

IntelliJ IDEA

Tutorial

2002.10.30 22:21
IDEA build 666
J2SE 1.4.1

ABOUT THIS DOC

this is the INTERNAL version of the Tutorial (red underlined text is internal commentary that does not appear in the PUBLIC version).

- This doc is being written by terry taylor (terry@intellij.com , (+ 7 921) 916 2841).
- The idea of this doc:
 - To introduce with simple step-by-step examples the IDEA functionality.
 - To determine what functionality is most important (ie, what is covered in this doc).
 - Verify the functionality of new builds by quickly doing the examples in this doc.

text coding

- red underlined text is internal commentary that does not appear in the PUBLIC version
- ~~text that should not appear in the final document marked by strikethru (the text will not be ready for this release, but should be included in future releases)~~
- empty chapters/sections are marked with "XXX"
- incomplete chapters/sections are marked with "X" (search for "X")
- terry commentary is marked with "TTT"
- unresolved questions marked with "??"

?? questions, suggestions (from TT)

- have users download the latest version of the documentation from www.intellij.com? the docs are changing too fast and i have too little finished, therefore letting users constantly download the latest docs would be the best answer. we could have the user download a zip, unzip to the idea/docs directory.

WHATS NEW

- 20021028 : changed
 - 4. Projects (page 63)
 - 6. Files (page 81)
 - 7. Editor X (page 107)
- 20021028 : major changes to chapters in
 - Part B. Projects / dirs / files (page 61).
- 20021028 : major changes to
 - 3. Basics (page 31).
- 20021025 : updated to build666
 - 2. Installation (page 19).
 - 3. Basics (page 31).
- 20021024 : edited
 - Copyright (page 5) added.
 - index entries added thruout release chapters.
 - 13. Compiler (page 283).
- 20021023 : edited
 - 11. Version control (page 255).
 - 12. Java Doc (page 269).
 - 13. Compiler (page 283).
 - 14. Ant (page 295).
 - 15. Debugger (page 301).
- 20021022 : edited
 - 9. Code Refactoring (page 201) finished.
- 20021018 : edited
 - page 1 of parts a,b,c,d updated.
 - 9. Code Refactoring (page 201)

- [10. Code Inspection X \(page 243\)](#)
- [20021017 : edited](#)
 - [8.3. Code templates \(page 184\) to 8.9. Comment \(page 199\)](#)
- [20021016 : edited](#)
 - [8.1. Code completion \(suggestion\) \(page 164\)](#)
- [20021015 : added](#)
 - [8.1. Code completion \(suggestion\) \(page 164\)](#)
 - [8.2. Code-completion OLD XXX \(page 179\)](#)
- [20021014 :](#)
 - [7. Editor X \(page 107\) text added.](#)
- [20021012 :](#)
 - [7. Editor X \(page 107\) text added.](#)
- [20021010 :](#)
 - [added 27. Keymaps X \(page 375\)](#)
 - [7. Editor X \(page 107\) reorganized.](#)
- [20021009 :](#)
 - [7. Editor X \(page 107\) reorganized.](#)
- [20021008 :](#)
 - [6. Files \(page 81\) rewritten.](#)
- [20021007 :](#)
 - [3. Basics \(page 31\) rewritten.](#)
 - [4. Projects \(page 63\) rewritten.](#)
 - [5. Directories \(packages\) \(page 75\) rewritten.](#)
- [20021004 :](#)
 - [2. Installation \(page 19\) rewritten.](#)
 - ~~[added conditional text style Hidden to text that should not appear in the final document \(this text is hidden \(marked by strikethru\)\)](#)~~
- [20021003: new chapters, deleted chapters.](#)
- [20021003: added text to](#)
 - [11.4. StarTeam \(page 265\)](#)
- [20021002: added text to](#)
 - [15. Debugger \(page 301\)](#)
 - [14. Ant \(page 295\)](#)
- [20021001: 12. Java Doc \(page 269\) text added.](#)
- [20020930:](#)
 - [About this doc \(page 7\). new chapter.](#)
 - [1. Documentation / Support \(page 15\) updated.](#)
 - [2. Installation \(page 19\). lots of pics add \(maybe delete later\).](#)
 - [8. Code Automation \(page 163\): new text.](#)
- [20020927: reorganized many chapters.](#)
- [20020925: updated:](#)
 - [7. Editor X \(page 107\)](#)
 - [8. Code Automation \(page 163\)](#)
 - [9. Code Refactoring \(page 201\)](#)
- [20020924: 7. Editor X \(page 107\) updated.](#)
- [20020923: a lot added to the chapters of Part B. Projects / dirs / files \(page 61\) \(projects, files\).](#)
- [20020922: a lot added to the chapters of Part B. Projects / dirs / files \(page 61\) \(projects, files\).](#)
- [20020919: idea examples added to Chapter 22. Plugins X \(page 345\).](#)
- [20020919: Reorganized the doc. Chapters removed/added.](#)

- [20020919 Chapter 3. Basics \(page 31\) updated.](#)
- [20020916: Reorganized the doc. Many chapters removed.](#)
- [20020912: Chapter 24. Tags \(page 91\) added.](#)
- [20020912: Chapter 21. EJB X \(page 329\) added \(uses Sun ejb example\).](#)
- [20020909: Chapter 9. Code Refactoring \(page 201\) partially completed.](#)
- [20020908: Chapter 11. Version control \(page 255\): history dialog described \(local vcs\).](#)

to create a PUBLIC version

1. [import latest faq/ts/glossary tables. convert to text.](#)
2. [delete any incomplete chapters from book. save book as public version.](#)
3. [import styles.](#)
4. [update book.](#)
5. [save as pdf.](#)
6. [check:](#)
 - [right/left sides](#)
7. [in acrobat: add bookmarks to other docs.](#)

Copyright

© 2002 JetBrains, Inc. All rights reserved.

JetBrains, Inc., JetBrains, IntelliJ, IDEA, and IntelliJ Labs are either registered trademarks or trademarks of JetBrains, Inc., s.r.o. in the Czech Republic and in other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Information in this document is subject to change without notice. JetBrains, Inc. makes no warranties, expressed nor implied, in this document. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise), or for any purpose, without the express written permission of JetBrains, Inc.

2002.10.30 22:21

About this doc

20020930TT added this chapter. need to expand.

This tutorial is designed to help you get started with IntelliJ IDEA, the Java development tool from JetBrains, as quickly as possible. IDEA integrates into a compact package an impressive array of functions, and this document introduces the functionality you require to start working effectively with IDEA.

This document contains the following major parts:

- **Part A. Quick Start (page 13)**
- **Part B. Projects / dirs / files (page 61)**
- **Part C. Editing files (page 105)**
- **Part D. Compile / Debug (page 281)**
- ~~Part E. Applications (page 317)~~
- ~~Part F. Tools and resources (page 343)~~
- ~~Part G. Appendices (page 369)~~

Each part contains several chapters, which are described on the first page of the part.

A **List of figures (page 381)** and **Index (page 387)** are also included at the end of this doc.

TOC

<i>Copyright</i>	5
<i>About this doc</i>	7

Part A. Quick Start..... 13

1. Documentation / Support	15
1.1. Doc newsgroup (jetbrains.intelliij.documen- tation)	15
1.2. Documentation sources	16
1.3. Documents / help	17
1.4. Support	18
2. Installation	19
2.1. Java 2 SDK (1.4.1 Windows i586)	19
2.2. IDEA	22
2.2.1. Download / Install	22
2.2.2. Get evaluation license	26
2.2.3. Start IDEA (enter license)	27
2.2.4. Enter non-evaluation license XXX	28
2.2.5. Register XXX	29
3. Basics	31
3.1. Create project	32
3.1.1. Specify project name / location	32
3.1.2. Specify target JDK	33
3.1.3. Specify project paths	35
3.2. Create package	38
3.3. Create class	39
3.4. Use the editor	41
3.4.1. Entering text	41
3.4.2. Find / Navigation	42
3.4.3. Colors and fonts	43
3.4.4. Code style	44
3.4.5. Error indication	45
3.4.6. Todo	46
3.5. Automate code	47
3.6. Refactor code	49
3.7. Use version control	50
3.8. View JavaDoc	51
3.9. Compile	53
3.10. Debug	54
3.10.1. Create application configuration	54
3.10.2. Run the configuration	56
3.10.3. Set breakpoint	57
3.10.4. Step through application	58

Part B. Projects / dirs / files . 61

4. Projects	63
4.1. Basic operations	64

4.1.1. New (New Project Wizard)	64
4.1.2. Open	64
4.1.3. Close	64
4.1.4. Reopen	65
4.1.5. Delete	65
4.2. View / Modify settings	66
4.2.1. Project properties dialog	66
4.2.2. Project tool	72
4.2.3. Other dialogs	73
4.2.4. Project xml file (.ipr)	73
5. Directories (packages)	75
5.1. Basic operations	75
5.1.1. Create	76
5.1.2. Delete	78
5.1.3. Cut / Copy / Paste	79
5.1.4. Recover XXX	79
5.1.5. Rename (refactor)	79
5.1.6. Move (refactor)	79
6. Files	81
6.1. Basic operations	81
6.1.1. Create	82
6.1.2. Rename (refactor)	83
6.1.3. Move (refactor)	83
6.1.4. Copy (refactor)	83
6.1.5. Delete (safe)	84
6.1.6. Recover	85
6.1.7. Export to HTML	86
6.1.8. Print	87
6.2. Types / Templates	88
6.2.1. Default file types	88
6.2.2. Custom file types	91
6.2.3. Class / File templates	93
6.3. Views	94
6.3.1. Editor	95
6.3.2. Project tool (show members)	95
6.3.3. Commander	95
6.3.4. (File) Structure tool	96
6.3.5. Hierarchy tool	100

Part C. Editing files 105

7. Editor X	107
7.1. File operations	108
7.1.1. Open	108
7.1.2. Synchronize	110
7.1.3. Save all	110
7.1.4. Close	110
7.2. Text editing X	111
7.2.1. Undo / Redo	111

7.2.2. Select	111	8.2. Code-completion OLD XXX	179
7.2.3. Cut / Copy / Paste / Duplicate / Delete	112	8.2.1. Code completion X	179
7.2.4. Move / Scroll	113	8.2.2. Lookup list XXX	183
7.2.5. Indent / tabs / lines	122	8.2.3. Parameter info XXX	183
7.2.6. Modify text	123	8.3. Code templates	184
7.3. Find / Navigation	124	8.3.1. Live Templates	184
7.3.1. Bookmarks	124	8.3.2. Surround with	191
7.3.2. Find / Replace	126	8.4. Code generation	192
7.3.3. Go to	133	8.5. Import optimization	193
7.4. Colors and fonts X	137	8.5.1. File	193
7.4.1. General X	137	8.5.2. Files in directory	193
7.4.2. Java X	138	8.6. Method override	194
7.4.3. HTML X	139	8.7. Interface implementation	196
7.4.4. XML X	140	8.8. Method delegation	197
7.4.5. JSP X	141	8.9. Comment	199
7.4.6. Custom X	142	9. Code Refactoring	201
7.4.7. Color Schemes	143	9.1. Migration XXX	202
7.5. Code style X	145	9.1.1. Migrate	202
7.5.1. General X	145	9.1.2. Create map	202
7.5.2. Indent and Braces X	146	9.2. Rename	203
7.5.3. Blank lines X	147	9.2.1. Package	203
7.5.4. Spaces X	148	9.2.2. Class	205
7.5.5. Imports X	149	9.2.3. Method	207
7.5.6. EJB Names X	150	9.2.4. Field	208
7.5.7. Code style schemes	151	9.2.5. Variable	209
7.5.8. Autoindent	153	9.2.6. Parameter	210
7.5.9. Reformat	154	9.3. Move	211
7.6. Error indication X	156	9.3.1. Package	211
7.6.1. Deprecated symbol	156	9.3.2. Class	213
7.6.2. Reparse delay	156	9.3.3. Members	214
7.6.3. Unused import	157	9.3.4. Inner to upper level	215
7.6.4. Unused symbol	157	9.4. Change method signature	216
7.6.5. Unused throws declaration XXX	157	9.4.1. Add parameter	216
7.6.6. Redundant type cast	157	9.4.2. Move parameter	218
7.6.7. Silly assignment XXX	158	9.4.3. Change name	219
7.6.8. Wrong package statement	158	9.4.4. Change type	219
7.6.9. Javadocs errors XXX	158	9.5. Copy class	220
7.6.10. Unknown Javadoc tags	158	9.6. Extract	221
7.6.11. EJB errors XXX	159	9.6.1. Method	221
7.6.12. EJB warnings XXX	159	9.6.2. Interface	222
7.7. Todo	160	9.6.3. Superclass	224
7.7.1. Basics	160	9.7. Use interface where possible	226
7.7.2. Patterns	161	9.8. Pull/push members	228
7.7.3. Filters	162	9.8.1. Pull Up	228
8. Code Automation	163	9.8.2. Push Down	229
8.1. Code completion (suggestion)	164	9.9. Introduce	230
8.1.1. Suggest after dot (auto)	166	9.9.1. Variable	230
8.1.2. Suggest after partial text (manual) ..	167	9.9.2. Field	231
8.1.3. Suggest in XML (auto) X	174	9.9.3. Constant	232
8.1.4. Suggest after @ (javadoc) (auto) X ..	174	9.9.4. Parameter	233
8.1.5. Options	175	9.10. Inline	234

9.10.1. Variable	234
9.10.2. Method	235
9.11. Encapsulate field	237
9.12. Replace temp with query	239
9.13. Convert Anonymous to Inner	240
10. Code Inspection X	243
10.1. Run inspection X	243
10.1.1. File X	243
10.1.2. Project X	245
10.1.3. Rerun X	245
10.2. Resolve problems X	246
10.2.1. Implement suggested resolution X	246
10.2.2. Manual X	249
10.3. Supported inspections XXX	250
10.3.1. Unreachable declaration XXX	250
10.3.2. Declaration redundancy XXX	250
10.3.3. Local code analysis XXX	250
10.3.4. Deprecated API usage XXX	251
10.3.5. equals() and hashCode() not paired XXX	251
10.3.6. EJB Errors & Warnings XXX	251
10.4. Entry points XXX	252
10.4.1. Add XXX	252
10.4.2. View in inspection (unreachable code) XXX	252
10.5. Export to html XXX	253
10.6. Offline inspection results XXX	254
10.6.1. Create XXX	254
10.6.2. View XXX	254
11. Version control	255
11.1. Local	256
11.1.1. Open History dialog	256
11.1.2. Insert	258
11.1.3. Modify	259
11.1.4. Delete	259
11.1.5. Rollback	260
11.1.6. Viewing changes	261
11.1.7. Labels	262
11.2. CVS XXX	263
11.3. SourceSafe XXX	264
11.4. StarTeam	265
11.4.1. Install	265
11.4.2. Configure connection	265
11.4.3. Add StarTeam dir to project	266
11.4.4. Create dir / file	266
11.4.5. Check in	267
11.4.6. Show differences	267
11.4.7. Check in	268
12. Java Doc	269
12.1. JDK JavaDoc	269

12.1.1. Install	269
12.1.2. View	271
12.2. Quick JavaDoc	273
12.2.1. JDK class	273
12.2.2. Own class	275
12.3. JavaDoc tags	276
12.3.1. Add tags	276
12.3.2. Display tags in quick JavaDoc	277
12.4. Own JavaDoc	278
12.4.1. Generate	278
12.4.2. Install	280
12.4.3. View	280

Part D. Compile / Debug.... 281

13. Compiler	283
13.1. Compile variations	283
13.1.1. Build / Rebuild	283
13.1.2. Make	285
13.1.3. Compile (single file)	285
13.2. Compiler messages dialog	286
13.2.1. Stop	286
13.2.2. Close	286
13.2.3. Previous / next message	286
13.2.4. Export to text file	286
13.2.5. Expand / collapse all	287
13.2.6. Hide warnings	287
13.2.7. Autoscroll to source	287
13.2.8. Compiler properties	288
13.3. Compiler options	289
13.3.1. General	289
13.3.2. Javac-specific	292
14. Ant	295
14.1. Download / Install	295
14.2. Create build.xml	297
14.3. Add Ant build to IDEA	298
14.4. Run build	299
15. Debugger	301
15.1. Run/Debug configurations	301
15.1.1. Types	301
15.1.2. Basic operations	302
15.2. Types of breakpoints / watchpoints	304
15.2.1. Line	304
15.2.2. Exception	304
15.2.3. Field (watchpoints)	306
15.2.4. Method	307
15.3. Run / debug actions	308
15.3.1. Pause / Resume / Stop / Close	308
15.3.2. Step over / Step into / Step out	309
15.3.3. Run to cursor	311
15.3.4. Show execution point	311

15.3.5. View breakpoints 311
16. JUnit XXX 313
16.1. xxx 313
17. Remote debugging XXX 315
17.1. xxx 315

Part E. Applications 317

18. Applications XXX 319
18.1. xxx 319
19. Applets XXX 321
19.1. xxx 321
20. Web applications XXX 323
20.1. Install Tomcat XXX 323
20.2. JSP XXX 324
20.3. Tags 325
20.3.1. Create HelloTag.java 325
20.3.2. Create mytaglib.tld 326
20.3.3. Create Hello.jsp 326
20.3.4. Test 327
20.4. Servlets XXX 328
21. EJB X 329
21.1. Install / start J2EE 329
21.1.1. Download / install J2EE from sun ... 329
21.1.2. Install 329
21.1.3. Start J2EE server 330
21.1.4. Start Cloudscape 331
21.2. Deploy Sun example application ... 332
21.2.1. Start deploy tool 332
21.2.2. Open the application 332
21.2.3. Generate SQL 333
21.2.4. Deploy 334
21.3. Set IDEA EJB project properties ... 337
21.4. Modify source in IDEA 340
21.5. Redeploy 341

Part F. Tools and resources 343

22. Plugins X 345
22.1. Install a plugin 345
22.1.1. Find a plugin 345
22.1.2. Install files 346
22.1.3. Using the plugin 346
22.2. Create a plugin 347
22.2.1. Application / project plugin 347
22.2.2. Action plugin 351
22.2.3. Tool window plugin 355
22.2.4. Virtual file system (Vfs) plugin 358
22.3. Publish a plugin XXX 362
23. External Tools X 363

23.1. Add 363
23.2. Open 365
24. External Resources XXX 367
24.1. XXX 367

Part G. Appendices 369

25. FAQ XXX 371
FAQ 1. Group of faqs..... 371
FAQ 2. Group of faqs..... 371
26. Trouble shooting XXX 373
TS 1. Group of TSs..... 373
TS 2. Group of TSs..... 373
27. Keymaps X 375
27.1. Select Active 375
27.2. Create (copy and modify) 376
28. Glossary XXX 379
List of figures 381
Index..... 387

Part A. Quick Start

[20021028TTT last edit.](#)

The chapters in this part introduce the basics required to start using IDEA. The contents of this part are very similar to the Quick Start document.

- 1. Documentation / Support (page 15).** Provides an overview of IDEA documentation and support.
- 2. Installation (page 19).** Demonstrates how to download and install the Java 2 SDK and IDEA.
- 3. Basics (page 31).** Demonstrates IDEA basis with simple examples.

1. Documentation / Support

add: show how to add components (for plugins), vote, add comments, etc.

20020906TTT: this chapter should list all available docs and info. i would also like to have links to the other pdf and help docs once the dir structure of the docs is finalized.

?? list plugin docs and internet websites.

This tutorial should hopefully provide all the information you need to quickly get started with IntelliJ IDEA without any other docs or assistance.

However, if you need other information or assistance, then you can refer to

- **1.1. Doc newsgroup (jetbrains.intelij.documentation) (page 15)**
- **1.2. Documentation sources (page 16)**
- **1.3. Documents / help (page 17)**
- **1.4. Support (page 18)**

1.1. Doc newsgroup (jetbrains.intelij.documentation)

jetbrains.intelij.documentation is the newsgroup for IntelliJ documentation and provides:

- Useful information about errors in docs.
- Feedback and recommendations.

You can subscribe to this newsgroup at <news://news.jetbrains.com> .

1.2. Documentation sources

IntelliJ is constantly developing the user documentation for IDEA. The currently available sources of documentation include:

- www.intellij.com
- www.intellij.net
- www.intellij.org

www.intellij.com

www.intellij.com is the official IntelliJ website and provides the following:

- **Features list.**
- **Downloads** (www.intellij.com/idea/download.jsp). You can download a fully functional version of IDEA.
- Temporary **license keys** (www.intellij.com/idea/evaluate.jsp)
- **Purchasing info.**
- **Online FAQ** (www.intellij.com/support/faq/)

www.intellij.net

www.intellij.net is the IntelliJ Technology Network (ITN) site. The site provides access to

- IntelliJ bugs/features database
- User forums
- Early access builds of IDEA
- Online support

www.intellij.org

www.intellij.org is the website driven by the IntelliJ products community. Topics include

- **Using IDEA** (features, hints, tips, problems)
- **OpenAPI**
- **Plugins**
- **Suggestions, ideas, requests d**

1.3. Documents / help

- [IDEA Tutorial \(this doc\)](#)
- [IDEA Quick Start](#)
- ~~[IDEA User Guide](#)~~
- [IDEA Help](#)
- [IDEA Tool Tips](#)

IDEA Tutorial (this doc)

This tutorial provides a very detailed step-by-step introduction to the functionality of IntelliJ IDEA.

IDEA Quick Start

The IDEA [Quick Start](#) is designed to introduce as quickly as possible the functionality of IntelliJ IDEA with simple step-by-step examples.

~~IDEA User Guide~~

~~The User Guide describes in detail the complete functionality of IntelliJ IDEA.~~

IDEA Help

IDEA includes context-sensitive online help.

IDEA Tool Tips

[20020906TTT ?? should we add a chapter at the end of this doc listing the tooltips?](#)

Each time IDEA is started, a tooltip will appear (if enabled). You can optionally scroll through the entire set of tooltips.

There are several internet websites that provide information and user support for IDEA.

1.4. Support

If you need support, then we first recommend that you consult

- **FAQ**
- ~~TS~~
- **ITN database**

~~If you still can't find the answers, then IntelliJ provides support via~~

- ~~Phone~~
- ~~Email~~

FAQ

For common questions refer to the available FAQs:

- ~~Chapter 25. FAQ XXX (page 374).~~
- <http://www.intellij.com/support/faq/> (may contain more recent info than this doc).
- **Online FAQ** at <http://www.iguru.com/faq/home.jsp?topic=IntelliJIDEA> (questions can be posted at <http://www.iguru.com/forums/home.jsp?topic=IntelliJIDEA>).

~~TS~~

~~For solutions to specific problems refer to~~

- ~~Chapter 26. Trouble shooting XXX (page 373).~~

ITN database

The IntelliJ Technology Network (ITN) database at <http://www.intellij.net> contains the following:

- **Forums.** IDEA users discuss various topics.
- **Tracker.** Tracks various bugs and fixes.
- **Early Access Program (EAP).** Contains the latest builds. The problem you are having may be associated with a particular build.

Phone

- ~~How to get phone support.~~
- ~~What info to gather before the call.~~
- ~~Where to call.~~

Email

~~[Do we have email support?](#)~~

2. Installation

20021029TTT: updated.

Contacts: zheka.

20021003ANN: Somewhere in the beginning of installation instructions, it would be nice either to explain that everything is going about Windows (with reference to the places where installation procedure for other systems is described), or to describe all possible installations. TTT: i will write a description for other operating systems later... this is too complicated right now.

This chapter describes the Windows installation of

- **2.1. Java 2 SDK (1.4.1 Windows i586) (page 19)**
- **2.2. IDEA (page 22)**

2.1. Java 2 SDK (1.4.1 Windows i586)

This section describes how to download and install the Java 2 Software Development Kit (J2SDK).

2.1. Download the J2SE 1.4.1 from <http://java.sun.com/j2se/downloads.html> (the downloaded file is `j2sdk-1_4_1-windows-i586.exe`).

Note: The rest of this book assumes that you have installed this version of the JDK.

describe path settings???

2.2. Double-click on `j2sdk-1_4_1-windows-i586.exe`. The files are extracted. The dialog “Welcome...” appears.

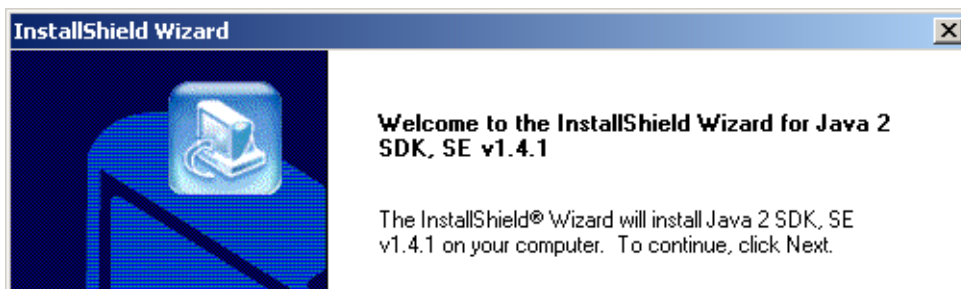


Figure 2.1. J2SDK installation welcome (321)

2.3. Click **Next**. The dialog “License agreement” appears.

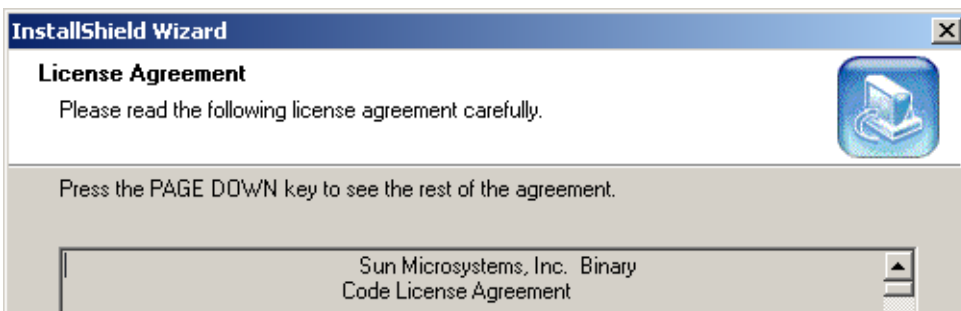


Figure 2.2. J2SDK installation license agreement (320)

2.4. Click **Yes**. The dialog “Choose destination location” appears.

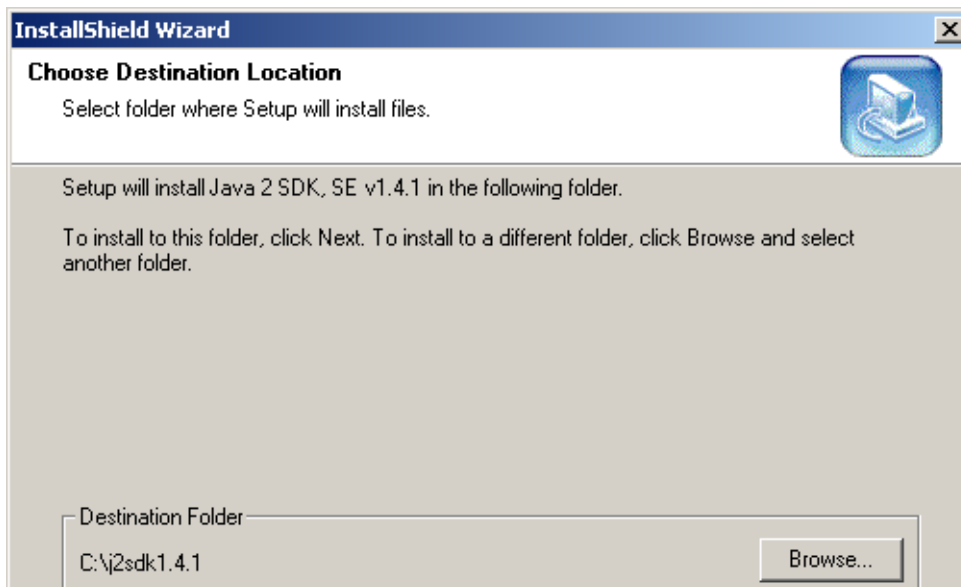


Figure 2.3. J2SDK installation destination (319)

2.5. Click **Next**. The dialog “Select components” appears.

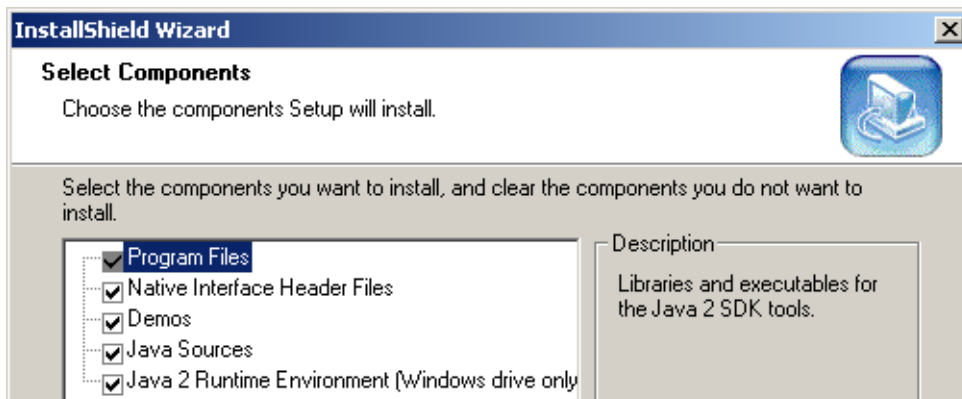


Figure 2.4. J2SDK installation components (317)

2.6. Click **Next**. The dialog “Select Browsers” appears.



Figure 2.5. J2SDK installation browser select (318)

2.7. Click **Next**. The files are copied. Finally the dialog “InstallShield Wizard Complete” appears.

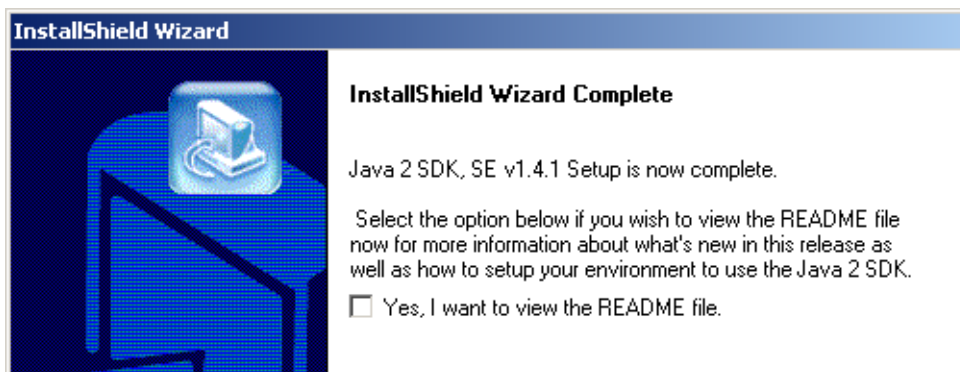


Figure 2.6. J2SDK installation complete [\(316\)](#)

2.8. Click **Finish**.

2.2. IDEA

This section describes the complete IDEA installation including the following:

- 2.2.1. Download / Install (page 22)
- 2.2.2. Get evaluation license (page 26)
- 2.2.3. Start IDEA (enter license) (page 27)
- ~~2.2.4. Enter non-evaluation license XXX (page 28)~~
- ~~2.2.5. Register XXX (page 29)~~

2.2.1. Download / Install

2.9. Download the installer from <http://www.intellij.com/idea/download.jsp>:

<http://www.intellij.com/idea/download.jsp>



Figure 2.7. Windows installer download page (322)

2.10. Double-click on **idea665.exe**. The dialog “Introduction” appears.

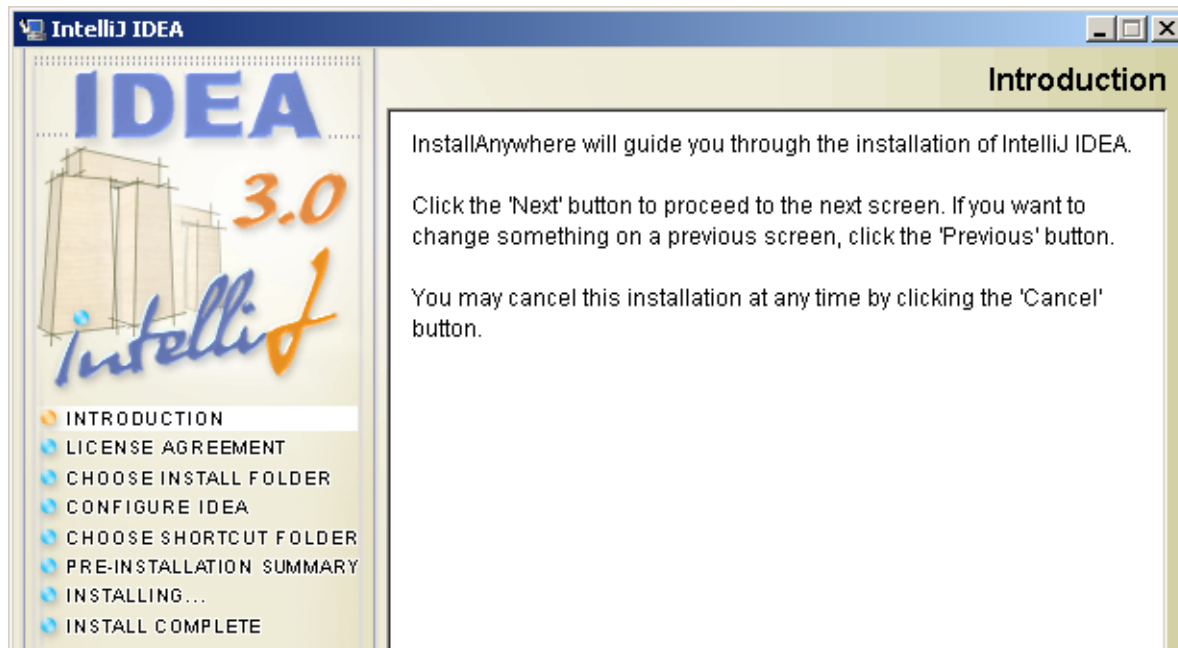


Figure 2.8. IDEA EXE installation introduction (336)

2.11. Click **Next**. The dialog “License agreement” appears.



Figure 2.9. IDEA EXE installation license agreement (335)

2.12. Select **I accept the terms**.

2.13. Click **Next**. The dialog “Choose install folder” appears.



Figure 2.10. IDEA EXE installation choose folder (333)

2.14. Click **Next**. The dialog for storing settings appears.

2.15. Select **In the IDEA installation directory**.



Figure 2.11. IDEA folder for storing settings (192)

2.16. Click **Next**. The dialog for storing the data cache appears.

2.17. Select **In the IDEA installation directory**.

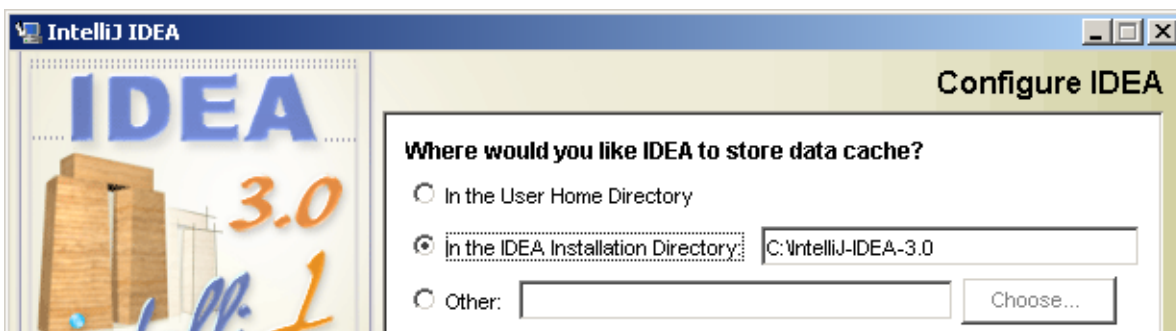


Figure 2.12. IDEA folder for data cache (191)

2.18. Click **Next**. The dialog “Choose shortcut folder” appears.



Figure 2.13. IDEA EXE installation shortcut folder (332)
2.19. Click **Next**. The dialog “File associations” appears.

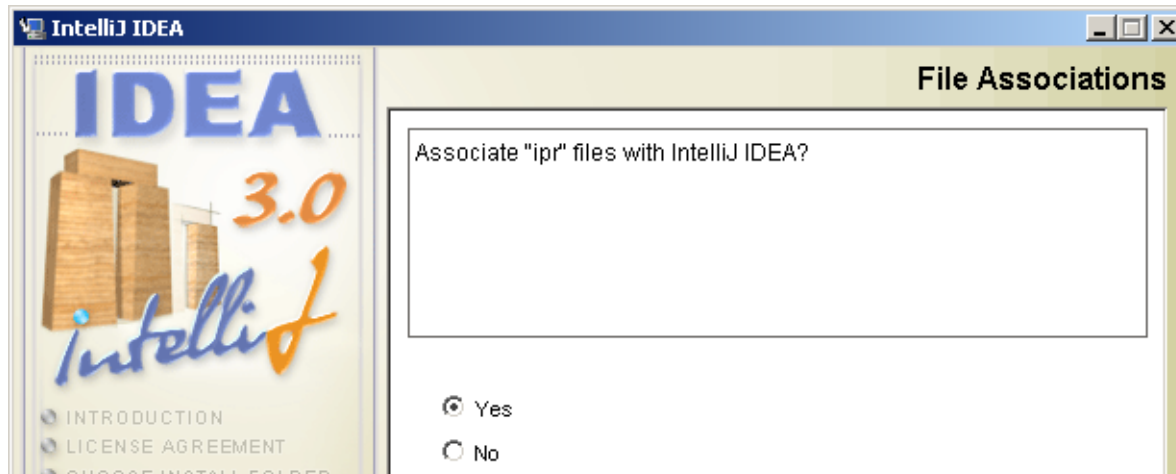


Figure 2.14. IDEA EXE installation file associations (331)
2.20. Click **Next**. The dialog “Pre-installation summary” appears.

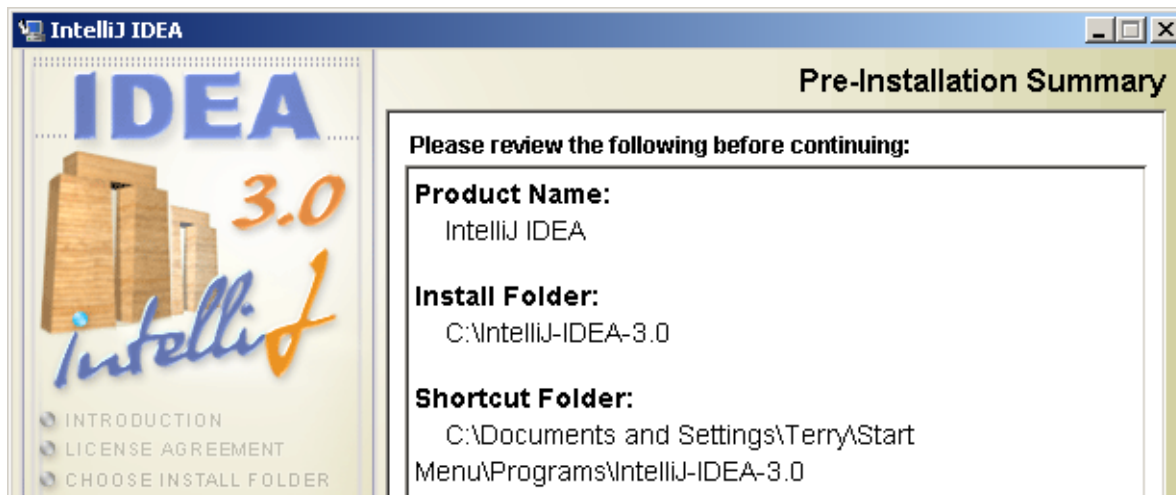


Figure 2.15. IDEA EXE installation pre-install summary (330)
2.21. Click **Install**. IDEA is installed. The dialog for the readme.txt file appears.

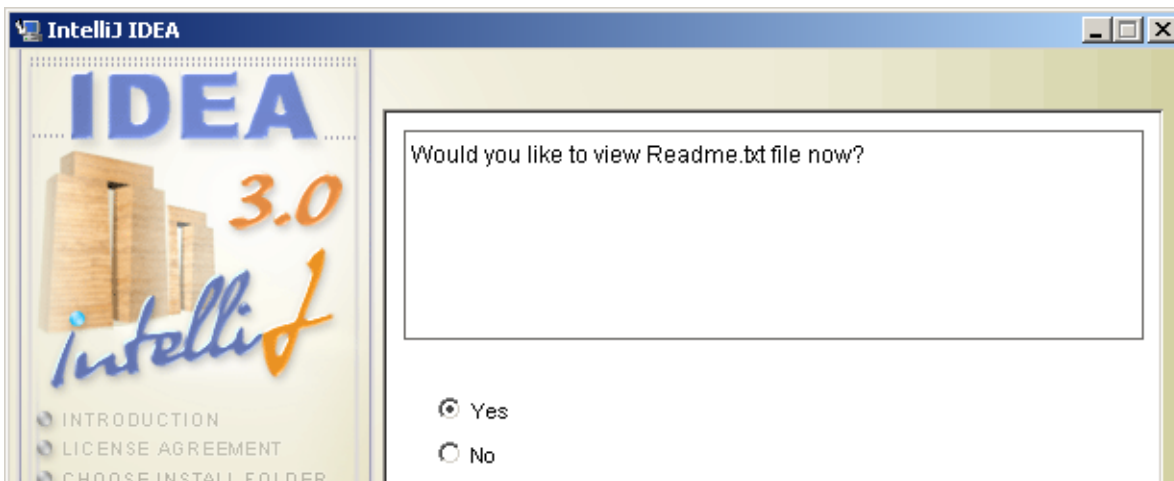


Figure 2.16. IDEA EXE installation readme (329)
2.22. Click **Next**. The readme.txt appears.

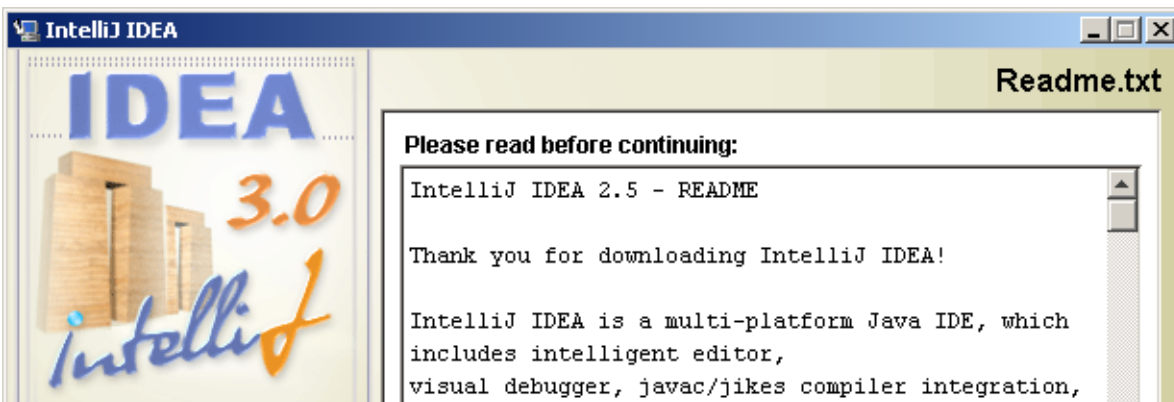


Figure 2.17. IDEA EXE installation readme.txt (328)
2.23. Click **Next**. The dialog "Install complete" appears.

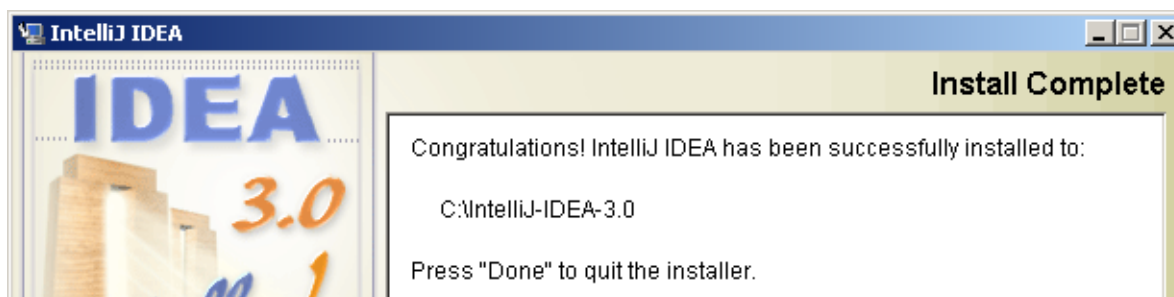
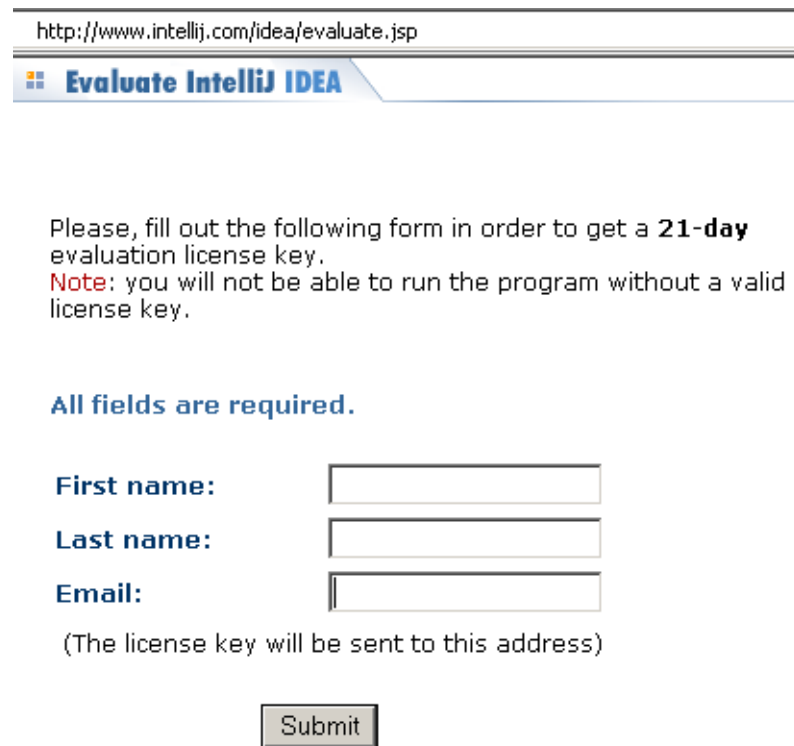


Figure 2.18. IDEA EXE install complete (327)
2.24. Click **Done**. The wizard closes. The installation is complete.

2.2.2. Get evaluation license

2.25. Fill out and submit the license form at <http://www.intellij.com/idea/evaluate.jsp>.



The screenshot shows the top of a web browser window with the address bar containing `http://www.intellij.com/idea/evaluate.jsp`. Below the address bar is a blue header with the IntelliJ IDEA logo and the text "Evaluate IntelliJ IDEA". The main content area contains the following text:

Please, fill out the following form in order to get a **21-day** evaluation license key.
Note: you will not be able to run the program without a valid license key.

All fields are required.

First name:

Last name:

Email:

(The license key will be sent to this address)

Figure 2.19. License form (254)

The following are sent to the email address entered in the form:

- User name.
- Evaluation key (alpha-numeric)

2.2.3. Start IDEA (enter license)

You are now ready to start IDEA.

2.26. From the Start menu select **Programs | IntelliJ IDEA 3.0 | IntelliJ IDEA**. The dialog “Enter license data” appears.

2.27. Enter your **User name** and **License key**.

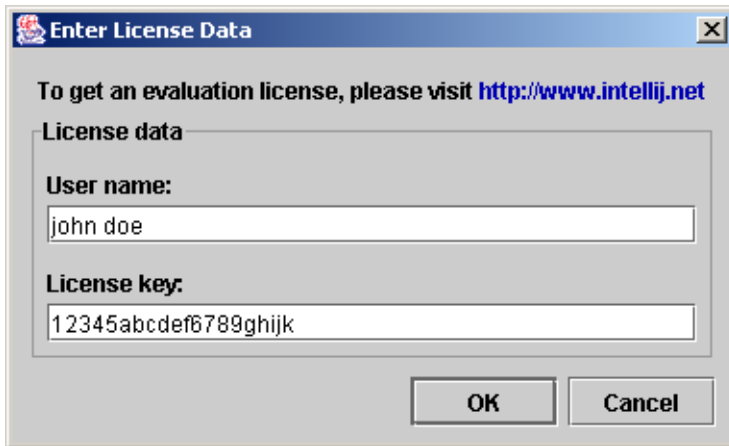


Figure 2.20. License data dialog (252)

2.28. Click **OK**.

2.29. The dialog “License agreement for IntelliJ IDEA” appears.

2.30. Check **Accept all terms of the license**.



Figure 2.21. License agreement dialog (251)

2.31. Click **OK**.

The new project wizard appears.

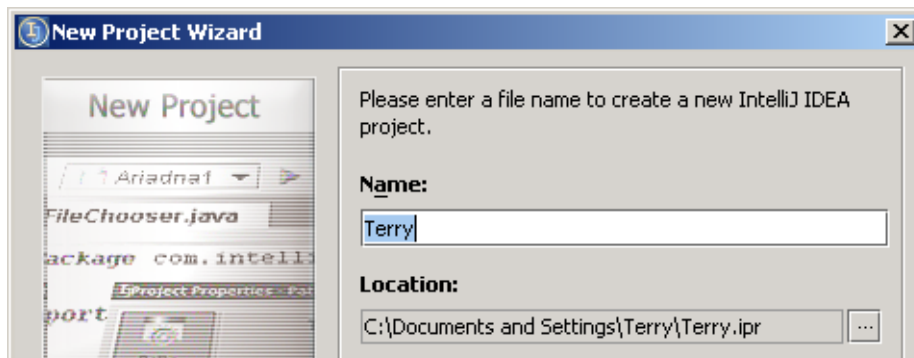


Figure 2.22. New project wizard (250)

~~2.2.4. Enter non-evaluation license XXX~~

~~describe how to enter license info after getting a real license.~~

~~2.2.5. Register XXX~~

~~Using the evaluation license: first, please note the reader that the license given from the site is just an evaluation one; more important is that in order to register the license within IDEA, the user has not to copy the "idea.license" file, since there is just no such file; the user should enter the registration data into the "Registration" dialog that appears when first starting the application, and then it automatically generates the "idea.license" file in the right directory; thus this section should go AFTER "Starting IDEA", not before it, since this dialog will appear only after start.~~

3. Basics

[20021028TTT: updated.](#)

[for help: zheka, anton](#)

This chapter describes basic IDEA functionality and concepts. In this chapter you will

- **3.1. Create project (page 32)**
- **3.2. Create package (page 38)**
- **3.3. Create class (page 39)**
- **3.4. Use the editor (page 41)**
- **3.5. Automate code (page 47)**
- **3.6. Refactor code (page 49)**
- **3.7. Use version control (page 50)**
- **3.8. View JavaDoc (page 51)**
- **3.9. Compile (page 53)**
- **3.10. Debug (page 54)**

Projects, packages (dirs), and files are described in more detail in Part B. Projects / dirs / files (page 61) which includes the following chapters

- *4. Projects (page 63)*
- *5. Directories (packages) (page 75)*
- *6. Files (page 81)*

Editor, code automation, code refactoring, version control, and JavaDoc are described in more detail in Part C. Editing files (page 105) which includes the following chapters

- *7. Editor X (page 107)*
- *8. Code Automation (page 163)*
- *9. Code Refactoring (page 201)*
- *11. Version control (page 255)*
- *12. Java Doc (page 269)*

Compilation and running/debugging are described in more detail in Part D. Compile / Debug (page 281) which includes the following chapters:

- *13. Compiler (page 283)*
- *15. Debugger (page 301)*

3.1. Create project

To create a project you will

- **3.1.1. Specify project name / location (page 32)**
- **3.1.2. Specify target JDK (page 33)**
- **3.1.3. Specify project paths (page 35)**


Projects will be discussed in more detail in Chapter 4. Projects (page 63).

3.1.1. Specify project name / location

3.1. For the project “Name”: Enter “**MyProject**”.



Figure 3.1. Project name (187)

3.2. For the project “Location”: Click on the ellipsis button (). The dialog “Select path” appears.

3.3. Right-click on **C:**. A popup dialog appears.

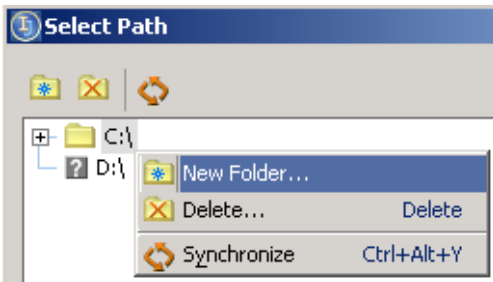


Figure 3.2. Popup “New Folder” (248)

3.4. Click on **New Folder**. The dialog “New Folder” appears.

3.5. For “Enter a new folder name”: Enter **MyProjectFolder**.

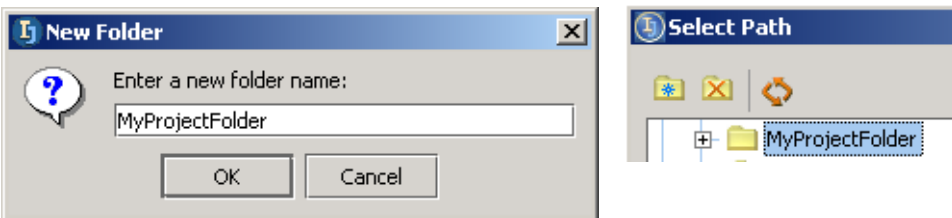


Figure 3.3. New folder name (052,186)

3.6. Click **OK**. The folder is created.

3.7. Double-click on **MyProjectFolder**. Note the new project location.

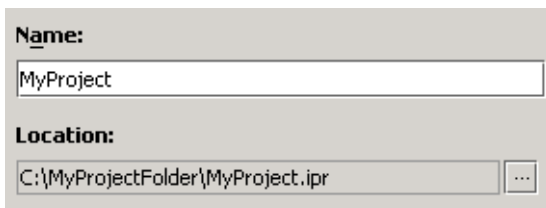



Figure 3.4. Project name / location (053)

3.1.2. Specify target JDK

3.8. For the project target “JDK”: Click on the ellipsis button (). The dialog “Select JDK” appears.

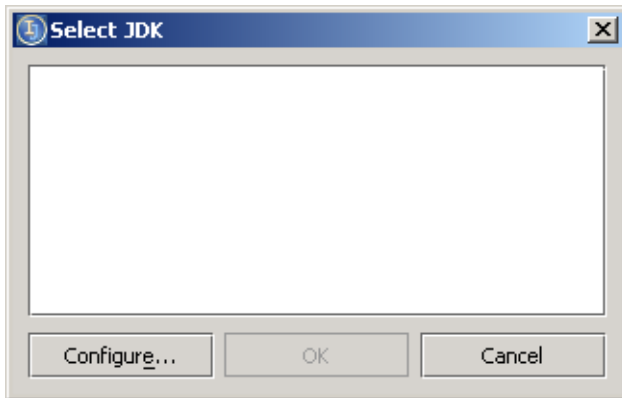


Figure 3.5. Select JDK (190)

3.9. Click on **Configure**.

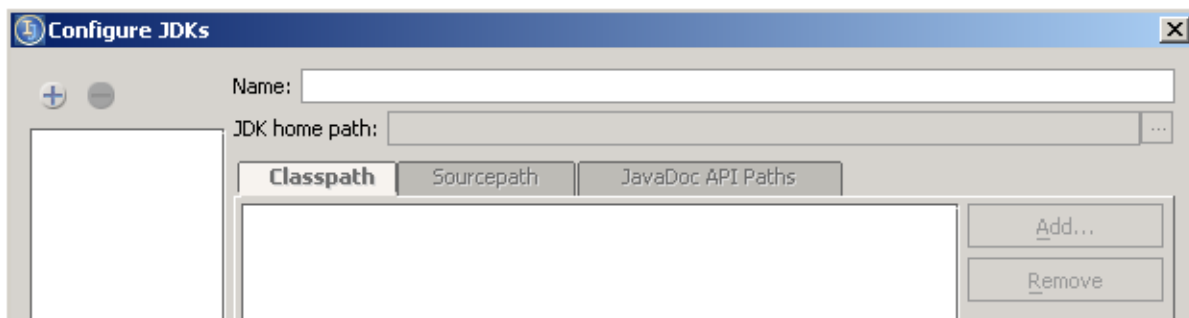



Figure 3.6. Dialog “Configure JDKs” (185)

3.10. Click on the button “Create JDK insert” (). The dialog “Select JDK Home Directory” appears.

3.11. Select **C:\j2sdk1.4.1**.



Figure 3.7. Dialog “Select JDK Home Directory” (183)

3.12. Click **OK**. The JDK is configured.

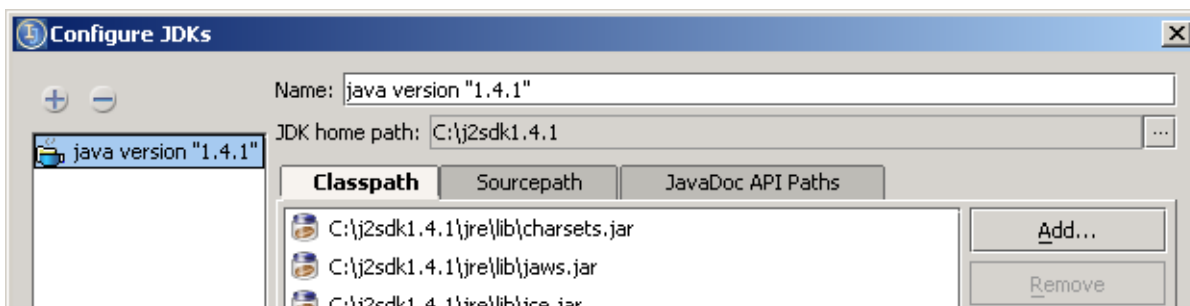


Figure 3.8. Dialog with configured JDK (182)

3.13. Click **OK**.

3.14. In the dialog “Select JDK” select **java version “1.4.1”**.

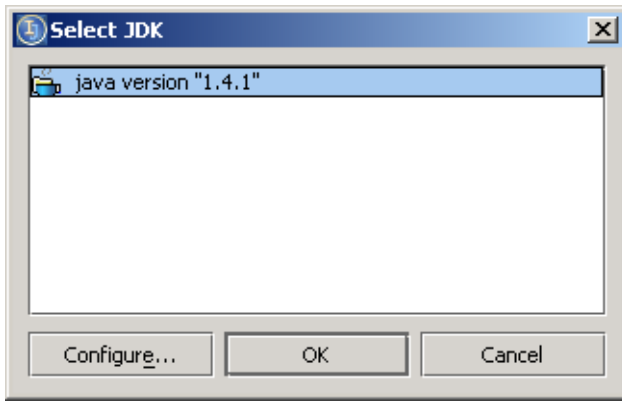


Figure 3.9. Configure JDKs [\(189\)](#)

3.15. Click **OK**. The JDK is shown in the new project wizard.

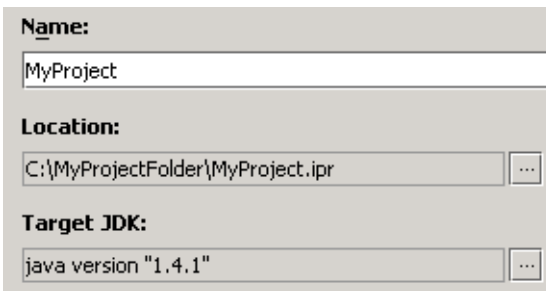


Figure 3.10. Configured JDK [\(188\)](#)

3.1.3. Specify project paths

You will now specify the following project paths:

- [3.1.3.1. Output path \(page 35\)](#)
- [3.1.3.2. Project path \(directories\) \(page 35\)](#)
- [3.1.3.3. Source path \(page 35\)](#)
- [3.1.3.4. Class path \(page 36\)](#)

3.1.3.1. Output path

- 3.16. Click on the **Compiler output path** ellipsis button . The dialog “Select Path” appears.
- 3.17. Right-click on **MyProjectFolder**. A context dialog appears.
- 3.18. Click **New Folder**. The dialog “New Folder” appears.
- 3.19. For the new folder name enter “**MyOutputFolder**”.

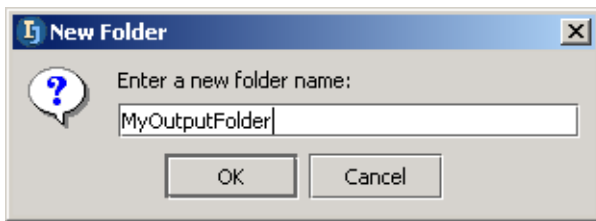


Figure 3.11. Output folder name [\(581\)](#)

- 3.20. Click **OK**. The folder is created.

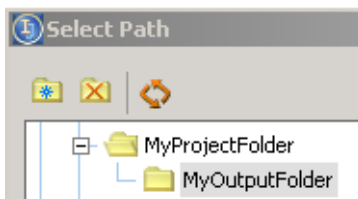


Figure 3.12. Select folder Output [\(246\)](#)

- 3.21. Click **OK**. The output folder is now selected.

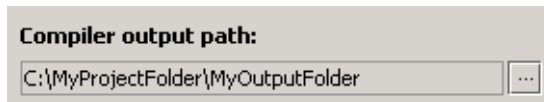


Figure 3.13. Compiler output folder [\(579\)](#)

3.1.3.2. Project path (directories)

[20020906TTT: ?? dialog should have a title.](#)

- 3.22. Click **Next**. The dialog with the project path appears. Note that the project path is **c:\MyProject-Folder**.

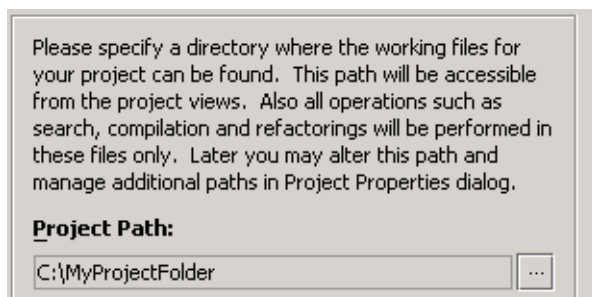


Figure 3.14. Project path [\(054\)](#)

3.1.3.3. Source path

3.23. Click **Next**. The dialog with the source paths appears. Note that the source path `c:\MyProject-Folder\src` is automatically suggested.

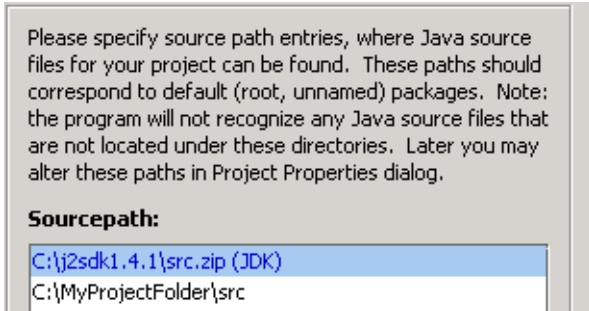


Figure 3.15. Source paths (055)

3.24. Click **Next**. A message dialog appears.

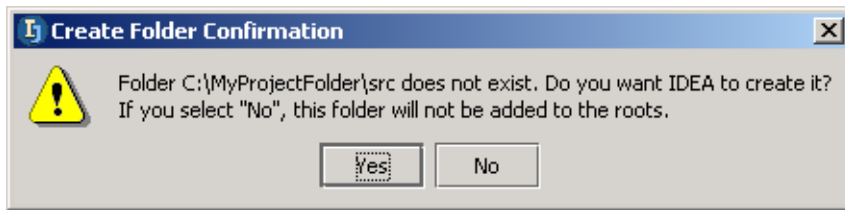


Figure 3.16. Create folder \src confirmation (056)

3.25. Click **Yes** to create the source path folder. The dialog with the class paths appears.

3.1.3.4. Class path

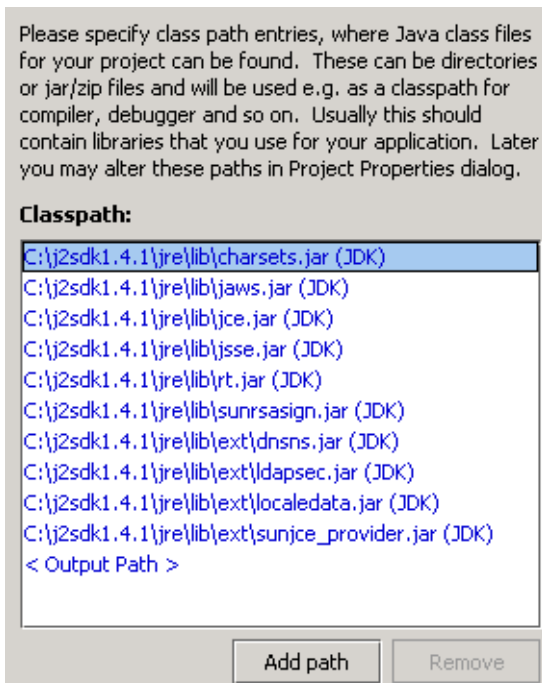


Figure 3.17. Class paths (057)

3.26. Click **Finish**. The project files are created (the project is built). The main dialog and the tip of the day appears.

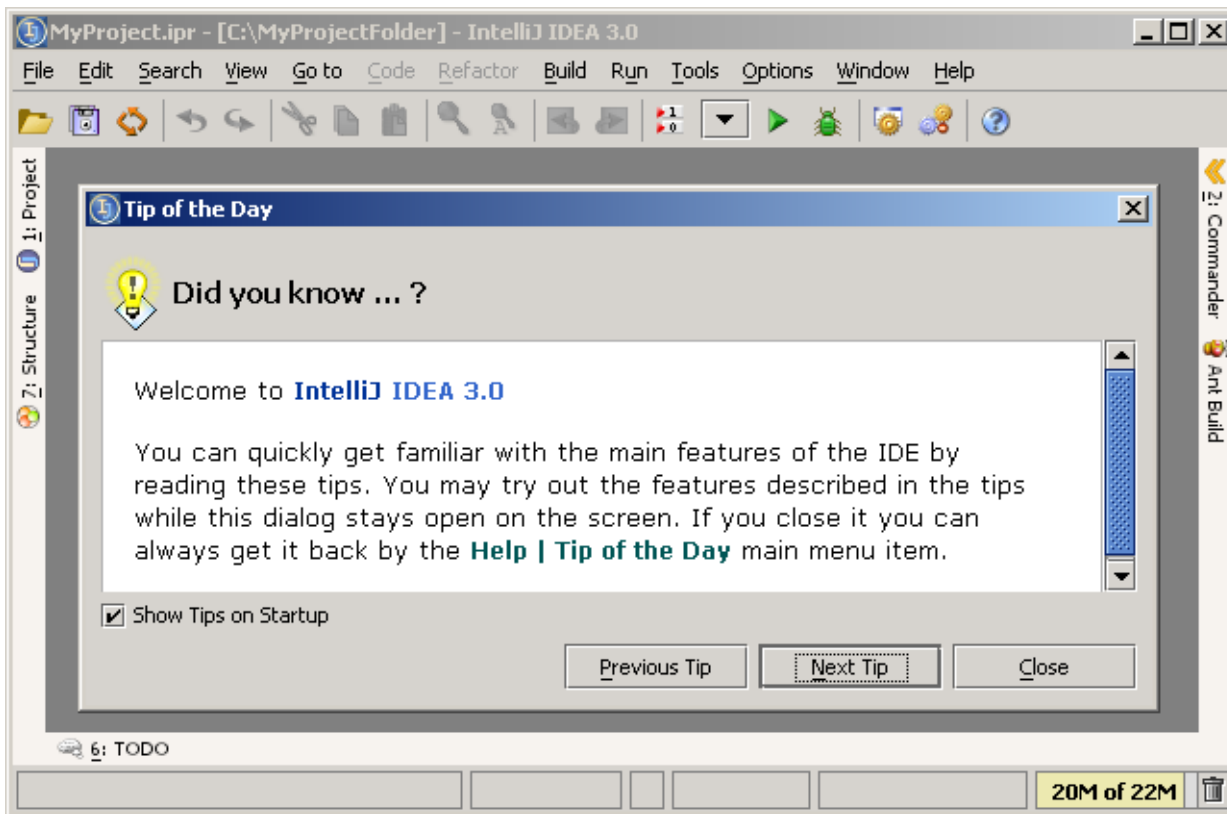


Figure 3.18. Main dialog and tip of the day (180)

3.27. Click **Close** to close the tip of the day.

3.28. Click on **Project** to open the project tool.

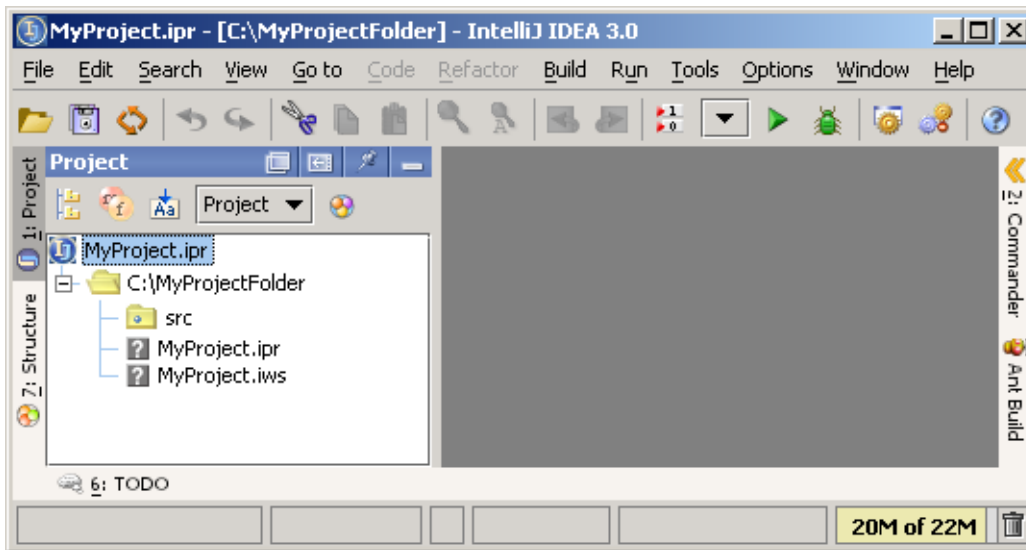


Figure 3.19. IDEA main dialog and project tool (591)

3.2. Create package

Packages (directories) will be discussed in more detail in [Chapter 5. Directories \(packages\) \(page 75\)](#).

[20021007TTT: why is the output dir not visible??](#)

3.29. In the Project tool: Right-click on folder `src`. A popup dialog appears.

3.30. Select **New | Package**. A dialog requesting the package name appears.

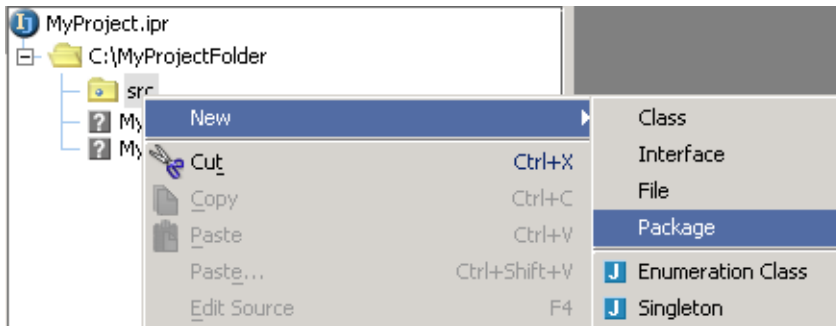


Figure 3.20. New package [\(574\)](#)

3.31. For the package name enter `myPackage`.

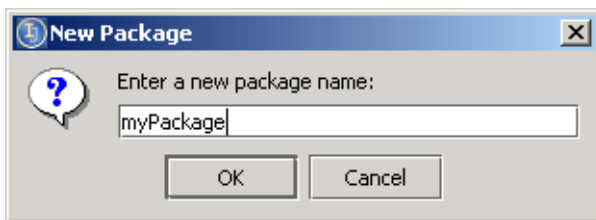


Figure 3.21. New package dialog [\(244\)](#)

3.32. Click **OK**. The package is created.

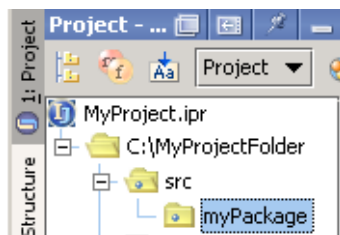


Figure 3.22. Created package [\(573\)](#)

3.3. Create class

Add about saving, resynch, etc.

You are now ready to create a Java source file.

*Creating class files will be discussed in more detail in **Chapter 6. Files (page 81)**.*

3.33. Right-click on **myPackage**.

3.34. From the context dialog select **New | Class**. A dialog requesting the class name appears.

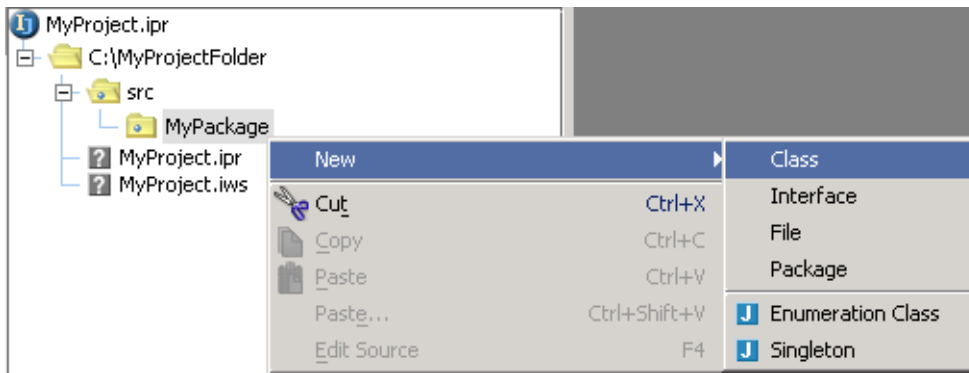


Figure 3.23. New class (243)

3.35. For the class name enter **MyClass**.

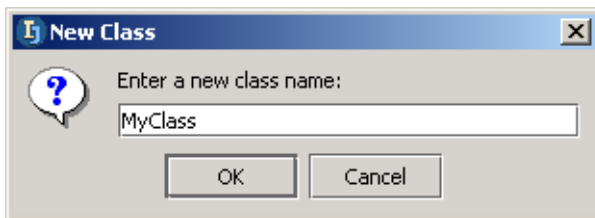


Figure 3.24. New class (242)

3.36. Click **OK**. A new class is created and opened in a text editor:

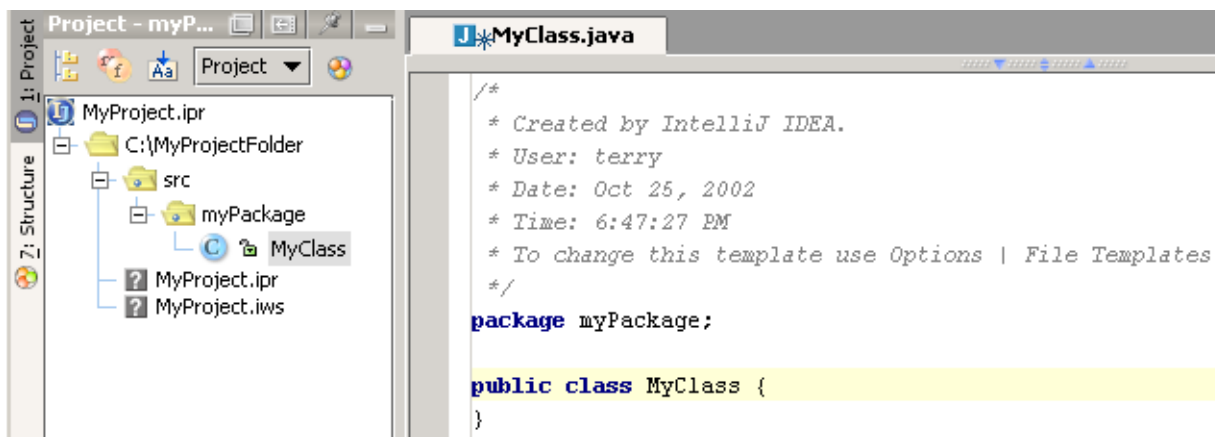
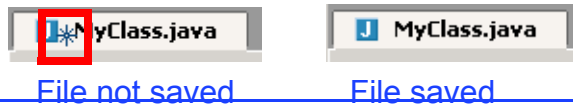


Figure 3.25. Class MyClass (572)

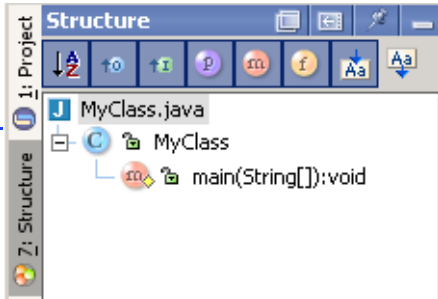
3.37. Press ~~CTRL-S~~. The file is saved.

~~Note the indicator for unsaved changes in a file.~~



~~Figure 3.26. Unsaved changes indicator (561.560)~~

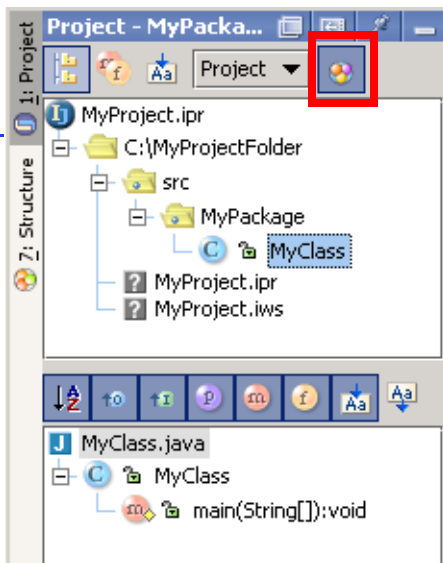
~~3.38. Click on the tab **Structure**. The structure of the class is displayed.~~



~~Figure 3.27. Class structure (559)~~

~~3.39. Click on tab **Project**.~~

~~3.40. Click on the icon . Both the Project and Structure panes are displayed.~~



~~Figure 3.28. Both Project and Structure panes are displayed (558)~~

3.4. Use the editor

You are now ready to use the IDEA editor to edit the Java source file:

- [3.4.1. Entering text \(page 41\)](#)
- [3.4.2. Find / Navigation \(page 42\)](#)
- [3.4.3. Colors and fonts \(page 43\)](#)
- [3.4.4. Code style \(page 44\)](#)
- [3.4.5. Error indication \(page 45\)](#)
- [3.4.6. Todo \(page 46\)](#)

Editing files will be described in more detail in 7. Editor X (page 107).

3.4.1. Entering text

This section demonstrates some of the basics of editing text with IDEA. Most things are very standard for most editors.

Entering text is described in more detail in 7.2. Text editing X (page 111).

3.41. Place the cursor (the “caret”) in line 2.

3.42. Click **CTRL-D** several times. The line is duplicated.

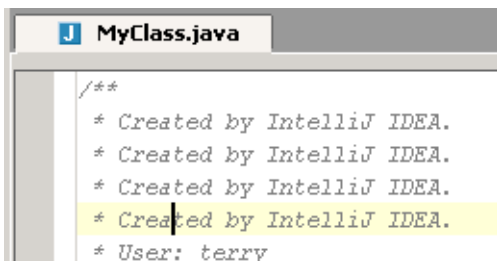


Figure 3.29. Duplicated lines [\(0002\)](#)

3.43. Use **CTRL-Y** to delete the added lines.

3.44. Experiment with the standard **CTRL-C**, **CTRL-X**, **CTRL-V** to copy, cut and paste text.

3.45. Press **CTRL-Z** multiple times to undo all of the changes you have made. Continue pressing until the following dialog appears:

3.46. Click **OK**. The class creation is undone.

3.47. Press **CTRL-SHIFT-Z** (redo). The question “Redo Create class?” appears.

3.48. Click **OK** to recreate the class.

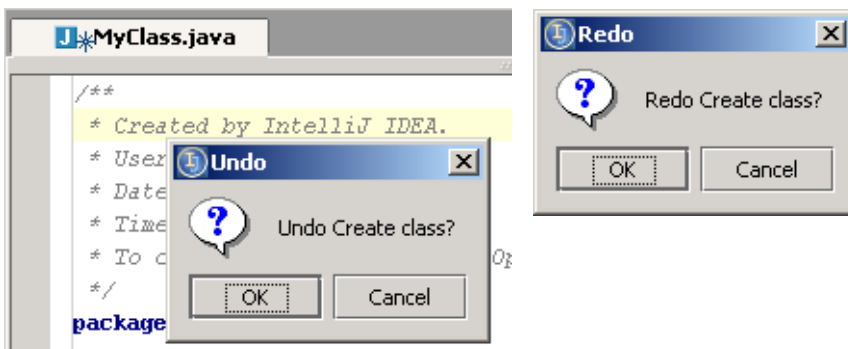


Figure 3.30. Undo / Redo create class [\(0003,0004\)](#)

3.4.2. Find / Navigation

This section demonstrates some of the basics of finding text, goto, and bookmarks.

Finding and navigation are described in more detail in 7.3. Find / Navigation (page 124).

3.49. Select **Search | Find**. The dialog “Find Text” appears.

3.50. For “Text to find” enter **my**.

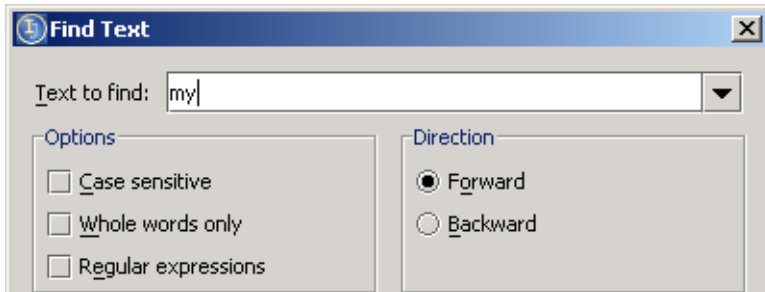


Figure 3.31. Find text (0005)

3.51. Click **Find**. An occurrence is found.

3.52. Click **F3** to find the next occurrences of the file.

Try options such as “Case sensitive”, “Whole words only”, “Regular expressions”.

3.53. Close **MyClass**.

3.54. Select **Goto | Class**.

3.55. For class name enter **MyClass**.

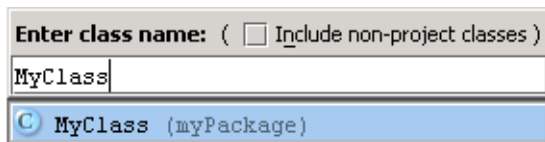


Figure 3.32. Goto class (0006)

3.56. Press **Enter**. MyClass is opened in an editor.

3.57. Place the cursor on a line in MyClass.

3.58. Select **Edit | Toggle bookmark**. A bookmark is added.

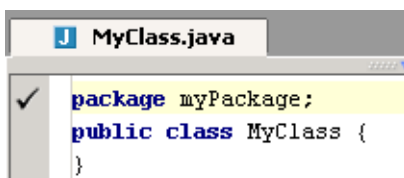


Figure 3.33. Added bookmark (0007)

3.59. Close MyClass.

3.60. Select **Edit | Show bookmarks**. The “Editor bookmarks” appears.

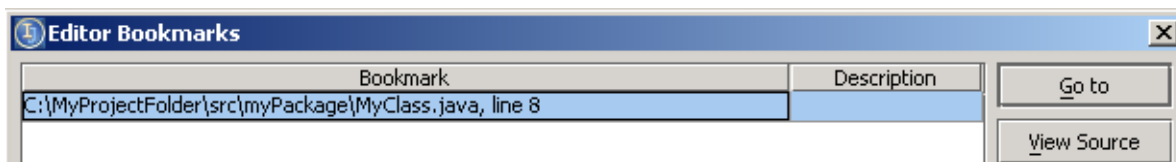


Figure 3.34. List of bookmarks (0008)

3.61. Double-click on the bookmark to open the file at the bookmark.

3.4.3. Colors and fonts

This section shows how the colors and fonts make code easier to understand.

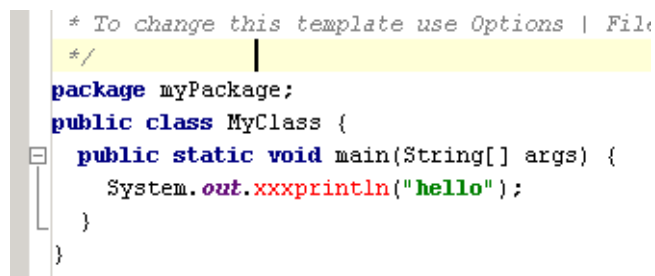
Colors and fonts are described in more detail in 7.4. Colors and fonts X (page 137).

3.62. Add the following to definition of MyClass

```
package myPackage;
public class MyClass {
    public static void main(String[] args) {
        System.out.println("hello");
    }
}
```

Note the colors and fonts

[20021028TTT find a better example and describe details.](#)

A screenshot of the IntelliJ IDEA code editor. The code is displayed with syntax highlighting: package names are in blue, keywords like 'public', 'class', 'static', and 'void' are in bold black, and the string 'hello' is in green. The code is as follows:

```
* To change this template use Options | File
*/
package myPackage;
public class MyClass {
    public static void main(String[] args) {
        System.out.println("hello");
    }
}
```

Figure 3.35. Colors and fonts (0009)

3.4.4. Code style

This section shows how code style makes code easier to understand.

Code style is described in more detail in 7.5. Code style X (page 145).

Like most editors, the IDEA editor makes it easy to maintain code style as you type. However, there are also functions for cleaning up the style of the code.

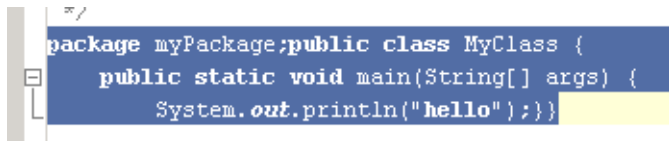
3.63. Modify the code for MyClass

```
package myPackage;public class MyClass {  
public static void main(String[] args) {  
System.out.println("hello");}}
```

Though error-free, the code is not easy to read.

3.64. Select the added code.

3.65. Select **Code | Auto-indent lines**. The lines are indented.



```
package myPackage;public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("hello");}}
```

Figure 3.36. Autoindented lines (0010)

3.66. Select **Tools | Reformat code**. The dialog “Reformat code” appears.

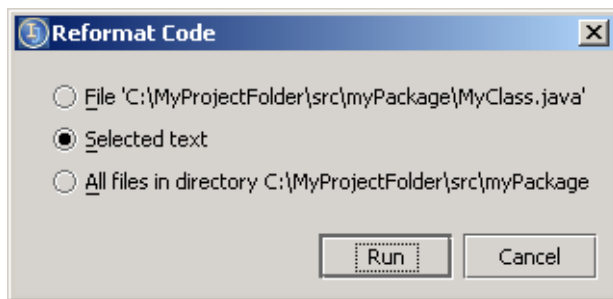
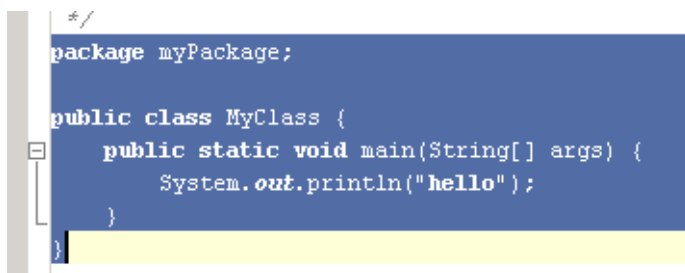


Figure 3.37. Reformat code (0010)

3.67. Click **Run**. The code is reformatted.



```
package myPackage;  
  
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("hello");  
    }  
}
```

Figure 3.38. Reformatted code (0012)

3.4.5. Error indication

This section shows how IDEA notifies you of errors while you type.

Error indication is described in more detail in 7.6. Error indication X (page 156)

3.68. Modify the code for MyClass

```
package myPackage;  
public xxclass MyClass {}
```

Note the indication of the error.

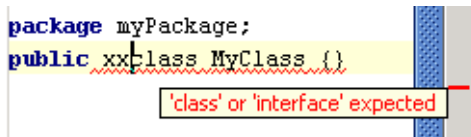


Figure 3.39. Error indication (0015)

3.69. Modify the code for MyClass

```
package myPackage;  
import java.util.*;  
public class MyClass {}
```

Though error-free, the import is not necessary.

3.70. Select the added code.

3.71. Select **Code | Auto-indent lines**. The lines are indented.

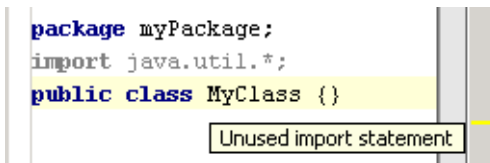


Figure 3.40. Unused import warning (0013)

3.72. Place the cursor at the end of the import statement. Note that a light-bulb figure appears.

3.73. Click on the suggestion icon to display the recommended solution for the problem.

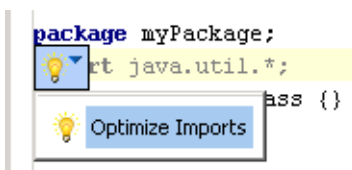


Figure 3.41. Optimize imports suggestion (0014)

3.74. Double-click on **Optimize imports**. The import statement is deleted.

3.4.6. Todo

This section shows how you can create “todo” messages.

ToDo is described in more detail in 7.7. Todo (page 160)

3.75. Modify the code for MyClass

```
package myPackage;  
//@todo add code  
public class MyClass {}
```

3.76. Close MyClass.

3.77. Select **Window | ToDo**. The TODO tool opens.

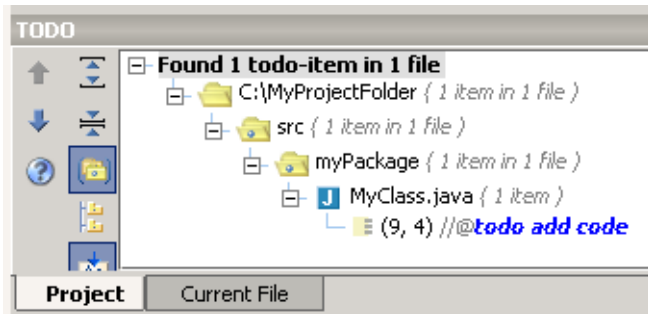


Figure 3.42. TODO tool (0016)

3.78. Double-click on the todo item. File MyClass.java is opened in an editor.

3.5. Automate code

This section shows how IDEA can automatically create code.

Code automation will be described in more detail in 8. Code Automation (page 163).

3.79. Enter the text “psvm” as shown in the diagram below:

```
public class MyClass{  
    psvm;
```

Figure 3.43. Enter text “psvm” (571)

3.80. Press the **TAB** key. Note that text is automatically entered:

```
public class MyClass{  
    public static void main(String[] args) {  
    }  
}
```

Figure 3.44. “psvm” replaced with live template text (570)

Entering text “psvm” and then pressing the TAB key will cause IDEA to replace the text “psvm” with “public status void main(String[] args) { }”. This is an example of a **Live template**. Live templates are described in more detail in **Section 8.3. Code templates (page 184)**.

3.81. Enter the text “itar” as shown in the diagram below:

```
public class MyClass{  
    public static void main(String[] args) {  
        itar  
    }  
}
```

Figure 3.45. Enter text “itar” (569)

3.82. Press the **TAB** key. Live template text is added.

```
public class MyClass{  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            String arg = args[i];
```

Figure 3.46. itar live template text (568)

```
package myPackage;  
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            String arg = args[i];  
            System.out.println(i + 1 + " - " + arg);  
        }  
    }  
}
```

3.83. Modify the text as shown below:

```
public class MyClass{  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            }  
        }  
    }
```

Figure 3.47. Modified itar text [\(566\)](#)

3.84. Enter “**sout**” live template text as shown below:

```
public static void main(String[] args) {  
    for (int i = 0; i < 5; i++) {  
        sout  
    }  
}
```

Figure 3.48. sout [\(565\)](#)

```
public class MyClass{  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            System.out.println("");  
        }  
    }
```

Figure 3.49. sout live template text [\(564\)](#)

3.6. Refactor code

This section shows how code can be refactored with IDEA.

Code refactoring will be described in more detail in 9. Code Refactoring (page 201)

3.85. Change MyClass:

```
package myPackage;
public class MyClass {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("i = " + i);
        }
    }
}
```

3.86. Select var **i**.

3.87. Select **Refactor | Rename**. The dialog “Rename” appears.

3.88. Enter **iRenamed**.

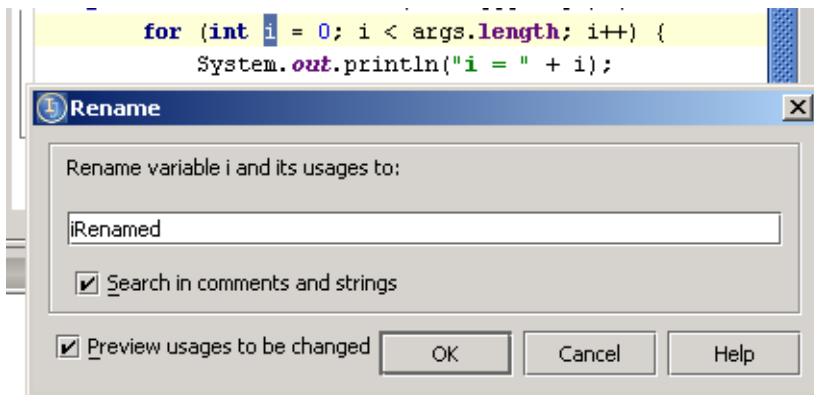


Figure 3.50. Rename variable (0017)

3.89. Click **OK**. The refactoring preview appears.

3.90. Click **Do Refactor**. The variable is renamed.

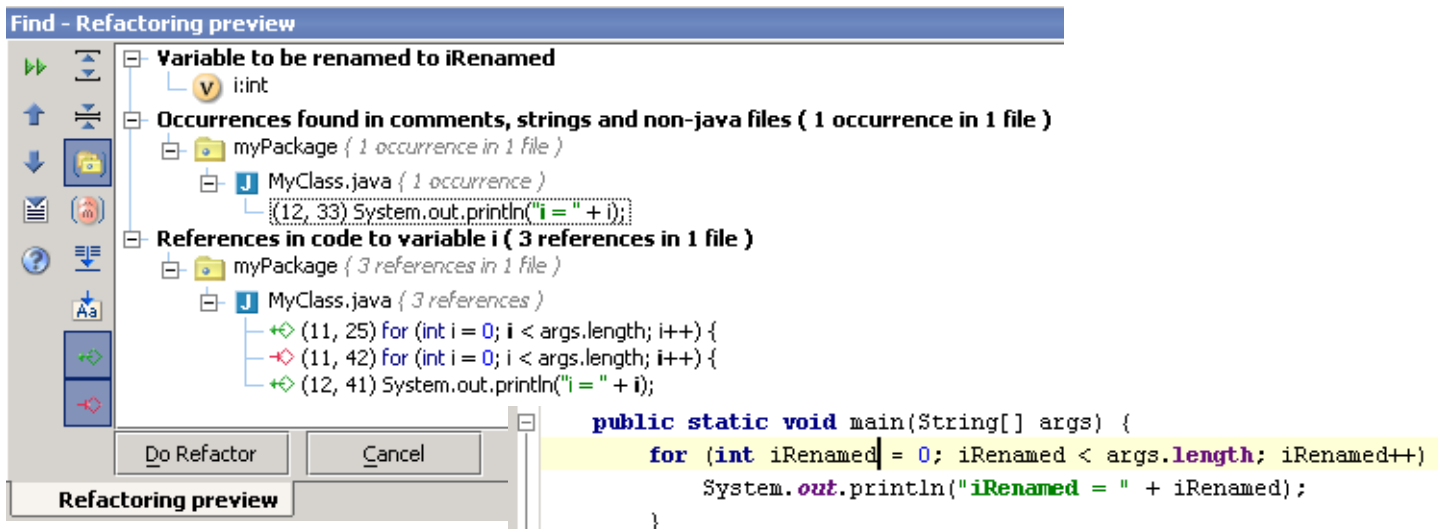


Figure 3.51. Refactoring preview / results (0018.0019)

3.7. Use version control

IDEA's built-in versioning tool makes it easy to recover previous versions.

Version control is described in more detail in 11. Version control (page 255).

3.91. Select **Tools | Local VCS | Show history**. The history tool appears.

3.92. Click on the action **Renaming variable i to iRenamed**.

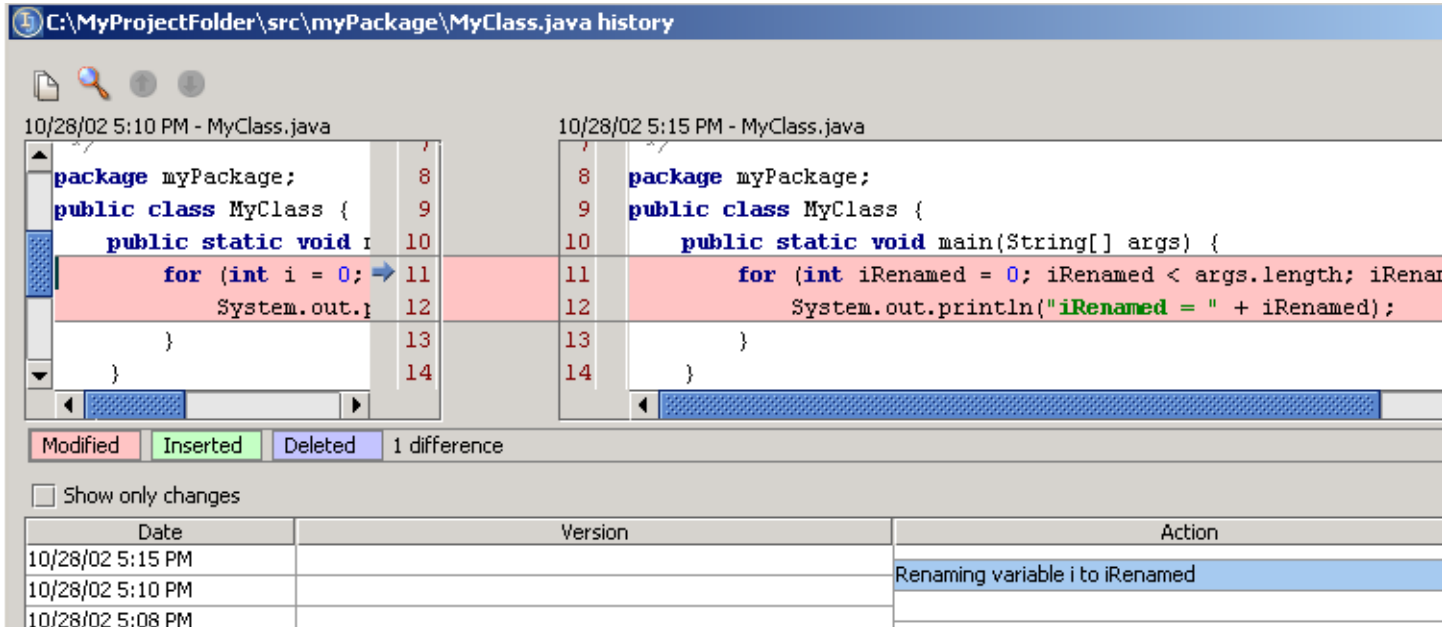


Figure 3.52. Renamed variable in history [\(0020\)](#)

3.93. Right-click on the action. A popup “Rollback” appears.

3.94. Click **Rollback**. A confirmation dialog appears.

3.95. Click **OK**. The change is rolled back.

```
package myPackage;
public class MyClass {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("i = " + i);
        }
    }
}
```

Figure 3.53. Rolled-back changes [\(0021\)](#)

3.8. View JavaDoc

IDEA's JavaDoc functionality makes it easy to create and work with JavaDoc.

JavaDoc is described in more detail in 12. Java Doc (page 269) which also includes a description of the installation of JDK javadoc and generation of own.

3.96. Place the cursor at the beginning of the line:

```
public static void main(String[] args) {
```

3.97. Enter `/**`.

3.98. Press **Enter**. The tags are entered:

```
public class MyClass {
  /**
   *
   * @param args
  */
  public static void main(String[] args) {
```

Figure 3.54. JavaDoc method tags (0022)

3.99. Place the cursor on **main**.

3.100. Click **Ctrl-Q**. Quick JavaDoc is displayed.

```
public class MyClass {
  /**
   *
   * @param args
  */
  public static void main(String[] args) {
    for myPackage.MyClass
  }
  public static void main(String[] args)
}
Parameters:
  args -
```

Figure 3.55. Quick JavaDoc for main (0023)

3.101. Select **Tools | Generate JavaDoc**. The dialog "Generate JavaDoc" appears.

3.102. For "Output directory" enter **C:\MyProjectFolder**.

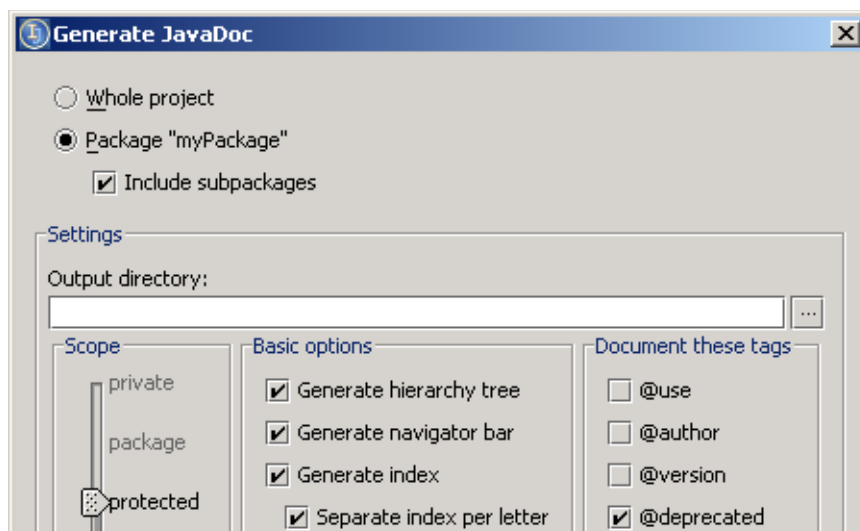


Figure 3.56. Quick JavaDoc for main (0023)

3.103. Click **Start**. JavaDoc is generated.

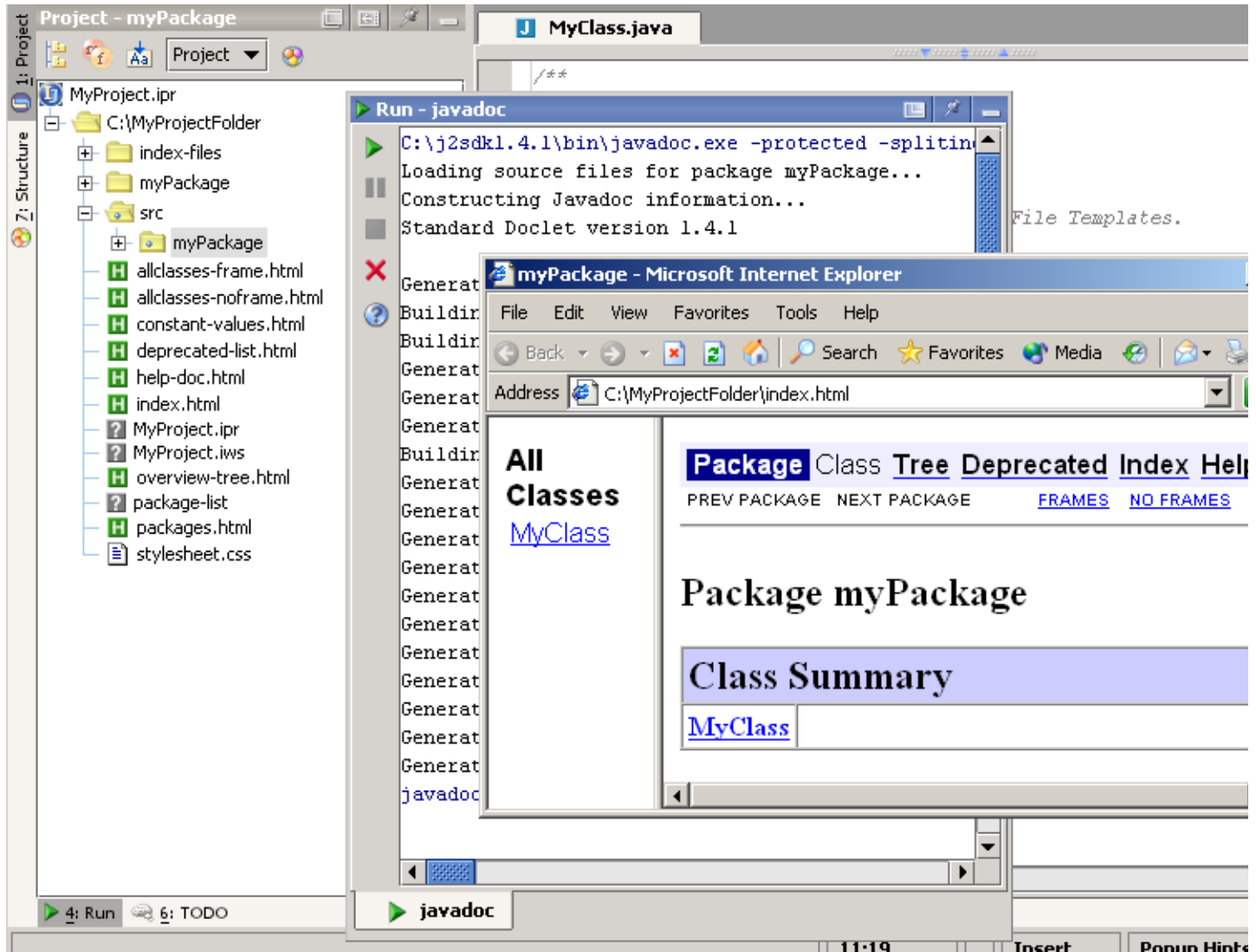


Figure 3.57. Generated JavaDoc for MyClass (0025)

3.9. Compile

You are now ready to compile the Java source file.

Compilation is described in more detail in Chapter 13. Compiler (page 283).

3.104. Modify **MyClass** (add an error)

```
package myPackage;
public class MyClass {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            System.out.println("i = " - i);
        }
    }
}
```

3.105. From the main menu select **Build | Compile “MyClass.java” Ctrl-Shift-F9**. A message pane appears with a description of the error.

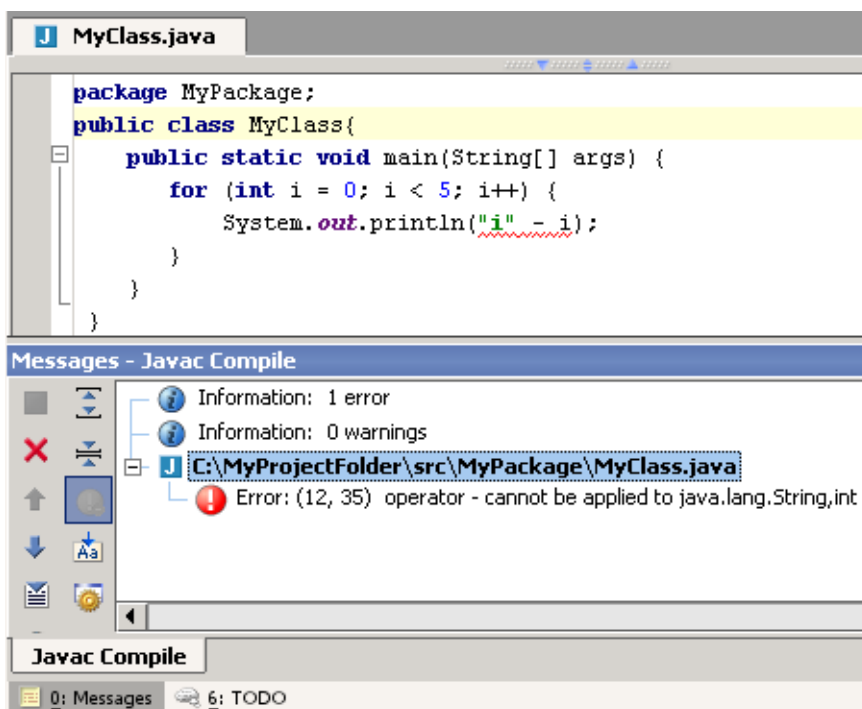


Figure 3.58. Message pane with error information (555)

3.106. Correct the error (change “-” to “+”).

3.107. Recompile. The file **C:\MyProjectFolder\MyOutputFolder\myPackage\MyClass.class** is created.

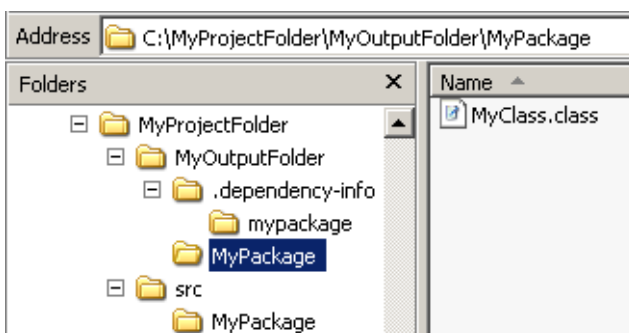


Figure 3.59. Compile file (241)

3.10. Debug

You are now ready to debug the application. To debug the application, you will

- [3.10.1. Create application configuration \(page 54\)](#)
- [3.10.2. Run the configuration \(page 56\)](#)
- [3.10.3. Set breakpoint \(page 57\)](#)
- [3.10.4. Step through application \(page 58\)](#)

Running/Debugging is described in more detail in Chapter 15. Debugger (page 301).

