1

Introduction to Eclipse

In this chapter you install and configure Eclipse. I then use the classical `HelloWorld` example to show how to effectively create Java programs under Eclipse. I first discuss the most important workbench preferences and then introduce various utilities for code creation.

Installing Eclipse

Installing Eclipse is very easy. In most cases, the only thing to do is to unpack the downloaded ZIP file onto a disk drive with sufficient free space. What do you need to run Eclipse? The following list shows what is required:

- **A suitable platform.** Eclipse 3.0 runs on a wide variety of platforms: Windows, Linux, Solaris, QNX, AIX, HP-UX, and Mac OS X. However, in this book I mostly refer to the Windows platform and occasionally give hints for the Linux platform.
- **Sufficient disk space.** 300 MB should be enough.
- **Sufficient RAM.** 256 MB should be fine.
- **Java SDK 1.4.** If this SDK is not installed on your machine, you can download it from `www.javasoft.com` and install it by following the instructions given on this site. You should specify the `bin` subdirectory of the SDK in your `PATH` environment variable so that you can call the Java Virtual Machine (JVM) by issuing the command `java` from the command prompt.
- **Eclipse SDK 3.0 for your platform.**
- The Eclipse example files (**eclipse-examples-3.0**) for your platform.

To install Eclipse, follow these steps:

1. Unpack the Eclipse SDK into the target directory. For example, on Windows that could be the root directory `C:\`. In effect, the Eclipse libraries will be contained in directory `C:\eclipse`. Under Linux you could use the `/opt/` directory so that the Eclipse files would be stored under `/opt/eclipse/`.
2. Immediately afterwards, unpack the Eclipse example files into the same root directory. By doing so, the example files are automatically placed into the just-created `eclipse` subdirectory.
3. That's all. Under Windows you can now invoke Eclipse by clicking the icon with the darkened sun (in the `eclipse` subdirectory). Under Linux you would issue the shell command `/eclipse` under the directory `/opt/eclipse/`.

   Eclipse then prompts you with the *Workspace Launcher*. Here you can select the location of the Eclipse workspace. This workspace will later contain all of your Eclipse projects. Usually the `\workspace\` folder is located in the Eclipse root directory `\eclipse\`. However, it makes more sense to install the workspace in a location separate from the Eclipse installation. This makes later upgrades to new Eclipse version easier (see also Appendix A). In addition, it becomes easier to back up the workspace.

   For example, you may want to specify `...\Own Files\eclipse-workspace` under Windows and `/root/eclipse-workspace` under Linux. The Eclipse Workspace Launcher is shown in Figure 1.1. Note that later when running Eclipse you can easily switch to a different workspace by invoking the function File > Open workspace.

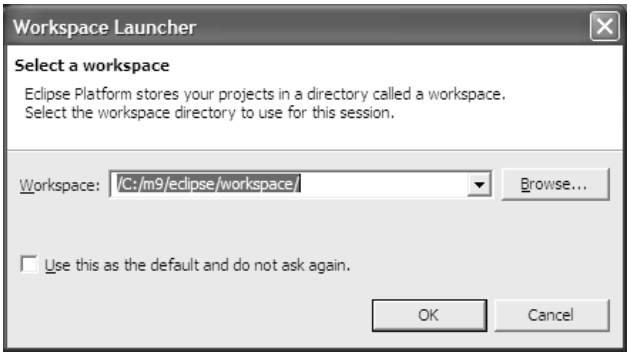

Figure 1.1

Important: When backing up the Eclipse workspace you should always create complete backups—never incremental backups. Eclipse treats the archive attribute of files in a somewhat unconventional way, which can lead to a corrupt workspace when restoring a workspace from an incremental backup. This is a known bug in Eclipse that has not been fixed with the release of Eclipse 3.0.0.

4. After a short while you should see the Welcome screen. Here you have the choice of various information sources such as help pages, tutorials, sample programs, and others:

   - In the Overview section you will find relevant chapters from the various user guides in the Eclipse help system.
   - In the Tutorials section you can learn how to create a simple Java program, a simple SWT application, and an Eclipse plug-in, and you will learn how to create and deploy an Eclipse feature. These tutorials come in form of *Cheat Sheets* that can be followed in a step-by-step fashion.
   - The Samples section contains ready-to-run example programs. These include samples for using the SWT and the Eclipse workbench. If you select such an example program, it will automatically be downloaded from www.eclipse.org (provided that you have established a connection to the Internet) and installed into the Eclipse workbench. Depending on your interests and requirements, it may be worthwhile to take a close look at the code of such an example program.
   - In the What's New section you will find a compilation of the new features contained in Eclipse 3 and also a migration guide for converting the Eclipse 2 application into Eclipse 3 (see also Appendix B). Furthermore, there is a link to the Eclipse Community page and a link to the Eclipse Update site, where you can update your Eclipse installation online.

   However, for the moment you continue the startup process by pressing the Workbench button. You should then see the Eclipse Welcome screen, as displayed in Figure 1.2. You can return at any time to this screen by invoking the function Help > Welcome. Figure 1.3 shows Eclipse running.

