Professional Eclipse 3 for Java™ Developers

Berthold Daum

WILEY

Professional Eclipse 3 for Java™ Developers

Professional Eclipse 3 for Java™ Developers

Berthold Daum

WILEY

Other Wiley Editorial Offices

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstr. 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 33 Park Road, Milton, Queensland 4064, Australia

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop #02-01, Jin Xing Distripark, Singapore 129809

John Wiley & Sons Canada Ltd, 22 Worcester Road, Etobicoke, Ontario, Canada M9W 1L1

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN: 0-470-02005-9

Typeset in Indianapolis, IN USA
Printed and bound by Malloy printing in Ann Arbor, MI USA
This book is printed on acid-free paper responsibly manufactured from sustainable forestry
in which at least two trees are planted for each one used for paper production.

Credits

Author
Berthold Daum

Executive Editor
Gaynor Redvers-Mutton

Production Editors
Felicia Robinson
Juliet Booker

Book Producer
Ryan Publishing Group, Inc.

Copy Editor
Linda Recktenwald

Compositor
Gina Rexrode

Illustrator
Nathan Clement

Vice President & Executive Group Publisher
Richard Swadley

Vice President & Publishing Director
Sarah Stevens

Vice President and Publisher
Joseph B. Wikert

Editorial Manager
Kathryn Malm

About the Author

Berthold Daum has a Ph.D. in Mathematics and is a professional Java and XML developer who has been using Eclipse since it was first developed. Mr. Daum specializes in innovative electronic business technology and electronic content production; his clients include SAP Integrated Services AG and Software AG. His experience in software training and ability to anticipate the needs of professional developers has been demonstrated in his previous books, including *Eclipse 2 for Java Developers* (Wiley) and *Modeling Business Objects with XML Schema* (Morgan-Kaufmann).

Mr. Daum studied photography in Melbourne and has both exhibited and published his images of Australia's natural beauty.

Introduction

The first version of Eclipse was released in November 2001. Eclipse was announced by IBM as a $40 million donation to the Open Source community. The first reactions to this gift, however, were mixed. While many Java programmers hailed the release of Eclipse enthusiastically (when would one not be enthusiastic about a $40 million present?), Sun Microsystems was initially less than amused.

In the meantime, Eclipse has taken the Java world (and not only the Java world) by storm, despite the fact that Sun Microsystems is still not onboard. Eclipse is now completely managed by eclipse.org, an independent, nonprofit organization in which, however, IBM plays a major role. Despite the fact that the membership fee is quite hefty ($250.00 per year) and commitment is asked in the form of staff members working actively toward the development of Eclipse, the membership circle is not at all small: the Eclipse consortium has about 150 member companies, and people from Ericsson, Genuitec LLC, IBM, Hewlett Packard, Intel, MontaVista Software, QNX Software Systems Ltd., SAP AG, SAS, Serena Software, and the University of Washington belong to the board (Microsoft, you guessed it, is not a member).

So, the question is, what is Eclipse? Is it a Java IDE? Is it a new GUI for Java applications? Is it an application platform or framework?

`Eclipse.org refers to` Eclipse as a platform for "everything and nothing in particular." That we can use Eclipse to develop Java programs (in fact, it is one of the finest Java IDEs) is just a special application of this platform. But its real application domain reaches far beyond Java development. Because of its plug-in architecture, Eclipse is as adaptable as a chameleon and can find a habitat in quite different environments. The Eclipse Java IDE is, in fact, only an eminent example of an Eclipse plug-in. A large number of other plug-ins have already been developed for Eclipse by various companies and developers or are currently in development (see Appendix A for a small selection of such developments). For example, there is a plug-in for a C++ IDE, while plug-ins for other programming languages such as RPG and COBOL are in preparation. In this book, however, we will concentrate on Java development with Eclipse.

Eclipse is more than a pure development environment. With its SWT and JFace libraries it provides an alternative to Sun's Java libraries, AWT and Swing. SWT and JFace allow the creation of Java applications that closely match native applications (i.e., applications written in C or C++) in both "look and feel" and in responsiveness. In contrast, applications implemented on the basis of Swing often lack responsiveness and sometimes differ—despite the possibility to switch skins—from the "look and feel" of a native application. Such applications are notoriously hard to sell, because end users expect applications that fulfill the standards of the host platform. SWT and JFace could therefore be a breakthrough for Java applications on the desktop. No wonder, therefore, that there is a heated debate for and against SWT/JFace in the respective discussion forums (for example, `www.javalobby.com`) and that the SWT was voted as the "most innovative Java component."