3.10.1. Create application configuration

3.108. From the main menu select **Run | Run Shift-F10**. Dialog **Run** tab **Application** appears.

3.109. Press the **Add** button . The application name “Unnamed” appears.

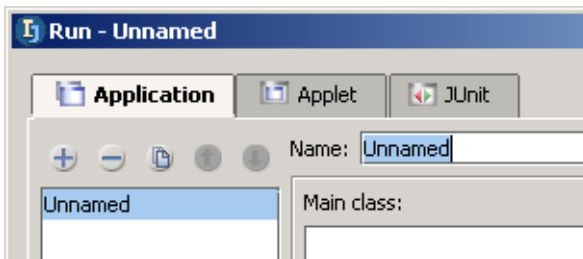


Figure 3.60. Application “Unnamed” [\(553\)](#)

3.110. Change Name to **MyApplication**.

3.111. Click on the ellipsis button for **Main class**:. The dialog “Choose Main Class” appears.

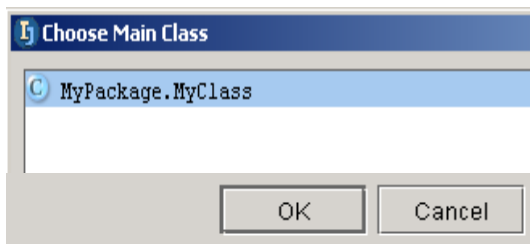


Figure 3.61. Choose main class [\(552,551\)](#)

3.112. Select **myPackage.MyClass**.

3.113. Click **OK**.

[20020906TTT: ?? what is the working directory used for?](#)

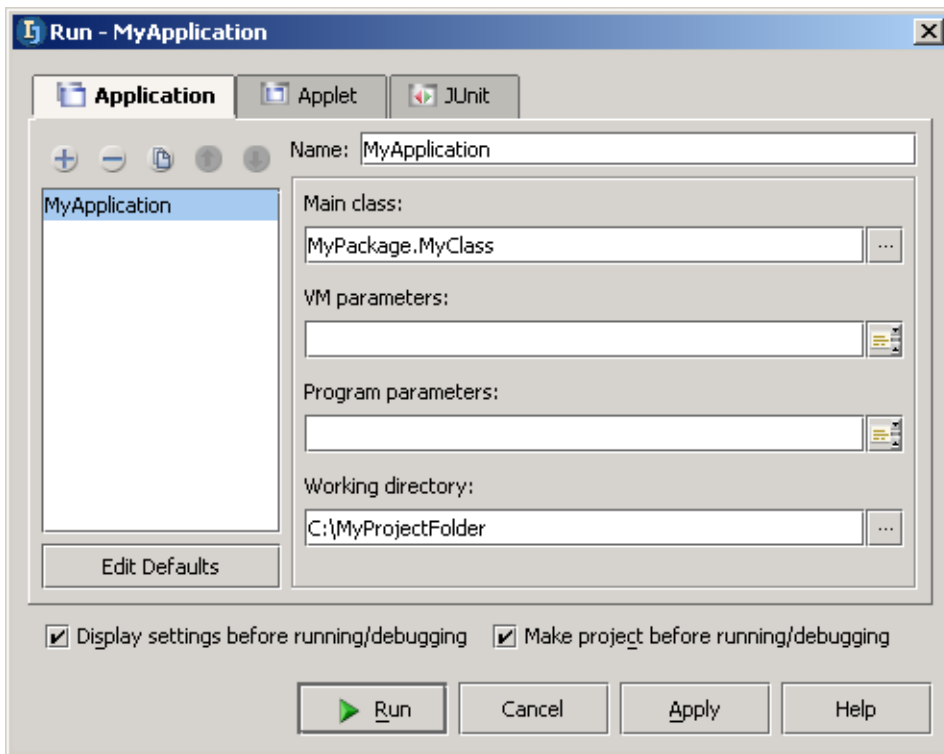


Figure 3.62. Application settings (550)

3.10.2. Run the configuration

3.114. Click **Run**. The pane “Run” appears with the program output.

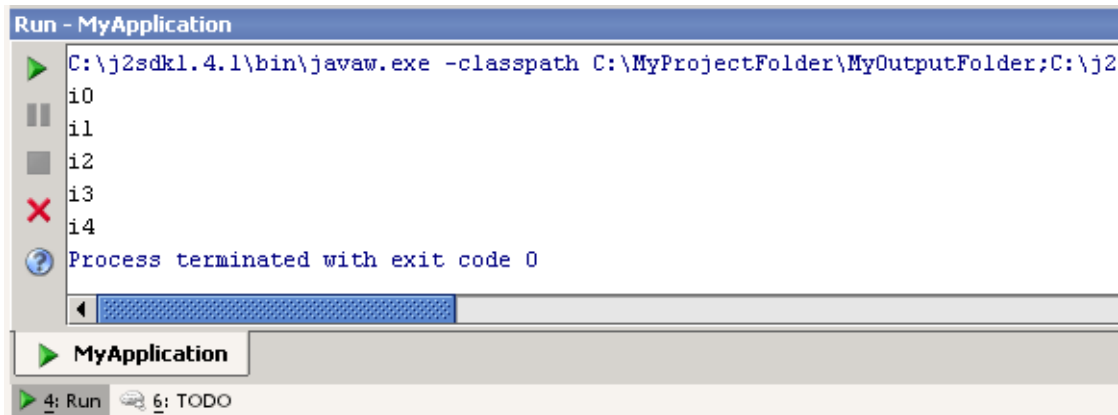


Figure 3.63. Run pane output (549)

3.115. Run with args.

```
package myPackage;  
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            String arg = args[i];  
            System.out.println(i + 1 + " - " + arg);  
        }  
    }  
}
```


3.10.3. Set breakpoint

3.116. Click to the left of the line “System.out.println...”. A line breakpoint marker appears.

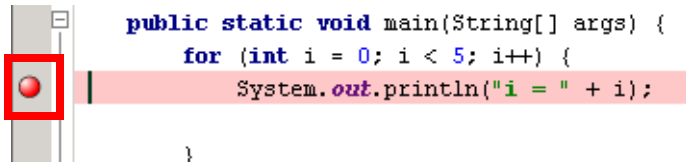


Figure 3.64. Set a line breakpoint (548)

3.117. Right-click on the marker.

3.118. From the context menu select **Properties**. The dialog **Breakpoints** appears.

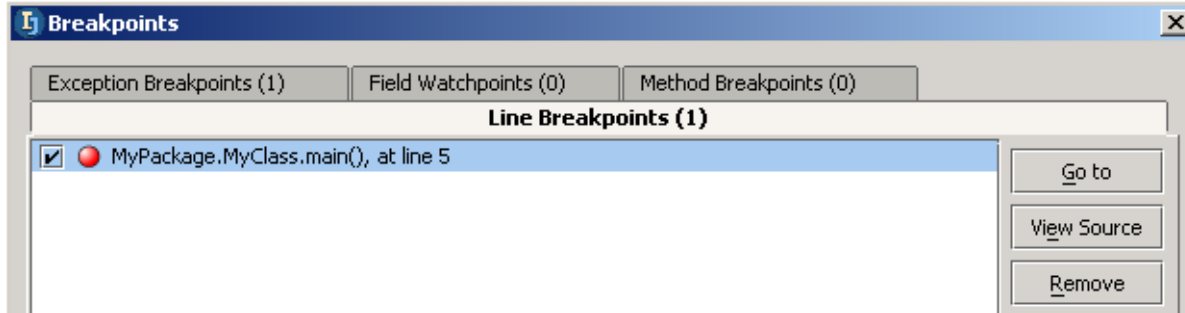


Figure 3.65. Dialog breakpoints (547)

Note: To open this dialog from the main menu select **Run / View Breakpoints Ctrl-Shift-F8**.

3.119. Close the dialog.

3.10.4. Step through application

3.120. Click on the debug icon  (or from the main menu select **Run / Debug Shift-F9**). The dialog “Debug - MyApplication” appears.

3.121. Click **Debug**. The application starts and halts at the breakpoint. The debug pane is displayed.

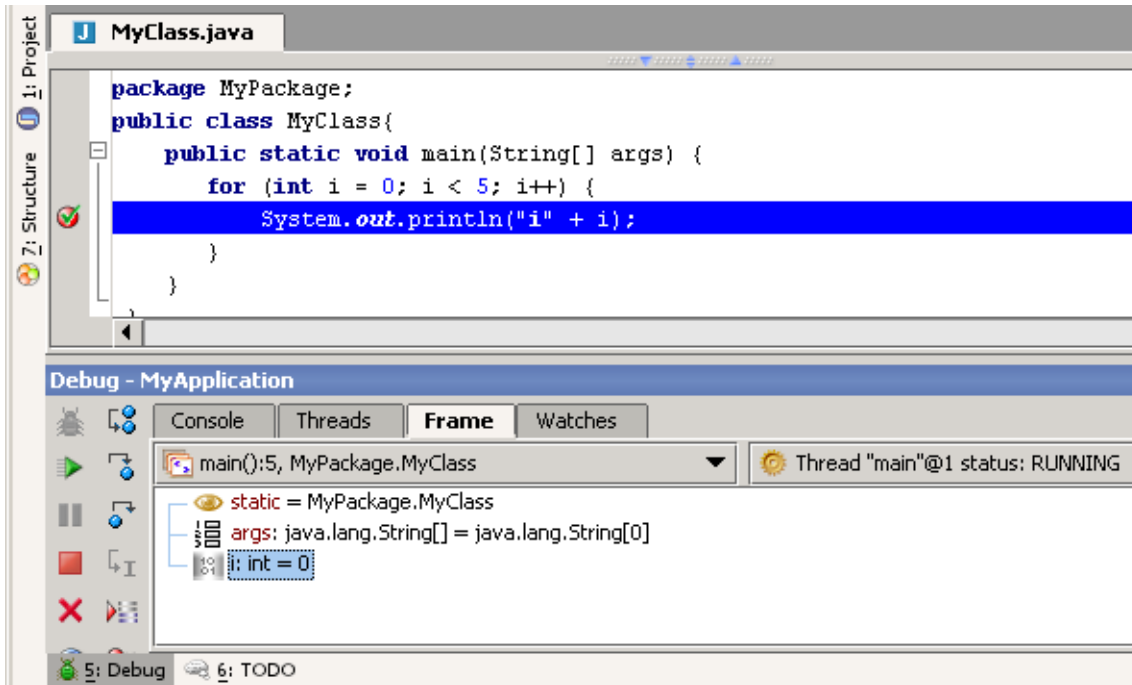



Figure 3.66. Debug pane [\(545\)](#)

3.122. Click on the icon **Step Over**  (or from the main menu select **Run | Step over F8**). The method is stepped over and the current line is highlighted in blue.

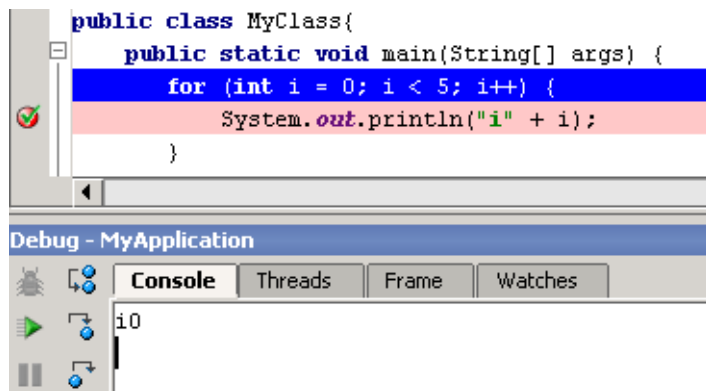


Figure 3.67. Step over [\(543\)](#)

3.123. Click on the icon **Resume**  (or from the main menu select **Run | Resume program F9**). Program execution halts at the breakpoint.

3.124. Right-click on the breakpoint marker.

3.125. Select **Disable**. The breakpoint marker is changed.

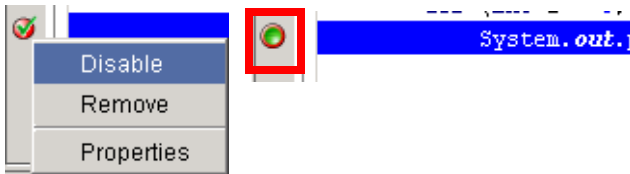


Figure 3.68. Disable breakpoint ([541,540](#))

3.126. Click **Resume**. The application finishes.

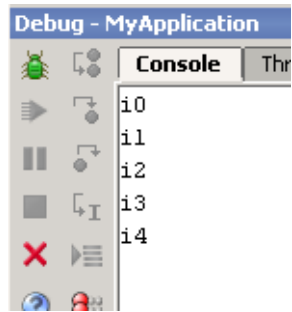


Figure 3.69. Application finishes ([539](#))

Part B. Projects / dirs / files

[20021029TTT last edit.](#)

The chapters in this part describe the extensive functionality IDEA provides for creating and managing

4. Projects (page 63). Demonstrates how IDEA project concepts make it as simple as possible to manage a software project.

5. Directories (packages) (page 75). Demonstrates how directories and packages can be created and managed within an IDEA project.

6. Files (page 81). Demonstrates how IDEA makes it as simple as possible to create and manage any type of file.

4. Projects

[20021030TTT: text added. some text taken from help. contact: valentin.](#)

There are 2 main groups of settings for your workspace:

- IDE settings
- Project properties

IDEA settings apply for the development workstation and do not vary for each project. To get an idea of what these settings are, look at the tabs in the dialog “IDEA Settings” (select **Options | IDEA Settings** to open):

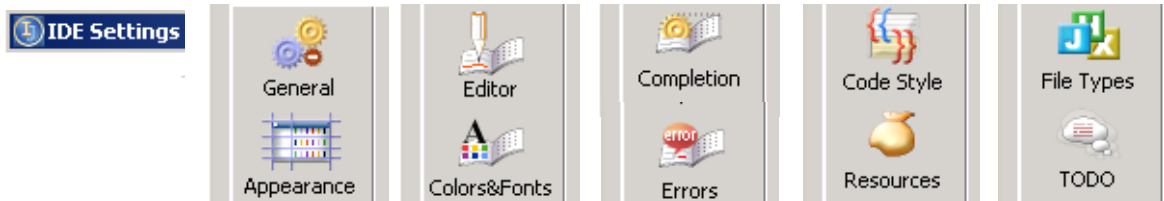


Figure 4.1. IDEA settings tab icons ([a003.a001.a002](#))

Project properties are specific to each project. To get an idea of what these settings are, look at the tabs in the dialog “Project Properties” (select **File | Project Properties** to open):



Figure 4.2. Project properties settings tab icons ([a004.a005.a006](#))

A project is specified by the

- **Project name**
- **Project location** (directory where all project files are stored)

The name and location specified above becomes the name and location of the project .ipr file (an XML file that contains all project settings)

The most basic project settings include the following:

- **Target JDK**
- **Paths:**
 - Output path
 - Project path
 - Source path
 - Class path

Thus, when you created a project in the quick start (3.1. Create project (page 32)) you did the following:

- **3.1.1. Specify project name / location (page 32)**
- **3.1.2. Specify target JDK (page 33)**
- **3.1.3. Specify project paths (page 35)**

In this chapter you will learn more of the basics for working with projects:

- **4.1. Basic operations (page 64).** Describes how create, open, reopen and close projects.
- **4.2. View / Modify settings (page 66).** Describes the various views of a project.

4.1. Basic operations

?? can you copy or delete projects??

This section describes how to

- Create a **New (New Project Wizard)** project
- **Open** an existing project
- **Close** an open project
- **Reopen** a closed (but previously opened) project
- **Delete** a project

4.1.1. New (New Project Wizard)

You already created a new project in [3.1. Create project \(page 32\)](#).

4.1.2. Open

To open a project, open the project `.ipr` file.

4.1. Select **File | Open project**. The dialog “Open project” appears.

4.2. Double-click on the **project (.ipr) file** that you want to open.

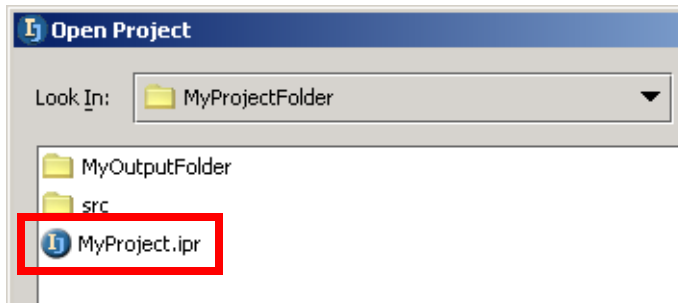


Figure 4.3. Open project `.ipr` file [\(617\)](#)

Note: If a project is already open in IDEA, then you will be asked whether to open the project in a new frame:

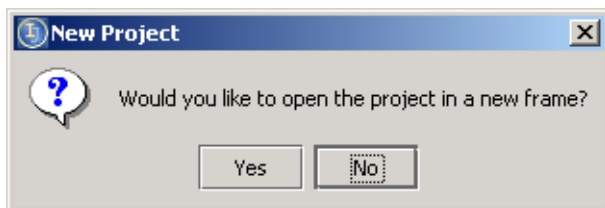


Figure 4.4. “Open in new frame?” [\(a007\)](#)

If you answer yes, then a new frame will appear.



Figure 4.5. New frame [\(a008\)](#)

4.1.3. Close

4.3. Select **File | Close Project**.

4.1.4. Reopen

4.4. Select **File | Reopen | (project .ipr file)**.

4.1.5. Delete

To delete a project, simply delete the project directory (for example `c:\MyProjectFolder`).

4.2. View / Modify settings

You can view all project properties in the

- **4.2.1. Project properties dialog (page 66)**

The project paths can also be modified in the

- **4.2.2. Project tool (page 72)**

Many project settings can also be modified in

- **4.2.3. Other dialogs (page 73)**

Although not recommended for making changes, all project properties are also specified in the

- **4.2.4. Project xml file (.ipr) (page 73)**

4.2.1. Project properties dialog

Dialog “Project Properties” provides access to all project settings.

The most important tab when creating projects is the

- **4.2.1.1. Paths tab (page 66)**

Other settings that you will become acquainted with throughout this tutorial are in the

- **4.2.1.2. Other tabs (page 71)**

4.2.1.1. Paths tab

The tab “Paths” specifies the project paths. You already specified project paths with the help of the new project wizard in section 3.1. **Create project (page 32)**. This section shows how to change these settings in tab “Paths” (without the wizard).

4.5. Select **File | Project properties...** The dialog “Project properties” tab “Paths” is opened.

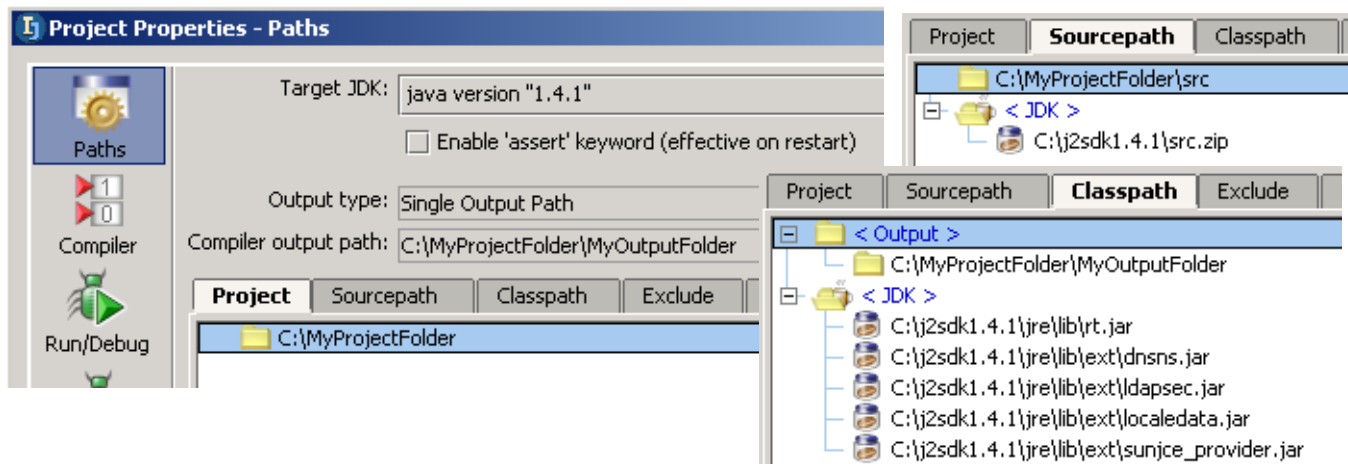


Figure 4.6. Project properties tab Paths ([468,467,466](#))

The following settings are shown:

- **4.2.1.1.1. Target JDK (page 67)**
- **4.2.1.1.2. Output path (page 68)**
- **4.2.1.1.3. Directory (page 69)**
- **4.2.1.1.4. Sourcepath (page 70)** (directory, jar, zip) or
- **4.2.1.1.5. Classpath (page 71)** (directory, jar)

4.2.1.1.1. Target JDK

This section demonstrates how to add and select a different target JDK (note that this example, for the sake of simplicity, uses the same installed JDK).

You have already done this with the wizard (see [3.1.2. Specify target JDK \(page 33\)](#)).

4.6. Display the drop-down list for “Targe JDK”. Note that only 1 target JDK is available.

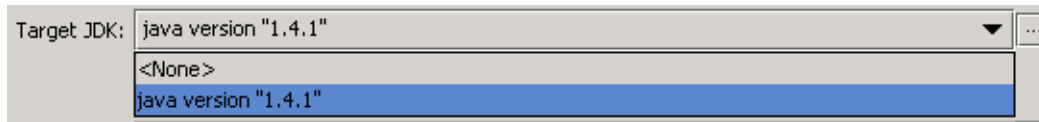



Figure 4.7. Folder added to project ([a011](#))

4.7. Click on the **Target JDK** ellipsis button (). The dialog “Configure JDKs” appears.

4.8. Click on the **Create JDK insert** button (). The dialog “Select JDK Home directory” appears.

4.9. Select **j2sdk1.4.1**.



Figure 4.8. Select JDK home directory ([a014](#))

4.10. Click **OK**. The configured JDK “java version 1.4.1 (1)” is shown.

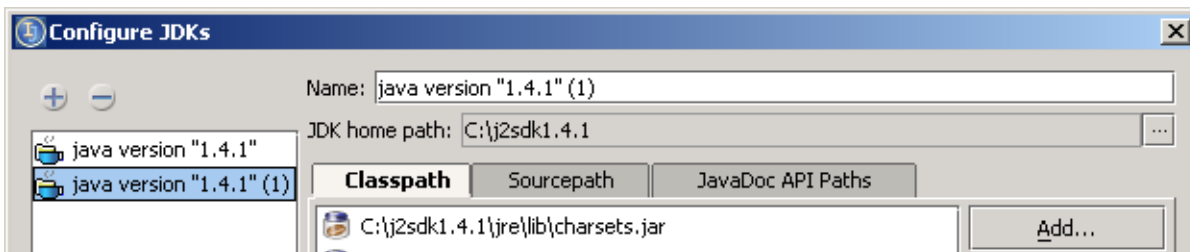


Figure 4.9. Configured JDK ([a015](#))

4.11. Click **OK**. The new target JDK is shown

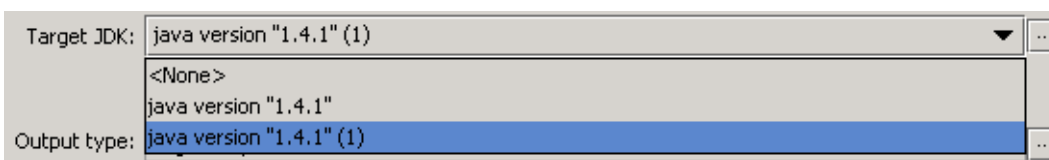



Figure 4.10. New target JDK ([a016](#))

4.2.1.1.2. Output path

This section demonstrates how to create and select a different output path.

You have already done this with the wizard (see [3.1.3.1. Output path \(page 35\)](#)).

4.12. Click on the **compiler output path** ellipsis button (). The dialog “Select path” appears.

4.13. Create dir **C:\MyProjectFolder\MyOutputFolder2**.

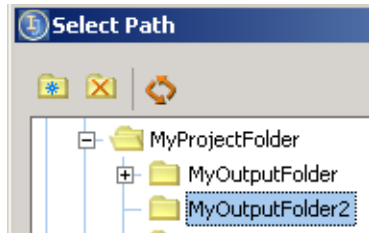


Figure 4.11. New output folder ([a017](#))

4.14. Click **OK**. The new compiler output path is shown.

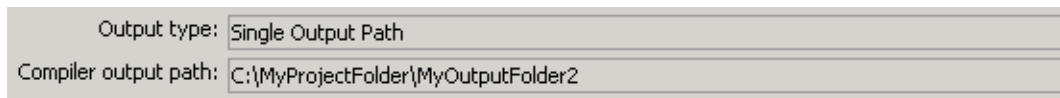


Figure 4.12. New compiler output path ([a020](#))

4.2.1.1.3. Directory

You can

- **Add**
- **Move**
- **Remove**

a directory.

4.2.1.1.3.1. Add

4.15. Select tab **Project**.

4.16. Click **Add**. The dialog “Select Path” appears.

4.17. Create folder **C:\MyProjectFolder2**.

4.18. Click **OK**. The folder is added (to the end of the list).

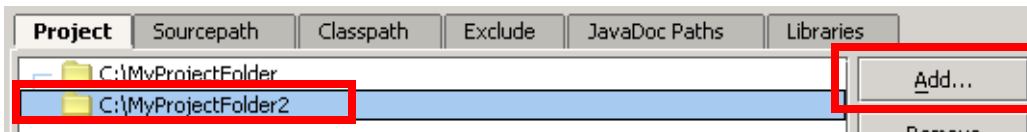


Figure 4.13. Folder added to project (612)

4.2.1.1.3.2. Move

4.19. Click on the added folder.

4.20. Click **Move up** and **Move down**. The location of the folder is changed.

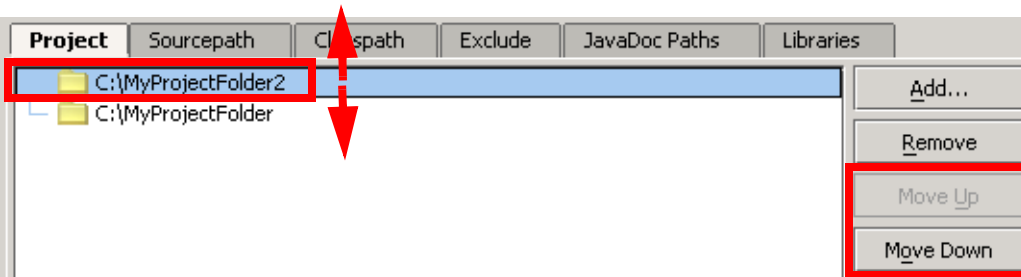


Figure 4.14. Folder moved (up and down) (613)

4.2.1.1.3.3. Remove

4.21. Click on **Remove** to remove from the project (not delete from the hard disk) the selected folder.

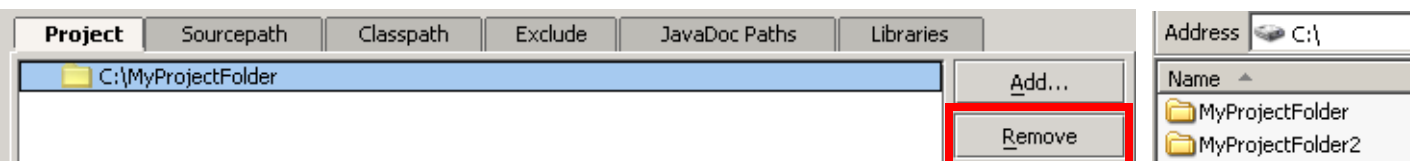


Figure 4.15. Folder removed (from project) (a018,a019)

4.2.1.1.4. Sourcepath

If the sources for classes are supplied as a

- jar (or zip) or
- directory of files

then you can

- **Add**
- **Move**
- **Remove**

them to/within/from the Sourcepath.

4.2.1.1.4.1. Add

4.22. Select the tab **Sourcepath**.

4.23. Select **Add**. The “Select Path” dialog appears.

4.24. Select the jar or dir.

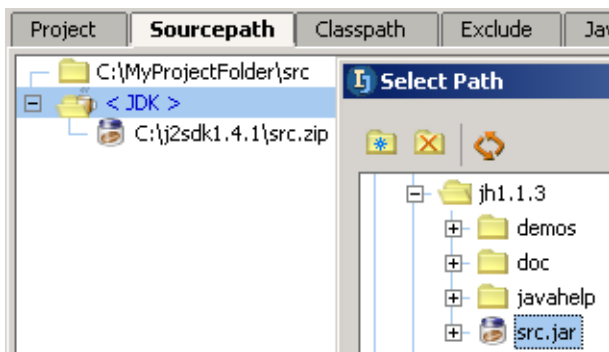


Figure 4.16. Select folder or jar (240)

4.25. Click **OK**. The jar or folder is added (to the end of the list).

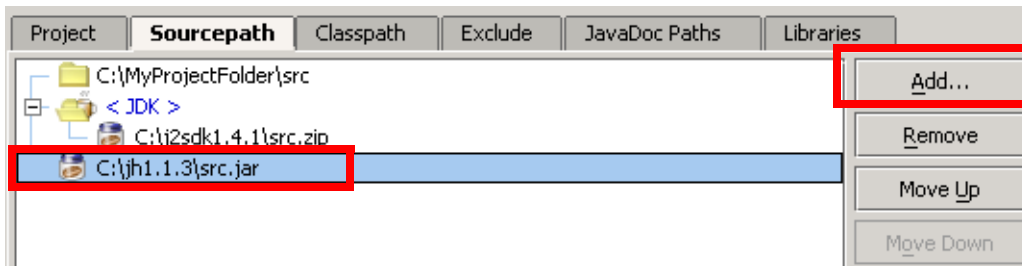


Figure 4.17. Folder added to project (615)

4.2.1.1.4.2. Move

Click to move up or down.

4.2.1.1.4.3. Remove

Click to remove (not delete from the file system).

4.2.1.1.5. Classpath

If the sources for classes are supplied as a

- jar or
- directory of files

then then you can

- **Add**
- **Move**
- **Remove**

them to the Classpath.

4.2.1.1.5.1. Add

4.26. Select the tab **Classpath**.

4.27. Select **Add**. The “Select Path” dialog appears.

4.28. Select **C:\MyProjectFolder\MyOutputFolder**.

4.29. Click **OK**. The directory is added to the classpath.

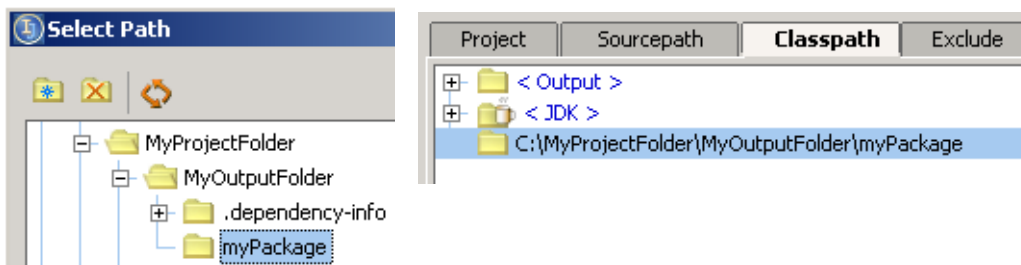


Figure 4.18. Added to classpath ([a021.a022](#))

4.30. Double-click on **MyClass** in the Project tool Classpath.

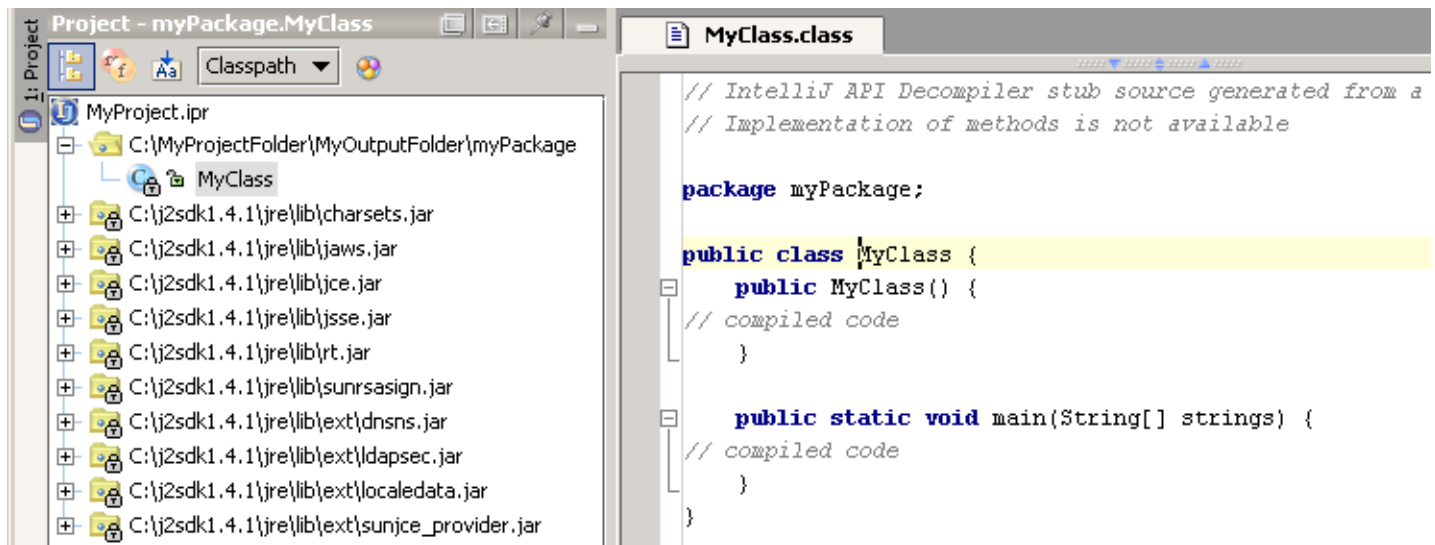


Figure 4.19. Class file opened in project tool ([a023](#))

4.2.1.1.5.2. Move

Click to move up or down.

4.2.1.1.5.3. Remove

Click to remove (not delete from the file system).

4.2.1.2. Other tabs

The settings in the other tabs will be introduced throughout this tutorial.

4.2.2. Project tool

In the project tool you can

- **Display paths**
- **Remove from paths** (directories, jars, files, etc.)

4.2.2.1. Display paths

4.31. In the frame: Click on the tab **Project**. The project path is displayed.

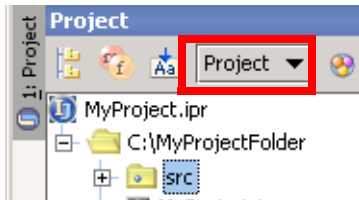


Figure 4.20. Project info in the project tool window (588)

4.32. Select from the drop-down list (with "Project" currently selected) **Sourcepath**. The source path info is displayed.

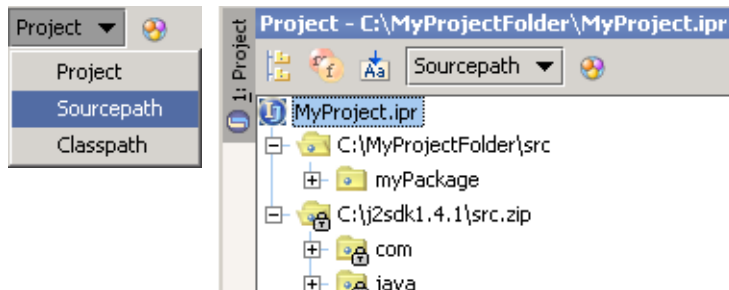


Figure 4.21. Sourcepath (575,587)

4.33. Select from the drop-down list **Classpath**. The classpath info is displayed.

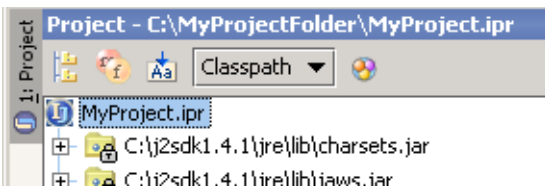


Figure 4.22. Classpath info in main dialog (586)

4.2.2.2. Remove from paths

4.34. In the classpath: Right-click on **C:\MyProjectFolder\MyOutputFolder\myPackage**.

4.35. Select **Remove from classpath**. A confirmation dialog appears.

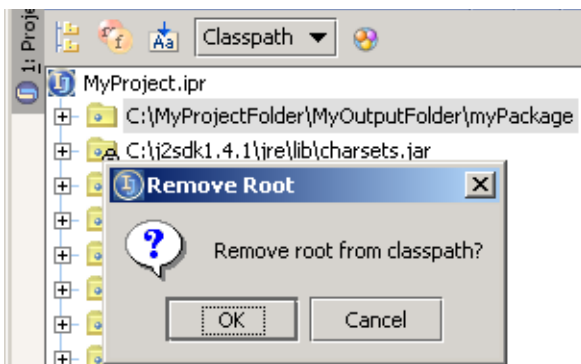


Figure 4.23. Removed from classpath (a024)

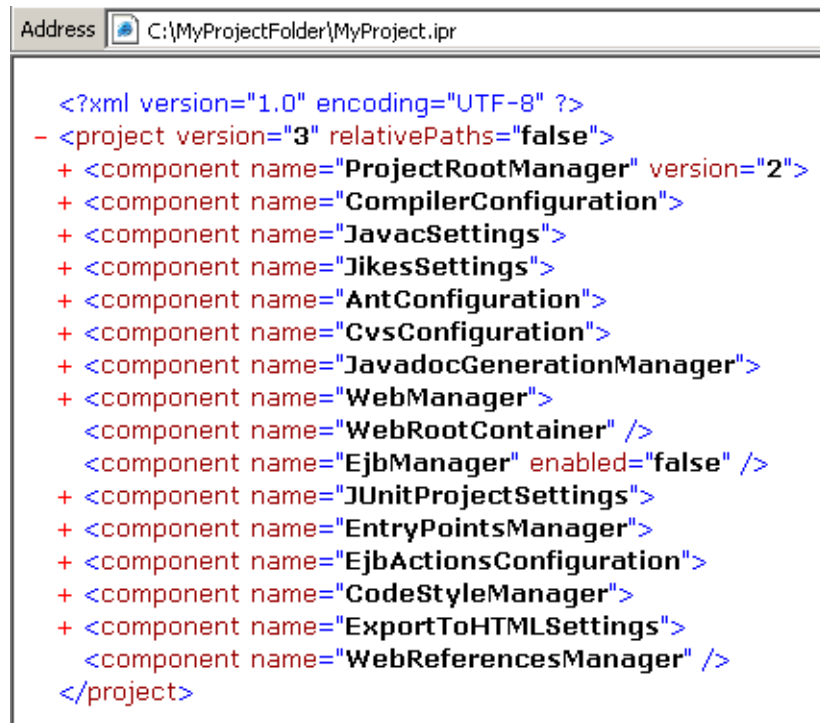
4.36. Click **OK**. The directory is removed from the classpath.

4.2.3. Other dialogs

Many of the project settings can be modified in other (ie, not the project properties) dialogs. You will see this throughout the rest of this tutorial.

4.2.4. Project xml file (.ipr)

4.37. Open **C:\MyProjectFolder\MyProject.ipr** in a web browser.



```
Address C:\MyProjectFolder\MyProject.ipr
<?xml version="1.0" encoding="UTF-8" ?>
- <project version="3" relativePaths="false">
+ <component name="ProjectRootManager" version="2">
+ <component name="CompilerConfiguration">
+ <component name="JavacSettings">
+ <component name="JikesSettings">
+ <component name="AntConfiguration">
+ <component name="CvsConfiguration">
+ <component name="JavadocGenerationManager">
+ <component name="WebManager">
  <component name="WebRootContainer" />
  <component name="EjbManager" enabled="false" />
+ <component name="JUnitProjectSettings">
+ <component name="EntryPointsManager">
+ <component name="EjbActionsConfiguration">
+ <component name="CodeStyleManager">
+ <component name="ExportToHTMLSettings">
  <component name="WebReferencesManager" />
</project>
```

Figure 4.24. Project .ipr files ([590 \(iws589\)](#))

This file specifies all project properties. Note, however, that it is not recommended to edit this file directly.

5. Directories (packages)

20021030TTT updated.

contacts: valentin

to select: view \ select in , popup dialog. ?? why is "hierarchy" not in the list in the popup??

This chapter introduces the basics for managing directories (packages) within IDEA:

- **5.1. Basic operations (page 75)** describes how create and delete directories and packages.

5.1. Basic operations

20021009TTT can also use Edit | New / Delete. XXX ??

The section shows how to

- **5.1.1. Create (page 76)**
- **5.1.2. Delete (page 78)**
- **5.1.3. Cut / Copy / Paste (page 79)**
- ~~**5.1.4. Recover XXX (page 79)**~~
- **5.1.5. Rename (refactor) (page 79)**
- **5.1.6. Move (refactor) (page 79)**

5.1.1. Create

You created a package already in section 3.2. **Create package (page 38)**.

~~20021030TTT where to put this??~~

~~Note that when viewing packages it is convenient to sometimes “flatten” packages.~~

~~4.38. Select the **Sourcepath**.~~

~~4.39. Click the **Flatten packages** icon (). Note how the packages are flattened.~~



~~Figure 4.25. Packages not flattened / flattened (645,646)~~

You can create a directory / package with the

- **5.1.1.1. Project tool (page 76)**
- **5.1.1.2. Commander (page 77)**

5.1.1.1. Project tool

4.40. Right-click on **\MyProjectFolder**.

4.41. Select **New | Directory**. The dialog “New directory” appears.

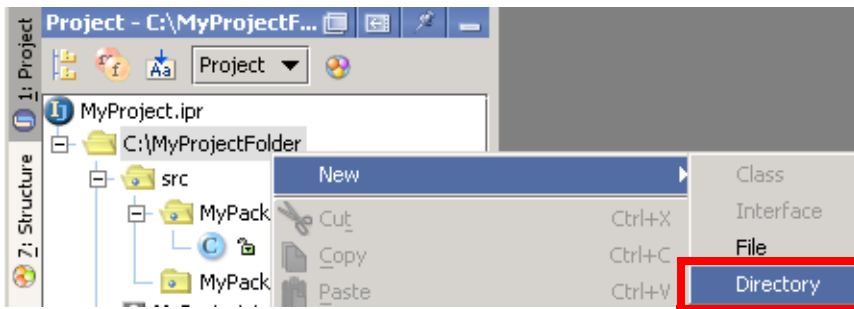


Figure 4.26. New directory (626)

4.42. For the directory name enter **MyDirectory**.

4.43. Click **OK**.

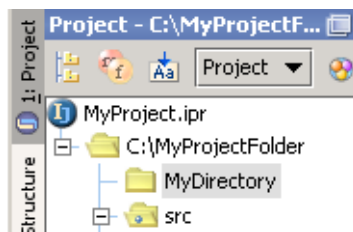


Figure 4.27. New directory (239)

5.1.1.2. Commander

- 4.44. Open the commander.
- 4.45. Display contents of **src**.
- 4.46. Right-click.
- 4.47. Select **New | Package**. The dialog “New package” appears.
- 4.48. Enter **myPackage2**.
- 4.49. Click **OK**. The package is created.

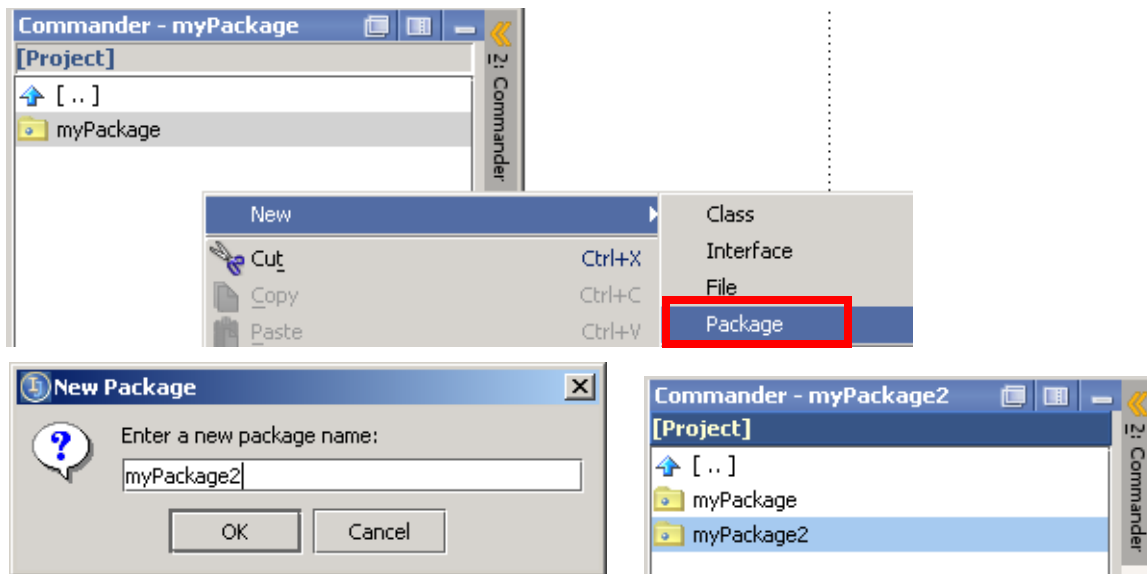


Figure 4.28. New package in Commander ([a025.a027.a026](#))

5.1.2. Delete

You can create a directory / package with the

- [5.1.2.1. Project tool \(page 78\)](#)
- [5.1.2.2. Commander \(page 78\)](#)

5.1.2.1. Project tool

4.50. Select the **package / directory**.

4.51. Click **DEL**. A warning dialog appears.

4.52. Click **OK**. The package / directory and any files / subdirectories are deleted.

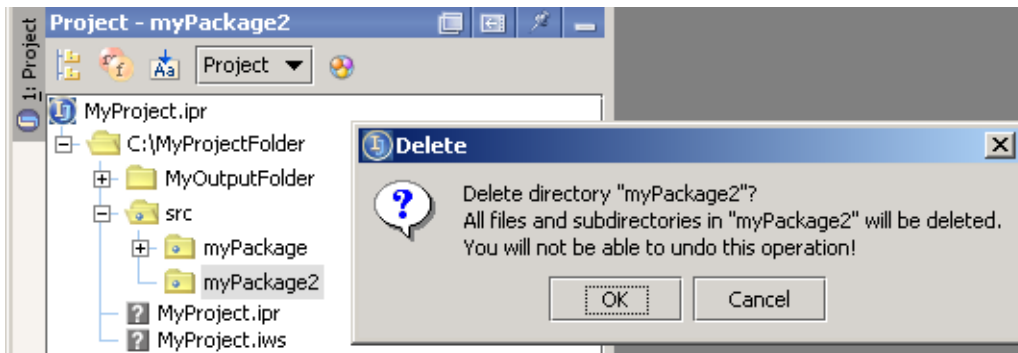


Figure 4.29. Delete package in project tool ([a028](#))

5.1.2.2. Commander

4.53. Select the **package / directory**.

4.54. Click **DEL**. A warning dialog appears.

4.55. Click **OK**. The package / directory and any files / subdirectories are deleted.

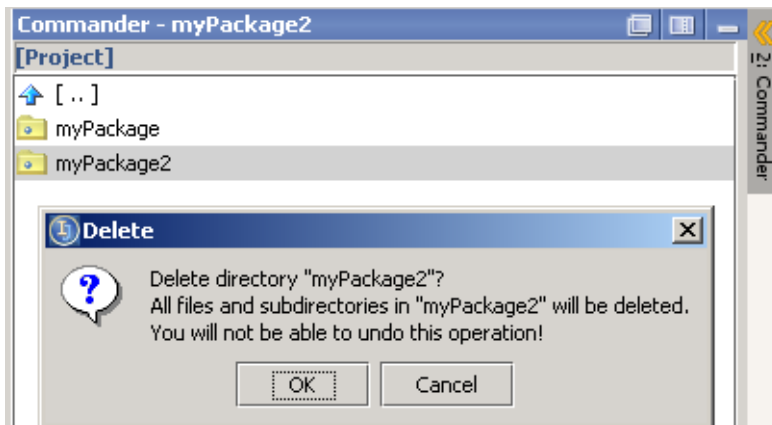


Figure 4.30. Delete package in commander ([a029](#))

5.1.3. Cut / Copy / Paste

- 4.56. Create dir **C:\MyProjectFolder\AFolder**.
- 4.57. Create dir **C:\MyProjectFolder\BFolder**.
- 4.58. In the project tool select **AFolder**.
- 4.59. Press **CTRL-X** (cut).
- 4.60. Select **BFolder**.
- 4.61. Press **CTRL-V** (paste). The dialog “Move” appears.
- 4.62. Click **OK** to move the folder.

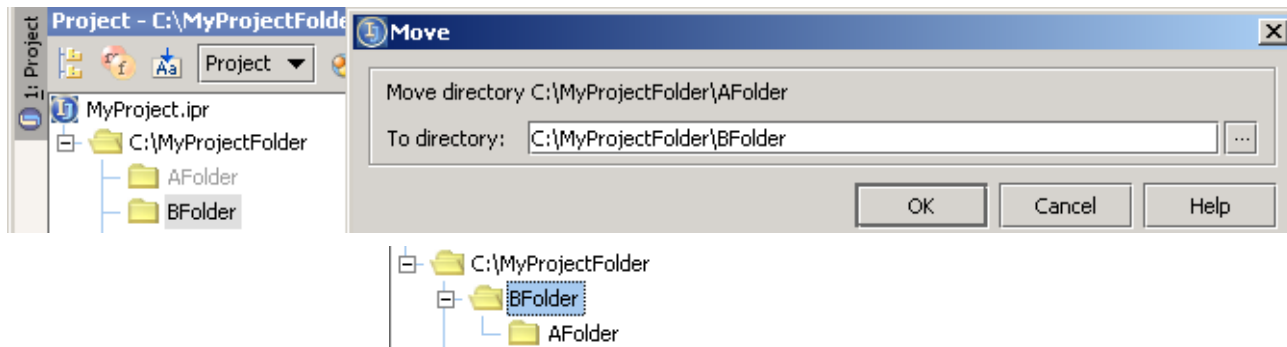


Figure 4.31. Cut / paste folder ([a030.a031](#))

- 4.63. Select **AFolder**.
- 4.64. Press **CTRL-C** (copy).
- 4.65. Select **C:\MyProjectFolder**.
- 4.66. Press **CTRL-V** (paste). The dialog “Copy” appears.
- 4.67. Click **OK** to copy the folder.

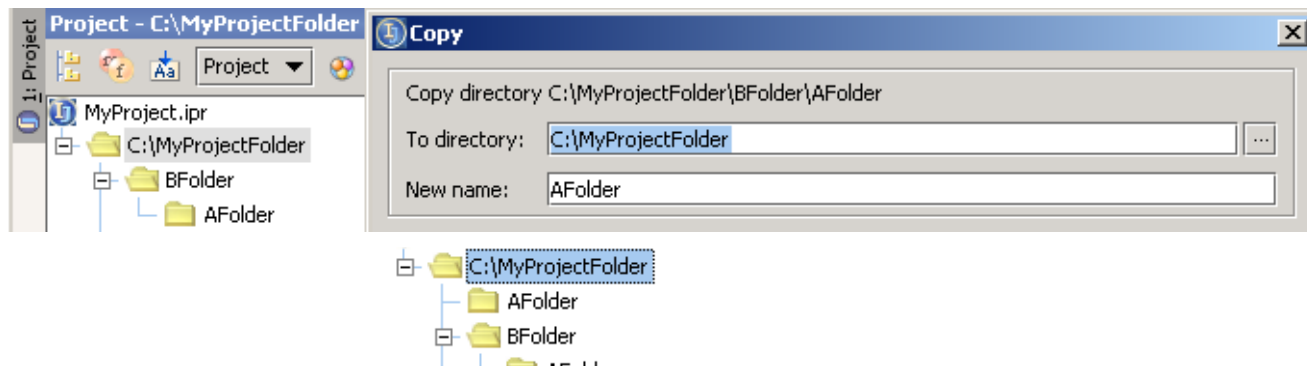


Figure 4.32. Copy / paste folder ([a032.a033](#))

5.1.4. Recover-XXX

5.1.5. Rename (refactor)

Renaming a directory / package involves refactoring the directory / package. This is demonstrated in chapter 9. [Code Refactoring \(page 201\)](#) section 9.2.1. [Package \(page 203\)](#).

5.1.6. Move (refactor)

Moving a directory / package involves refactoring the directory / package. This is demonstrated in chapter 9. [Code Refactoring \(page 201\)](#) section 9.3.1. [Package \(page 211\)](#).

6. Files

[20021030TTT last edit.](#)

[contacts: valentin](#)

[to select: view \ select in , popup dialog. ?? why is "hierarchy" not in the list in the popup??](#)

[mention custom file types.](#)

This chapter introduces the basics for working with files in IDEA:

- **6.1. Basic operations (page 81)** describes how create, rename, delete, etc. files.
- **6.2. Types / Templates (page 88)** describes the types of files.
- **6.3. Views (page 94)** describes the various views of files.

6.1. Basic operations

The following file operations are supported:

- **6.1.1. Create (page 82)**
- **6.1.2. Rename (refactor) (page 83)**
- **6.1.3. Move (refactor) (page 83)**
- **6.1.4. Copy (refactor) (page 83)**
- **6.1.5. Delete (safe) (page 84)**
- **6.1.6. Recover (page 85)**
- **6.1.7. Export to HTML (page 86)**
- **6.1.8. Print (page 87)**

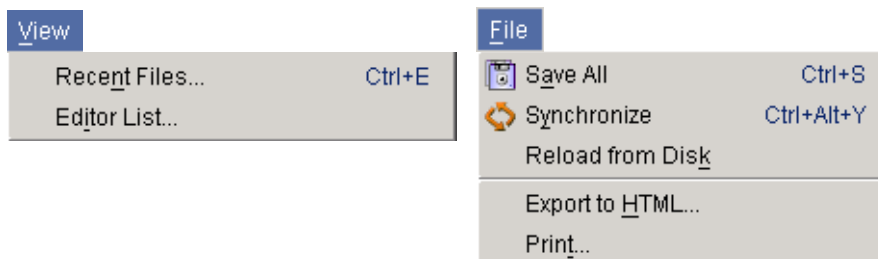


Figure 5.1. Basic file operations ([444,443,446,445](#))

6.1.1. Create

You can create the following types of files:

- **Class**
- **Interface**
- **File (text)**

6.1.1.1. Class

[20021009TTT](#) can also use [Edit | New. XXX ??](#)

5.1. Right-click on a package.

5.2. Select **New | Class**. The dialog “New Class” appears.

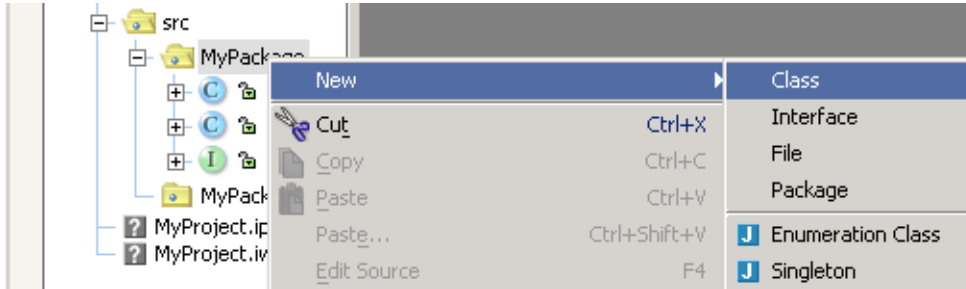


Figure 5.2. New class [\(627\)](#)

5.3. Enter the new class name.

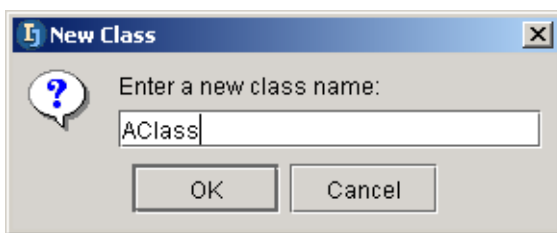


Figure 5.3. New class name [\(628\)](#)

5.4. Click **OK**. The class is created.

6.1.1.2. Interface

5.5. Right-click on a package.

5.6. Select **New | Interface**. The dialog “New Interface” appears.

5.7. Enter the new Interface name.

5.8. Click **OK**. The interface files is created.

6.1.1.3. File (text)

5.9. Right-click on a package or directory.

5.10. Select **New | File**. The dialog “New File” appears.

5.11. Enter the new file name and extension.

5.12. Click **OK**. The text file is created.

6.1.2. Rename (refactor)

Renaming a file may be a basic operation, but when a file is renamed IDEA checks for any references to the file and does any required refactoring. This is first demonstrated in Chapter 9. **Code Refactoring (page 201)** section 9.2. **Rename (page 203)**.

6.1.3. Move (refactor)

Similar to renaming, moving a file may be a basic operation, but when a file is moved IDEA checks for any references to the file and does any required refactoring. This is first demonstrated in Chapter 9. **Code Refactoring (page 201)** section 9.3. **Move (page 211)**.

6.1.4. Copy (refactor)

Similar to renaming, copying a file may be a basic operation, but when a file is copied IDEA checks for any references to the file and does any required refactoring. This is first demonstrated in Chapter 9. **Code Refactoring (page 201)** section 9.3. **Move (page 211)**.

6.1.5. Delete (safe)

[20021009TTT can also use Edit | Delete. XXX ??](#)

5.13. Select a class in the project directory.

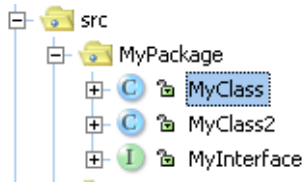


Figure 5.4. Select class (205)

5.14. Press **DEL**. The dialog “Safe delete” appears.

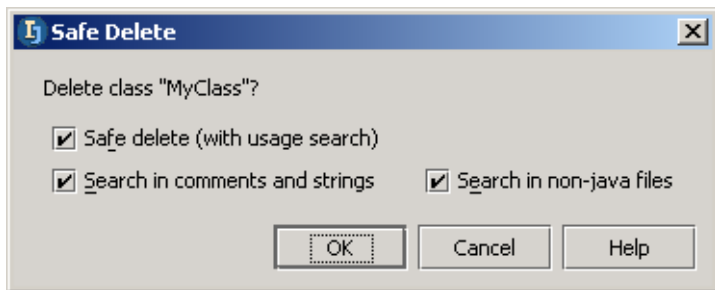


Figure 5.5. Safe delete (204)

5.15. Click **OK**. If the file is used anywhere, then the dialog “Usages detected” appears.

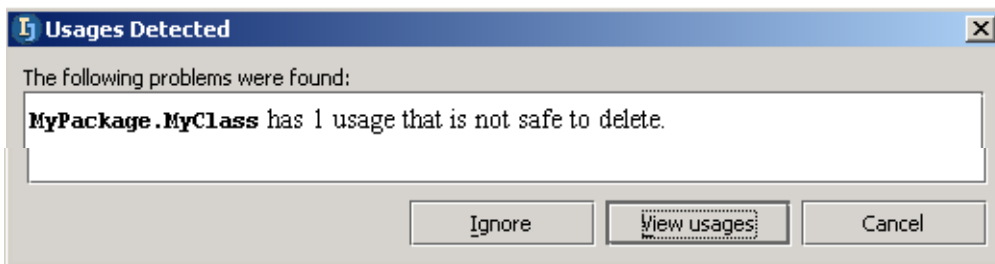


Figure 5.6. Usages detected (203,202)

5.16. Click **View usages**. The “Find” tool appears.

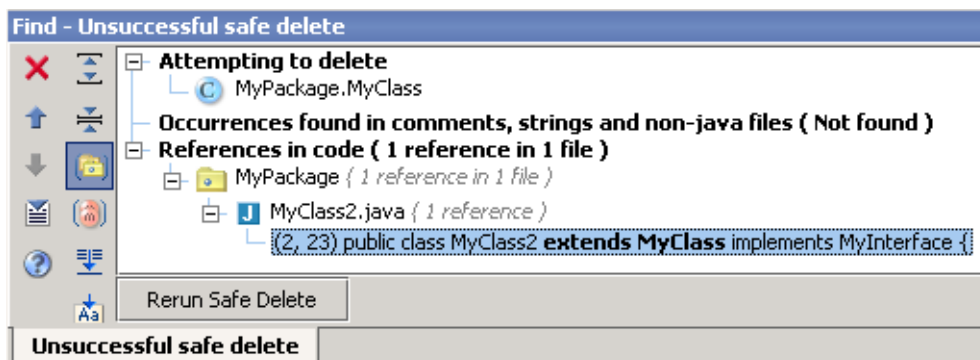


Figure 5.7. Found usages (201)

5.17. Click **Rerun safe delete** to again open the dialog “Safe delete”.

5.18. Click **OK**. Again the dialog “Usages detected” appears.

5.19. Click **Ignore**. The file is deleted.

6.1.6. Recover

To recover a deleted file:

- 5.20. Select the package the file was deleted from.
- 5.21. Select **Tools | Local VCS | Show history....** The History dialog appears.
- 5.22. Double-click on the **Safe delete** action.

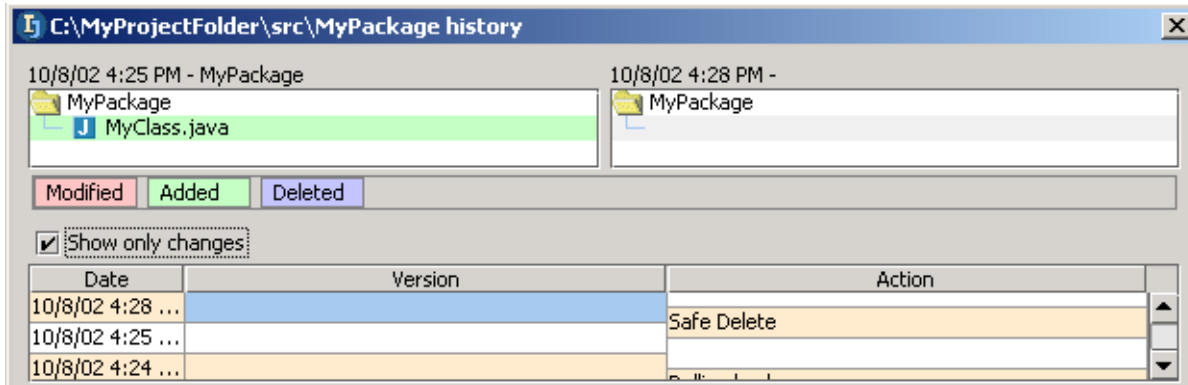


Figure 5.8. Safe delete action in history ([200.199](#))

- 5.23. Right-click.
- 5.24. Click on **Rollback**. A confirmation dialog appears.
- 5.25. Click **OK**. The file is restored (the deletion is rolled back).

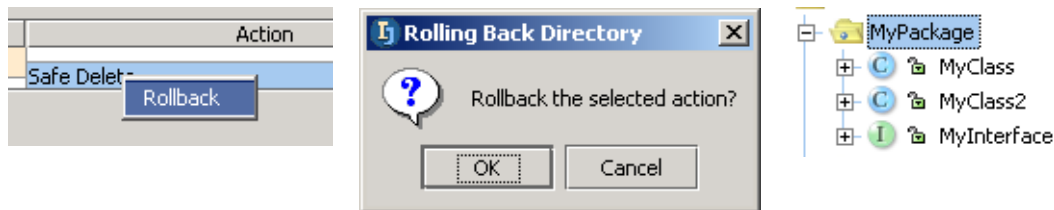


Figure 5.9. Rollback of deletion ([198.197.196](#))

6.1.7. Export to HTML

5.26. Open the file in the editor.

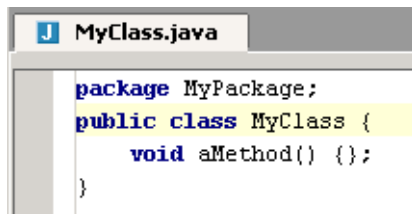


Figure 5.10. File for export in editor (657)

5.27. Select **File | Export to HTML....** The dialog “Export to HTML” appears.

5.28. Select the output directory.

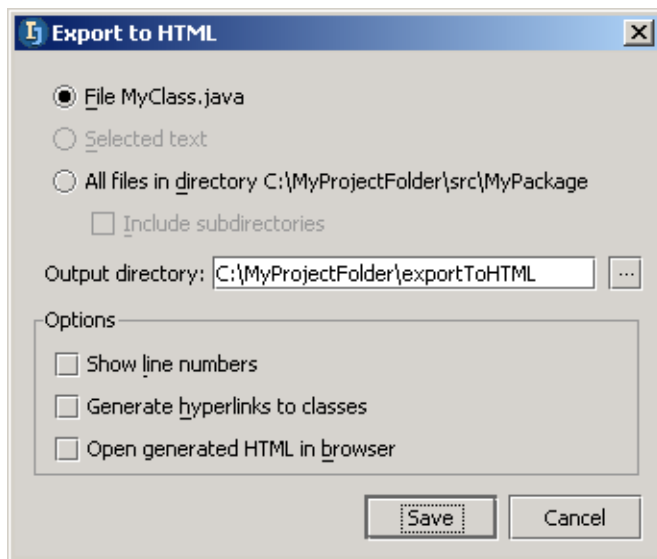


Figure 5.11. Export to html dialog (656)

5.29. Click **Save**. **C:\IntelliJ-IDEA-3.0\exportToHTML\hello\HelloWorld.java.html** is created and opened in a browser.

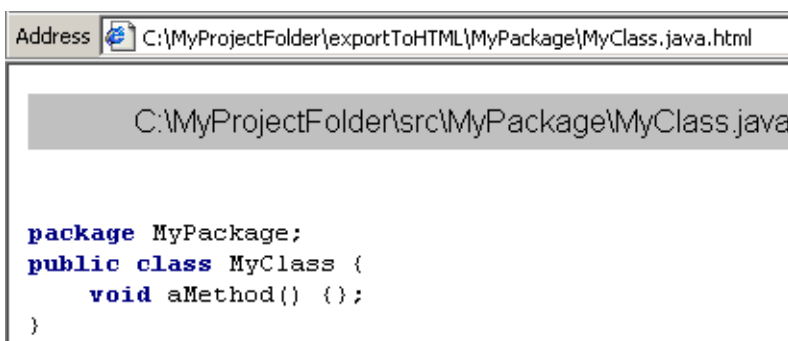


Figure 5.12. File exported to HTML (658)

6.1.8. Print

5.30. Open the file in the editor.

5.31. Select **File | Print...**. The dialog “Print” appears.

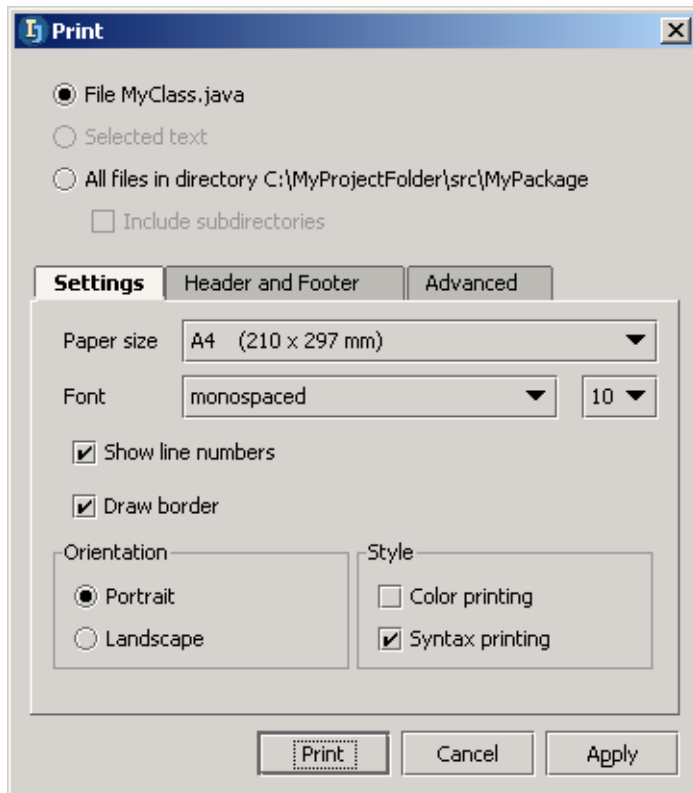


Figure 5.13. Export to html dialog (659)

5.32. Click **Print**. The standard print dialog appears.

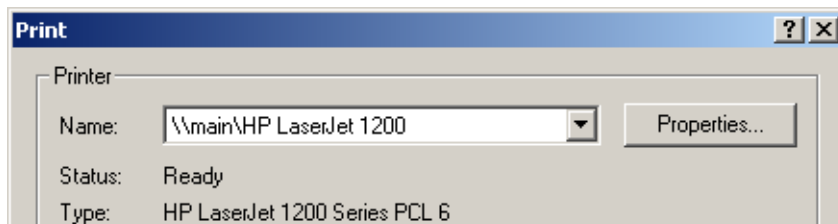


Figure 5.14. Windows print dialog (670)

5.33. Click **OK** to print.

6.2. Types / Templates

IDEA recognizes

- [6.2.1. Default file types \(page 88\)](#)
- [6.2.2. Custom file types \(page 91\)](#)

You can specify the initial contents of a file of a specific type using

- [6.2.3. Class / File templates \(page 93\)](#)

IDEA make it easier to work with various types of files by displaying recognized file types with special

- [7.4. Colors and fonts X \(page 137\)](#) and
- [7.5. Code style X \(page 145\)](#)

6.2.1. Default file types

IDEA recognizes the following default file types:

- **Archive**
- **Java class files**
- **HTML**
- **IDL**
- **Java source**
- **JSP**
- **JavaScript**
- **Text**
- **XML**

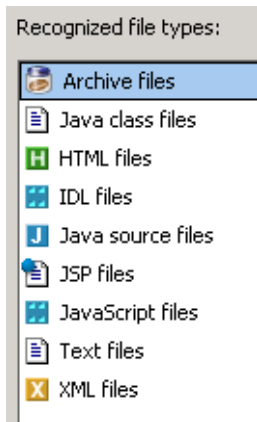


Figure 5.15. Default recognized file types [\(811\)](#)

Archive

In [section 3.1.3.3. Source path \(page 35\)](#) you added the J2SDK source zip to the IDEA source path. This allows you to easily view the contents of the zip files in IDEA.

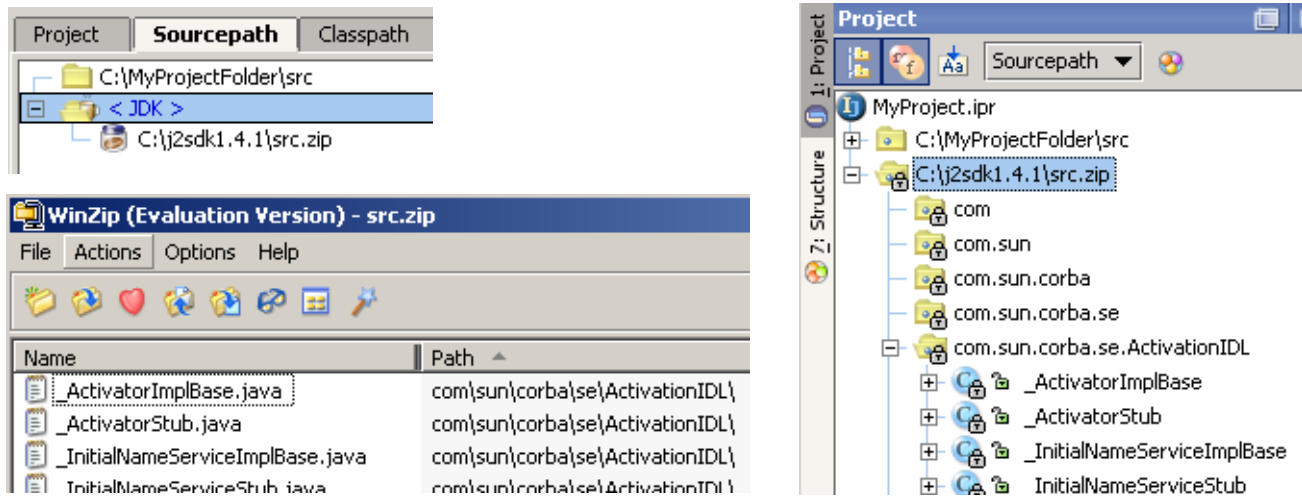


Figure 5.16. Archive file displayed in IDEA (805,806,807)

Java class files

IDEA recognizes automatically Java class files.

HTML

Colors and fonts for HTML files are described in [section 7.4.3. HTML X \(page 139\)](#).

IDL

IDL files are displayed with special colors or fonts.

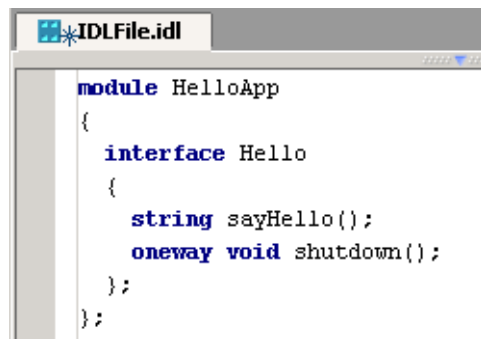


Figure 5.17. IDL (810)

Java source

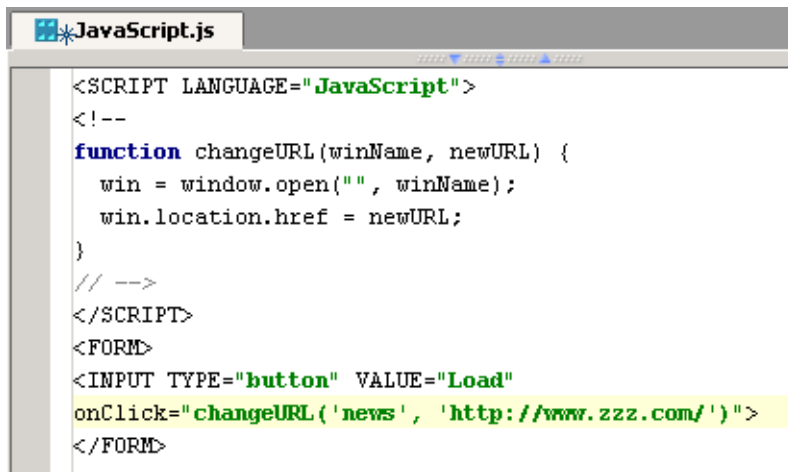
Colors and fonts for Java sources files are described in [section 7.4.2. Java X \(page 138\)](#).

JSP

Colors and fonts for JSP files are described in [section 7.4.5. JSP X \(page 141\)](#).

JavaScript

JavaScript files are displayed with special colors or fonts.

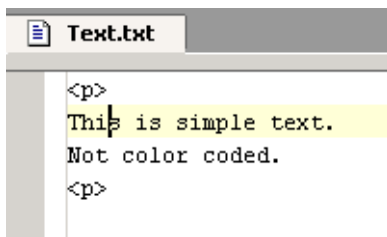


```
*JavaScript.js
<SCRIPT LANGUAGE="JavaScript">
<!--
function changeURL(winName, newURL) {
  win = window.open("", winName);
  win.location.href = newURL;
}
// -->
</SCRIPT>
<FORM>
<INPUT TYPE="button" VALUE="Load"
onClick="changeURL('news', 'http://www.zzz.com/')">
</FORM>
```

Figure 5.18. JavaScript (809)

Text

Text files are displayed with special colors or fonts.



```
Text.txt
<p>
This is simple text.
Not color coded.
<p>
```

Figure 5.19. Simple text file (808)

XML

Colors and fonts for JSP files are described in [section 7.4.4. XML X \(page 140\)](#).

6.2.2. Custom file types

IDEA automatically recognizes files types based on the file extension.

In the **IDEA Options | File types** you can

- **6.2.2.1. Add an extension (page 91)**
- **6.2.2.2. Add a file type and extension (page 91)**

6.2.2.1. Add an extension

5.34. In “Recognized file types” select **Text files**.

5.35. In “Registered extensions” click **Add...** The dialog “Add extension” appears.

5.36. As the new extension enter **text**.

5.37. Click **OK**. The new extension is in the list.

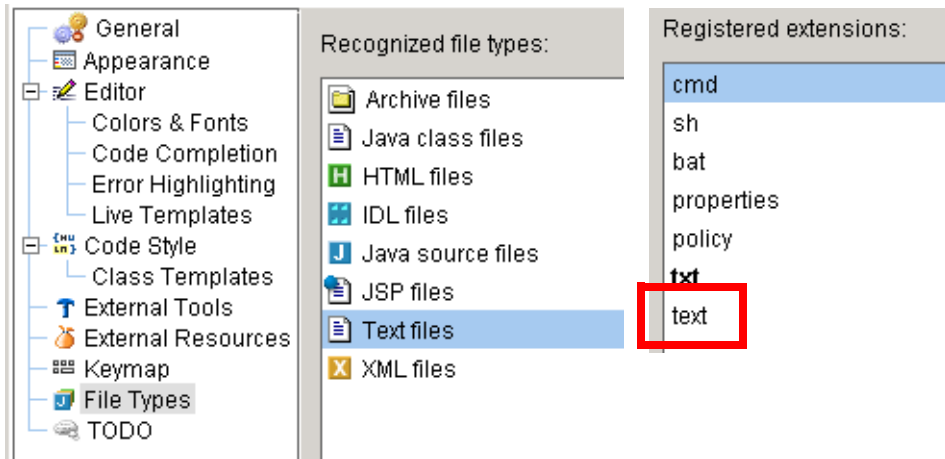


Figure 5.20. New recognized extension [\(421,420\)](#)

6.2.2.2. Add a file type and extension

5.38. In “Recognized file types” click **Add...** The dialog “Add New File Type” appears.

5.39. Enter the parameters for the file type (see the diagram).

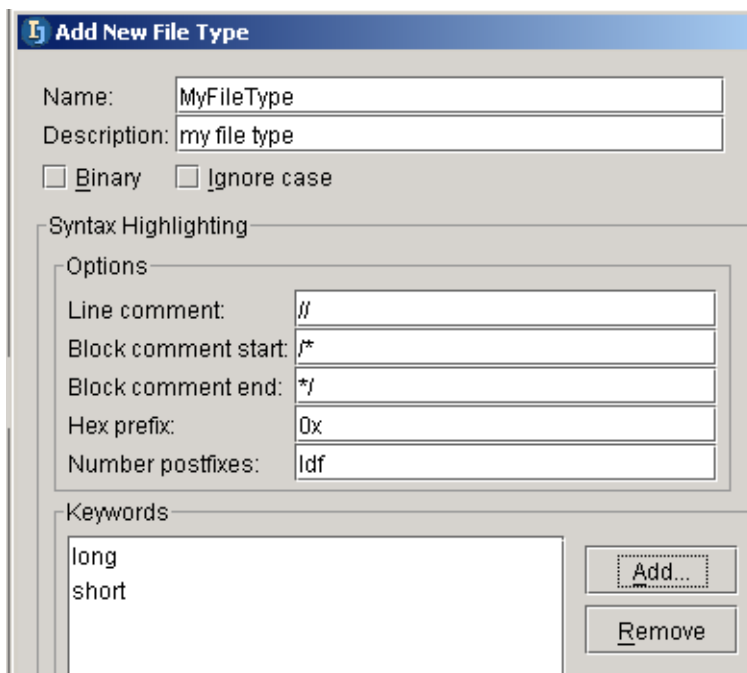


Figure 5.21. New file type parameters [\(419\)](#)

5.40. Add keywords.

5.41. Click **OK**. The file type appears in the list of recognized file types.

5.42. Add registered extension **myext**.

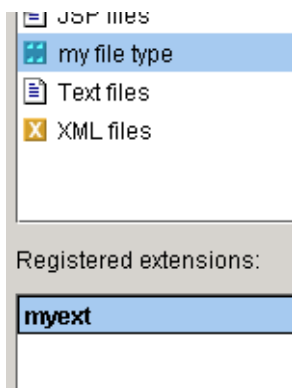


Figure 5.22. New file type and extensions ([418](#))

6.2.3. Class / File templates

The initial contents of class / interface files are determined by a template.

5.43. Select **Options | File templates....** The dialog “File templates” tab “Templates” is opened.

5.44. Select **New Class**. Note the default contents.

5.45. Select tab **Includes**. Note the default contents.

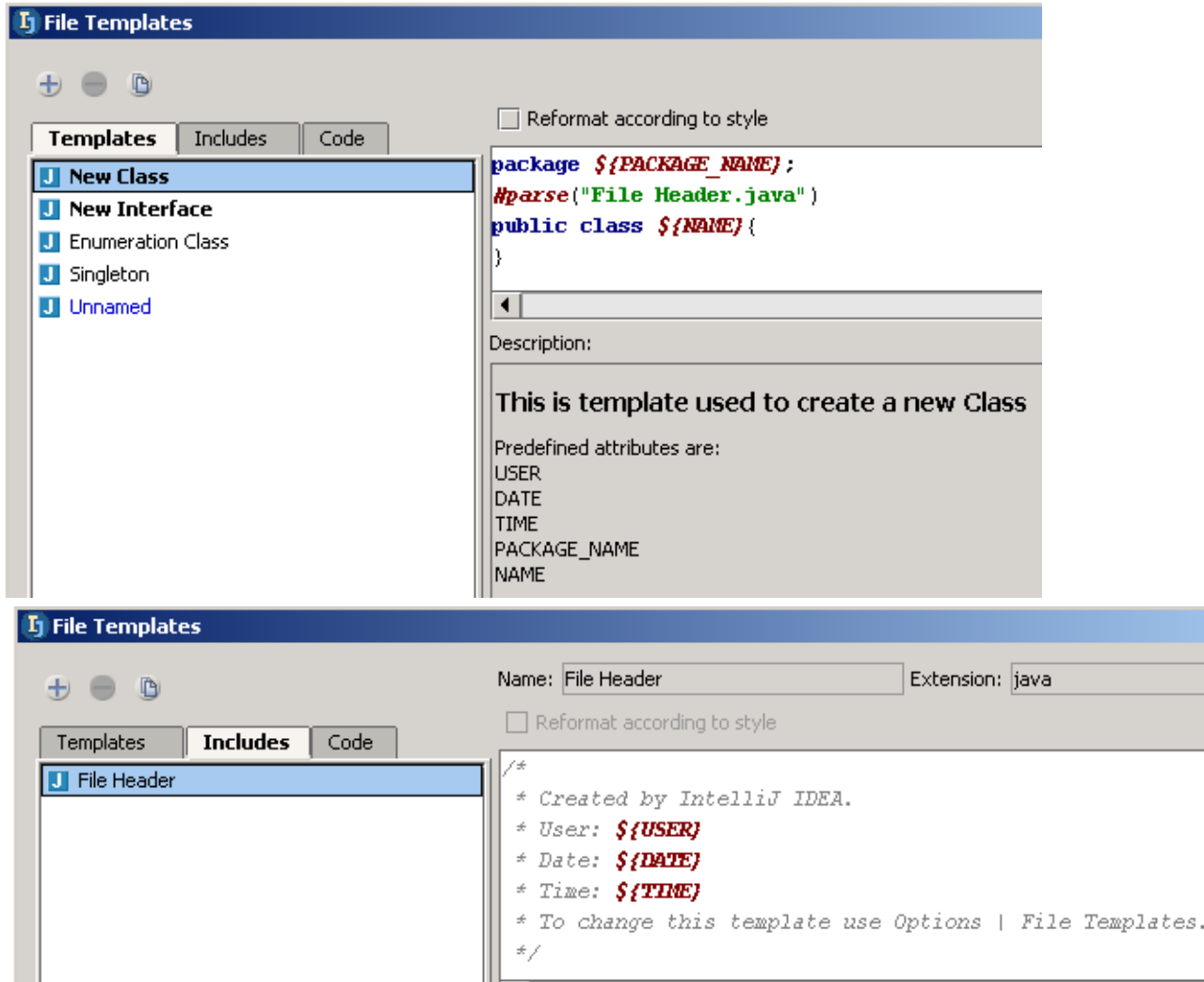


Figure 5.23. Class template (611,195)

Check **Reformat according to style** to reformat existing classes.

6.3. Views

There are 4 views that show the contents of a file

- 6.3.1. Editor (page 95)
- 6.3.2. Project tool (show members) (page 95)
- 6.3.3. Commander (page 95)
- 6.3.4. (File) Structure tool (page 96)

Also the

- 6.3.5. Hierarchy tool (page 100)

shows how the contents of the current file are referenced, implemented, etc. in other files.

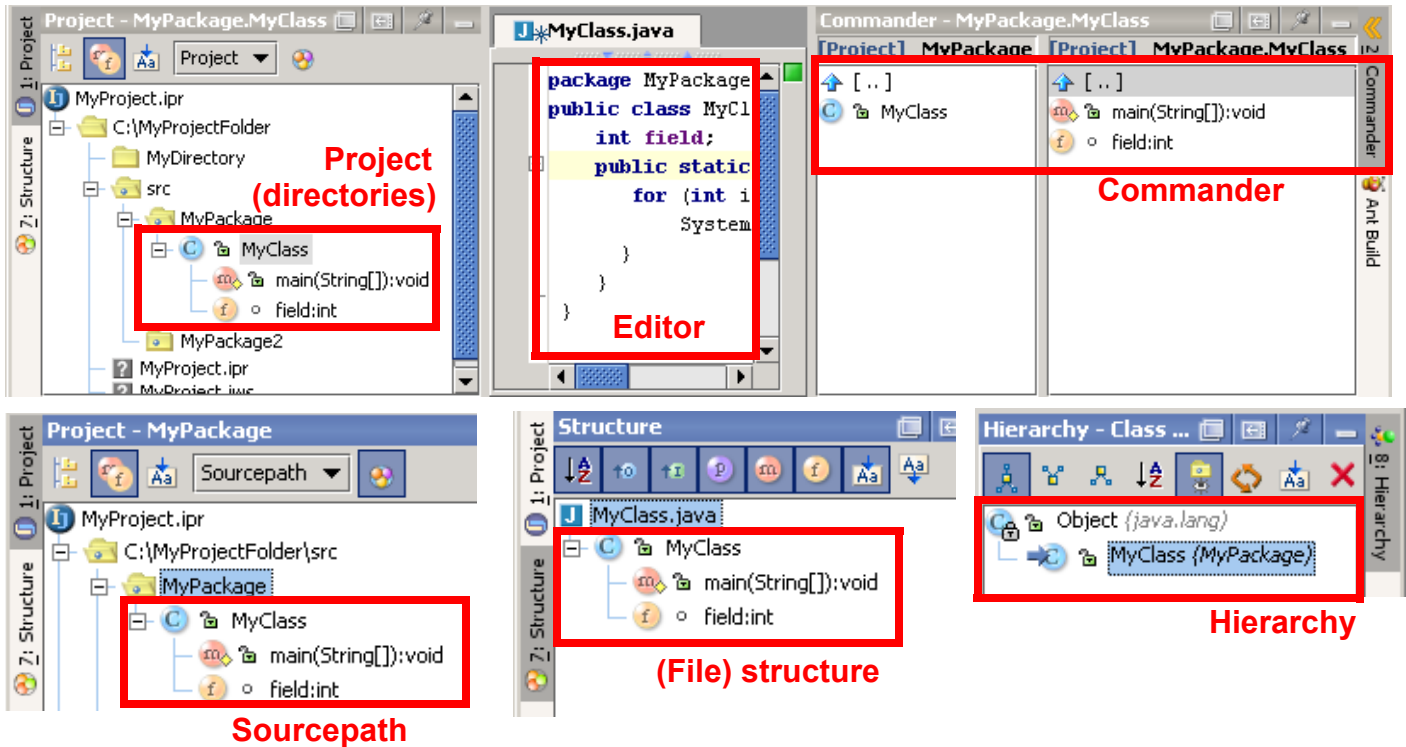


Figure 5.24. 5 views of a file (238.237.236.235)

6.3.1. Editor

The editor is the primary tool for making changes to text (add methods, fields, etc.).

The editor and the tools available within it are described in detail in

- **Chapter 7. Editor X (page 107)**
- **Chapter 8. Code Automation (page 163)**
- **Chapter 9. Code Refactoring (page 201)**

6.3.2. Project tool (show members)

The Project (directories) tool shows the methods, fields, etc. of a class.

5.46. Add the following to MyClass:

```
public class MyClass{  
    int field;
```

5.47. Select the **Project** view.

5.48. Click the **Show members** icon () to show the class members.

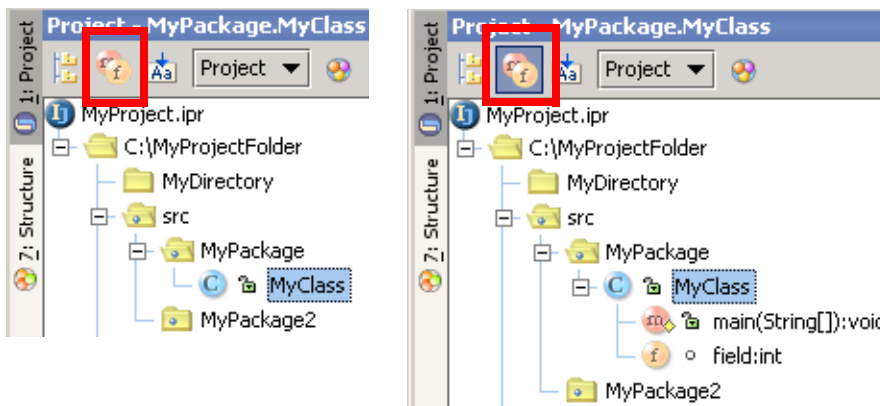


Figure 5.25. Members not shown / shown ([234,233](#))

6.3.3. Commander

The commander allows you to view the contents of a file.

5.49. In the Commander: Navigate to **MyPackage.MyClass**.

5.50. Double-click on **main**. MyClass.java appears in the editor.

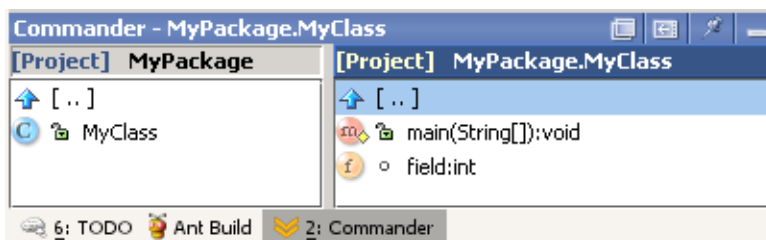


Figure 5.26. Viewing file contents with Commander ([232](#))

6.3.4. (File) Structure tool

The Structure tool shows the structure of a class.

- [6.3.4.1. Tool variations \(page 96\)](#)
- [6.3.4.2. Sort alphabetically \(page 98\)](#)
- [6.3.4.3. Group overriding methods \(page 98\)](#)
- [6.3.4.4. Group implementation methods \(page 98\)](#)
- [6.3.4.5. Show properties \(page 98\)](#)
- [6.3.4.6. Show methods \(page 99\)](#)
- [6.3.4.7. Show fields \(page 99\)](#)
- [6.3.4.8. Autoscroll to source \(page 99\)](#)
- [6.3.4.9. Autoscroll from source \(page 99\)](#)

6.3.4.1. Tool variations

The file structure can be shown in 3 variations of the structure tool:

- **Normal**
- **From project tool**
- **Popup**

6.3.4.1.1. Normal

5.51. Modify **MyClass**:

```
package MyPackage;  
public class MyClass {  
    void aMethod() {};  
}
```

5.52. Create **MyInterface**

```
package MyPackage;  
public interface MyInterface {  
    public void iMethod();  
}
```

5.53. Create **MyClass2**

```
package MyPackage;  
public class MyClass2 extends MyClass implements MyInterface {  
    void aMethod(){};  
    public void iMethod(){};  
    int iField;  
    public int getiField() {  
        return iField;  
    }  
}
```

5.54. Select the **MyClass2** editor.

5.55. Click on **7. Structure** to display the structure.

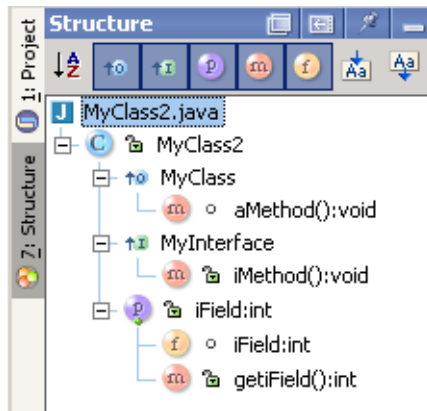


Figure 5.27. Structure tool (231)

6.3.4.1.2. From project tool

5.56. Close the 7. Structure tool.

5.57. Open the 1. Project tool.

5.58. Click on the Show Structure icon (). The structure is shown.

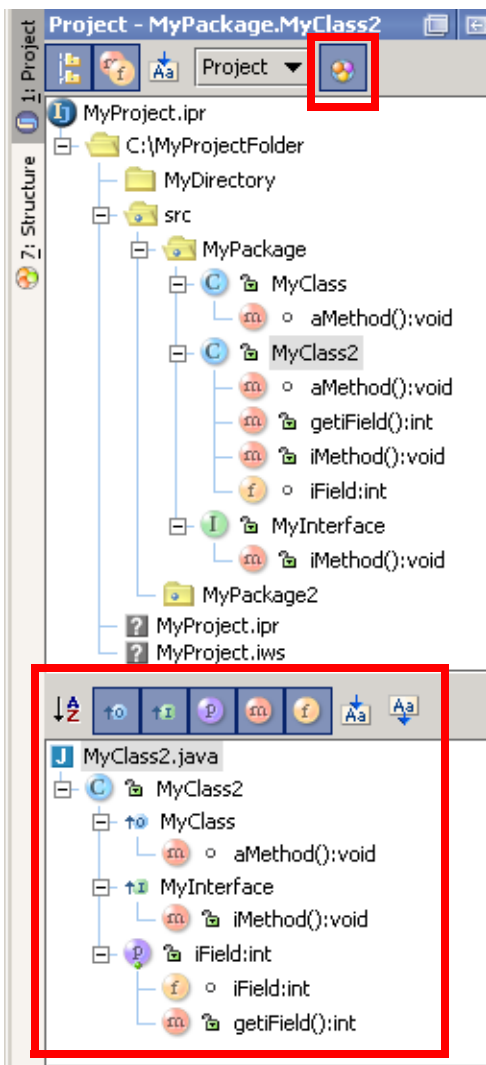


Figure 5.28. Structure tool as subset of Project tool (230)

6.3.4.1.3. Popup

- 5.59. Click in the **MyClass2** source text.
- 5.60. Select **View | File structure popop...**

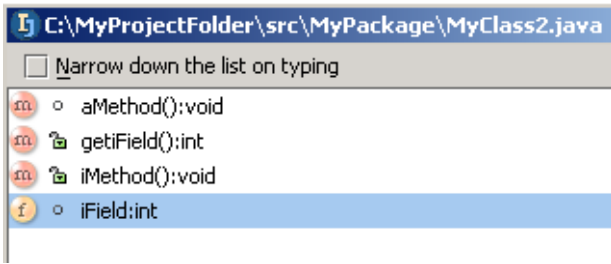




Figure 5.29. File structure popup (607)

6.3.4.2. Sort alphabetically

Select to sort alphabetically.

6.3.4.3. Group overriding methods

- 5.61. Unselect all.
- 5.62. Select the **Show methods** () icon.
- 5.63. Select the **Group overriding methods** () icon.

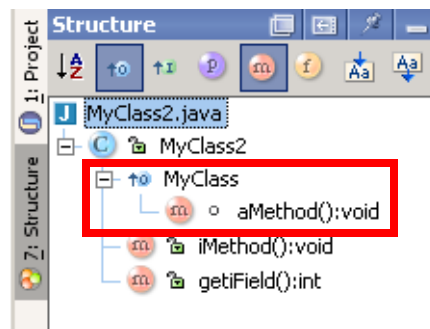




Figure 5.30. Group overriding methods (217)

6.3.4.4. Group implementation methods

- 5.64. Unselect the **Group overriding methods** () icon.
- 5.65. Select the **Group implementation methods** () icon.

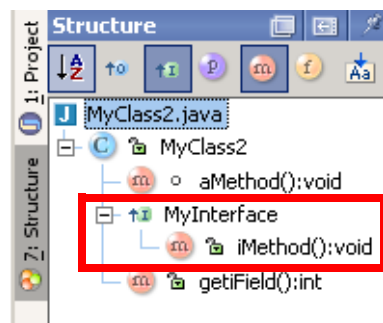




Figure 5.31. Group implementation methods (215)

6.3.4.5. Show properties

- 5.66. Unselect the **Group implementation methods** () icon.
- 5.67. Select the **Show properties** () icon.

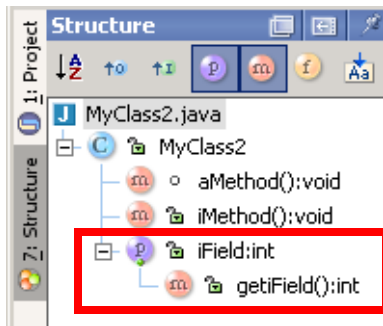


Figure 5.32. Show properties (213)

6.3.4.6. Show methods

5.68. Select the **Show methods** () icon.

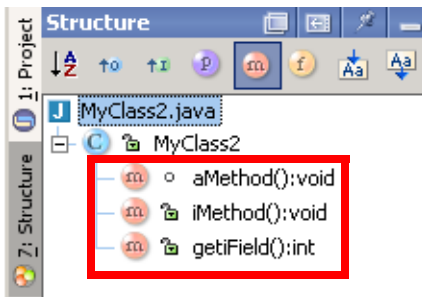


Figure 5.33. Show methods (212)

6.3.4.7. Show fields

5.69. Unselect the **Show methods** () icon.

5.70. Select the **Show fields** () icon.

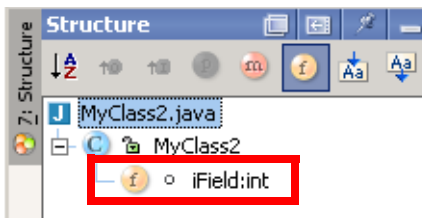


Figure 5.34. Show fields (210)

6.3.4.8. Autoscroll to source

20021008TTT is this right??

If selected: Scrolling within the Structure will open the source file.

6.3.4.9. Autoscroll from source

20021008TTT is this right??

If selected: Opening the source will automatically cause the source to be shown in the Structure.

6.3.5. Hierarchy tool

The Hierarchy tool shows the

- [6.3.5.1. Type hierarchy \(page 100\)](#)
- [6.3.5.2. Method hierarchy \(page 101\)](#)
- [6.3.5.3. Call hierarchy \(page 102\)](#)

6.3.5.1. Type hierarchy

Shows the type hierarchy.

- [6.3.5.1.1. Class hierarchy \(page 100\)](#)
- [6.3.5.1.2. Supertypes hierarchy \(page 100\)](#)
- [6.3.5.1.3. Subtypes hierarchy \(page 101\)](#)
- [6.3.5.1.4. Sort alphabetically \(page 101\)](#)
- [6.3.5.1.5. Show packages \(page 101\)](#)
- [6.3.5.1.6. Refresh \(page 101\)](#)
- [6.3.5.1.7. Autoscroll to source \(page 101\)](#)

6.3.5.1.1. Class hierarchy

5.71. Modify **MyClass**:

```
package MyPackage;
public class MyClass{
    int field;
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            System.out.println("i" + i);
        }
    }
    void aMethod() {};
    void aMethod1(){
        MyClass2 mc2 = new MyClass2();
        mc2.aMethod2();
    };
}
```

5.72. Modify **MyClass2**

```
package MyPackage;
public class MyClass2 extends MyClass {
    void aMethod(){};
    void aMethod2(){};
}
```

5.73. Select the editor for **MyClass**.

5.74. Select **View | Type Hierarchy**. The hierarchy is shown.

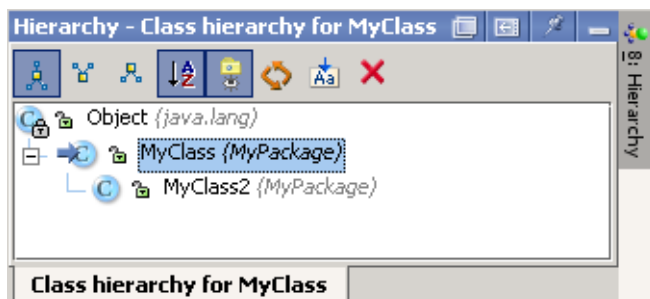


Figure 5.35. Class hierarchy ([605](#))

6.3.5.1.2. Supertypes hierarchy

5.75. Click on the **Supertypes hierarchy** () icon.

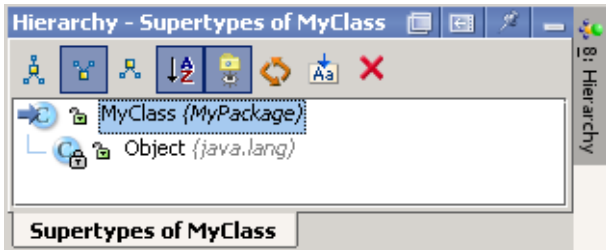


Figure 5.36. Supertypes hierarchy (228)

6.3.5.1.3. Subtypes hierarchy

5.76. Click on the **Subtypes hierarchy** () icon.

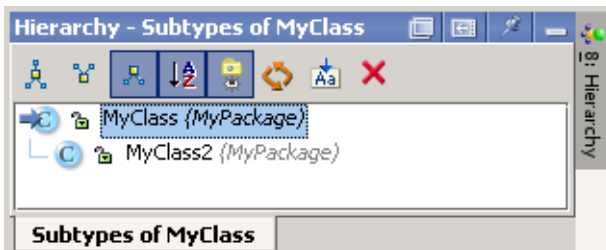


Figure 5.37. Subtypes hierarchy (225)

6.3.5.1.4. Sort alphabetically

Sorts the listed files alphabetically.

6.3.5.1.5. Show packages

Shows/hides the package names.

6.3.5.1.6. Refresh

Refreshes the display.

6.3.5.1.7. Autoscroll to source

If selected: Scrolling within the hierarchy will open the source file for the selected class.

6.3.5.2. Method hierarchy

Shows the method hierarchy.

- 6.3.5.2.1. Show hierarchy (page 101)
- 6.3.5.2.2. Sort alphabetically (page 102)
- 6.3.5.2.3. Show packages (page 102)
- 6.3.5.2.4. Refresh (page 102)
- 6.3.5.2.5. Autoscroll to source (page 102)

6.3.5.2.1. Show hierarchy

5.77. In **MyClass** select method **aMethod()**.

5.78. Select **View | Method Hierarchy**. The method hierarchy is shown.

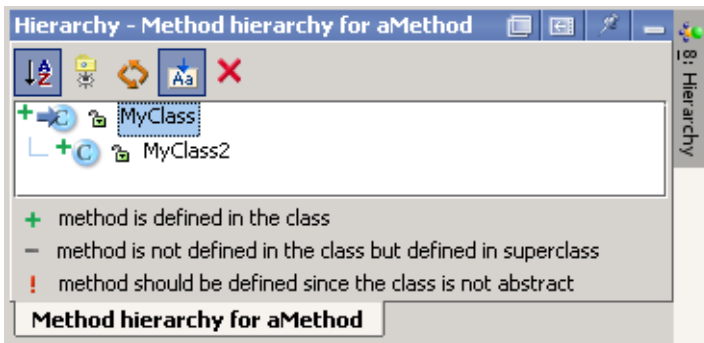


Figure 5.38. Method hierarchy (224)

6.3.5.2.2. Sort alphabetically

Sorts the listed files alphabetically.

6.3.5.2.3. Show packages

Shows/hides the package names.

6.3.5.2.4. Refresh

Refreshes the display.

6.3.5.2.5. Autoscroll to source

If selected: Scrolling within the hierarchy will open the source file for the selected class.

6.3.5.3. Call hierarchy

Shows the call hierarchy.

- 6.3.5.3.1. Show hierarchy (page 102)
- 6.3.5.3.2. Caller methods hierarchy (page 102)
- 6.3.5.3.3. Callee methods hierarchy (page 102)
- 6.3.5.3.4. Sort alphabetically (page 103)
- 6.3.5.3.5. Show packages (page 103)
- 6.3.5.3.6. Scope (page 103)
- 6.3.5.3.7. Refresh (page 103)
- 6.3.5.3.8. Autoscroll to source (page 103)

6.3.5.3.1. Show hierarchy

5.79. In **MyClass** select method **aMethod2()**.

5.80. Select **View | Call Hierarchy**. The call hierarchy is shown.

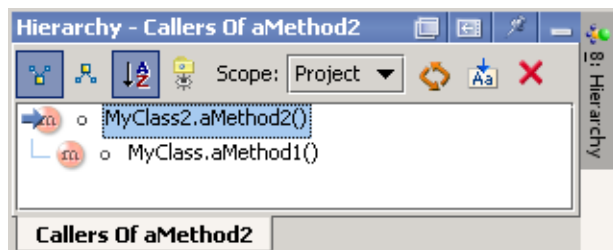


Figure 5.39. Call hierarchy (caller methods) (223)

6.3.5.3.2. Caller methods hierarchy

5.81. Click on the **Caller methods hierarchy** () icon (same as above).

6.3.5.3.3. Callee methods hierarchy

5.82. Click on the **Callee methods hierarchy** () icon (same as above).

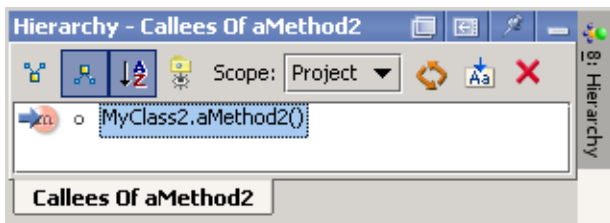


Figure 5.40. Call hierarchy (callee methods) (220)

6.3.5.3.4. Sort alphabetically

Sorts the listed files alphabetically.

6.3.5.3.5. Show packages

Shows/hides the package names.

6.3.5.3.6. Scope

Select:

- **Project** to search within the entire project
- **This class** to search only within the selected class

6.3.5.3.7. Refresh

Refreshes the display.

6.3.5.3.8. Autoscroll to source

If selected: Scrolling within the hierarchy will open the source file for the selected class.

Part C. Editing files

[20021018TTT last edit.](#)

The chapters in this part describe the extensive functionality provided by IDEA for editing files.

- 7. Editor X (page 107).** Demonstrates how the ease-of-use and extensive functionality of the editor makes it the tool of choice for editing any type of code file.
- 8. Code Automation (page 163).** Demonstrates how the code automation functions provided by IDEA allow you to more easily and quickly create, modify, and improve code.
- 9. Code Refactoring (page 201).** Demonstrates how IDEA's extensive refactoring support makes code maintenance much easier.
- ~~**10. Code Inspection X (page 243).** Demonstrates how IDEA can automatically inspect your code and recommend / implement solutions or improvements.~~
- 11. Version control (page 255).** Demonstrates how IDEA's local versioning tool and compatible external tools (CVS, SourceSafe, StarTeam) provide complete and robust version control.
- 12. Java Doc (page 269).** Demonstrates how IDEA not only provides its own JavaDoc functionality but also makes it easy to integrate Sun's JavaDoc tools for use from within IDEA.

7. Editor X

20021030TTT: last edit.

contacts: valentin (maxim?)

This chapter introduces editing features provided by IDEA and includes the following sections:

- **7.1. File operations (page 108)**
- **7.2. Text editing X (page 111)**
- **7.3. Find / Navigation (page 124)**
- **7.4. Colors and fonts X (page 137)**
- **7.5. Code style X (page 145)**
- **7.6. Error indication X (page 156)**
- **7.7. Todo (page 160)**

Note that chapters

- **8. Code Automation (page 163)**
- **9. Code Refactoring (page 201)**

also introduce editing functionality.

7.1. File operations

The IDEA editor supports the following file operations

- 7.1.1. Open (page 108)
- 7.1.2. Synchronize (page 110)
- 7.1.3. Save all (page 110)
- 7.1.4. Close (page 110)

7.1.1. Open

You can open a file in an editor

- From a view
- From “Open file” dialog
- Autoscroll to source
- Reload from disk
- Recent files

7.1.1.1. From a view

Simply double-click on a file (or a file member, field, etc.) to open the file from within any view (views were introduced in 6.3. Views (page 94))

7.1.1.2. Autoscroll to source

6.1. Close the source file for **MyClass** (if open).

6.2. Click the **Autoscroll to source** icon ().

6.3. Click on **MyPackage**.

6.4. Move the focus (with the **arrow down key**) to **MyClass**. **MyClass.java** is opened in the editor.

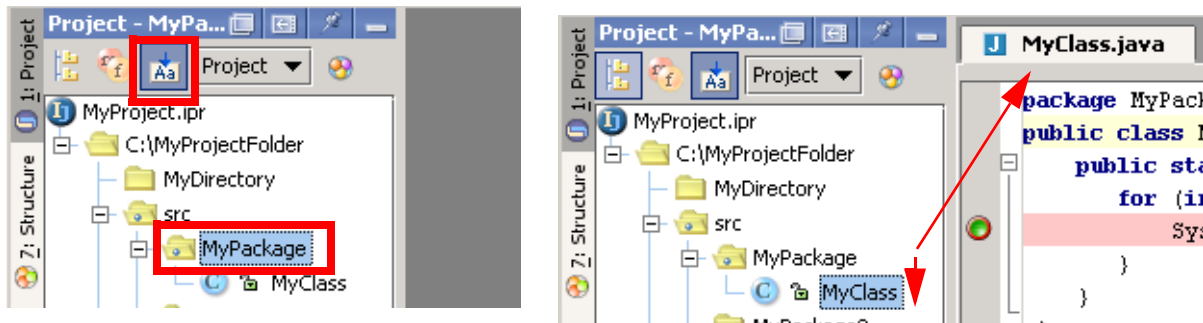


Figure 6.1. Autoscroll to source (647.648)

7.1.1.3. From “Open file” dialog

6.5. Select **File | Open file...**. The dialog “Open file” appears.

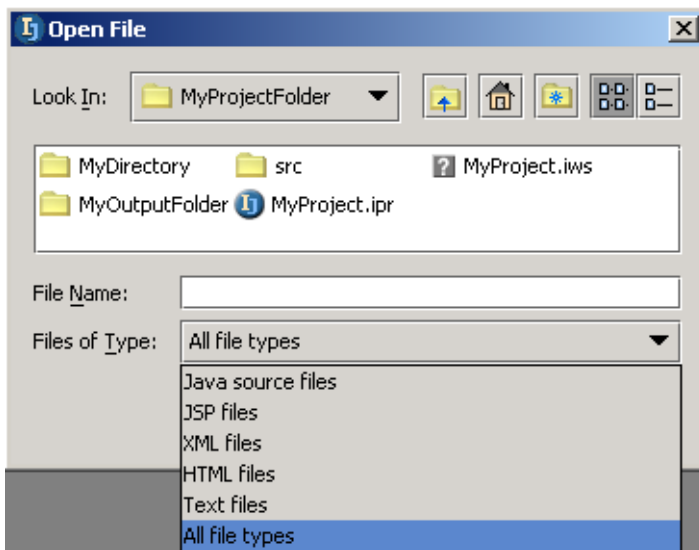


Figure 6.2. Dialog “Open file” (653)

6.6. Double-click on a file to open.

7.1.1.4. Reload from disk

6.7. Add any text to MyClass.java.

6.8. Select **File | Reload from disk**.

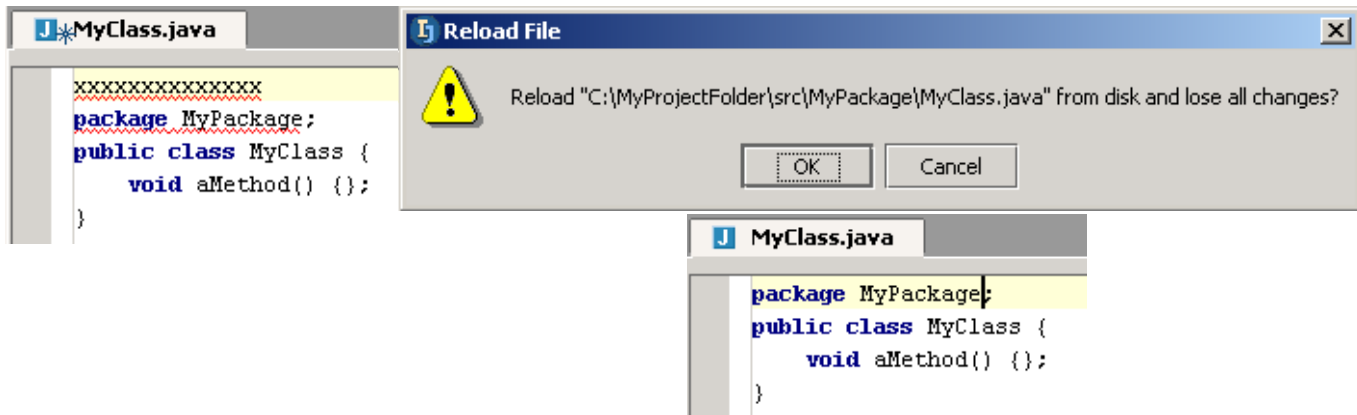


Figure 6.3. Reload file (209.208.207)

7.1.1.5. Recent files

6.9. Select **View | Recent files...**. A dialog with a list of recently opened files appears.

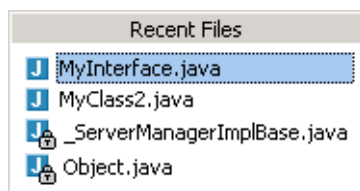


Figure 6.4. List of recent files (651)

6.10. Click on a file to open in an editor.

7.1.2. Synchronize

[20021008TTT: how do i demo synchronize??](#)

6.11. Select **File | Synchronize** to synchronize the file.

7.1.3. Save all

6.12. To save all open files: Select **File | Save All**.

Note: IDEA does not allow an individual file to be saved. There are several very compelling reasons for this which will become apparent during the course of this tutorial.

see <http://www.intellij.net/forums/thread.jsp?forum=1&thread=8721&message=237663&q=53617665#237663>

7.1.4. Close

You can

- **Close Active**
- **Close All**
- **Close All but current**

7.1.4.1. Close Active

6.13. To close the active editor do one of the following:

- Select **File | Close Active Editor**.
- Right-click on the editor tab and select **Close**.