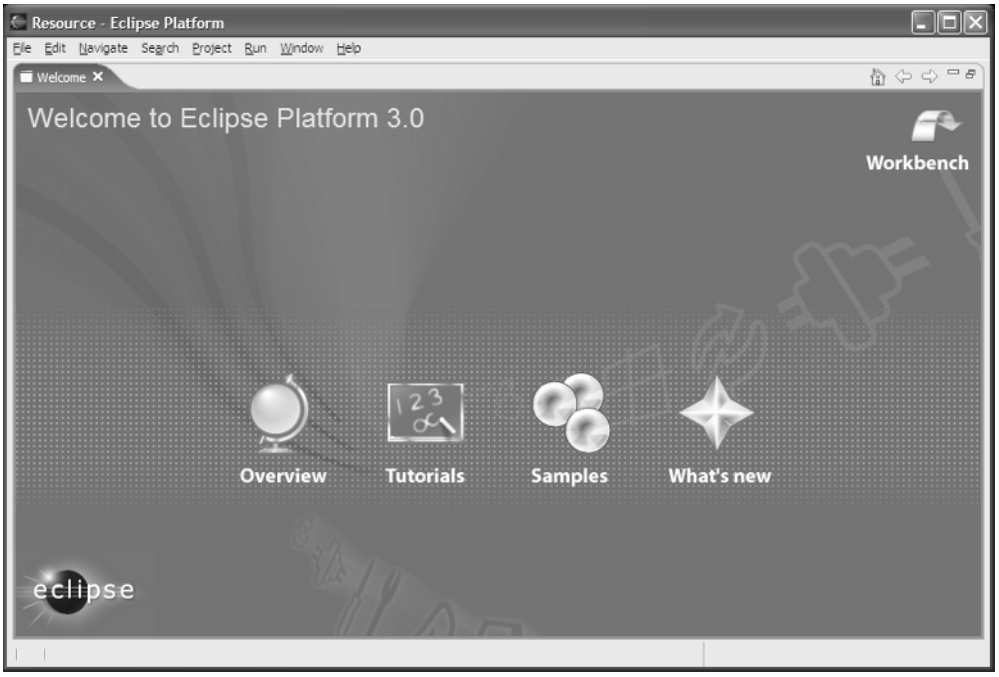

Figure 1.2

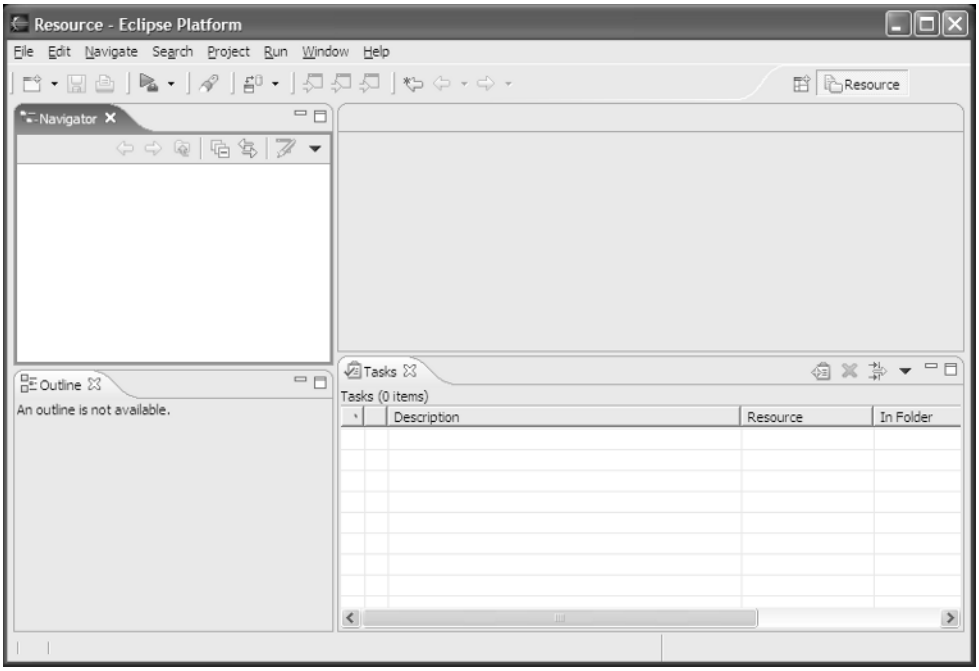

Figure 1.3

5. It is a good idea to create a desktop shortcut for Eclipse. Under Windows simply pull the Eclipse icon onto the desktop by pressing the right mouse button. From the context menu select Create Shortcut Here. Now you can add additional command-line options to this shortcut, for example, the `-vm` option discussed below. To do so, right-click the shortcut and select Properties from the context menu.

   To learn which command-line options are available for Eclipse, check the Eclipse help system by choosing Help > Help Contents. Then select Workbench User Guide, expand the Tasks item, and choose Running Eclipse.

   Under Linux you can similarly create a desktop shortcut under KDE or Gnome and add the required command-line options.

   A further list of command line options is found at Help > Help Contents > Platform Plug-in Developer Guide > Reference > Other reference information > Runtime options. This section lists all command line parameters and the corresponding System Property keys. (For example, the key `osgi.instance.data` is equivalent to the command line parameter `-data`.) These keys can be used to configure Eclipse via the configuration file `\eclipse\configuration\config.ini`. Modifying this file allows you starting Eclipse in different configurations without having to use command line parameters.

6. One of the most important command-line options deals with the selection of the Java Virtual Machine (JVM) under which the Eclipse platform is executed. If you don't want to use the standard JVM (the one executed when invoking the `java` command), you can specify a different JVM by using the command-line option `-vm`.

   When the Eclipse loader is invoked it uses a three-stage strategy to determine the JVM under which the platform is executed. If a JVM is explicitly specified with the command-line option `-vm`, then this VM is used. Otherwise, the loader will look for a specific Java Runtime Environment (JRE) that was deployed with the Eclipse platform. Such a JRE must be located in the directory `\eclipse\jre\`. If such a JRE does not exist (as in our case), then the location of the VM is derived from the `PATH` environment variable.

   By the way, this strategy affects only the JVM under which the platform is executed. Which JVM and which SDK are used for Java development is specified separately in the Eclipse workbench.

   The command-line option `-vmargs` can be used to specify parameters for the Java Virtual Machine. For example:

```
eclipse.exe -vm C:\java13\bin\javaw -vmargs -Xmx256M
```

   Here Eclipse is started with a specific JVM and sets the JVM heap to 256 MB. With very large projects this can help to prevent instabilities of the workbench.

   Another important command-line parameter is the parameter `-data` for specifying the location of the workspace. In this case, the *Workspace Launcher* dialog discussed previously is skipped. This parameter allows you to create different Eclipse desktop shortcuts for different workspaces.

The First Application: Hello World

Until now you haven't seen much of a Java development environment. Eclipse—which is advertised as a platform for everything and nothing in particular—shows, in fact, nothing in particular when invoked for the first time. You are now going to change this radically.

Perspectives

To see something "particular" in Eclipse, you first must open an Eclipse perspective. Perspectives consist of a combination of windows and tools best suited for specific tasks. Perspectives are added to the Eclipse workbench by various Eclipse plug-ins. This is, for example, the case with the user interface of the Java IDE, which is nothing more than a large plug-in for the Eclipse workbench. To start developing Java programs, you therefore must first open the Java perspective. To do so, click the Open Perspective icon, as shown in Figure 1.4.

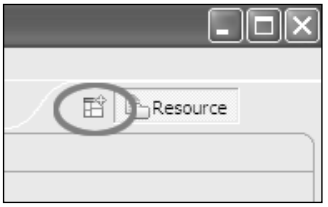

Figure 1.4

Use the Open Perspective icon to open new perspectives. By the way, by clicking the perspective bar with the right mouse button and invoking the function Dock On, you can change the position of the perspective bar. If you were used to Eclipse 2.1, you may want to dock the perspective bar at the left border of the Eclipse workbench.

From the list that appears, select Java. You should then see the screen shown in Figure 1.5.

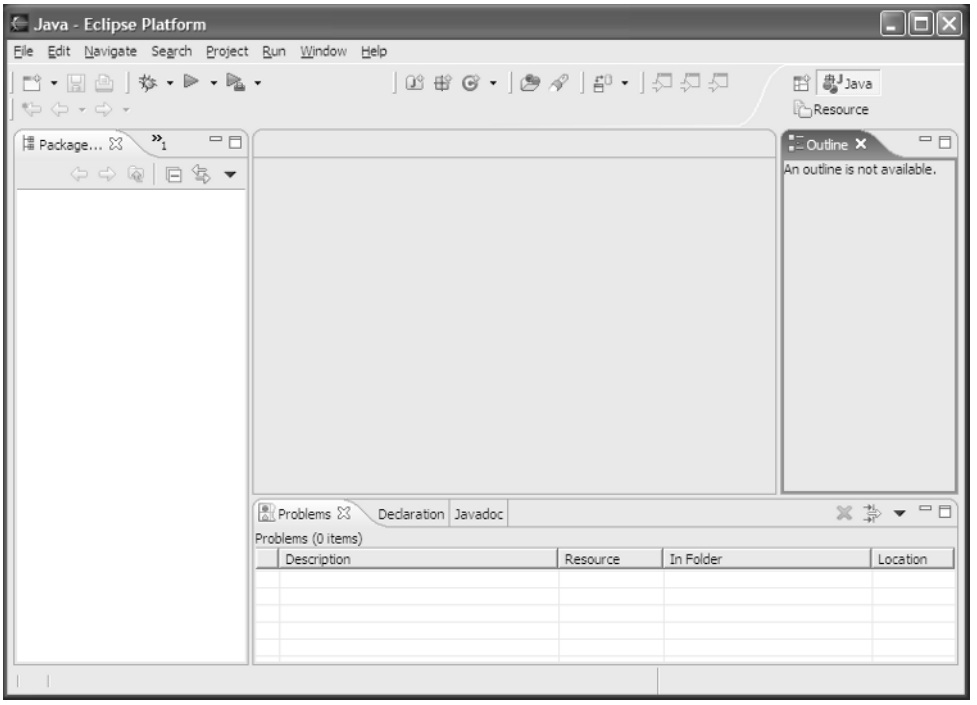

Figure 1.5

The Java perspective shows the windows (Package Explorer, Hierarchy), menu items, and toolbar icons that are typical for Java development. On the left you see a new icon denoting the Java perspective. Above this icon is the icon for the Resource perspective that was active before you opened the Java perspective. You can quickly switch between different perspectives by clicking these icons.

