IntelliJ IDEA

Quick Start

2002.10.18 20:49
IDEA build 651
J2SE 1.4.1

ABOUT THIS DOC

this is the INTERNAL version of the QS guide (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 witih “XXX”
- incomplete chapters/sections are marked witih “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.
- change this to a “tutorial” or “getting started”, and take a subportion and call it “quick start”? i have really made this file too big to be a quick start.

WHATS NEW

- 20021018 : editted
  - page 1 of parts a,b,c,d updated.
  - 9. Code Refactoring X (page 183)
  - 10. Code Inspection X (page 223)
- 20021017 : editted
  - 8.3. Code templates (page 166) to 8.9. Comment (page 181)
- 20021016 : editted
  - 8.1. Code completion (suggestion) (page 146)
- 20021015 : added
  - 8.1. Code completion (suggestion) (page 146)
  - 8.2. Code-completion OLD XXX (page 161)
- 20021014 :
  - 7. Editor X (page 91) text added.
- 20021012 :
  - 7. Editor X (page 91) text added.
- 20021010 :
  - added 27. Keymaps X (page 341)
  - 7. Editor X (page 91) reorganized.
- 20021009 :
  - 7. Editor X (page 91) reorganized.
- 20021008 :
  - 6. Files (page 63) rewritten.
- 20021007 :
  - 3. Basic Application (page 31) rewritten.

- 4. Projects (page 51) rewritten.
- 5. Directories (packages) (page 59) 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 243)
- 20021002: added text to
  - 15. Debugger X (page 269)
  - 14. Ant X (page 263)
- 20021001: 12. Java Doc X (page 247) 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 145): new text.
- 20020927: reorganized many chapters.
- 20020925: udpated:
  - 7. Editor X (page 91)
  - 8. Code Automation (page 145)
  - 9. Code Refactoring X (page 183)
- 20020924: 7. Editor X (page 91) updated.
- 20020923: a lot added to the chapters of Part B. Projects / dirs / files (page 49) (projects, files).
- 20020922: a lot added to the chapters of Part B. Projects / dirs / files (page 49) (projects, files).
- 20020919: idea examples added to Chapter 22. Plugins X (page 311).
- 20020919: Reorganized the doc. Chapters removed/added.
- 20020919 Chapter 3. Basic Application (page 31) updated.
- 20020916: Reorganized the doc. Many chapters removed.
- 20020912: Chapter 24. Tags (page 91) added.
- 20020912: Chapter 21. EJB X (page 295) added (uses Sun ejb example).
- 20020909: Chapter 9. Code Refactoring X (page 183) partially completed.
- 20020908: Chapter 11. Version control X (page 235): history dialog described (local vcs).

INDEX TEST

This is a marker for the index.

This is a bigmarker for the index.

This is a secondlevel bigmarker.

hypertext marker

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 XXX~~

About this doc

20020930TT added this chapter. need to expand.

This document is designed to help you get started with the IDEA Java development from IntelliJ as quickly as possible. Although it is a “quick” start, it is not a short document. IDEA provides integrates into a compact package an impressive array of functions, and this document introduces as much of this functionality as possible (as quickly as possible).

This document contains the following major parts:

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

A **List of figures (page 347)** and **Index XXX (page 353)** are also included at the end of this doc.

TOC

Part A. Basics

20021018TTT last edit.

The chapters in this part introduce the basics required to start using IDEA.

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. Basic Application (page 31). Demonstrates how to create the required project / package / source files and then compile / run / debug a simple Java application with IDEA.

1. Documentation / Support

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

20020906TTT: this chapter should list all availabe 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 Quick Start 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.intellij.documentation)

jetbrains.intellij.documentation is the newsgroup for IntelliJ documentation.

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

1.2. Documentation sources

IntelliJ is constantly developing the 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 provides the following:

- **Features list**.
- **Downloads** (www.intellij.com/idea/download.jsp). You can download of 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. It provides various types of documentation.

www.intellij.org

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

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

1.3. Documents / help

- **IDEA Quick Start (this doc)**
- **IDEA User Guide**
- **IDEA Help**
- **IDEA Tool Tips**

IDEA Quick Start (this doc)

This 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 UserGuide 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 cant find the answers, then IntelliJ provides support via

- **Phone**
- **Email**

FAQ

For common questions refer to the available FAQs:

- **Chapter 25. FAQ XXX (page 337)**.
- **http://www.intellij.com/support/faq/** (may contain more recent info than this doc).
- **Online 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).

TS

For solutions to specific problems refer to

- **Chapter 26. Trouble shooting XXX (page 339)**.

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

20021004TTT: 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)

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

2.1.1. Download

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.

2.1.2. Install

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.

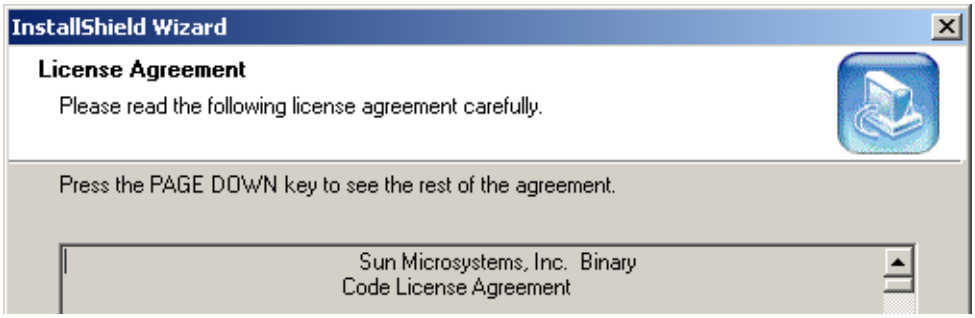


Figure 2.2. J2SDK installation license agreement (320)

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

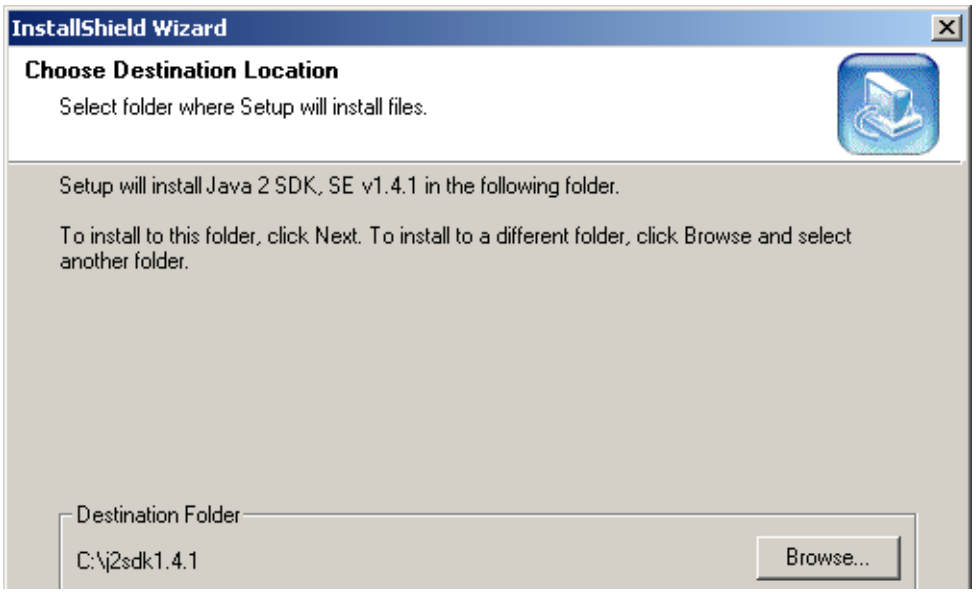


Figure 2.3. J2SDK installation destination (319)

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

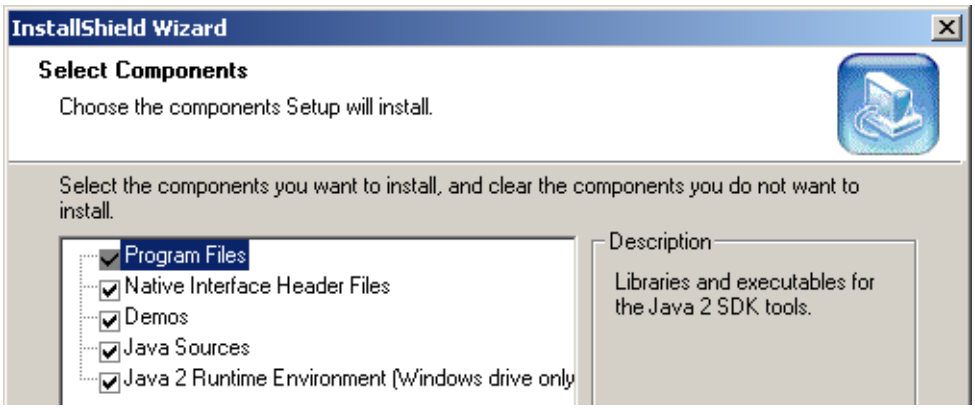


Figure 2.4. J2SDK installation components (317)

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

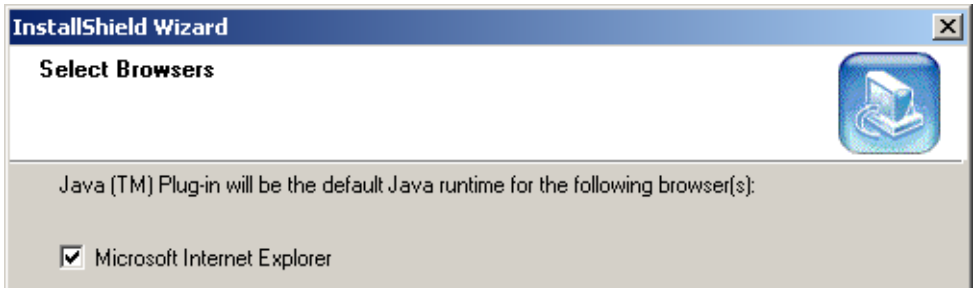


Figure 2.5. J2SDK installation browser select (318)

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

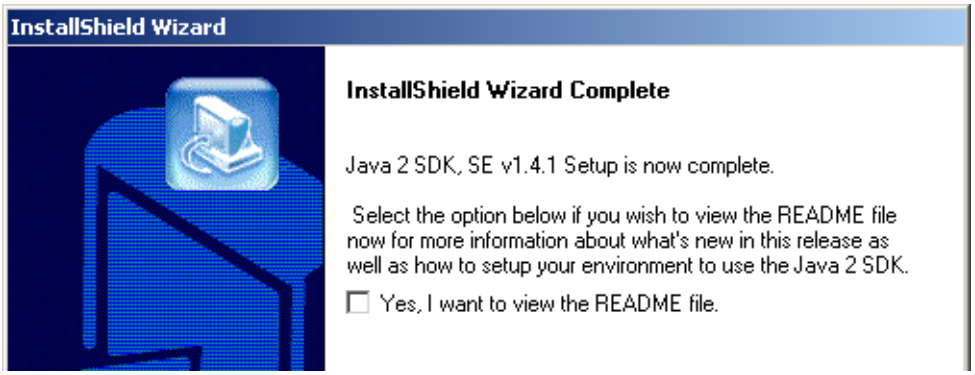


Figure 2.6. J2SDK installation complete (316)

2.8. Click **Finish**.

20021004TTT the following did not occur.

Figure 2.7. xxxx (315)

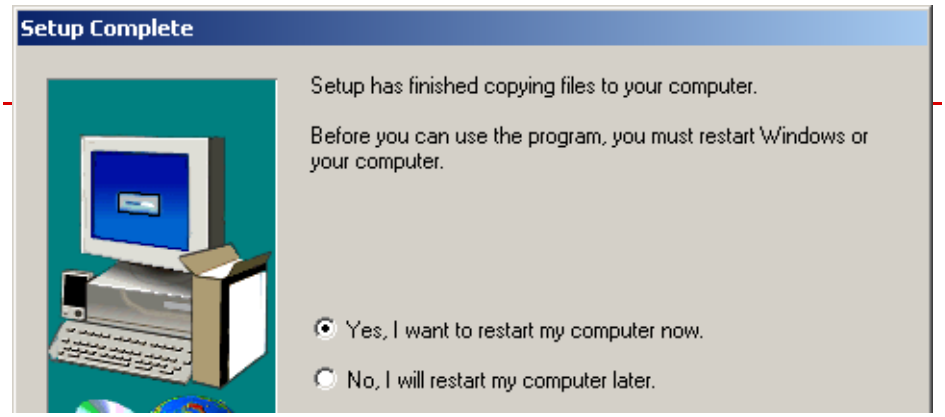


Figure 2.8. xxxx (314)

2.2. IDEA

This section describes the complete IDEA installation including the following:

2.2.1. Download Windows EXE/ZIP installer

The IDEA installer for Windows is available as an

- EXE
- ZIP

You need to download one or the other for installation.

2.9. Download from **www.intellij.com/idea/download.jsp** the Windows EXE (idea651.exe) or ZIP (idea651.zip) installer:

Figure 2.9. Windows EXE/ZIP installer download page (322)

2.2.2. Automatic installation (EXE)

This section describes installation for both the exe and zip file.

2.10. Double-click on **idea651.exe**. The dialog “Introduction” appears.

Figure 2.10. IDEA EXE installation introduction (336)

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

Figure 2.11. IDEA EXE installation license agreement (335)

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

2.13. Click **Next**. The dialog “Choose JDK” appears.

Figure 2.12. IDEA EXE installation choose JDK (334)

2.14. Click **Next**. The dialog “Where to install” appears.

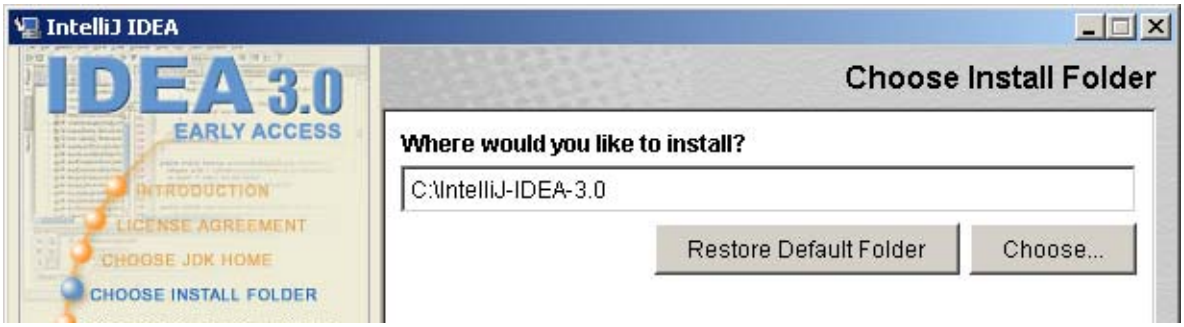


Figure 2.13. IDEA EXE installation choose folder (333)

2.15. Click **Next**. The dialog “Where to create project icons” appears.

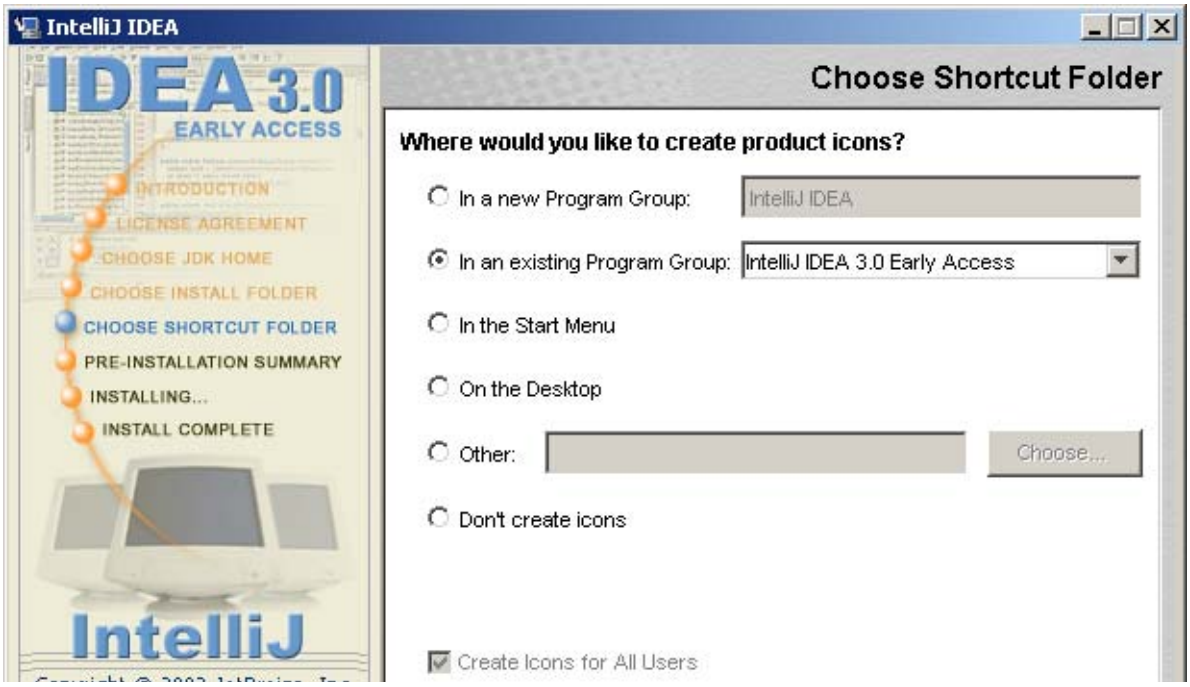


Figure 2.14. IDEA EXE installation icon location (332)

2.16. Click **Next**. The dialog “File associations” appears.

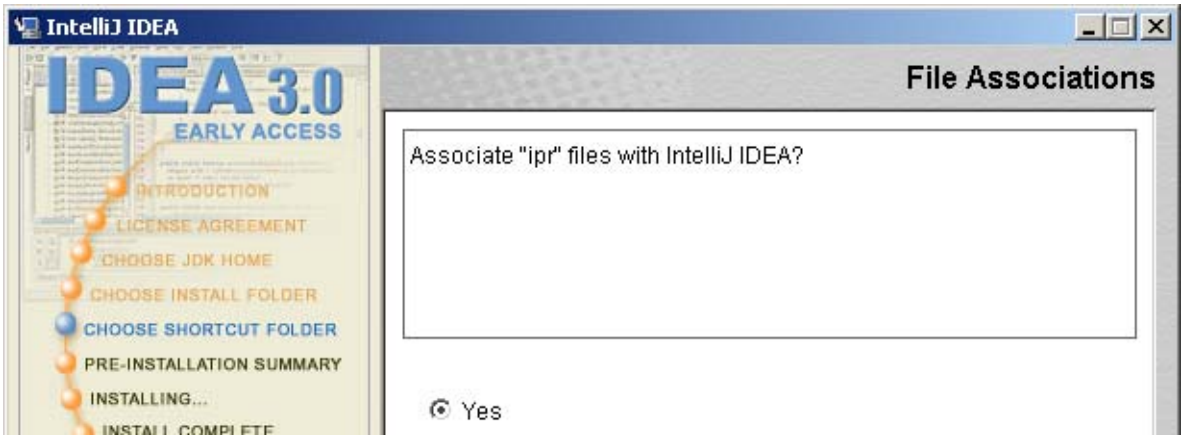


Figure 2.15. IDEA EXE installation file associations (331)

2.17. Click **Next**. The dialog “Pre-installation summary” appears.

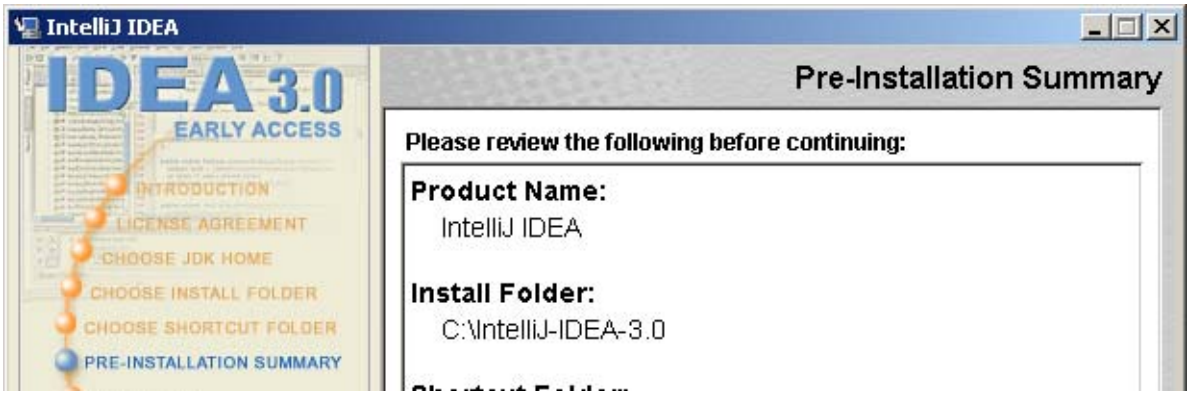


Figure 2.16. IDEA EXE installation pre-install summary (330)

2.18. Click **Install**. IDEA is installed. The dialog for the readme.txt file appears.

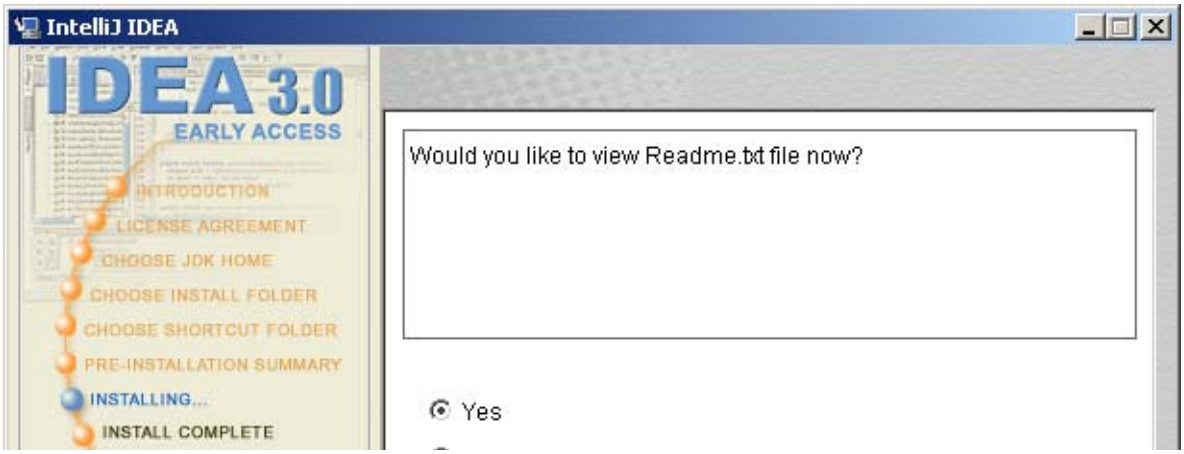


Figure 2.17. IDEA EXE installation readme (329)

2.19. Click **Next**. The readme.txt appears.

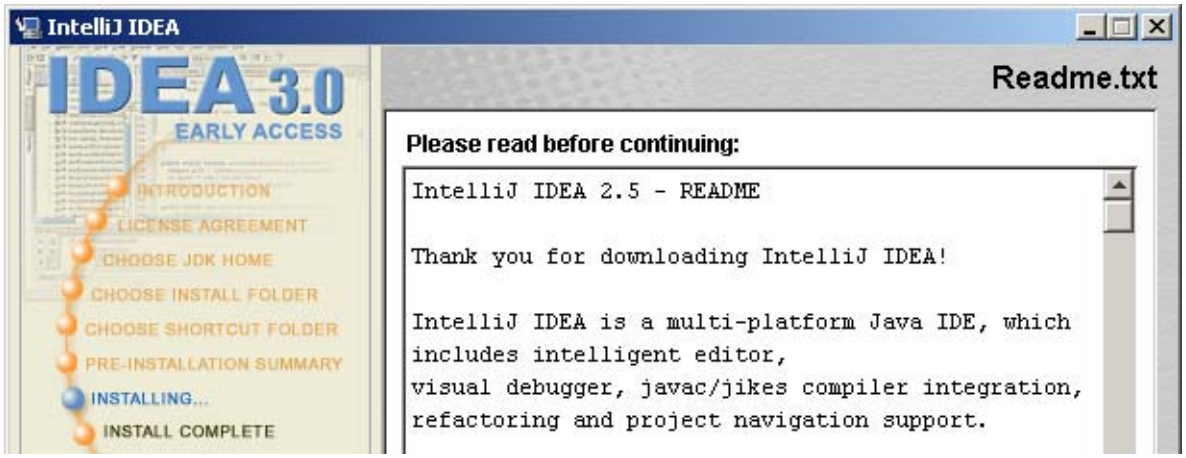


Figure 2.18. IDEA EXE installation readme.txt (328)

2.20. Click **Next**. The dialog “Install complete” appears.

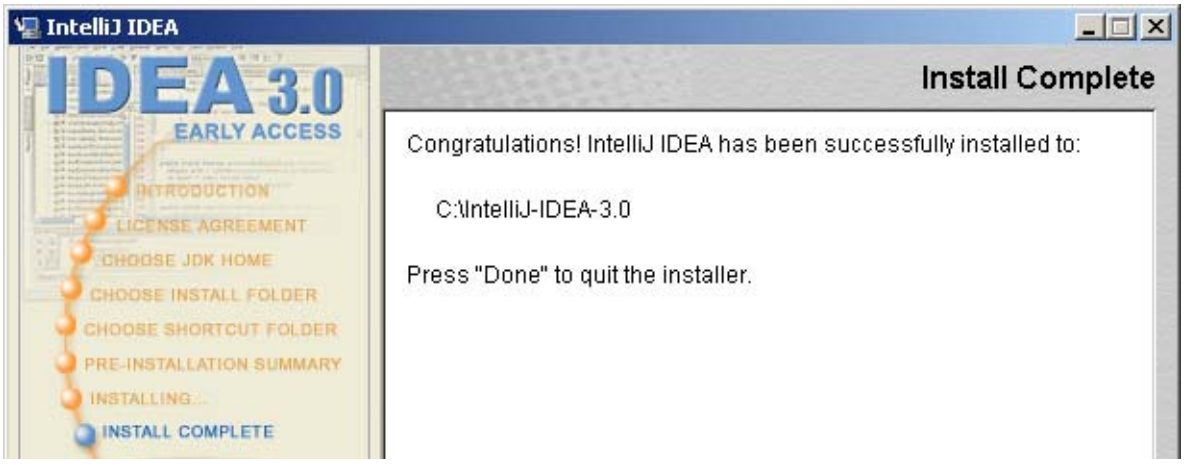


Figure 2.19. IDEA EXE install complete (327)

2.21. Click **Done**. The wizard closes. The installation is complete.

2.2.3. Manual installation (ZIP)

2.22. Unzip **idea651.zip** to **C:\ideaZIP**.

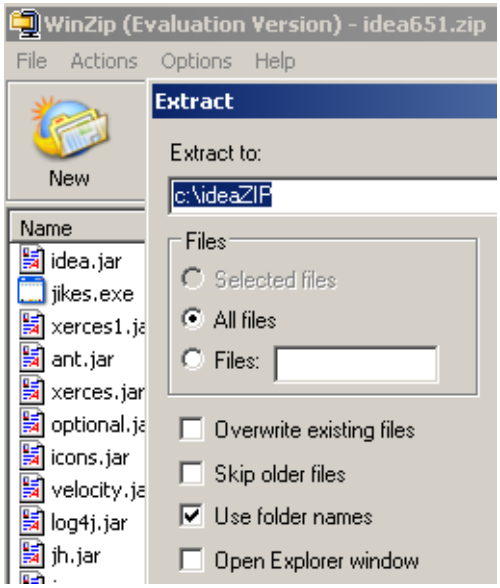


Figure 2.20. WinZIP settings for IDEA ZIP install (255)

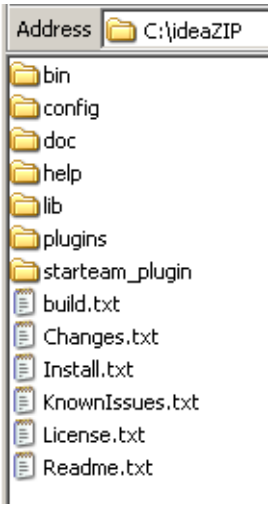


Figure 2.21. Unzipped IDEA ZIP install files (323)

2.23. Specify the location of the J2SDK in the file **C:\ideaZIP\bin\idea.bat**:

```
IF NOT EXIST "%IDEA_JAVA_HOME%" SET IDEA_JAVA_HOME=c:\j2sdk1.4.1
```

2.2.4. Get evaluation license

2.24. Fill out and submit the license form at **http://www.intellij.com/idea/evaluate.jsp**.

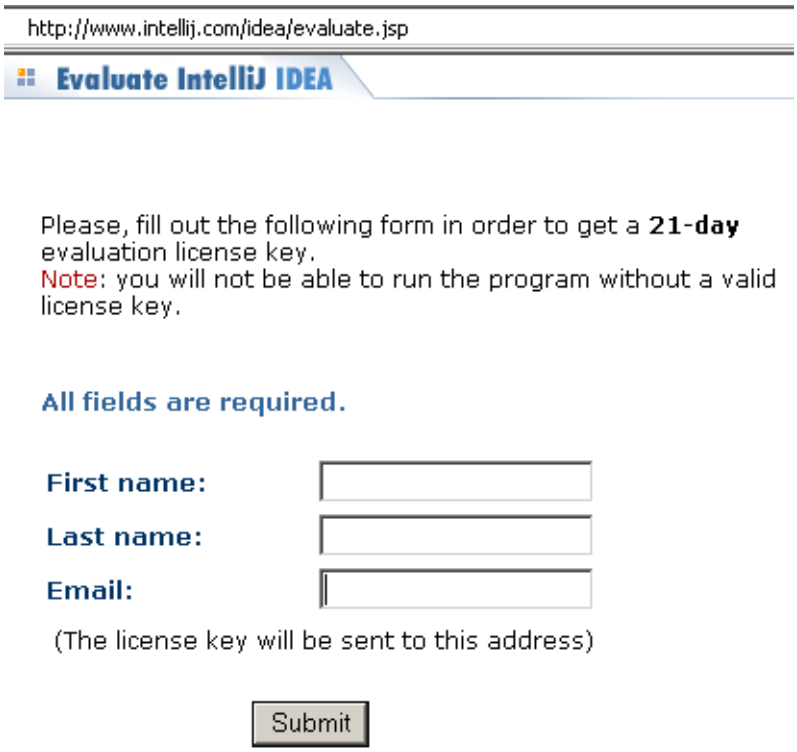


Figure 2.22. License form (254)

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

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

```
Dear john doe,

Thank you for trying IDEA from IntelliJ Software!

You will need the following information to run the software.
Your user name: john doe
Your evaluation key: 012345abcdef6789ghijk
The key will expire on October 25, 2002
Email address registered: john@doe.com

Please, follow these installation instructions:
 - Unpack the downloaded file,
 - Run the idea.bat file under the bin subdirectory,
 - Enter the license information.

If you require assistance, please email mailto:support@intellij.com

 -- IntelliJ Software Team
```

Figure 2.23. License received by email (253)

2.2.5. Start IDEA (enter license)

You are now ready to start IDEA.

2.2.5.1. EXE

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

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

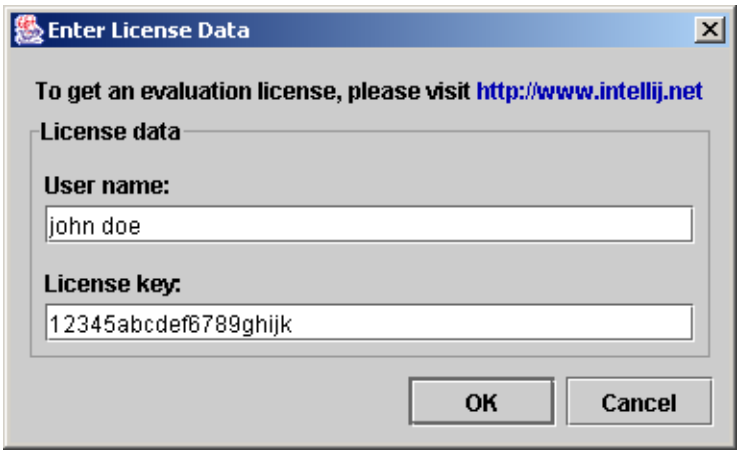


Figure 2.24. License data dialog (252)

2.27. Click **OK**. The dialog “License agreement for IntelliJ IDEA” appears.

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

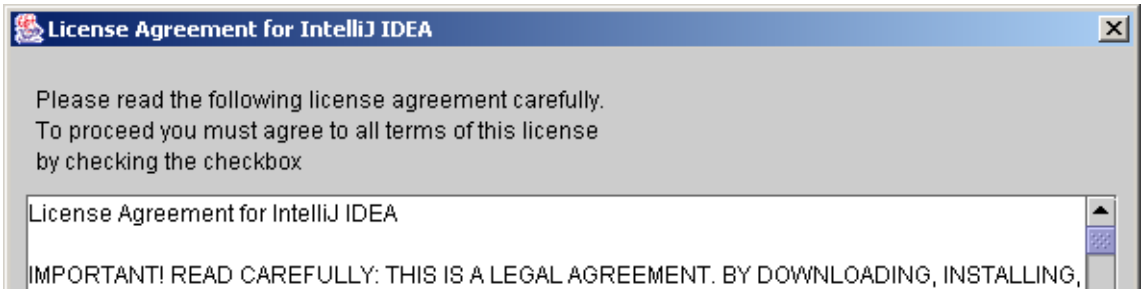


Figure 2.25. License agreement dialog (251)

2.29. Click **OK**. The new project wizard appears.

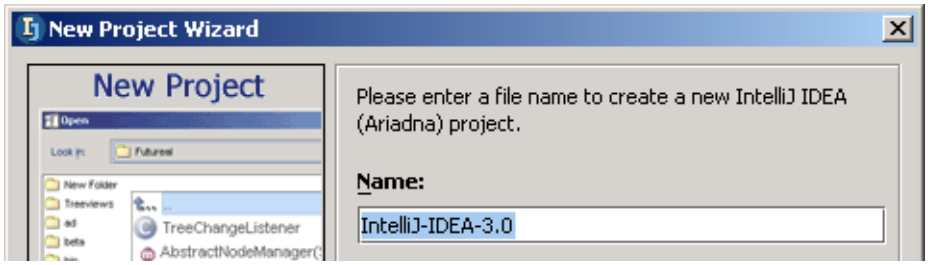


Figure 2.26. New project wizard (250)

2.2.5.2. ZIP

2.30. Double-click on **C:\ideaZIP\bin\idea.bat**. The dialog “Enter license data” appears.
2.31. Enter your **User name** and **License key**.

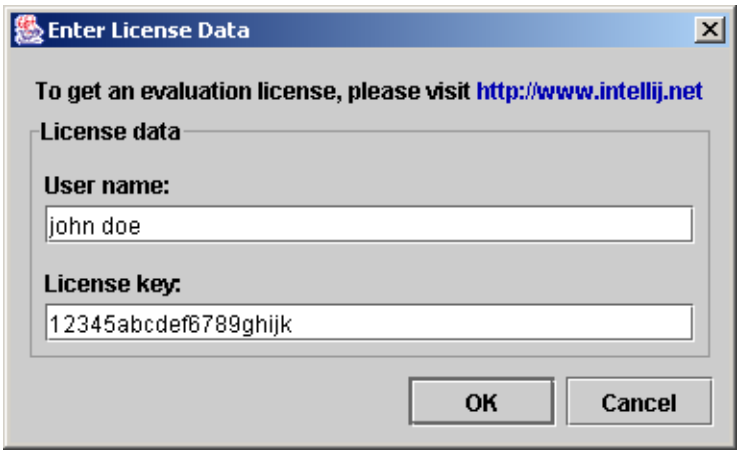


Figure 2.27. License data dialog (252)

2.32. Click **OK**. The dialog “License agreement for IntelliJ IDEA” appears.
2.33. Check **Accept all terms of the license**.

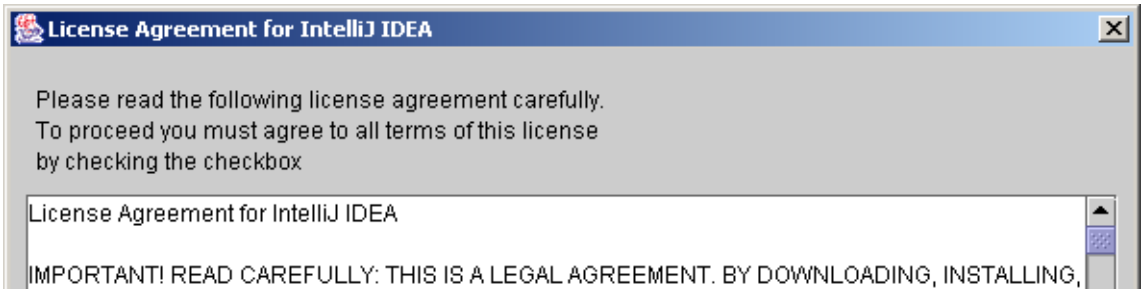


Figure 2.28. License agreement dialog (251)

2.34. Click **OK**. The new project wizard appears.

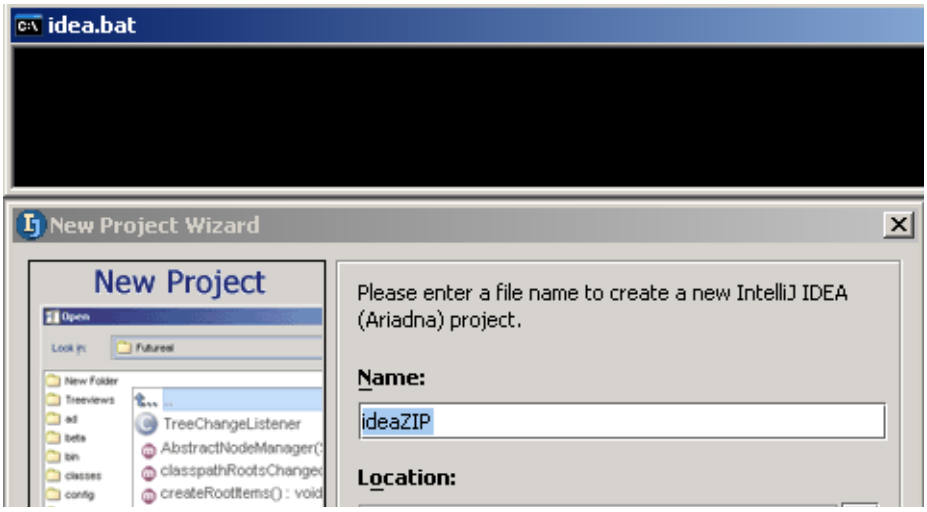


Figure 2.29. Command window and New Project wizard (597)

~~2.2.6. 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. Basic Application

20021007TTT: updated.

for help: zheka, anton

This chapter describes the steps required to create, run and debug the simplest Java application use IDEA. It also introduces some very important basic concepts.

In this chapter you will

- **3.1. Create project (page 32)**
- **3.2. Create package (page 38)**
- **3.3. Create file (page 39)**

  Projects, packages (dirs), and files are described in more detail in Part B. Projects / dirs / files (page 49) which included the following chapters

  - *4. Projects (page 51)*
  - *5. Directories (packages) (page 59)*
  - *6. Files (page 63)*
- **3.4. Edit file (page 40)**

  Editting files is described in more detail in Part C. Editting files (page 89) which includes the following chapters

  - *7. Editor X (page 91)*
  - *8. Code Automation (page 145)*
  - *9. Code Refactoring X (page 183)*
- **3.5. Compile (page 43)**
- **3.6. Run (page 44)**
- **3.7. Debug (page 46)**

  Compilation and running/debugging are described in more detail in Part D. Compile / Debug (page 257) which includes the the following chapters:

  - *13. Compiler XXX (page 259)*
  - *15. Debugger X (page 269)*

3.1. Create project

To create a project you will

- **3.1.1. Specify project name / location (page 32)**
- ~~**3.1.2. Specify JDK (page 32)**~~
- **3.1.3. Specify project paths (page 34)**
- ~~**3.1.4. Specify output type / path(s) (page 37)**~~

Projects will be discussed in more detail in Chapter 4. Projects (page 51).

3.1.1. Specify project name / location

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

3.2. For the project “Location”: Click on the directory button (...). The dialog “Select path” appears.
3.3. Right-click on **C:**. A popup dialog appears.

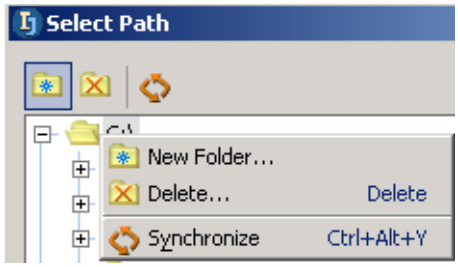


Figure 3.1. Popup “New Folder” (248)

3.4. Click on **New Folder**. The dialog “New Folder” appears.
3.5. For “Enter a new folder name”: Enter **MyProjectFolder**.

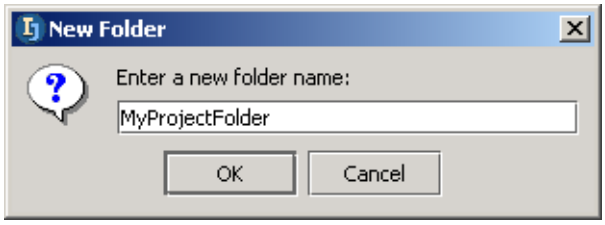


Figure 3.2. New folder name (052)

3.6. Click **OK**.
3.7. Click **OK**. Note the new project location.
20020906TTT: ?? why is it called “location”?

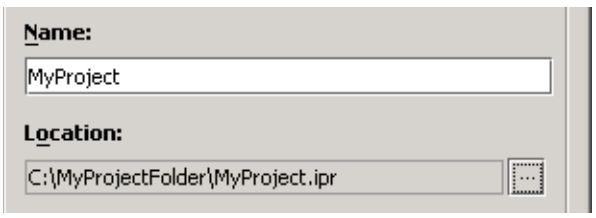


Figure 3.3. Project name / location (053)

~~3.1.2. Specify JDK~~

~~3.8. From the main menu: Select **File | Project properties...**. The project properties dialog appears.~~
~~3.9. Select tab **Paths**.~~

3.10. For “Target JDK”: Select **java version “1.4.0_01”**.

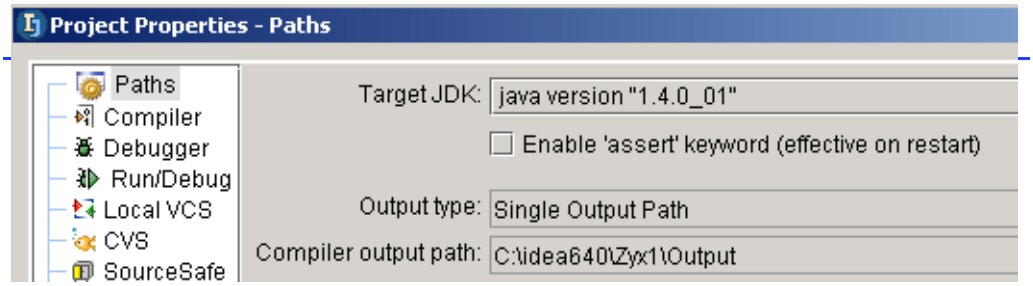


Figure 3.4. Target JDK selected (469)

3.1.3. Specify project paths

You will now specify the following project paths:

3.1.3.1. Output path

3.11. Click on the **Compiler output path** directory button  . The dialog “Select Path” appears.
3.12. Right-click on **MyProjectFolder**. A context dialog appears.
3.13. Click **New Folder**. The dialog “New Folder” appears.
3.14. For the new folder name enter “**MyOutputFolder**”.

Figure 3.5. Output folder name (581)

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

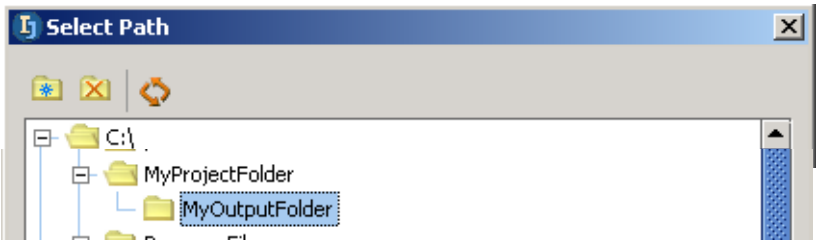


Figure 3.6. Select folder Output (246,247)

3.16. Click **OK**. The output folder is now selected.

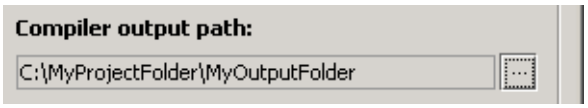


Figure 3.7. Compiler output folder (579)

3.1.3.2. Project path (directories)

20020906TTT: ?? dialog should have a title.

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

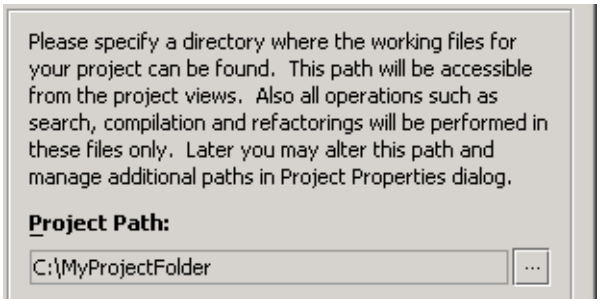


Figure 3.8. Project path (054)

3.1.3.3. Source path

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

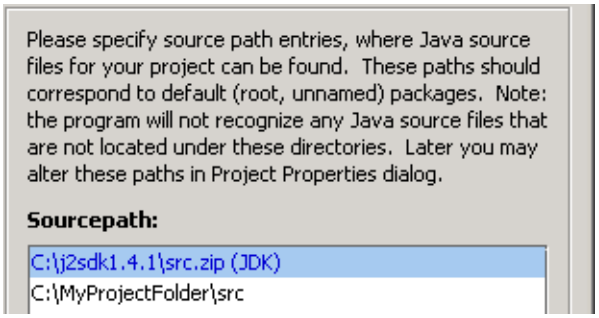


Figure 3.9. Source paths (055)

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

Figure 3.10. Create folder \src confirmation (056)

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

3.1.3.4. Class path

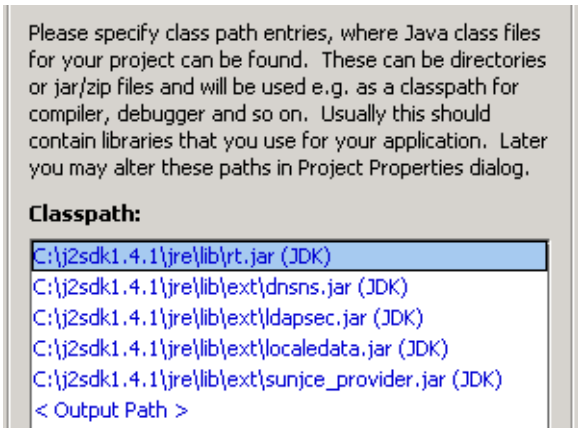


Figure 3.11. Class paths (057)

3.21. Click **Finish**. The project files are created (the project is built). Finally the main dialog appears.

20020906TTT: ?? i would prefer to see the project window opened automatically

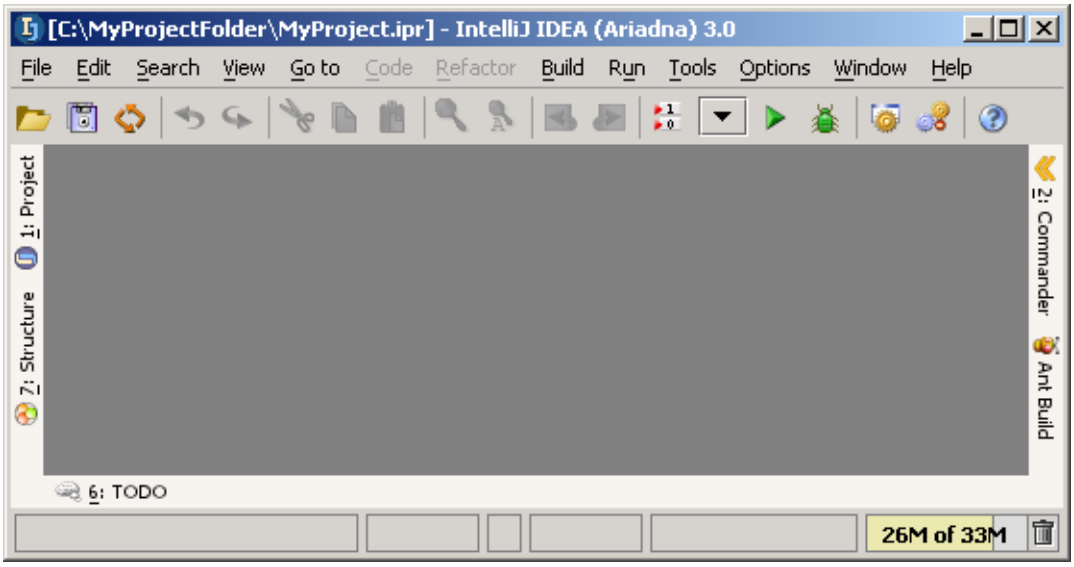


Figure 3.12. IDEA main dialog (591)

3.1.4. Specify output type / path(s)

3.1.4.1. Output type (single)

3.22. Open the project settings dialog.
3.23. For “Output type”: Click on the directory button. The dialog “Output type” appears.
3.24. Select **Single output path**.

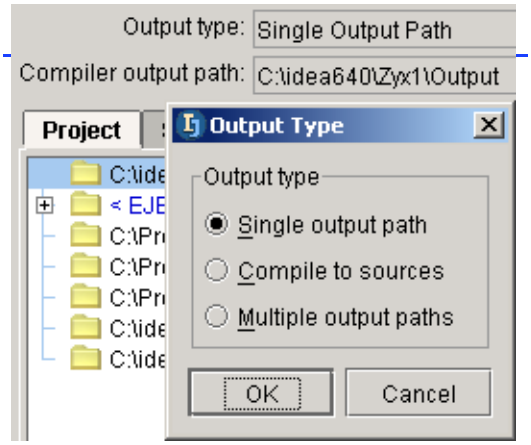


Figure 3.13. Single output path selected (471)

3.1.4.2. Output path

3.25. For “Compiler output path”: Click on the directory button. The dialog “” appears.
3.26. Right-click. Dialog “New folder” appears.
3.27. For “Enter a new folder name” enter **MyOutput**.

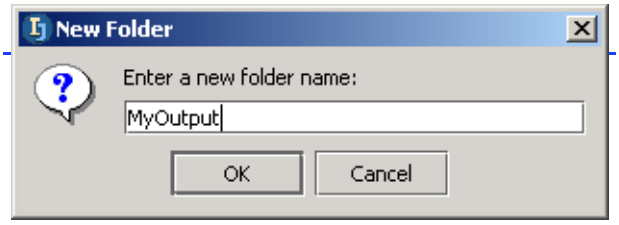


Figure 3.14. Compiler output path name (071)
3.28. Click **OK**.
3.29. For “Compiler output path”: Select **c:\MyProjectFolder\MyOutput**.

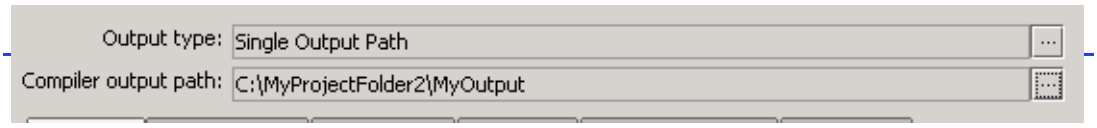


Figure 3.15. Compiler output path selected (073)

3.2. Create package

*Packages (directories) will be discussed in more detail in **Chapter 5. Directories (packages) (page 59)**.*

3.30. Click on the Project tool tab. The Project tool appears.

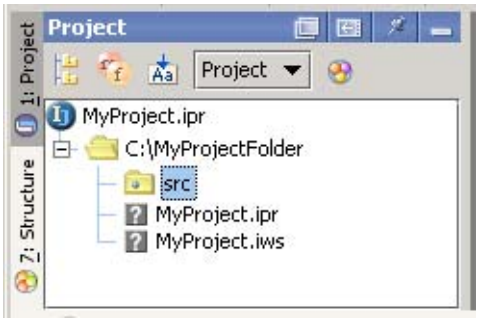


Figure 3.16. Project tool (245)

20021007TTT: why is the output dir not visible??

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

3.32. Select **New / Package**. A dialog requesting the package name appears.

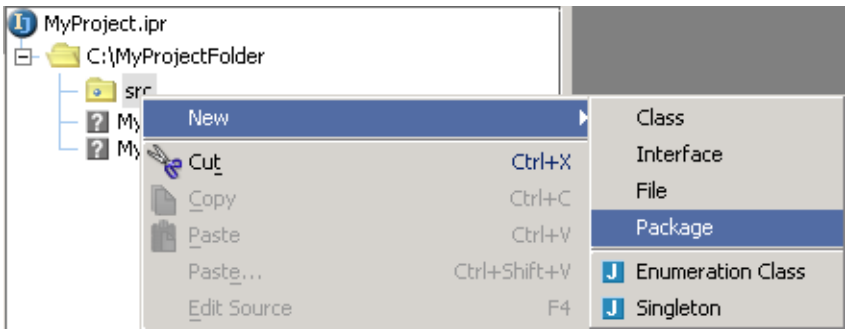


Figure 3.17. New package (574)

3.33. For the package name enter **MyPackage**.

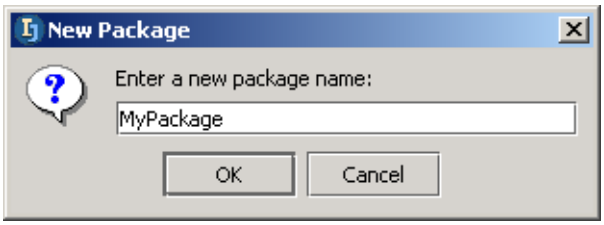


Figure 3.18. New package dialog (244)

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

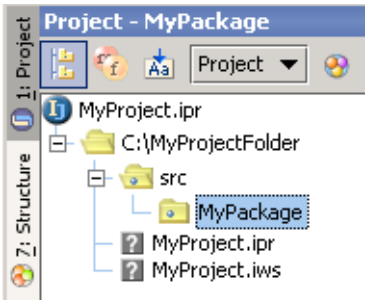


Figure 3.19. Created package (573)

3.3. Create file

You are now ready to create a Java source file.

*Files will be discussed in more detail in **Chapter 6. Files (page 63)**.*

3.35. Right-click on **MyPackage**.

3.36. From the context dialog select **New / Class**. A dialog requesting the class name appears.

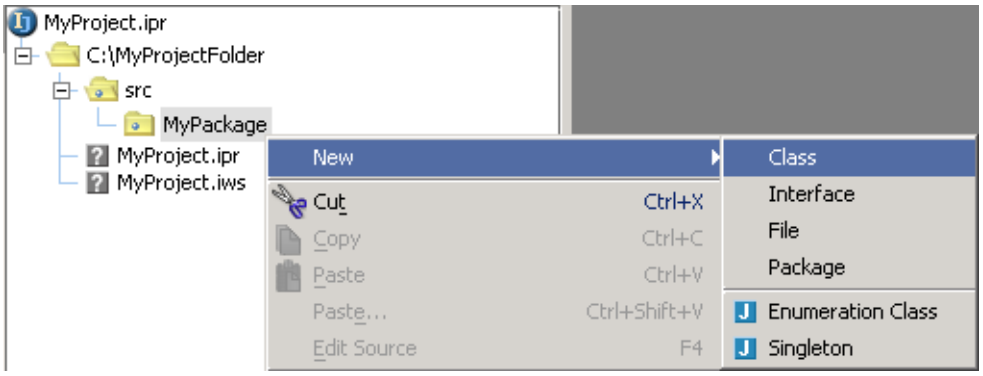


Figure 3.20. New class (243)

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

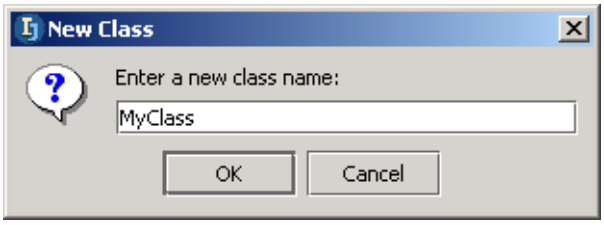


Figure 3.21. New class (242)

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

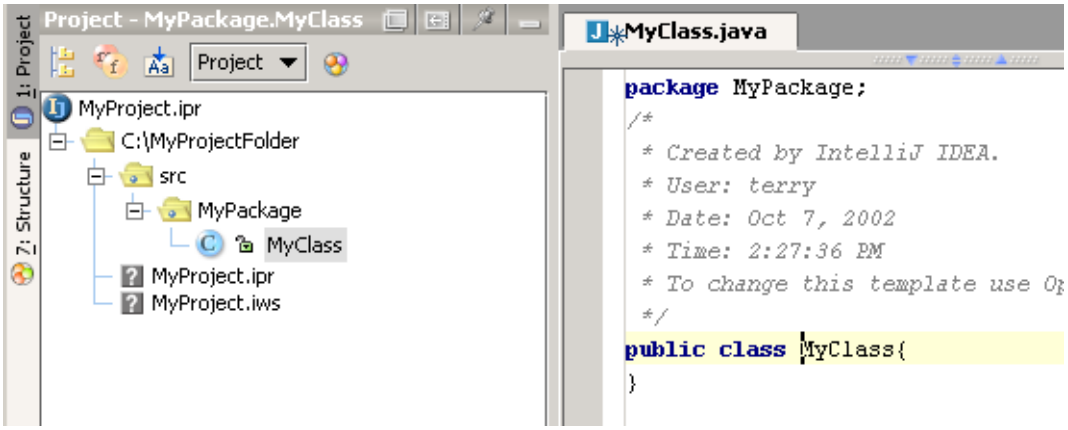


Figure 3.22. Class MyClass (572)

3.4. Edit file

You are now ready to edit the Java source file. You will

- **3.4.1. Add code (page 40)**
- **3.4.2. View class structure (page 42)**

Editting files will be described in more detail in

- *7. Editor X (page 91)*
- *8. Code Automation (page 145)*
- *9. Code Refactoring X (page 183)*

3.4.1. Add code

3.39. Enter the text "**psvm**" as shown in the diagram below:

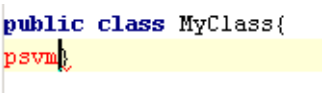


Figure 3.23. Enter text "psvm" (571)

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

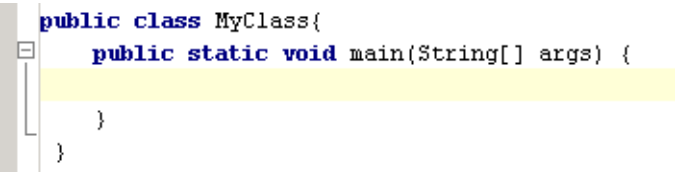


Figure 3.24. "psvm" replaced with live template text (570)

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 166)**.

3.41. Enter the text "**itar**" as shown in the diagram below:

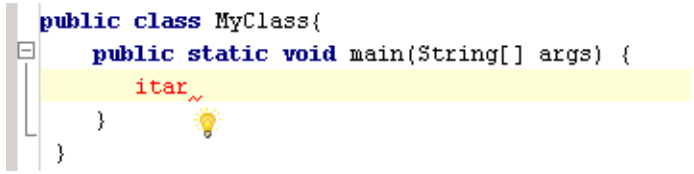


Figure 3.25. Enter text "itar" (569)

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

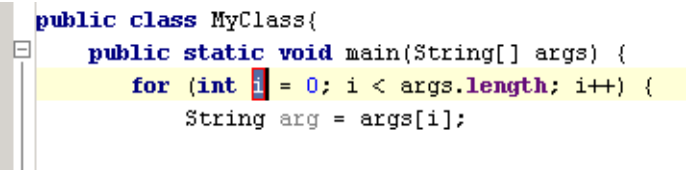


Figure 3.26. itar live template text (568)

