

JAVA DEVELOPER'S GUIDE

murach's beginning Java 2 JDK 5

(Chapter 1)

Thanks for downloading this chapter from [Murach's Beginning Java 2, JDK 5](#). To view the full table of contents for this book, you can go to

<http://www.murach.com/books/jav5/toc.htm>

From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our book on Java web programming, [Murach's Java Servlets and JSP](#).

Thanks for your interest in our books!

Doug Lowe
Joel Murach
Andrea Steelman



MIKE MURACH & ASSOCIATES, INC.

1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • www.murach.com

Copyright © 2005 Mike Murach & Associates. All rights reserved.

Section 1

Essential Java skills

This section gets you started quickly with Java programming. First, chapter 1 shows you how to compile and run Java applications, and chapter 2 introduces you to the basic skills that you need for developing Java applications. When you complete these chapters, you'll be able to write, test, and debug simple applications of your own.

After that, chapter 3 presents the details for working with numeric data. Chapter 4 presents the details for coding control statements. And chapter 5 shows you how to validate the data that's entered by the user. These are the essential skills that you'll use in almost every Java application that you develop. When you finish these chapters, you'll be able to write solid programs of your own. And you'll have the background that you need for learning how to develop object-oriented programs.

1

How to get started with Java

Before you can begin learning the Java language, you need to install Java and learn how to use some tools for working with Java. So that's what you'll learn in this chapter. Since most Java developers use Windows, the examples in this chapter show how to use Java with Windows. However, the principles illustrated by these examples apply to all operating systems including Linux, Mac (OS X), and Solaris.

Introduction to Java	4
Toolkits and platforms	4
Java compared to C++	4
Java compared to C#	4
Applications, applets, and servlets	6
How Java compiles and interprets code	8
How to prepare your system for using Java	10
How to install the JDK	10
A summary of the directories and files of the JDK	12
How to set the command path	14
How to set the class path	16
How to use TextPad to work with Java	18
How to install TextPad	18
How to use TextPad to save and edit source code	20
How to use TextPad to compile source code	22
How to use TextPad to run an application	22
Common error messages and solutions	24
How to use the command prompt to work with Java	26
How to compile source code	26
How to run an application	26
How to compile source code with a switch	28
Essential DOS skills for working with Java	30
How to use the documentation for the J2SE API	32
How to install the API documentation	32
How to navigate the API documentation	34
Introduction to Java IDEs	36
Overview of Eclipse	36
When we recommend using a professional IDE	36
Overview of BlueJ	38
Perspective	40

Introduction to Java

In 1996, Sun Microsystems released a new programming language called Java. This language had great promise as a language that could be used on all platforms. Today, Java has fulfilled much of that promise and has established itself as one of the most widely used object-oriented programming languages.

Toolkits and platforms

Figure 1-1 describes all major releases of Java to date starting with version 1 and ending with version 5. Throughout Java's history, Sun has used the terms *Java Development Kit (JDK)* and *Software Development Kit (SDK)* to describe the Java toolkit. In version 5 of Java, Sun favors the term *JDK*. In this book, we'll use the term *JDK* since it's the most current term. In practice, though, *SDK* and *JDK* are often used interchangeably.

Versions 1.2 through 5 of the *JDK* are sometimes referred to as *Java 2* because they both run under the *Java 2 Platform*. This book will show you how to use the *Java 2 Platform, Standard Edition (J2SE)*. Once you master the Standard Edition, you will have all the skills you need for learning how to use the *Java 2 Platform, Enterprise Edition (J2EE)*.

Java compared to C++

When Sun's developers created Java, they tried to keep the syntax for Java similar to the syntax for C++ so it would be easy for C++ programmers to learn Java. In addition, they designed Java so its applications can be run on any computer platform. In contrast, C++ needs to have a specific compiler for each platform. Java was also designed to automatically handle many operations involving the creation and destruction of memory. This led to improved productivity for Java programmers, and it's a key reason why it's easier to develop programs and write bug-free code with Java than with C++.

To provide these features, the developers of Java had to sacrifice some speed (or performance) when compared to C++. For many types of applications, however, Java's relative slowness is not an issue.

Java compared to C#

Microsoft's new Visual C# language is similar to Java in many ways. Like Java, C# uses a syntax that's similar to C++ and that automatically handles memory operations. Also like Java, C# applications can run on any system that has the appropriate interpreter. Currently, however, only Windows provides the interpreter needed to run C# applications. In addition, C# applications are optimized for Windows. Because of that, C# is a good choice for developing applications for a Windows-only environment. However, many of the server computers that store critical enterprise data use Solaris or Linux. As a result, Java remains popular for developing programs that run on these servers.