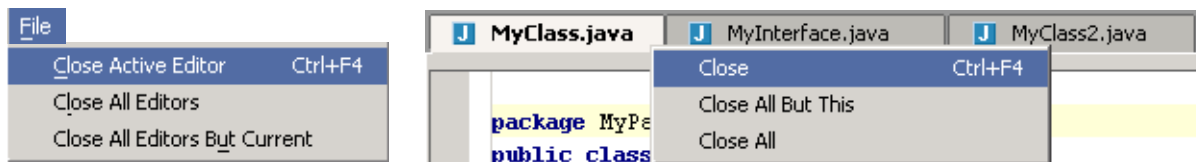


Figure 6.5. Close file menus [\(654,655,206\)](#)

7.1.4.2. Close All

6.14. To close all editors do one of the following (reference diagram above):

- Select **File | Close All Editors**.
- Right-click on the editor tab and select **Close All**.

7.1.4.3. Close All but current

6.15. To close all editors do one of the following (reference diagram above):

- Select **File | Close All Editors But Current**.
- Right-click on the editor tab and select **Close All But This**.

7.2. Text editing X

The IDEA provides extensive text editing functionality

- 7.2.1. Undo / Redo (page 111)
- 7.2.2. Select (page 111)
- 7.2.3. Cut / Copy / Paste / Duplicate / Delete (page 112)
- 7.2.4. Move / Scroll (page 113)
- 7.2.5. Indent / tabs / lines (page 122)
- 7.2.6. Modify text (page 123)

7.2.1. Undo / Redo

2 of the most important editing functions (and one that most people use quite often) are:

Undo CTRL-Z

Undo the previous action.

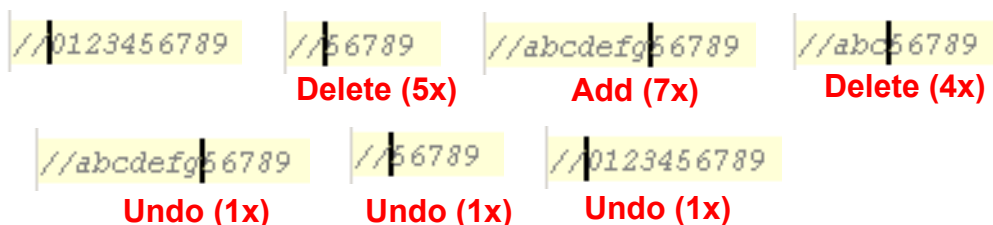


Figure 6.6. Undo (813-819)

Redo CTRL-SHIFT-Z

Redo the an undone action.

7.2.2. Select

The following select functions are available:

Select line at caret

Select the entire line of text in the line that contains the caret.

Select word at caret CTRL-W

Select the entire word that contains the caret.



Figure 6.7. Select line / word at caret (820,821,822)

Unselect word at caret CTRL-SHIFT-W

Unselect the entire word that contains the caret.

7.2.3. Cut / Copy / Paste / Duplicate / Delete

The following functions are supported:

Cut CTRL-X

Standard Cut.

Copy CTRL-C

Standard Copy.

Paste CTRL-V

Standard Paste.

Duplicate line or block CTRL-D

Select a block of text. Clicking CTRL-D will create a copy of the block immediately following the block.

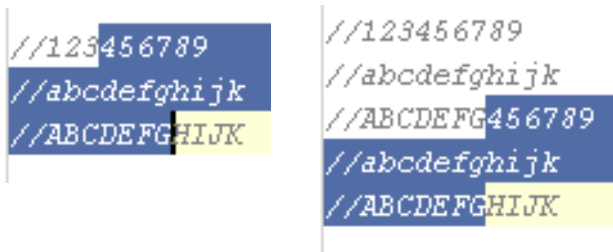


Figure 6.8. Duplicated block ([732,733](#))

Delete line at caret CTRL-Y

Place the caret on a line. Clicking CTRL-Y will delete the line.

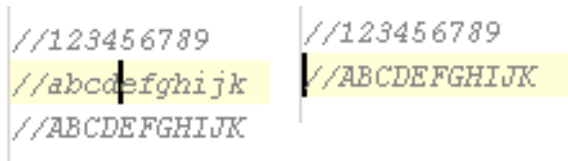


Figure 6.9. Delete line at caret ([734,735](#))

Delete to word end CTRL-DEL

Place the caret in a word. Clicking CTRL-DEL will delete all characters in the word after the caret.

Delete to word start CTRL-BACKSPACE

Place the caret in a word. Clicking CTRL-BACKSPACE will delete all characters in the word before the caret.

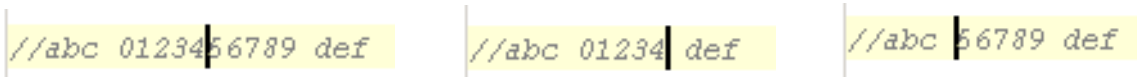


Figure 6.10. Delete to word end / start ([736,737,738](#))

Toggle insert/override INSERT

Click INSERT. The caret changes from a vertical blinking line to a blinking rectangle (that encloses a character). Typing a character will overwrite the enclosed character.



Figure 6.11. Non-insert mode (overwrite) ([739,740](#))

7.2.4. Move / Scroll

IDEA provides a wide variety of move and scroll functions:

- [7.2.4.1. Move basics \(left/right/up/down\) \(page 113\)](#)
- [7.2.4.2. Within code block \(page 114\)](#)
- [7.2.4.3. Within line \(page 115\)](#)
- [7.2.4.4. Within text \(page 116\)](#)
- [7.2.4.5. Page \(page 118\)](#)
- [7.2.4.6. Scroll \(page 120\)](#)

7.2.4.1. Move basics (left/right/up/down)

BACKSPACE

Move cursor back 1 character.

LEFT

Move cursor left 1 character.

RIGHT

Move cursor right 1 character.

UP

Move cursor to the next upper line.

DOWN

Move cursor to the next lower line.

Left with selection SHIFT-LEFT

Place the caret in a line. Press and hold SHIFT-LEFT to select the line to the left of the caret.



```
//abc 0123456789 def
```

Figure 6.12. Left with selection ([741,742](#))

Right with selection SHIFT-RIGHT

Place the caret in a line. Press and hold SHIFT-RIGHT to select the line text to the right of the caret.

Up with selection SHIFT-UP

Place the caret in a line. Press and hold SHIFT-UP to select the text as shown below.



```
//0123456789  
//abcdefghijkl  
//ABCDEFGHIJK
```

Figure 6.13. Up with selection ([743,744](#))

Down with selection SHIFT-DOWN

Place the caret in a line. Press and hold SHIFT-DOWN to select the text as shown below.

7.2.4.2. Within code block

Move to code block start CTRL-]

Place the caret within a code block. Clicking CTRL-] will cause the caret to be placed at the start of the code block.

Move to code block end CTRL-[

Place the caret within a code block. Clicking CTRL-[will cause the caret to be placed at the end of the code block.

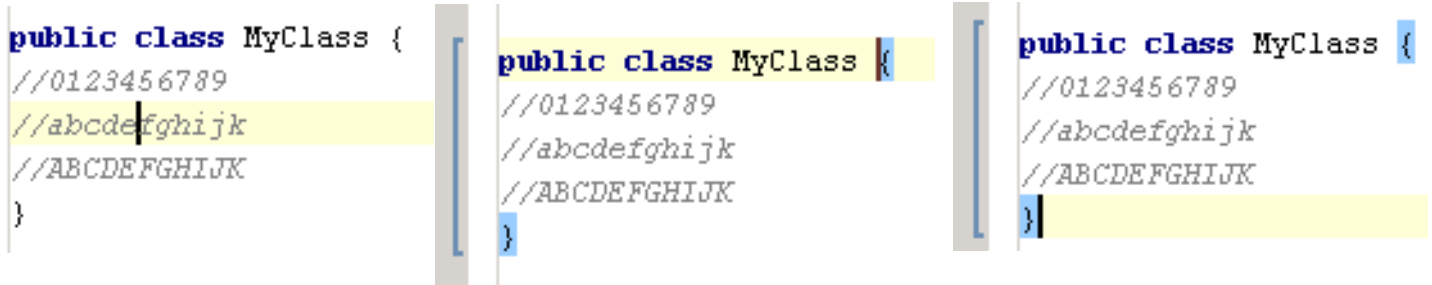


Figure 6.14. Move to code block start / end (745,746,747)

Move to code block start with selection CTRL-SHIFT-]

Place the caret within a code block. Clicking CTRL-SHIFT-] will select the text between the caret and the start of the code block (and place the caret at the beginning of the code block).

20021010TTT does this work??

Move to code block end with selection CTRL-SHIFT-[

Place the caret within a code block. Clicking CTRL-SHIFT-[will select the text between the caret and the end of the code block (and place the caret at the end of the code block).

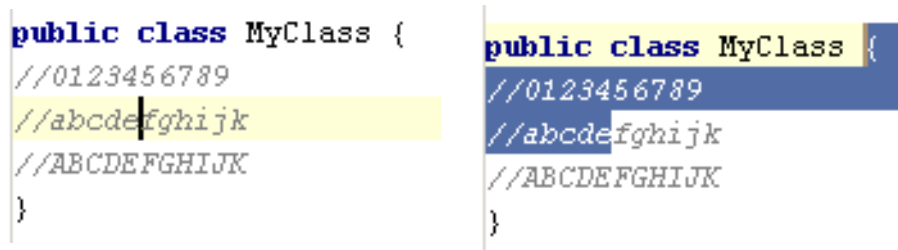


Figure 6.15. Move to code block start with select (748,749)

7.2.4.3. Within line

Move line end END

Place the caret in a line. Clicking END will move the caret to the end of the line.



```
//abc 0123456789 def | //abc 0123456789 def |
```

Figure 6.16. Move to line end ([750,751](#))

Move line start HOME

Place the caret in a line. Clicking HOME will move the caret to the start of the line.

Move line end with selection SHIFT-END

Place the caret in a line. Clicking SHIFT-HOME will move the caret to the start of the line and select the text between the caret initial and final position.



```
//abc 0123456789 def | //abc 0123456789 def |
```

Figure 6.17. Move to line end with select ([750,752](#))

Move line start with selection SHIFT-HOME

Place the caret in a line. Clicking SHIFT-HOME will move the caret to the start of the line and select the text between the caret initial and final position.

7.2.4.4. Within text

Move to next word CTRL-RIGHT

Place the caret anywhere in the class text. Clicking CTRL-RIGHT will move the caret to the start of the next word.

Move to previous word CTRL-LEFT

Place the caret anywhere in the class text. Clicking CTRL-LEFT will move the caret to the start of the word the caret was placed in or (if the caret was not placed in a word) to the start of the word before the caret.



```
public class MyClass {   public class MyClass {   public class MyClass {
```

Figure 6.18. Move to next / previous word ([756.757.758](#))

Move to next word with selection CTRL-SHIFT-RIGHT

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-RIGHT will move the caret to the start of the next word and select the text between the caret initial and final position.

Move to previous word with selection CTRL-SHIFT-LEFT

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-LEFT will move the caret [to the start of the word the caret was placed in or (if the caret was not placed in a word) to the start of the word before the caret] and [select the text between the caret initial and final position].

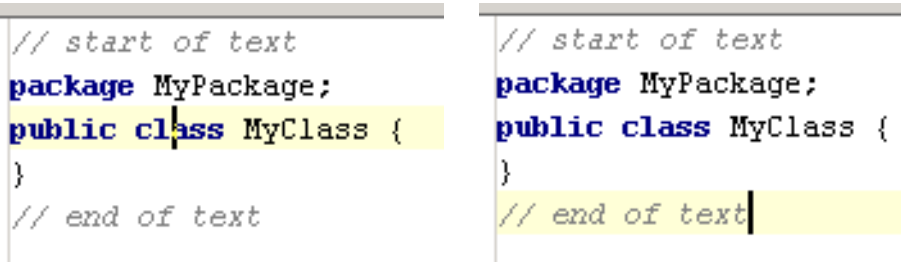


```
public class MyClass {   public class MyClass {   public class MyClass {
```

Figure 6.19. Move to next / previous word with select ([756.759.760](#))

Move text end CTRL-END

Place the caret anywhere in the class text. Clicking CTRL-END will move the caret to the end of the text.



```
// start of text
package MyPackage;
public class MyClass {
}
// end of text

// start of text
package MyPackage;
public class MyClass {
}
// end of text
```

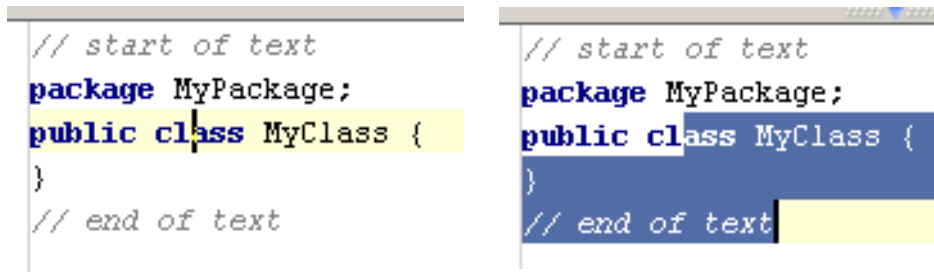
Figure 6.20. Move to text end ([753.754](#))

Move text start CTRL-HOME

Place the caret anywhere in the class text. Clicking CTRL-HOME will move the caret to the start of the text.

Move text end with selection CTRL-SHIFT-END

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-END will move the caret to the end of the text and select the text between the caret initial and final position.



```
// start of text
package MyPackage;
public class MyClass {
}
// end of text
```

```
// start of text
package MyPackage;
public class MyClass {
}
// end of text
```

Figure 6.21. Move to text end with select ([753.755](#))

Move text start with selection CTRL-SHIFT-HOME

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-HOME will move the caret to the start of the text and select the text between the caret initial and final position.

7.2.4.5. Page

Page down PAGE DOWN

Place the caret anywhere in the class text. Clicking PAGE DOWN will scroll down to the next visible page.



Figure 6.22. Page down ([761.762](#))

Page up PAGE UP

Place the caret anywhere in the class text. Clicking PAGE UP will scroll up to the next visible page.

Page down with selection SHIFT-PAGE DOWN

Place the caret anywhere in the class text. Clicking PAGE DOWN will scroll down to the next visible page and select the text between the caret initial and final position (the caret final position will be in the same relative location on the new page).

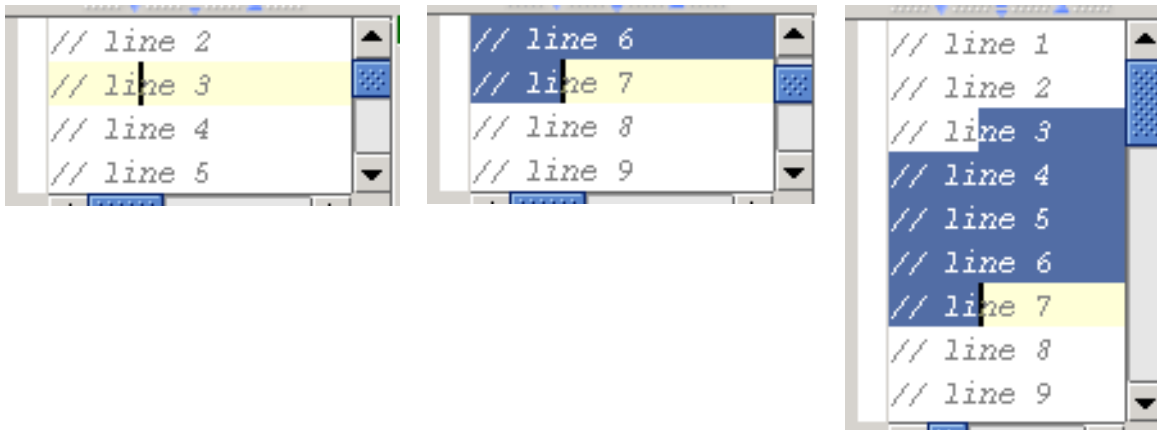


Figure 6.23. Page down with selection ([763.764.765](#))

Page up with selection SHIFT-PAGE UP

Place the caret anywhere in the class text. Clicking PAGE UP will scroll up to the next visible page and select the text between the caret initial and final position (the caret final position will be in the same relative location on the new page).

Go page bottom CTRL-PAGE DOWN

Place the caret anywhere in the class text. Clicking CTRL-PAGE DOWN will move the cursor to the bottom of the page (in the same vertical location).



Figure 6.24. Go page bottom ([766.767](#))

Go page top CTRL-PAGE UP

Place the caret anywhere in the class text. Clicking CTRL-PAGE UP will move the cursor to the top of the page (in the same vertical location).

Go page bottom with selection CTRL-SHIFT-PAGE DOWN

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-PAGE DOWN will move the cursor to the bottom of the page (in the same vertical location) and select the text between the cursor initial and final location.

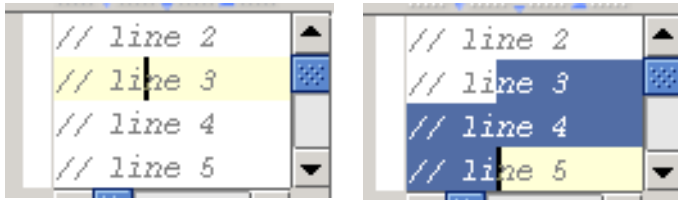


Figure 6.25. Go page bottom with selection ([766,768](#))

Go page top with selection CTRL-SHIFT-PAGE UP

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-PAGE UP will move the cursor to the top of the page (in the same vertical location) and select the text between the cursor initial and final location.

7.2.4.6. Scroll

Scroll down CTRL-DOWN, CTRL-SHIFT-DOWN

Place the caret anywhere in the class text. Clicking CTRL-DOWN will move the displayed page down 1 line and keep the cursor in the same relative page position.



Figure 6.26. Scroll down ([769.770](#))

Scroll up CTRL-UP, CTRL-SHIFT-UP

Place the caret anywhere in the class text. Clicking CTRL-UP will move the displayed page up 1 line and keep the cursor in the same relative page position.

Scroll to center CTRL-M

Place the caret anywhere in the class text. Clicking CTRL-M will move the displayed page so that the caret is in the center.

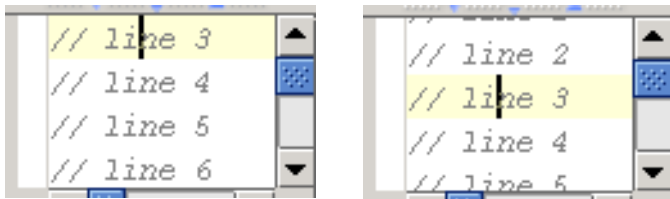


Figure 6.27. Scroll to center ([772.773](#))

[20021012TTT no hotkeys by default for move scroll XX ?? are these repeated functions??](#)

Move down and scroll

Place the caret anywhere in the class text. Move down and scroll will move the displayed page down 1 line and keep the cursor in the same relative page position.



Figure 6.28. Move down and scroll ([769.770](#))

Move up and scroll

Place the caret anywhere in the class text. Move up and scroll will move the displayed page up 1 line and keep the cursor in the same relative page position.

Move down and scroll with selection

Place the caret anywhere in the class text. Move down and scroll with selection will [move the displayed page down 1 line and keep the cursor in the same relative page position] and [select the text between the cursor initial and final location].

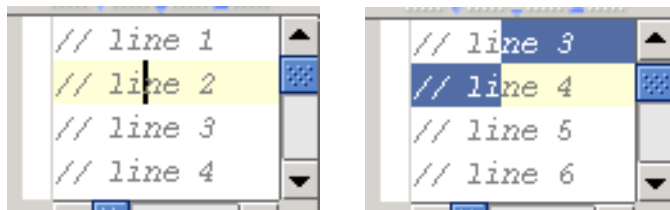


Figure 6.29. Move down and scroll with selection ([769.771](#))

Move up and scroll with selection

Place the caret anywhere in the class text. Move up and scroll with selection will [move the displayed page up 1 line and keep the cursor in the same relative page position] and [select the text between the cursor initial and final location].

7.2.5. Indent / tabs / lines

Tab TAB

Indent selection TAB

Place the caret anywhere in the class text. Clicking TAB will insert several spaces before the caret.

The image shows two side-by-side code snippets. The left snippet shows a line of code with a caret at the end: `// line 3`. The right snippet shows the same line of code after pressing TAB, with the text indented: `// li ne 3`.

Figure 6.30. Indent selection (tab) [\(774,775\)](#)

Unindent selection SHIFT-TAB

Place the caret within the indent of indented text (the indent must be at the beginning of the line). Clicking SHIFT-TAB will removed 4 (default) spaces (equivalent to a TAB). The caret will not move.

The image shows two side-by-side code snippets. The left snippet shows a line of code with a caret at the beginning of the indent: `123456789 */
// line 3`. The right snippet shows the same line of code after pressing SHIFT-TAB, with the text unindented: `123456789 */
// line 3`.

Figure 6.31. Unindent selection (tab) [\(776,777\)](#)

Start new line SHIFT-ENTER

Place the caret anywhere in the class text. Clicking SHIFT-ENTER will insert a new line after the line the caret is located in and move the caret to the start of the new line.

The image shows two side-by-side code snippets. The left snippet shows a line of code with a caret at the end: `// line 3
// line 4`. The right snippet shows the same code after pressing SHIFT-ENTER, with a new line inserted: `// line 3

// line 4`.

Figure 6.32. Start new line [\(778,779\)](#)

Split line CTRL-ENTER

Place the caret anywhere in a line. Clicking CTRL-ENTER will [insert a new line after the line the caret is located in] and [move the caret and the text located after the caret to the new line].

The image shows two side-by-side code snippets. The left snippet shows a line of code with a caret in the middle: `// line 3
// line 4`. The right snippet shows the same code after pressing CTRL-ENTER, with the line split: `// li
ne 3
// line 4`.

Figure 6.33. Split line [\(778,780\)](#)

Join lines CTRL-SHIFT-J

Line with caret and next line

Place the caret anywhere in the class text. Clicking CTRL-SHIFT-J will join the line that the caret is in and the next line.

The image shows two side-by-side code snippets. The left snippet shows two lines of code with a caret at the end of the first line: `// line 3
// line 4`. The right snippet shows the same code after pressing CTRL-SHIFT-J, with the lines joined: `// line 3 line 4`.

Figure 6.34. Join line with caret and next line [\(781,782\)](#)

Selected lines

Select text in multiple lines. Clicking CTRL-SHIFT-J will join the lines in which text is selected.

The image shows two side-by-side code snippets. The left snippet shows three lines of code with the first two lines selected: `// line 3
// line 4
// line 5`. The right snippet shows the same code after pressing CTRL-SHIFT-J, with the selected lines joined: `// line 3 line 4 line 5`.

Figure 6.35. Join selected lines [\(783,784\)](#)

7.2.6. Modify text

Toggle case CTRL-SHIFT-U

Select text. Clicking CTRL-SHIFT-U will:

- [if the text contains upper-case characters] : Make the selected text lower case.
- [if the text contains NO upper-case characters] : Make the selected upper case.

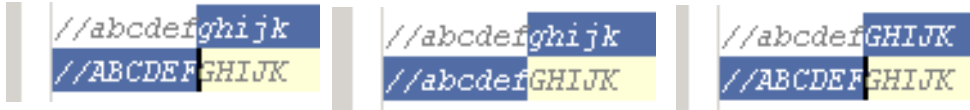


Figure 6.36. Toggle text case ([785.786.787](#))

~~?? Choose lookup item~~

~~?? Choose lookup item replace~~

~~?? Next template variable~~

~~?? Previous template variable~~

7.3. Find / Navigation

IDEA provides the following find and navigation functions:

- 7.3.1. Bookmarks (page 124)
- 7.3.2. Find / Replace (page 126)
- 7.3.3. Go to (page 133)

7.3.1. Bookmarks

IDEA support the following functions for bookmarks:

- 7.3.1.1. Create (page 124)
- 7.3.1.2. Show (page 124)
- 7.3.1.3. View source (page 124)
- 7.3.1.4. Go to (page 125)
- 7.3.1.5. Describe (page 125)
- 7.3.1.6. Move (page 125)
- 7.3.1.7. Remove (page 125)

7.3.1.1. Create

6.16. Place the cursor in a file.

6.17. Select **Edit | Toggle Bookmark**. A bookmark appears.

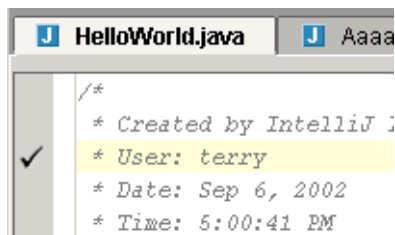


Figure 6.37. Bookmark in file (672)

6.18. Create bookmarks in several other files.

7.3.1.2. Show

6.19. Select **Edit | Show Bookmarks**. The dialog “Editor bookmarks” appears.

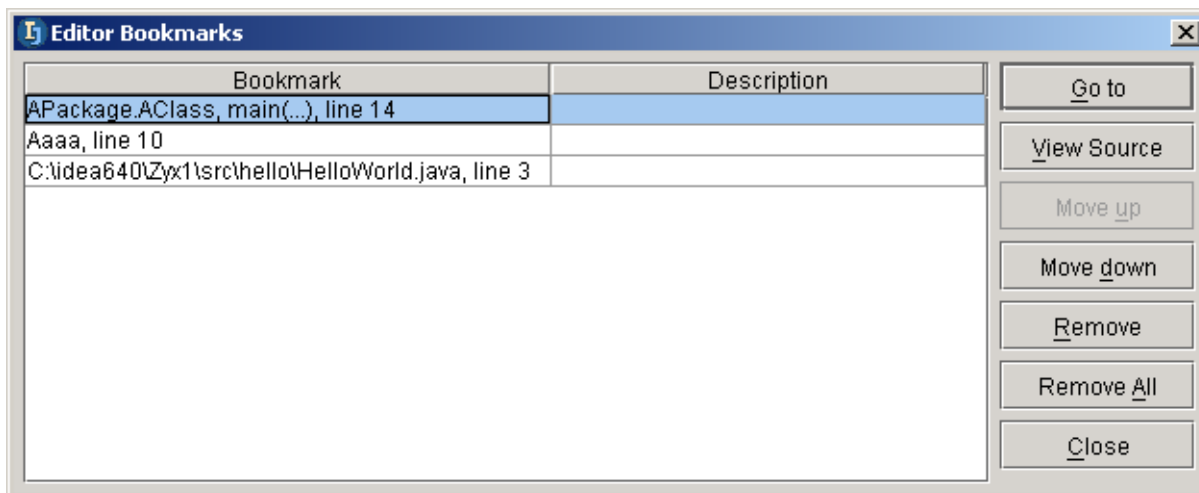


Figure 6.38. Bookmarks editor (673)

7.3.1.3. View source

6.20. Select a bookmark.

6.21. Click **View Source**. The editor of the bookmark source is displayed (and the bookmark editor is not closed).

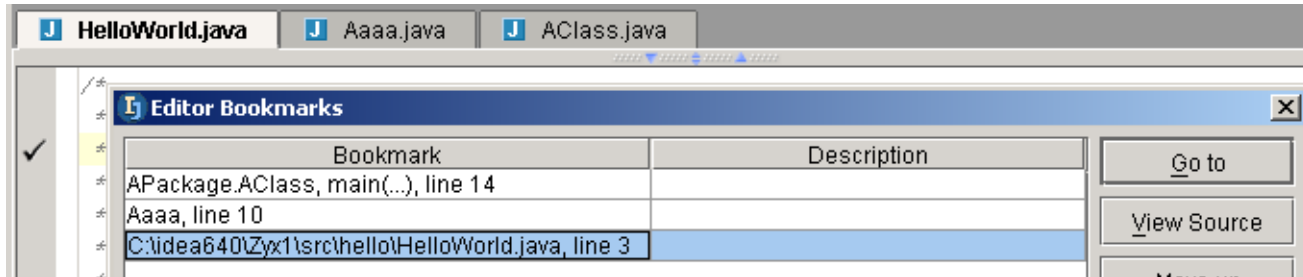


Figure 6.39. View source (674)

7.3.1.4. Go to

Goto is the same as View Source, except that the bookmark editor is closed.

7.3.1.5. Describe

6.22. Click in the column **Description** for a bookmark.

6.23. Type in the description text.

Bookmark	Description
APackage.AClass, main(...), line 14	My first bookmark
Aaaa, line 10	My second bookmark
C:\idea640\Zyx1\src\hello\HelloWorld.java, line 3	

Figure 6.40. Bookmark description (675)

7.3.1.6. Move

6.24. Select a bookmark.

6.25. Click **Move up** / **Move down** to move the bookmark.

7.3.1.7. Remove

6.26. Select a bookmark.

6.27. Click **Remove** to delete the bookmark.

Click **Remove all** to delete all bookmarks.

7.3.2. Find / Replace

IDEA supports the following Find / Replace variations:

- 7.3.2.1. Find (in file) (page 126)
- 7.3.2.2. Highlight usages in file (page 127)
- 7.3.2.3. Find usages in file (page 127)
- 7.3.2.4. Find Next (page 127)
- 7.3.2.5. Find Previous (page 128)
- 7.3.2.6. Find in path (page 128)
- 7.3.2.7. Find usages (in path) (page 129)
- 7.3.2.8. Replace (in file) (page 129)
- 7.3.2.9. Replace in path (page 130)
- 7.3.2.10. Find Word at caret (page 131)
- 7.3.2.11. Incremental search (page 132)

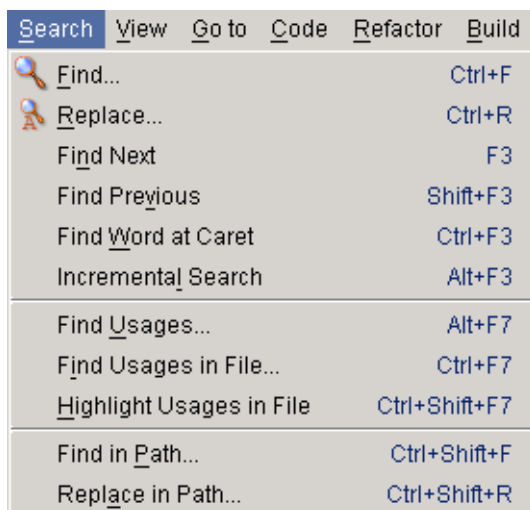


Figure 6.41. Find/replace menu (676)

7.3.2.1. Find (in file)

6.28. Open a file.

6.29. Select a word.

6.30. Select **Search | Find**. The dialog “Find text” appears.

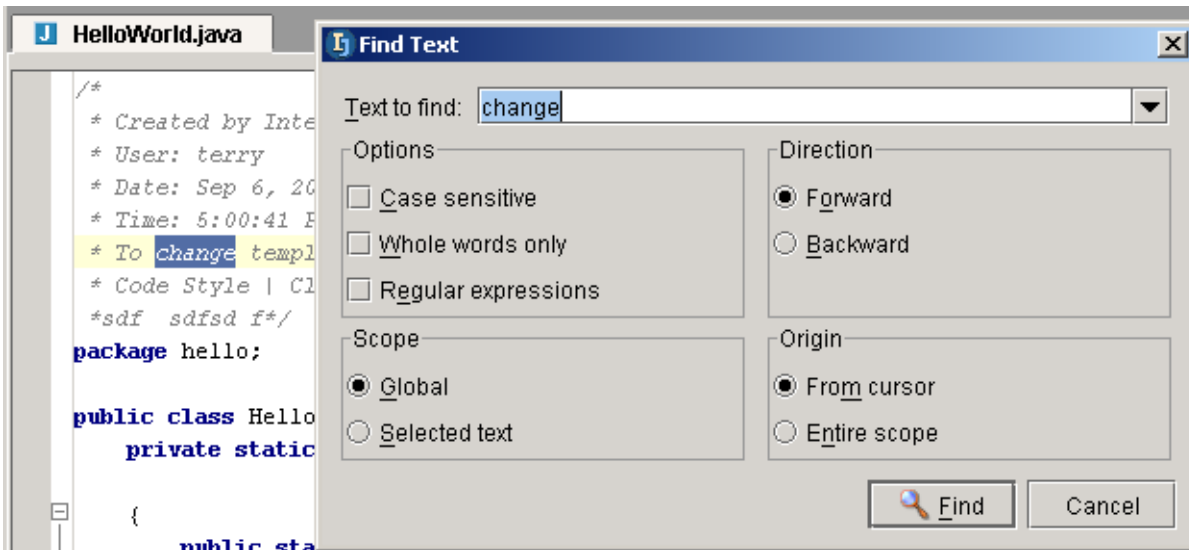


Figure 6.42. Dialog “Find text” (677)

Note that previously searched text is available in the drop-down list “Text to find”.

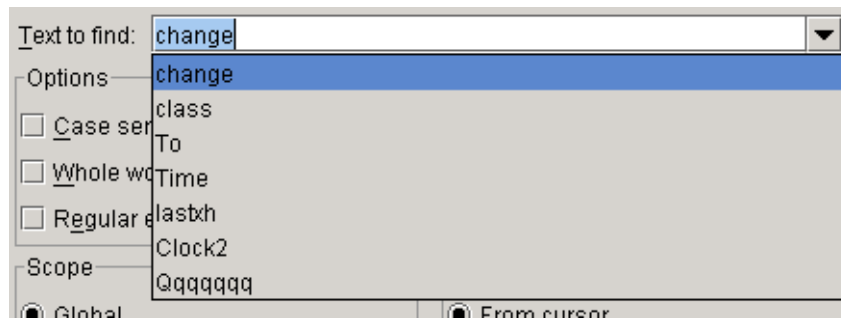


Figure 6.43. Drop-down list “Text to find” (678)

6.31. Click **Find** to find the text.

7.3.2.2. Highlight usages in file

6.32. Place the cursor within a code element (do not select).

6.33. Select **Search | Highlight usages in file**. The usage occurrences of the word is highlighted.

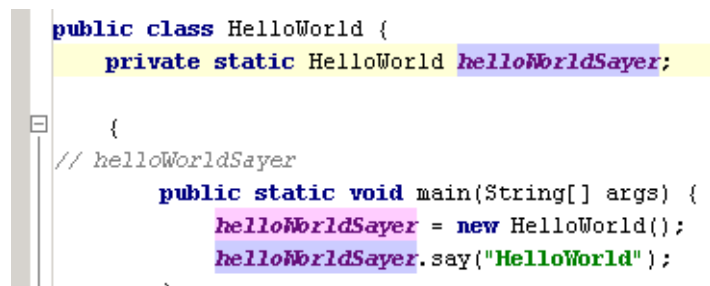


Figure 6.44. Highlight usages (687)

Note in the above example that “helloWorldSayer” in the comment was not highlighted.

7.3.2.3. Find usages in file

This is the same as highlighting the usages, except that the usages are found (ie, selected, and you can use F3 to find the next).

7.3.2.4. Find Next

6.34. Select **Search | Find next** to find the next occurrence.

7.3.2.5. Find Previous

6.35. Select **Search | Find previous** to find the previous occurrence.

7.3.2.6. Find in path

6.36. Select a word.

6.37. Select **Search | Find in path...** The dialog “Find in path” appears.

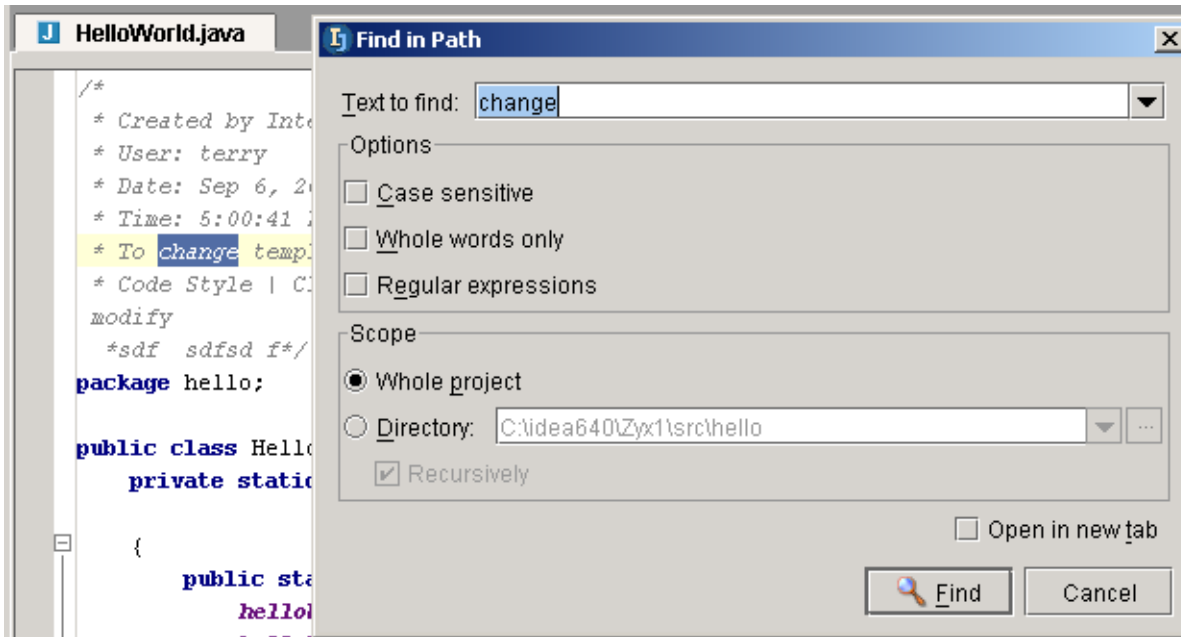


Figure 6.45. Dialog “Find in path” (681)

6.38. Click **Find** to find the text. The tool “Find” appears with a list of packages and dirs in which the text was found.

6.39. Expand a package or directory and double-click on an occurrence. The source file is opened.

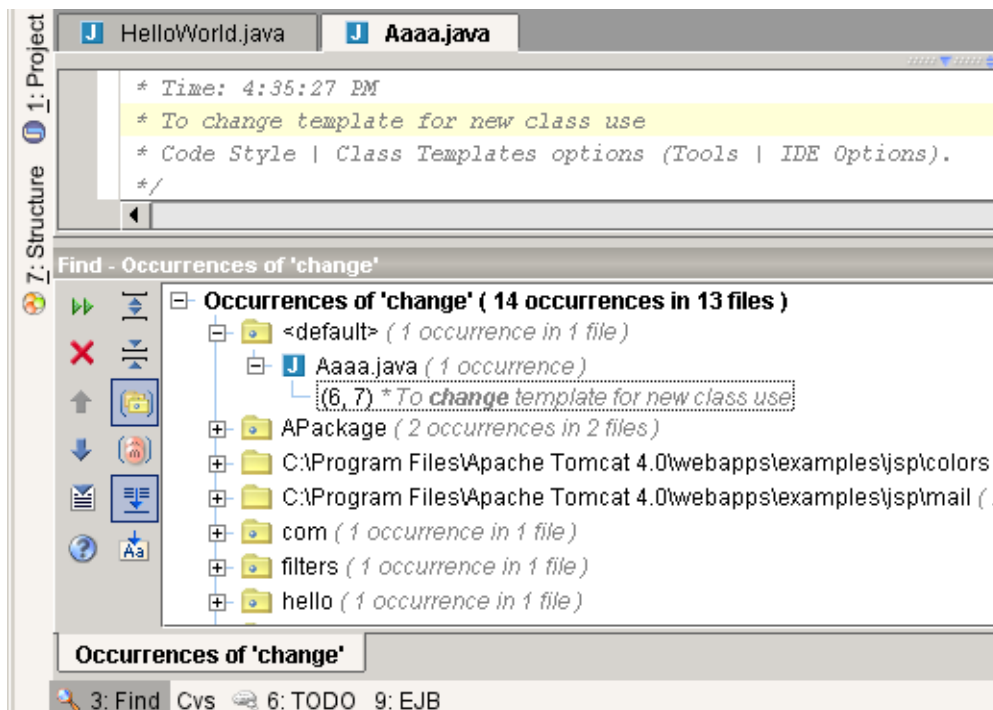


Figure 6.46. Tool “Find” (682)

7.3.2.7. Find usages (in path)

6.40. Place the cursor within a code element (do not select).

6.41. Select **Search | Find usages**. The “Find usages” dialog appears.

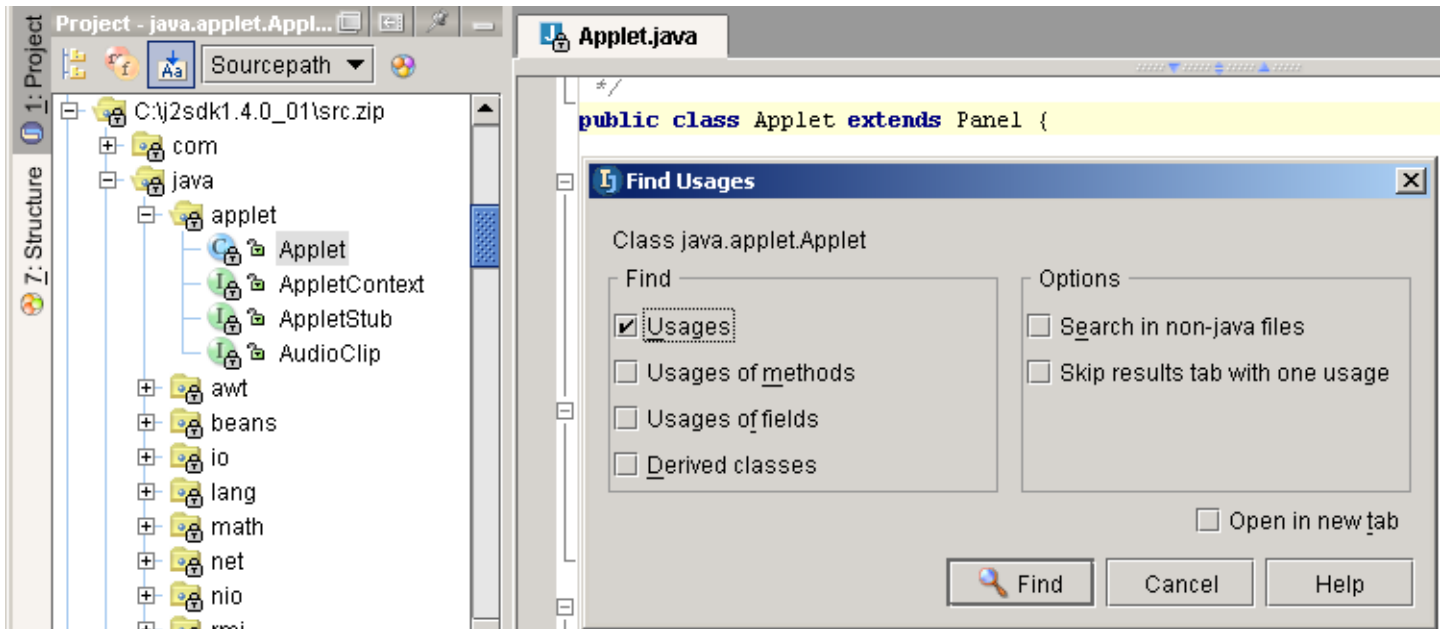


Figure 6.47. Dialog “Find usages” (688)

6.42. Click **Find**. The results are displayed.

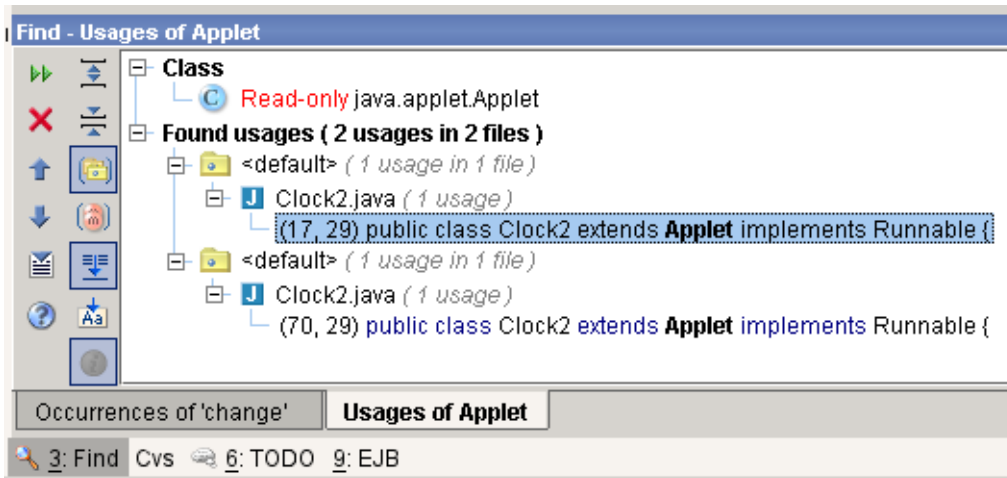


Figure 6.48. Find tool results (689)

7.3.2.8. Replace (in file)

6.43. Open a file.

6.44. Select a word.

6.45. Select **Search | Replace**. The dialog “Replace text” appears.

6.46. For “Replace with:” Enter the new text.

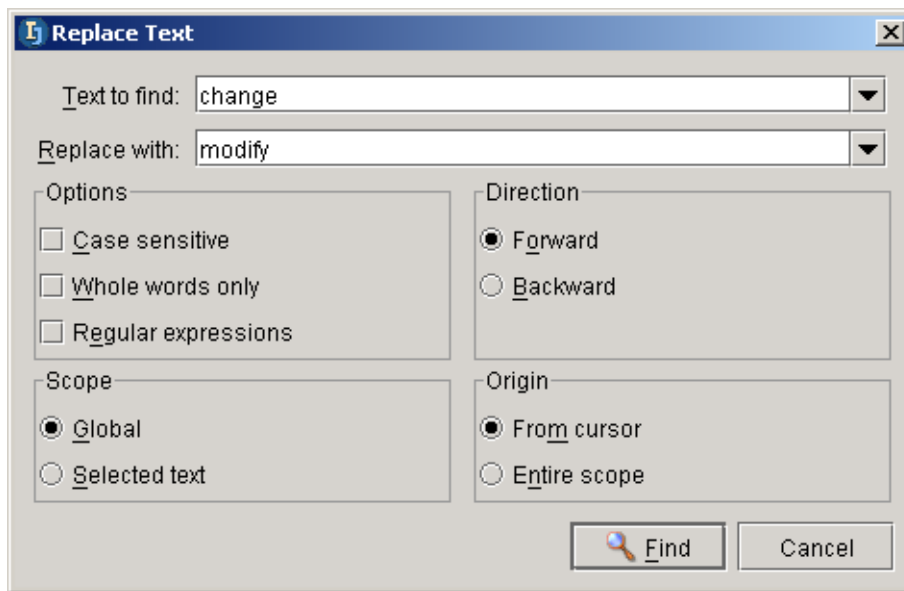


Figure 6.49. Dialog “Replace text” (679)

6.47. Select **Find**. The dialog “Replace” appears.

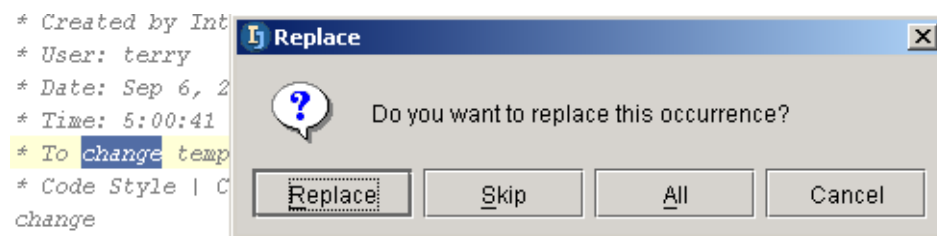


Figure 6.50. Dialog “Replace text” (680)

6.48. Select

- **Replace** to replace the selection and find the next occurrence.
- **Skip** to not replace the selection and to find the next occurrence.
- **All** to replace all occurrences.
- **Cancel** to not replace any remaining occurrences.

7.3.2.9. Replace in path

6.49. Open a file.

6.50. Select a word.

6.51. Select **Search | Replace in path**. The dialog “Replace in path” appears.

6.52. For “Replace with:”: Enter the new text.

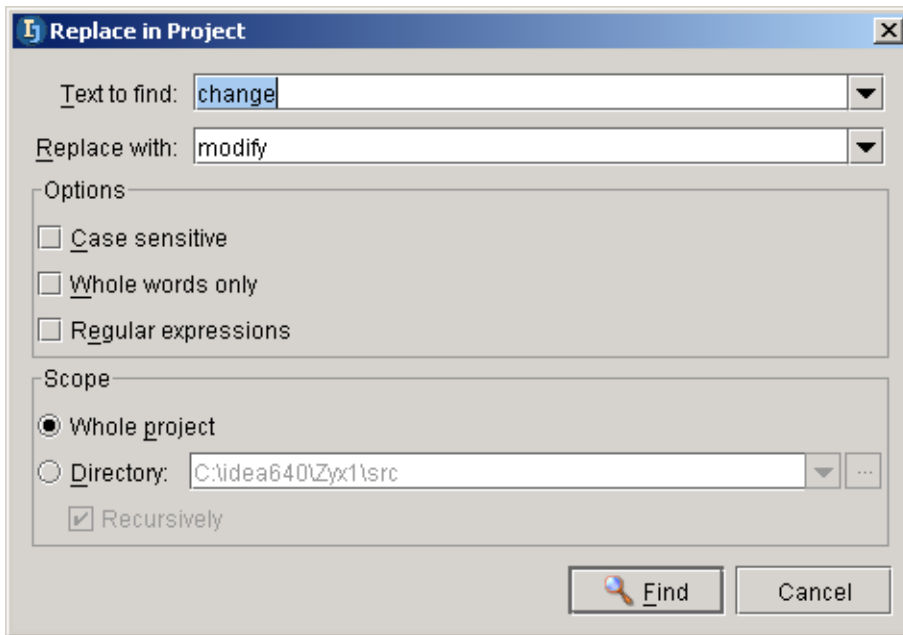


Figure 6.51. Dialog “Replace in project” (683)

6.53. Select **Find**. The dialog “Replace” appears.

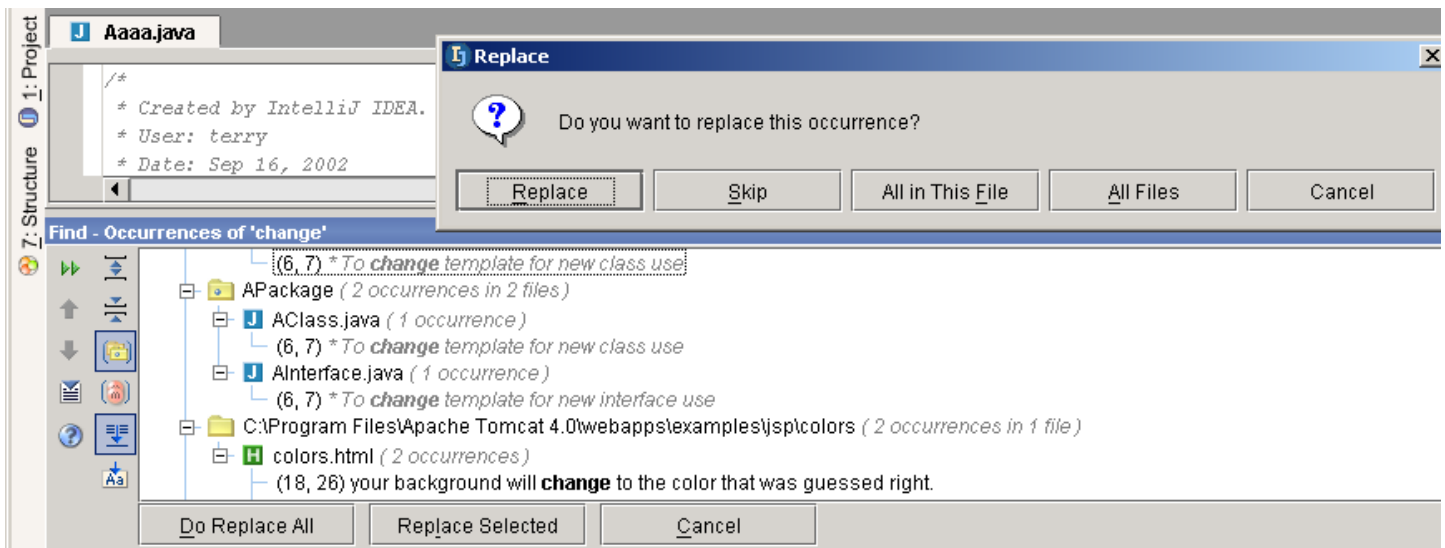


Figure 6.52. Dialog “Replace” (684)

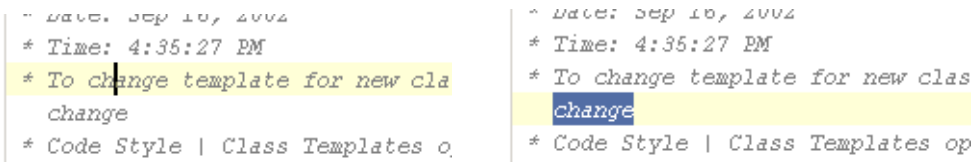
6.54. Select

- **Replace** to replace the selection and find the next occurrence.
- **Skip** to not replace the selection and to find the next occurrence.
- **All in this file** to replace all occurrences in the selected file.
- **All file** to replace all occurrences in all files.
- **Cancel** to not replace any remaining occurrences.

7.3.2.10. Find Word at caret

6.55. Place the cursor within a word (do not select the word).

6.56. Select **Search | Find word at caret**. The next occurrence of the word is highlighted.



The image shows two side-by-side screenshots of a code editor. Both screenshots display the same text: `~ Date: Sep 10, 2004`, `* Time: 4:35:27 PM`, `* To change template for new cla`, `change`, and `* Code Style | Class Templates o`. In the left screenshot, a vertical caret is positioned at the start of the word 'change' on the third line. In the right screenshot, the word 'change' is highlighted in blue, indicating the search results.

Figure 6.53. Find word at caret ([685,686](#))

7.3.2.11. Incremental search

Performs an incremental search

7.3.3. Go to

IDEA provides the following “go to” functions:

- **7.3.3.1. Class (page 133)**
- **7.3.3.2. File (page 133)**
- **7.3.3.3. Line (page 134)**
- **7.3.3.4. Declaration (page 134)**
- **7.3.3.5. Implementation (page 134)**
- **7.3.3.6. Type declaration (page 134)**
- **7.3.3.7. Super method (page 135)**
- **7.3.3.8. Next / Previous highlighted error (page 135)**
- **7.3.3.9. Next / Previous method (page 135)**
- **7.3.3.10. Back / Forward (page 136)**
- **7.3.3.11. Last Edit location (page 136)**

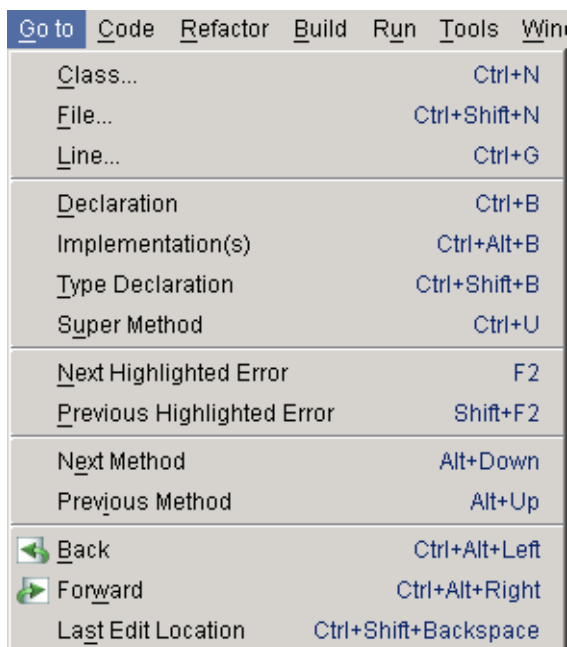


Figure 6.54. Goto menu (690)

7.3.3.1. Class

6.57. Select **Go to | Class**. The dialog “Enter class name” appears.

6.58. Enter the class name. Note that IDEA will suggest names based on the letters you type in.

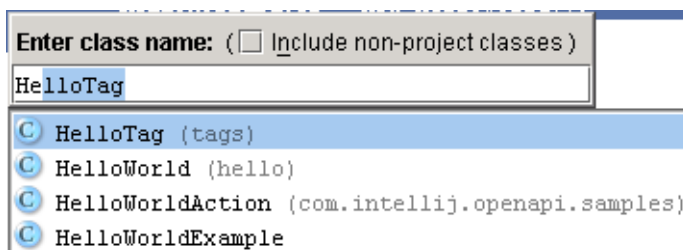


Figure 6.55. Dialog “Enter class name” (691)

6.59. Click **Enter**. The source for the class is opened in an editor.

7.3.3.2. File

6.60. Select **Go to | File**. The dialog “Enter file name” appears.

6.61. Enter the file name. Note that IDEA will suggest names based on the letters you type in.

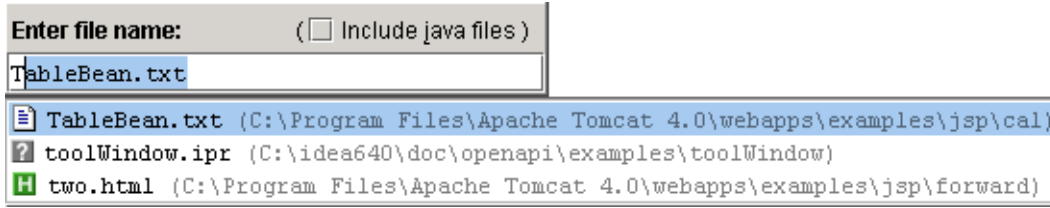


Figure 6.56. Dialog “Enter file name” (692)

6.62. Click **Enter**. The file is opened in an editor.

7.3.3.3. Line

Select **Go to / Line** to go to the specified line in the selected editor.

7.3.3.4. Declaration

6.63. Place the cursor in a code element.

6.64. Select **Go to | Declaration**. The source file for the declaration is opened.

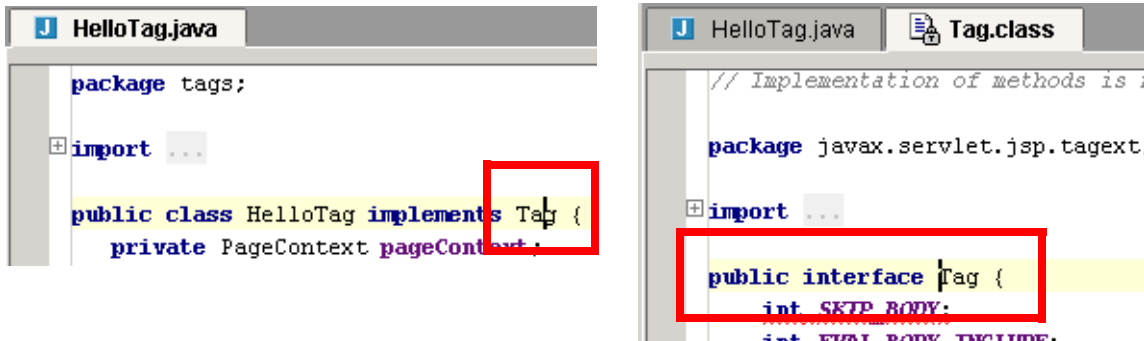


Figure 6.57. Go to declaration (693,694)

7.3.3.5. Implementation

6.65. Open an Interface file.

6.66. Place the cursor on the interface name.

6.67. Select **Go to | Implementation**. A list of the implementing classes appears in a popup (if more than 1 exists).

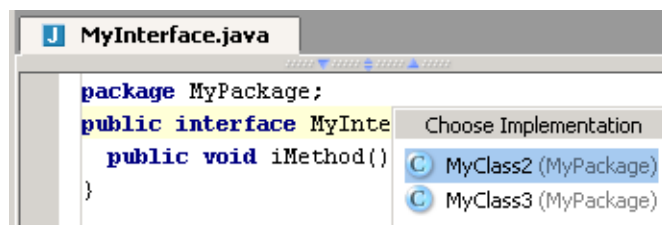


Figure 6.58. Go to implementation (194)

6.68. Select an implementor to open.

7.3.3.6. Type declaration

6.69. Place the cursor in a code element.

6.70. Select **Go to | Type declaration**. The source file for the declaration of the type is opened.

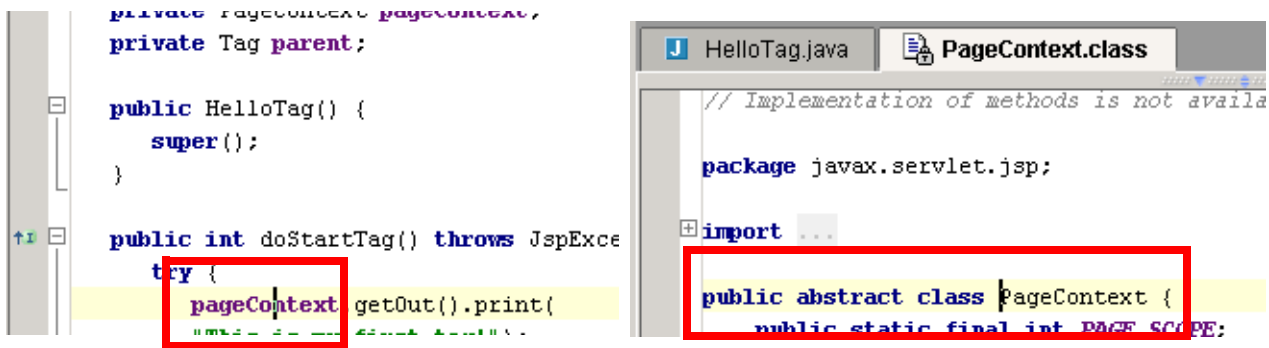


Figure 6.59. Go to type declaration (695,696)

7.3.3.7. Super method

6.71. Place the cursor in a code element.

6.72. Select **Go to | Super method**. The source file for the method of the super class is opened.

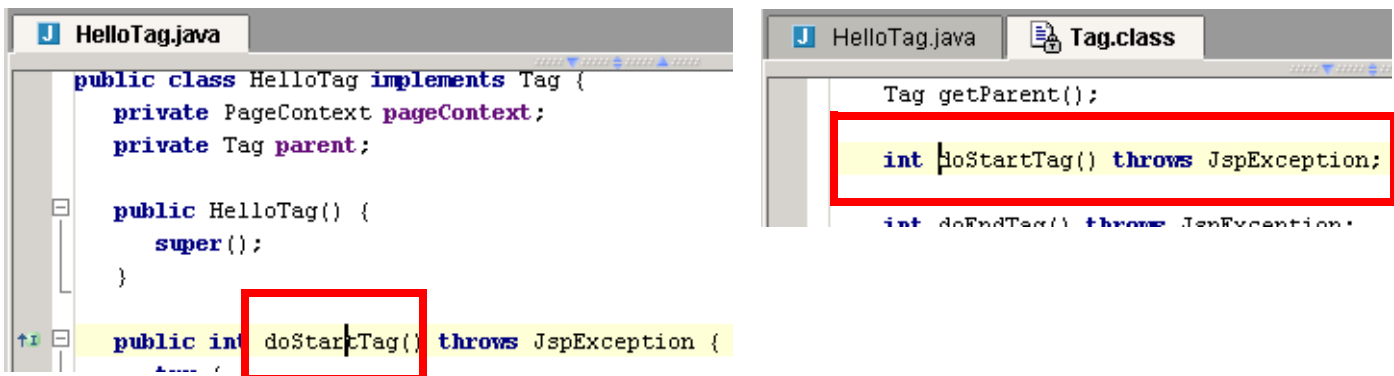


Figure 6.60. Go to super method (698,697)

7.3.3.8. Next / Previous highlighted error

Select **Go to | Next highlighted error** to place the cursor at the next highlighted error (typically red) in a source file.

Select **Go to | Previous highlighted error** for the previous error.

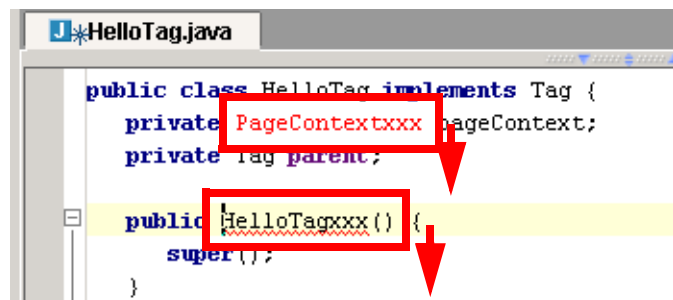
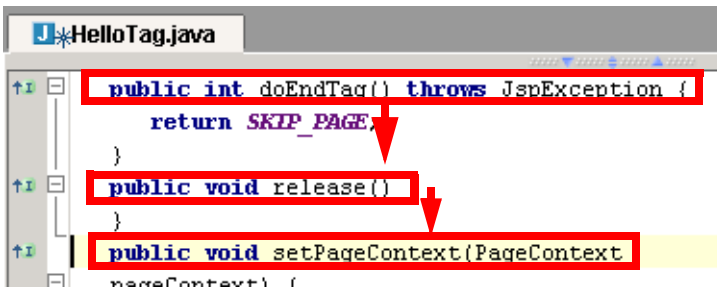


Figure 6.61. Next highlighted error (699)

7.3.3.9. Next / Previous method

Select **Go to | Next method** to place the cursor at the next method

Select **Go to | Previous method** for the previous method.



```
public int doEndTag() throws JspException {  
    return SKIP_PAGE;  
}  
  
public void release()  
}  
  
public void setPageContext(PageContext  
pageContext) {
```

Figure 6.62. Next method ([700](#))

7.3.3.10. Back / Forward

The **Back** and **Forward** buttons () are used to goto to different editors. The following simple example shows how it works.

- 6.73. Create 3 classes: **Class1**, **Class2**, **Class3**.
- 6.74. Open **Class1**.
- 6.75. Open **Class2**.
- 6.76. Open **Class3**.
- 6.77. Select **Go to | Back**. The Class2 editor is selected.
- 6.78. Close **Class1**.
- 6.79. Close **Class3**.
- 6.80. Select **Go to | Forward**. Class3 is opened in an editor.
- 6.81. Select **Go to | Back**. The Class2 editor is selected.
- 6.82. Select **Go to | Back**. Class1 is opened in an editor.

7.3.3.11. Last Edit location

Select **Go to / Last edit location** to place the cursor at the location of the last edit made in IDEA.

7.4. Colors and fonts X

IDEA simplifies editing tasks with colors and fonts that make it easier to recognize text functionality:

- 7.4.1. General X (page 137)
- 7.4.2. Java X (page 138)
- 7.4.3. HTML X (page 139)
- 7.4.4. XML X (page 140)
- 7.4.5. JSP X (page 141)
- 7.4.6. Custom X (page 142)
- 7.4.7. Color Schemes (page 143)

7.4.1. General X

Tab **General** shows general colors and fonts.

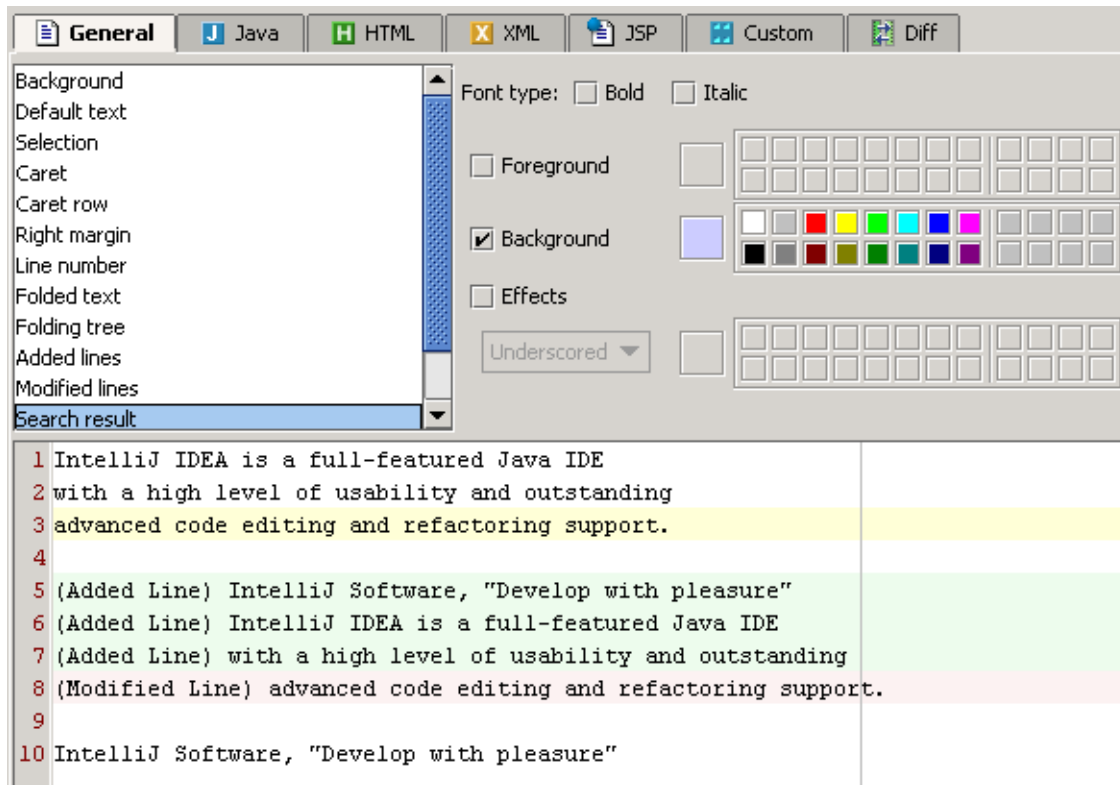


Figure 6.63. General colors/fonts (438)

7.4.2. Java X

Tab **Java** shows colors and fonts for Java files.

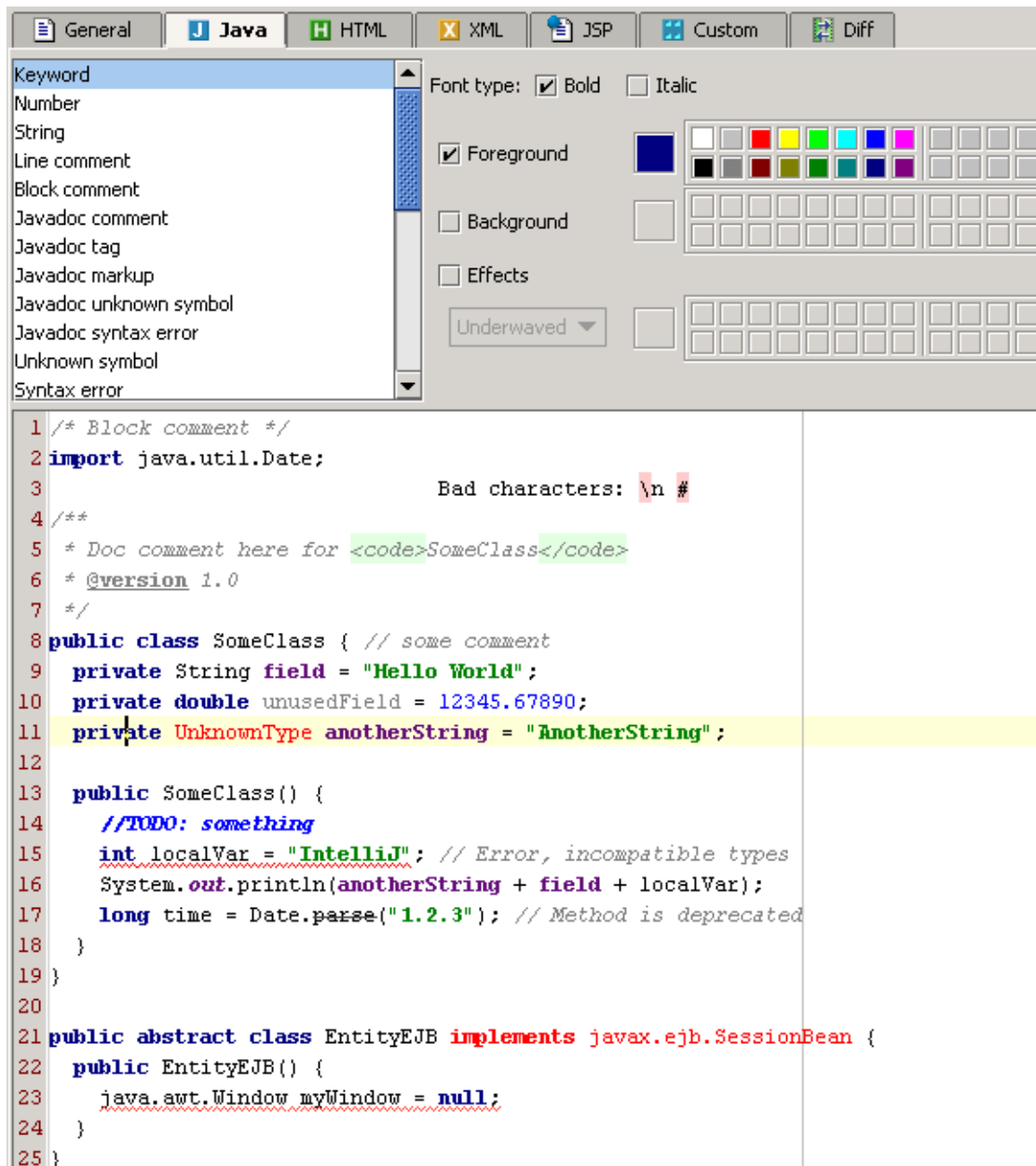


Figure 6.64. Colors/fonts for Java files (437)

7.4.3. HTML_X

Tab **HTML** shows colors and fonts for HTML files.

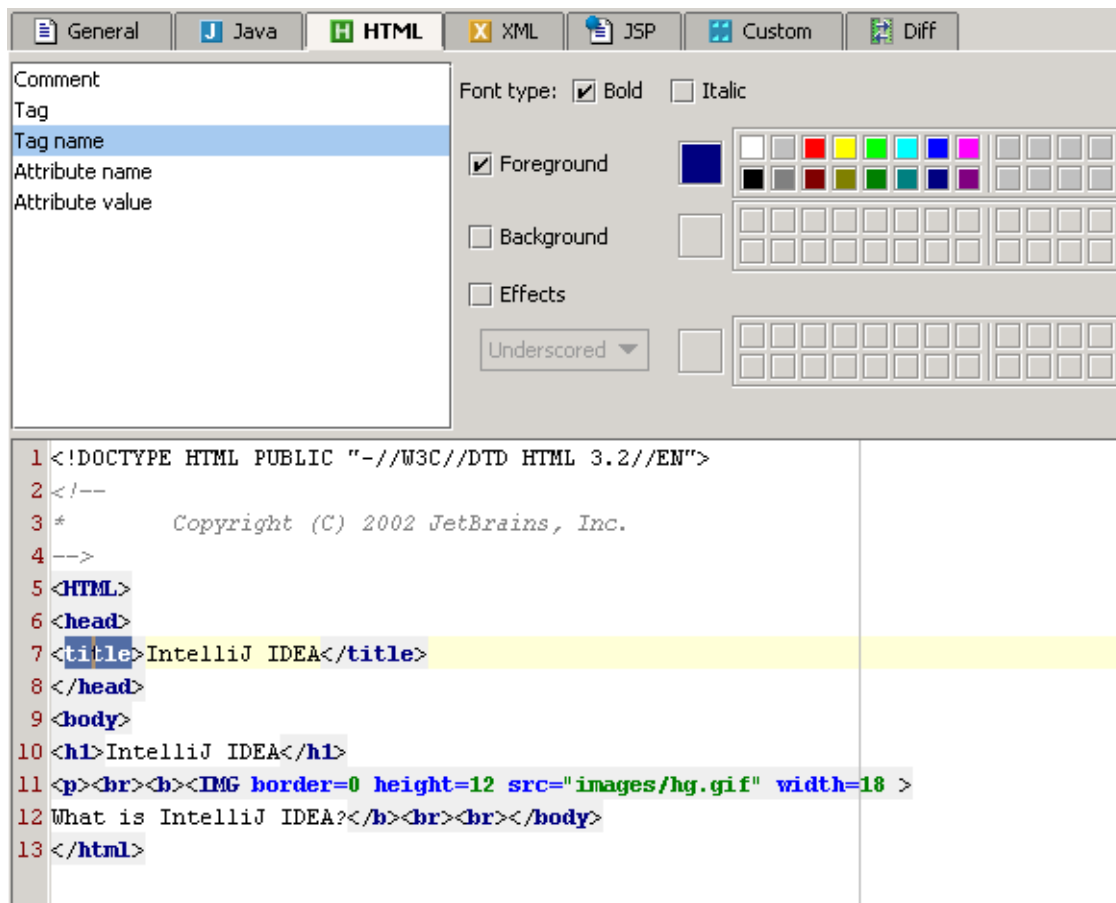


Figure 6.65. Colors/fonts for HTML files (436)

7.4.4. XML_X

Tab **XML** shows colors and fonts for XML files.

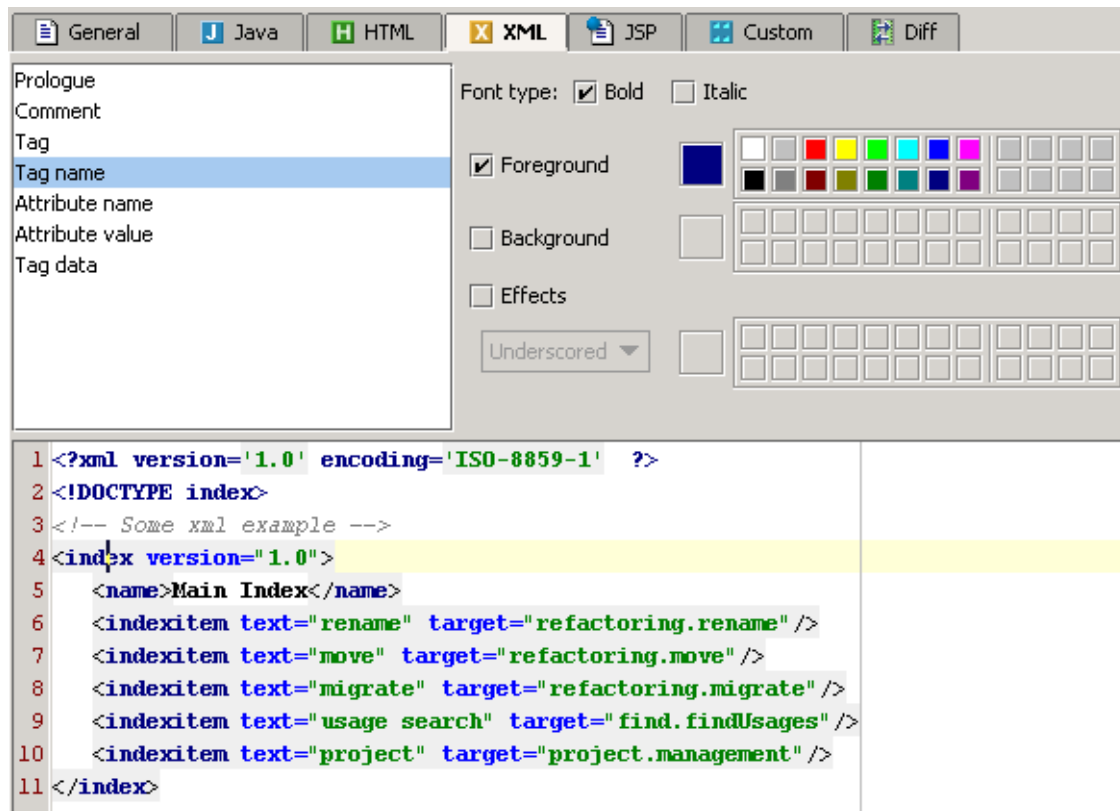


Figure 6.66. Colors/fonts for XML files [\(435\)](#)

7.4.5. JSP X

Tab **JSP** shows colors and fonts for JSP files.

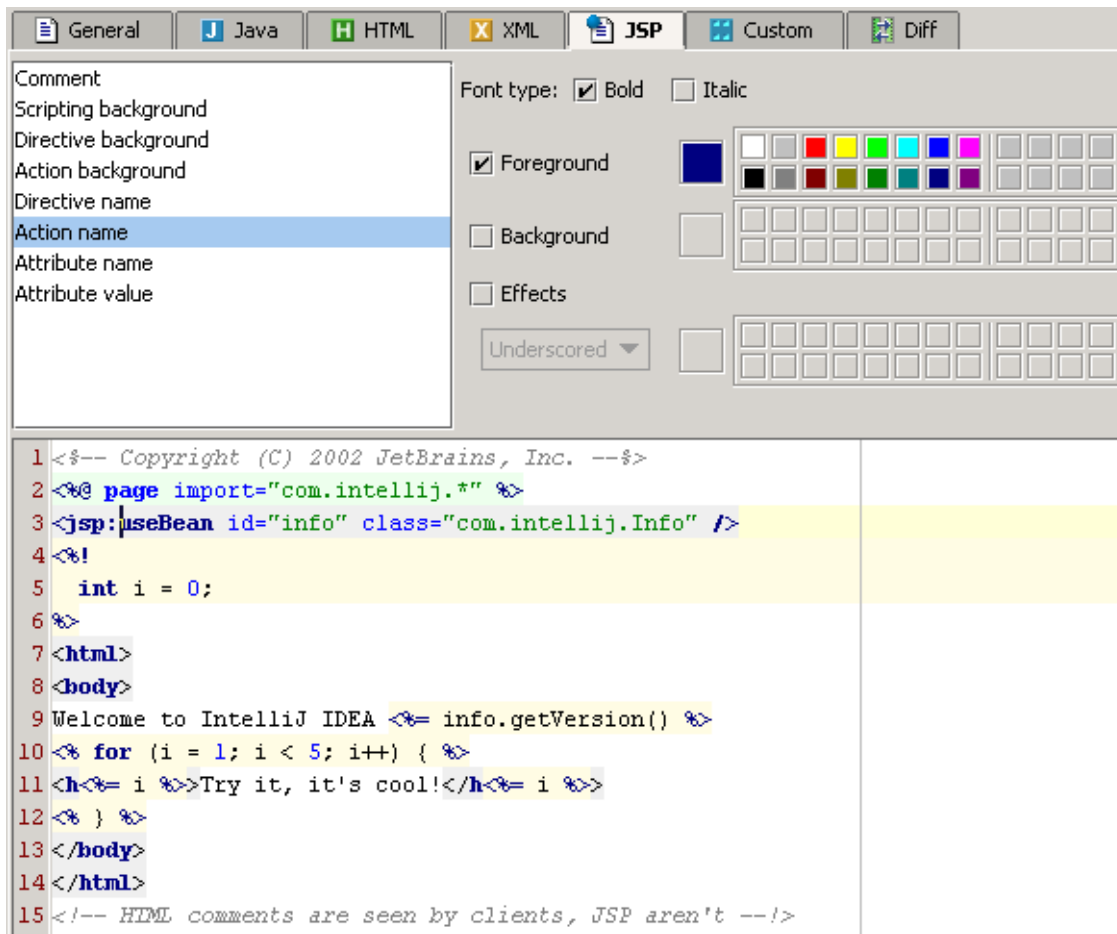


Figure 6.67. Colors/fonts for JSP files (434)

7.4.6. Custom X

Tab **Custom** shows custom colors and fonts.

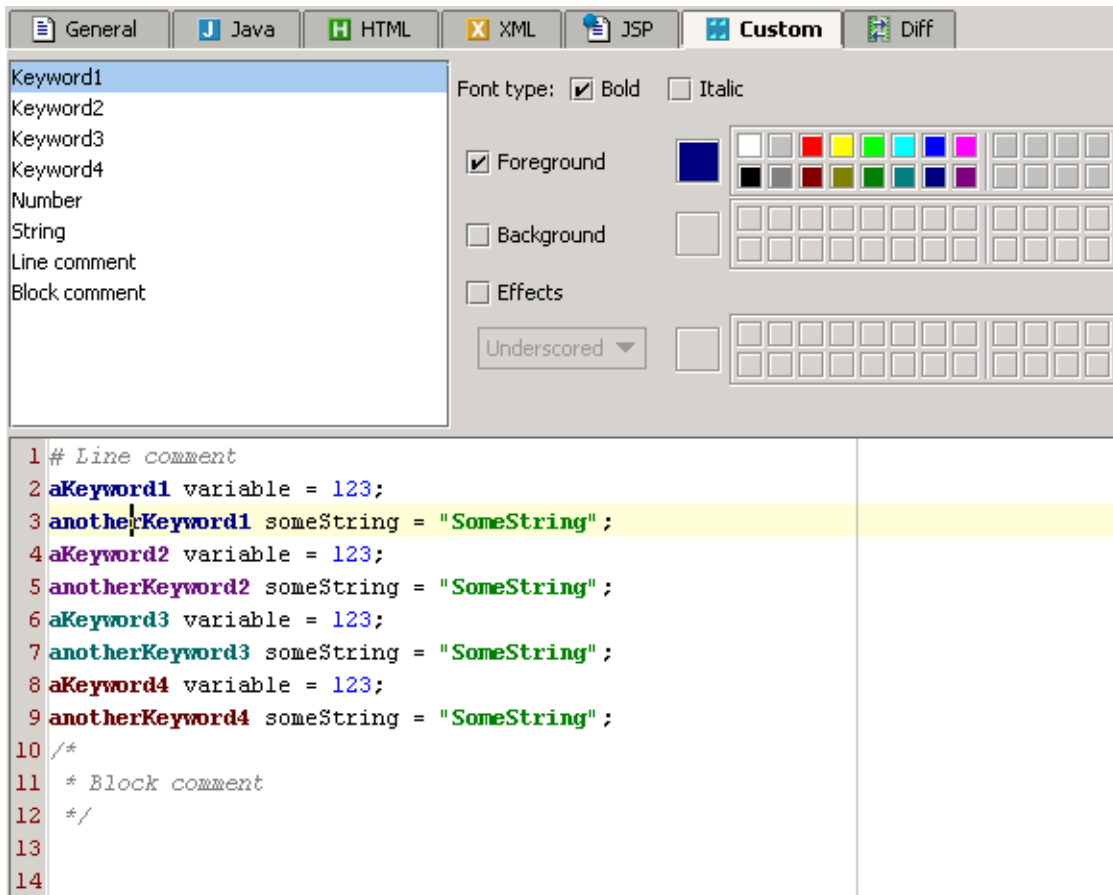


Figure 6.68. Custom colors/fonts (433)

7.4.7. Color Schemes

To use a custom color scheme, do the following:

- 7.4.7.1. Create (page 143)
- 7.4.7.2. Select (page 143)
- 7.4.7.3. Modify (page 143)
- 7.4.7.4. Apply (page 144)

7.4.7.1. Create

6.83. Click **Save as...**. The dialog “Save color scheme as” appears.

6.84. For “Name” enter **MyColorScheme**.

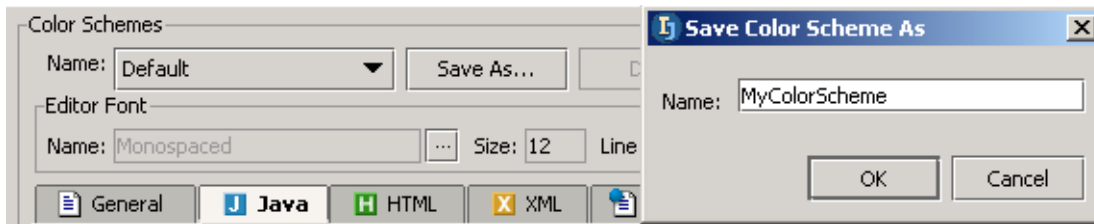


Figure 6.69. Custom color scheme name (789)

6.85. Click **OK**. The color scheme is created.

7.4.7.2. Select

6.86. Select from “Name” drop-down list **MyColorScheme**.

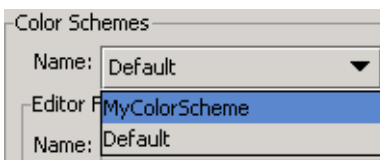


Figure 6.70. Selecting a color scheme (790)

7.4.7.3. Modify

6.87. In tab “General”: Click on **line 1**. Note that the “Default text” is selected.

6.88. Check the checkbox **Background**.

6.89. For the background color select grey.

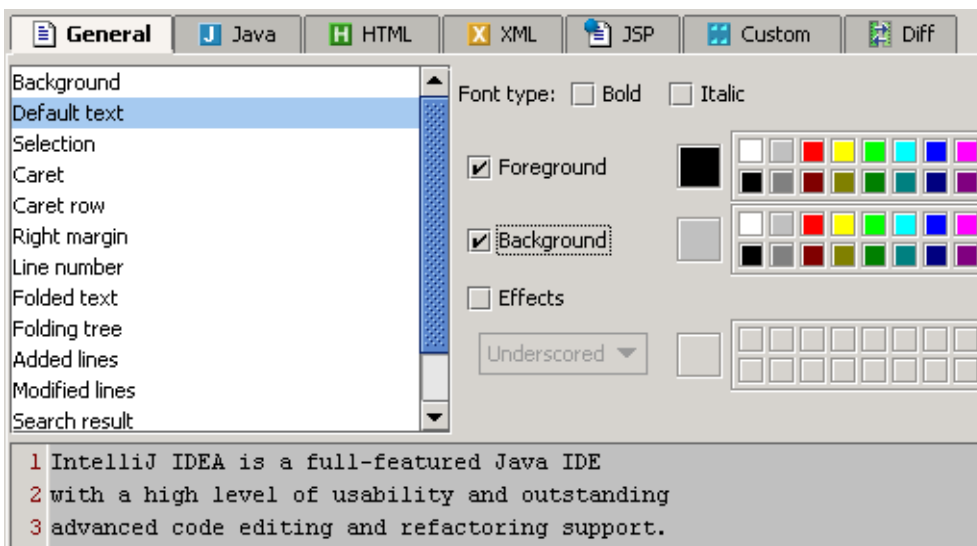
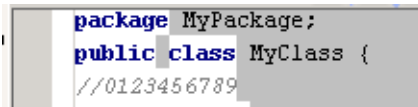


Figure 6.71. Selecting a background color (791)

7.4.7.4. Apply

6.90. Click **Apply**. The text background is changed.



```
package MyPackage;  
public class MyClass {  
    //0123456789
```

Figure 6.72. New background color ([792](#))

7.5. Code style X

IDEA provides the following code style functionality to make it easier to recognize code:

- 7.5.1. General X (page 145)
- 7.5.2. Indent and Braces X (page 146)
- 7.5.3. Blank lines X (page 147)
- 7.5.4. Spaces X (page 148)
- 7.5.5. Imports X (page 149)
- 7.5.6. EJB Names X (page 150)
- 7.5.7. Code style schemes (page 151)
- 7.5.8. Autoindent (page 153)
- 7.5.9. Reformat (page 154)

7.5.1. General X

Tab **General** shows general code styles.

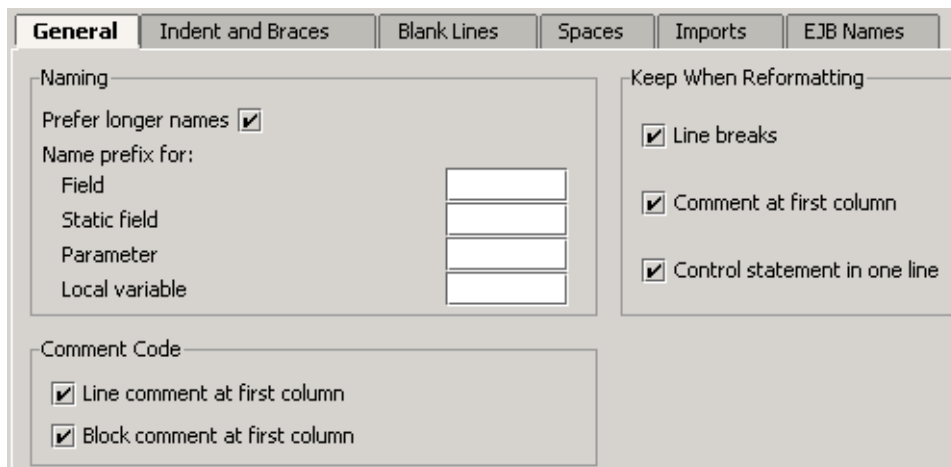


Figure 6.73. General code style [\(431\)](#)

7.5.2. Indent and Braces X

Tab **Indents and braces** shows indents and braces for different types of code and text.

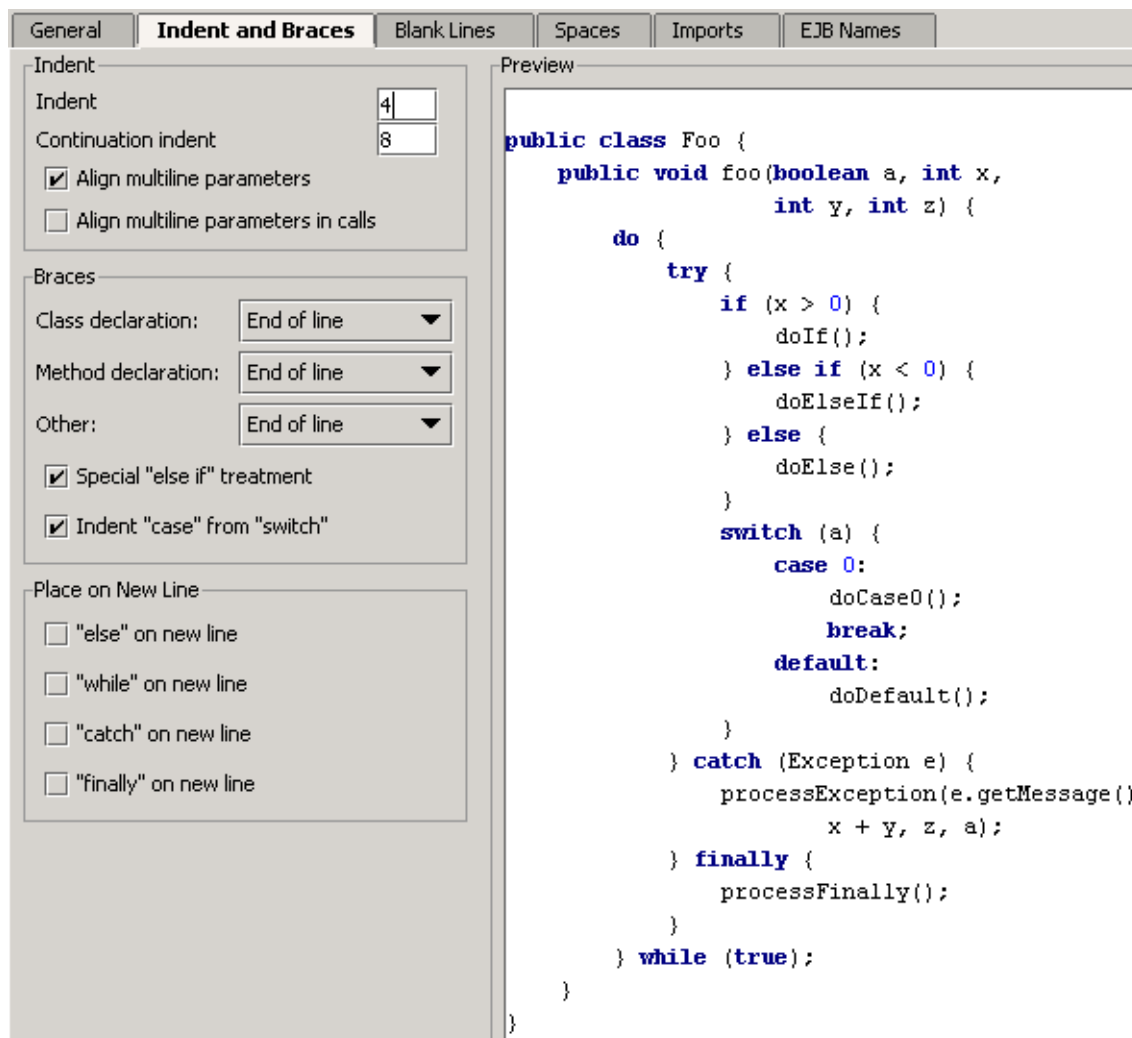


Figure 6.74. Code style / indents and braces (430)

7.5.3. Blank lines X

Tab **Blank lines** shows how blank lines are added.

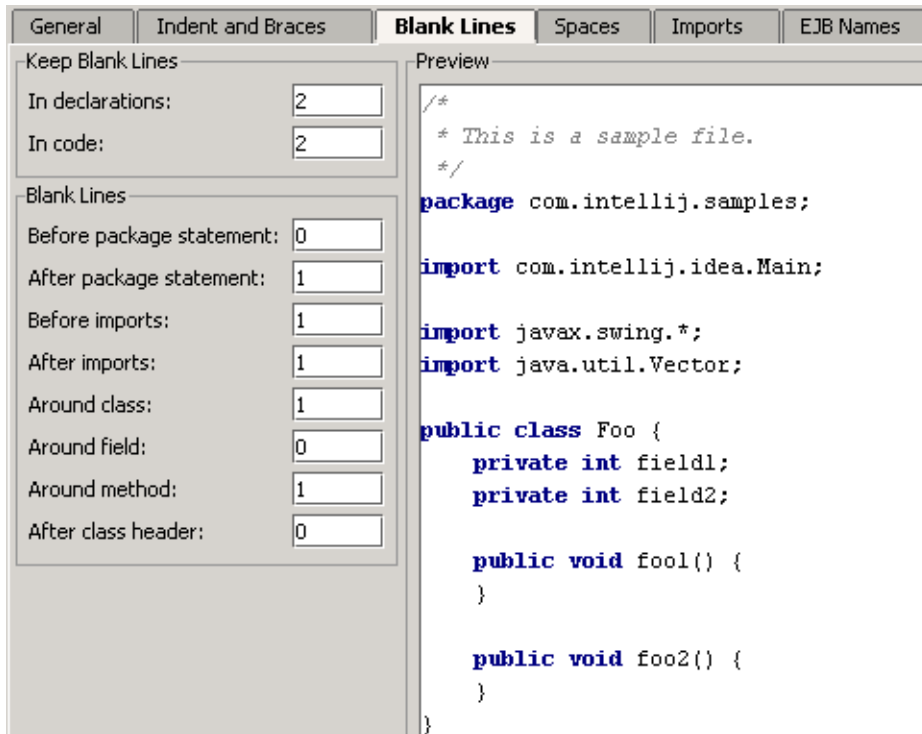


Figure 6.75. Code style / blank lines (429)

7.5.4. Spaces X

Tab Spaces shows how spaces are added.

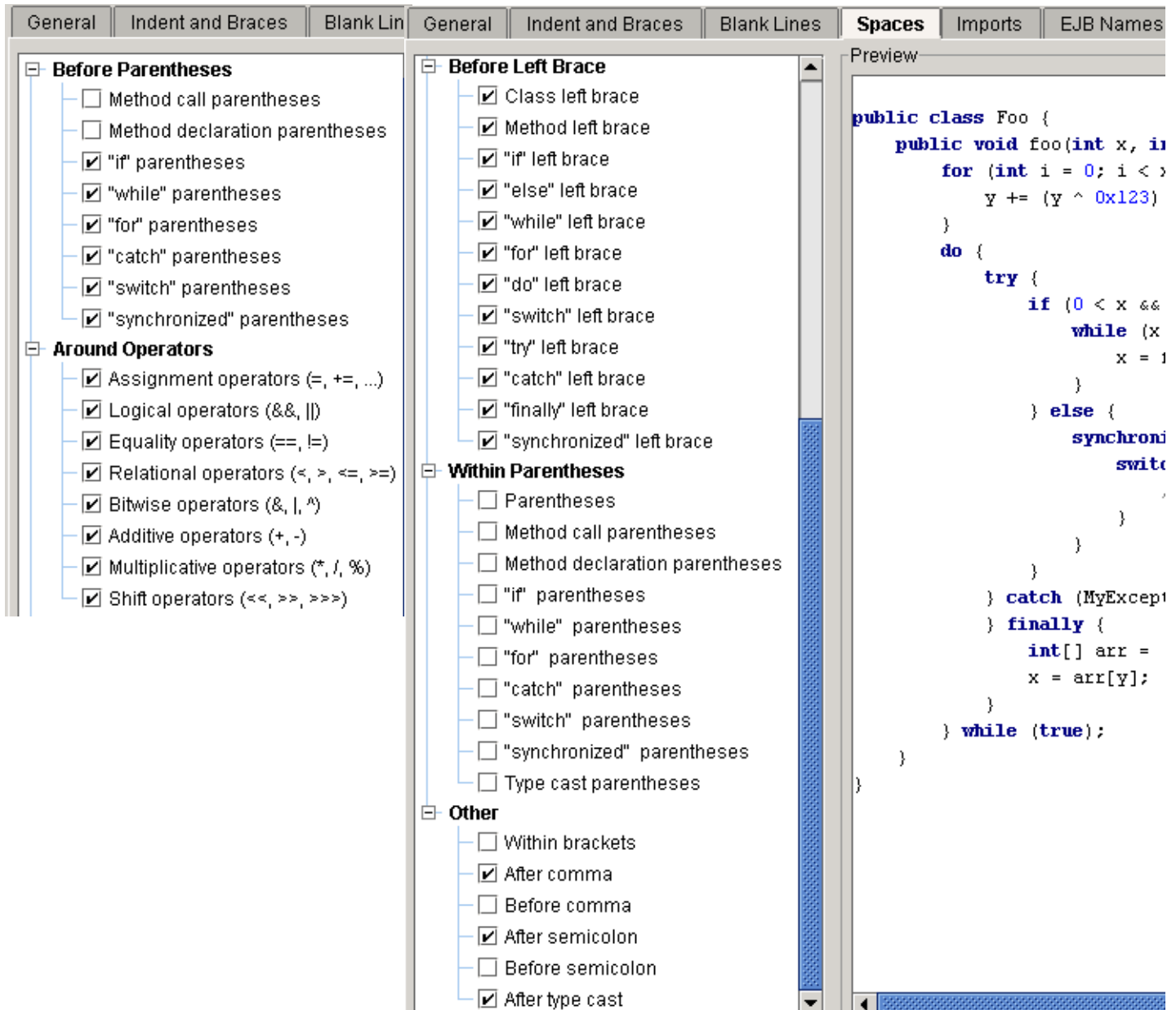


Figure 6.76. Code style / spaces (428.427)

7.5.5. Imports X

Tab **Imports** shows how imports are configured.

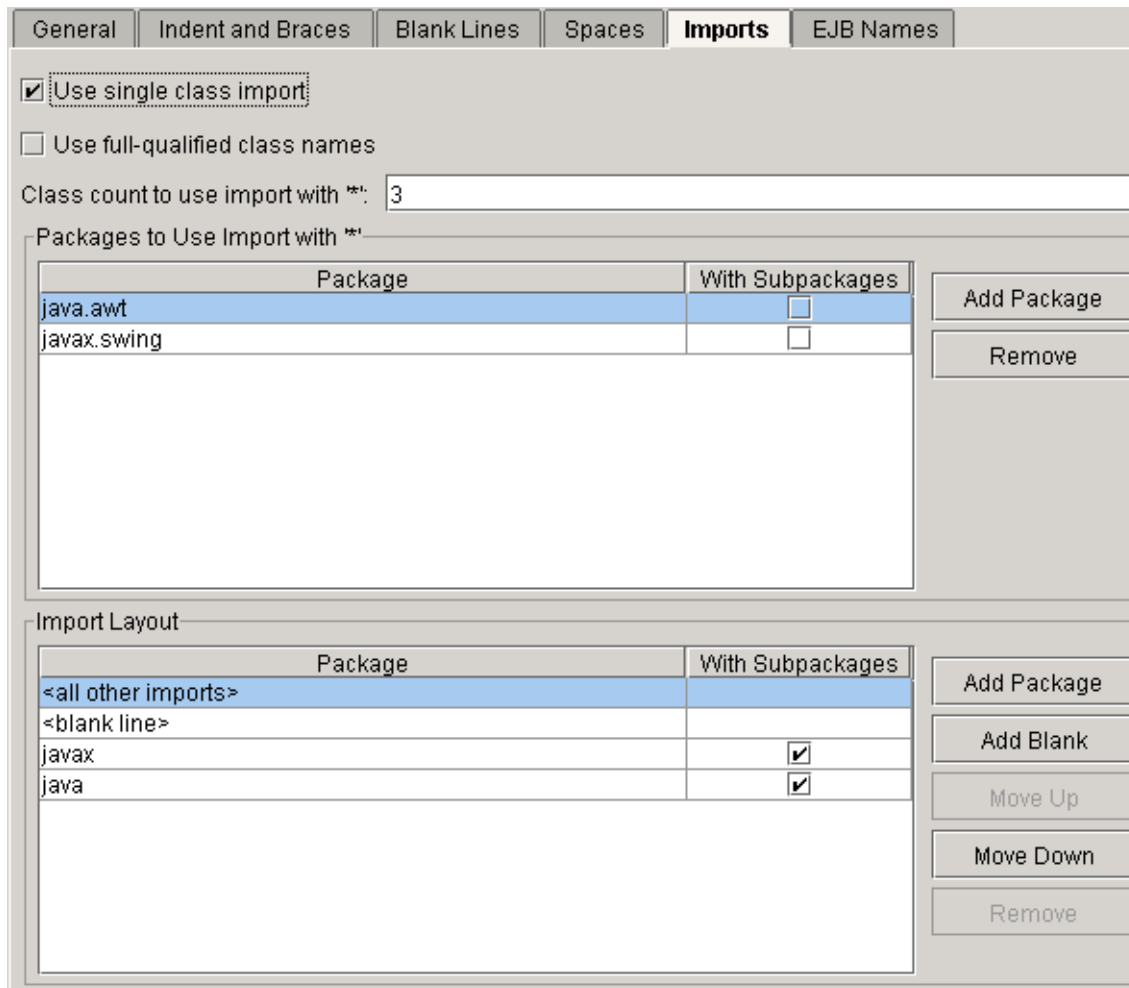


Figure 6.77. Code style / imports [\(426\)](#)

7.5.6. EJB Names X

[20021014TTT move this somewhere else?? are EJB names really code style relevant??.](#)

Tab **EJB Names** shows how EJB's are named by default.

General	Indent and Braces	Blank Lines	Spaces	Imports	EJB Names
Entity Bean					
EJB Class Prefix:	<input type="text"/>	Suffix:	<input type="text" value="Bean"/>		
Home Interface Prefix:	<input type="text"/>	Suffix:	<input type="text" value="Home"/>		
Remote Interface Prefix:	<input type="text"/>	Suffix:	<input type="text"/>		
Local Home Interface Prefix:	<input type="text" value="Local"/>	Suffix:	<input type="text" value="Home"/>		
Local Interface Prefix:	<input type="text" value="Local"/>	Suffix:	<input type="text"/>		
<ejb-name> Prefix:	<input type="text"/>	Suffix:	<input type="text" value="EJB"/>		
Default PK Class:	<input type="text" value="java.lang.String"/>				
Session Bean					
EJB Class Prefix:	<input type="text"/>	Suffix:	<input type="text" value="Bean"/>		
Home Interface Prefix:	<input type="text"/>	Suffix:	<input type="text" value="Home"/>		
Remote Interface Prefix:	<input type="text"/>	Suffix:	<input type="text"/>		
Local Home Interface Prefix:	<input type="text" value="Local"/>	Suffix:	<input type="text" value="Home"/>		
Local Interface Prefix:	<input type="text" value="Local"/>	Suffix:	<input type="text"/>		
<ejb-name> Prefix:	<input type="text"/>	Suffix:	<input type="text" value="EJB"/>		
Message Driven Bean					
EJB Class Prefix:	<input type="text"/>	Suffix:	<input type="text" value="Bean"/>		
<ejb-name> Prefix:	<input type="text"/>	Suffix:	<input type="text" value="EJB"/>		

Figure 6.78. Code style / EJB names [\(793\)](#)

7.5.7. Code style schemes

To use a custom code style scheme, do the following:

- 7.5.7.1. Create (page 151)
- 7.5.7.2. Select (page 151)
- 7.5.7.3. Modify (page 151)
- 7.5.7.4. Apply (page 151)

7.5.7.1. Create

6.91. Click **Save as...**. The dialog “Save code style scheme as” appears.

6.92. For “Name” enter **MyCodeStyleScheme**.

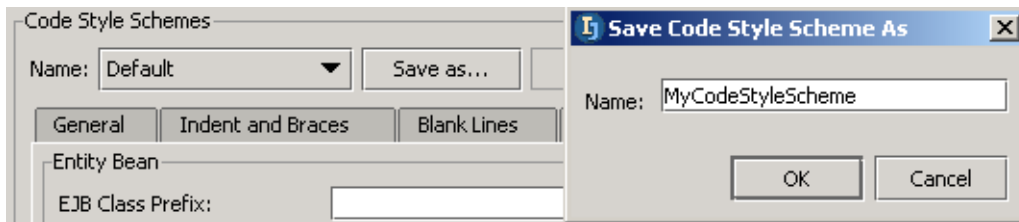


Figure 6.79. Custom code style scheme name (794)

6.93. Click **OK**. The code style scheme is created.

7.5.7.2. Select

6.94. Select from “Name” drop-down list **MyCodeStyleScheme**.

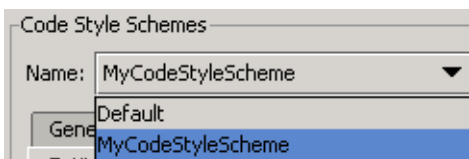


Figure 6.80. Selecting a code style scheme (795)

7.5.7.3. Modify

6.95. In tab “Indent and braces”: In section “Braces”: For “Class declaration” select **Next line**. Note the different style for indenting class declarations.