Projects

Now it's time to say Hello to the world and to create your first program. To do so, first create a new Java project. On the toolbar click the Create a Java Project icon, as shown in Figure 1.6. By clicking the icons of this group you can create new Java projects, packages, classes, interfaces, and JUnit Test Cases.

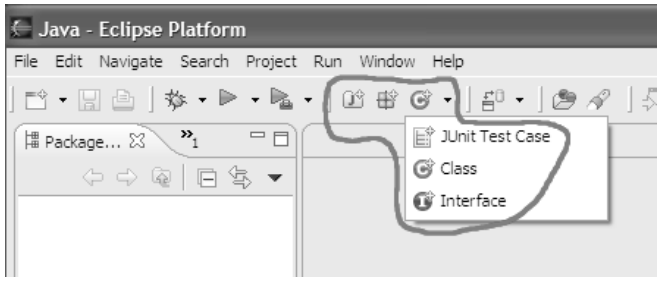

Figure 1.6

In the dialog that appears, name the project with `HelloWorld`. The Package Explorer now shows an entry for the new project.

Create a New Class

In the next step click the C icon on the toolbar (Create a Java Class). In the following dialog make sure that

- The Source Folder is specified as `HelloWorld`.
- The name of the new class is specified as `HelloWorld`.
- `public` is selected as Modifier.
- `java.lang.Object` is specified as Superclass.
- The option to `public static void main()` is checked.

The Create a New Class Wizard (Figure 1.7) is able to generate some class code. The wizard can generate stubs for the inherited methods, especially if a super class and interfaces are specified.

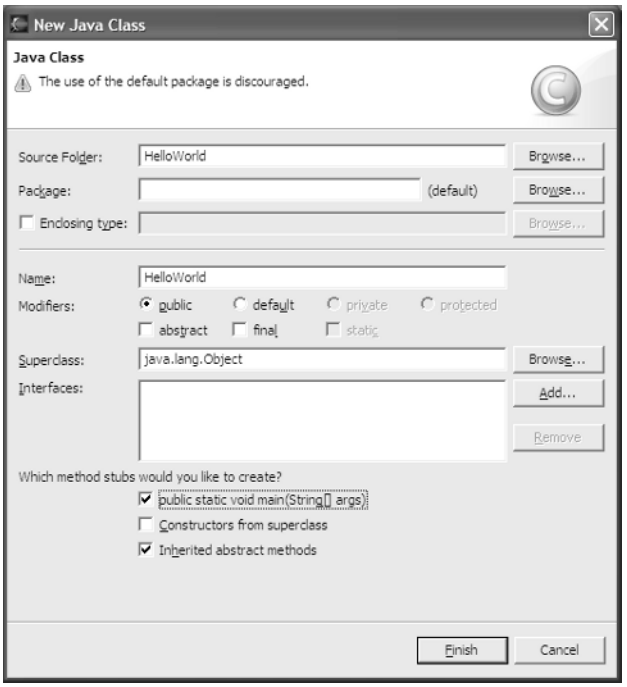

Figure 1.7

After you click the Finish button, the Eclipse workbench looks a bit more like a workbench in use (Figure 1.8).

The Package Explorer shows the contents of the new project, including the libraries of the Java runtime environment. At any time you can open the classes belonging to these libraries and look at their source code. The center window holds the Java source editor, which currently contains the pregenerated code for the HelloWorld class. At the right-hand side you can see the Outline window showing the current class with its methods. You quickly navigate to any method or variable in the source editor by clicking it in the Outline View.

Now you complete the pregenerated code. You change the `main()` method in the following way:

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

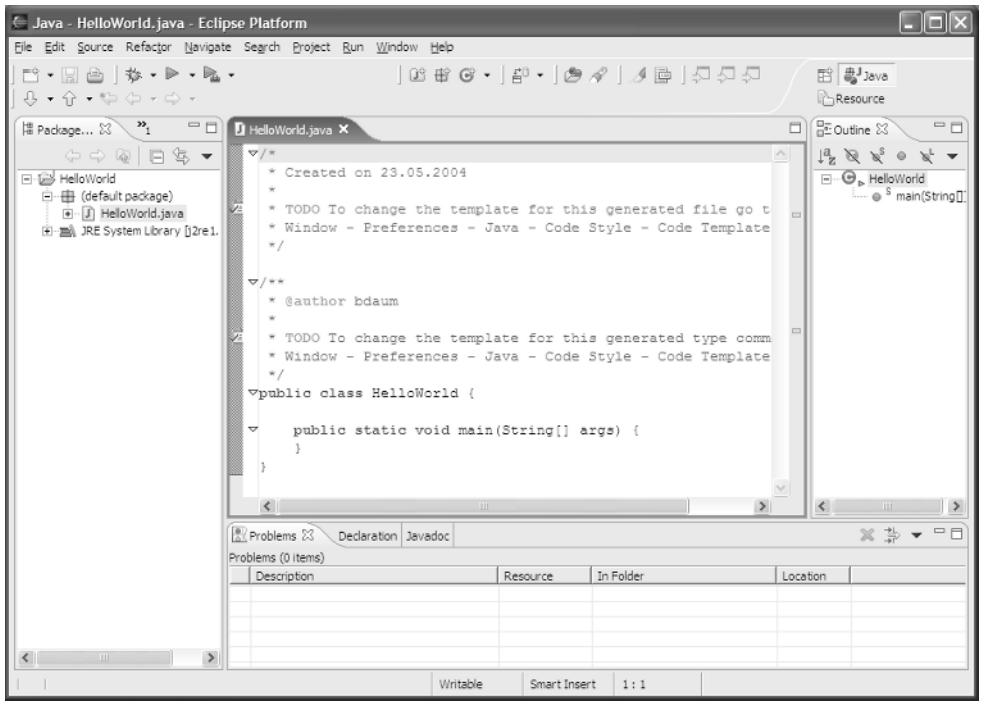

Figure 1.8

By doing this you have finished the programming work for your first project. Save the new class `HelloWorld` to disk by clicking the floppy disk icon on the toolbar. (Alternatively, you can use the keyboard shortcut Ctrl+S.) This will also compile this class. The program is now ready for execution.

Launch

The Run icon is positioned on the right side of the bug icon. Here, you activate the drop-down menu by clicking the arrow at the right of the Run icon. From this drop-down menu select Run As > Java Application to start program execution. Now, a new tag with the label Console should appear in the Tasks View area. With a click on that tag you can open the Console View (see Figure 1.9), which should display the text "Hello World." Done!

During this first execution, Eclipse creates a new Run Configuration named HelloWorld. A list of all available Run Configurations is found under the arrow on the right side of the Run icon. The Run icon itself is always associated with the Run Configuration that was executed last. To execute the program again, simply click the Run icon.

The console window opens automatically when a program writes to `System.out` or `System.err`.