Java timeline

Year	Month	Event
1996	January	Sun releases Java Development Kit 1.0 (JDK 1.0).
1997	February	Sun releases Java Development Kit 1.1 (JDK 1.1).
1998	December	Sun releases the Java 2 Platform with version 1.2 of the Software Development Kit (SDK 1.2).
1999	August	Sun releases Java 2 Platform, Standard Edition (J2SE).
	December	Sun releases Java 2 Platform, Enterprise Edition (J2EE).
2000	May	Sun releases J2SE with version 1.3 of the SDK.
2002	February	Sun releases J2SE with version 1.4 of the SDK.
2004	September	Sun releases J2SE with version 5.0 (instead of 1.5) of the JDK.

Operating systems supported by Sun

Windows (NT, 95, 98, 2000, XP) Solaris
Linux Macintosh (OS X)

Java compared to C++

Feature	Description
Syntax	Java syntax is similar to C++ syntax.
Platforms	Compiled Java code can be run on any platform that has a Java interpreter. C++ code must be compiled once for each type of system that it is going to be run on.
Speed	C++ runs faster than Java, but Java is getting faster with each new version.
Memory	Java handles most memory operations automatically, while C++ programmers must write code that manages memory.

Java compared to C#

Feature	Description
Syntax	Java syntax is similar to C# syntax.
Platforms	Like compiled Java code, compiled C# code (MSIL) can be run on any system that has the appropriate interpreter. Currently, only Windows has an interpreter for MSIL.
Speed	C# runs faster than Java.
Memory	Both C# and Java handle most memory operations automatically.

Description

- Versions 1.0, 1.1, and 5.0 of the Java toolkit are called the *Java Development Kit*, or *JDK*.
- Versions 1.2, 1.3, and 1.4 of the Java toolkit are called the *Software Development Kit*, or *SDK*.
- The *Java 2 Platform, Standard Edition (J2SE)* supports versions 1.2 to 5.0 of the JDK.
- The *Java 2 Platform, Enterprise Edition (J2EE)* can be used to create enterprise-level, server-side applications.

Applications, applets, and servlets

Figure 1-2 describes the three types of programs that you can create with Java. First, you can use Java to create *applications*. This figure shows an application that uses a *graphical user interface (GUI)* to get user input and perform a calculation. In this book, you'll be introduced to a variety of applications with the emphasis on GUI applications that get data from files and databases.

One of the unique characteristics of Java is that you can use it to create a special type of web-based application known as an *applet*. For instance, this figure shows an applet that works the same way as the application above it. The main difference between an application and an applet is that an applet can be downloaded from a web server and can run inside a Java-enabled browser. As a result, you can distribute applets via the Internet or an intranet.

Although applets can be useful for creating a complex user interface within a browser, they have their limitations. First, you need to make sure a plug-in is installed on each client machine to be able to use the newer Java GUI components (such as Swing components). Second, since an applet runs within a browser on the client, it's not ideal for working with resources that run on the server, such as enterprise databases.

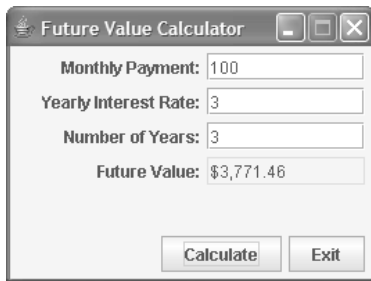
To provide access to enterprise databases, many developers use the Enterprise Edition of the Java 2 Platform (J2EE) to create applications that are based on servlets. A *servlet* is a special type of Java application that runs on the server and can be called by a client, which is usually a web browser. This is also illustrated in this figure. Here, you can see that the servlet works much the same way as the applet. The main difference is that the code for the application runs on the server.

When a web browser calls a servlet, the servlet performs its task and returns the result to the browser, typically in the form of an HTML page. For example, suppose a browser requests a servlet that displays all unprocessed invoices that are stored in a database. Then, when the servlet is executed, it reads data from the database, formats that data within an HTML page, and returns the HTML page to the browser.

When you create a servlet-based application like the one shown here, all the processing takes place on the server and only HTML is returned to the browser. That means that anyone with an Internet or intranet connection, a web browser, and adequate security clearance can access and run a servlet-based application. Because of that, you don't need to install any special software on the client.