Finally, Eclipse provides a large framework for implementing Java applications. Besides the GUI libraries SWT and JFace, we find higher-level components such as editors, viewers, resource management, task and problem management, a help system, and various assistants and wizards. Eclipse uses all these components to implement features such as the Java IDE or the workbench, but they can also be used for your own applications. In particular, the Rich Client Platform that was introduced with Eclipse 3 provides a generic framework for a wide class of applications. The Eclipse license model allows users to embed these components into their own applications, to modify them, and to deploy them as part of their own applications—all without paying a cent in license fees. The complete Eclipse code is available as source code, can be browsed online, and can be used within you own projects.

The Eclipse Culture

Of course, Eclipse was not just "invented": it has a history. The author of this book, who has used Visual Age for Java for years, can detect many of the Visual Age construction elements within Eclipse. In fact, the same company that stood behind the development of Visual Age is also responsible for the development of Eclipse. This company is OTI (`www.oti.com`). As long ago as 1988, OTI developed a collaborative development environment for Smalltalk called ENVY, which was later licensed to IBM under the name Visual Age. What followed was the development of Visual Age for Java, but this was still implemented in Smalltalk. Now, OTI has started the next generation of development tools with Eclipse. Of course, we find many of the design elements of Visual Age in Eclipse. The difference is, however, that Eclipse is implemented in Java and that it features a much more open architecture than Visual Age.

Eclipse was licensed by IBM and than donated to the Open Source community. This was not done without self-interest: Eclipse basically is nothing more than the community edition of IBM's WebSphere Studio Application Developer (WSAD). The core platform and the core plug-ins are all the same. The main difference is that Eclipse 3.0 consists of about 90 plug-ins, while WSAD features about 500–700 plug-ins, thus offering greatly extended functionality, such as plug-ins for developing web and database applications.

About This Book

It is practically impossible to write a single book about Eclipse. The sheer complexity of Eclipse would require quite a few books. I have tried to emphasize those topics where Eclipse makes significant contributions to the Java world. In particular, these are the new GUI libraries (SWT and JFace) and the use of Eclipse as a platform and framework for desktop applications. What had to be excluded from this book are WebSphere-specific topics such as J2EE and servlet development. Developing desktop applications is currently one of the strong points of Eclipse.

This book is not an introduction to Java programming. We assume that readers have a good knowledge of Java and of object-oriented programming concepts. Most of the examples used in this book are not trivial. Two examples come from the multimedia area. Here, readers have the possibility of "getting their feet wet" with cutting-edge Java technology such as speech processing and MP3 (all in pure Java!). In the third example, we do something useful and implement a spell checker plug-in for Eclipse. I am sick and tired of bad orthography in Java comments! The last example is a board game implemented on the basis of the Rich Client Platform, just to burn some of the programmer's spare time gained by productivity enhancements of the Eclipse IDE.

This book, therefore, addresses Java programmers—from the student to the professional—who want to implement their own desktop applications with the help (or on the basis) of Eclipse. You will learn all the techniques that are required to create applications of professional quality.

How This Book Is Organized

The novice to Eclipse—or even an experienced Java programmer—is at first overwhelmed by the sheer number of functions. But the functions visible to the user are only the tip of the iceberg. If we start to explore the inner workings of Eclipse, its API, we can get lost easily. Currently the Eclipse download has a size of 83 MB.

Faced with this huge amount of information, this book uses a pragmatic approach. Following the motto that "perception works from the outside to the inside," I first investigate how Eclipse presents itself to the end user. The benefit is twofold: first, each programmer is an end user of the Eclipse Java IDE; second, the various components of the Eclipse workbench, such as editors, views, menus, dialogs, and much more, can also be used in personal applications. Experienced programmers, however, may find an introduction into the Java IDE trivial and superfluous. Nevertheless, it is useful to get well acquainted with the Eclipse user interface, because many of the concepts and details can be later utilized when designing you own applications.

In Chapters 1 through 7 of this book I first introduce practical work with Eclipse, in particular with the Java development environment. Eclipse presents itself as a very powerful Java IDE that continues the positive traditions of Visual Age for Java but also introduces new concepts such as code completion, strong refactoring facilities, assistants that make intelligent proposals for fixing program errors, and a local history that allows a return to previous code versions.

In these chapters I also discuss the organization of the workbench, the resources of the Eclipse workspace such as projects, folders, and files, how these resources are related to the native file system, and the tools for navigation. I explain what perspectives are and how they can be used effectively. The Eclipse Java debugger and the integration of JUnit into Eclipse are discussed, and a short introduction about Eclipse's support for working in a team is given.

The examples used in this part are still all based on AWT and Swing.

However, this will quickly change in the second part of the book, Chapters 8 through 10. Here, I introduce the secrets of the SWT and JFace libraries. For SWT, event processing is discussed, along with the various GUI elements such as text fields, tables, buttons, and trees; the various layout options; graphics operations and how Java2D can coexist with the SWT; and printer output. I also explain the specialties of thread and resource management in the context of the SWT and the integration of SWT widgets with Swing facilities.