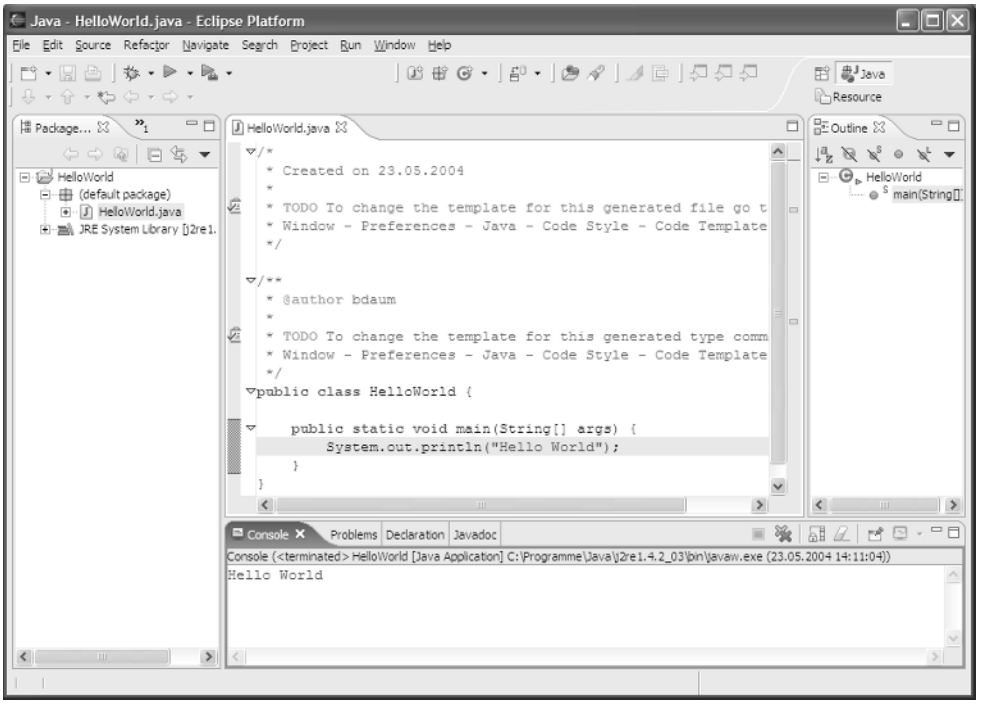

Figure 1.9

The Most Important Preferences for Java Development

Before you continue in your programming efforts, you should first explore your working environment. The Window > Preferences menu gives you access to all Eclipse preferences (see Figure 1.10).

On the left of the Preferences dialog you can select from several preference categories. On the right-hand side of the dialog the details of the selected preference category are shown. All settings made here can be stored into an external file by clicking the Export button or loaded from an external file by clicking the Import button.

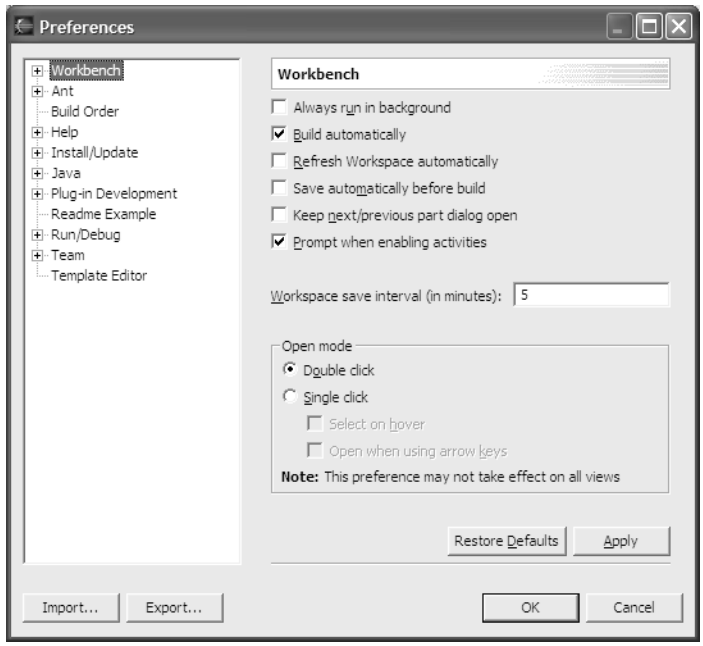

Figure 1.10

At first sight, the sheer mass of preferences shown in this dialog may be overwhelming, because each plug-in may contribute its own set of preference categories to this dialog. In this chapter, I will discuss only those preferences that are most relevant in the context of this book. You should take the time to step systematically through all preference categories to get an overview of the possibilities. Some of the categories have subcategories. To expand a category, click the + sign in front of the category name.

Some of the preference settings will make sense only during the discussion of the corresponding Eclipse function. In such cases I will postpone the discussion of the preference settings to the discussion of the corresponding workbench function.

Workbench Preferences

If you previously have worked with Emacs, it may make sense to switch the Key Bindings in Eclipse so you can continue to use the familiar Emacs shortcuts. To do so, expand the Workbench category, select the subcategory Keys, and click the Keyboard Shortcuts tag. In the drop-down list named Active Configuration you can choose between Emacs and Default. You can even define your own keyboard shortcuts. First, go to the Command group and select a command via the Category and Name fields. The existing keyboard shortcut assignments appear in the Assignments list. A keyboard shortcut can consist of a single key combination or a series of key combinations. Edit the sequence of key combinations by placing the cursor into the Name field of the Key Sequence group and pressing the key combination to

be added to the sequence. Use the Backspace key to delete entries. To add a new key sequence, don't select an entry in the Assignments list; simply enter the key sequences in the described way, and then press the Add button.

On the Advanced page of the Key Bindings preferences you can enable an assistant that will help you with completing multistroke keyboard shortcuts.

Installed JREs

You probably don't always want to create Java applications that require a Java 1.3 or Java 1.4 platform. In some cases you may need to run on Java 1.2 platforms. Within the preference category *Java*, in the subcategory Installed JREs, you can list all Java Runtime Environments that are installed on the host computer (see Figure 1.11).

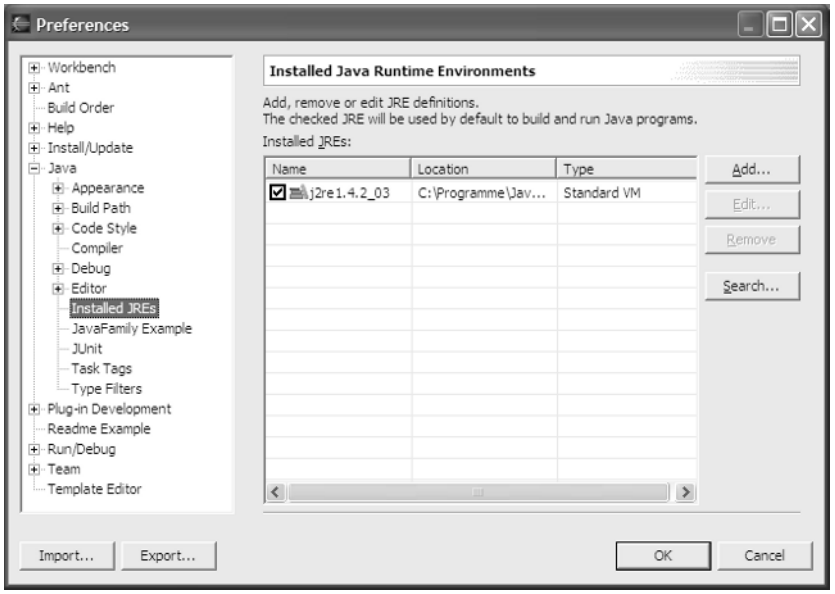

Figure 1.11

In this preference category you can declare all the Java Runtime Environments (SDK or JRE) that are installed on the host computer for Eclipse. Among the JREs listed here, Checkmark One is the default JRE. This JRE will be assigned to all new Java projects. You will learn later how this can be changed in the project settings and how different JREs can be used in different Launch Configurations.

To add a new JRE, just click the Add button (alternatively you can click the Search button to scan a whole directory for a JRE or SDK). Then complete the following dialog (see Figure 1.12).