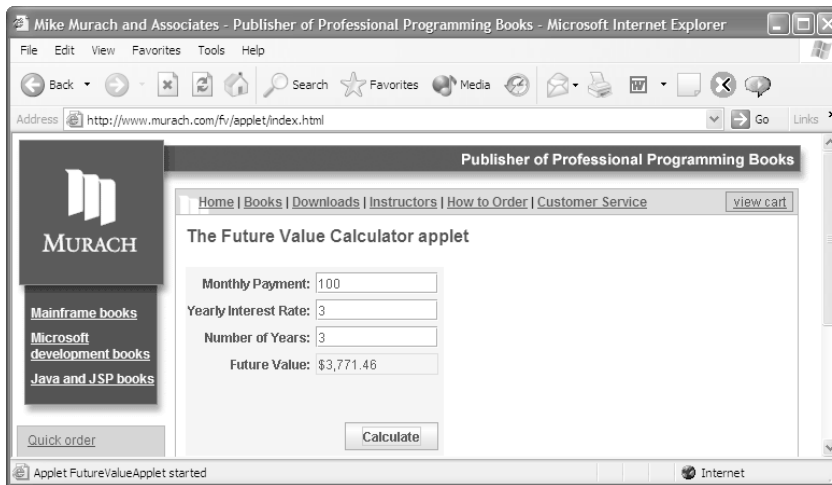
To make it easy to store the results of a servlet within an HTML page, the J2EE specification provides for *JavaServer Pages (JSPs)*. Most developers use JSPs together with servlets when developing server-side Java applications. Although servlets and JSPs aren't presented in this book, we cover this topic in a companion book, *Murach's Java Servlets and JSP*. For more information about this book, please visit our web site at www.murach.com.

An application



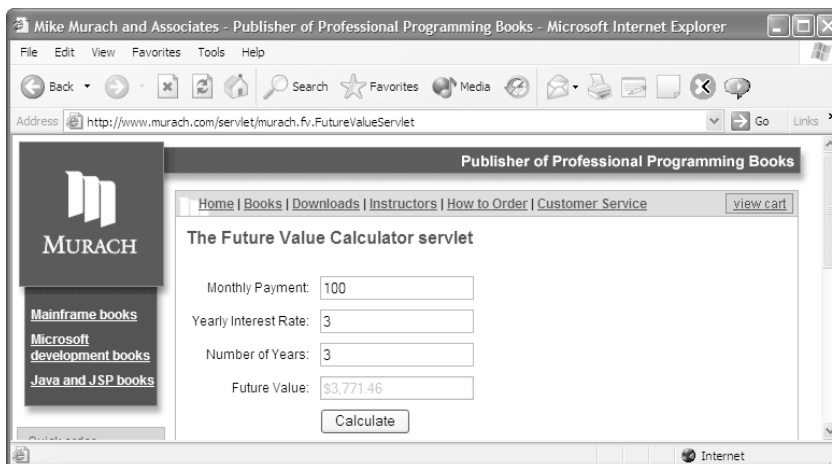
A screenshot of a Java application window titled "Future Value Calculator". The window contains four input fields: "Monthly Payment" with the value 100, "Yearly Interest Rate" with the value 3, "Number of Years" with the value 3, and "Future Value" with the calculated value \$3,771.46. At the bottom of the window are two buttons: "Calculate" and "Exit".

An applet



A screenshot of a Microsoft Internet Explorer browser window. The address bar shows "http://www.murach.com/fv/applet/index.html". The page title is "Publisher of Professional Programming Books". The page content includes a navigation menu with links for Home, Books, Downloads, Instructors, How to Order, and Customer Service. The main content area is titled "The Future Value Calculator applet" and displays the same form as the application window, with input fields for Monthly Payment (100), Yearly Interest Rate (3), Number of Years (3), and Future Value (\$3,771.46), and a Calculate button. The status bar at the bottom indicates "Applet FutureValueApplet started".

A servlet



A screenshot of a Microsoft Internet Explorer browser window. The address bar shows "http://www.murach.com/servlet/murach.fv.FutureValueServlet". The page title is "Publisher of Professional Programming Books". The page content includes a navigation menu with links for Home, Books, Downloads, Instructors, How to Order, and Customer Service. The main content area is titled "The Future Value Calculator servlet" and displays the same form as the application window, with input fields for Monthly Payment (100), Yearly Interest Rate (3), Number of Years (3), and Future Value (\$3,771.46), and a Calculate button. The status bar at the bottom indicates "Internet".

Description

- You can run the applet and servlet versions of the Future Value Calculator application shown above by going to www.murach.com/fv.

How Java compiles and interprets code

When you develop a Java application, you develop one or more *classes*. For each class, you write the Java statements that direct the operation of the class. Then, you use a Java tool to translate the Java statements into instructions that can be run by the computer. This process is illustrated in figure 1-3.