In the case of the JFace library, I present the higher user interface levels such as windows, dialogs, viewers, actions, menus, text processing, wizards, and preferences. As an example, an MP3 player that can be deployed independently of the Eclipse platform is implemented completely with SWT and JFace. An interesting detail in this example is how the SWT library is used in a multithreaded application.

In Chapters 11 through 16 I explain how to develop your own products on the basis of the Eclipse platform: either as a plug-in to Eclipse or as a stand-alone application under the Rich Client Platform. Since Eclipse consists more or less only of plug-ins, I first introduce the plug-in architecture of Eclipse. The requirements for a minimal platform are discussed, and I show how workspace resources are used in Eclipse and how plug-ins are declared via a manifest. Then the various components of the Eclipse workbench such as editors, views, actions, dialogs, forms, wizards, preferences, perspectives, and the help system are introduced. All these components are available to the application programmer as building blocks, a fact that can speed up application development considerably.

Then, I show how your own products can be packaged for deployment. Eclipse offers integrated support for all tasks here, too: from the creation of a feature, to the creation of nation language fragment and the definition of an update site, to the automated installation of updates. As an example, a universal and fully functional plug-in for spell checking on Eclipse platforms is implemented.

Finally, I discuss the Rich Client Platform (RCP) that was introduced with Eclipse 3 and serves as a generic platform for a wide range of applications. The board game Hex is implemented as an example of such an RCP application.

In Appendix A some more interesting third-party plug-ins are listed. In Appendix B I discuss the migration to another version of the Eclipse platform. Appendix C contains download addresses for the third-party software and the source code used in the examples.

Acknowledgements

Books are always teamwork, even if only the author's name appears below the title. This is also the case with this book, and here is the place to acknowledge the contribution of all the other team members.

Special thanks go to the publisher John Wiley & Sons and Wrox, in particular to Gaynor Redvers-Mutton who acted as the publishing editor. Thanks go also to the publisher of the original German edition, dpunkt verlag, and the responsible editor there, René Schönfeldt.

Thanks also to Tim Ryan's group who handled the production of this book, especially Linda Recktenwald for copyediting, Gina Rexrode for composition, and Nathan Clement for his technical illustrations.

Many important tips that found their way into this book came from the (anonymous) reviewers but also from developers and employees of OTI who had looked at the first manuscript version. Many thanks! And of course, without the development of Eclipse this book would not have been written, and Eclipse is indeed a tool that I wouldn't want to miss. Thanks again!

Berthold Daum
June 2004
berthold.daum@bdaum.de

Contents

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:

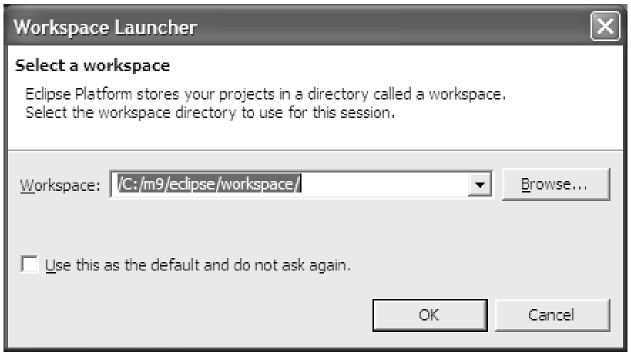
- **A suitable platform.** Eclipse 3.0 runs on a wide variety of platforms: Windows, Linux, Solaris, QNX, AIX, HP-UX, and Mac OS X. However, in this book I mostly refer to the Windows platform and occasionally give hints for the Linux platform.
- **Sufficient disk space.** 300 MB should be enough.
- **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.


Workspace Launcher

Select a workspace

Eclipse Platform stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: /C:/m9/eclipse/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.