3.43. Modify the text as shown below:

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

        }
    }
```

Figure 3.27. Modified itar text (566)

3.44. Enter "**sout**" live template text as shown below:

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

Figure 3.28. sout (565)

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

Figure 3.29. sout live template text (564)

3.45. Modify the sout live template text as shown below (the error will be corrected in the next section):

MyClass.java

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

Figure 3.30. Modified sout live template text (563)

3.46. Press **CTRL-S**. The file is saved.

Note the indicator for unsaved changes in a file.

File not saved File saved

Figure 3.31. Unsaved changes indicator (561,560)

3.4.2. View class structure

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

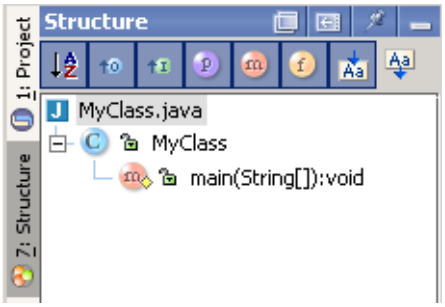


Figure 3.32. Class structure (559)

3.48. Click on tab **Project**.

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

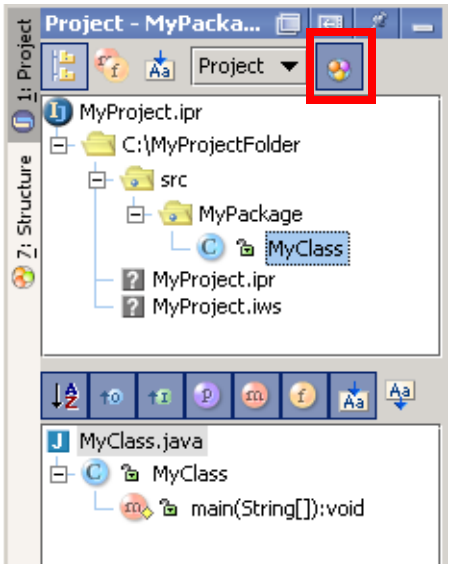


Figure 3.33. Both Project and Structure panes are displayed (558)

3.5. Compile

You are now ready to compile the Java source file.

Compilation is described in more detail in Chapter 13. Compiler XXX (page 259).

3.50. From the main menu select **Build / Compile "MyClass.java" Ctrl-Shift-F9**. A message pane appears with a description of the error.

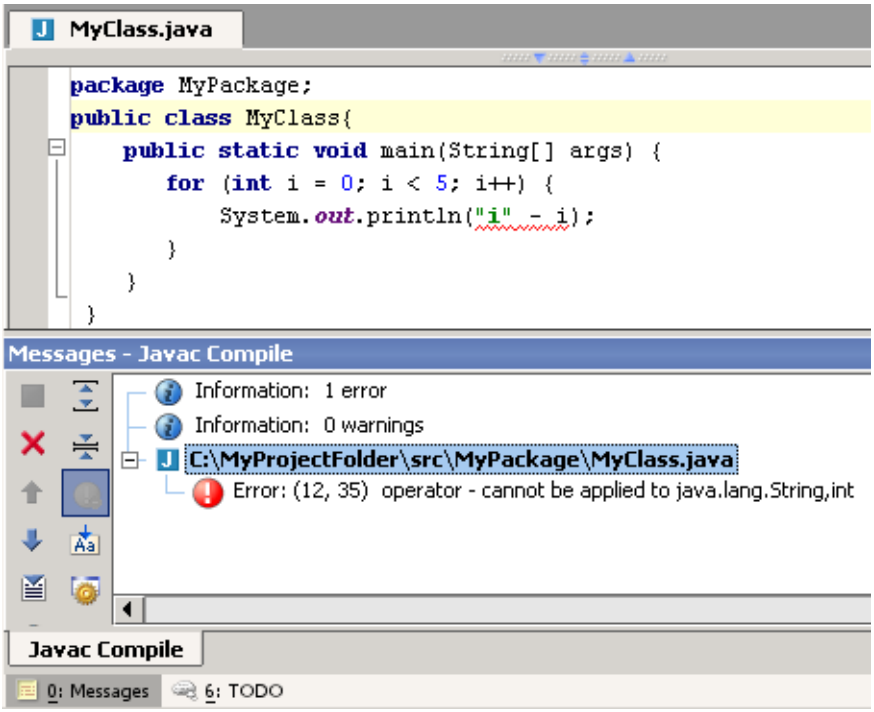


Figure 3.34. Message pane with error information (555)

3.51. Correc the error (change "-" to "+").

3.52. Recompile. The file **C:\MyProjectFolder\MyOutputFolder\MyPackage\MyClass.class** is created.

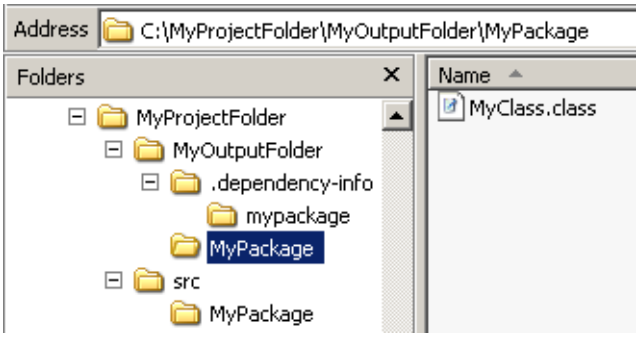


Figure 3.35. Compile file (241)

3.6. Run

You are now ready to run the application.
To run the application, you must

- **3.6.1. Create application configuration (page 44)**
- **3.6.2. Run the configuration (page 45)**

Running an application is described in more detail in Chapter 15. Debugger X (page 269).

3.6.1. Create application configuration

3.53. From the main menu select **Run / Run Shift-F10**. Dialog **Run** tab **Application** appears.

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

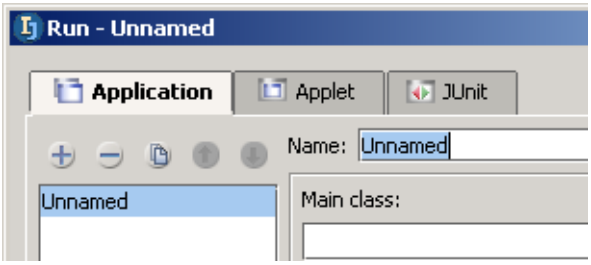


Figure 3.36. Application “Unnamed” (553)

3.55. Change Name to **MyApplication**.
3.56. Click on the directory button for **Main class:**. The dialog “Choose Main Class” appears.

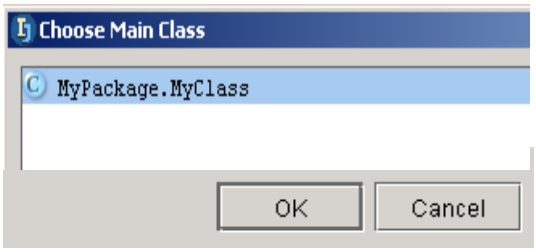


Figure 3.37. Choose main class (552,551)

3.57. Select **MyPackage.MyClass**.
3.58. Click **OK**.

20020906TTT: ?? what is the working directory used for?

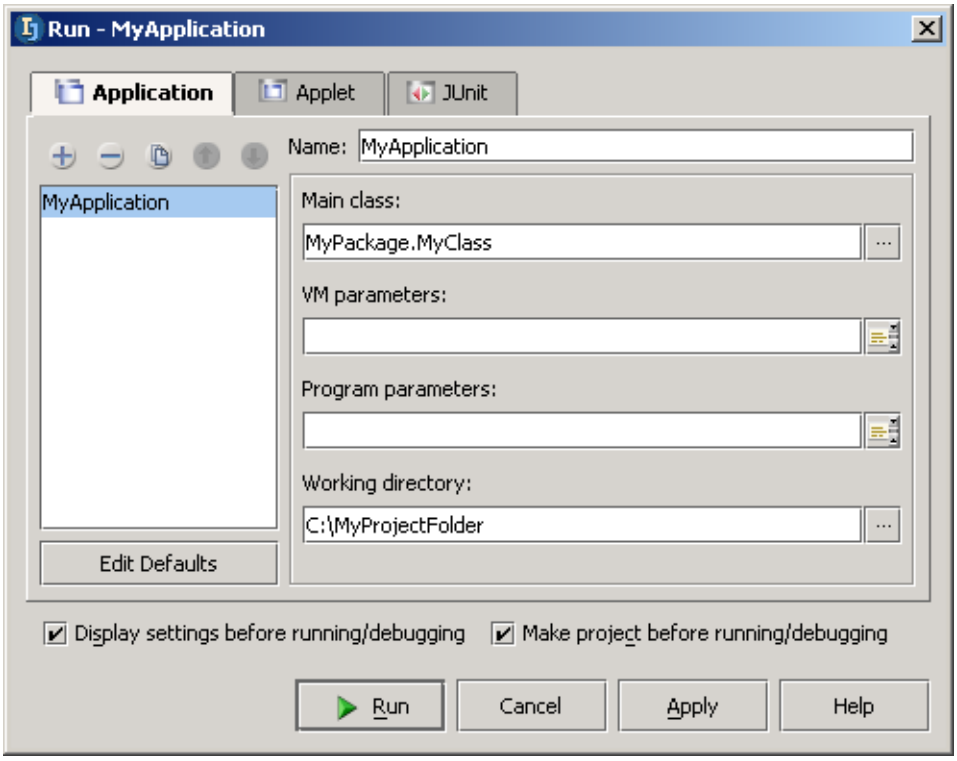


Figure 3.38. Application settings (550)

3.6.2. Run the configuration

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

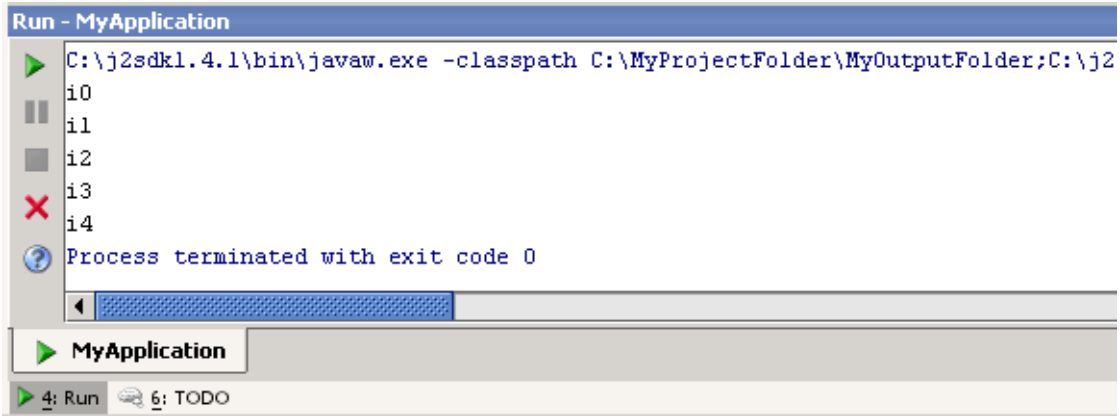


Figure 3.39. Run pane output (549)

3.7. Debug

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

Debugging is described in more detail in Chapter 15. Debugger X (page 269).

3.7.1. Set breakpoint

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

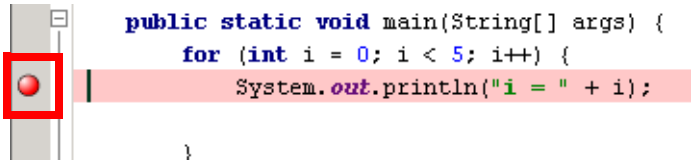


Figure 3.40. Set a line breakpoint (548)

3.61. Right-click on the marker.
3.62. From the context menu select **Properties**. The dialog **Breakpoints** appears.

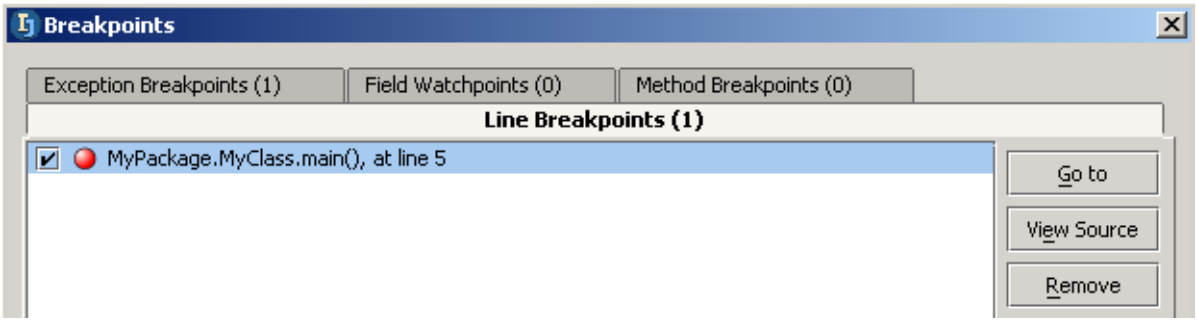


Figure 3.41. Dialog breakpoints (547)

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

3.63. Close the dialog.

3.7.2. Step through application

3.64. Click on the debug icon (or from the main menu select **Run / Debug Shift+F9**). The dialog "Debug - MyApplication" appears.
3.65. Click **Debug**. The application starts and halts at the breakpoint. The debug pane is displayed.

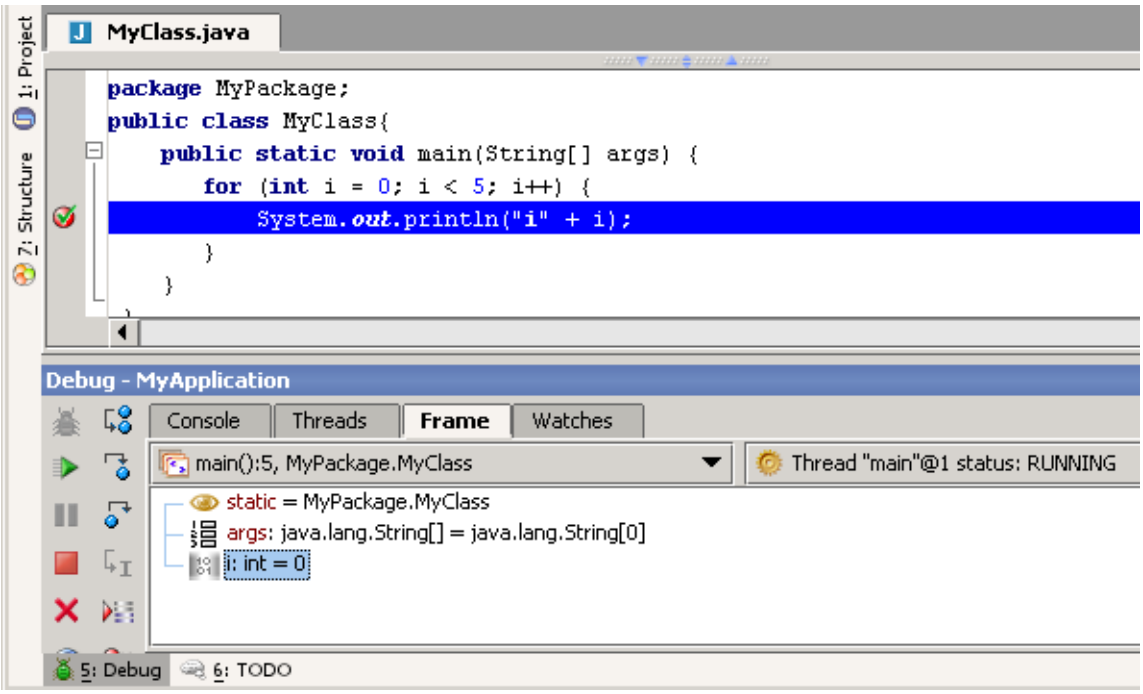


Figure 3.42. Debug pane (545)

3.66. 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.

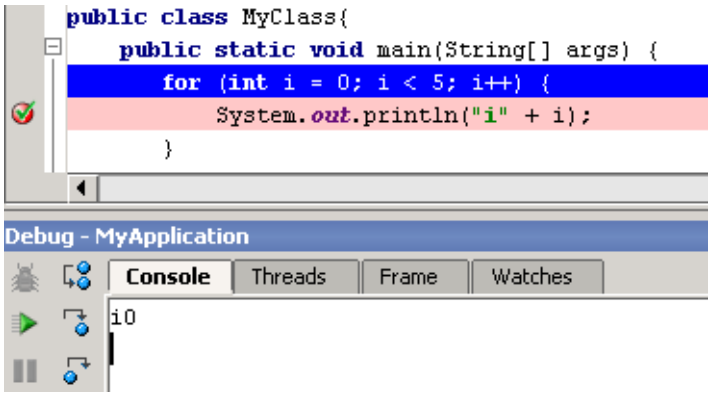


Figure 3.43. Step over (543)

3.67. Click on the icon **Resume** (or from the main menu select **Run / Resume program F9**). Program execution halts at the breakpoint.
3.68. Right-click on the breakpoint marker.
3.69. Select **Disable**. The breakpoint marker is changed.

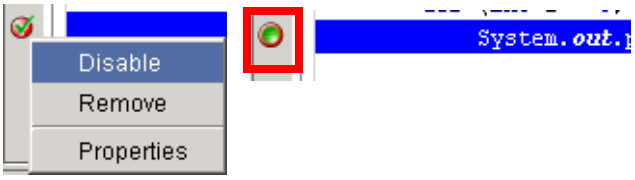


Figure 3.44. Disable breakpoint (541,540)

3.70. Click **Resume**. The application finishes.

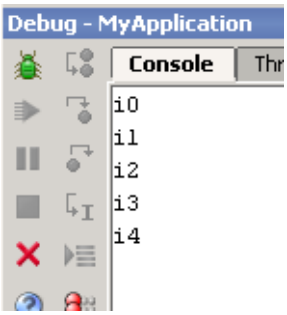


Figure 3.45. Application finishes (539)

Part B. Projects / dirs / files

20021018TTT last edit.

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

4. Projects (page 51). Demonstrates how IDEA project concepts make it as simple as possible to manage a software project.

5. Directories (packages) (page 59). Demonstrates how directories and packages can be created and managed within an IDEA project.

6. Files (page 63). Demonstrates how IDEA makes it as simple as possible to create and manage any type of file.

4. Projects

20021007TTT: editted.. reorganized.

?? what about project output dir (single, multiple), javadoc api, required libraries??

contacts: valentin.

This chapter introduces the basics for managing projects within IDEA:

- **4.1. New / Open / Close / Reopen (page 51)** describes how create, open, reopen and close projects.
- **4.2. Views (page 52)** describes the various views of the project that IDEA provides.
- **4.3. Modify project content (page 55)** shows how to add and delete items from a project.

4.1. New / Open / Close / Reopen

?? can you copy or delete projects??

With IDEA you can

- Create a **New** project.
- **Open** an existing project.
- **Close** an open project.
- **Reopen** a closed (but previously open) project.

4.1.1. New

You already created a new project in **3.1. Create project (page 32)**.

4.1.2. Open

You open a project by opening 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.

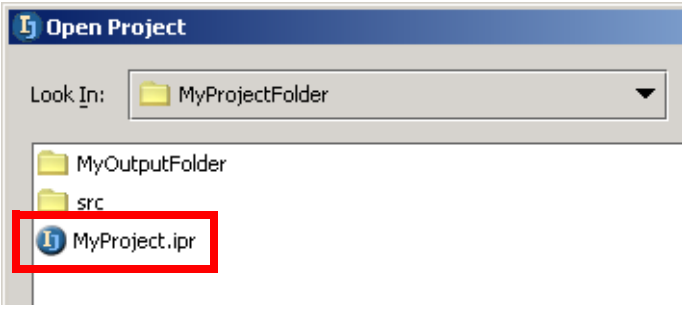


Figure 4.1. Open project .ipr file (617)

4.1.3. Close

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

4.1.4. Reopen

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

4.2. Views

this was introduced previously, but maybe more details here, or introduce here?

There are several tools for viewing project properties:

4.2.1. Project properties dialog

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

4.6. In the dialog “Project properties”: View the tabs **Project**, **Sourcepath**, and **Classpath**.

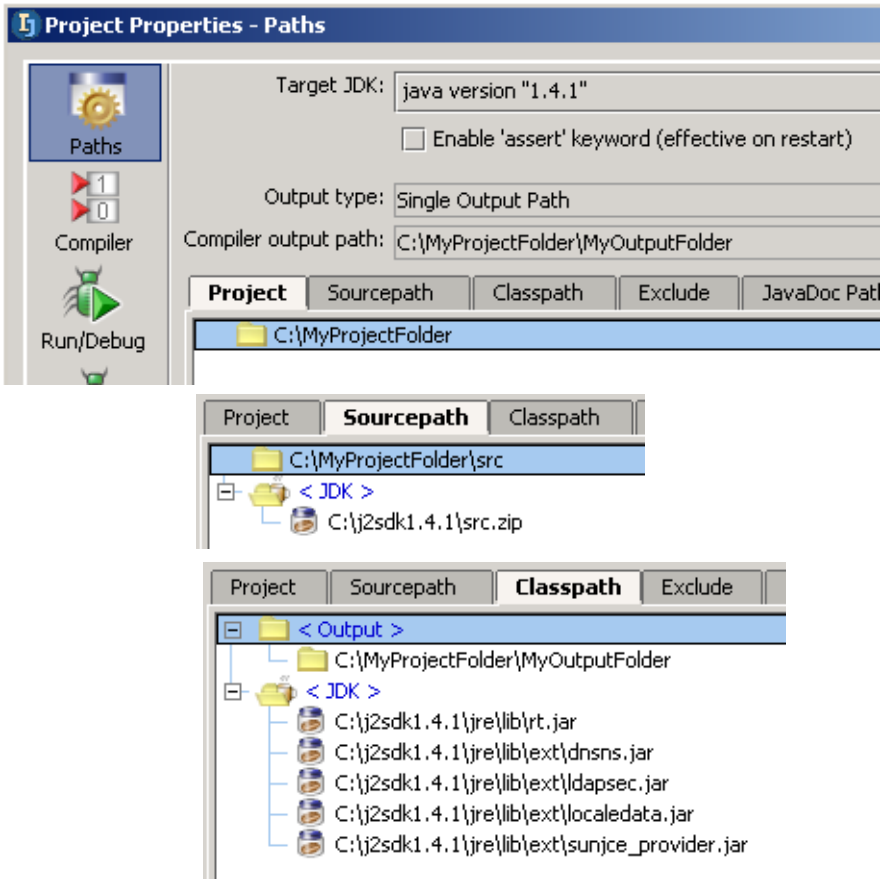


Figure 4.2. Project, Sourcepath and Classpath info (468,467,466)

4.2.2. Project tool window

4.7. In the project tool window: Click on the tab **Project**. The project tree appears.

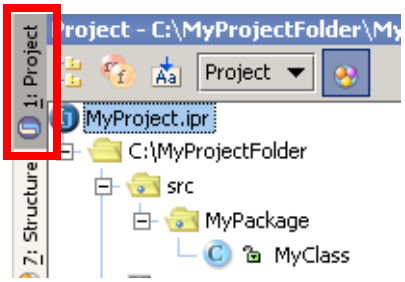


Figure 4.3. Project info in the project tool window (588)

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

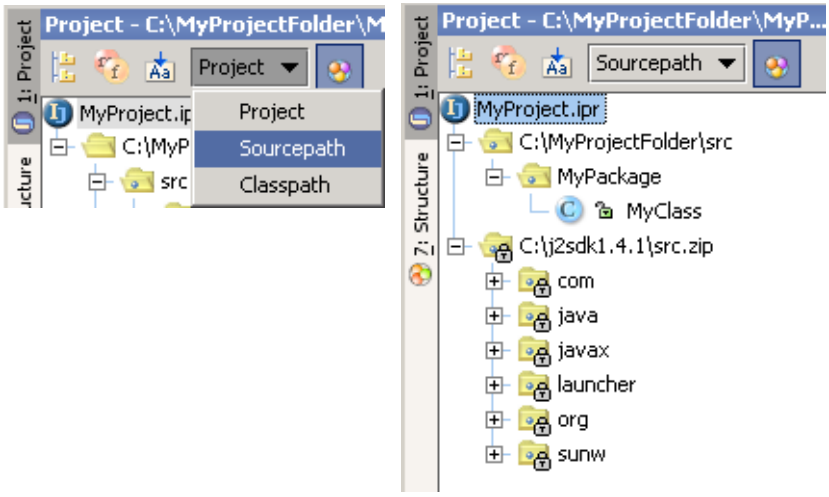


Figure 4.4. Sourcepath (575,587)

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

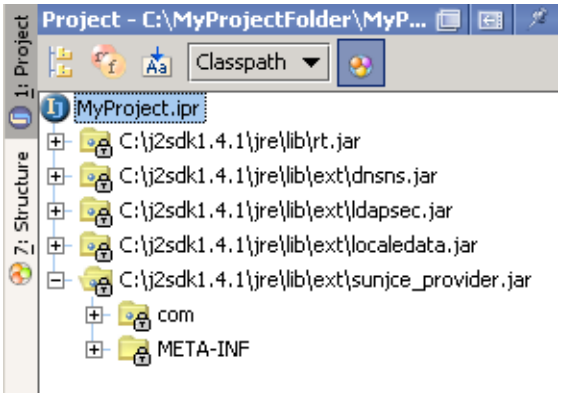


Figure 4.5. Classpath info in main dialog (586)

4.2.3. Commander

The commander shows the “Project view” info in the “commander” tool style.

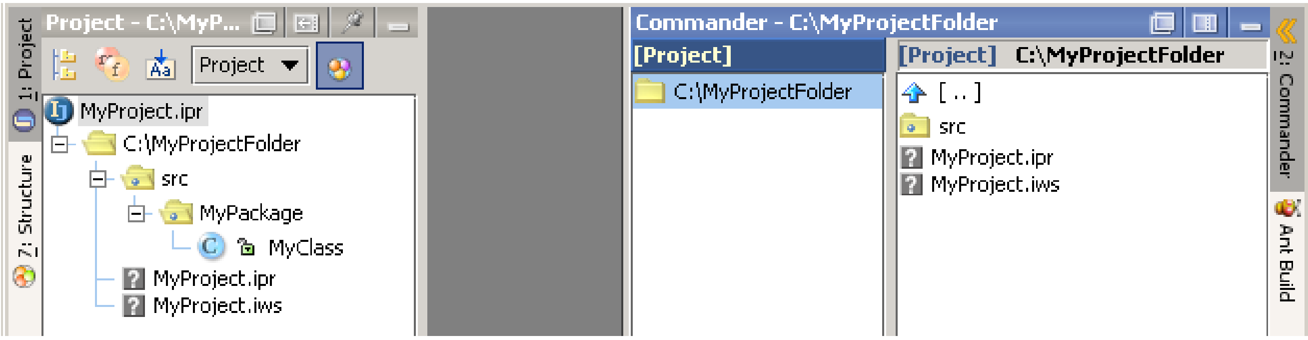


Figure 4.6. Commander (604)

4.2.4. Project xml files (.ipr/.iws)

4.10. Open the following files in a web browser:

- **C:\MyProjectFolder\MyProject.ipr**
- **C:\MyProjectFolder\MyProject.iws**

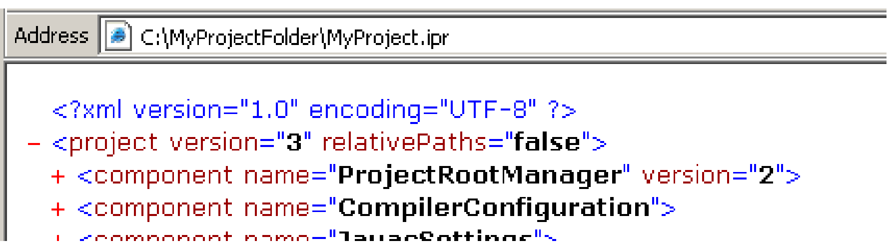


Figure 4.7. Project .ipr file contents (590)

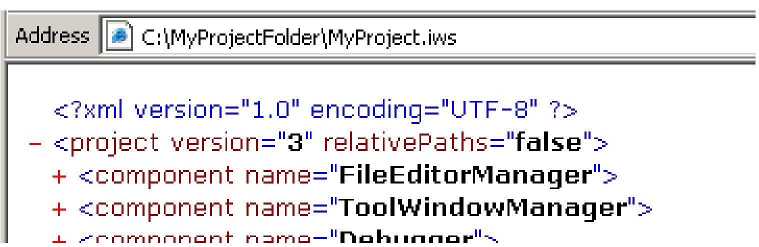


Figure 4.8. Project .iws file contents (589)

You will become acquainted with many of the components shown above while doing example during the rest of this Quick Start.

4.3. Modify project content

You can modify the following project content:

- **4.3.1. Directory (page 55)**
- **4.3.2. Sourcepath (page 56)** (directory, jar, zip) or
- **4.3.3. Classpath (page 57)** (directory, jar)

4.3.1. Directory

You can

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

a directory.

4.3.1.1. Add

4.11. Select **File | Project properties** section **Paths** tab **Project**.
4.12. Select **Add**. The dialog “Select Path” appears.
4.13. Double-click on a folder. The folder is added (to the end of the list).

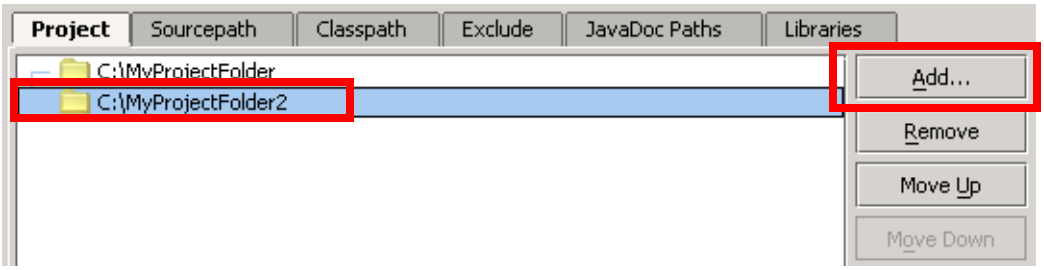


Figure 4.9. Folder added to project (612)

4.3.1.2. Move

4.14. Click on the added folder.
4.15. Click **Move up** and **Move down**. The location of the folder is changed.

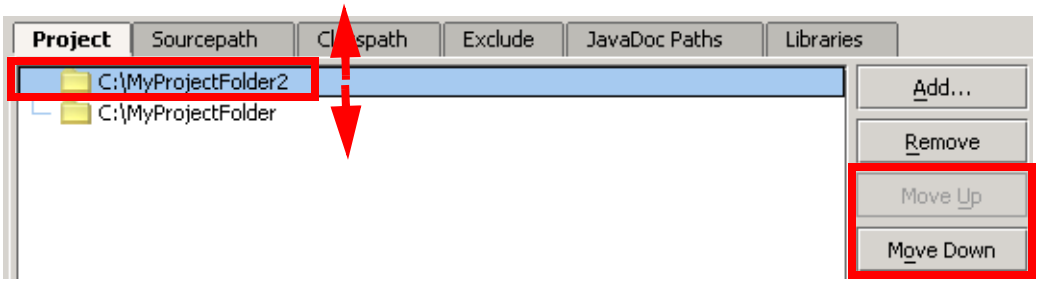


Figure 4.10. Folder moved (up and down) (613)

4.3.1.3. Remove

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

4.3.2. 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.

4.3.2.1. Add

4.17. Select the tab **Sourcepath**.
4.18. Select **Add**. The “Select Path” dialog appears.
4.19. Select the jar or dir.

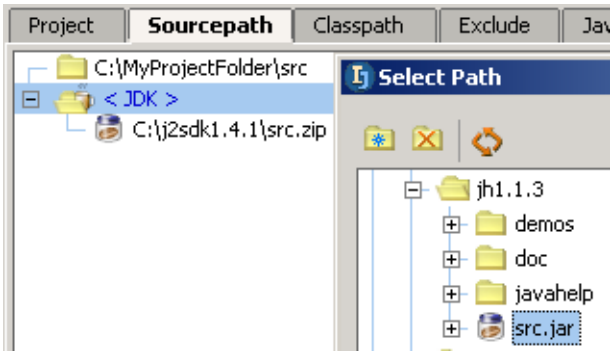


Figure 4.11. Select folder or jar (240)

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

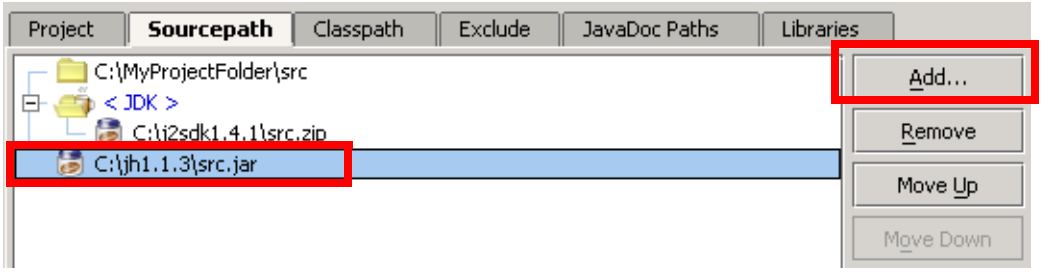


Figure 4.12. Folder added to project (615)

4.3.2.2. Move

Similar to **4.3.1.2. Move (page 55)**.

4.3.2.3. Remove

Similar to **4.3.1.3. Remove (page 55)**.

4.3.3. 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.

4.3.3.1. Add

4.21. Select the tab **Classpath**.
4.22. Select **Add**. The “Select Path” dialog appears.
4.23. Select the jar or dir.

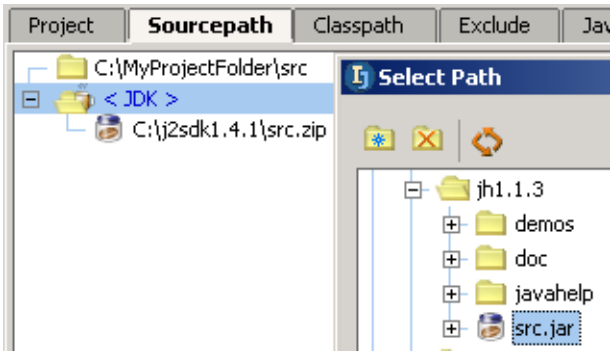


Figure 4.13. Select folder or jar (240)

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

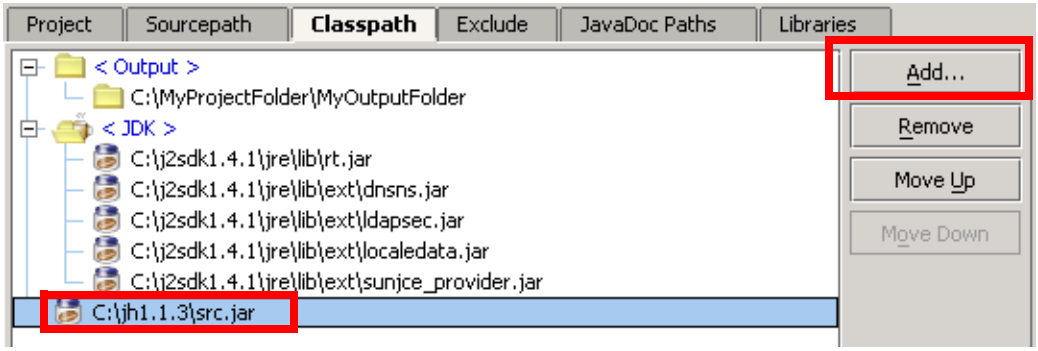


Figure 4.14. Class jar added to project (616)

4.3.3.2. Move

Similar to **4.3.1.2. Move (page 55)**.

4.3.3.3. Remove

Similar to **4.3.1.3. Remove (page 55)**.

5. Directories (packages)

20021007TTT 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. New / Delete (page 59)** describes how create and delete directories and packages.
- **5.2. Views (page 61)** describes the various views of directories that IDEA provides.

5.1. New / Delete

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

With IDEA you can

- Create a **New package**
- Create a **New directory**
- **Delete package / directory**

5.1.1. New package

Note: You cannot create a package on an output directory.

5.1. Right-click on **\src**.
5.2. Select **New | Package**. The dialog “New package” appears.

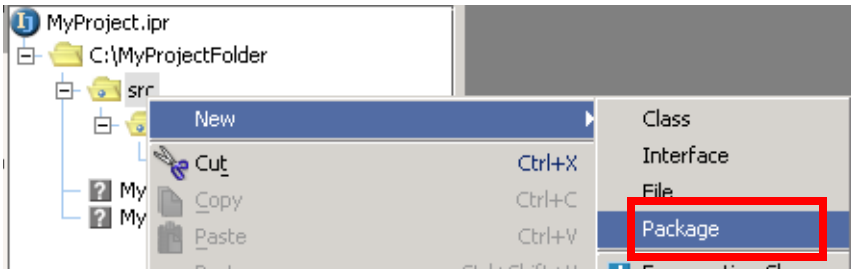


Figure 5.1. New package (623)

5.3. For the package name enter **MyPackage2**.
5.4. Click **OK**.

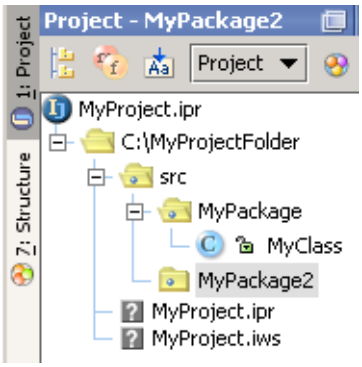


Figure 5.2. New package (625)

5.1.2. New directory

Note: You cannot create a subdirectory of a package.

5.5. Right-click on **\MyProjectFolder**.
5.6. Select **New | Directory**. The dialog “New directory” appears.

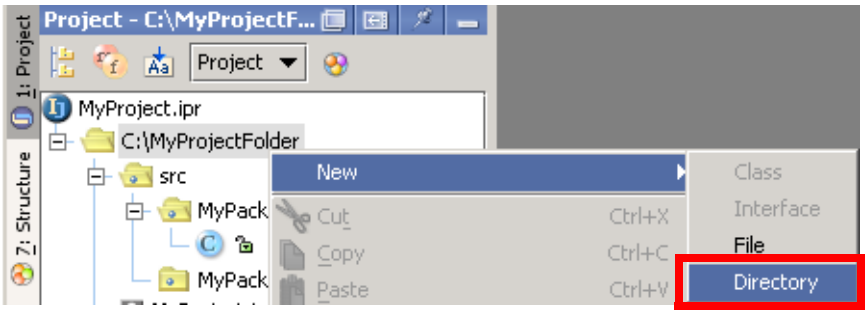


Figure 5.3. New directory (626)

5.7. For the directory name enter **MyDirectory**.
5.8. Click **OK**.

Figure 5.4. New directory (239)

5.1.3. Delete package / directory

5.9. Select the **package / directory**.
5.10. Click **DEL**. A warning dialog appears.
5.11. Click **OK**. The package / directory and any files / subdirectories are deleted.

5.2. Views

There are 3 views that show packages / directories

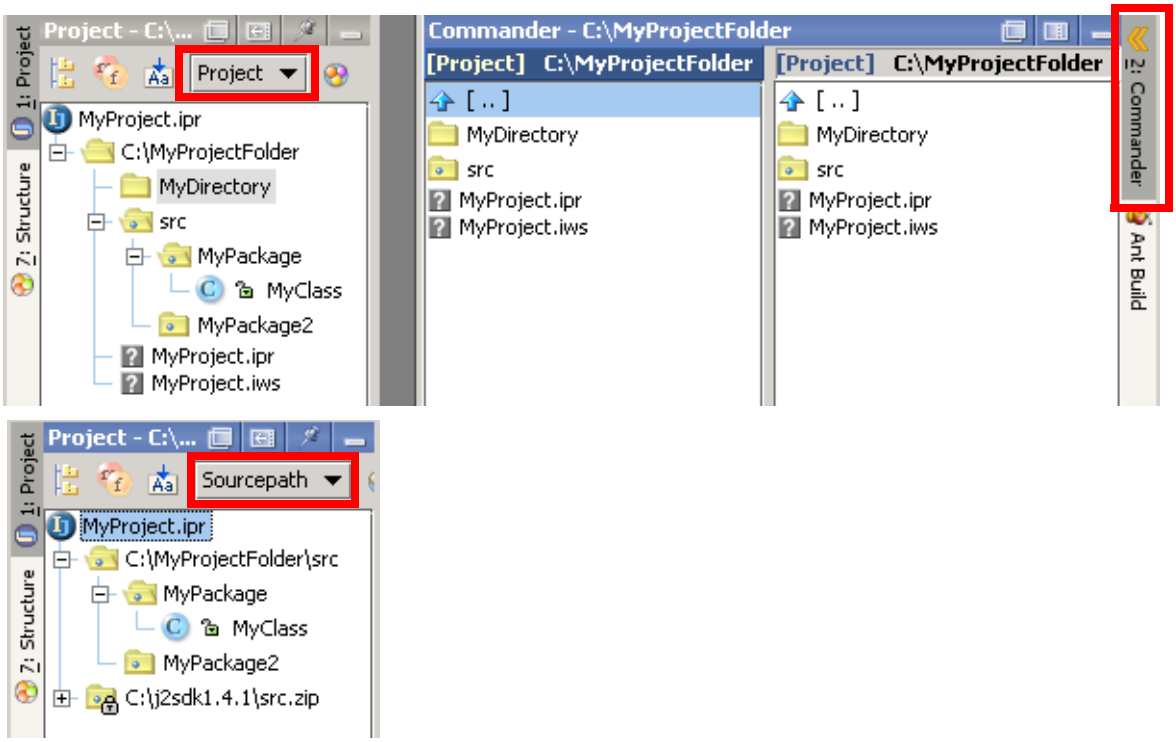


Figure 5.5. 3 views of packages / directories (640,641)

5.2.1. Project tool

Flatten packages

5.12. Select the **Sourcepath**.

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

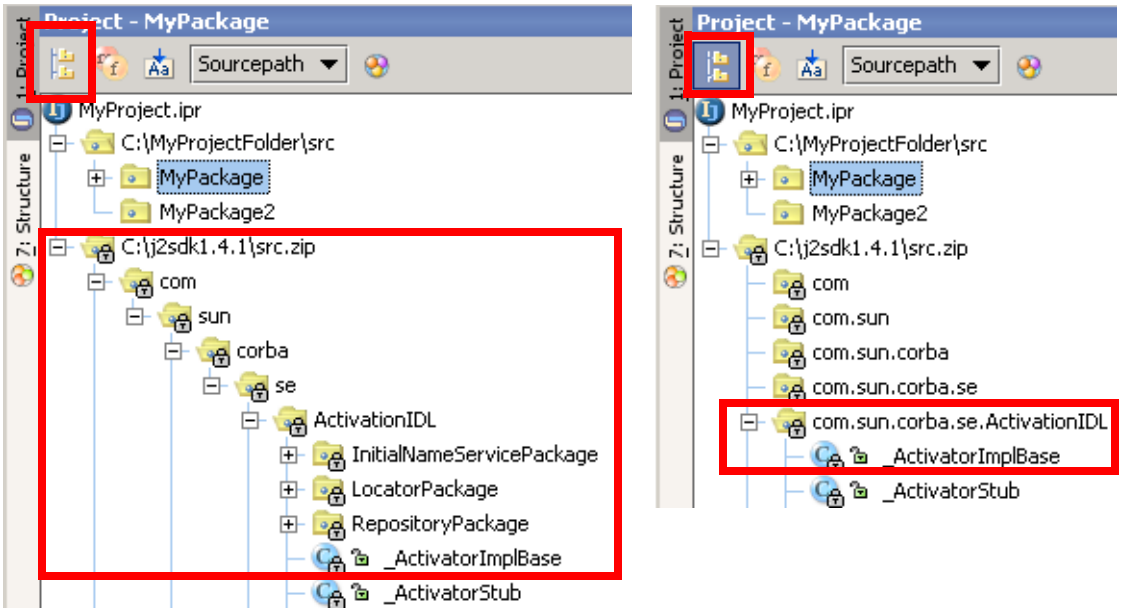


Figure 5.6. Packages not flattened / flattened (645,646)

5.2.2. Commander

The commander allows you to view directory contents and open files using a Windows Commander style interface.

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

5.15. Double-click on **main**. MyClass.java is opened in an editor.

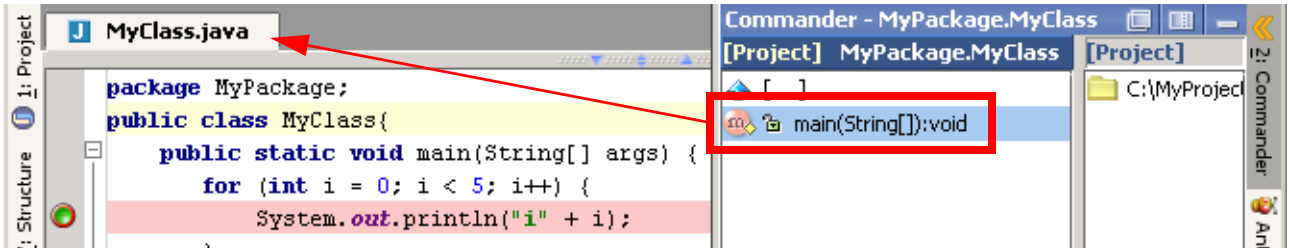


Figure 5.7. Opening a file from within Commander (650)

6. Files

20021014TTT last edit.
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 and files within IDEA:

- **6.1. Basic file operations (page 63)** describes how create, open, close, etc. files.
- **6.2. Types (page 73)** describes the types of files.
- **6.3. Views (page 77)** describes the various views of files.
- **6.4. Modify content (page 87)** describes the file editors and default settings.

6.1. Basic file operations

The following file operations are supported:

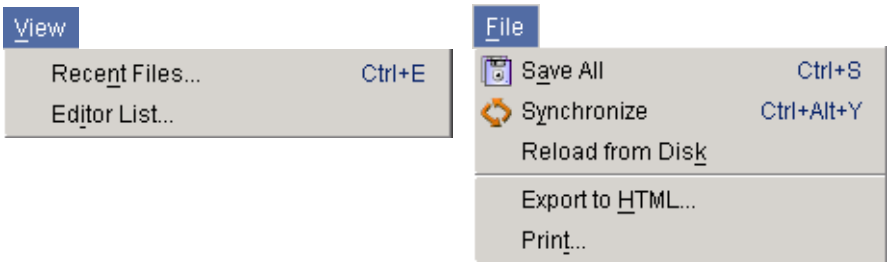


Figure 6.1. Basic file operations (444,443,446,445)

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 ??

6.1. Right-click on a package.
6.2. Select **New | Class**. The dialog “New Class” appears.

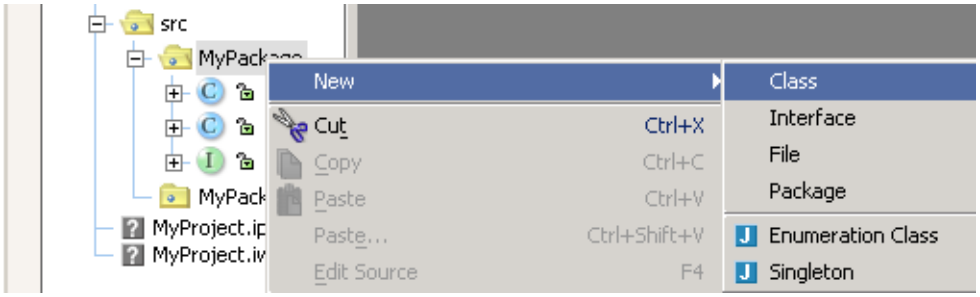


Figure 6.2. New class (627)

6.3. Enter the new class name.

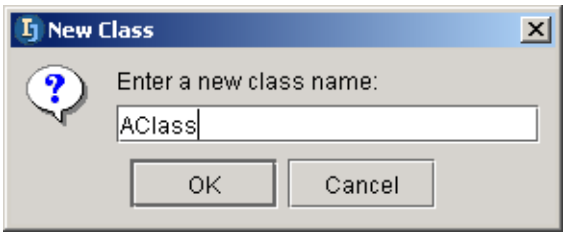


Figure 6.3. New class name (628)

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

To change the default contents of the class, see **6.4.2.1. Class template (page 88)**.

6.1.1.2. Interface

6.5. Right-click on a package.
6.6. Select **New | Interface**. The dialog “New Interface” appears.
6.7. Enter the new Interface name.
6.8. Click **OK**. The interface files is created.

To change the default contents of the interface, see **6.4.2.2. Interface template (page 88)**.

6.1.1.3. File (text)

6.9. Right-click on a package or directory.
6.10. Select **New | File**. The dialog “New File” appears.
6.11. Enter the new file name and extension.
6.12. Click **OK**. The text file is created.

6.1.2. Open

You can open a file

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

6.1.2.1. 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 77)**)

6.1.2.2. Autoscroll to source

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

6.14. Click the **Autoscroll to source** icon ( ).
6.15. Click on **MyPackage**.
6.16. Move the focus (with the **arrow down key**) to MyClass. MyClass.java is opened in the editor.

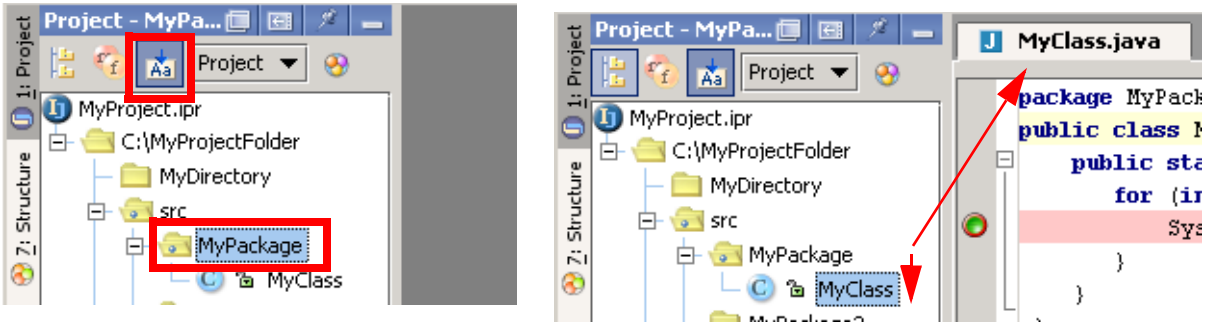


Figure 6.4. Autoscroll to source (647,648)

6.1.2.3. From “Open file” dialog

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

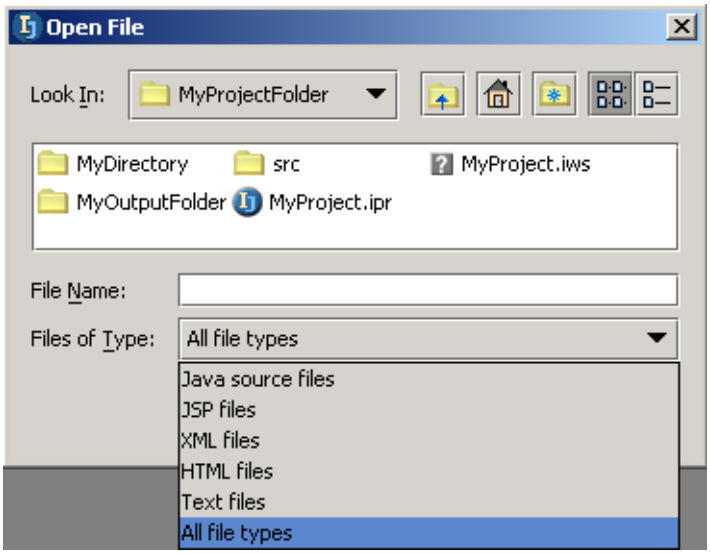


Figure 6.5. Dialog “Open file” (653)

6.18. Double-click on a file to open.

6.1.2.4. Reload from disk

6.19. Add any text to MyClass.java.
6.20. Select **File | Reload from disk**.

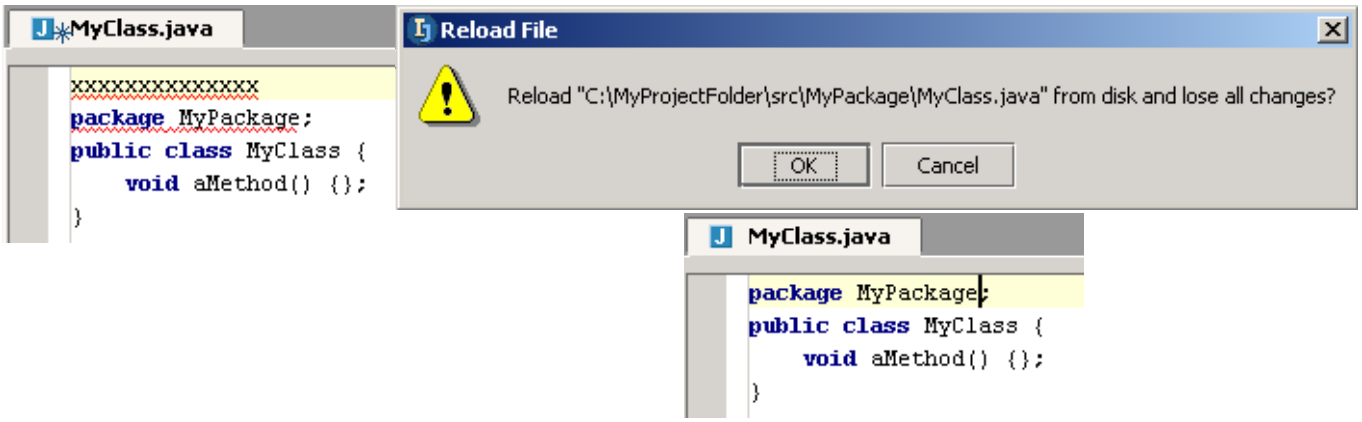


Figure 6.6. Reload file (209,208,207)

6.1.2.5. Recent files

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

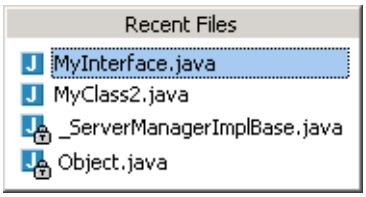


Figure 6.7. List of recent files (651)

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

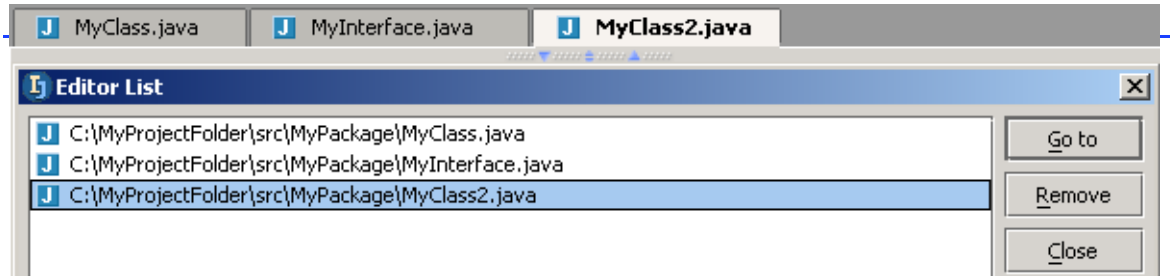
6.1.3. Synchronize

20021008TTT: how do i demo synchronize??

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

~~6.1.4. Editor list~~

~~6.24. Select **View | Editor list...**. A list of the open editors appears.~~



~~Figure 6.8. Editor list (652)~~

6.1.5. Save all

6.25. 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 quick start.

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

6.1.6. Close

You can

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

6.1.6.1. Close Active

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

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

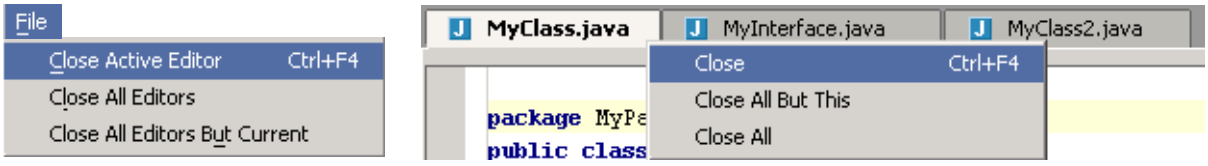


Figure 6.9. Close file menus (654,655,206)

6.1.6.2. Close All

6.27. 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**.

6.1.6.3. Close All but current

6.28. 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**.

6.1.7. Delete

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

- **Safe delete**
- **Recover**

6.1.7.1. Safe delete

6.29. Select a class in the project directory.

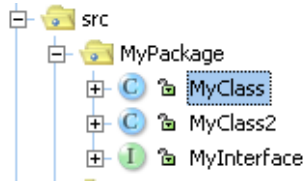


Figure 6.10. Select class (205)

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

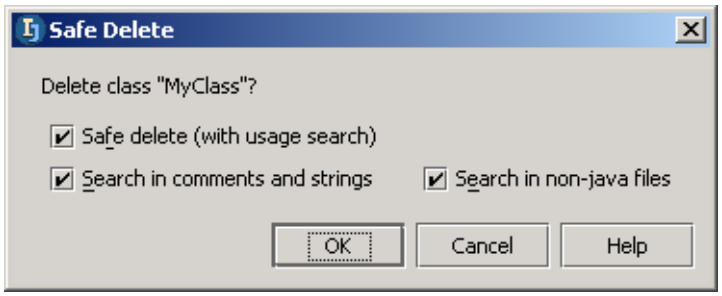


Figure 6.11. Safe delete (204)

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

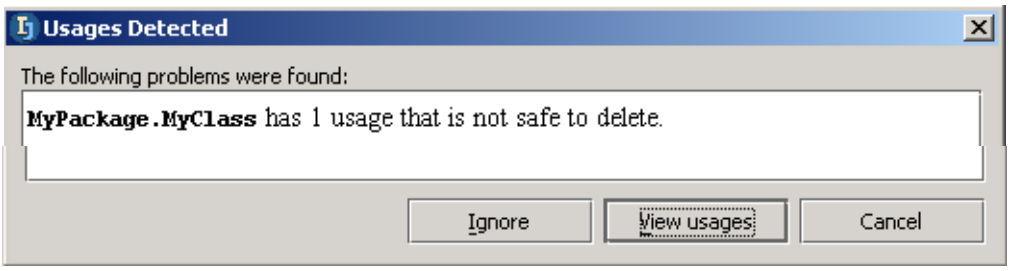


Figure 6.12. Usages detected (203,202)

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

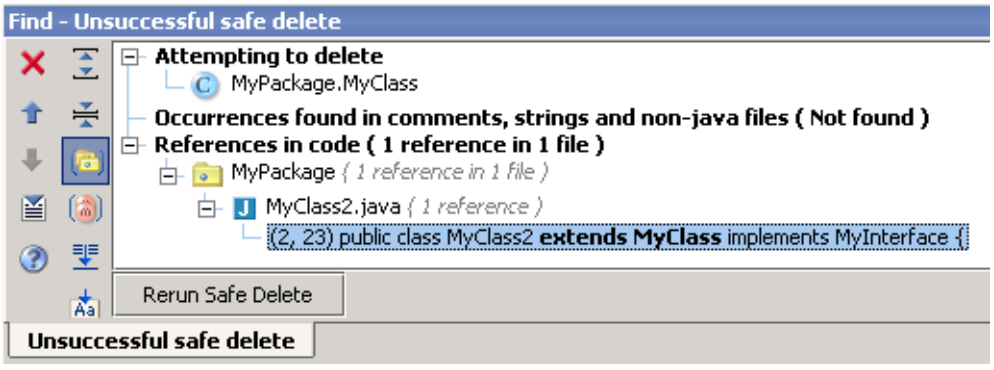


Figure 6.13. Found usages (201)

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

6.34. Click **OK**. Again the dialog “Usages detected” appears.
6.35. Click **Ignore**. The file is deleted.

6.1.7.2. Recover

To recover a deleted file:

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

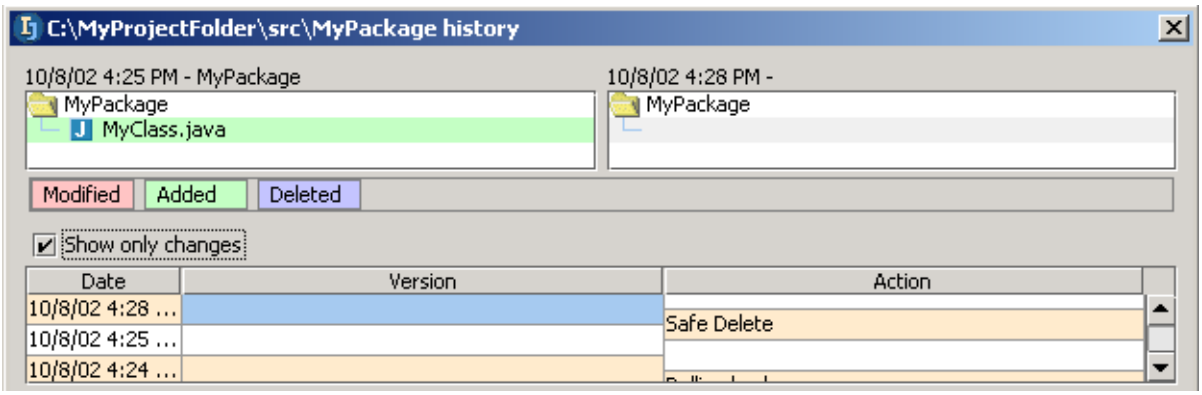


Figure 6.14. Safe delete action in history (200,199)

6.39. Right-click.
6.40. Click on **Rollback**. A confirmation dialog appears.
6.41. Click **OK**. The file is restored (the deletion is rolled back).

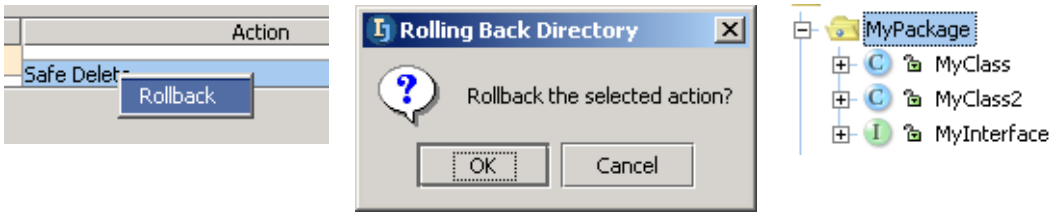


Figure 6.15. Rollback of deletion (198,197,196)

6.1.8. Export to HTML

6.42. Open the file in the editor.

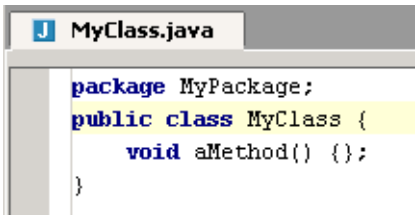


Figure 6.16. File for export in editor (657)

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

6.44. Select the output directory.

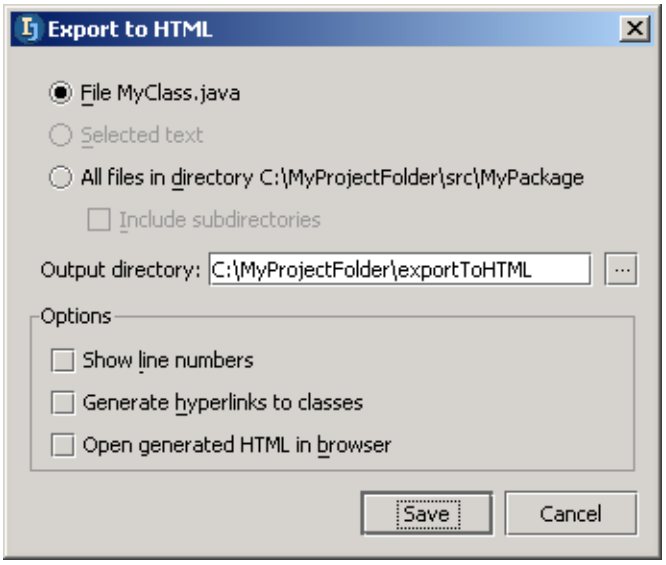


Figure 6.17. Export to html dialog (656)

6.45. Click **Save**. C:\idea640**\exportToHTML\hello\HelloWorld.java.html** is created and opened in a browser.

Figure 6.18. File exported to HTML (658)

6.1.9. Print

6.46. Open the file in the editor.
6.47. Select **File | Print...**. The dialog “Print” appears.

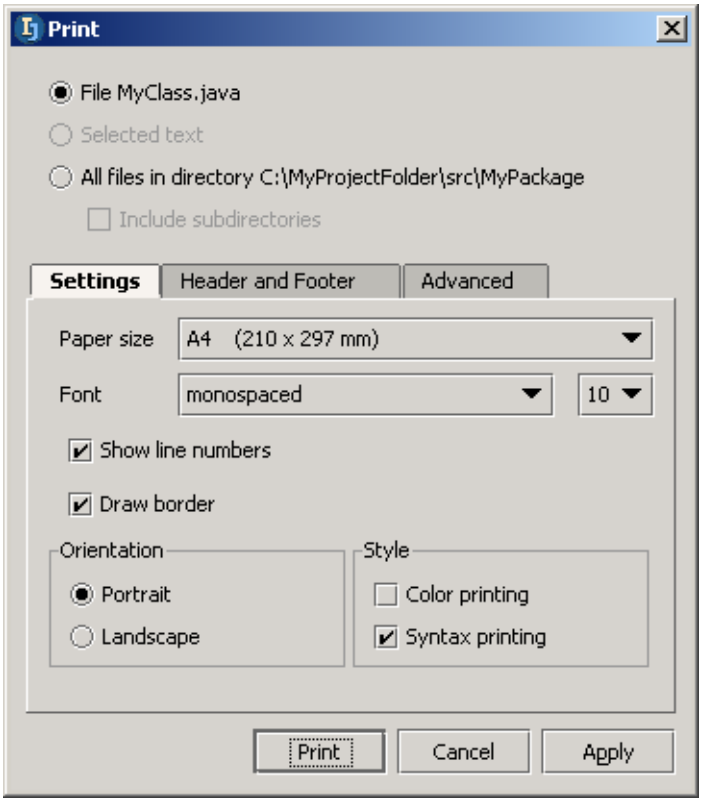


Figure 6.19. Export to html dialog (659)

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

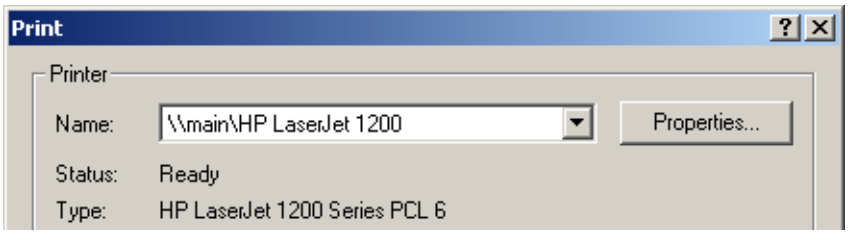


Figure 6.20. Windows print dialog (670)

6.49. Click **OK** to print.

6.2. Types

IDEA recognizes

- **6.2.1. Default file types (page 73)**
- **6.2.2. Custom filetypes (page 75)**

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

- **7.3. Colors and fonts X (page 118)** and
- **7.4. Code style X (page 126)**

6.2.1. Default file types

IDEA recognizes the following default file types:

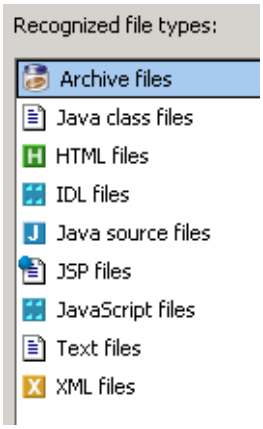


Figure 6.21. Default recognized file types (811)

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.

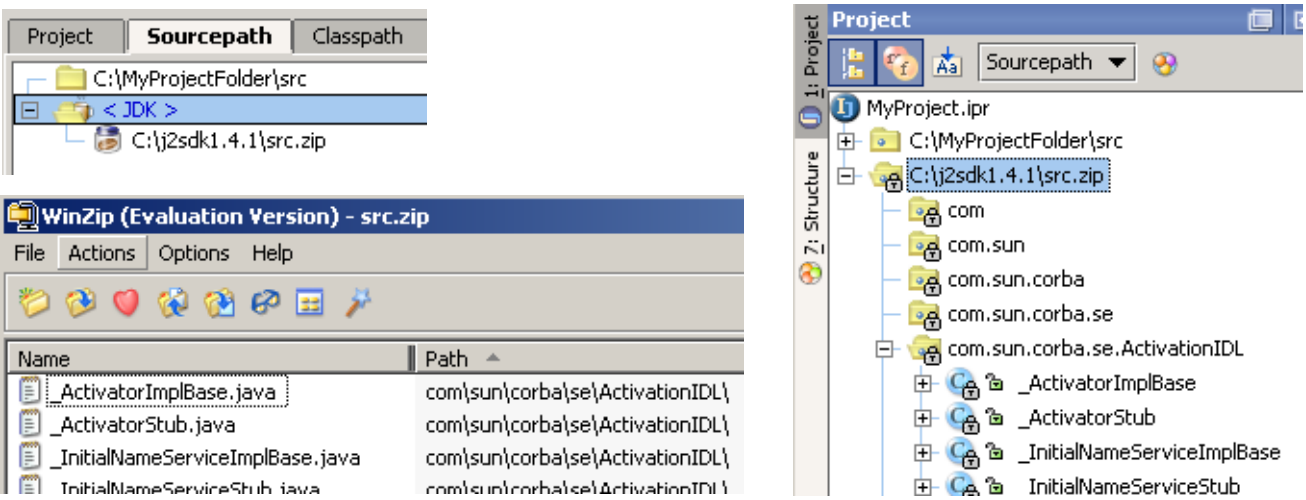


Figure 6.22. Archive file displayed in IDEA (805,806,807)

Java class files

IDEA recognizes automatically Java class files.

HTML

Colors and fonts for HTML files are described in **section 7.3.3. HTML X (page 120)**.

IDL

IDL files are displayed with special colors or fonts.

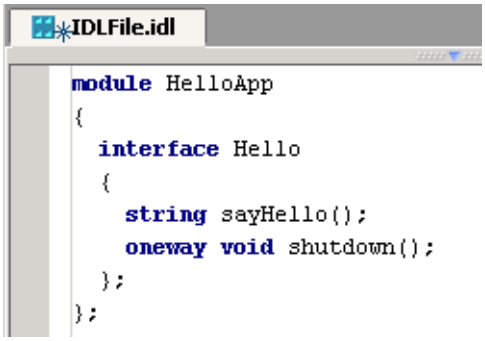


Figure 6.23. IDL (810)

Java source

Colors and fonts for Java sources files are described in **section 7.3.2. Java X (page 119)**.

JSP

Colors and fonts for JSP files are described in **section 7.3.5. JSP X (page 122)**.

JavaScript

JavaScript files are displayed with special colors or fonts.

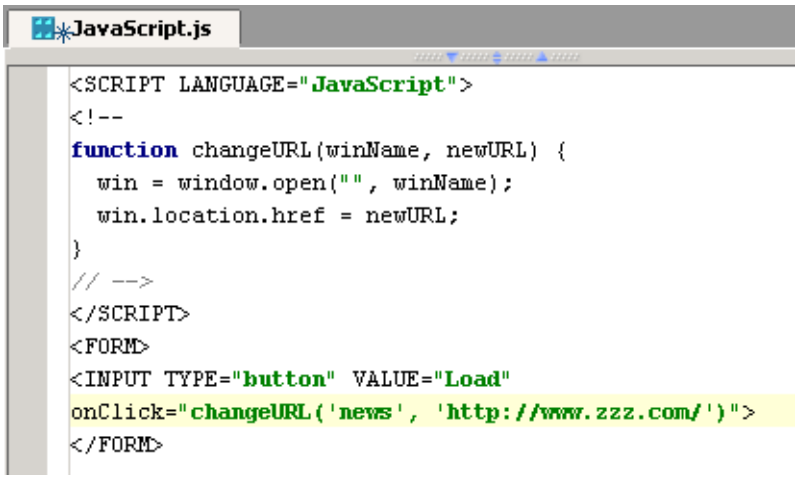


Figure 6.24. JavaScript (809)

Text

Text files are displayed with special colors or fonts.

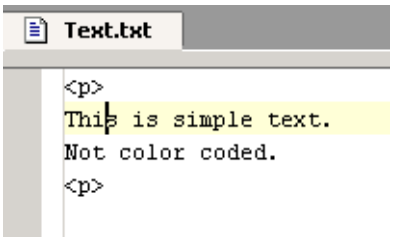


Figure 6.25. Simple text file (808)

XML

Colors and fonts for JSP files are described in **section 7.3.4. XML X (page 121)**.

6.2.2. Custom filetypes

IDEA automatically recognizes files types based on the file extension.
In the IDEA Options / Filetypes you can

6.2.2.1. Add an extension

6.50. In “Recognized file types” select **Text files**.
6.51. In “Registered extensions” click **Add...**. The dialog “Add extension” appears.
6.52. As the new extension enter **text**.
6.53. Click **OK**. The new extension is in the list.

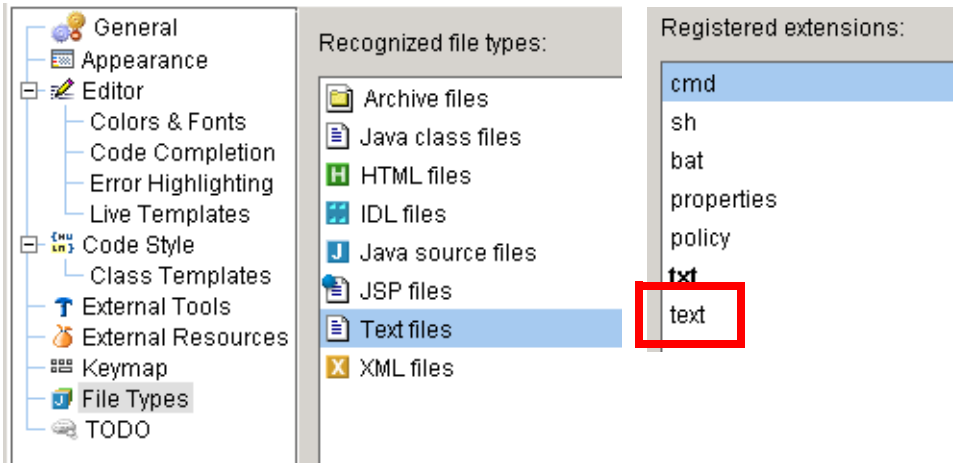


Figure 6.26. New recognized extension (421,420)

6.2.2.2. Add a file type and extension

6.54. In “Recognized file types” click **Add...**. The dialog “Add New File Type” appears.
6.55. Enter the parameters for the file type (see the diagram).

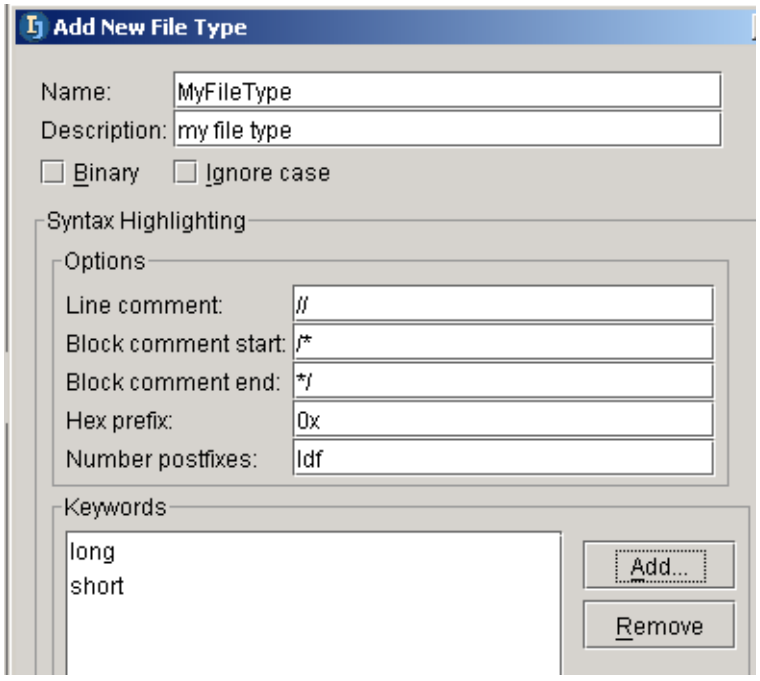


Figure 6.27. New file type parameters (419)

6.56. Add keywords.
6.57. Click **OK**. The file type appears in the list of recognized file types.
6.58. Add registered extension **myext**.

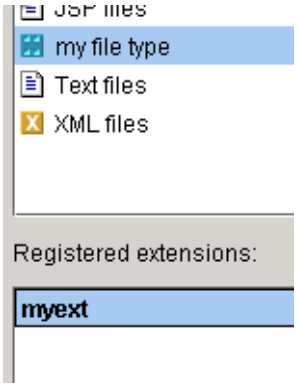


Figure 6.28. New file type and extensions (418)

6.3. Views

There are 4 views that show the contents of a file

- **6.3.1. Editor (page 78)**
- **6.3.2. Project tool (show members) (page 78)**
- **6.3.3. Commander (page 78)**
- **6.3.4. (File) Structure tool (page 79)**

Also the

- **6.3.5. Hierarchy tool (page 83)**

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

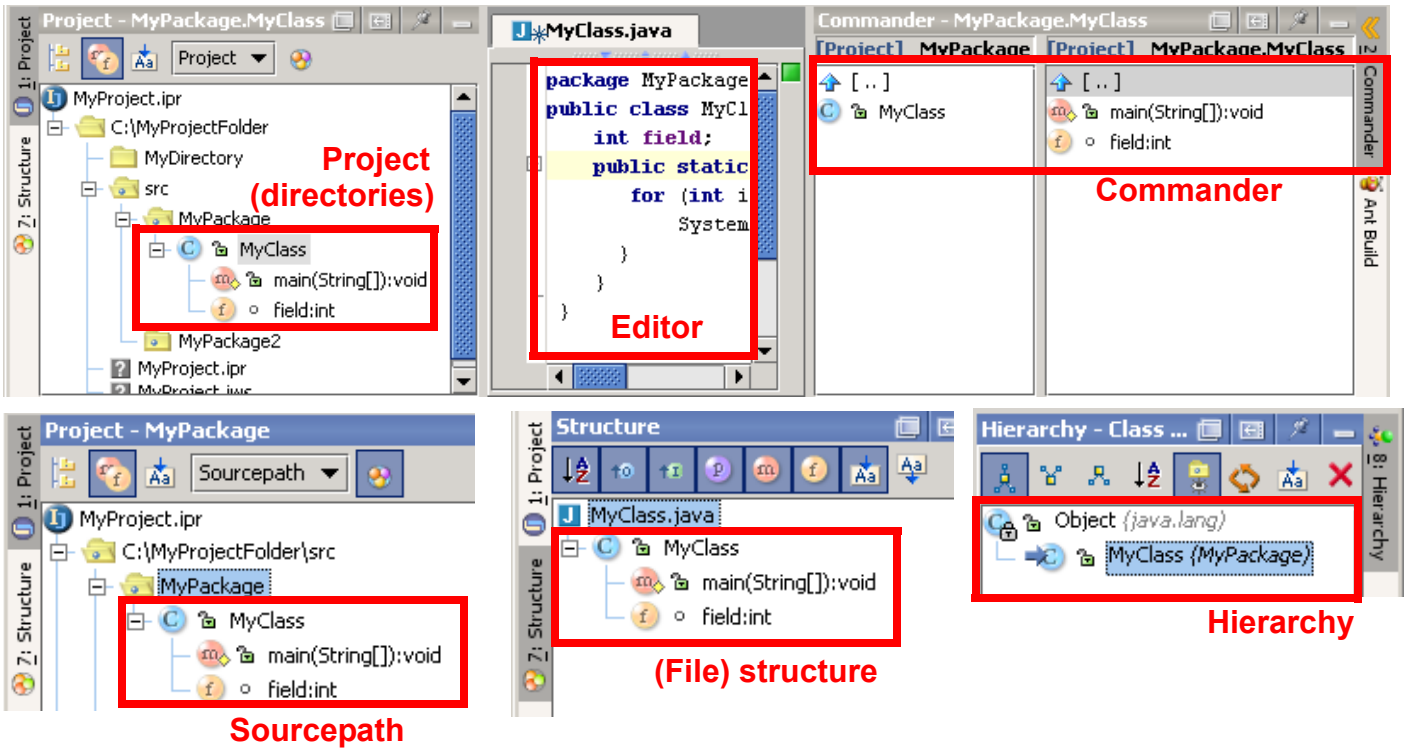


Figure 6.29. 5 views of a file (238,237,236,235)

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 91)**
- **Chapter 8. Code Automation (page 145)**
- **Chapter 9. Code Refactoring X (page 183)**

6.3.2. Project tool (show members)

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

6.59. Add the following to MyClass:

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

6.60. Select the **Project** view.

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

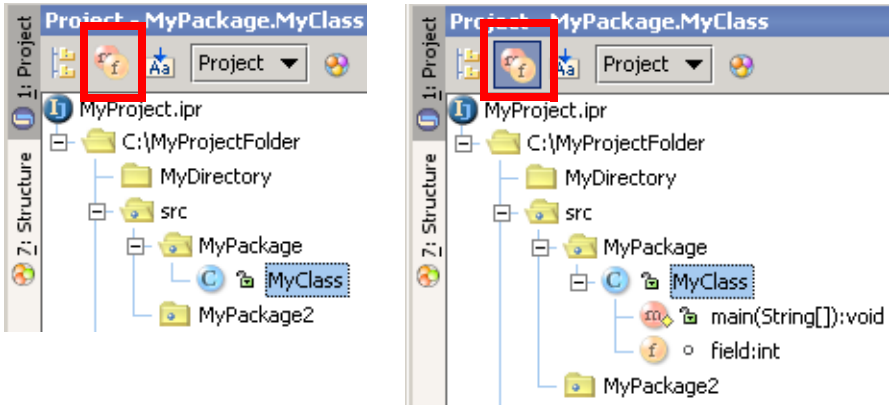


Figure 6.30. Members not shown / shown (234,233)

6.3.3. Commander

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

6.62. In the Commander: Navigate to **APackage.AClass**.

6.63. Double-click on **main**. AClass.java appears in the editor.

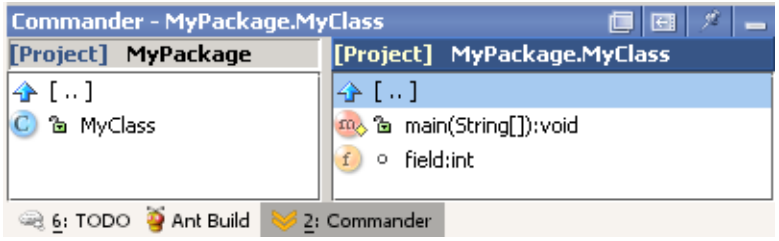


Figure 6.31. Viewing file contents with Commander (232)

6.3.4. (File) Structure tool

The Structure tool shows the structure of a class.

6.3.4.1. Tool variations

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

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

6.3.4.1.1. Normal

6.64. Modify **MyClass**:

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

6.65. Create **MyInterface**

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

6.66. Create **MyClass2**

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

6.67. Select the **MyClass2** editor.

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

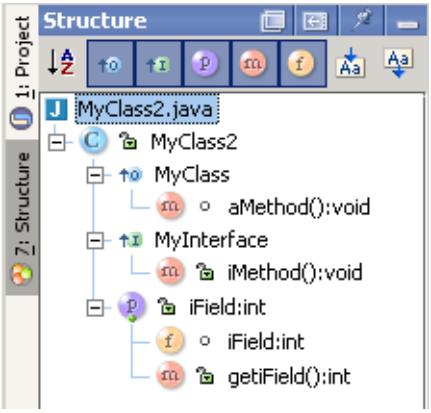


Figure 6.32. Structure tool (231)

6.3.4.1.2. From project tool

6.69. Close the **7. Structure** tool.

6.70. Open the **1. Project** tool.

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

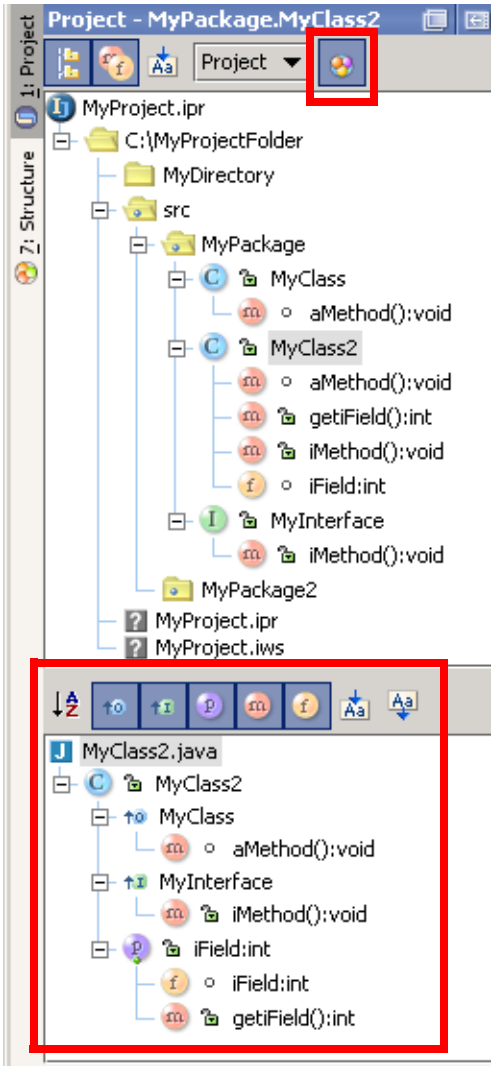


Figure 6.33. Structure tool as subset of Project tool (230)

6.3.4.1.3. Popup

6.72. Click in the **MyClass2** source text.

6.73. Select **View | File structure popop...**.

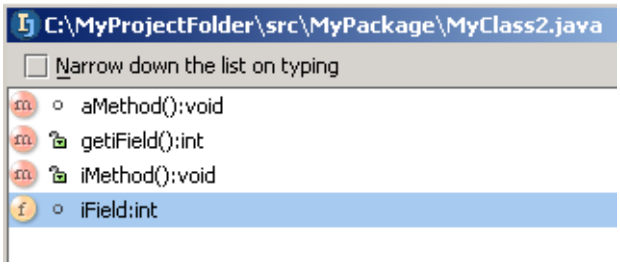


Figure 6.34. File structure popup (607)

6.3.4.2. Sort alphabetically

Select to sort alphabetically.

6.3.4.3. Group overriding methods

6.74. Unselect all.

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

6.76. Select the **Group overriding methods** ( ) icon.

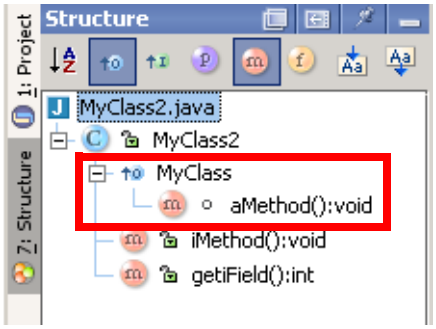


Figure 6.35. Group overriding methods (217)

6.3.4.4. Group implementation methods

6.77. Unselect the **Group overriding methods** ( ) icon.

6.78. Select the **Group implementation methods** ( ) icon.

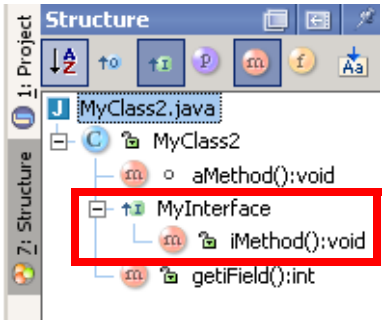


Figure 6.36. Group implementation methods (215)

6.3.4.5. Show properties

6.79. Unselect the **Group implementation methods** ( ) icon.

6.80. Select the **Show properties** ( ) icon.

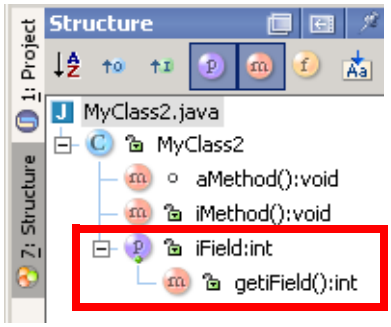


Figure 6.37. Show properties (213)

6.3.4.6. Show methods

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

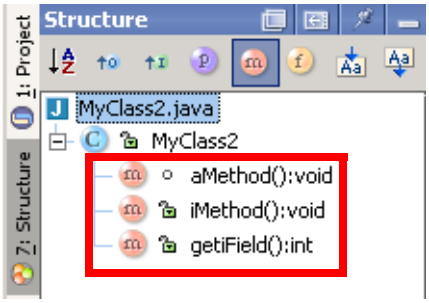


Figure 6.38. Show methods (212)

6.3.4.7. Show fields

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

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

Figure 6.39. 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

Shows the type hierarchy.

6.3.5.1.1. Class hierarchy

6.84. 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();
    };
 }
```

6.85. Modify **MyClass2**

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

6.86. Select the editor for **MyClass**.

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

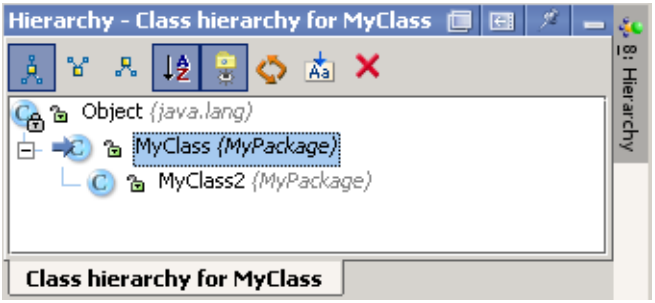


Figure 6.40. Class hierarchy (605)

6.3.5.1.2. Supertypes hierarchy

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

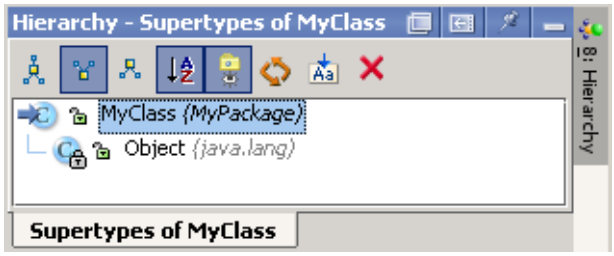


Figure 6.41. Supertypes hierarchy (228)

6.3.5.1.3. Subtypes hierarchy

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

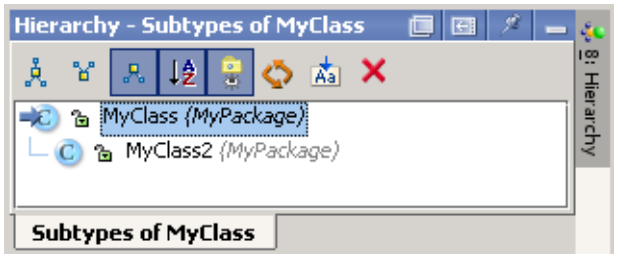


Figure 6.42. Subtypes hierarchy (225)

6.3.5.1.4. Sort alpabetically

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

6.90. In **MyClass** select method **aMethod()**.
6.91. Select **View | Method Hierarchy**. The method hierarchy is shown.

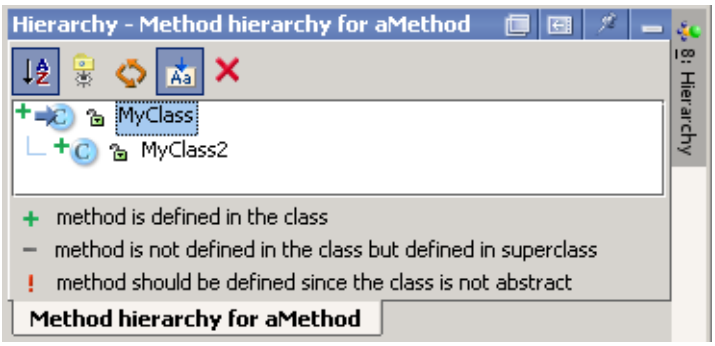


Figure 6.43. Method hierarchy (224)

6.3.5.2.2. Sort alpabetically

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

6.92. In **MyClass** select method **aMethod2()**.
6.93. Select **View | Call Hierarchy**. The call hierarchy is shown.

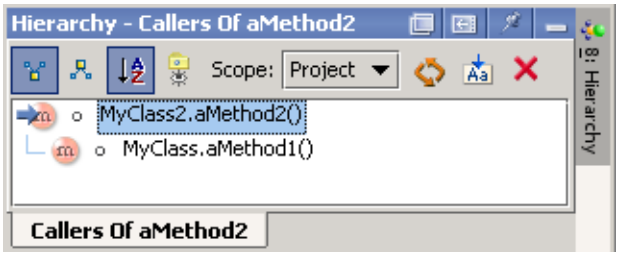


Figure 6.44. Call hierarchy (caller methods) (223)

6.3.5.3.2. Caller methods hierarchy

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

6.3.5.3.3. Callee methods hierarchy

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

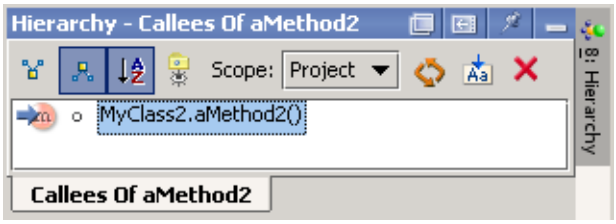


Figure 6.45. Call hierarchy (callee methods) (220)

6.3.5.3.4. Sort alpabetically

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.

6.4. Modify content

You can modify content with

6.4.1. Editors

In IDEA, file content is only modified in the editor.

Although members can be deleted from some views (Project for example), no view other than the editor allows creation of members, fields, etc.

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

6.4.2. Defaults

IDEA allows you to specify default contents with

- **Class template**
- **Interface template**

6.4.2.1. Class template

The class template determines the default text of a created class.

6.96. Select **Options | File templates...**. The dialog “File templates” tab “Templates” is opened.
6.97. Select **New Class**. Note the default contents.
6.98. Select tab **Includes**. Note the default contents.

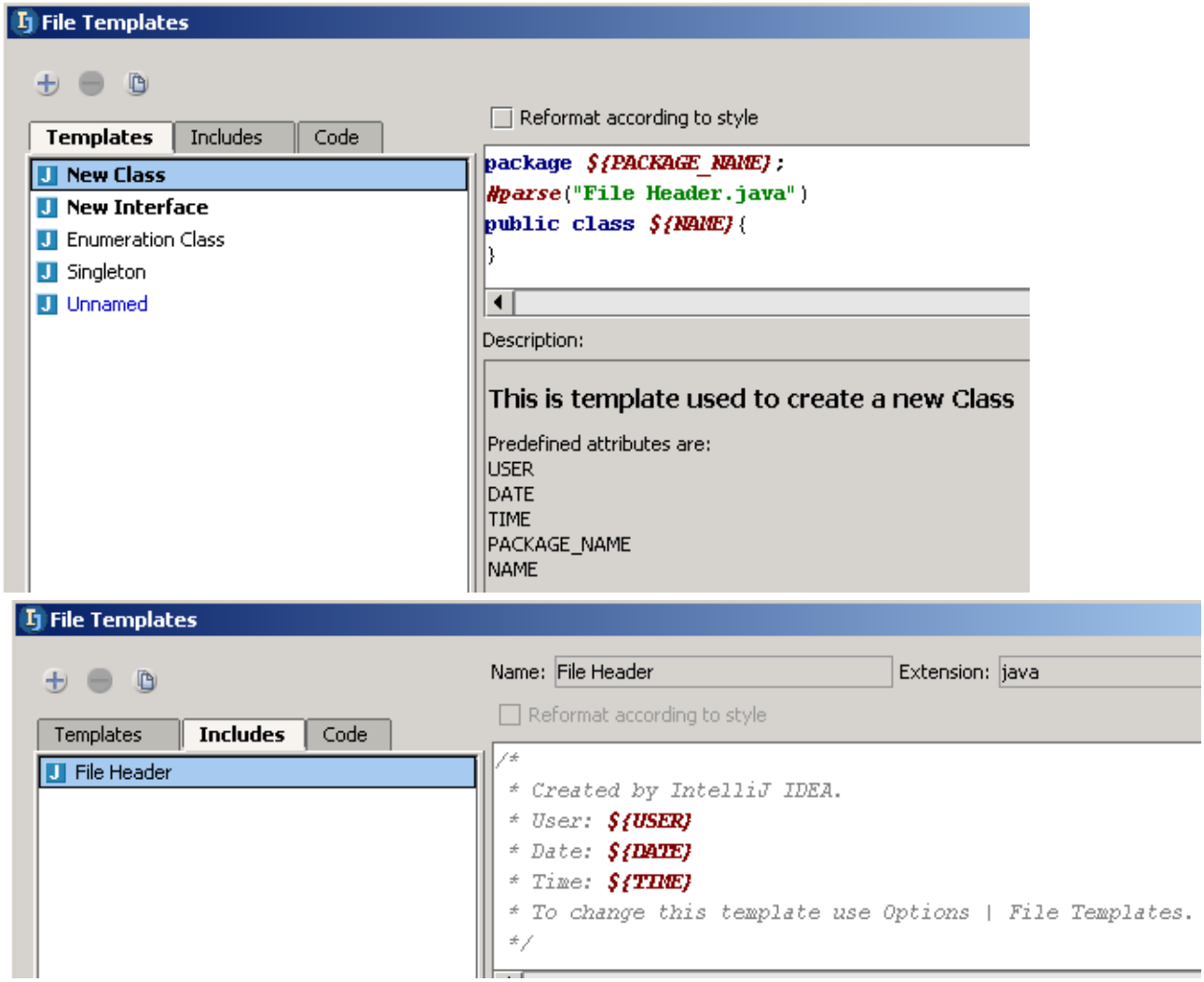


Figure 6.46. Class template (611,195)

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

6.4.2.2. Interface template

The interface template is analogous to the New Class template.

Part C. Editting files

20021018TTT last edit.

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

7. Editor X (page 91). Demonstrates how the ease-of-use and extensive functionality of the editor makes it the tool of choice for editting any type of code file.

8. Code Automation (page 145). Demonstrates how the code automation functions provided by IDEA allow you to more easily and quickly create, modify, and improve code.

9. Code Refactoring X (page 183). Demonstrates how IDEA's extensive refactoring support makes code maintenance much easier.

10. Code Inspection X (page 223). Demonstrates how IDEA can automaticallly inspect your code and recommend / implement solutions or improvements.

11. Version control X (page 235). Demonstrates how IDEA's local versioning tool and compatible external tools (CVS, SourceSafe, StarTeam) provide complete and robust version control.

12. Java Doc X (page 247). 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

20021012TTT: last edit.

contacts: valentin (maxim?)

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

Note that chapters

also introduce editting functionality.

7.1. Editting text X

IDEA provides extensive functionality for basic text editting tasks

7.1.1. Undo / Redo

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

Undo CTRL-Z

Undo the previous action.

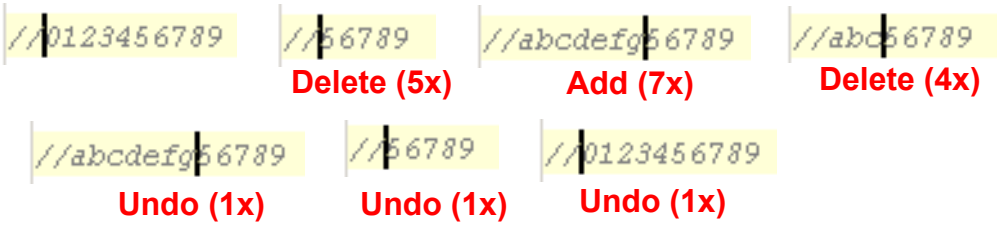


Figure 7.1. Undo (813-819)

Redo CTRL-SHIFT-Z

Redo the an undone action.

7.1.2. Select

The folllowing 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 7.2. Select line / word at caret (820,821,822)

Unselect word at caret CTRL-SHIFT-W

Unselect the entire word that contains the caret.

7.1.3. 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.

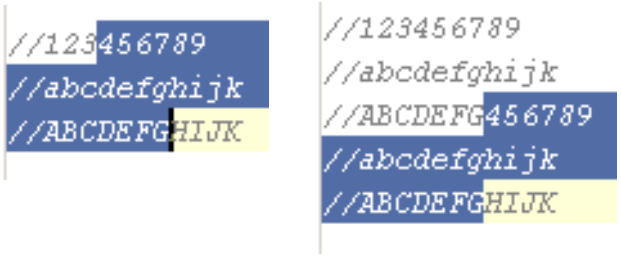


Figure 7.3. Duplicated block (732,733)

Delete line at caret CTRL-Y

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

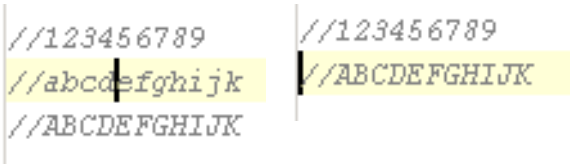


Figure 7.4. Delete line at caret (734,735)

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.

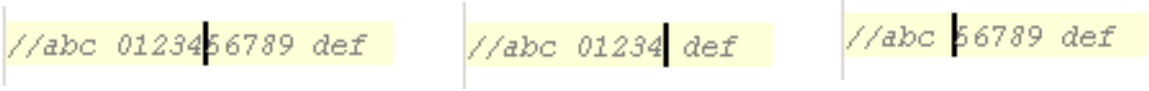


Figure 7.5. Delete to word end / start (736,737,738)

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 7.6. Non-insert mode (overwrite) (739,740)

7.1.4. Move / Scroll

IDEA provides a wide variety of move and scroll functions:

7.1.4.1. 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.

Figure 7.7. 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.

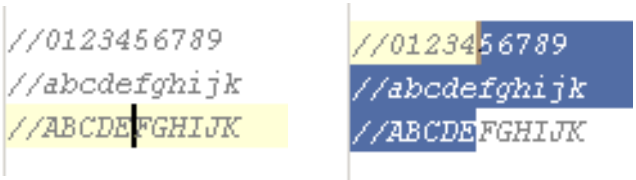


Figure 7.8. Up with selection (743,744)

Down with selection SHIFT-DOWN

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

7.1.4.2. 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.