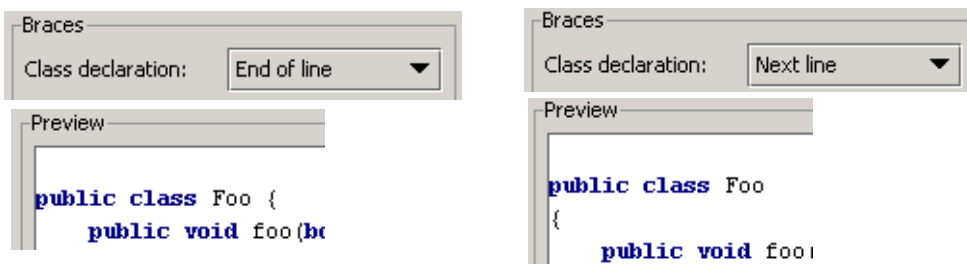


Figure 6.81. Class declaration indents (796.797.798.799)

6.96. Click **OK**. The code style is saved.

7.5.7.4. Apply

- 7.5.7.4.1. To existing class (page 151)
- 7.5.7.4.2. To class template (page 152)

7.5.7.4.1. To existing class

6.97. Open **MyClass.java**.

6.98. Select **Tools | Reformat code...**. The dialog “Reformat code” appears.

6.99. Select **File**.

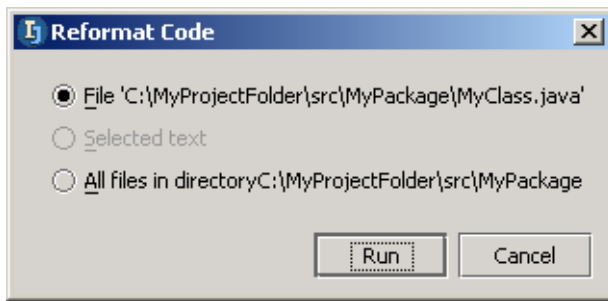


Figure 6.82. Dialog “Reformat code” (801)
6.100. Click **Run**. The file is reformatted.

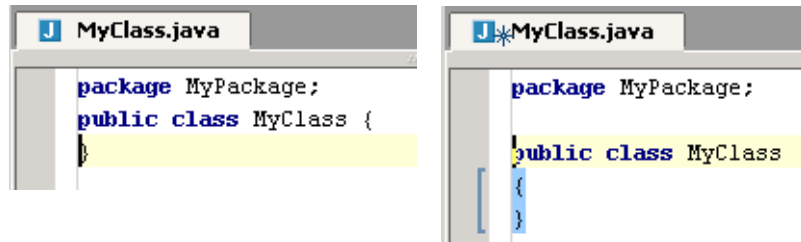


Figure 6.83. Reformatted source file (800.802)

7.5.7.4.2. To class template

6.101. Select **Options | File templates**. The dialog “File templates” appears.

6.102. For tab “Templates” class type “New class” check checkbox **Reformat according to style**.

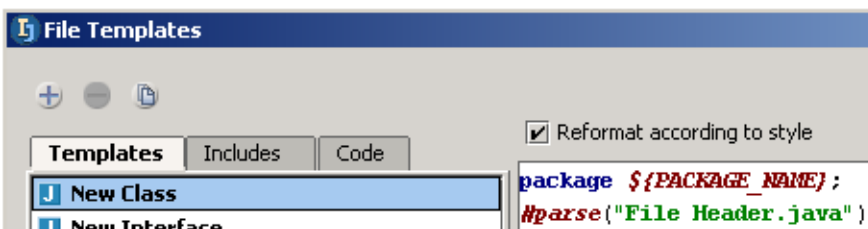


Figure 6.84. Reformat according to style checkbox (803)

6.103. Click **OK**.

6.104. Create a new class. Note the code style.

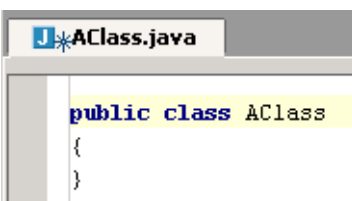


Figure 6.85. New class with code style (804)

7.5.8. Autoindent

6.105. Create class **Class1**:

```
public class Class1 {  
    String s1 = "string1";  
    String s2 = "string2";  
    public static void main(String[] args) {  
        System.out.println("Hello");  
    }  
}
```

6.106. Select all the text (CTRL-A).

6.107. Select **Code | Autoindent lines**. The lines of text are auto-indented.

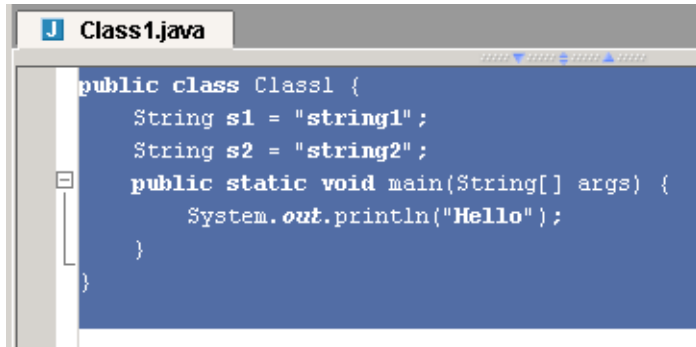


Figure 6.86. Autoindented code ([346](#))

7.5.9. Reformat

IDEA can reformat

- 7.5.9.1. File (page 154)
- 7.5.9.2. Selected text (page 155)
- 7.5.9.3. Files in directory (page 155)

7.5.9.1. File

6.108. Create class **Class1**:

```
public class Class1 {String s1 = "string1"; String s2 = "string2";  
    public static void main(String[] args) {  
        System.out.println("Hello");}}
```

6.109. Select **Tools | Reformat code...** The dialog “Reformat code” appears.

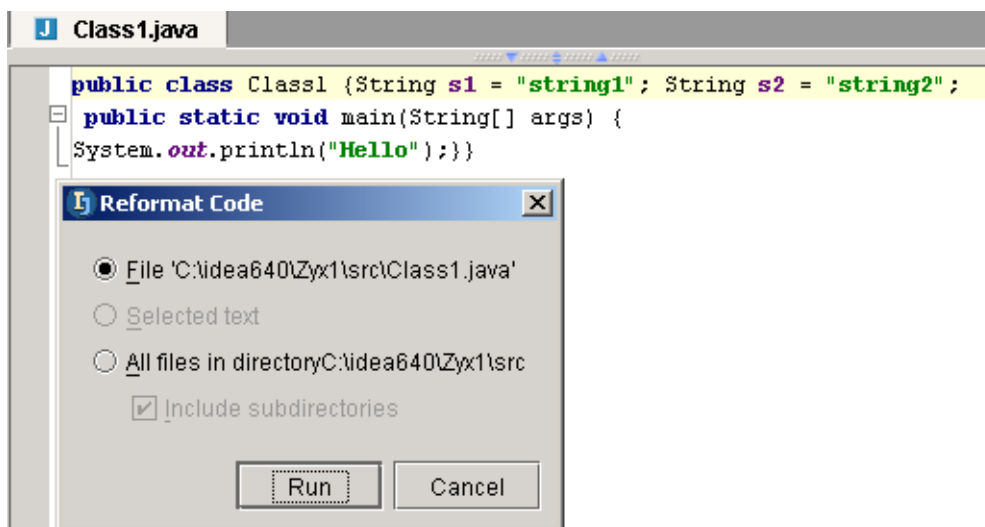


Figure 6.87. Dialog “Reformat code” (345)

6.110. Select **Run**.

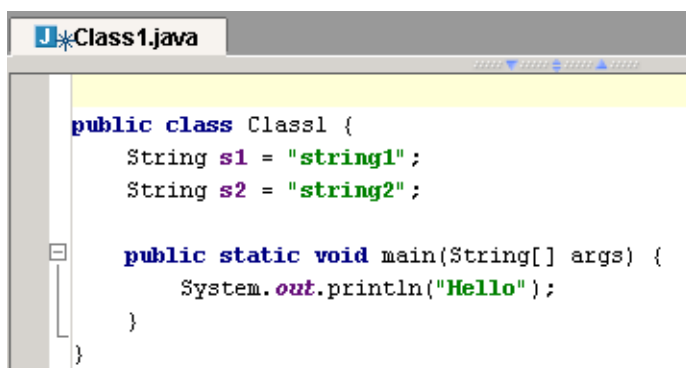


Figure 6.88. Reformatted code (344)

7.5.9.2. Selected text

6.111. Undo the reformatting for **Class1**:

6.112. Select the lines that are in bold below.

```
public class Class1 {String s1 = "string1"; String s2 = "string2";  
    public static void main(String[] args) {  
    System.out.println("Hello");}}
```

6.113. Select **Tools | Reformat code...**. The dialog “Reformat code” appears.

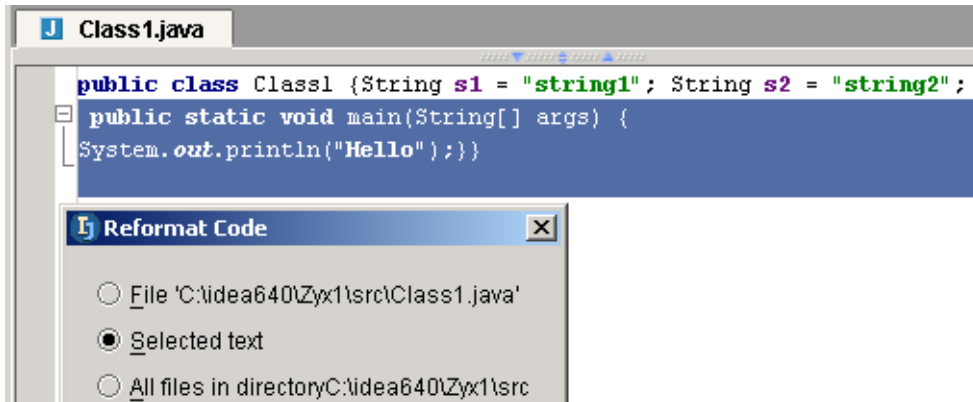


Figure 6.89. Dialog “Reformat code” (selected text) (343)

6.114. Select **Run**. Note that only the selected text is reformatted.

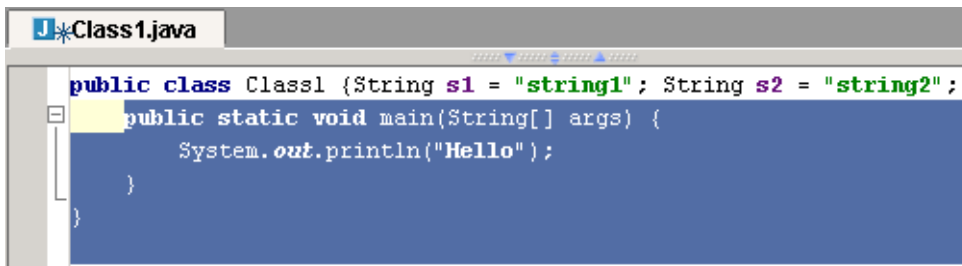


Figure 6.90. Reformatted code (342)

7.5.9.3. Files in directory

6.115. Select a directory.

6.116. Select **Tools | Reformat code...**. The dialog “Reformat code” appears.

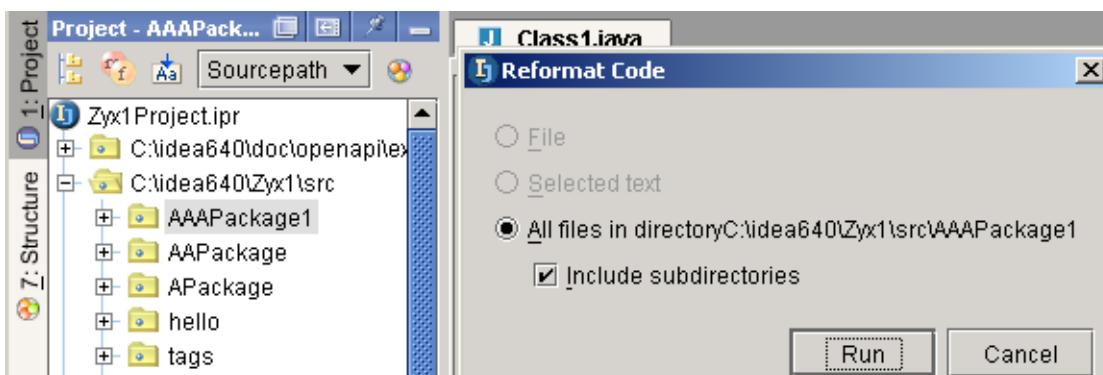


Figure 6.91. Dialog “Reformat code” (selected directory) (341)

If you click **Run**, then the files in the chosen directory and all subdirectories will be reformatted.

7.6. Error indication X

The IDEA editor checks for the following types of errors while editing:

- 7.6.1. **Deprecated symbol** (page 156)
- 7.6.2. **Reparse delay** (page 156)
- 7.6.3. **Unused import** (page 157)
- 7.6.4. **Unused symbol** (page 157)
- ~~7.6.5. Unused throws declaration XXX (page 157)~~
- 7.6.6. **Redundant type cast** (page 157)
- ~~7.6.7. Silly assignment XXX (page 158)~~
- 7.6.8. **Wrong package statement** (page 158)
- ~~7.6.9. Javadocs errors XXX (page 158)~~
- 7.6.10. **Unknown Javadoc tags** (page 158)
- ~~7.6.11. EJB errors XXX (page 159)~~
- ~~7.6.12. EJB warnings XXX (page 159)~~

7.6.1. Deprecated symbol

6.117. Change **MyClass** as shown:

```
package MyPackage;  
public class MyClass {  
    void aMethod() {  
        System.out.println(java.awt.Frame.CROSSHAIR_CURSOR);  
    }  
};
```

An deprecation warning is generated.

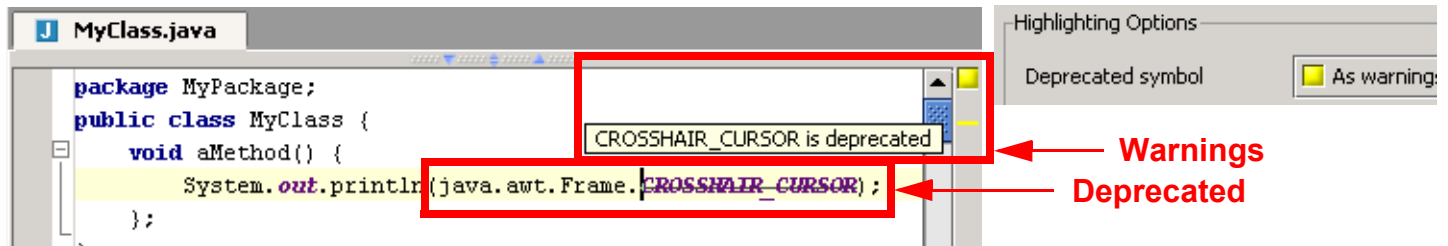


Figure 6.92. Deprecation (720,722)

7.6.2. Reparse delay

6.118. In IDE Settings - Errors: Set **Autoreparse delay** to **5000** (5 seconds).

6.119. In an editor: Make an error (in the example below, “package” was changed to “pacxxxxkage”).

6.120. Move the cursor to a different line. Note that the error is marked (ie, the file is reparsed) after 5 seconds.

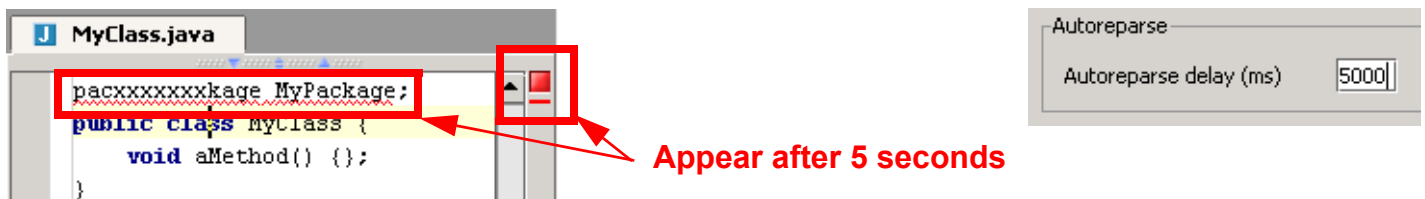


Figure 6.93. Reparse delay (714,716)

6.121. In IDE Settings - Errors: Set **Autoreparse delay** to **300**.

6.122. Click **Apply**.

7.6.3. Unused import

6.123. Change **MyClass** as shown:

```
import java.beans.*;  
package MyPackage;  
public class MyClass {  
    void aMethod() {};  
}
```

An “Unused import statement” warning is generated.

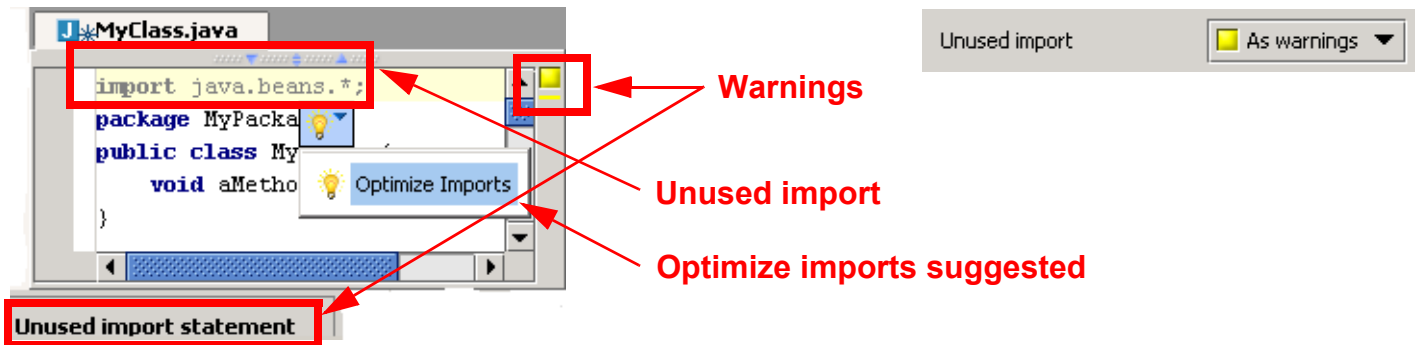


Figure 6.94. Unused import (715,717)

7.6.4. Unused symbol

6.124. Change **MyClass** as shown:

```
package MyPackage;  
public class MyClass {  
    void aMethod() {  
        int a;  
    };  
}
```

An “Unused symbol” warning is generated.

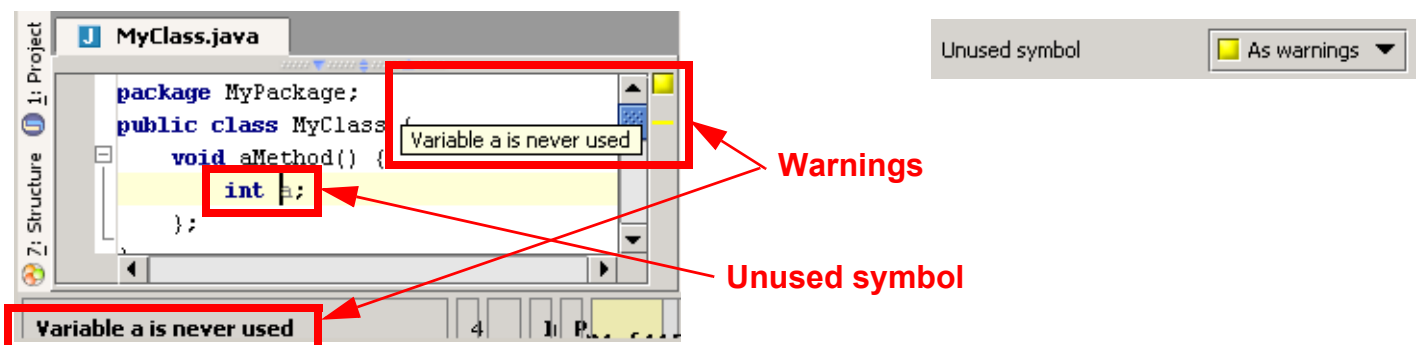


Figure 6.95. Unused symbol (718,719)

~~7.6.5. Unused throws declaration XXX~~

7.6.6. Redundant type cast

6.125. Change **MyClass** as shown:

```
package MyPackage;  
public class MyClass {  
    Object a = new Integer (1);  
}
```

```
Object b = (Object)a;  
}
```

An “Redundant type cast” warning is generated.

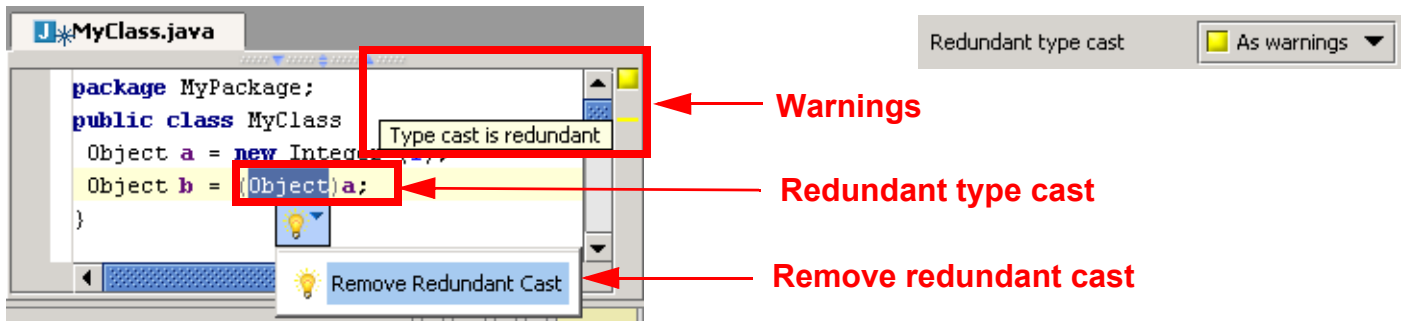


Figure 6.96. Redundant type cast (721,723)

~~7.6.7. Silly assignment XXX~~

7.6.8. Wrong package statement

6.126. Change **MyClass** as shown:

```
package MyPackage2;  
public class MyClass {  
}
```

An wrong package warning is generated.

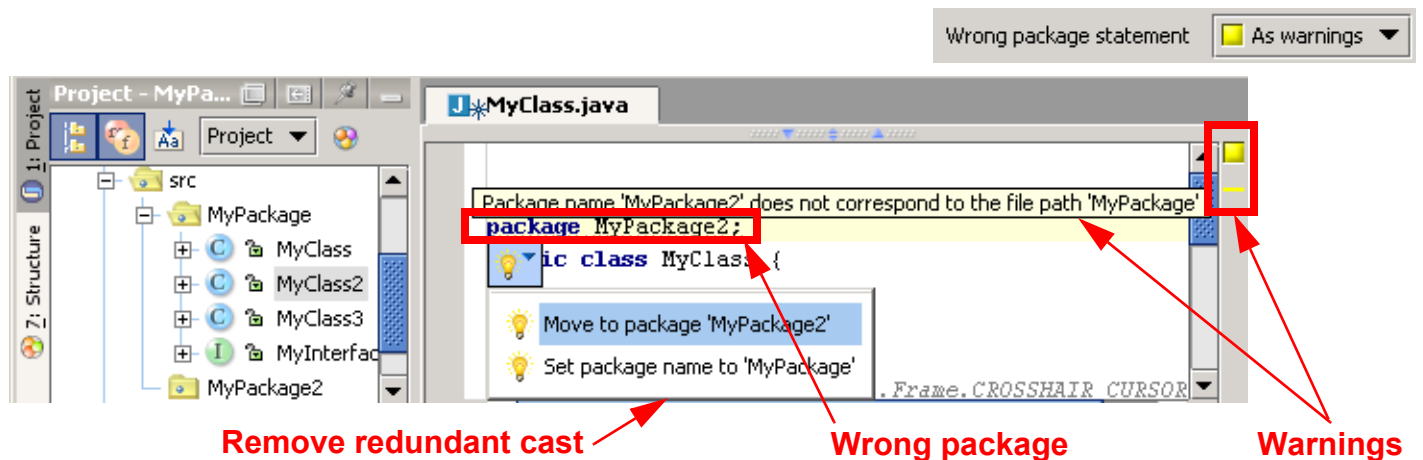


Figure 6.97. Wrong package statement (724,725)

~~7.6.9. Javadocs errors XXX~~

7.6.10. Unknown Javadoc tags

6.127. Change **MyClass** as shown:

```
package MyPackage;  
public class MyClass {  
/**  
 * @param p1
```

```
* @param p2  
* @return  
*/  
public int foo(int p1, int p2)  
{ return p1 + p2; }  
}
```

An wrong javadoc tag is generated.

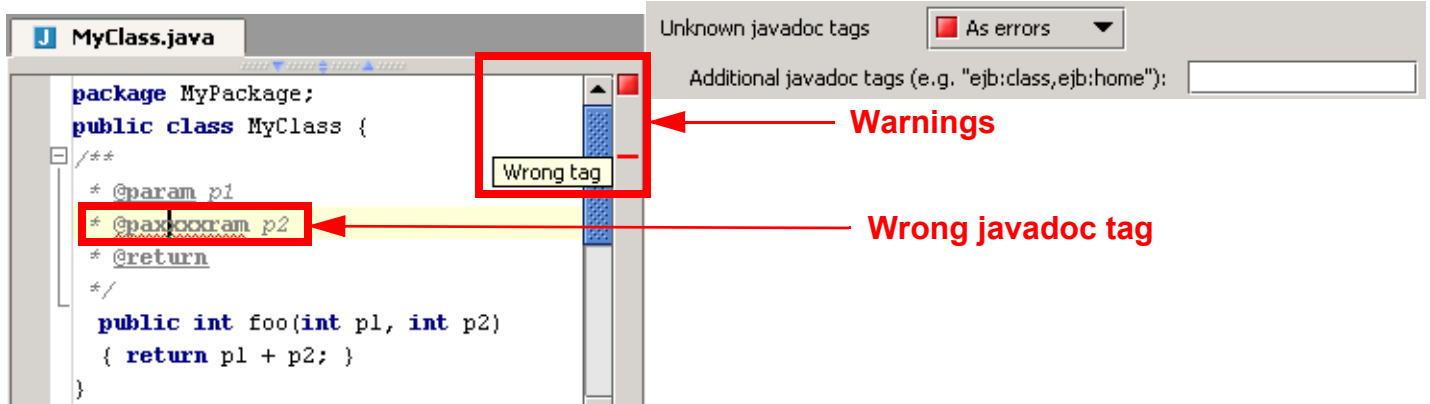


Figure 6.98. Wrong javadoc tag (726.727)

~~7.6.11. EJB errors XXX~~



Figure 6.99. EJB error (728)

~~7.6.12. EJB warnings XXX~~



Figure 6.100. EJB error (729)

7.7. Todo

The IDEA todo functionality makes it easy to make and find todo notes in a file or project.

IDEA todo includes:

- 7.7.1. Basics (page 160)
- 7.7.2. Patterns (page 161)
- 7.7.3. Filters (page 162)

7.7.1. Basics

6.128. In Class1 add the line

```
//@todo finish class1
```

6.129. In Class2 add the lines

```
//@todo finish class2  
//@todo get ready for class3
```

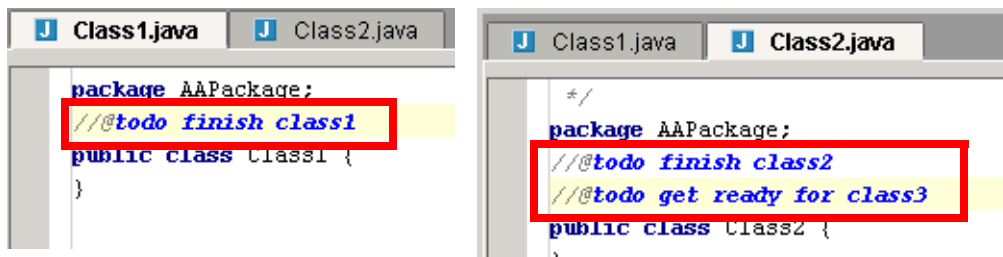


Figure 6.101. Added todo text (702,703)

6.130. Open the **TODO** tool.

6.131. Select the tab **Project**. The todos for the entire project are shown.

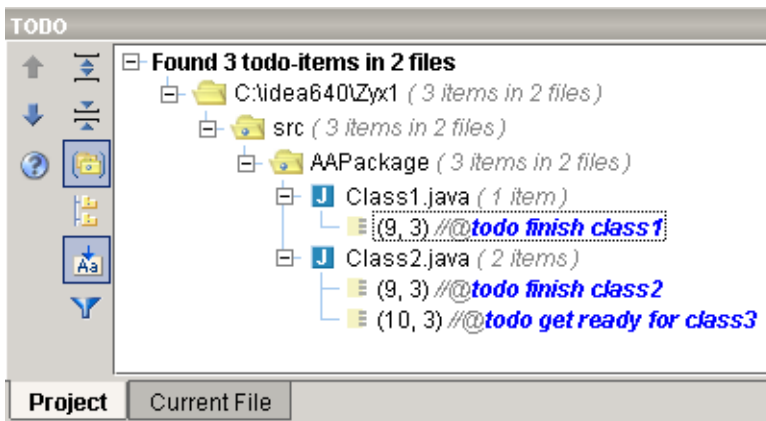


Figure 6.102. TODO items for the project (704)

6.132. Select tab **Current file**. The todos for the current file are shown.

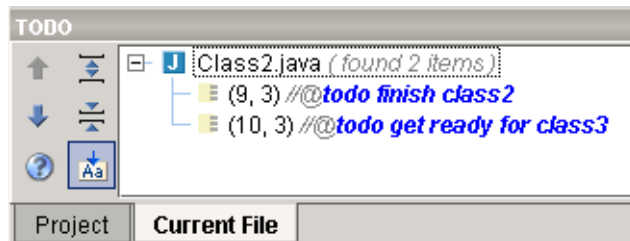


Figure 6.103. Todo items for a file (705)

6.133. Select a different file. The todo items for that file are shown in the TODO tool.

You can also open a file by double-clicking on a todo item.

7.7.2. Patterns

You can create custom patterns that are interpreted by IDEA as todo items.

- Create pattern
- Add pattern to file

7.7.2.1. Create pattern

6.134. Select **Options | IDEA settings**. The “IDE Options” dialog appears.

6.135. Select **TODO**.

6.136. For Patterns: Select **Add**. The dialog “Add Pattern” appears.

6.137. For Pattern: Enter `\btodoNOW\b.*`.

6.138. Select the red icon.

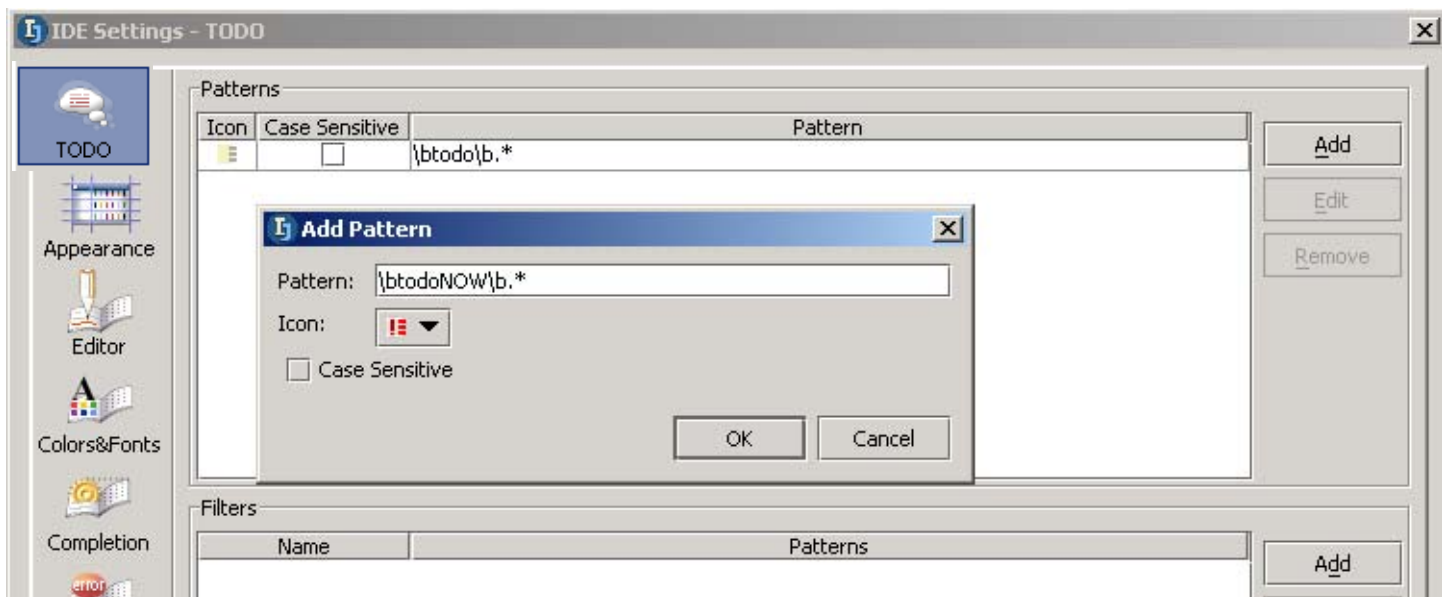


Figure 6.104. New TODO pattern (706)

6.139. Click **OK**. The new pattern appears in the list.

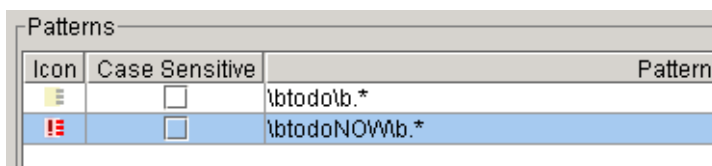


Figure 6.105. New TODO pattern (707)

6.140. Click **OK** to close.

7.7.2.2. Add pattern to file

6.141. In Class1 add the line

```
//@todonow urgent stuff
```

6.142. View the todo in teh TODO dialog:



Figure 6.106. New TODO in the TODO tool (708)

7.7.3. Filters

You can create filters that only show certain todo patterns.

- **Create filter**
- **Display filtered TODOs**

7.7.3.1. Create filter

6.143. In dialog “IDEA options” tab “TODO”: For **Filters** click **Add**. The dialog “Add filter” appears.

6.144. For “Name” enter **NOW**.

6.145. Check `\btodoNOW\b.*`.

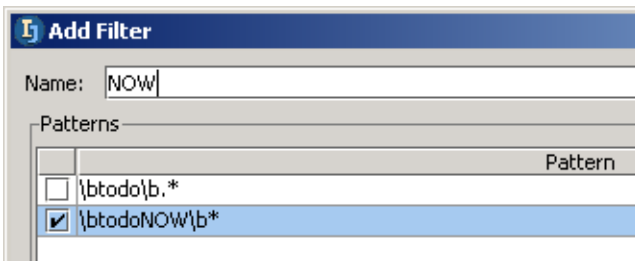


Figure 6.107. New TODO filter (709)

6.146. Click **OK**. The new filter appears in the list.

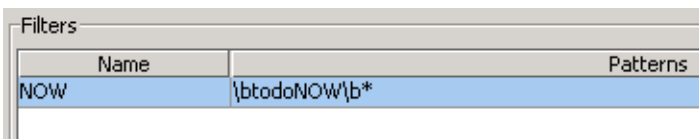


Figure 6.108. New TODO filter (710)

6.147. Click **OK** to close.

7.7.3.2. Display filtered TODOs

6.148. In the TODO dialog: Click on filter icon (). A popup for selecting the filter appears.

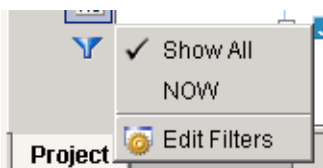


Figure 6.109. Popup for selecting filter (711)

6.149. Select **NOW**. Only the NOW TODOs are shown.

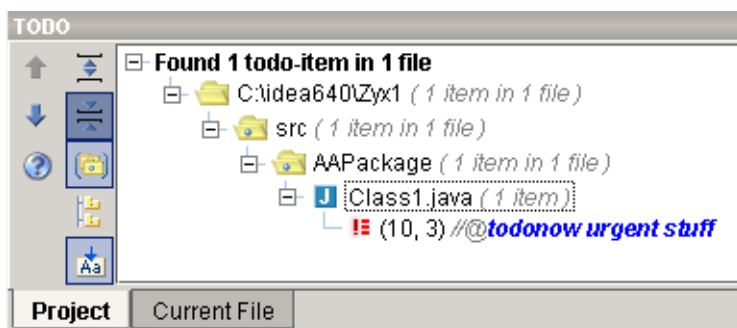


Figure 6.110. Only filtered TODOs shown (713)

8. Code Automation

[contacts: many, maxim, valya](#)

IDEA offers the following code automation functions:

- 8.1. Code completion (suggestion) (page 164)
- ~~8.2. Code completion OLD XXX (page 179)~~
- 8.3. Code templates (page 184)
- 8.4. Code generation (page 192)
- 8.5. Import optimization (page 193)
- 8.6. Method override (page 194)
- 8.7. Interface implementation (page 196)
- 8.8. Method delegation (page 197)
- 8.9. Comment (page 199)

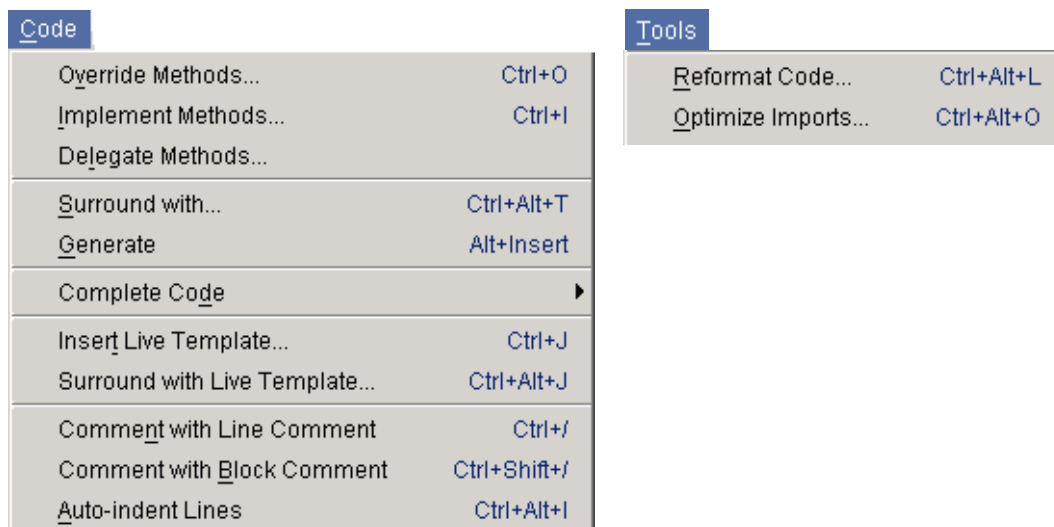
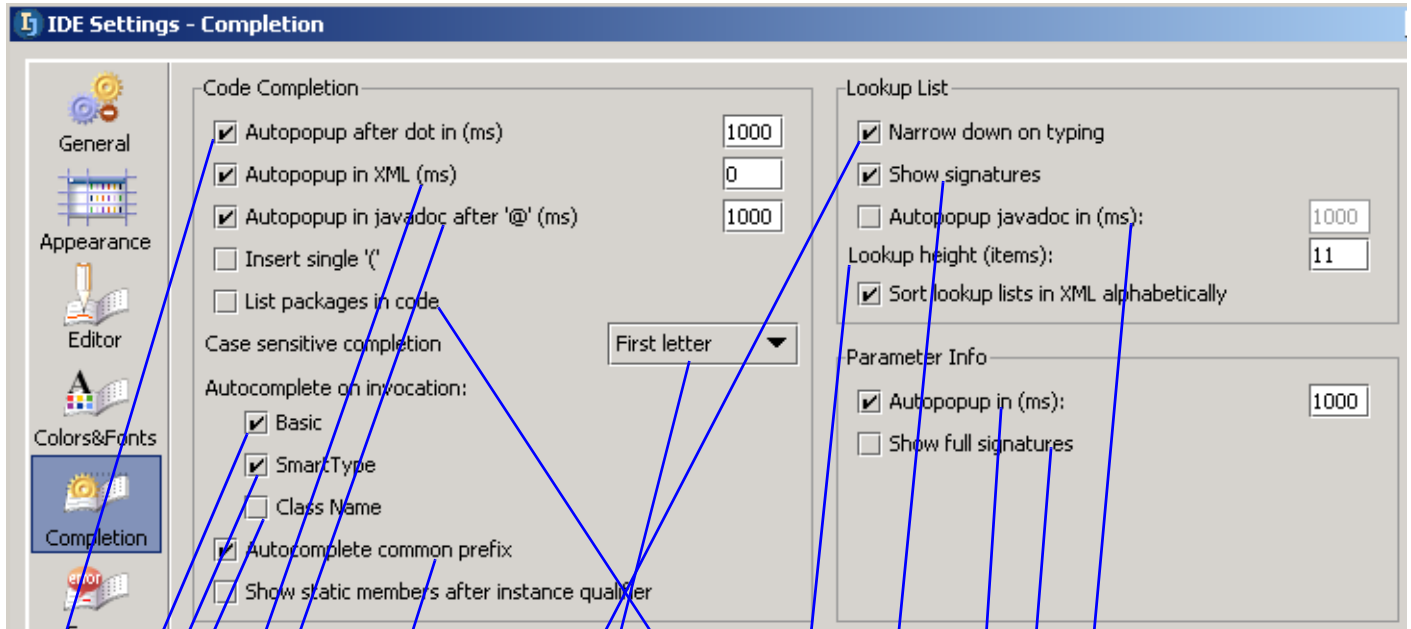


Figure 7.1. Code automation menu items ([417,416,415](#))

8.1. Code completion (suggestion)

20021016TTT recommended dialog changes.



Code suggestion

- After dot
- After partial text entry**
 - Any fvmc(p) in import CTRL-SPACE
 - Recommended (smarttype) fvmc in import CTRL-SHIFT-SPACE
 - Any c in any package CTRL-ALT-SPACE
- Options**
 - Case sensitivity None
First letter
All
 - Auto-narrow**
 - List as text entered
 - Entered text
- in XML
- Sort
- After javadoc @

Options

- Lookup list size
- Insert single (
- Include packages
- Show signature
 - In popup
 - In ()
 - Manual CTRL-P
 - Autopopup ms
 - Full
- Show javadoc

Figure 7.2. current and recommended dialog (812)

IDEA can suggest how to complete partially entered code:

- 8.1.1. Suggest after dot (auto) (page 166)
- 8.1.2. Suggest after partial text (manual) (page 167)
- 8.1.3. Suggest in XML (auto) X (page 174)
- 8.1.4. Suggest after @ (javadoc) (auto) X (page 174)

The above functions have several

• 8.1.5. Options (page 175)

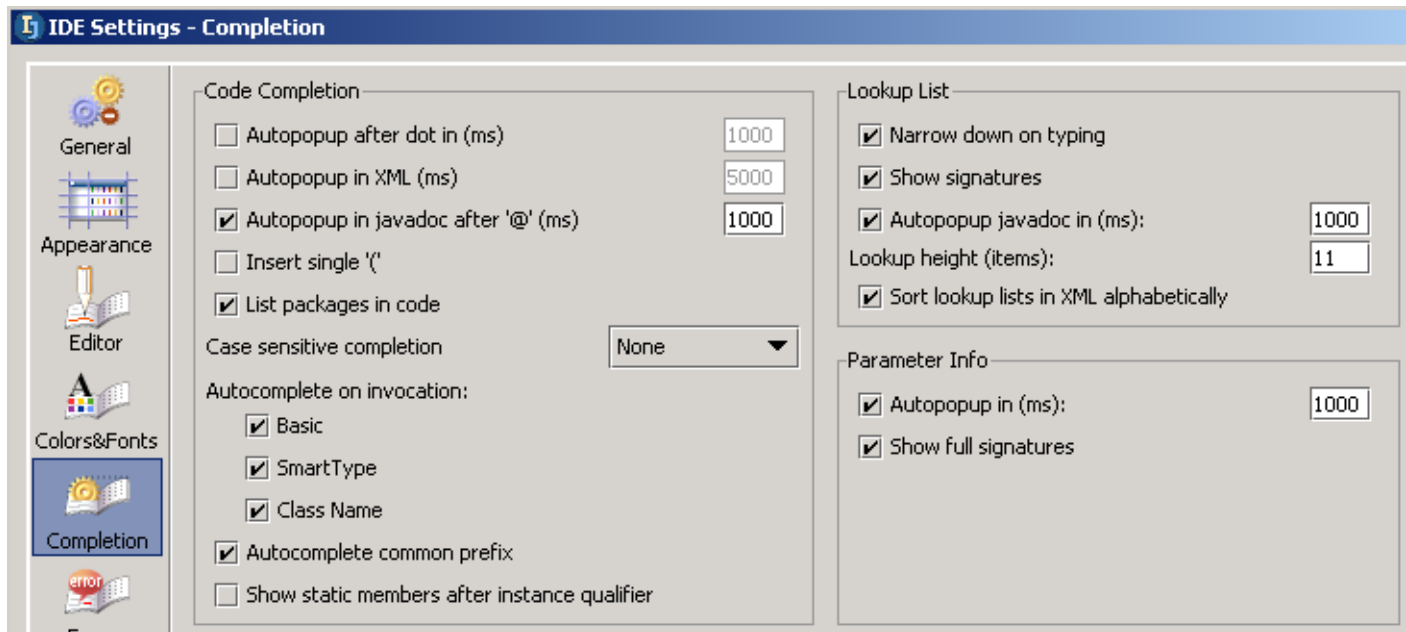


Figure 7.3. IDEA Settings / Completion (858)

8.1.1. Suggest after dot (auto)

Determines if and after what delay an autopopup will appear after a period has been typed at the end of a declaration (assuming that the cursor does not move and nothing else is typed in after the dot).

7.1. Open **IDEA Settings | Completion**.

7.2. Check the checkbox **Autopopup after dot in (ms)**.

7.3. For the value enter **5000** (5 secs).

7.4. Click **OK**.

7.5. In the editor: Enter “**System.out.**”. Note that after 5 seconds an autopopup appears.

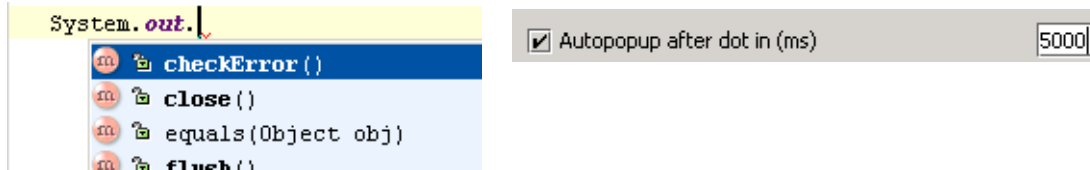


Figure 7.4. Autopopup [\(824.823\)](#)

7.6. Open **IDEA Settings | Completion**.

7.7. Uncheck the checkbox.

7.8. Click **OK**.

7.9. Again in the editor: Enter “**System.out.**”. Note that the popup does not appear.

7.10. Click **CTRL-SPACE**. The popup appears.

7.11. Open **IDEA Settings | Completion**.

7.12. Check the checkbox.

7.13. For the value enter **1000** (1 sec).

7.14. Click **OK**.

8.1.2. Suggest after partial text (manual)

IDEA supports the following variations for code suggestion after part of the code has been entered and a hot key has been pressed:

- [8.1.2.1. fvmc\(p\) in import CTRL-SPACE \(basic\) \(page 167\)](#)
- [8.1.2.2. fvmc recommended in import CTRL-SHIFT-SPACE \(smarctype\) \(page 168\)](#)
- [8.1.2.3. Classes recommended in all packages CTRL-ALT-SPACE \(page 169\)](#)

The above functions have several

- [8.1.2.4. Options \(page 170\)](#)

8.1.2.1. fvmc(p) in import CTRL-SPACE (basic)

This function suggests

- fields
- variables
- methods
- classes
- packages (optional)

that

- match the partial text that has been entered and
- are imported

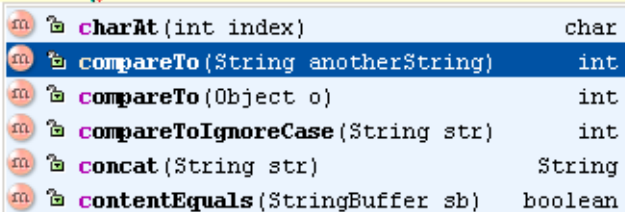
7.15. Enter the following code:

```
String myString = "hello";  
myString.c
```

7.16. Leave the cursor positioned after "myString.c".

7.17. Press **CTRL-SPACE**. A list of suggestions appears.

```
void aMethod(){  
    String myString = "hello";  
    myString.c
```



m	charAt (int index)	char
m	compareTo (String anotherString)	int
m	compareTo (Object o)	int
m	compareToIgnoreCase (String str)	int
m	concat (String str)	String
m	contentEquals (StringBuffer sb)	boolean

Figure 7.5. Matching fvmc(p) in import [\(842\)](#)

7.18. Double-click on a suggestion to complete the code.

8.1.2.2. fvmc recommended in import CTRL-SHIFT-SPACE (smarttype)

This function suggests

- fields
- variables
- methods
- classes

that

- match the partial text that has been entered and
- are imported and
- are most likely required in the given situation

7.19. Enter the following code:

```
String myString = "hello";  
myString.compareTo(
```

7.20. Leave the cursor positioned after "myString.compareTo".

7.21. Press **CTRL-SHIFT-SPACE**. A list of suggestions that are most likely required in this situation appears.

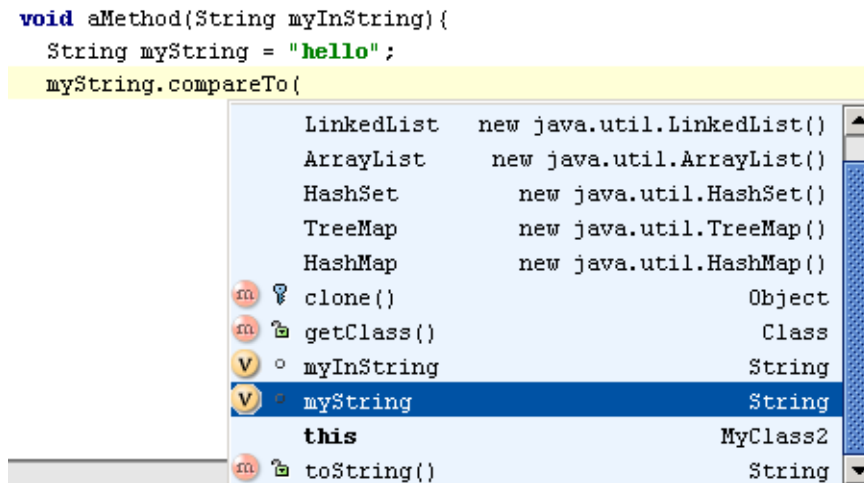


Figure 7.6. Recommended fvmc in import (843)

7.22. Double-click on a suggestion to complete the code.

8.1.2.3. Classes recommended in all packages CTRL-ALT-SPACE

This function suggests

- classes

that

- match the partial text that has been entered

Note that all classes in all packages are listed (not just those in the import).

7.23. Enter the following code (the letter O):

O

7.24. Leave the cursor positioned after “O”.

7.25. Press **CTRL-ALT-SPACE**. A list of all classes in all packages that start with “O” are listed.

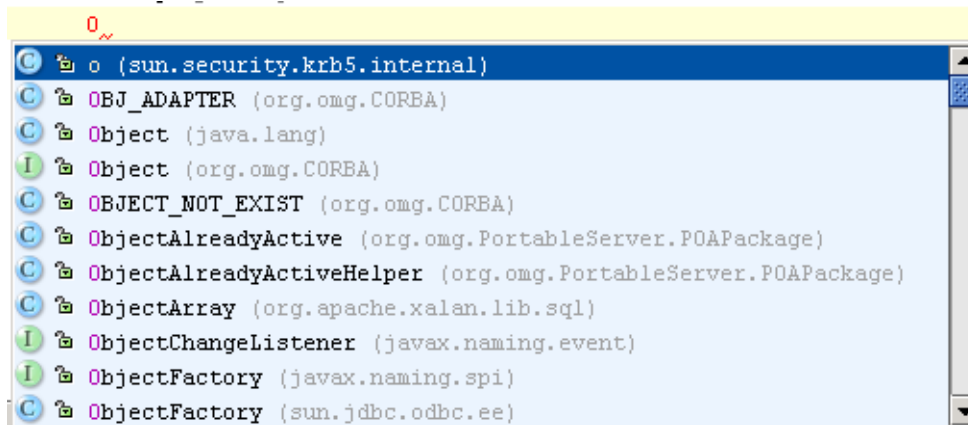


Figure 7.7. All matching classes ([844](#))

7.26. Double-click on a suggestion to complete the code.

8.1.2.4. Options

The following options are available

- 8.1.2.4.1. Case sensitivity (page 170)
- 8.1.2.4.2. Auto-narrow (page 172)

8.1.2.4.1. Case sensitivity

IDEA can limit the list of suggestions to only those possibilities whose

- 8.1.2.4.1.1. None (page 170)
- 8.1.2.4.1.2. First letter (page 170)
- 8.1.2.4.1.3. All (page 170)

letters have the same case as the entered partial text.

8.1.2.4.1.1. None

7.27. Open **IDEA Settings | Completion.**

7.28. For “Case sensitive completion” select **None.**

7.29. Uncheck the 3 checkboxes for “Autocomplete on invocation”.

7.30. Click **OK.**

7.31. Create Class2 with the following content:

```
package MyPackage;  
public static class MyClass2 {  
    static int ABCDEF = 1;  
    static int Abcdef = 1;  
    static void abcdef(){};  
    MyClass2.AB  
}
```

7.32. Place the cursor at the end of “MyClass2.AB”.

7.33. Press **CTRL-SPACE.** A popup appears with all 3 options.



Figure 7.8. Case sensitive completion: None ([835,836](#))

8.1.2.4.1.2. First letter

7.34. Open **IDEA Settings | Completion.**

7.35. For “Case sensitive completion” select **First letter.**

7.36. Click **OK.**

7.37. Place the cursor at the end of “MyClass2.AB”.

7.38. Press **CTRL-SPACE.** A popup appears with 2 options.

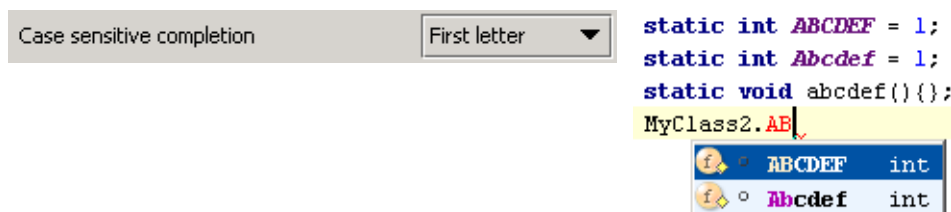


Figure 7.9. Case sensitive completion: First letter ([837,838](#))

8.1.2.4.1.3. All

- 7.39. Open **IDEA Settings | Completion**.
- 7.40. For “Case sensitive completion” select **All**.
- 7.41. Click **OK**.
- 7.42. Place the cursor at the end of “MyClass2.AB”.
- 7.43. Press **CTRL-SPACE**. A popup appears with 1 option.

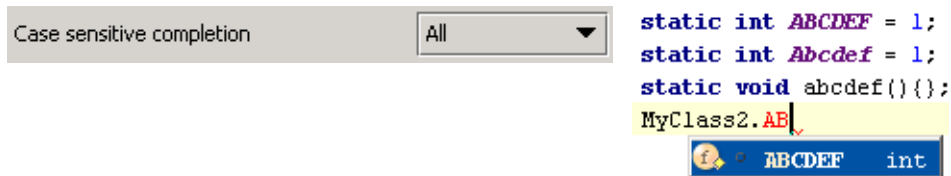


Figure 7.10. Case sensitive completion: All [\(839,840\)](#)

8.1.2.4.2. Auto-narrow

IDEA can automatically narrow the

- 8.1.2.4.2.1. List as text entered (page 172)
- 8.1.2.4.2.2. Entered text (page 172)

8.1.2.4.2.1. List as text entered

7.44. Check the checkbox **Narrow down on typing**.

7.45. Enter the following code:

```
String myString = "hello";  
myString.c
```

7.46. Leave the caret at the end of "myString.c". Note the auto

7.47. Click **CTRL-SPACE**. A list of suggestions appears.

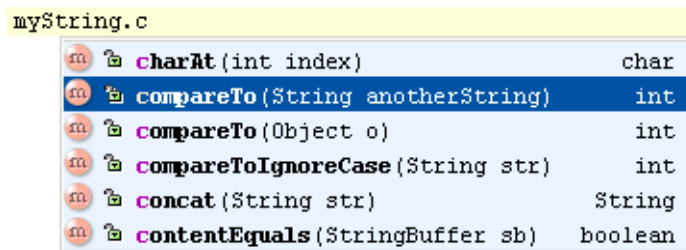


Figure 7.11. Suggestions after myString.c (845)

7.48. Press the "o" key. The list is narrowed.

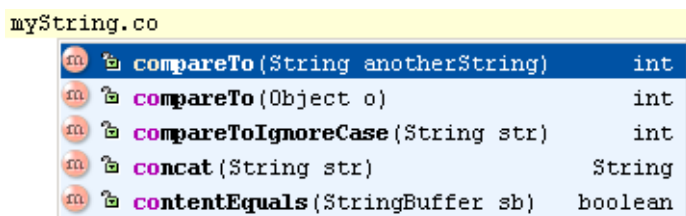


Figure 7.12. Suggestions after myString.co (846)

7.49. Press the "m" key. The list is further narrowed.

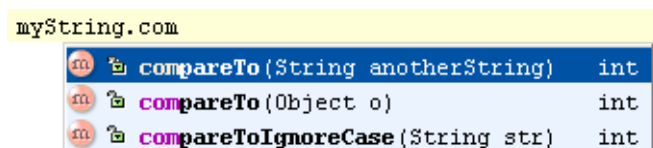


Figure 7.13. Suggestions after myString.com (847)

8.1.2.4.2.2. Entered text

7.50. Check the checkbox **Autocomplete common prefix**.

7.51. Enter the following code:

```
String myString = "hello";  
myString.com
```

7.52. Leave the caret at the end of "myString.com".

7.53. Click **CTRL-SPACE**. Note that the letters "pareTo" are added and the corresponding suggestion list displayed.

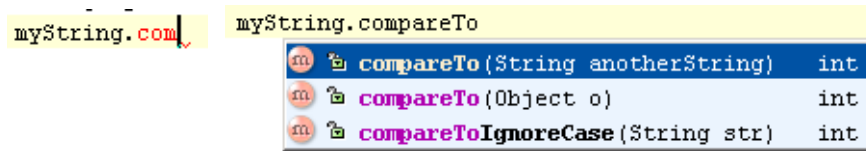


Figure 7.14. Entered text narrowed ([848.849](#))

8.1.3. Suggest in XML (auto)X

Similar to 8.1.1. Suggest after dot (auto) (page 166), but for XML files.

~~8.1.3.1. Sort XXX~~

8.1.4. Suggest after @ (javadoc) (auto)X

Similar to 8.1.1. Suggest after dot (auto) (page 166), but for popups after the javadoc '@'.

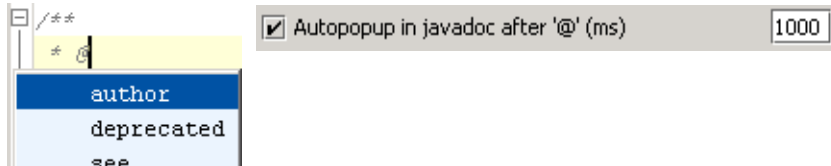


Figure 7.15. Autopopup in javadoc after '@' ([826.825](#))

8.1.5. Options

The following options for code recommendations are available:

- [8.1.5.1. Lookup list height \(page 175\)](#)
- [8.1.5.2. Insert single ‘\(‘ \(page 175\)](#)
- [8.1.5.3. Include packages \(page 176\)](#)
- [8.1.5.4. Show signature \(page 177\)](#)
- [8.1.5.5. Show javadoc \(page 178\)](#)

8.1.5.1. Lookup list height

Determines the number of lines in the suggestion popup.

7.54. Set “Lookup list height” to 2.

7.55. Display a list. Note that the list displays only 2 lines (you can scroll to view the other lines).

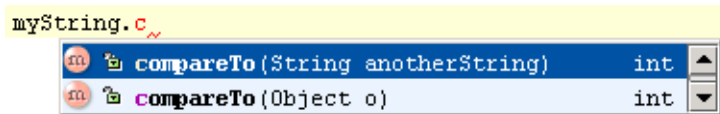


Figure 7.16. Lookup list height = 2 ([856](#))

8.1.5.2. Insert single ‘(‘

If checked: Only a single ‘(‘ is inserted after a suggestion has been selected.

7.56. Check the checkbox **Insert single ‘(‘**.

7.57. In the editor: Add the following:

```
String aString = "Hello";  
aString.getBytes
```

7.58. Place the cursor at the end of “getBytes”.

7.59. Press **CTRL-SPACE**. A single ‘)’ appears.

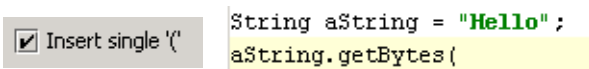


Figure 7.17. Insert single ‘)’ ([827,828](#))

7.60. Delete the single ‘)’.

7.61. Uncheck the checkbox **Insert single ‘)’**.

7.62. Place the cursor at the end of “getBytes”.

7.63. Press **CTRL-SPACE**. ‘)’ appears.

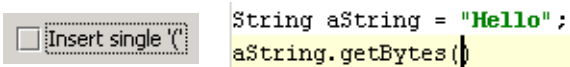


Figure 7.18. Insert ‘)’ ([829,830](#))

8.1.5.3. Include packages

If checked: Available packages are included in the list of suggestions.

7.64. Check the checkbox **List packages in code**.

7.65. In the editor: Add the following:

```
String comString = "Hello";  
c
```

7.66. Place the cursor at the end of “c”.

7.67. Press **CTRL-SPACE**. A popup appears that includes packages.

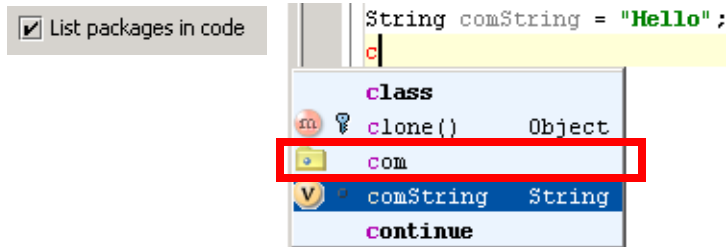


Figure 7.19. List packages in code (832.831)

7.68. Uncheck the checkbox **List packages in code**.

7.69. Place the cursor at the end of “c”.

7.70. Press **CTRL-SPACE**. A popup appears that does not include packages.

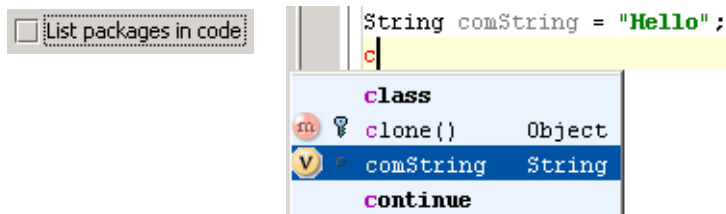


Figure 7.20. No packages in code (833.834)

8.1.5.4. Show signature

The signature can be optionally shown

- 8.1.5.4.1. In popup (page 177)
- 8.1.5.4.2. In ‘()’ (page 177)

8.1.5.4.1. In popup

7.71. Check the checkbox **Show signatures**.

7.72. In the editor: Add the following:

```
String aString = "Hello";
aString.c
```

7.73. Place the cursor at the end of “c”.

7.74. Press **CTRL-SPACE**. A popup appears that includes the signatures.

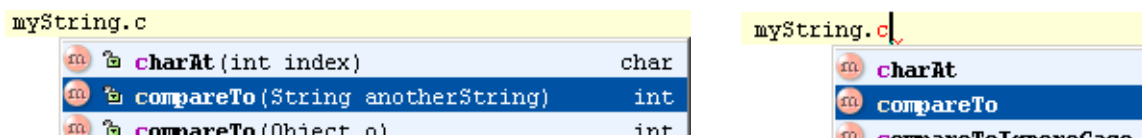


Figure 7.21. With / without signature in popup (850,857)

8.1.5.4.2. In ‘()’

IDEA can show the signature within ‘()’ with the following variations

- 8.1.5.4.2.1. Manual CTRL-P (page 177)
- 8.1.5.4.2.2. Autopopup (ms) (page 177)
- 8.1.5.4.2.3. Full (page 178)

8.1.5.4.2.1. Manual CTRL-P

The signature can be displayed manually with CTRL-P.

7.75. In the editor: Add the following:

```
String aString = "Hello";
aString.compareTo()
```

7.76. Place the cursor at the end of “compareTo”.

7.77. Press **CTRL-P**. A popup appears with a suggestion that includes signatures.

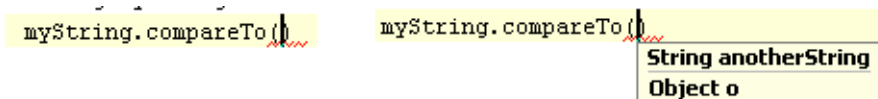


Figure 7.22. CTRL-P signature (851,852)

8.1.5.4.2.2. Autopopup (ms)

7.78. Check the checkbox **Autopopup in ms**.

7.79. In the editor: Add the following:

```
String aString = "Hello";
aString.c
```

7.80. Place the cursor at the end of “c”.

7.81. Click **CTRL-SPACE**.

7.82. Double-click on **compareTo(String anotherString)**. The caret is placed between ‘()’. After 1 sec the popup appears.

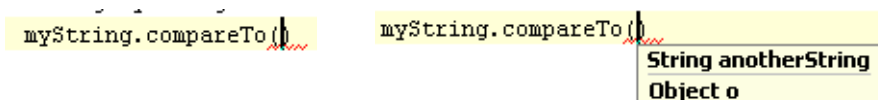


Figure 7.23. Signature autopopup (851,852)

8.1.5.4.2.3. Full

7.83. Check the checkbox **Show full signatures**.

7.84. In the editor: Add the following:

```
String aString = "Hello";  
aString.compareTo()
```

7.85. Place the cursor at the end of “compareTo”.

7.86. Press **CTRL-P**. A popup appears with suggestions that include full signatures.

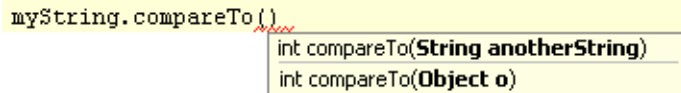


Figure 7.24. Full signatures (853)

8.1.5.5. Show javadoc

7.87. Check the checkbox **Autopopup javadoc in (ms)**.

7.88. In the editor: Add the following:

```
String aString = "Hello";  
aString.c
```

7.89. Place the cursor at the end of “aString.c”.

7.90. Press **CTRL-SPACE**. A popup appears with suggestions.

7.91. After 1 sec the Java doc for the highlighted suggestion is displayed.

7.92. Click the **DOWN** arrow key. The following occurs:

- Java doc disappears
- The next suggestion is selected
- After 1 sec the javadoc for the selected suggestion appears.

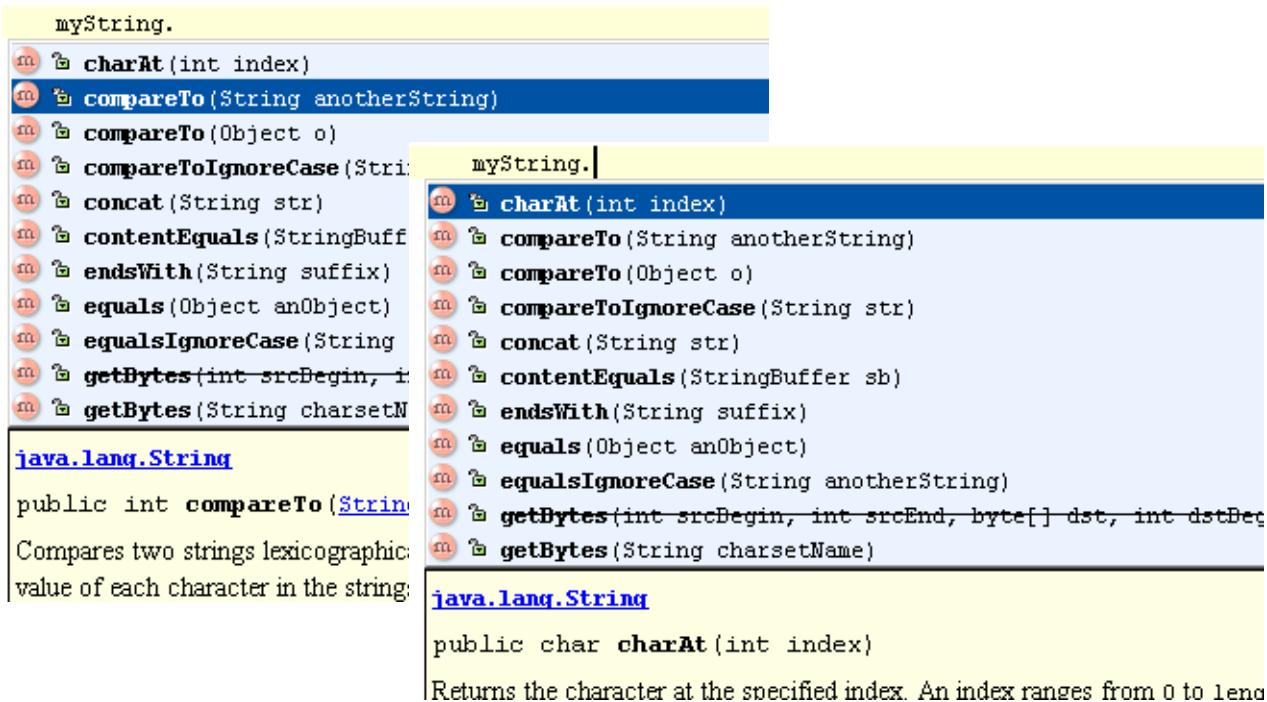


Figure 7.25. Autopopup of JavaDoc for suggestion (854,855)

~~8.2. Code completion OLD XXX~~

- ~~▲ 8.2.1. Code completion X (page 179)~~
- ~~▲ 8.2.2. Lookup list XXX (page 183)~~
- ~~▲ 8.2.3. Parameter info XXX (page 183)~~

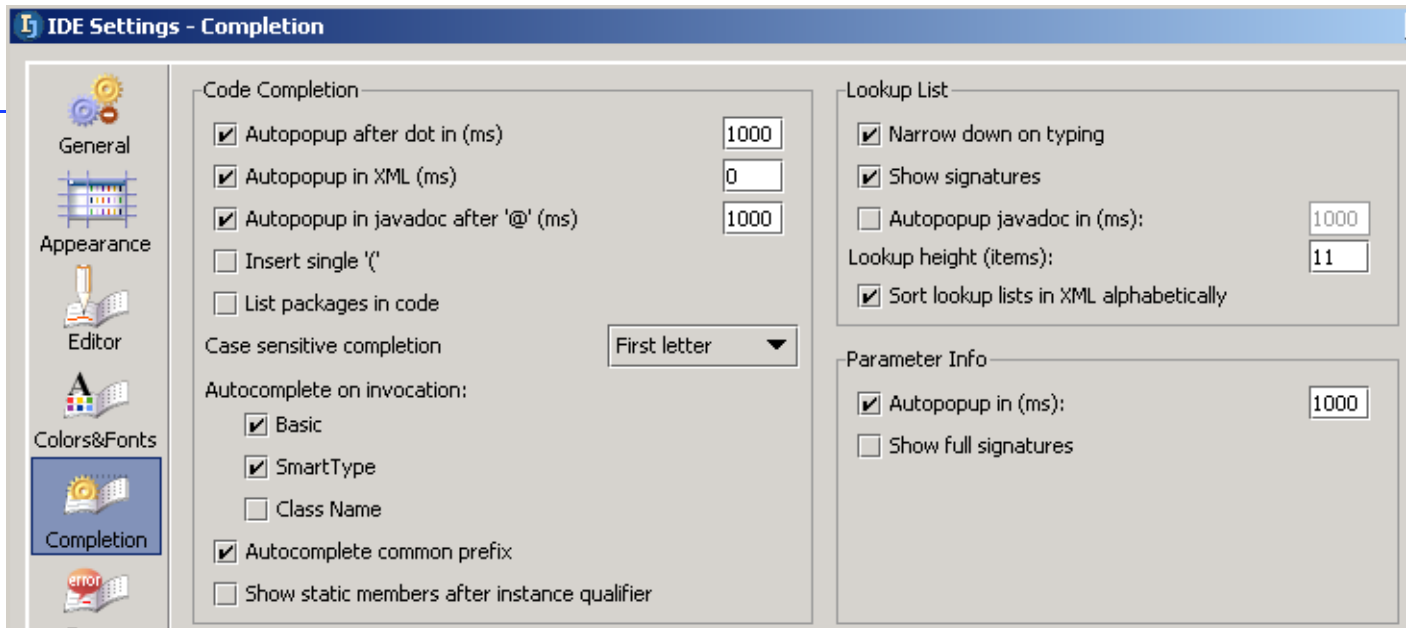


Figure 7.26. Completion ~~(812)~~

~~8.2.1. Code completion X~~

~~<help/topic/idea/preferences/editor/codeCompletion.html>~~

~~IDEA provides the following functions for automatically completing code:~~

- ~~▲ 8.2.1.1. Autopopup after dot (page 179)~~
- ~~▲ 8.2.1.2. Autopopup in XML X (page 180)~~
- ~~▲ 8.2.1.3. Autopopup in javadoc after '@' (page 180)~~
- ~~▲ 8.2.1.4. Insert single '' (page 180)~~
- ~~▲ 8.2.1.5. List packages in code (page 181)~~
- ~~▲ 8.2.1.6. Case sensitive completion (page 181)~~
- ~~▲ 8.2.1.7. Autocompletion on invocation XXX (page 182)~~
- ~~▲ 8.2.1.8. Autocomplete common prefix XXX (page 183)~~
- ~~▲ 8.2.1.9. Show static member after instance qualifier XXX (page 183)~~

~~8.2.1.1. Autopopup after dot~~

~~Determines if and after what delay an autopopup will appear after a period has been typed at the end of a declaration (assuming that the cursor does not move and nothing else is typed in after the dot).~~

~~7.93. Open **IDEA Settings | Completion**.~~

~~7.94. Check the checkbox **Autopopup after dot in (ms)**.~~

~~7.95. For the value enter **5000** (5 secs).~~

~~7.96. Click **OK**.~~

~~7.97. In the editor: Enter "**System.out.**". Note that after 5 seconds an autopopup appears.~~

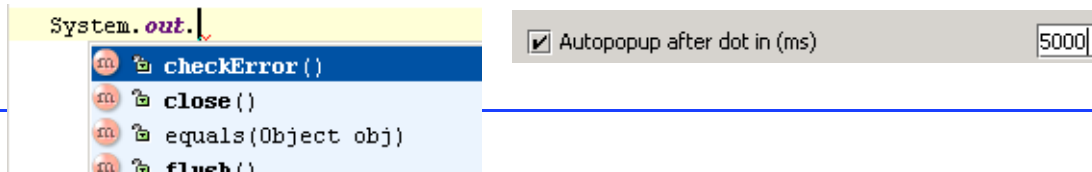


Figure 7.27. Autopopup (824.823)

7.98. Open **IDEA Settings | Completion**.

7.99. Uncheck the checkbox.

7.100. Click **OK**.

7.101. Again in the editor: Enter “**System.out.**”. Note that the popup does not appear.

7.102. Click **CTRL SPACE**. The popup appears.

7.103. Open **IDEA Settings | Completion**.

7.104. Check the checkbox.

7.105. For the value enter **1000** (1 sec).

7.106. Click **OK**.

8.2.1.2. Autopopup in XML X

[20021015TTT ?? example of this??](#)

Similar to “Autopop after dot”, but for XML files.

8.2.1.3. Autopopup in javadoc after ‘@’

Similar to “Autopopup after dot”, but for popups after the javadoc ‘@’.

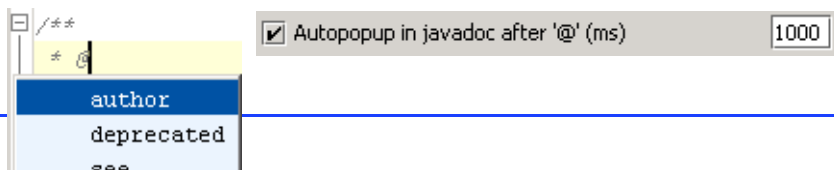


Figure 7.28. Autopopup in javadoc after ‘@’ (826.825)

8.2.1.4. Insert single ‘)’

7.107. Open **IDEA Settings | Completion**.

7.108. Check the checkbox **Insert single ‘)’**.

7.109. Click **OK**.

7.110. In the editor: Add the following:

```
String aString = "Hello";  
aString.getBytes
```

7.111. Place the cursor at the end of “getBytes”.

7.112. Press **CTRL SPACE**. A single ‘)’ appears.

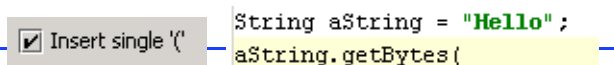


Figure 7.29. Insert single ‘)’ (827.828)

7.113. Delete the single ‘)’.

7.114. Open **IDEA Settings | Completion**.

7.115. Uncheck the checkbox **Insert single ‘)’**.

7.116. Click **OK**.

7.117. Place the cursor at the end of “getBytes”.

7.118. Press **CTRL SPACE**. ‘)’ appears.

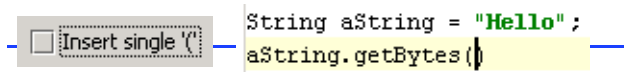


Figure 7.30. Insert '()' (829,830)

8.2.1.5. List packages in code

- 7.119. Open **IDEA Settings | Completion**.
- 7.120. Check the checkbox **List packages in code**.
- 7.121. Click **OK**.
- 7.122. In the editor: Add the following:
`String comString = "Hello";`
`e`
- 7.123. Place the cursor at the end of "e".
- 7.124. Press **CTRL SPACE**. A popup appears that includes packages.

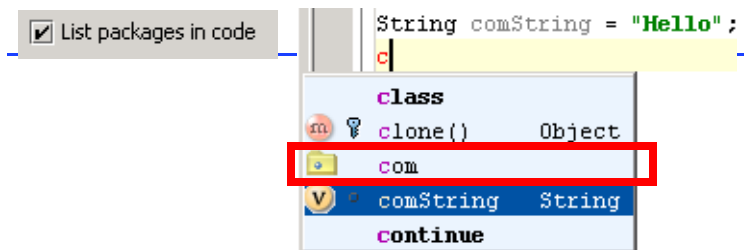


Figure 7.31. List packages in code (832,831)

- 7.125. Open **IDEA Settings | Completion**.
- 7.126. Uncheck the checkbox **List packages in code**.
- 7.127. Click **OK**.
- 7.128. Place the cursor at the end of "c".
- 7.129. Press **CTRL SPACE**. A popup appears that does not include packages.

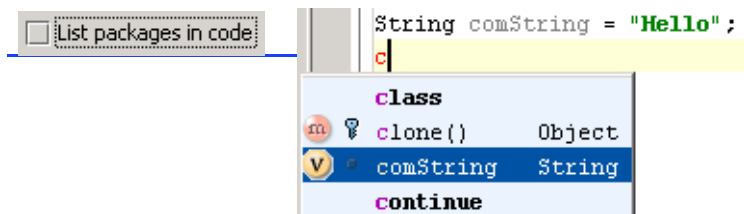


Figure 7.32. No packages in code (833,834)

8.2.1.6. Case sensitive completion

You can set the case sensitivity of IDEA to one of the following:

- ▲ **8.2.1.6.1. None (page 181)**
- ▲ **8.2.1.6.2. First letter (page 182)**
- ▲ **8.2.1.6.3. All (page 182)**

8.2.1.6.1. None

- 7.130. Open **IDEA Settings | Completion**.
- 7.131. For "Case sensitive completion" select **None**.
- 7.132. Uncheck the 3 checkboxes for "Autocomplete on invocation".
- 7.133. Click **OK**.
- 7.134. Create Class2 with the following content:
`package MyPackage;`
`public static class MyClass2 {`
`static int ABCDEF = 1;`

```
static int Abcdef = 1;  
static void abcdef(){};  
MyClass2.AB  
}
```

7.135. Place the cursor at the end of “MyClass2.AB”.

7.136. Press **CTRL SPACE**. A popup appears with all 3 options.

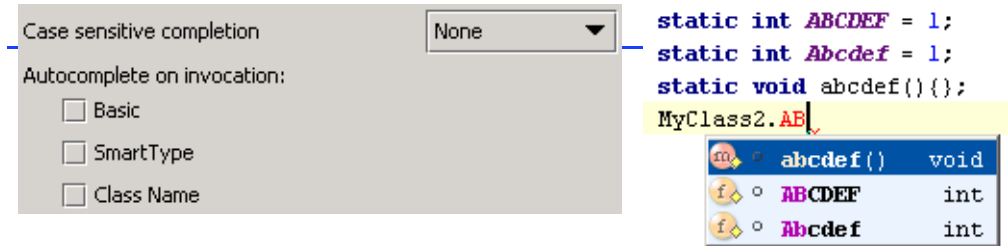


Figure 7.33. Case sensitive completion: None (835,836)

8.2.1.6.2. First letter

7.137. Open **IDEA Settings | Completion**.

7.138. For “Case sensitive completion” select **First letter**.

7.139. Click **OK**.

7.140. Place the cursor at the end of “MyClass2.AB”.

7.141. Press **CTRL SPACE**. A popup appears with 2 options.

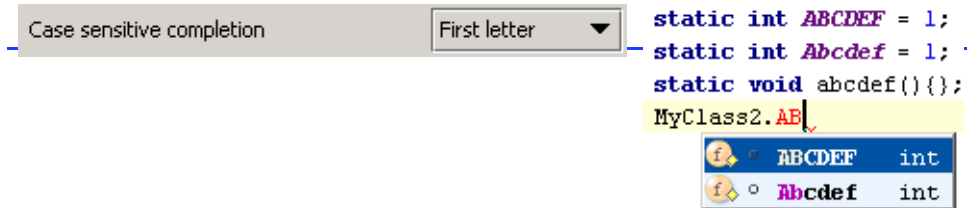


Figure 7.34. Case sensitive completion: First letter (837,838)

8.2.1.6.3. All

7.142. Open **IDEA Settings | Completion**.

7.143. For “Case sensitive completion” select **All**.

7.144. Click **OK**.

7.145. Place the cursor at the end of “MyClass2.AB”.

7.146. Press **CTRL SPACE**. A popup appears with 1 option.

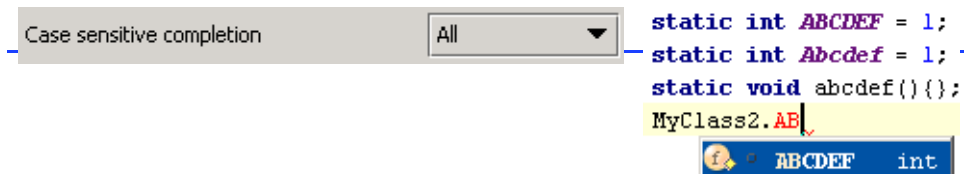


Figure 7.35. Case sensitive completion: All (839,840)

8.2.1.7. Autocompletion on invocation XXX

▲ **8.2.1.7.1. Basic XXX (page 182)**

▲ **8.2.1.7.2. SmartType XXX (page 183)**

▲ **8.2.1.7.3. Class Name XXX (page 183)**

8.2.1.7.1. Basic XXX

~~8.2.1.7.2. SmartType XXX~~

~~8.2.1.7.3. Class Name XXX~~

~~8.2.1.8. Autocomplete common prefix XXX~~

~~8.2.1.9. Show static member after instance qualifier XXX~~

~~8.2.2. Lookup list XXX~~

- ~~▲ 8.2.2.1. Narrow done on typing XXX (page 183)~~
- ~~▲ 8.2.2.2. Show signatures XXX (page 183)~~
- ~~▲ 8.2.2.3. Autopopup javadoc in XXX (page 183)~~
- ~~▲ 8.2.2.4. Lookup height (items) XXX (page 183)~~
- ~~▲ 8.2.2.5. Sort lookup lists in XML alphabetically XXX (page 183)~~

~~8.2.2.1. Narrow done on typing XXX~~

~~8.2.2.2. Show signatures XXX~~

~~8.2.2.3. Autopopup javadoc in XXX~~

~~8.2.2.4. Lookup height (items) XXX~~

~~8.2.2.5. Sort lookup lists in XML alphabetically XXX~~

~~8.2.3. Parameter info XXX~~

- ~~▲ 8.2.3.1. Autopopup in XXX (page 183)~~
- ~~▲ 8.2.3.2. Show full signatures XXX (page 183)~~

~~8.2.3.1. Autopopup in XXX~~

~~8.2.3.2. Show full signatures XXX~~

8.3. Code templates

IDEA provides the following code template functionality

- [8.3.1. Live Templates \(page 184\)](#)
- [8.3.2. Surround with \(page 191\)](#)

8.3.1. Live Templates

There are many different

- [8.3.1.1. Type of insert \(page 184\)](#)
- [8.3.1.2. Context \(page 187\)](#)
- [8.3.1.3. Edit / Add / Remove \(page 189\)](#)

8.3.1.1. Type of insert

The following types of live templates are available:

- [8.3.1.1.1. Plain text \(page 184\)](#)
- [8.3.1.1.2. With variables \(page 184\)](#)
- [8.3.1.1.3. Surround \(page 185\)](#)

8.3.1.1.1. Plain text

7.147. Create class **Class1**:

```
public class Class1 {  
    public static void main(String[] args) {  
  
    }  
}
```

7.148. Place the cursor inside main().

7.149. Enter **St**.

7.150. Click **TAB**. Note that the live template code is added.

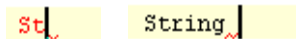


Figure 7.36. Insert plain text live template ([870,871](#))

Note: You can also enter the code by selecting **Code | Insert live template**. A list of live templates appears in a popup dialog. Double-clicking on a selection will add it.

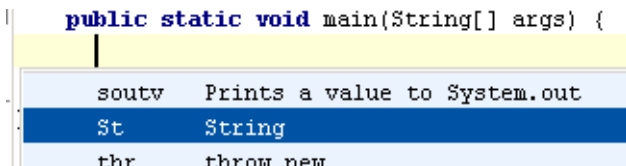


Figure 7.37. Insert using list ([872](#))

8.3.1.1.2. With variables

7.151. Create class **Class1**:

```
public class Class1 {  
    public static void main(String[] args) {  
  
    }  
}
```

7.152. Add live template **sout**. Note the final location of the cursor.


```
System.out.println("|");
```

Figure 7.38. Final location of cursor (873)

7.153. Select **Options | Live templates...**. The dialog “Live templates” appears.

7.154. Select **sout**. Note the variable “\$END\$”. This specifies the location of the cursor after the live template has been entered.

sout	Prints a string to System.out
soutm	Prints current class and method names to System.out

```
System.out.println("$END$");
```

Figure 7.39. \$END\$ variable (874)

7.155. Insert live template **itar**. An iteration is inserted.

```

J*\Class1.java
public class Class1 {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++) {
            String arg = args[i];
        }
    }
}

```

Figure 7.40. Inserted iteration (352)

7.156. Press the **j** key. Note that the iteration variable is changed.

```

public static void main(String[] args) {
    for (int j = 0; j < args.length; j++) {
        String arg = args[j];
    }
}

```

Figure 7.41. Changed iteration variable (351)

7.157. Click the **Tab** key. Note that the red square (the focus) has move to “args”.

```

for (int j = 0; j < args.length; j++) {
    String arg = args[j];
}

```

Figure 7.42. Focus moved (350)

The variable definitions explain the above behavior.

itar	Iterate elements of array
------	---------------------------

```

for (int $INDEX$ = 0; $INDEX$ < $ARRAY$.length; $INDEX$++) {
    $ELEMENT_TYPE$ $VAR$ = $ARRAY[$INDEX$];
    $END$
}

```

Figure 7.43. itar variables (876,875)

8.3.1.1.3. Surround

7.158. Select the contents of main

7.159. Select **Code | Surround with live template**. A list of surround live templates appears in a popup dialog.

7.160. Click on **B: Surround with {}**. The code is surrounded.

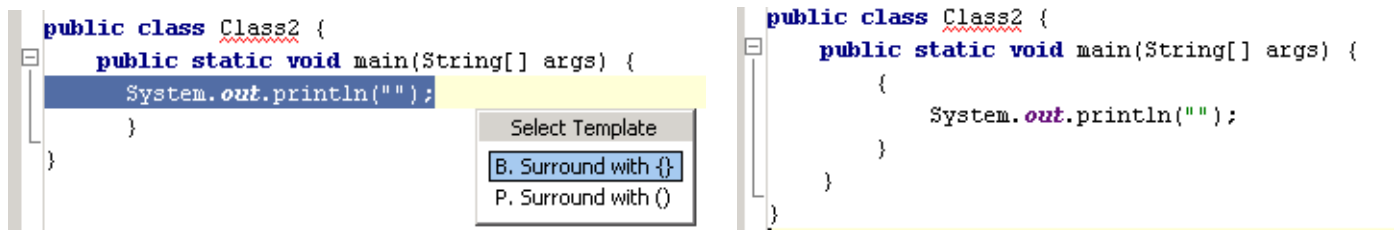


Figure 7.44. Surround with {} (877,878)

8.3.1.2. Context

The context determines how the live template can be used. The following contexts are supported:

- **8.3.1.2.1. Java code (page 187)**
- **8.3.1.2.2. Java comment (page 187)**
- **8.3.1.2.3. Java String (page 187)**
- **8.3.1.2.4. Smart type completion (page 188)**
- **8.3.1.2.5. HTML (page 188)**
- **8.3.1.2.6. XML (page 188)**
- **8.3.1.2.7. JSP (page 188)**
- **8.3.1.2.8. Other (page 188)**

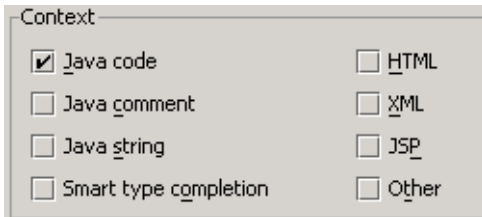


Figure 7.45. Contexts (879)

8.3.1.2.1. Java code

If this context is selected, then the live template can be inserted into java code. In the examples above, you added live templates to java code.

7.161. Add `<` to the code.

7.162. Click **TAB**. The live template is not entered since this template is only supported in HTML, XML, and JSP files.

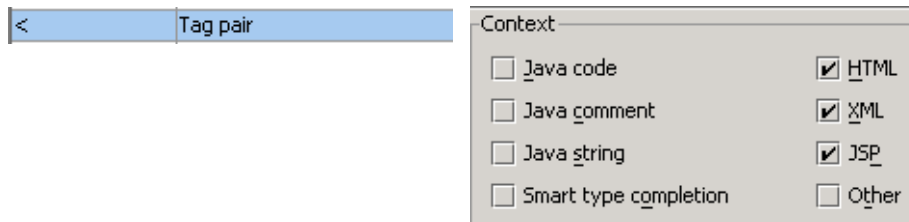


Figure 7.46. Tag pair contexts (880,881)

8.3.1.2.2. Java comment

If this context is selected, then the live template can be inserted into java comments.

7.163. For Tag pair: Check content **Java comment**.

7.164. Add `<` to the code comment.

7.165. Click **TAB**. The live template is entered.

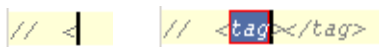


Figure 7.47. Tag pair in comments (882,883)

8.3.1.2.3. Java String

If this context is selected, then the live template can be inserted into java strings.

7.166. For Tag pair: Check content **Java string**.

7.167. Add `<` to the string in System.out.println.

7.168. Click **TAB**. The live template is entered.

```
System.out.println("a st<ring>"); System.out.println("a st<tag></tag>ring");
```

Figure 7.48. Tag pair in a string (884,885)

8.3.1.2.4. Smart type completion

If this context is selected, then the live template will be listed in a Smart type completion.

7.169. Add the following:

```
String s = "";  
s.compareTo()
```

7.170. Place the cursor between '()'

7.171. Click **CTRL-SHIFT-SPACE**. A list of the Smart type completions appear. Note the live templates (all of which have the checkbox "Smarttype completion" checked).

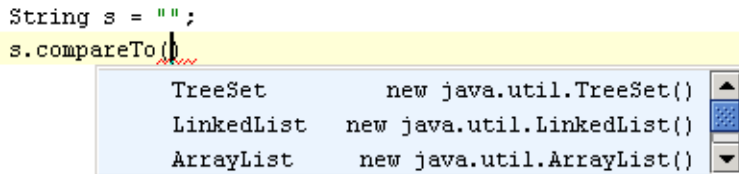


Figure 7.49. Smart type completion (886)

8.3.1.2.5. HTML

If this context is selected, then the live template can be inserted into HTML files.

7.172. Add the < live template to an HTML file.



Figure 7.50. Live template < in HTML file (859,860,861)

8.3.1.2.6. XML

If this context is selected, then the live template can be inserted into XML files.

8.3.1.2.7. JSP

If this context is selected, then the live template can be inserted into JSP files.

8.3.1.2.8. Other

[20021017TTT what is this used for ??](#)

If this context is selected, then the live template can be inserted into other types of files.

8.3.1.3. Edit / Add / Remove

The following functions are available for managing live templates:

- **8.3.1.3.1. Edit (page 189)**
- **8.3.1.3.2. Add / Copy (page 190)**
- **8.3.1.3.3. Remove (page 190)**

8.3.1.3.1. Edit

7.173. Select **Options | Live templates....** The dialog “Live templates” appears.

7.174. Double-click on **soutm**. The dialog “Edit Live Template” appears.

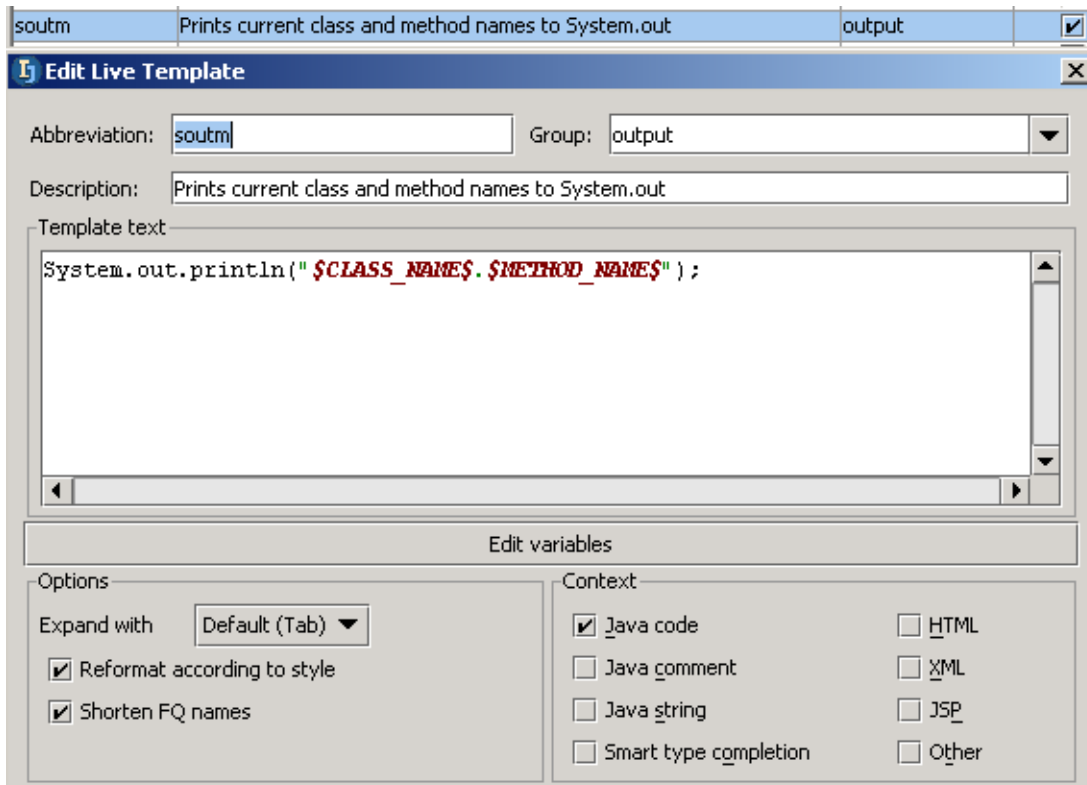


Figure 7.51. Edit live template for soutm (887)

Note the following properties.

Abbreviation

The abbreviation appears in dialogs and is used to insert the live template using {abbreviation}+TAB.

Group

The group is used to organize live templates in the “Live Templates” dialog.

Description

Description of the live template.

Template text

The code that is inserted. The code can contain variables.

Edit variables

Click to open the dialog “Edit template variables”.

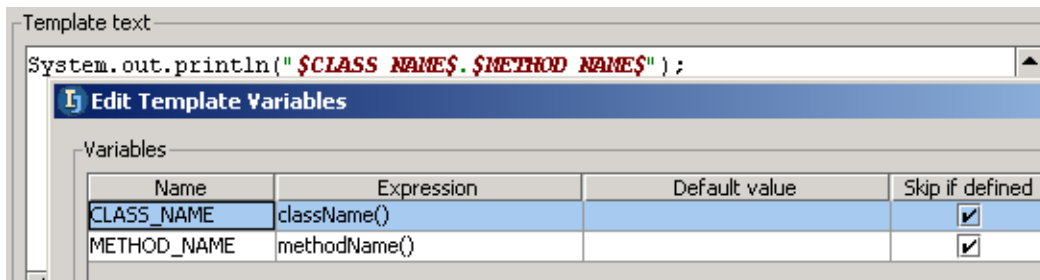


Figure 7.52. Edit template variables (888)

[20021017TTT what are these used for ??](#)

Name

Name of the variable.

Expression

Variable expression

Default value

The default value.

Skip if defined

Skip if defined if checked.

Expand with

Specifies the key that is pressed after the live template abbreviation in order to add the template.

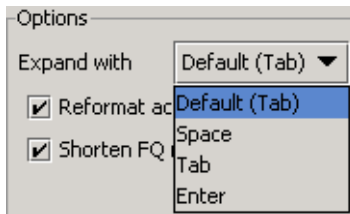


Figure 7.53. Expand with options (889)

Reformat according to style

If checked: Reformat according to style.

Shorten FQ names

If checked: FQ names are shortened.

Context

Check to enable the live templates in contexts (demonstrated earlier).

8.3.1.3.2. Add / Copy

Click **Add** to open an empty “Edit Live Template” dialog.

Click **Copy** to open a copy of the selected live template in the “Edit Live Template” dialog.

8.3.1.3.3. Remove

Click **Remove** to remove the selected template.

8.3.2. Surround with

7.175. Create class **Class1**:

```
public class Class1 {  
    public void aMethod() {  
        System.out.println("Hello");  
    }  
}
```

7.176. Place the caret in “System.out....”.

7.177. Select **Code | Surround with....** A popup dialog appears.

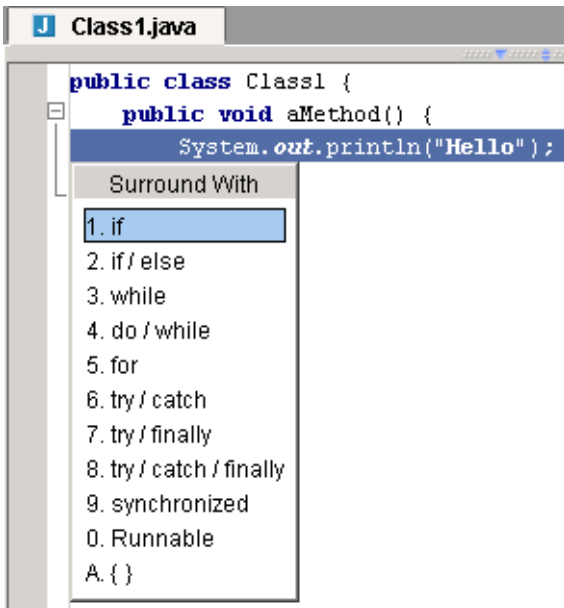


Figure 7.54. Dialog “Surround with” [\(361\)](#)

7.178. Select **If**. The line is surrounded with an “if” clause.

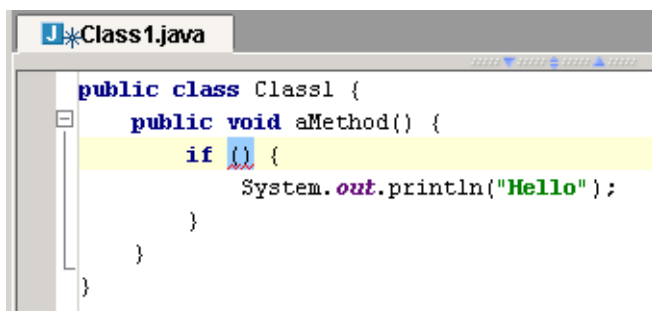


Figure 7.55. Surrounded with an “if” clause [\(360\)](#)

8.4. Code generation

IDEA can automatically generate some types of code (such as getter and setter methods).

7.179. Create class **Class1**:

```
public class Class1 {  
    int int1;  
    int int2;  
}
```

7.180. Select **Code | Generated....** A popup dialog appears.

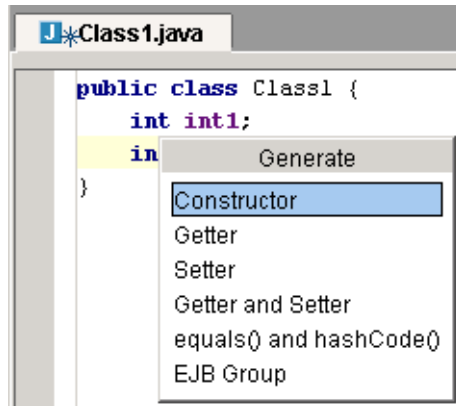


Figure 7.56. Popup “Generate” (359)

7.181. Select **Constructor**. The dialog “Choose field to initialize by constructor” appears.

7.182. Select both **int1** and **int2** (using CTRL key).

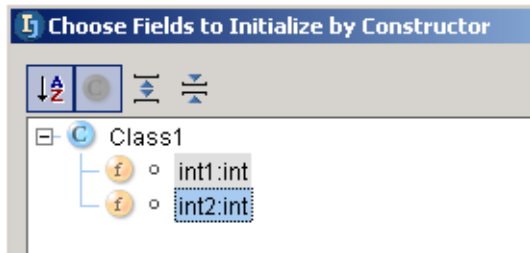


Figure 7.57. Dialog “Choose field to initialize by constructor” (358)

7.183. Click **OK**. The constructor is generated.

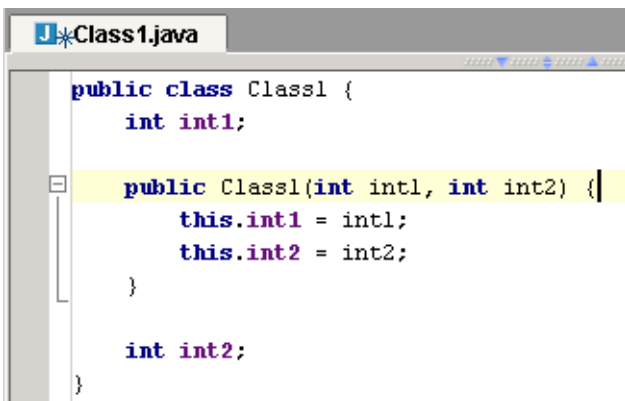


Figure 7.58. Generated constructor (357)

8.5. Import optimization

IDEA can optimize imports for

- [8.5.1. File \(page 193\)](#)
- [8.5.2. Files in directory \(page 193\)](#)

8.5.1. File

7.184. Create class **MyClass2**:

```
import java.util.*;  
import com.sun.*;  
package MyPackage  
public class MyClass2 {}
```

7.185. Select **Tools | Optimize imports...** The dialog “Optimize imports” appears.

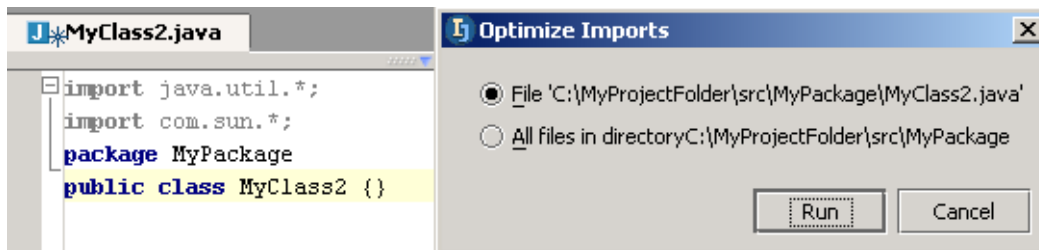


Figure 7.59. Dialog “Optimize imports” [\(339\)](#)

7.186. Select **Run**. Imports are optimized.

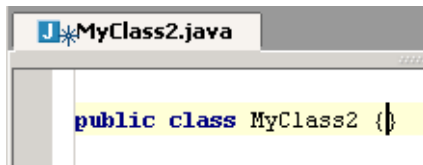


Figure 7.60. Optimized imports [\(338\)](#)

8.5.2. Files in directory

In the dialog “Optimize Imports”: Select **All files in directory...** to optimize the imports for all files in the directory.

8.6. Method override

IDEA makes it easy to override methods.

7.187. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {}
```

7.188. Place the cursor on the line “public...”.

7.189. Select **Code | Override methods**. The dialog “Select methods to override” appears.

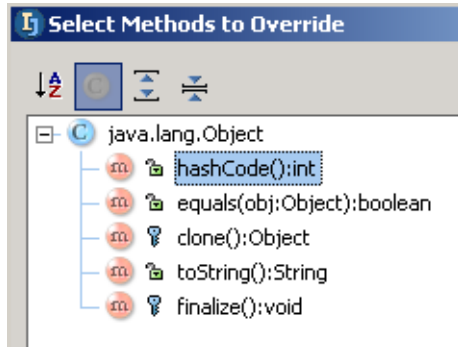


Figure 7.61. Override methods (453)

7.190. Double-click on **toString**. The code to override the method is added:

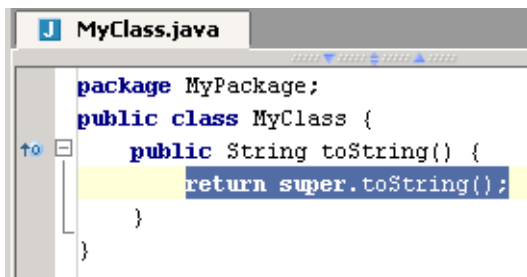


Figure 7.62. Overridden method toString() (452)

7.191. Place the cursor on the overridden method icon (). Note the message that appears:

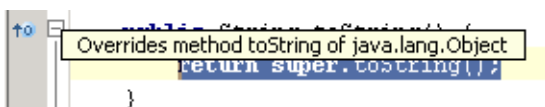


Figure 7.63. Overridden method message (451)

7.192. Click on the icon. The source file for the overridden method is opened.

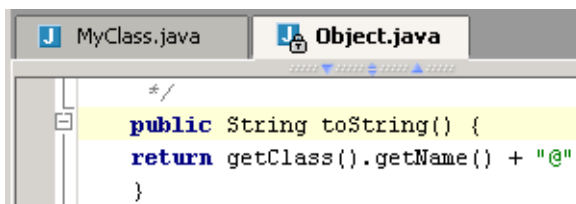


Figure 7.64. Overridden method (450)

7.193. Create class **MyClass2**:

```
package MyPackage;  
public class MyClass2 extends MyClass {}
```

7.194. Place the cursor on the line “public...”.

7.195. Select **Code | Override methods**. The dialog “Select methods to override” appears. Note that the methods for all superclasses are shown.

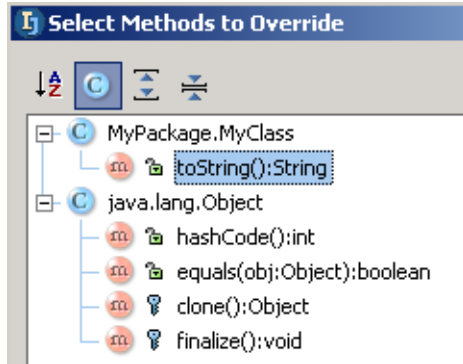


Figure 7.65. Override methods [\(448\)](#)

8.7. Interface implementation

IDEA makes it easy to implement interfaces.

7.196. Create class **Interface1**:

```
public interface Interface1 {  
    void doSomething();  
}
```

7.197. Create class **ImplementInterface1**:

```
public class ImplementInterface1 implements Interface1 {  
}
```

7.198. Place the caret with “Class1” after “new”.

7.199. Select **Code | Implement methods**. The dialog “Select methods to implement” appears.

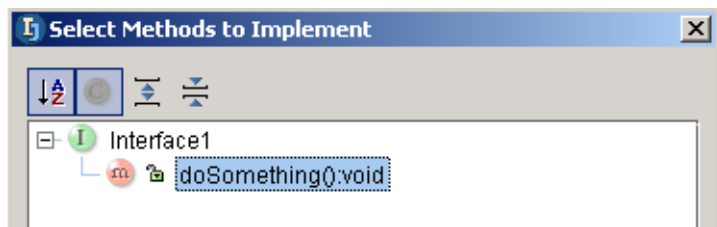


Figure 7.66. Dialog “Select methods to implement” (366)

7.200. Select **doSomething():void**.

7.201. Click **OK**. The method is implemented.

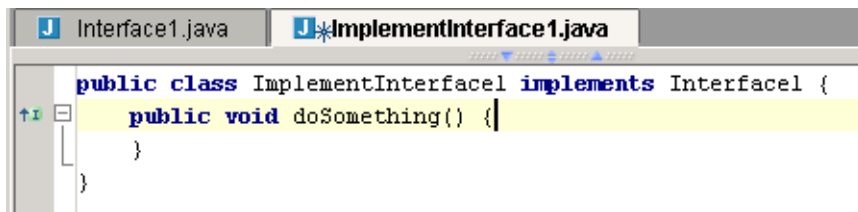


Figure 7.67. Method implemented (365)

8.8. Method delegation

IDEA makes it easy to delegate methods.

7.202. Create class **Class1**:

```
public class Class1 {  
    public Class2 c2;  
    public Class3 c3;  
}
```

7.203. Create class **Class2**:

```
public class Class2 {  
    public void method2a() {}  
    public void method2b() {}  
}
```

7.204. Create class **Class3**:

```
public class Class3 {  
    public void method3() {}  
}
```

7.205. Place the caret in Class1.

7.206. Select **Code | Delegate methods**. The dialog “Select target to generate delegates for” appears.

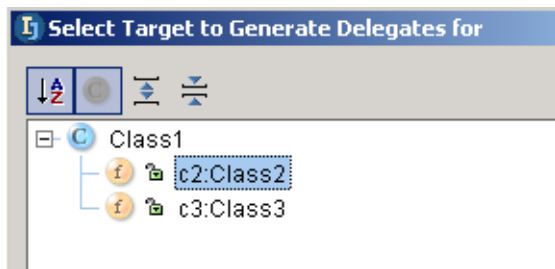


Figure 7.68. Dialog “Select target to generate delegates for” [\(364\)](#)

7.207. Select **c2:Class2**. The dialog “Select methods to generate delegates for” appears.

7.208. Select both **c2** and **c3** (using CTRL key).

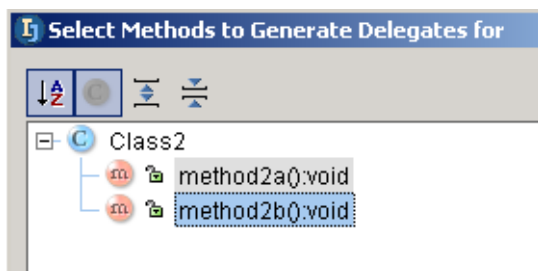
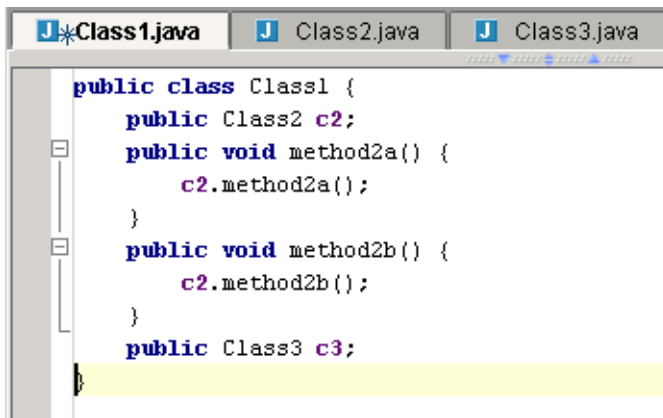


Figure 7.69. Dialog “Select methods to generate delegates for” [\(363\)](#)

7.209. Click **OK**. The delegates are generated.