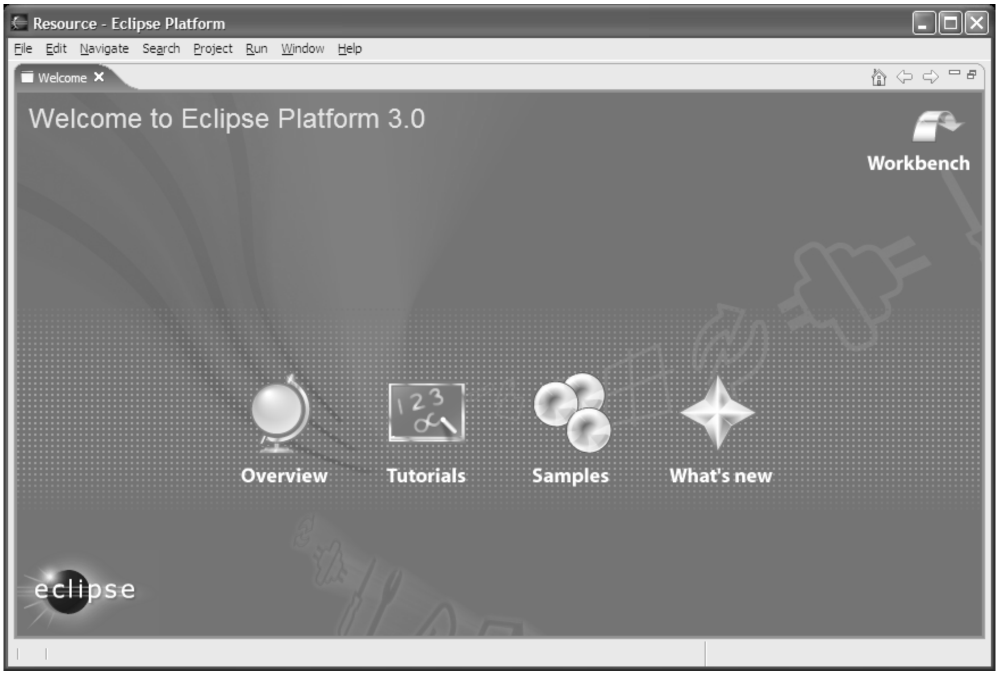

Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse Platform 3.0
Workbench
Overview
Tutorials
Samples
What's new
eclipse

Figure 1.2

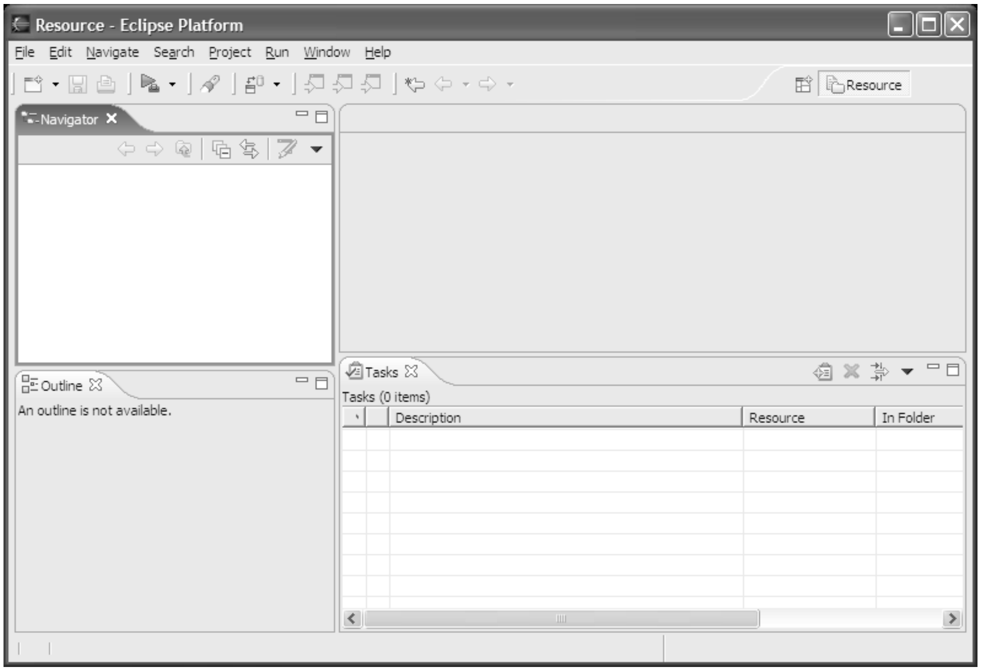

Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Resource
Navigator
Outline
An outline is not available.
Tasks
Tasks (0 items)
Description
Resource
In Folder

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.