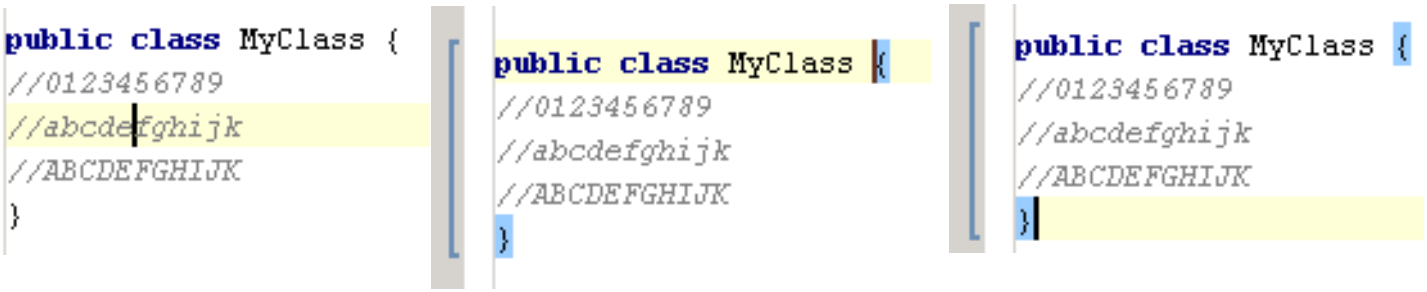


Figure 7.9. Move to code block start / end (745,746,747)

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).

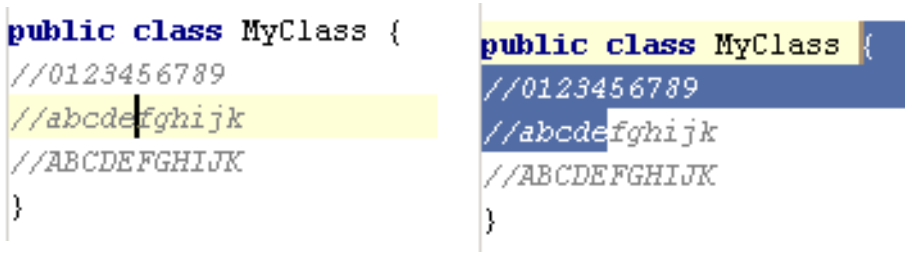


Figure 7.10. Move to code block start with select (748,749)

7.1.4.3. Within line

Move line end END

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

Figure 7.11. 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.

Figure 7.12. 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.

7.1.4.4. 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 place in or (if the caret was not placed in a word) to the start of the word before the caret.

Figure 7.13. 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 place 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].

Figure 7.14. 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.

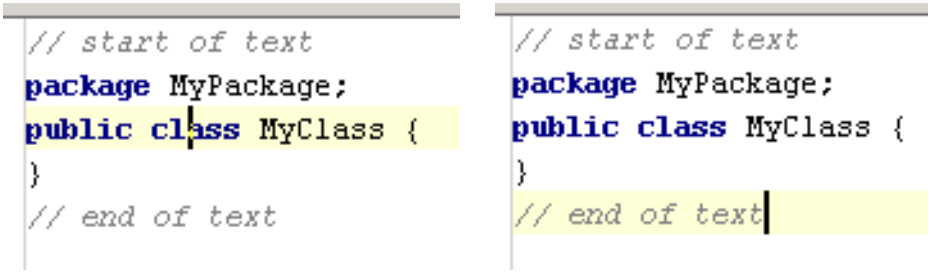


Figure 7.15. Move to text end (753,754)

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.

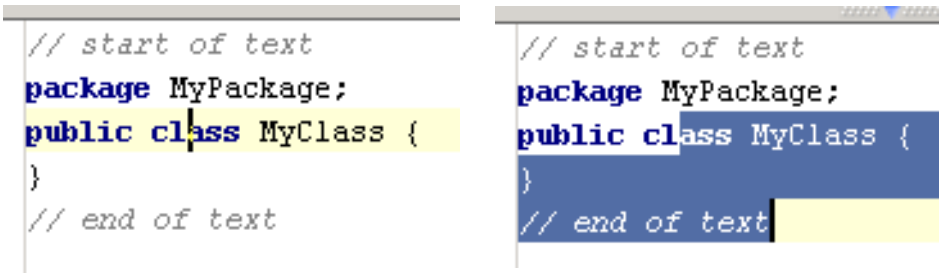


Figure 7.16. Move to text end with select (753,755)

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.

7.1.4.5. Page

Page down PAGE DOWN

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

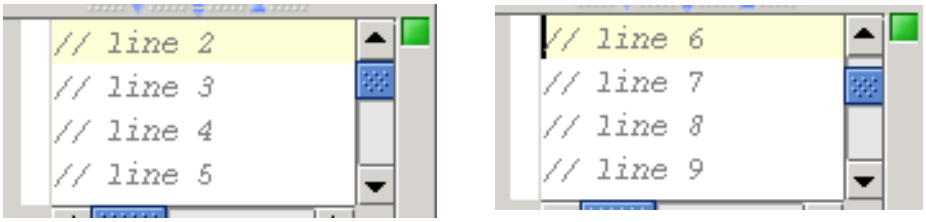


Figure 7.17. Page down (761,762)

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).

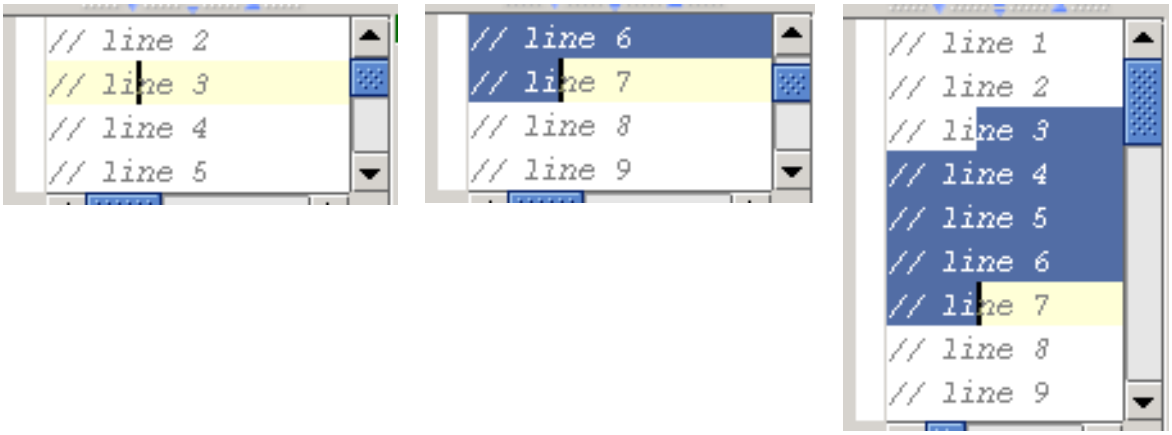


Figure 7.18. Page down with selection (763,764,765)

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).

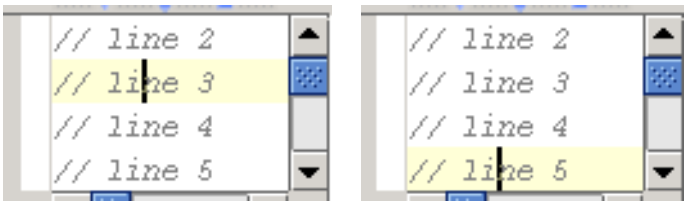


Figure 7.19. Go page bottom (766,767)

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.

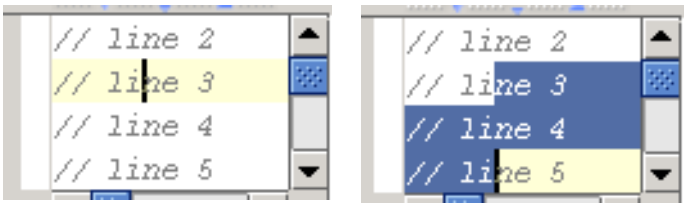


Figure 7.20. Go page bottom with selection (766,768)

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.

7.1.4.6. 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.

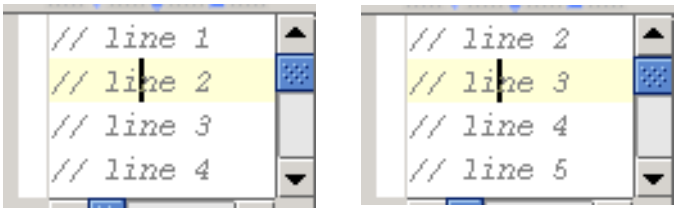


Figure 7.21. Scroll down (769,770)

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 sot that the caret is in the center.

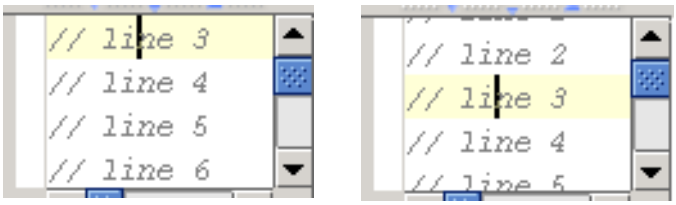


Figure 7.22. Scroll to center (772,773)

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.

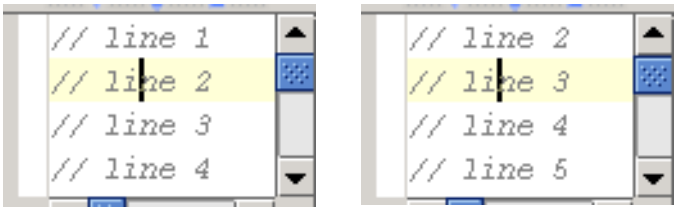


Figure 7.23. Move down and scroll (769,770)

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].

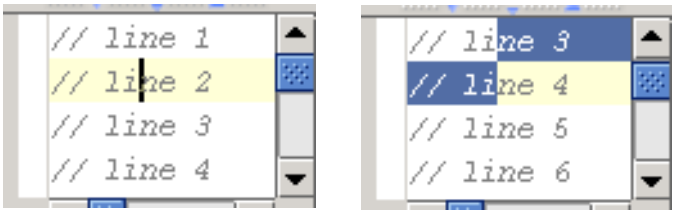


Figure 7.24. Move down and scroll with selection (769,771)

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].

7.1.5. 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.

Figure 7.25. 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.

Figure 7.26. Unident 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.

Figure 7.27. 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].

Figure 7.28. 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.

Figure 7.29. 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.

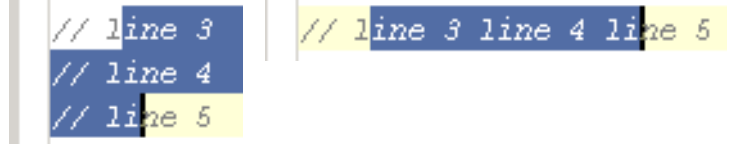


Figure 7.30. Join selected lines (783,784)

7.1.6. Modify text

Toggle case CTRL-SHIFT-U

Select text. Clicking CTRL-SHIFT-J 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.

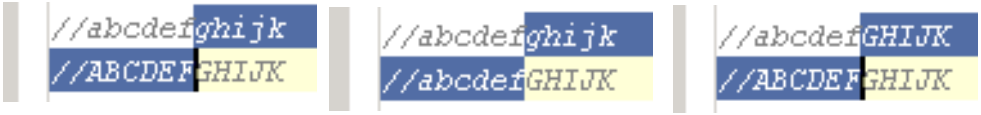


Figure 7.31. Toggle text case (785,786,787)

~~?? Choose lookup item~~
~~?? Choose lookup item replace~~
~~?? Next template variable~~
~~?? Previous template variable~~

7.2. Find / Navigation

IDEA provides the following find and navigation functions:

7.2.1. Bookmarks

IDEA support the following functions for bookmarks:

7.2.1.1. Create

7.1. Place the cursor in a file.

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

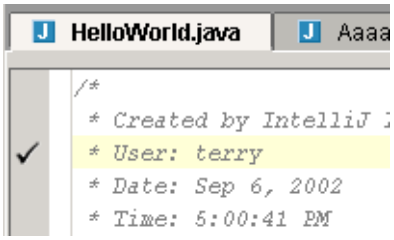


Figure 7.32. Bookmark in file (672)

7.3. Create bookmarks in several other files.

7.2.1.2. Show

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

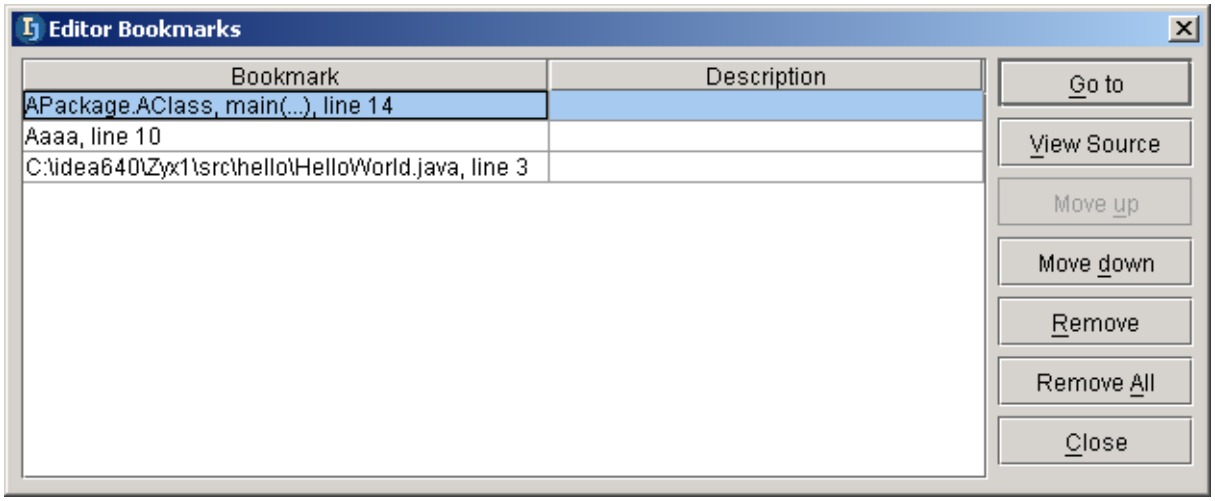


Figure 7.33. Bookmarks editor (673)

7.2.1.3. View source

7.5. Select a bookmark.

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

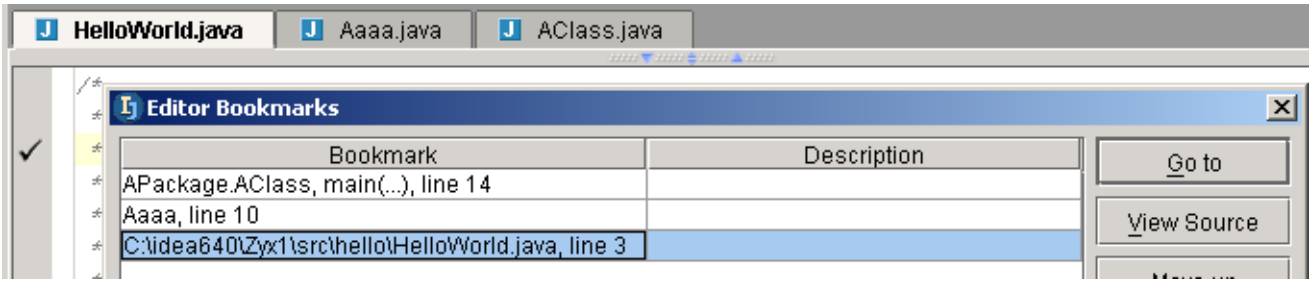


Figure 7.34. View source (674)

7.2.1.4. Go to

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

7.2.1.5. Describe

7.7. Click in the column **Description** for a bookmark.
7.8. Type in the description text.

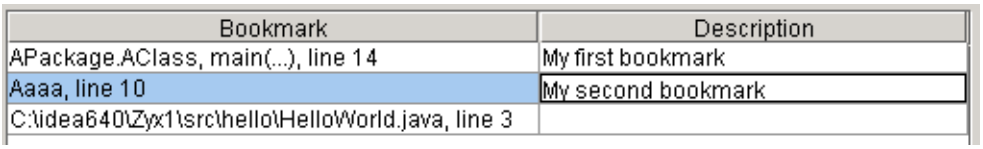


Figure 7.35. Bookmark description (675)

7.2.1.6. Move

7.9. Select a bookmark.
7.10. Click **Move up / Move down** to move the bookmark.

7.2.1.7. Remove

7.11. Select a bookmark.
7.12. Click **Remove** to delete the bookmark.

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

7.2.2. Find / Replace

IDEA supports the following Find / Replace variations:

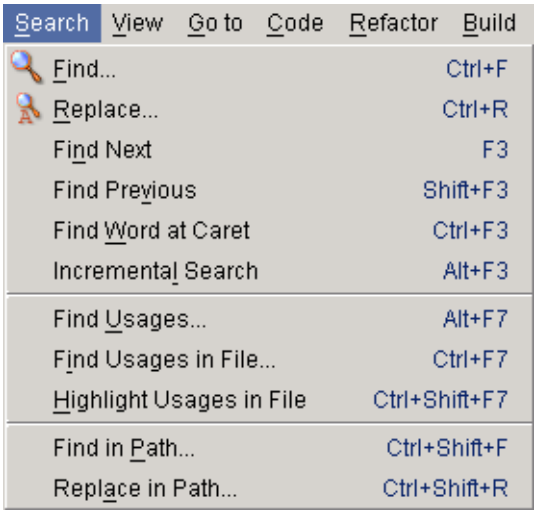


Figure 7.36. Find/replace menu (676)

7.2.2.1. Find (in file)

7.13. Open a file.
7.14. Select a word.
7.15. Select **Search | Find**. The dialog “Find text” appears.

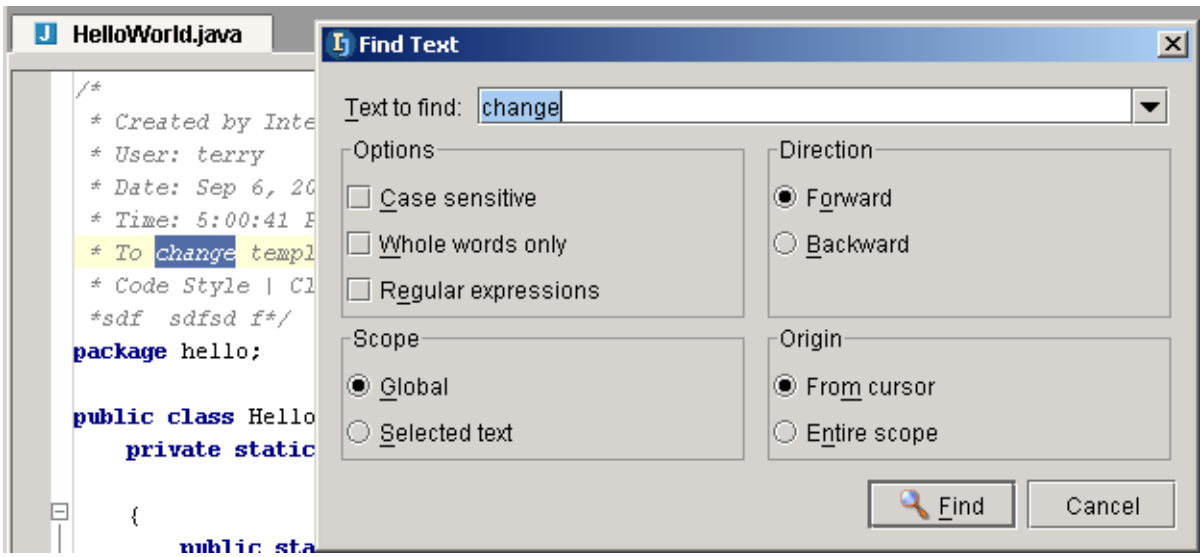


Figure 7.37. Dialog “Find text” (677)

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

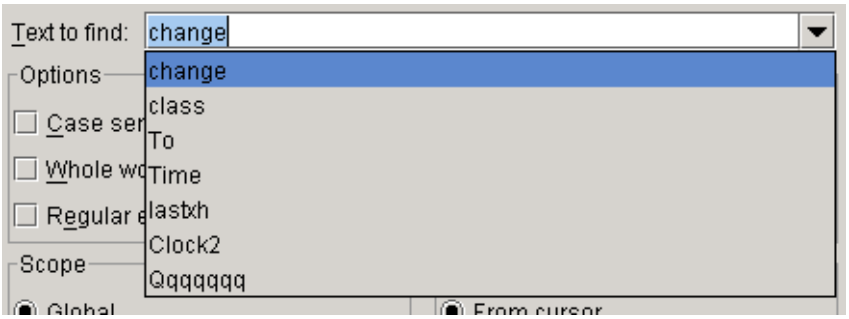


Figure 7.38. Drop-down list “Text to find” (678)

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

7.2.2.2. Highlight usages in file

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

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

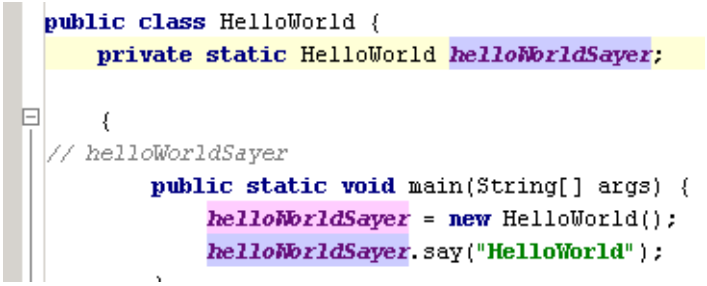


Figure 7.39. Highlight usages (687)

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

7.2.2.3. 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).

7.2.2.4. Find Next

7.19. Select **Search | Find next** to find the next occurence.

7.2.2.5. Find Previous

7.20. Select **Search | Find previous** to find the previous occurence.

7.2.2.6. Find in path

7.21. Select a word.

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

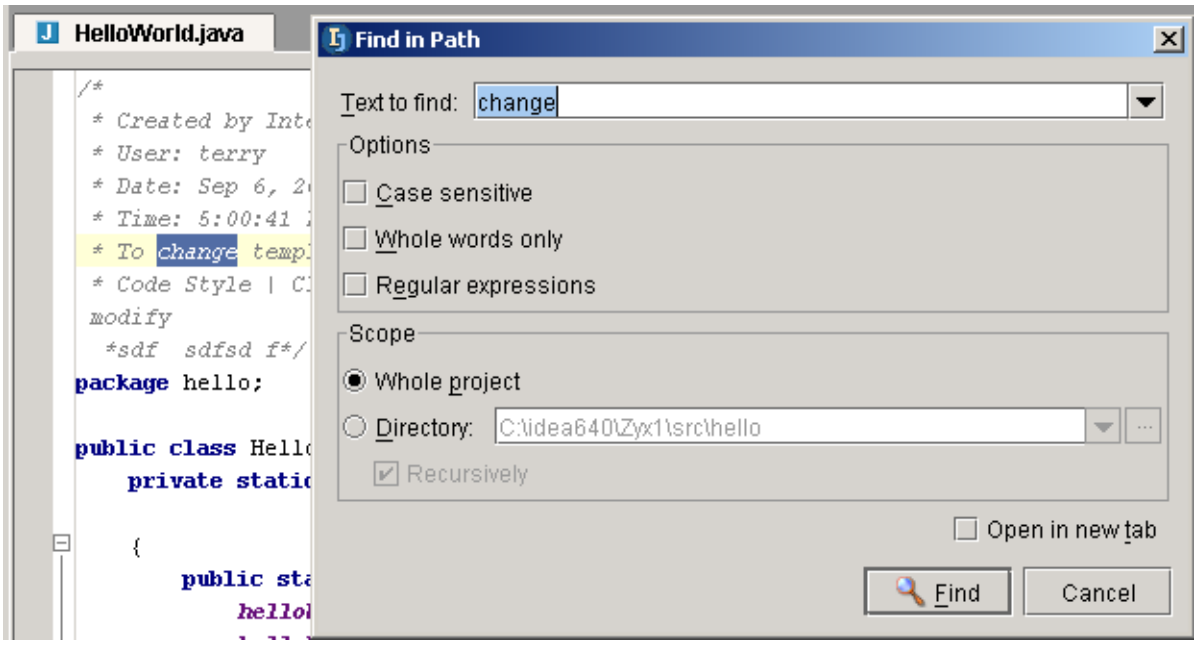


Figure 7.40. Dialog “Find in path” (681)

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

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

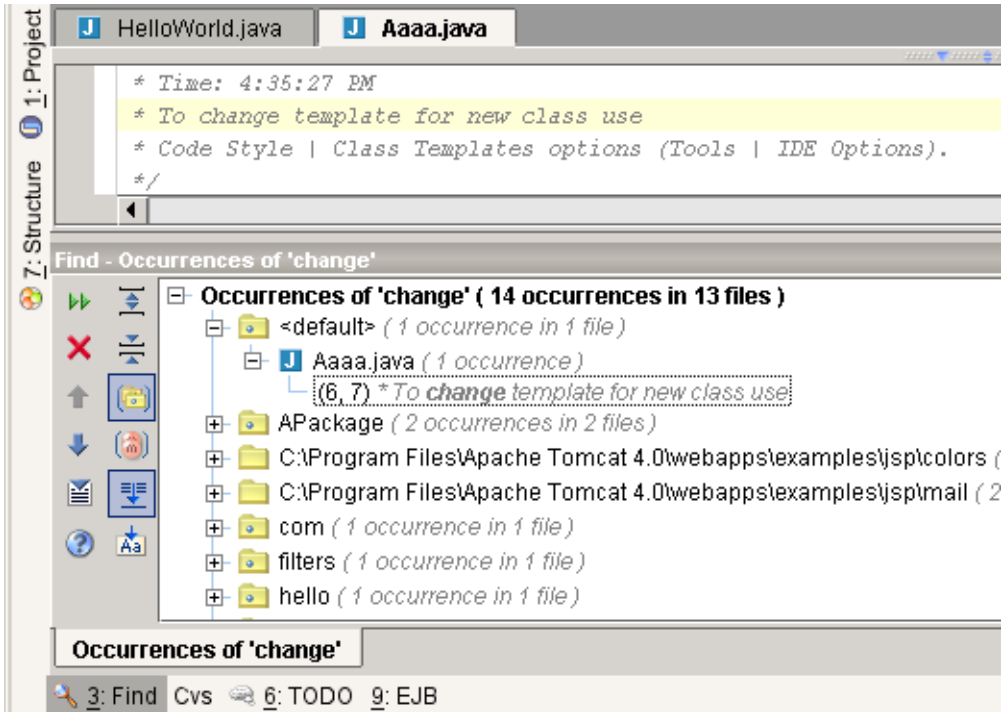


Figure 7.41. Tool “Find” (682)

7.2.2.7. Find usages (in path)

7.25. Place the cursor within a code element (do not select).
7.26. Select **Search | Find usages**. The “Find usages” dialog appears.

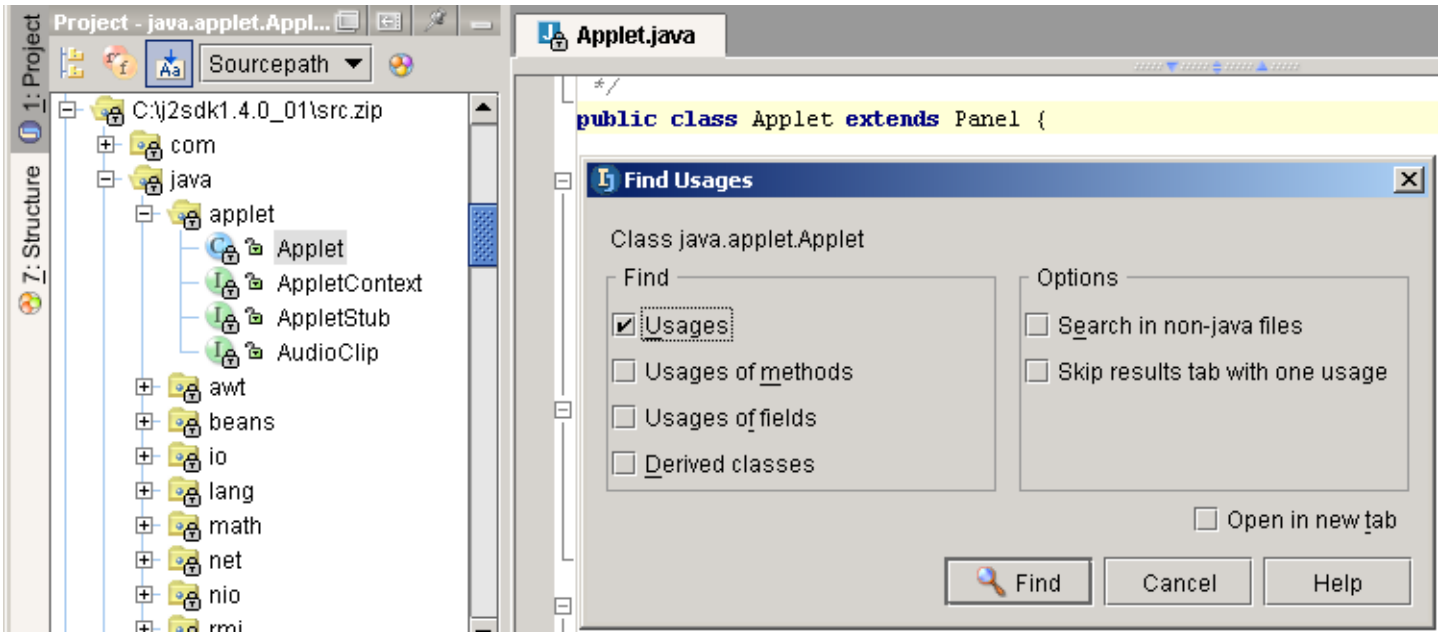


Figure 7.42. Dialog “Find usages” (688)

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

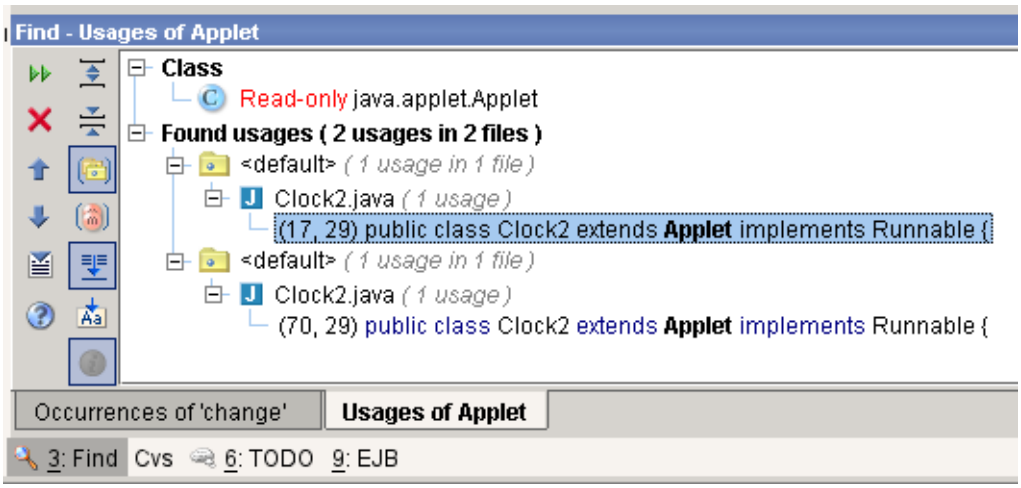


Figure 7.43. Find tool results (689)

7.2.2.8. Replace (in file)

7.28. Open a file.
7.29. Select a word.
7.30. Select **Search | Replace**. The dialog “Replace text” appears.
7.31. For “Replace with:”: Enter the new text.

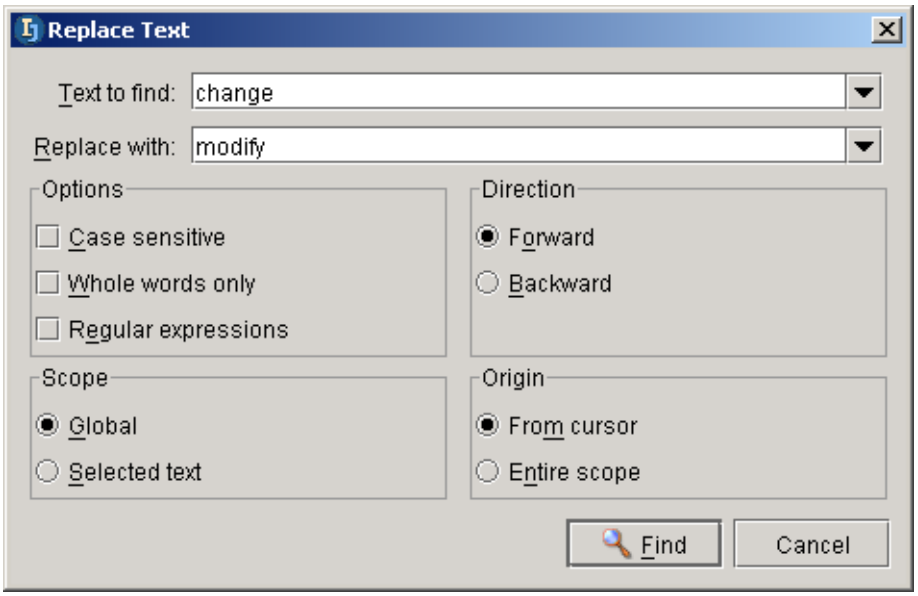


Figure 7.44. Dialog “Replace text” (679)

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

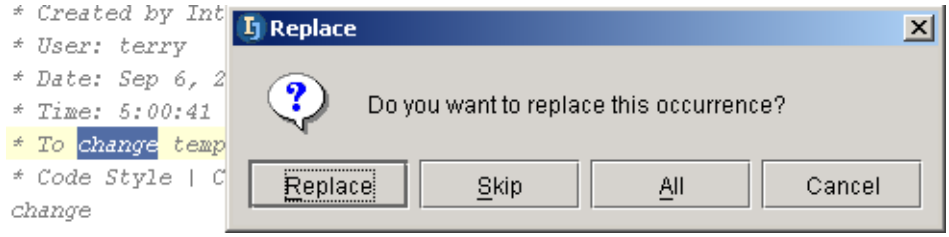


Figure 7.45. Dialog “Replace text” (680)

7.33. Select

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

7.2.2.9. Replace in path

7.34. Open a file.

7.35. Select a word.

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

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

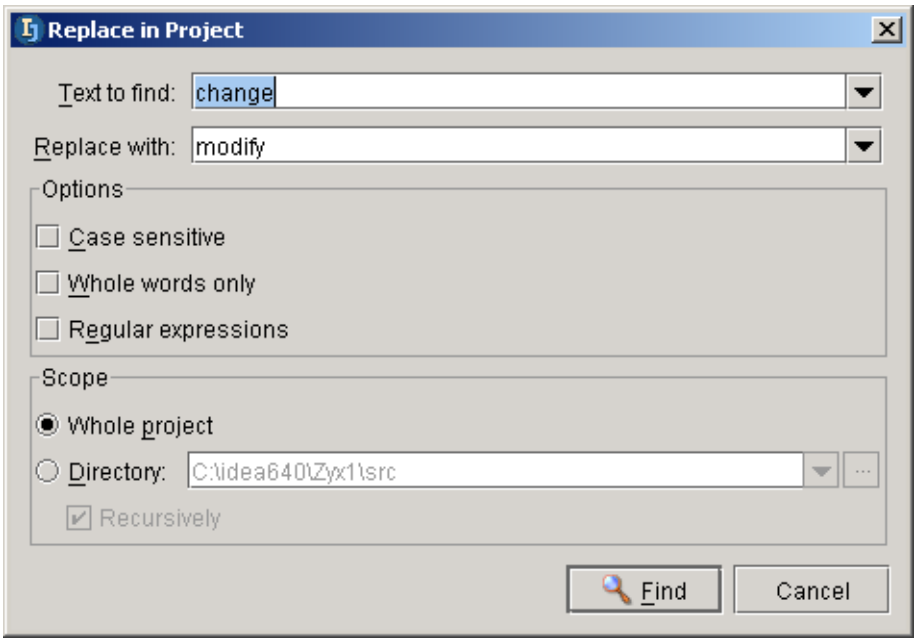


Figure 7.46. Dialog “Replace in project” (683)

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

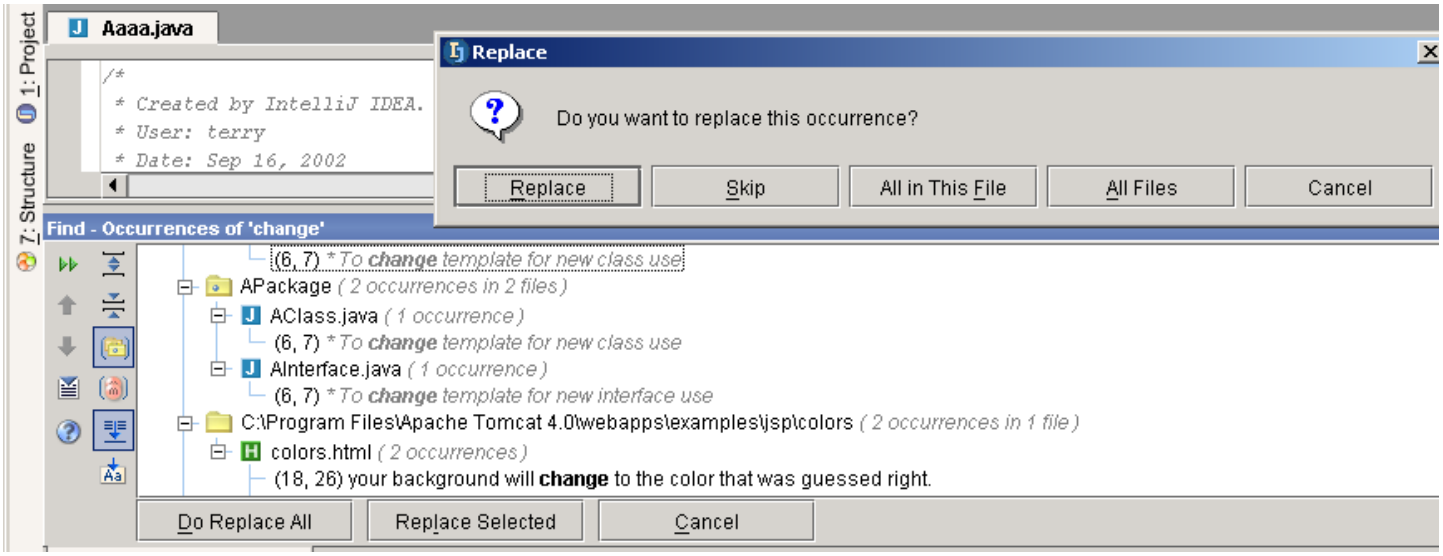


Figure 7.47. Dialog “Replace” (684)

7.39. Select

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

7.2.2.10. Find Word at caret

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

7.41. Select **Search | Find word at caret**. The next occurence of the word is highlighted.

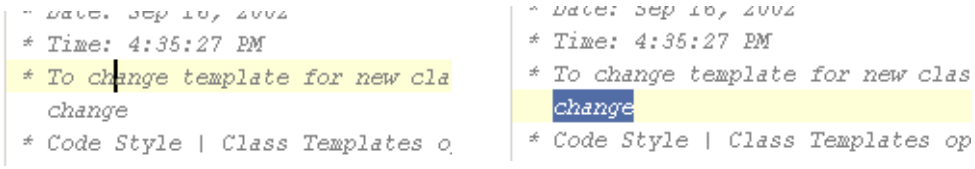


Figure 7.48. Find word at caret (685,686)

7.2.2.11. Incremental search ?? XXX

7.2.3. Go to

IDEA provides the following “go to” functions:

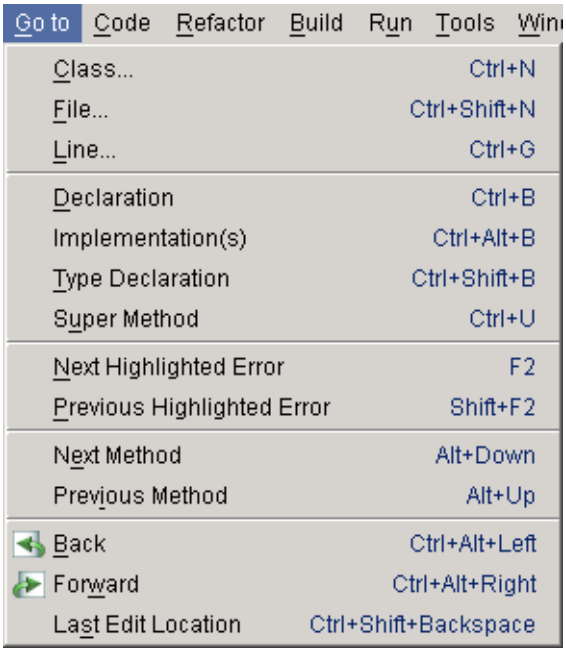


Figure 7.49. Goto menu (690)

7.2.3.1. Class

7.42. Select **Go to | Class**. The dialog “Enter class name” appears.
7.43. Enter the class name. Note that IDEA will suggest names based on the letters you type in.

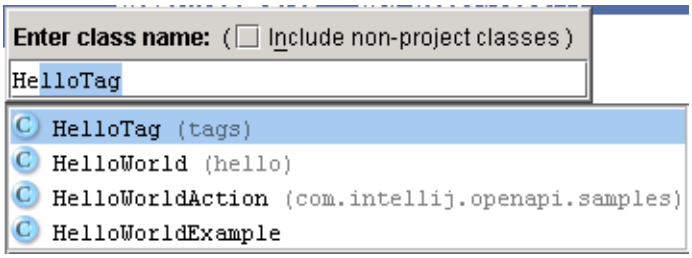


Figure 7.50. Dialog “Enter class name” (691)

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

7.2.3.2. File

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

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

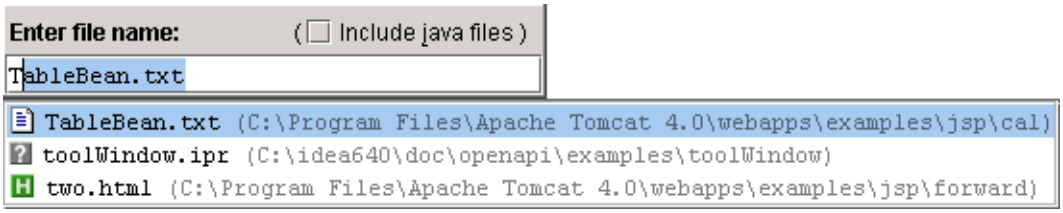


Figure 7.51. Dialog “Enter file name” (692)

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

7.2.3.3. Line

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

7.2.3.4. Declaration

7.48. Place the cursor in a code element.

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

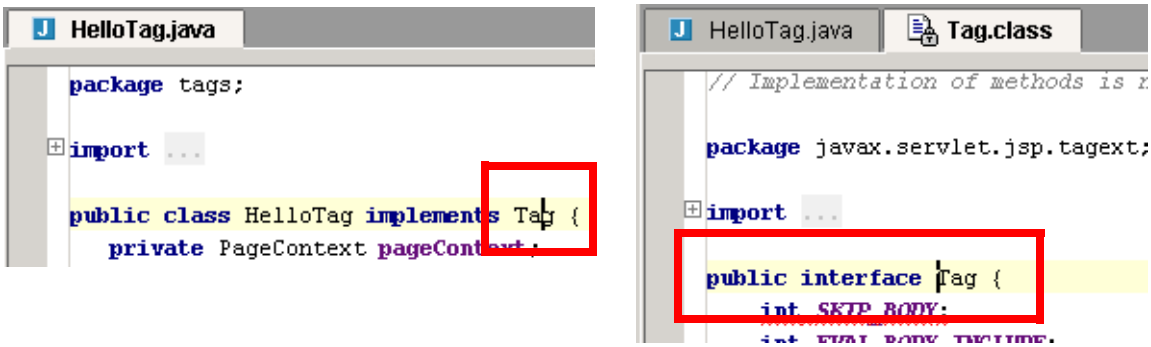


Figure 7.52. Go to declaration (693,694)

7.2.3.5. Implementation

7.50. Open an Interface file.

7.51. Place the cursor on the interface name.

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

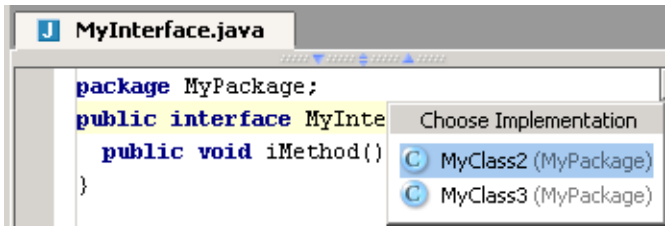


Figure 7.53. Go to implementation (194)

7.53. Select an implementor to open.

7.2.3.6. Type declaration

7.54. Place the cursor in a code element.

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

Figure 7.54. Go to type declaration (695,696)

7.2.3.7. Super method

7.56. Place the cursor in a code element.
7.57. Select **Go to | Super method**. The source file for the method of the super class is opened.

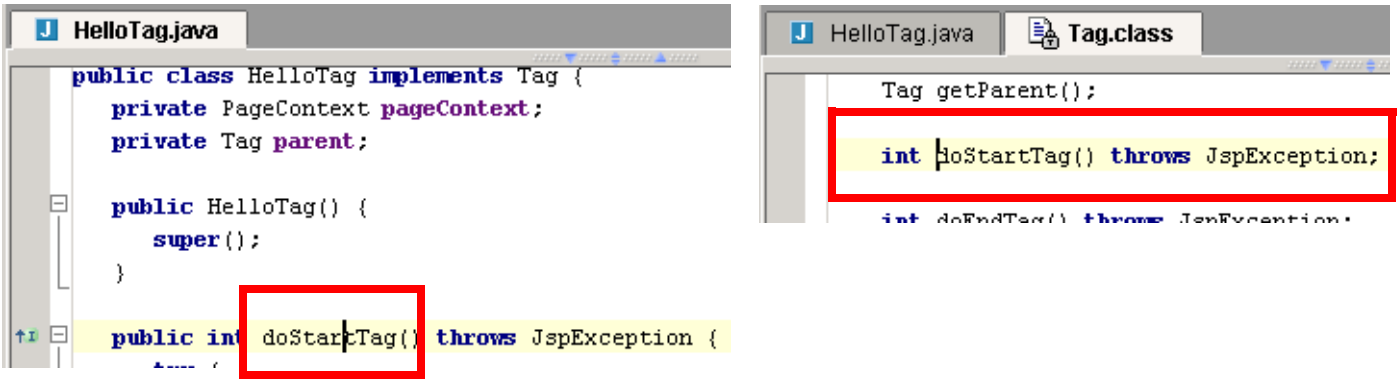


Figure 7.55. Go to super method (698,697)

7.2.3.8. 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.

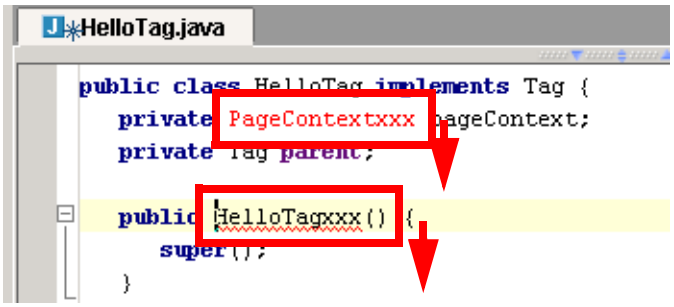


Figure 7.56. Next highlighted error (699)

7.2.3.9. 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.

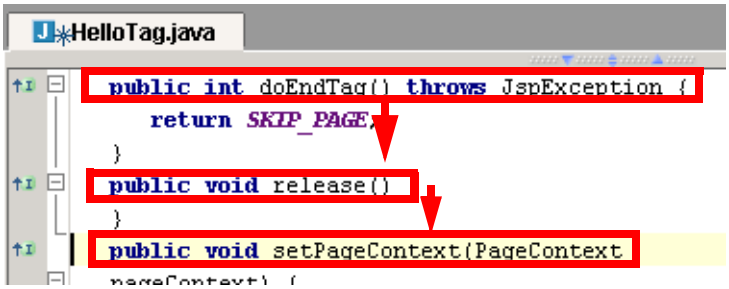


Figure 7.57. Next method (700)

7.2.3.10. Back / Forward

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

7.58. Create 3 classes: **Class1**, **Class2**, **Class3**.
7.59. Open **Class1**.
7.60. Open **Class2**.
7.61. Open **Class3**.
7.62. Select **Go to | Back**. The Class2 editor is selected.
7.63. Close **Class1**.
7.64. Close **Class3**.
7.65. Select **Go to | Forward**. Class3 is opened in an editor.
7.66. Select **Go to | Back**. The Class2 editor is selected.
7.67. Select **Go to | Back**. Class1 is opened in an editor.

7.2.3.11. Last Edit location

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

7.3. Colors and fonts X

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

7.3.1. General X

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

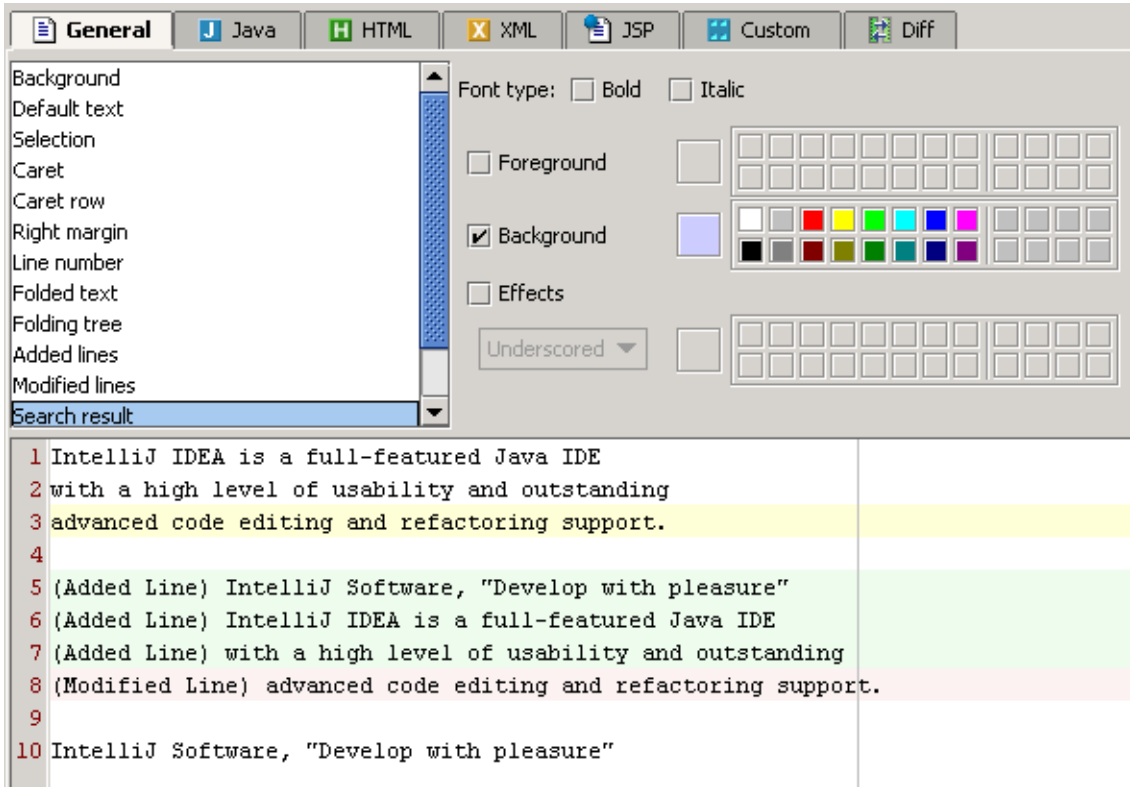


Figure 7.58. General colors/fonts (438)

7.3.2. Java X

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

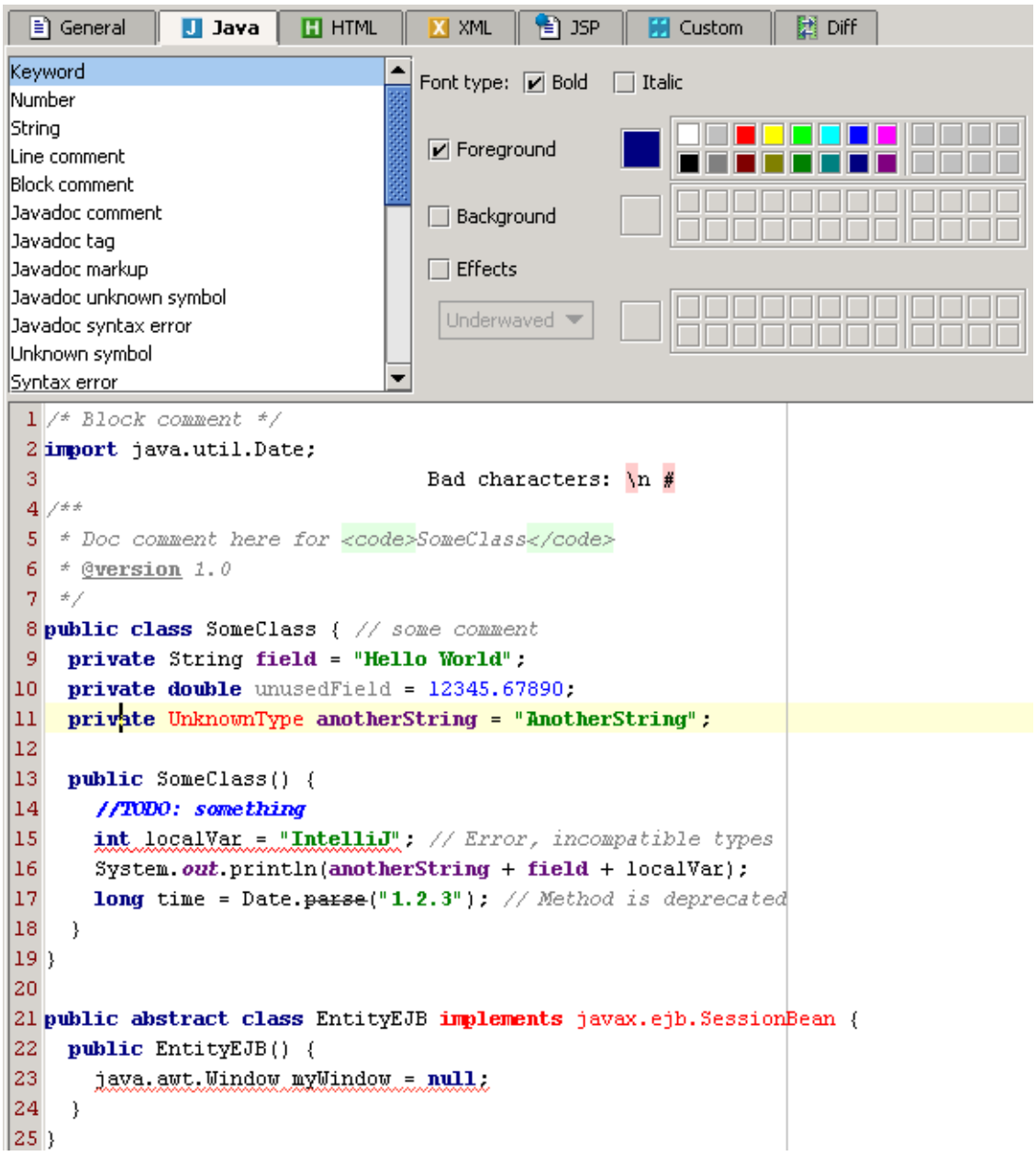


Figure 7.59. Colors/fonts for Java files (437)

7.3.3. HTML X

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

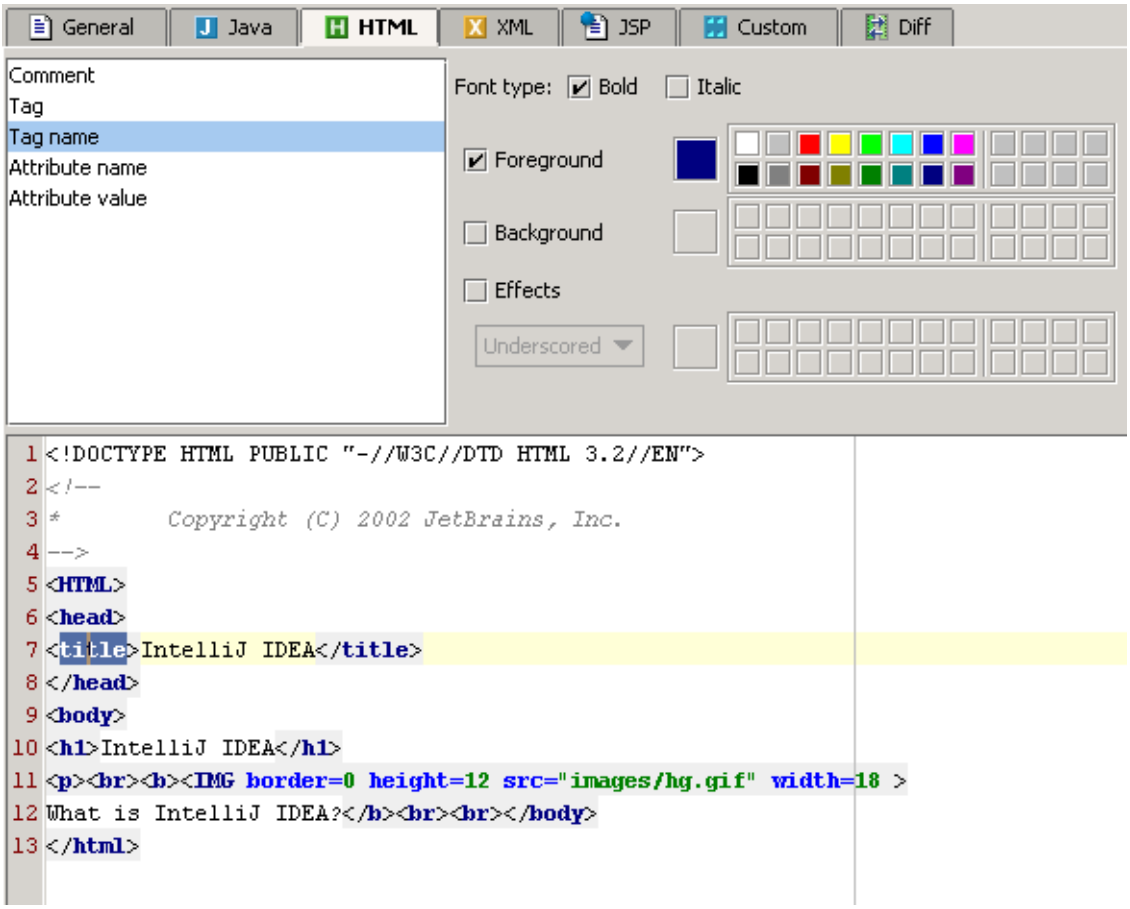


Figure 7.60. Colors/fonts for HTML files (436)

7.3.4. XML X

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

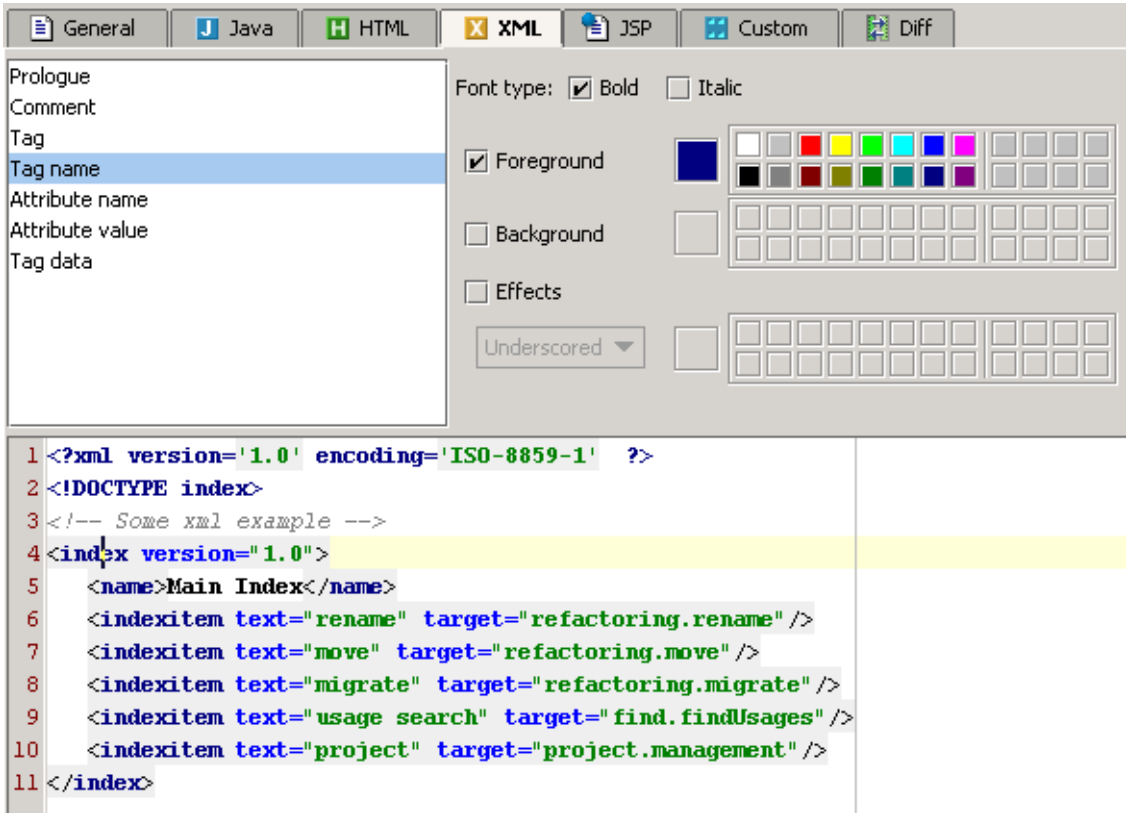


Figure 7.61. Colors/fonts for XML files (435)

7.3.5. JSP X

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

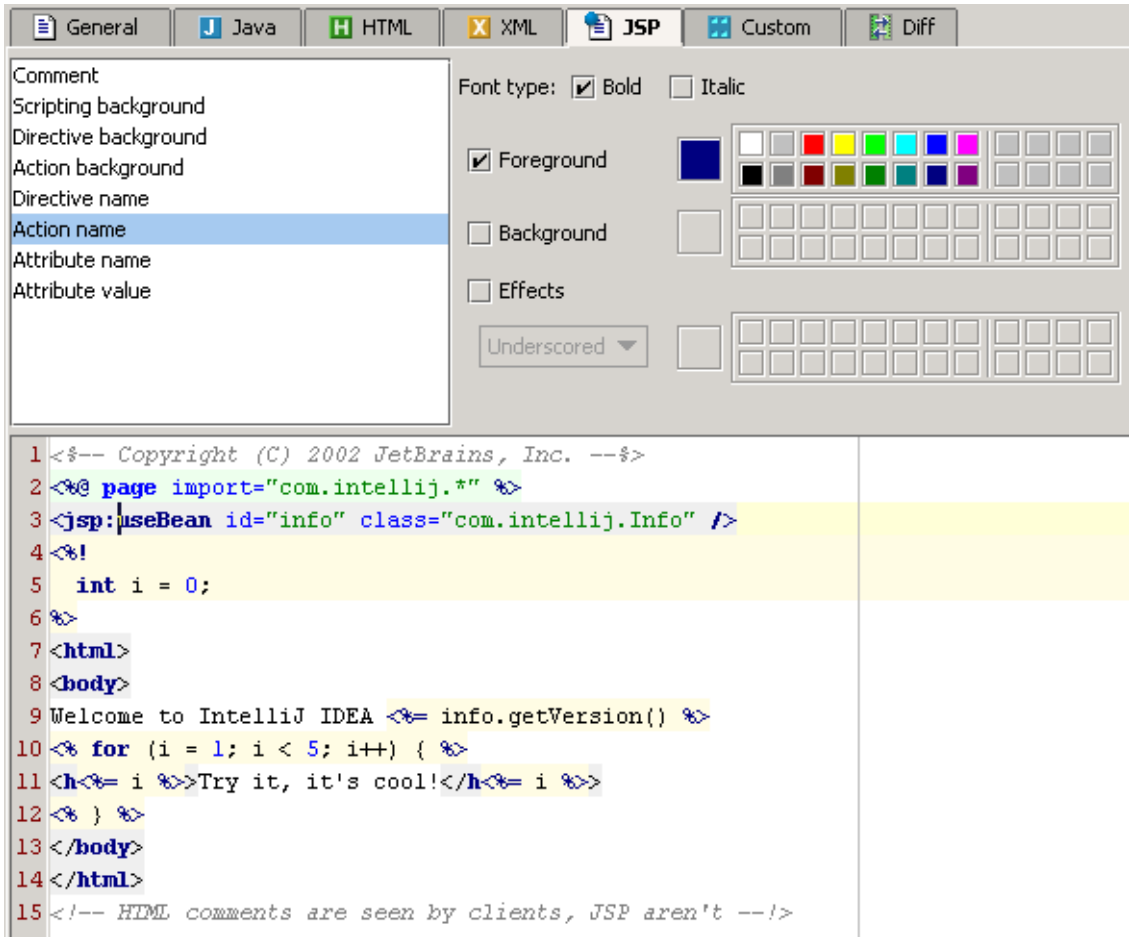


Figure 7.62. Colors/fonts for JSP files (434)

7.3.6. Custom X

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

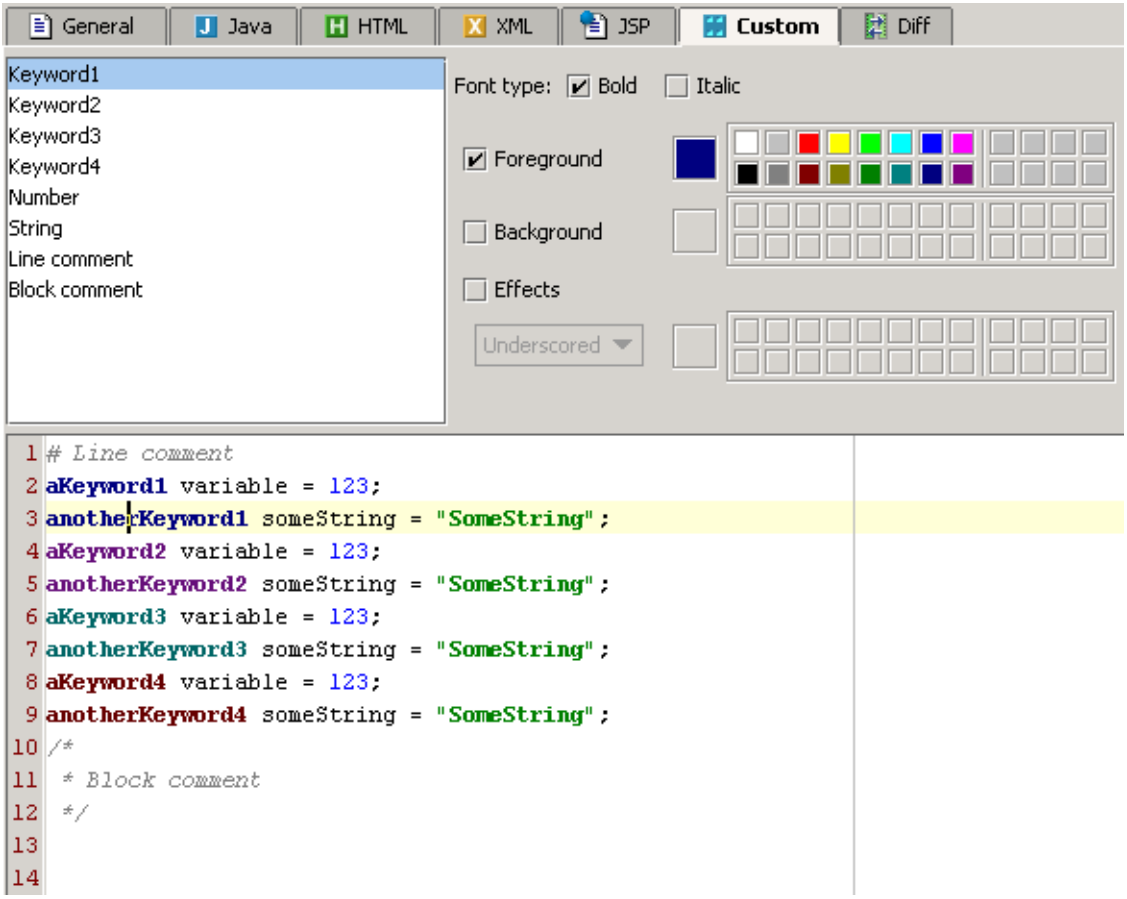


Figure 7.63. Custom colors/fonts (433)

7.3.7. Color Schemes

To use a custom color scheme, do the following:

7.3.7.1. Create

7.68. Click **Save as...**. The dialog “Save color scheme as” appears.
7.69. For “Name” enter **MyColorScheme**.

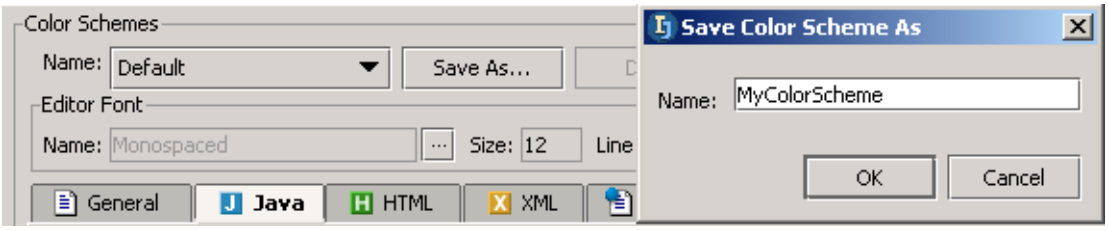


Figure 7.64. Custom color scheme name (789)

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

7.3.7.2. Select

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

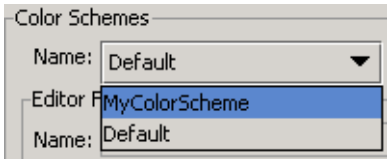


Figure 7.65. Selecting a color scheme (790)

7.3.7.3. Modify

7.72. In tab “General”: Click on **line 1**. Note that the “Default text” is selected.
7.73. Check the checkbox **Background**.
7.74. For the background color select grey.

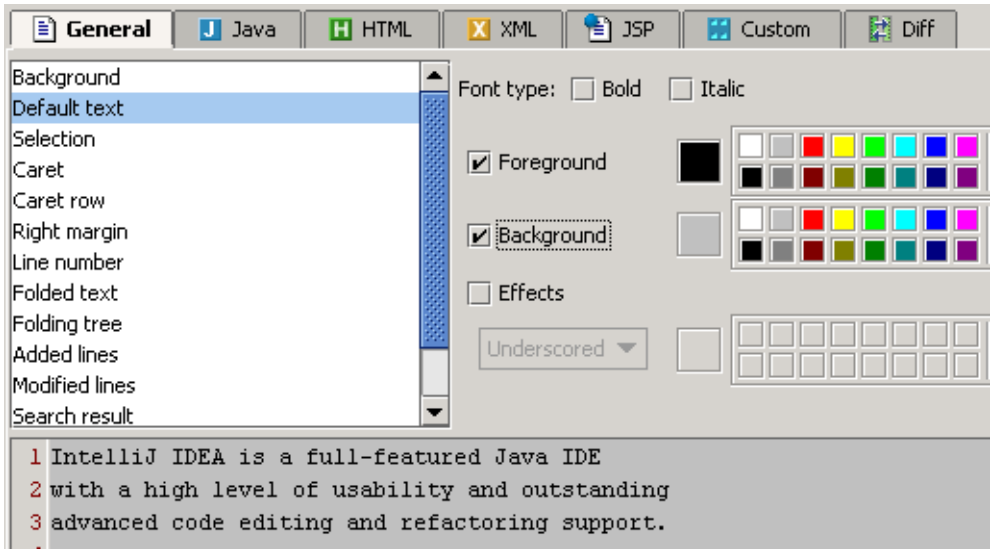


Figure 7.66. Selecting a background color (791)

7.3.7.4. Apply

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

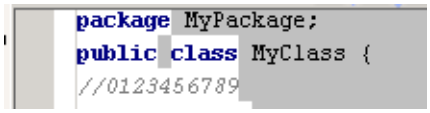


Figure 7.67. New background color (792)

7.4. Code style X

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

7.4.1. General X

Tab **General** shows general code styles.

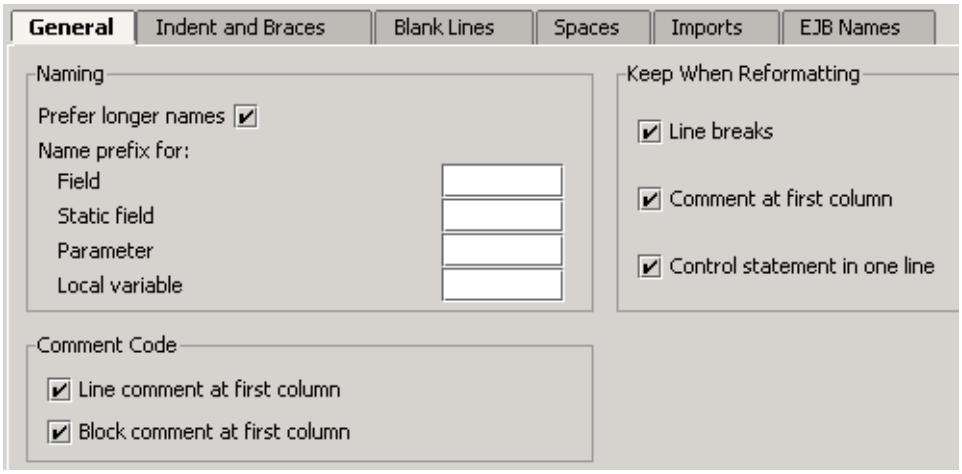


Figure 7.68. General code style (431)

7.4.2. Indent and Braces X

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

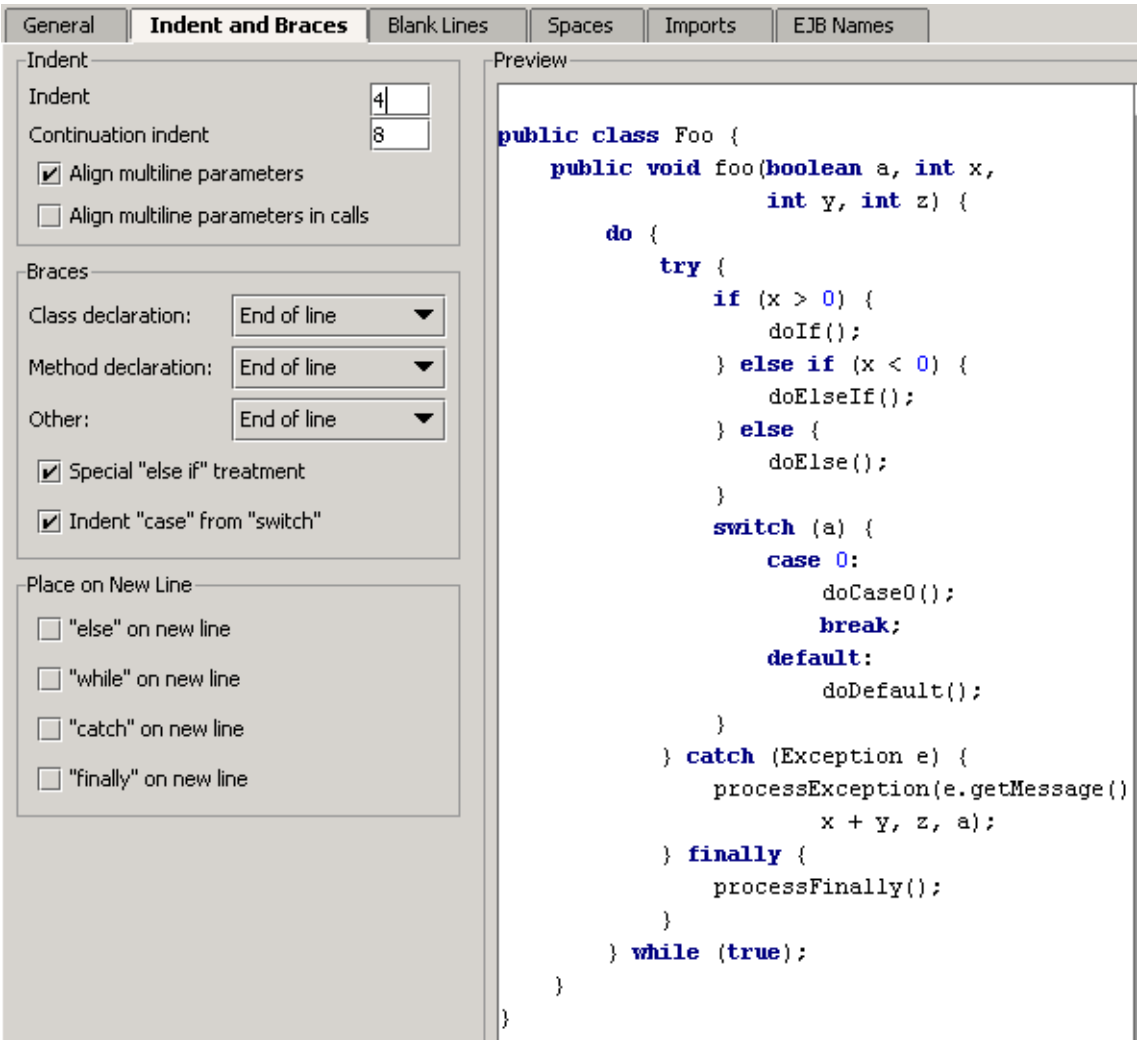


Figure 7.69. Code style / indents and braces (430)

7.4.3. Blank lines X

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

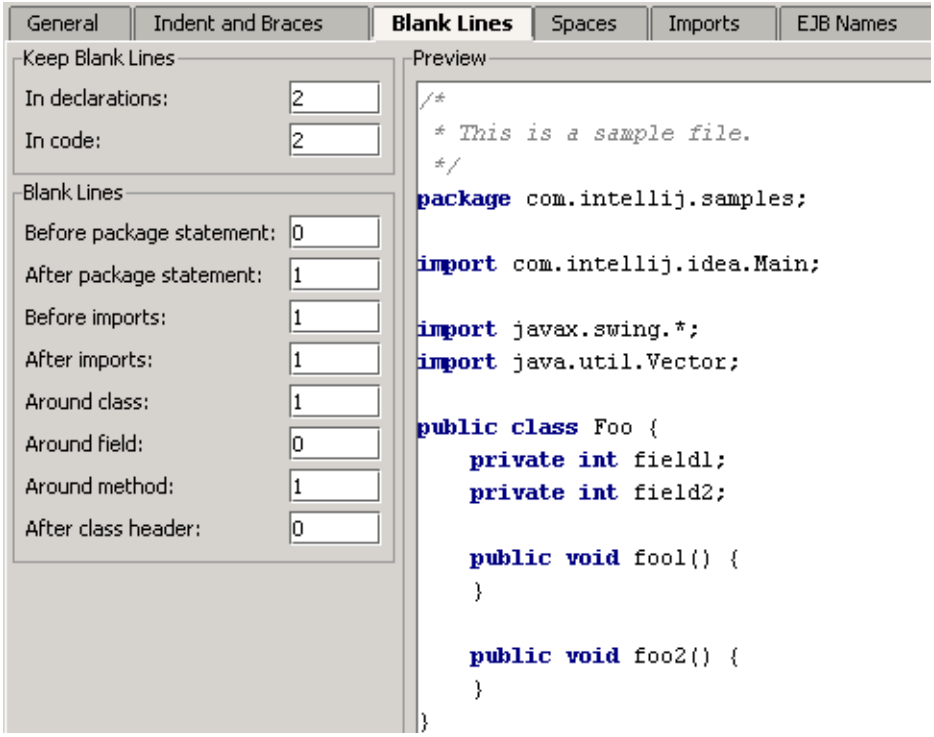


Figure 7.70. Code style / blank lines (429)

7.4.4. Spaces X

Tab **Spaces** shows how spaces are added.