```
public class Class1 {  
    public Class2 c2;  
    public void method2a() {  
        c2.method2a();  
    }  
    public void method2b() {  
        c2.method2b();  
    }  
    public Class3 c3;  
}
```

Figure 7.70. Delegates generated [\(362\)](#)

8.9. Comment

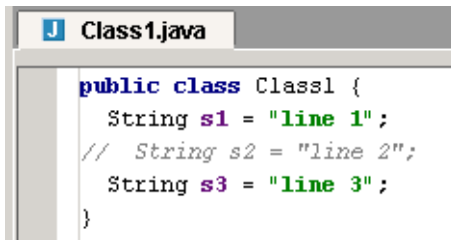
IDEA makes it easy to comment out code.

7.210. Create class **Class1**:

```
public class Class1 {  
    String s1 = "line 1";  
    String s2 = "line 2";  
    String s3 = "line 3";  
}
```

7.211. Place the cursor on a line.

7.212. Select **Code | Comment with line comment..** The line is commented.

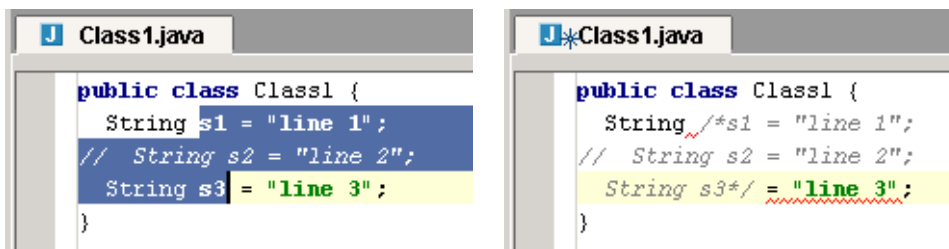


```
J Class1.java  
public class Class1 {  
    String s1 = "line 1";  
    // String s2 = "line 2";  
    String s3 = "line 3";  
}
```

Figure 7.71. Line comment ([356](#))

7.213. Select a portion of the code.

7.214. Select **Code | Comment with block comment..** The block is commented.



```
J Class1.java  
public class Class1 {  
    String s1 = "line 1";  
    // String s2 = "line 2";  
    String s3 = "line 3";  
}
```

```
J*Class1.java  
public class Class1 {  
    String /*s1 = "line 1";  
    // String s2 = "line 2";  
    String s3*/ = "line 3";  
}
```

Figure 7.72. Block comment ([355.354](#))

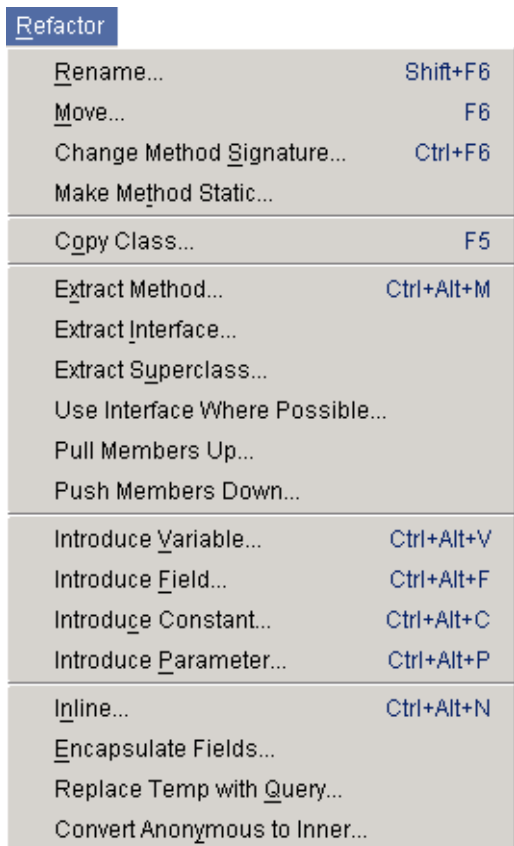
9. Code Refactoring

[20021022TTT last update.](#)

[Consult dima. valya.](#)

The following refactoring operations are supported:

- ~~9.1. Migration XXX (page 202)~~
- 9.2. Rename (page 203)
- 9.3. Move (page 211)
- 9.4. Change method signature (page 216)
- 9.5. Copy class (page 220)
- 9.6. Extract (page 221)
- 9.7. Use interface where possible (page 226)
- 9.8. Pull/push members (page 228)
- 9.9. Introduce (page 230)
- 9.10. Inline (page 234)
- 9.11. Encapsulate field (page 237)
- 9.12. Replace temp with query (page 239)
- 9.13. Convert Anonymous to Inner (page 240)



Refactor	
Rename...	Shift+F6
Move...	F6
Change Method Signature...	Ctrl+F6
Make Method Static...	
Copy Class...	F5
Extract Method...	Ctrl+Alt+M
Extract Interface...	
Extract Superclass...	
Use Interface Where Possible...	
Pull Members Up...	
Push Members Down...	
Introduce Variable...	Ctrl+Alt+V
Introduce Field...	Ctrl+Alt+F
Introduce Constant...	Ctrl+Alt+C
Introduce Parameter...	Ctrl+Alt+P
Inline...	Ctrl+Alt+N
Encapsulate Fields...	
Replace Temp with Query...	
Convert Anonymous to Inner...	

Figure 8.1. Refactoring menu items [\(414\)](#)

~~9.1. Migration XXX~~

- ~~• 9.1.1. Migrate (page 202)~~
- ~~• 9.1.2. Create map (page 202)~~

~~9.1.1. Migrate~~

~~9.1.2. Create map~~

9.2. Rename

IDEA refactoring allows you rename a

- [9.2.1. Package \(page 203\)](#)
- [9.2.2. Class \(page 205\)](#)
- [9.2.3. Method \(page 207\)](#)
- [9.2.4. Field \(page 208\)](#)
- [9.2.5. Variable \(page 209\)](#)
- [9.2.6. Parameter \(page 210\)](#)

9.2.1. Package

8.1. Create class MyClass:

```
package MyPackage;  
import MyPackage2.MyClass2;  
public class MyClass {  
    static int field1 = 1;  
    static int field2;  
    public static void main(String[] args) {  
        MyClass2 mc2 = new MyClass2();  
        field2 = mc2.method2(field1);  
        System.out.println(field2);  
    }  
}
```

8.2. Create class MyClass2:

```
package MyPackage2;  
public class MyClass2 {  
    public int method2(int param2) {  
        int var2 = 2;  
        return var2 + param2;  
    }  
}
```

8.3. Right-click on **MyPackage**.

8.4. Select **Refactor | Rename**. The dialog “Rename” appears.

8.5. For the new name enter **MyPackageNEW**.

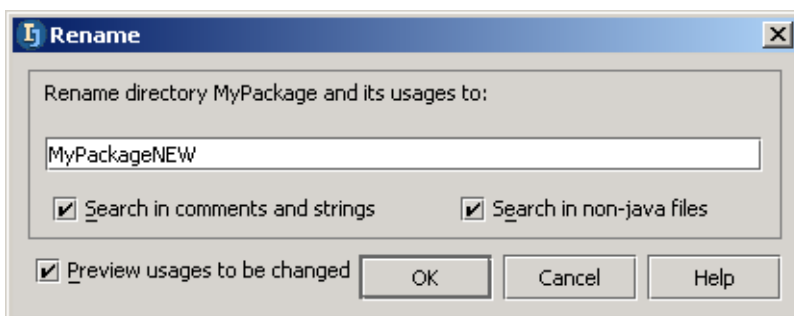


Figure 8.2. Dialog “Rename” (for package) [\(515\)](#)

8.6. Click **OK**. The panel “Find - Refactoring preview” appears.

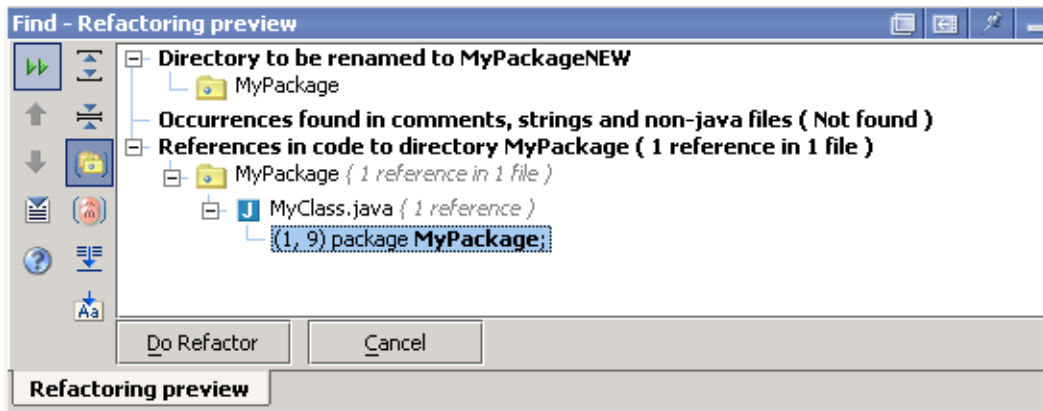


Figure 8.3. Panel “Find - Refactoring preview” (for package) (514)
8.7. Click on **Do Refactor**. The package is renamed.

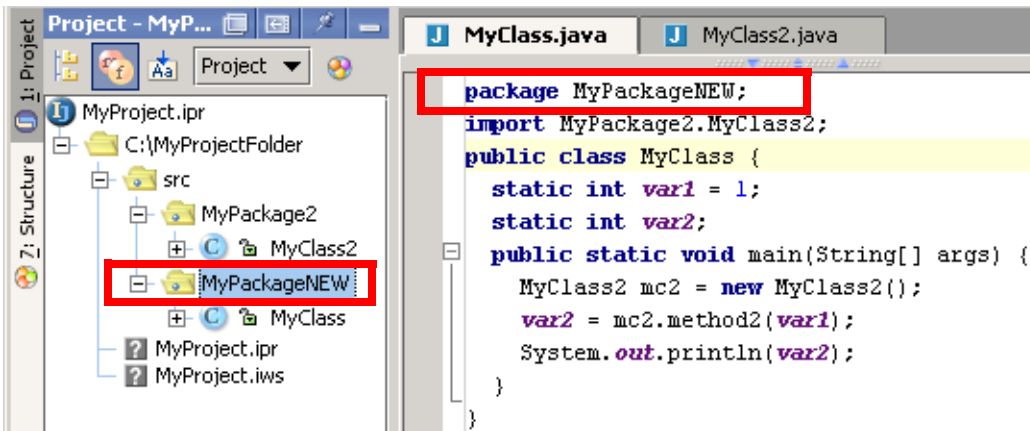


Figure 8.4. Renamed package (513)

9.2.2. Class

- 8.8. In the “Project” window: Right-click on **MyClass2**.
- 8.9. Select **Refactor | Rename**. The dialog “Rename” appears.
- 8.10. For the new name enter **MyClass2NEW**.

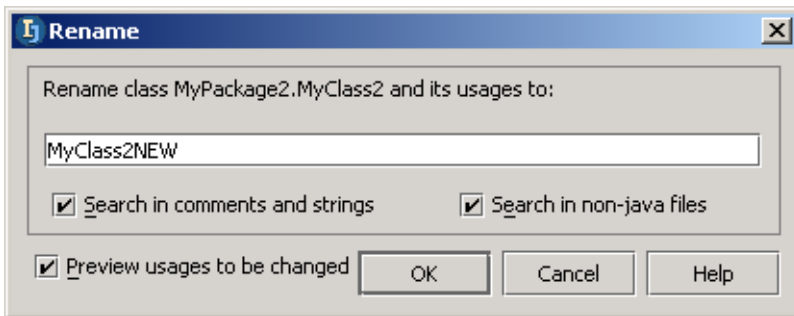


Figure 8.5. Dialog “Rename” (for class) (511)

- 8.11. Click **OK**. The panel “Find - Refactoring preview” appears.

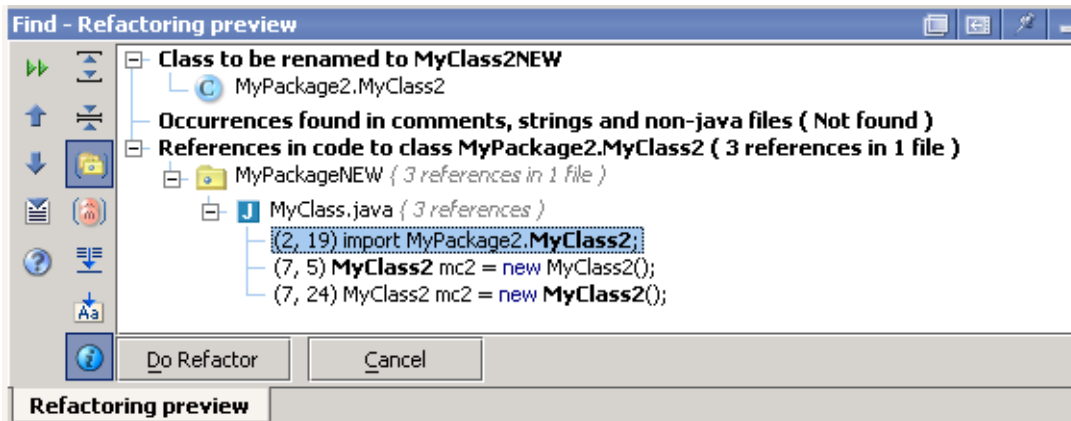


Figure 8.6. Panel “Find - Refactoring preview” (for class) (510)

- 8.12. Click on **Do Refactor**. The class is renamed.

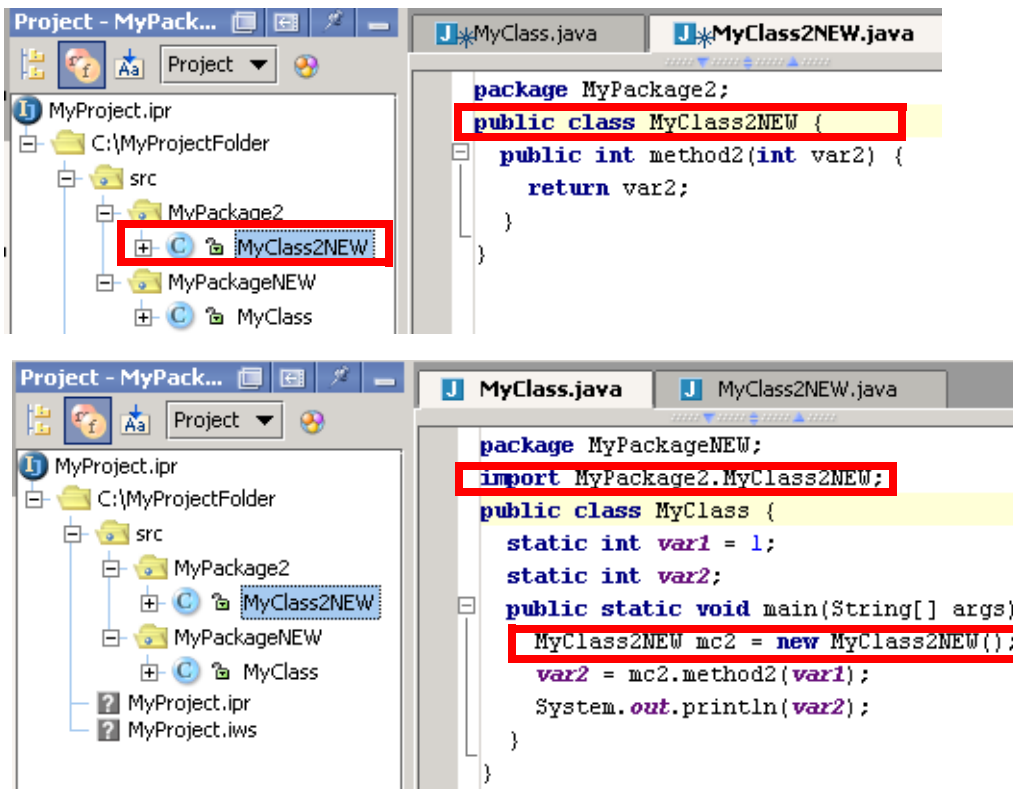


Figure 8.7. Renamed class (509,508)

9.2.3. Method

8.13. In MyClass2NEW: Place the cursor on the declaration of **method2**.

8.14. Right-click.

8.15. Select **Refactor | Rename**. The dialog “Rename” appears.

8.16. For the new name enter **method2NEW**.

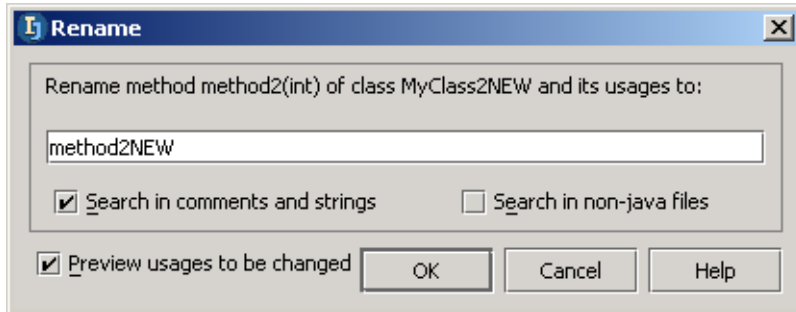


Figure 8.8. Dialog “Rename” (for method) (507)

8.17. Click **OK**. The panel “Find - Refactoring preview” appears.

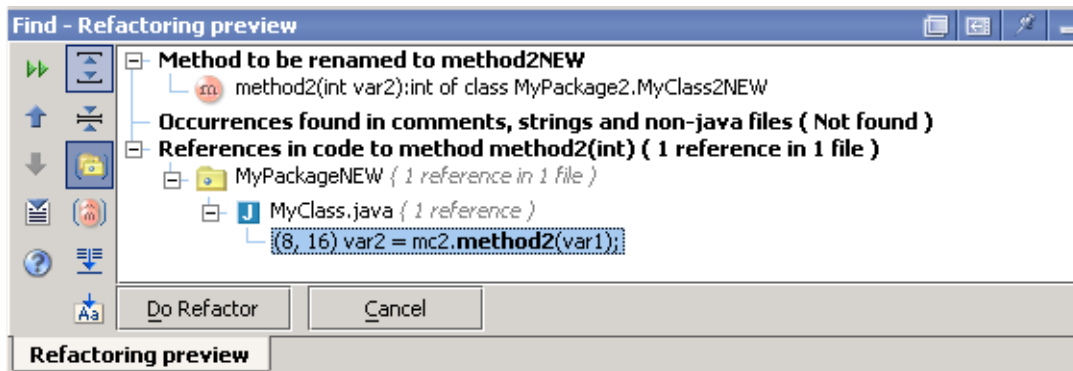


Figure 8.9. Panel “Find - Refactoring preview” (for method) (506)

8.18. Click on **Do Refactor**. The method is renamed.

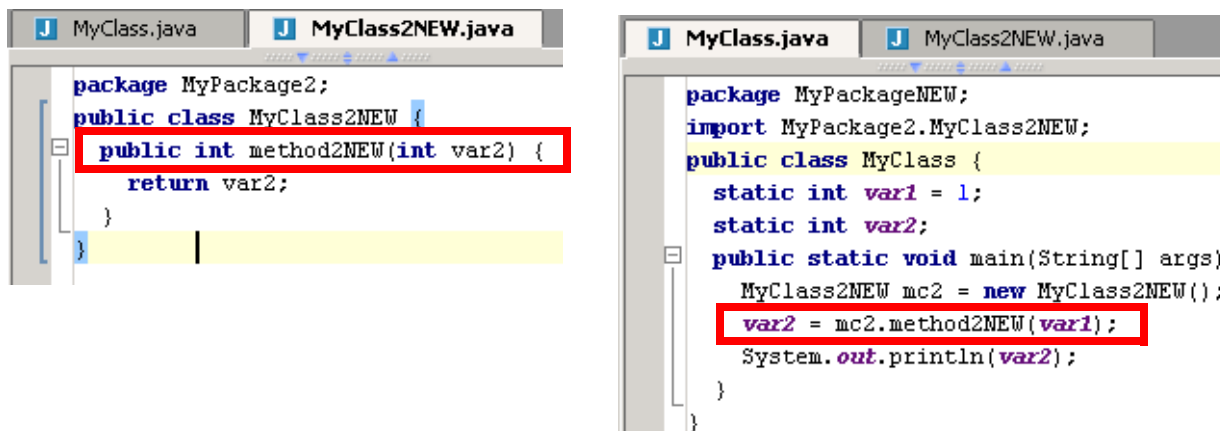


Figure 8.10. Renamed method (505,504)

9.2.4. Field

8.19. In the code for MyClass: Place the cursor on the declaration of **field1**.

8.20. Right-click.

8.21. Select **Refactor | Rename**. The dialog “Rename” appears.

8.22. For the new name enter **field1NEW**.

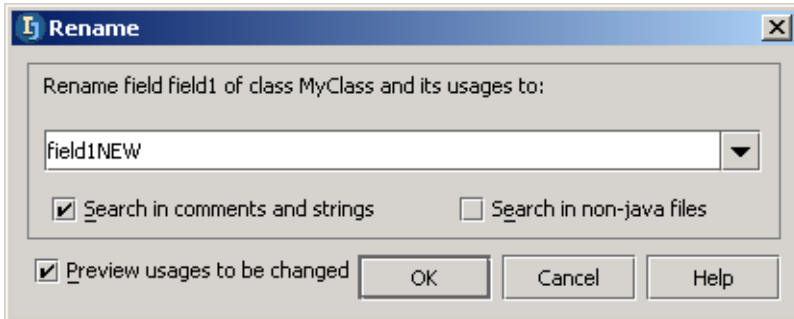


Figure 8.11. Dialog “Rename” (for field) (503)

8.23. Click **OK**. The panel “Find - Refactoring preview” appears.

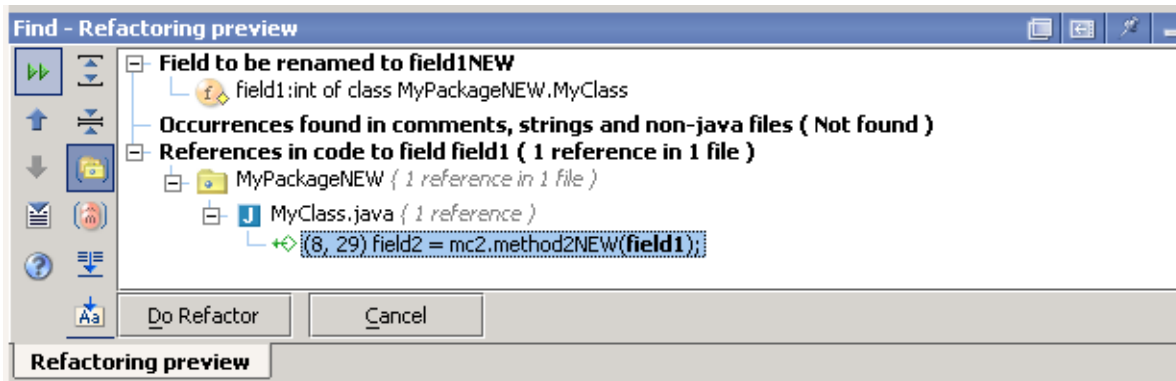


Figure 8.12. Panel “Find - Refactoring preview” (for field) (502)

8.24. Click on **Do Refactor**. The field is renamed.

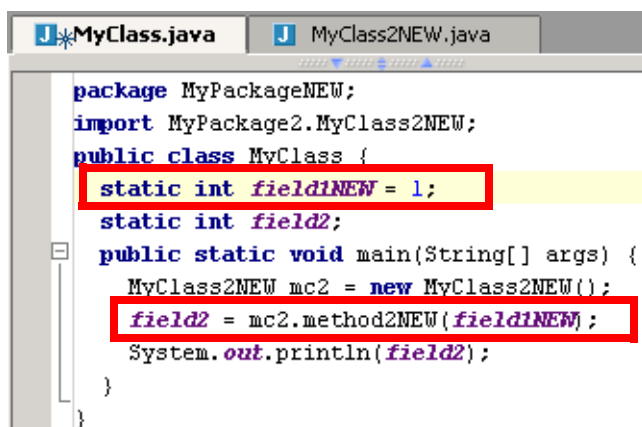


Figure 8.13. Renamed field (501)

9.2.5. Variable

8.25. In the code for MyClass2NEW: Place the cursor on the declaration of **var2**.

8.26. Right-click.

8.27. Select **Refactor | Rename**. The dialog “Rename” appears.

8.28. For the new name enter **var2NEW**.

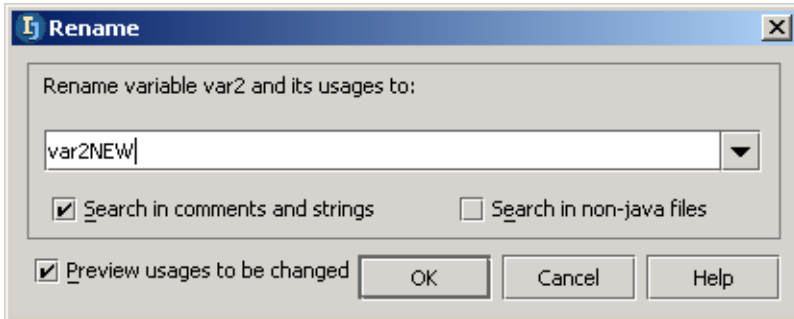


Figure 8.14. Dialog “Rename” (for variable) (918)

8.29. Click **OK**. The panel “Find - Refactoring preview” appears.

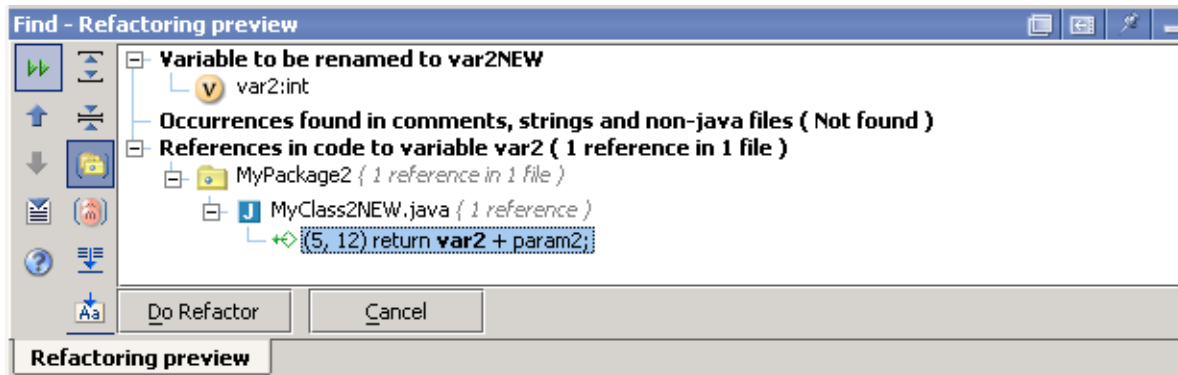


Figure 8.15. Panel “Find - Refactoring preview” (for field) (919)

8.30. Click on **Do Refactor**. The variable is renamed.

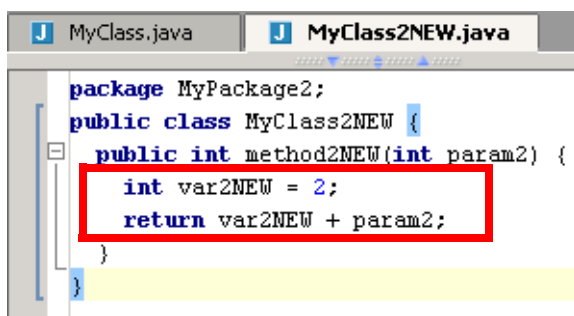


Figure 8.16. Renamed variable (920)

9.2.6. Parameter

- 8.31. In the code for MyClass2NEW: Place the cursor on **param2**.
- 8.32. Right-click.
- 8.33. Select **Refactor | Rename**. The dialog “Rename” appears.
- 8.34. For the new name enter **param2NEW**.

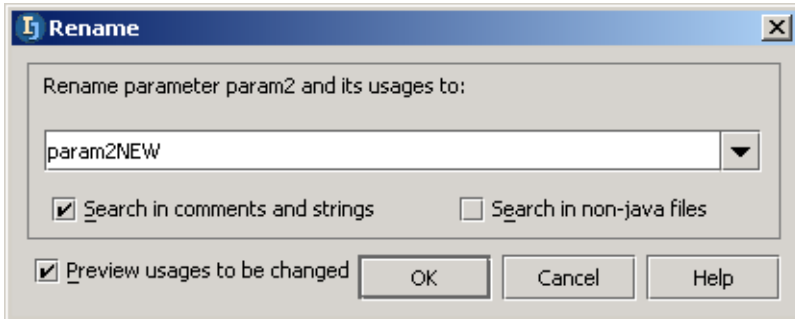


Figure 8.17. Dialog “Rename” (for parameter) (500)

- 8.35. Click **OK**. The panel “Find - Refactoring preview” appears.

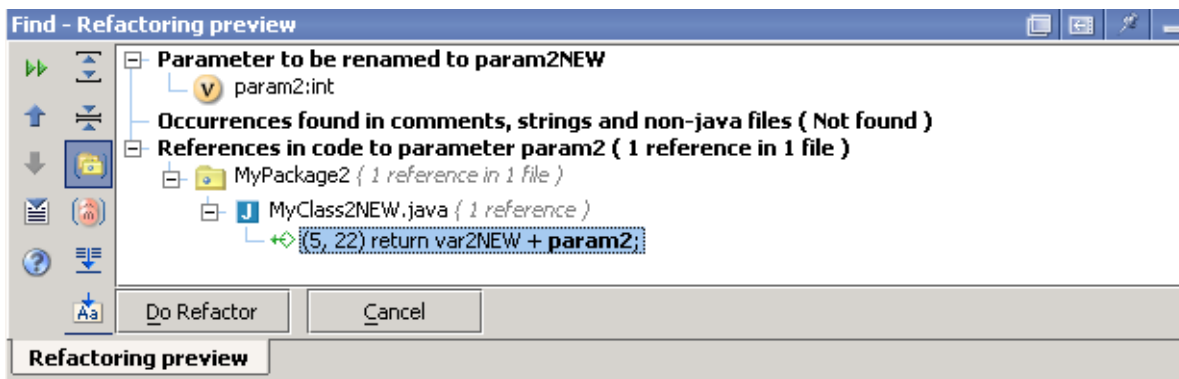


Figure 8.18. Panel “Find - Refactoring preview” (for parameter) (499)

- 8.36. Click on **Do Refactor**. The parameter is renamed.

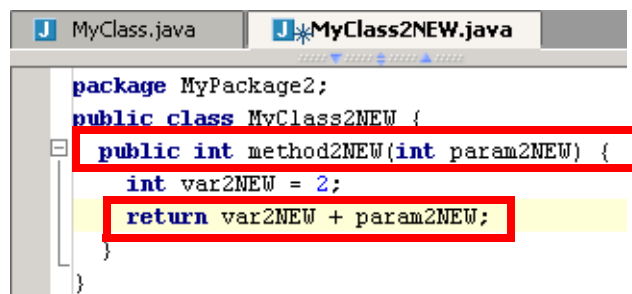


Figure 8.19. Renamed paramter (498)

9.3. Move

IDEA refactoring allows you move a

- [9.3.1. Package \(page 211\)](#)
- [9.3.2. Class \(page 213\)](#)
- [9.3.3. Members \(page 214\)](#)
- [9.3.4. Inner to upper level \(page 215\)](#)

9.3.1. Package

8.37. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {}
```

8.38. Create class **MyClass2**:

```
package MyPackage2;  
public class MyClass2 {  
    public static void myMethod2A() {}  
    public void myMethod2B() {  
        myMethod2A();  
    }  
    public class MyInnerClass2 {  
        public void myInnerClass2Method() {  
            myMethod2A();  
        }  
    }  
}
```

8.39. Right-click on **MyPackage2**.

8.40. Select **Refactor | Move**. The dialog “Move” appears.

8.41. For “To package:” enter **MyPackage**.

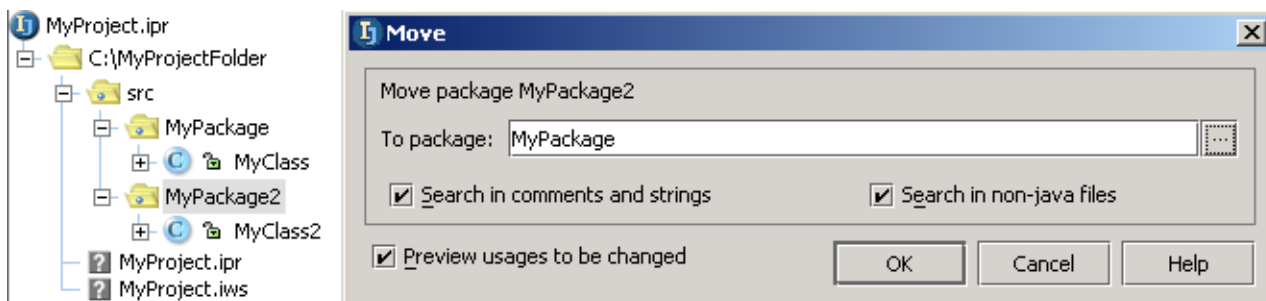


Figure 8.20. Move package dialog [\(380\)](#)

8.42. Click **OK**. The “Find - Refactoring preview” dialog appears.

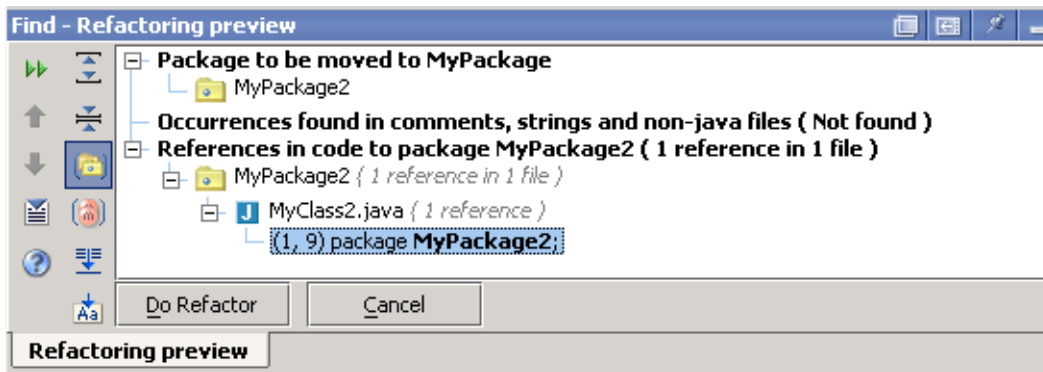


Figure 8.21. Find Refactoring preview (379)

8.43. Click **Do Refactor**. The package is moved.

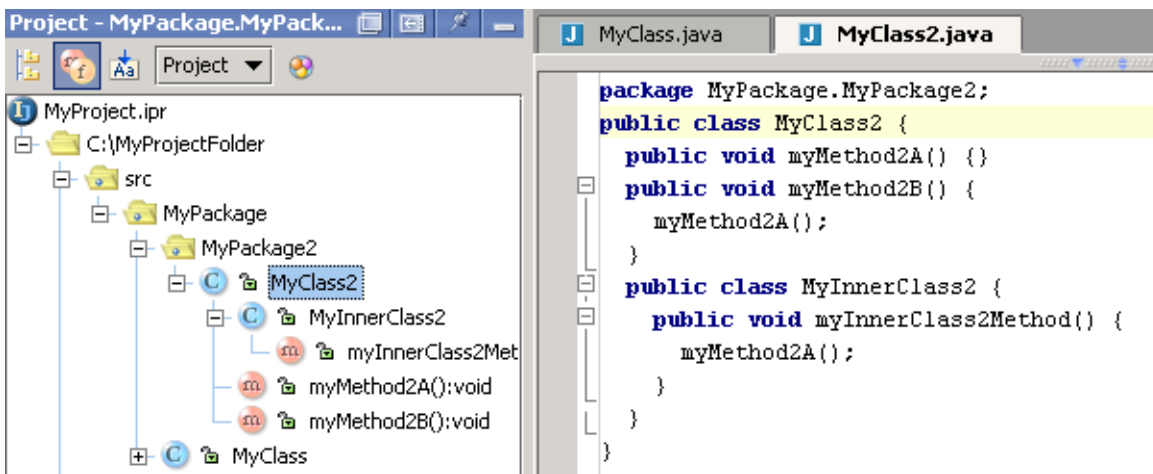


Figure 8.22. Package moved (378)

9.3.2. Class

8.44. Right-click on **MyClass2**.

8.45. Select **Refactor | Move**. The dialog “Move” appears.

8.46. For “To package” select **MyPackage**.

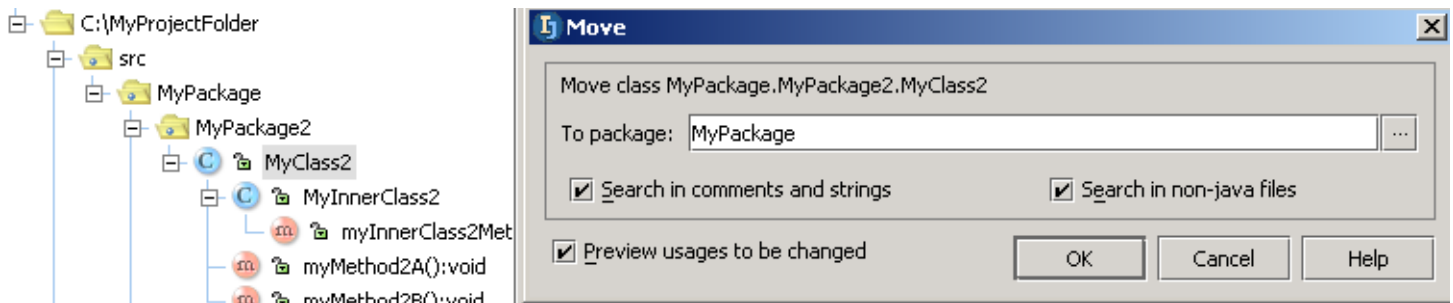


Figure 8.23. Move class dialog (377)

8.47. Click **OK**. The “Find - Refactoring preview” dialog appears.

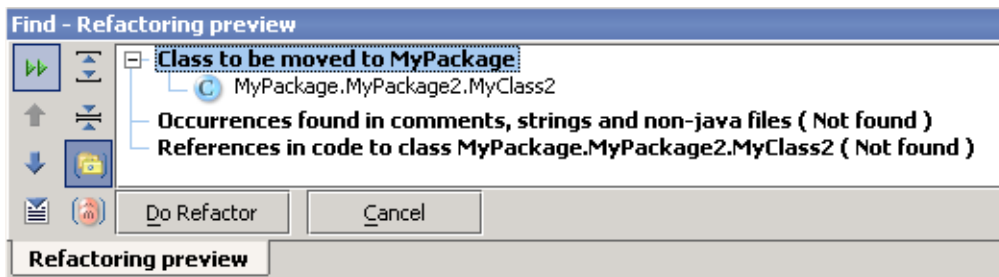


Figure 8.24. Find Refactoring preview (376)

8.48. Click **Do Refactor**. The class is moved.

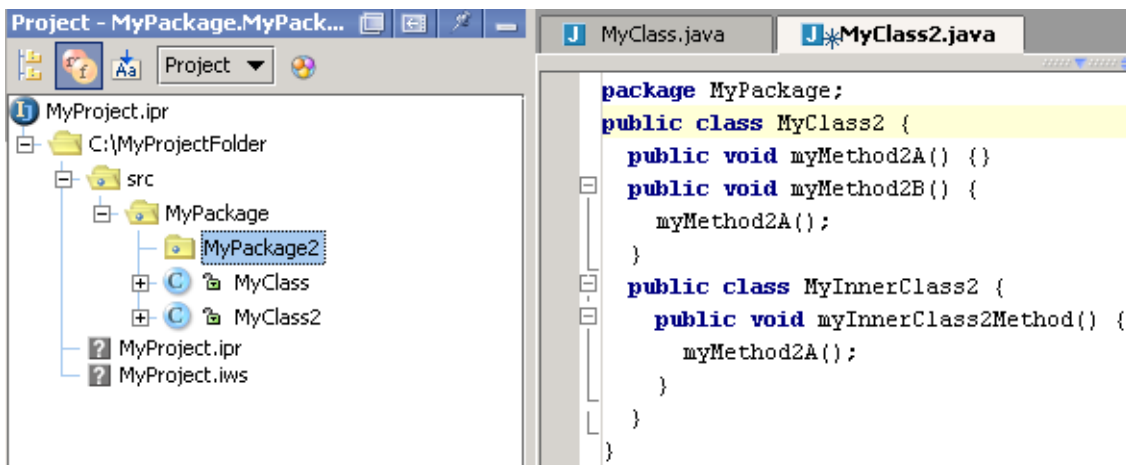


Figure 8.25. Class moved (375)

9.3.3. Members

- 8.49. Right-click on **myMethod2A**.
- 8.50. Selected **Refactor | Move**. The dialog “Move members” appears.
- 8.51. For “To” select **MyPackage.MyClass**.

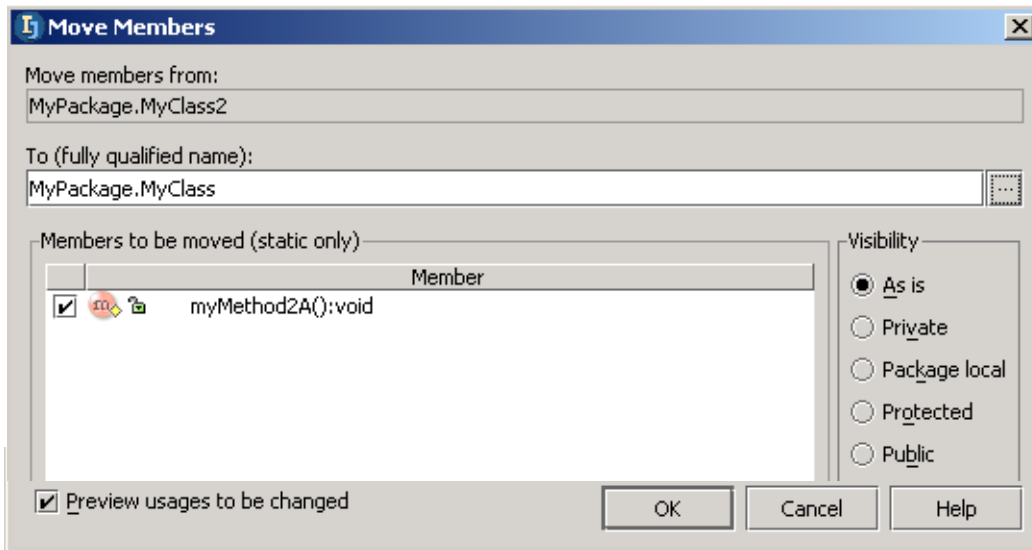


Figure 8.26. Dialog Move Members (374,921)

- 8.52. Click **OK**. The tool “Refactoring preview” appears.

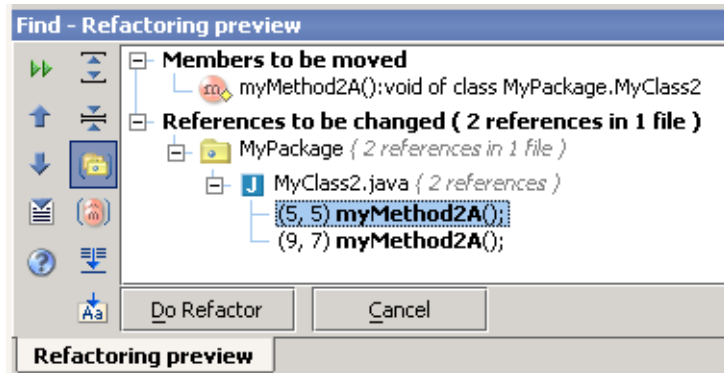


Figure 8.27. Find - Refactoring preview (373)

- 8.53. Click **Do Refactor**. The class is moved.

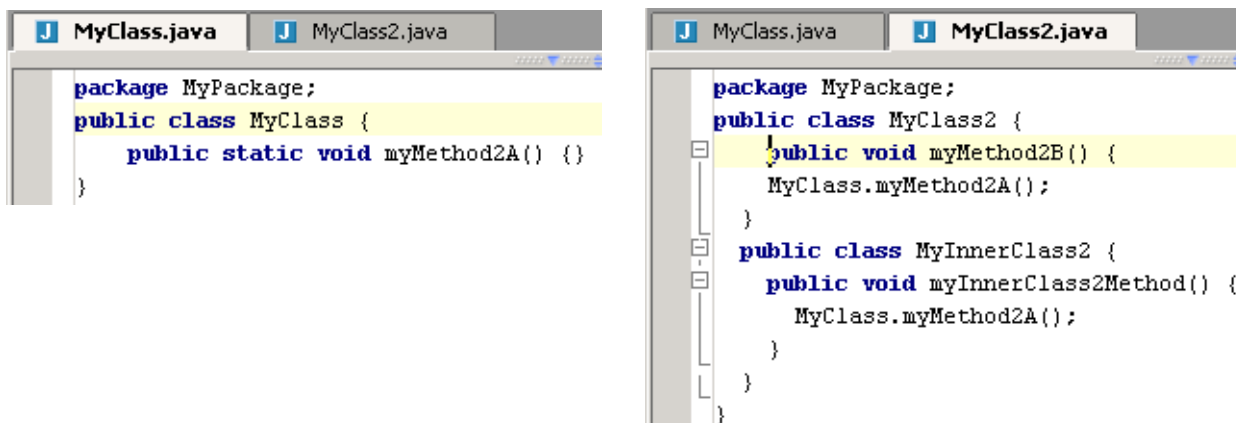


Figure 8.28. Moved class (372,371)

9.3.4. Inner to upper level

8.54. Place the cursor on **MyInnerClass2**.

8.55. Selected **Refactor | Move**. The dialog “Move inner to upper” appears.

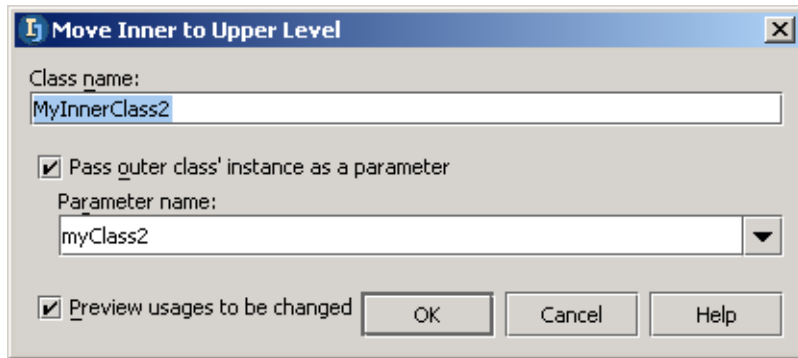


Figure 8.29. Move inner to upper dialog [\(370\)](#)

8.56. Click **OK**. The tool “Refactoring preview” appears.

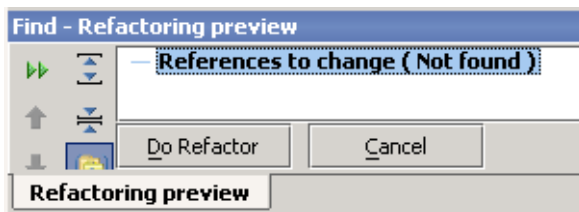


Figure 8.30. Find - Refactoring preview [\(369\)](#)

8.57. Click **Do Refactor**. The class is moved.

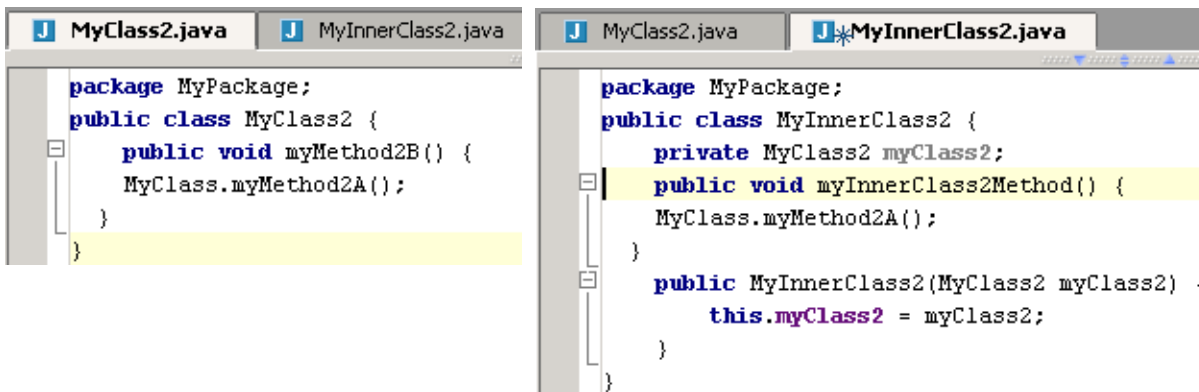


Figure 8.31. Moved inner class [\(368,367\)](#)

9.4. Change method signature

IDEA make it easy to change the signature of a method globally.
In this section you will

- [9.4.1. Add parameter \(page 216\)](#)
- [9.4.2. Move parameter \(page 218\)](#)
- [9.4.3. Change name \(page 219\)](#)
- [9.4.4. Change type \(page 219\)](#)

9.4.1. Add parameter

8.58. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public static void main(String[] args) {  
        Class1 cla = new Class1(1, "1");  
        Class1 clb = new Class1(2, "2");  
    }  
    public class Class1 {  
        private int aInt;  
        private String aString;  
        public Class1(int aIntPar, String aStringPar) {  
            this.aInt = aIntPar;  
            this.aString = aStringPar;  
        }  
    }  
}
```

8.59. Place the caret on the Class1 constructor in the line

```
Class1 cla = new Class1(1, "1");
```

8.60. Select **Refactor | Change method signature**. The dialog “Change method signature” appears.

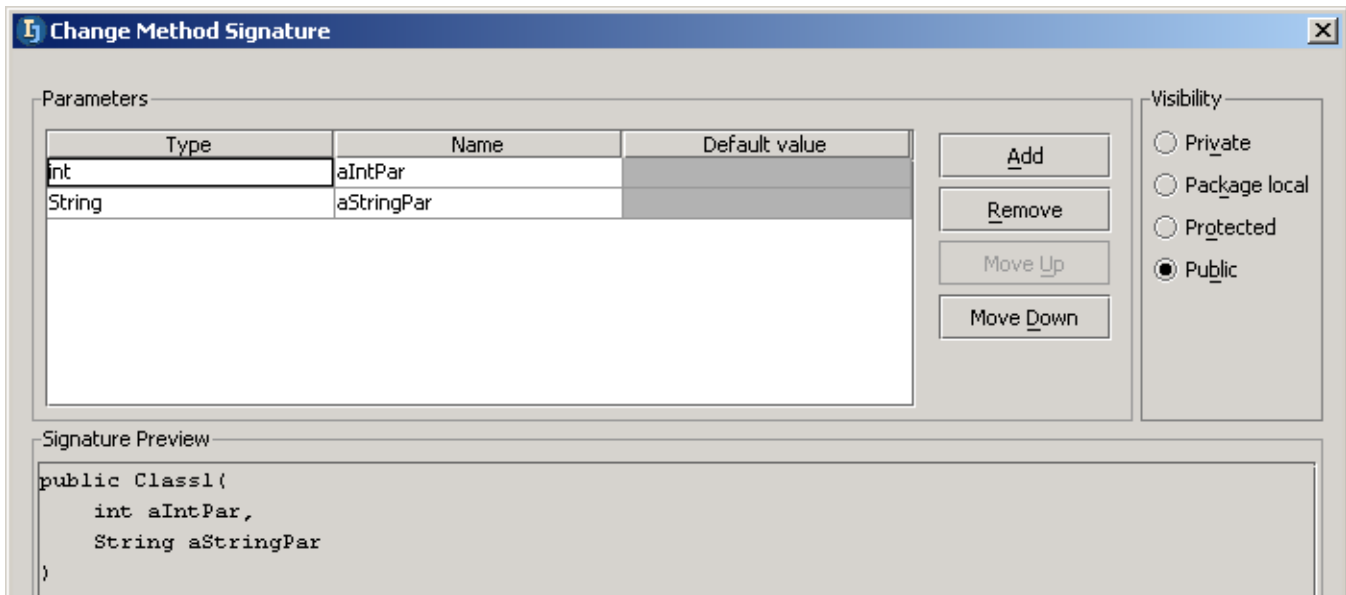


Figure 8.32. Dialog “Change method signature” (413)

8.61. Click **Add**. A new row appears at the end of the parameter list. The text box in row “Type” is selected.

8.62. For “Type” enter **boolean**.

- 8.63. For “Name” enter **aBoolean**.
- 8.64. For “Default value” enter **true**.

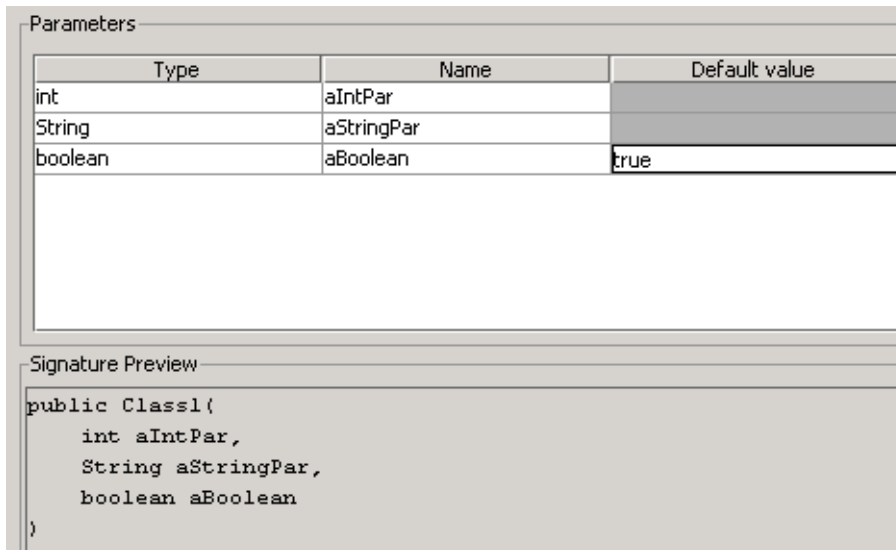


Figure 8.33. Add parameter (412)

- 8.65. Check checkbox **Preview usages to be changed**.
- 8.66. Click **OK**. The tool “Refactoring preview” appears.

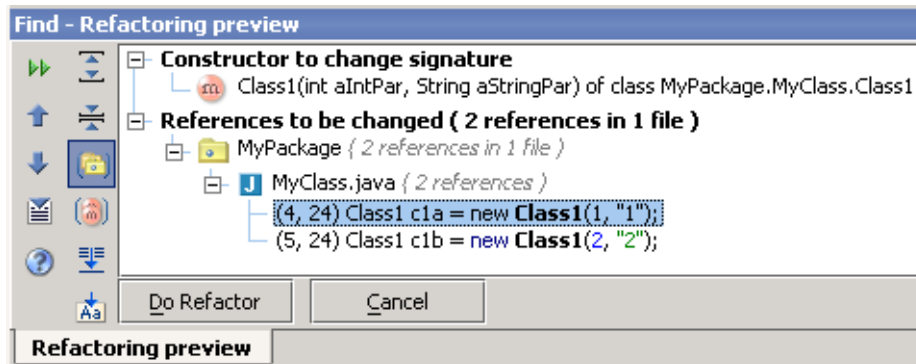


Figure 8.34. Find - Refactoring preview (411)

- 8.67. Click **Do Refactor**. The class is refactored.

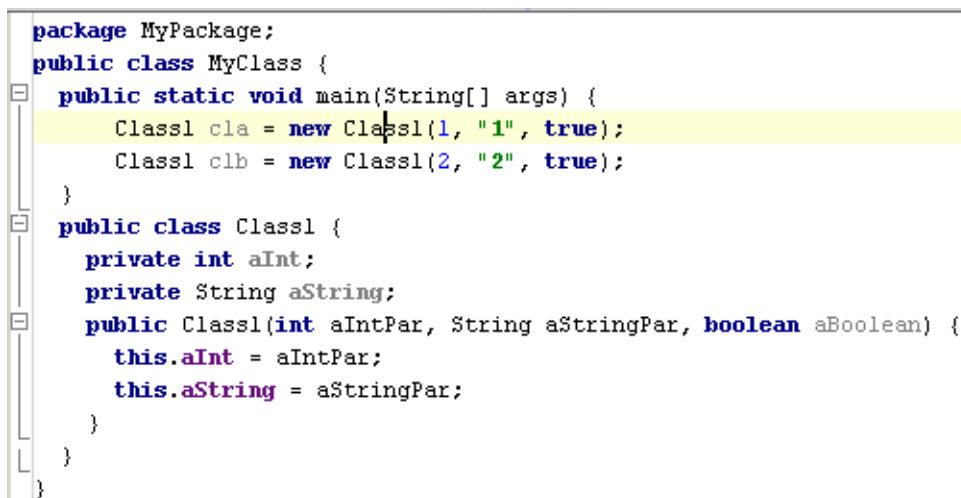


Figure 8.35. Parameter added (410)

9.4.2. Move parameter

8.68. Reopen “Change method signature” for Class1.

8.69. Select **aBoolean**.

8.70. Click **MoveUp** until the parameter is first.

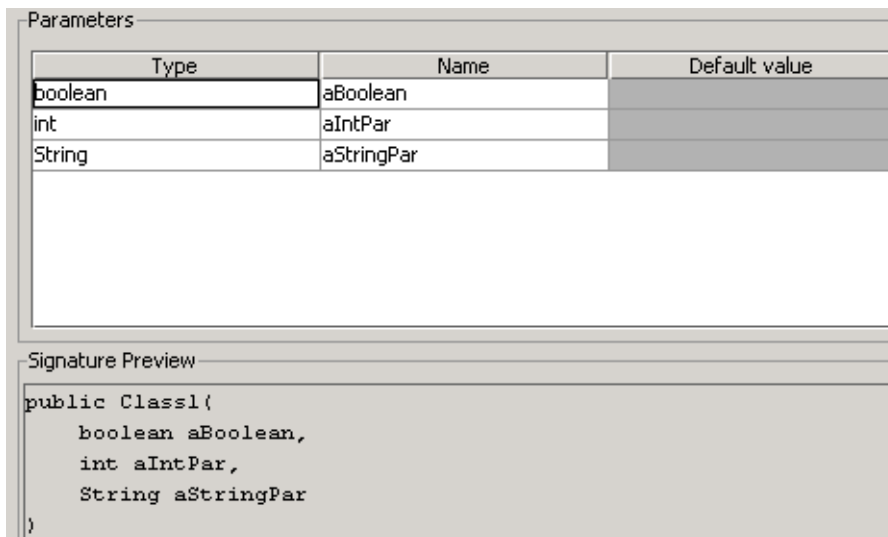


Figure 8.36. Parameter moved (409)

8.71. Click **OK**.

8.72. Click **Do Refactor**. The class is refactored.

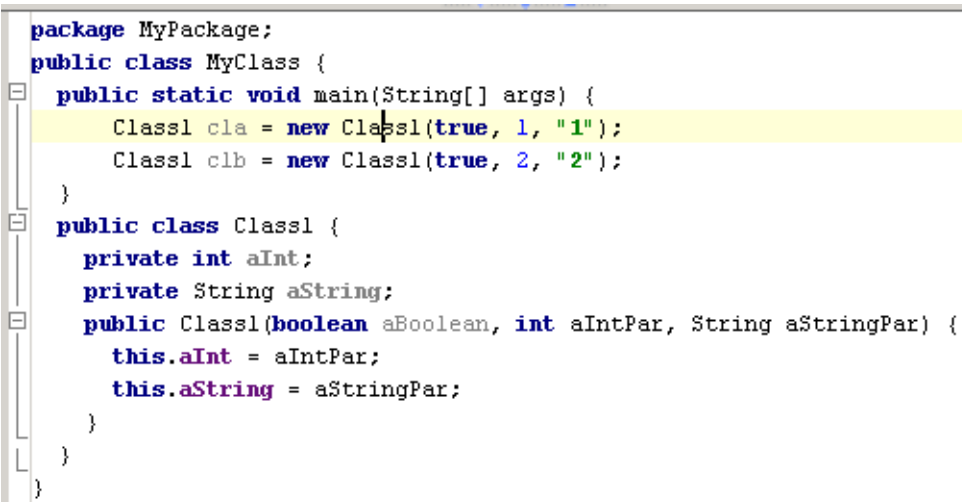


Figure 8.37. Parameter moved (922)

9.4.3. Change name

- 8.73. Select **aBoolean**.
- 8.74. Change to **aBooleanPar**.
- 8.75. Click **OK**.
- 8.76. Click **Do Refactor**. The class is refactored.

9.4.4. Change type

- 8.77. Reopen “Change method signature” for Class1.
- 8.78. Change type “boolean” to **char**.

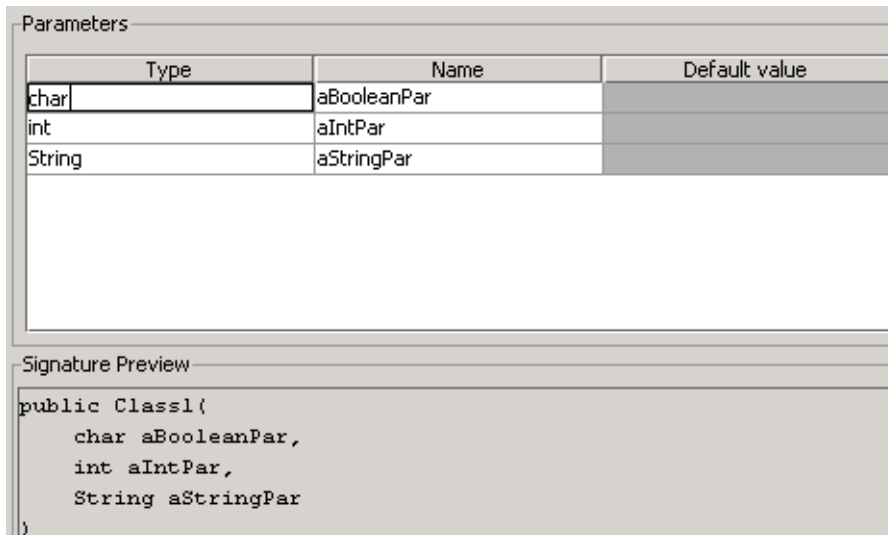


Figure 8.38. Parameter type change (408)

- 8.79. Click **OK**.
- 8.80. Click **Do Refactor**. The class is refactored.
- 8.81. Change

```

Class1 cla = new Class1(true, 1, "1");
Class1 clb = new Class1(true, 2, "2");
    
```

to

```

Class1 cla = new Class1('1', 1, "1");
Class1 clb = new Class1('2', 2, "2");
    
```

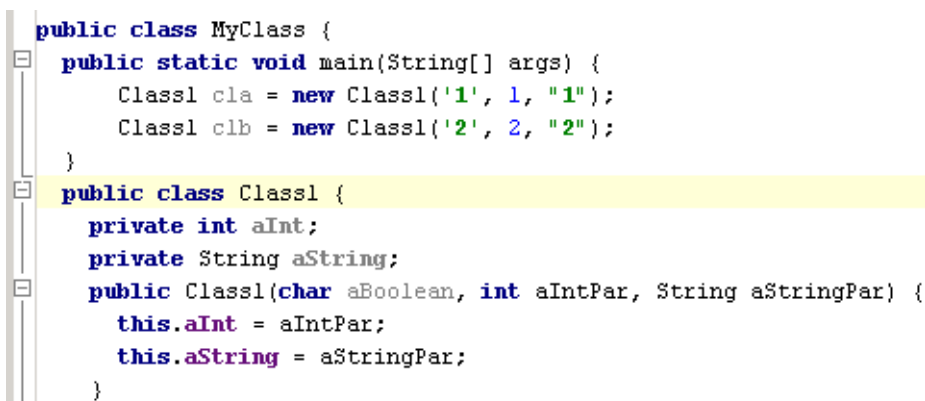


Figure 8.39. Parameter type changed (407)

9.5. Copy class

IDEA make it easy to copy a class.

8.82. Select **MyClass**.

8.83. Select **Refactor | Copy**. The dialog “Copy Class” appears.

8.84. For “Name” enter **MyClassCopy**.

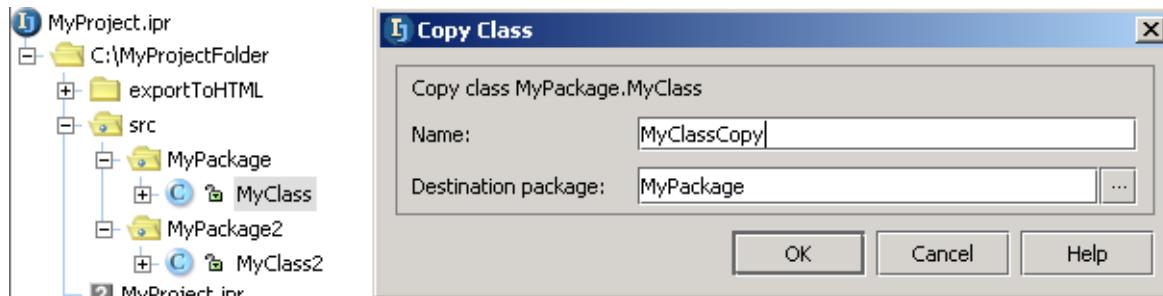


Figure 8.40. Dialog “Copy Class” (890)

8.85. Click **OK**. The class is copied.

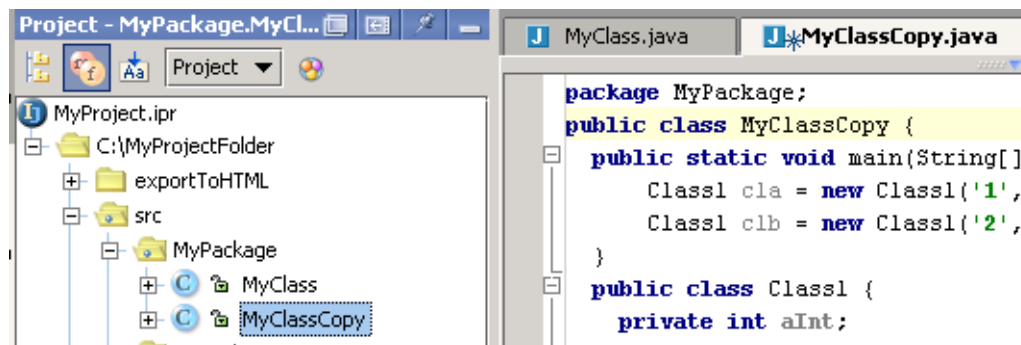


Figure 8.41. Copied class (891)

9.6. Extract

IDEA allows you to extract the following:

- [9.6.1. Method \(page 221\)](#)
- [9.6.2. Interface \(page 222\)](#)
- [9.6.3. Superclass \(page 224\)](#)

9.6.1. Method

8.86. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public void aMethod() {  
        System.out.println("Hello");  
    }  
}
```

8.87. Place the cursor on line

```
        System.out.println("Hello");
```

8.88. Selected **Refactor | Extract method**. The dialog “Extract method” appears.

8.89. For “Name” enter **extractedMethod**.

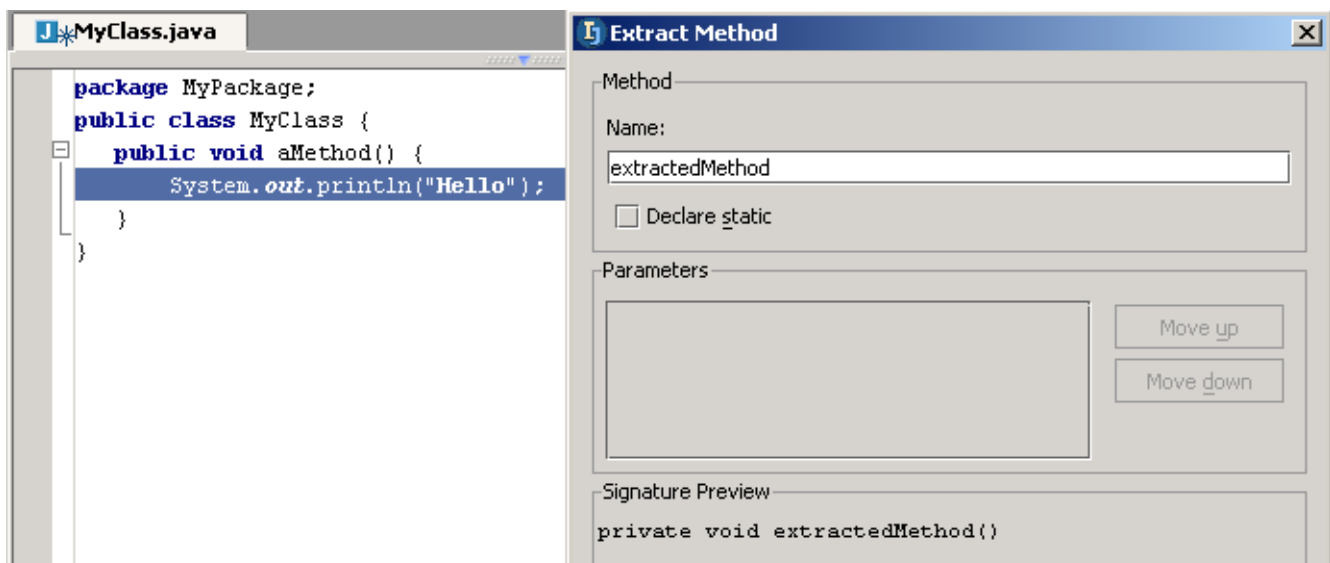


Figure 8.42. Method to extract [\(406\)](#)

8.90. Click **OK**. The method is extracted.

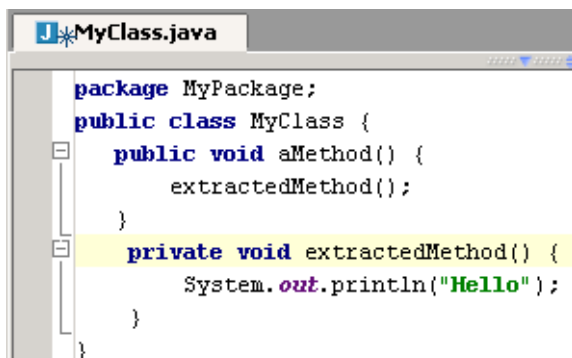


Figure 8.43. Extracted method [\(405\)](#)

9.6.2. Interface

8.91. Click on line

```
private void extractedMethod() {
```

8.92. Select **Refactor | Extract interface**. The “Extract Interface” dialog appears.

8.93. For “Interface name” enter **ExtractedInterface**.

8.94. Check the checkbox for **aMethod():void**.

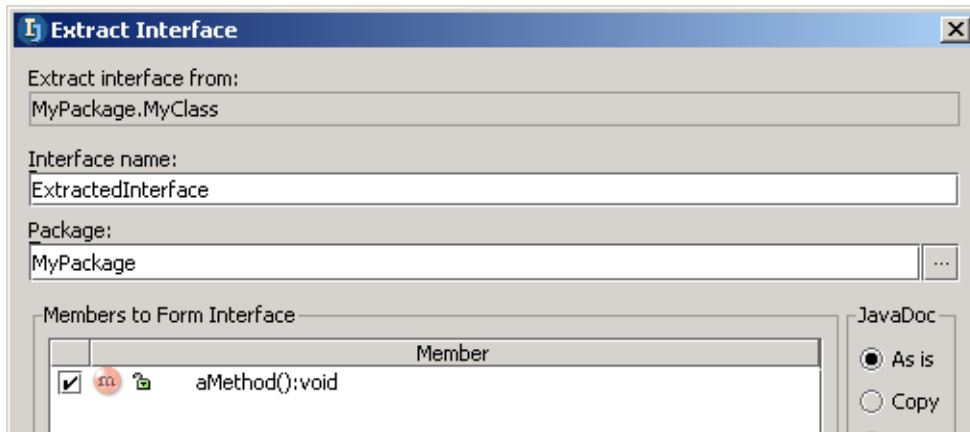


Figure 8.44. Extract interface (404)

8.95. Click **OK**. The dialog “Search usages” appears.

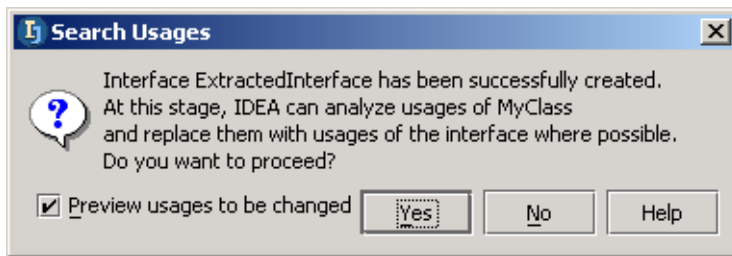


Figure 8.45. Search usages (924)

8.96. Click **Yes**.

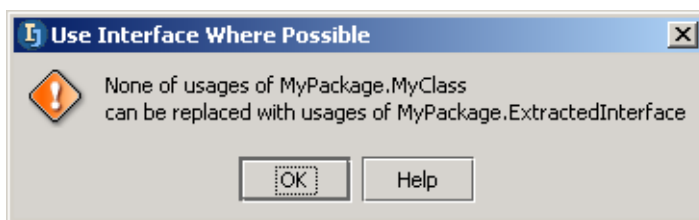


Figure 8.46. No usages can be replaced (925)

8.97. Click **OK**. The interface is extracted.

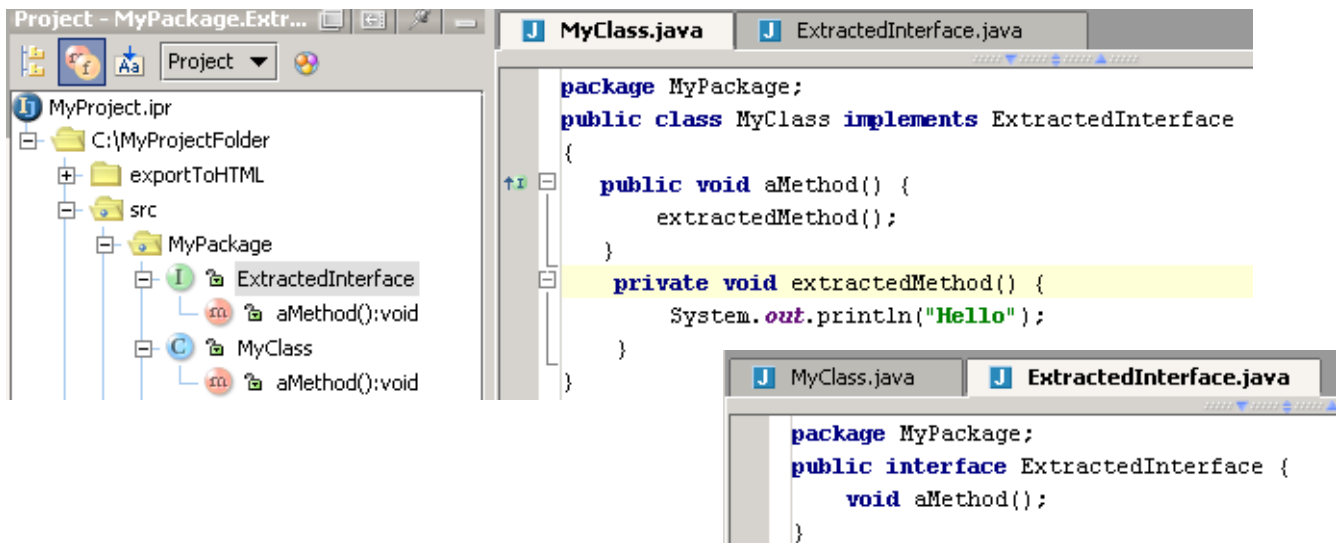


Figure 8.47. Extracted interface [\(401,400\)](#)

9.6.3. Superclass

8.98. Select **MyClass**.

8.99. Select **Refactor | Extract superclass**. The “Extract Superclass” dialog appears.

8.100. For “Superclass name” enter **ExtractedSuperClass**.

8.101. Check the checkbox in “Members to Form Superclass”.

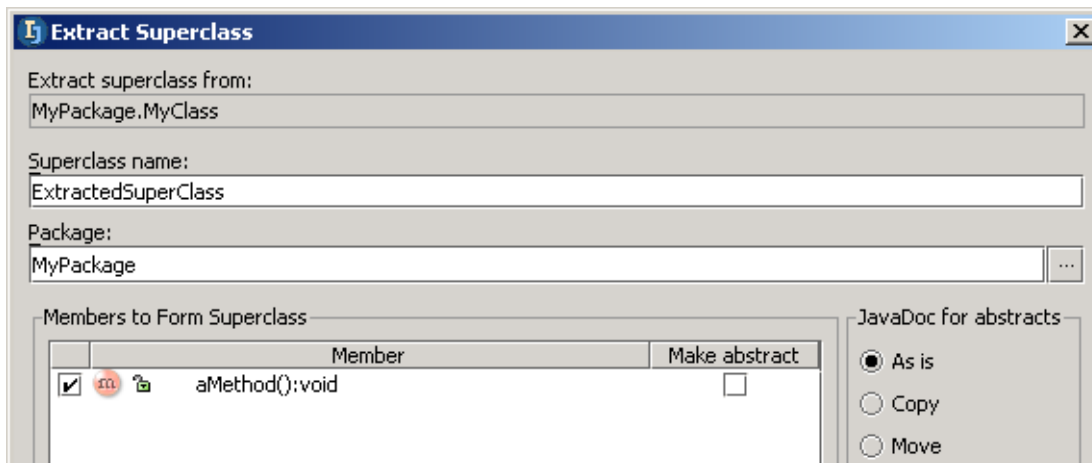


Figure 8.48. Extract superclass (399)

8.102. Click **OK**. A question dialog appears.

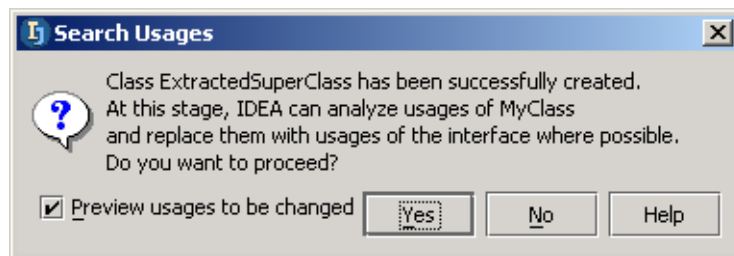


Figure 8.49. Proceed question (398)

8.103. Click **Yes**.

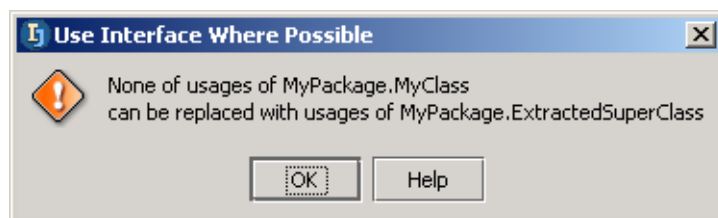


Figure 8.50. User interface where possible (926)

8.104. Click **OK**. The superclass is extracted.

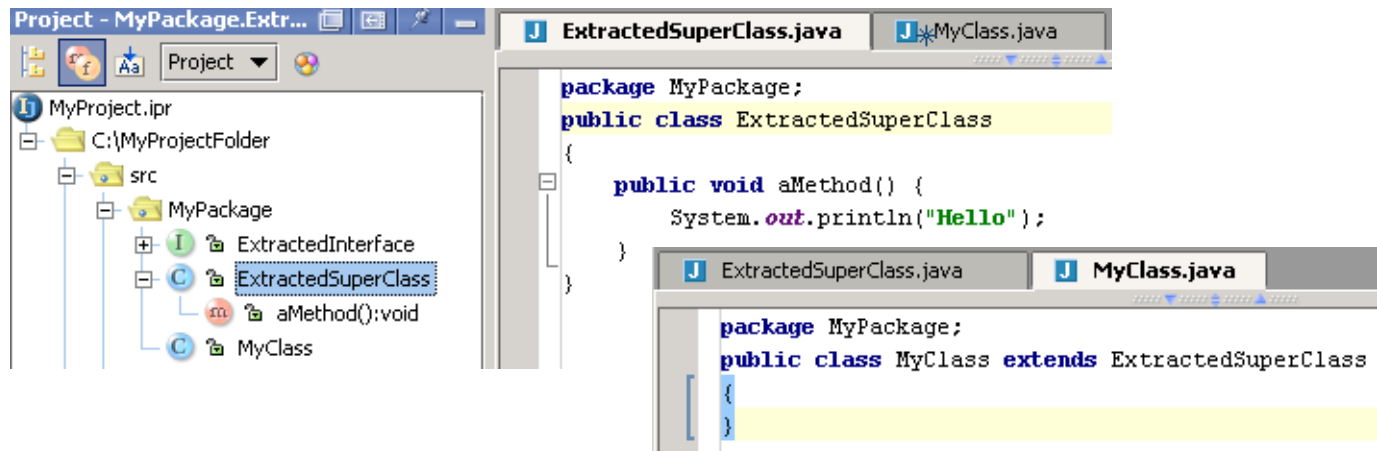


Figure 8.51. Extracted superclass [\(397,396\)](#)

9.7. Use interface where possible

IDEA can replace with an interface if possible.

8.105. Create interface **MyInterface**:

```
package MyPackage;  
public interface MyInterface {  
    public void iMethod();  
}
```

8.106. Create class **MyClass**:

```
package MyPackage;  
public class MyClass implements MyInterface {  
    public void iMethod() {}  
}
```

8.107. Create class **MyClass2**:

```
package MyPackage;  
public class MyClass2 extends MyClass {  
    public void test (MyClass myClass) {  
        myClass.iMethod();  
    }  
}
```

8.108. In the editor for **MyClass**: Select **Refactor | Use interfaces where possible....** The dialog “Use Interface Where Possible” appears.

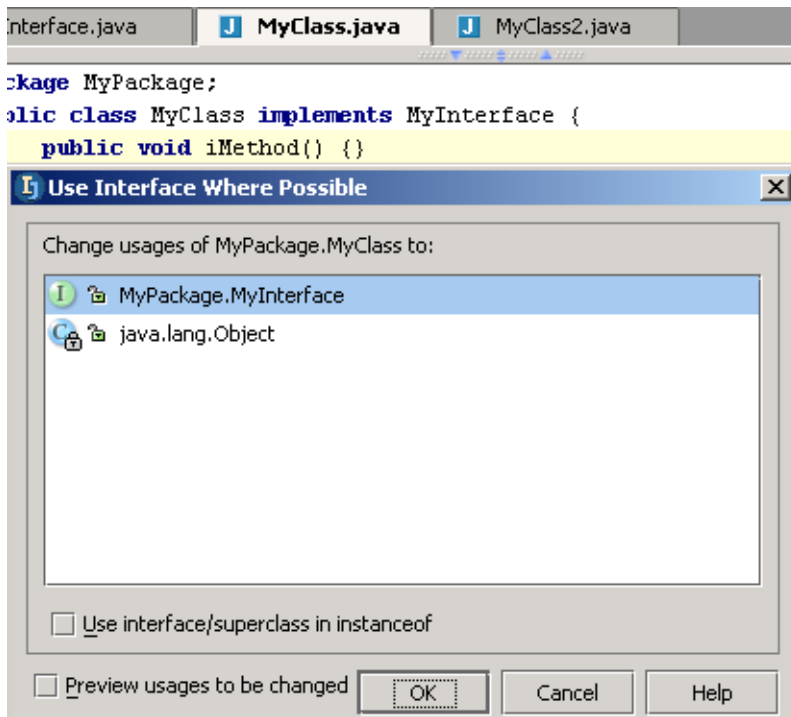


Figure 8.52. User interfaces where possible (892)

8.109. Select **MyPackage.MyInterface**.

8.110. Check **Preview usages to be changed**.

8.111. Click **OK**. The refactoring preview appears.

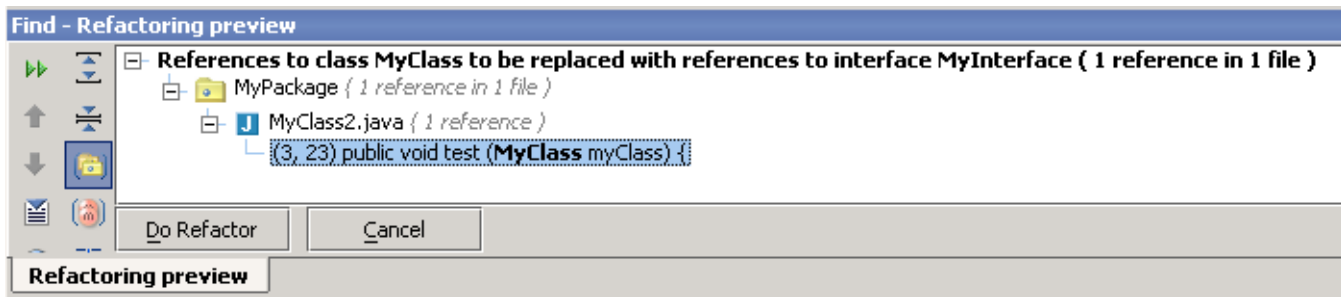


Figure 8.53. Refactoring preview (893)

8.112. Click **Do Refactor**. MyClass2 is refactored.

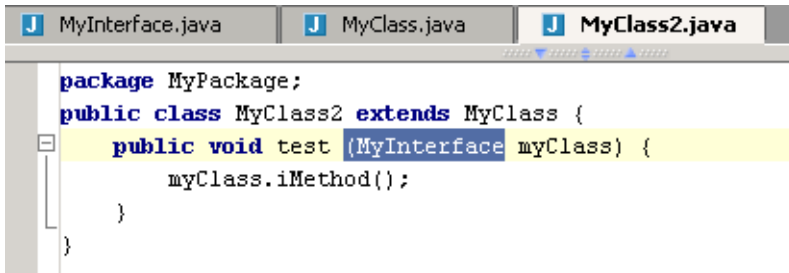


Figure 8.54. Refactored class (894)

9.8. Pull/push members

IDEA can

- 9.8.1. Pull Up (page 228)
- 9.8.2. Push Down (page 229)

class members.

9.8.1. Pull Up

8.113. Create class **MyClass**:

```
package MyPackage;  
public class MyClass implements MyInterface {  
    public void iMethod() {};  
    public void iMethod2() {};  
}
```

8.114. Create interface **MyInterface**:

```
package MyPackage;  
public interface MyInterface {  
}
```

8.115. In the MyClass editor: Select **Refactor | Pull members up...**

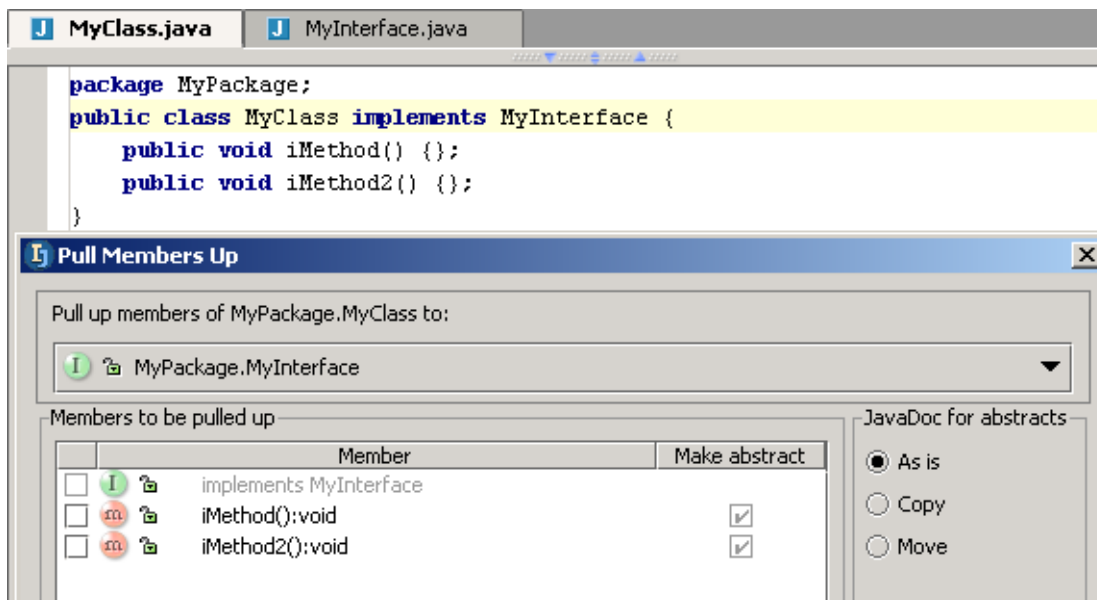


Figure 8.55. Dialog “Pull members up” (895)

8.116. Check **iMethod():void**.

8.117. Check **iMethod2():void**.

8.118. Click **OK**. The methods are “pulled up” to MyInterface.

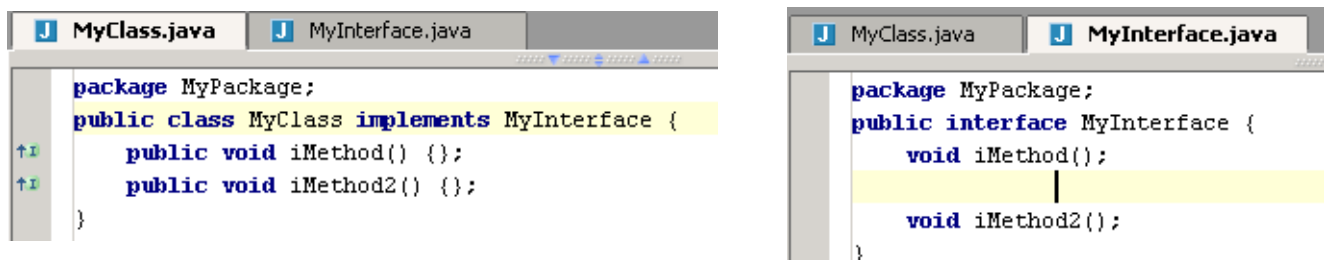


Figure 8.56. Members pulled up (896,897)

9.8.2. Push Down

8.119. Delete the methods in MyClass:

```
package MyPackage;
public class MyClass implements MyInterface {}
```

8.120. In the MyInterface editor: Select **Refactor | Push members down....**

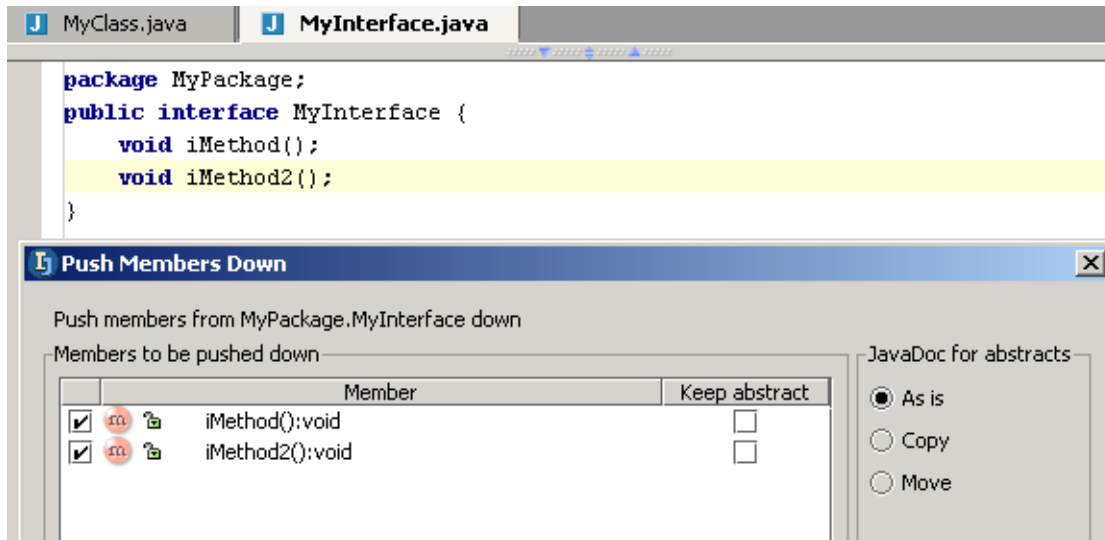


Figure 8.57. Dialog “Push members down” (898)

8.121. Check **iMethod():void**.

8.122. Check **iMethod2():void**.

8.123. Check **Preview usages to be changed**.

8.124. Click **OK**. A Refactoring preview appears.

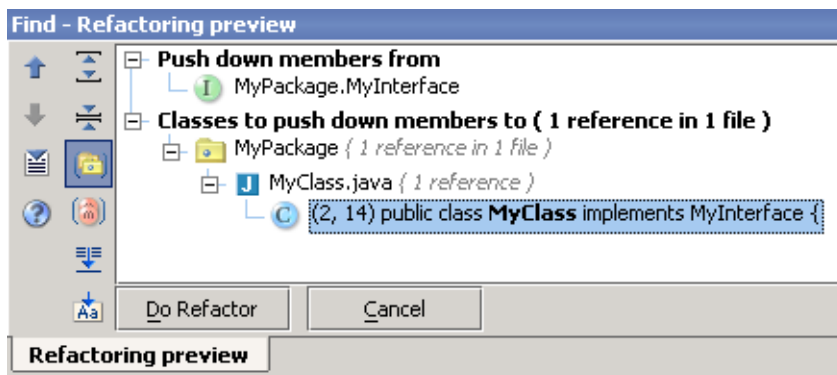


Figure 8.58. Push members down refactoring preview (899)

8.125. Click **OK**. The members are pushed down to MyClass.

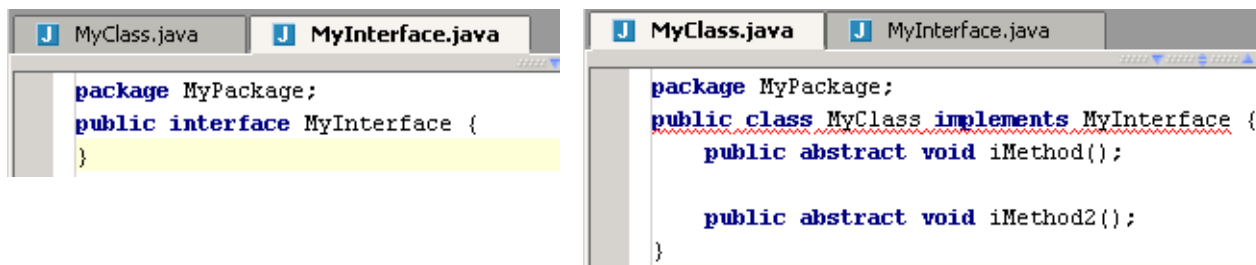


Figure 8.59. Members pushed down (900,901)

9.9. Introduce

IDEA can introduce the following:

- [9.9.1. Variable \(page 230\)](#)
- [9.9.2. Field \(page 231\)](#)
- [9.9.3. Constant \(page 232\)](#)
- [9.9.4. Parameter \(page 233\)](#)

9.9.1. Variable

8.126. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
    }  
}
```

8.127. Select **"System.out"**.

8.128. Select **Refactor | introduce variable**. The dialog "Introduce variable" appears.

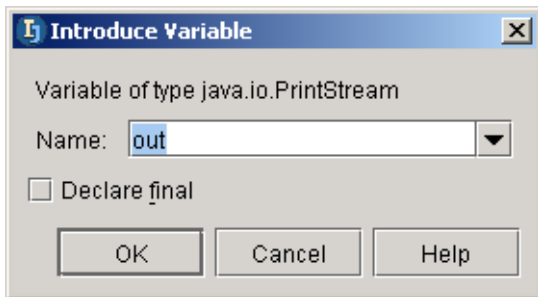


Figure 8.60. Introduce variable [\(382\)](#)

8.129. Click **OK**. The variable is introduced.

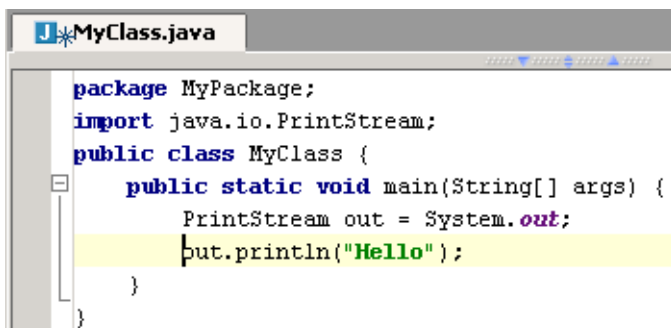


Figure 8.61. Variable introduced [\(381\)](#)

9.9.2. Field

The following example shows how to convert a local variable to a field.

8.130. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public void aMethod() {  
        String aString = new String("Hello");  
    }  
}
```

8.131. Place the cursor on **aString**.

8.132. Select **Refactor | introduce field**.

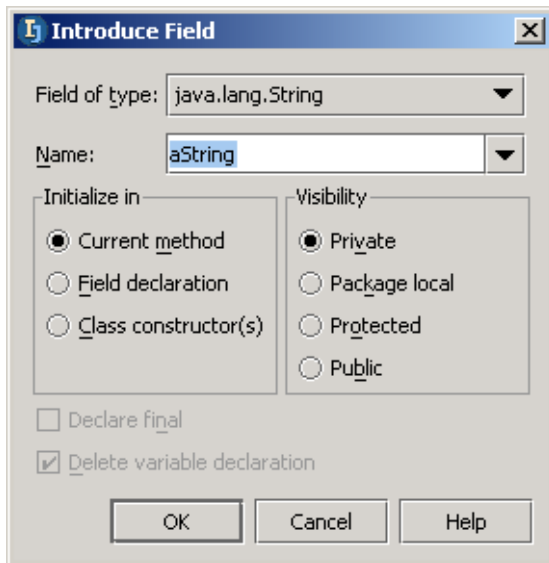


Figure 8.62. Introduce field (384)

8.133. Click **OK**. The field is introduced.

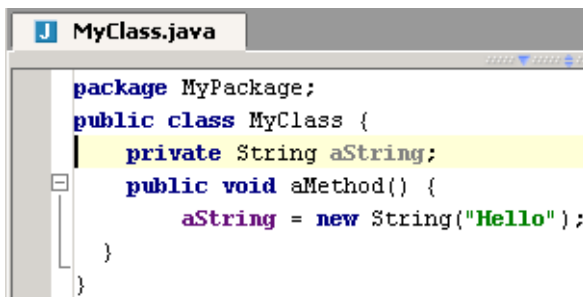


Figure 8.63. Field introduced (383)

9.9.3. Constant

The following example shows how to convert to a constant.

8.134. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public int aMethod() {  
        int aInt = 1;  
        return aInt;  
    }  
}
```

8.135. Place the cursor on **aInt**.

8.136. Select **Refactor | Introduce constant**.

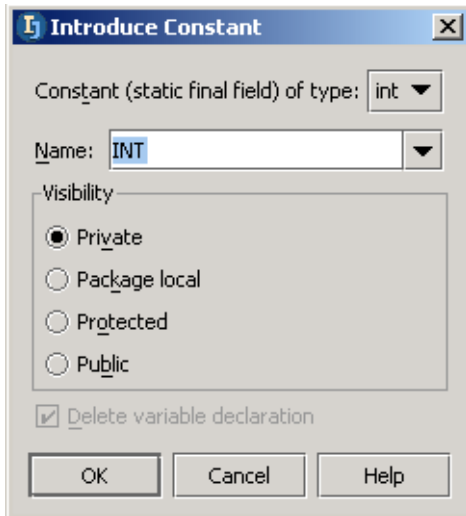


Figure 8.64. Introduce constant (927???)

8.137. Click **OK**. The constant is introduced.

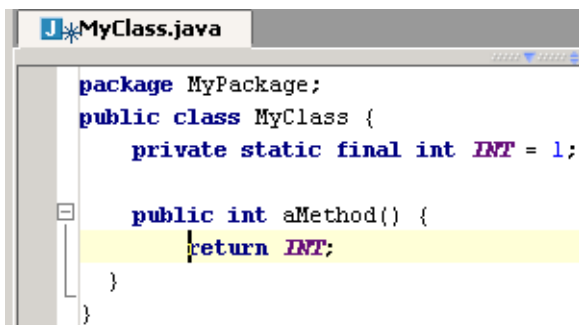


Figure 8.65. Constant introduced (928)

9.9.4. Parameter

The following example shows how to convert to a parameter.

8.138. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public void aMethod() {  
        int aInt = 1;  
    }  
}
```

8.139. Place the cursor on **aInt**.

8.140. Select **Refactor | Introduce parameter**.

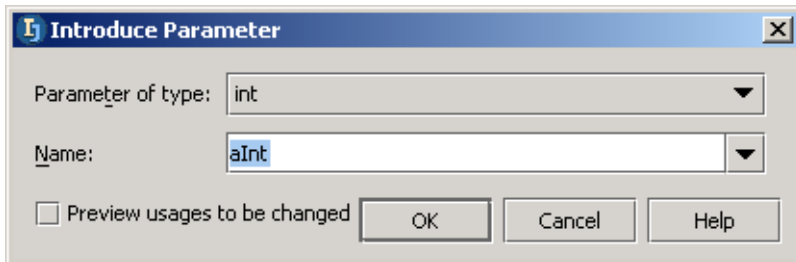


Figure 8.66. Introduce parameter [\(929\)](#)

8.141. Click **OK**. The parameter is introduced.

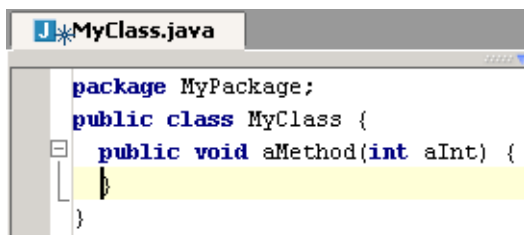


Figure 8.67. Parameter introduced [\(930\)](#)

9.10. Inline

IDEA can inline a

- [9.10.1. Variable \(page 234\)](#)
- [9.10.2. Method \(page 235\)](#)

9.10.1. Variable

8.142. Create class **MyPackage**:

```
package MyPackage;
public class MyClass {
    public int aMethod() {
        int aVar = 1;
        return aVar;
    }
}
```

8.143. Place the cursor on “aVar”.

8.144. Selected **Refactor | Inline**. A message appears “Inline local variable aVar?”.

8.145. Click **OK**. The variable is inlined.

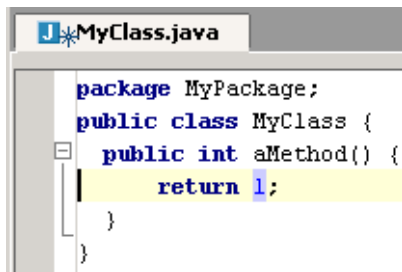


Figure 8.68. Inlined variable [\(395\)](#)

9.10.2. Method

8.146. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public int methodToInline(int param) {  
        return aMethod() + param;  
    }  
    public int aMethod() {  
        return 1;  
    }  
}
```

8.147. Create class **MyClass2**:

```
package MyPackage;  
public class MyClass2 {  
    void aMethod (MyClass mc) {  
        int res = mc.methodToInline(1);  
    }  
}
```

8.148. Place the cursor on “methodToInline”.

8.149. Selected **Refactor | Inline**. A message appears.

8.150. Select **All invocations and remove the method**.

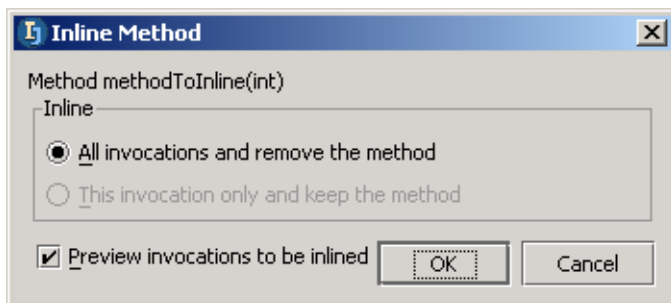


Figure 8.69. Inline method (394)

8.151. Click **OK**. A refactoring preview appears.

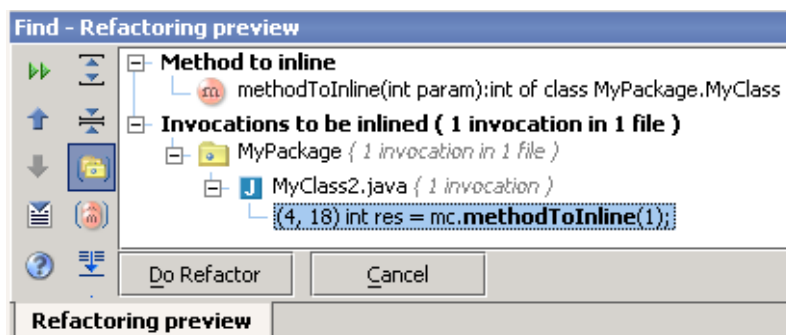
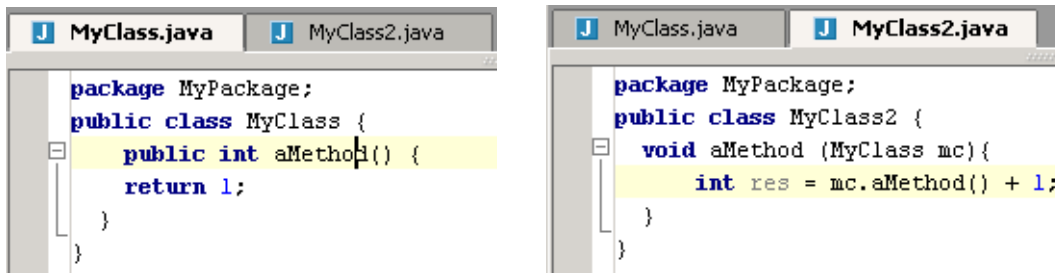


Figure 8.70. Inline method preview (931)

8.152. Click **Do Refactor**. The method is inlined.