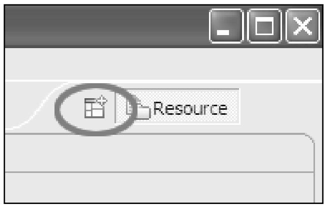
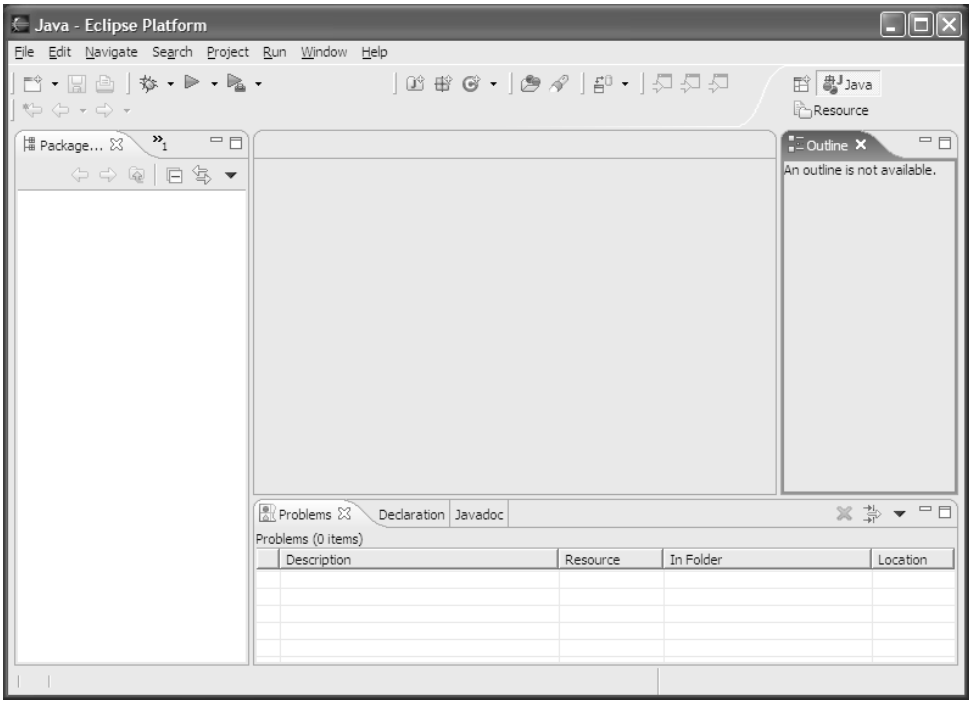


Figure 1.4

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.



An outline is not available.

Problems (0 items)

	Description	Resource	In Folder	Location

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.

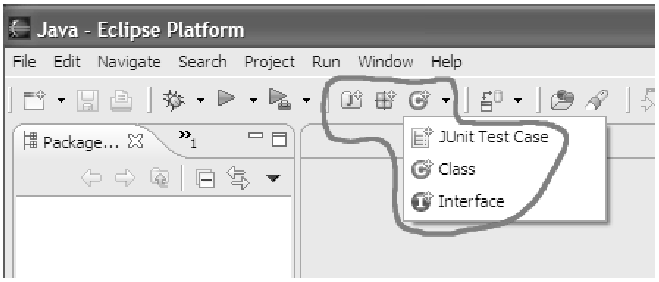


Figure 1.6

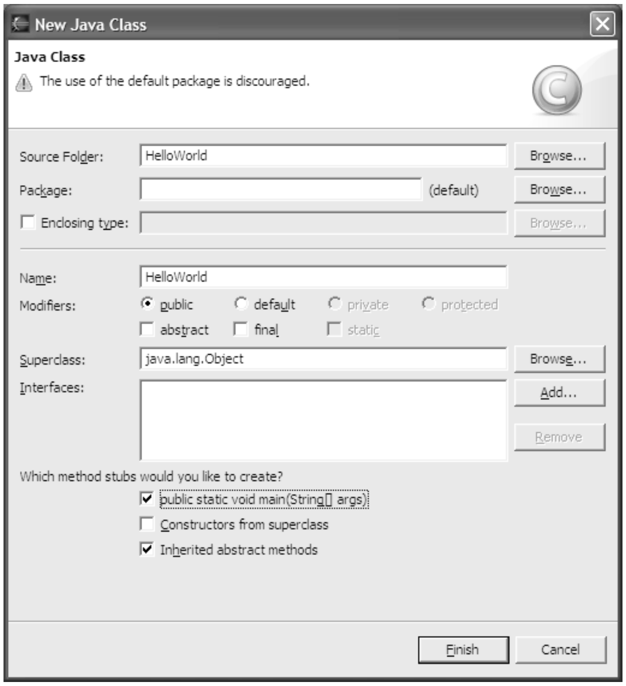
In the dialog that appears, name the project with `HelloWorld`. The Package Explorer now shows an entry for the new project.

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.


New Java Class

Java Class
The use of the default package is discouraged.