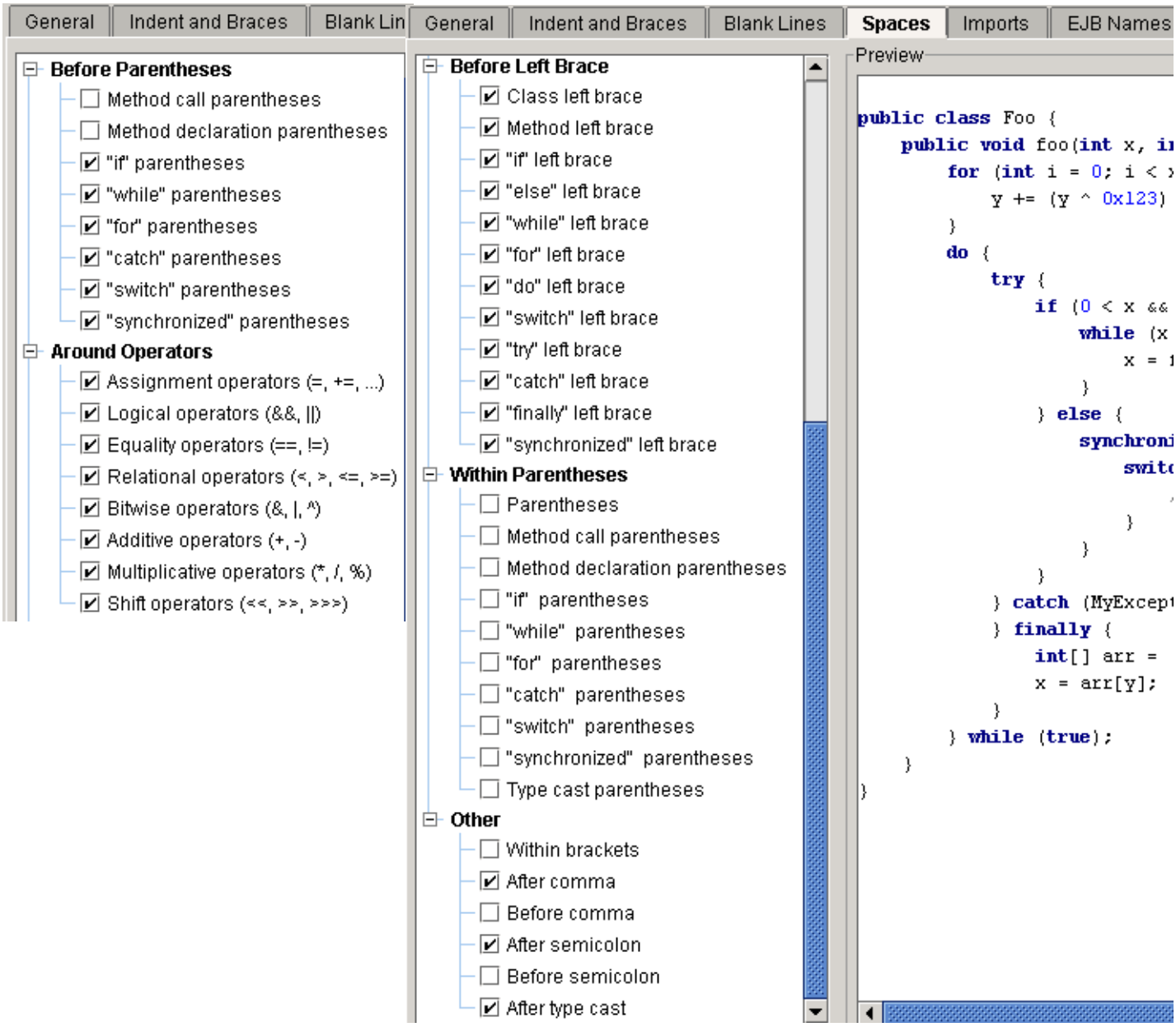


Figure 7.71. Code style / spaces (428,427)

7.4.5. Imports X

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

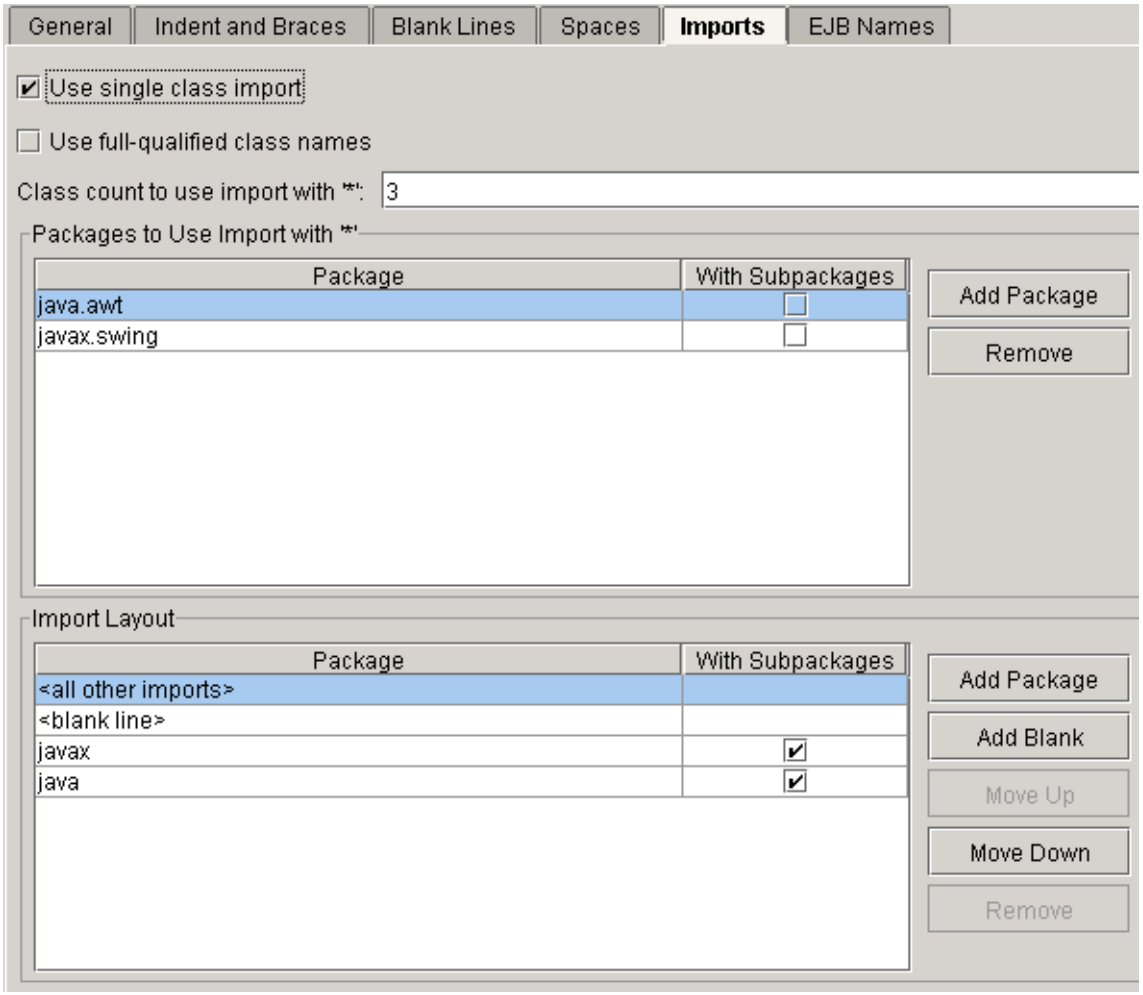


Figure 7.72. Code style / imports (426)

7.4.6. 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.

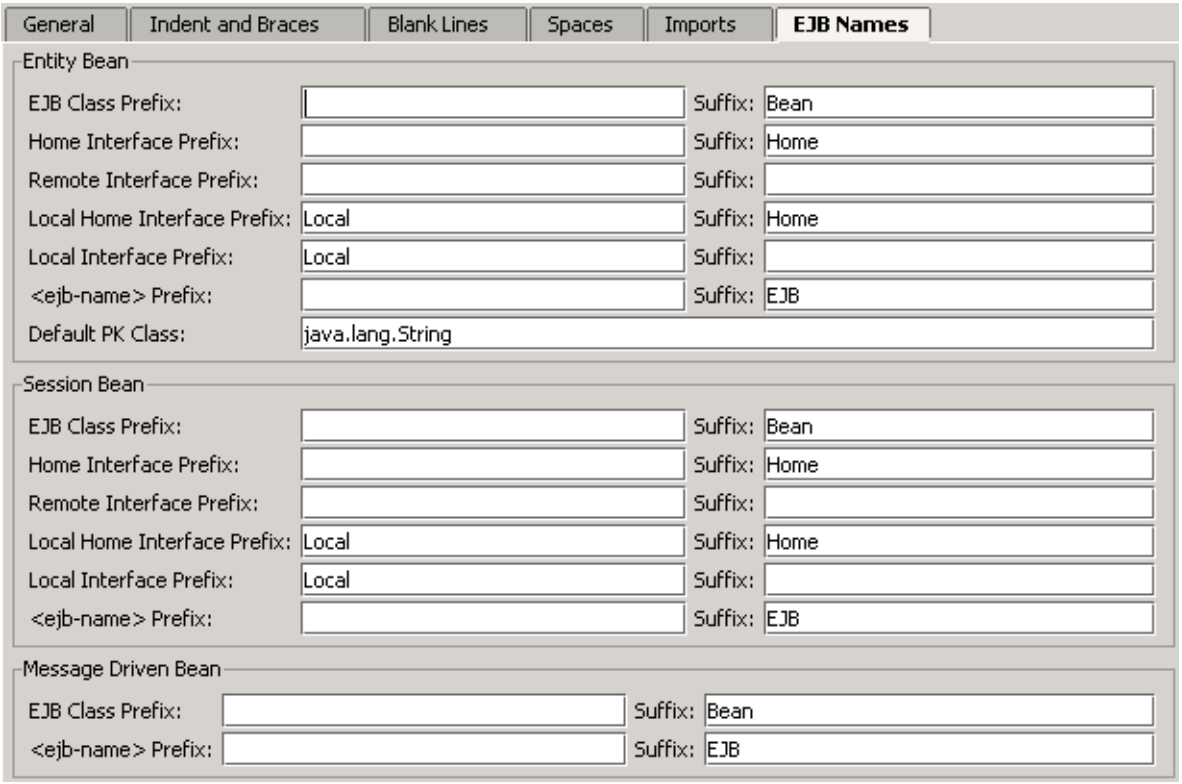


Figure 7.73. Code style / EJB names (793)

7.4.7. Code style schemes

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

7.4.7.1. Create

7.76. Click **Save as...**. The dialog “Save code style scheme as” appears.
7.77. For “Name” enter **MyCodeStyleScheme**.

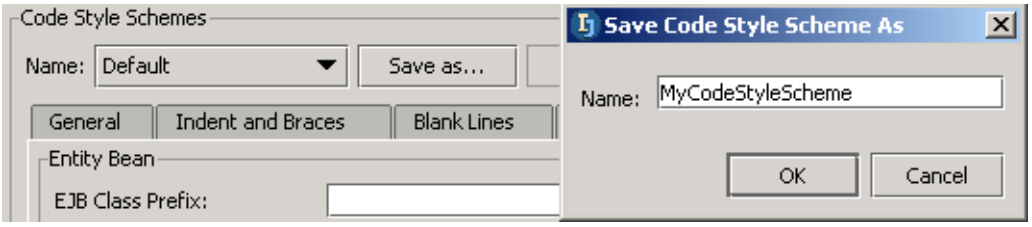


Figure 7.74. Custom code style scheme name (794)

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

7.4.7.2. Select

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

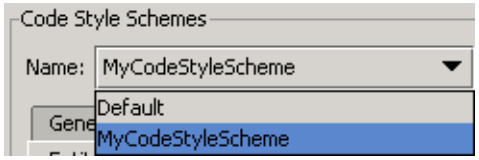


Figure 7.75. Selecting a code style scheme (795)

7.4.7.3. Modify

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

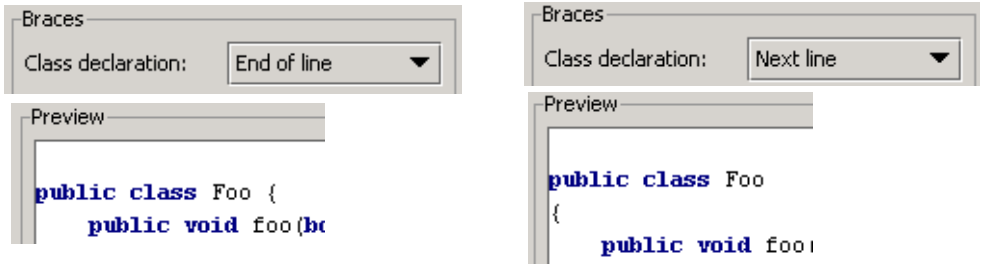


Figure 7.76. Class declaration indents (796,797,798,799)

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

7.4.7.4. Apply

7.4.7.4.1. To existing class

7.82. Open **MyClass.java**.
7.83. Select **Tools | Reformat code...**. The dialog “Reformat code” appears.
7.84. Select **File**.

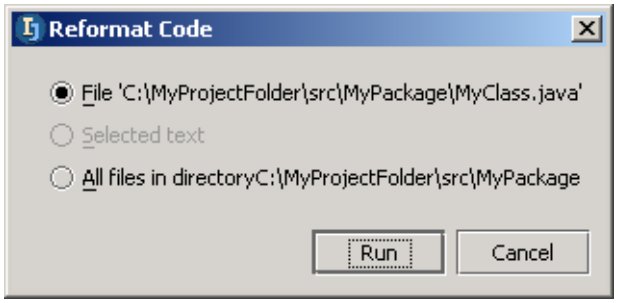


Figure 7.77. Dialog “Reformat code” (801)

7.85. Click **Run**. The file is reformatted.

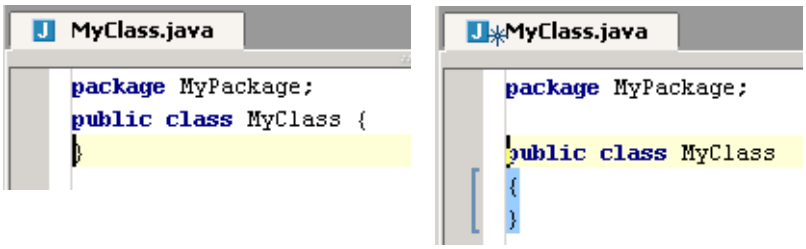


Figure 7.78. Reformatted source file (800,802)

7.4.7.4.2. To class template

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

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

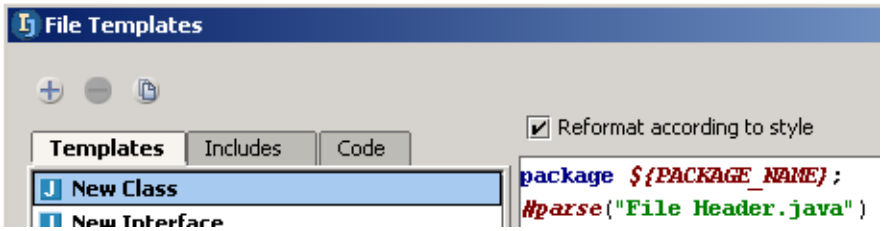


Figure 7.79. Reformat according to style checkbox (803)

7.88. Click **OK**.

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

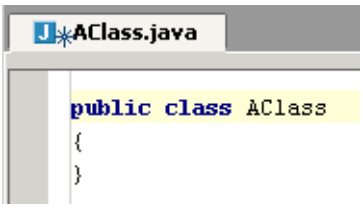


Figure 7.80. New class with code style (804)

7.4.8. Autoindent

7.90. Create class **Class1**:

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

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

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

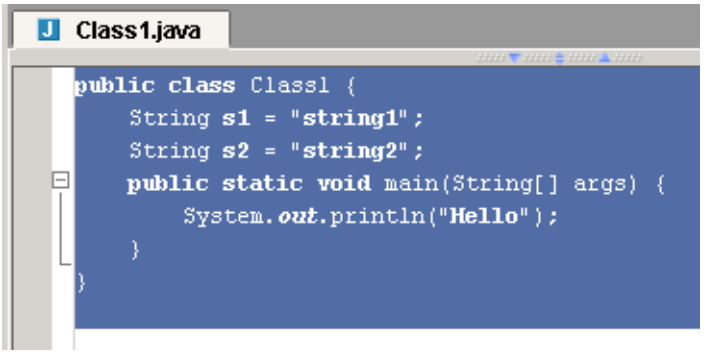


Figure 7.81. Autoindented code (346)

7.4.9. Reformat

IDEA can reformat

7.4.9.1. File

7.93. Create class **Class1**:

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

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

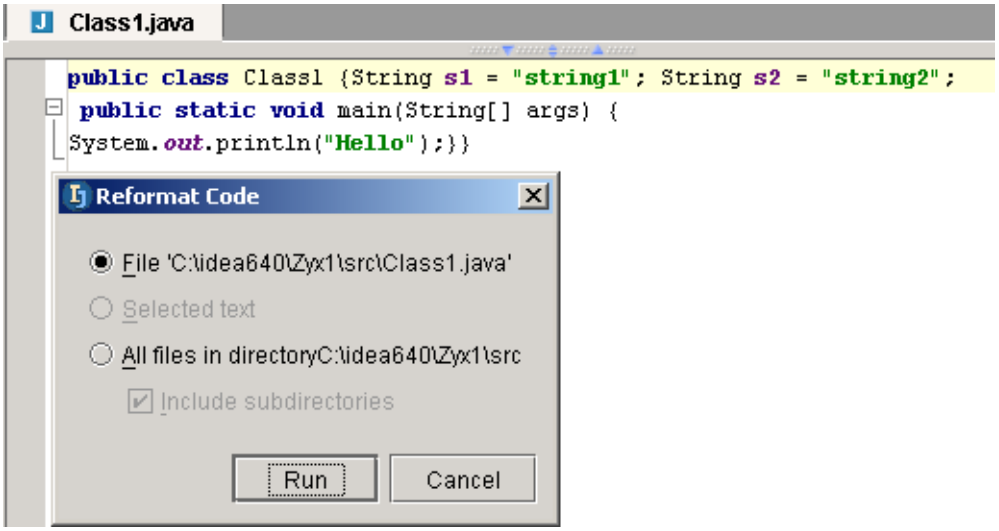


Figure 7.82. Dialog “Reformat code” (345)

7.95. Select **Run**.

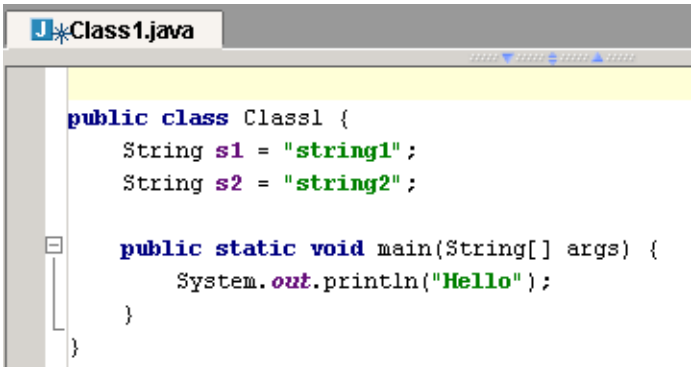


Figure 7.83. Reformatted code (344)

7.4.9.2. Selected text

7.96. Undo the reformatting for **Class1**:
7.97. 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");}}
```

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

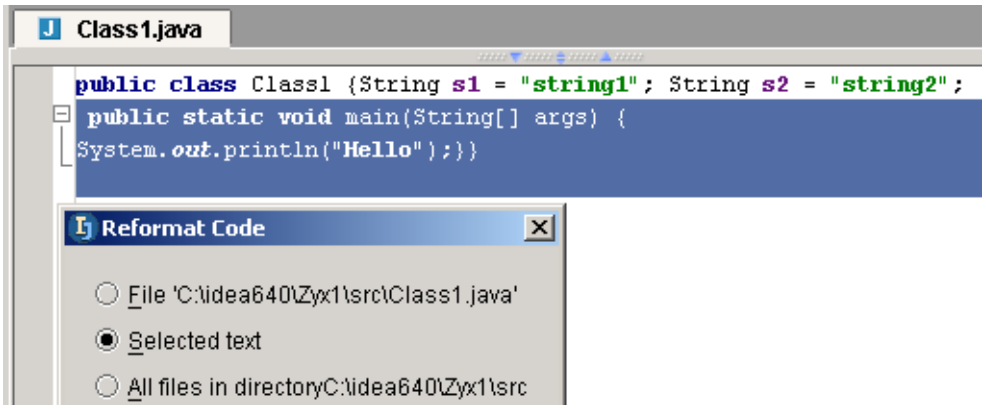


Figure 7.84. Dialog “Reformat code” (selected text) (343)

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

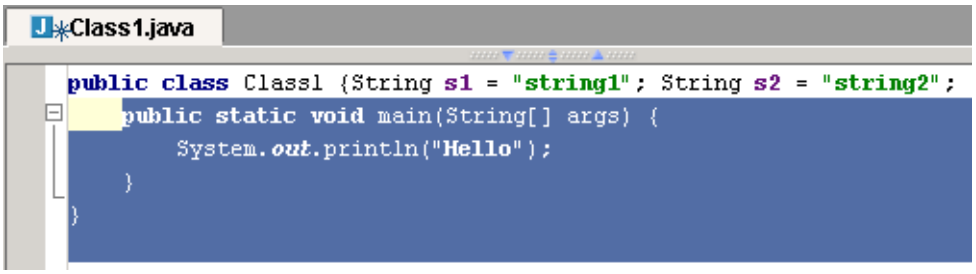


Figure 7.85. Reformatted code (342)

7.4.9.3. Files in directory

7.100. Select a directory.
7.101. Select **Tools | Reformat code...**. The dialog “Reformat code” appears.

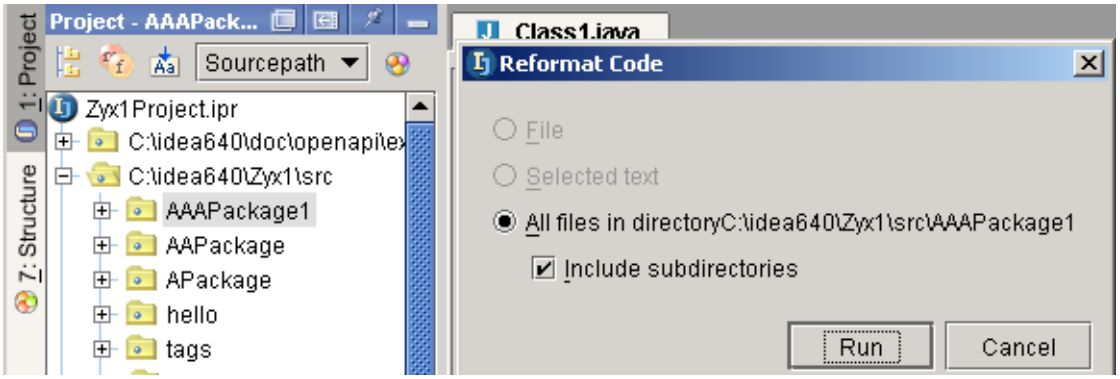


Figure 7.86. Dialog “Reformat code” (selected directory) (341)

If you click **Run**, then the files in the chosen directory and all subdirectories will be reformatted.

7.5. Error indication X

The IDEA editor checks for the following types of errors while editting:

7.5.1. Deprecated symbol

7.102. Change **MyClass** as shown:

```
package MyPackage;
public class MyClass {
    void aMethod() {
        System.out.println(java.awt.Frame.CROSSHAIR_CURSOR);
    };
}
```

An deprecation warning is generated.

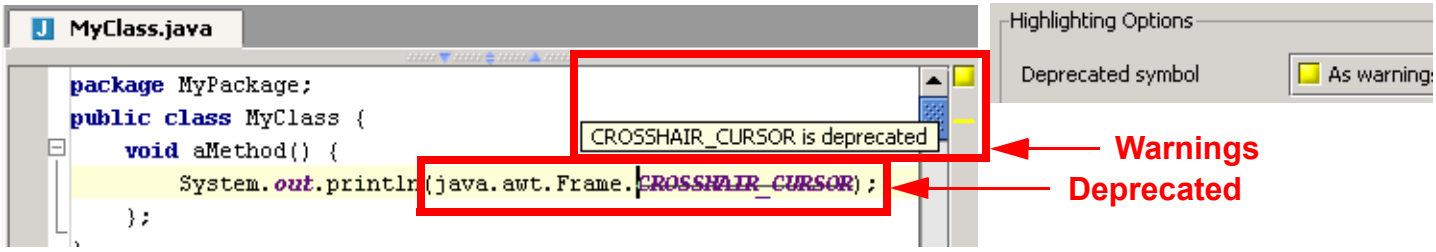


Figure 7.87. Deprecation (720,722)

7.5.2. Reparse delay

7.103. In IDE Settings - Errors: Set **Autoreparse delay** to **5000** (5 seconds).
7.104. In an editor: Make an error (in the example below, “package” was changed to “pacxxxkage”).
7.105. Move the cursor to a different line. Note that the error is marked (ie, the file is reparsed) after 5 seconds.

Figure 7.88. Reparse delay (714,716)

7.106. In IDE Settings - Errors: Set **Autoreparse delay** to **300**.
7.107. Click **Apply**.

7.5.3. Unused import

7.108. Change **MyClass** as shown:

```
import java.beans.*;
package MyPackage;
public class MyClass {
    void aMethod() {};
}
```

An “Unused import statement” warning is generated.

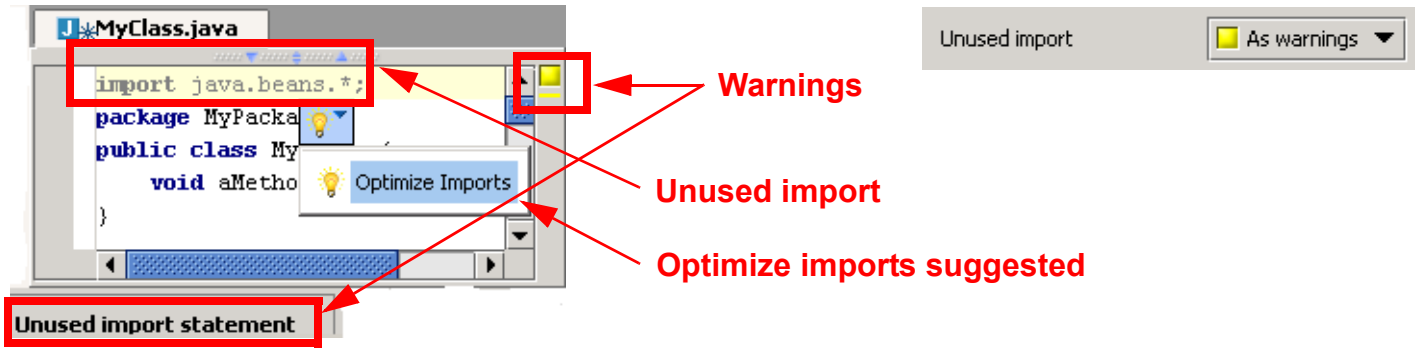


Figure 7.89. Unused import (715,717)

7.5.4. Unused symbol

7.109. Change **MyClass** as shown:

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

An “Unused symbol” warning is generated.

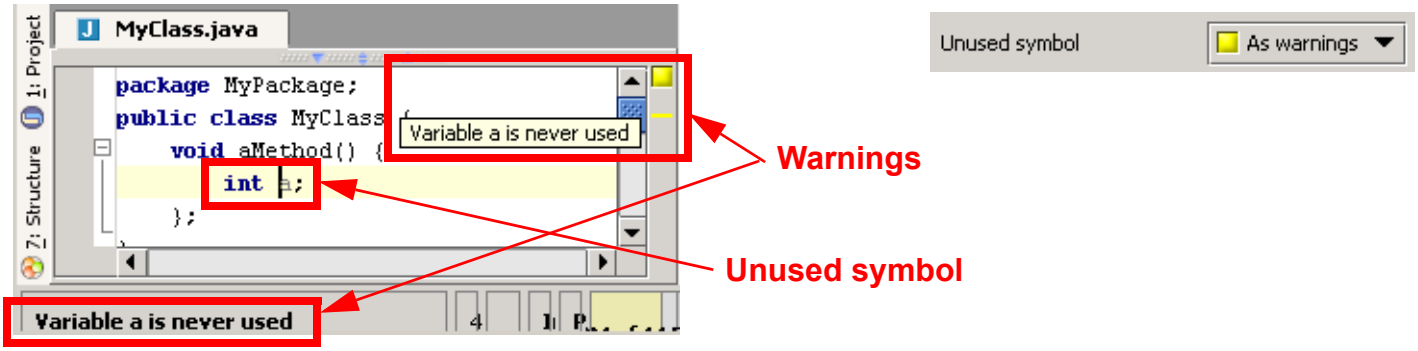


Figure 7.90. Unused symbol (718,719)

7.5.5. Unused throws declaration XXX

7.5.6. Redundant type cast

7.110. 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.

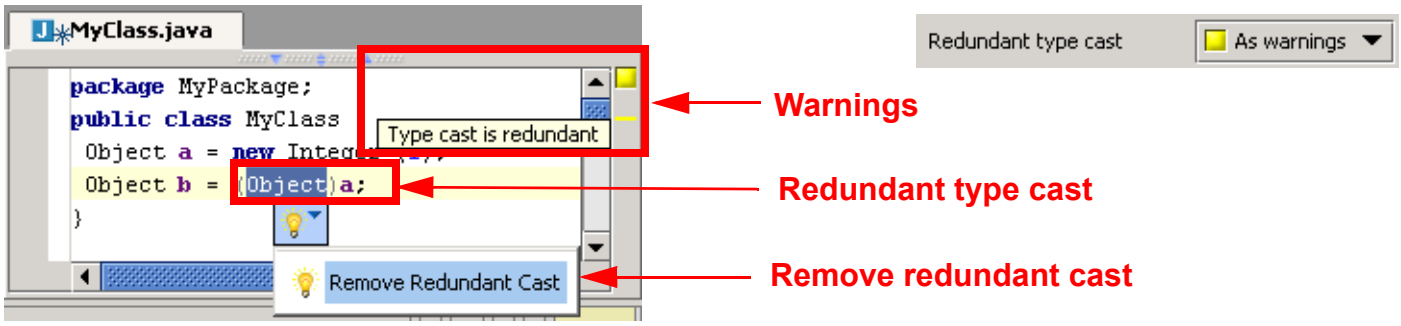


Figure 7.91. Redundant type cast (721,723)

7.5.7. Silly assignment XXX

7.5.8. Wrong package statement

7.111. Change **MyClass** as shown:

```
package MyPackage2;
public class MyClass {
}
```

An wrong package warning is generated.

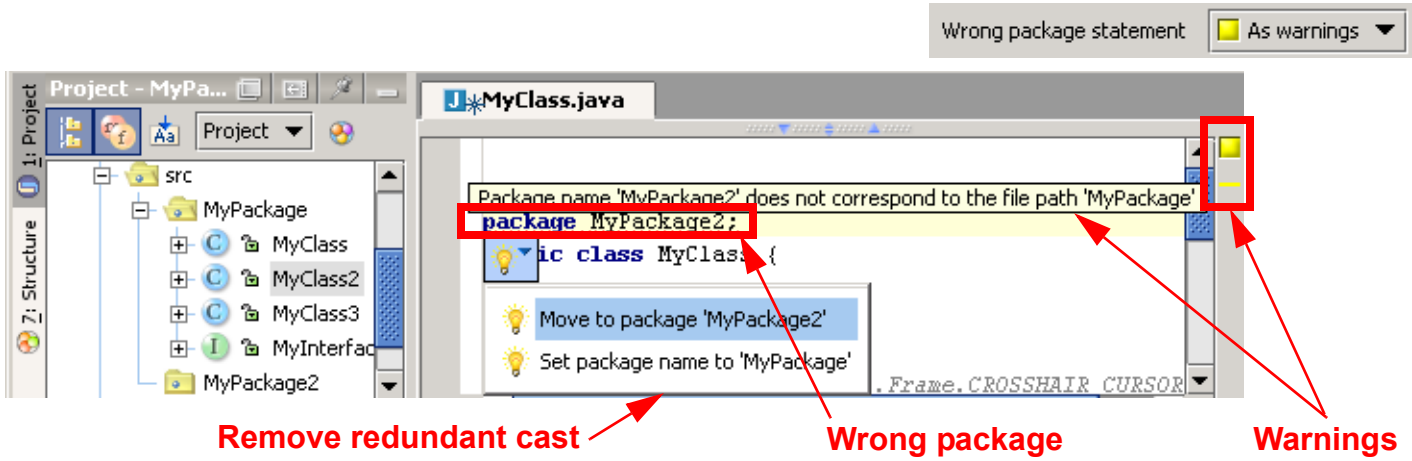


Figure 7.92. Wrong package statement (724,725)

7.5.9. Javadocs errors XXX

7.5.10. Unknown Javadoc tags

7.112. Change **MyClass** as shown:

```
package MyPackage;
public class MyClass {
/**
 * @param p1
```

```
 * @paxxxxram p2
 * @return
 */
  public int foo(int p1, int p2)
   { return p1 + p2; }
}
```

An wrong javadoc tag is generated.

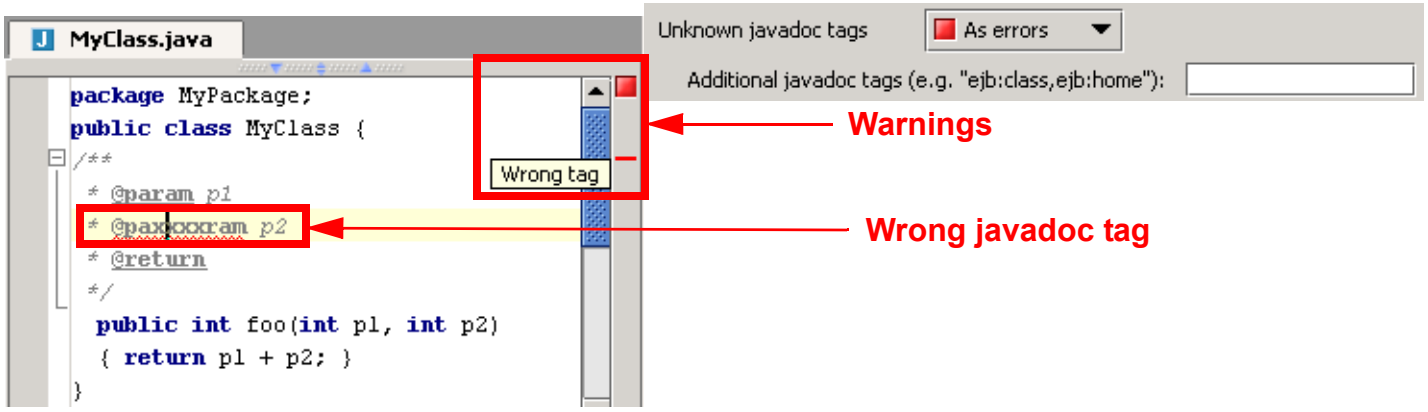


Figure 7.93. Wrong javadoc tag (726,727)

7.5.11. EJB errors XXX

Figure 7.94. EJB error (728)

7.5.12. EJB warnings XXX

Figure 7.95. EJB error (729)

7.6. Todo

The IDEA todo functionality makes it easy to make and find todo notes in a file or project.
IDEA todo includes:

7.6.1. Basics

7.113. In Class1 add the line

```
//@todo finish class1
```

7.114. In Class2 add the lines

```
//@todo finish class2
//@todo get ready for class3
```

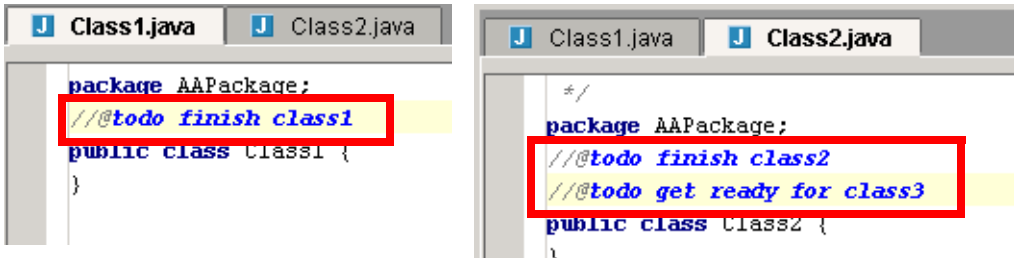


Figure 7.96. Added todo text (702,703)

7.115. Open the **TODO** tool.

7.116. Select the tab **Project**. The todos for the entire project are shown.

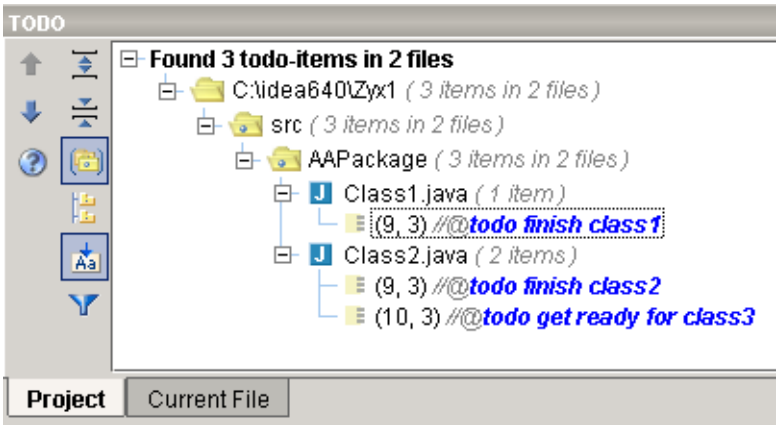


Figure 7.97. TODO items for the project (704)

7.117. Select tab **Current file**. The todos for the current file are shown.

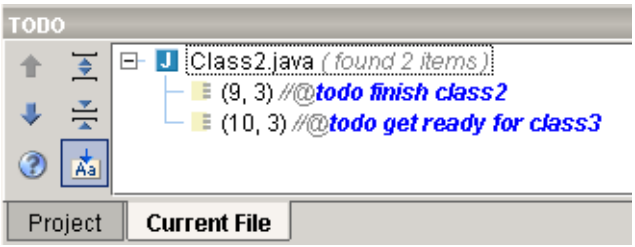


Figure 7.98. Todo items for a file (705)

7.118. 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.

7.6.2. Patterns

You can create custom patterns that are interpreted by IDEA as todo items.

- **Create pattern**
- **Add pattern to file**

7.6.2.1. Create pattern

7.119. Select **Options | IDEA settings**. The “IDE Options” dialog appears.
7.120. Select **TODO**.
7.121. For Patterns: Select **Add**. The dialog “Add Pattern” appears.
7.122. For Pattern: Enter **\btodoNOW\b.***.
7.123. Select the red icon.

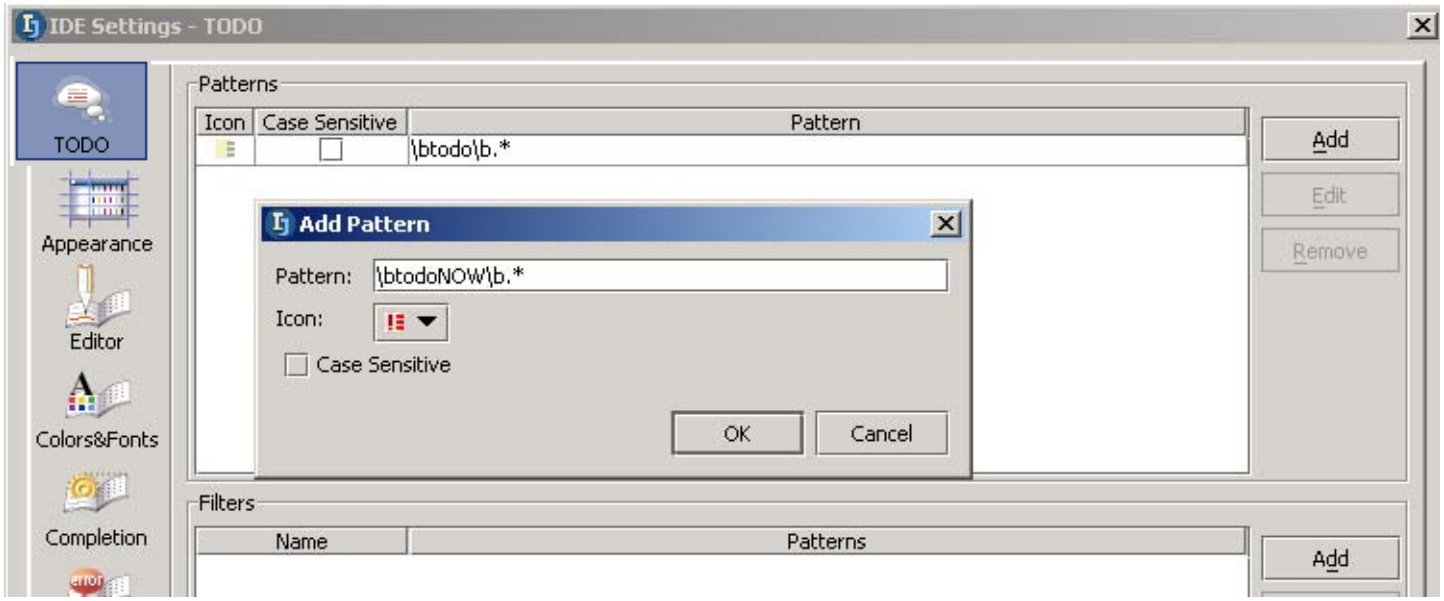


Figure 7.99. New TODO pattern (706)

7.124. Click **OK**. The new pattern appears in the list.

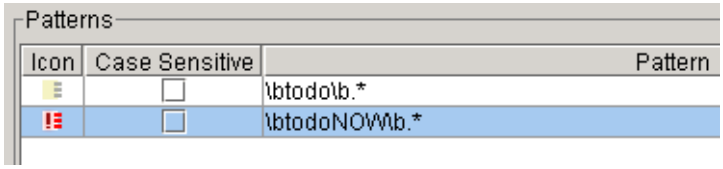


Figure 7.100. New TODO pattern (707)

7.125. Click **OK** to close.

7.6.2.2. Add pattern to file

7.126. In Class1 add the line

```
//@todonow urgent stuff
```

7.127. View the todo in teh TODO dialog:

Figure 7.101. New TODO in the TODO tool (708)

7.6.3. Filters

You can create filters that only show certain todo patterns.

- **Create filter**
- **Display filtered TODOs**

7.6.3.1. Create filter

7.128. In dialog “IDEA options” tab “TODO”: For **Filters** click **Add**. The dialog “Add filter” appears.
7.129. For “Name” enter **NOW**.
7.130. Check **\btodoNOW\b.***.

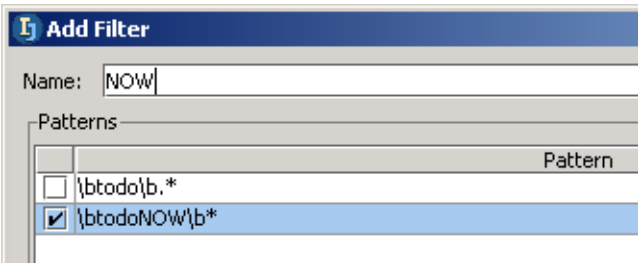


Figure 7.102. New TODO filter (709)

7.131. Click **OK**. The new filter appears in the list.

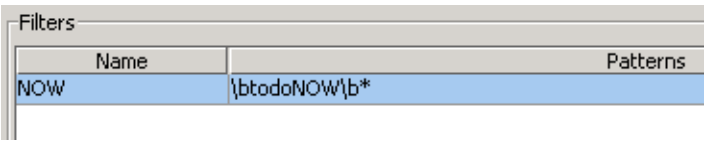


Figure 7.103. New TODO filter (710)

7.132. Click **OK** to close.

7.6.3.2. Display filtered TODOs

7.133. In the TODO dialog: Click on filter icon ( ). A popup for selecting the filter appears.

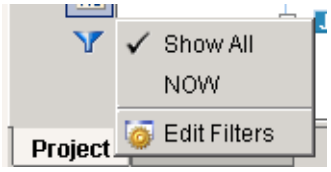


Figure 7.104. Popup for selecting filter (711)

7.134. Select **NOW**. Only the NOW TODOs are shown.

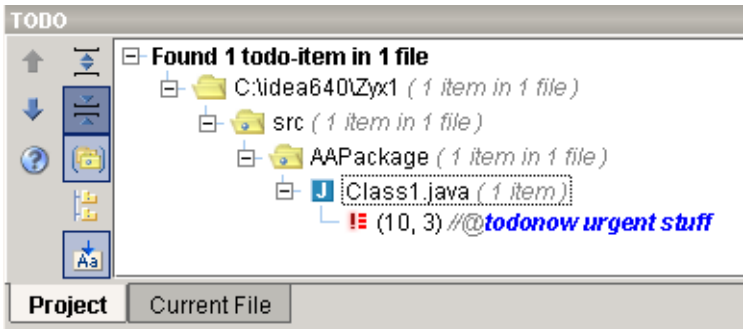


Figure 7.105. Only filtered TODOs shown (713)

8. Code Automation

contacts: many, maxim, valya

IDEA offers the following code automation functions:

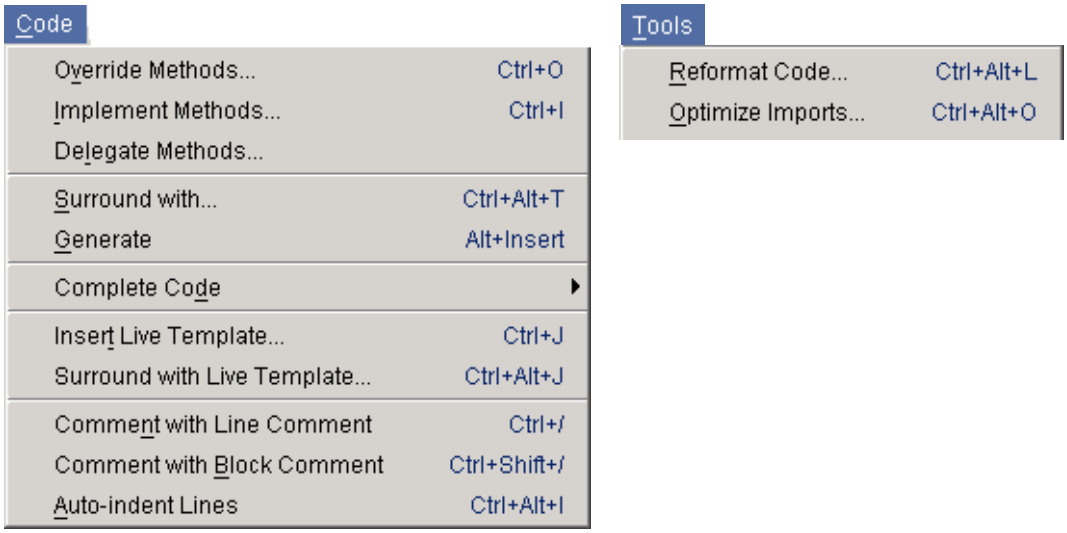


Figure 8.1. Code automation menu items (417,416,415)

8.1. Code completion (suggestion)

20021016TTT recommended dialog changes.

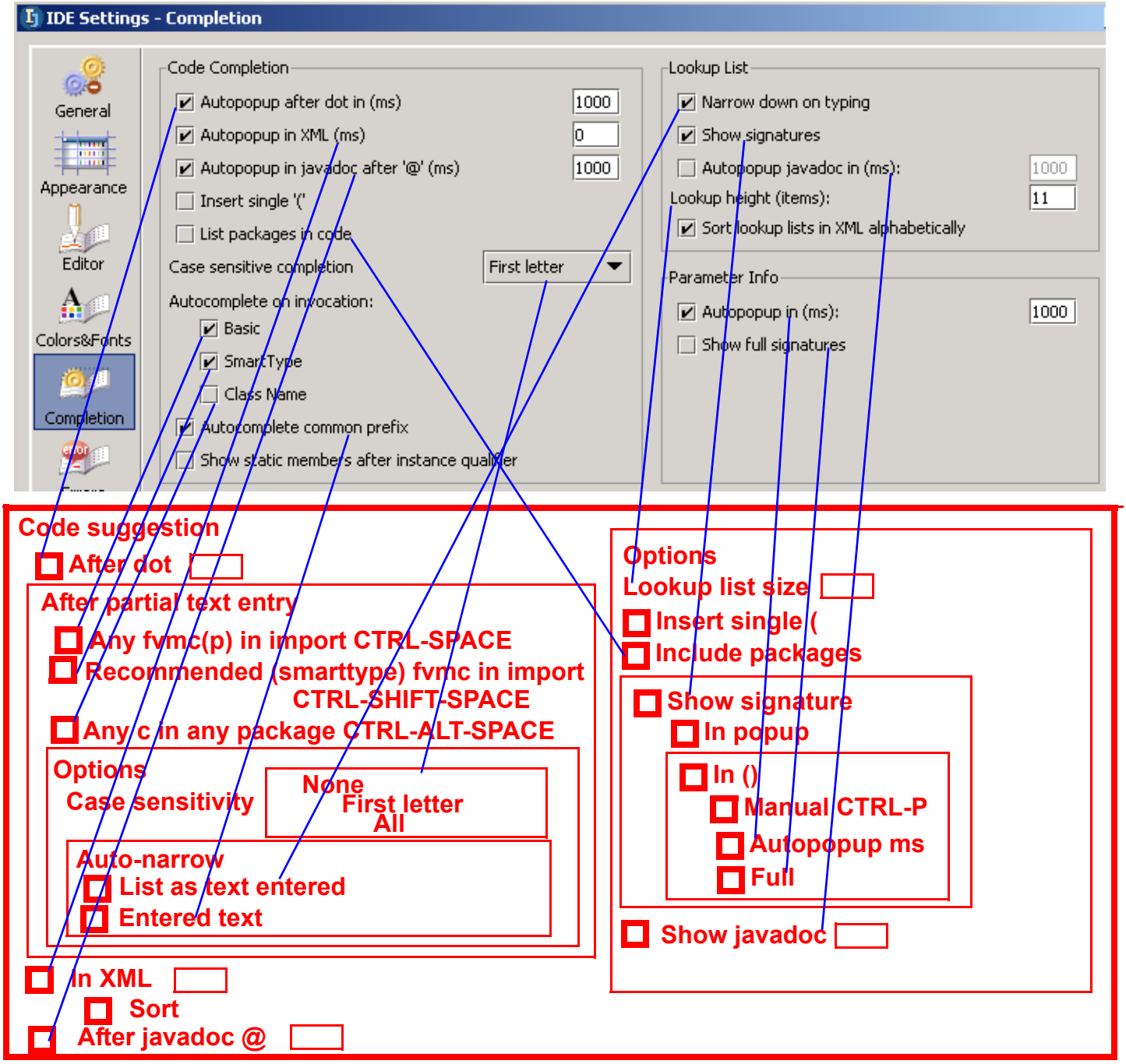


Figure 8.2. current and recommended dialog (812)

IDEA can suggest how to complete partially entered code:

- 8.1.1. Suggest after dot (auto) (page 148)
- 8.1.2. Suggest after partial text (manual) (page 149)
- 8.1.3. Suggest in XML (auto) X (page 156)
- 8.1.4. Suggest after @ (javadoc) (auto) X (page 156)

The above functions have several

- **8.1.5. Options (page 157)**

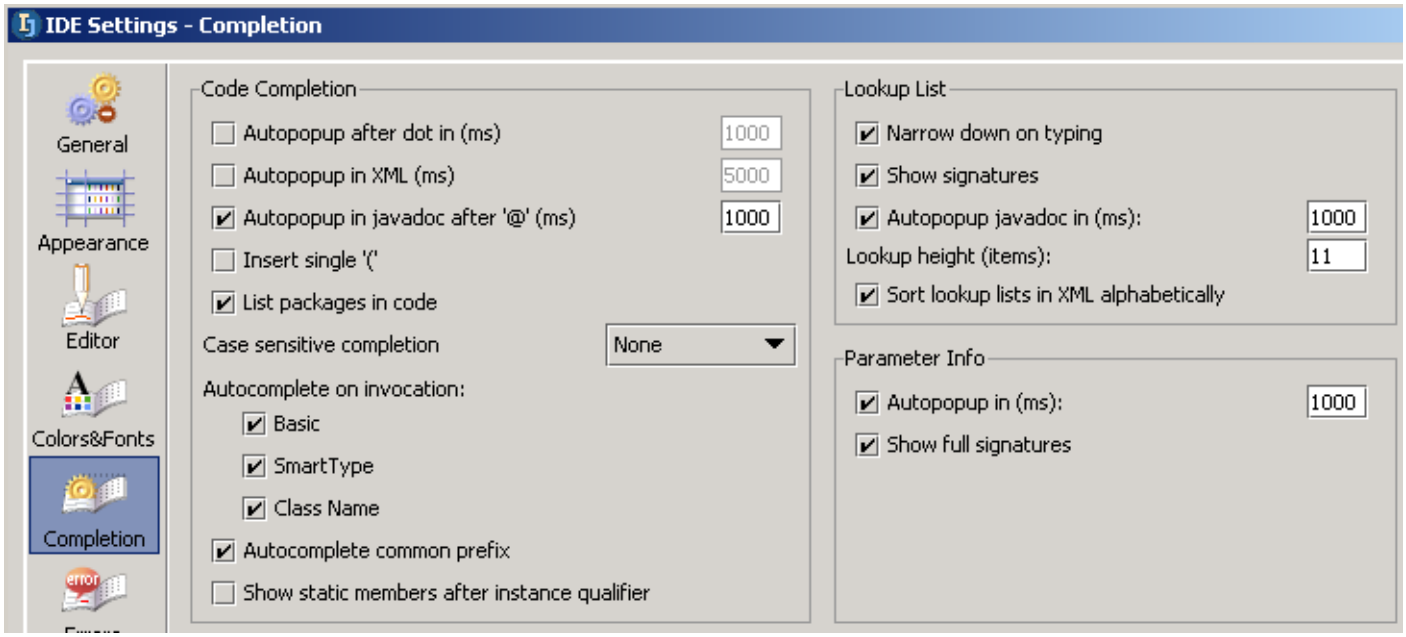


Figure 8.3. IDEA Settings / Completion (858)

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).

8.1. Open **IDEA Settings | Completion**.
8.2. Check the checkbox **Autopopup after dot in (ms)**.
8.3. For the value enter **5000** (5 secs).
8.4. Click **OK**.
8.5. In the editor: Enter "**System.out.**". Note that after 5 seconds an autopopup appears.

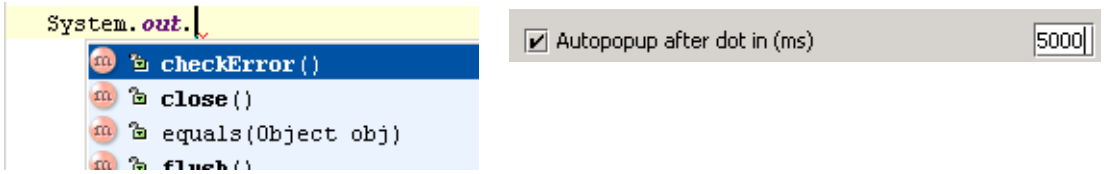


Figure 8.4. Autopopup (824,823)

8.6. Open **IDEA Settings | Completion**.
8.7. Uncheck the checkbox.
8.8. Click **OK**.
8.9. Again in the editor: Enter "**System.out.**". Note that the popup does not appear.
8.10. Click **CTRL-SPACE**. The popup appears.
8.11. Open **IDEA Settings | Completion**.
8.12. Check the checkbox.
8.13. For the value enter **1000** (1 sec).
8.14. 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 149)**
- **8.1.2.2. fvmc recommended in import CTRL-SHIFT-SPACE (smarttype) (page 150)**
- **8.1.2.3. Classes recommended in all packages CTRL-ALT-SPACE (page 151)**

The above functions have several

- **8.1.2.4. Options (page 152)**

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

8.15. Enter the following code:

```
String myString = "hello";
myString.c
```

8.16. Leave the cursor positioned after "myString.c".

8.17. Press **CTRL-SPACE**. A list of suggestions appears.

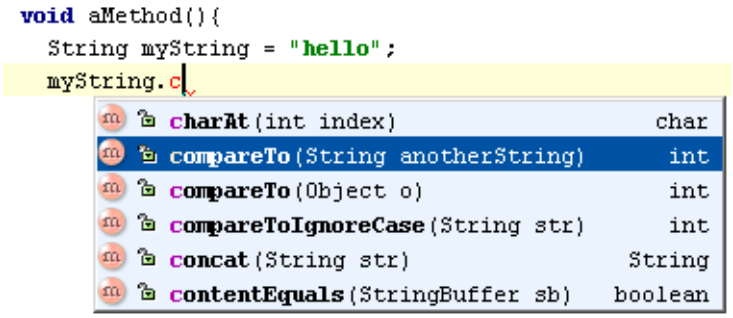


Figure 8.5. Matching fvmc(p) in import (842)

8.18. 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

8.19. Enter the following code:

```
String myString = “hello”;
myString.compareTo(
```

8.20. Leave the cursor positioned after “myString.compareTo(”.

8.21. Press **CTRL-SHIFT-SPACE**. A list of suggestions that are most likely required in this situation appears.

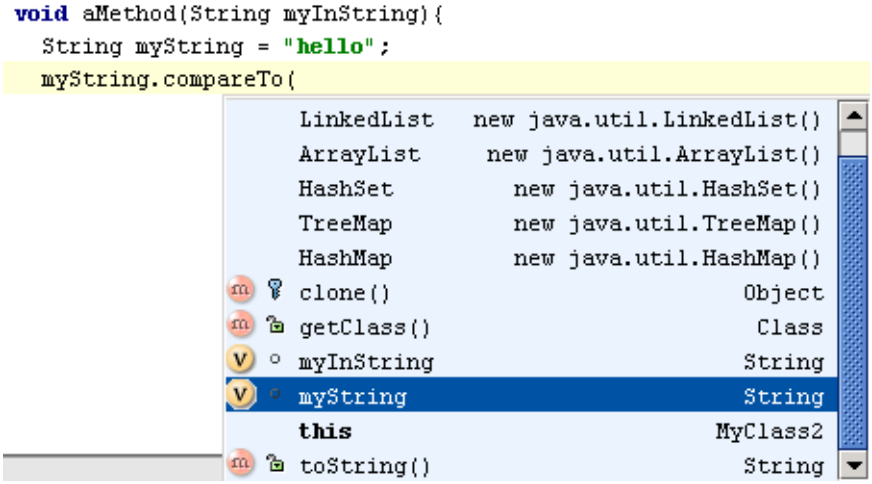


Figure 8.6. Recommended fvmc in import (843)

8.22. 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).

8.23. Enter the following code (the letter O):

O

8.24. Leave the cursor positioned after “O”.

8.25. Press **CTRL-ALT-SPACE**. A list of all classes in all packages that start with “O” are listed.

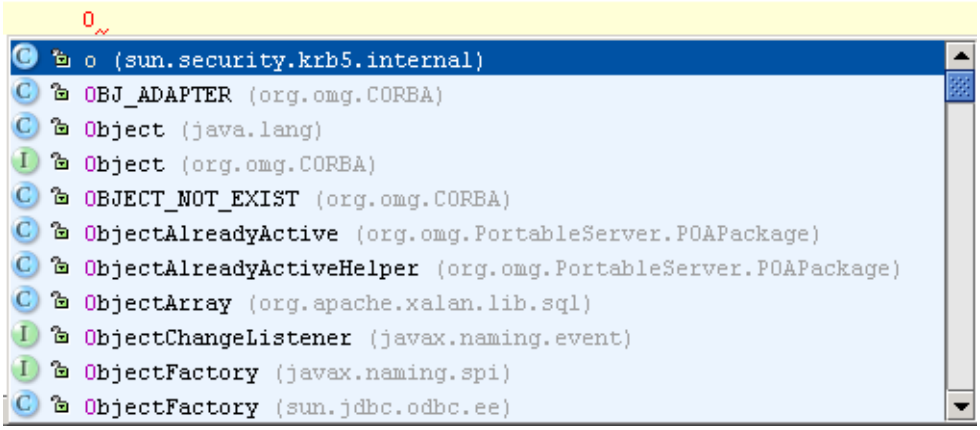


Figure 8.7. All matching classes (844)

8.26. 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

IDEA can limit the list of suggestions to only those possibilities whose

letters have the same case as the entered partial text.

8.1.2.4.1.1. None

8.27. Open **IDEA Settings | Completion**.
8.28. For “Case sensitive completion” select **None**.
8.29. Uncheck the 3 checkboxes for “Autocomplete on invocation”.
8.30. Click **OK**.
8.31. 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
}
```

8.32. Place the cursor at the end of “MyClass2.AB”.
8.33. Press **CTRL-SPACE**. A popup appears with all 3 options.

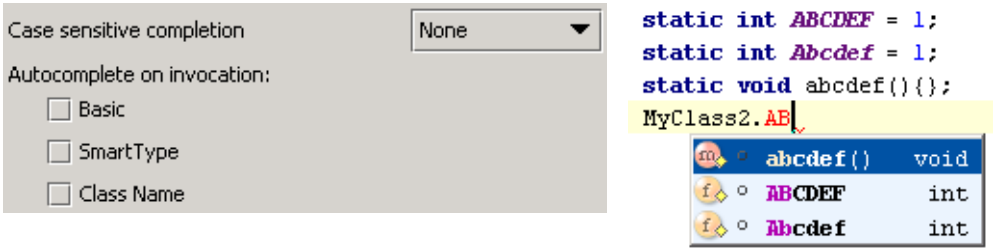


Figure 8.8. Case sensitive completion: None (835,836)

8.1.2.4.1.2. First letter

8.34. Open **IDEA Settings | Completion**.
8.35. For “Case sensitive completion” select **First letter**.
8.36. Click **OK**.
8.37. Place the cursor at the end of “MyClass2.AB”.
8.38. Press **CTRL-SPACE**. A popup appears with 2 options.

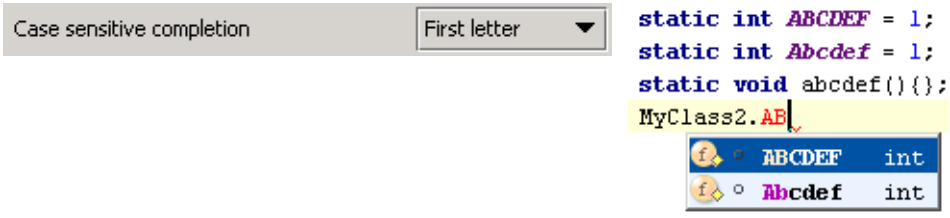


Figure 8.9. Case sensitive completion: First letter (837,838)

8.1.2.4.1.3. All

8.39. Open **IDEA Settings | Completion**.
8.40. For “Case sensitive completion” select **All**.
8.41. Click **OK**.
8.42. Place the cursor at the end of “MyClass2.AB”.
8.43. Press **CTRL-SPACE**. A popup appears with 1 option.

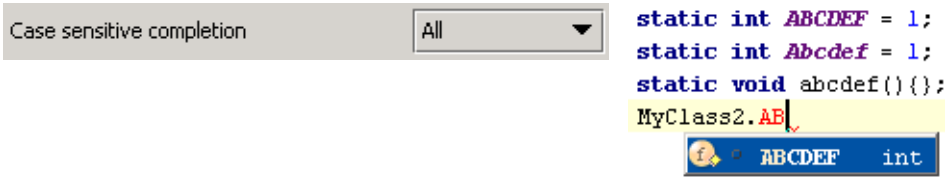


Figure 8.10. Case sensitive completion: All (839,840)

8.1.2.4.2. Auto-narrow

IDEA can automatically narrow the

8.1.2.4.2.1. List as text entered

8.44. Check the checkbox **Narrow down on typing**.
8.45. Enter the following code:

```
String myString = "hello";
myString.c
```

8.46. Leave the caret at the end of "myString.c". Note the auto
8.47. Click **CTRL-SPACE**. A list of suggestions appears.

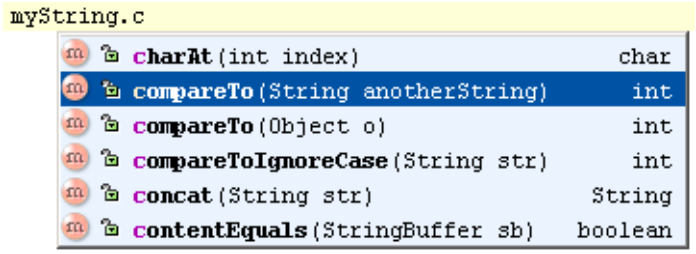


Figure 8.11. Suggestions after myString.c (845)

8.48. Press the "o" key. The list is narrowed.

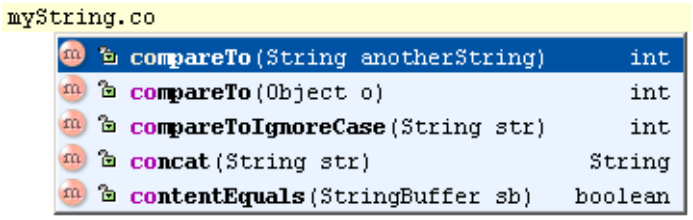


Figure 8.12. Suggestions after myString.co (846)

8.49. Press the "m" key. The list is further narrowed.

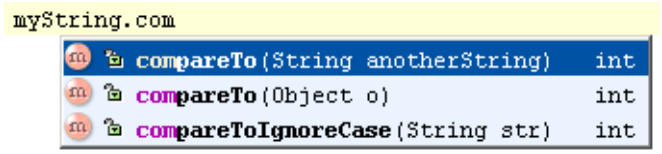


Figure 8.13. Suggestions after myString.com (847)

8.1.2.4.2.2. Entered text

8.50. Check the checkbox **Autocomplete common prefix**.
8.51. Enter the following code:

```
String myString = "hello";
myString.com
```

8.52. Leave the caret at the end of "myString.com".
8.53. Click **CTRL-SPACE**. Note that the letters "pareTo" are added and the corresponding suggestion list displayed.

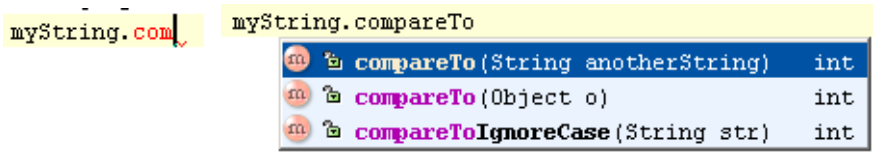


Figure 8.14. Entered text narrowd (848,849)

8.1.3. Suggest in XML (auto) X

Similar to **8.1.1. Suggest after dot (auto) (page 148)**, 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 148)**, but for popups after the javadoc '@'.

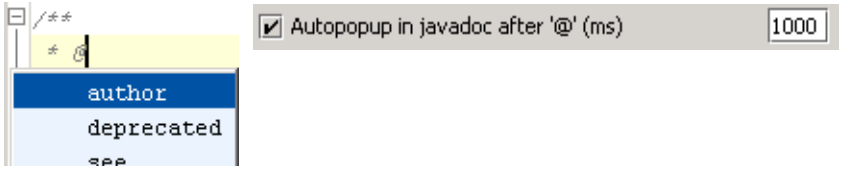


Figure 8.15. Autopopup in javadoc after '@' (826,825)

8.1.5. Options

The following options for code recommendations are available:

8.1.5.1. Lookup list height

Determines the number of lines in the suggestion popup.

8.54. Set "Lookup list height" to **2**.
8.55. Display a list. Note that the list displays only 2 lines (you can scroll to view the other lines).

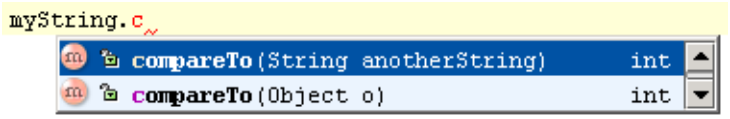


Figure 8.16. Lookup list height = 2 (856)

8.1.5.2. Insert single '('

If checked: Only a single '(' is inserted after a suggestion has been selected.

8.56. Check the checkbox **Insert single ')'**.
8.57. In the editor: Add the following:

```
String aString = "Hello";
aString.getBytes
```

8.58. Place the cursor at the end of "getBytes".
8.59. Press **CTRL-SPACE**. A single ')' appears.

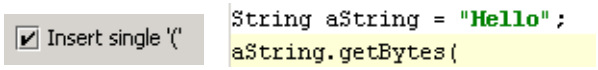


Figure 8.17. Insert single ')' (827,828)

8.60. Delete the single ')'.
8.61. Uncheck the checkbox **Insert single ')'**.
8.62. Place the cursor at the end of "getBytes".
8.63. Press **CTRL-SPACE**. '( )' appears.

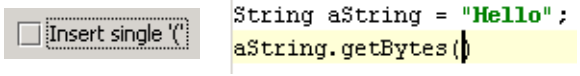


Figure 8.18. Insert '()' (829,830)

8.1.5.3. Include packages

If checked: Available packages are included in the list of suggestions.

8.64. Check the checkbox **List packages in code**.

8.65. In the editor: Add the following:

```
String comString = "Hello";
c
```

8.66. Place the cursor at the end of “c”.

8.67. Press **CTRL-SPACE**. A popup appears that includes packages.

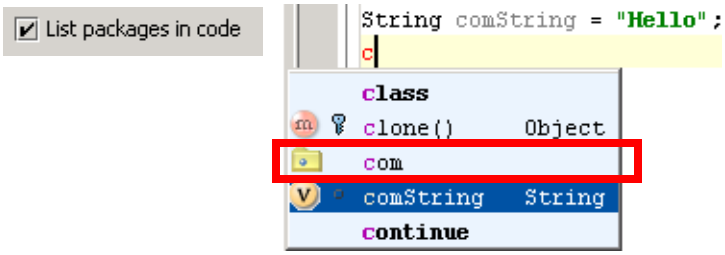


Figure 8.19. List packages in code (832,831)

8.68. Uncheck the checkbox **List packages in code**.

8.69. Place the cursor at the end of “c”.

8.70. Press **CTRL-SPACE**. A popup appears that does not include packages.

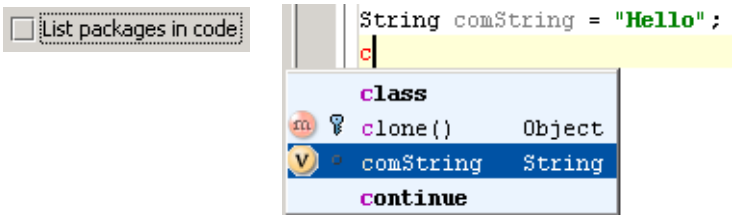


Figure 8.20. No packages in code (833,834)

8.1.5.4. Show signature

The signature can be optionally shown

8.1.5.4.1. In popup

8.71. Check the checkbox **Show signatures**.
8.72. In the editor: Add the following:

```
String aString = "Hello";
aString.c
```

8.73. Place the cursor at the end of "c".
8.74. Press **CTRL-SPACE**. A popup appears that includes the signatures.

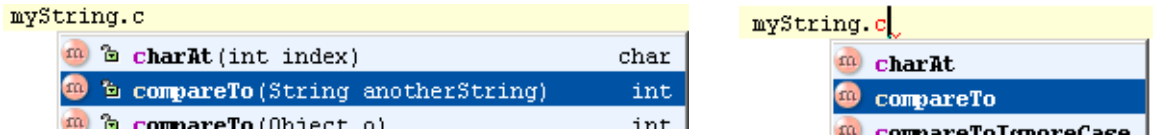


Figure 8.21. With / without signature in popup (850,857)

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

The signature can be displayed manually with CTRL-P.

8.75. In the editor: Add the following:

```
String aString = "Hello";
aString.compareTo()
```

8.76. Place the cursor at the end of "compareTo(".
8.77. Press **CTRL-P**. A popup appears with a suggestion that includes signatures.

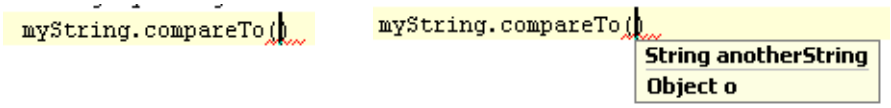


Figure 8.22. CTRL-P signature (851,852)

8.1.5.4.2.2. Autopopup (ms)

8.78. Check the checkbox **Autopopup in ms**.
8.79. In the editor: Add the following:

```
String aString = "Hello";
aString.c
```

8.80. Place the cursor at the end of "c".
8.81. Click **CTRL-SPACE**.
8.82. Double-click on **compareTo(String anotherString)**. The caret is placed between '()'. After 1 sec the popup appears.

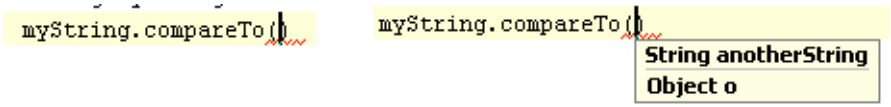


Figure 8.23. Signature autopopup (851,852)

8.1.5.4.2.3. Full

8.83. Check the checkbox **Show full signatures**.
8.84. In the editor: Add the following:

```
String aString = "Hello";
aString.compareTo()
```

8.85. Place the cursor at the end of “compareTo(”.
8.86. Press **CTRL-P**. A popup appears with suggestions that include full signatures.

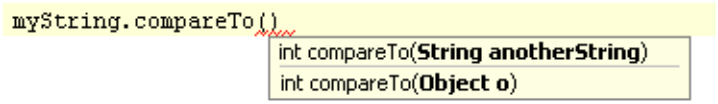


Figure 8.24. Full signatures (853)

8.1.5.5. Show javadoc

8.87. Check the checkbox **Autopopup javadoc in (ms)**.
8.88. In the editor: Add the following:

```
String aString = "Hello";
aString.c
```

8.89. Place the cursor at the end of “aString.c”.
8.90. Press **CTRL-SPACE**. A popup appears with suggestions.
8.91. After 1 sec the Java doc for the highlighted suggestion is displayed.
8.92. 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.

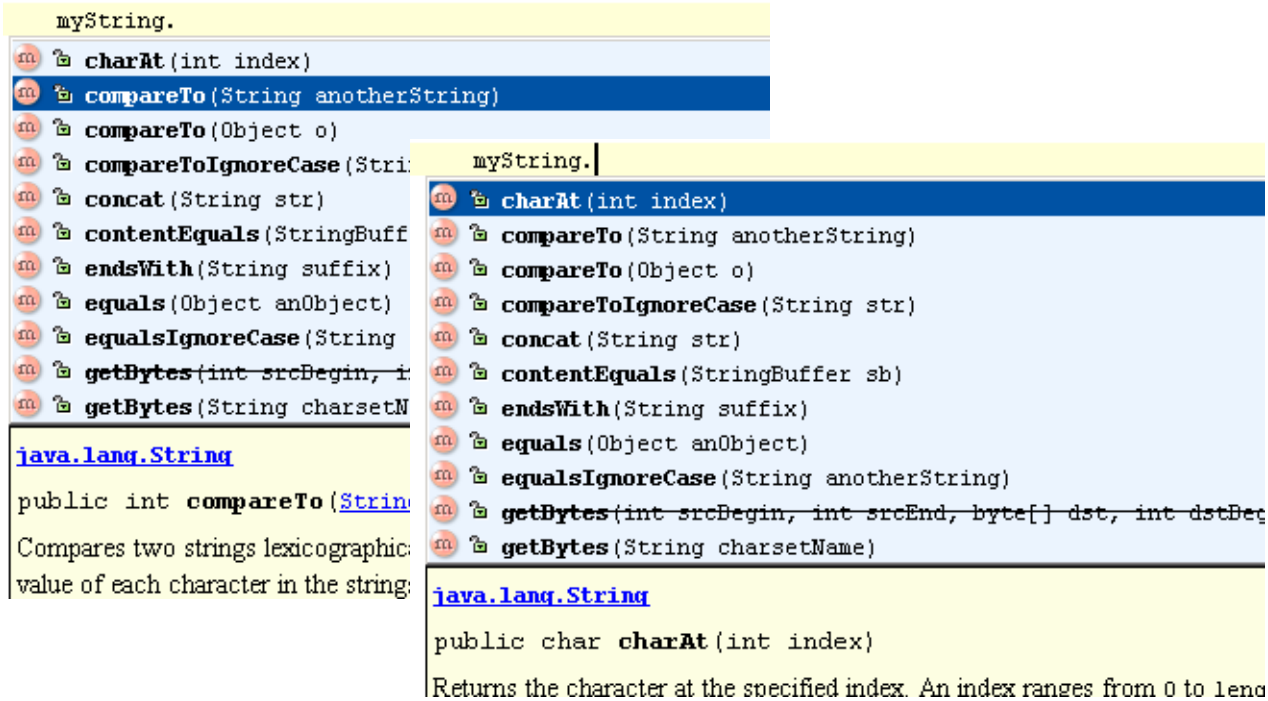


Figure 8.25. Autopopup of JavaDoc for suggestion (854,855)

8.2. Code-completion OLD XXX

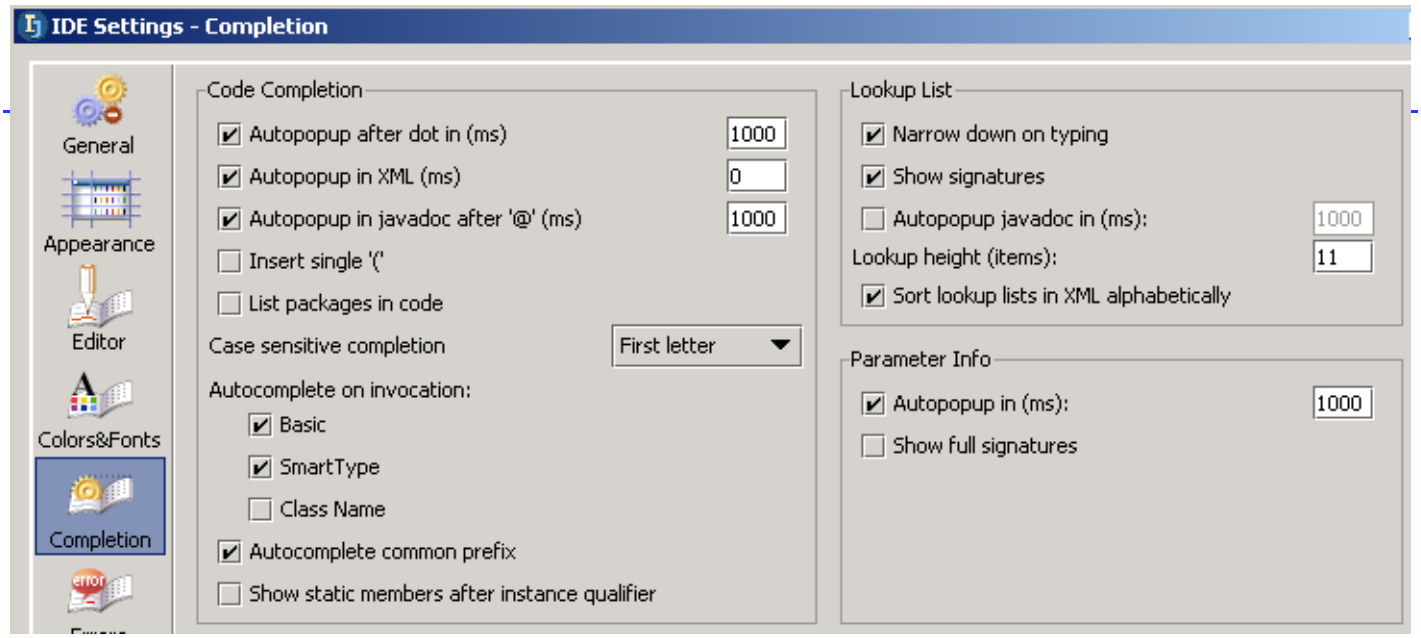


Figure 8.26. Completion (812)

8.2.1. Code completion X

helptopic idea\preferences\editor\codeCompletion.html

IDEA provides the following functions for automatically completing code:

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).

8.93. Open **IDEA Settings | Completion**.
8.94. Check the checkbox **Autopopup after dot in (ms)**.
8.95. For the value enter **5000** (5 secs).
8.96. Click **OK**.
8.97. In the editor: Enter "**System.out.**". Note that after 5 seconds an autopopup appears.

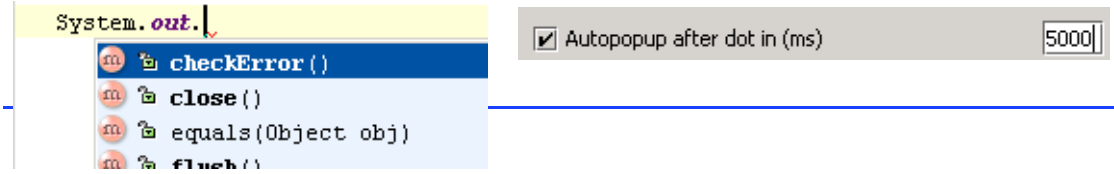


Figure 8.27. Autopopup (824,823)

8.98. Open **IDEA Settings | Completion**.
8.99. Uncheck the checkbox.
8.100. Click **OK**.
8.101. Again in the editor: Enter "**System.out.**". Note that the popup does not appear.
8.102. Click **CTRL SPACE**. The popup appears.
8.103. Open **IDEA Settings | Completion**.
8.104. Check the checkbox.
8.105. For the value enter **1000** (1 sec).
8.106. 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 '@'.

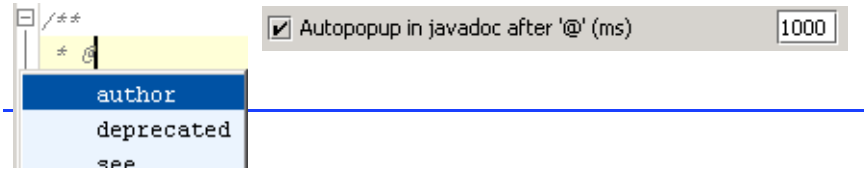


Figure 8.28. Autopopup in javadoc after '@' (826,825)

8.2.1.4. Insert single ')'

8.107. Open **IDEA Settings | Completion**.
8.108. Check the checkbox **Insert single ')'**.
8.109. Click **OK**.
8.110. In the editor: Add the following:

```
String aString = "Hello";
aString.getBytes
```

8.111. Place the cursor at the end of "getBytes".
8.112. Press **CTRL SPACE**. A single ')' appears.

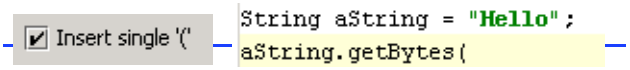


Figure 8.29. Insert single ')' (827,828)

8.113. Delete the single ')'.
8.114. Open **IDEA Settings | Completion**.
8.115. Uncheck the checkbox **Insert single ')'**.
8.116. Click **OK**.
8.117. Place the cursor at the end of "getBytes".
8.118. Press **CTRL SPACE**. '( )' appears.