```
package MyPackage;
public class MyClass {
    public int aMethod() {
        return 1;
    }
}

package MyPackage;
public class MyClass2 {
    void aMethod (MyClass mc){
        int res = mc.aMethod() + 1;
    }
}
```

Figure 8.71. Inlined method [\(393.392\)](#)

9.11. Encapsulate field

8.153. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public int field;  
    public void method1(){  
        field = 1;  
    }  
    public int method2(int param){  
        return field + param;  
    }  
}
```

8.154. Create class **MyClass2**:

```
package MyPackage;  
public class MyClass2 {  
    private MyClass mc;  
    public int method3(int param){  
        return mc.field + param;  
    }  
}
```

8.155. Place the cursor in the class **MyClass**.

8.156. Selected **Refactor | Encapsulate fields....** The dialog “Encapsulate fields” appears.

8.157. Check the checkbox for **field:int**.

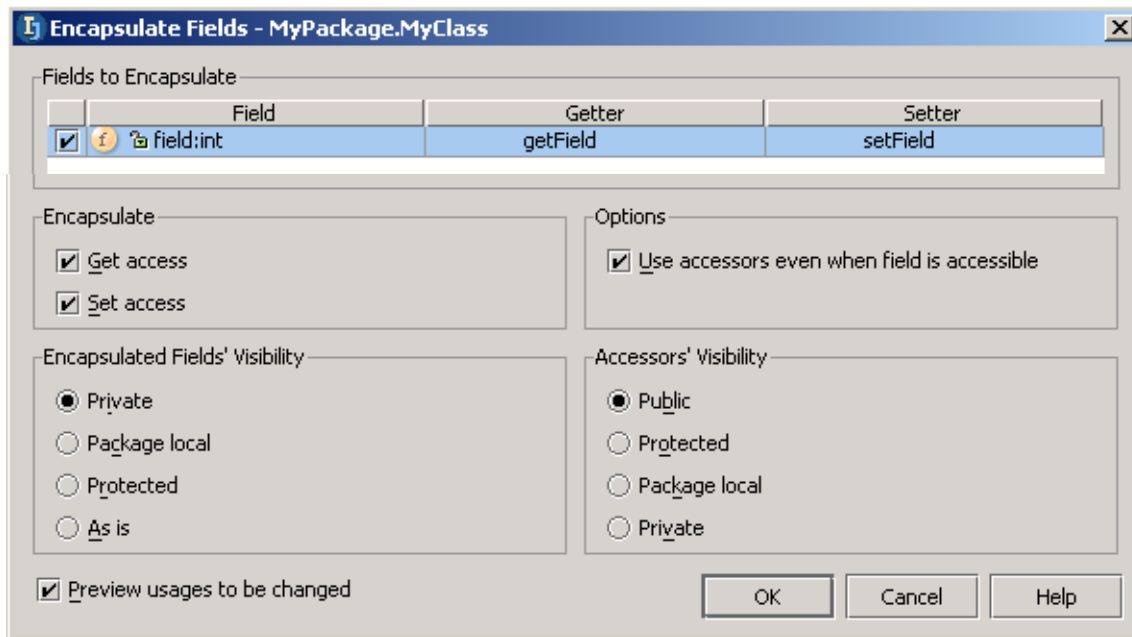


Figure 8.72. Dialog “Encapsulate fields” (391,390)

8.158. Click **OK**. A refactoring preview appears.

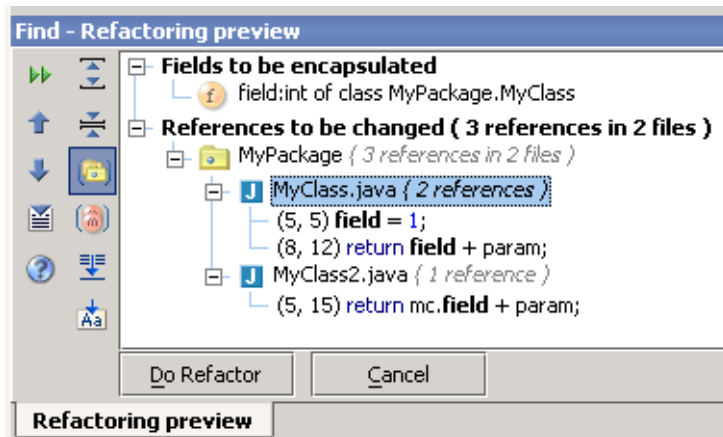


Figure 8.73. Encapsulate field refactoring preview (389)

8.159. Click **Do Refactor**. The field is encapsulated.

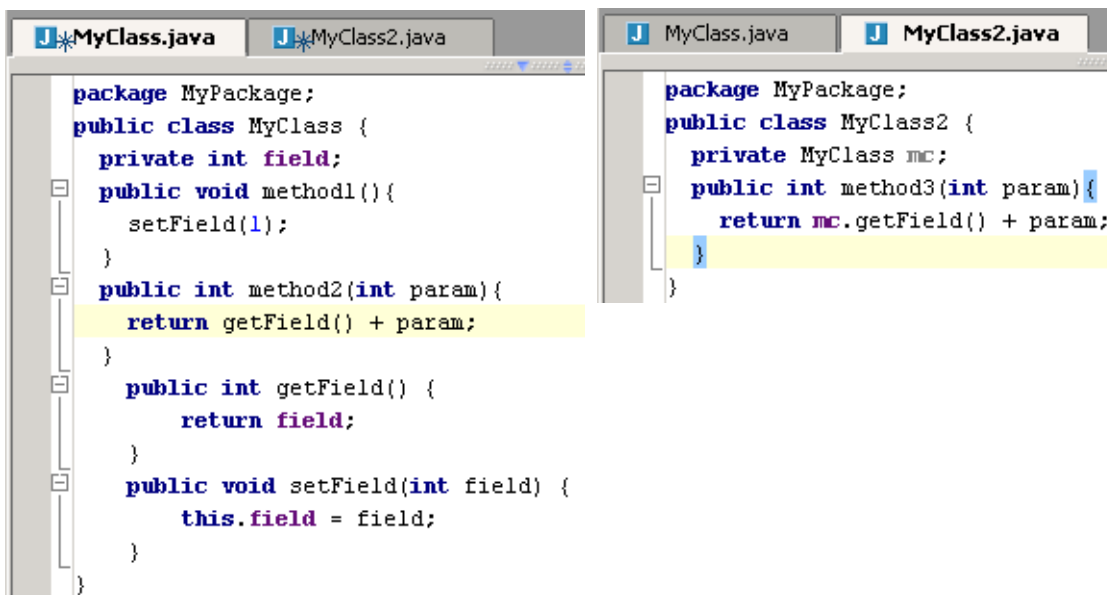


Figure 8.74. Encapsulated fields (388,387)

9.12. Replace temp with query

8.160. Create class **MyClass**:

```
package MyPackage;
public class MyClass {
    void aMethod() {
        int temp1 = query();
        int temp2 = query();
        temp2 = temp1;
    }
    int query()
    {
        return 1;
    }
}
```

8.161. Place the caret on **temp1**.

8.162. Click **Refactor | Replace temp with query....** The dialog “Replace temp with query” appears.

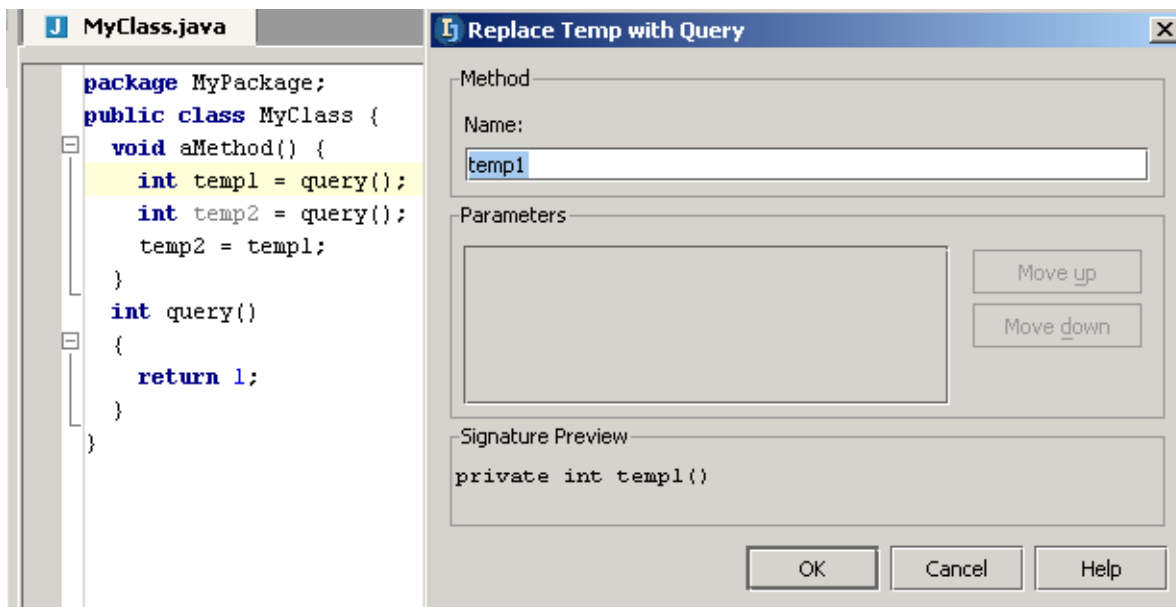


Figure 8.75. Replace temp with query (902)

8.163. Click **OK**. The temporary variable is replaced with a query.

```
package MyPackage;
public class MyClass {
    void aMethod() {
        int temp2 = query();
        temp2 = temp1();
    }
    private int temp1() {
        return query();
    }
    int query()
    {
        return 1;
    }
}
```

Figure 8.76. Temp replaced with query (903)

9.13. Convert Anonymous to Inner

8.164. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public MyInterface aMethod() {  
        final int aInt = 1;  
        return new MyInterface() {  
            public int aFunction() {  
                return aInt;  
            }  
        };  
    }  
}
```

8.165. Create class **MyInterface**:

```
package MyPackage;  
public interface MyInterface {  
    int aFunction();  
}
```

8.166. Place the cursor inside

```
        return new MyInterface() {  
            public int aFunction() {  
                return aInt;  
            }  
        };
```

8.167. Selected **Refactor | Convert anonymous to inner....** The dialog “Convert Anonymous to Inner” appears.

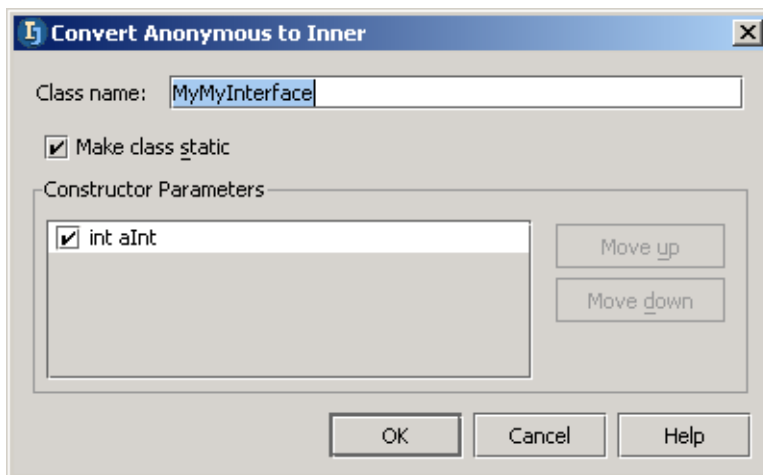
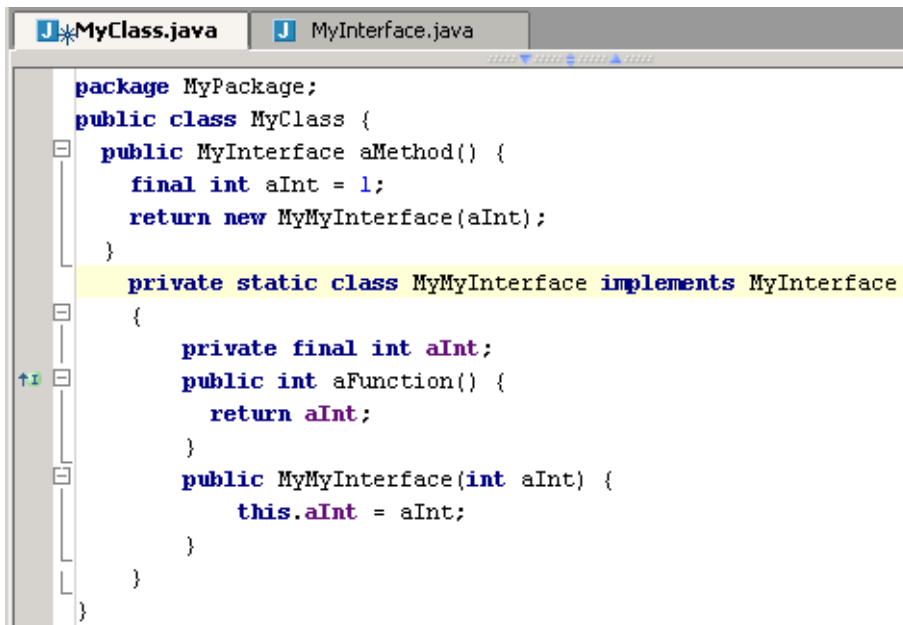


Figure 8.77. Dialog “Convert anonymous to inner” (932)

8.168. Click **OK**. The anonymous class is converted to an inner class.



```
package MyPackage;
public class MyClass {
    public MyInterface aMethod() {
        final int aInt = 1;
        return new MyMyInterface(aInt);
    }
    private static class MyMyInterface implements MyInterface
    {
        private final int aInt;
        public int aFunction() {
            return aInt;
        }
        public MyMyInterface(int aInt) {
            this.aInt = aInt;
        }
    }
}
```

Figure 8.78. Anonymous class converted to inner [\(385\)](#)

10. Code Inspection X

contacts: max

- 10.1. Run inspection X (page 243)
- 10.2. Resolve problems X (page 246)
- 10.3. Supported inspections XXX (page 250)
- 10.4. Entry points XXX (page 252)
- 10.5. Export to html XXX (page 253)
- 10.6. Offline inspection results XXX (page 254)

10.1. Run inspection X

- 10.1.1. File X (page 243)
- 10.1.2. Project X (page 245)
- 10.1.3. Rerun X (page 245)

10.1.1. File X

9.1. Create file:

```
package MyPackage;  
public class MyClass {  
    void aMethod() {  
        int temp2 = query();  
        temp2 = temp1();  
    }  
    private int temp1() {  
        return query();  
    }  
    int query() {  
        return 1;  
    }  
}
```

9.2. Select **Tools | Inspect code...**. The dialog “Choose Inspection Scope” appears.

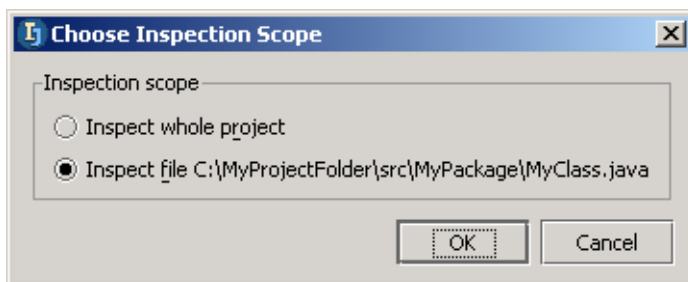


Figure 9.1. Choose inspection scope (904)

9.3. Select **Inspect file**.

9.4. Click **OK**. The dialog “Inspect code in file” appears.

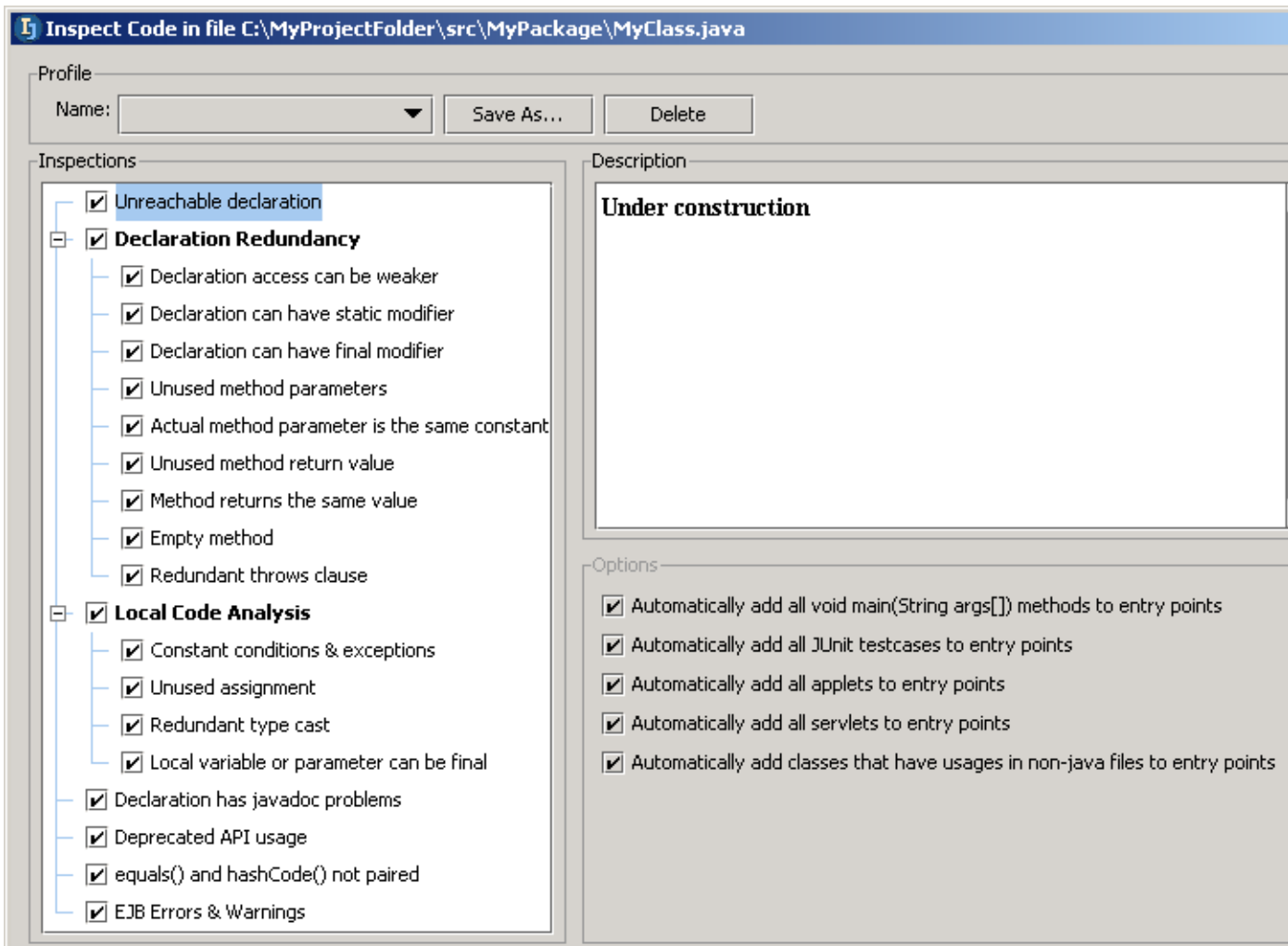


Figure 9.2. Dialog “Inspect code in file” (905)
9.5. Click **Run**. The inspection tool appears.

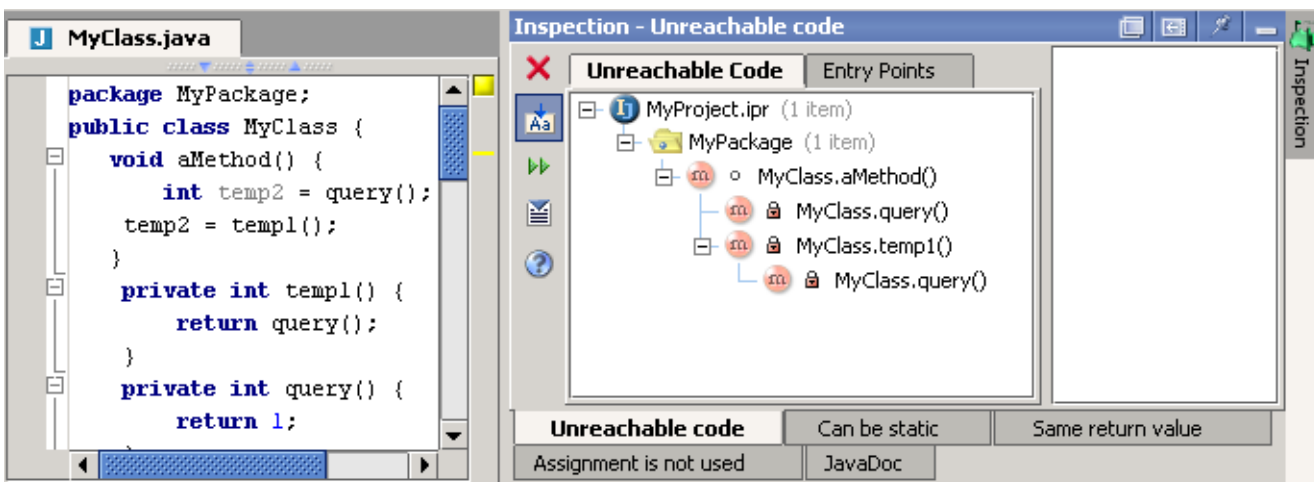


Figure 9.3. Inspection tool (906)

Inspection results

- Unreachable code
- Can be static

- Same return value
- Assignment is not used
- Javadoc

10.1.2. Project X

- 9.6. Close the inspection tool.
- 9.7. Select **Tools | Inspect code...**. The dialog “Choose Inspection Scope” appears.
- 9.8. Select **Inspect whole project**.
- 9.9. Click **OK**. The dialog “Inspect code in project” appears.
- 9.10. Click **Run**. The inspection tool appears.

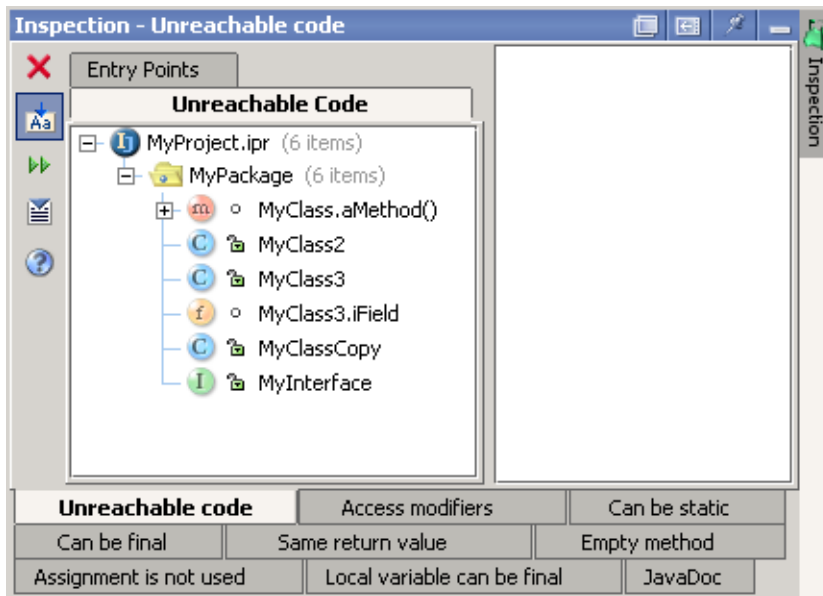


Figure 9.4. Inspection tool (project) (907)

Inspection results

- Unreachable code
- Access modifiers
- Can be static
- Can be final
- Same return value
- Empty method
- Assignment is not used
- Local variable can be final
- Javadoc

10.1.3. Rerun X

- 9.11. Click on the rerun icon () to rerun the inspection.

10.2. Resolve problems X

- 10.2.1. Implement suggested resolution X (page 246)
- 10.2.2. Manual X (page 249)

10.2.1. Implement suggested resolution X

typical suggested solutions

- 10.2.1.1. Modify X (page 246)
- 10.2.1.2. Comment out X (page 247)
- 10.2.1.3. Safe delete X (page 248)

10.2.1.1. Modify X

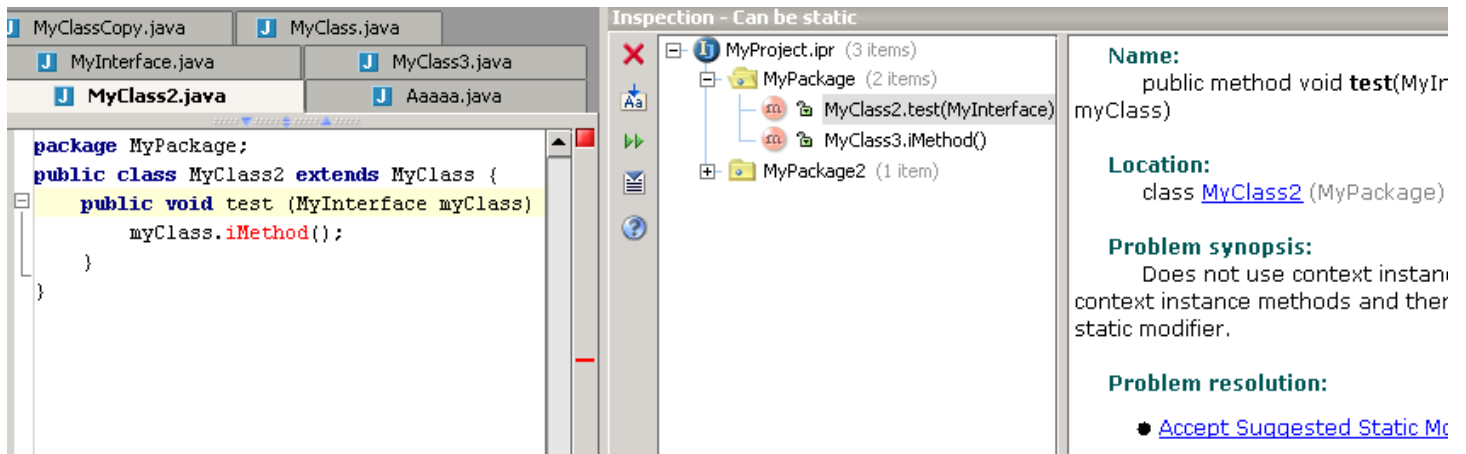


Figure 9.5. Modification suggested (915)

9.12. Click on **Accept suggested static modifier.**

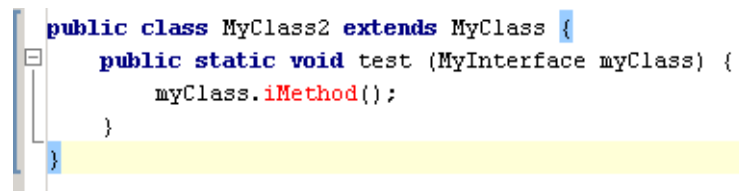


Figure 9.6. Modified (916)

10.2.1.2. Comment out X

9.13.

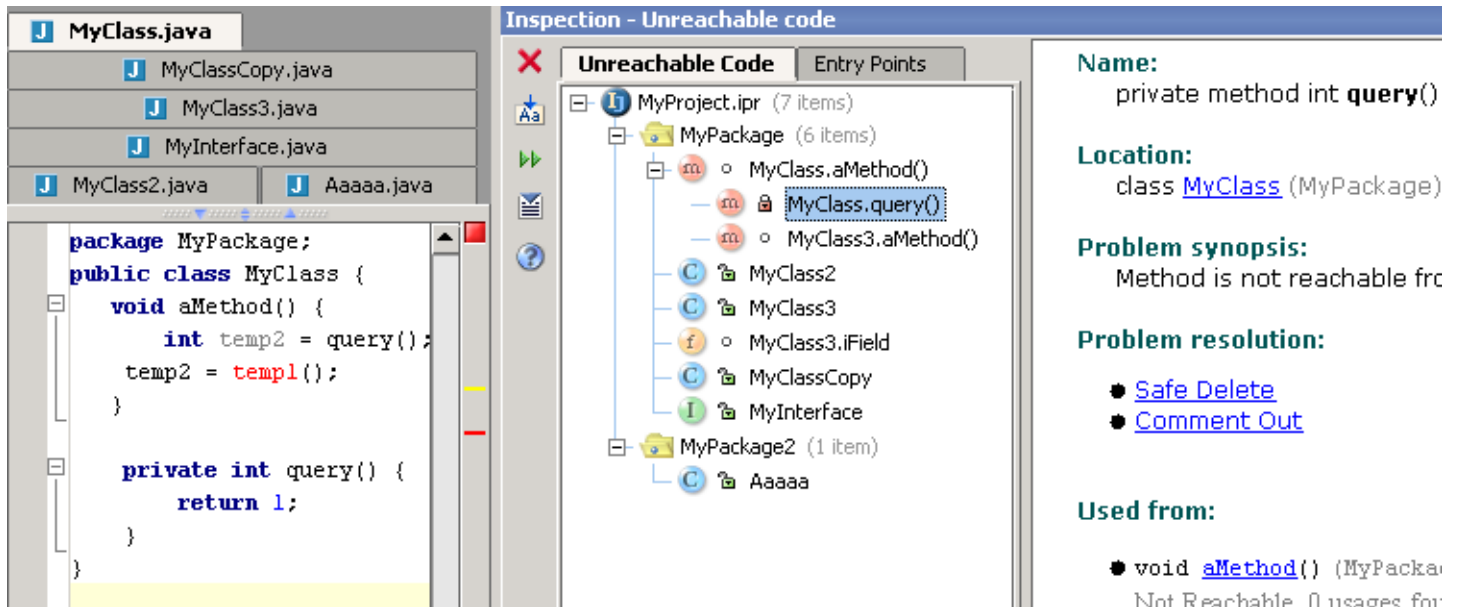


Figure 9.7. Safe delete (913)
9.14. Click on **Comment out**.

```

package MyPackage;
public class MyClass {
    void aMethod() {
        int temp2 = query();
        temp2 = templ();
    }

    private int query() {
        return 1;
    }
}

// --Recycle Bin START (10/18/02 4:37 PM):
//     private int query() {
//         return 1;
//     }
// --Recycle Bin STOP (10/18/02 4:37 PM)
}

```

Figure 9.8. Comment out (914)

10.2.1.3. Safe delete **X**

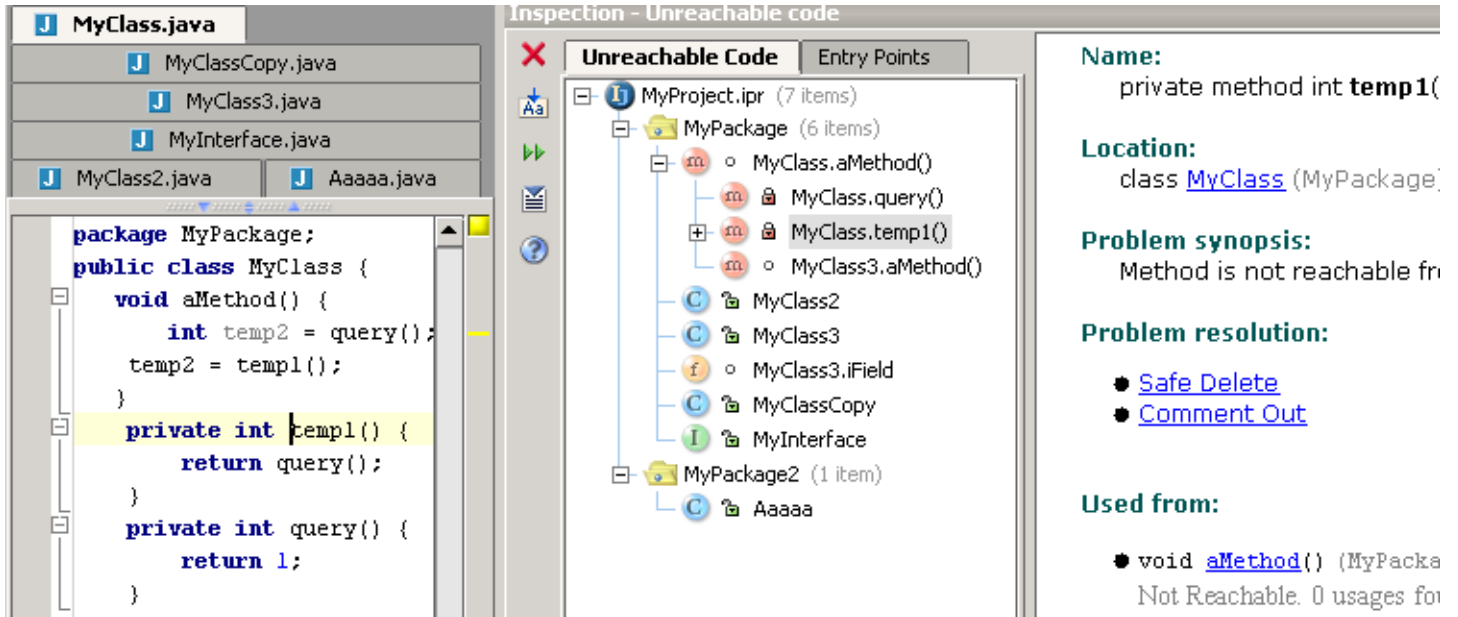


Figure 9.9. Safe delete (909)
9.15. Click on **Safe delete**.

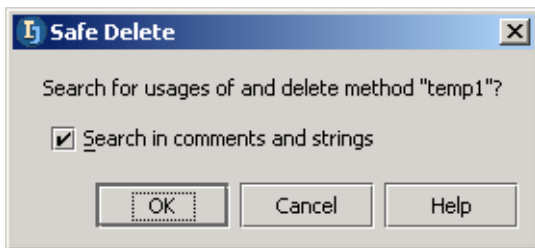


Figure 9.10. Safe delete (910)
9.16. Click **OK**.

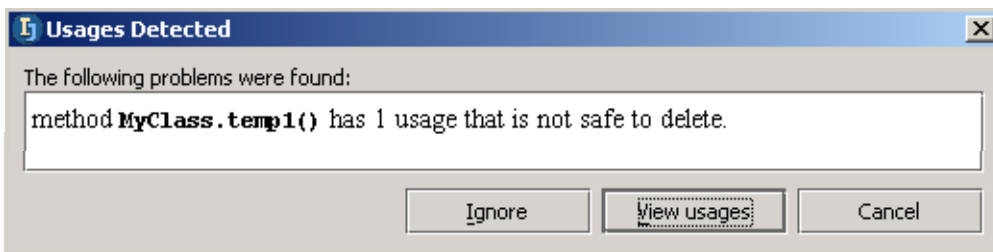


Figure 9.11. Usages detected (911,912)
9.17. Ignore and delete.

10.2.2. Manual X

Autoscroll to source.

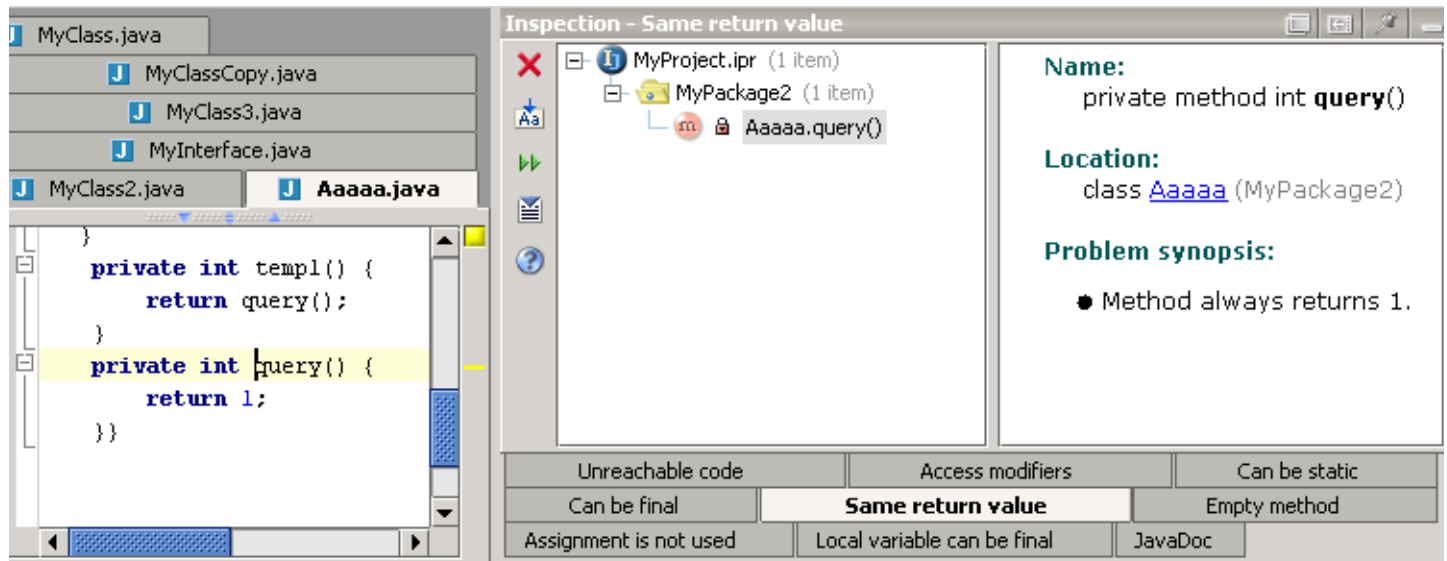


Figure 9.12. xxx (917)

10.3. Supported inspections XXX

10.3.1. Unreachable declaration XXX

10.3.1.1. Options XXX

10.3.1.1.1. Automatically add to entry points XXX

void main(String args[]) methods XXX

JUnit testcases XXX

Applets XXX

Servlets XXX

Classes that have usages in non-java files XXX

10.3.2. Declaration redundancy XXX

10.3.2.1. Declaration access can be weaker XXX

10.3.2.1.1. Options XXX

10.3.2.1.1.1. Suggest XXX

Package local visibility level for class members XXX

Package local visibility level top-level classes XXX

Private for inner class member when referenced from outer class only XXX

10.3.2.2. Declaration can have static modifier XXX

10.3.2.3. Declaration can have final modifier XXX

10.3.2.3.1. Options XXX

10.3.2.3.1.1. Report XXX

Classes XXX

Methods XXX

Fields XXX

10.3.2.4. Unused method parameters XXX

10.3.2.5. Actual method parameter is the same constant XXX

10.3.2.6. Unused method return value XXX

10.3.2.7. Method returns the same value XXX

10.3.2.8. Empty method XXX

10.3.2.9. Redundant throws clause XXX

10.3.3. Local code analysis XXX

10.3.3.1. Constant conditions & exceptions XXX

10.3.3.2. Unused assignment XXX

10.3.3.3. Redundant type cast XXX

10.3.3.4. Local variable or parameter can be final XXX

10.3.3.4.1. Options XXX

10.3.3.4.1.1. Report XXX

Local variables XXX

Method parameters XXX

10.3.3.5. Declaration has javadoc problems XXX

10.3.3.5.1. Options XXX

10.3.3.5.1.1. Class XXX

10.3.3.5.1.1.1. Scope XXX

10.3.3.5.1.1.2. Required tags XXX

- @author
- @version
- @since

10.3.3.5.1.2. Method XXX

10.3.3.5.1.2.1. Scope XXX

10.3.3.5.1.2.2. Required tags XXX

- @return
- @param
- @throws or @exception

10.3.3.5.1.3. Field XXX

10.3.3.5.1.3.1. Scope XXX

10.3.3.5.1.4. Inner class XXX

10.3.3.5.1.4.1. Scope XXX

10.3.4. Deprecated API usage XXX

10.3.5. equals() and hashCode() not paired XXX

10.3.6. EJB Errors & Warnings XXX

10.3.6.1. Options XXX

10.3.6.1.1. Report XXX

Errors XXX

Warnings XXX

10.4. Entry points XXX

10.4.1. Add XXX

10.4.2. View in inspection (unreachable code) XXX

10.5. Export to html XXX

10.6. Offline inspection results XXX

10.6.1. Create XXX

10.6.2. View XXX

11. Version control

[20021022TTT last update.](#)

IDEA supports the following versioning tools:

- Local (built-in)
- StarTeam
- CVS
- SourceSafe

This chapter demonstrates:

- **11.1. Local (page 256)**
- ~~11.2. CVS XXX (page 263)~~
- ~~11.3. SourceSafe XXX (page 264)~~
- ~~11.4. StarTeam (page 265)~~

11.1. Local

20021012TTT: Add colors and fonts for local

CONTACT: valya or mike.

20020908TTT: ?? recommended changes to the history dialog:

current...

Modified	Inserted	Deleted	No differences
Date	Comment		Action
8/3/02 1:15 PM	Project opened		
9/9/02 1:00 PM	After lunch		

?? Current history dialog (516)

recommended...

Differences				
Date	Before change / Current		Before change / After change	
	view	Label (comment)	view	Label (comment)

Figure 9.13. ?? Recommended changes to History dialog (517)

- In the "view" column: A button you click to view the selected change.
- In the "label" column: The text comment.

Also... the rows would be side-by-side (not zigzag as current).

IDEA's local versioning tool provides all-around versioning functionality. This section demonstrates the following local VCS functionality:

- **11.1.1. Open History dialog (page 256)**
- **11.1.2. Insert (page 258)**
- **11.1.3. Modify (page 259)**
- **11.1.4. Delete (page 259)**
- **11.1.5. Rollback (page 260)**
- **11.1.6. Viewing changes (page 261)**
- **11.1.7. Labels (page 262)**

11.1.1. Open History dialog

10.1. Create class **MyVCSLocal** (by selecting **New | Class** and not making any changes).

10.2. Right-click on **MyVCSLocal**.

10.3. Select **Local VCS | Show history**. The History dialog appears.

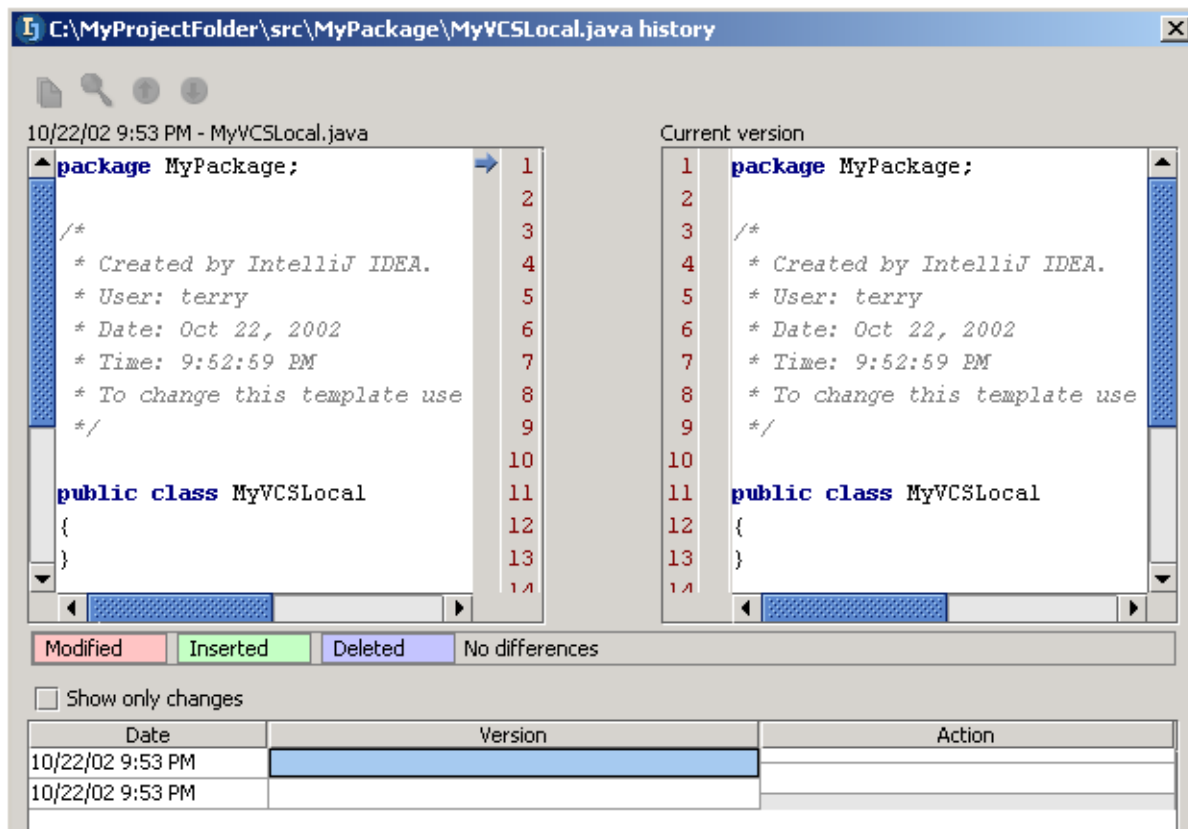


Figure 10.1. History dialog (527)

Note that only the current version is shown (no changes have been made).

11.1.2. Insert

20020909TTT: XXX need to add more detail about where to put the cursor so that you get an “insert” and not a “modify”.

This section shows how inserted text is displayed in local VCS.

10.4. Close the history dialog.

10.5. Add the following lines (type in, do not paste) to MyVCSLocal:

```
public static void main(String[] args) {  
    System.out.println("Hello");  
}
```

10.6. Reopen the history dialog.

10.7. Click on the lowest cell in the “Comment” column. The dialog shows the inserted text.

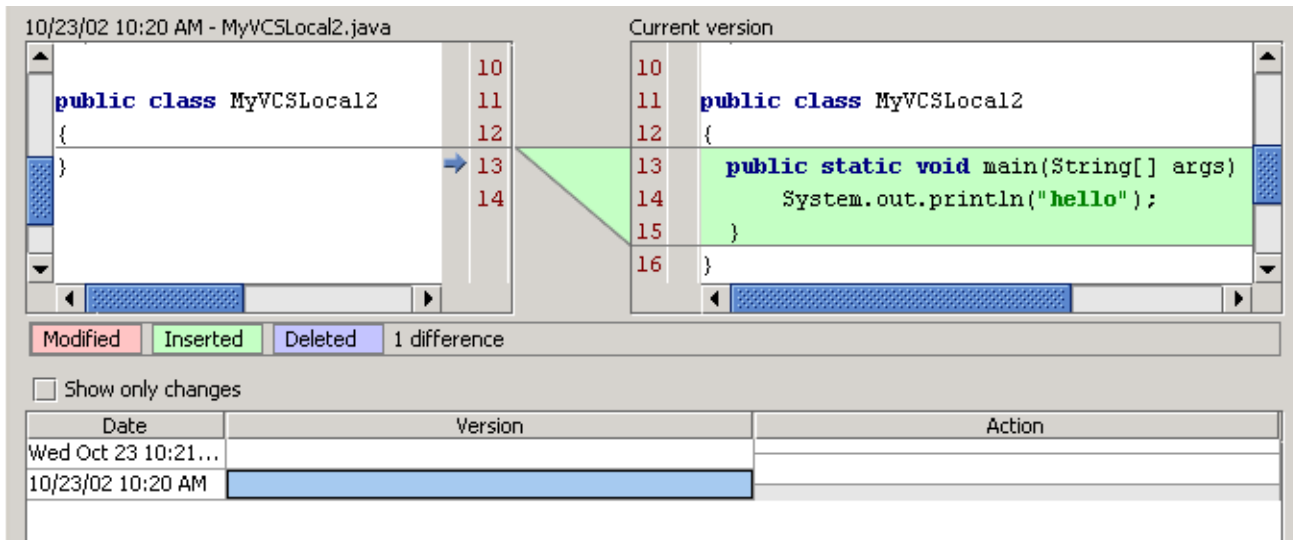


Figure 10.2. Inserted changes in history dialog (526)

11.1.3. Modify

- 10.8. Close the history dialog.
- 10.9. Change “Hello” to “**Hello World**”.
- 10.10. Open the history dialog.
- 10.11. Click on the second cell from the bottom in the “Comment” column. The dialog shows the modified line.

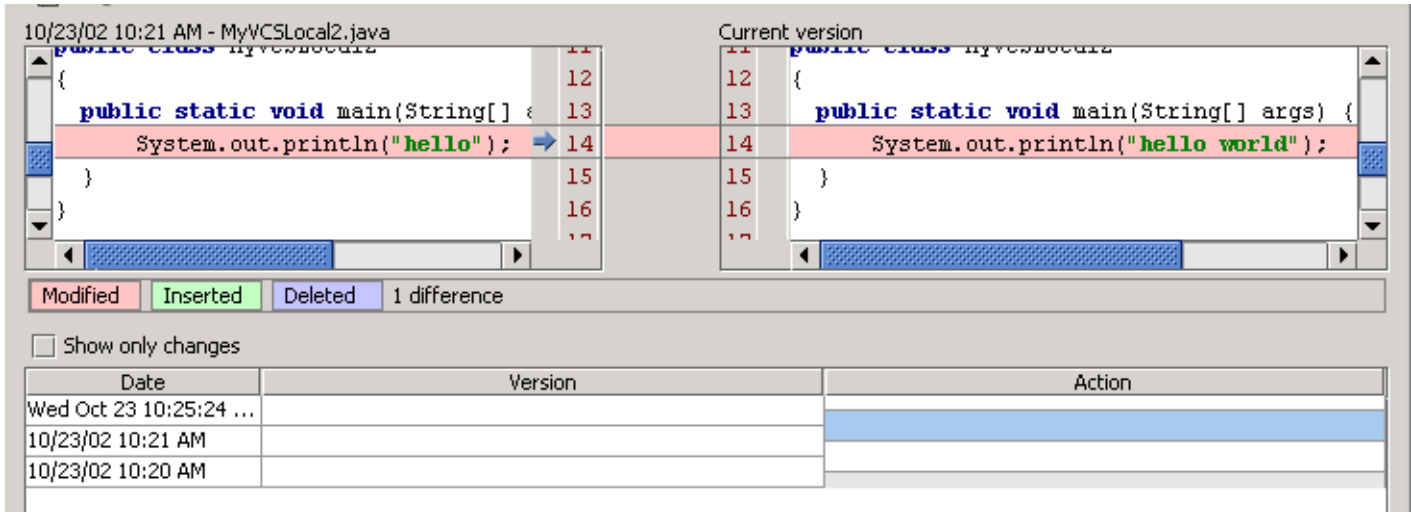


Figure 10.3. Modification in history dialog (523)

11.1.4. Delete

- 10.12. Close the history dialog.
- 10.13. Delete the line
`System.out.println("Hello World");`
- 10.14. Open the history dialog.
- 10.15. Click on the 3rd cell from the bottom in the “Comment” column. The deleted line is shown.

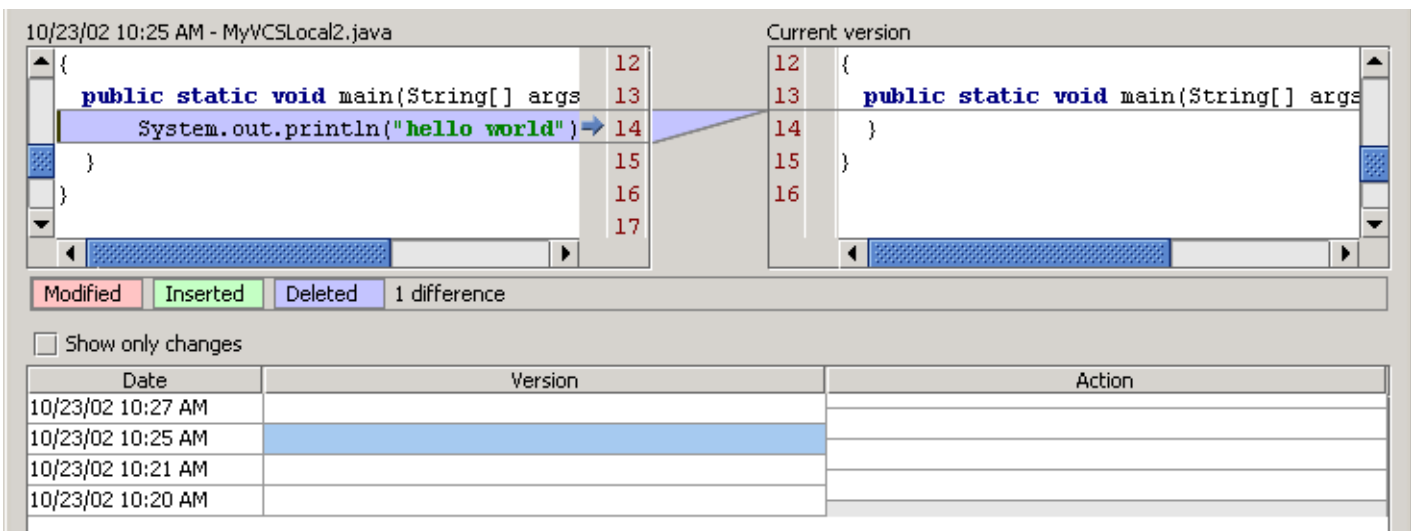


Figure 10.4. Deleted line in history dialog (522)

11.1.5. Rollback

10.16. Right-click on the 3rd cell from the bottom in the “Comment” column. A context menu appears.

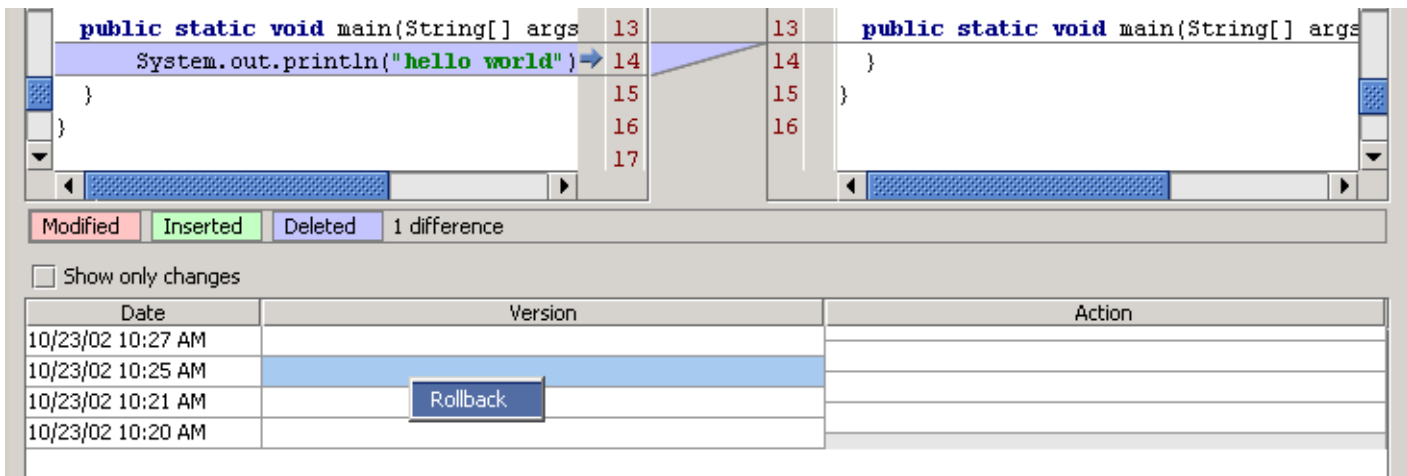


Figure 10.5. Rollback context menu (521)

10.17. Click **Rollback**. A message dialog appears “Rollback to the selected state?”.

10.18. Click **OK**. The code line is added and the history dialog is closed.

10.19. Reopen the history dialog.

10.20. Click on the 4th cell from the bottom in the “Comment” column. The dialog shows the rolled back change.

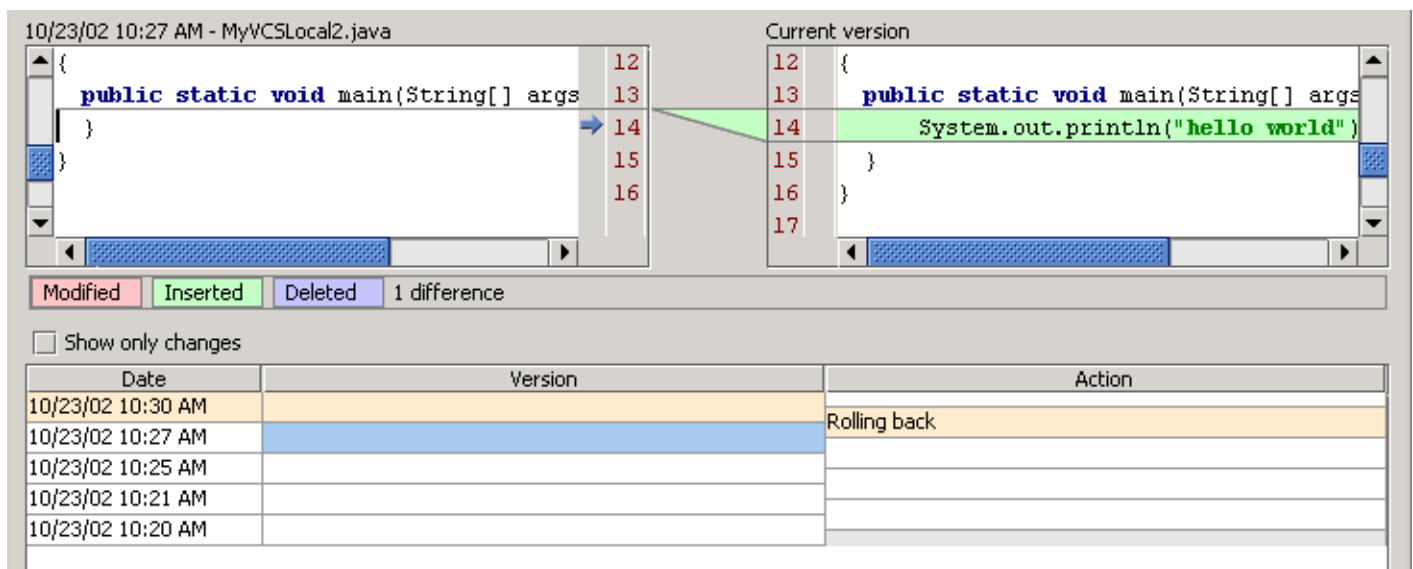


Figure 10.6. Rollback in history dialog (520)

11.1.6. Viewing changes

The history dialog shows the differences

- 11.1.6.1. Between version before change and current version (page 261)
- 11.1.6.2. Between versions before/after change (page 261)

11.1.6.1. Between version before change and current version

10.21. Click on the lowest cell in the “Comment” column. The dialog shows the changes between

- The file before the action was made
- The current (final) version of the file

20020909TTT: ?? why is the “modify”ed text not shown in pink?

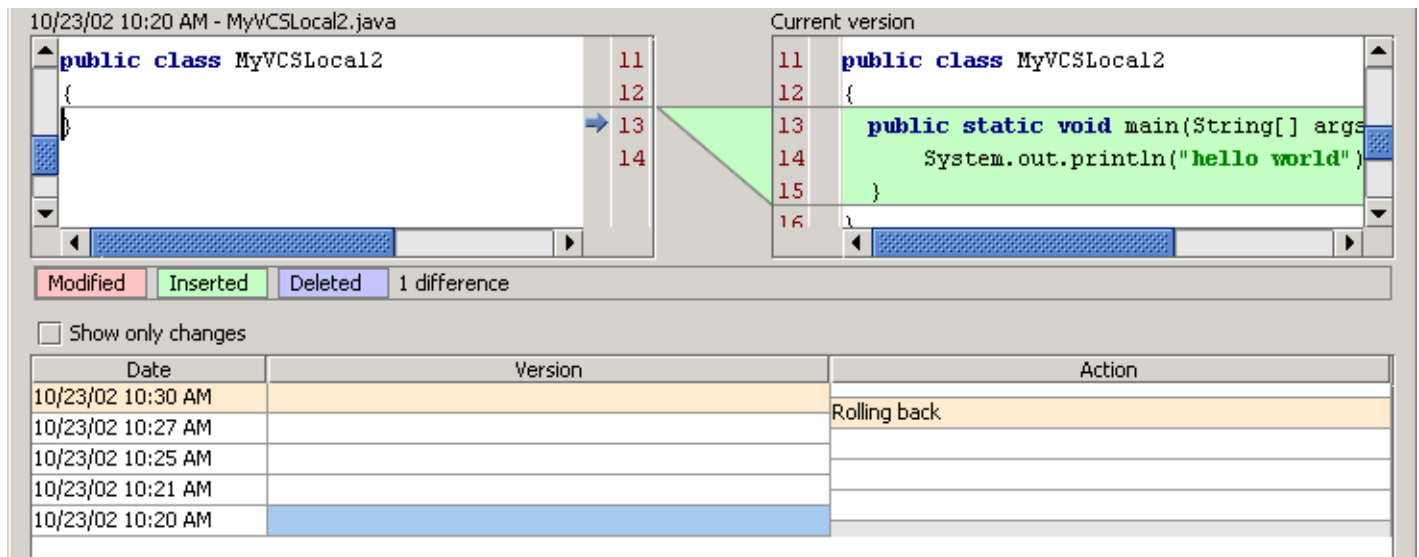


Figure 10.7. Differences between the original and the current version (525)

11.1.6.2. Between versions before/after change

10.22. Click on the lowest cell in the “Action” column. The dialog shows the changes between

- The file before the action was made
- The file after the action was made

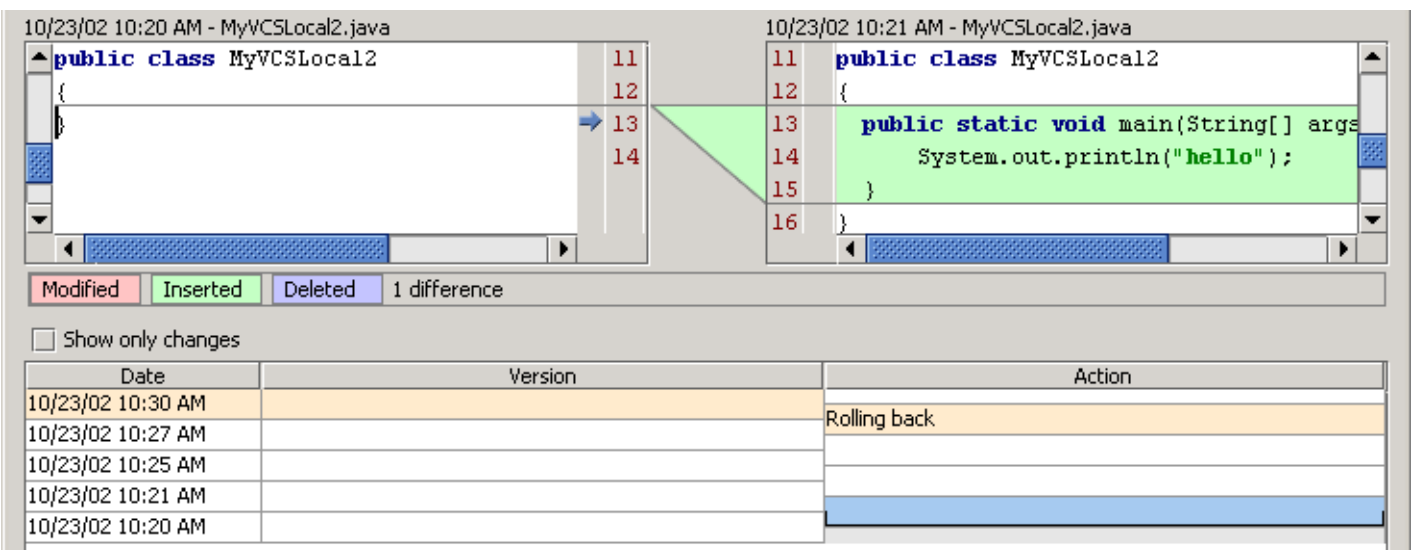


Figure 10.8. First action taken on file (524)

11.1.7. Labels

- 10.23. Close the History dialog.
- 10.24. Right-click on **MyVCSLocal**.
- 10.25. Select **Local VCS | Add label**. The dialog “Add label” appears.
- 10.26. For “Label name” enter **Label1**.

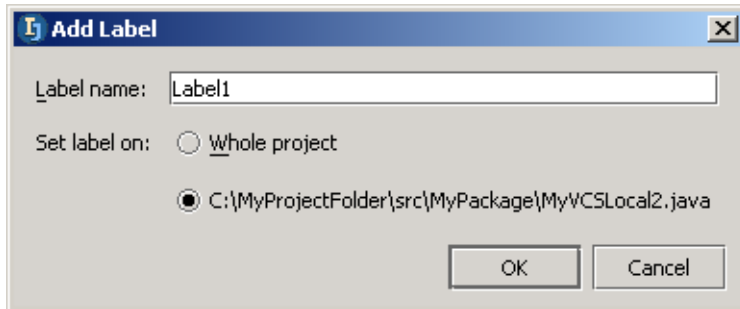


Figure 10.9. Add label dialog (519)

- 10.27. Click **OK**.
- 10.28. Open the History dialog. Note the label.

Date	Version	Action
10/23/02 10:39 AM	Label1	
10/23/02 10:30 AM		
10/23/02 10:27 AM		Rolling back
10/23/02 10:25 AM		
10/23/02 10:21 AM		
10/23/02 10:20 AM		

Figure 10.10. Label in history dialog (518)

~~11.2. CVS XXX~~

CONTACT: zheka

~~11.3. SourceSafe XXX~~

CONTACT: vova

11.4. StarTeam

~~20021023TT text added.~~

~~CONTACT: vova~~

~~idea.bat: SET CLASS_PATH=%CLASS_PATH%;%IDEA_HOME%\lib\starteam-sdk.jar~~

11.4.1. Install

~~10.29. Copy C:\idea650\starteam_plugin\starteam.jar to C:\idea650\plugins.~~

~~10.30. Add C:\Program Files\Starbase\StarGate SDK\Lib\starteam-sdk.jar to idea classpath.~~



Figure 10.11. StarTeam classpath ~~(266)~~

11.4.2. Configure connection

~~10.31. For VCS Support select StarTeam.~~

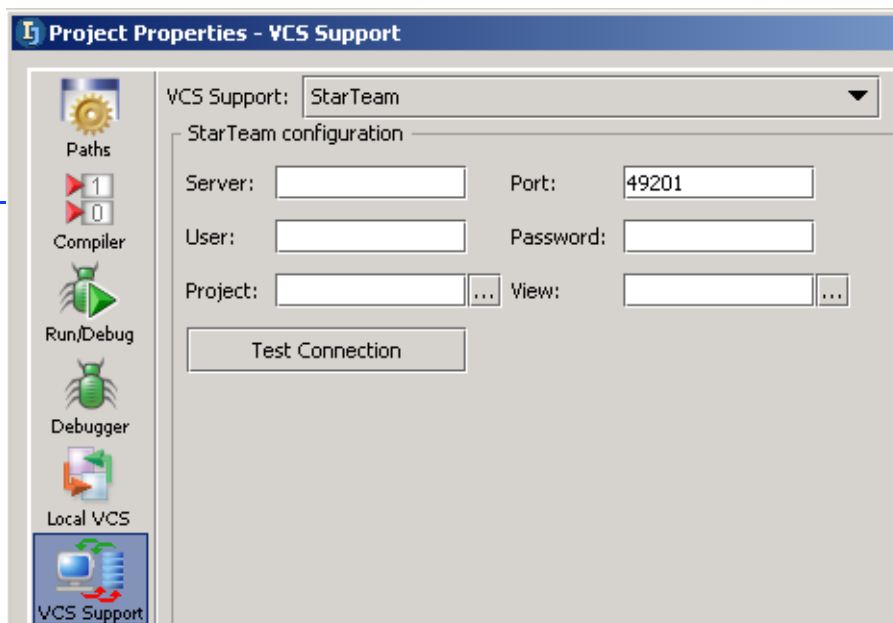


Figure 10.12. VCS Support StarTeam ~~(265)~~

~~10.32. Enter following:~~

- ~~▲ Server~~
- ~~▲ Port~~
- ~~▲ User~~
- ~~▲ Password~~
- ~~▲ Project~~
- ~~▲ View~~

~~10.33. Click Test Connection. The dialog "Connection successful" appears.~~

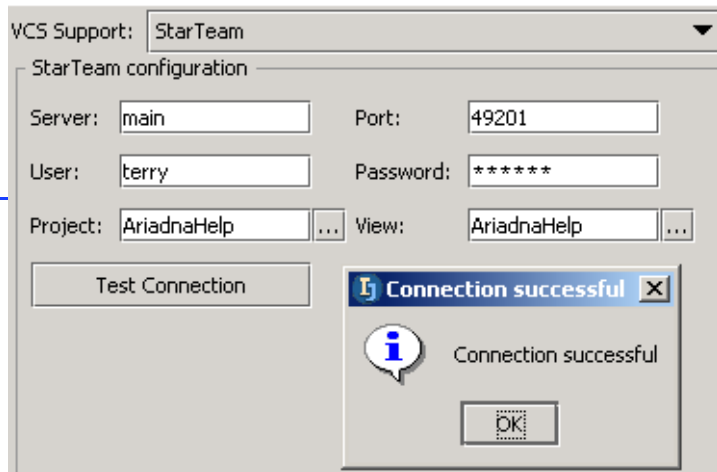


Figure 10.13. StarTeam configuration (264)
40.34. Click **OK**.

11.4.3. Add StarTeam dir to project

40.35. Add the folder:

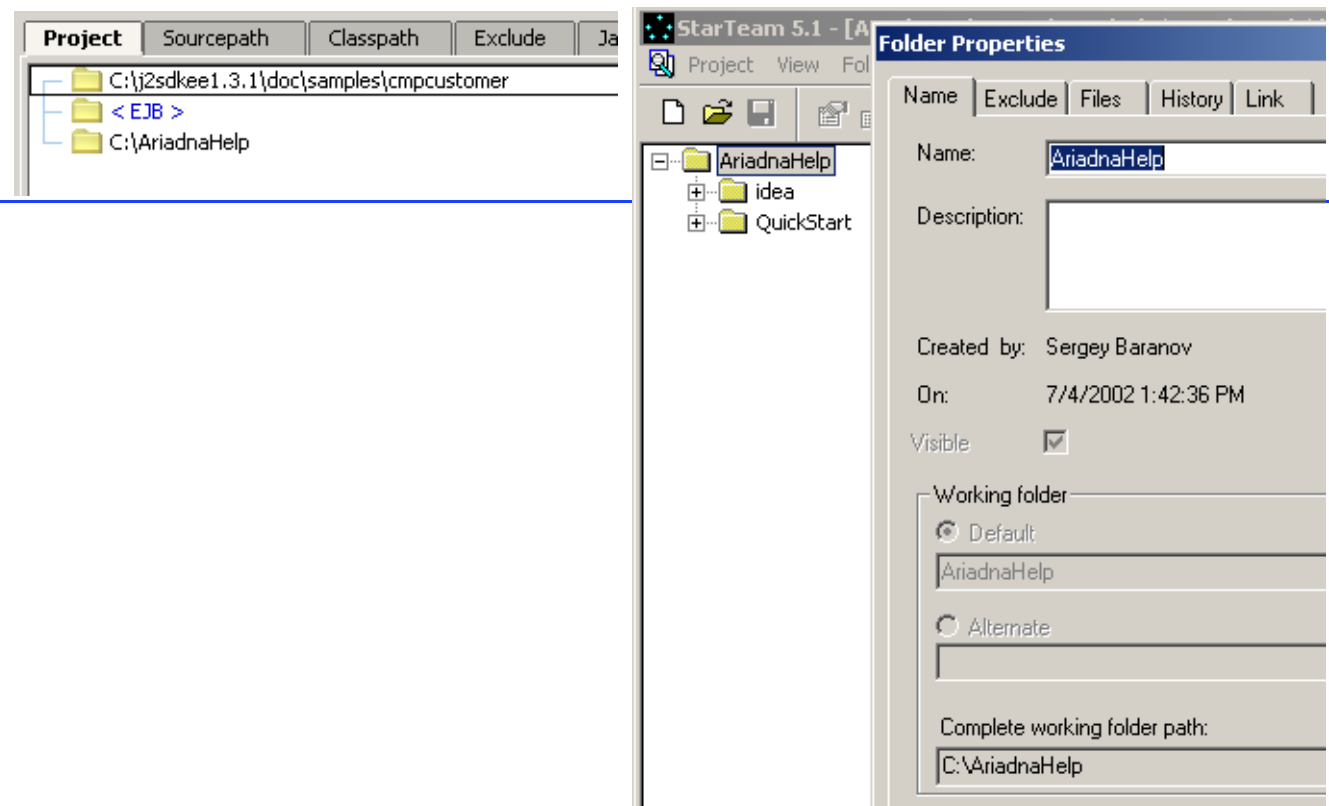


Figure 10.14. StarTeam folder added to project (263,262)

11.4.4. Create dir / file

40.36.

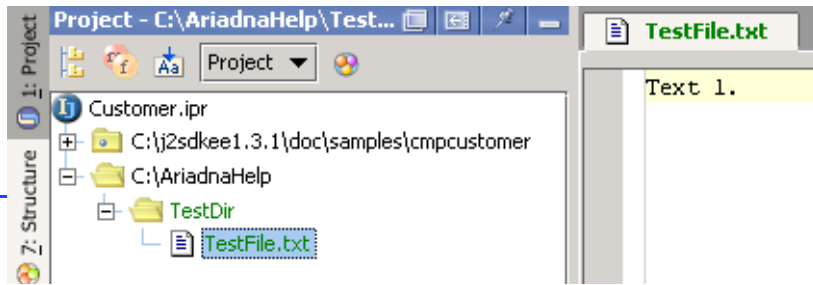


Figure 10.15. Dir, folder added (261)

11.4.5. Check in

10.37. Right click on TestFile.txt.

10.38. Select Check in Project.

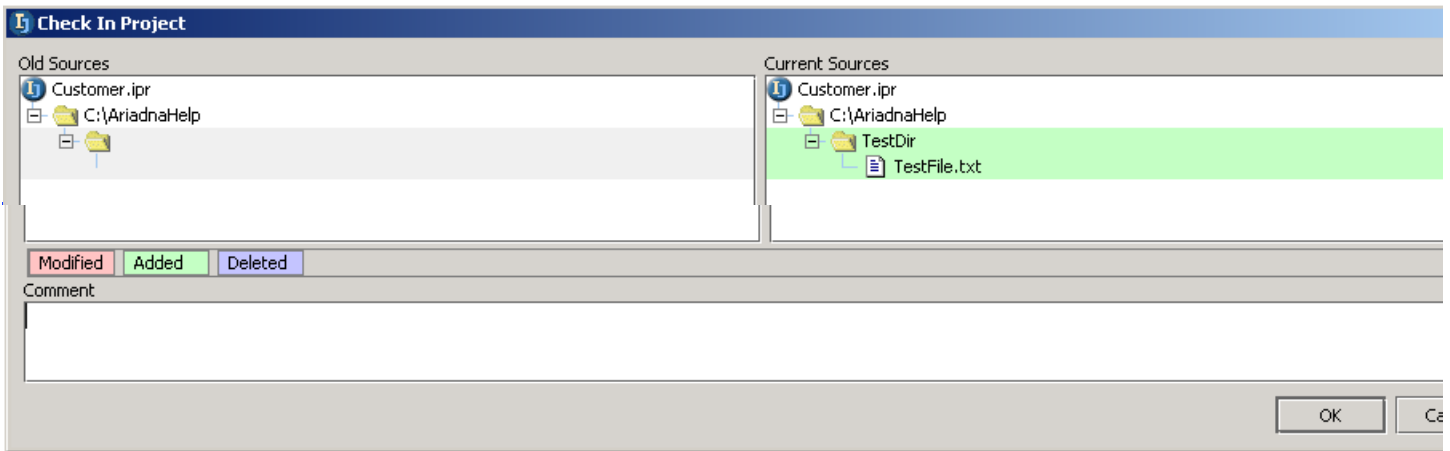


Figure 10.16. xxx (260,259)

10.39. Click OK.

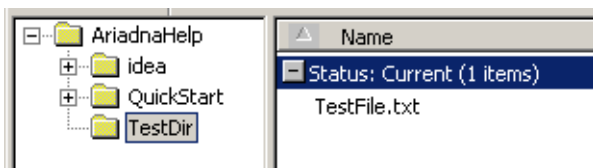


Figure 10.17. xxx (258)

11.4.6. Show differences

10.40. Add second line "Text 2." to TestFile.txt.

10.41. Select StarTeam / Show difference.

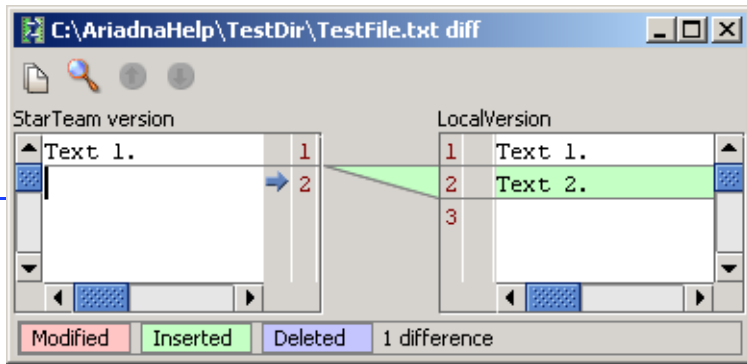


Figure 10.18. ~~xxxx (257)~~

11.4.7. ~~Check in~~

~~40.42. Right click on TestFile.txt.~~

~~40.43. Select Check in Project.~~

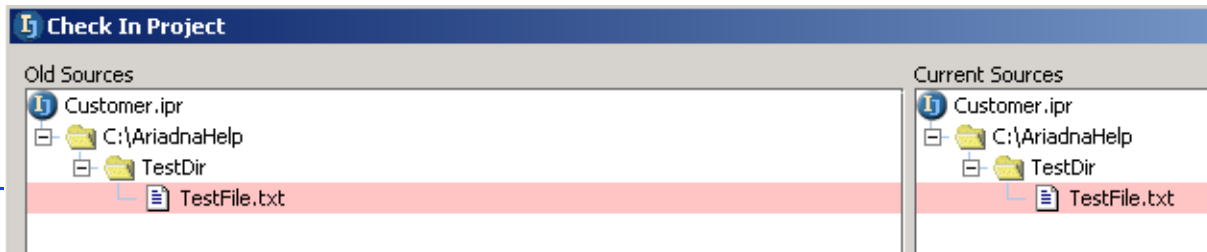


Figure 10.19. ~~xxx (256)~~

12. Java Doc

[20021023TT last edit.](#)

[contacts: anton](#)

This chapter demonstrates the following:

- [12.1. JDK JavaDoc \(page 269\)](#)
- [12.2. Quick JavaDoc \(page 273\)](#)
- [12.3. JavaDoc tags \(page 276\)](#)
- [12.4. Own JavaDoc \(page 278\)](#)

12.1. JDK JavaDoc

This section shows how to

- [12.1.1. Install \(page 269\)](#)
- [12.1.2. View \(page 271\)](#)

the JDK JavaDoc.

12.1.1. Install

11.1. Download the Java 2 SDK documentation ([j2sdk-1_4_1-doc.zip](#)) from [java.sun.com](#).

11.2. Extract to `c:\IntelliJ-IDEA-3.0` (use folder names).

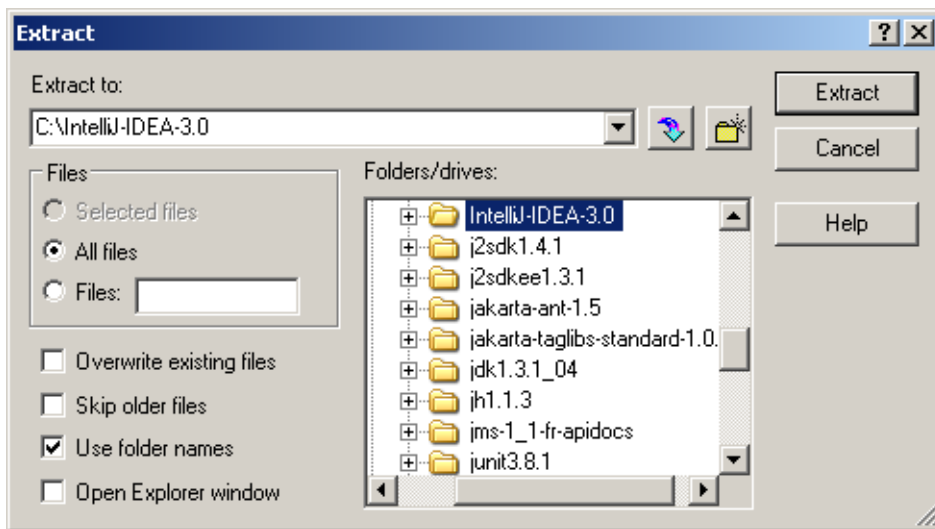


Figure 11.1. J2SDK docs extraction [\(933\)](#)

The javadoc api for the JDK classes are now located in `c:\IntelliJ-IDEA-3.0\docs\api`.

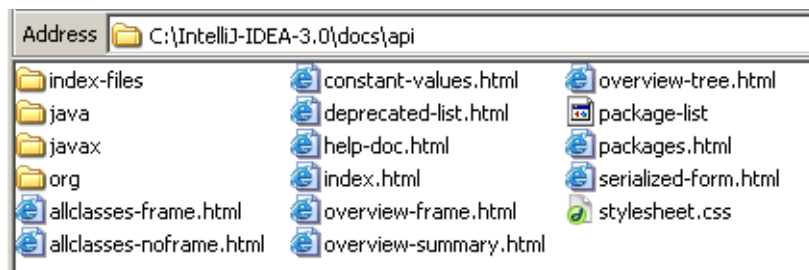


Figure 11.2. JavaDoc API dir [\(313\)](#)

11.3. In dialog “Project Properties” tab “Paths” tab “JavaDoc Paths”: Add the directory **c:\IntelliJ-IDEA-3.0\docs\api**.

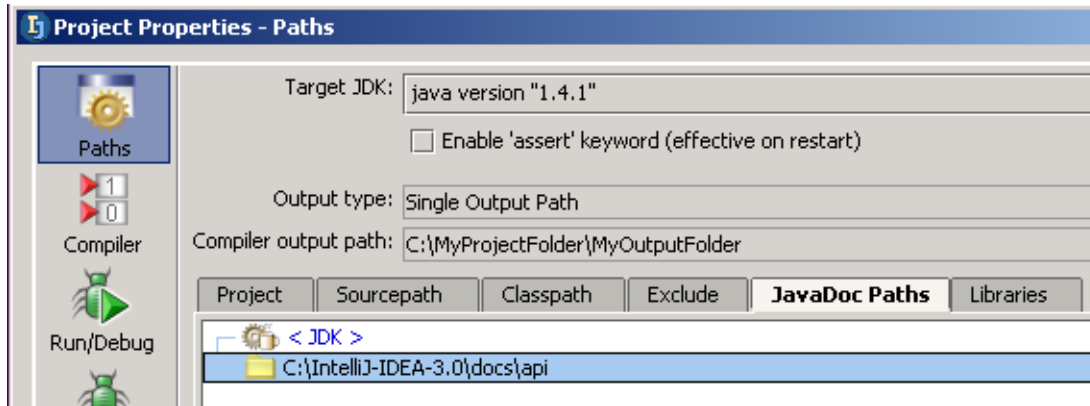


Figure 11.3. JavaDoc path [\(934\)](#)

12.1.2. View

This section shows how to open JavaDoc for

- [12.1.2.1. All \(page 271\)](#)
- [12.1.2.2. For element \(page 271\)](#)

12.1.2.1. All

11.4. Open `C:\IntelliJ-IDEA-3.0\docs\api\index.html`. The JavaDoc API docs start page appears.

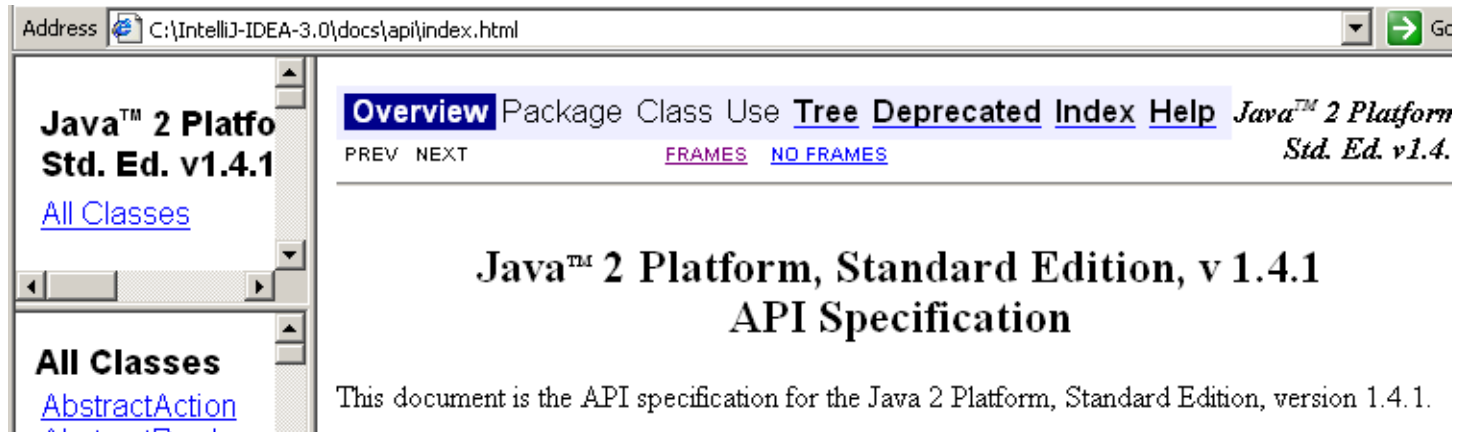


Figure 11.4. JavaDoc start page [\(312\)](#)

12.1.2.2. For element

You can open the JavaDoc description for an API class direct from within IDEA.

11.5. Select class Applet as shown below.

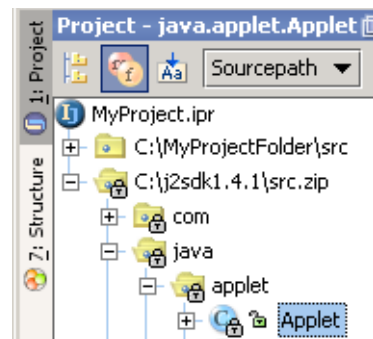


Figure 11.5. Select Applet [\(311\)](#)

11.6. Select **View | External Java doc**. The java documentation for class Applet is opened in an editor.

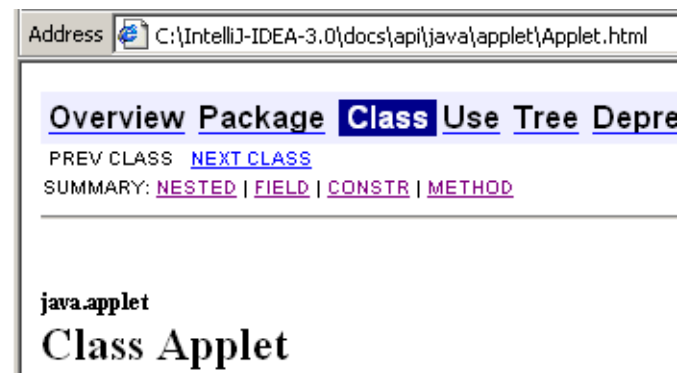


Figure 11.6. JavaDoc for Applet [\(310\)](#)

11.7. Open the source code for Applet.java as shown below.

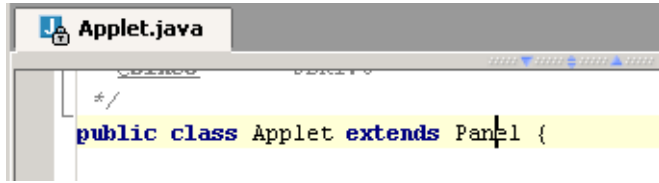


Figure 11.7. Applet source (309)

11.8. Place the cursor on **Panel**.

11.9. Click **Shift-F1**. JavaDoc for Panel is opened.

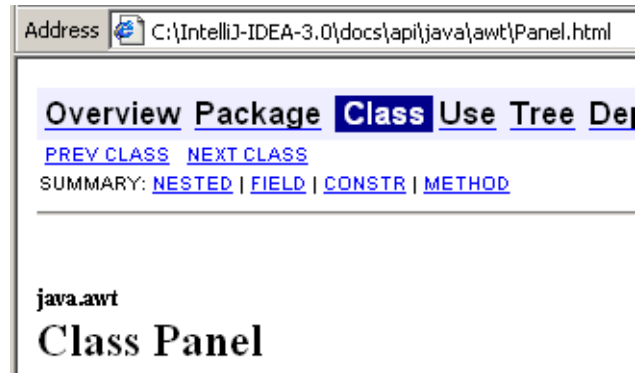


Figure 11.8. JavaDoc for Panel (308)

12.2. Quick JavaDoc

This section shows how to display IDEA's quick JavaDoc for

- 12.2.1. JDK class (page 273)
- 12.2.2. Own class (page 275)

12.2.1. JDK class

When opening quick JavaDoc for a JDK class, you can

- 12.2.1.1. Open for selected (page 273)
- 12.2.1.2. Open for other (page 274)
- 12.2.1.3. Open JDK JavaDoc (page 274)

12.2.1.1. Open for selected

11.10. Place the cursor on "Applet" in the class definition.

11.11. Press **Ctrl-Q**. The quick javadoc popup for "Applet" appears.

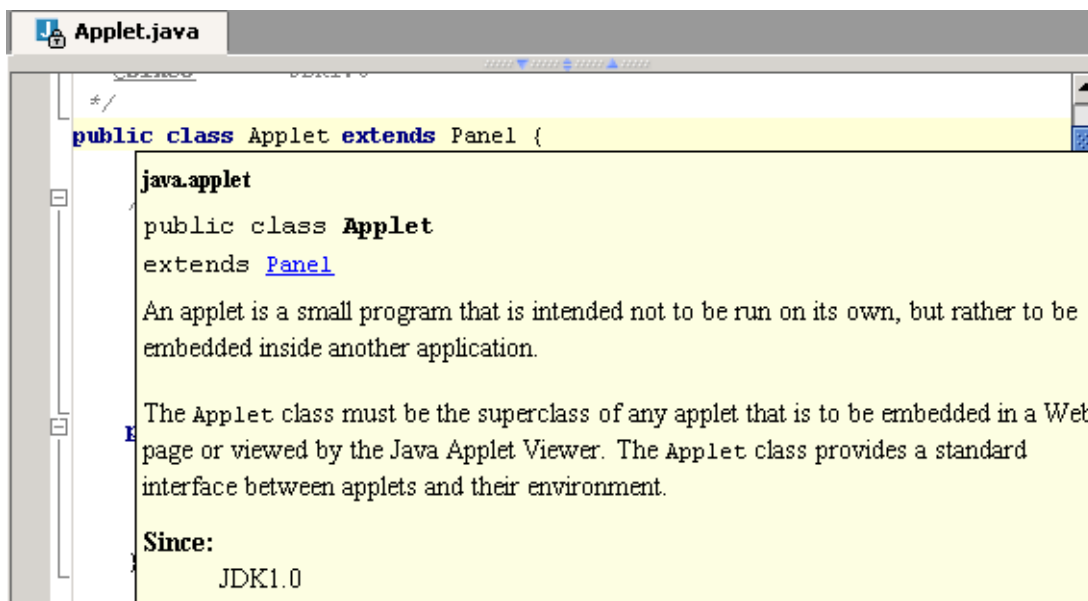


Figure 11.9. Quick JavaDoc for Applet (306)

Note that the above javadoc was generated directly from the contents of the file.

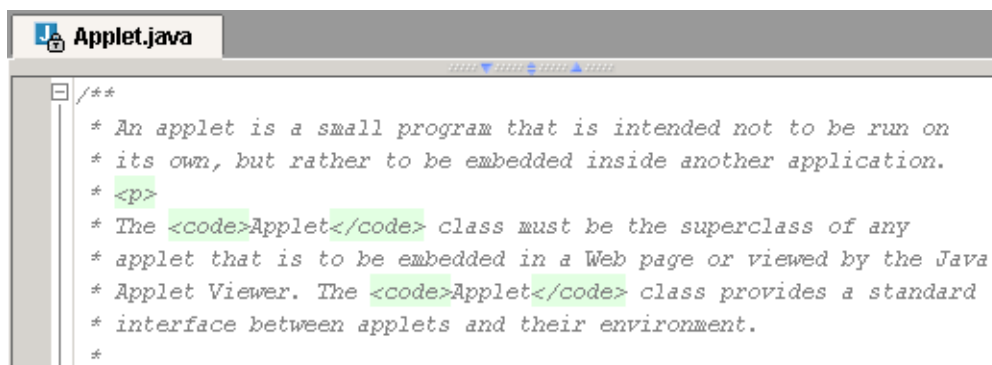


Figure 11.10. Applet definition (307)

12.2.1.2. Open for other

11.12. Click on the hyperlink for **Panel**. The quick javadoc for “Panel” is opened.

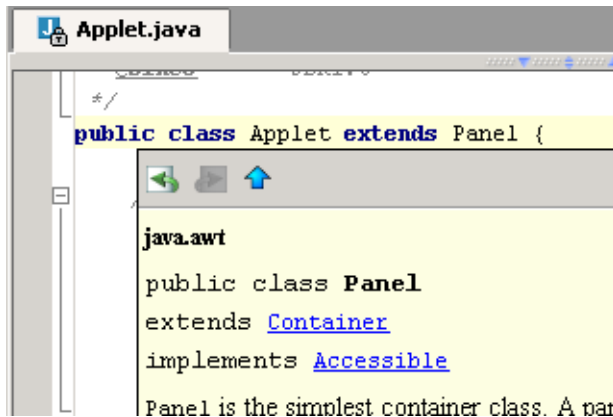


Figure 11.11. Quick JavaDoc for Panel (305)

Note: You can go back to the previous quick javadoc popup dialog by pressing the arrow icon ( ).

12.2.1.3. Open JDK JavaDoc

11.13. Click on the icon “View external JavaDoc” (). The external JavaDoc for Panel is opened.

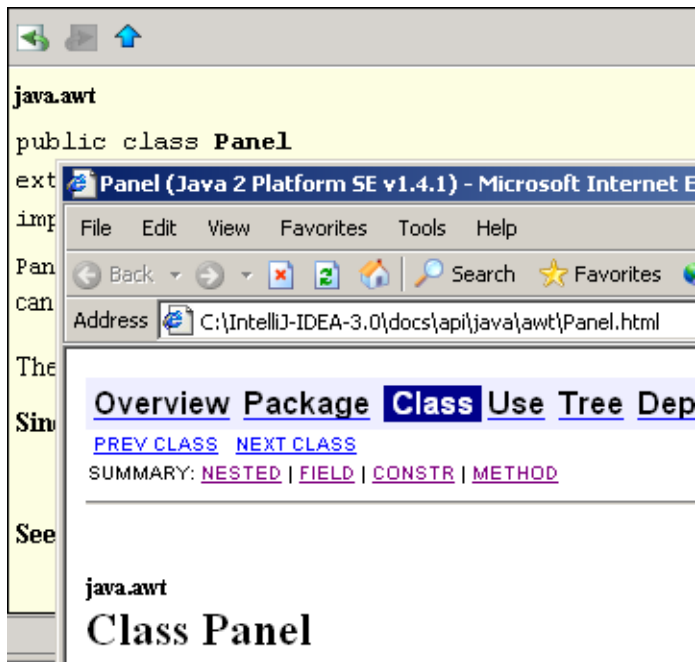


Figure 11.12. JavaDoc for Panel (302)

12.2.2. Own class

This section shows how to

- **12.2.2.1. Open (page 275)**

quick JavaDoc for your own class. You will also

- **12.2.2.2. Attempt to open JDK JavaDoc (none) (page 275)**

for your own class (this will not work at this point).

12.2.2.1. Open

11.14. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {}
```

11.15. Place the cursor on **MyClass**.

11.16. Click **Ctrl-Q**. The quick javadoc is opened.

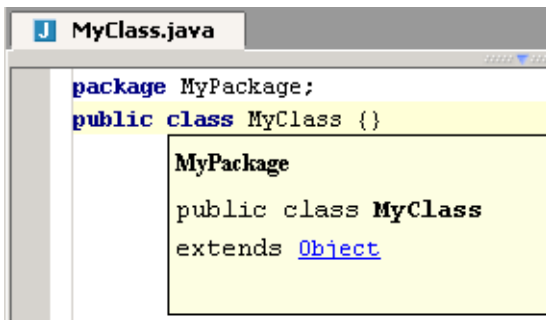


Figure 11.13. Quick JavaDoc for own class (301)

12.2.2.2. Attempt to open JDK JavaDoc (none)

20021023TTT the javadoc arrow doesnt appear above.

11.17. Try to open JavaDoc for MyClass. Note the error:

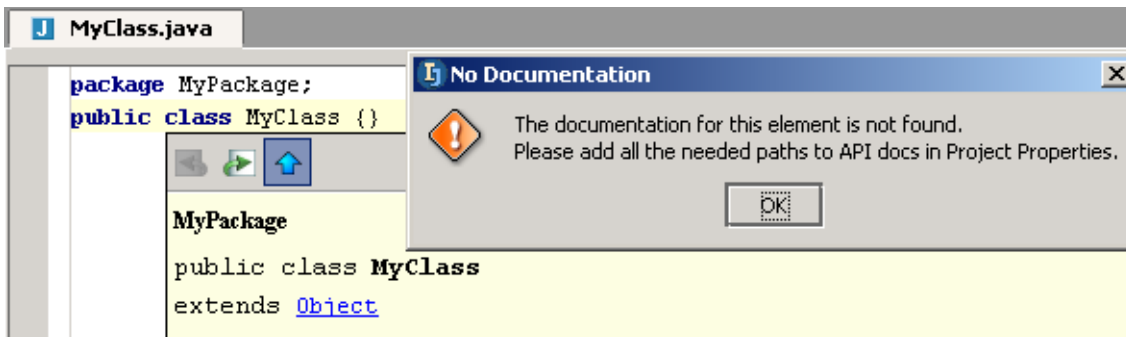


Figure 11.14. JavaDoc error for MyClass (300)

12.3. JavaDoc tags

This section shows how to

- [12.3.1. Add tags \(page 276\)](#)
- [12.3.2. Display tags in quick JavaDoc \(page 277\)](#)

12.3.1. Add tags

11.18. Create class:

```
package MyPackage;  
public class MyClass {  
    public int aMethod(int param1, int param2) {  
        return param1 + param2;  
    }  
}
```

11.19. Place the cursor at the beginning of the line:

```
public int aMethod(int param1, int param2) {
```

11.20. Enter `/**`.

11.21. Press **Enter**. The tags for the method are entered:

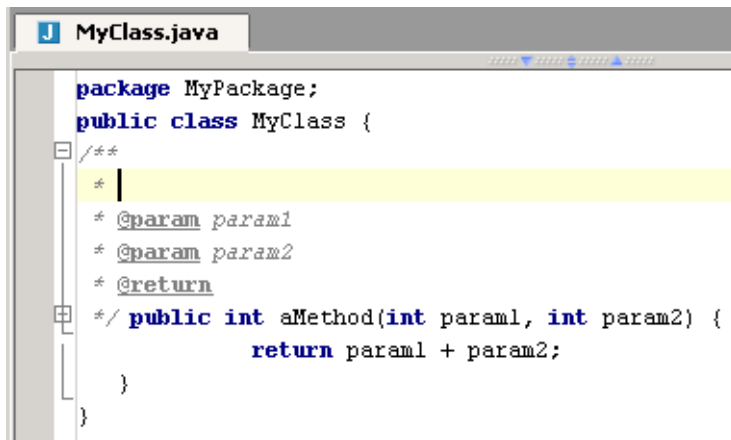


Figure 11.15. JavaDoc method tags [\(299\)](#)

11.22. Place the cursor after the `*` on line 4 (it is probably already there).

11.23. Enter `@`. Note that a popup dialog appears.

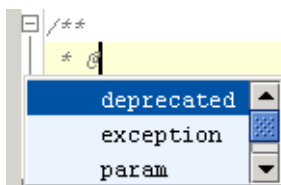


Figure 11.16. `@` popup [\(298\)](#)

11.24. Double-click **exception**. “exception” is added.

11.25. Click **Space**.

11.26. Click **Ctrl-Space**. A popup dialog with a list of exceptions appears.

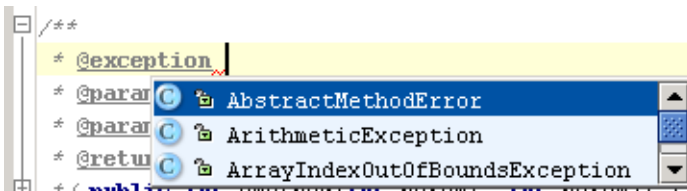


Figure 11.17. Exception list (297)

11.27. Double-click on **Unknown error**.

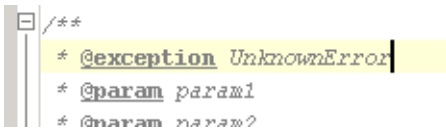


Figure 11.18. UnknownError tag added (296)

12.3.2. Display tags in quick JavaDoc

11.28. Place the cursor on **aMethod**.

11.29. Click **Ctrl-Q**. Quick JavaDoc is displayed.

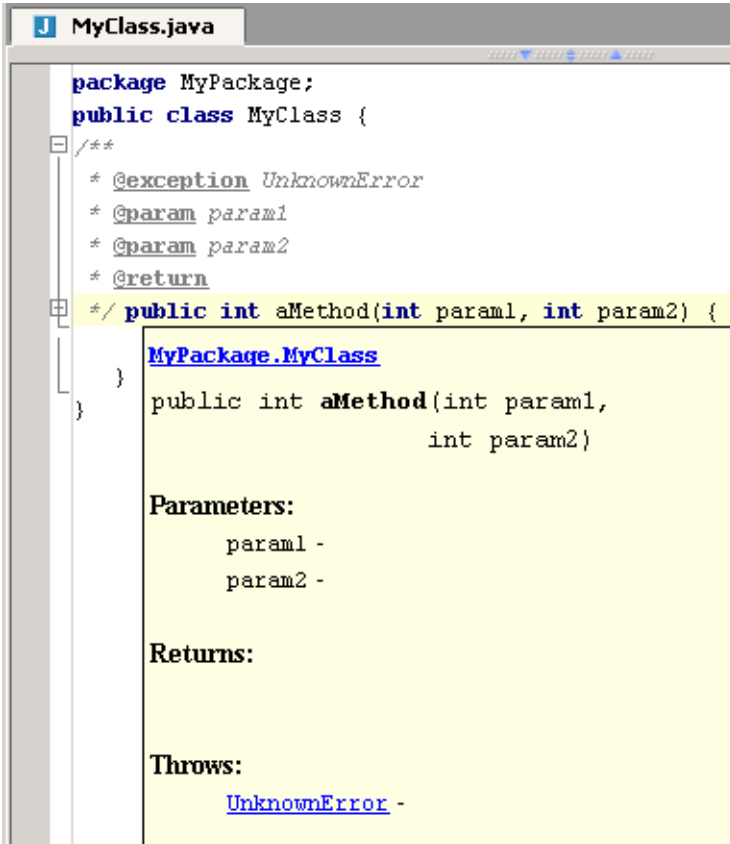


Figure 11.19. Quick JavaDoc for MyClass (295)

12.4. Own JavaDoc

This section demonstrates how to

- **12.4.1. Generate** (page 278)
- **12.4.2. Install** (page 280)
- **12.4.3. View** (page 280)

your own JavaDoc.

12.4.1. Generate

11.30. Select **MyPackage**.

11.31. Select **Tools | Generate JavaDoc....** The dialog “Generate JavaDoc” appears.

11.32. Select **Package “MyPackage”**.

11.33. For “Output directory” enter **c:\IntelliJ-IDEA-3.0\MyJavaDoc** (create the directory).

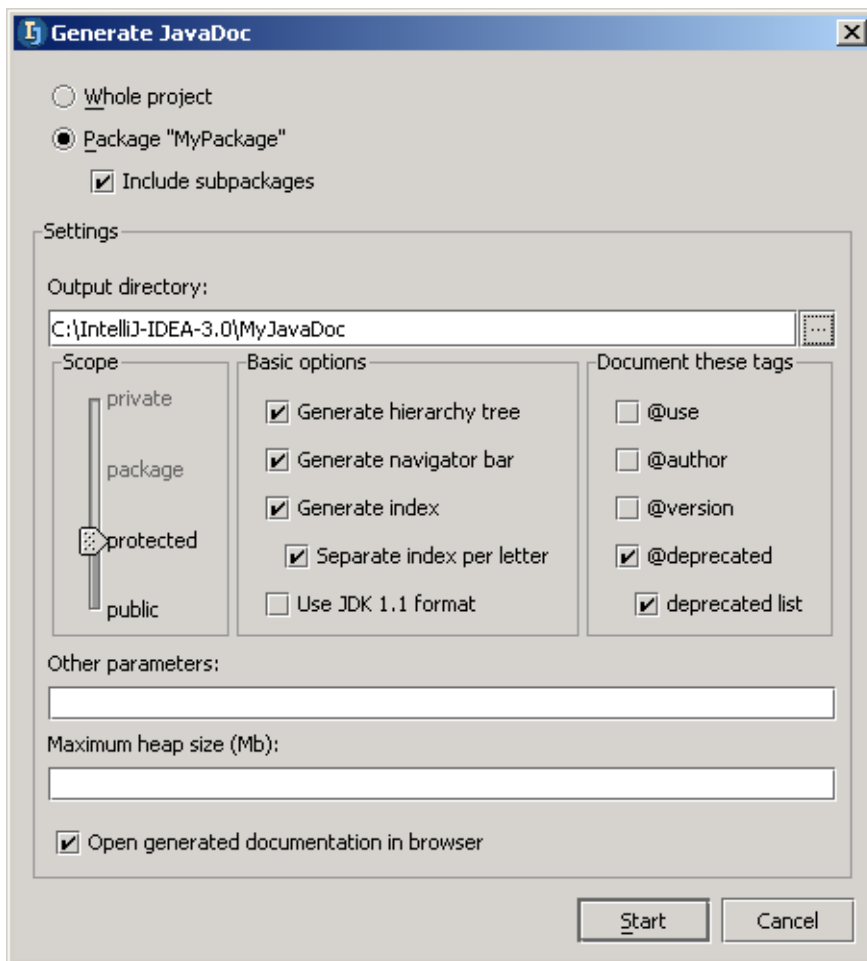


Figure 11.20. Dialog “Generate JavaDoc” (294)

11.34. Click **Start**. The tool “Run - JavaDoc” appears.

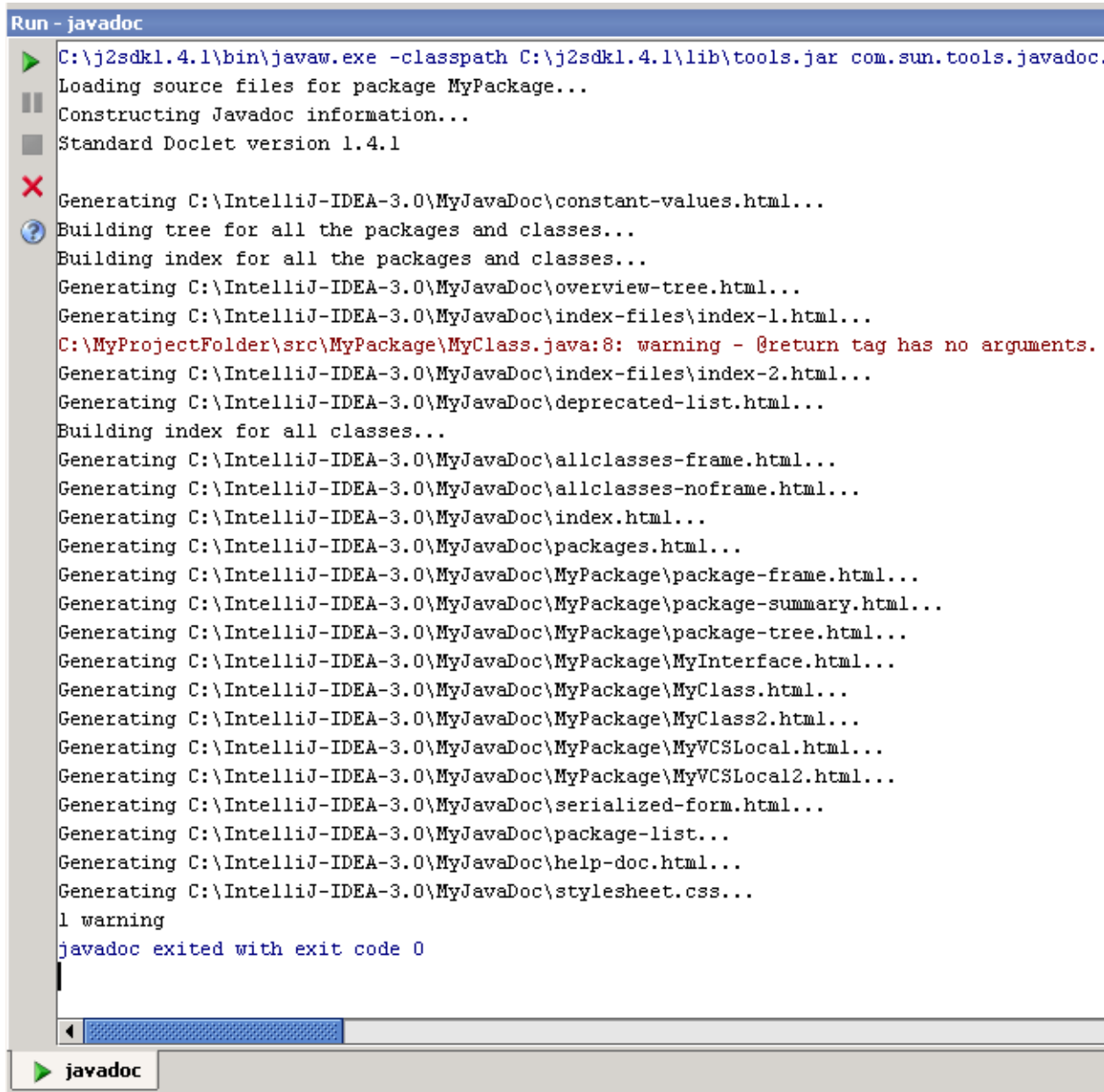


Figure 11.21. Run JavaDoc tool (293)

The generated JavaDoc is opened in a browser:



Figure 11.22. Generated JavaDoc (935)

12.4.2. Install

11.35. Add `c:\IntelliJ-IDEA-3.0\MyJavaDoc` to the JavaDoc API paths.

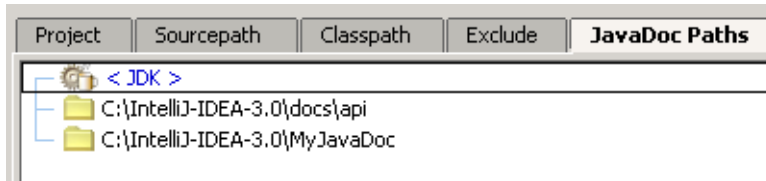


Figure 11.23. Add to JavaDoc API paths (292)

12.4.3. View

11.36. Place the cursor on `aMethod`.

11.37. Select **View | External JavaDoc**. JavaDoc is displayed.

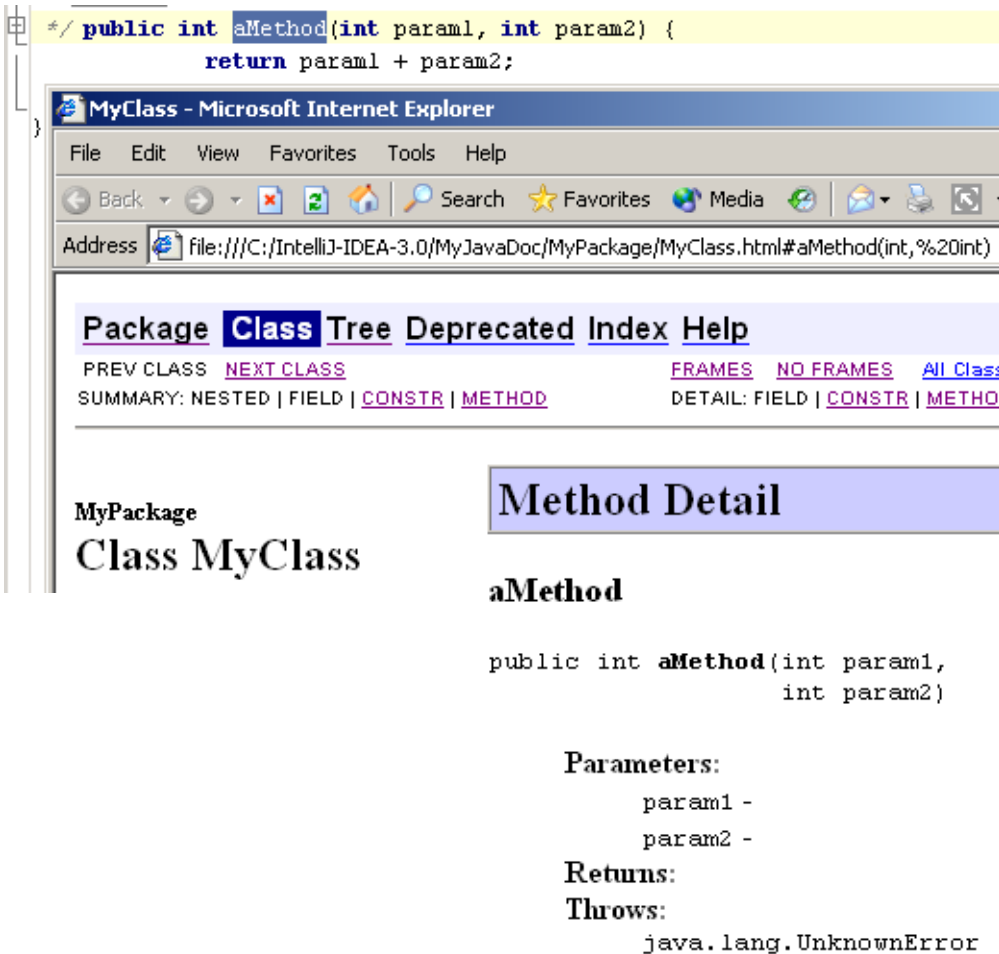


Figure 11.24. JavaDoc for MyClass (291.936)

Part D. Compile / Debug

[20021018TTT last edit.](#)

The chapters in this part describe the extensive compile / debug functionality provided and supported by IDEA.