Source Folder: HelloWorld
Package: (default)
Enclosing type: [ ]

Name: HelloWorld
Modifiers: [✓ public] default private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[✓] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods

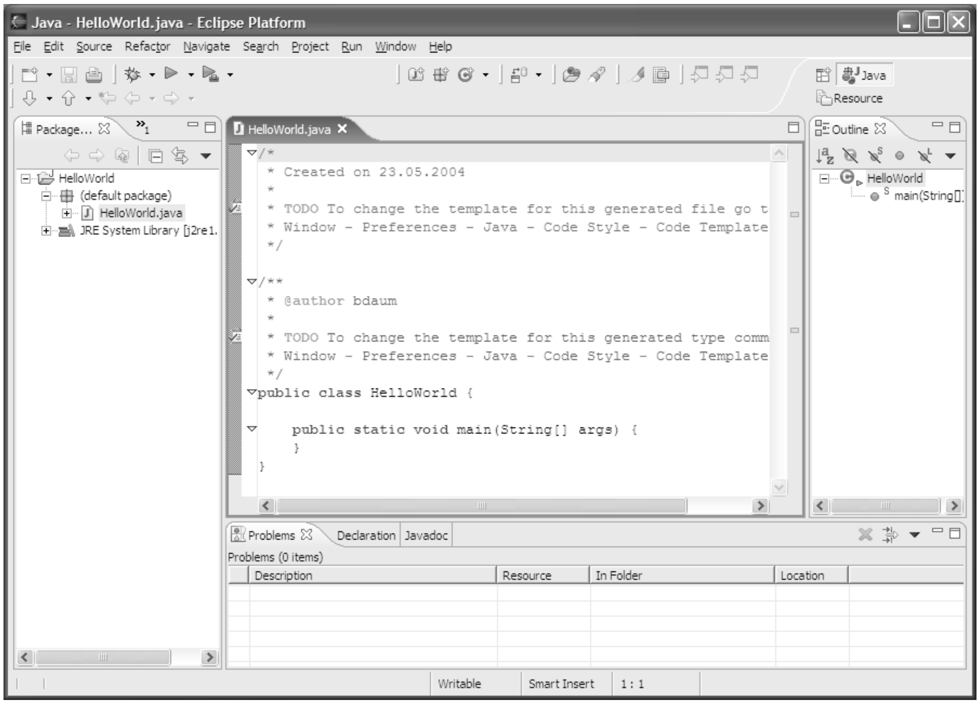
Figure 1.7

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");
}
```



```
/*
 * Created on 23.05.2004
 *
 * TODO To change the template for this generated file go t
 * Window - Preferences - Java - Code Style - Code Template
 */

/**
 * @author bdaum
 *
 * TODO To change the template for this generated type comm
 * Window - Preferences - Java - Code Style - Code Template
 */
public class HelloWorld {

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

Problems (0 items)

Description	Resource	In Folder	Location

Figure 1.8

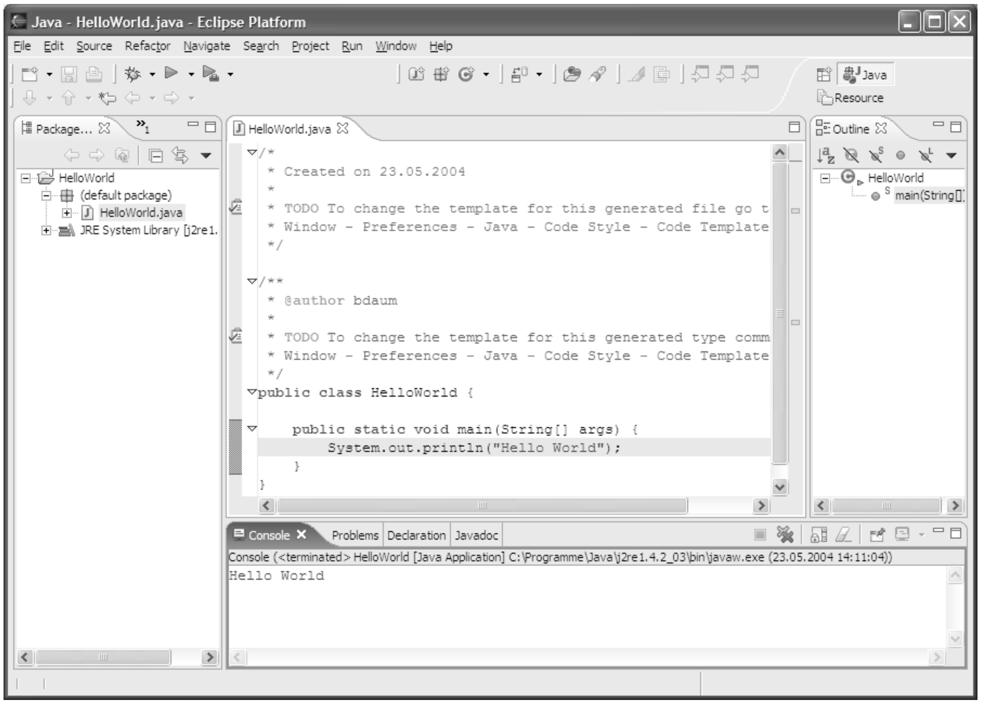
By doing this you have finished the programming work for your first project. Save the new class `HelloWorld` to disk by clicking the floppy disk icon on the toolbar. (Alternatively, you can use the keyboard shortcut Ctrl+S.) This will also compile this class. The program is now ready for execution.