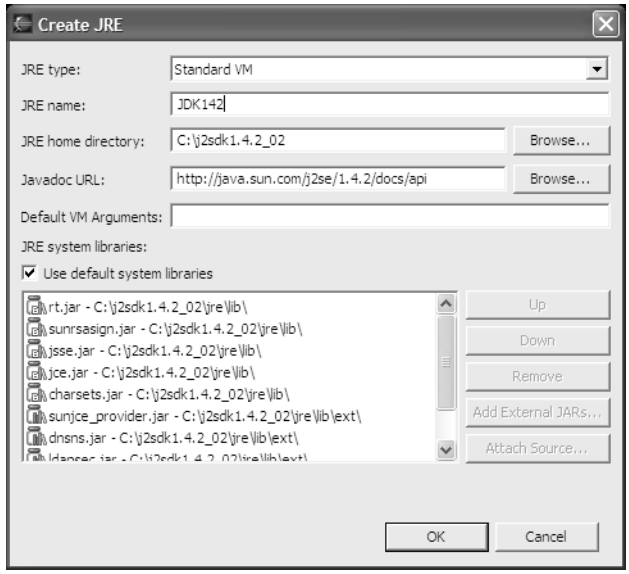

Figure 1.12

A new JRE is added to the Eclipse workbench. I have provided the name and location of the JRE home directory. The location of the corresponding Javadoc is preset by Eclipse and points to the JavaSoft Web site. If the documentation is available locally, you should modify this entry accordingly. The entry Default VM Arguments may specify VM command-line parameters to be used with this VM.

For further customization you could uncheck the Use Default System Libraries item. This would allow you to add further JAR libraries. If any of the JARs does not contain source code, you can attach external source code by pressing Attach Source.

If you want to add a version 1.1 JRE (this is necessary when you want to run your application on a Microsoft VM), you must also change the JRE type to the value Standard 1.1.x VM.

Of course, it is possible to execute an application on a JVM that is different from the JVM under which the application was developed. For example, if you developed an application under Java SDK 1.1.8 and want to test how the application performs under a version 1.3.1 JVM, you must change the runtime environment before executing the program. You can do this by choosing the appropriate JVM in the Eclipse Launch Configurator. You can open the Launch Configurator by invoking the menu function Run > Run.

For the remainder of this book I use the Java 1.4 SDK.

Compiler Preferences

Now take a closer look at the compiler preferences. In the Preferences dialog select the category Java and the subcategory Compiler. Note that all adjustments made here affect the whole workbench. On project level (see the "Project Properties" section in Chapter 4), however, you have the possibility of overriding the global settings made here under Preferences.

Warnings and Errors

On the right-hand side of the Java > Compiler category you see a tabbed notebook. The Style, Advanced, Unused Code, and Javadoc pages show which compiler events create errors or warnings and which compiler events should be ignored (see Figure 1.13).

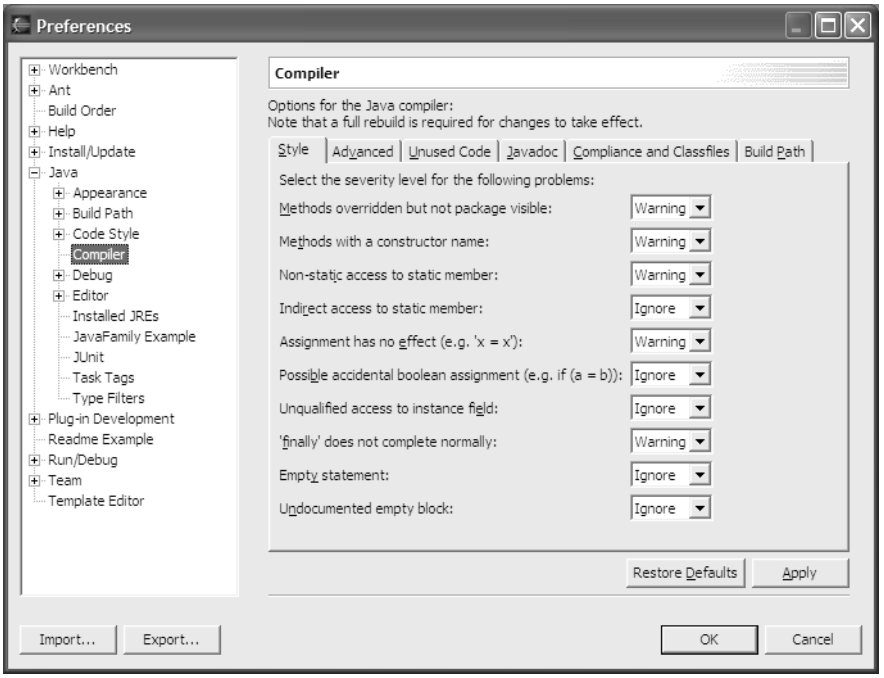

Figure 1.13

Because a lot of third-party code is used in the examples, you need to reset the settings for unused imports, never-read local variables, and never-read parameters on the Unused Code page to Ignore. Otherwise, you could face an overwhelming flood of error messages. But if you develop your own applications, it makes sense to set these settings to Warning because these settings help you to detect

and remove garbage from your code. Just try the following: set Parameter Is Never Read to Warning and press OK. The project is recompiled. At the program line

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

you now see a warning icon, and in the Problems window you see the entry

```
The argument args is never read
```

Quite right! The `HelloWorld` program did not make use of the parameter that contains the command-line arguments.

Classfiles and JDK Compliance

On the Compliance & Classfiles page you can specify which symbolic information, such as variable names and line numbers, is to be included in the generated classfiles. This information is required for debugging, and therefore you may want to leave the proposed settings unchanged. However, for a well-tested program it may make sense to remove this information from the classfiles; generated files are much smaller without the symbol tables.

On the same page you can determine whether the compiler must comply with the Java 1.4 or Java 1.3 syntax. With Java 1.4, one new instruction was added to the language: `assert`. Consequently, the word "assert" can no longer be used as a field or method name. In addition, `assert` requires support from the JVM. Classes that use this instruction cannot be executed by older JVMs. Since `assert` is not used in the first example program, leave this setting at the proposed value of Java 1.3.

Formatting Code

Formatting code can be very helpful, because it is easier to detect violations of the control structures of a program (such as open `if` or `while` statements) when the program is formatted. In the preference category Java > Code Style > Code Formatter you can configure how the Eclipse code formatter works, as shown in Figure 1.14. The best method is to try some of the settings and to select those that work best for your application. To modify these settings you must first create a new profile (by pressing the New button). Then you can edit this profile by pressing the *Edit* button. You can create multiple profiles and switch easily among them. When you publish your code, for example, you may use different profiles for different sorts of publications.

But how do you apply code formatting? Very simply: just click with the right mouse button on the source code and select Source > Format from the context menu (Figure 1.14). The key shortcut Ctrl+Shift+F works even faster. Note that it is also possible to select only a portion of the source code to format just that portion.