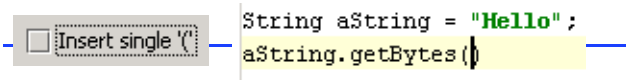


Figure 8.30. Insert '()' (829,830)

8.2.1.5. List packages in code

8.119. Open **IDEA Settings | Completion**.
8.120. Check the checkbox **List packages in code**.
8.121. Click **OK**.
8.122. In the editor: Add the following:

```
String comString = "Hello";
c
```

8.123. Place the cursor at the end of "c".
8.124. Press **CTRL-SPACE**. A popup appears that includes packages.

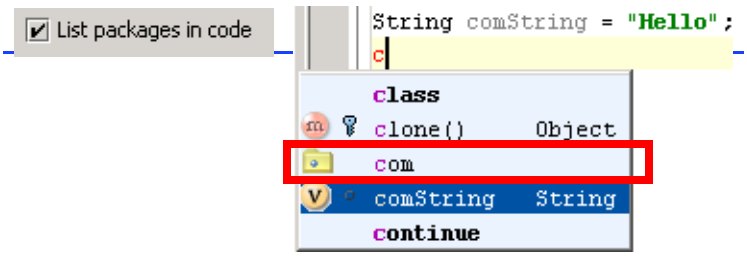


Figure 8.31. List packages in code (832,831)

8.125. Open **IDEA Settings | Completion**.
8.126. Uncheck the checkbox **List packages in code**.
8.127. Click **OK**.
8.128. Place the cursor at the end of "c".
8.129. Press **CTRL-SPACE**. A popup appears that does not include packages.

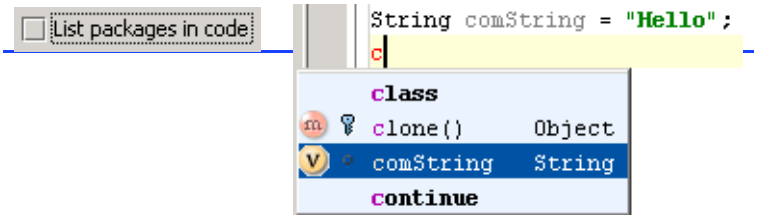


Figure 8.32. No packages in code (833,834)

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 163)**
- **8.2.1.6.2. First letter (page 164)**
- **8.2.1.6.3. All (page 164)**

8.2.1.6.1. None

8.130. Open **IDEA Settings | Completion**.
8.131. For "Case sensitive completion" select **None**.
8.132. Uncheck the 3 checkboxes for "Autocomplete on invocation".
8.133. Click **OK**.
8.134. 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
}
```

8.135. Place the cursor at the end of “MyClass2.AB”.
8.136. Press **CTRL-SPACE**. A popup appears with all 3 options.

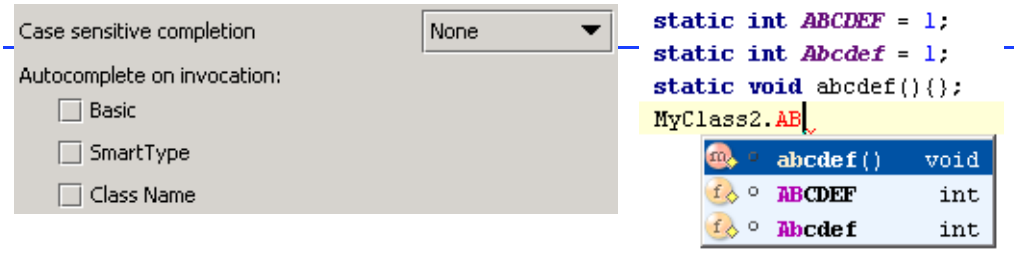


Figure 8.33. Case sensitive completion: None (835,836)

8.2.1.6.2. First letter

8.137. Open **IDEA Settings | Completion**.
8.138. For “Case sensitive completion” select **First letter**.
8.139. Click **OK**.
8.140. Place the cursor at the end of “MyClass2.AB”.
8.141. Press **CTRL-SPACE**. A popup appears with 2 options.

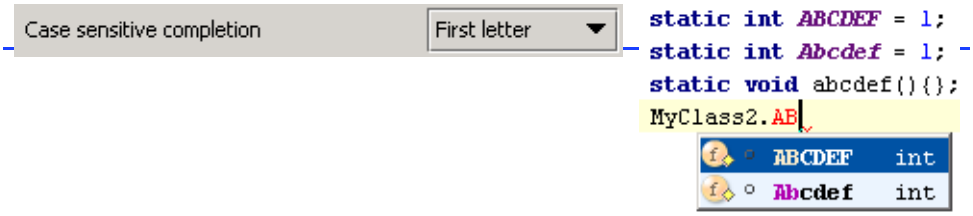


Figure 8.34. Case sensitive completion: First letter (837,838)

8.2.1.6.3. All

8.142. Open **IDEA Settings | Completion**.
8.143. For “Case sensitive completion” select **All**.
8.144. Click **OK**.
8.145. Place the cursor at the end of “MyClass2.AB”.
8.146. Press **CTRL-SPACE**. A popup appears with 1 option.

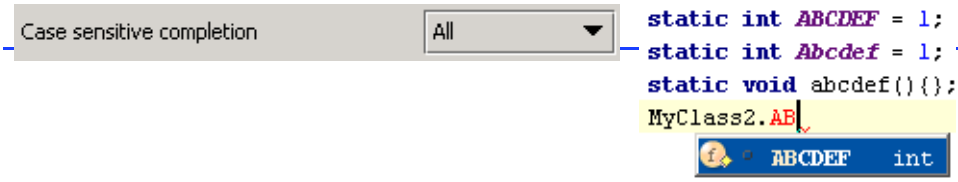


Figure 8.35. Case sensitive completion: All (839,840)

8.2.1.7. Autocompletion on invocation XXX

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~~

~~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~~

~~8.2.3.2. Show full signatures XXX~~

8.3. Code templates

IDEA provides the following code template functionality

8.3.1. Live Templates

There are many different

8.3.1.1. Type of insert

The following types of live templates are available:

8.3.1.1.1. Plain text

8.147. Create class **Class1**:

```
public class Class1 {
    public static void main(String[] args) {

    }
}
```

8.148. Place the cursor inside main().

8.149. Enter **St**.

8.150. Click **TAB**. Note that the live template code is added.

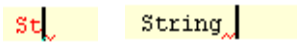


Figure 8.36. Insert plain text live template (870,871)

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.

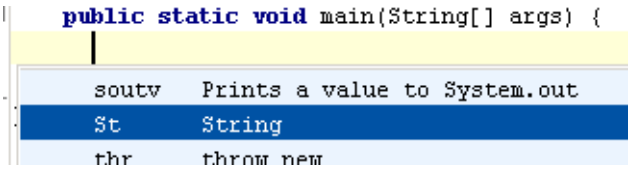


Figure 8.37. Insert using list (872)

8.3.1.1.2. With variables

8.151. Create class **Class1**:

```
public class Class1 {
    public static void main(String[] args) {

    }
}
```

8.152. Add live template**sout**. Note the final location of the cursor.

```
System.out.println("");
```

Figure 8.38. Final location of cursor (873)

8.153. Select **Options | Live templates...**. The dialog “Live templates” appears.

8.154. Select **sout**. Note the variable “**END**”. This specifies the location of the cursor after the live template has been entered.

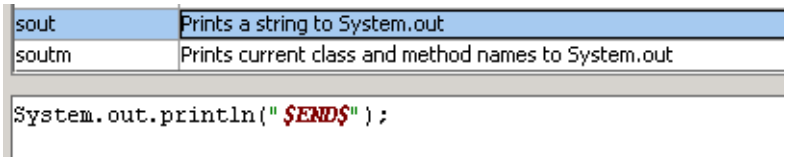


Figure 8.39. END variable (874)

8.155. Insert live template **itar**. An iteration is inserted.

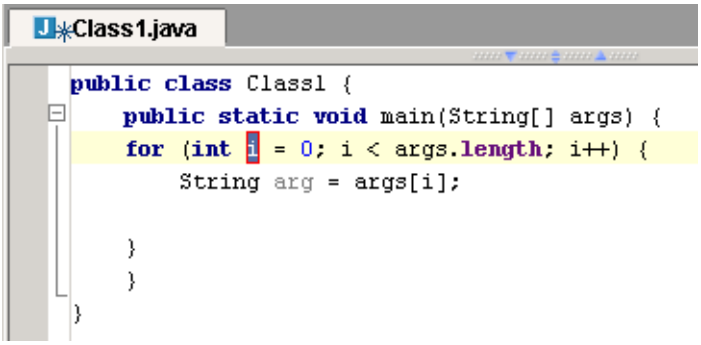


Figure 8.40. Inserted iteration (352)

8.156. Press the **j** key. Note that the iteration variable is changed.

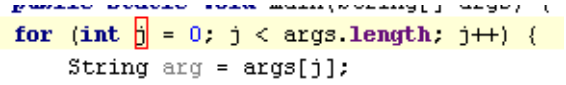


Figure 8.41. Changed iteration variable (351)

8.157. Click the **Tab** key. Note that the red square (the focus) has move to “args”.

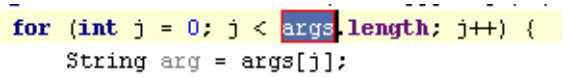


Figure 8.42. Focus moved (350)

The variable definitions explain the above behavior.

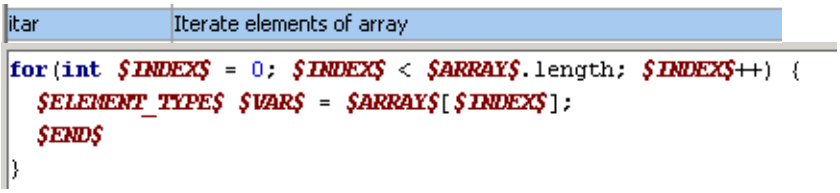


Figure 8.43. itar variables (876,875)

8.3.1.1.3. Surround

8.158. Select the contents of main

8.159. Select **Code | Surround with live template**. A list of surround live templates appears in a popup dialog.

8.160. Click on **B: Surround with {}**. The code is surrounded.

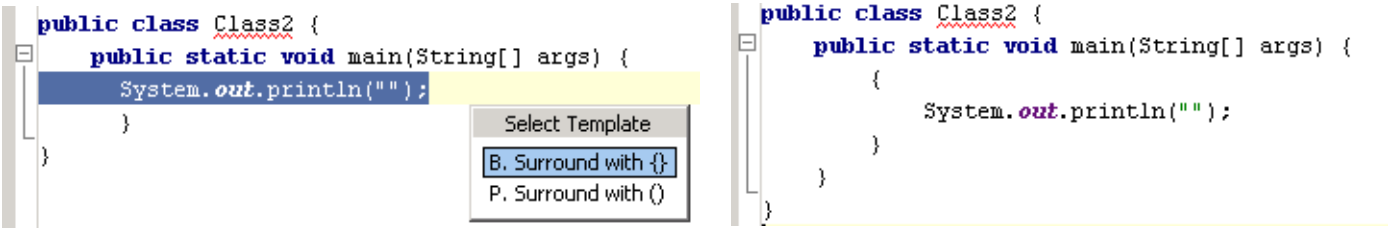


Figure 8.44. Surround with {} (877,878)

8.3.1.2. Context

The context determines how the live template can be used. The following contexts are supported:

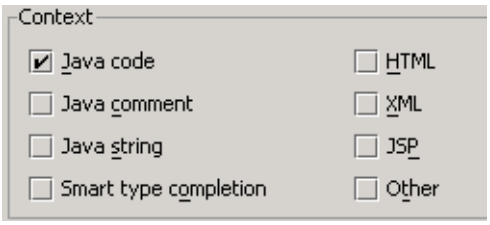


Figure 8.45. Contexts (879)

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.

8.161. Add **<** to the code.
8.162. Click **TAB**. The live template is not entered since this template is only supported in HTML, XML, and JSP files.

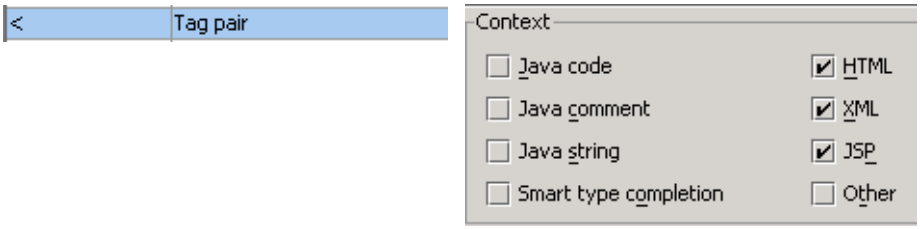


Figure 8.46. Tag pair contexts (880,881)

8.3.1.2.2. Java comment

If this context is selected, then the live template can be inserted into java comments.

8.163. For Tag pair: Check content **Java comment**.
8.164. Add **<** to the code comment.
8.165. Click **TAB**. The live template is entered.

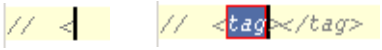


Figure 8.47. Tag pair in comments (882,883)

8.3.1.2.3. Java String

If this context is selected, then the live template can be inserted into java strings.

8.166. For Tag pair: Check content **Java string**.
8.167. Add **<** to the string in System.out.println.
8.168. Click **TAB**. The live template is entered.

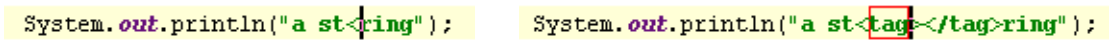


Figure 8.48. Tag pair in a string (884,885)

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.

8.169. Add the following:

```
String s = "";
s.compareTo()
```

8.170. Place the cursor between '()'.

8.171. 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.

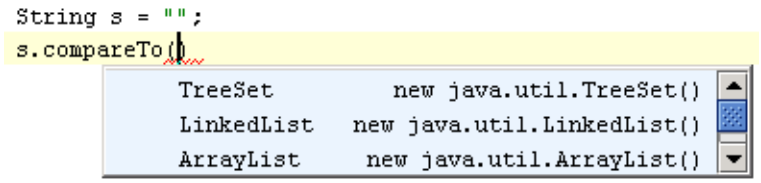


Figure 8.49. Smart type completion (886)

8.3.1.2.5. HTML

If this context is selected, then the live template can be inserted into HTML files.

8.172. Add the **<** live template to an HTML file.

Figure 8.50. 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 171)**
- **8.3.1.3.2. Add / Copy (page 172)**
- **8.3.1.3.3. Remove (page 172)**

8.3.1.3.1. Edit

8.173. Select **Options | Live templates...**. The dialog “Live templates” appears.
8.174. Double-click on **soutm**. The dialog “Edit Live Template” appears.

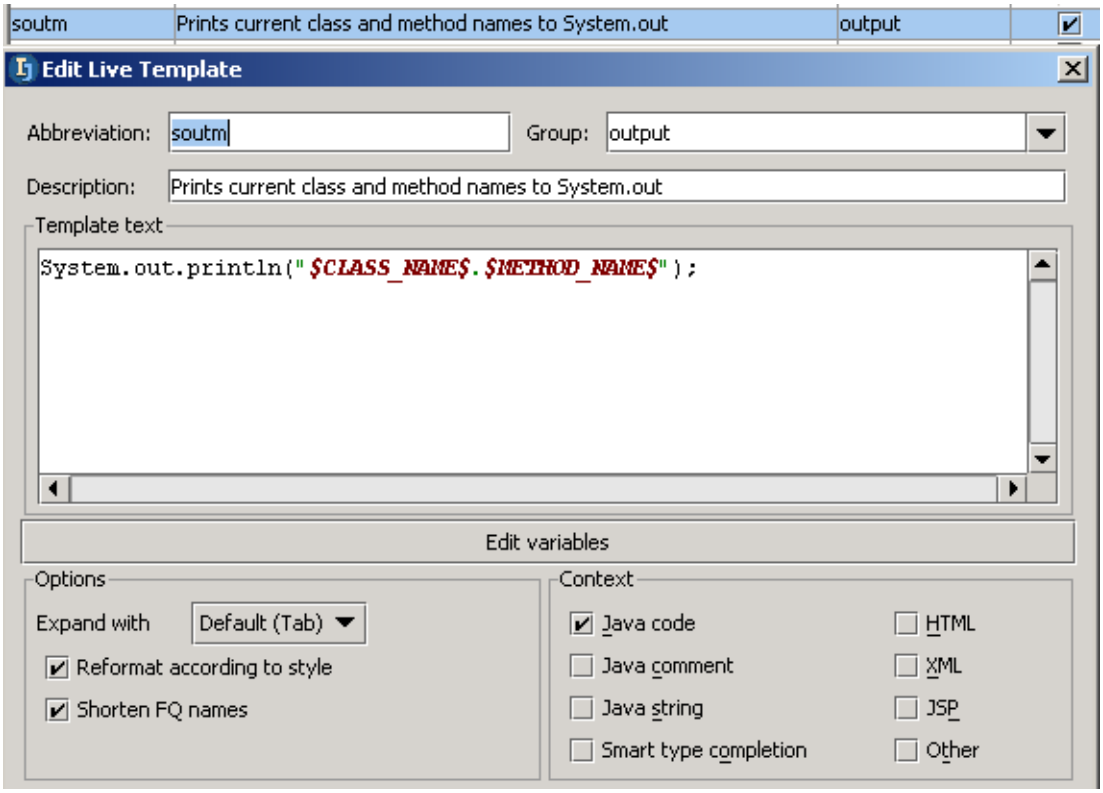


Figure 8.51. Edit live template for soutm (887)

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”.

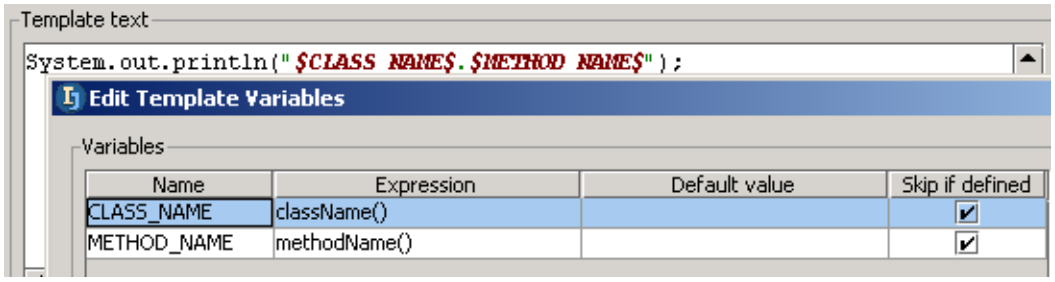


Figure 8.52. Edit template variables (888)

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.

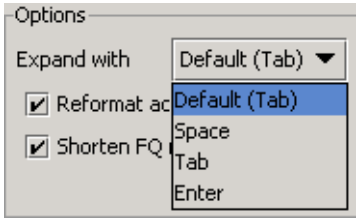


Figure 8.53. Expand with options (889)

Reformat according to style

If checked: Reformat according to style.

Shorten FQ names XXX

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

8.175. Create class **Class1**:

```
public class Class1 {
    public void aMethod() {
        System.out.println("Hello");
    }
}
```

8.176. Place the caret in “System.out....”.

8.177. Select **Code | Surround with...**. A popup dialog appears.

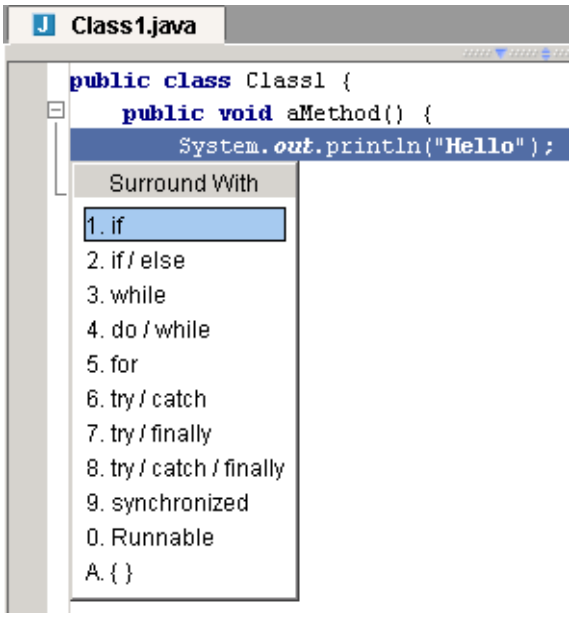


Figure 8.54. Dialog “Surround with” (361)

8.178. Select **If**. The line is surrounded with an “if” clause.

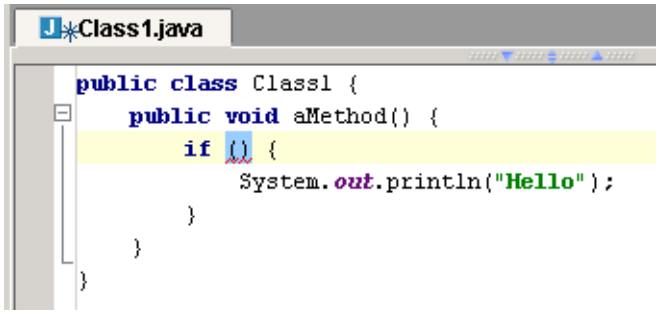


Figure 8.55. Surrounded with an “if” clause (360)

8.4. Code generation

IDEA can automatically generate some types of code (such as getter and setter methods).

8.179. Create class **Class1**:

```
public class Class1 {
    int int1;
    int int2;
}
```

8.180. Select **Code | Generated...**. A popup dialog appears.

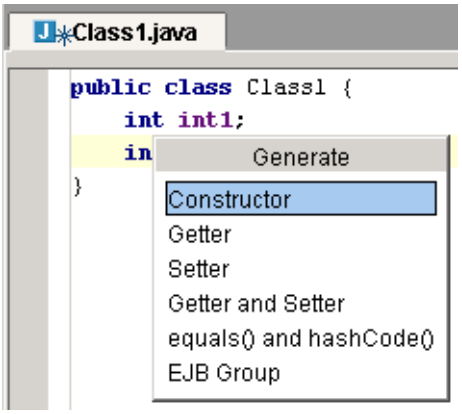


Figure 8.56. Popup “Generarte” (359)

8.181. Select **Constructor**. The dialog “Choose field to initialize by constructor” appears.

8.182. Select both **int1** and **int2** (using CTRL key).

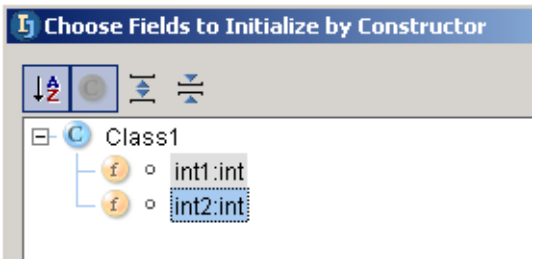


Figure 8.57. Dialog “Choose field to initialize by constructor” (358)

8.183. Click **OK**. The constructor is generated.

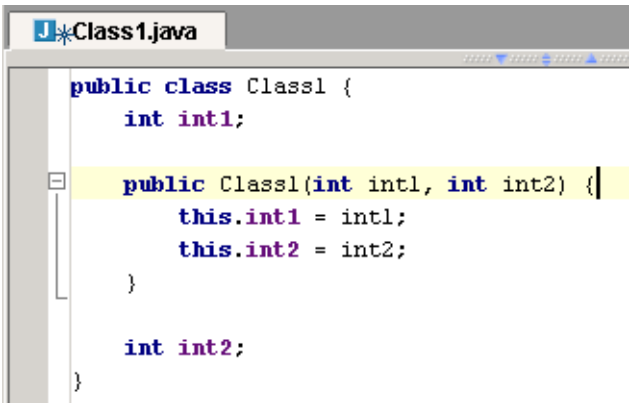


Figure 8.58. Generated constructor (357)

8.5. Import optimization

IDEA can optimize imports for

8.5.1. File

8.184. Create class **MyClass2**:

```
import java.util.*;
import com.sun.*;
package MyPackage
public class MyClass2 {}
```

8.185. Select **Tools | Optimize imports...**. The dialog “Optimize imports” appears.

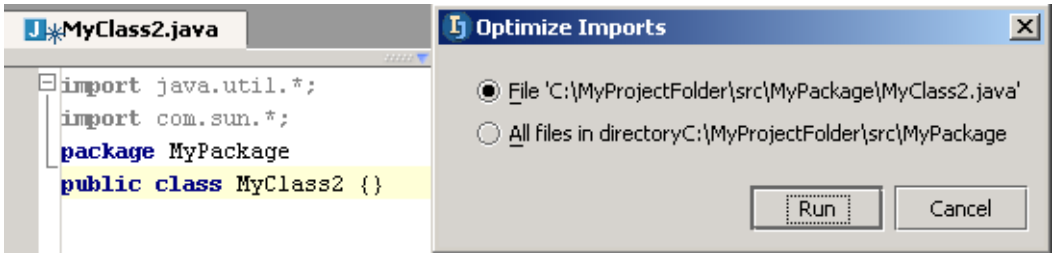


Figure 8.59. Dialog “Optimize imports” (339)

8.186. Select **Run**. Imports are optimized.

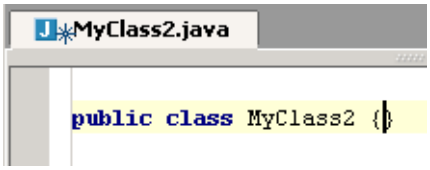


Figure 8.60. Optimized imports (338)

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.

8.187. Create class **MyClass**:

```
package MyPackage;
public class MyClass {}
```

8.188. Place the cursor on the line “public...”.

8.189. Select **Code | Override methods**. The dialog “Select methods to override” appears.

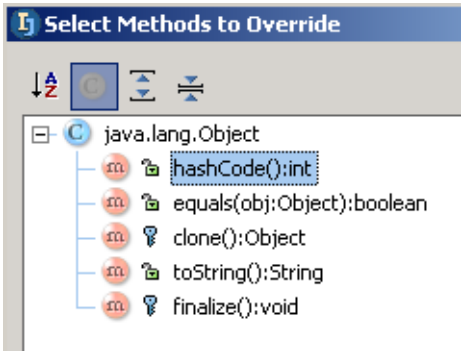


Figure 8.61. Override methods (453)

8.190. Double-click on **toString**. The code to override the method is added:

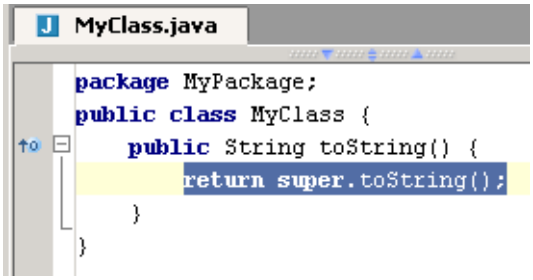


Figure 8.62. Overridden method toString() (452)

8.191. Place the cursor on the overriden method icon ( ). Note the message that appears:

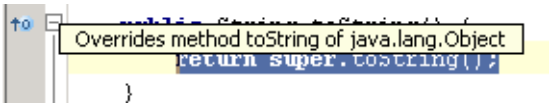


Figure 8.63. Overridden method message (451)

8.192. Click on the icon. The source file for the overriden method is opened.

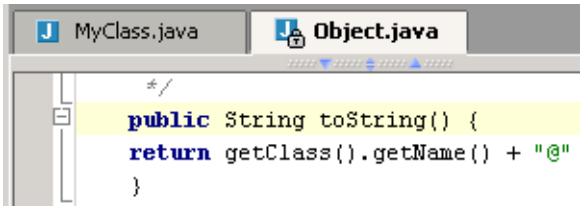


Figure 8.64. Overridden method (450)

8.193. Create class **MyClass2**:

```
package MyPackage;
public class MyClass2 extends MyClass {}
```

8.194. Place the cursor on the line “public...”.

8.195. Select **Code | Override methods**. The dialog “Select methods to override” appears. Note that the methods for all superclasses are shown.

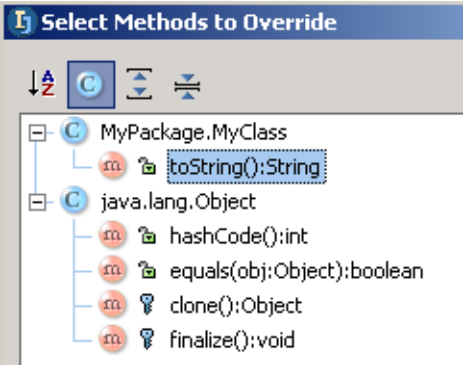


Figure 8.65. Override methods (448)

8.7. Interface implementation

IDEA makes it easy to implement interfaces.

8.196. Create class **Interface1**:

```
public interface Interface1 {
  void doSomething();
}
```

8.197. Create class **ImplementInterface1**:

```
public class ImplementInterface1 implements Interface1 {
}
```

8.198. Place the caret with “Class1” after “new”.

8.199. Select **Code | Implement methods**. The dialog “Select methods to implement” appears.

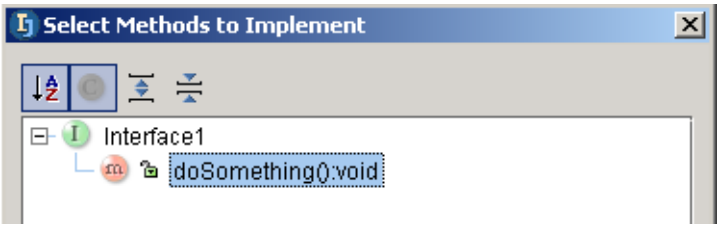


Figure 8.66. Dialog “Select methods to implement” (366)

8.200. Select **doSomething():void**.

8.201. Click **OK**. The method is implemented.

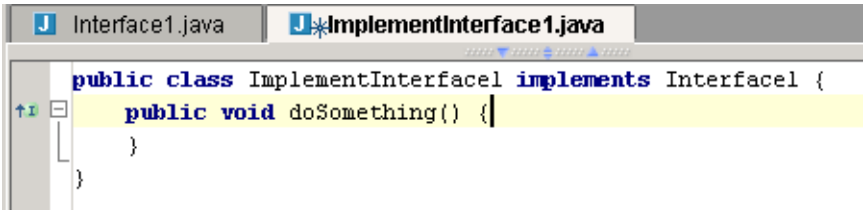


Figure 8.67. Method implemented (365)

8.8. Method delegation

IDEA makes it easy to delegate methods.

8.202. Create class **Class1**:

```
public class Class1 {
    public Class2 c2;
    public Class3 c3;
}
```

8.203. Create class **Class2**:

```
public class Class2 {
    public void method2a() {}
    public void method2b() {}
 }
```

8.204. Create class **Class3**:

```
public class Class3 {
  public void method3() {}
}
```

8.205. Place the caret in Class1.

8.206. Select **Code | Delegate methods**. The dialog “Select target to generate delegates for” appears.

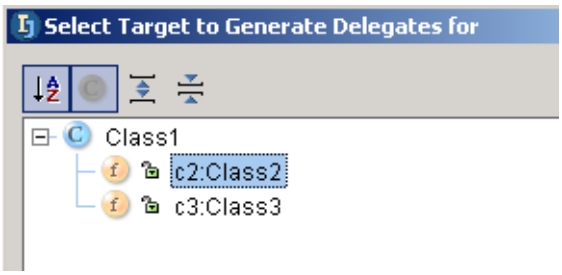


Figure 8.68. Dialog “Select target to generate delegates for” (364)

8.207. Select **c2:Class2**. The dialog “Select methods to generate delegates for” appears.

8.208. Select both **c2** and **c3** (using CTRL key).

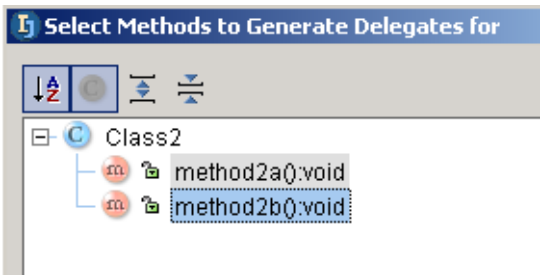


Figure 8.69. Dialog “Select methods to generate delegates for” (363)

8.209. Click **OK**. The delegates are generated.

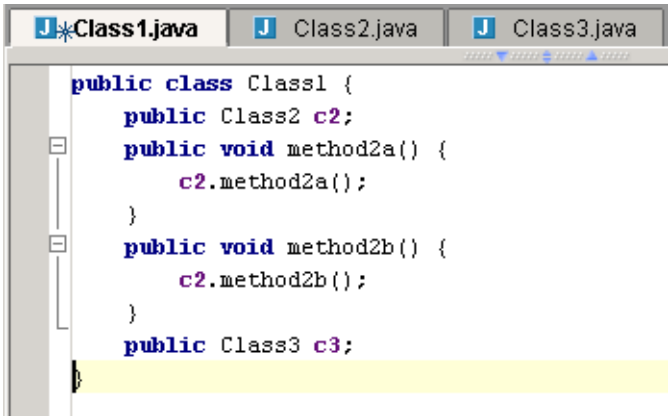


Figure 8.70. Delegates generated (362)

8.9. Comment

IDEA makes it easy to comment out code.

8.210. Create class **Class1**:

```
public class Class1 {
  String s1 = "line 1";
  String s2 = "line 2";
  String s3 = "line 3";
}
```

8.211. Place the cursor on a line.

8.212. Select **Code | Comment with line comment.**. The line is commented.

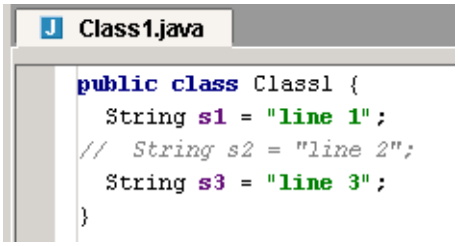


Figure 8.71. Line comment (356)

8.213. Select a portion of the code.

8.214. Select **Code | Comment with block comment.**. The block is commented.

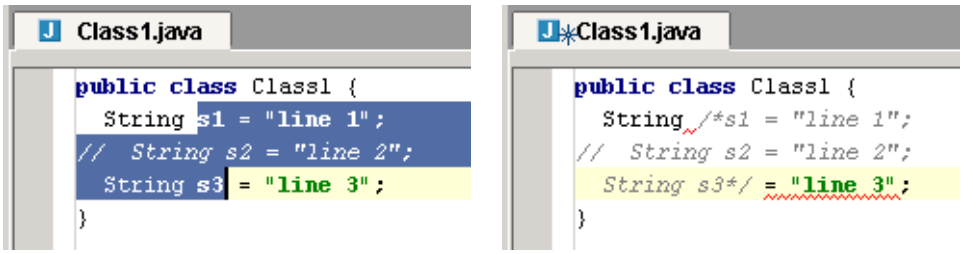


Figure 8.72. Block comment (355,354)

9. Code Refactoring X

20021018TTT last update.

Consult dima, valya.

Try to create example code that can be used thruout this entire chapter.

The following refactoring operations are supported:

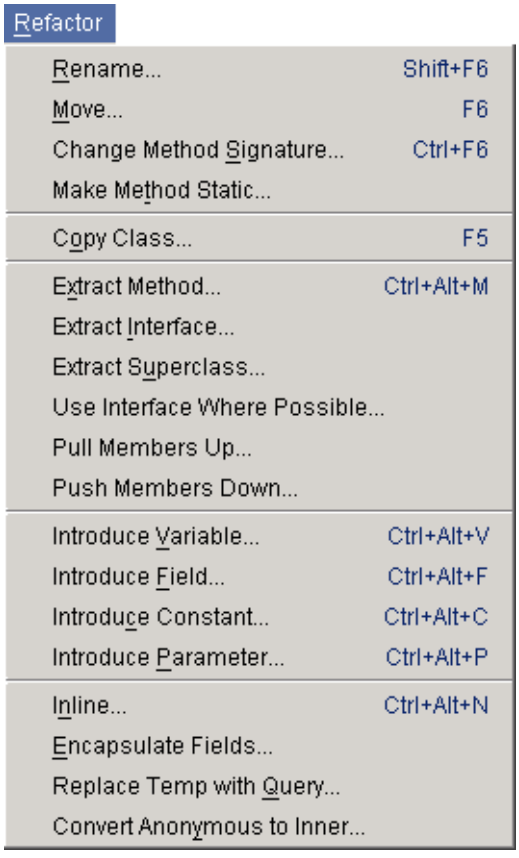


Figure 9.1. Refactoring menu items (414)

9.1. Migration XXX

9.1.1. Migrate

9.1.2. Create map

9.2. Rename

IDEA refactoring allows you rename a

9.2.1. Package

9.1. 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);
  }
}
```

9.2. Create class MyClass2:

```
package MyPackage2;
public class MyClass2 {
  public int method2(int param2) {
    int var2 = 2;
    return var2 + param2;
  }
}
```

9.3. Right-click on **MyPackage**.
9.4. Select **Refactor | Rename**. The dialog “Rename” appears.
9.5. For the new name enter **MyPackageNEW**.

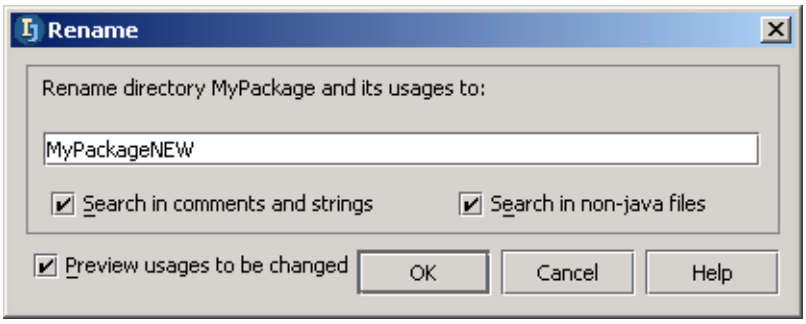


Figure 9.2. Dialog “Rename” (for package) (515)

9.6. Click **OK**. The panel “Find - Refactoring preview” appears.

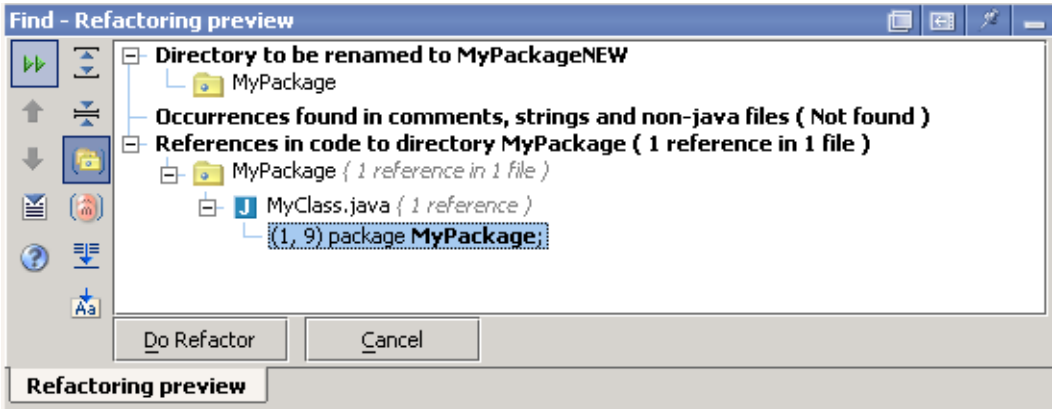


Figure 9.3. Panel “Find - Refactoring preview” (for package) (514)

9.7. Click on **Do Refactor**. The package is renamed.

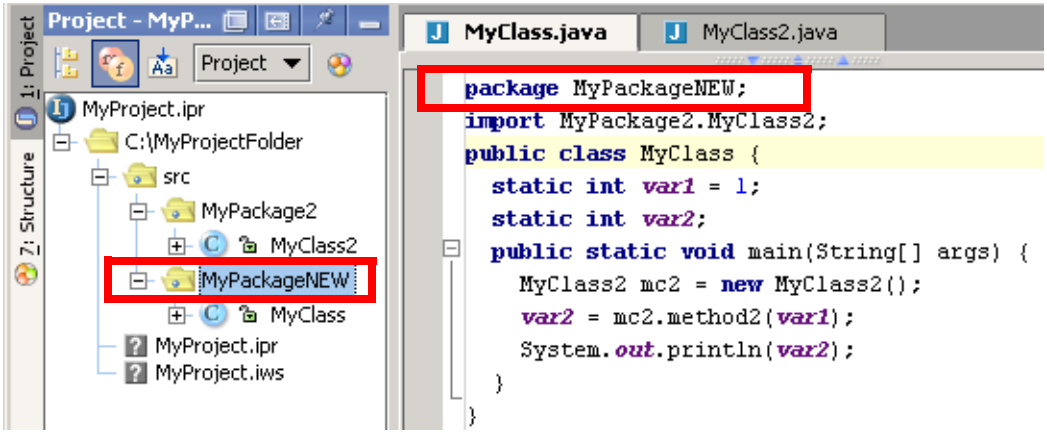


Figure 9.4. Renamed package (513)

9.2.2. Class

9.8. In the “Project” window: Right-click on **MyClass2**.
9.9. Select **Refactor | Rename**. The dialog “Rename” appears.
9.10. For the new name enter **MyClass2NEW**.

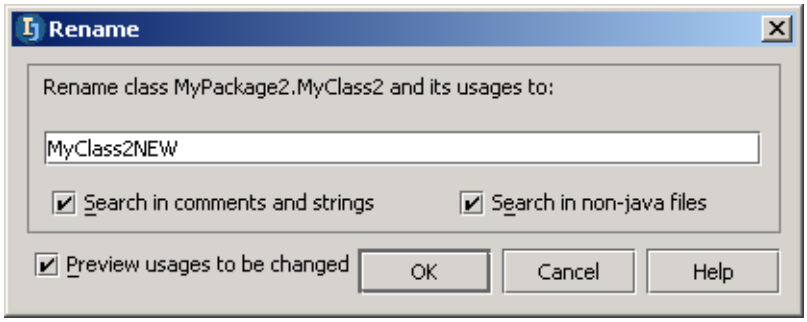


Figure 9.5. Dialog “Rename” (for class) (511)

9.11. Click **OK**. The panel “Find - Refactoring preview” appears.

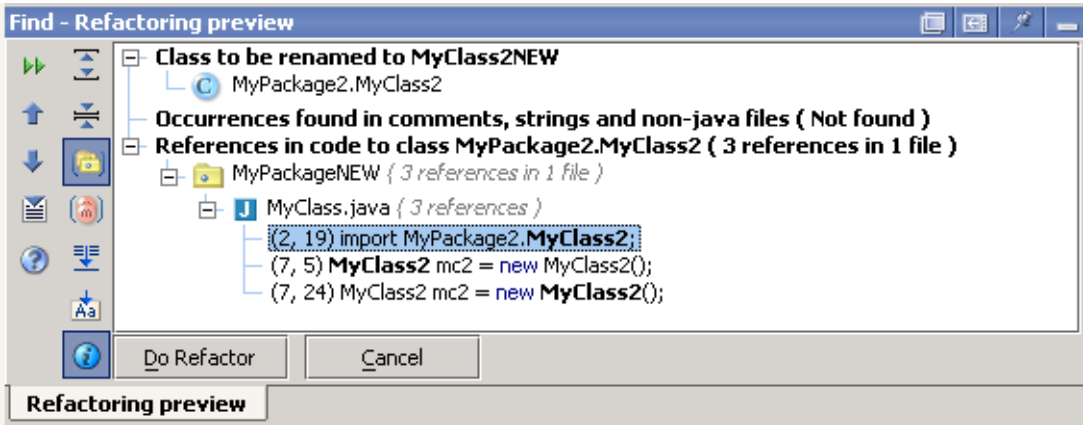


Figure 9.6. Panel “Find - Refactoring preview” (for class) (510)

9.12. Click on **Do Refactor**. The class is renamed.

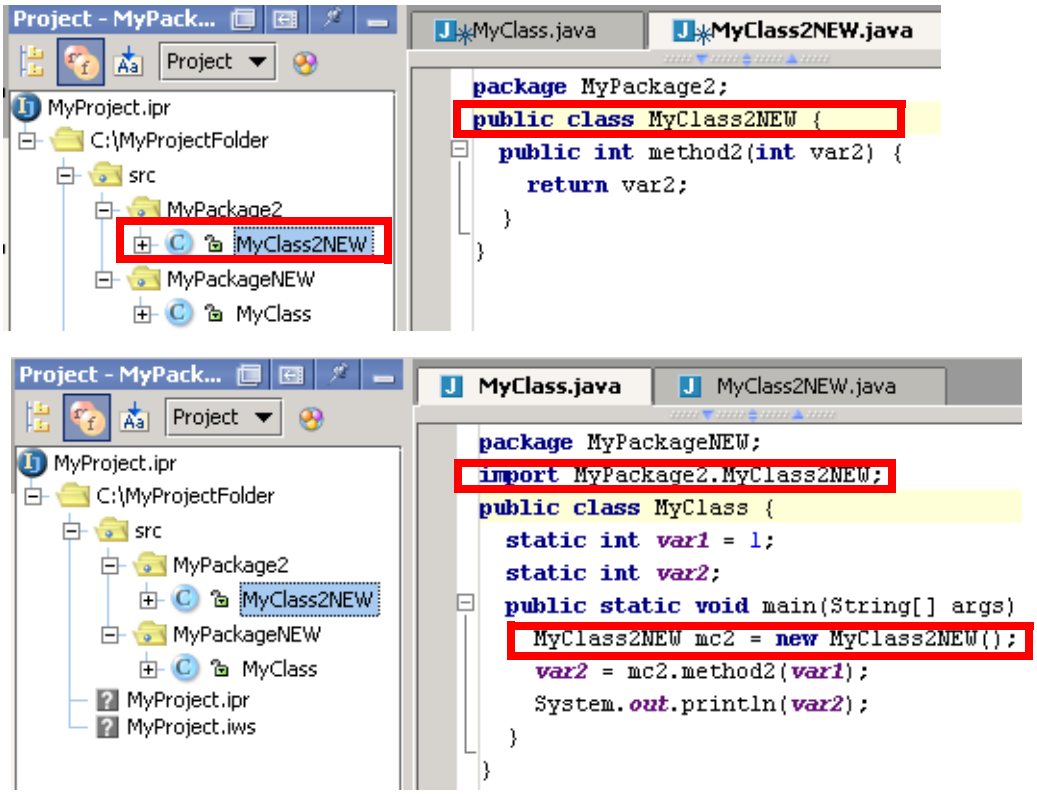


Figure 9.7. Renamed class (509,508)

9.2.3. Method

9.13. In MyClass2NEW: Place the cursor on the declaration of **method2**.
9.14. Right-click.
9.15. Select **Refactor | Rename**. The dialog “Rename” appears.
9.16. For the new name enter **method2NEW**.

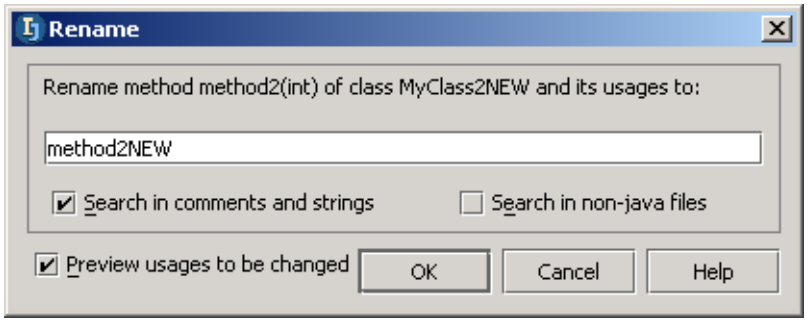


Figure 9.8. Dialog “Rename” (for method) (507)

9.17. Click **OK**. The panel “Find - Refactoring preview” appears.

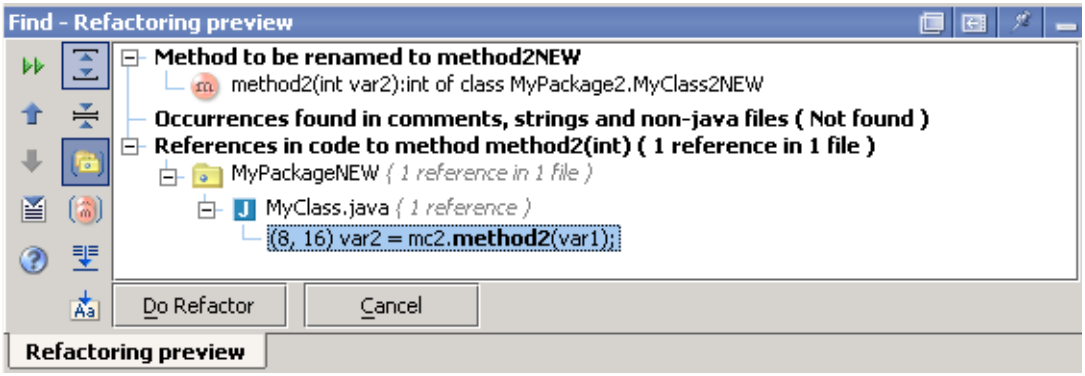


Figure 9.9. Panel “Find - Refactoring preview” (for method) (506)

9.18. Click on **Do Refactor**. The method is renamed.

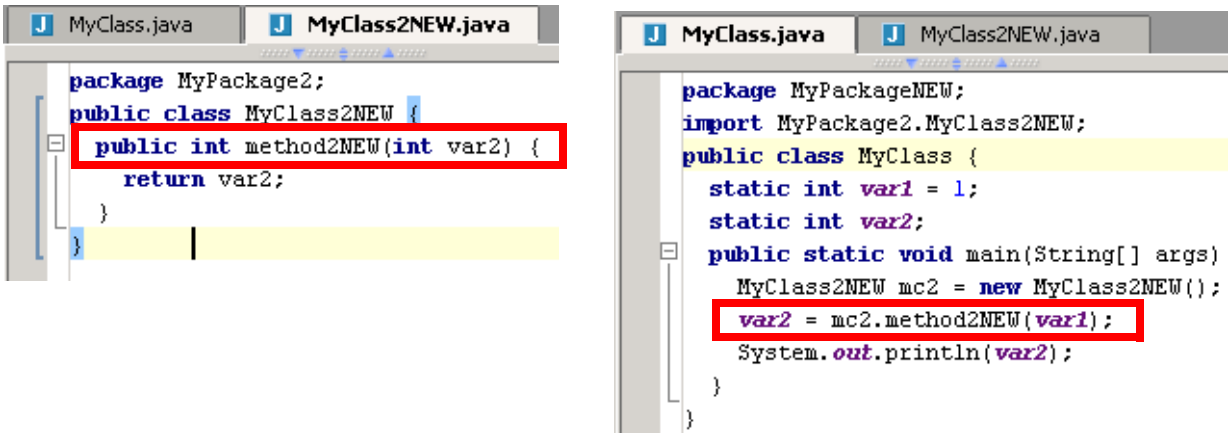


Figure 9.10. Renamed method (505,504)

9.2.4. Field

9.19. In the code for MyClass: Place the cursor on the declaration of **field1**.
9.20. Right-click.
9.21. Select **Refactor | Rename**. The dialog “Rename” appears.
9.22. For the new name enter **field1NEW**.

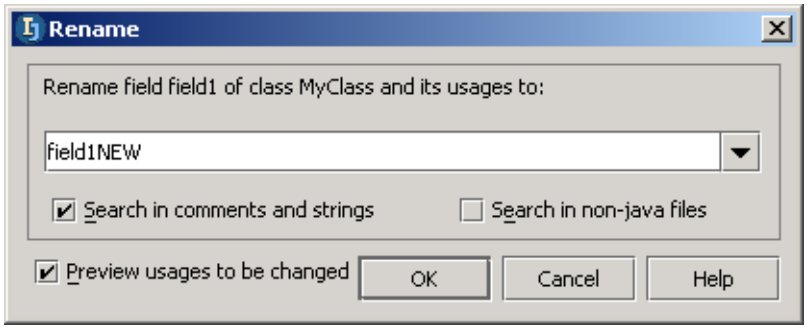


Figure 9.11. Dialog “Rename” (for field) (503)

9.23. Click **OK**. The panel “Find - Refactoring preview” appears.

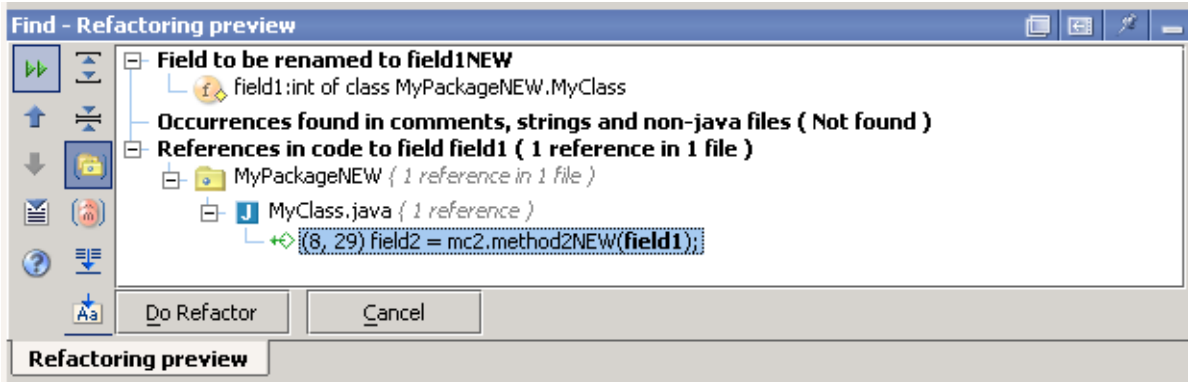


Figure 9.12. Panel “Find - Refactoring preview” (for field) (502)

9.24. Click on **Do Refactor**. The field is renamed.

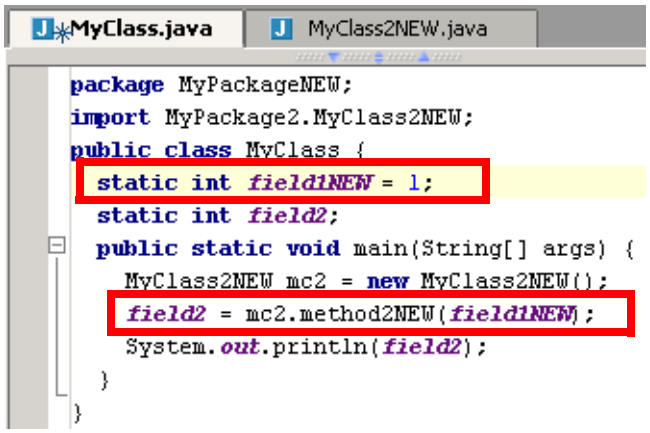


Figure 9.13. Renamed field (501)

9.2.5. Variable

9.25. In the code for MyClass2NEW: Place the cursor on the declaration of **var2**.
9.26. Right-click.
9.27. Select **Refactor | Rename**. The dialog “Rename” appears.
9.28. For the new name enter **var2NEW**.

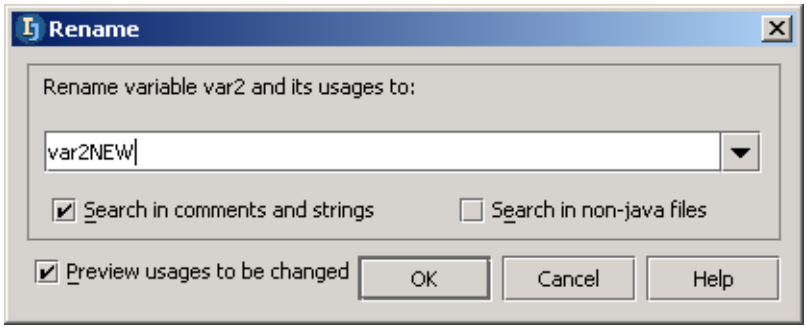


Figure 9.14. Dialog “Rename” (for variable) (918)

9.29. Click **OK**. The panel “Find - Refactoring preview” appears.

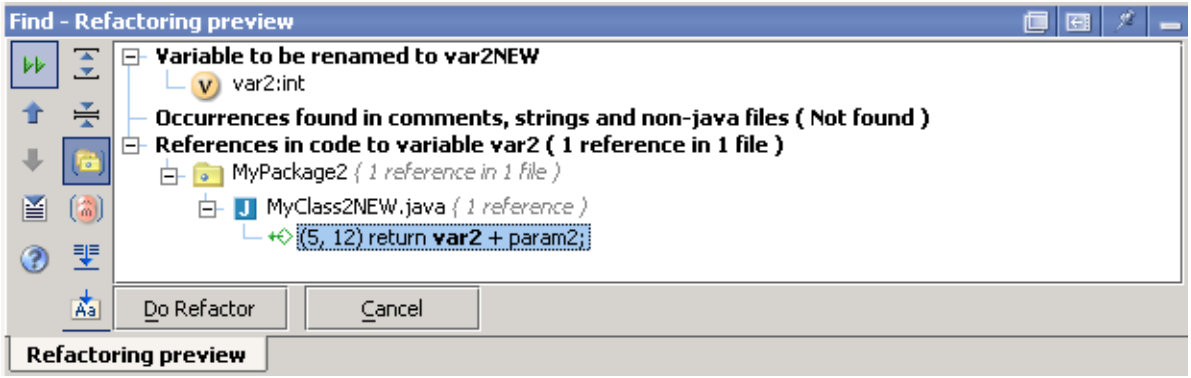


Figure 9.15. Panel “Find - Refactoring preview” (for field) (919)

9.30. Click on **Do Refactor**. The variable is renamed.

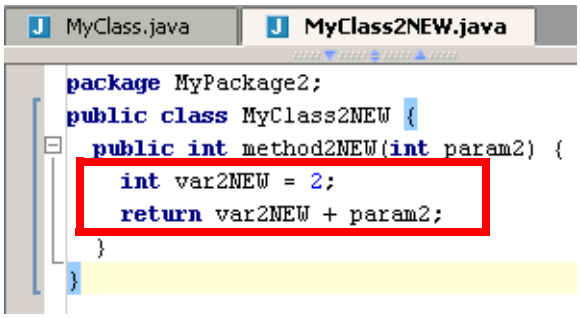


Figure 9.16. Renamed variable (920)

9.2.6. Parameter

9.31. In the code for MyClass2NEW: Place the cursor on **param2**.
9.32. Right-click.
9.33. Select **Refactor | Rename**. The dialog “Rename” appears.
9.34. For the new name enter **param2NEW**.

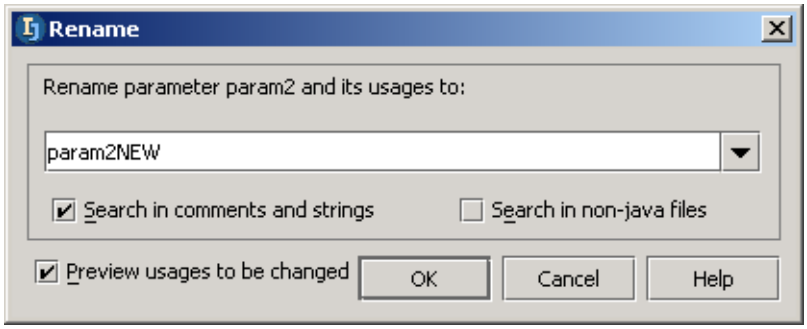


Figure 9.17. Dialog “Rename” (for parameter) (500)

9.35. Click **OK**. The panel “Find - Refactoring preview” appears.

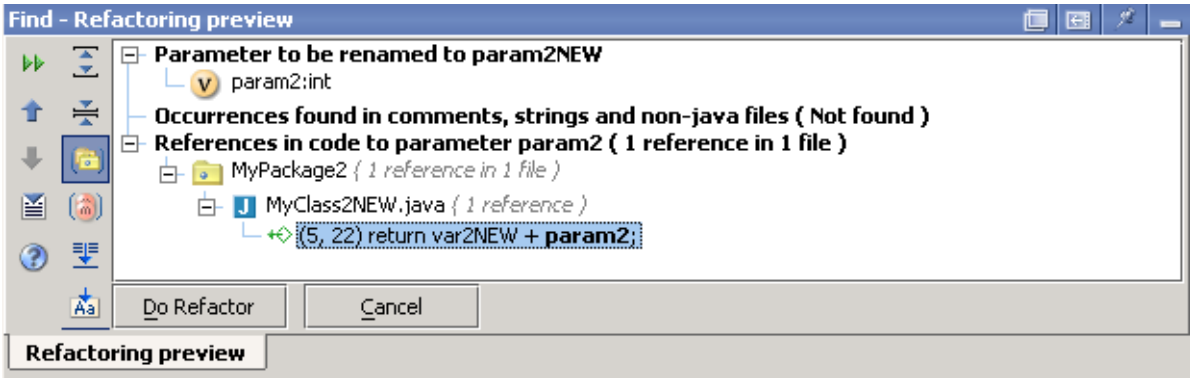


Figure 9.18. Panel “Find - Refactoring preview” (for parameter) (499)

9.36. Click on **Do Refactor**. The parameter is renamed.

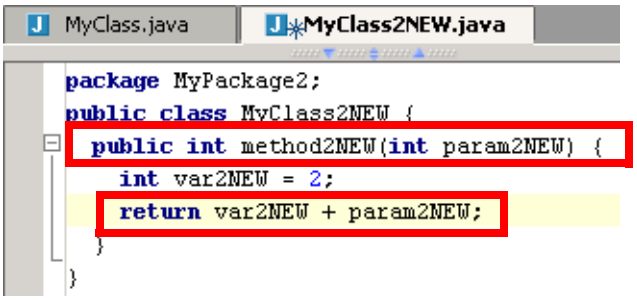


Figure 9.19. Renamed paramter (498)

9.3. Move

IDEA refactoring allows you move a

9.3.1. Package

9.37. Create class **MyClass**:

```
package MyPackage;
public class MyClass {}
```

9.38. Create class **MyClass2**:

```
package MyPackage2;
public class MyClass2 {
  public static void myMethod2A() {}
  public void myMethod2B() {
    myMethod2A();
  }
  public class MyInnerClass2 {
    public void myInnerClass2Method() {
      myMethod2A();
    }
  }
}
```

9.39. Right-click on **MyPackage2**.
9.40. Select **Refactor | Move**. The dialog “Move” appears.
9.41. For “To package:” enter **MyPackage**.

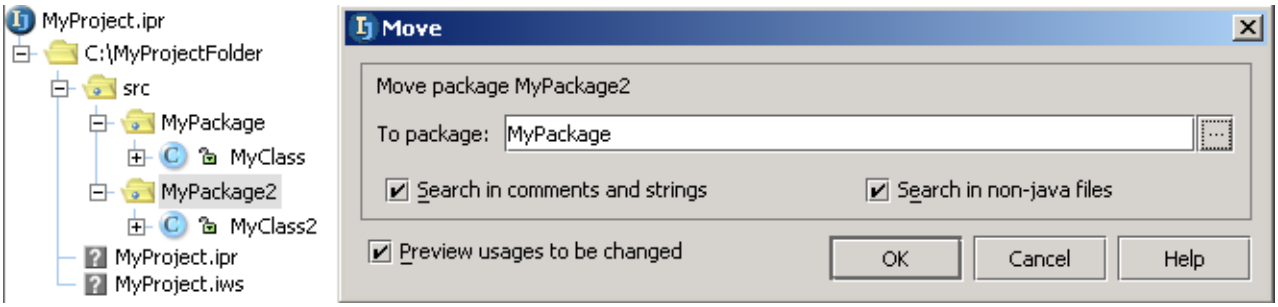


Figure 9.20. Move package dialog (380)

9.42. Click **OK**. The “Find - Refactoring preview” dialog appears.

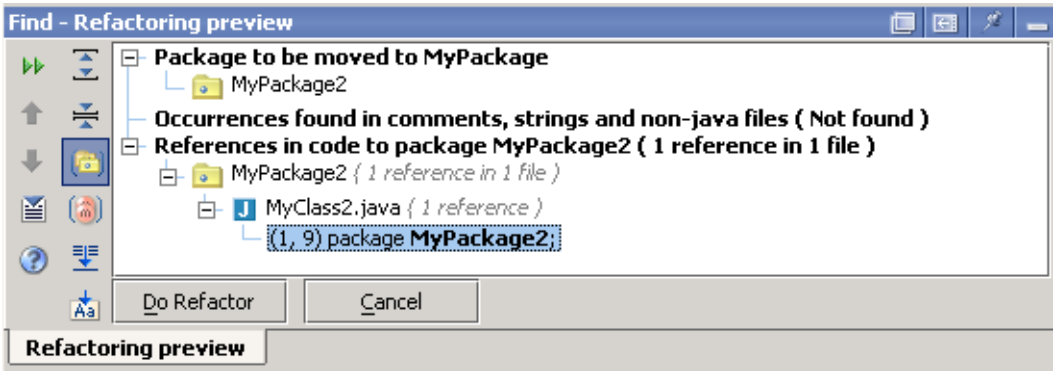


Figure 9.21. Find Refactoring preview (379)

9.43. Click **Do Refactor**. The package is moved.

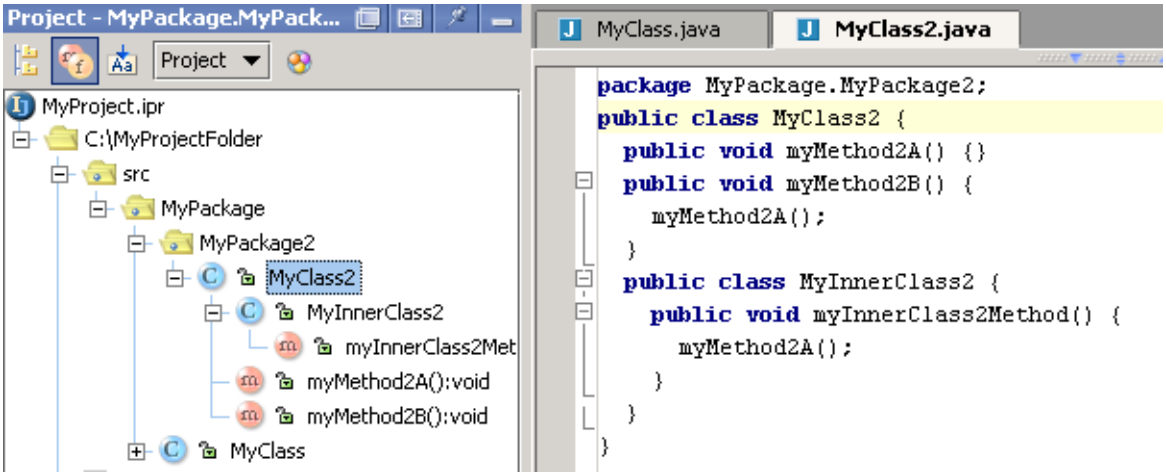


Figure 9.22. Package moved (378)

9.3.2. Class

9.44. Right-click on **MyClass2**.
9.45. Select **Refactor | Move**. The dialog “Move” appears.
9.46. For “To package” select **MyPackage**.

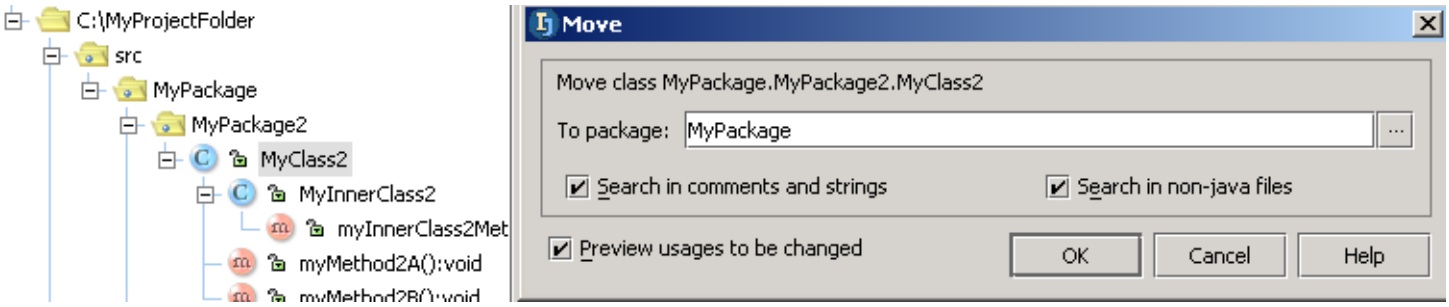


Figure 9.23. Move class dialog (377)

9.47. Click **OK**. The “Find - Refactoring preview” dialog appears.

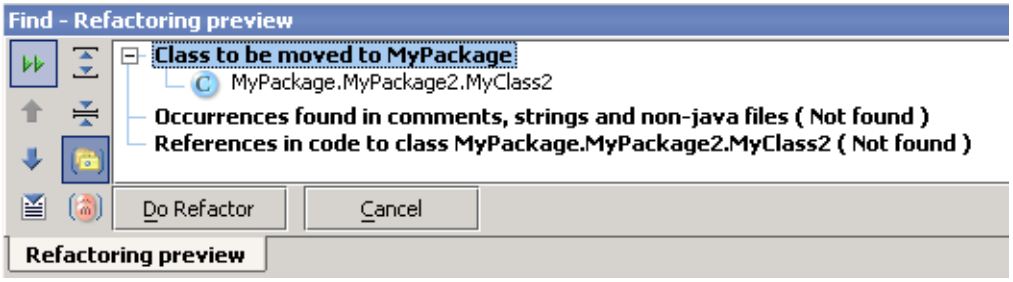


Figure 9.24. Find Refactoring preview (376)

9.48. Click **Do Refactor**. The class is moved.

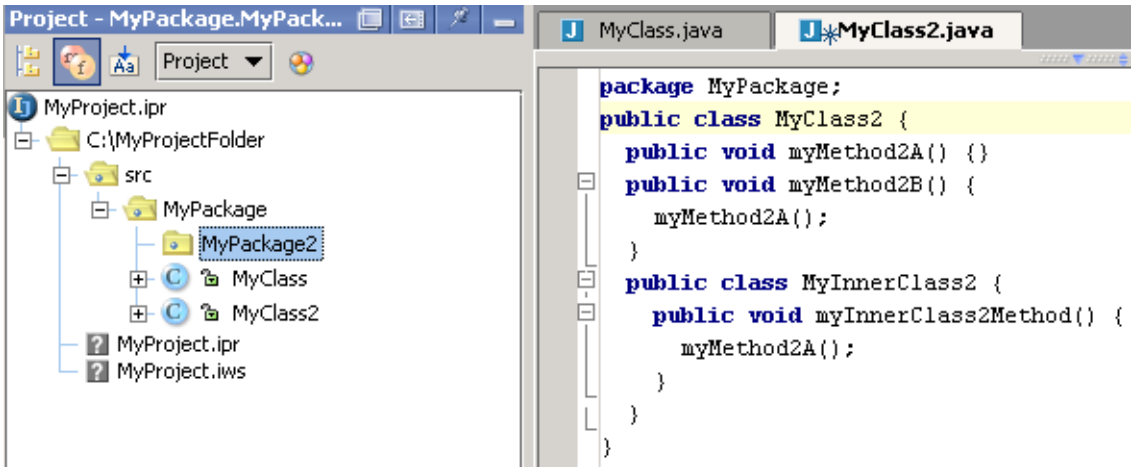


Figure 9.25. Class moved (375)

9.3.3. Members

9.49. Right-click on **myMethod2A**.
9.50. Selected **Refactor | Move**. The dialog “Move members” appears.
9.51. For “To” select **MyPackage.MyClass**.

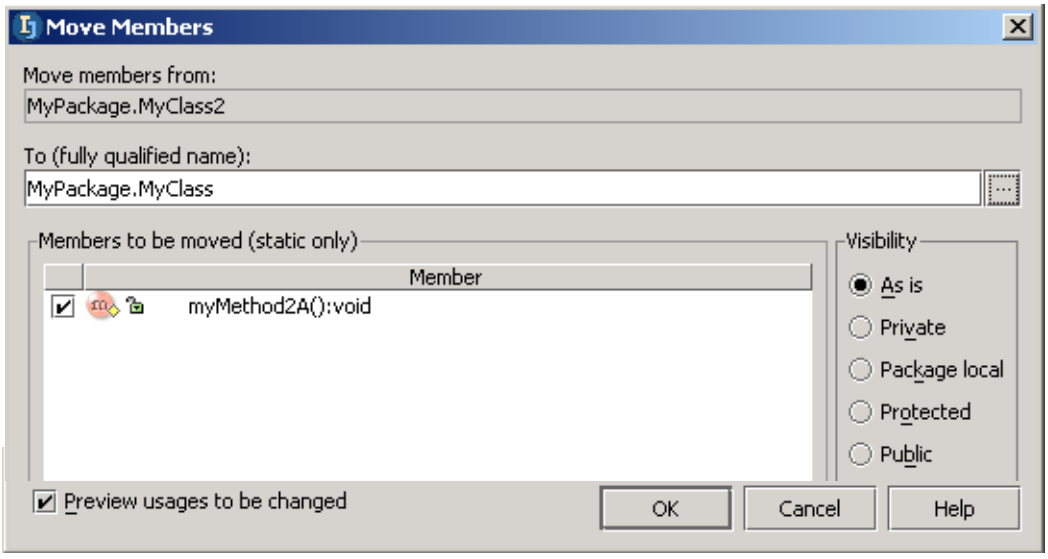


Figure 9.26. Dialog Move Members (374,921)

9.52. Click **OK**. The tool “Refactoring preview” appears.

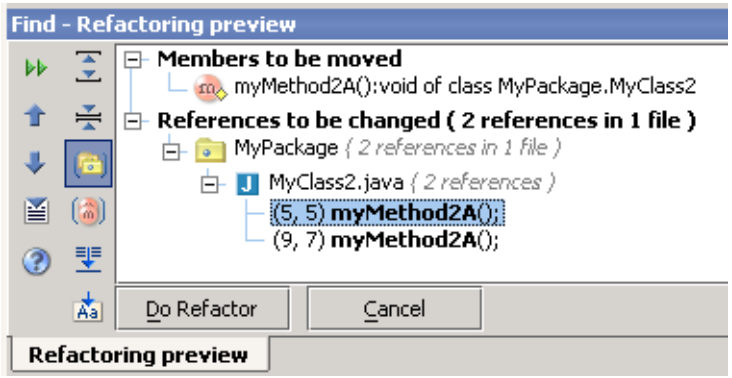


Figure 9.27. Find - Refactoring preview (373)

9.53. Click **Do Refactor**. The class is moved.

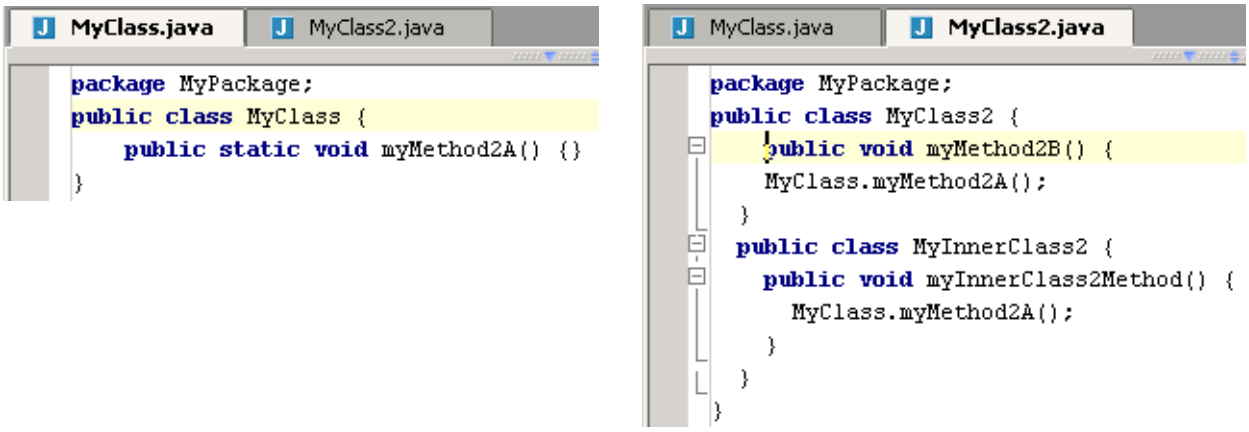


Figure 9.28. Moved class (372,371)

9.3.4. Inner to upper level

9.54. Place the cursor on **MyInnerClass2**.
9.55. Selected **Refactor | Move**. The dialog “Move inner to upper” appears.

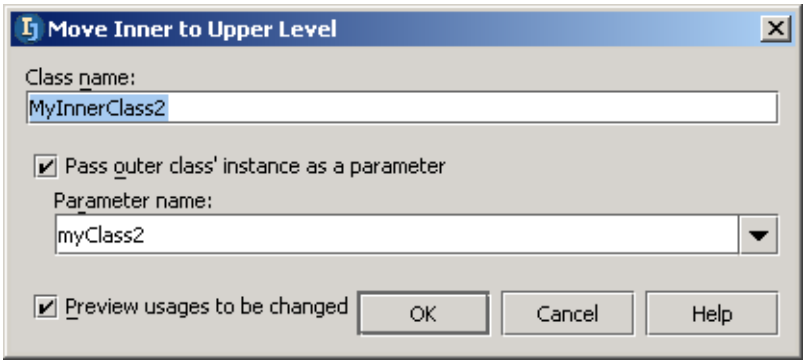


Figure 9.29. Move inner to upper dialog (370)

9.56. Click **OK**. The tool “Refactoring preview” appears.

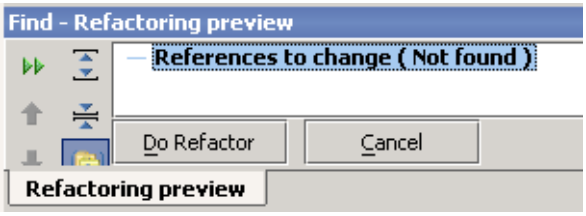


Figure 9.30. Find - Refactoring preview (369)

9.57. Click **Do Refactor**. The class is moved.

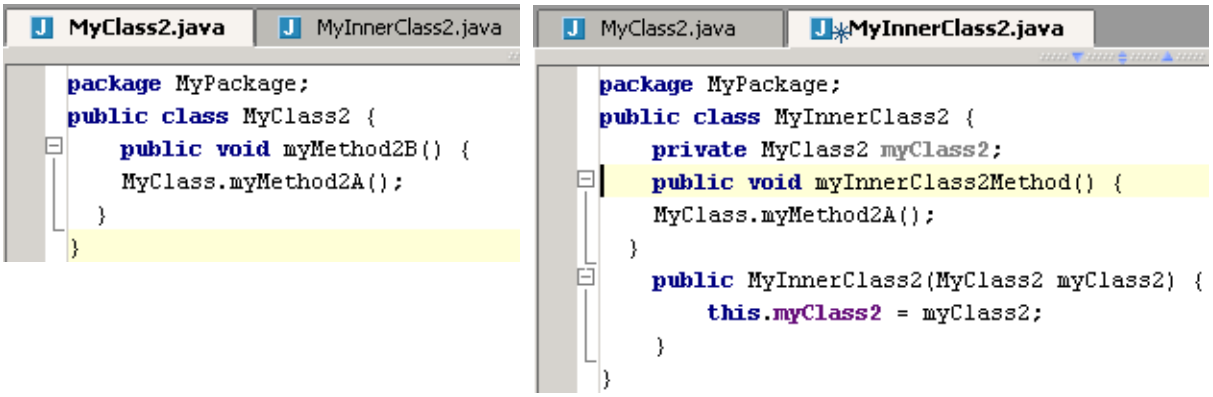


Figure 9.31. Moved inner class (368,367)

9.4. Change method signature X

IDEA make it easy to change the signature of a method globally.

In this section you will

9.4.1. Add parameter

9.58. Create class **ChangeMethodSignature**:

```
import java.util.*;
public class ChangeMethodSignature {
      public static void main(String[] args) {
      }
    public class Test {
        Class1 c1 = new Class1(111, "name1");
        Class1 c2 = new Class1(222, "name2");
    }
      public class Class1 {
        private String cName;
        private int cNumber;
        public Class1(int cNumber, String cName) {
          this.cName = cName;
          this.cNumber = cNumber;
        }
   }
}
```

9.59. Place the caret with “Class1” after “new”.

9.60. Select **Refactor | Change method signature**. The dialog “Change method signature” appears.

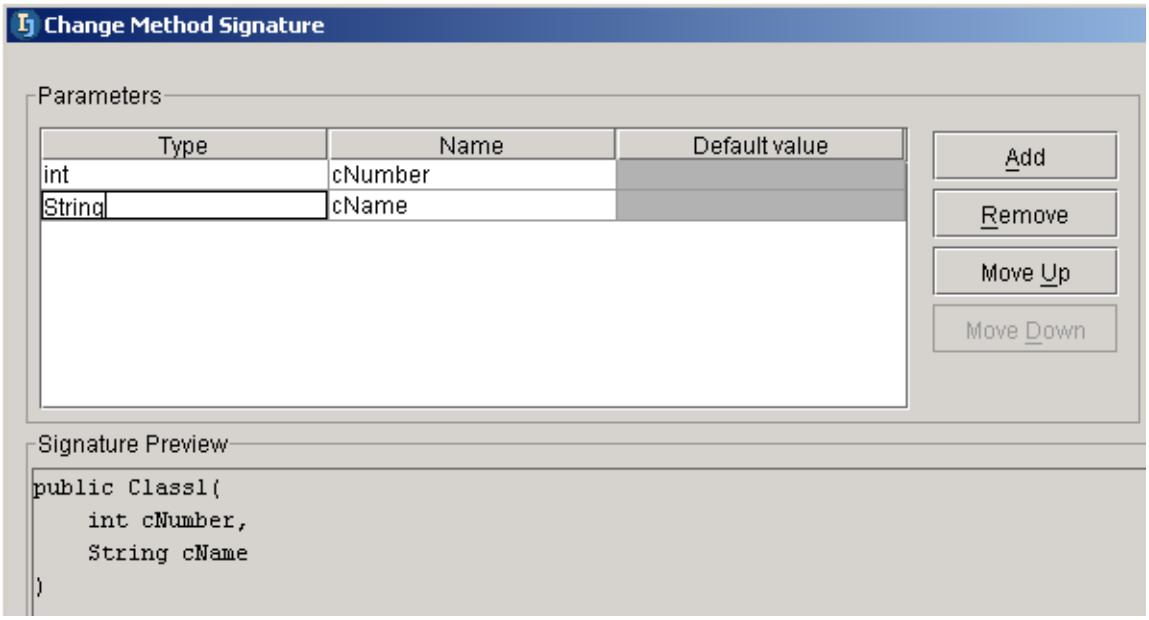


Figure 9.32. Dialog “Change method signature” (413)

9.61. Click **Add**. A new row appears at the end of the parameter list. The text box in row “Type” is selected.

9.62. For “Type” enter **Boolean**.
9.63. For “Name” enter **cYesNo**.
9.64. For “Default value” enter **true**.
9.65. Check checkbox **Preview usages to be changed**.