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`.



```
/*
 * Created on 23.05.2004
 *
 * TODO To change the template for this generated file go t
 * Window - Preferences - Java - Code Style - Code Template
 */

/**
 * @author bdaum
 *
 * TODO To change the template for this generated type comm
 * Window - Preferences - Java - Code Style - Code Template
 */
public class HelloWorld {

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

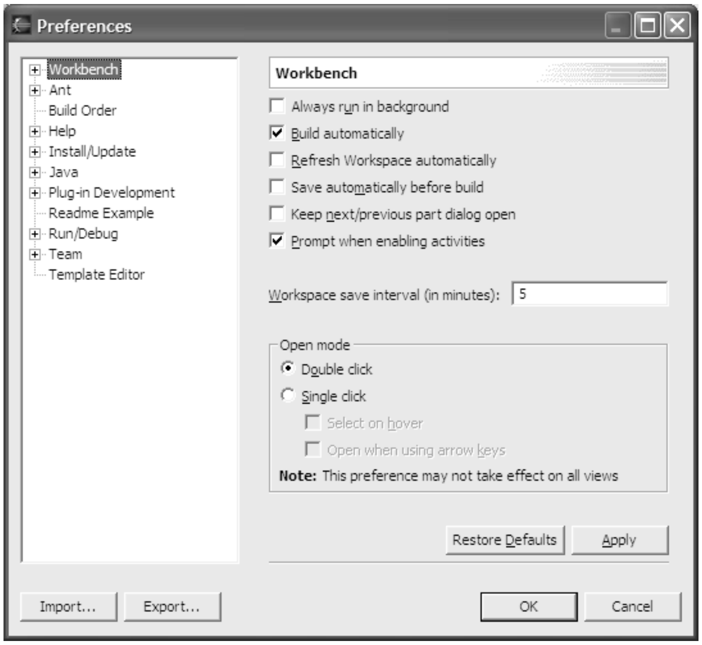
```
Console (<terminated> HelloWorld [Java Application] C:\Programme\Java\j2re1.4.2_03\bin\javaw.exe (23.05.2004 14:11:04))
Hello World
```

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.



Workbench

- [ ] Always run in background
- [✓] Build automatically
- [ ] Refresh Workspace automatically
- [ ] Save automatically before build
- [ ] Keep next/previous part dialog open
- [✓] Prompt when enabling activities

Workspace save interval (in minutes): 5

Open mode
- [● Double click]
- ○ Single click
  - [ ] Select on hover
  - [ ] Open when using arrow keys

Note: This preference may not take effect on all views

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.

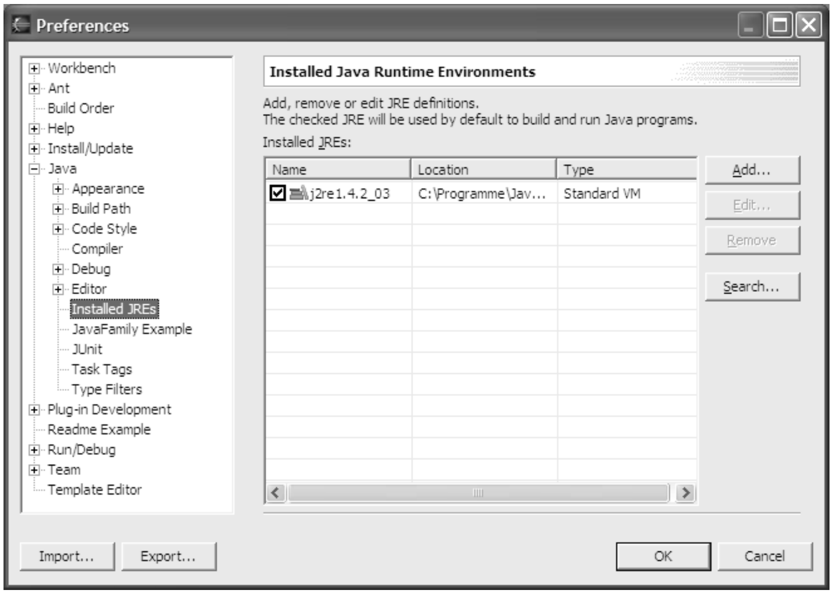
Workbench Preferences

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).



Installed Java Runtime Environments

Add, remove or edit JRE definitions.
The checked JRE will be used by default to build and run Java programs.

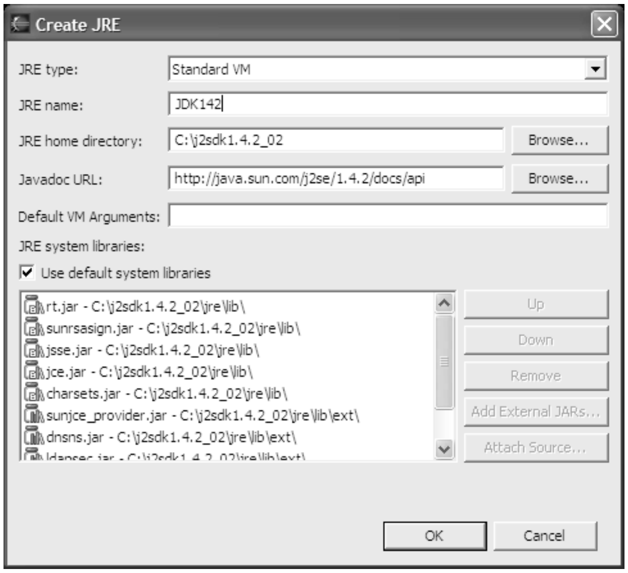
Installed JREs:

Name	Location	Type
☑ j2re1.4.2_03	C:\Programme\Jav...	Standard VM

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).



JRE type: Standard VM
JRE name: JDK142
JRE home directory: C:\j2sdk1.4.2_02
Javadoc URL: http://java.sun.com/j2se/1.4.2/docs/api
Default VM Arguments:
JRE system libraries:
☑ Use default system libraries