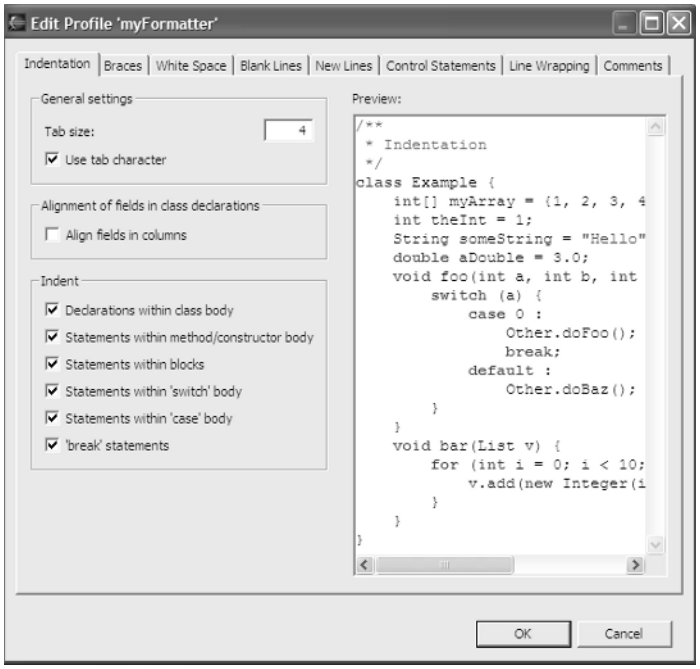

Figure 1.14

Templates

When you created the new `HelloWorld` class, text similar to the following was generated at the top of the new compilation unit:

```
/*
 * Created on 27.04.2004
 *
 * To change this generated comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
/**
 * @author Berthold Daum
 *
 * To change this generated comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
```

The first comment was generated for the new Java file, and the second comment was created for the new type (`HelloWorld`). You now should follow the advice given in these comments and modify the code generation preferences according to your requirements. Just open the preferences category Java > Code Style > Code Templates (see Figure 1.15).

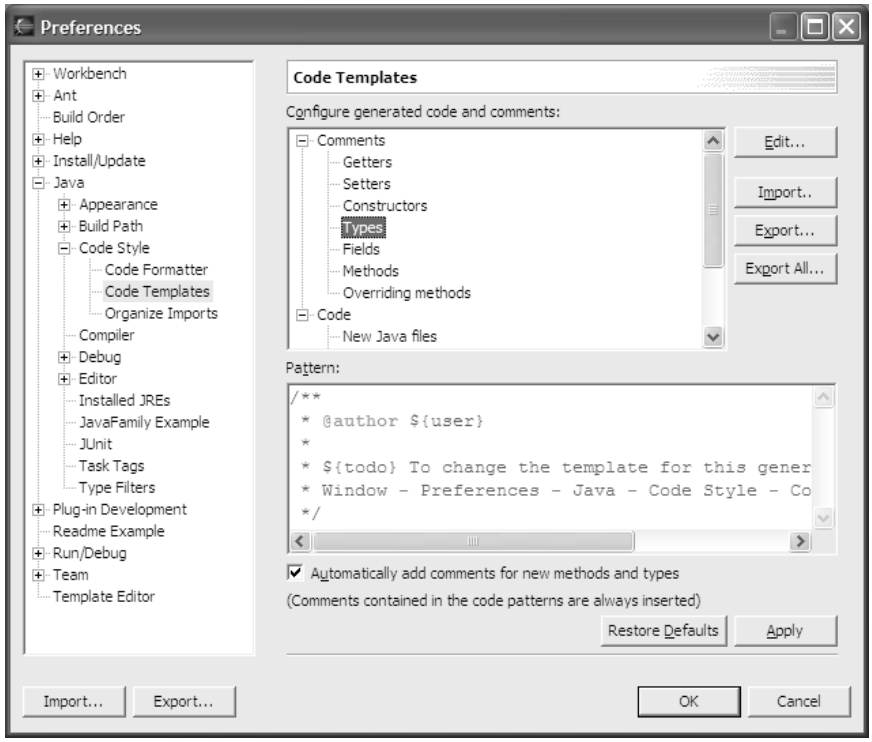

Figure 1.15

For various events, such as the creation of a new type, a new method, or a new constructor, you can specify which code is to be generated.

Select the Types entry and press the Edit button. In the dialog that appears replace the text provided by Eclipse with the string "created first in project." Then press the Insert Variable button. From the list select the variable named `project_name`. The result should look like that shown in Figure 1.16.

After you have committed these changes, new classes and interfaces will be created with a comment containing the user name and the project name.

Figure 1.16 shows the process of editing a code generation template. All variables are prefixed with the $ character and are enclosed in curly brackets. Apart from the template pattern, you can also supply a description (which will appear in the overview) and a template context (Java or Javadoc).

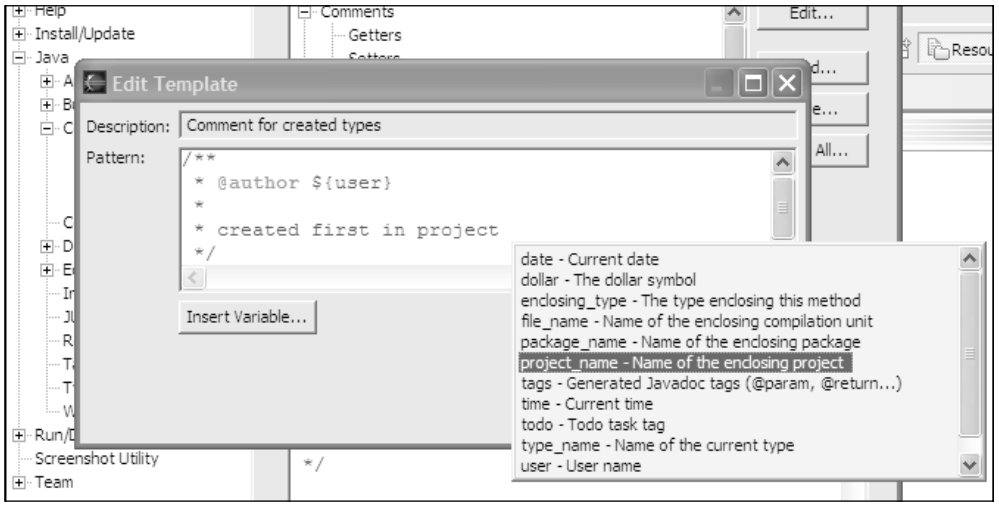

Figure 1.16

Later you may also change the entry for New Java File. The predefined text is shown here:

```
/*
 * Created on ${date}
 *
 * To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
${package_declaration}
${typecomment}
${type_declaration}
```

The variables used here define the sequence of the different code parts. For example, you just specified what happens under `typecomment` in the previous template. Here, delete only the text lines `To change...Code Templates`, and leave everything else as is.

Tasks and Problems

Eclipse uses the *Tasks* and *Problems* views to notify the user about pending tasks. The Problems view lists problems such as errors or warnings. By clicking on such an entry you can quickly navigate to an erroneous program line.

Other task entries are hints about pending development actions and are shown in the Tasks view. Some of these hints are created by Eclipse. For example, when you create a new class or a new method, Eclipse creates a hint that the new construct must still be completed. Programmers may create similar task entries at their own discretion.

Problems, Problems

In the Compiler Preferences section of this chapter, you saw the Problems window in action. The entries in the Problems window correlated to pending problems in the Eclipse workbench. In Figure 1.17 I purposely created a syntax error by inserting a blank into the parameter name args. This resulted in three error messages.