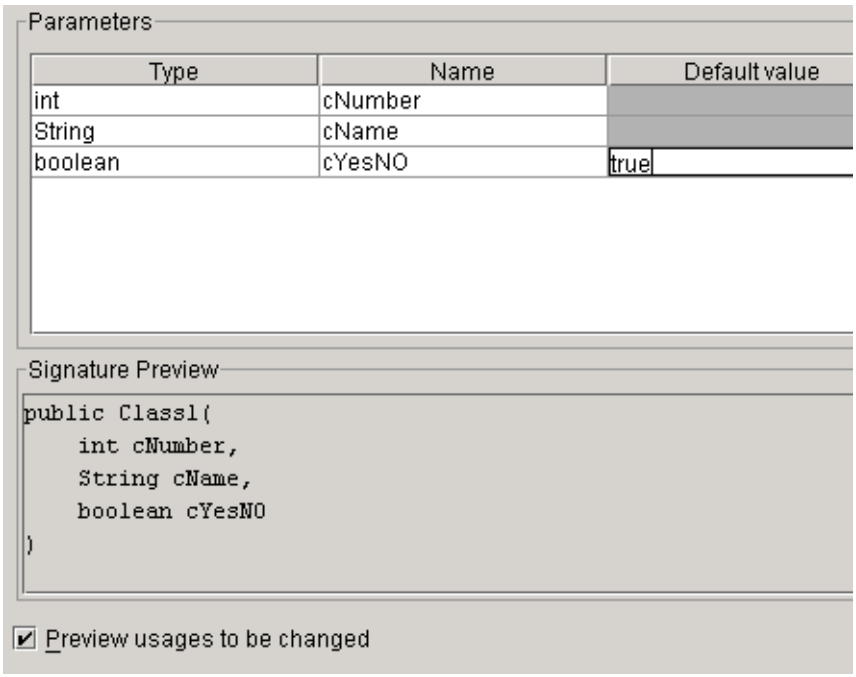


Figure 9.33. Add parameter (412)

9.66. Click **OK**. The tool “Refactoring preview” appears.

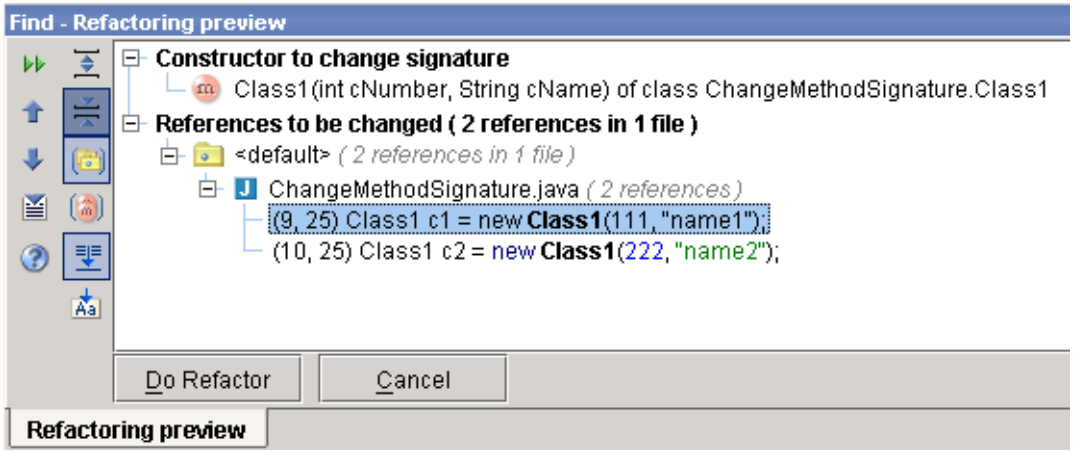


Figure 9.34. Find - Refactoring preview (411)

9.67. Click **Do Refactor**. The class is refactored.

```
public class ChangeMethodSignature {
      public static void main(String[] args) {
      }
    public class Test {
        Classl c1 = new Classl(111, "name1", true);
        Classl c2 = new Classl(222, "name2", true);
    }
      public class Classl {
        private String cName;
        private int cNumber;
        public Classl(int cNumber, String cName, boolean cYesNO) {
          this.cName = cName;
          this.cNumber = cNumber;
        }
```

Figure 9.35. Parameter added (410)

9.4.2. Move parameter

9.68. Reopen “Change method signature” for Class1.
9.69. Select **cYesNo**.
9.70. Click **MoveUp** until the parameter is first.

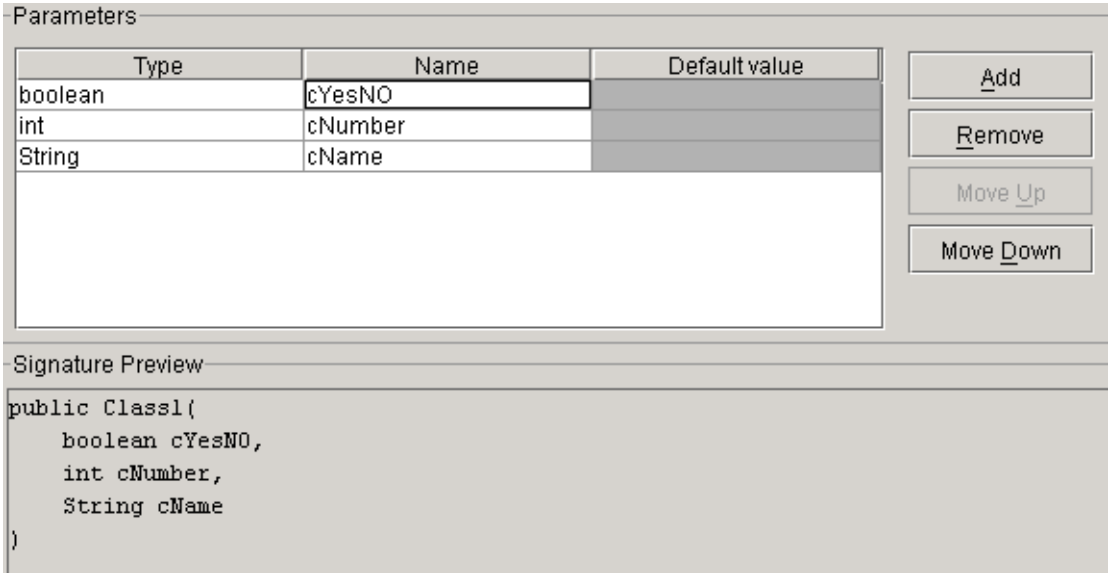


Figure 9.36. Parameter moved (409)

9.4.3. Change name

9.71. For the type “boolean” change the name to **cChar**.

9.4.4. Change type

9.72. For the type “boolean” change the type to **char**.

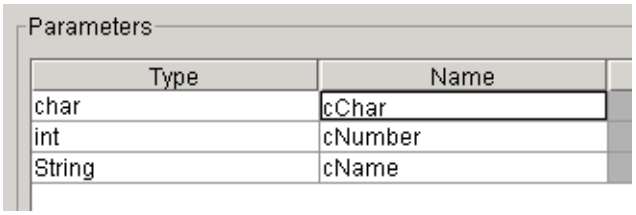


Figure 9.37. Parameter name and type change (408)

9.73. Refactor the class.

20020925TTT this did not quite work... i changed the char initial value myself.

```
public class ChangeMethodSignature {
      public static void main(String[] args) {
      }
    public class Test {
        Classl c1 = new Classl('c', 111, "name1");
        Classl c2 = new Classl('c', 222, "name2");
    }
      public class Classl {
        private String cName;
        private int cNumber;
        public Classl(char cChar, int cNumber, String cName) {
          this.cName = cName;
          this.cNumber = cNumber;
        }
    }
}
```

Figure 9.38. Parameter name and type changed (407)

9.5. Copy class

9.74. Select **MyClass**.
9.75. Select **Refactor | Copy**. The dialog “Copy Class” appears.
9.76. For “Name” enter **MyClassCopy**.

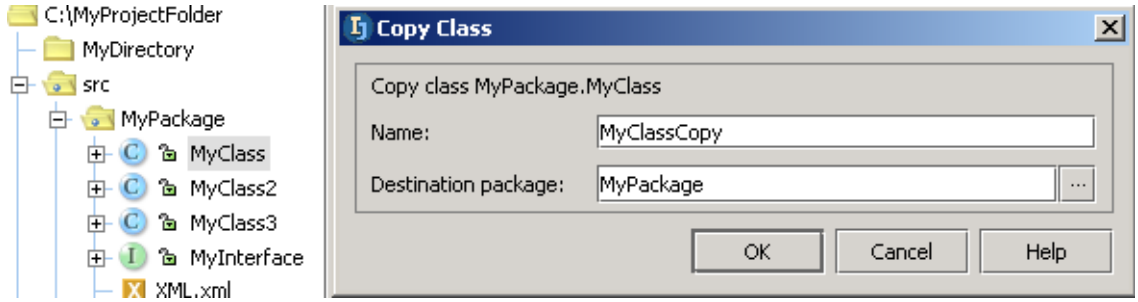


Figure 9.39. Dialog Copy class (890)

9.77. Click **OK**. The class is copied.

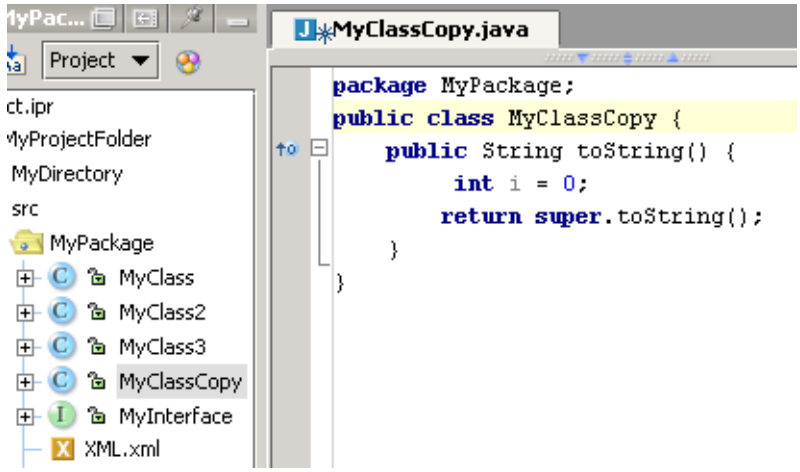


Figure 9.40. Copied class (891)

9.6. Extract

IDEA allows you to extract the following:

9.6.1. Method

9.78. Create class **ExtractMethod**:

```
public class ExtractMethod {
    public void doSomething() {
        System.out.println("Hello");
    }
}
```

9.79. Place the cursor on the “System.out...” line.
9.80. Selected **Refactor | Extract method**. The dialog “Extract method” appears.
9.81. For “Name” enter **extractedMethod**.

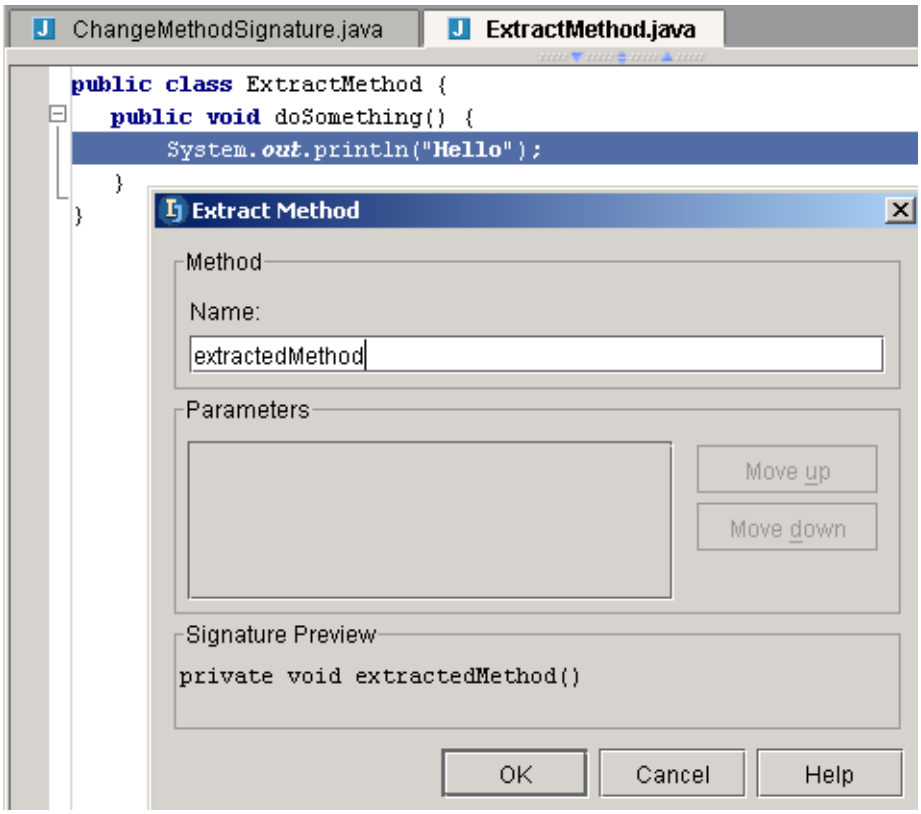


Figure 9.41. Method to extract (406)

9.82. Click **OK**. The method is extracted.

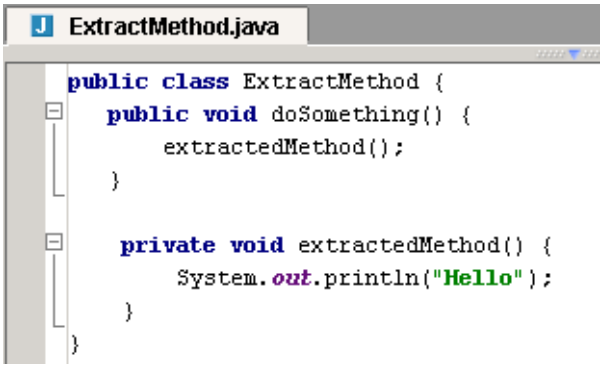


Figure 9.42. Extracted method (405)

9.6.2. Interface

9.83. Select **Refactor | Extract interface**. The “Extract Interface” dialog appears.
9.84. For “Interrface name” enter **ExtractedInterface**.
9.85. Check the checkbox for **doSomething():void**.

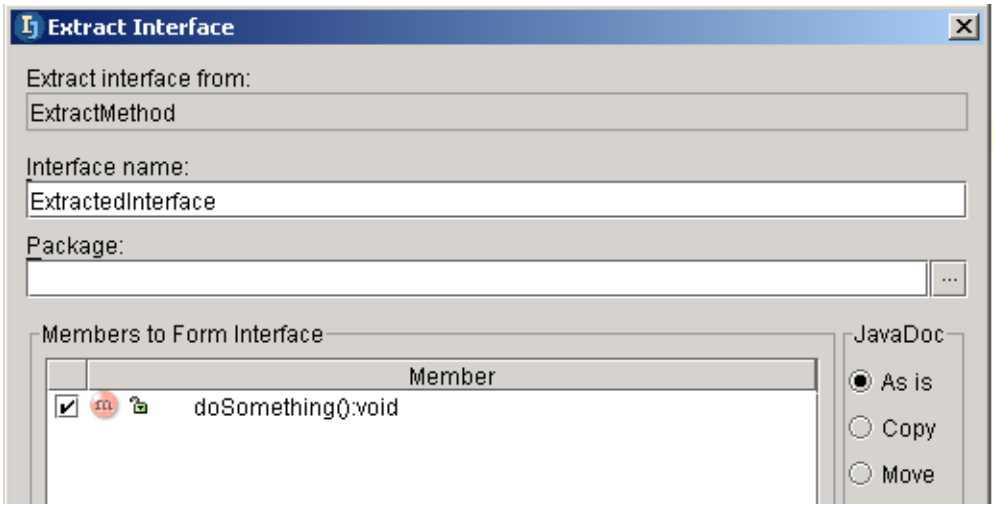


Figure 9.43. Extract interface (404)

9.86. Click **OK**. The “Choose destination directory” dialog appears.

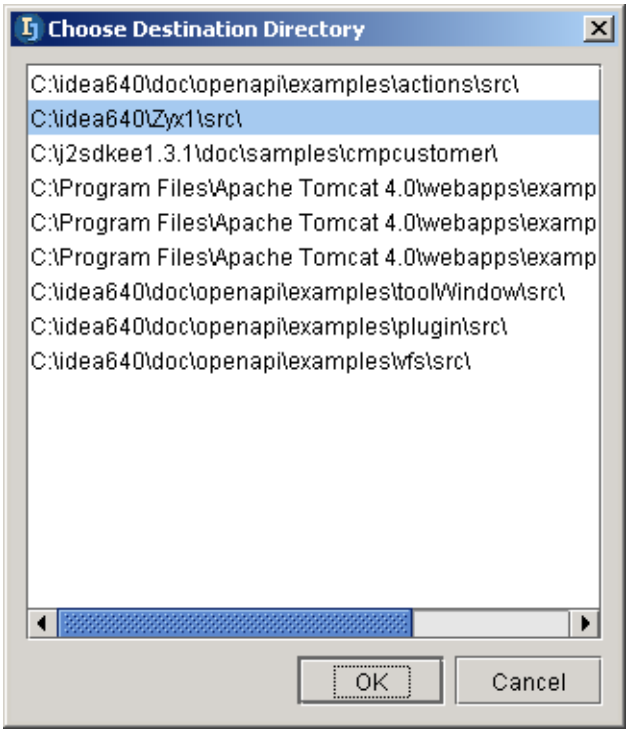


Figure 9.44. Choose destination directory (403)

9.87. Click **OK**. A question dialog appears.

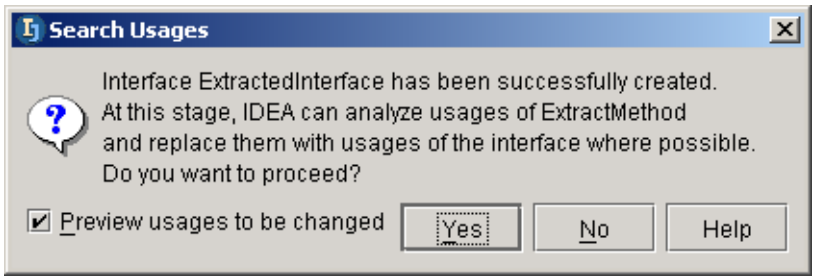


Figure 9.45. Proceed question (402)

9.88. Click **Yes**. The interface is extracted.

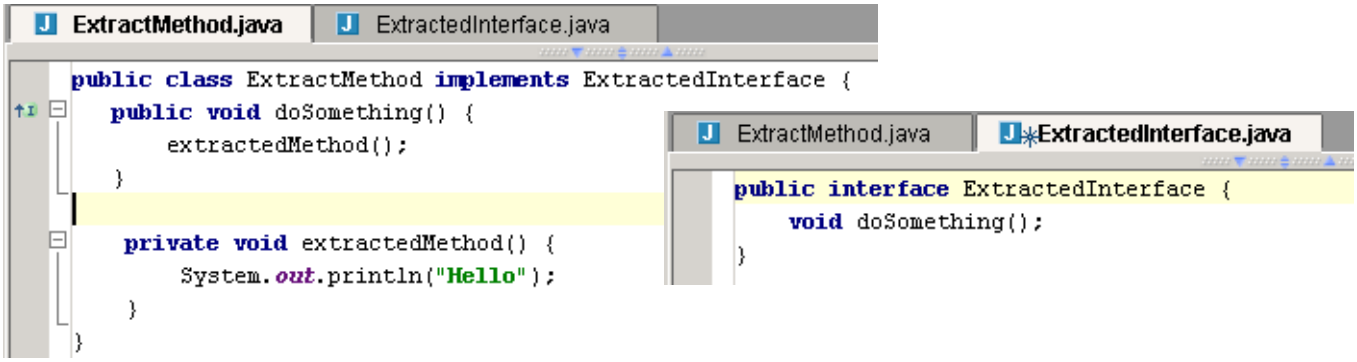


Figure 9.46. Extracted interface (401,400)

9.6.3. Superclass

9.89. Select **Refactor | Extract superclass**. The “Extract Superclass” dialog appears.

9.90. For “Superclass name” enter **ExtractedSuperClass**.

9.91. Check the checkboxes in “Member to Form Superclass”.

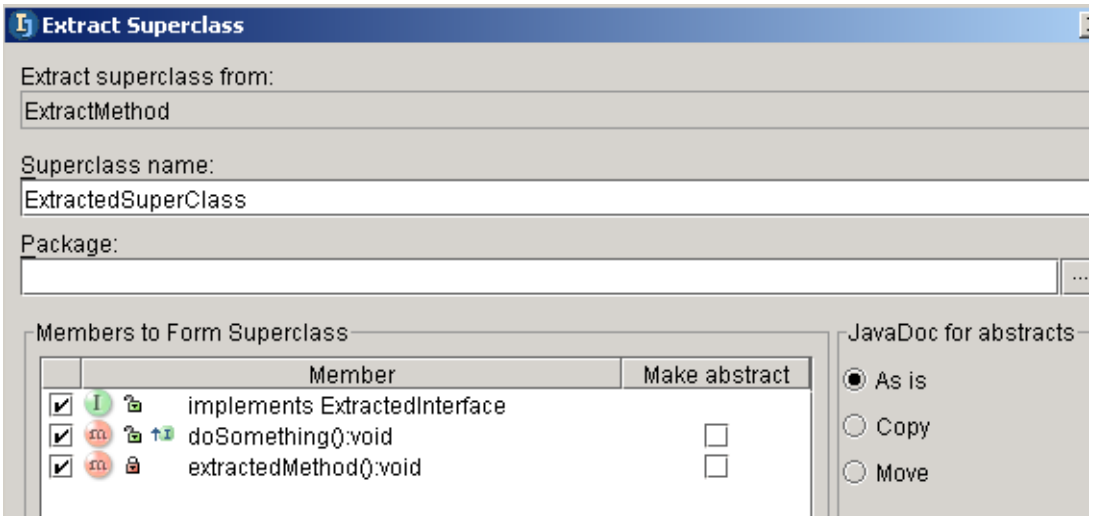


Figure 9.47. Extract superclass (399)

9.92. Click **OK**. The “Choose destination directory” dialog appears.

9.93. Click **OK**. A question dialog appears.

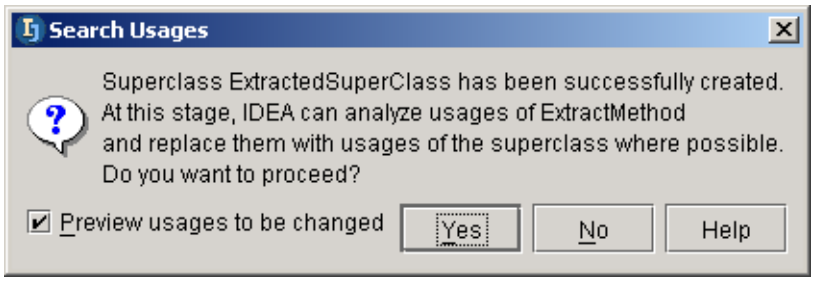


Figure 9.48. Proceed question (398)

9.94. Click **Yes**. The superclass is extracted.

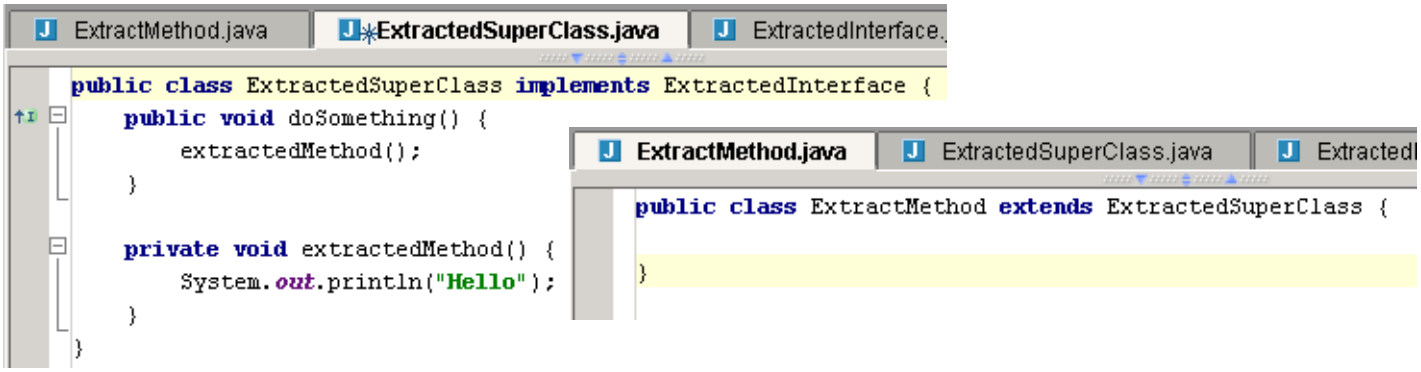


Figure 9.49. Extracted superclass (397,396)

9.7. Use interface where possible

9.95.

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

9.96.

```
package MyPackage;
public class MyClass implements MyInterface {
     public void iMethod() {}
}
```

9.97.

```
package MyPackage;
public class MyClass2 extends MyClass {
    public void test (MyClass myClass) {
        myClass.iMethod();
    }
}
```

9.98. In MyClass editor: Select **Refactor | Use interfaces where possible...**.

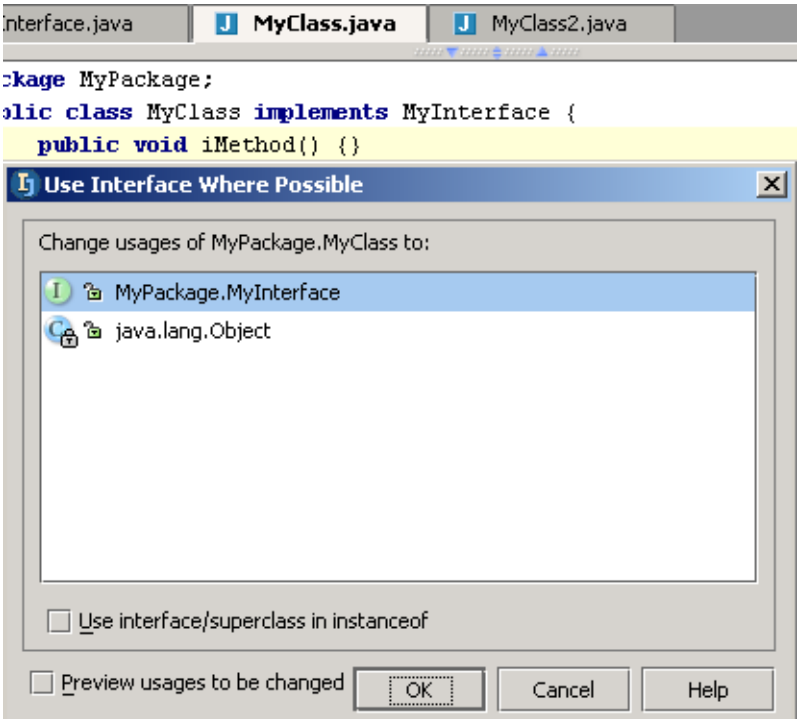


Figure 9.50. User interfaces where possible (892)

9.99. Check **Preview usages to be changed**.

9.100. Click **OK**. Refactoring preview appears.

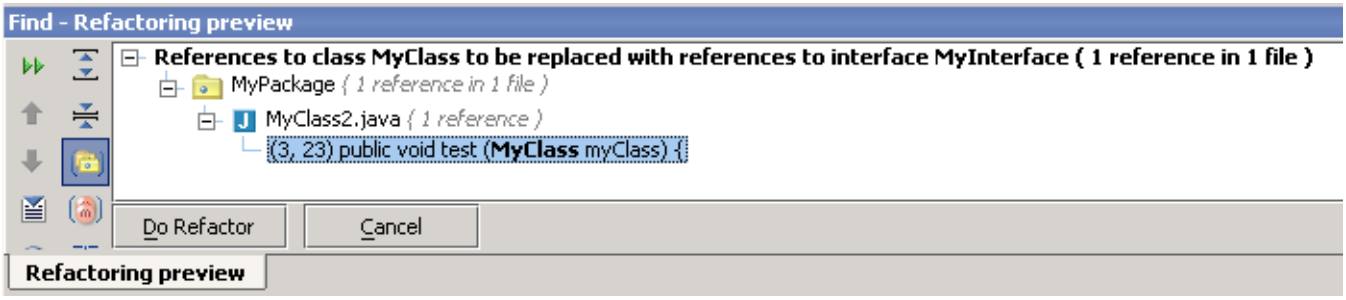


Figure 9.51. Refactoring preview (893)

9.101. Click **Do Refactor**. MyClass2 is refactored.

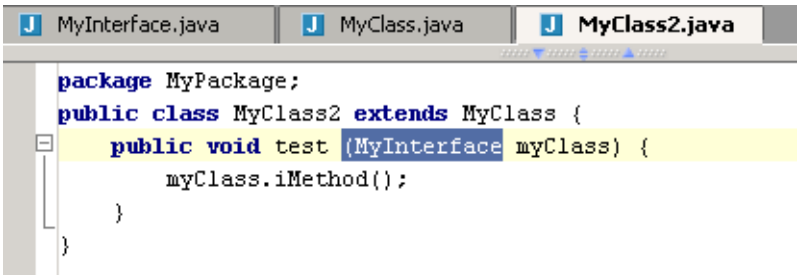


Figure 9.52. Refactored class (894)

9.8. Pull/push members

9.8.1. Pull Up

```
package MyPackage;
public class MyClass implements MyInterface {
    public void iMethod() {};
    public void iMethod2() {};
}
```

```
package MyPackage;
public interface MyInterface {
}
```

9.102. In MyClass editor: Select **Refactor | Pull members up...**.

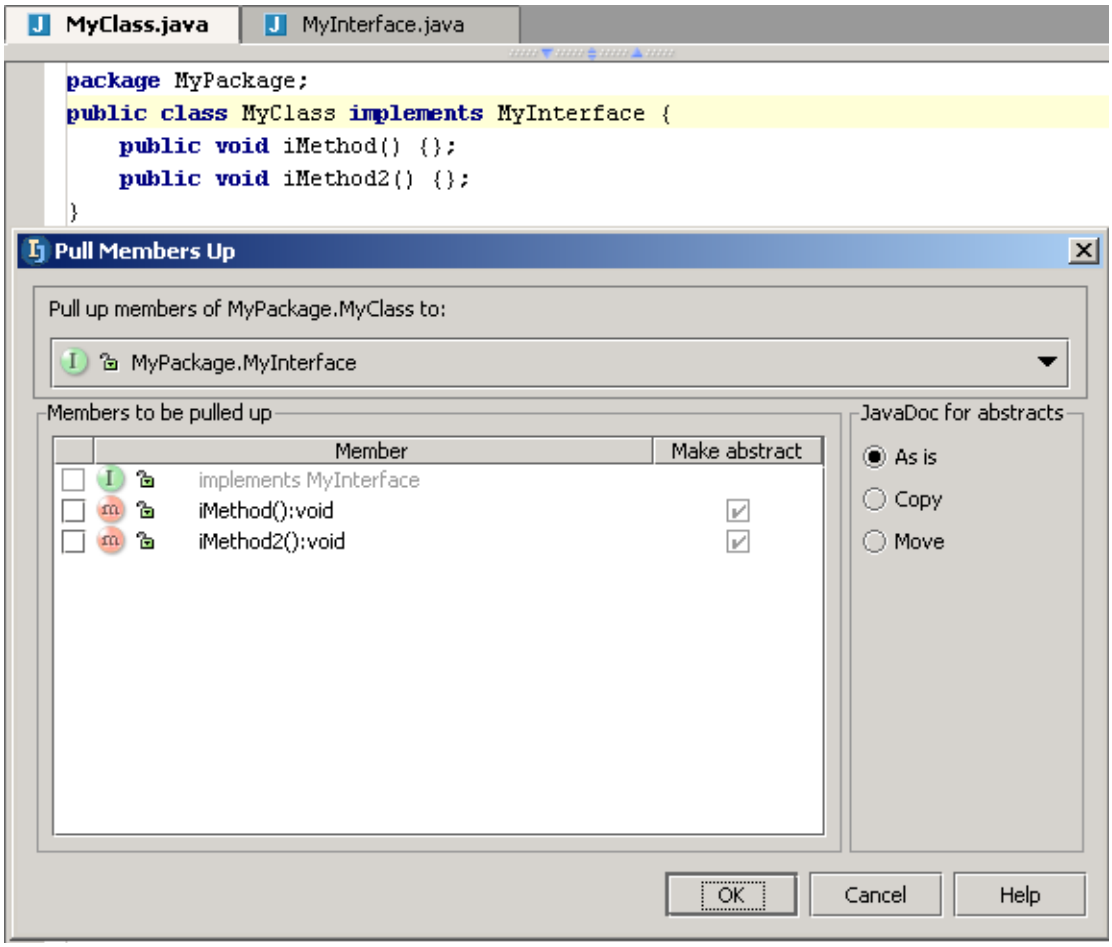
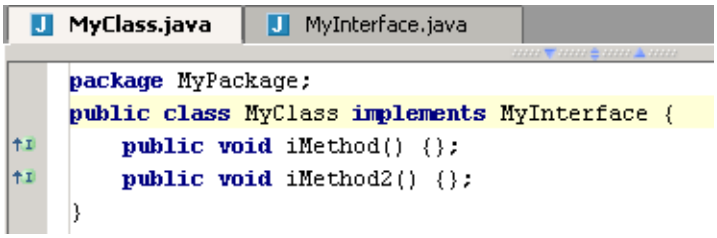


Figure 9.53. xxx (895)

9.103. Check **iMethod():void**.
9.104. Check **iMethod2():void**.

9.105. Click **OK**. MyInterface is refactored.



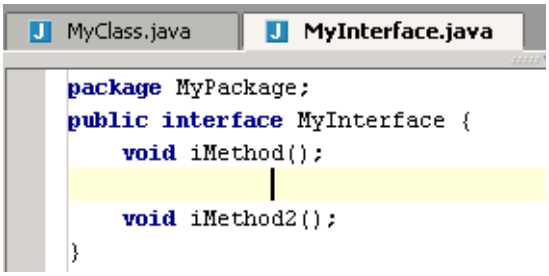


Figure 9.54. (896,897)

9.8.2. Push Down

9.106. Deleted methods.

```
package MyPackage;
public class MyClass implements MyInterface {}
```

9.107. In MyInterface editor: Select **Refactor | Push members down...**.

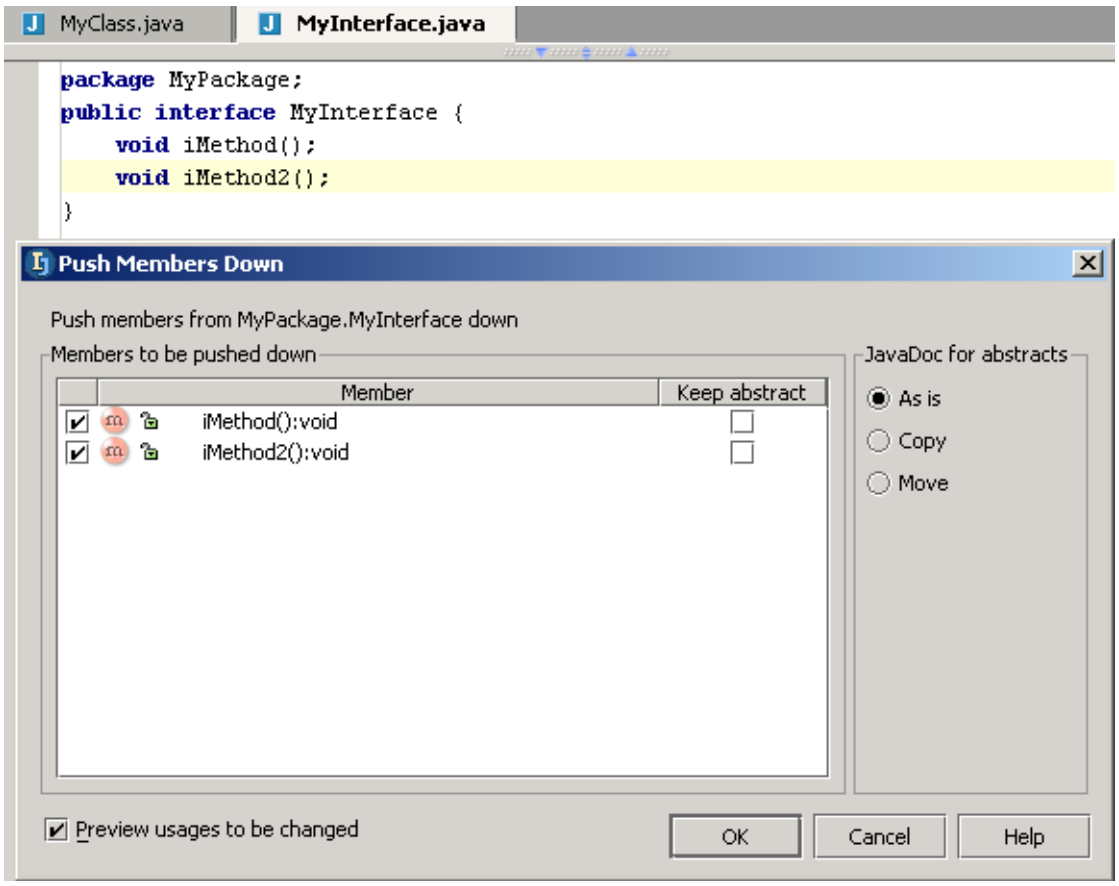


Figure 9.55. xxx (898)

9.108. Check **iMethod():void**.

9.109. Check **iMethod2():void**.

9.110. Check **Preview usages to be changed**.

9.111. Click **OK**. Preview.

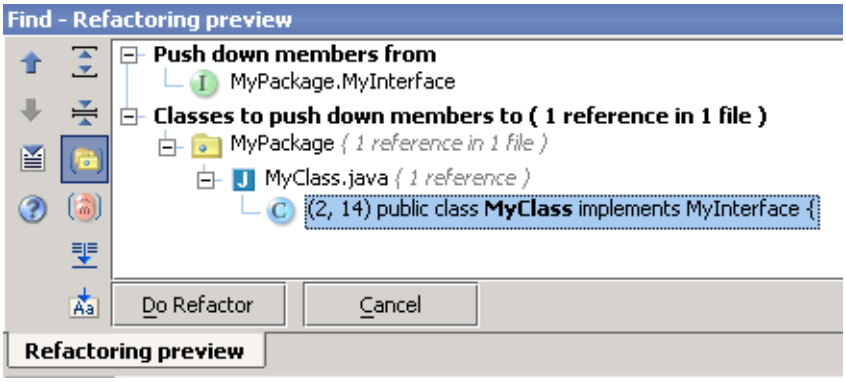


Figure 9.56. (899)

9.112. Click **OK**. MyClass and MyInterface are refactored.

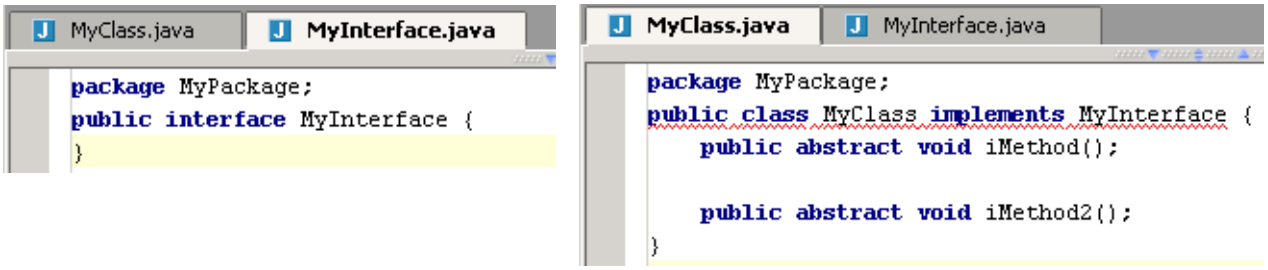


Figure 9.57. (900,901)

9.9. Introduce

9.9.1. Variable

Introduce a variable.

9.113. Create class **IntroduceVariable**:

```
public class IntroduceVariable {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

9.114. Select “**System.out**”.

9.115. Select **Refactor | introduce variable**. The dialog “Introduce variable” appears.

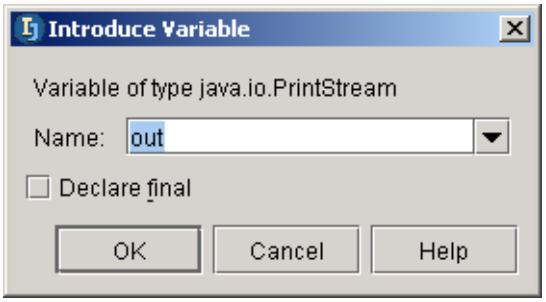


Figure 9.58. Introduce variable (382)

9.116. Click **OK**. The variable is introduced.

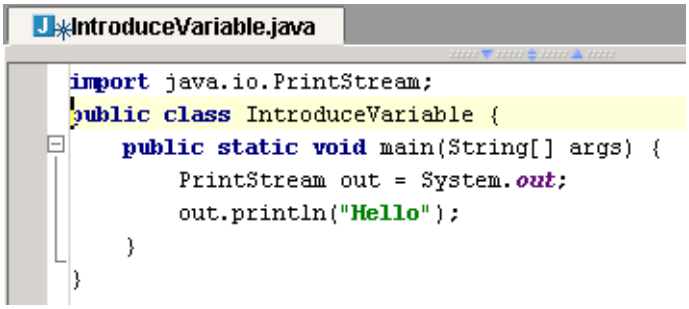


Figure 9.59. Variable introduced (381)

9.9.2. Field

Convert a local variable to a field.

9.117. Create class **IntroduceField**:

```
public class IntroduceField {
  public void aMethod() {
    String jb = new String("Button");
  }
}
```

9.118. Place the cursor on **jb**.

9.119. Select **Refactor | introduce field**.

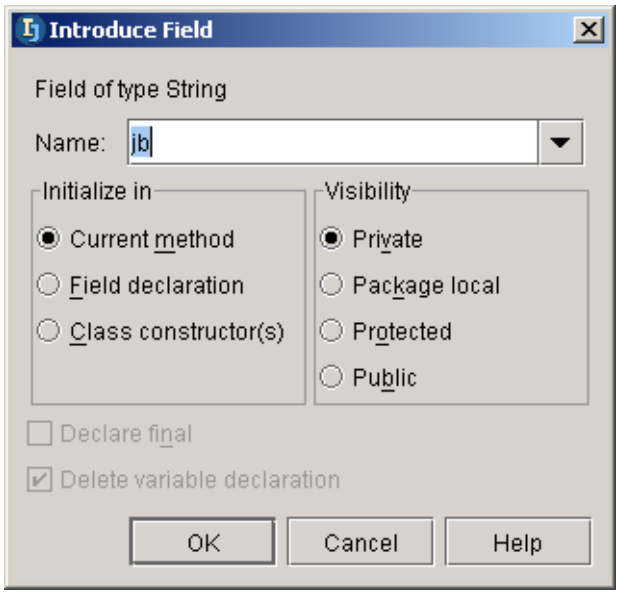


Figure 9.60. Introduce field (384)

9.120. Click **OK**. The field is introduced.

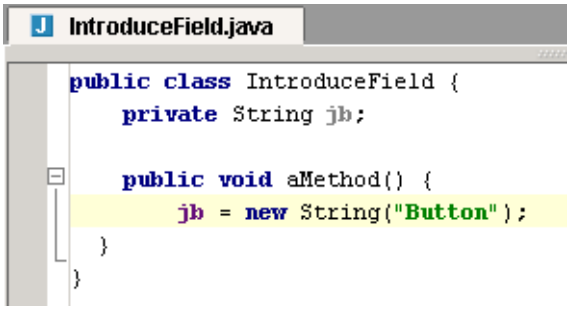


Figure 9.61. Field introduced (383)

9.9.3. Constant XXX

9.9.4. Parameter XXX

9.10. Inline

You can inline a

9.10.1. Variable

9.121. Create class **InlineVariable**:

```
public class InlineVariable {
public int aMethod() {
int aVar = 1;
return aVar;
  }
}
```

9.122. Place the cursor on “aVar”.
9.123. Selected **Refactor | Inline**. A message appears “Inline local variable aVar?”.
9.124. Click **Yes**. The variable is inlined.

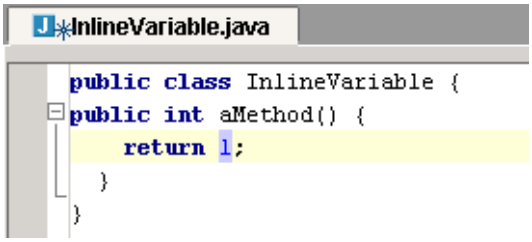


Figure 9.62. Inlined variable (395)

9.10.2. Method

9.125. Create class **InlineMethod**:

```
public class InlineMethod {
  public int methodToInline(int param) {
    return methodB() + param;
  }
  public int methodB() {
    return 1;
  }
}
```

9.126. Create class **AClass**:

```
public class AClass {
  void methodB (InlineMethod Im){
    int res = Im.methodToInline(1);
  }
}
```

9.127. Place the cursor on “methodToInline”.
9.128. Selected **Refactor | Inline**. A message appears.
9.129. Select **All invocations and remove the method**.

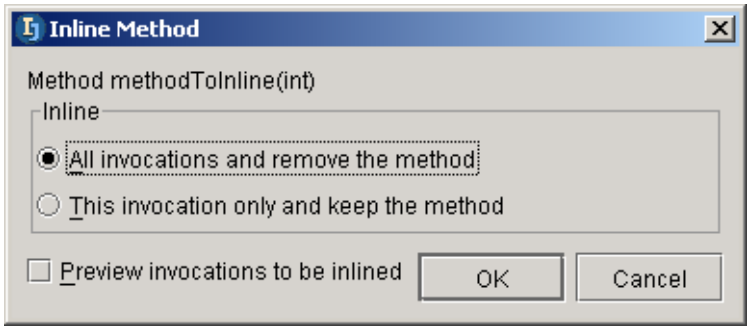


Figure 9.63. Inline level (394)

9.130. Click **OK**. The method is inlined.

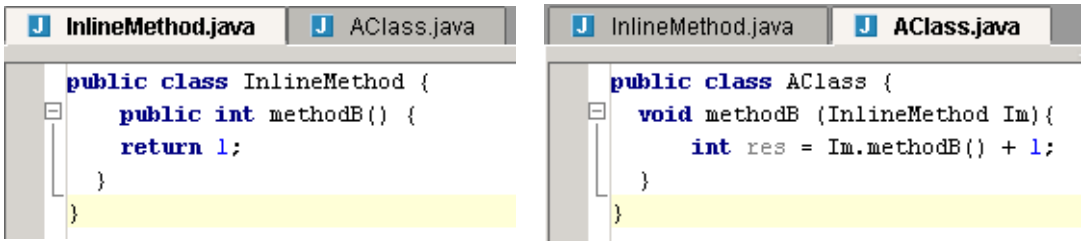


Figure 9.64. Inlined method (393,392)

9.11. Encapsulate field

9.131. Create class **EncapsulateField**:

```
public class EncapsulateField {
  public int field;
  public EncapsulateField(){
    field = 5;
  }
  public int method2(int value){
    return field + value;
  }
}
```

9.132. Create class **EFClass**:

```
class EFClass {
  private EncapsulateField ef;
  public int aMethod3(int value){
    return ef.field + value;
  }
}
```

9.133. Place the cursor in the class **EncapsulateField**.

9.134. Selected **Refactor | Encapsulate fields...**. The dialog “Encapsulate fields” appears.

9.135. Check the checkbox for **ef.EncapsulateField**.

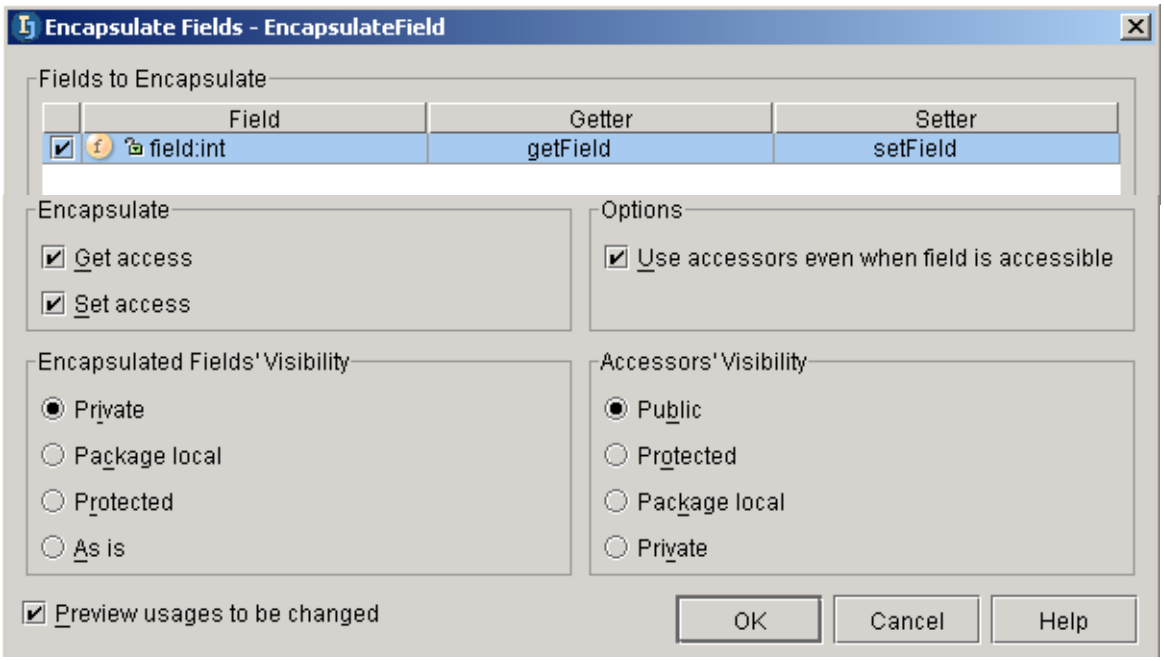


Figure 9.65. Dialog “Encapsulate fields” (391,390)

9.136. Click **OK**. A refactoring preview appears.

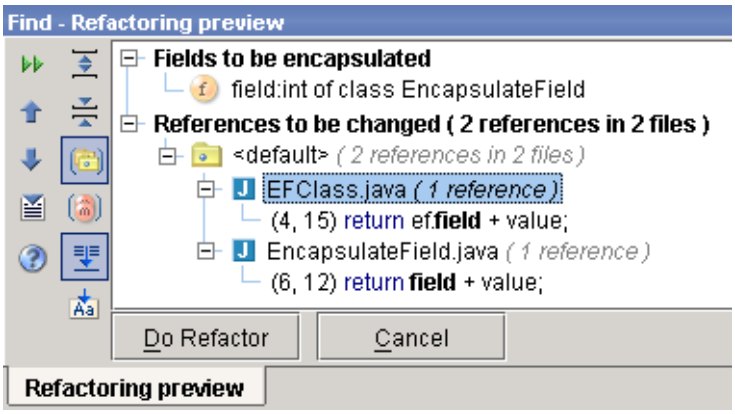


Figure 9.66. Encapsulate field refactoring preview (389)

9.137. Click **Do Refactor**. The field is encapsulated.

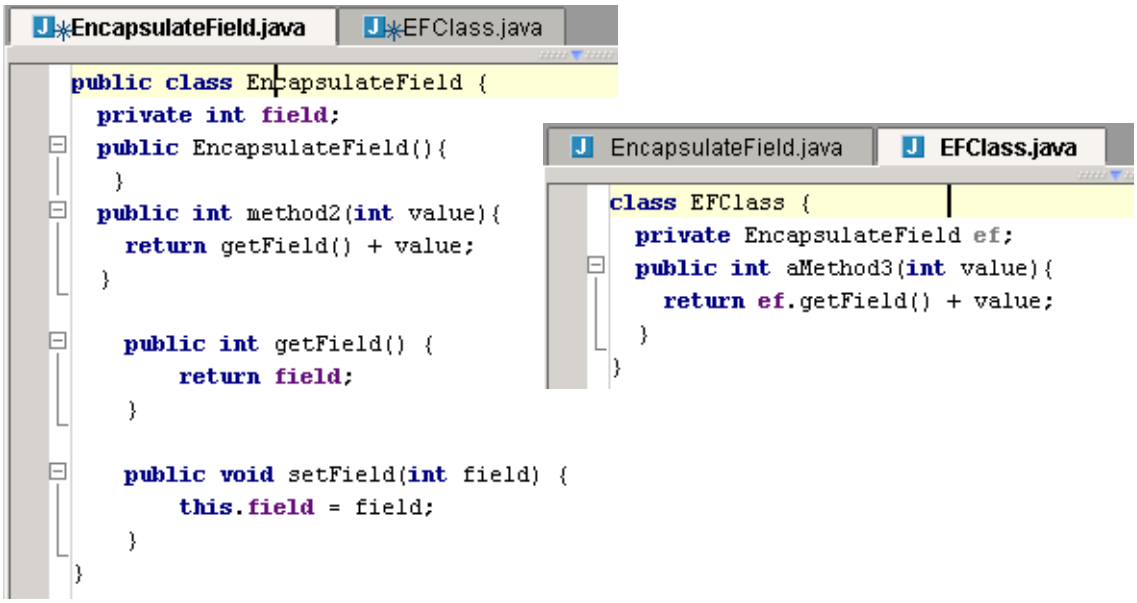


Figure 9.67. Encapsulate fields (388,387)

9.12. Replace temp with query

```
package MyPackage;
public class MyClass {
    void aMethod() {
     int temp1 = query();
     int temp2 = query();
     temp2 = temp1;
    }
     int query()
     {
         return 1;
     }
}
```

9.138. Click **Refactor | Replace temp with query...**.

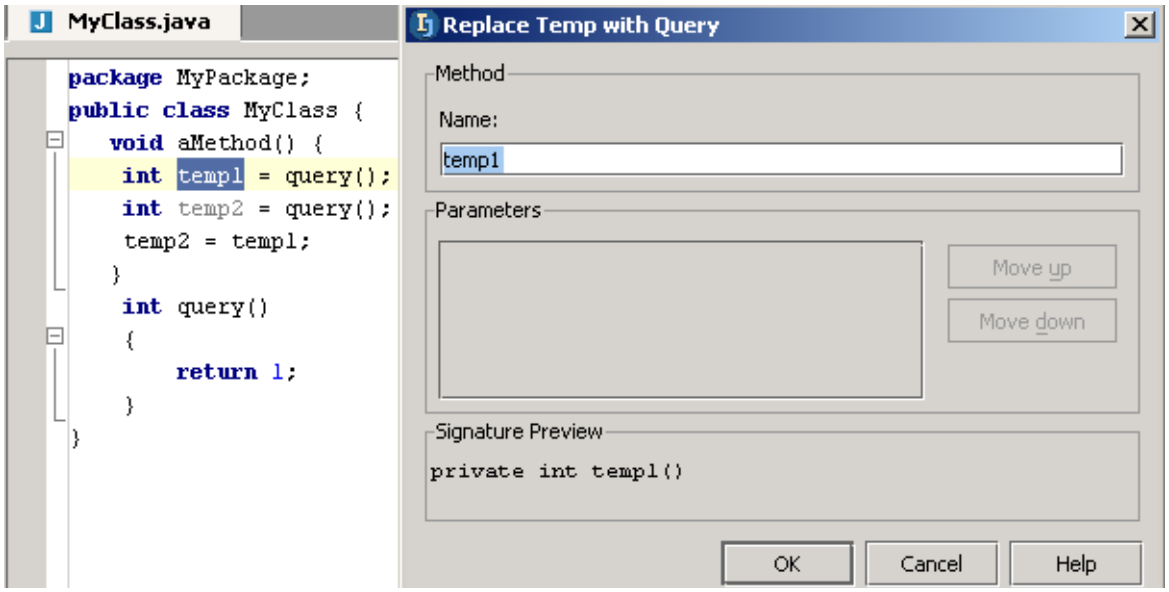


Figure 9.68. xxx (902)

9.139. Click **OK**.

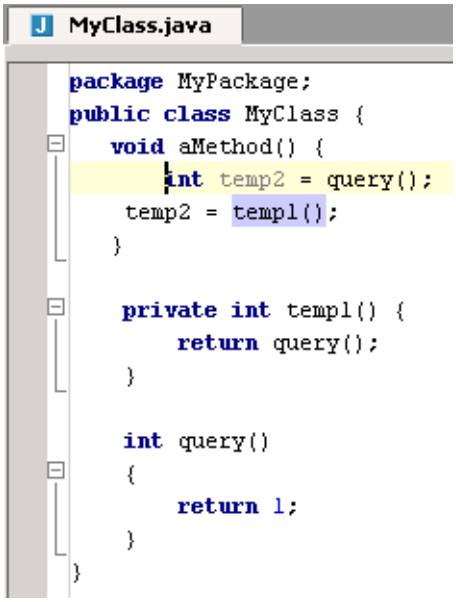


Figure 9.69. xxx (903)

9.13. Convert Anonymous to Inner

9.140. Create class **AnonToInner**:

```
public class AnonToInner {
  public AnInterface foo() {
    final int local = 1;
    return new AnInterface() {
      public int function() { return local; }
    };
  }
}
```

9.141. Create class **AnInterface**:

```
public interface AnInterface {
   int function();
}
```

9.142. Place the cursor in the class **AnInterface**.

9.143. Selected **Refactor | Convert anonymous to inner...**. The dialog “Convert Anonymous to Inner” appears.

9.144. Click **OK**. The anonymous class is converted to an inner class.

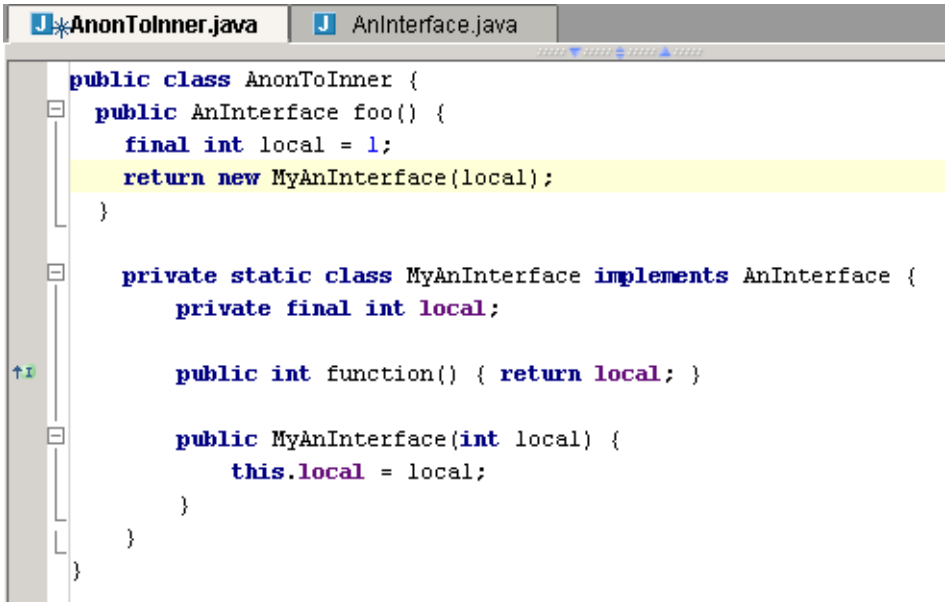


Figure 9.70. Anonymous class converted to inner (385)

10. Code Inspection X

contacts: max

10.1. Run inspection X

10.1.1. File X

10.1. Create file:

```
package MyPackage;
public class MyClass {
    void aMethod() {
        int temp2 = query();
     temp2 = temp1();
    }
     private int temp1() {
         return query();
     }
     int query() {
         return 1;
     }
}
```

10.2. Select **Tools | Inspect code...**. The dialog “Choose Inspection Scope” appears.

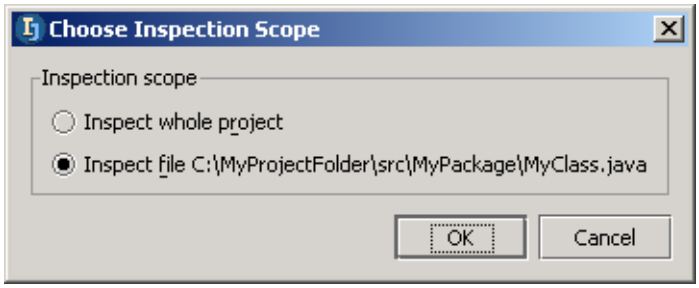


Figure 10.1. Choose inspection scope (904)

10.3. Select **Inspect file**.
10.4. Click **OK**. The dialog “Inspect code in file” appears.

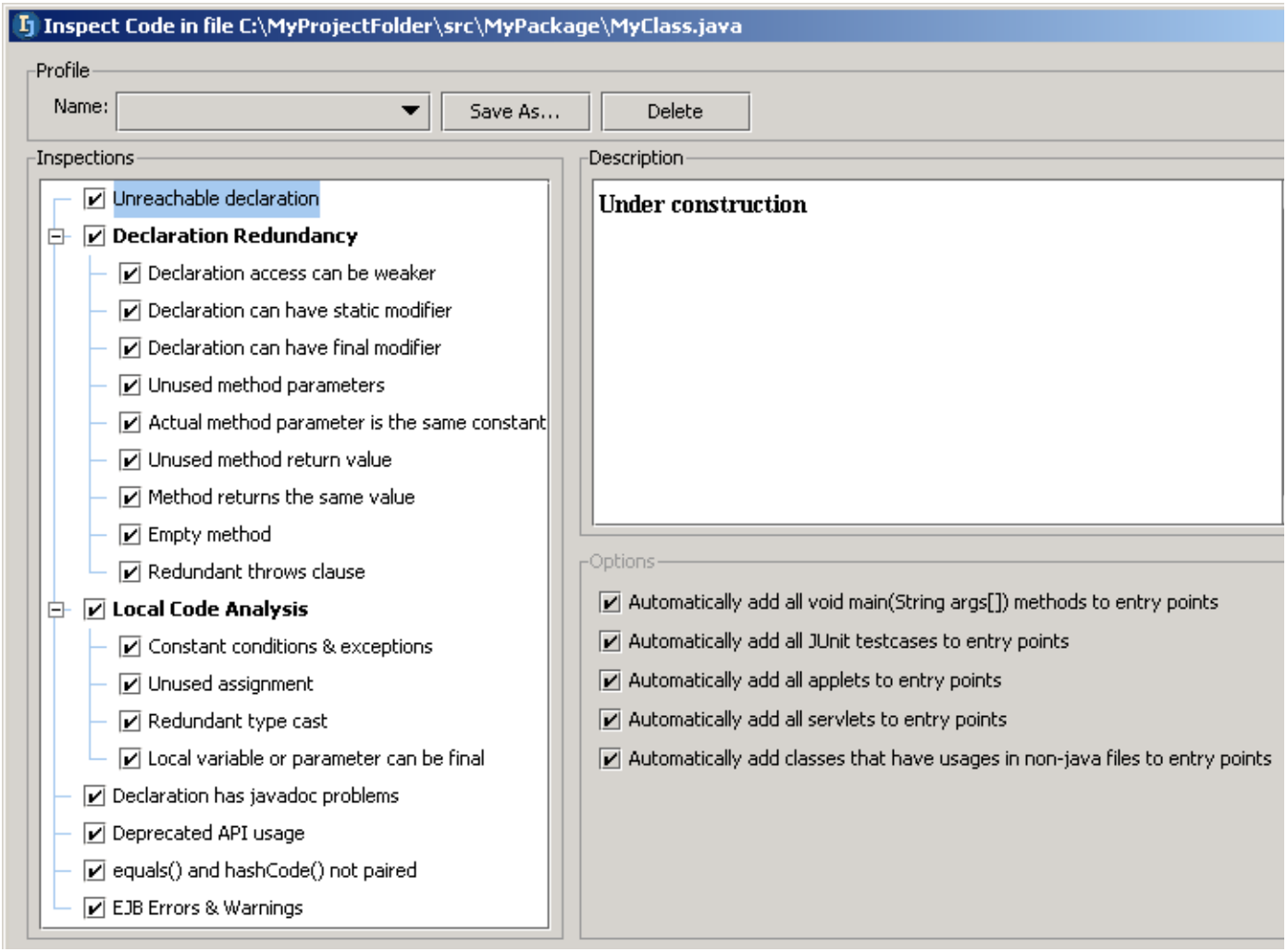


Figure 10.2. Dialog “Inspect code in file” (905)

10.5. Click **Run**. The inspection tool appears.

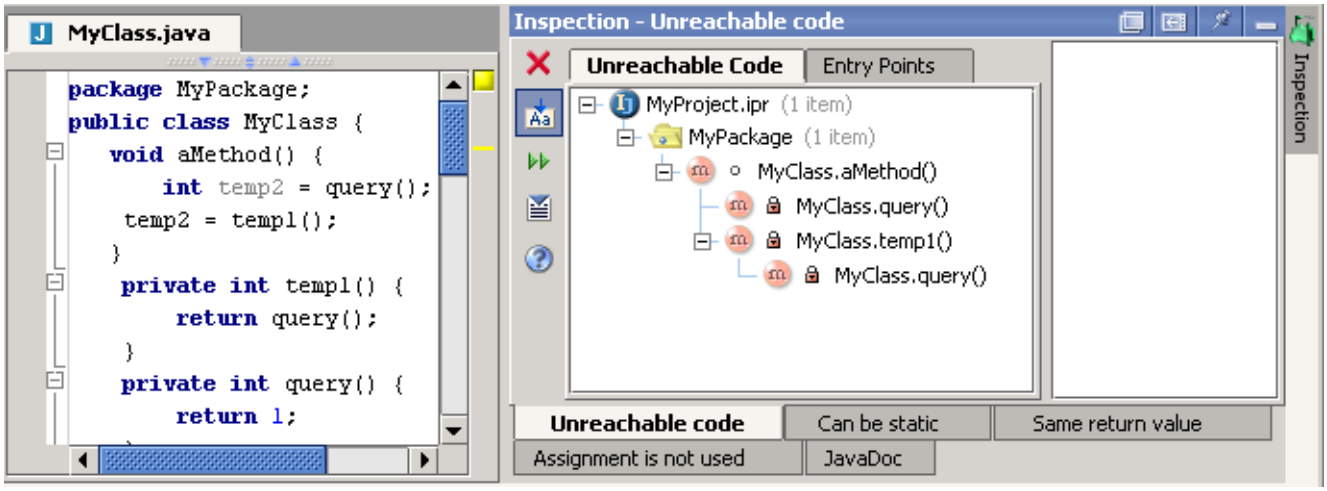


Figure 10.3. Inspection tool (906)

Inspection results

- Unreachable code
- Can be static

- Same return value
- Assignment is not used
- Javadoc

10.1.2. Project X

10.6. Close the inspection tool.
10.7. Select **Tools | Inspect code...**. The dialog “Choose Inspection Scope” appears.
10.8. Select **Inspect whole project**.
10.9. Click **OK**. The dialog “Inspect code in project” appears.
10.10. Click **Run**. The inspection tool appears.

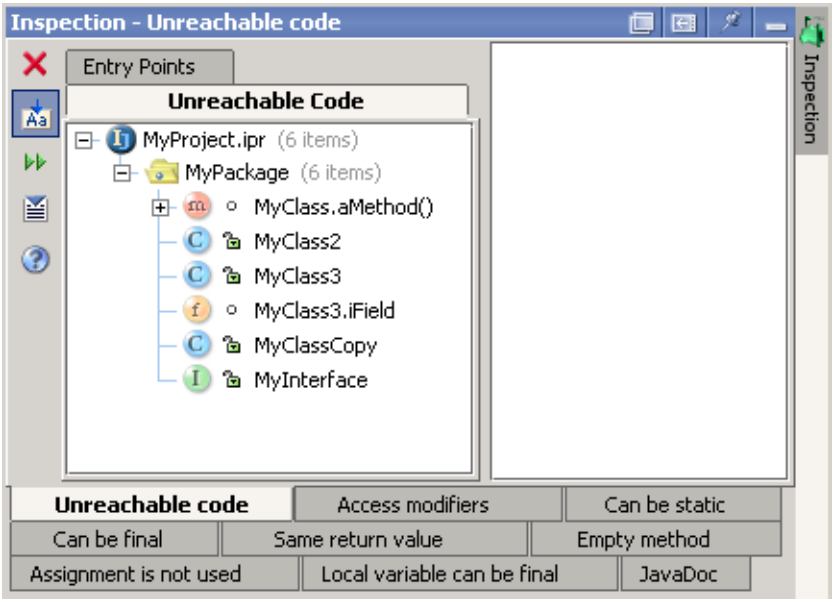


Figure 10.4. Inspection tool (project) (907)

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

10.11. Click on the rerun icon ( ) to rerun the inspection.

10.2. Resolve problems X

10.2.1. Implement suggested resolution X

typical suggested solutions

10.2.1.1. Modify X

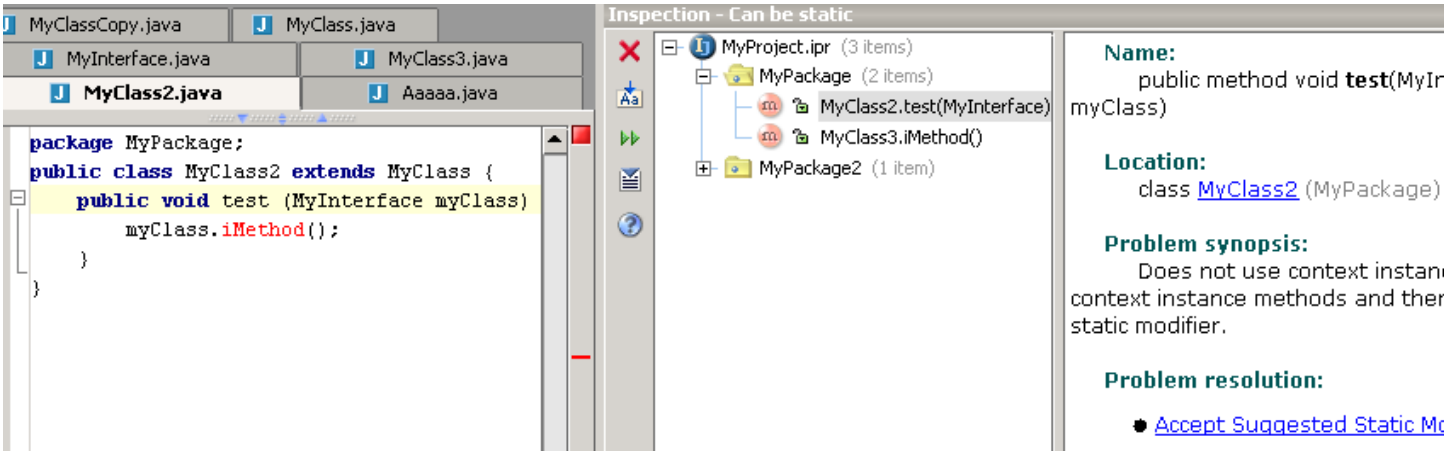


Figure 10.5. Modification suggested (915)

10.12. Click on **Accept suggested static modifier**.