13. Compiler (page 283). Demonstrates how to use the IDEA compiler.

14. Ant (page 295). Demonstrates how to use Ant with IDEA.

15. Debugger (page 301). Demonstrates how to use the IDEA debugger.

~~**16. JUnit XXX (page 313).**~~

~~**17. Remote debugging XXX (page 315).**~~

13. Compiler

[20021024TTT: last edit.](#)

[contacts: zheka](#)

[Libraries??](#)

[Add Jikes somewhere](#)

Compilation in IDEA is easy to use and provies many compilation options.

- **13.1. Compile variations (page 283)**
- **13.2. Compiler messages dialog (page 286)**
- **13.3. Compiler options (page 289)**

13.1. Compile variations

The following compile variations are available

- **13.1.1. Build / Rebuild (page 283)**
- **13.1.2. Make (page 285)**
- **13.1.3. Compile (single file) (page 285)**

13.1.1. Build / Rebuild

Build / rebuild compiles all classes in the project (even if the classes have not changed).

12.1. Create class **MyClass**:

```
package MyPackage;
public class MyClass {
    public static void main(String[] args) {
        MyClass2 mc2 = new MyClass2();
        mc2.aMethod();
        MyClass3 mc3 = new MyClass3();
        mc3.aMethod();
    }
}
```

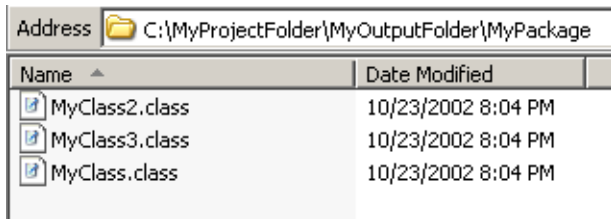
12.2. Create class **MyClass2**:

```
package MyPackage;
public class MyClass2
{
    public void aMethod() {
        System.out.println("A");
    }
}
```

12.3. Create class **MyClass3**:

```
package MyPackage;
public class MyClass3
{
    public void aMethod() {
        System.out.println("A");
    }
}
```

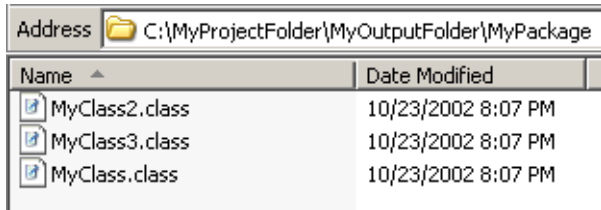
12.4. Select **Build | Rebuild project**. The project is built. All 3 files are compiled.



Name	Date Modified
MyClass2.class	10/23/2002 8:04 PM
MyClass3.class	10/23/2002 8:04 PM
MyClass.class	10/23/2002 8:04 PM

Figure 12.1. Built class files [\(964\)](#)

12.5. Select **Build | Rebuild project**. Note that again all 3 files are compiled (even though no changes were made).



Name	Date Modified
MyClass2.class	10/23/2002 8:07 PM
MyClass3.class	10/23/2002 8:07 PM
MyClass.class	10/23/2002 8:07 PM

Figure 12.2. Rebuilt class files [\(965\)](#)

13.1.2. Make

Make involves compilation of only those classes that have been modified since the last compilation.

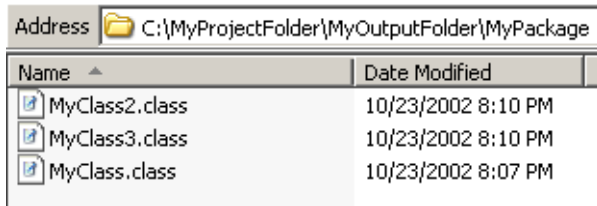
12.6. Modify class **MyClass2**:

```
System.out.println("B");
```

12.7. Modify class **MyClass3**:

```
System.out.println("B");
```

12.8. Select **Build | Make project**. The project is made: Only the modified files are compiled.



Name	Date Modified
MyClass2.class	10/23/2002 8:10 PM
MyClass3.class	10/23/2002 8:10 PM
MyClass.class	10/23/2002 8:07 PM

Figure 12.3. Made class files [\(966\)](#)

13.1.3. Compile (single file)

A single file can also be compiled.

12.9. Modify class **MyClass2**:

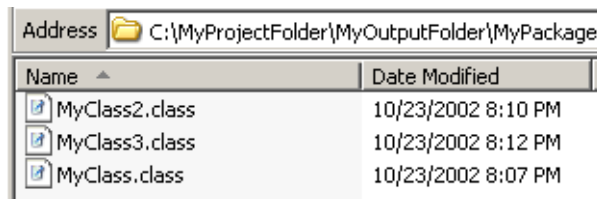
```
System.out.println("C");
```

12.10. Modify class **MyClass3**:

```
System.out.println("C");
```

12.11. Select the editor for **MyClass3**.

12.12. Select **Build | Compile "MyClass3.java"**. MyClass3 is compiled.



Name	Date Modified
MyClass2.class	10/23/2002 8:10 PM
MyClass3.class	10/23/2002 8:12 PM
MyClass.class	10/23/2002 8:07 PM

Figure 12.4. Compiled class file [\(967\)](#)

13.2. Compiler messages dialog

During compilation, if any problems occur the “Messages - Javac Compile” dialog will appear. In this dialog the following functions are available:

- 13.2.1. Stop (page 286)
- 13.2.2. Close (page 286)
- 13.2.3. Previous / next message (page 286)
- 13.2.4. Export to text file (page 286)
- 13.2.5. Expand / collapse all (page 287)
- 13.2.6. Hide warnings (page 287)
- 13.2.7. Autoscroll to source (page 287)
- 13.2.8. Compiler properties (page 288)

13.2.1. Stop

12.13. Click **Stop** () to stop compilation.

13.2.2. Close

12.14. Click **Close** () to close the dialog “Messages - Java Compile”.

13.2.3. Previous / next message

12.15. Click **Previous** () or **Next** () to select the previous / next message.

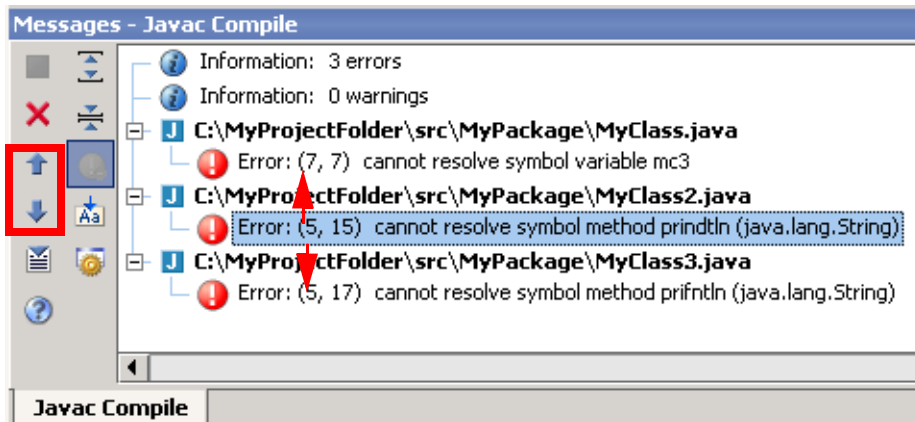



Figure 12.5. View previous / next message (968)

13.2.4. Export to text file

12.16. Click **Export to text file** (). The dialog “Export preview” appears.

12.17. For “Export to file” enter `C:\MyProjectFolder\export.txt`.

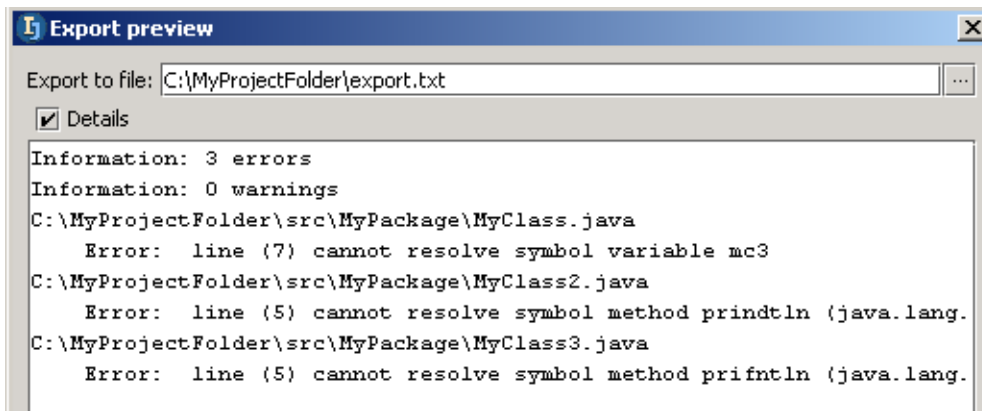


Figure 12.6. Dialog “Export preview” (972)

12.18. Click **Save**. The file is exported.

13.2.5. Expand / collapse all

12.19. Click **Expand all** () or **Collapse all** ().

13.2.6. Hide warnings

12.20. Modify **MyClass2**.

```
package MyPackage;
public class MyClass2
{
    public void aMethod() {
        System.out.println("C");
        Thread.currentThread().resume();
    }
}
```

12.21. Recompile.

12.22. Click **Hide warnings** () to hide the warning.

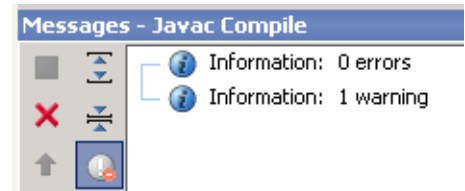
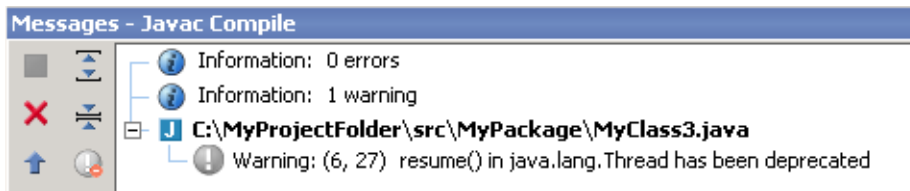


Figure 12.7. Hide warnings (974,975)

13.2.7. Autoscroll to source

12.23. Close all editors.

12.24. Click (enable) **Autoscroll to source** ().

12.25. Click on the warning. The source is opened.

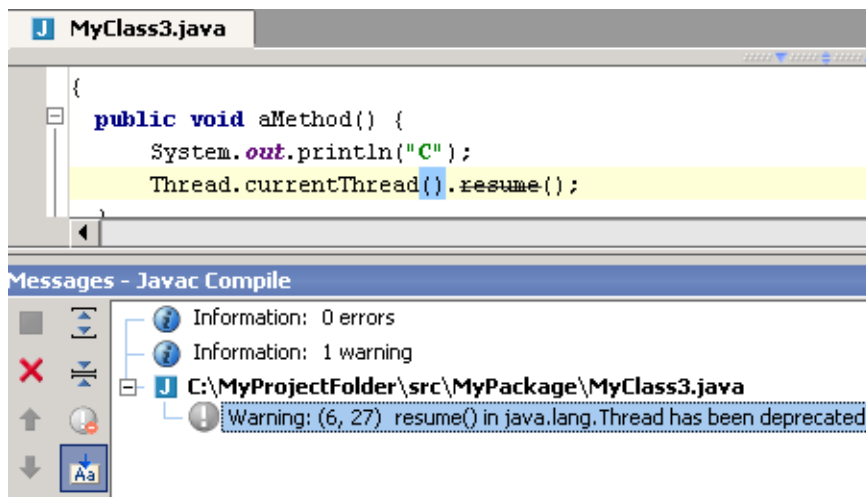


Figure 12.8. Autoscroll to source (978)

13.2.8. Compiler properties

12.26. Click **Compiler properties** () to open the dialog “Project properties” tab “Compiler”.

13.3. Compiler options

IDEA provides the following groups of compiler options:

- **13.3.1. General** (page 289)
- **13.3.2. Javac-specific** (page 292)

13.3.1. General

General compiler options include the following:

- **13.3.1.1. Exclude** (page 289)
- **13.3.1.2. Choose compiler** (page 290)
- **13.3.1.3. Resource patterns** (page 290)
- **13.3.1.4. Compile in background** (page 291)
- **13.3.1.5. Synchronize output directory** (page 291)

13.3.1.1. Exclude

The following can be excluded from compilation:

- **13.3.1.1.1. File** (page 289)
- **13.3.1.1.2. Directory (recursively)** (page 289)

13.3.1.1.1. File

12.27. Open the dialog “Project properties” tab “Compiler”.

12.28. Click **Add file**. The dialog “Select Path” appears.

12.29. Double-click on **C:\MyProjectFolder\src\MyPackage\MyClass3.java**. The file is excluded.

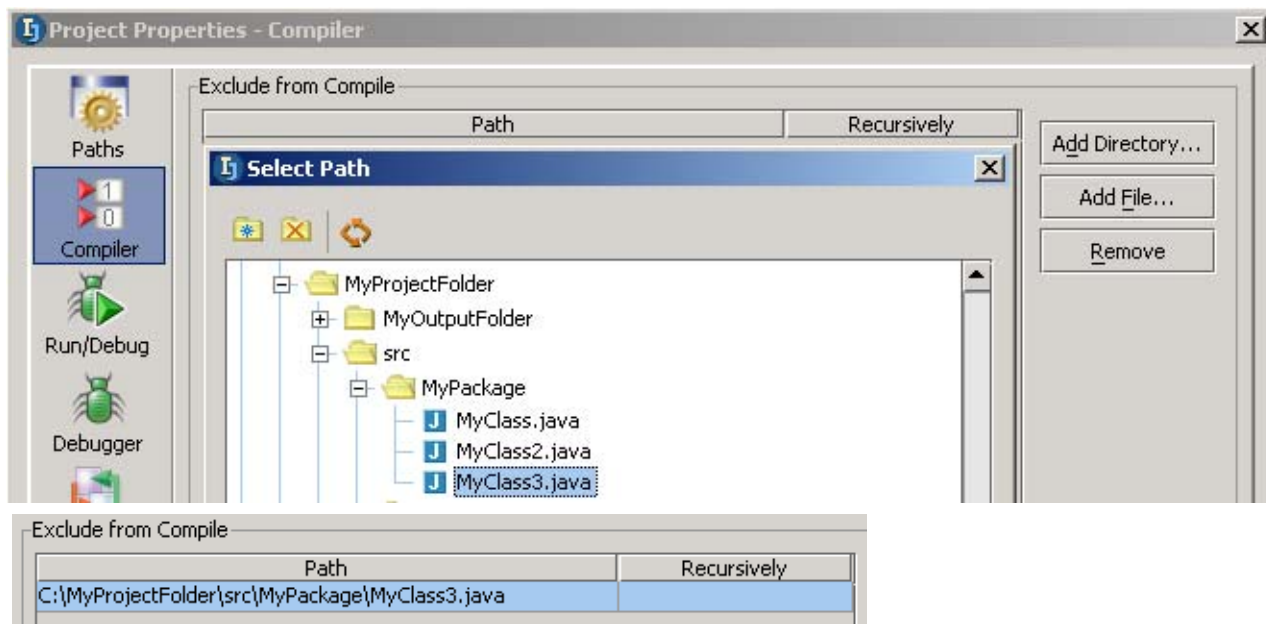


Figure 12.9. Exclude file (980,981)

The excluded file will not be compiled (even in a build).

12.30. Remove the file from the exclude list.

13.3.1.1.2. Directory (recursively)

12.31. Open the dialog “Project properties” tab “Compiler”.

12.32. Click **Add directory**. The dialog “Select Path” appears.

12.33. Double-click on **C:\MyProjectFolder\src\MyPackage**. The directory is excluded.

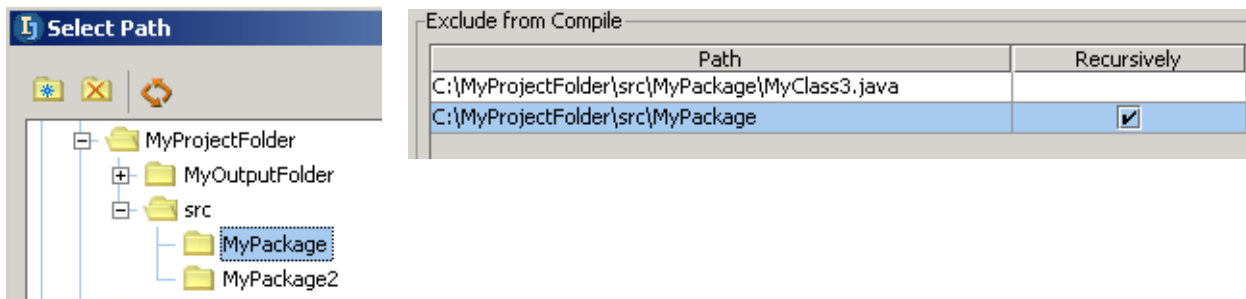


Figure 12.10. Exclude directory (983.982)

The files in the directory will not be compiled (even in a build).

If checkbox **Recursively** is checked, then the files in subdirectories of MyPackage will also not be compiled.

12.34. Remove the directory from the exclude list.

13.3.1.2. Choose compiler

12.35. Select tab **Jikes**.

12.36. Click **Set active**. Jikes is now the active compiler.

12.37. Select tab **Javac**.

12.38. Click **Set active**. Javac is now the active compiler.

13.3.1.3. Resource patterns

Files matching a resource pattern will be copied to the output.

Note: Make sure that the file and directory above were removed from the exclude list.

12.39. Create class **MyClass**:

```
package MyPackage;
import java.util.ResourceBundle;
import java.util.PropertyResourceBundle;
import java.io.IOException;
public class MyClass {
    public static void main(String[] args) {
        try {
            ResourceBundle res = new PropertyResourceBundle (
                MyClass.class.getResourceAsStream("messages.txt"));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

12.40. Create file **messages.txt**:

```
error.e1=Error 1 text
error.e2=Error 2 text
```

12.41. Open the dialog “Project properties” tab “Compiler”.

12.42. For “Resource patterns:” add the following text in bold:

```
.\. (properties|xml|html|txt); .+\. (gif|png|jpeg)
```

12.43. Click **OK**.

12.44. Build the project. Note that messages.txt was copied to the output.

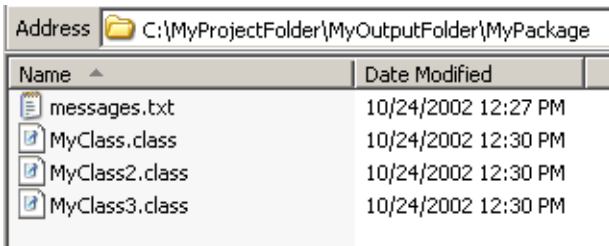


Figure 12.11. Resource copied to output (984)

13.3.1.4. Compile in background

12.45. Open the dialog “Project properties” tab “Compiler”.

12.46. Check the checkbox **Compile in background**. Compilations will now occur in the background.

This is convenient when compiling large projects. The dialog “Compile progress” will not be shown.

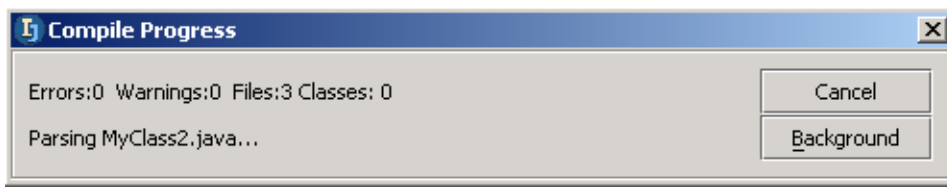


Figure 12.12. Compile progress (985)

12.47. Uncheck **Compile in background**.

13.3.1.5. Synchronize output directory

12.48. Open the dialog “Project properties” tab “Compiler”.

12.49. Check the checkbox **Synchronize output directory**.

12.50. Delete **MyClass2**.

12.51. Delete **MyClass3**.

12.52. **Build** the project. Note that the class files for MyClass2 and MyClass3 were deleted.

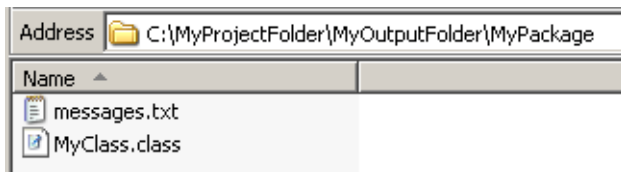


Figure 12.13. Synchronized output directory (986)

13.3.2. Javac-specific

Javac-specific compiler options include the following:

- [13.3.2.1. Generate debug info \(page 292\)](#)
- [13.3.2.2. Report uses of deprecated features \(page 292\)](#)
- [13.3.2.3. Generate no warnings \(page 293\)](#)
- [13.3.2.4. Additional Javac command parameters \(page 293\)](#)
- [13.3.2.5. Max heap size \(page 294\)](#)

13.3.2.1. Generate debug info

12.53. Open the dialog “Project properties” tab “Compiler”.

12.54. In tab “Javac (active)” uncheck **Generate debugging info**.

12.55. Click **OK**.

12.56. Add a breakpoint at

```
ResourceBundle res = new PropertyResourceBundle
```

12.57. Build the project.

12.58. Debug the application. Note that at the breakpoint no debug info is available.

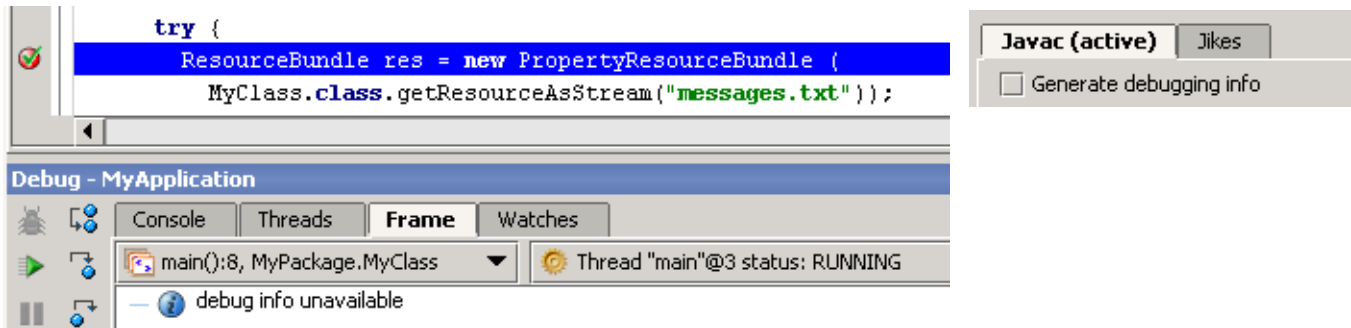


Figure 12.14. No debug info available [\(987,988\)](#)

12.59. Open the dialog “Project properties” tab “Compiler”.

12.60. In tab “Javac (active)” check **Generate debugging info**.

12.61. Click **OK**.

12.62. Build the project.

12.63. Debug the application. Note that at the breakpoint debug info is available.

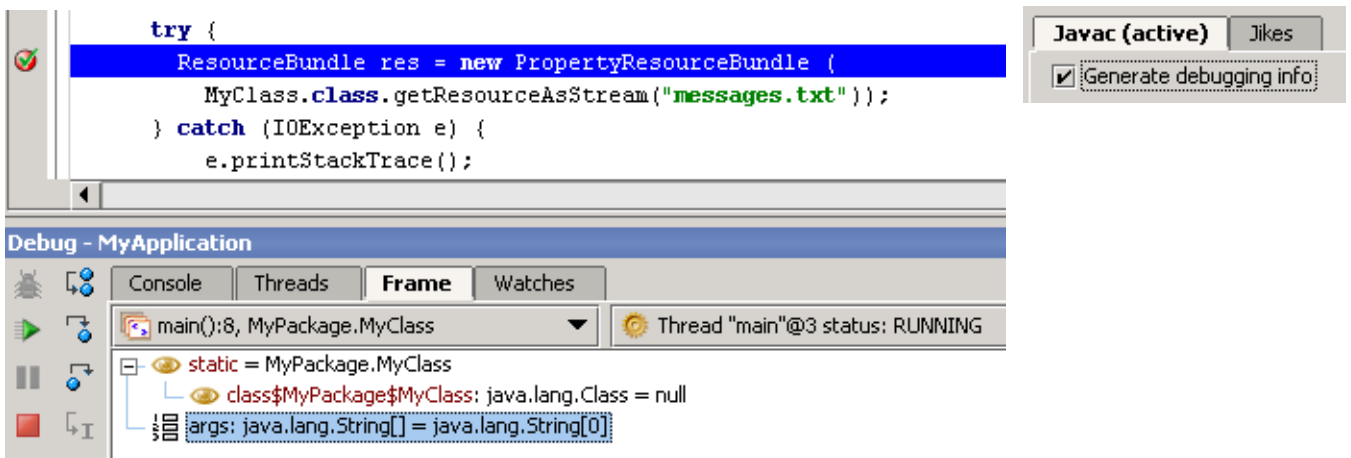


Figure 12.15. Debug info available [\(990,989\)](#)

13.3.2.2. Report uses of deprecated features

12.64. Create class **MyClass**

```
package MyPackage;
```

```
public class MyClass {  
    public void aMethod() {  
        Thread.currentThread().resume();  
    }  
}
```

- 12.65. Open the dialog “Project properties” tab “Compiler”.
- 12.66. In tab “Javac (active)” check **Generate debugging info**.
- 12.67. In tab “Javac (active)” check **Report uses of deprecated features**.
- 12.68. In tab “Javac (active)” uncheck **Generate no warnings**.
- 12.69. Click **OK**.
- 12.70. Compile. Note the warning.

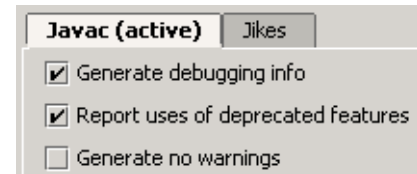
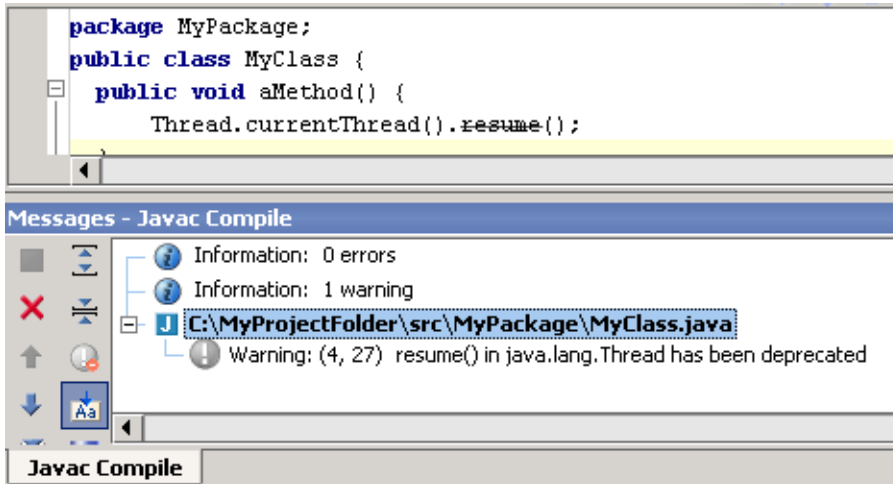


Figure 12.16. Deprecation warning ([992.991](#))

13.3.2.3. Generate no warnings

- 12.71. Open the dialog “Project properties” tab “Compiler”.
- 12.72. In tab “Javac (active)” check **Generate debugging info**.
- 12.73. In tab “Javac (active)” check **Report uses of deprecated features**.
- 12.74. In tab “Javac (active)” check **Generate no warnings**.

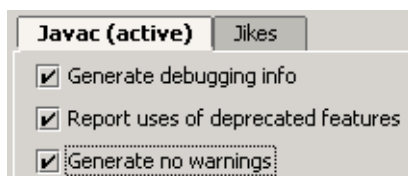


Figure 12.17. No warnings ([993](#))

- 12.75. Click **OK**.
- 12.76. Compile. Note that no warning is generated.

13.3.2.4. Additional Javac command parameters

12.77. Create class **MyClass**

```
package MyPackage;  
public class MyClass {  
    public static void main(String[] args) {  
        System.out.println("hello");  
    }  
}
```

- 12.78. Open the dialog “Project properties” tab “Compiler”.
- 12.79. In tab “Javac (active)” check **Generate debugging info**.

- 12.80. In tab “Javac (active)” check **Report uses of deprecated features**.
- 12.81. In tab “Javac (active)” uncheck **Generate no warnings**.
- 12.82. For “Additional Javac command line parameters:” enter **-g:none**.
- 12.83. Click **OK**.
- 12.84. Build the project.
- 12.85. Add a breakpoint to line
`System.out.println("hello");`
- 12.86. Debug. Note that execution does not stop at the breakpoint.

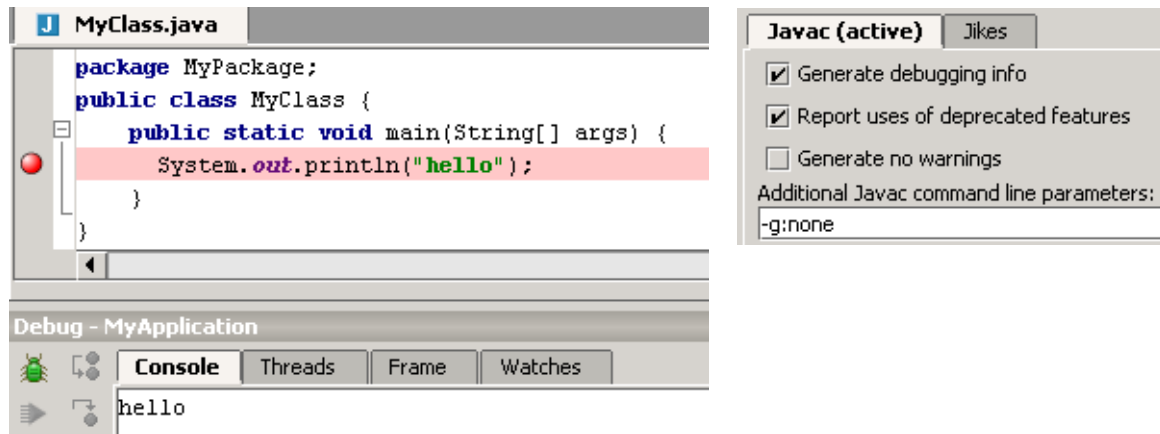


Figure 12.18. No breakpoint stop (996.995)

- 12.87. Open the dialog “Project properties” tab “Compiler”.
- 12.88. For “Additional Javac command line parameters:” leave blank.
- 12.89. Click **OK**.
- 12.90. Build the project.
- 12.91. Debug. Note that execution does not stop at the breakpoint.

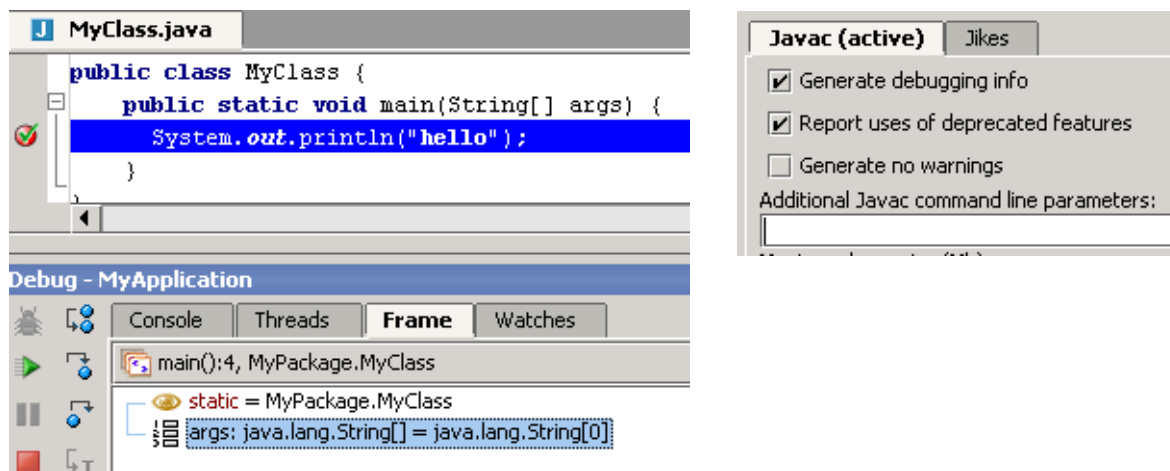


Figure 12.19. Breakpoint stop (998.997)

13.3.2.5. Max heap size

The maximum heap size should be enlarged if the compiler runs out of memory.

14. Ant

[20021023TT: last edit.](#)

[contacts: zheka](#)

IDEA supports integration with Jakarta Ant. This chapter describes how to

- [14.1. Download / Install \(page 295\)](#)
- [14.2. Create build.xml \(page 297\)](#)
- [14.3. Add Ant build to IDEA \(page 298\)](#)
- [14.4. Run build \(page 299\)](#)

14.1. Download / Install

Refer to <http://jakarta.apache.org/ant/index.html> for the latest information on Ant.

13.1. Download from <http://jakarta.apache.org/builds/jakarta-ant/release/v1.5/bin/> file **jakarta-ant-1.5-bin.zip**.

13.2. Extract to **c:** (use folder names).

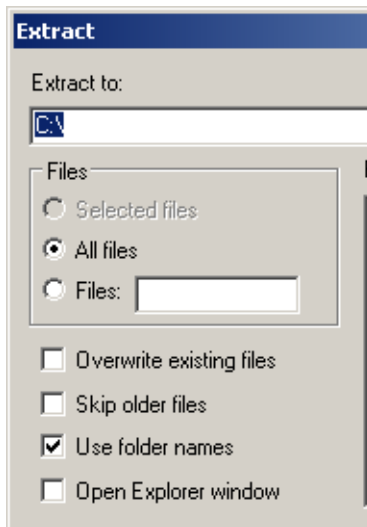


Figure 13.1. Ant extraction [\(938\)](#)

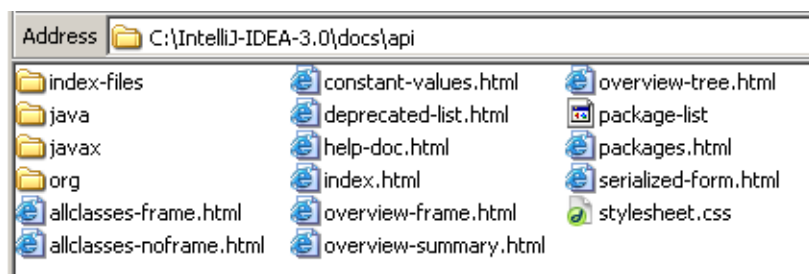


Figure 13.2. JavaDoc API dir [\(313\)](#)

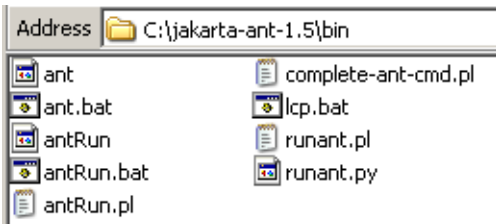


Figure 13.3. Ant directories (273)

13.3. Add environment variable **ANT_HOME** with value **C:\jakarta-ant-1.5**.

13.4. Add to environment variable **Path** dir **C:\jakarta-ant-1.5\bin**.

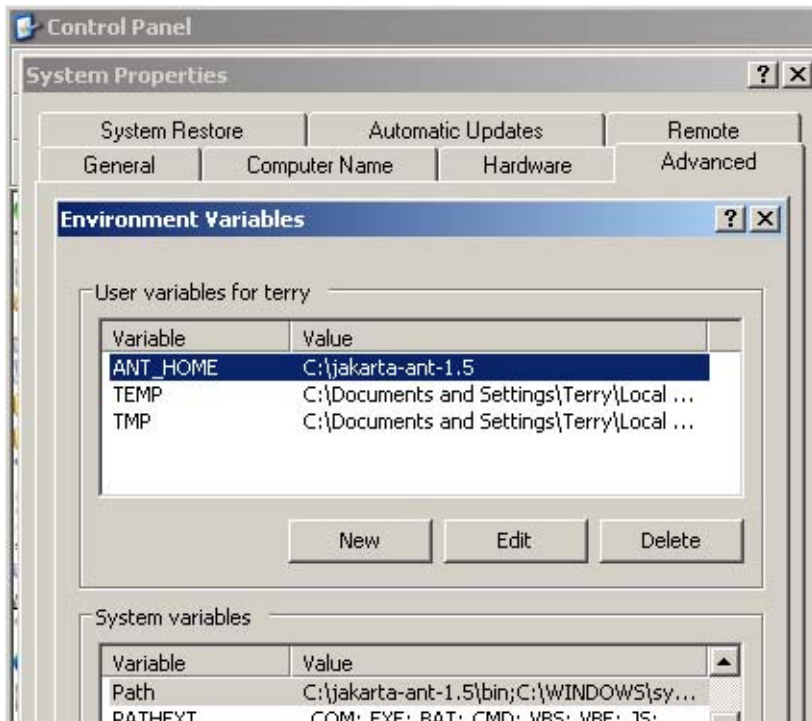


Figure 13.4. Environment variables for Ant (272)

14.2. Create build.xml

13.5. Create file C:\MyProjectFolder\build.xml:

```
<project name="MyAntProject" default="all" basedir=".">
<target name="init">
  <property name = "class_path" value = "c:\MyProjectFolder" />
  <property name = "build"      value = "${class_path}\MyOutputFolder" />
  <property name = "source"     value = "${class_path}\src" />
</target>
<target name="all" depends="init">
  <mkdir dir="${build}" />
  <javac srcdir   = "${source}"
        destdir  = "${build}"
        classpath = "${class_path}" />
</target>
</project>
```



Figure 13.5. build.xml (271)

14.3. Add Ant build to IDEA

- 13.6. Open the Ant Build tool.
- 13.7. Click on **+**. The dialog “Select Ant build” appears.
- 13.8. Select build.xml.

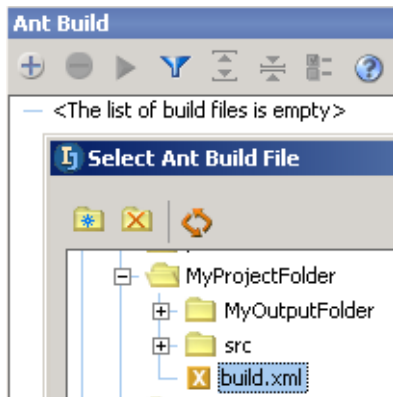


Figure 13.6. Adding build.xml to Ant tool [\(270\)](#)

- 13.9. Click **OK**. The file is added.

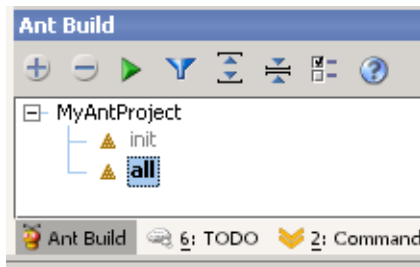


Figure 13.7. Build file added [\(269\)](#)

14.4. Run build

13.10. Click on the **Run** icon. The files are built.

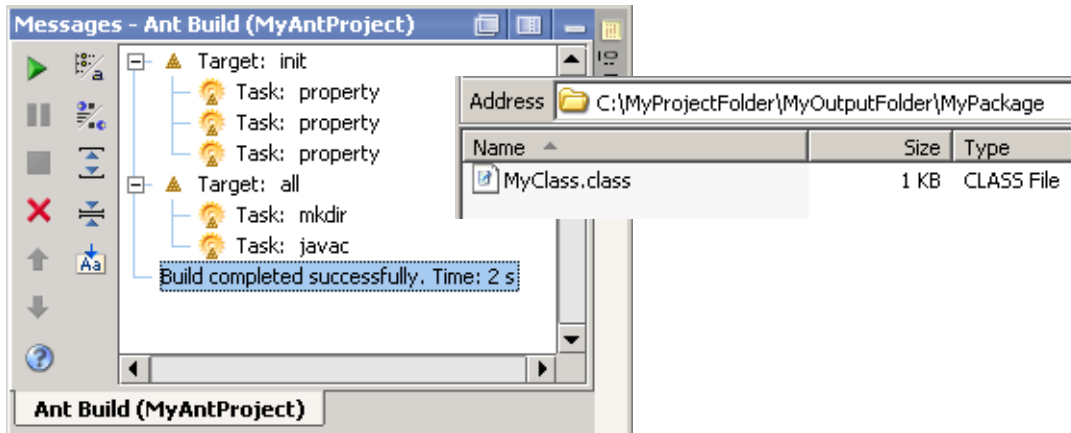


Figure 13.8. Files are built ([268.267](#))

15. Debugger

[20021023TT: last edit](#)

[contacts: zheka](#)

[Add: remote debugging.](#)

The chapter describes

- **15.1. Run/Debug configurations (page 301)**
- **15.2. Types of breakpoints / watchpoints (page 304)**
- **15.3. Run / debug actions (page 308)**

Note: This chapter only describes in detail running/debugging for an Application. The particular details of running/debugging for other run/debug configurations (Applet, WebApp, etc.) are described in ~~Part E. Applications (page 317).~~

15.1. Run/Debug configurations

This section describes the run/debug configuration

- **15.1.1. Types (page 301)**
- **15.1.2. Basic operations (page 302)**

15.1.1. Types

The following 5 types of configurations are available

- **Application** (you already created a basic application configuration in **Section 3.10.1. Create application configuration (page 54)**)
- **Applet**
- **JUnit**
- **Remote** (debug configuration ONLY)
- **WebApp**
- ~~15.1.1.1. Application (page 301)~~
- ~~15.1.1.2. Applet (page 301)~~
- ~~15.1.1.3. JUnit (page 301)~~
- ~~15.1.1.4. Remote XXX ?? (page 301) (debug configuration ONLY)~~
- ~~15.1.1.5. WebApp (page 301)~~

15.1.1.1. Application

You already created a basic application configuration in ~~Section 3.10.1. Create application configuration (page 54).~~

15.1.1.2. Applet

Will be introduced in ~~Chapter 19. Applets XXX (page 321).~~

15.1.1.3. JUnit

Will be introduced in ~~Chapter 16. JUnit XXX (page 313).~~

15.1.1.4. Remote XXX ??

15.1.1.5. WebApp

Will be introduced in ~~Chapter 20. Web applications XXX (page 323).~~

15.1.2. Basic operations

The following operations are supported for run/debug configurations:

- 15.1.2.1. Add (create) (page 302)
- 15.1.2.2. Edit (page 302)
- 15.1.2.3. Modify defaults (page 302)
- 15.1.2.4. Copy (page 303)
- 15.1.2.5. Remove (delete) (page 303)
- 15.1.2.6. Run (page 303)
- 15.1.2.7. Debug (page 303)

15.1.2.1. Add (create)

You created a basic application configuration in [Section 3.10.1. Create application configuration \(page 54\)](#).

15.1.2.2. Edit

14.1. Click on the **Run** or **Debug** icon ( ). The dialog appears.

14.2. Make the required changes.

14.3. Click **Apply**.

15.1.2.3. Modify defaults

14.4. Click **Edit Defaults**. The “Application default settings” page appears.

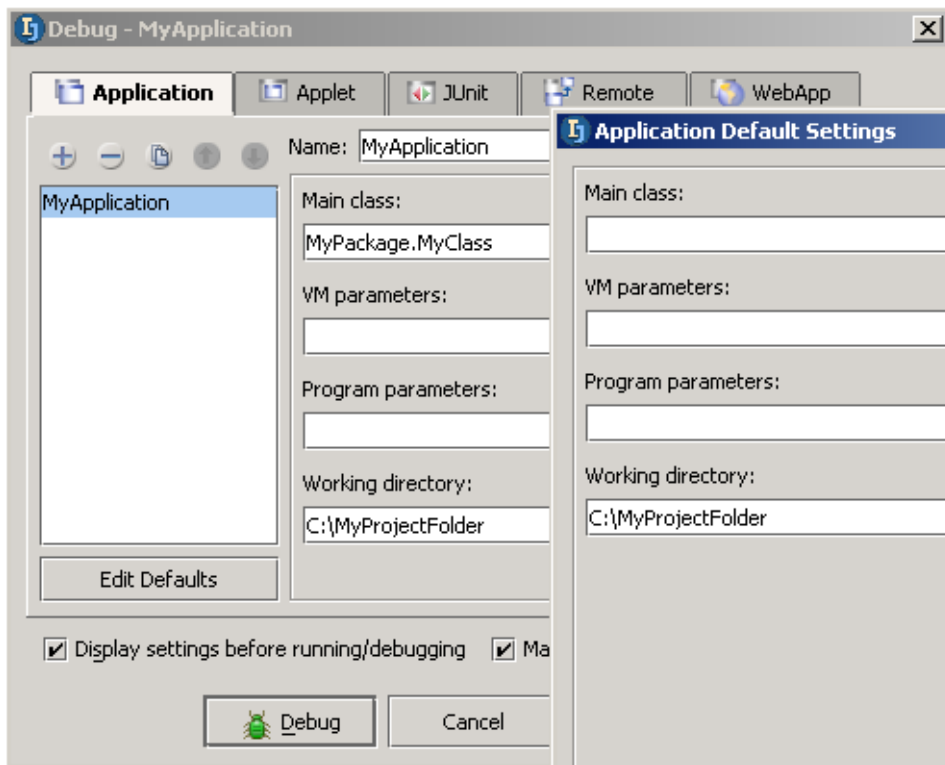


Figure 14.1. Application default settings ([289](#))

14.5. Enter defaults.

14.6. Click **OK**.

15.1.2.4. Copy

14.7. Select the configuration.

14.8. Click on the **Copy configuration** icon (). A copy of the configuration is created.

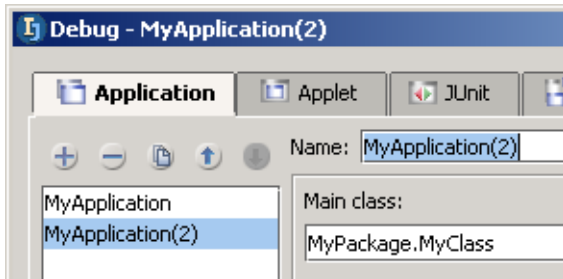


Figure 14.2. Copied configuration (287)

15.1.2.5. Remove (delete)

14.9. Select the configuration.

14.10. Click on the **Remove configuration** icon (). A confirmation dialog appears.

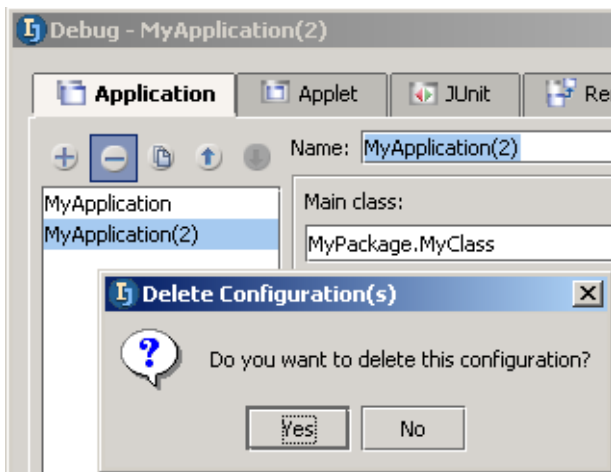


Figure 14.3. Delete configuration confirmation (940)

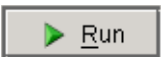
14.11. Click **Yes**. The configuration is deleted.

15.1.2.6. Run

14.12. Close the configuration dialog.

14.13. Click on the **Run** icon (). The Run configuration dialog appears.

14.14. Select a configuration.

14.15. Click **Run** (). The application is run.

15.1.2.7. Debug

14.16. Close the configuration dialog.

14.17. Click on the **Debug** icon (). The Debug configuration dialog appears.

14.18. Select a configuration.

14.19. Click **Debug** (). The application is debugged (details of debugging an application are introduced later in this chapter).

15.2. Types of breakpoints / watchpoints

[eventually make better example code here.](#)

The following types of breakpoints / watchpoints exist:

- [15.2.1. Line \(page 304\)](#)
- [15.2.2. Exception \(page 304\)](#)
- [15.2.3. Field \(watchpoints\) \(page 306\)](#)
- [15.2.4. Method \(page 307\)](#)

15.2.1. Line

You created a line breakpoint in [section 3.10.3. Set breakpoint \(page 57\)](#).

15.2.2. Exception

14.20. Create class **MyClass**:

```
package MyPackage;
public class MyClass {
    public static void main(String[] args) {
        MyClass2 mc2 = new MyClass2();
        mc2.aMethod();
    }
}
class MyClass2 {
    int anInt = 1;
    void aMethod() throws ArithmeticException {
        throw new ArithmeticException();
    }
}
```

14.21. Debug the class. Note the exception:

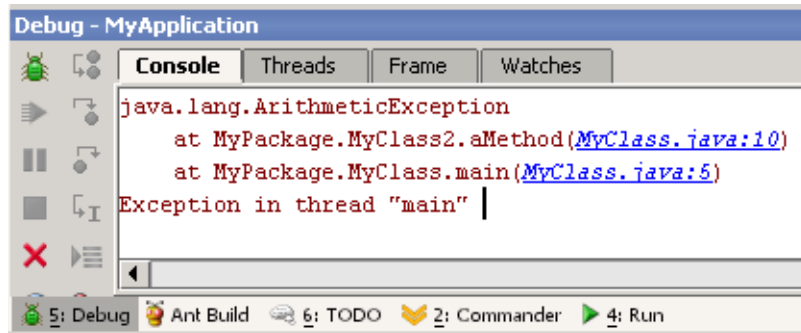


Figure 14.4. Exception [\(941\)](#)

14.22. Close the debug tool.

14.23. Select **Run | View breakpoints....**

14.24. Select tab **Exception breakpoints.**

14.25. Click **Add**. The dialog "Enter exception class" appears.

14.26. For the name enter **ArithmeticException**.

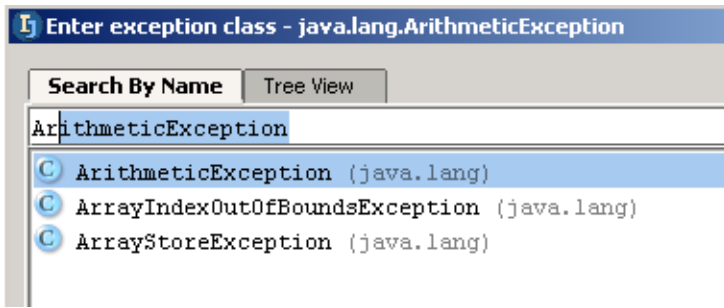


Figure 14.5. Enter Arithmetic exception (942)
14.27. Click **OK**. The exception breakpoint is added.

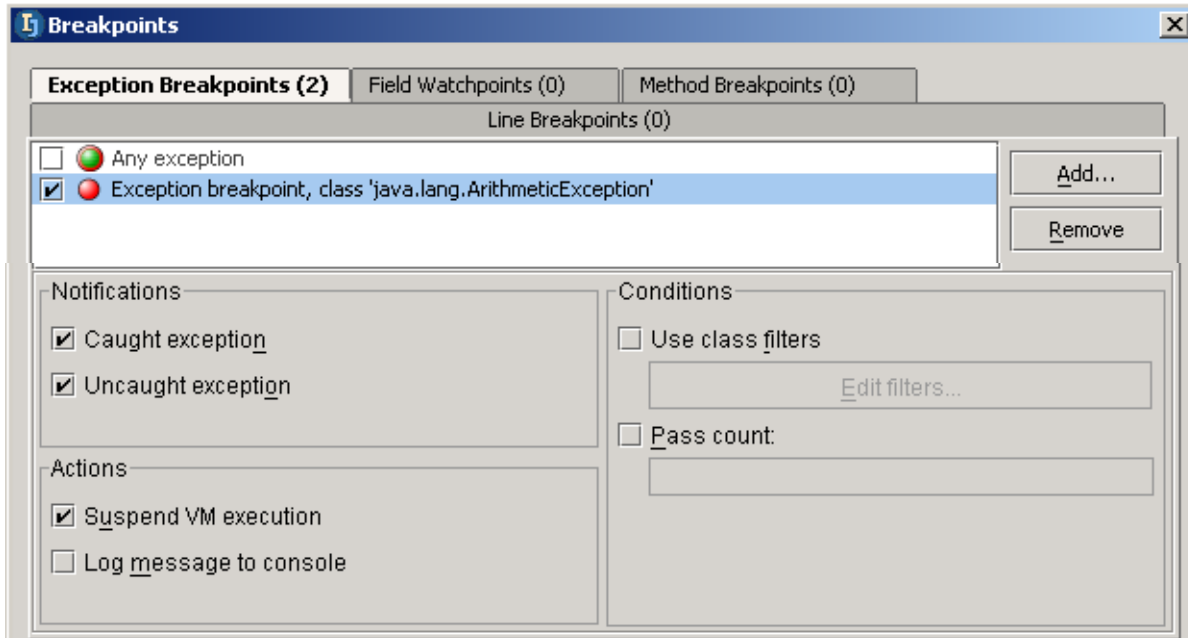


Figure 14.6. Exception breakpoint (282,281)
14.28. Click **Close**.
14.29. Debug the application. Note the debug window.

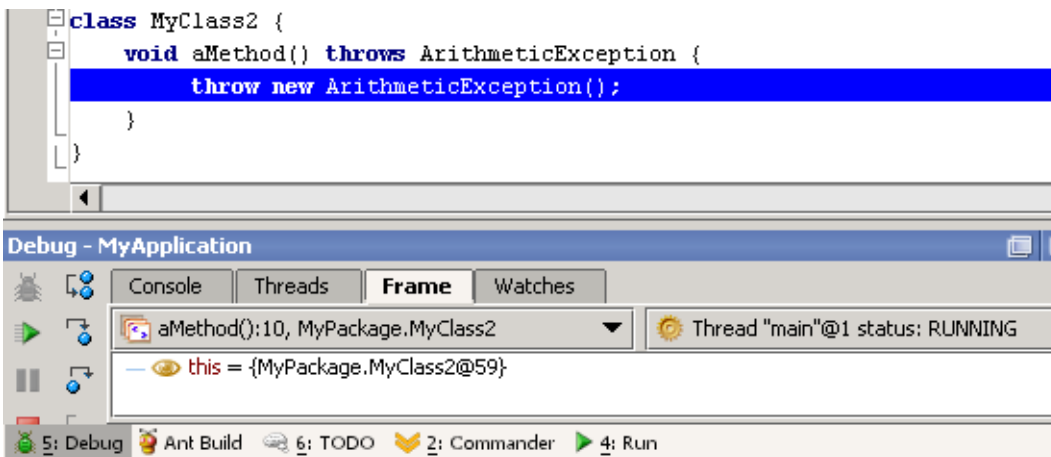


Figure 14.7. Exception breakpoint (943)

15.2.3. Field (watchpoints)

- 14.30. Delete the exception breakpoint.
- 14.31. Select tab **Field watchpoints**.
- 14.32. Click **Add**. The dialog “Add field watchpoint” appears.
- 14.33. For “Fully qualified name of a class” enter **MyPackage.MyClass2**.
- 14.34. For “Field name” enter **anInt**.

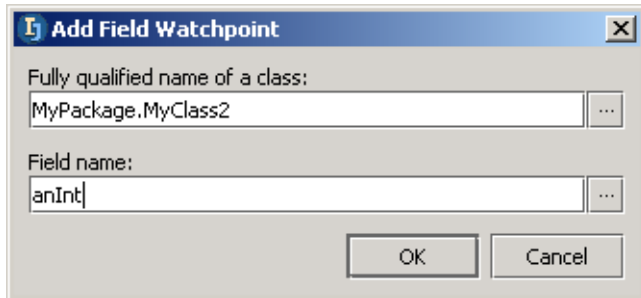


Figure 14.8. Add field watchpoint (279)

- 14.35. Click **OK**. The field watchpoint is added.

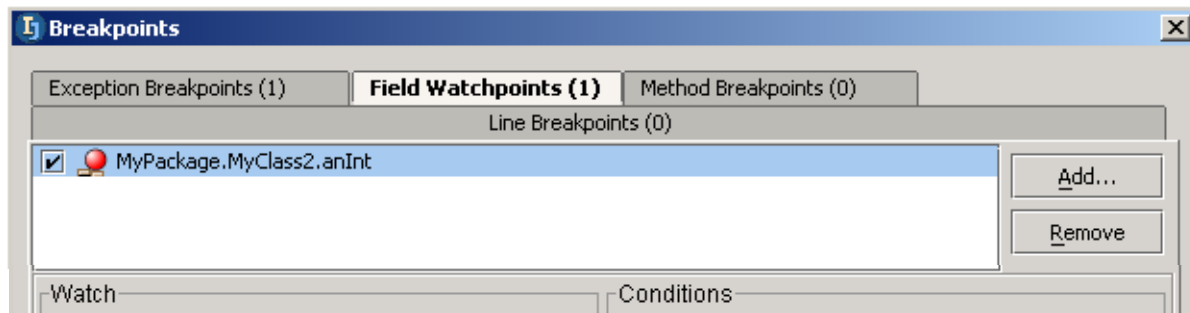


Figure 14.9. Field watchpoint added (278,277)

- 14.36. Click **Close**.
- 14.37. Debug the application. Note the debug window.

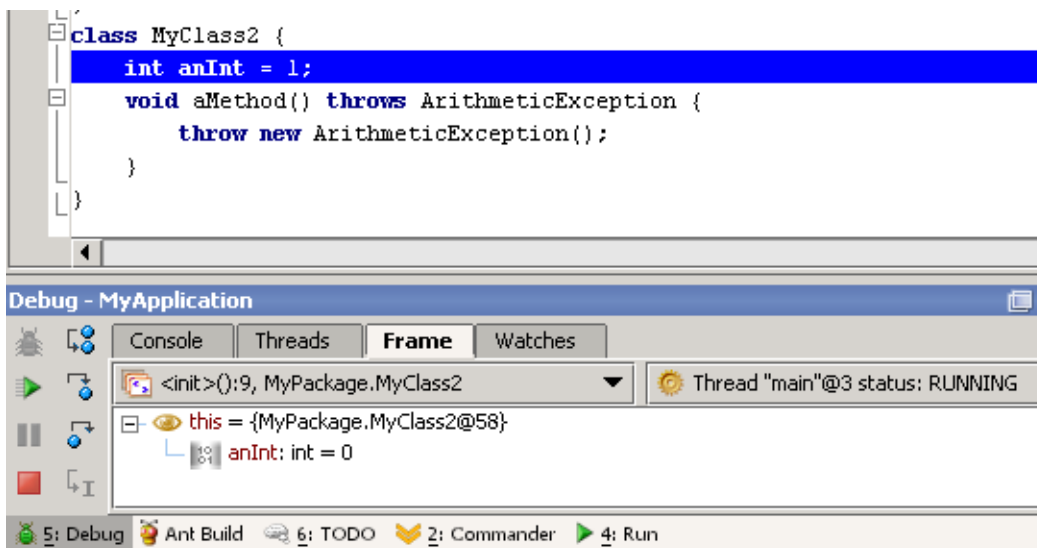


Figure 14.10. Field watchpoint info in debug (276)

15.2.4. Method

14.38. Delete the field watchpoint.

14.39. Place the cursor within the definition for **aMethod()**.

14.40. Select **Run | Add method breakpoint...**. The dialog “Add method breakpoint” appears with the breakpoint defined.

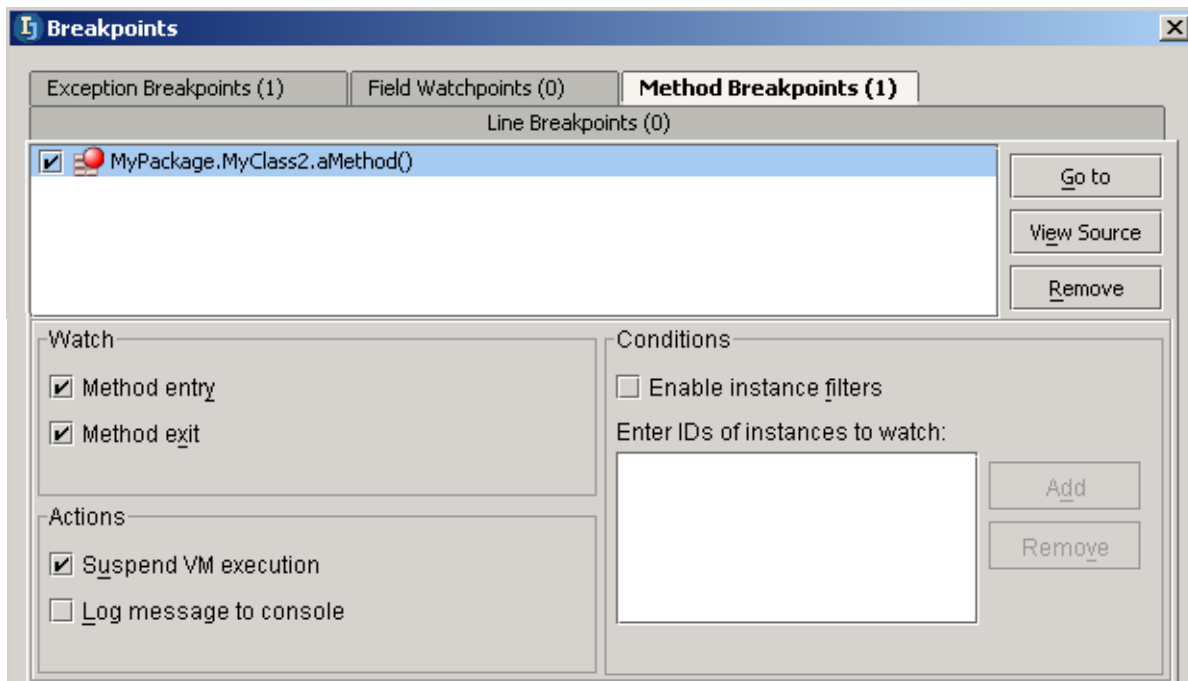


Figure 14.11. Method breakpoint [\(275,274\)](#)

14.41. Click **Close**.

14.42. Debug the application.

15.3. Run / debug actions

The following run / debug actions are available:

- 15.3.1. **Pause / Resume / Stop / Close** (page 308)
- 15.3.2. **Step over / Step into / Step out** (page 309)
- 15.3.3. **Run to cursor** (page 311)
- 15.3.4. **Show execution point** (page 311)
- 15.3.5. **View breakpoints** (page 311)

15.3.1. Pause / Resume / Stop / Close

14.43. Create class **MyClass**:

```
package MyPackage;  
public class MyClass {  
    public static void main(String[] args) {  
        for (int i = 0; i < 1000000; i++) {  
            System.out.println(i);  
            try { Thread.sleep(100);  
            } catch (InterruptedException ie) {}  
        }  
    }  
}
```

14.44. Run the application.

14.45. Click **Pause** (). Execution is paused.

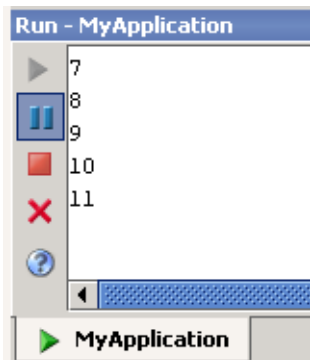


Figure 14.12. Pause (956)

14.46. Click **Pause** () again to continue.

14.47. Click **Stop** () to stop.

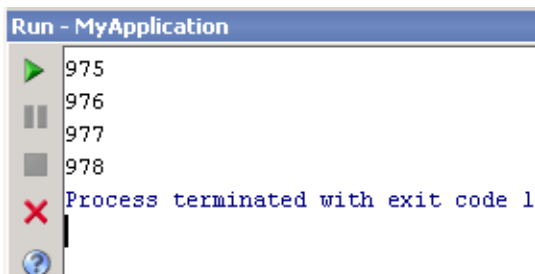


Figure 14.13. Stop (958)

14.48. Click **Close** () to close.

15.3.2. Step over / Step into / Step out

14.49. Create class **MyClass**:

```
package MyPackage;
public class MyClass {
    public static void main(String[] args) {
        MyClass2 mc2 = new MyClass2();
        mc2.aMethod2();
        mc2.aMethod2();
        mc2.aMethod2();
    }
}
class MyClass2 {
    int anInt2 = 1;
    void aMethod2() {
        anInt2 = 2;
    }
}
```

14.50. Set a line breakpoint for the first

```
mc2.aMethod2();
```

14.51. Debug the class. Execution stops at the breakpoint.

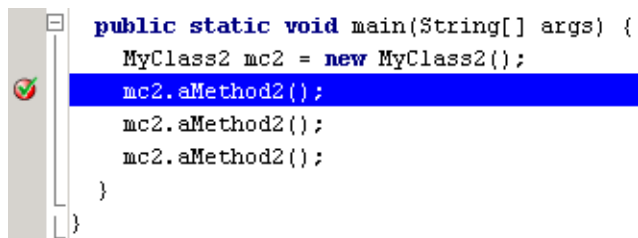


Figure 14.14. Breakpoint (945)

14.52. Click **Step over** (). The method is stepped over.

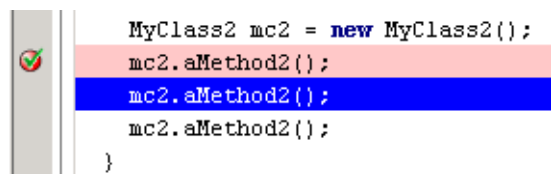
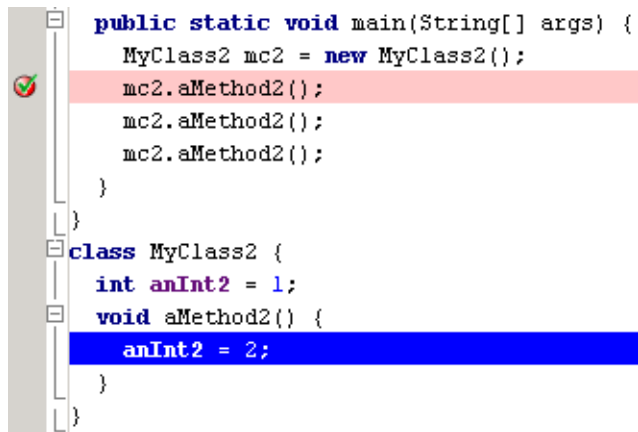


Figure 14.15. Step over (947)

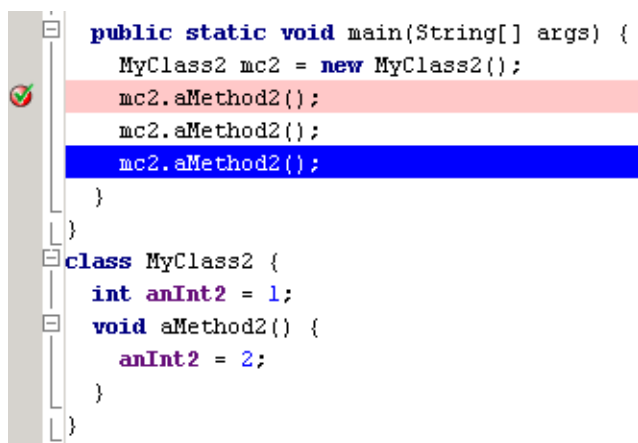
14.53. Click **Step into** (). The method is stepped into.



```
public static void main(String[] args) {  
    MyClass2 mc2 = new MyClass2();  
    mc2.aMethod2();  
    mc2.aMethod2();  
    mc2.aMethod2();  
}  
  
class MyClass2 {  
    int anInt2 = 1;  
    void aMethod2() {  
        anInt2 = 2;  
    }  
}
```

Figure 14.16. Step into (949)

14.54. Click **Step out** (). The method is stepped out of.



```
public static void main(String[] args) {  
    MyClass2 mc2 = new MyClass2();  
    mc2.aMethod2();  
    mc2.aMethod2();  
    mc2.aMethod2();  
}  
  
class MyClass2 {  
    int anInt2 = 1;  
    void aMethod2() {  
        anInt2 = 2;  
    }  
}
```

Figure 14.17. Step out (951)

15.3.3. Run to cursor

- 14.55. Terminate the process.
- 14.56. Debug again. Execution stops at the breakpoint.
- 14.57. Place the cursor (caret) on the 3rd (last)

```
mc2.aMethod2();
```

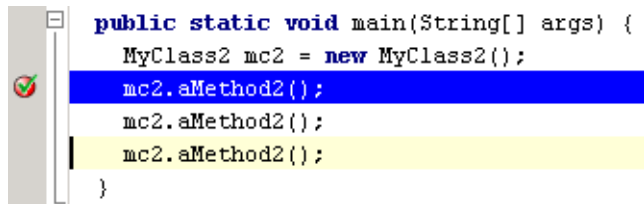


Figure 14.18. Cursor on 3rd method (952)

- 14.58. Click **Run to cursor** (). Execution continues to the cursor location.

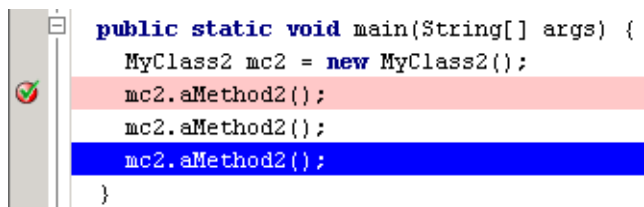


Figure 14.19. Run to cursor (954)

15.3.4. Show execution point

- 14.59. Close the editor for **MyClass.java**.
- 14.60. Click **Show execution point** (). The execution point is shown.

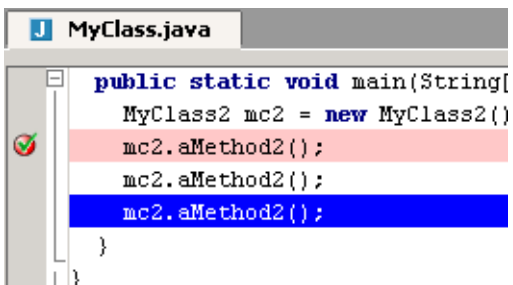


Figure 14.20. Show execution point (961)

15.3.5. View breakpoints

- 14.61. Click **View breakpoints** (). The dialog “Breakpoints” appears.

16. JUnit XXX

[contacts: zheka](#)

15.1.

16.1. xxx

Download from <http://www.junit.org/index.htm>.

Unzip.

Set CLASSPATH.

Test

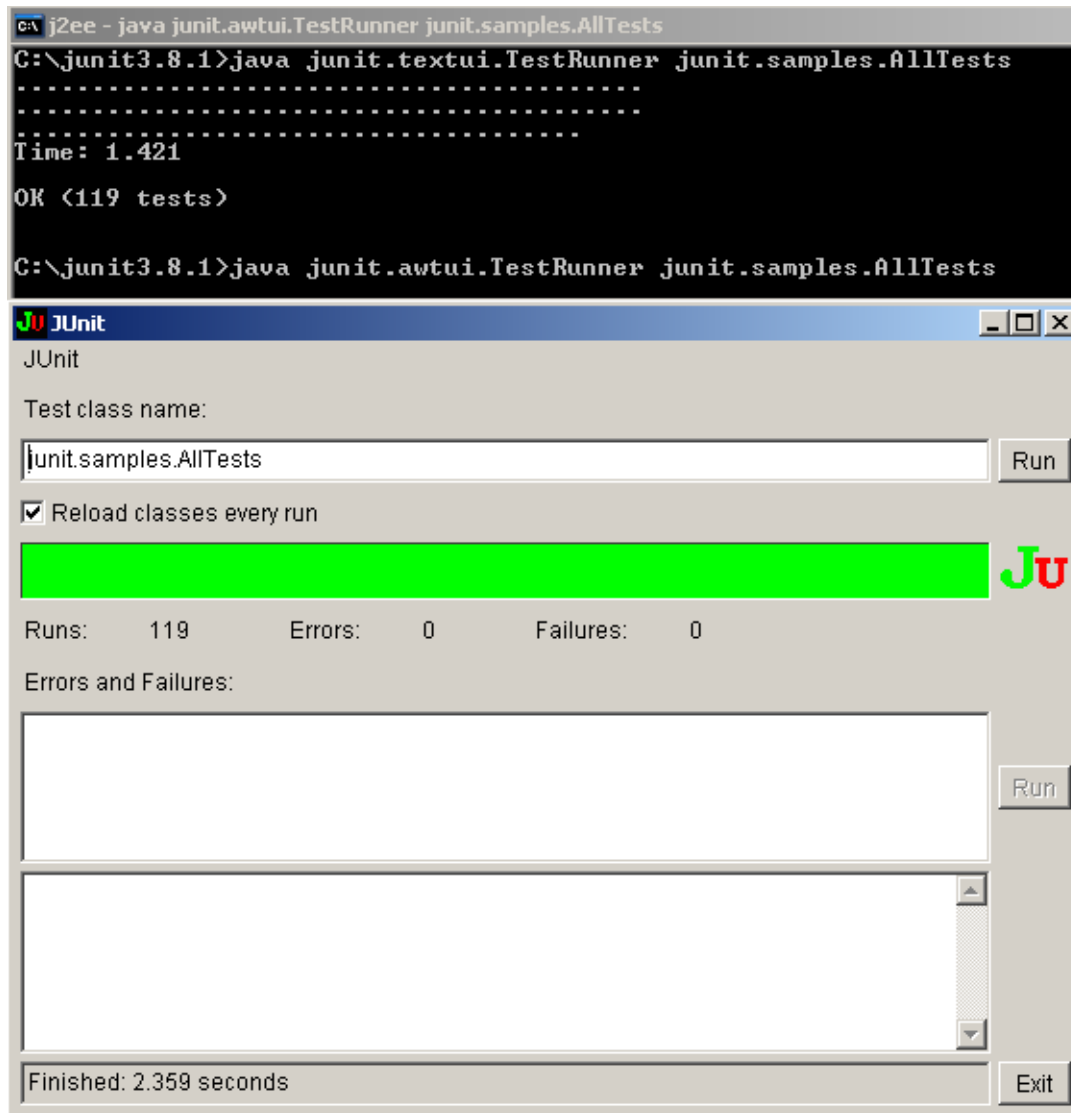


Figure 15.1. Junit test [\(472\)](#)

~~17. Remote debugging XXX~~

~~[contacts: ???](#)~~

~~17.1. xxx~~

~~16.1-~~

~~Figure 16.1. xxx (1)~~

Part E. Applications

The chapters in this part describe IDEA's support for various types of applications.

18. Applications XXX (page 319).

19. Applets XXX (page 321).

20. Web applications XXX (page 323). Demonstrates how web applications can be created and managed with IDEA.

21. EJB X (page 329). Demonstrates how to create EJB using IDEA.

18. Applications XXX

[contacts: valentin](#)

18.1. xxx

17.1.

Figure 17.1.

19. Applets XXX

[contacts: valentin](#)

19.1. xxx

18.1.

Figure 18.1.

20. Web applications XXX

[contacts: mike. and aleksei kudravtsefv](#)

19.1.

Figure 19.1.

- [20.1. Install Tomcat XXX \(page 323\)](#)
- [20.2. JSP XXX \(page 324\)](#)
- [20.3. Tags \(page 325\)](#)
- [20.4. Servlets XXX \(page 328\)](#)

20.1. Install Tomcat XXX

20.2. JSP XXX

20.3. Tags

[migrate??? to different jdk.... valya](#)
[20020911TTT: created.](#)

- **20.3.1. Create HelloTag.java (page 325)**
- **20.3.2. Create mytaglib.tld (page 326)**
- **20.3.3. Create Hello.jsp (page 326)**
- **20.3.4. Test (page 327)**

[example from http://developer.java.sun.com/developer/technicalArticles/xml/WebAppDev3/.](http://developer.java.sun.com/developer/technicalArticles/xml/WebAppDev3/)

20.3.1. Create HelloTag.java

19.2. Create C:\Program Files\Apache Tomcat 4.0\webapps\examples\WEB-INF\classes\tags\HelloTag.java with code:

```
package tags;

import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HelloTag implements Tag {
    private PageContext pageContext;
    private Tag parent;

    public HelloTag() {
        super();
    }

    public int doStartTag() throws JspException {
        try {
            pageContext.getOut().print(
                "This is my first tag!");
        } catch (IOException ioe) {
            throw new JspException("Error: IOException while writing to client"
+ ioe.getMessage());
        }
        return SKIP_BODY;
    }

    public int doEndTag() throws JspException {
        return SKIP_PAGE;
    }

    public void release() {
    }

    public void setPageContext(PageContext
pageContext) {
        this.pageContext = pageContext;
    }

    public void setParent(Tag parent) {
```

```
        this.parent = parent;
    }

    public Tag getParent() {
        return parent;
    }
}
```

19.3. Compile.

20.3.2. Create mytaglib.tld

19.4. Create **C:\Program Files\Apache Tomcat 4.0\webapps\examples\WEB-INF\jsp\mytaglib.tld** with code:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.1//EN"
"http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<!-- a tag library descriptor -->

<taglib>

    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>first</shortname>
    <uri></uri>
    <info>A simple tag library for the
examples</info>

    <tag>
        <name>hello</name>
        <tagclass>tags.HelloTag</tagclass>
        <bodycontent>empty</bodycontent>
        <info>Say Hi</info>
    </tag>
</taglib>
```

20.3.3. Create Hello.jsp

19.5. Create **C:\Program Files\Apache Tomcat 4.0\webapps\examples\jsp\Hello.jsp** with code:

```
<%@ taglib uri="/WEB-INF/jsp/mytaglib.tld"
    prefix="first" %>
<HTML>
<HEAD>
<TITLE>Hello Tag</TITLE>
</HEAD>

<BODY bgcolor="#ffffcc">

<B>My first tag prints</B>:

<first:hello/>

</BODY>
</HTML>
```

20.3.4. Test

19.6. In IE: Open <http://localhost:8080/examples/jsp/Hello.jsp>.

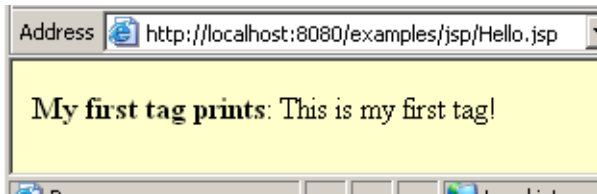


Figure 19.2. xxx [\(473\)](#)

20.4. Servlets XXX

[migrate??? to different jdk... valya](#)
[contacts: valya](#)

21. EJB X

CONSULT: [mike, aleksei kudravtsev.](#)

20020911TTT: [created. describes doing the sun tutorial example using IDEA. this chapter needs to be highly modified \(i really have no idea what i am doing\), but documents how to do the sun example.](#)

- [21.1. Install / start J2EE \(page 329\)](#)
- [21.2. Deploy Sun example application \(page 332\)](#)
- [21.3. Set IDEA EJB project properties \(page 337\)](#)
- [21.4. Modify source in IDEA \(page 340\)](#)
- [21.5. Redeploy \(page 341\)](#)

21.1. Install / start J2EE

- [21.1.1. Download / install J2EE from sun \(page 329\)](#)
- [21.1.2. Install \(page 329\)](#)
- [21.1.3. Start J2EE server \(page 330\)](#)
- [21.1.4. Start Cloudscape \(page 331\)](#)

21.1.1. Download / install J2EE from sun

20.1. Download [j2sdkee-1_3_1-win.exe](#) (Windows install) from <http://java.sun.com/j2ee/download.html#sdk>.

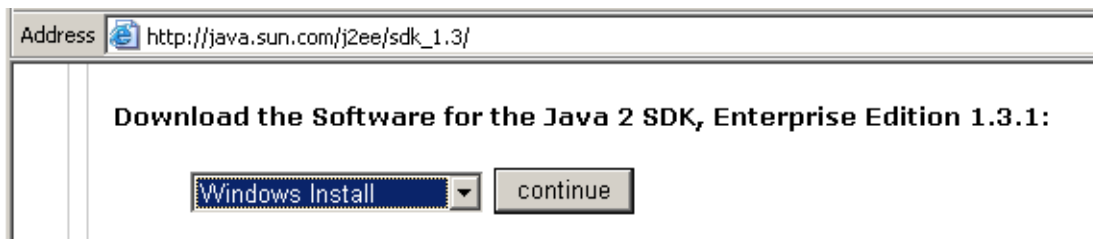


Figure 20.1. xxx [\(485\)](#)

21.1.2. Install

- 20.2. Double-click on [j2sdkee-1_3_1-win.exe](#). Follow the directions to install (use default settings).
- 20.3. Set JAVA_HOME system variable to
- 20.4. **Start | Settings | Control Panel.**
- 20.5. Double-click **System**.
- 20.6. In tab **Advanced**: Select **Environment variables**.
- 20.7. Add System variable **JAVA_HOME** with value **c:\jdk1.4.0_01** (or the location of your JDK if different).
- 20.8. Add System variable **J2EE_HOME** with value **C:\j2sdkee1.3.1**.

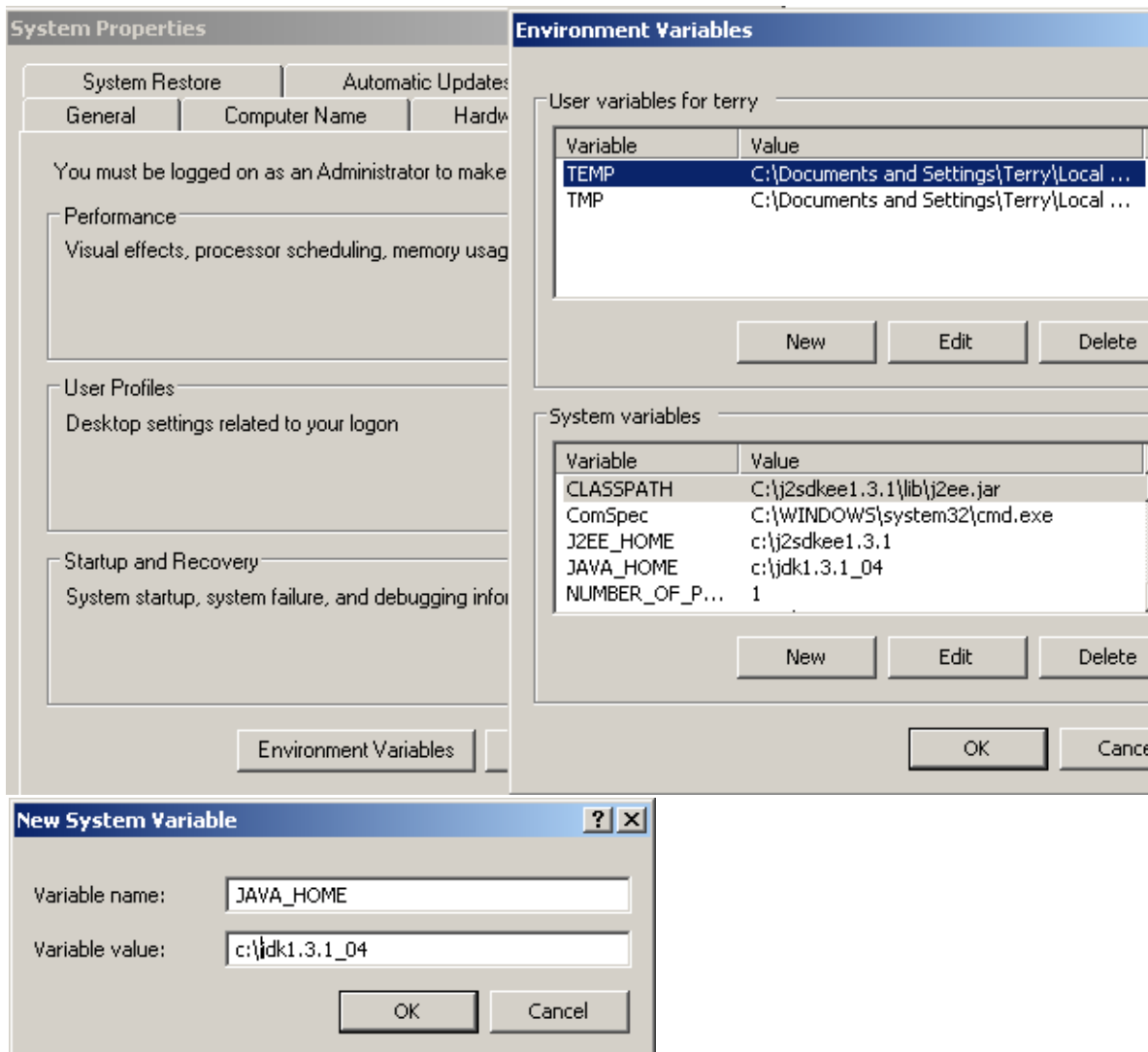


Figure 20.2. xxx (497,484)

21.1.3. Start J2EE server

20.9. Start J2EE server.

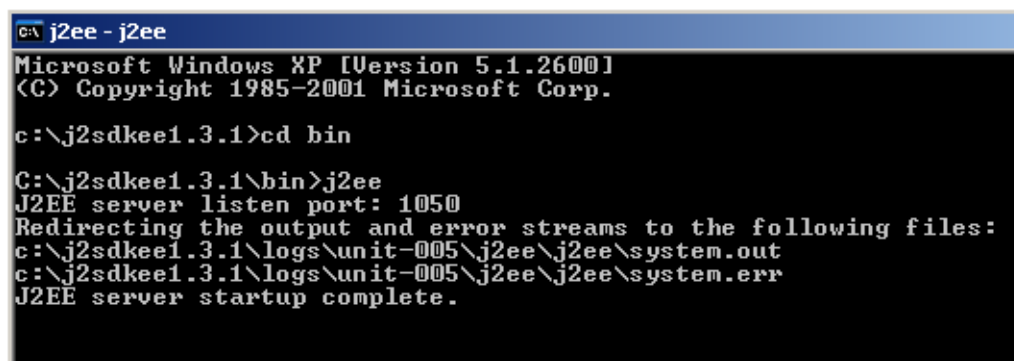


Figure 20.3. xxx (496)

20.10. Open <http://localhost:8000> to verify.

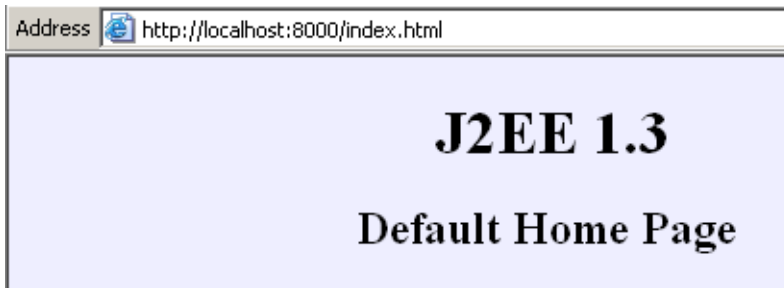


Figure 20.4. xxx (483)

21.1.4. Start Cloudscape

20.11. Start Cloudscape server.

```
C:\j2sdkee1.3.1\bin>cloudscape -start
Wed Sep 11 17:04:43 MSD 2002: [RmiJdbc] Starting Cloudscape RmiJdbc Server Version 1.7.2 ...
Wed Sep 11 17:04:43 MSD 2002: [RmiJdbc] COM.cloudscape.core.JDBCdriver registered in DriverManager
Wed Sep 11 17:04:43 MSD 2002: [RmiJdbc] Binding RmiJdbcServer...
Wed Sep 11 17:04:43 MSD 2002: [RmiJdbc] No installation of RMI Security Manager.
Wed Sep 11 17:04:43 MSD 2002: [RmiJdbc] RmiJdbcServer bound in rmi registry
```

Figure 20.5. xxx (495)

20.12. Open <http://???> to verify.

21.2. Deploy Sun example application

- 21.2.1. Start deploy tool (page 332)
- 21.2.2. Open the application (page 332)
- 21.2.3. Generate SQL (page 333)
- 21.2.4. Deploy (page 334)

21.2.1. Start deploy tool

20.13. Double-click on `C:\j2sdkee1.3.1\bin\deploytool.bat` to start the deploy tool.

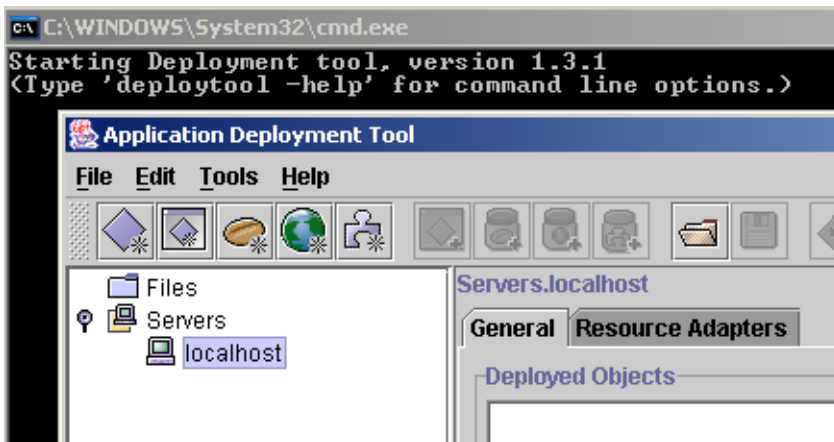


Figure 20.6. xxx (494)

21.2.2. Open the application

Select `File | Open`.

Select `C:\j2sdkee1.3.1\doc\samples\cmpcustomer\cmpcustomer.ear`.

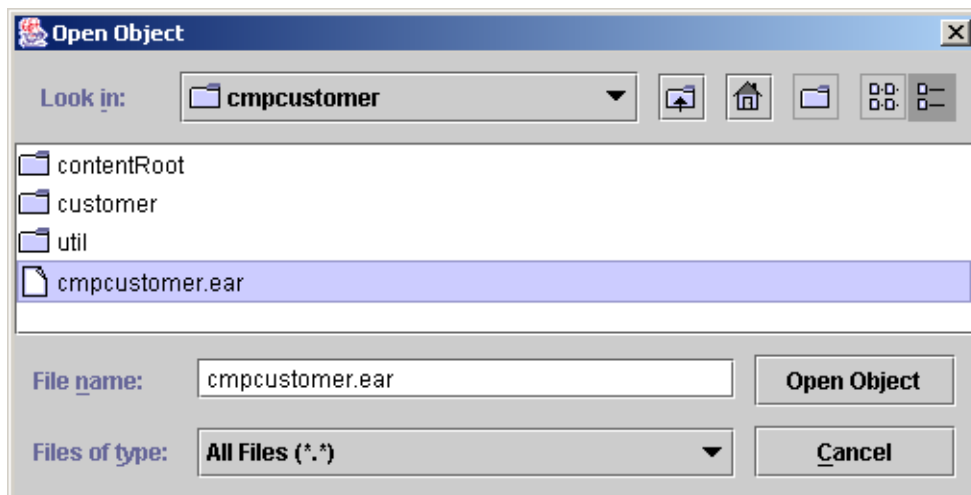


Figure 20.7. xxx (482)

Click `Open Object`.

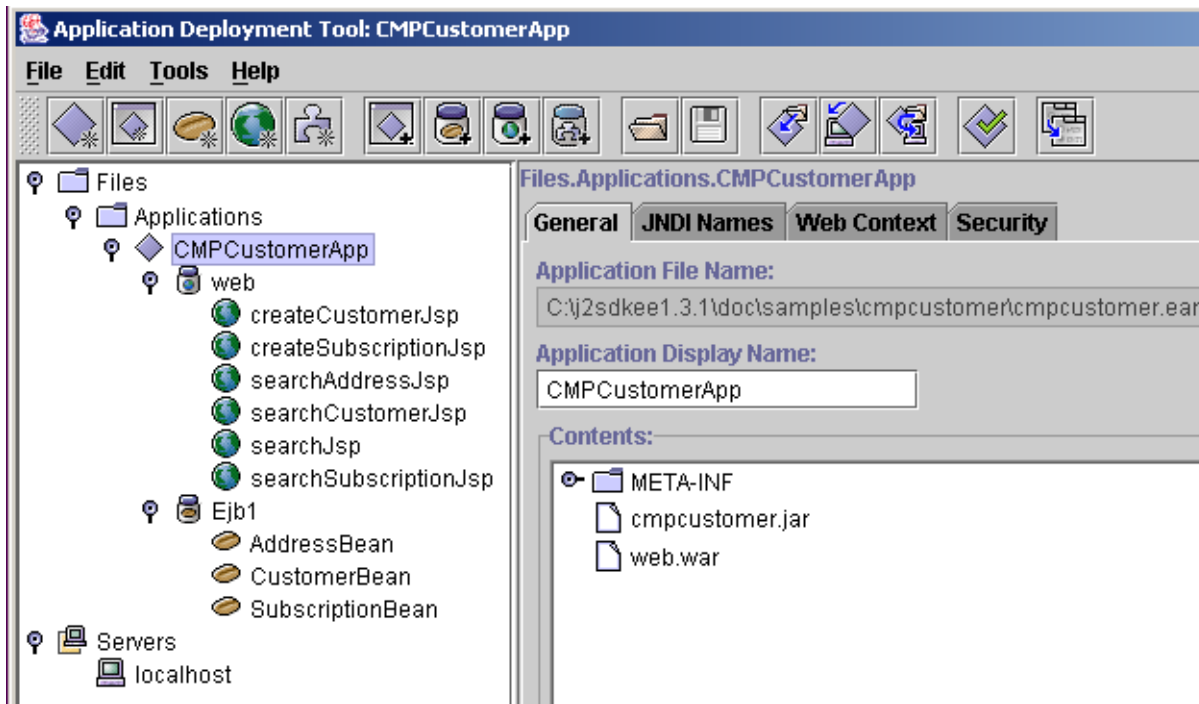


Figure 20.8. xxx (492)

21.2.3. Generate SQL

[from C:\j2sdkee1.3.1\doc\release\CMP-RI.html](http://C:\j2sdkee1.3.1\doc\release\CMP-RI.html)

Before you can run the sample application, be sure that the application has been deployed and that the enterprise bean SQL code has been generated. Use the deploytool's Deploy function (available from the Tools menu) to deploy the application.

To generate the SQL:

- Select Ejb1 beneath the CMPCustomerApp application.
- Select the General tab.
- Select Deployment Settings.
- From the Deployment Settings screen, select Generate SQL Now. This operation generates the SQL statements.

You may now run the application. From a browser window, enter the following location:

<http://localhost:8000/customer>

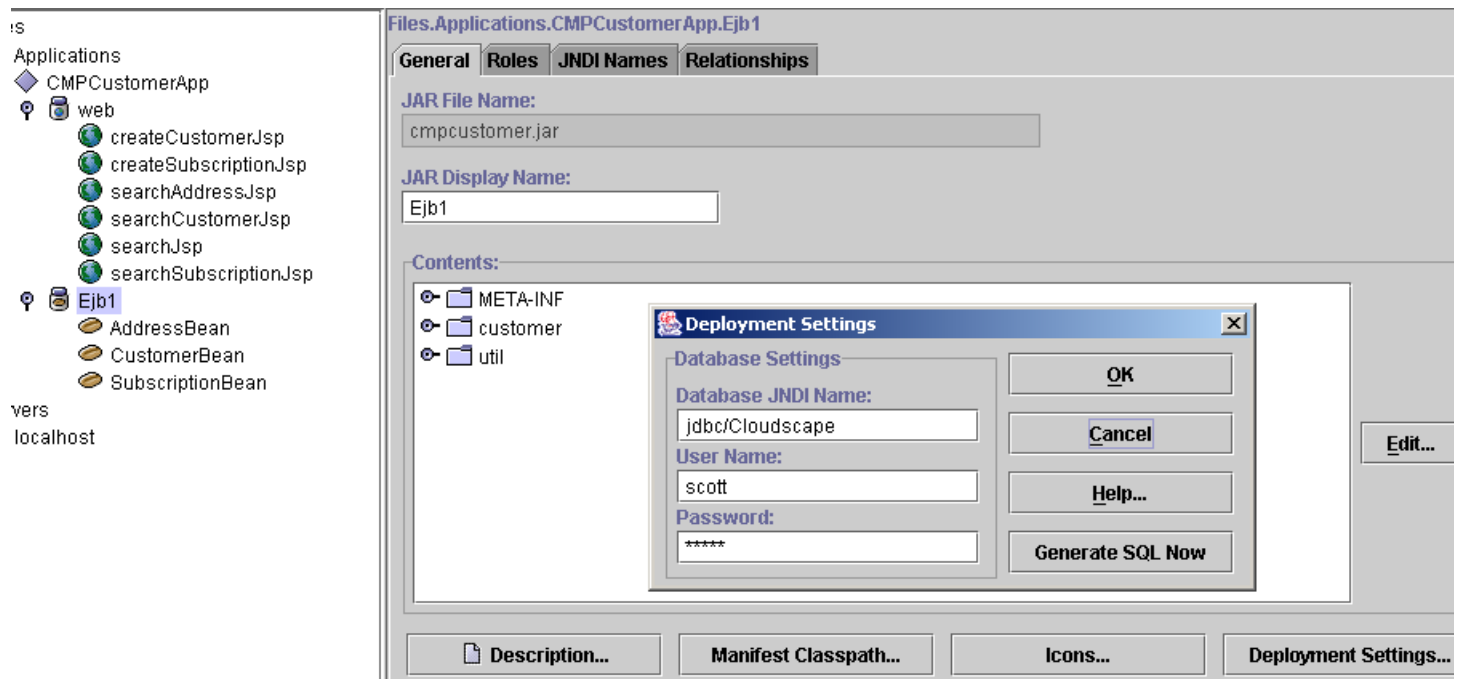


Figure 20.9. xxx (481)

Dialog appears “SQL Generation Complete”. Click OK. Dialog Deployment Settings. Click OK.

21.2.4. Deploy

Select Tools | Deploy. The appears.
Check Return Client Jar.

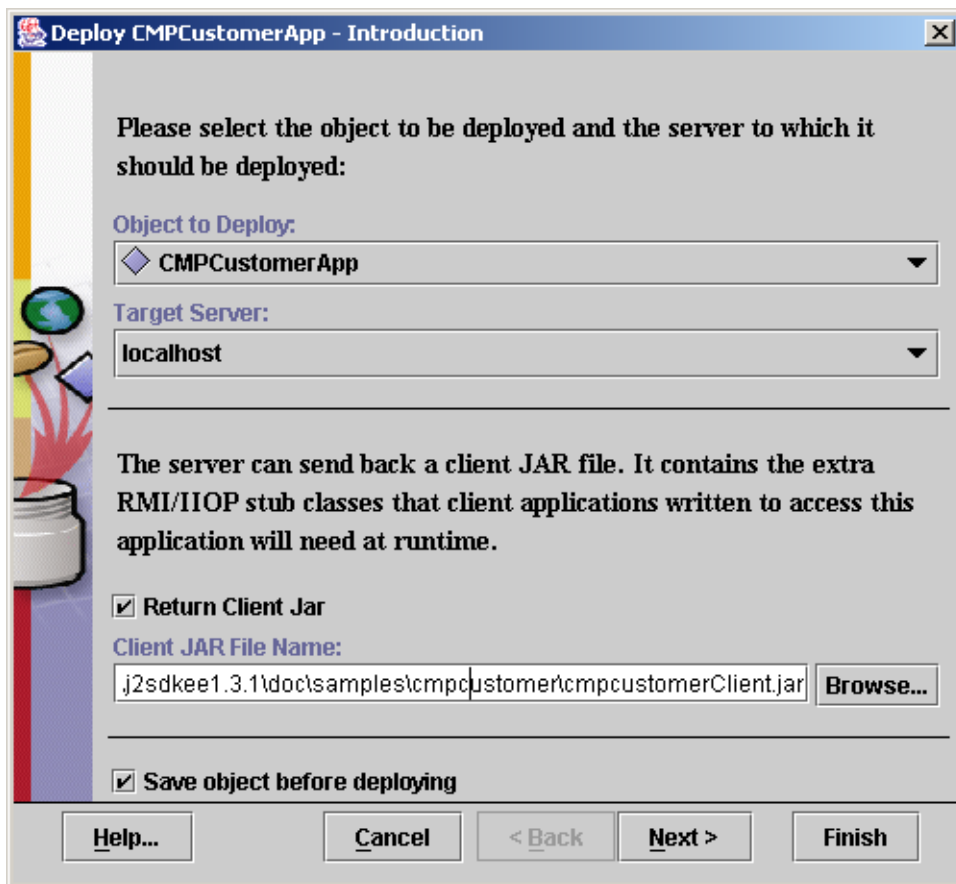


Figure 20.10. xxx (481)

Click Next.

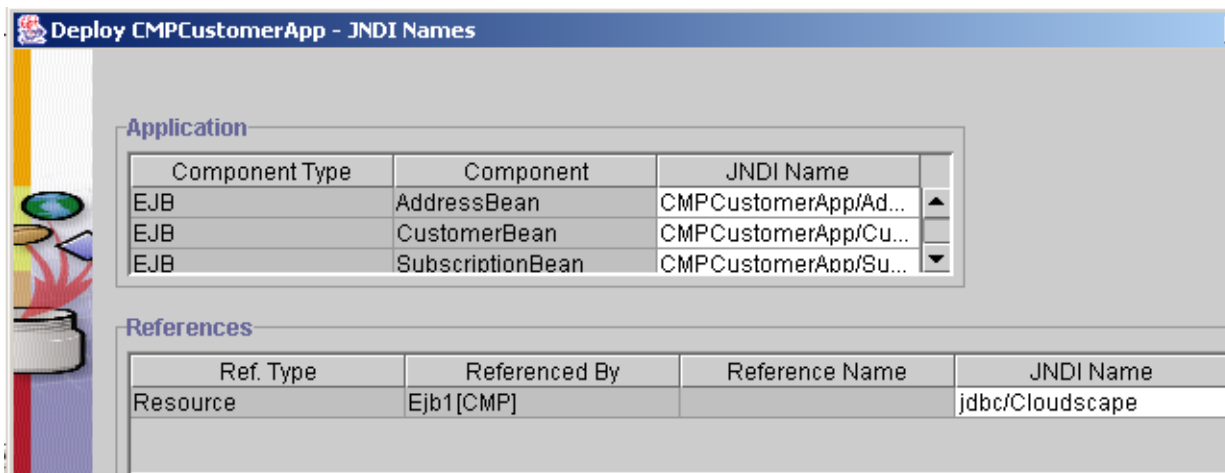


Figure 20.11. xxx (479)

Click Next.

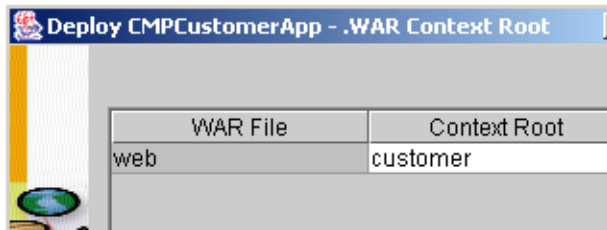


Figure 20.12. xxx (478)

Click Finish. Appears Deployment Progress dialog.

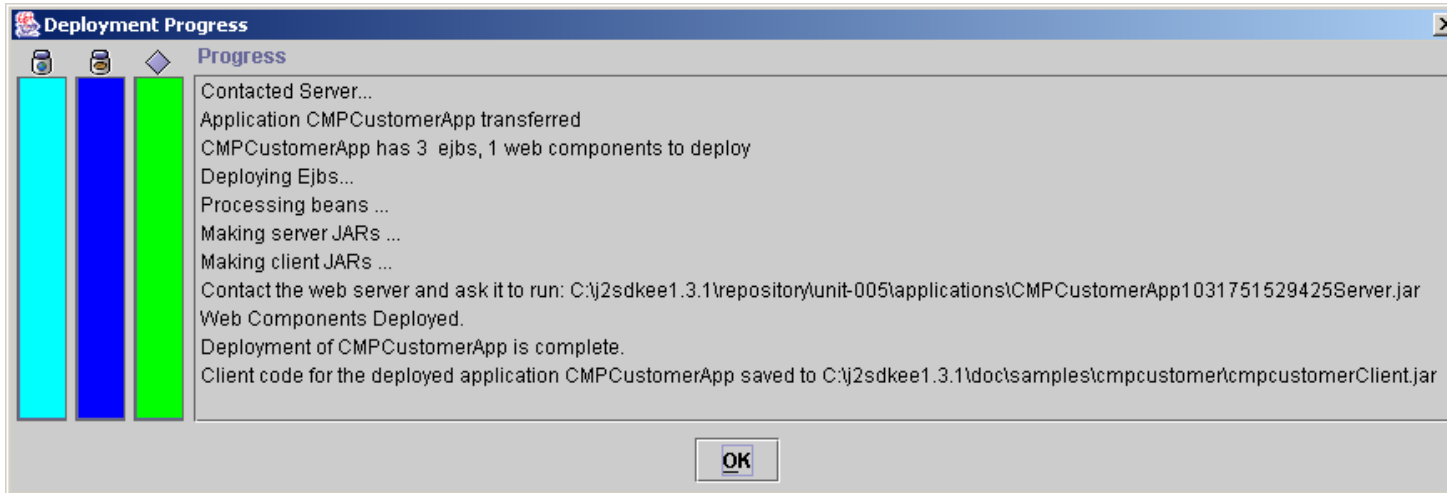


Figure 20.13. xxx (477)

Click OK.

20.14. In IE: Open <http://localhost:8000/customer/index.html>.

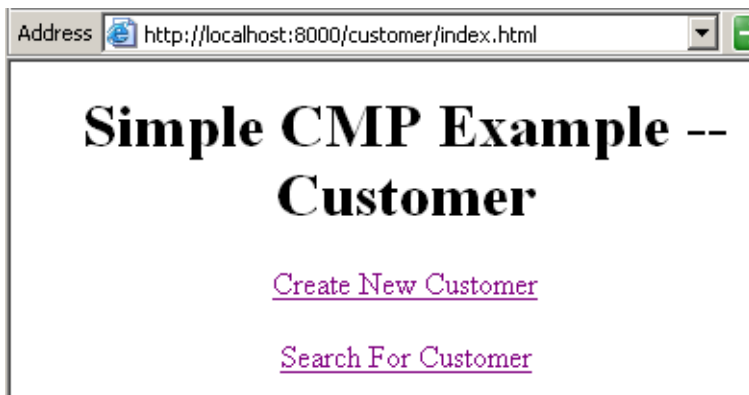


Figure 20.14. xxxut (476)

21.3. Set IDEA EJB project properties

20.15. In IDE: Select **File | Project properties**.

20.16. Select tab **EJB**.

20.17. Click **+**. The dialog “Create EJB Group” appears.

20.18. For Name: Enter **EJBGroup1**.

20.19. For path to ejb-jar.xml: Enter xxx.

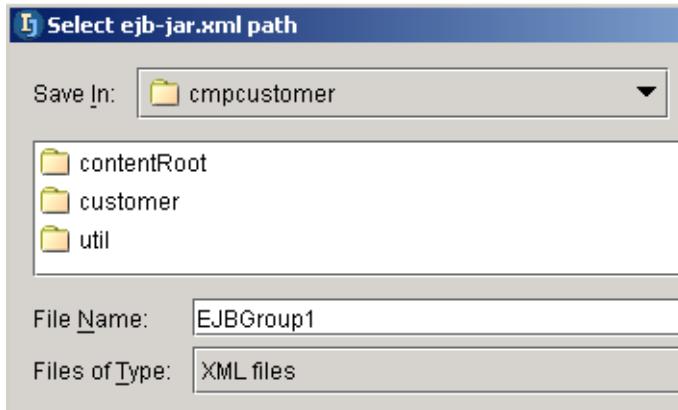


Figure 20.15. xxxut (475)

20.20. Check Sourcepath.

20.21. For Sourcepath enter xxx:

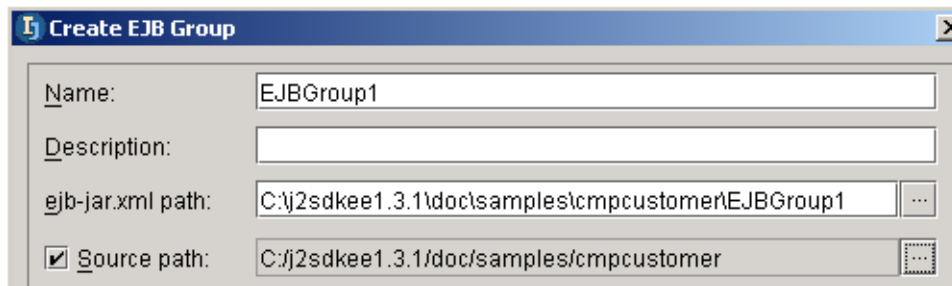


Figure 20.16. xxxut (474)

20.22. Click **OK**.

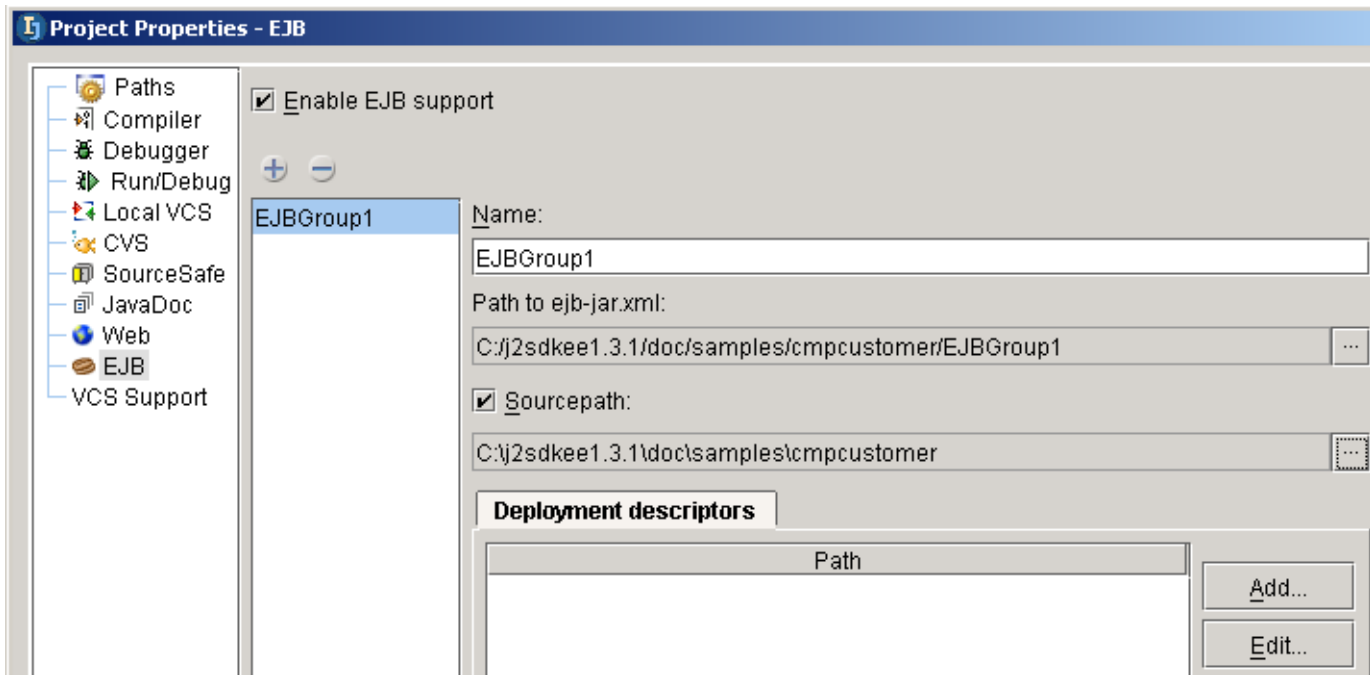


Figure 20.17. xxxut (491)
20.23. Click **OK**.