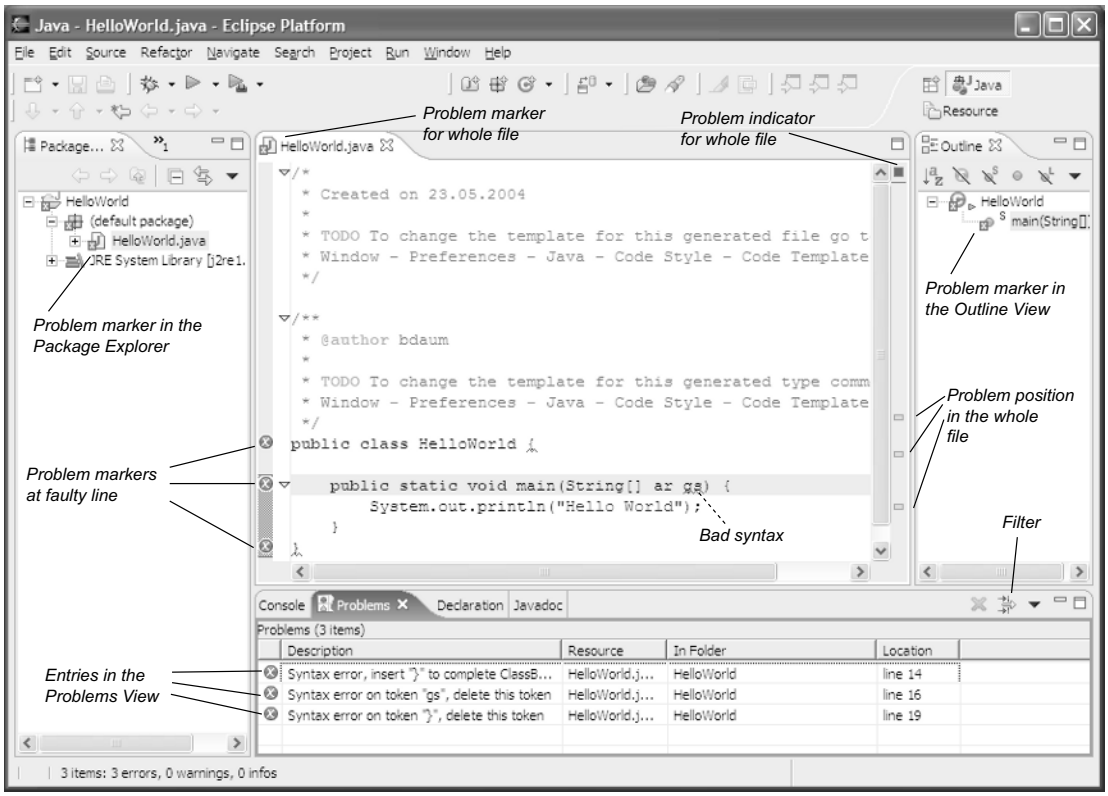

Figure 1.17

After you double-click the problem entries, the faulty expressions are underlined in red. Red error markers on the source editor's left margin mark the faulty lines. The markers on the right margin show the position of the errors relative to the whole file. To scroll to the error position, it is only necessary to pull the scroll bar to the markers. Clicking the markers works just as well.

Error messages are represented by a white cross in a red circle. In contrast, warnings are represented by a yellow triangle. The third problem type is information tasks, which are represented by a blue `i` character.

Double-clicking the problem entry in the Problems view lets you quickly navigate to the problem location. Should the problem be located in a file that currently is not open, the file will be opened in the

editor, and the editor window will be positioned on the error location. Just try it, and click on one of the entries in the Problems view.

As the workbench gets busier, the Problems view often overflows with errors and warnings. At times it can become difficult to find the Problems entries that are related to the current project or file, because the Problems view by default shows all problems and other tasks within the whole workspace. Of course, you can suppress some of the warnings by setting the compiler options accordingly (see the "Compiler Preferences" section). But there is another way to reduce the information overload: by using the Problems Filter (Figure 1.18). You can open the Problems Filter dialog by clicking the Filter button in the toolbar of the Problems window.

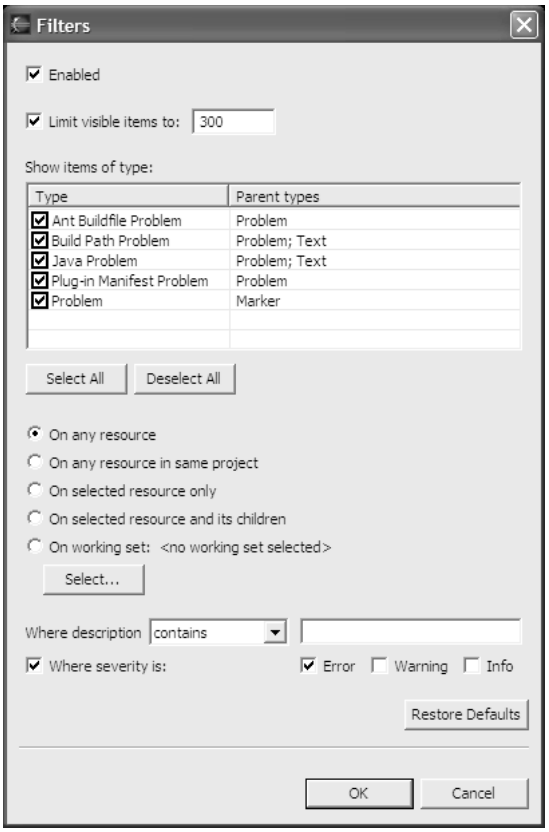

Figure 1.18

The setting shown here allows only entries from the current project to appear in the Problems view. The type of the entry is irrelevant.

Here, in the Problems Filter, you may restrict the entries shown in the Problems view to specific types. For example, you may opt to show only Java problems. The entry types shown in this window depend on the installed plug-ins.

In addition, you can restrict the entries by their origin. The On Any Resource option shows all problems and tasks from the whole workbench. On Any Resource in Same Project shows only problems from the current project. An interesting option is also the definition of a Working Set—a freely configurable and named set of resources. Select On Any Resource in Same Project if you want to see only the tasks and problems of the project on which you are currently working.

You also have the option of filtering problems according their severity. To do so, mark the Where Problem Severity Is check box and also the Error check box. By doing so you can suppress all warnings and information entries.

General Tasks

Task entries generated by the compiler are only a specific type of task entry. In addition, you have the option of creating entries manually. When writing code it often happens that you want to postpone a certain task to a later time. In this case, you can create a task entry that later reminds you of the unfinished work.

Just click with the right mouse button on the left margin of the source editor at the line where you want to create the task marker. Select Add Task from the context menu. In the New Task dialog, enter a task description. The result could look like Figure 1.19.

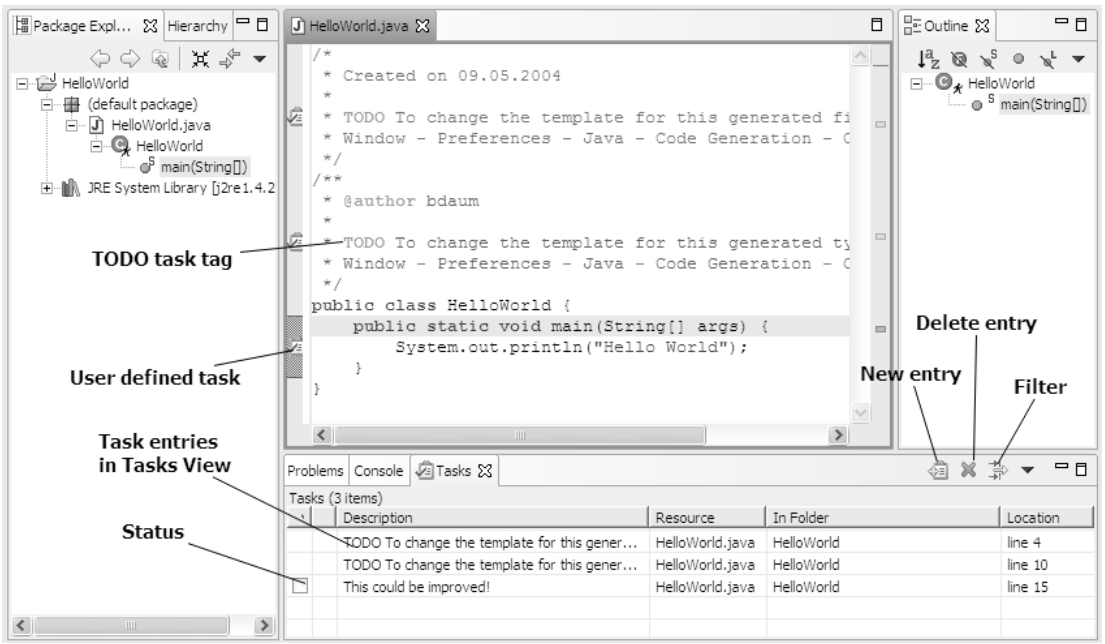

Figure 1.19

This task entry was created by the user. By clicking the status field you can mark the entry as completed. You can click the Delete button to delete one or several selected tasks. You can create task entries that are not related to specific locations with the New Entry button. For example, you could create a task called Don't Forget to Buy Milk!