```
rt.jar - C:\j2sdk1.4.2_02\jre\lib\
sunrsasign.jar - C:\j2sdk1.4.2_02\jre\lib\
jsse.jar - C:\j2sdk1.4.2_02\jre\lib\
jce.jar - C:\j2sdk1.4.2_02\jre\lib\
charsets.jar - C:\j2sdk1.4.2_02\jre\lib\
sunjce_provider.jar - C:\j2sdk1.4.2_02\jre\lib\ext\
dnsns.jar - C:\j2sdk1.4.2_02\jre\lib\ext\
ldapsec.jar - C:\j2sdk1.4.2_02\jre\lib\ext\
```

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.

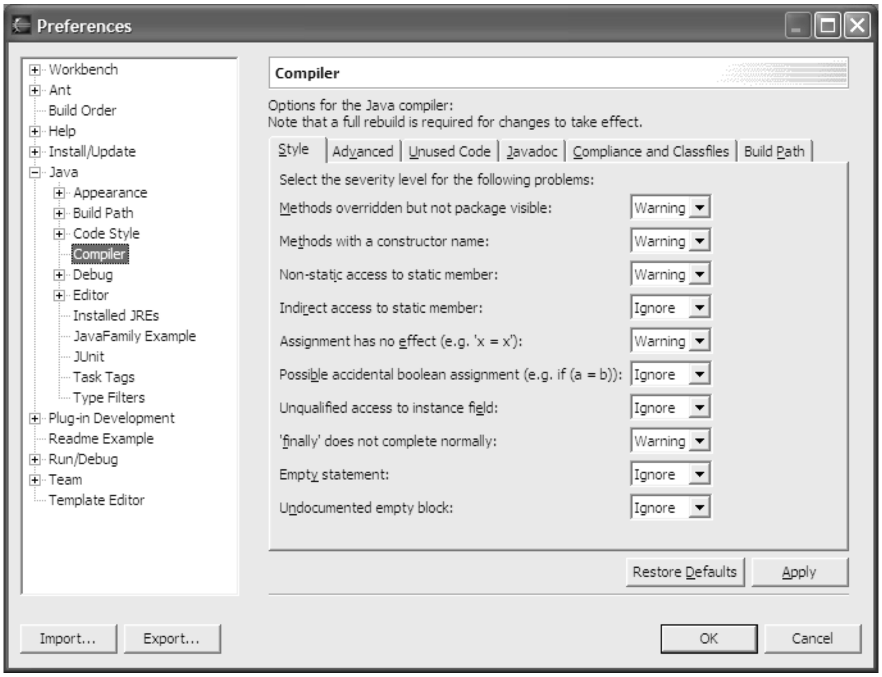
For the remainder of this book I use the Java 1.4 SDK.

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).



Options for the Java compiler:
Note that a full rebuild is required for changes to take effect.

Select the severity level for the following problems:

Methods overridden but not package visible:	Warning
Methods with a constructor name:	Warning
Non-static access to static member:	Warning
Indirect access to static member:	Ignore
Assignment has no effect (e.g. 'x = x'):	Warning
Possible accidental boolean assignment (e.g. if (a = b)):	Ignore
Unqualified access to instance field:	Ignore
'finally' does not complete normally:	Warning
Empty statement:	Ignore
Undocumented empty block:	Ignore

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