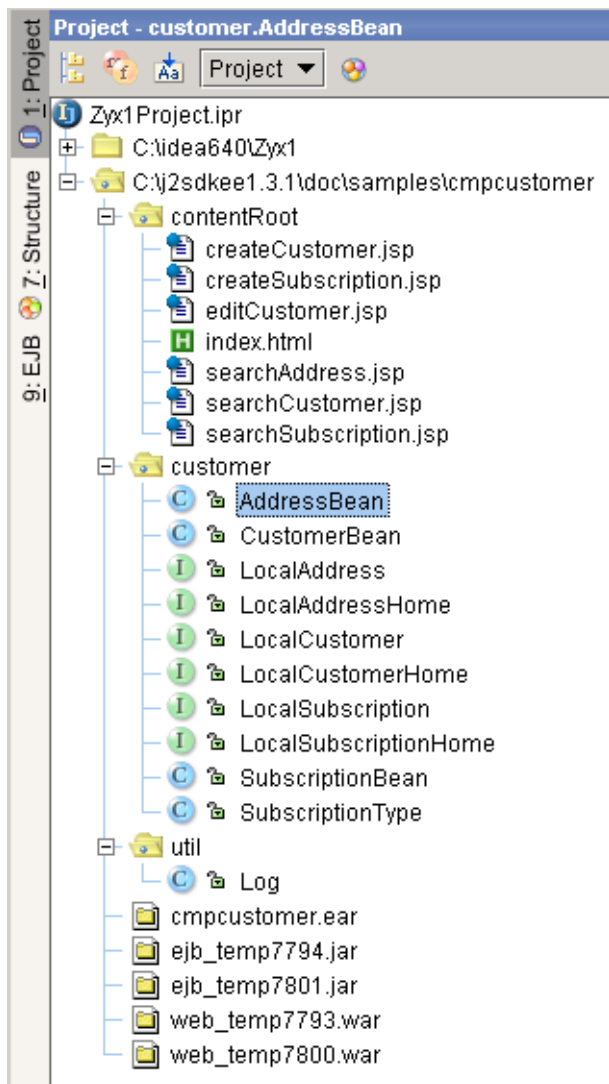


Figure 20.18. xxxxut (490)

21.4. Modify source in IDEA



Figure 20.19. xxxut (489)

21.5. Redeploy

20.24. Select **Tools** | **Update and redeploy**.

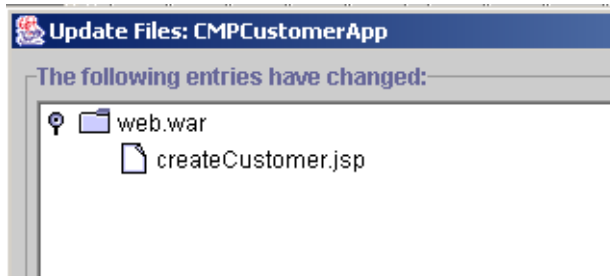


Figure 20.20. xxxut (488)

20.25. Click **OK**. The dialog “Deployment progress” appears.

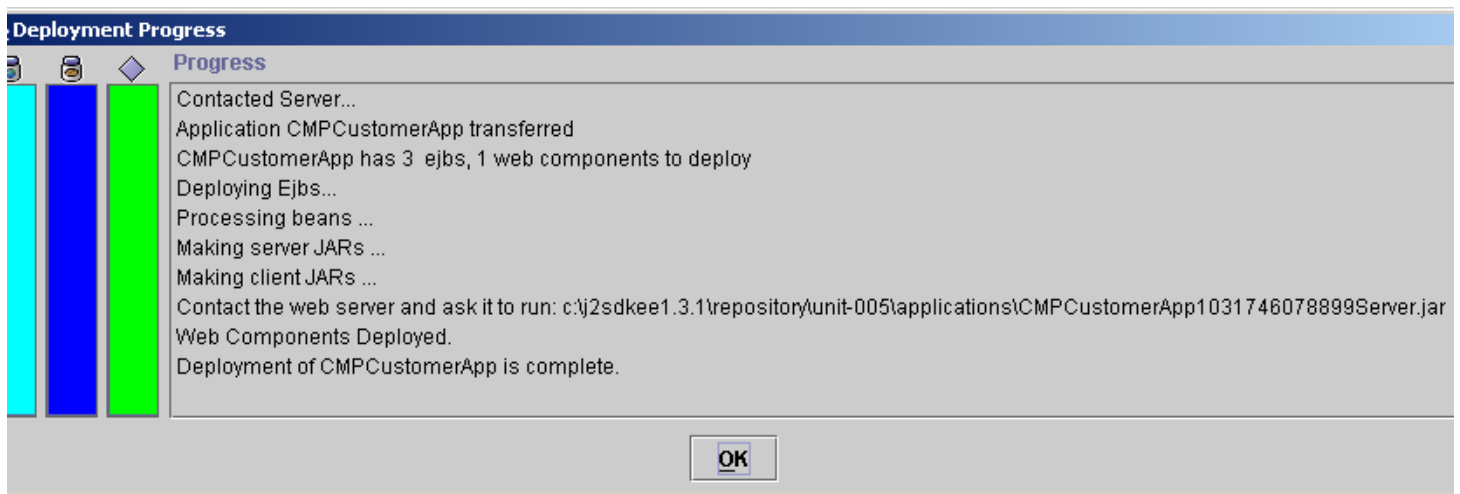


Figure 20.21. xxxut (487)

20.26. **OK**.

20.27. Click on **Create a new customer**.



Figure 20.22. xxxut (486)

Part F. Tools and resources

The chapters in this part describe how to integrate external tools (plugins) and resources within IDEA.

22. Plugins X (page 345).

23. External Tools X (page 363).

~~**24. External Resources XXX (page 367).**~~

22. Plugins X

20020919TTT: description of idea examples added.
see C:\idea640\doc\openapi\plugins.html
Consult ?? for details.

- **22.1. Install a plugin (page 345)**
- **22.2. Create a plugin (page 347)**
- **22.3. Publish a plugin XXX (page 362)**

22.1. Install a plugin

- **22.1.1. Find a plugin (page 345)**
- **22.1.2. Install files (page 346)**
- **22.1.3. Using the plugin (page 346)**

22.1.1. Find a plugin

21.1. Open www.intellij.org. The page lists (among other things) available plugins for IDEA.

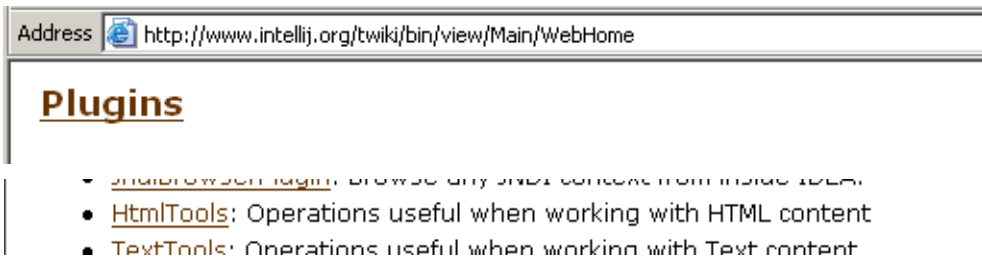


Figure 21.1. www.intellij.org start page (538,537)

21.2. Click on HtmlTools. A listing of available HTML tools appears.

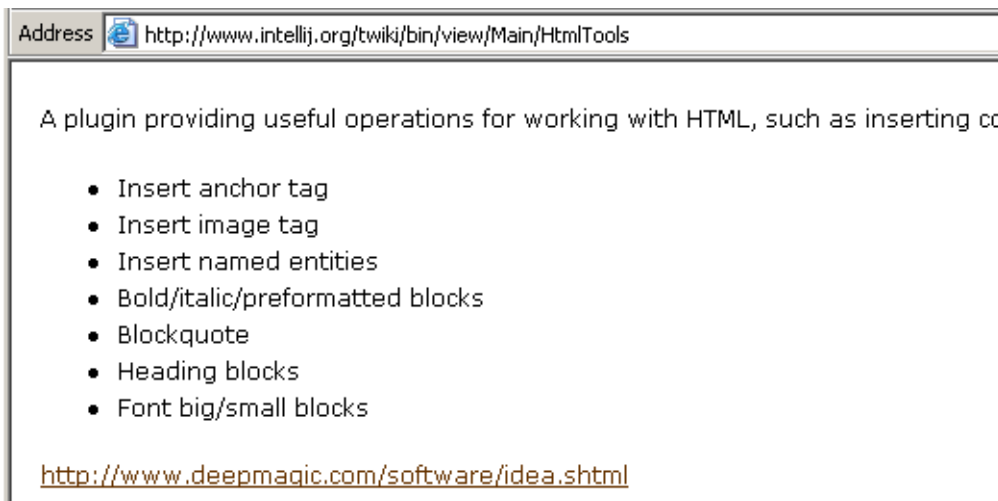


Figure 21.2. HtmlTools (536)

21.3. Click on the link. Private website.

21.4. Download HTML Tools Plugin v1.3 (zip file).

22.1.2. Install files

22.1.2.1. Class / XML files XXX

22.1.2.2. JAR file

21.5. Unzip the file.

21.6. Copy html.jar to C:\idea640\plugins.

21.7. Restart IDEA. Note the HTML menu item in the main menu.

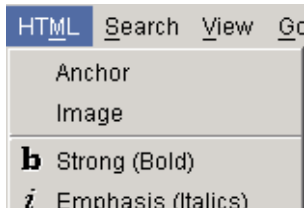


Figure 21.3. HTML main menu item (535)

22.1.3. Using the plugin

21.8. Create an HTML file.

21.9. Add the text shown in the following diagram.

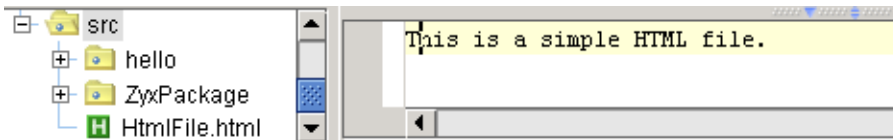


Figure 21.4. Simple HTML file (534)

21.10. Select a word in the text.

21.11. From the main menu select HTML / Strong(Bold). The html text is added.

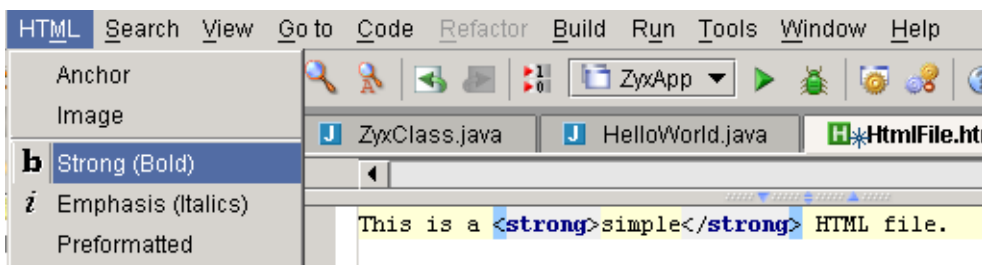


Figure 21.5. HTML bold tags added to text (533)

22.2. Create a plugin

Note: The code for the following examples are available in dir %idea%\doc\openapiexamples.
20020919TTT: you must first add the idea640\lib jars files to the class path to compile.

- 22.2.1. Application / project plugin (page 347)
- 22.2.2. Action plugin (page 351)
- 22.2.3. Tool window plugin (page 355)
- 22.2.4. Virtual file system (Vfs) plugin (page 358)

22.2.1. Application / project plugin

The code for this example is available in dir %idea%\doc\openapiexamples\plugin.

- 22.2.1.1. Create jar (page 347)
- 22.2.1.2. Install plugin (page 350)
- 22.2.1.3. Test plugin (page 350)

22.2.1.1. Create jar

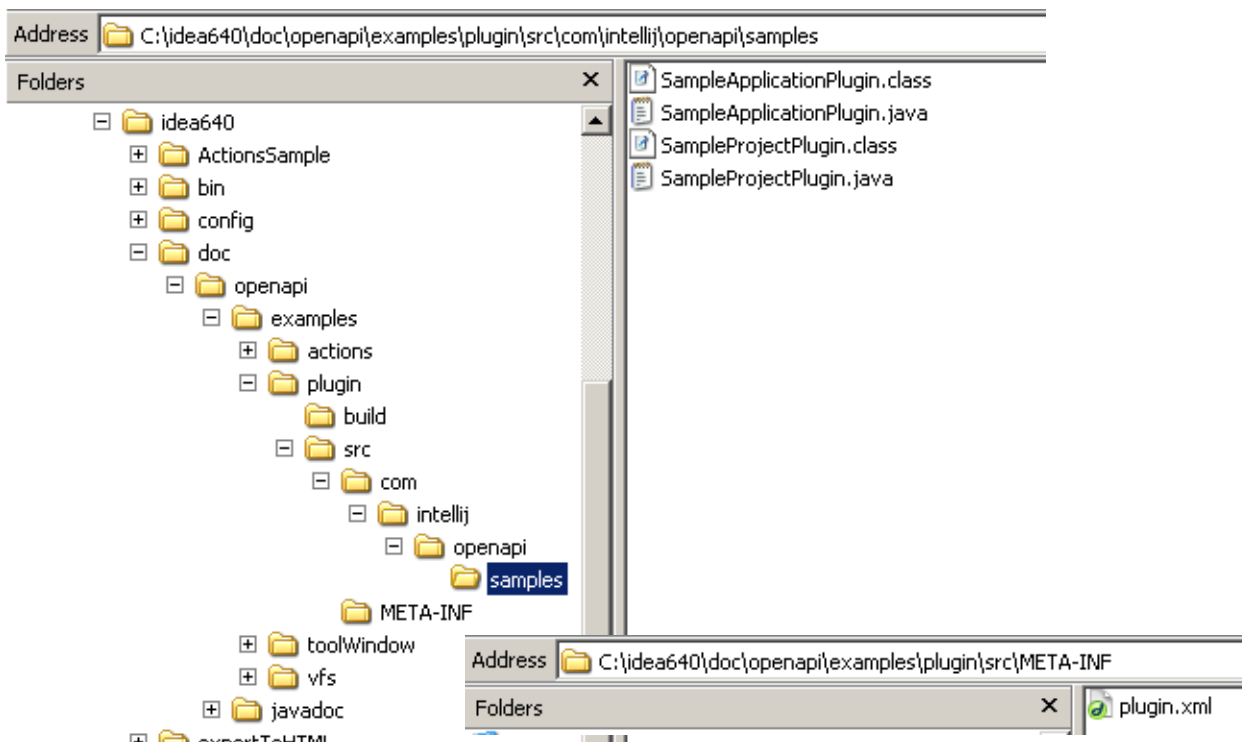


Figure 21.6. xxx (465,464)

21.12. Create and compile file com\intelliij\openapi\samples\SampleApplicationPlugin.java:

```
package com.intellij.openapi.samples;  
import com.intellij.openapi.components.ProjectComponent;  
/**  
 * <h3>SampleProjectPlugin</h3>  
 * Project level plugin sample showing IDEA <b>OpenAPI</b> basics.<br>  
 * Implements <code>ApplicationComponent</code> interface.
```

```
*/
public class SampleProjectPlugin implements ProjectComponent {
    /**
     * Method is called after plugin is already created and configured. Plugin
     * can start to communicate with
     * other plugins only in this method.
     */
    public void initComponents() {
        System.out.println("SampleProjectPlugin: initComponents");
    }

    /**
     * This method is called on plugin disposition.
     */
    public void disposeComponent() {
        System.out.println("SampleProjectPlugin: disposeComponent");
    }

    /**
     * Invoked when project is opened.
     */
    public void projectOpened() {
        System.out.println("SampleProjectPlugin: projectOpened");
    }

    /**
     * Invoked when project is closed.
     */
    public void projectClosed() {
        System.out.println("SampleProjectPlugin: projectClosed");
    }

    /**
     * Returns the name of component
     * @return String representing component name. Use
     * plugin_name.component_name notation.
     */
    public String getComponentName() {
        return "Sample.SampleProjectPlugin";
    }
}
```

21.13. Create and compile file `com\intellij\openapi\samples\SampleApplicationPlugin.java`:

```
package com.intellij.openapi.samples;
import com.intellij.openapi.components.ApplicationComponent;
/**
 * <h3>SampleApplicationPlugin</h3>
 * Application level plugin sample showing IDEA <b>OpenAPI</b> basics.<br>
 * Implements <code>ApplicationComponent</code> interface.
 */
public class SampleApplicationPlugin implements ApplicationComponent {
    /**
     * Method is called after plugin is already created and configured. Plugin
     * can start to communicate with
     * other plugins only in this method.
     */
    public void initComponents() {
```

```
        System.out.println("SampleApplicationPlugin: initComponents");
    }

    /**
     * This method is called on plugin disposal.
     */
    public void disposeComponent() {
        System.out.println("SampleApplicationPlugin: disposeComponent");
    }

    /**
     * Returns the name of component
     * @return String representing component name. Use
     plugin_name.component_name notation.
     */
    public String getComponentName() {
        return "Sample.SampleApplicationPlugin";
    }
}
```

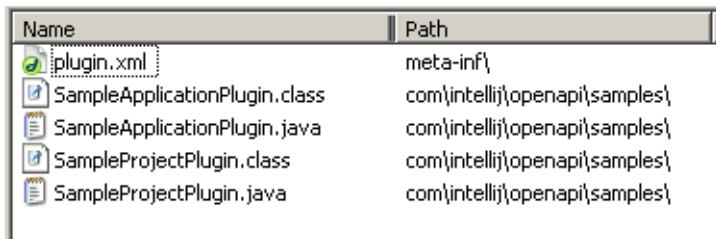
21.14. Create file META-INF\plugin.xml:

```
<!--
  plugin name
-->
<name>Sample</name>
<!--
  description
-->
<description>SamplePlugin</description>
<!--
  plugin version
-->
<version>1.0</version>
<vendor>IntelliJ</vendor>
<!--
  minimum and maximum IDEA version plugin is supposed to work with
-->
<idea-version min="3.0" max="3.1" />
<!--
  application components of the plugin
-->
<application-components>
  <component>
    <!--
      component implementation class
    -->
    <implementation-
class>com.intellij.openapi.samples.SampleApplicationPlugin</implementation-
class>
    <!--
      component interface class
    -->
    <interface-class>com.intellij.openapi.samples.SampleApplicationPlugin</
interface-class>
  </component>
</application-components>
```

```
- <!--  
  project components of the plugin  
-->  
_ <project-components>  
_ <component>  
<implementation-class>com.intellij.openapi.samples.SampleProjectPlugin</  
implementation-class>  
<interface-class>com.intellij.openapi.samples.SampleProjectPlugin</  
interface-class>  
</component>  
</project-components>  
</idea-plugin>
```

21.15. Compile.

21.16. Pack the files into **Sample.jar**.



Name	Path
plugin.xml	meta-inf\
SampleApplicationPlugin.class	com\intellij\openapi\samples\
SampleApplicationPlugin.java	com\intellij\openapi\samples\
SampleProjectPlugin.class	com\intellij\openapi\samples\
SampleProjectPlugin.java	com\intellij\openapi\samples\

Figure 21.7. Sample.jar contents [\(463\)](#)

22.2.1.2. Install plugin

21.17. Close IDEA (if open).

21.18. Copy Sample.jar to **C:\idea640\plugins**.

22.2.1.3. Test plugin

21.19. Start IDEA.

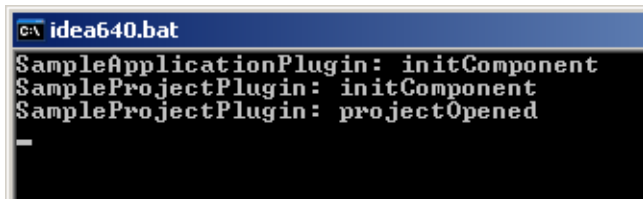


Figure 21.8. xxx [\(462\)](#)

22.2.2. Action plugin

The code for this example is available in dir `%idea%\doc\openapi\examples\actions`.

- [22.2.2.1. Create jar \(page 351\)](#)
- [22.2.2.2. Install plugin \(page 354\)](#)
- [22.2.2.3. Test plugin \(page 354\)](#)

22.2.2.1. Create jar

21.20. Create and compile file `com\intelliij\openapi\samples\ActionsPlugin.java`:

```
package com.intelliij.openapi.samples;
import com.intelliij.openapi.components.ApplicationComponent;
/**
 * <h3>SampleApplicationPlugin</h3>
 *
 * Application level plugin sample showing IDEA <b>OpenAPI</b> basics.<br>
 * Implements <code>ApplicationComponent</code> interface.
 */
public class ActionsPlugin implements ApplicationComponent {

    /**
     * Method is called after plugin is already created and configured. Plugin
     can start to communicate with
     * other plugins only in this method.
     */
    public void initComponents() {

    }

    /**
     * This method is called on plugin disposal.
     */
    public void disposeComponent() {

    }

    /**
     * Returns the name of component
     *
     * @return String representing component name. Use
     PluginName.ComponentName notation
     * to avoid conflicts.
     */
    public String getComponentName() {
        return "ActionsSample.ActionsPlugin";
    }
}
```

21.21. Create and compile file `com\intelliij\openapi\samples\HelloWorldAction.java`:

```
package com.intelliij.openapi.samples;

import com.intelliij.openapi.actionSystem.AnAction;
import com.intelliij.openapi.actionSystem.AnActionEvent;
import com.intelliij.openapi.actionSystem.DataConstants;
import com.intelliij.openapi.ui.Messages;
import com.intelliij.openapi.project.Project;
```

```
import javax.swing.*;

public class HelloWorldAction extends AnAction {
    public void actionPerformed(AnActionEvent event) {
        Project project =
        (Project)event.getDataContext().getData(DataConstants.PROJECT);
        Messages.showMessageDialog(project, "Hello World!", "Information",
        Messages.getInformationIcon());
    }
}
```

21.22. Create and compile file **com\intellij\openapi\samples\GarbageCollectionAction.java**:

```
package com.intellij.openapi.samples;

import com.intellij.openapi.actionSystem.ActionPlaces;
import com.intellij.openapi.actionSystem.AnAction;
import com.intellij.openapi.actionSystem.AnActionEvent;
import com.intellij.openapi.actionSystem.Presentation;

import javax.swing.*;

public class GarbageCollectionAction extends AnAction {
    private ImageIcon myIcon;

    public GarbageCollectionAction() {
        super("GC", "Run garbage collection", null);
    }

    public void actionPerformed(AnActionEvent event) {
        System.gc();
    }

    public void update(AnActionEvent event) {
        super.update(event);
        Presentation presentation = event.getPresentation();
        if (ActionPlaces.MAIN_TOOLBAR.equals(event.getPlace())) {
            if (myIcon == null) {
                java.net.URL resource = GarbageCollectionAction.class.getResource("/icons/garbage.png");
                myIcon = new ImageIcon(resource);
            }
            presentation.setIcon(myIcon);
        }
    }
}
```

21.23. Create file **META-INF\plugin.xml**:

```
<?xml version="1.0" encoding="UTF-8" ?>
<idea-plugin>
  <!--
    Plugin name
  -->
  <name>ActionsSample</name>
  <!--
    Description
  -->
  <description>ActionSamplePlugin</description>
</idea-plugin>
```



```
- <!--
  Plugin version
-->
<version>1.0</version>
- <!--
  Plugin's vendor
-->
<vendor>IntelliJ</vendor>
- <!--
  Minimum and maximum IDEA version plugin is supposed to work with
-->
<idea-version min="3.0" max="3.1" />
- <!--
  Plugin's application components
-->
_ <application-components>
_ <component>
- <!--
  Component's implementation class
-->
<implementation-class>com.intellij.openapi.samples.ActionsPlugin</
implementation-class>
- <!--
  Component's interface class
-->
<interface-class>com.intellij.openapi.samples.ActionsPlugin</interface-
class>
</component>
</application-components>
- <!--
  Component's actions
-->
_ <actions>
- <!--
  We use "PluginName.ComponentName.ActionName" notation for "id" to avoid
conflicts
-->
_ <action id="ActionsSample.ActionsPlugin.GarbageCollection"
class="com.intellij.openapi.samples.GarbageCollectionAction" text="Collect
_garbage" description="Run garbage collector">
<shortcut first-keystroke="control alt G" second-keystroke="C"
keymap="$default" />
</action>
<action id="Actions.ActionsPlugin.HelloWorld1"
class="com.intellij.openapi.samples.HelloWorldAction" text="Hello World1"
description="" />
_ <group id="Actions.ActionsPlugin.SampleGroup" text="S_ample"
description="Sample group">
<reference id="ActionsSample.ActionsPlugin.GarbageCollection" />
<separator />
<action id="Actions.ActionsPlugin.HelloWorld"
class="com.intellij.openapi.samples.HelloWorldAction" text="Hello World"
description="" />
- <!--
adds this group to the main menu
-->
<add-to-group group-id="MainMenu" anchor="last" />
```

```
- <!--  
adds this group to the main toolbar before the Help action  
-->  
<add-to-group group-id="MainToolBar" anchor="before" relative-to-  
action="HelpTopics" />  
</group>  
- <!--  
the group below contains only the "Hello World" action defined above  
-->  
_ <group>  
<reference id="Actions.ActionsPlugin.HelloWorld1" />  
- <!--  
the group is added to the editor popup menu  
-->  
<add-to-group group-id="EditorPopupGroup" anchor="after" relative-to-  
action="GotoImplementation" />  
</group>  
</actions>  
</idea-plugin>
```

21.24. Compile.

21.25. Pack the files into **ActionsSample.jar**.

Name	Path
plugin.xml	meta-inf\
ActionsPlugin.class	com\intellij\openapi\samples\
GarbageCollectionAction.class	com\intellij\openapi\samples\
HelloWorldAction.class	com\intellij\openapi\samples\

Figure 21.9. ActionsSample.jar contents (461)

22.2.2.2. Install plugin

21.26. Close IDEA (if open).

21.27. Copy ActionsSample.jar to **C:\idea640\plugins**.

22.2.2.3. Test plugin

21.28. Start IDEA.

21.29. Select **Sample | HelloWorld**. An information dialog appears.



Figure 21.10. xxx (460,459)

21.30. Click **OK**.

21.31. Select **Sample | Collection garbage**. The garbage is collected (note the difference in the heap size **28M of 47M**).

22.2.3. Tool window plugin

The code for this example is available in dir `%idea%\doc\openapi\examples\toolWindow`.

- [22.2.3.1. Create jar \(page 355\)](#)
- [22.2.3.2. Install plugin \(page 357\)](#)
- [22.2.3.3. Test plugin \(page 357\)](#)

22.2.3.1. Create jar

21.32. Create and compile file `com\intelliij\openapi\samples\SimpleToolWindowPlugin.java`:

```
package com.intelliij.openapi.samples;

import com.intelliij.openapi.components.ProjectComponent;
import com.intelliij.openapi.project.Project;
import com.intelliij.openapi.wm.ToolWindow;
import com.intelliij.openapi.wm.ToolWindowAnchor;
import com.intelliij.openapi.wm.ToolWindowManager;

import javax.swing.*;
import java.awt.*;

public class SimpleToolWindowPlugin implements ProjectComponent {
    private Project myProject;

    private ToolWindow myToolWindow;
    private JPanel myContentPanel;

    public static final String TOOL_WINDOW_ID = "SimpleToolWindow";

    public SimpleToolWindowPlugin(Project project) {
        myProject = project;
    }

    public void projectOpened() {
        initToolWindow();
    }

    public void projectClosed() {
        unregisterToolWindow();
    }

    public void initComponents() {
        // empty
    }

    public void disposeComponent() {
        // empty
    }

    public String getComponentName() {
        return "SimpleToolWindow.SimpleToolWindowPlugin";
    }

    private void initToolWindow() {
```

```
    ToolWindowManager toolWindowManager =
ToolWindowManager.getInstance(myProject);

    myContentPanel = new JPanel(new BorderLayout());

myContentPanel.setBackground(UIManager.getColor("Tree.textBackground"));
    myContentPanel.add(new JLabel("Hello World!", JLabel.CENTER),
BorderLayout.CENTER);

    myToolWindow = toolWindowManager.registerToolWindow(TOOL_WINDOW_ID,
myContentPanel, ToolWindowAnchor.LEFT);
    myToolWindow.setTitle("SimpleWindow");
}

private void unregisterToolWindow() {
    ToolWindowManager toolWindowManager =
ToolWindowManager.getInstance(myProject);
    toolWindowManager.unregisterToolWindow(TOOL_WINDOW_ID);
}
}
```

21.33. Create file **META-INF\plugin.xml**:

```
_ <idea-plugin>
- <!--
    Plugin name
-->
<name>SimpleToolWindow</name>
- <!--
    Description
-->
<description>An example on installing a tool window</description>
- <!--
    Plugin version
-->
<version>1.0</version>
- <!--
    Plugin's vendor
-->
<vendor>IntelliJ</vendor>
- <!--
    Minimum and maximum IDEA version plugin is supposed to work with
-->
<idea-version min="3.0" max="3.1" />
- <!--
    Plugin's application components
-->
_ <project-components>
_ <component>
- <!--
    Component's implementation class
-->
<implementation-class>com.intellij.openapi.samples.SimpleToolWindowPlugin</
implementation-class>
- <!--
    Component's interface class
```

```
-->  
<interface-class>com.intellij.openapi.samples.SimpleToolWindowPlugin</  
interface-class>  
</component>  
</project-components>  
</idea-plugin>
```

21.34. Compile.

21.35. Pack the files into **SimpleToolWindow.jar**.

Name	Path
plugin.xml	meta-inf\
SimpleToolWindowPlugin.class	com\intellij\openapi\samples\

Figure 21.11. SimpleToolWindow.jar contents [\(457\)](#)

22.2.3.2. Install plugin

21.36. Close IDEA (if open).

21.37. Copy SimpleToolWindow.jar to **C:\idea640\plugins**.

22.2.3.3. Test plugin

21.38. Start IDEA. Note the tool window SimpleToolWindow.

21.39. Open **SimpleToolWindow**.

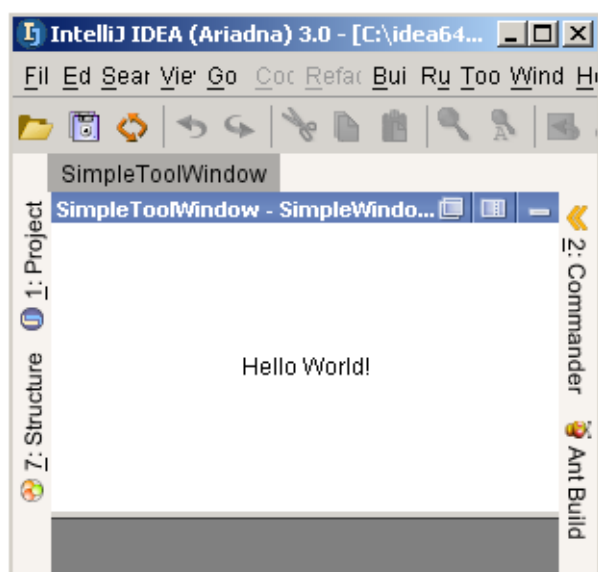


Figure 21.12. xxx [\(456\)](#)

22.2.4. Virtual file system (Vfs) plugin

The code for this example is available in dir %idea%\doc\openapi\examples\vfs.

- [22.2.4.1. Create jar \(page 358\)](#)
- [22.2.4.2. Install plugin \(page 360\)](#)
- [22.2.4.3. Test plugin \(page 360\)](#)

22.2.4.1. Create jar

21.40. Create and compile file `com\intellij\openapi\samples\VfsSamplePlugin.java`:

```
package com.intellij.openapi.samples;

import com.intellij.openapi.components.ProjectComponent;
import com.intellij.openapi.project.Project;
import com.intellij.openapi.projectRoots.ProjectRootManager;
import com.intellij.openapi.projectRoots.ProjectRootType;
import com.intellij.openapi.vfs.*;
import com.intellij.openapi.fileTypes.FileTypeManager;
import com.intellij.openapi.fileTypes.FileType;

public class VfsSamplePlugin implements ProjectComponent {
    private Project myProject;
    private MyVfsListener myVfsListener;

    private static int ourJavaFilesCount;

    public VfsSamplePlugin(Project project) {
        myProject = project;
    }

    public void projectOpened() {
        ProjectRootManager projectRootManager =
        ProjectRootManager.getInstance(myProject);
        VirtualFile[] sourceRoots =
        projectRootManager.getRootFiles(ProjectRootType.SOURCE);

        ourJavaFilesCount = 0;

        for (int i = 0; i < sourceRoots.length; i++) {
            VirtualFile sourceRoot = sourceRoots[i];
            countJavaFiles(sourceRoot);
        }

        myVfsListener = new MyVfsListener();
        VirtualFileManager.getInstance().addVirtualFileListener(myVfsListener);
    }

    public void projectClosed() {
        VirtualFileManager.getInstance().removeVirtualFileListener(myVfsListener);
    }

    public void initComponents() {
        // empty
    }
}
```

```
    }

    public void disposeComponent() {
        // empty
    }

    public String getComponentName() {
        return "VfsSample.VfsSamplePlugin";
    }

    private void updateCount(VirtualFile file, int increase) {
        FileTypeManager fileTypeManager = FileTypeManager.getInstance();
        if (!fileTypeManager.isFileIgnored(file.getName())
            && fileTypeManager.getFileTypeByFile(file) == FileType.JAVA) {
            ourJavaFilesCount += increase;
            System.out.println("ourJavaFilesCount = " + ourJavaFilesCount);
        }
    }

    private void countJavaFiles(VirtualFile virtualFile) {
        VirtualFile[] children = virtualFile.getChildren();
        if (children == null) return;
        for (int i = 0; i < children.length; i++) {
            VirtualFile child = children[i];
            updateCount(child, +1);
            countJavaFiles(child);
        }
    }

    // -----
    // MyVfsListener
    // -----
    -----

    private class MyVfsListener extends VirtualFileAdapter {
        public void fileCreated(VirtualFileEvent event) {
            updateCount(event.getFile(), +1);
        }

        public void fileDeleted(VirtualFileEvent event) {
            updateCount(event.getFile(), -1);
        }
    }
}
```

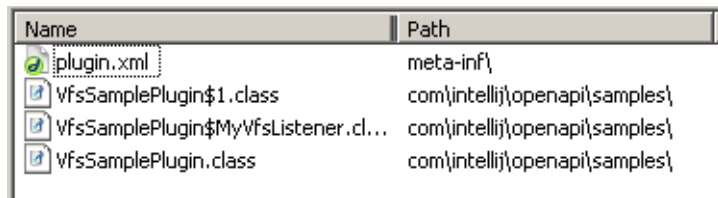
21.41. Create file **META-INF\plugin.xml**:

```
= <idea-plugin>
- <!--
  Plugin name
-->
<name>VfsSample</name>
- <!--
  Description
-->
```

```
<description>An example of using IDEA's Vfs, dynamically keeps the count of
Java files in sourcepaths</description>
- <!--
  Plugin version
-->
<version>1.0</version>
- <!--
  Plugin's vendor
-->
<vendor>IntelliJ</vendor>
- <!--
  Minimum and maximum IDEA version plugin is supposed to work with
-->
<idea-version min="3.0" max="3.1" />
- <!--
  Plugin's application components
-->
_ <project-components>
_ <component>
- <!--
  Component's implementation class
-->
<implementation-class>com.intellij.openapi.samples.VfsSamplePlugin</
implementation-class>
- <!--
  Component's interface class
-->
<interface-class>com.intellij.openapi.samples.VfsSamplePlugin</interface-
class>
</component>
</project-components>
</idea-plugin>
```

21.42. Compile.

21.43. Pack the files into **VfsSample.jar**.



Name	Path
plugin.xml	meta-inf\
VfsSamplePlugin\$1.class	com\intellij\openapi\samples\
VfsSamplePlugin\$MyVfsListener.cl...	com\intellij\openapi\samples\
VfsSamplePlugin.class	com\intellij\openapi\samples\

Figure 21.13. VfsSample.jar contents [\(455\)](#)

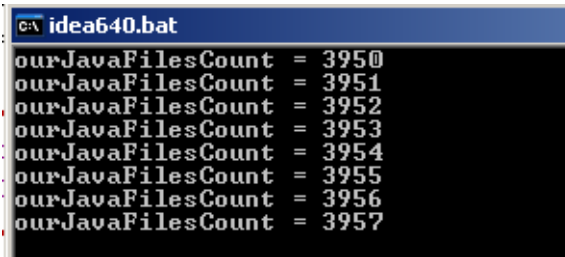
22.2.4.2. Install plugin

21.44. Close IDEA (if open).

21.45. Copy VfsSample.jar to **C:\idea640\plugins**.

22.2.4.3. Test plugin

21.46. Start IDEA. Note that the number of files are shown in the DOS window.



```
C:\> idea640.bat
ourJavaFilesCount = 3950
ourJavaFilesCount = 3951
ourJavaFilesCount = 3952
ourJavaFilesCount = 3953
ourJavaFilesCount = 3954
ourJavaFilesCount = 3955
ourJavaFilesCount = 3956
ourJavaFilesCount = 3957
```

Figure 21.14. xxx [\(454\)](#)

If you create a Java file, the count will be incremented.

22.3. Publish a plugin XXX

23. External Tools X

[Consult mike for details.](#)

[20020906TTT: not sure about this chapter.](#)

- [23.1. Add \(page 363\)](#)
- [23.2. Open \(page 365\)](#)

23.1. Add

22.1. From the main menu select **Tools / IDE options.**

22.2. Select **External Tools.**

22.3. Click **Add.** The dialog **Edit Tool** appears.

22.4. For **Name** enter "ZyxTool".

22.5. For **Group** enter 'ZyxGroup'.

22.6. For **Description** enter "ZyxTool description".

22.7. For **Program** click on the dir button. The dialog "Select path" appears.

22.8. Select a tool.

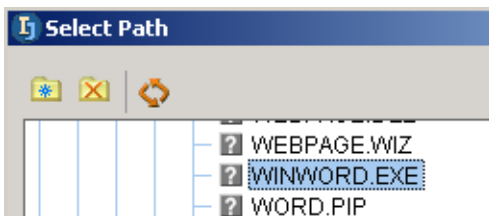


Figure 22.1. Selecting the external tool [\(532\)](#)

22.9. Click **OK.** The external tools info has been entered.

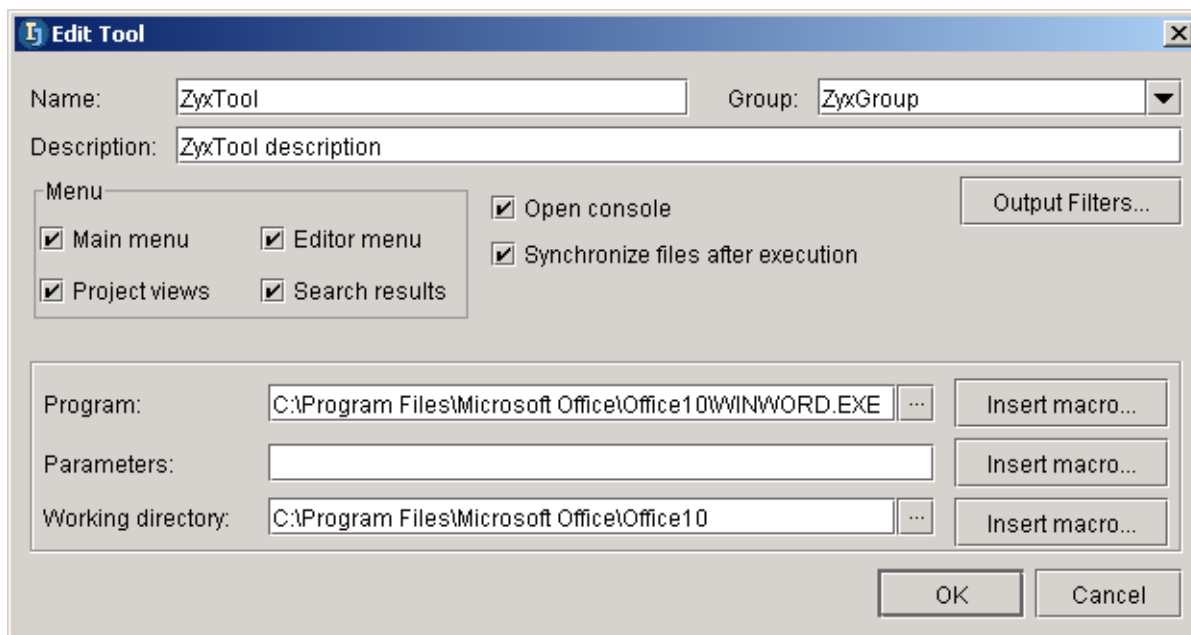


Figure 22.2. External tool settings [\(531\)](#)

22.10. Click **OK.** The tool is now included in the list.

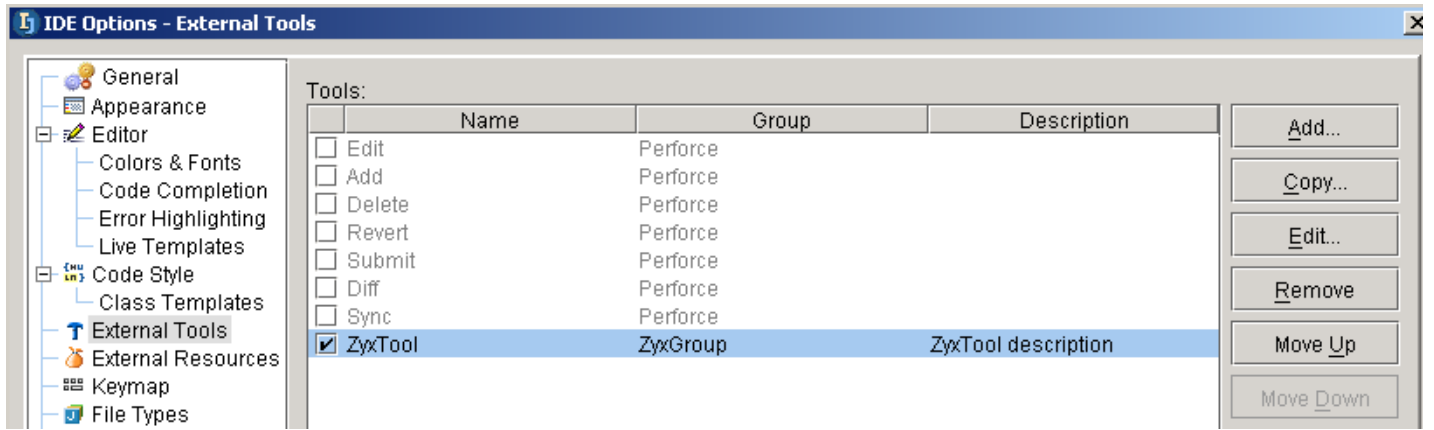


Figure 22.3. Tool in list of external tools (530)
22.11. Click **OK**.

23.2. Open

22.12. Select from the main menu **Tools / ZyxGroup / ZyxTool**.

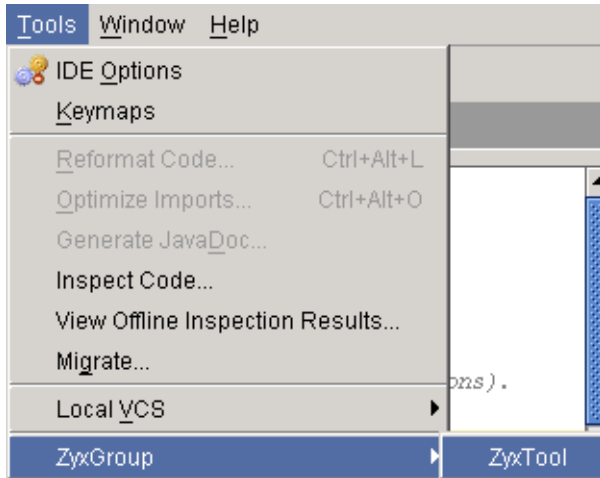


Figure 22.4. External tool in Tools menu (529)

The tool is opened. A message appears in IDEA.

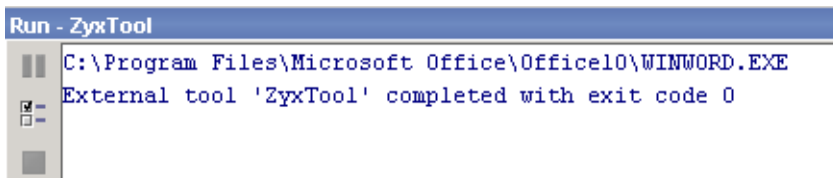


Figure 22.5. Message about external tool being opened (528)

~~24. External Resources XXX~~

~~[contacts:mike](#)~~

~~24.1. XXX~~

~~[23.1.](#)~~

~~[Figure 23.1.](#)~~

~~Part G. Appendices~~

~~This last part contains~~

~~[25. Views / User Interface XXX \(page 171\). Describes in detail the IDEA user interface.](#)~~

~~[26. Default Settings \(page 173\). Describes various default settings for keymaps, etc.](#)~~

~~[25. FAQ XXX \(page 371\).](#)~~

~~[26. Trouble shooting XXX \(page 373\).](#)~~

~~[27. Keymaps X \(page 375\).](#)~~

~~[28. Glossary XXX \(page 379\).](#)~~

~~25. FAQ XXX~~

~~This chapter contains the answers to many FAQs.~~

~~For more FAQ info see also:~~

- ~~• <http://www.intellij.com/support/faq/>~~
- ~~• Online IDEA FAQ at <http://www.jguru.com/faq/home.jsp?topic=IntelliJIDEA> (questions can be posted at <http://www.jguru.com/forums/home.jsp?topic=IntelliJIDEA>)~~

~~FAQ 1. Group of faqs~~

~~A FAQ section as a table~~

FAQ 1.1. A single faq	Faq text.
FAQ 1.2. A single faq	Faq text.

~~FAQ 2. Group of faqs~~

~~A FAQ section as text (all tables would be converted to text in the released PUBLIC doc~~

~~FAQ 2.1. A single faq~~

~~Faq text.~~

~~FAQ 2.2. A single faq~~

~~Faq text.~~

~~26. Trouble shooting XXX~~

~~TS 1. Group of TSs~~

~~A TS section as a table~~

TS 1.1. A single TS	TS text.
TS 1.2. A single TS	TS text.

~~TS 2. Group of TSs~~

~~A TS section as text (all tables would be converted to text in the released PUBLIC doc~~

~~TS 2.1. A single TS~~

~~TS text.~~

~~TS 2.2. A single TS~~

~~TS text.~~

~~27. Keymaps X~~

Hot keys allow you perform almost all actions within IDEA without having to select the action from a menu.

In the IDEA Options / Keymaps settings dialog you can

- **27.1. Select Active (page 375)**
- **27.2. Create (copy and modify) (page 376)**

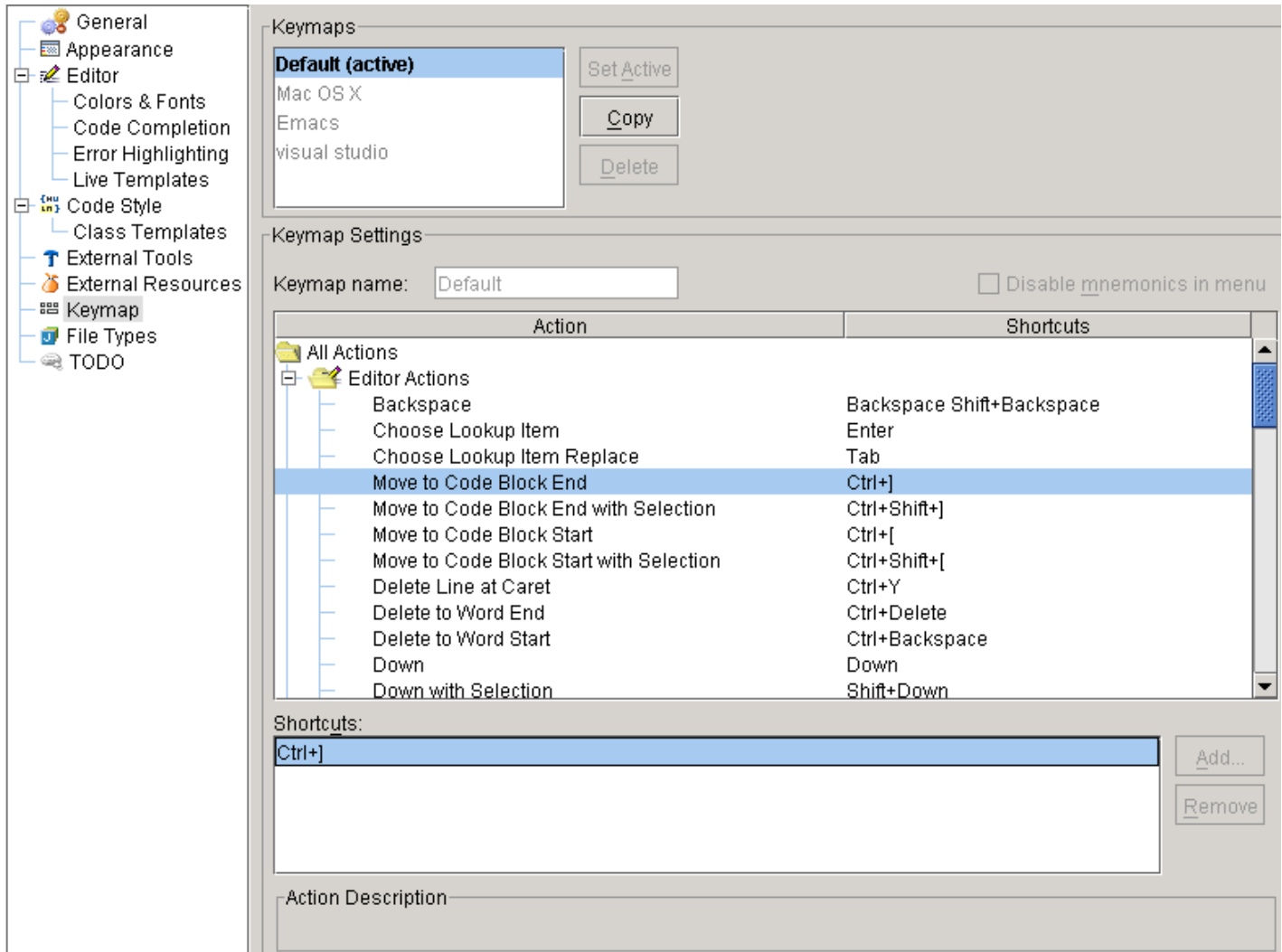


Figure 24.1. IDEA Options / Keymaps (425)

27.1. Select Active

- 24.1. Select a Keymap.
- 24.2. Click **Set Active**.

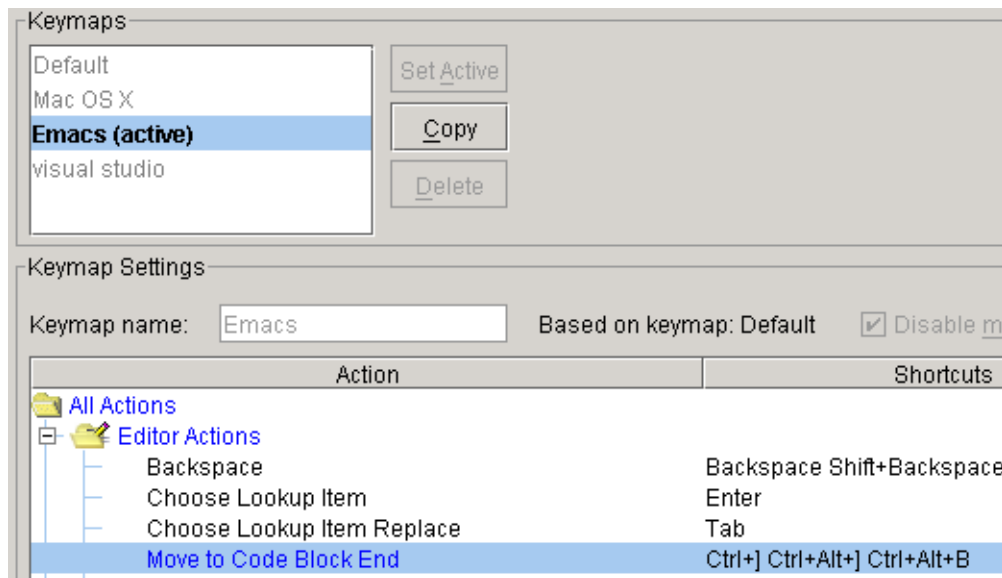


Figure 24.2. Set active keymap (424)

Note: Be sure to reset the default keymap as the active.

27.2. Create (copy and modify)

You can use your own custom keymap by

- copy an existing keymap
- modify the copy
- set the copy as the active keymap

24.3. Select the **Default** keymap.

24.4. Click **Copy**. The question “Make the new keymap active?” appears.

24.5. Click **Yes**.

24.6. Change the “Keymap name” to **MyKeymap**.

24.7. Select **All actions / Editor actions / Move to Code Block End**.

24.8. In “Shortcuts”: Select **Ctrl+]**.

24.9. Click **Add**. The dialog “Enter shortcut” appears.

24.10. Press simultaneously **Ctrl** and ****. The key combination appears in the dialog.

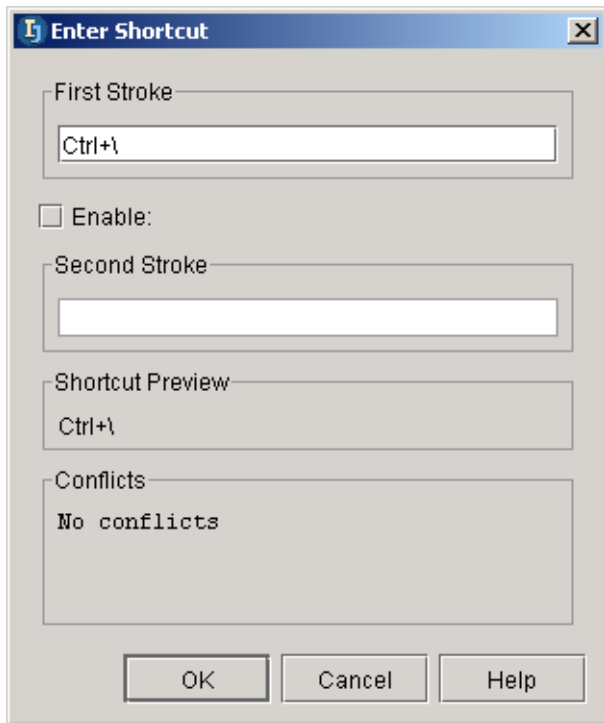


Figure 24.3. New keystroke (423)

24.11. Click **OK**. The keystroke appears in the list.

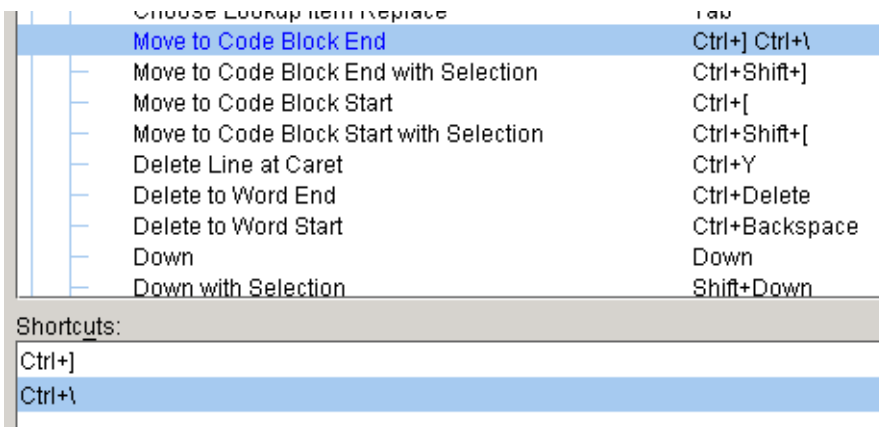


Figure 24.4. New keystroke in the list (422)

~~28. Glossary XXX~~

~~As a table~~

Term1PPP	Definition of term 1. Multiple paragraphs and pics ok.
Term2PPP	Definition of term 1. Multiple paragraphs and pics ok.

~~As text (in alphabetical order)~~

~~Term1. Definition of term 1.~~

~~Multiple paragraphs and pics ok.~~

~~Term2. Definition of term 1.~~

~~Multiple paragraphs and pics ok.~~

List of figures

2.1. J2SDK installation welcome (321).....	19
2.2. J2SDK installation license agreement (320).	19
2.3. J2SDK installation destination (319).....	20
2.4. J2SDK installation components (317).....	20
2.5. J2SDK installation browser select (318).....	20
2.6. J2SDK installation complete (316).....	21
2.7. Windows installer download page (322).....	22
2.8. IDEA EXE installation introduction (336).....	22
2.9. IDEA EXE installation license agreement (335)	23
2.10. IDEA EXE installation choose folder (333) .	23
2.11. IDEA folder for storing settings (192).....	23
2.12. IDEA folder for data cache (191).....	23
2.13. IDEA EXE installation shortcut folder (332)	24
2.14. IDEA EXE installation file associations (331)	24
2.15. IDEA EXE installation pre-install summary (330)	24
2.16. IDEA EXE installation readme (329).....	25
2.17. IDEA EXE installation readme.txt (328).....	25
2.18. IDEA EXE install complete (327).....	25
2.19. License form (254).....	26
2.20. License data dialog (252).....	27
2.21. License agreement dialog (251).....	27
2.22. New project wizard (250).....	27
3.1. Project name (187).....	32
3.2. Popup “New Folder” (248).....	32
3.3. New folder name (052,186).....	32
3.4. Project name / location (053).....	32
3.5. Select JDK (190).....	33
3.6. Dialog “Configure JDKs” (185).....	33
3.7. Dialog “Select JDK Home Directory” (183)...	33
3.8. Dialog with configured JDK (182).....	33
3.9. Configure JDKs (189).....	34
3.10. Configured JDK (188).....	34
3.11. Output folder name (581).....	35
3.12. Select folder Output (246).....	35
3.13. Compiler output folder (579).....	35
3.14. Project path (054).....	35
3.15. Source paths (055).....	36
3.16. Create folder \src confirmation (056).....	36
3.17. Class paths (057).....	36
3.18. Main dialog and tip of the day (180).....	37
3.19. IDEA main dialog and project tool (591).....	37
3.20. New package (574).....	38
3.21. New package dialog (244).....	38
3.22. Created package (573).....	38
3.23. New class (243).....	39
3.24. New class (242).....	39
3.25. Class MyClass (572).....	39
3.26. Unsaved changes indicator (561,560).....	40
3.27. Class structure (559).....	40
3.28. Both Project and Structure panes are displayed	(558).....
3.29. Duplicated lines (0002).....	41
3.30. Undo / Redo create class (0003,0004).....	41
3.31. Find text (0005).....	42
3.32. Goto class (0006).....	42
3.33. Added bookmark (0007).....	42
3.34. List of bookmarks (0008).....	42
3.35. Colors and fonts (0009).....	43
3.36. Autoindented lines (0010).....	44
3.37. Reformat code (0010).....	44
3.38. Reformatted code (0012).....	44
3.39. Error indication (0015).....	45
3.40. Unused import warning (0013).....	45
3.41. Optimize imports suggestion (0014).....	45
3.42. TODO tool (0016).....	46
3.43. Enter text “psvm” (571).....	47
3.44. “psvm” replaced with live template text (570)	47
3.45. Enter text “itar” (569).....	47
3.46. itar live template text (568).....	47
3.47. Modified itar text (566).....	48
3.48. sout (565).....	48
3.49. sout live template text (564).....	48
3.50. Rename variable (0017).....	49
3.51. Refactoring preview / results (0018,0019)..	49
3.52. Renamed variable in history (0020).....	50
3.53. Rolled-back changes (0021).....	50
3.54. JavaDoc method tags (0022).....	51
3.55. Quick JavaDoc for main (0023).....	51
3.56. Quick JavaDoc for main (0023).....	51
3.57. Generated JavaDoc for MyClass (0025)....	52
3.58. Message pane with error information (555)	53
3.59. Compile file (241).....	53
3.60. Application “Unnamed” (553).....	54
3.61. Choose main class (552,551).....	54
3.62. Application settings (550).....	55
3.63. Run pane output (549).....	56
3.64. Set a line breakpoint (548).....	57
3.65. Dialog breakpoints (547).....	57
3.66. Debug pane (545).....	58
3.67. Step over (543).....	58
3.68. Disable breakpoint (541,540).....	59
3.69. Application finishes (539).....	59
4.1. IDEA settings tab icons (a003,a001,a002)...	63
4.2. Project properties settings tab icons	(a004,a005,a006).....
4.3. Open project .ipr file (617).....	64

4.4. "Open in new frame?" (a007).....	64	5.23. Class template (611,195)	93
4.5. New frame (a008).....	64	5.24. 5 views of a file (238,237,236,235)	94
4.6. Project properties tab Paths (468,467,466)..	66	5.25. Members not shown / shown (234,233)	95
4.7. Folder added to project (a011)	67	5.26. Viewing file contents with Commander (232)	95
4.8. Select JDK home directory (a014).....	67	5.27. Structure tool (231).....	97
4.9. Configured JDK (a015).....	67	5.28. Structure tool as subset of Project tool (230)	97
4.10. New target JDK (a016)	67	5.29. File structure popup (607)	98
4.11. New output folder (a017)	68	5.30. Group overriding methods (217)	98
4.12. New compiler output path (a020).....	68	5.31. Group implementation methods (215).....	98
4.13. Folder added to project (612)	69	5.32. Show properties (213)	99
4.14. Folder moved (up and down) (613)	69	5.33. Show methods (212)	99
4.15. Folder removed (from project) (a018,a019)	69	5.34. Show fields (210).....	99
4.16. Select folder or jar (240)	70	5.35. Class hierarchy (605)	100
4.17. Folder added to project (615)	70	5.36. Supertypes hierarchy (228)	101
4.18. Added to classpath (a021,a022).....	71	5.37. Subtypes hierarchy (225)	101
4.19. Class file opened in project tool (a023)	71	5.38. Method hierarchy (224)	102
4.20. Project info in the project tool window (588)	72	5.39. Call hierarchy (caller methods) (223)	102
4.21. Sourcepath (575,587).....	72	5.40. Call hierarchy (callee methods) (220)	103
4.22. Classpath info in main dialog (586)	72	6.1. Autoscroll to source (647,648)	108
4.23. Removed from classpath (a024).....	72	6.2. Dialog "Open file" (653)	109
4.24. Project .ipr files (590 (iws589))	73	6.3. Reload file (209,208,207)	109
4.25. Packages not flattened / flattened (645,646)	76	6.4. List of recent files (651)	109
4.26. New directory (626)	76	6.5. Close file menus (654,655,206)	110
4.27. New directory (239)	76	6.6. Undo (813-819)	111
4.28. New package in Commander (a025,a027,a026)	77	6.7. Select line / word at caret (820,821,822)....	111
4.29. Delete package in project tool (a028).....	78	6.8. Duplicated block (732,733).....	112
4.30. Delete package in commander (a029).....	78	6.9. Delete line at caret (734,735).....	112
4.31. Cut / paste folder (a030,a031).....	79	6.10. Delete to word end / start (736,737,738)..	112
4.32. Copy / paste folder (a032,a033)	79	6.11. Non-insert mode (overwrite) (739,740)	112
5.1. Basic file operations (444,443,446,445)	81	6.12. Left with selection (741,742)	113
5.2. New class (627).....	82	6.13. Up with selection (743,744).....	113
5.3. New class name (628).....	82	6.14. Move to code block start / end (745,746,747).	114
5.4. Select class (205)	84	6.15. Move to code block start with select (748,749)	114
5.5. Safe delete (204)	84	6.16. Move to line end (750,751).....	115
5.6. Usages detected (203,202)	84	6.17. Move to line end with select (750,752).....	115
5.7. Found usages (201).....	84	6.18. Move to next / previous word (756,757,758)	116
5.8. Safe delete action in history (200,199)	85	6.19. Move to next / previous word with select	
5.9. Rollback of deletion (198,197,196)	85	(756,759,760)	116
5.10. File for export in editor (657).....	86	6.20. Move to text end (753,754)	116
5.11. Export to html dialog (656).....	86	6.21. Move to text end with select (753,755).....	117
5.12. File exported to HTML (658).....	86	6.22. Page down (761,762)	118
5.13. Export to html dialog (659).....	87	6.23. Page down with selection (763,764,765) .	118
5.14. Windows print dialog (670)	87	6.24. Go page bottom (766,767)	118
5.15. Default recognized file types (811)	88	6.25. Go page bottom with selection (766,768).	119
5.16. Archive file displayed in IDEA (805,806,807)	89	6.26. Scroll down (769,770)	120
5.17. IDL (810).....	89	6.27. Scroll to center (772,773).....	120
5.18. JavaScript (809).....	90	6.28. Move down and scroll (769,770)	120
5.19. Simple text file (808)	90	6.29. Move down and scroll with selection (769,771)	121
5.20. New recognized extension (421,420)	91	6.30. Indent selection (tab) (774,775)	122
5.21. New file type parameters (419).....	91		
5.22. New file type and extensions (418).....	92		

6.31. Unident selection (tab) (776,777)	122
6.32. Start new line (778,779).....	122
6.33. Split line (778,780).....	122
6.34. Join line with caret and next line (781,782)	122
6.35. Join selected lines (783,784).....	122
6.36. Toggle text case (785,786,787)	123
6.37. Bookmark in file (672).....	124
6.38. Bookmarks editor (673)	124
6.39. View source (674).....	125
6.40. Bookmark description (675).....	125
6.41. Find/replace menu (676).....	126
6.42. Dialog “Find text” (677).....	127
6.43. Drop-down list “Text to find” (678)	127
6.44. Highlight usages (687).....	127
6.45. Dialog “Find in path” (681).....	128
6.46. Tool “Find” (682).....	128
6.47. Dialog “Find usages” (688)	129
6.48. Find tool results (689).....	129
6.49. Dialog “Replace text” (679).....	130
6.50. Dialog “Replace text” (680).....	130
6.51. Dialog “Replace in project” (683).....	131
6.52. Dialog “Replace” (684).....	131
6.53. Find word at caret (685,686).....	132
6.54. Goto menu (690).....	133
6.55. Dialog “Enter class name” (691).....	133
6.56. Dialog “Enter file name” (692).....	134
6.57. Go to declaration (693,694).....	134
6.58. Go to implementation (194)	134
6.59. Go to type declaration (695,696)	135
6.60. Go to super method (698,697).....	135
6.61. Next highlighted error (699).....	135
6.62. Next method (700).....	136
6.63. General colors/fonts (438).....	137
6.64. Colors/fonts for Java files (437).....	138
6.65. Colors/fonts for HTML files (436).....	139
6.66. Colors/fonts for XML files (435)	140
6.67. Colors/fonts for JSP files (434).....	141
6.68. Custom colors/fonts (433).....	142
6.69. Custom color scheme name (789).....	143
6.70. Selecting a color scheme (790).....	143
6.71. Selecting a background color (791).....	143
6.72. New background color (792).....	144
6.73. General code style (431)	145
6.74. Code style / indents and braces (430).....	146
6.75. Code style / blank lines (429)	147
6.76. Code style / spaces (428,427).....	148
6.77. Code style / imports (426).....	149
6.78. Code style / EJB names (793).....	150
6.79. Custom code style scheme name (794) ...	151
6.80. Selecting a code style scheme (795).....	151
6.81. Class declaration indents (796,797,798,799) ..	151
6.82. Dialog “Reformat code” (801).....	152
6.83. Reformatted source file (800,802).....	152
6.84. Reformat according to style checkbox (803)	152
6.85. New class with code style (804).....	152
6.86. Autoindented code (346).....	153
6.87. Dialog “Reformat code” (345).....	154
6.88. Reformatted code (344)	154
6.89. Dialog “Reformat code” (selected text) (343) ..	155
6.90. Reformatted code (342)	155
6.91. Dialog “Reformat code” (selected directory)	
(341).....	155
6.92. Deprecation (720,722).....	156
6.93. Reparse delay (714,716).....	156
6.94. Unused import (715,717).....	157
6.95. Unused symbol (718,719)	157
6.96. Redundant type cast (721,723).....	158
6.97. Wrong package statement (724,725).....	158
6.98. Wrong javadoc tag (726,727)	159
6.99. EJB error (728).....	159
6.100. EJB error (729).....	159
6.101. Added todo text (702,703).....	160
6.102. TODO items for the project (704).....	160
6.103. Todo items for a file (705)	160
6.104. New TODO pattern (706)	161
6.105. New TODO pattern (707)	161
6.106. New TODO in the TODO tool (708)	161
6.107. New TODO filter (709).....	162
6.108. New TODO filter (710).....	162
6.109. Popup for selecting filter (711)	162
6.110. Only filtered TODOs shown (713)	162
7.1. Code automation menu items (417,416,415)	163
7.2. current and recommended dialog (812)	164
7.3. IDEA Settings / Completion (858)	165
7.4. Autopopup (824,823).....	166
7.5. Matching fvmc(p) in import (842).....	167
7.6. Recommended fvmc in import (843)	168
7.7. All matching classes (844)	169
7.8. Case sensitive completion: None (835,836)	170
7.9. Case sensitive completion: First letter (837,838)	170
7.10. Case sensitive completion: All (839,840) ..	171
7.11. Suggestions after myString.c (845).....	172
7.12. Suggestions after myString.co (846).....	172
7.13. Suggestions after myString.com (847).....	172
7.14. Entered text narrowed (848,849).....	173
7.15. Autopopup in javadoc after ‘@’ (826,825)	174
7.16. Lookup list height = 2 (856).....	175
7.17. Insert single ‘)’ (827,828).....	175
7.18. Insert ‘(’ (829,830)	175
7.19. List packages in code (832,831)	176
7.20. No packages in code (833,834)	176

7.21. With / without signature in popup (850,857)	177	7.69. Dialog “Select methods to generate delegates for” (363)	197
7.22. CTRL-P signature (851,852)	177	7.70. Delegates generated (362)	198
7.23. Signature autopopup (851,852)	177	7.71. Line comment (356)	199
7.24. Full signatures (853)	178	7.72. Block comment (355,354)	199
7.25. Autopopup of JavaDoc for suggestion (854,855)	178	8.1. Refactoring menu items (414)	201
7.26. Completion (812)	179	8.2. Dialog “Rename” (for package) (515)	203
7.27. Autopopup (824,823)	180	8.3. Panel “Find - Refactoring preview” (for package) (514)	204
7.28. Autopopup in javadoc after ‘@’ (826,825)	180	8.4. Renamed package (513)	204
7.29. Insert single ‘)’ (827,828)	180	8.5. Dialog “Rename” (for class) (511)	205
7.30. Insert ‘(’ (829,830)	181	8.6. Panel “Find - Refactoring preview” (for class) (510)	205
7.31. List packages in code (832,831)	181	8.7. Renamed class (509,508)	206
7.32. No packages in code (833,834)	181	8.8. Dialog “Rename” (for method) (507)	207
7.33. Case sensitive completion: None (835,836)	182	8.9. Panel “Find - Refactoring preview” (for method) (506)	207
7.34. Case sensitive completion: First letter (837,838)	182	8.10. Renamed method (505,504)	207
7.35. Case sensitive completion: All (839,840)	182	8.11. Dialog “Rename” (for field) (503)	208
7.36. Insert plain text live template (870,871)	184	8.12. Panel “Find - Refactoring preview” (for field) (502)	208
7.37. Insert using list (872)	184	8.13. Renamed field (501)	208
7.38. Final location of cursor (873)	185	8.14. Dialog “Rename” (for variable) (918)	209
7.39. \$END\$ variable (874)	185	8.15. Panel “Find - Refactoring preview” (for field) (919)	209
7.40. Inserted iteration (352)	185	8.16. Renamed variable (920)	209
7.41. Changed iteration variable (351)	185	8.17. Dialog “Rename” (for parameter) (500)	210
7.42. Focus moved (350)	185	8.18. Panel “Find - Refactoring preview” (for parameter) (499)	210
7.43. itar variables (876,875)	185	8.19. Renamed paramter (498)	210
7.44. Surround with {} (877,878)	186	8.20. Move package dialog (380)	211
7.45. Contexts (879)	187	8.21. Find Refactoring preview (379)	212
7.46. Tag pair contexts (880,881)	187	8.22. Package moved (378)	212
7.47. Tag pair in comments (882,883)	187	8.23. Move class dialog (377)	213
7.48. Tag pair in a string (884,885)	188	8.24. Find Refactoring preview (376)	213
7.49. Smart type completion (886)	188	8.25. Class moved (375)	213
7.50. Live template < in HTML file (859,860,861)	188	8.26. Dialog Move Members (374,921)	214
7.51. Edit live template for soutm (887)	189	8.27. Find - Refactoring preview (373)	214
7.52. Edit template variables (888)	190	8.28. Moved class (372,371)	214
7.53. Expand with options (889)	190	8.29. Move inner to upper dialog (370)	215
7.54. Dialog “Surround with” (361)	191	8.30. Find - Refactoring preview (369)	215
7.55. Surrounded with an “if” clause (360)	191	8.31. Moved inner class (368,367)	215
7.56. Popup “Generarte” (359)	192	8.32. Dialog “Change method signature” (413)	216
7.57. Dialog “Choose field to initialize by constructor” (358)	192	8.33. Add parameter (412)	217
7.58. Generated constructor (357)	192	8.34. Find - Refactoring preview (411)	217
7.59. Dialog “Optimize imports” (339)	193	8.35. Parameter added (410)	217
7.60. Optimized imports (338)	193	8.36. Parameter moved (409)	218
7.61. Override methods (453)	194	8.37. Parameter moved (922)	218
7.62. Overridden method toString() (452)	194	8.38. Parameter type change (408)	219
7.63. Overridden method message (451)	194	8.39. Parameter type changed (407)	219
7.64. Overridden method (450)	194	8.40. Dialog “Copy Class” (890)	220
7.65. Override methods (448)	195	8.41. Copied class (891)	220
7.66. Dialog “Select methods to implement” (366)	196		
7.67. Method implemented (365)	196		
7.68. Dialog “Select target to generate delegates for” (364)	197		

8.42. Method to extract (406).....	221	10.1. History dialog (527)	257
8.43. Extracted method (405)	221	10.2. Inserted changes in history dialog (526) ..	258
8.44. Extract interface (404)	222	10.3. Modification in history dialog (523).....	259
8.45. Search usages (924)	222	10.4. Deleted line in history dialog (522).....	259
8.46. No usages can be replaced (925).....	222	10.5. Rollback context menu (521).....	260
8.47. Extracted interface (401,400)	223	10.6. Rollback in history dialog (520)	260
8.48. Extract superclass (399).....	224	10.7. Differences between the original and the current version (525)	261
8.49. Proceed question (398)	224	10.8. First action taken on file (524)	261
8.50. User interface where possible (926).....	224	10.9. Add label dialog (519)	262
8.51. Extracted superclass (397,396).....	225	10.10. Label in history dialog (518)	262
8.52. User interfaces where possible (892)	226	10.11. StarTeam classpath (266).....	265
8.53. Refactoring preview (893).....	227	10.12. VCS Support StarTeam (265)	265
8.54. Refactored class (894).....	227	10.13. StarTeam configuration (264).....	266
8.55. Dialog “Pull members up” (895).....	228	10.14. StarTeam folder added to project (263,262) . 266	
8.56. Members pulled up (896,897).....	228	10.15. Dir, folder added (261)	267
8.57. Dialog “Push members down” (898).....	229	10.16. xxx (260,259).....	267
8.58. Push members down refactoring preview (899) 229		10.17. xxx (258).....	267
8.59. Members pushed down (900,901).....	229	10.18. xxxx (257).....	268
8.60. Introduce variable (382).....	230	10.19. xxx (256).....	268
8.61. Variable introduced (381)	230	11.1. J2SDK docs extraction (933).....	269
8.62. Introduce field (384).....	231	11.2. JavaDoc API dir (313)	269
8.63. Field introduced (383).....	231	11.3. JavaDoc path (934).....	270
8.64. Introduce constant (927???).....	232	11.4. JavaDoc start page (312).....	271
8.65. Constant introduced (928)	232	11.5. Select Applet (311).....	271
8.66. Introduce parameter (929).....	233	11.6. JavaDoc for Applet (310).....	271
8.67. Parameter introduced (930).....	233	11.7. Applet source (309).....	272
8.68. Inlined variable (395)	234	11.8. JavaDoc for Panel (308).....	272
8.69. Inline method (394).....	235	11.9. Quick JavaDoc for Applet (306)	273
8.70. Inline method preview (931)	235	11.10. Applet definition (307)	273
8.71. Inlined method (393,392).....	236	11.11. Quick JavaDoc for Panel (305)	274
8.72. Dialog “Encapsulate fields” (391,390).....	237	11.12. JavaDoc for Panel (302).....	274
8.73. Encapsulate field refactoring preview (389) 238		11.13. Quick JavaDoc for own class (301).....	275
8.74. Encapsulated fields (388,387)	238	11.14. JavaDoc error for MyClass (300)	275
8.75. Replace temp with query (902).....	239	11.15. JavaDoc method tags (299)	276
8.76. Temp replaced with query (903).....	239	11.16. @ popup (298)	276
8.77. Dialog “Convert anonymous to inner” (932) 240		11.17. Exception list (297).....	277
8.78. Anonymous class converted to inner (385) 241		11.18. UnknownError tag added (296).....	277
9.1. Choose inspection scope (904)	243	11.19. Quick JavaDoc for MyClass (295).....	277
9.2. Dialog “Inspect code in file” (905).....	244	11.20. Dialog “Generate JavaDoc” (294)	278
9.3. Inspection tool (906)	244	11.21. Run JavaDoc tool (293).....	279
9.4. Inspection tool (project) (907).....	245	11.22. Generated JavaDoc (935).....	279
9.5. Modification suggested (915).....	246	11.23. Add to JavaDoc API paths (292).....	280
9.6. Modified (916).....	246	11.24. JavaDoc for MyClass (291,936).....	280
9.7. Safe delete (913)	247	12.1. Built class files (964)	284
9.8. Comment out (914).....	247	12.2. Rebuilt class files (965)	284
9.9. Safe delete (909)	248	12.3. Made class files (966)	285
9.10. Safe delete (910)	248	12.4. Compiled class file (967)	285
9.11. Usages detected (911,912)	248	12.5. View previous / next message (968)	286
9.12. xxx (917).....	249	12.6. Dialog “Export preview” (972).....	287
9.13. ?? Recommended changes to History dialog (517)	256	12.7. Hide warnings (974,975)	287

12.8. Autoscroll to source (978).....	288	20.7. xxx (482).....	332
12.9. Exclude file (980,981).....	289	20.8. xxx (492).....	333
12.10. Exclude directory (983,982).....	290	20.9. xxx (481).....	334
12.11. Resource copied to output (984).....	291	20.10. xxx (481).....	335
12.12. Compile progress (985).....	291	20.11. xxx (479).....	335
12.13. Synchronized output directory (986).....	291	20.12. xxx (478).....	336
12.14. No debug info available (987,988).....	292	20.13. xxx (477).....	336
12.15. Debug info available (990,989).....	292	20.14. xxxut (476).....	336
12.16. Deprecation warning (992,991).....	293	20.15. xxxut (475).....	337
12.17. No warnings (993).....	293	20.16. xxxut (474).....	337
12.18. No breakpoint stop (996,995).....	294	20.17. xxxut (491).....	338
12.19. Breakpoint stop (998,997).....	294	20.18. xxxut (490).....	339
13.1. Ant extraction (938).....	295	20.19. xxxut (489).....	340
13.2. JavaDoc API dir (313).....	295	20.20. xxxut (488).....	341
13.3. Ant directories (273).....	296	20.21. xxxut (487).....	341
13.4. Environment variables for Ant (272).....	296	20.22. xxxut (486).....	341
13.5. build.xml (271).....	297	21.1. www.intellij.org start page (538,537).....	345
13.6. Adding build.xml to Ant tool (270).....	298	21.2. HtmlTools (536).....	345
13.7. Build file added (269).....	298	21.3. HTML main menu item (535).....	346
13.8. Files are built (268,267).....	299	21.4. Simple HTML file (534).....	346
14.1. Application default settings (289).....	302	21.5. HTML bold tags added to text (533).....	346
14.2. Copied configuration (287).....	303	21.6. xxx (465,464).....	347
14.3. Delete configuration confirmation (940)....	303	21.7. Sample.jar contents (463).....	350
14.4. Exception (941).....	304	21.8. xxx (462).....	350
14.5. Enter Arithmetic exception (942).....	305	21.9. ActionsSample.jar contents (461).....	354
14.6. Exception breakpoint (282,281).....	305	21.10. xxx (460,459).....	354
14.7. Exception breakpoint (943).....	305	21.11. SimpleToolWindow.jar contents (457)....	357
14.8. Add field watchpoint (279).....	306	21.12. xxx (456).....	357
14.9. Field watchpoint added (278,277).....	306	21.13. VfsSample.jar contents (455).....	360
14.10. Field watchpoint info in debug (276).....	306	21.14. xxx (454).....	361
14.11. Method breakpoint (275,274).....	307	22.1. Selecting the external tool (532).....	363
14.12. Pause (956).....	308	22.2. External tool settings (531).....	363
14.13. Stop (958).....	308	22.3. Tool in list of external tools (530).....	364
14.14. Breakpoint (945).....	309	22.4. External tool in Tools menu (529).....	365
14.15. Step over (947).....	309	22.5. Message about external tool being opened (528)	365
14.16. Step into (949).....	310	23.1.	367
14.17. Step out (951).....	310	24.1. IDEA Options / Keymaps (425).....	375
14.18. Cursor on 3rd method (952).....	311	24.2. Set active keymap (424).....	376
14.19. Run to cursor (954).....	311	24.3. New keystroke (423).....	377
14.20. Show execution point (961).....	311	24.4. New keystroke in the list (422).....	377
15.1. Junit test (472).....	313		
16.1. xxx ().....	315		
17.1.	319		
18.1.	321		
19.1.	323		
19.2. xxx (473).....	327		
20.1. xxx (485).....	329		
20.2. xxx (497,484).....	330		
20.3. xxx (496).....	330		
20.4. xxx (483).....	331		
20.5. xxx (495).....	331		
20.6. xxx (494).....	332		

Index

20021024TTT index entries added thruout doc (for release version chapters)
(the following number is the last page marker.... required by Framemaker).

394			
Symbols			
\$END\$	185/7.154		
/**	51/3.97, 276/11.20		
@	276/11.23		
A			
Abbreviation (live template)	189/7.51		
Accept all terms of the license	27/2.30		
Action column (history dialog)	261/10.22		
Add (directory)	69/4.2.1.1.3.1		
Add (live template)	190/8.3.1.3.2		
Add (to classpath)	71/4.2.1.1.5.1		
Add (to sourcepath)	70/4.2.1.1.4.1		
All	171/7.40		
All files in directory...	193/8.5.2		
All invocations and remove the method (button)	235/8.150		
Application configuration (create)	54/3.10.1		
Application default settings (dialog)	302/14.4		
Apply (code style scheme)	151/7.5.7.4		
Apply (color scheme)	144/6.90		
Archive	88/5.15		
Autocomplete common prefix	172/7.50		
Autoindent	153/7.5.8		
Auto-narrow	172/8.1.2.4.2		
Autopopup after dot (checkbox)	166/7.2		
Autopopup in ()	177/8.1.5.4.2.2		
Autopopup javadoc (checkbox)	178/7.87		
Autoreparse delay	156/6.118		
Autoscroll from source (structure tool)	99/6.3.4.9		
Autoscroll to source	101/6.3.5.1.7, 102/6.3.5.2.5, 103/6.3.5.3.8, 108/7.1.1.2		
Autoscroll to source (icon)	287/13.2.7		
Autoscroll to source (structure tool)	99/6.3.4.8		
B			
B			
Surround with {}	185/7.160		
Back	136/7.3.3.10		
BACKSPACE	113/7.2.4.1		
Basic	167/8.1.2.1		
Blank lines (code style)	147/7.5.3		
Bookmarks	124/7.3.1		
Breakpoint			
Step into	309/15.3.2		
Step out	309/15.3.2		
Step over	309/15.3.2		
Breakpoint set	57/3.10.3		
Breakpoints	304/15.2		
Exception	304/15.2.2		
Line	304/15.2.1		
Method	307/15.2.4		
Bugs/features database	16/		
Build	283/13.1.1		
Build / Compile	53/3.105		
Build Compile	285/12.12		
Build Make project	285/12.8		
Build Rebuild project	283/12.4, 284/12.5		
C			
C:\IntelliJ-IDEA-3.0\docs\api	269/11.1		
C:\IntelliJ-IDEA-3.0\docs\api\index.html	271/11.4		
C:\IntelliJ-IDEA-3.0\MyJavaDoc	278/11.33		
C:\MyProjectFolder\export.txt	286/12.17		
C:\MyProjectFolder\MyOutputFolder\MyPackage\MyClass.class	53/3.107		
C:\MyProjectFolder\MyProject.ipr	73/4.37		
C:\MyProjectFolder\src	36/3.23		
Call hierarchy	102/6.3.5.3		
Callee methods hierarchy	102/6.3.5.3.3		
Caller methods hierarchy	102/6.3.5.3.2		
Case sensitivity	170/8.1.2.4.1		
All	170/8.1.2.4.1.3		
Auto-narrow	172/8.1.2.4.2		
First letter	170/8.1.2.4.1.2		
None	170/8.1.2.4.1.1		
Change method signature (refactor)	216/9.4		
Add param	216/9.4.1		
Change name	219/9.4.3		
Change type	219/9.4.4		
Move param	218/9.4.2		
Choose Main Class (dialog)	54/3.111		
Class			
Go to	133/7.3.3.1		
Class hierarchy	100/6.3.5.1.1		
Class path	36/3.1.3.4		
Classpath	71/4.2.1.1.5		
Close	110/7.1.4		

Close (icon)	286/13.2.2, 308/14.48	CTRL-D	112/7.2.3
Close (project)	64/4.1.3	CTRL-DEL	112/6.9
Close Active	110/7.1.4.1	CTRL-DOWN	120/7.2.4.6
Close All	110/7.1.4.2	CTRL-END	116/6.19
Close All but current	110/7.1.4.3	CTRL-ENTER	122/6.32
Close All But This	110/*	CTRL-HOME	116/6.20
Code Autoindent lines	153/6.107	CTRL-LEFT	116/7.2.4.4
Code Comment with block comment	199/7.214	CTRL-M	120/6.26
Code Comment with line comment	199/7.212	CTRL-P	177/8.1.5.4.2.1
Code Delegate methods	197/7.206	CTRL-PAGE DOWN	118/6.23
Code Generated...	192/7.180	CTRL-PAGE UP	118/6.24
Code Implement methods	196/7.199	Ctrl-Q	277/11.29
Code Insert live template	184/7.36	CTRL-RIGHT	116/7.2.4.4
Code Override methods	194/7.189	CTRL-S	39/3.37
Code Surround with live template	185/7.159	CTRL-SHIFT-]	114/6.14
Code Surround with...	191/7.177	CTRL-SHIFT-DOWN	120/7.2.4.6
Code Automation	163/8	CTRL-SHIFT-END	116/6.20
Code block	114/7.2.4.2	Ctrl-Shift-F8	57/3.65
Code completion	164/8.1	CTRL-SHIFT-HOME	117/6.21
Code generation	192/8.4	CTRL-SHIFT-J	122/6.33
Code style	145/7.5	CTRL-SHIFT-LEFT	116/6.18
Code templates	184/8.3	CTRL-SHIFT-PAGE DOWN	119/6.24
Collapse all (icon)	287/13.2.5	CTRL-SHIFT-PAGE UP	119/6.25
Color Schemes	143/7.4.7	CTRL-SHIFT-RIGHT	116/6.18
Colors	137/7.4	CTRL-SHIFT-SPACE	168/8.1.2.2
Commander	95/6.3.3	CTRL-SHIFT-U	123/7.2.6
Comment	199/8.9	CTRL-SHIFT-UP	120/6.26
Comment column (history dialog)	261/10.21	CTRL-SHIFT-W	111/6.7
Compile	53/3.9, 285/13.1.3	CTRL-SHIFT-Z	111/6.6
Compiler	283/13	CTRL-SPACE	166/7.10, 167/8.1.2.1
Compiler messages	286/13.2	CTRL-UP	120/6.26
Constructor	192/7.181	CTRL-V	112/7.2.3
Context (live template)	187/8.3.1.2, 190/7.53	CTRL-W	111/7.2.2
Convert Anonymous to Inner (refactor)	240/9.13	CTRL-X	112/7.2.3
Copy	112/7.2.3	CTRL-Y	112/6.8
Copy (live template)	190/8.3.1.3.2	CTRL-Z	111/7.2.1
Copy class (refactor)	220/9.5	Current file (tab)	160/6.132
Copy configuration (icon)	303/14.8	Custom (colors and fonts)	142/7.4.6
Copyright	5/	Custom file types	91/6.2.2
Create		Cut	112/7.2.3
Class	82/6.1.1.1		
File	82/6.1.1.3	D	
Interface	82/6.1.1.2		
Create (code style scheme)	151/7.5.7.1	Debug	54/3.10
Create (color scheme)	143/7.4.7.1	Debug (configuration)	301/15.1
Create (package)	38/3.2	Debug (icon)	303/14.17
Create (project)	32/3.1	Debugger	301/15
Create bookmark	124/7.3.1.1	Declaration	
CTRL-]	114/7.2.4.2	Go to	134/7.3.3.4
CTRL-ALT-SPACE	169/8.1.2.3	Default file types	88/6.2.1
CTRL-BACKSPACE	112/6.9	Default value (live template variable)	190/7.52
CTRL-C	112/7.2.3	Delete (project)	65/4.1.5

Delete directory	78/5.1.2
Delete line at caret	112/6.8
Delete package	78/5.1.2
Delete to word end	112/6.9
Delete to word start	112/6.9
Deprecated symbol	156/7.6.1
Describe bookmark	125/7.3.1.5
Description (live template)	189/7.51
Directories	75/5
Directory	69/4.2.1.1.3
Disable	58/3.125
Do Refactor	204/8.7
Documentation	15/1
DOWN	113/7.2.4.1
Down with selection	113/6.13
Downloads	16/•
Duplicate line or block	112/7.2.3

E

EAP	18/•
Early access builds	16/•
Edit Show Bookmarks	124/6.19
Edit Toggle Bookmark	124/6.17
Edit Live Template (dialog)	189/7.174
Edit template variables (dialog)	189/7.51
Edit variables (live template)	189/7.51
Editor	95/6.3.1, 107/7
Editor bookmarks (dialog)	124/6.19
EJB errors	159/7.6.11
EJB Names (code style)	150/7.5.6
EJB warnings	159/7.6.12
Email	18/•
Encapsulate field (refactor)	237/9.11
END	115/7.2.4.3
Error indication	156/7.6
Evaluation license	26/2.2.2
Exception breakpoints (tab)	304/14.24
Expand all (icon)	287/13.2.5
Expand with (live template)	190/7.52
Export preview (dialog)	286/12.16
Export to HTML	86/6.1.7
Export to text file (icon)	286/13.2.4
Expression (live template variable)	190/7.52
Extension (file)	91/6.2.2.1
Extract (refactor)	221/9.6
Interface	222/9.6.2
Method	221/9.6.1
Superclass	224/9.6.3

F

F8	58/3.122
F9	58/3.123
FAQ	16/•
Features list	16/•
Field watchpoints (tab)	306/14.31
File	
Go to	133/7.3.3.2
File (reformat)	154/7.5.9.1
File Close Active Editor	110/•
File Close All Editors	110/•
File Close All Editors But Current	110/•
File Close Project	64/4.3
File Export to HTML...	86/5.27
File Open file...	108/6.5
File Open project	64/4.1
File Print...	87/5.31
File Project properties...	66/4.5
File Reload from disk	109/6.8
File Reopen	65/4.4
File Save All	110/6.12
File Synchronize	110/6.11
File templates (dialog)	93/5.43
File types	88/6.2
Files	81/6
Files in directory (reformat)	155/7.5.9.3
Filters	162/7.7.3
Find (in file)	126/7.3.2.1
Find in path	128/7.3.2.6
Find Next	127/7.3.2.4
Find Previous	128/7.3.2.5
Find text (dialog)	126/6.30
Find usages (dialog)	129/6.41
Find usages (in path)	129/7.3.2.7
Find usages in file	127/7.3.2.3
Find Word at caret	131/7.3.2.10
First letter	170/7.35
Flatten packages (icon)	76/4.39
Fonts	137/7.4
Forums	18/•
Forward	136/7.3.3.10
From a view (open)	108/7.1.1.1
Full signatures	178/8.1.5.4.2.3

G

General (code style)	145/7.5.1
General (colors and fonts)	137/7.4.1
Go page bottom	118/6.23
Go page bottom with selection	119/6.24
Go page top	118/6.24

Go page top with selection	119/6.25
Go to	133/7.3.3
Go to (bookmark)	125/7.3.1.4
Go to / Last edit location	136/7.3.3.11
Go to / Line	134/7.3.3.3
Go to Class	133/6.57
Go to Declaration	134/6.64
Go to File	133/6.60
Go to Implementation	134/6.67
Go to Next highlighted error	135/7.3.3.8
Go to Next method	135/7.3.3.9
Go to Previous highlighted error	135/7.3.3.8
Go to Previous method	135/7.3.3.9
Go to Super method	135/6.72
Go to Type declaration	134/6.70
Group (live template)	189/7.51
Group implementation methods (structure tool)	98/6.3.4.4
Group overriding methods (structure tool)	98/6.3.4.3

H

Help	17/•
Hide warnings (icon)	287/13.2.6
Hierarchy tool	100/6.3.5
Highlight usages in file	127/7.3.2.2
History dialog	
Delete	259/11.1.4
Insert	258/11.1.2
Modify	259/11.1.3
Open	256/11.1.1
Rollback	260/11.1.5
HOME	115/6.16
HTML (colors and fonts)	139/7.4.3
HTML (context)	188/8.3.1.2.5
HTML (file type)	89/5.16
http://java.sun.com/j2se/downloads.html	19/2.1
http://www.intellij.com/idea/download.jsp	22/2.9
http://www.intellij.com/idea/evaluate.jsp	26/2.25

I

IDEA Options File types	91/6.2.2
IDEA Settings Completion	166/7.1
IDL (file type)	89/5.16
If (surround with)	191/7.178
Implementation	
Go to	134/7.3.3.5
Import optimization	193/8.5
File	193/8.5.1
Files in dir	193/8.5.2
Imports (code style)	149/7.5.5

Include packages	176/8.1.5.3
Incremental search	132/7.3.2.11
Indent selection	122/7.2.5
Indents and braces (code style)	146/7.5.2
Inline (refactor)	234/9.10
Method	235/9.10.2
Variable	234/9.10.1
INSERT	112/6.10
Insert single ‘(175/8.1.5.2
Installation	19/2
Installation (IDEA)	22/2.2
Interface implementation	196/8.7
Introduce (refactor)	230/9.9
Constant	232/9.9.3
Field	231/9.9.2
Parameter	233/9.9.4
Variable	230/9.9.1
ipr	64/4.1.2, 73/4.2.4
itar	47/3.81, 185/7.155
ITN	18/•

J

j2sdk-1_4_1-doc.zip	269/11.1
j2sdk-1_4_1-windows-i586.exe	19/2.1
Java (colors and fonts)	138/7.4.2
Java 2 SDK	19/2.1
Java class files	89/5.16
Java code (context)	187/8.3.1.2.1
Java comment (context)	187/8.3.1.2.2
Java source (file type)	89/5.17
Java string (context)	187/8.3.1.2.3
java.sun.com	269/11.1
JavaDoc	269/12
JDK	269/12.1
Tags	276/12.3
View	271/12.1.2
JavaDoc (own)	
Generate	278/12.4.1
Install	280/12.4.2
View	280/12.4.3
JavaDoc Paths (tab)	270/11.3
JavaDocQuick	273/12.2
JDK class	273/12.2.1
Own class	275/12.2.2
Javadocs errors	158/7.6.9
JavaScript (file type)	89/5.17
jetbrains.intellij.documentation	15/1.1
Join lines	122/6.33
JSP (colors and fonts)	141/7.4.5
JSP (context)	188/8.3.1.2.7
JSP (file type)	89/5.17

L

Labels (history)	262/11.1.7
Last edit location	
Go to	136/7.3.3.11
LEFT	113/7.2.4.1
Left with selection	113/7.2.4.1
License key	27/2.27
License keys	16/•
Line	
Go to	134/7.3.3.3
List packages in code (checkbox)	176/7.64
Llve template	
Edit	189/8.3.1.3.1
Live Templates	184/8.3.1
Local (version control)	256/11.1
Local VCS Add label	262/10.25
Local VCS Show history	256/10.3
Lookupt list height	175/8.1.5.1

M

Main class	54/3.111
Make	285/13.1.2
Manual show in ()	177/8.1.5.4.2.1
Method delegation	197/8.8
Method hierarchy	101/6.3.5.2
Method override	194/8.6
Modify (code style scheme)	151/7.5.7.3
Modify (color scheme)	143/7.4.7.3
Move	113/7.2.4
Move (directory)	69/4.2.1.1.3.2
Move (refactor)	211/9.3
Class	213/9.3.2
Inner to upper level	215/9.3.4
Members	214/9.3.3
Package	211/9.3.1
Move (within classpath)	71/4.2.1.1.5.2
Move (within sourcepath)	70/4.2.1.1.4.2
Move down (directory)	69/4.2.0
Move down and scroll	120/6.27
Move down and scroll with selection	120/6.28
Move down bookmark	125/6.25
Move line end	115/7.2.4.3
Move line end with selection	115/6.16
Move line start	115/6.16
Move line start with selection	115/6.17
Move text end	116/6.19
Move text end with selection	116/6.20
Move text start	116/6.20
Move text start with selection	117/6.21
Move to code block end	114/7.2.4.2

Move to code block end with selection	114/6.14
Move to code block start	114/7.2.4.2
Move to code block start with selection	114/6.14
Move to next word	116/7.2.4.4
Move to next word with selection	116/6.18
Move to previous word	116/7.2.4.4
Move to previous word with selection	116/6.18
Move up (directory)	69/4.20
Move up and scroll	120/6.28
Move up and scroll with selection	121/6.29
Move up bookmark	125/6.25
MoveUp	218/8.70
MyApplication	54/3.110
MyCodeStyleScheme	151/6.94
MyColorScheme	143/6.84
myPackage	38/3.31
MyProject	32/3.1
MyProjectFolder	32/3.5

N

Name (live template variable)	190/7.52
Narrow down on typing (checkbox)	172/7.44
New (project)	64/4.1.1
New Class	39/3.34, 82/5.2, 256/10.1
New Directory	76/4.41
New File	82/5.10
New Interface	82/5.6
New Package	38/3.30
New Folder	32/3.4
news://news.jetbrains.com	15/•
Newsgroup	15/1.1
Next highlighted error	
Go to	135/7.3.3.8
Next message (icon)	286/13.2.3
Next method	
Go to	135/7.3.3.9
None	170/7.28

O

Online support	16/•
Open	108/7.1.1
Open (project)	64/4.1.2
OpenAPI	16/•
Optimize imports (dialog)	193/7.185
Options File templates	152/6.101
Options File templates...	93/5.43
Options Live templates...	185/7.153
Other (context)	188/8.3.1.2.8
Output path	35/3.1.3.1

P

- Packages **75/5**
- PAGE DOWN **118/7.2.4.5**
- Page down **118/7.2.4.5**
- Page down with selection **118/6.22**
- PAGE UP **118/6.22**
- Page up **118/6.22**
- Page up with selection **118/6.23**
- Paste **112/7.2.3**
- Patterns (todo) **161/7.7.2**
- Pause (icon) **308/14.45**
- Phone **18/•**
- Plugins **16/•**
- Preview usages to be changed . **226/8.110, 229/8.123**
- Preview usages to be changed (checkbox) **217/8.65**
- Previous highlighted error
 - Go to **135/7.3.3.8**
- Previous message (icon) **286/13.2.3**
- Previous method
 - Go to **135/7.3.3.9**
- Print **87/6.1.8**
- Programs | IntelliJ IDEA 3.0 | IntelliJ IDEA **27/2.26**
- Project location **32/3.1.1**
- Project name **32/3.1.1**
- Project path **35/3.1.3.2**
- Project paths **35/3.1.3**
- Project properties (dialog) **66/4.2.1**
- Project tool **72/4.2.2, 95/6.3.2**
- Project xml files **73/4.2.4**
- Projects **63/4**
- psvm **47/3.7.9**
- Pull members (refactor) **228/9.8**
- Purchasing info **16/•**
- Push down (refactor) **229/9.8.2**
- Push members (refactor) **228/9.8**

Q

- Quick Start **17/•**

R

- Rebuild **283/13.1.1**
- Recent files **109/7.1.1.5**
- Recover **85/6.1.6**
- Redo **111/7.2.1**
- Redundant type cast **157/7.6.6**
- Refactor | Change method signature **216/8.60**
- Refactor | Convert anonymous to inner... **240/8.167**
- Refactor | Copy **220/8.83**
- Refactor | Encapsulate fields... **237/8.156**

- Refactor | Extract interface **222/8.92**
- Refactor | Extract method **221/8.88**
- Refactor | Extract superclass **224/8.99**
- Refactor | Inline **234/8.144, 235/8.149**
- Refactor | Introduce constant **232/8.136**
- Refactor | introduce field **231/8.132**
- Refactor | Introduce parameter **233/8.140**
- Refactor | introduce variable **230/8.128**
- Refactor | Move .. **211/8.40, 213/8.45, 214/8.50, 215/8.55**
- Refactor | Pull members up... **228/8.115**
- Refactor | Push members down... **229/8.120**
- Refactor | Rename **203/8.4, 205/8.9, 207/8.15, 208/8.21, 209/8.27, 210/8.33**
- Refactor | Replace temp with query... **239/8.162**
- Refactor | Use interfaces where possible... **226/8.108**
- Refactoring **201/9**
- Reformat **154/7.5.9**
- Reformat according to style (checkbox) . . . **152/6.102**
- Reformat according to style (live template) . **190/7.53**
- Refresh **101/6.3.5.1.6, 102/6.3.5.2.4, 103/6.3.5.3.7**
- Reload from disk **109/7.1.1.4**
- Remove (directory) **69/4.2.1**
- Remove (from classpath) **71/4.2.1.1.5.3**
- Remove (live template) **190/8.3.1.3.3**
- Remove (within sourcepath) **70/4.2.1.1.4.3**
- Remove all bookmarks **125/6.27**
- Remove bookmark **125/7.3.1.7**
- Remove configuration (icon) **303/14.10**
- Rename (refactor) **203/9.2**
 - Class **205/9.2.2**
 - Field **208/9.2.4**
 - Method **207/9.2.3**
 - Package **203/9.2.1**
 - Parameter **210/9.2.6**
 - Variable **209/9.2.5**
- Reopen (project) **65/4.1.4**
- Reparse delay **156/7.6.2**
- Replace (in file) **129/7.3.2.8**
- Replace in path **130/7.3.2.9**
- Replace temp with query (refactor) **239/9.12**
- Replace text (dialog) **129/6.45**
- Rerun safe delete **84/5.17**
- Resume **58/3.123**
- RIGHT **113/7.2.4.1**
- Right with selection **113/6.12**
- Rollback **85/5.24**
- Rollback (history dialog) **260/11.1.5**
- Run
 - Close **308/15.3.1**
 - Pause **308/15.3.1**
 - Resume **308/15.3.1**

Stop	308/15.3.1	SHIFT-HOME	115/6.17
Run (configuration)	301/15.1	SHIFT-LEFT	113/7.2.4.1
Run (icon)	303/14.13	SHIFT-PAGE DOWN	118/6.22
Run / Debug	58/3.120	SHIFT-PAGE UP	118/6.23
Run / View Breakpoints	57/3.65	SHIFT-RIGHT	113/6.12
Run Add method breakpoint...	307/14.40	SHIFT-TAB	122/6.30
Run Resume program	58/3.123	SHIFT-UP	113/6.12
Run Run	54/3.108	Shorten FQ names (live template)	190/7.53
Run Step	58/3.122	Show bookmark	124/7.3.1.2
Run View breakpoints...	304/14.23	Show execution point	311/15.3.4
Run configuration	56/3.10.2	Show execution point (icon)	311/14.60
Run to cursor	311/15.3.3	Show fields (structure tool)	99/6.3.4.7
Run/Debug configuration		Show full signatures (checkbox)	178/7.83
Edit	302/15.1.2.2	Show hierarchy	101/6.3.5.2.1, 102/6.3.5.3.1
Run/debug configuration		Show javadoc	178/8.1.5.5
Add	302/15.1.2.1	Show members (icon)	95/5.48
Copy	303/15.1.2.4	Show methods (structure tool)	99/6.3.4.6
Debug	303/15.1.2.7	Show packages	101/6.3.5.1.5, 102/6.3.5.2.3, 103/6.3.5.3.5
Delete	303/15.1.2.5	Show properties (structure tool)	98/6.3.4.5
Edit Defaults	302/14.4	Show signature	177/8.1.5.4
Modify defaults	302/15.1.2.3	Silly assignment	158/7.6.7
Remove	303/15.1.2.5	Skip if defined (live template variable)	190/7.52
Run	303/15.1.2.6	Smart type completion (context)	188/8.3.1.2.4
S		Smarttype	168/8.1.2.2
Save all	110/7.1.3	Sort alphabetically	101/6.3.5.1.4, 102/6.3.5.2.2, 103/6.3.5.3.4
Schemes (code style)	151/7.5.7	Sort alphabetically (structure tool)	98/6.3.4.2
Scope	103/6.3.5.3.6	Source path	35/3.1.3.3
Scroll	113/7.2.4, 120/7.2.4.6	sout	48/3.84, 184/7.152
Scroll down	120/7.2.4.6	soutm	189/7.174
Scroll to center	120/6.26	Spaces (code style)	148/7.5.4
Scroll up	120/6.26	Split line	122/6.32
Search Find	126/6.30	Start IDEA	27/2.2.3
Search Find in path...	128/6.37	Start new line	122/6.31
Search Find next	127/6.34	Step into (icon)	309/14.53
Search Find previous	128/6.35	Step out (icon)	310/14.54
Search Find usages	129/6.41	Step Over	58/3.122
Search Find word at caret	131/6.56	Step through application	58/3.10.4
Search Replace	129/6.45	Stop (icon)	286/13.2.1, 308/14.47
Search Replace in path	130/6.51	Structure (tab)	40/3.38
Select	111/7.2.2	Structure tool	96/6.3.4
Select (code style scheme)	151/7.5.7.2	Subtypes hierarchy	101/6.3.5.1.3
Select (color scheme)	143/7.4.7.2	Suggest after @	174/8.1.4
Select line at caret	111/7.2.2	Suggest after dot	166/8.1.1
Select word at caret	111/7.2.2	Suggest after partial text	167/8.1.2
Selected text (reformat)	155/7.5.9.2	Suggest in XML	174/8.1.3
SHIFT-DOWN	113/6.13	Suggestions	16/
SHIFT-END	115/6.16	Super method	
SHIFT-ENTER	122/6.31	Go to	135/7.3.3.7
Shift-F1	272/11.9	Supertypes hierarchy	100/6.3.5.1.2
Shift-F9	58/3.120	Support	15/1, 18/1.4

Surround with **191**/8.3.2
Synchronize **110**/7.1.2

T

TAB **47**/3.80, **184**/7.150
Tab **122**/7.2.5
Template text (live template) **189**/7.51
Text (file type) **90**/5.18
Text editing **111**/7.2
Todo **160**/7.7
Toggle case **123**/7.2.6
Toggle insert/override **112**/6.10
Tool Tips **17**/
Tools | Generate JavaDoc... **278**/11.31
Tools | Local VCS | Show history... **85**/5.21
Tools | Optimize imports... **193**/7.185
Tools | Reformat code... . **151**/6.98, **154**/6.109, **155**/
6.113
Tracker **18**/
Tutorial **17**/
Type declaration
 Go to **134**/7.3.3.6
Type hierarchy **100**/6.3.5.1

U

Undo **111**/7.2.1
Unindent selection **122**/6.30
Unknown Javadoc tags **158**/7.6.10
Unselect word at caret **111**/6.7
Unused import **157**/7.6.3
Unused symbol **157**/7.6.4
Unused throws declaration **157**/7.6.5
UP **113**/7.2.4.1
Up with selection **113**/6.12
Use interface (refactor) **226**/9.7
User Guide **17**/
User name **27**/2.27

V

Version control **255**/11
View | External Java doc **271**/11.6
View | External JavaDoc **280**/11.37
View | Recent files... **109**/6.9
View breakpoints **311**/15.3.5
View breakpoints (icon) **311**/14.61
View source of bookmark **124**/7.3.1.3
View usages **84**/5.16
Views
 File **94**/6.3

W

Watchpoints **304**/15.2
 Field **306**/15.2.3
Wrong package statement **158**/7.6.8
www.intellij.com **16**/
www.intellij.net **16**/
www.intellij.org **16**/

X

XML (colors and fonts) **140**/7.4.4
XML (context) **188**/8.3.1.2.6
XML (file type) **90**/5.19