A function that was introduced with Eclipse 2.1 is even simpler. Just type a comment starting with one of the words `TODO`, `FIXME`, or `XXX` in a new line. This line will automatically appear in the Tasks window as soon as you save the source code. By the way, in Preferences > Java > Task Tags you may define alternative or additional tags such as `TUNE`, `UGLY`, etc. Of course, these workbench-wide definitions can be overridden at the project level.

> **If you work in a team, you should always create tasks that are important for other team member, too, in this way (as a comment in the source code) so the tasks can be exchanged as part of the source code.**

Bookmarks

Eclipse also has a construct that is quite similar to tasks: *bookmarks*. In the same way that you created a task entry, you can also create a bookmark. Such a bookmark, however, does not appear in the Tasks view but appears in a separate Bookmark view. Since this view is not a standard part of the Java perspective, you first must open it. Select Window > Show View > Other > Basic > Bookmarks (see also the "Arranging Editors and Views" section in Chapter 4). Bookmarks should be used when you want to mark a specific position in the code but it is not related to a pending task.

The Scrapbook

Eclipse also inherited the *Scrapbook* from Visual Age. A scrapbook page is nothing other than a small file in which you can try out Java expressions or just jot down a new idea.

You can create a new scrapbook page by invoking the function File > New > Other. In the wizard select Java > Java Run/Debug > Scrapbook Page. In the dialog that appears specify a name for the new page and, if necessary, the target folder. The result is the creation of a new empty scrapbook page in the target folder. Scrapbook pages have the file extension `.jpage`.

Now, how do you use a scrapbook page? You simply type in arbitrary Java expressions. If you use external types in these expressions, you either have to qualify the type names fully or add `import` statements. The context function Set Imports allows you to add `import` statements for single types or whole packages.

Then select the expressions that you want to execute and call the Execute context function with the right mouse button (see Figure 1.20).

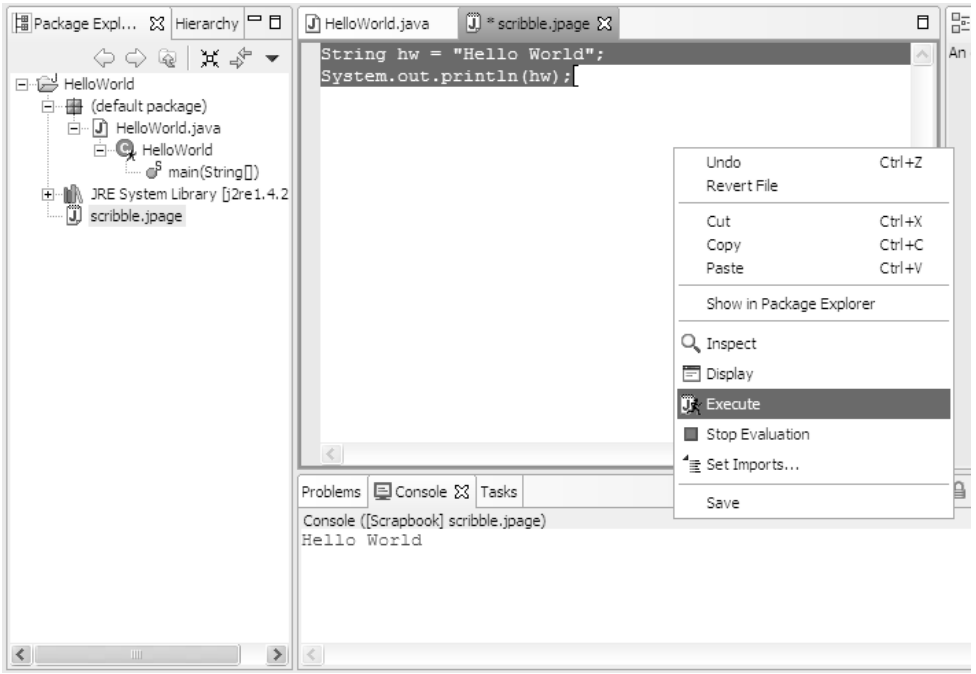

Figure 1.20

The selected expression is executed with the help of the Execute context function. The scrapbook context function appears on the workbench's toolbar at the far right.

It is not necessary to save the scrapbook page before executing the selected code. The selected code is compiled by the Execute function. In the case of a compilation or execution error, Eclipse shows the error message in a pop-up window. You may insert it into the current scrapbook content by pressing Ctrl+Shift+D. You can easily remove it again by applying the Undo function (Ctrl+Z).

Execute is not the only function that you can use to run a Java expression. In cases where you want to know the result of an expression, it would be better to use the Display function. For example, executing the expression

```
6*7
```

with the Display function returns the result

```
(int) 42
```

Eclipse shows the result in a pop-up window. You may insert it into the current scrapbook content by pressing Ctrl+Shift+D. You can easily remove it again by applying the *U*ndo function (Ctrl+Z).

A further function for executing selected expressions is Inspect. This function first appears in a pop-up window, but by pressing Ctrl+Shift+I you can move it to the separate Expressions View (see Figure 1.21) that opens automatically when needed. This function is particularly useful when the result of the executed expression is a complex object. In the Expressions window you can open the resultant object and analyze it in detail.

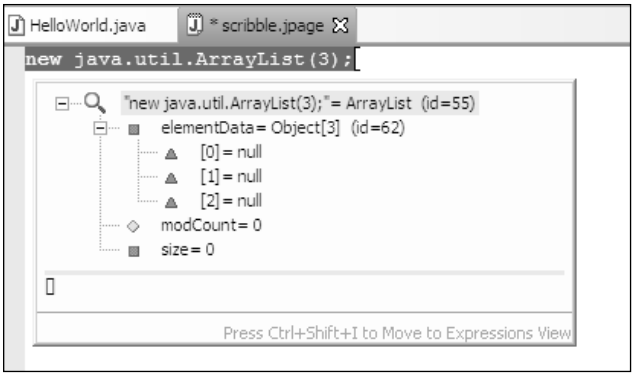

Figure 1.21

The results shown here are displayed in a pop-up window after applying the Inspect function on the expression `new java.util.ArrayList(3);`.

Summary

After this first chapter you should be able to create, compile, and run simple Java program with Eclipse. You should now know how to install Eclipse, create projects, and launch programs. You have become acquainted with the most important preferences and should take some time now to browse through the remaining preferences. However, the purpose of some preferences may become clear only during the course of this book.

Source code annotations such as tasks and problem markers are powerful concepts during the development of a software project. In Chapter 16 you will see that these concepts can be used to adopt a more natural programming style.

Finally, the scrapbook encourages experimenting with Java so that you can try out new program constructs in isolation before integrating them into an application.

In the next chapter I will introduce into the various productivity techniques found in Eclipse.