```
public class MyClass2 extends MyClass {
    public static void test (MyInterface myClass) {
        myClass.iMethod();
    }
}
```

Figure 10.6. Modified (916)

10.2.1.2. Comment out X

10.13.

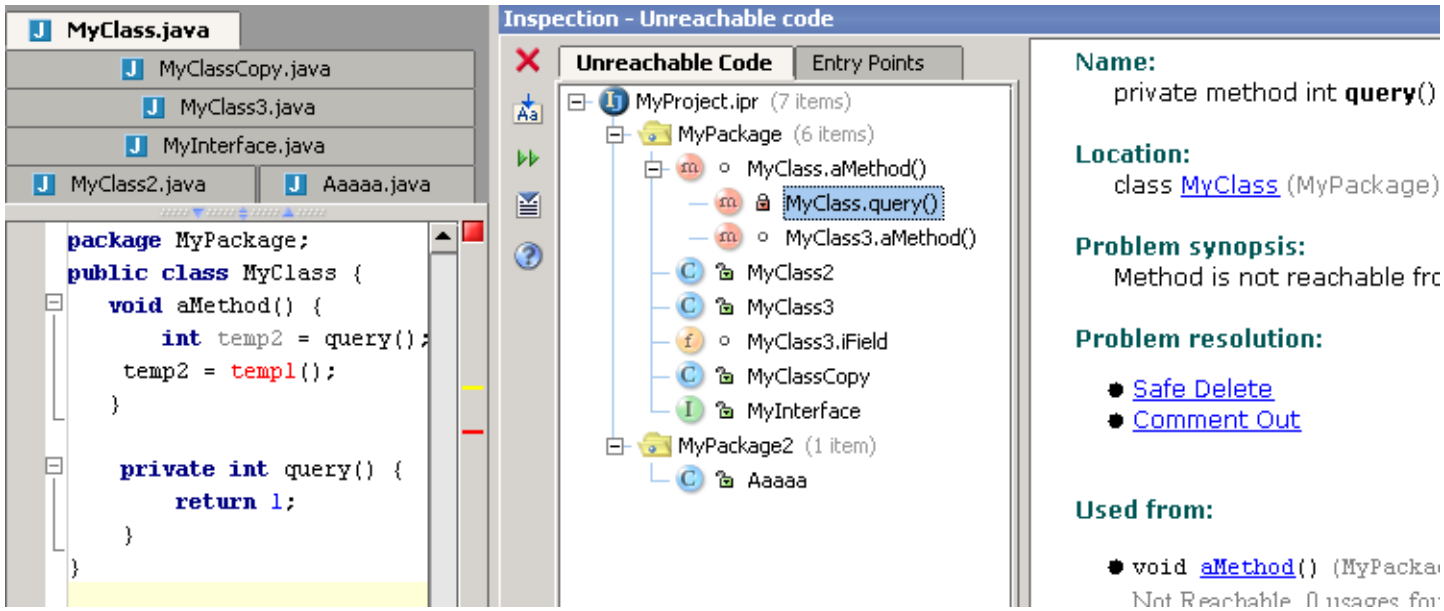


Figure 10.7. Safe delete (913)

10.14. Click on **Comment out**.

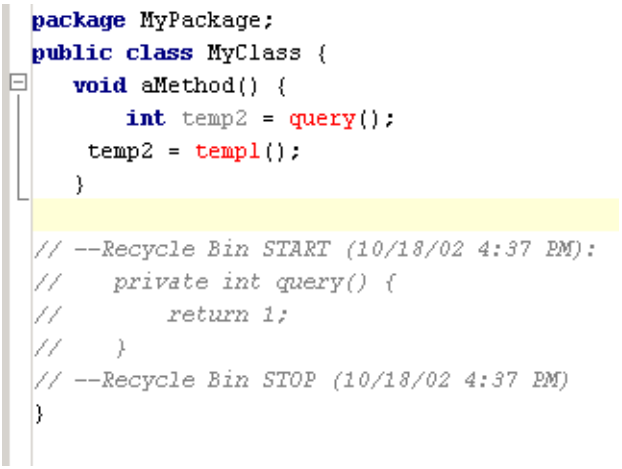


Figure 10.8. Comment out (914)

10.2.1.3. Safe delete X

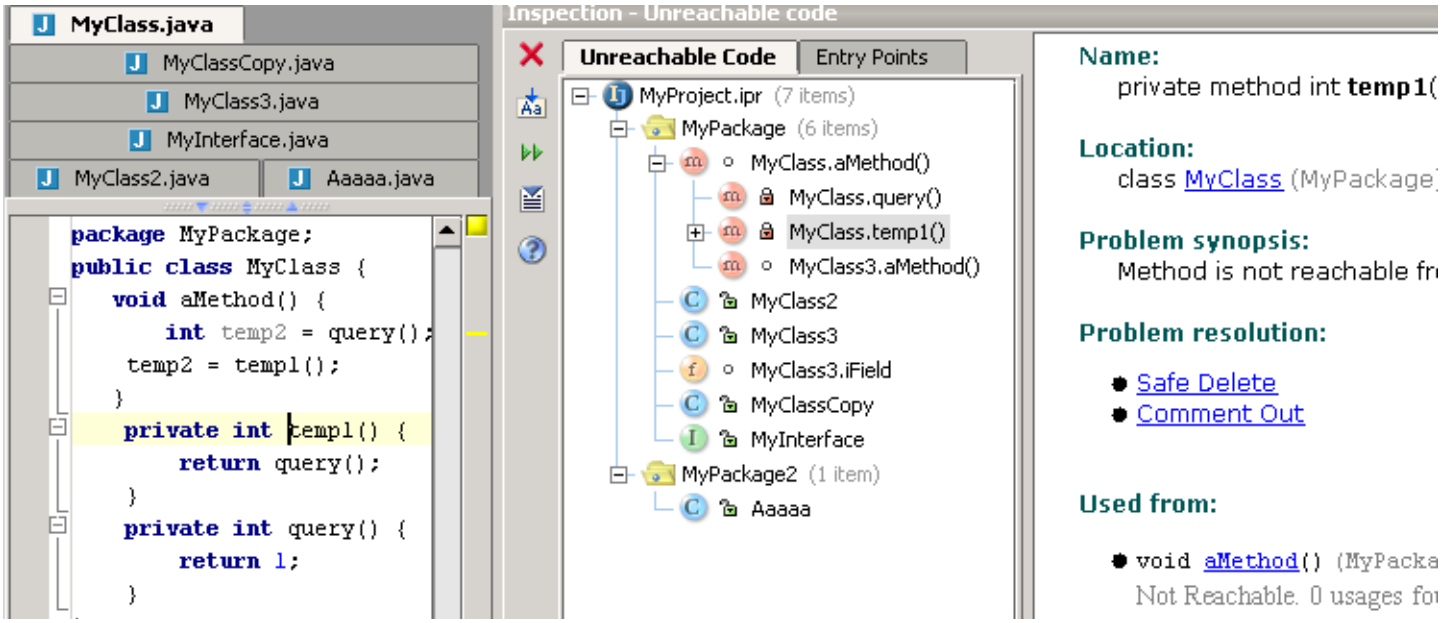


Figure 10.9. Safe delete (909)

10.15. Click on **Safe delete**.

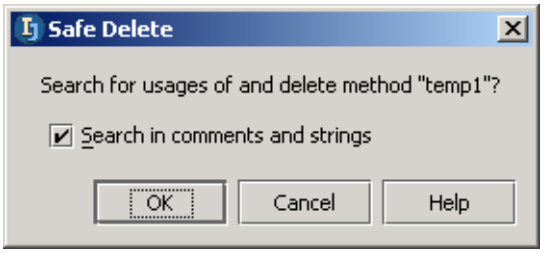


Figure 10.10. Safe delete (910)

10.16. Click **OK**.

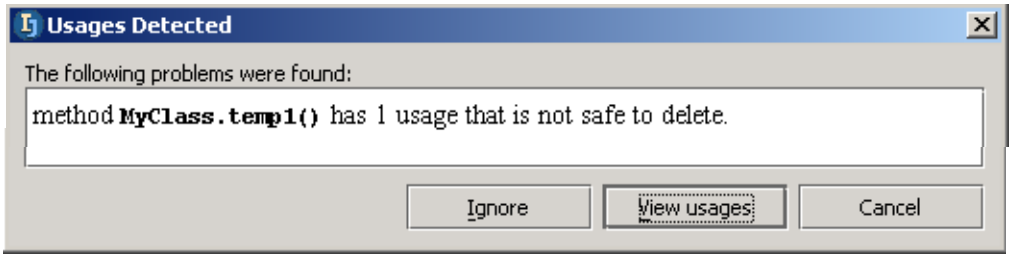


Figure 10.11. Usages detected (911,912)

10.17. Ignore and delete.

10.2.2. Manual X

Autoscroll to source.

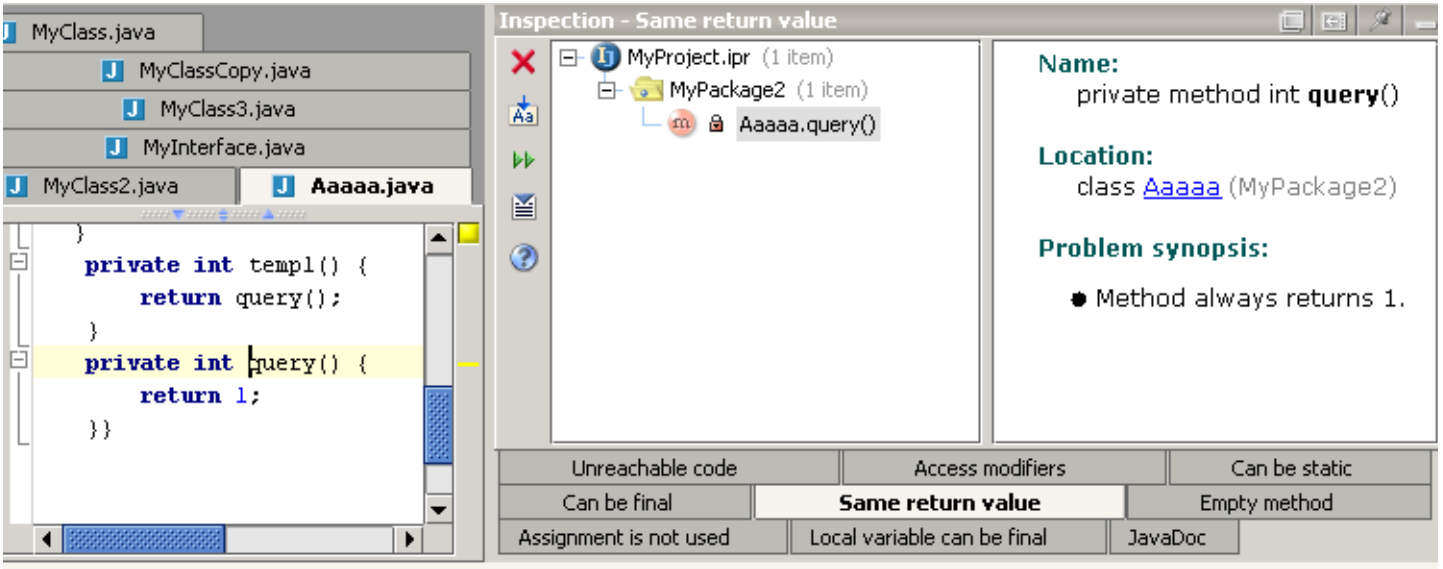


Figure 10.12. xxx (917)

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 visiblity 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

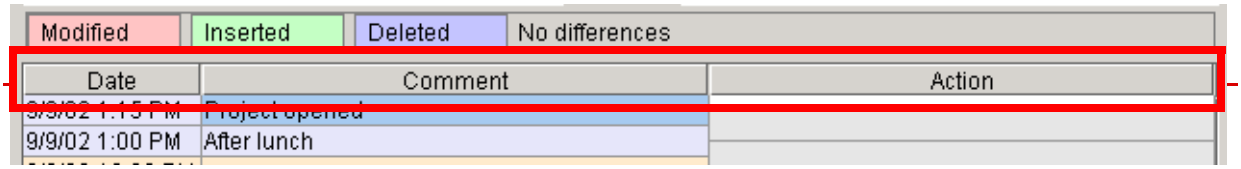
10.6.1. Create XXX

10.6.2. View XXX

11. Version control X

11.1. Local

20021012TTT: Add colors and fonts for local
CONTACT: valya or mike.
20020908TTT: ?? recommended changes to the history dialog:
current...



?? Current history dialog (516)
recommended...

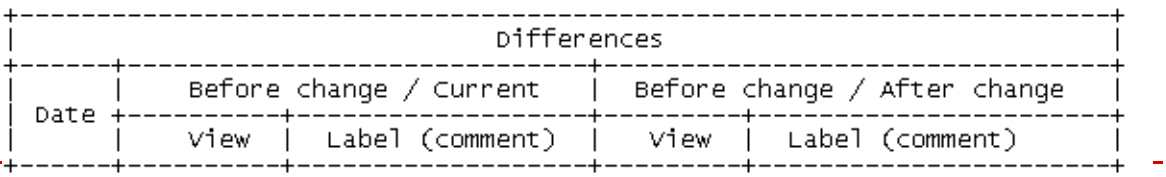


Figure 10.13. ?? Recommended changes to History dialog (517)

- 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).

11.1.1. Open History dialog

11.1. Create class file **VCSClass.java**.
11.2. Select **Tools | Local VCS | Show history**. The History dialog appears.

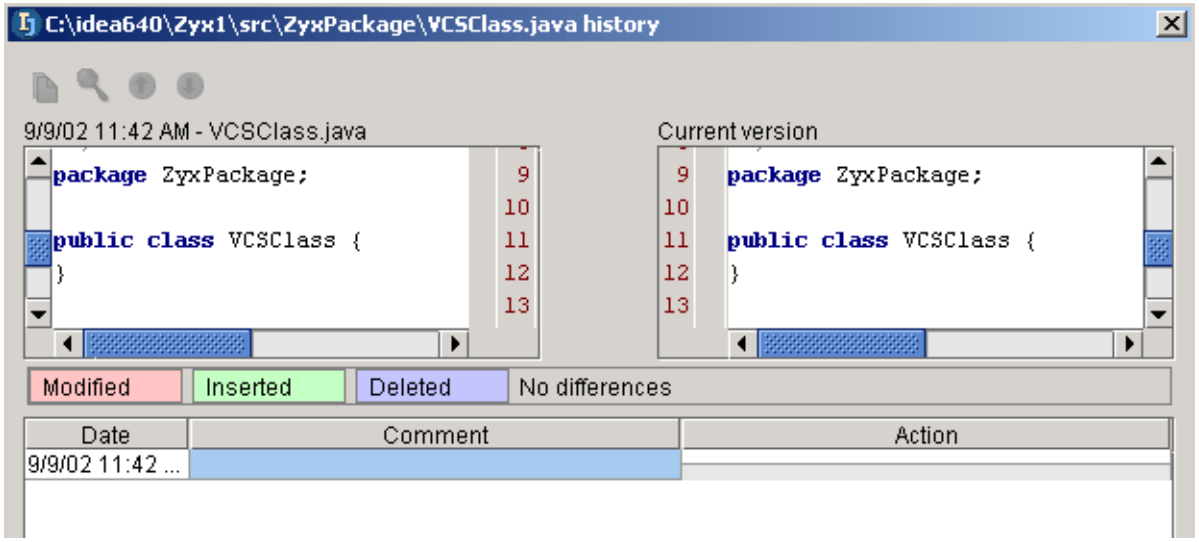


Figure 11.1. History dialog (527)

Note that only the current version is shown (no changes have been made).

11.3. Close the History dialog.

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”.

11.4. Insert the following line into the Add the fo

```
    public static void main(String[] args) {
        System.out.println("Hello");
    }
```

11.5. Open the history dialog.

11.6. Click on the lowest cell in the “Comment” column.

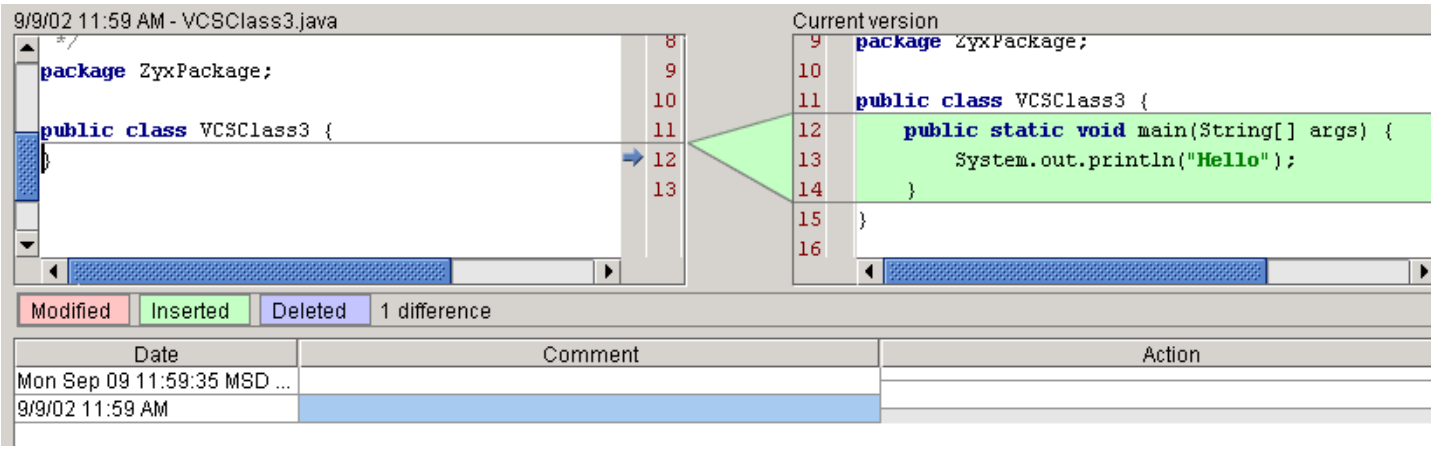


Figure 11.2. Inserted changes in the History dialog (526)

The dialog shows the inserted text.

11.1.3. Modify

11.7. Change “Hello” to “Hello World”.

11.8. Open the history dialog.

11.9. Click on the second cell from the bottom in the “Comment” column.

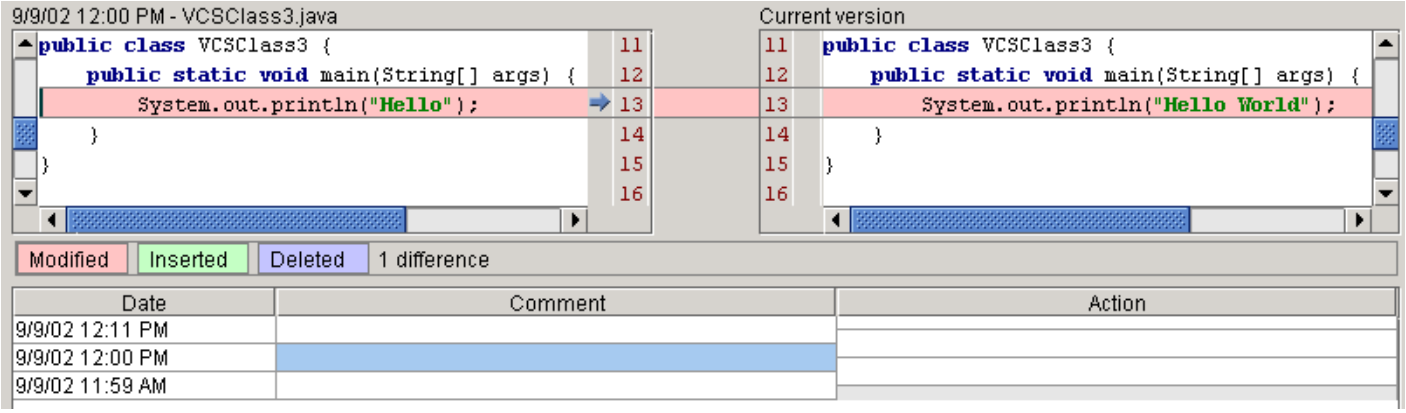


Figure 11.3. Modification in the History dialog (523)

The dialog shows the changes made during the first action.

11.1.4. Delete

11.10. Delete the line

```
System.out.println("Hello World");
```

11.11. Open the history dialog.

11.12. Click on the 3rd cell from the bottom in the “Comment” column.

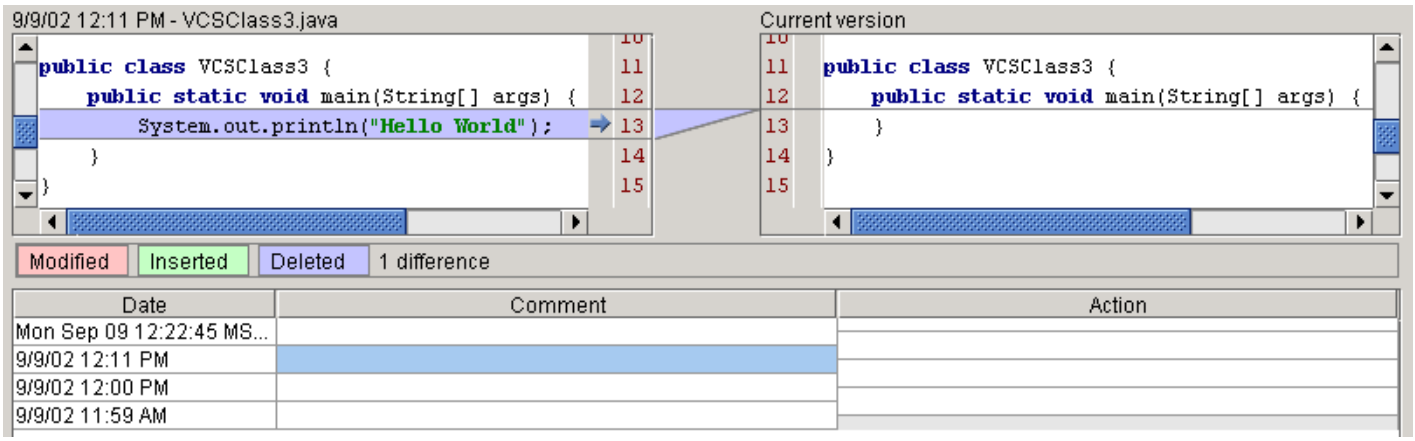


Figure 11.4. Deleted line in the History dialog (522)

The dialog shows the deleted link.

11.1.5. Rollback

11.13. Right-click on the 3rd cell from the bottom in the “Comment” column. A context menu appears.

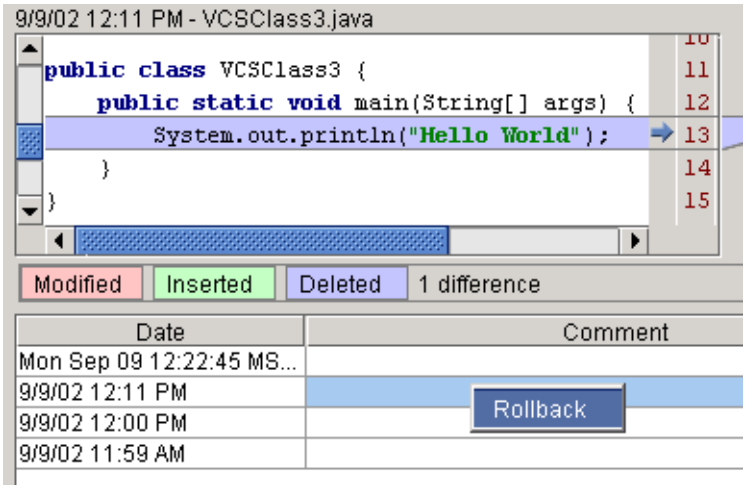


Figure 11.5. Rollback context menu (521)

11.14. Click **Rollback**. A message dialog appears “Rollback to the selected state?”.
11.15. Click **Yes**. The code line is added and the History dialog is closed.
11.16. Reopen the history dialog.
11.17. Click on the 4th cell from the bottom in the “Comment” column.

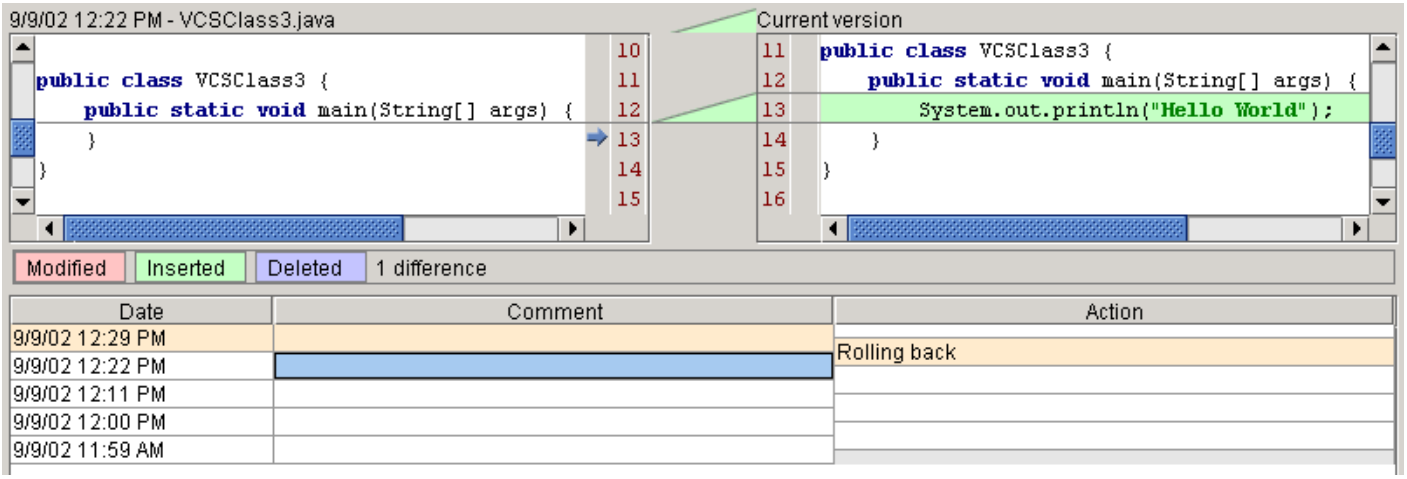


Figure 11.6. Rollback in the History dialog (520)

The dialog shows the added (“rolledback”) code.

11.1.6. Viewing changes

11.1.6.1. Between version before change and current version

11.18. Click on the lowest cell in the “Comment” column.

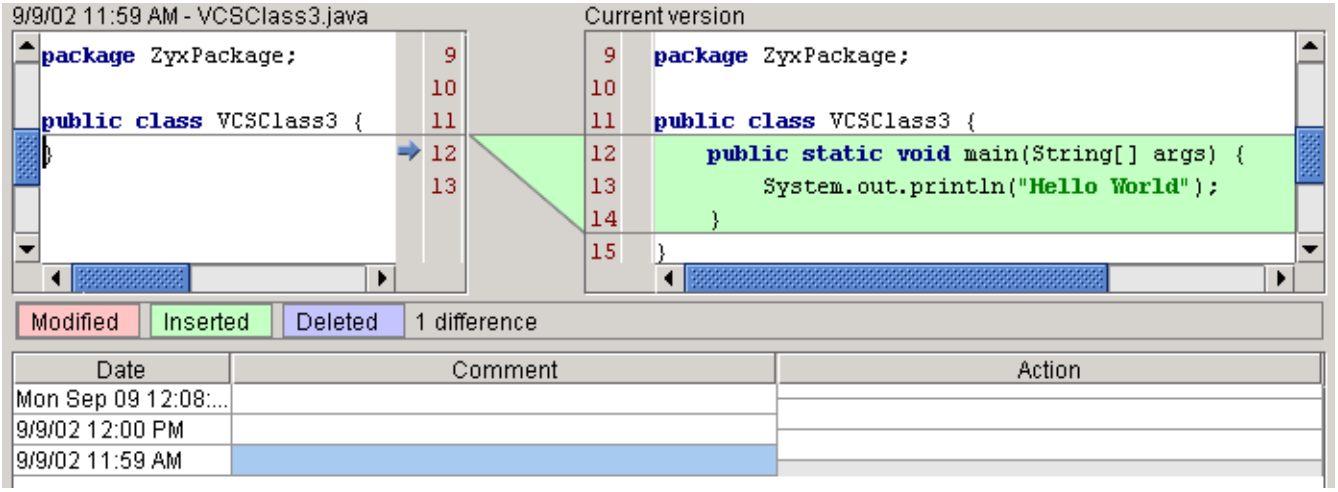


Figure 11.7. Differences between the original and the current version (525)

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?

11.1.6.2. Between versions before/after change

11.19. Click on the lowest cell in the “Action” column.

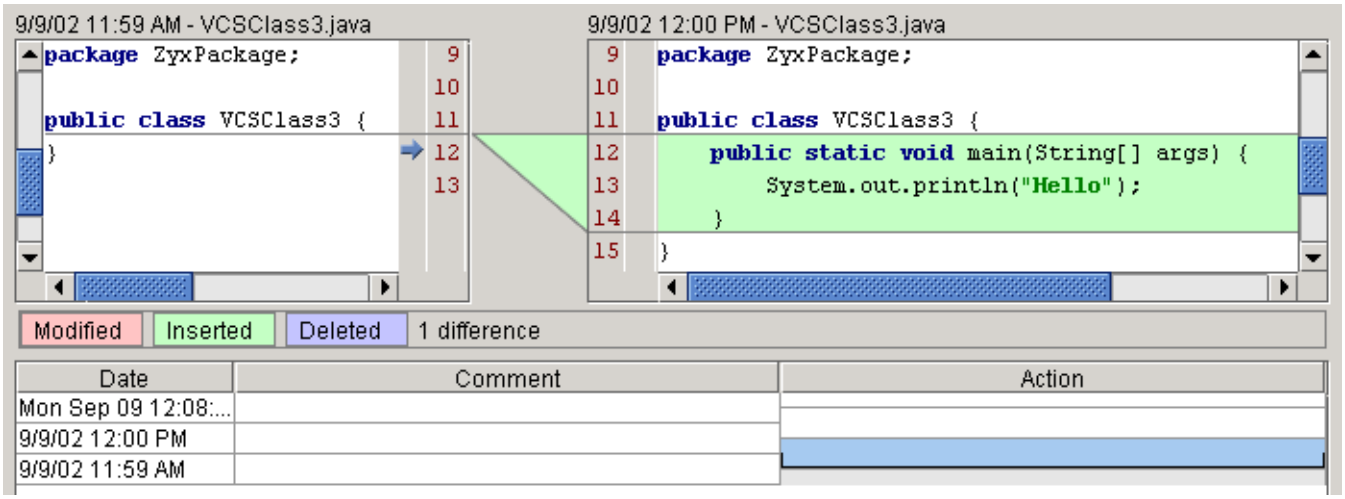


Figure 11.8. First action taken on file (524)

The dialog shows the changes between

- The file before the action was made
- The file after the action was made

11.1.7. Labels

11.20. Close the History dialog.
11.21. Select **Tools | Local VCS | Add label**. The dialog “Add label” appears.
11.22. For “Label name” enter “**After lunch**”.

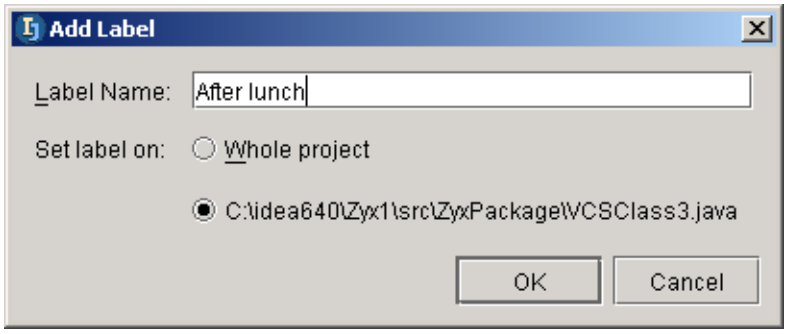


Figure 11.9. Add label dialog (519)

11.23. Click **OK**.

11.24. Open the History dialog. Note the label.

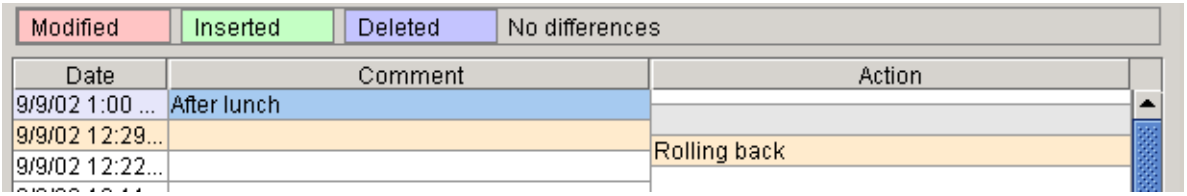


Figure 11.10. Label in history dialog (518)

11.2. CVS XXX

CONTACT: zheka

11.3. SourceSafe XXX

CONTACT: vova

11.4. StarTeam

20021003TT text added.
CONTACT: vova
idea.bat: SET CLASS_PATH=%CLASS_PATH%;%IDEA_HOME%\lib\starteam-sdk.jar

11.4.1. Install

11.25. Copy **C:\idea650\starteam_plugin\starteam.jar** to **C:\idea650\plugins**.
11.26. Add **C:\Program Files\Starbase\StarGate SDK\Lib\starteam-sdk.jar** to idea **classpath**.

Figure 11.11. StarTeam classpath (266)

11.4.2. Configure connection

11.27. For VCS Support select **StarTeam**.

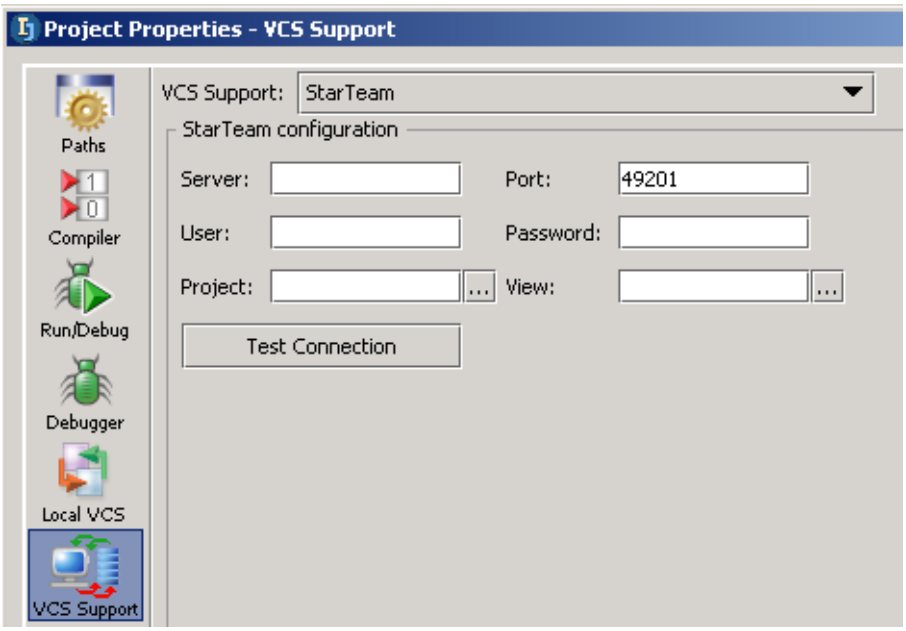


Figure 11.12. VCS Support StarTeam (265)

11.28. Enter following:

- Server
- Port
- User
- Password
- Project
- View

11.29. Click **Test Connection**. The dialog “Connection successful” appears.

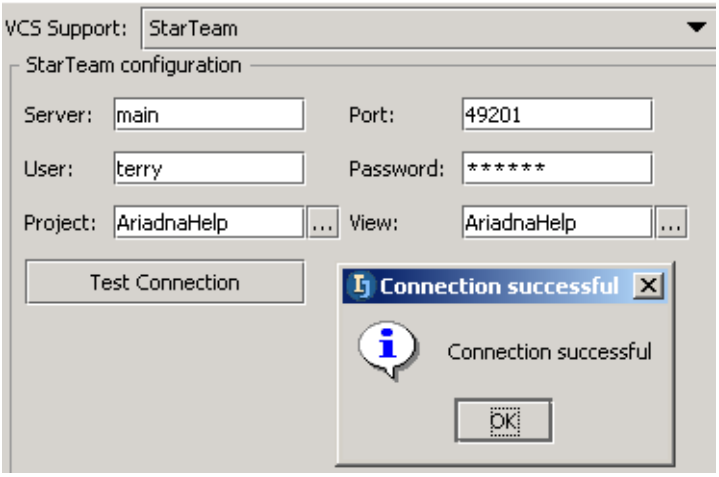


Figure 11.13. StarTeam configuration (264)

11.30. Click **OK**.

11.4.3. Add StarTeam dir to project

11.31. Add the folder:

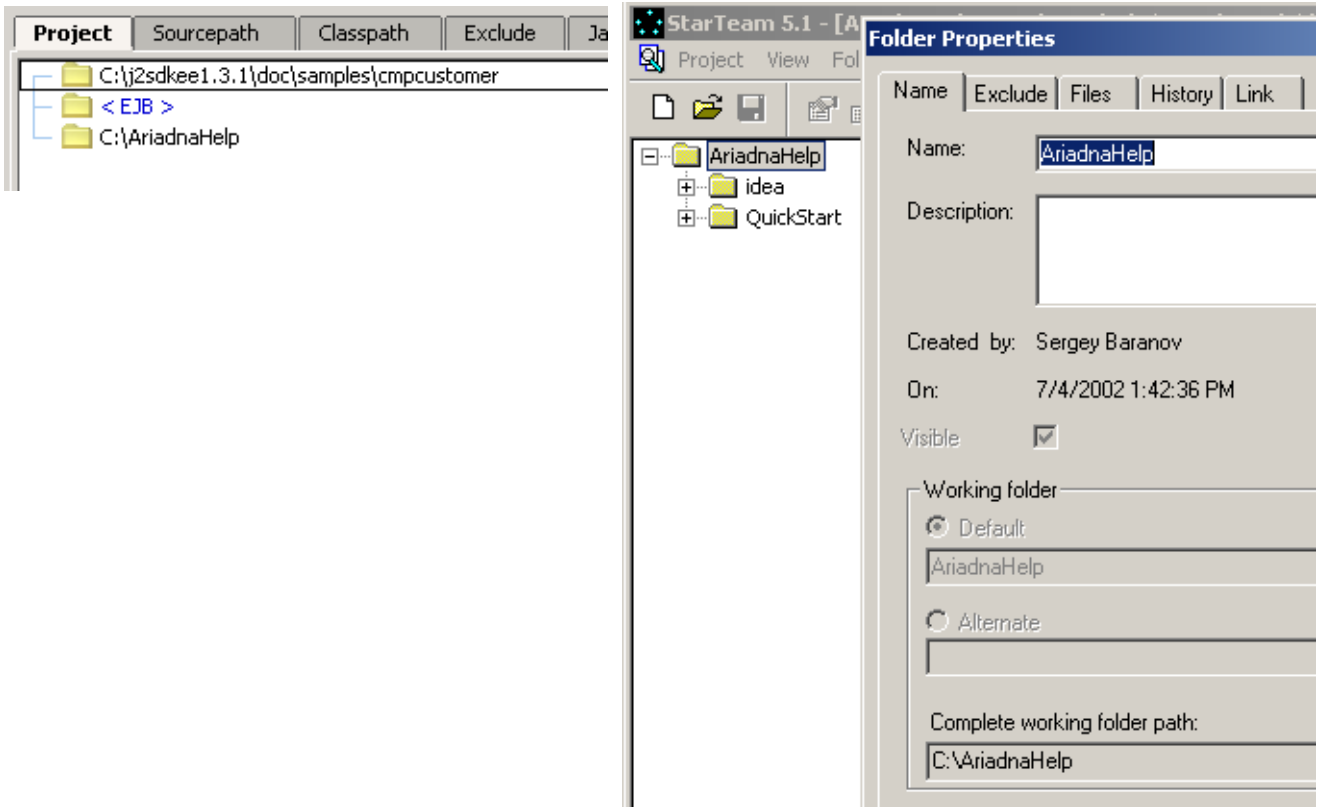


Figure 11.14. StarTeam folder added to project (263,262)

11.4.4. Create dir / file

11.32.

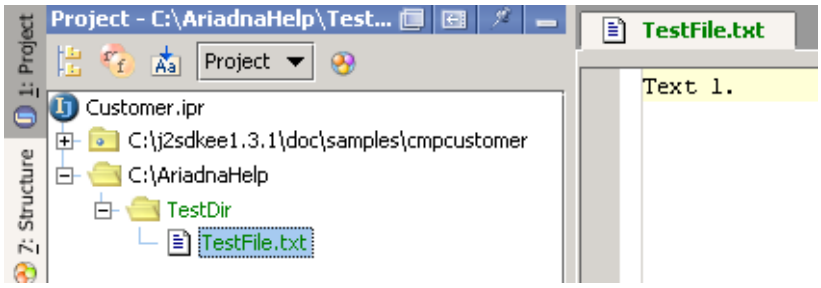


Figure 11.15. Dir, folder added (261)

11.4.5. Check in

11.33. Right-click on TestFile.txt.
11.34. Select Check in Project.

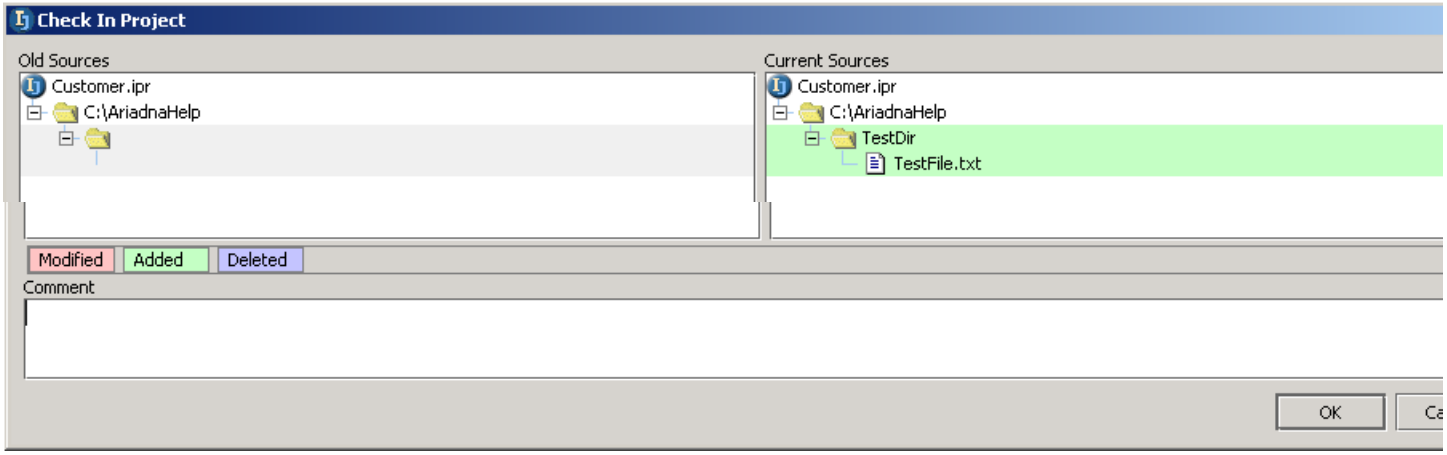


Figure 11.16. xxx (260,259)
11.35. Click **OK**.

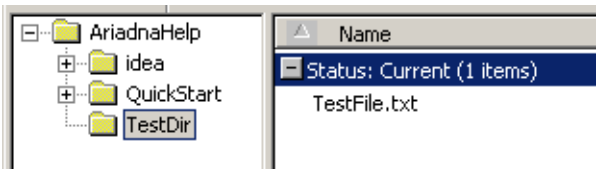


Figure 11.17. xxx (258)

11.4.6. Show differences

11.36. Add second line “Text 2.” to TestFile.txt.
11.37. Select StarTeam / Show difference.

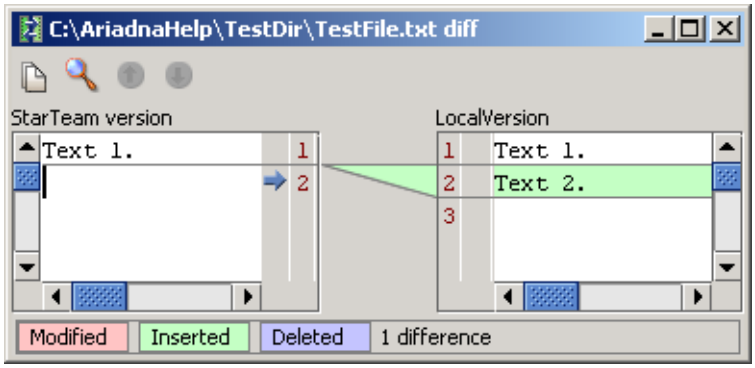


Figure 11.18. xxxx (257)

11.4.7. Check in

11.38. Right-click on TestFile.txt.
11.39. Select Check in Project.

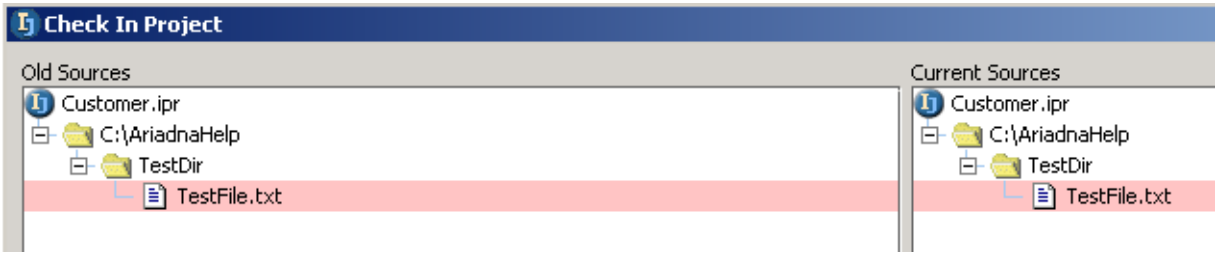


Figure 11.19. xxx (256)

12. Java Doc X

20021001TT text added.
contacts: anton

12.1. JDK JavaDoc

12.1.1. Install

12.1. Download **j2sdk-1_4_1-doc.zip** from java.sun.com.
12.2. Extract to c:\idea640 (use folder names). The javadocs api for the JDK classes are now located in c:\idea640\docs\api.

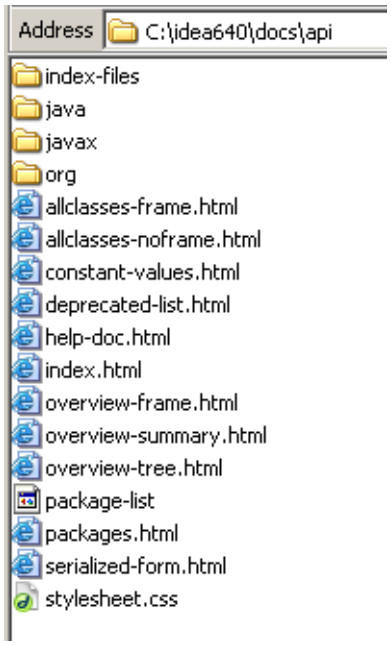


Figure 12.1. xxxx (313)

12.1.2. View

12.1.2.1. All

12.3. Open **c:\idea640\docs\api\index.html**.

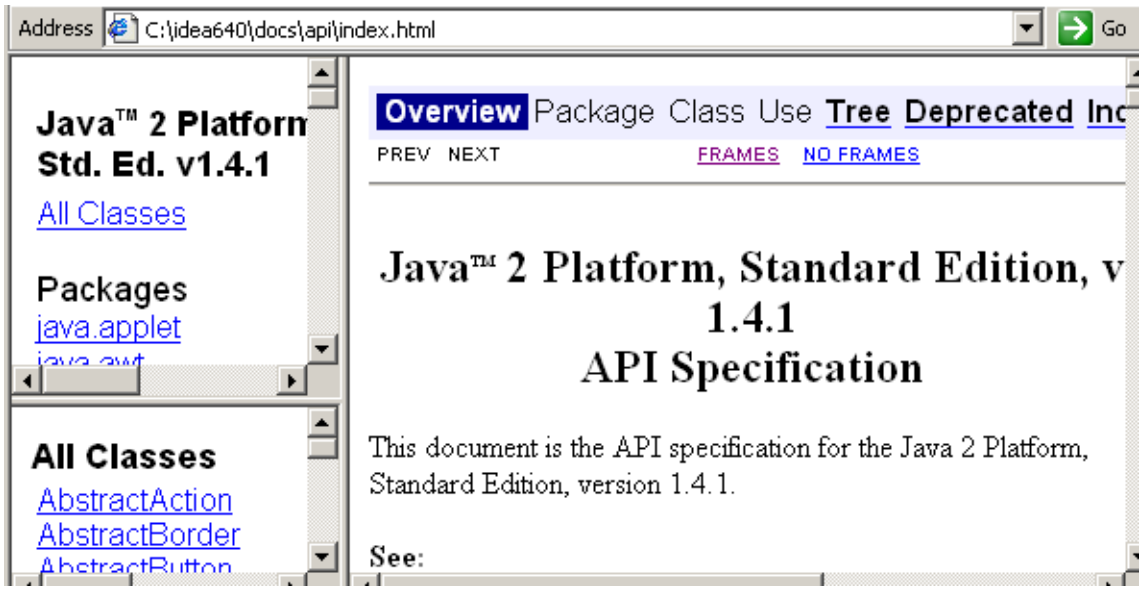


Figure 12.2. xxxx (312)

12.1.2.2. For element

12.4. Select class Applet as shown below.

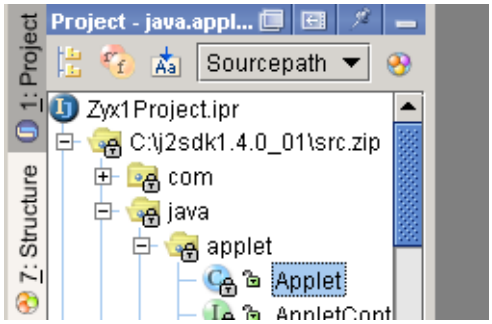


Figure 12.3. xxxx (311)

12.5. Select **View | External Java doc**. The java documentation is opened in an editor.

Figure 12.4. xxxx (310)

12.6. Open the source code for Applet.java as shown below.

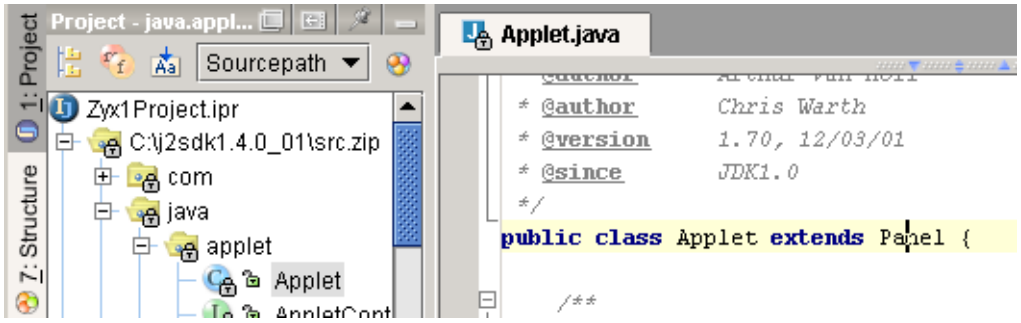


Figure 12.5. xxxx (309)

12.7. Place the cursor on **Panel**.

12.8. Click **Shift-F1**. JavaDoc for Panel is opened.

Figure 12.6. xxxx (308)

12.2. Quick JavaDoc

12.2.1. JDK class

12.2.1.1. Open for selected

12.9. Place the cursor on “Applet” in the class definition.
12.10. Press**Ctrl-Q**. The quick javadoc popup for “Applet” appears.

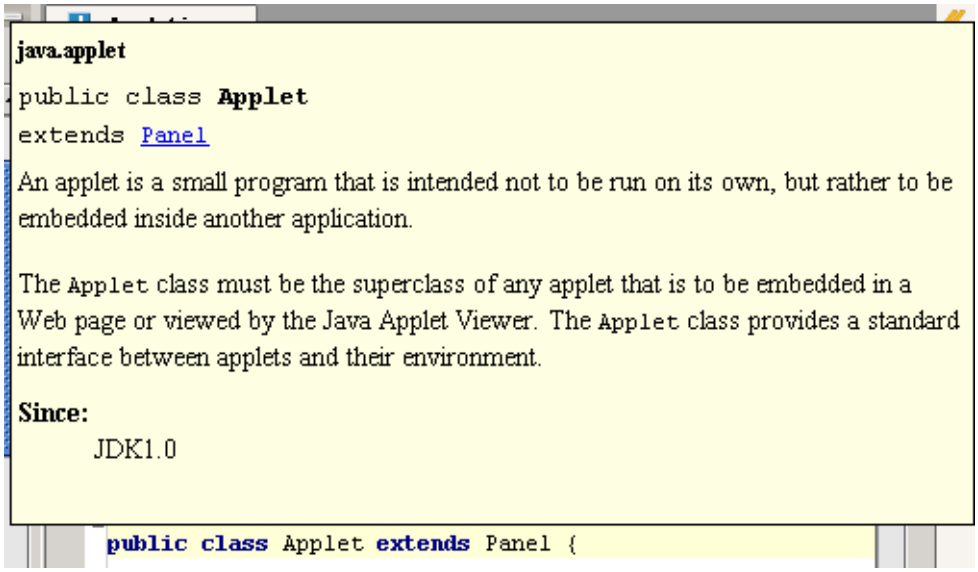


Figure 12.7. xxxx (306)

Note that the above javadoc was generated directly from the contents of the file.

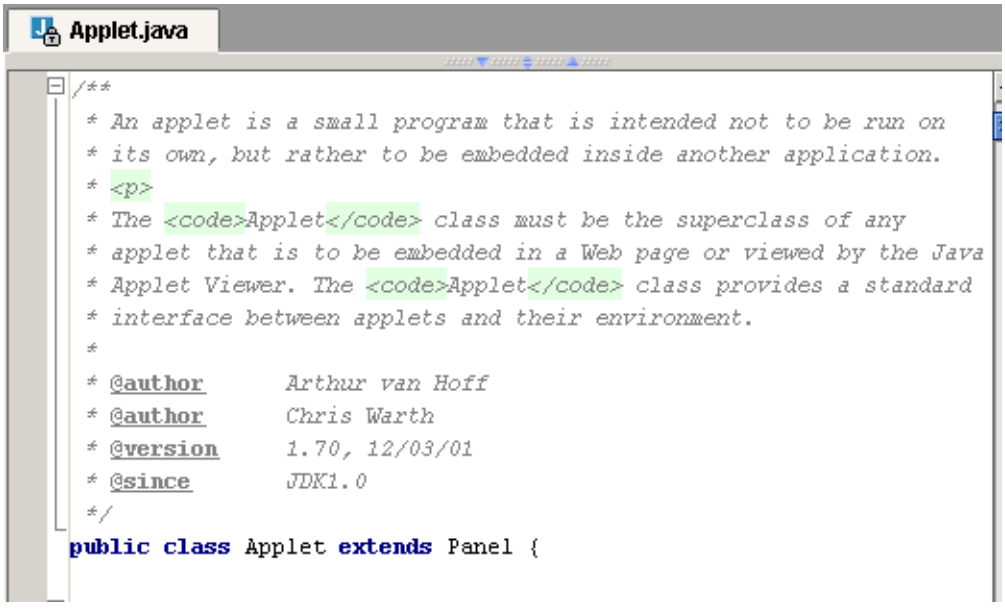


Figure 12.8. xxxx (307)

12.2.1.2. Open for other

12.11. Click on the hyperlink for **Panel**. The quick javadoc for “Panel” is opened.

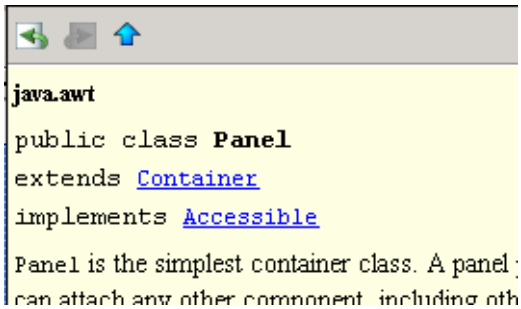


Figure 12.9. xxxx (305)

Note: You can go back to the previous quick javadoc popup dialog by pressing the arrow icon ( ).

12.2.1.3. Open JDK JavaDoc

12.12. Click on the arrow key ( ). JavaDoc is opened.

Figure 12.10. xxxx (302)

12.2.2. Own class

12.2.2.1. Open

12.13. Create class:

```
package AAAPackage1;
public class AClass1 {}
```

12.14. Place the cursor on **AClass1**.
12.15. Click **Ctrl-Q**. The quick javadoc is opened.

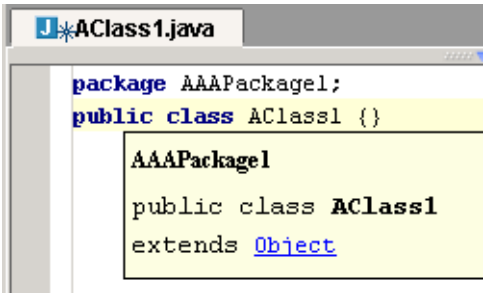


Figure 12.11. xxxx (301)

12.2.2.2. Attempt to open JDK JavaDoc (none)

12.16. Try to open JavaDoc for AClass1. Note the error:

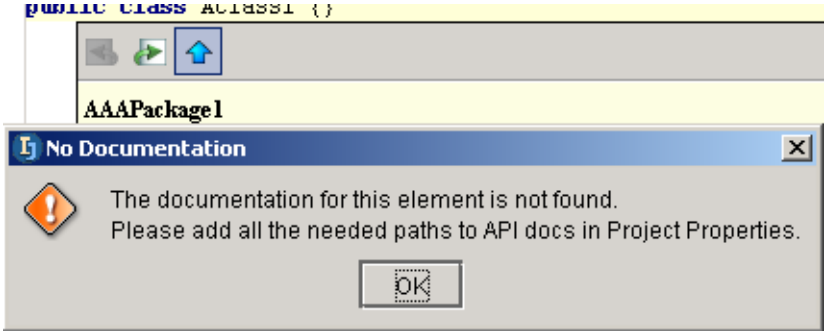


Figure 12.12. xxxx (300)

12.3. JavaDoc tags

Show what tags generate what.

12.3.1. Add tags

12.17. Create class:

```
package AAAPackage1;
public class AClass1 {
public int foo(int p1, int p2) {
              return p1 + p2;
    }
}
```

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

```
public int foo(int p1, int p2) {
```

12.19. Enter "**/****".

12.20. Press **Enter**. The tags for the method are entered:

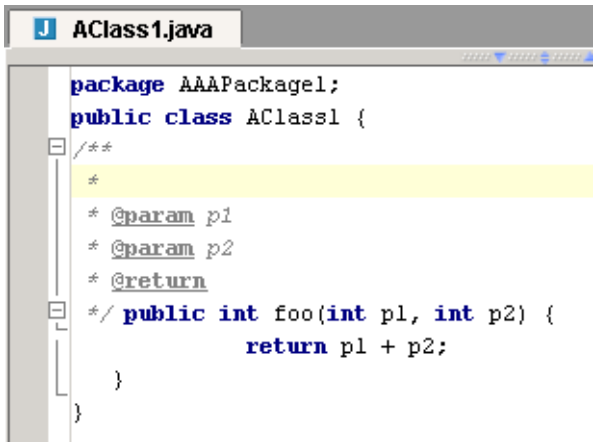


Figure 12.13. xxxx (299)

12.21. Place the cursor after the "*" on line 4.

12.22. Enter "@". Note that a popup dialog appears.

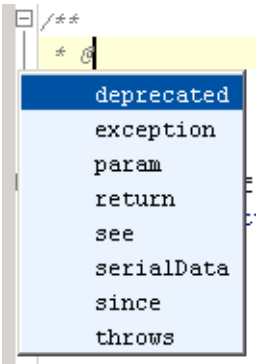


Figure 12.14. xxxx (298)

12.23. Click **exception**.

12.24. Click **space**.

12.25. Click **Ctrl-Space**. A popup dialog appears.

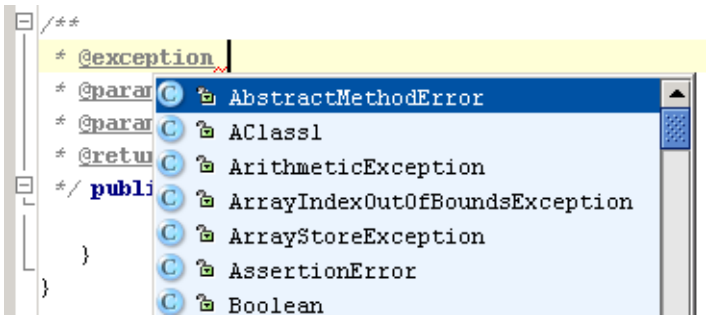


Figure 12.15. xxxx (297)

12.26. Select **Unknown error**.

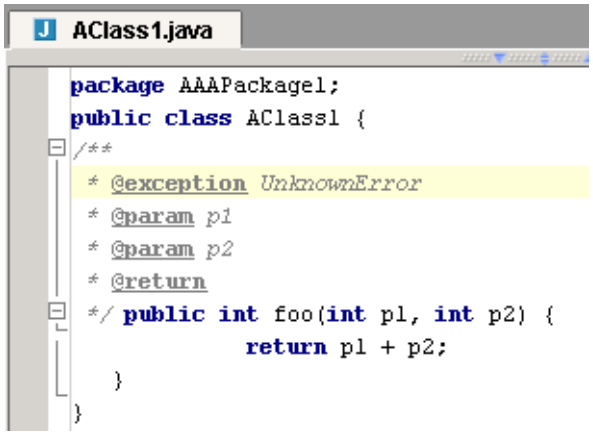


Figure 12.16. xxxx (296)

12.3.2. Display tags in quick JavaDoc

12.27. Place the cursor on **foo**.

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

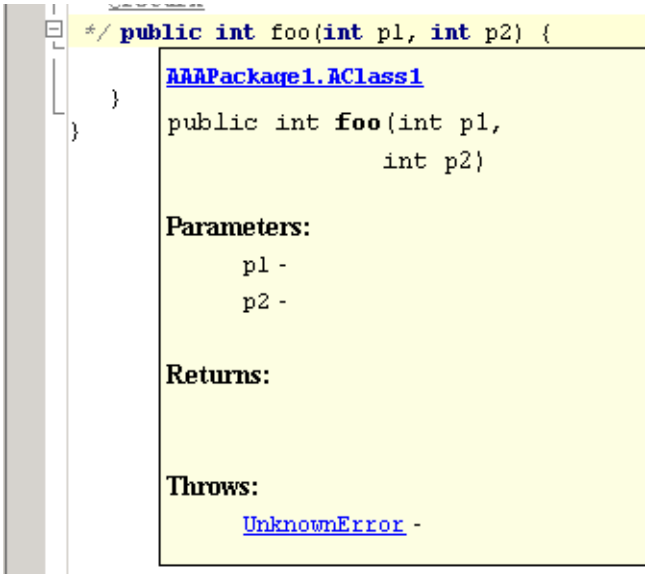


Figure 12.17. xxxx (295)

12.4. Own JavaDoc

12.4.1. Generate

12.29. Select **AAAPackage1**.
12.30. Select **Tools | Generate JavaDoc...**. The dialog “Generate JavaDoc” appears.

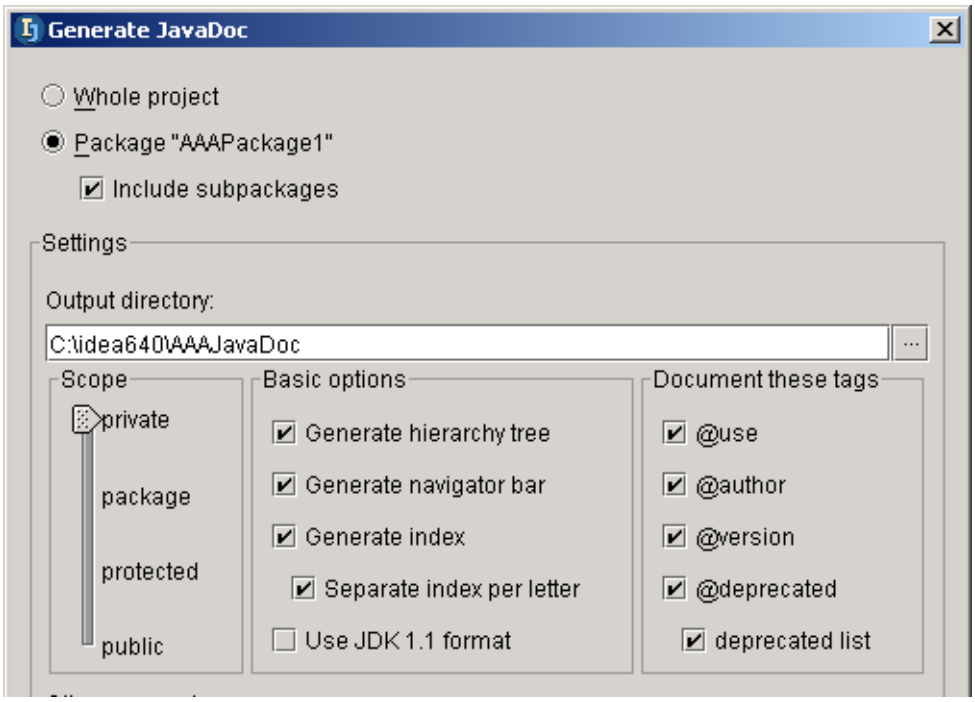


Figure 12.18. xxxx (294)

12.31. Click **Run**. The tool “Run - JavaDoc” appears.

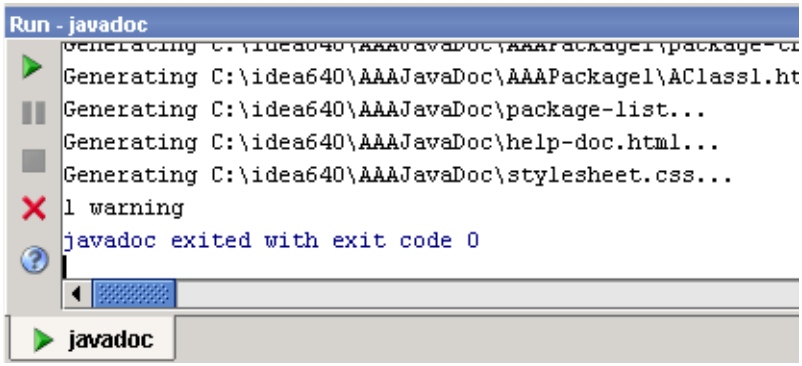


Figure 12.19. xxxx (293)

File C:\idea640\AAAJavaDoc\index.html is the main page.

12.4.2. Install

12.32. Add the JavaDoc directory.

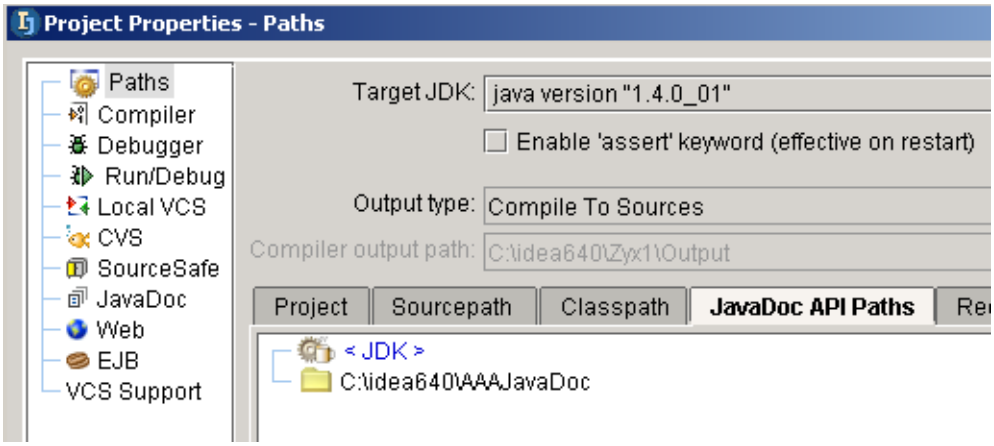


Figure 12.20. xxxx (292)

12.4.3. View

12.33. Place the cursor on **foo**.
12.34. Select **View | External JavaDoc**. JavaDoc is displayed.

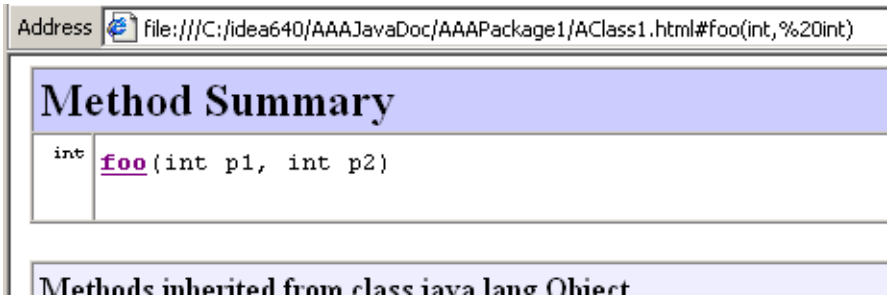


Figure 12.21. xxxx (291)

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 XXX (page 259). Demonstrates how to use the IDEA compiler.

14. Ant X (page 263). Demonstrates how to use Ant with IDEA.

15. Debugger X (page 269). Demonstrates how to use the IDEA debugger.

~~**16. JUnit XXX (page 279)**~~.

~~**17. Remote debugging XXX (page 281)**~~.

13. Compiler XXX

contacts: zheka

13.1.

Figure 13.1.

Libraries??

Add Jikes somewhere

13.1. Single file

13.2. Make

compile modified + dependencies.

13.3. Build

14. Ant X

20021002TT: text added. need to simplify example.
contacts: zheka

14.1. Download / Install

14.1.1. Download

http://jakarta.apache.org/ant/index.html
http://jakarta.apache.org/builds/jakarta-ant/release/v1.5/bin/
14.1. File downloaded is jakarta-ant-1.5-bin.zip.

14.1.2. Install

14.2. Unzip to c: using directories.

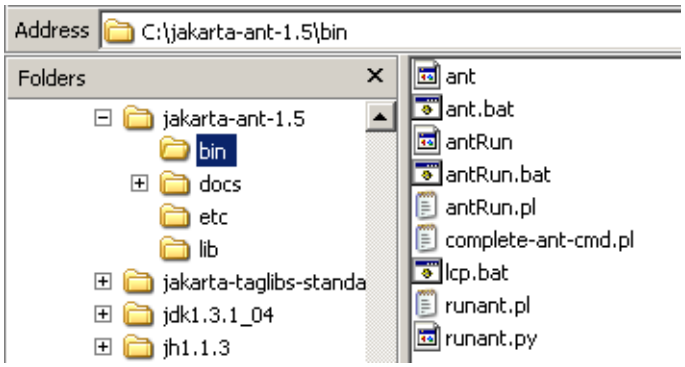


Figure 14.1. Ant directories (273)

14.3. Add environment variable **ANT_HOME** with value **C:\jakarta-ant-1.5**.
14.4. Add to environment variable **Path** dir **C:\jakarta-ant-1.5\bin**.

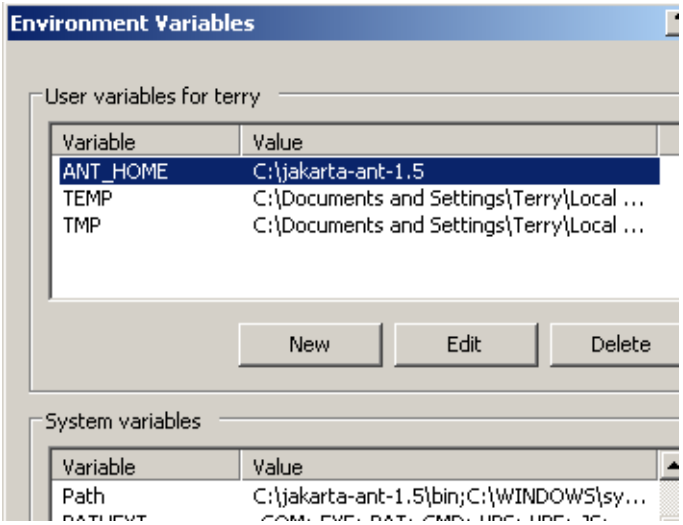


Figure 14.2. Environment variables for Ant (272)

14.2. Create build.xml

14.5. Create **C:\idea640\Zyx1\build.xml**:

```
<project name="ZyxProject" default="all" basedir=".">

<target name="init">
  <property name = "build"      value = "Output" />
  <property name = "source"     value = "src\AAAPackage1" />
  <property name = "class_path" value = "c:\idea640\Zyx1" />
</target>

<target name="all" depends="init">
  <mkdir dir="${build}" />
  <javac srcdir    = "${source}"
         destdir   = "${build}"
         classpath = "${class_path}" />
</target>

</project>
```

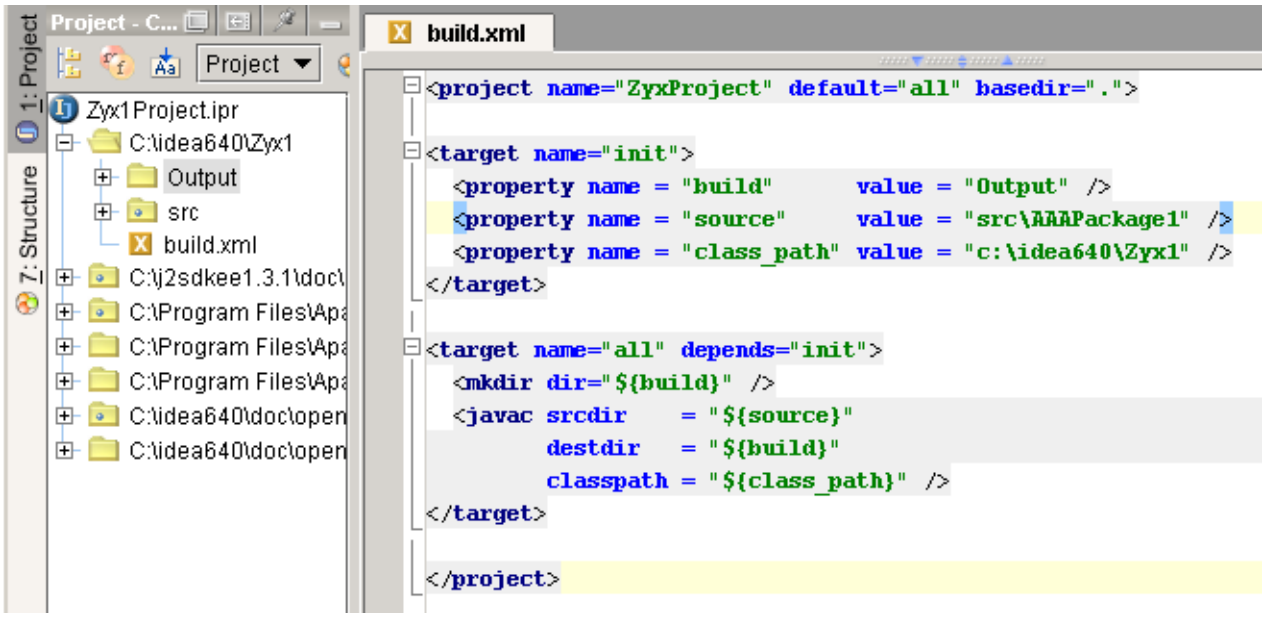


Figure 14.3. build.xml (271)

14.3. Add Ant build to IDEA

14.6. Open the Ant Build tool.
14.7. Click on the +. The dialog “Select Ant build” appears.
14.8. Select build.xml.

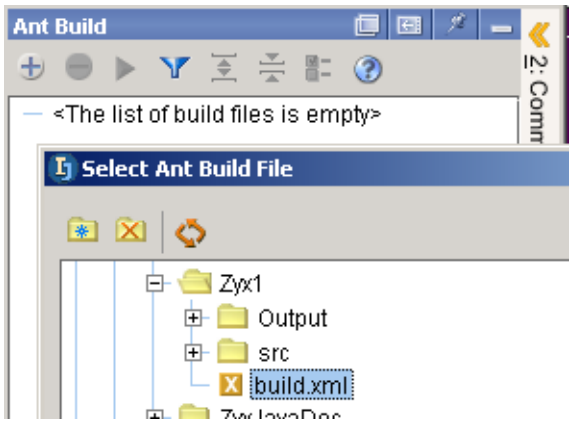


Figure 14.4. build.xml (270)

14.9. Click **OK**. The file is added.

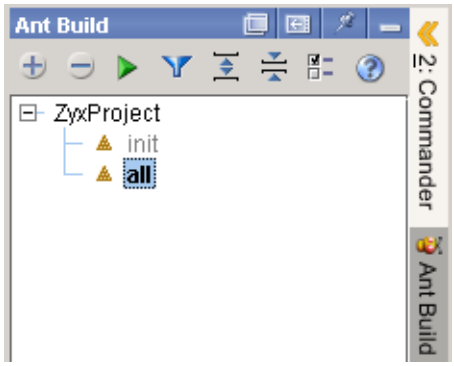


Figure 14.5. Build file added (269)

14.4. Run build

14.10. Click on the **Run**. icon. The files are built.

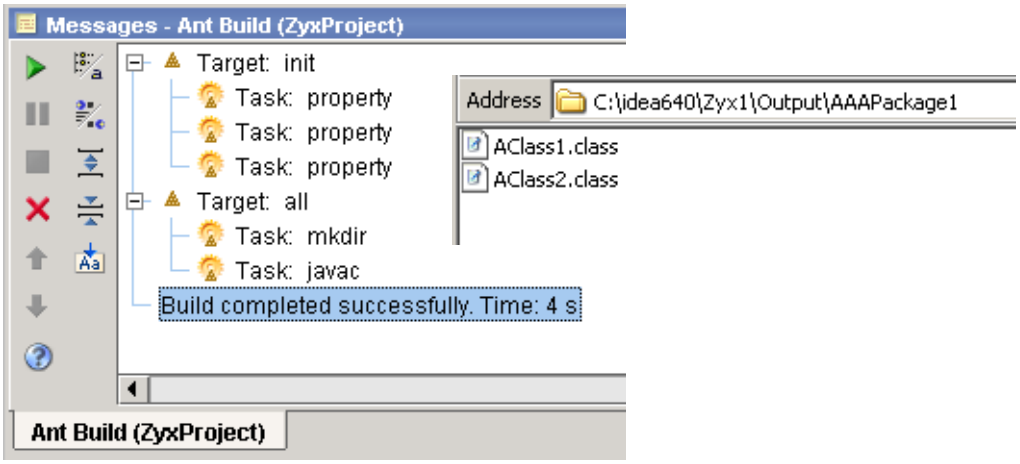


Figure 14.6. Files are built (268,267)

15. Debugger X

20021002TT: text added.

contacts: zheka

Add: remote debugging.

Note: This chapter only describes in detail running/debugging for an Application. The particular detais of running/debugging for other run/debug configurations (Applet, WebApp, etc.) are described in **Part E. Applications (page 283)**.

15.1. Run/Debug configurations

15.1.1. Types

The following 5 types of configurations are available

15.1.1.1. Application

You already created a basic application configuration in **Section 3.6.1. Create application configuration (page 44)**.

15.1.1.2. Applet

Will be introduced in **Chapter 19. Applets XXX (page 287)**.

15.1.1.3. JUnit

Will be introduced in **Chapter 16. JUnit XXX (page 279)**.

15.1.1.4. Remote XXX ??

15.1.1.5. WebApp

Will be introduced in **Chapter 20. Web applications XXX (page 289)**.

15.1.2. Basic operations

15.1.2.1. Add (create)

You already created a basic application configuration in **Section 3.6.1. Create application configuration (page 44)**.

15.1.2.2. Edit

15.1. Click on the **Run** or **Debug** icon (). The dialog appears.
15.2. Make the required changes.
15.3. Click **Apply**.

15.1.2.3. Modify defaults

15.4. Click **Edit Defaults**. The “Application default settings” page appears.

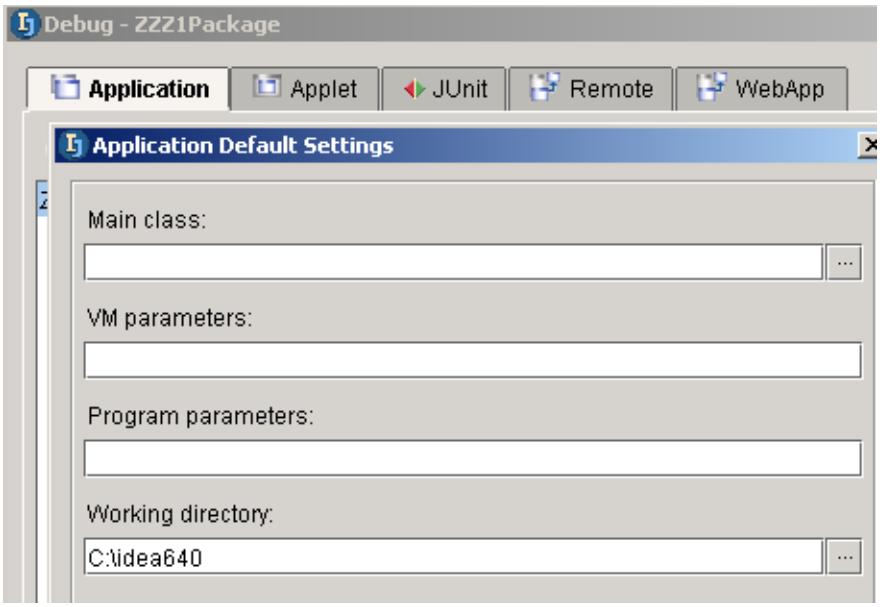


Figure 15.1. Application configuration default settings (289)

15.5. Enter defaults.
15.6. Click **OK**.

15.1.2.4. Copy

15.7. Select the configuration.

15.8. Click on the **Copy configuration** icon (). A copy of the configuration is created.

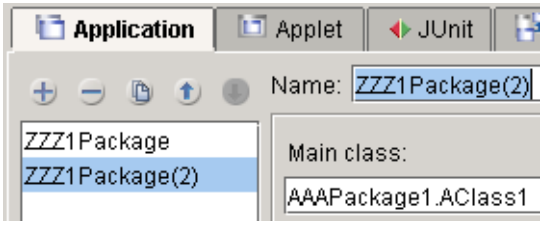


Figure 15.2. Copied configuration (287)

15.1.2.5. Remove (delete)

15.9. Select the configuration.

15.10. Click on the **Remove configuration** icon ( ). The configuration is deleted.

15.1.2.6. Run

15.11. Close the configuration dialog.

15.12. Click on the **Run** icon ( ). The Run configuratuion dialog appears.
15.13. Select a configuration.

15.14. Click **Run** Run. The application is run.

15.1.2.7. Debug

15.15. Close the configuration dialog.

15.16. Click on the **Debug** icon ( ). The Debug configuratuion dialog appears.
15.17. Select a configuration.

15.18. Click **Debug**. Debug. The application is debugged (details of debugging an application are introduced later in this chapter).

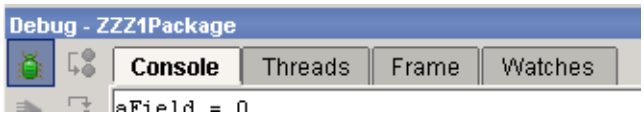


Figure 15.3. Debug dialog opened (283)

15.2. Types of breakpoints

eventually make better example code here.

15.2.1. Line

You set a line breakpoint already in **section 3.7.1. Set breakpoint (page 46)**.

15.2.2. Exception

15.19. Create class:

```
package AAAPackage1;
import java.lang.*;
public class AClass1 {
    public static void main(String[] args) {
      int aField = 0;
      AClass2 c2 = new AClass2();
      for (int i = 0; i < 100000; i++) {
          aField += 1;
          System.out.println("aField = " + aField);
          c2.aMethod();
      }
    }
}
class AClass2 {
    int xxx =1;
    void aMethod() throws ArithmeticException {
      xxx += 1;
      if (xxx == 3) {
         throw new ArithmeticException();
      }
    }
}
```

15.20. Select **Run | View breakpoints...**.
15.21. Select tab **Exception breakpoints**.
15.22. Click **Add**. The dialog “Add exception breakpoint” appears.
15.23. For the name enter **ArithmeticException**.
15.24. Click **OK**. The exception breakpoint is added.

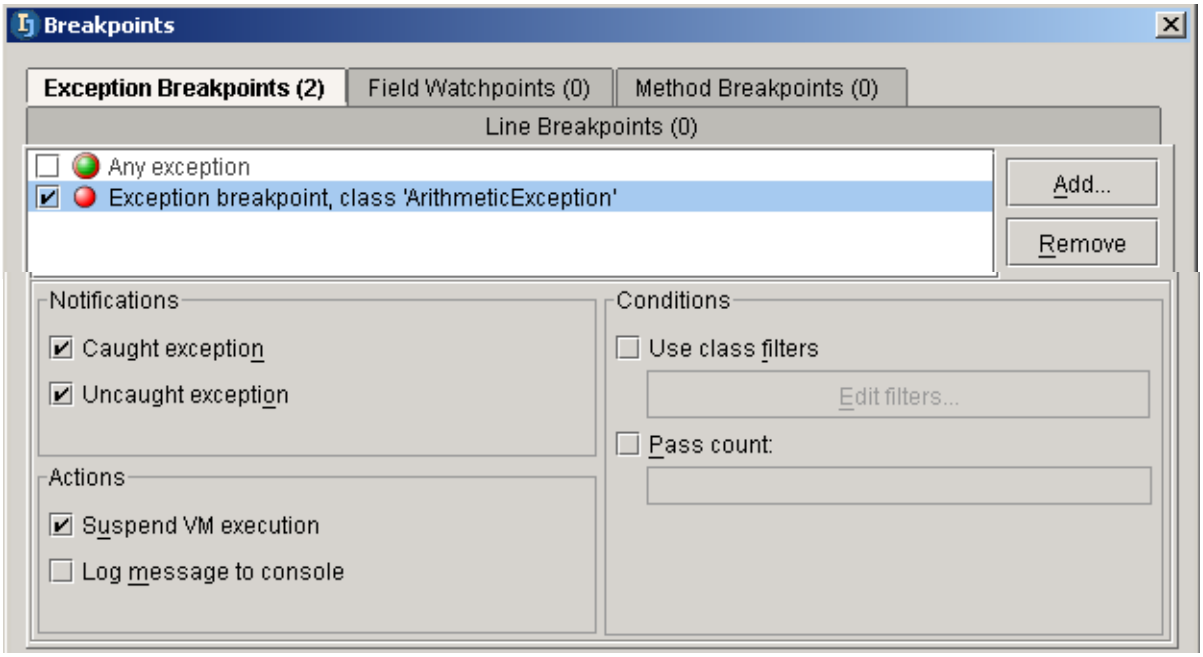


Figure 15.4. Exception breakpoint (282,281)

15.25. Click **Close**.

15.26. Debug the application. Note the debug window.

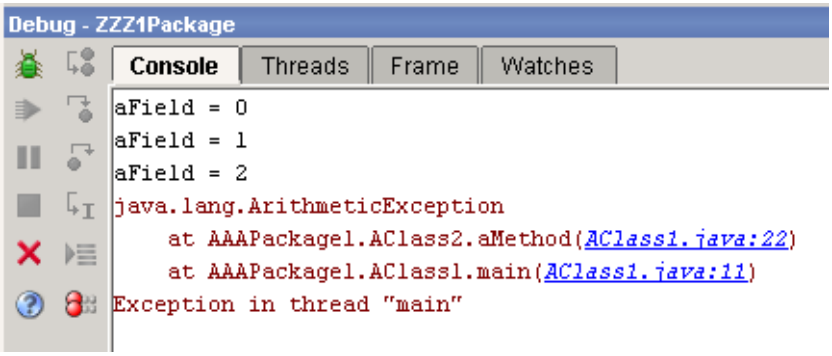


Figure 15.5. Exception info in debug (280)

15.2.3. Field (watchpoints)

15.27. Delete the exception breakpoint.

15.28. Select tab **Field watchpoints**.

15.29. Click **Add**. The dialog “Add field watchpoint” appears.

15.30. For “Fully qualified name of a class” enter **AAAPackage1.AClass2**.

15.31. For “Field name” enter **xxx**.

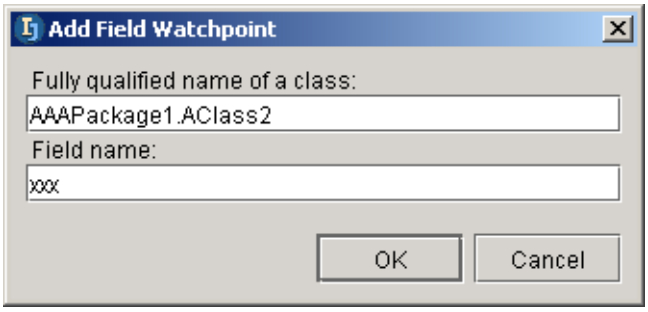


Figure 15.6. Add field watchpoint (279)

15.32. Click **OK**. The field watchpoint is added.

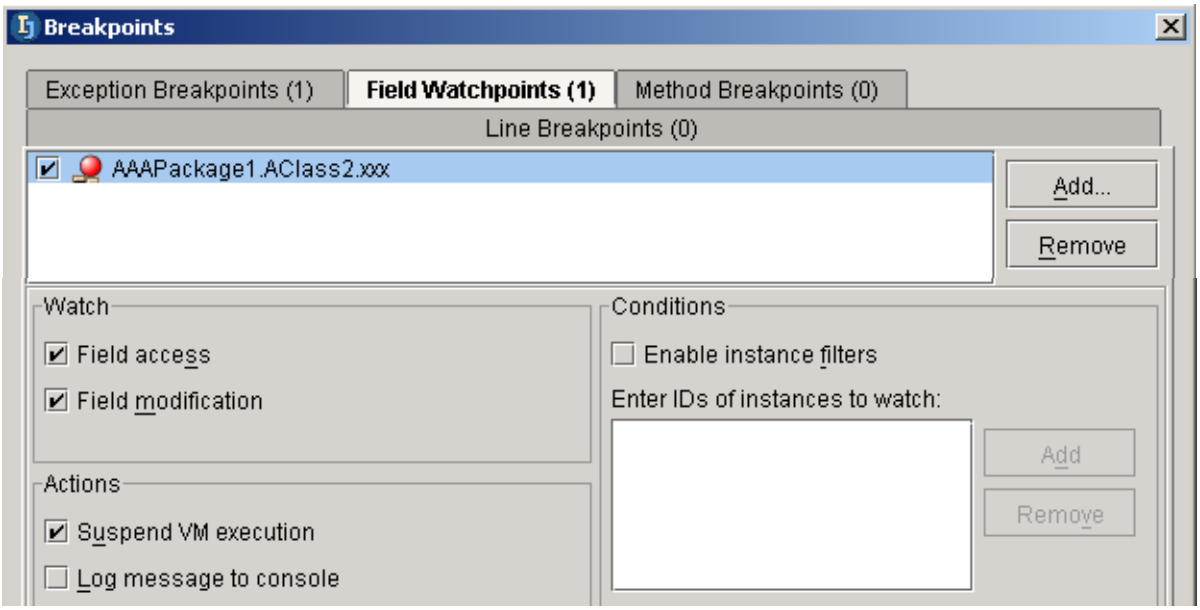


Figure 15.7. Field watchpoint added (278,277)

15.33. Click **Close**.

15.34. Debug the application. Note the debug window.

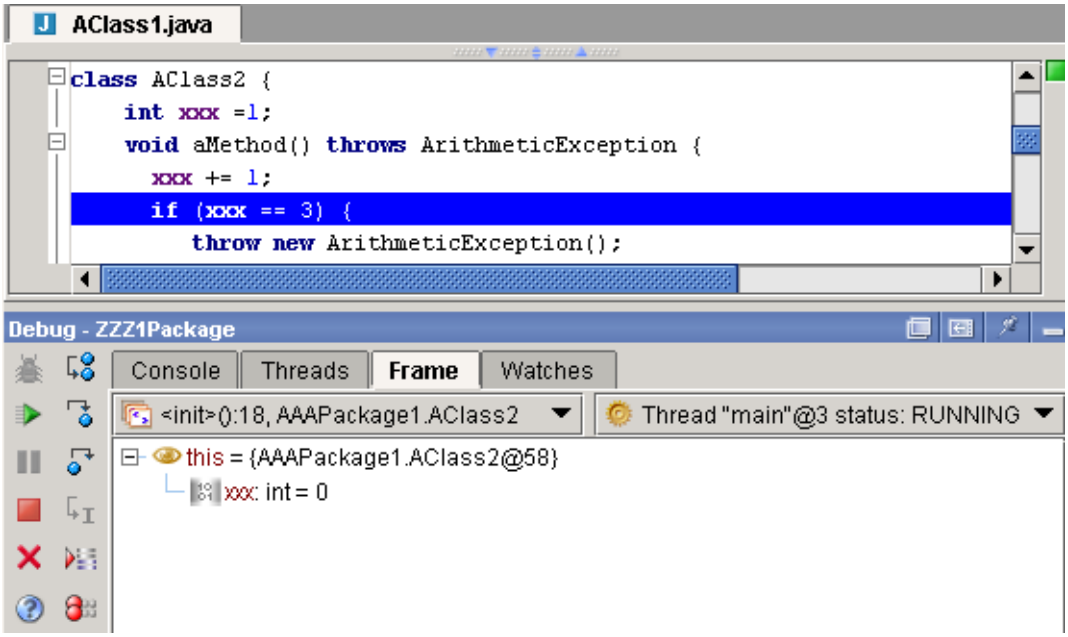


Figure 15.8. Field watchpoint info in debug (276)

15.2.4. Method

15.35. Delete the field watchpoint.
15.36. Place the cursor within the definition for **aMethod()**.
15.37. Select **Run | Add method breakpoint...**.
15.38. Click **Add**. The dialog “Add method breakpoint” appears with the breakpoint defined.

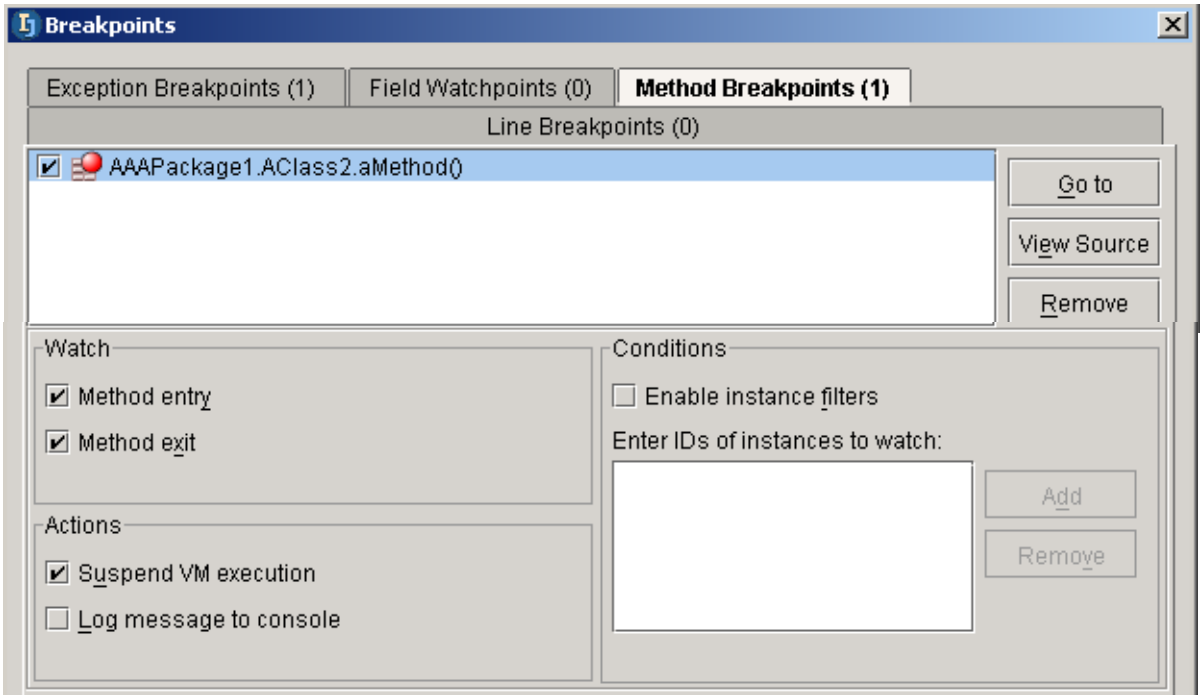


Figure 15.9. Method breakpoint (275,274)

15.39. Click **Close**.

15.40. Debug the application. Note the debug window.

Figure 15.10. xxxxpoint info in debug ()

15.3. Breakpoint actions XXX

15.3.1. Step over / Step into / Step out

15.3.2. Run to cursor

15.3.3. Pause / Resume

15.3.4. Disconnect

15.3.5. Evaluate expression

15.3.6. Show execution point

15.3.7. Toggle

15.3.8. Export threads ??

16. JUnit XXX

contacts: zheka

16.1.

16.1. xxx

Download from **http://www.junit.org/index.htm**.
Unzip.
Set CLASSPATH.
Test

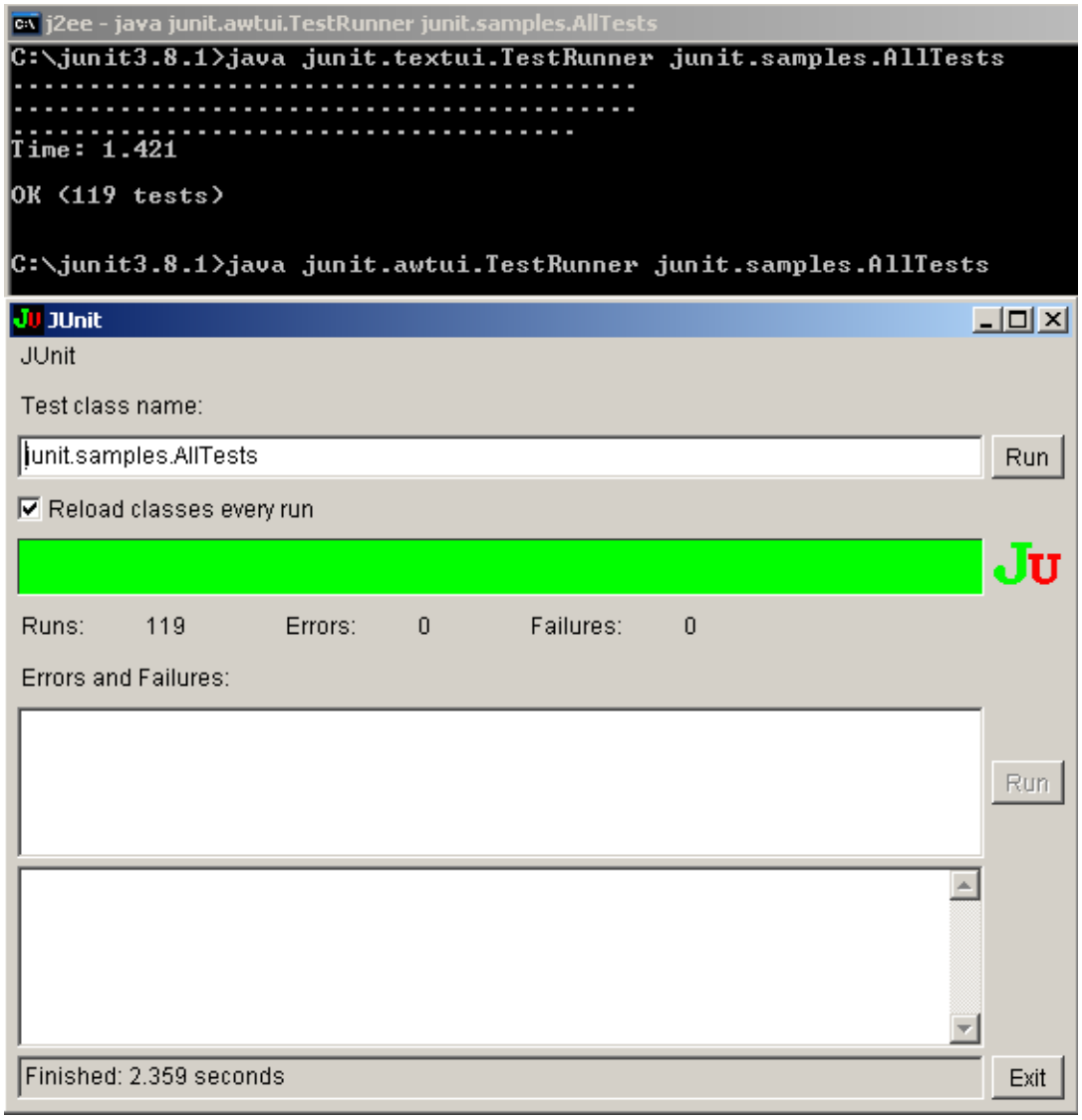


Figure 16.1. Junit test (472)

17. Remote debugging XXX

contacts: ???

17.1. xxx

17.1.
Figure 17.1. xxx ()

Part E. Applications

The chapters in this part describe IDEA's support for various types of applications.

18. Applications XXX (page 285).

19. Applets XXX (page 287).

20. Web applications XXX (page 289). Demonstrates how web applications can be created and managed with IDEA.

21. EJB X (page 295). Demonstrates how to create EJB using IDEA.

18. Applications XXX

contacts: valentin

18.1. xxx

18.1.
Figure 18.1.

19. Applets XXX

contacts: valentin

19.1. xxx

19.1.
Figure 19.1.

20. Web applications XXX

contacts: mike, and aleksei kudravtsefv

20.1.

Figure 20.1.

20.1. Install Tomcat XXX

20.2. JSP XXX

20.3. Tags

migrate??? to different jdk.... valya
20020911TTT: created.

example from http://developer.java.sun.com/developer/technicalArticles/xml/WebAppDev3/.

20.3.1. Create HelloTag.java

20.2. 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;
    }
}
```

20.3. Compile.

20.3.2. Create mytaglib.tld

20.4. 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 tab 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

20.5. 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

20.6. In IE: Open **http://localhost:8080/examples/jsp/Hello.jsp**.

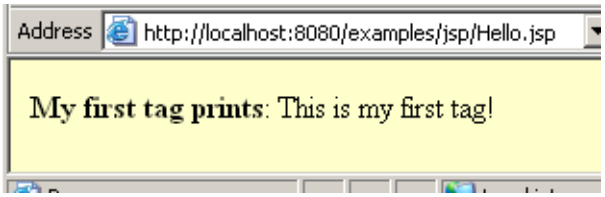


Figure 20.2. xxx (473)

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 reallly have no idea what i am doing), but documents how to do the sun example.

21.1. Install / start J2EE

21.1.1. Download / install J2EE from sun

21.1. Download **j2sdkee-1_3_1-win.exe** (Windows install) from **http://java.sun.com/j2ee/download.html#sdk**.

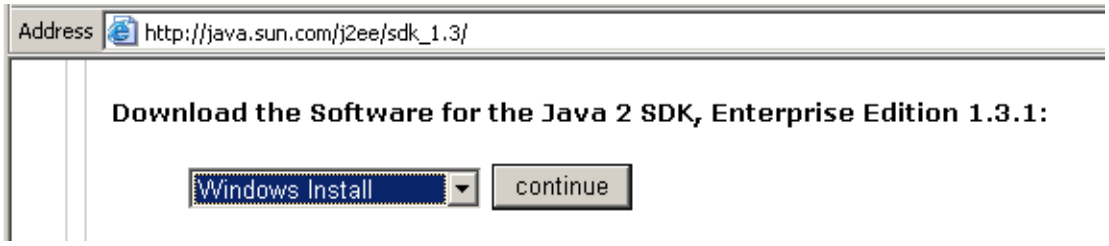


Figure 21.1. xxx (485)

21.1.2. Install

21.2. Double-click on **j2sdkee-1_3_1-win.exe**. Follow the directions to install (use default settings).
21.3. Set JAVA_HOME system variable to
21.4. **Start | Settings | Control Panel**.
21.5. Double-click **System**.
21.6. In tab **Advanced**: Select **Environment variables**.
21.7. Add System variable **JAVA_HOME** with value **c:\jdk1.4.0_01** (or the location of your JDK if different).
21.8. Add System variable **J2EE_HOME** with value **C:\j2sdkee1.3.1**.

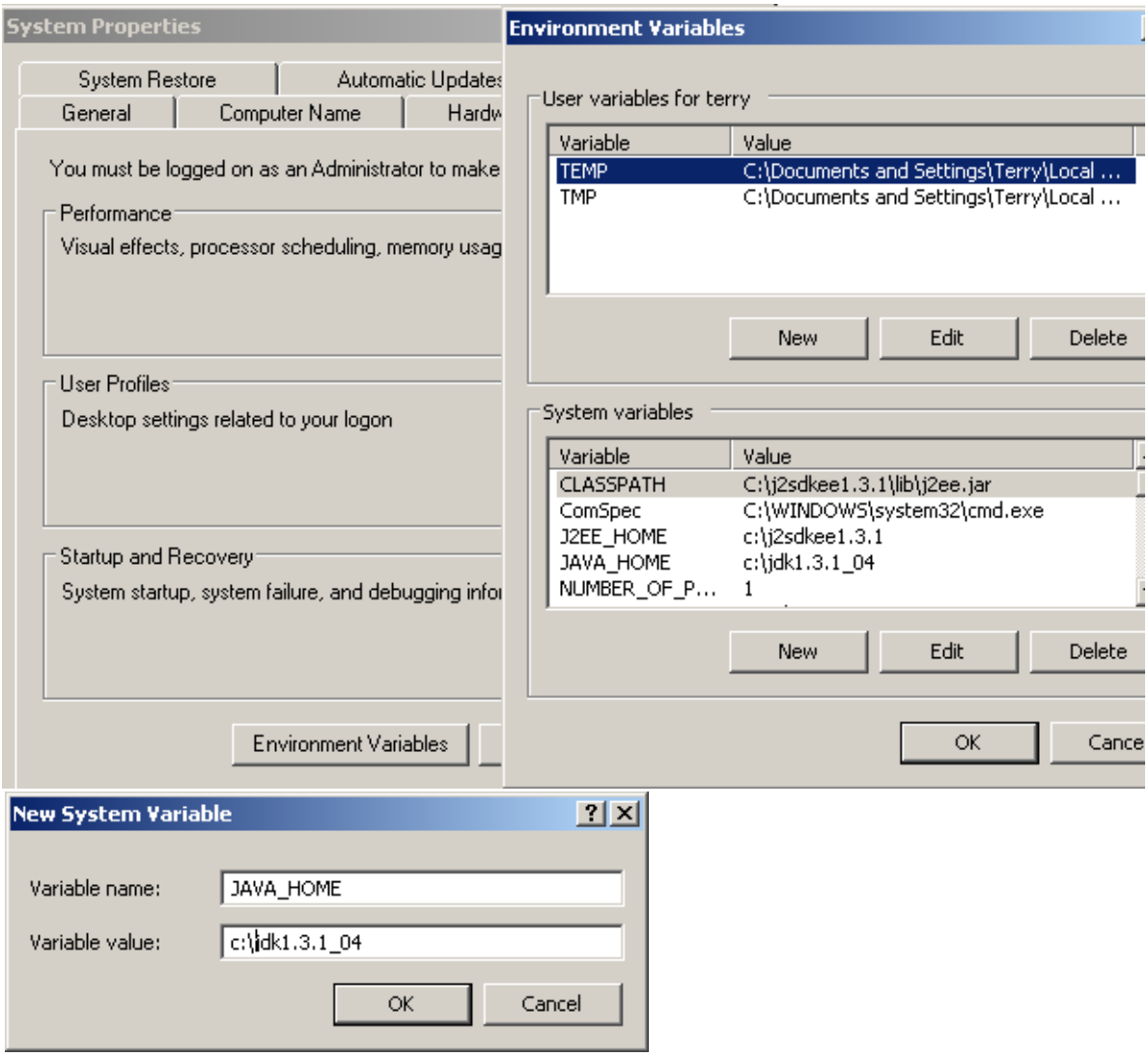


Figure 21.2. xxx (497,484)

21.1.3. Start J2EE server

21.9. Start J2EE server.

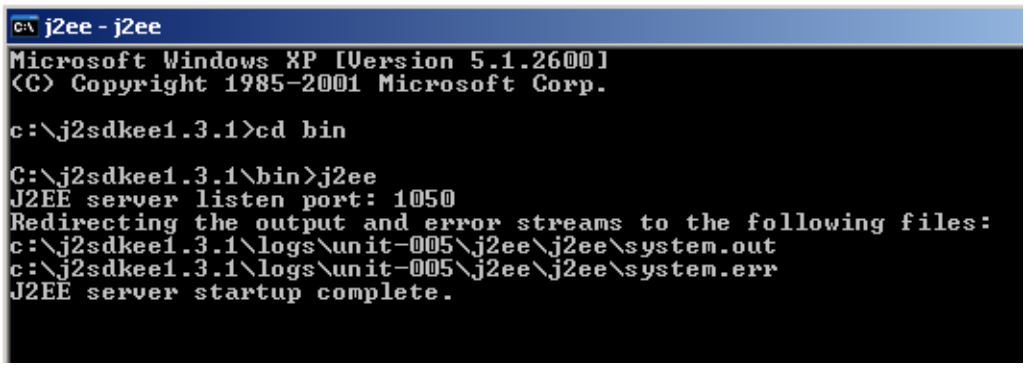


Figure 21.3. xxx (496)

21.10. Open **http://localhost:8000** to verify.

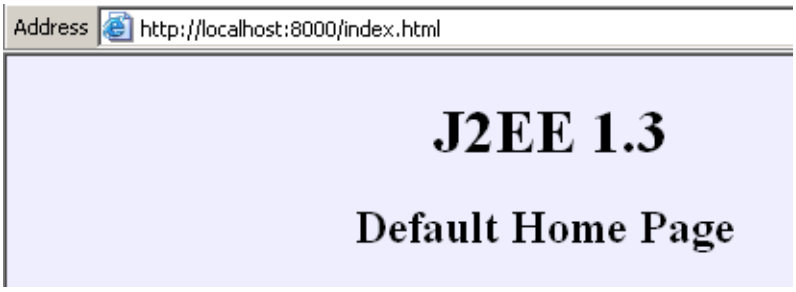


Figure 21.4. xxx (483)

21.1.4. Start Cloudscape

21.11. Start Cloudscape server.

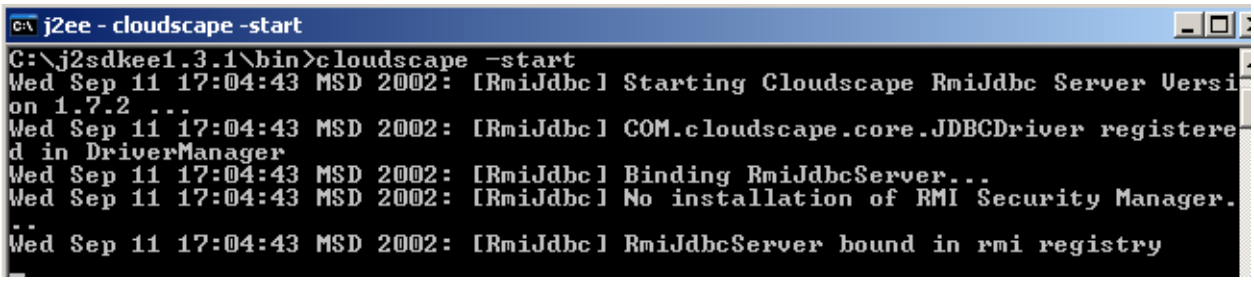


Figure 21.5. xxx (495)

21.12. Open **http://???** to verify.

21.2. Deploy Sun example application

21.2.1. Start deploy tool

21.13. Double-click on **C:\j2sdkee1.3.1\bin\deploytool.bat** to start the deploy tool.

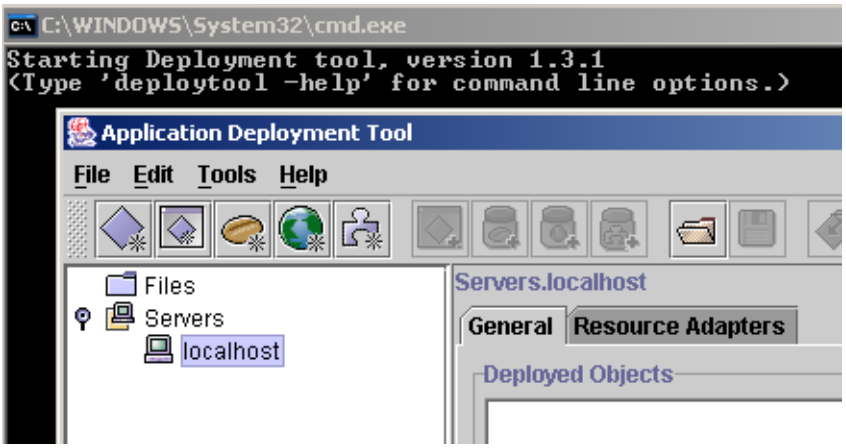


Figure 21.6. xxx (494)

21.2.2. Open the application

Select File | Open.
Select C:\j2sdkee1.3.1\doc\samples\cmpcustomer\cmpcustomer.ear.

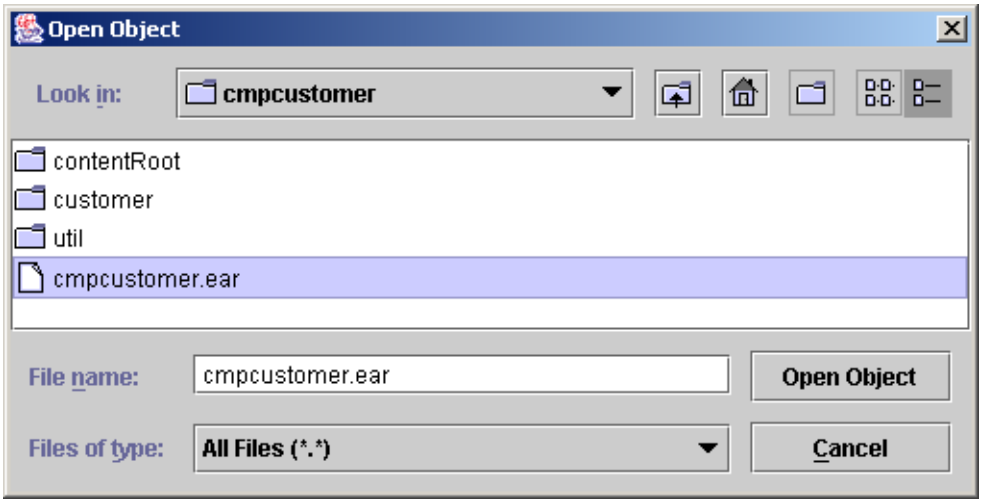


Figure 21.7. xxx (482)

Click Open Object.

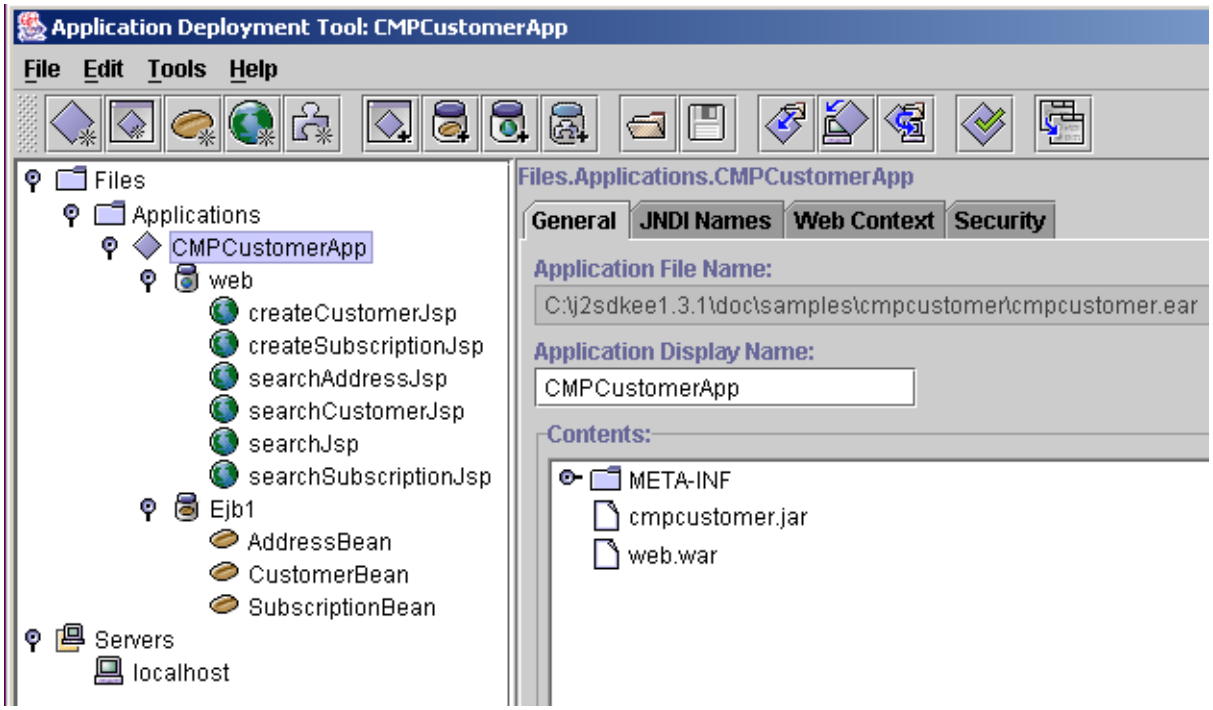


Figure 21.8. xxx (492)

21.2.3. Generate SQL

from 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

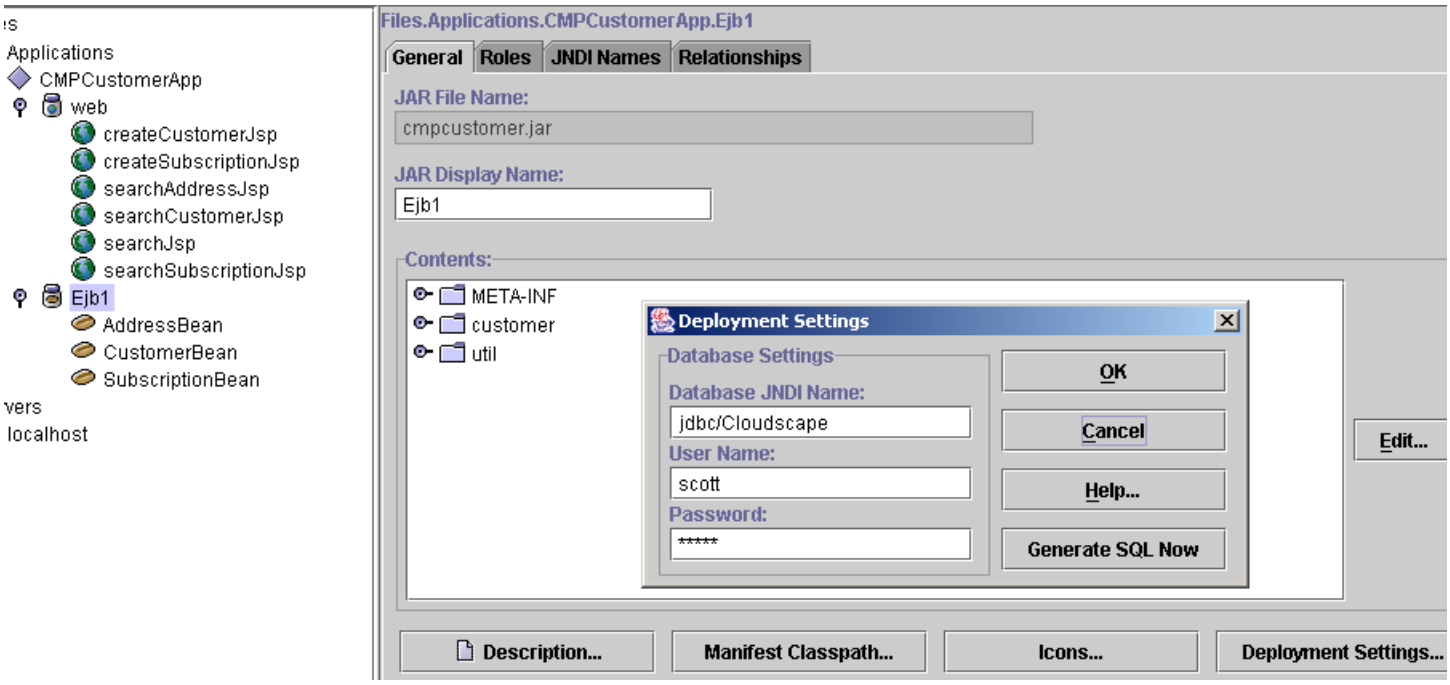


Figure 21.9. xxx (481)

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.

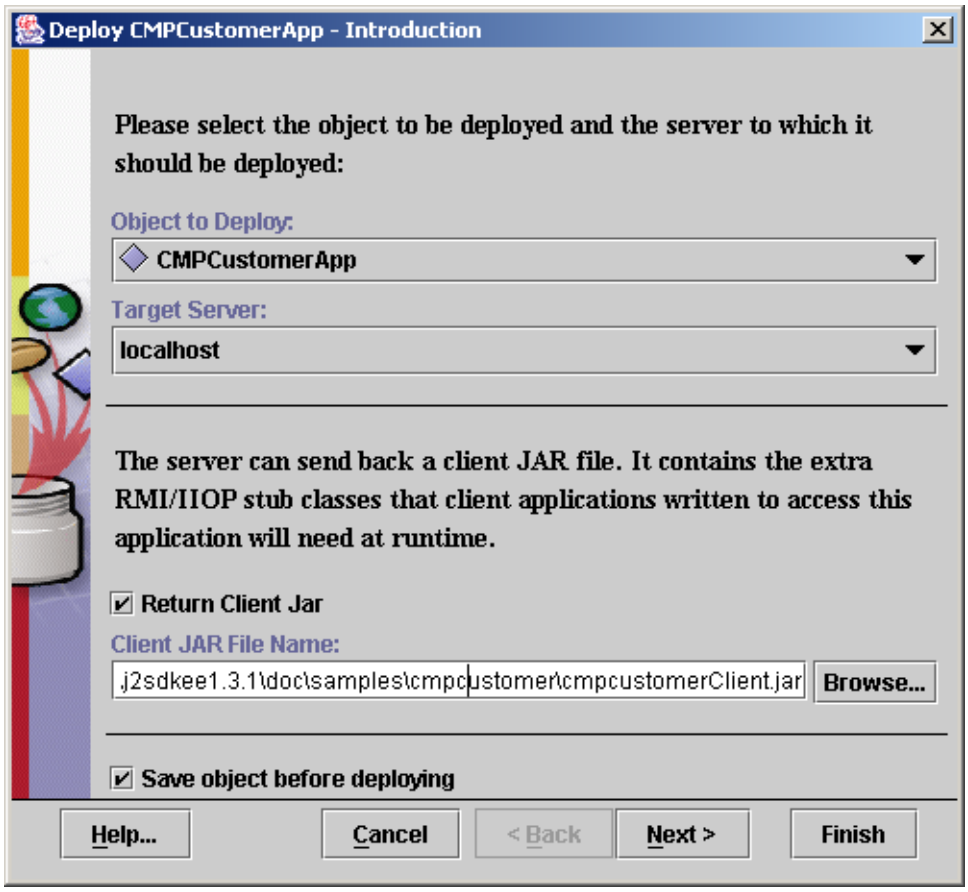


Figure 21.10. xxx (481)

Click Next.

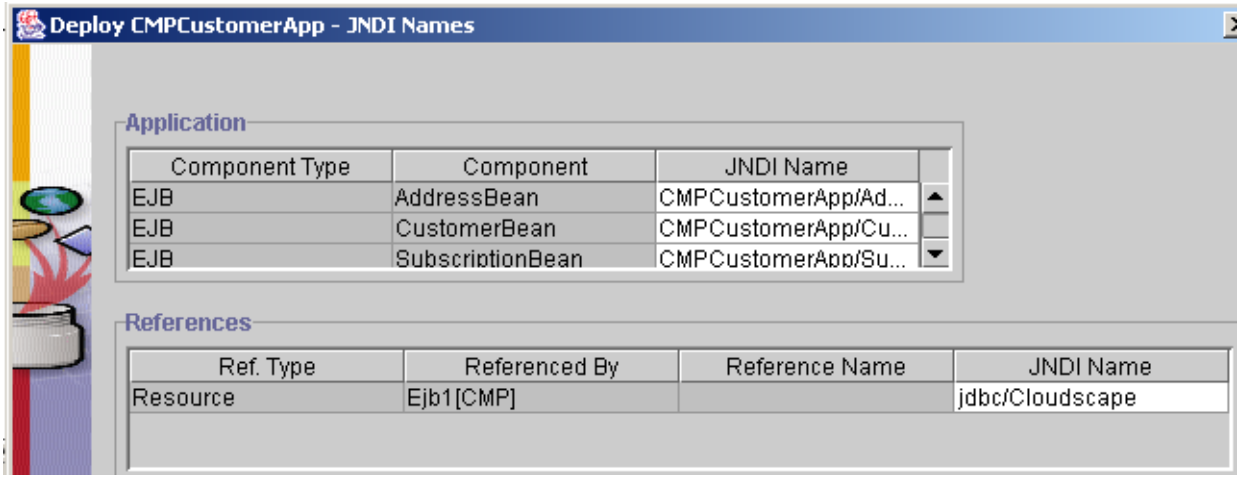


Figure 21.11. xxx (479)

Click Next.

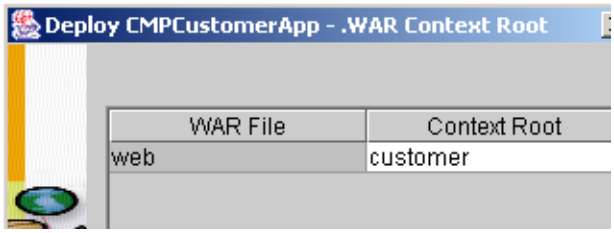


Figure 21.12. xxx (478)

Click Finish. Appears Deployment Progress dialog.

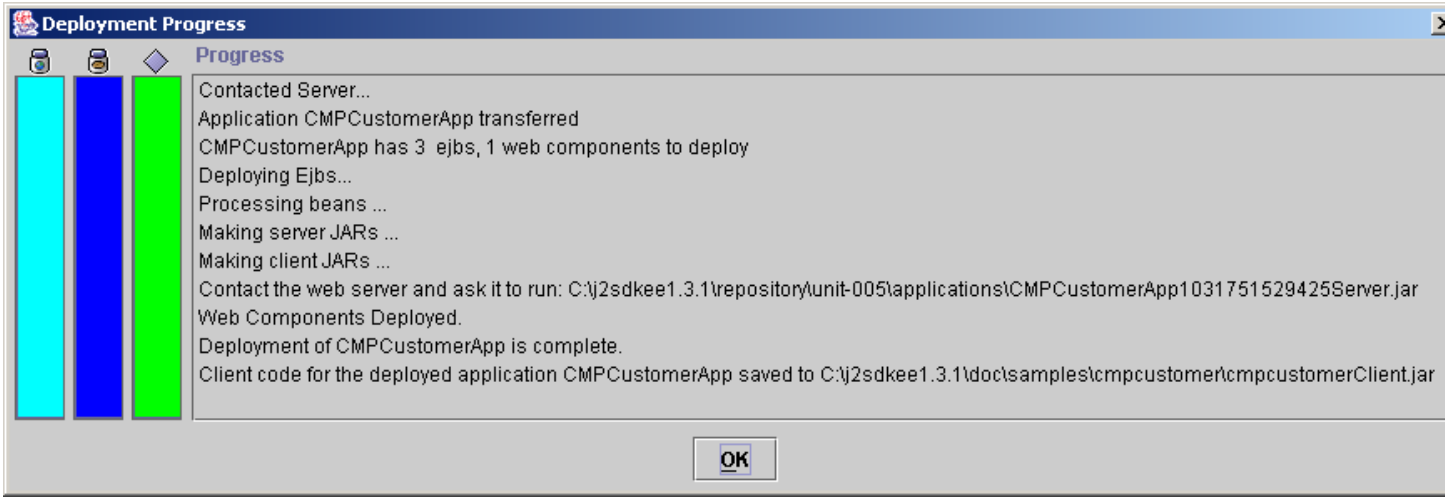


Figure 21.13. xxx (477)

Click OK.

21.14. In IE: Open http://localhost:8000/customer/index.html.

Figure 21.14. xxxxut (476)

21.3. Set IDEA EJB project properties

21.15. In IDE: Select **File | Project properties**.
21.16. Select tab **EJB**.
21.17. Click **+**. The dialog “Create EJB Group” appears.
21.18. For Name: Enter **EJBGroup1**.
21.19. For path to ejb-jar.xml: Enter xxx.

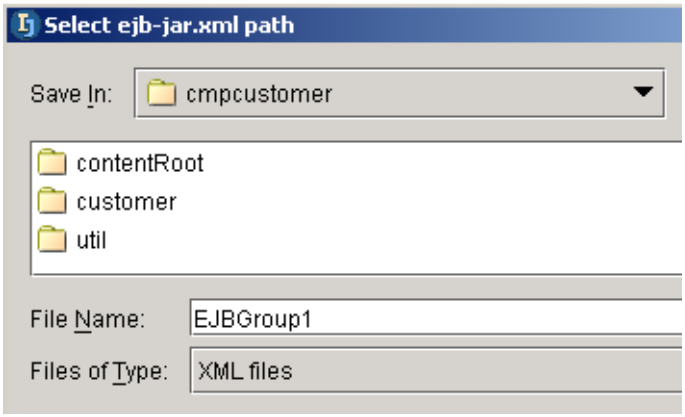


Figure 21.15. xxxxut (475)

21.20. Check Sourcepath.
21.21. For Sourcepath enter xxx:

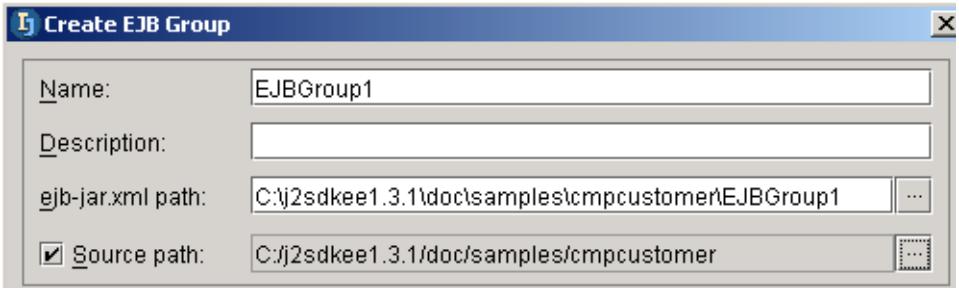


Figure 21.16. xxxxut (474)

21.22. Click **OK**.

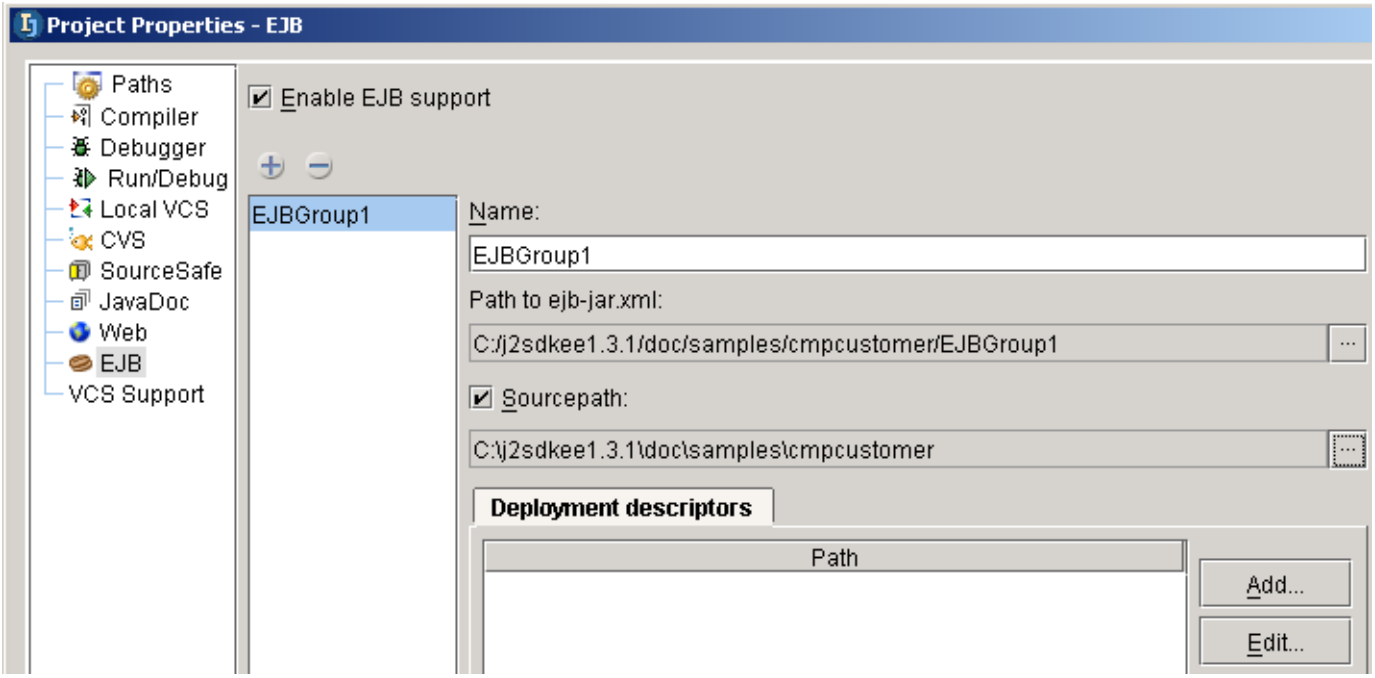


Figure 21.17. xxxxut (491)

21.23. Click **OK**.

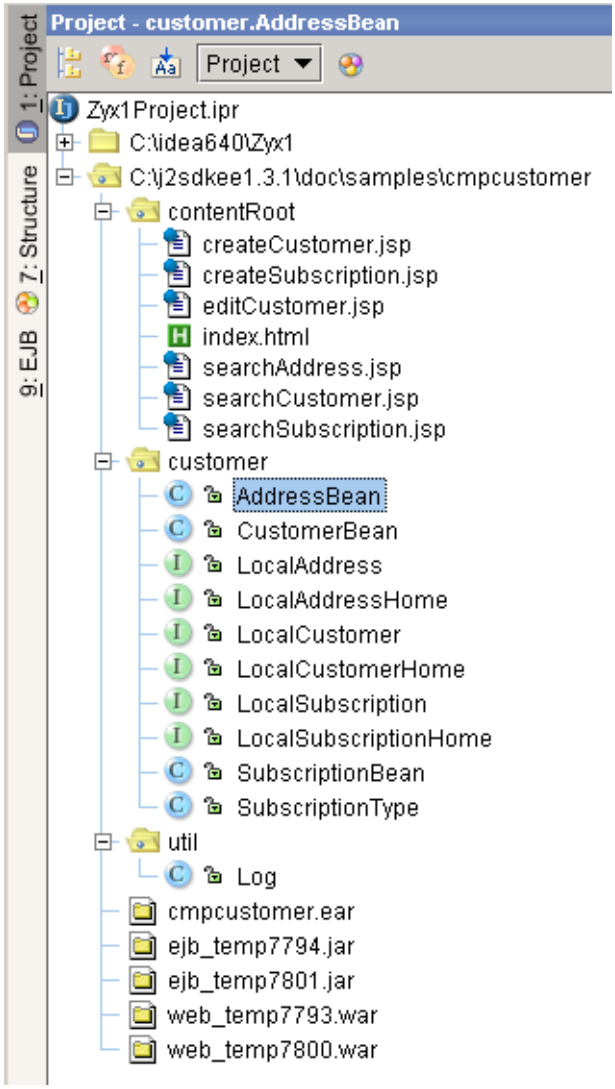


Figure 21.18. xxxxut (490)

21.4. Modify source in IDEA

Figure 21.19. xxxxut (489)

21.5. Redeploy

21.24. Select **Tools | Update and redeploy**.

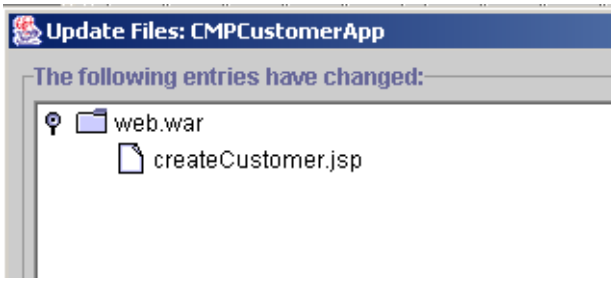


Figure 21.20. xxxxut (488)

21.25. Click **OK**. The dialog “Deployment progress” appears.

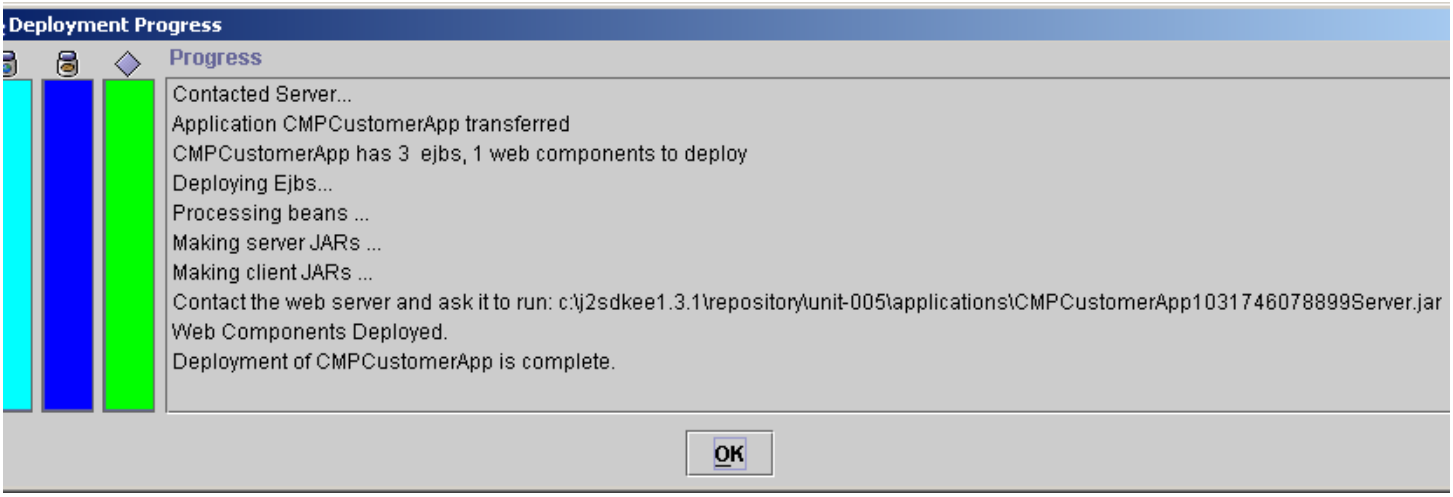


Figure 21.21. xxxxut (487)

21.26. **OK**.

21.27. Click on Create a new customer.

Figure 21.22. xxxxut (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

20020919TTT: description of idea examples added.
see C:\idea640\doc\openapi\plugins.html
Consult ?? for details.

22.1. Install a plugin

22.1.1. Find a plugin

22.1. Open www.intellij.org. The page lists (among other things) available plugins for IDEA.

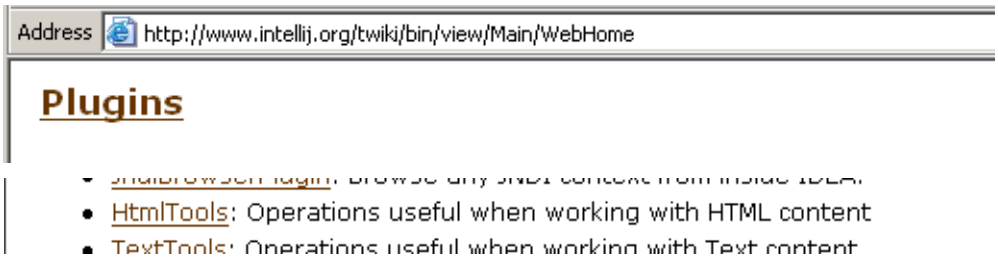


Figure 22.1. www.intellij.org start page (538,537)

22.2. Click on HtmlTools. A listing of available HTML tools appears.

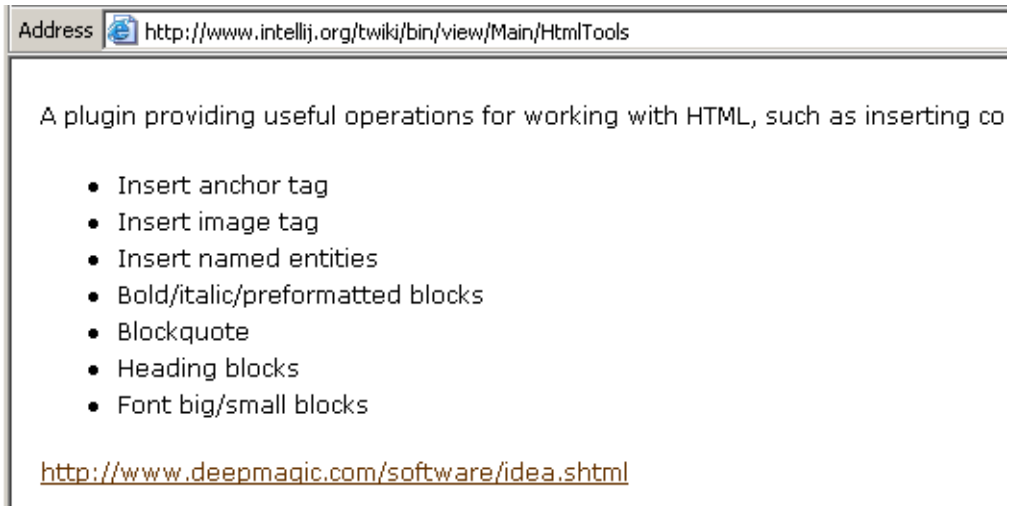


Figure 22.2. HtmlTools (536)

22.3. Click on the link. Private website.

22.4. 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

22.5. Unzip the file.
22.6. Copy html.jar to C:\idea640\plugins.
22.7. Restart IDEA. Note the HTML menu item in the main menu.

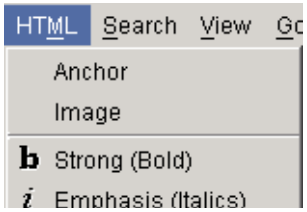


Figure 22.3. HTML main menu item (535)

22.1.3. Using the plugin

22.8. Create an HTML file.
22.9. Add the text shown in the following diagram.

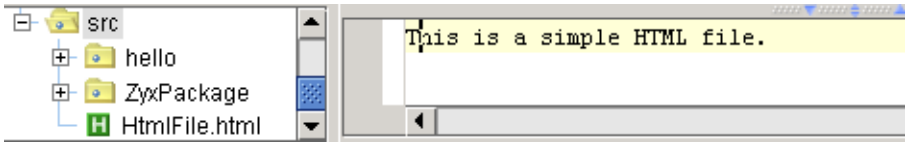


Figure 22.4. Simple HTML file (534)

22.10. Select a word in the text.
22.11. From the main menu select HTML / Strong(Bold). The html text is added.

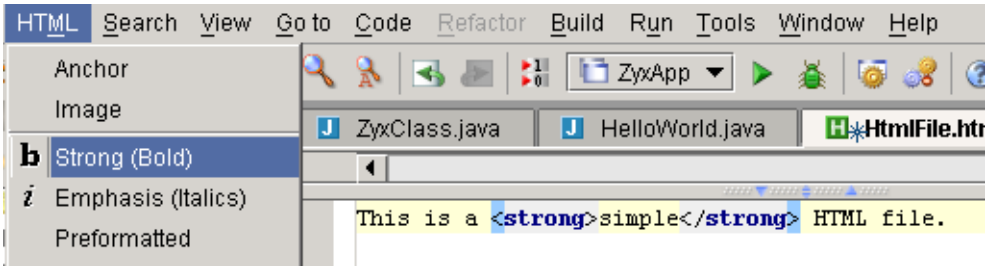


Figure 22.5. HTML bold tags added to text (533)

22.2. Create a plugin

Note: The code for the following examples are available in dir **%idea%\doc\openapi\examples**.
20020919TTT: you must first add the idea640\lib jars files to the class path to compile.

22.2.1. Application / project plugin

The code for this example is available in dir **%idea%\doc\openapi\examples\plugin**.

22.2.1.1. Create jar

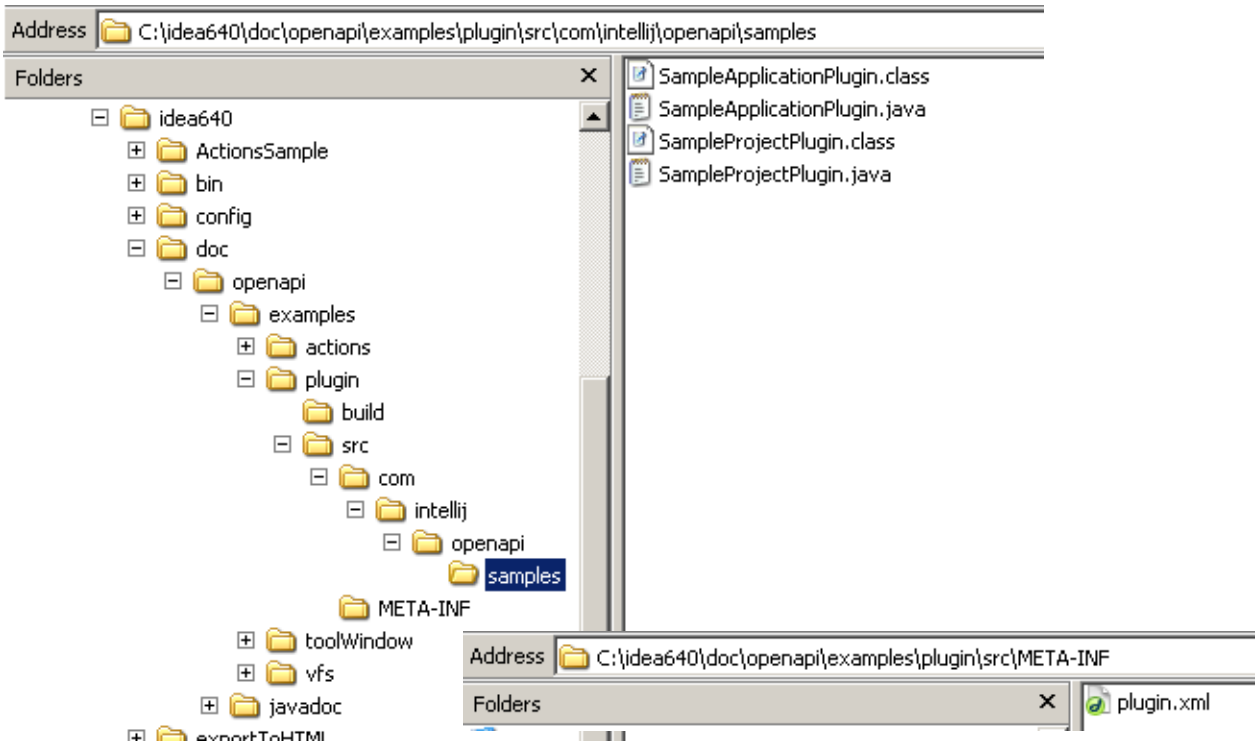


Figure 22.6. xxx (465,464)

22.12. Create and compile file **com\intellij\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 initComponent() {
    System.out.println("SampleProjectPlugin: initComponent");
  }

  /**
   * This method is called on plugin disposion.
   */
  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";
  }
}
```

22.13. 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 initComponent() {
```

```
    System.out.println("SampleApplicationPlugin: initComponent");
  }

  /**
   * This method is called on plugin disposion.
   */
  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";
  }
}
```

22.14. Create file **META-INF\plugin.xml**:

```
- <idea-plugin>
- <!--
 plugin name
-->
<name>Sample</name>
- <!--
 description
-->
<description>SamplePlugin</description>
- <!--
 plugin versuin
-->
<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>
```

22.15. Compile.

22.16. Pack the files into **Sample.jar**.

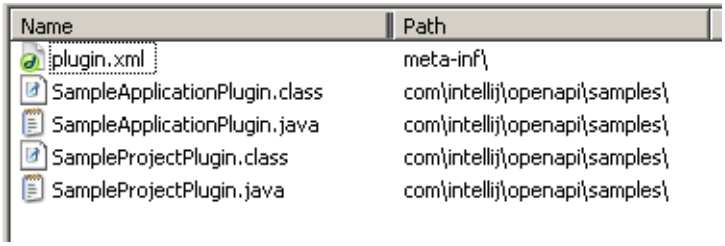


Figure 22.7. Sample.jar contents (463)

22.2.1.2. Install plugin

22.17. Close IDEA (if open).

22.18. Copy Sample.jar to **C:\idea640\plugins**.

22.2.1.3. Test plugin

22.19. Start IDEA.

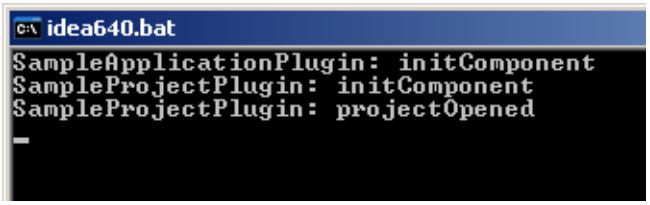


Figure 22.8. xxx (462)

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

22.20. Create and compile file **com\intellij\openapi\samples\ActionsPlugin.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 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 initComponent() {
  }

  /**
   * This method is called on plugin disposion.
   */
  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";
  }
}
```

22.21. Create and compile file **com\intellij\openapi\samples\HelloWorldAction.java**:

```
package com.intellij.openapi.samples;

import com.intellij.openapi.actionSystem.AnAction;
import com.intellij.openapi.actionSystem.AnActionEvent;
import com.intellij.openapi.actionSystem.DataConstants;
import com.intellij.openapi.ui.Messages;
import com.intellij.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());
  }
}
```

22.22. 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);
    }
  }
}
```

22.23. Create file **META-INF\plugin.xml**:

```
- <idea-plugin>
- <!--
 Plugin name
-->
<name>ActionsSample</name>
- <!--
 Description
-->
<description>ActionSamplePlugin</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
-->
- <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>
```

22.24. Compile.

22.25. Pack the files into **ActionsSample.jar**.

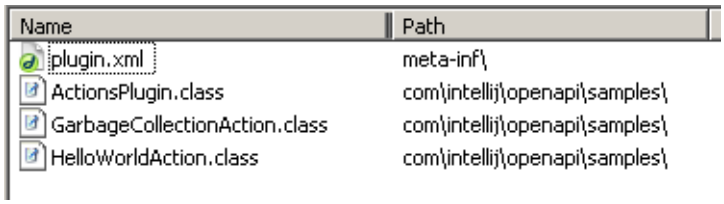


Figure 22.9. ActionsSample.jar contents (461)

22.2.2.2. Install plugin

22.26. Close IDEA (if open).

22.27. Copy ActionsSample.jar to **C:\idea640\plugins**.

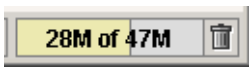
22.2.2.3. Test plugin

22.28. Start IDEA.

22.29. Select **Sample | HelloWorld**. An information dialog appears.

Figure 22.10. xxx (460,459)

22.30. Click **OK**.

22.31. Select **Sample | Collection garbage**. The garbage is collected (note the difference in the heap size 

).

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

22.32. Create and compile file **com\intellij\openapi\samples\SimpleToolWindowPlugin.java**:

```
package com.intellij.openapi.samples;

import com.intellij.openapi.components.ProjectComponent;
import com.intellij.openapi.project.Project;
import com.intellij.openapi.wm.ToolWindow;
import com.intellij.openapi.wm.ToolWindowAnchor;
import com.intellij.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 initComponent() {
    // 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);
  }
}
```

22.33. 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>
```

22.34. Compile.
22.35. Pack the files into **SimpleToolWindow.jar**.

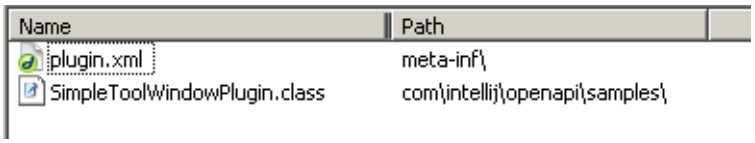


Figure 22.11. SimpleToolWindow.jar contents (457)

22.2.3.2. Install plugin

22.36. Close IDEA (if open).
22.37. Copy SimpleToolWindow.jar to **C:\idea640\plugins**.

22.2.3.3. Test plugin

22.38. Start IDEA. Note the tool window SimpleToolWindow.
22.39. Open **SimpleToolWindow**.

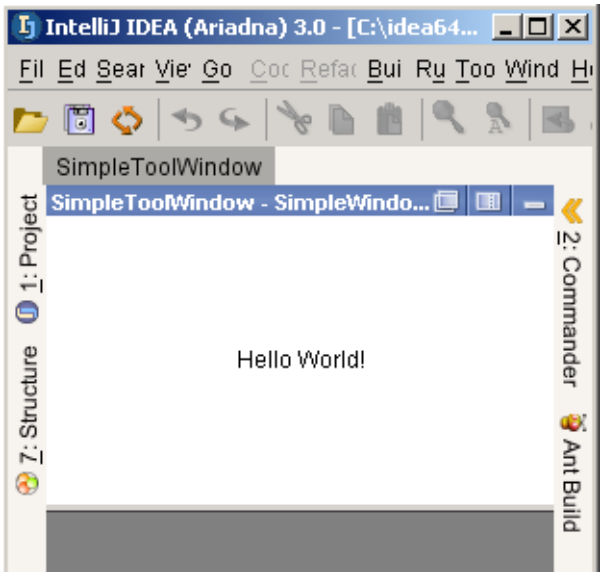


Figure 22.12. xxx (456)

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

22.40. 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 initComponent() {
    // 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);
    }
  }
}
```

22.41. 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>
```

22.42. Compile.

22.43. Pack the files into **VfsSample.jar**.

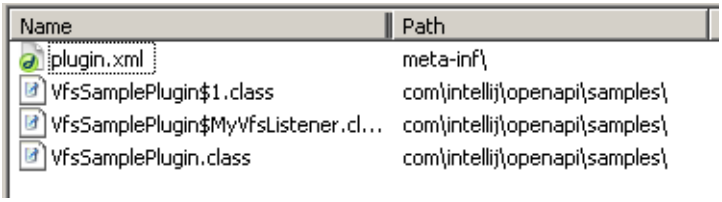


Figure 22.13. VfsSample.jar contents (455)

22.2.4.2. Install plugin

22.44. Close IDEA (if open).

22.45. Copy VfsSample.jar to **C:\idea640\plugins**.

22.2.4.3. Test plugin

22.46. Start IDEA. Note that the number of files are shown in the DOS window.

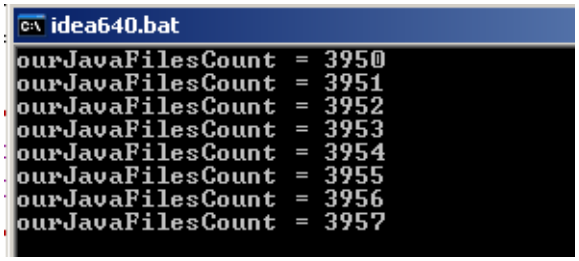


Figure 22.14. xxx (454)

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

23.1. From the main menu select **Tools / IDE options**.
23.2. Select **External Tools**.
23.3. Click **Add**. The dialog **Edit Tool** appears.
23.4. For **Name** enter “ZyxTool”.
23.5. For **Group** enter 'ZyxGroup”.
23.6. For **Description** enter “ZyxTool description”.
23.7. For **Program** click on the dir button. The dialog “Select path” appears.
23.8. Select a tool.

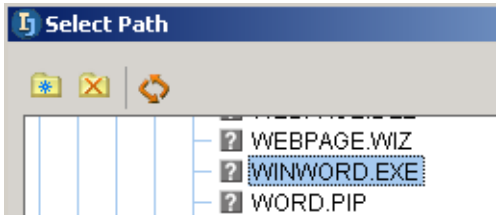


Figure 23.1. Selecting the external tool (532)

23.9. Click **OK**. The external tools info has been entered.

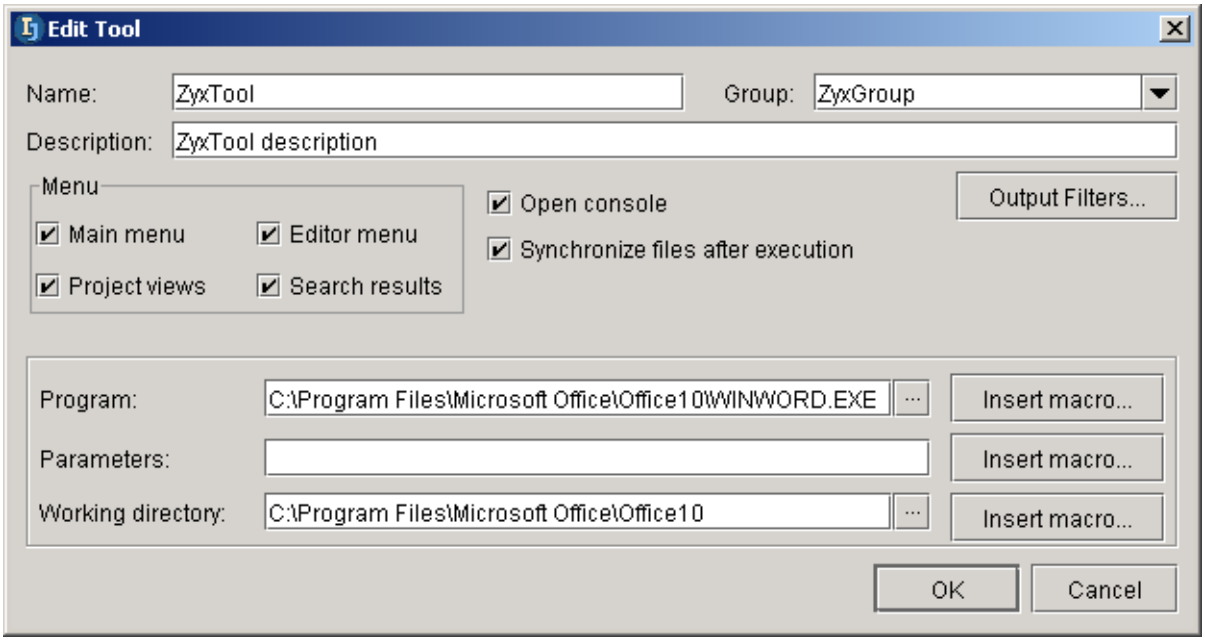


Figure 23.2. External tool settings (531)

23.10. Click **OK**. The tool is now included in the list.

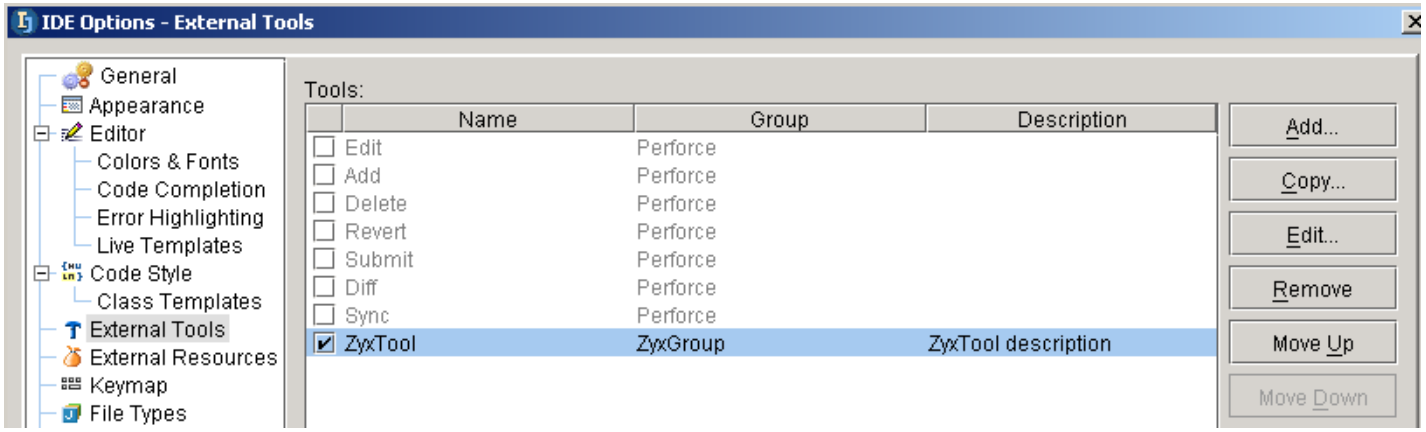


Figure 23.3. Tool in list of external tools (530)

23.11. Click **OK**.

23.2. Open

23.12. Select from the main menu **Tools / ZyxGroup / ZyxTool**.

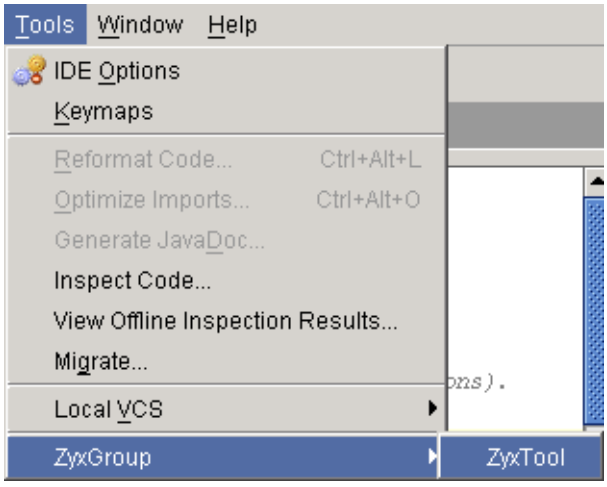


Figure 23.4. External tool in Tools menu (529)

The tool is opened. A message appears in IDEA.

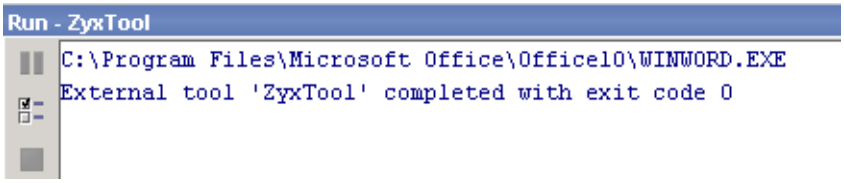


Figure 23.5. Message about external tool being opened (528)

24. External Resources XXX

contacts: mike

24.1. XXX

24.1.
Figure 24.1.

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 337).

26. Trouble shooting XXX (page 339).

27. Keymaps X (page 341).

28. Glossary XXX (page 345).

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 341)**
- **27.2. Create (copy and modify) (page 342)**

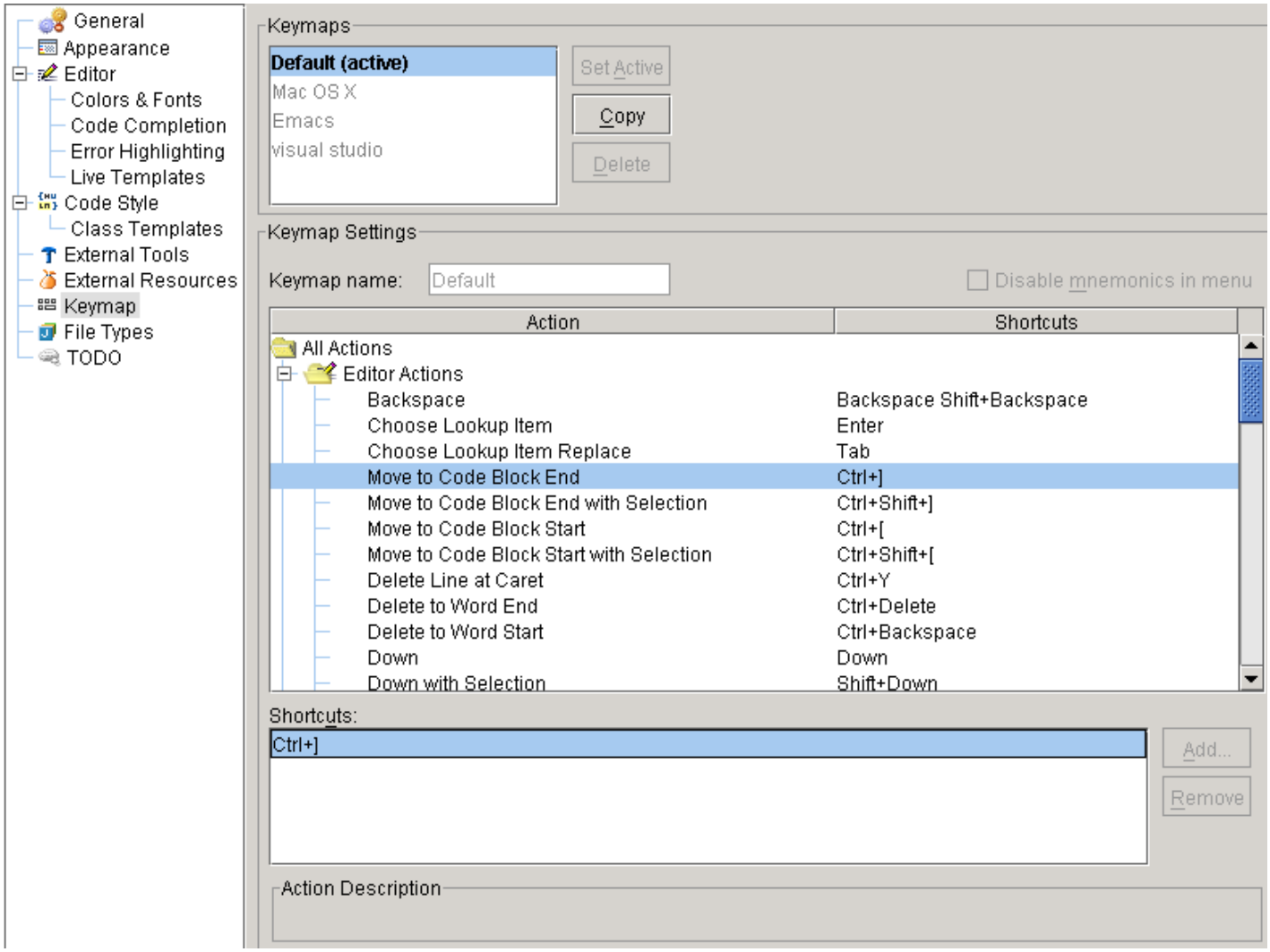


Figure 25.1. IDEA Options / Keymaps (425)

27.1. Select Active

25.1. Select a Keymap.
25.2. Click **Set Active**.

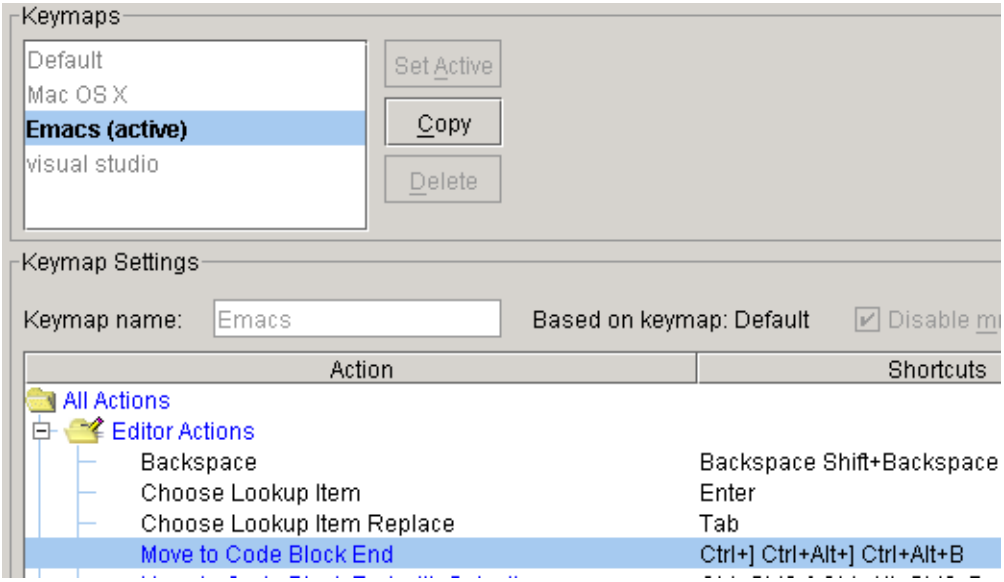


Figure 25.2. Set active keymap (424)

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

25.3. Select the **Default** keymap.
25.4. Click **Copy**. The question “Make the new keymap active?” appears.
25.5. Click **Yes**.
25.6. Change the “Keymap name” to **MyKeymap**.
25.7. Select **All actions / Editor actions / Move to Code Block End**.
25.8. In “Shortcuts”: Select **Ctrl+]**.
25.9. Click **Add**. The dialog “Enter shortcut” appears.
25.10. Press simultaneously **Ctrl** and \. The key combination appears in the dialog.

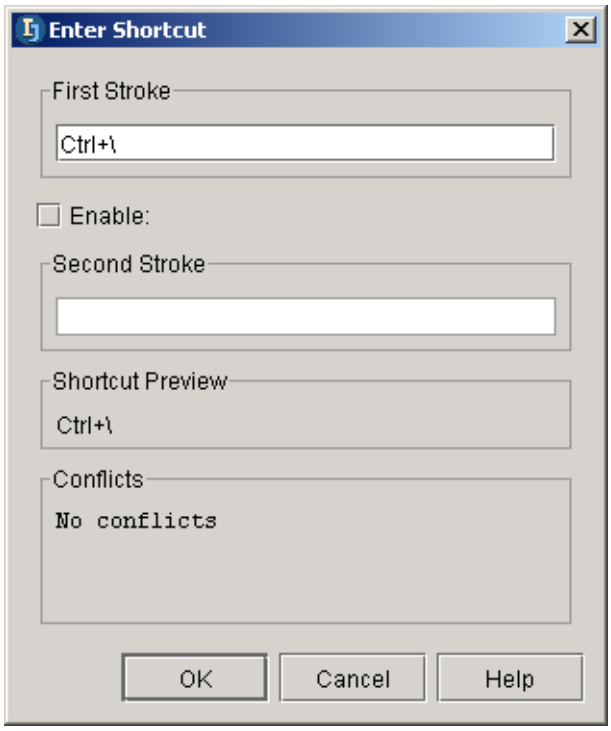


Figure 25.3. New keystroke (423)

25.11. Click **OK**. The keystroke appears in the list.

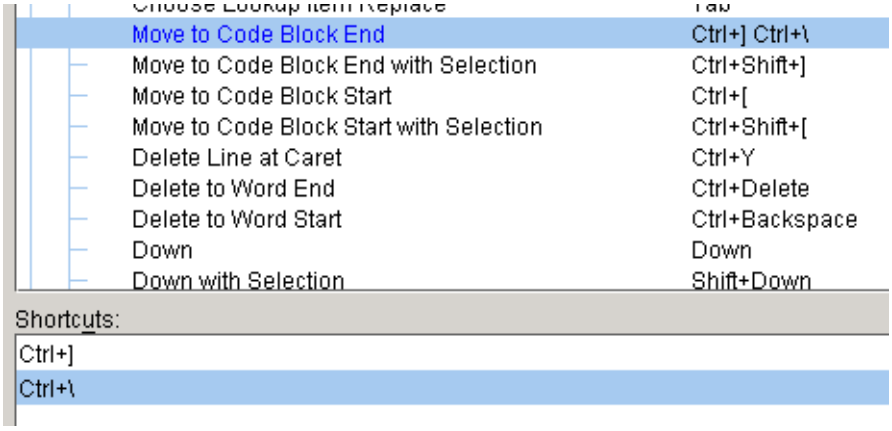


Figure 25.4. New keystroke in the list (422)

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

Index XXX

The index is the last thing to finish.

(the following number is the last page marker.... required by Framemaker).

354

B

M