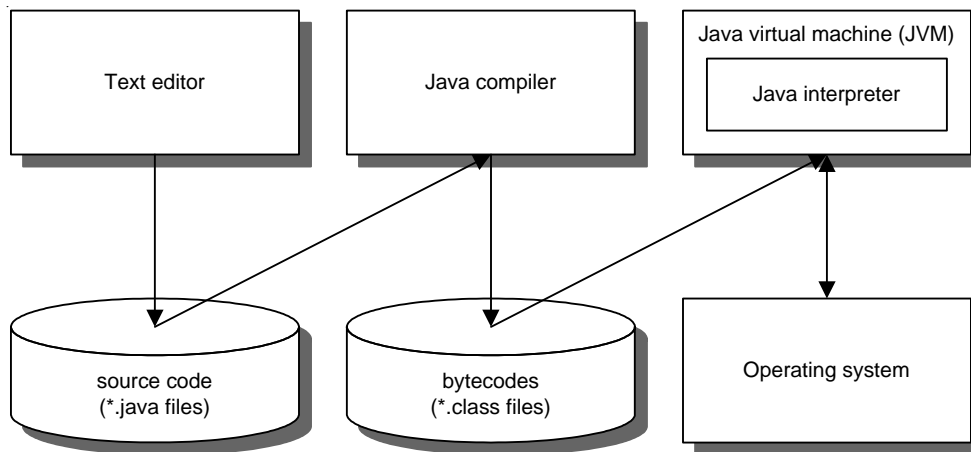
To start, you can use any text editor to enter and edit the Java *source code* for a class. These are the Java statements that tell the application what to do. Then, you use the *Java compiler* to compile the source code into a format known as Java *bytecodes*. At this point, the bytecodes can be run on any platform that has a *Java interpreter* to *interpret* (or translate) the Java bytecodes into code that can be understood by the underlying operating system. The Java interpreter is a concrete implementation of an abstract specification known as the *Java virtual machine (JVM)*. As a result, these two terms are often used interchangeably.

Since Java interpreters are available for all major operating systems, you can run Java on most platforms. This is what gives Java applications their *platform independence*. In contrast, C++ requires a specific compiler for each type of platform that its programs are going to run on.

In addition, some web browsers like Netscape and the Internet Explorer are Java enabled. In other words, these browsers contain Java interpreters. This allows applets, which are bytecodes that are downloaded from the Internet or an intranet, to run within a web browser.

The problem with this is that both Netscape and the Internet Explorer only support older versions of the Java interpreter. In addition, Netscape and the Internet Explorer support slightly different subsets of the Java language. To solve this problem, Sun has developed a tool known as the *Java Plug-in*, which lets the user upgrade the interpreter to a later version. This makes it possible to develop applets that take advantage of the latest features of Java. You'll learn more about this in chapter 18.

How Java compiles and interprets code



Description

- When you develop a Java application, you develop one or more *classes*.
- You can use any text editor to create, edit, and save the *source code* for a Java class. Source code files have the *java* extension.
- The *Java compiler* translates Java source code into a *platform-independent* format known as *Java bytecodes*. Files that contain Java bytecodes have the *class* extension.
- The *Java interpreter* executes Java bytecodes. Since Java interpreters exist for all major operating systems, Java bytecodes can be run on most platforms. A Java interpreter is an implementation of a *Java virtual machine (JVM)*.
- Some web browsers like Netscape and the Internet Explorer contain Java interpreters. This lets applets run within these browsers. However, both Netscape and the Internet Explorer only provide older versions of the Java interpreter.
- Sun provides a tool known as the *Java Plug-in* that allows you to specify the version of the Java virtual machine that you want to use.

Figure 1-3 How Java compiles and interprets code

How to prepare your system for using Java

Before you can develop Java applications, the JDK must be installed on your system. In addition, your system may need to be configured to work with the JDK. Once you install the JDK, you'll be ready to create your first Java application.

How to install the JDK

Figure 1-4 shows how to install the JDK. To start, you download the exe file for the setup program for the most recent version of the JDK from the Java web site. Then, you navigate to the directory that holds the exe file, run the setup file, and respond to the resulting dialog boxes.

Since Sun periodically updates the Java web site, we've kept the procedure shown in this figure somewhat general. As a result, you may have to do some searching to find the current version of the JDK. In general, you can start by looking for products for the Java 2 Platform, Standard Edition. Then, you can find the most current version of the JDK for your operating system.

By the way, all of the examples in this book have been tested against version 5.0 of the JDK. Since Java has a good track record of being upwards compatible, however, these examples should work equally well with later versions of the JDK.

The Java web site

`java.sun.com`

How to download the JDK from the Java web site

1. Go to the Java web site.
2. Locate the list of Java products, and display the page for the Java 2 Platform, Standard Edition.
3. Display the J2SE Downloads page and select the link for the most current JDK version that's available for your operating system.
4. Click the Download link for the JDK (not the JRE), and follow the instructions.
5. Save the exe file for the setup program to your hard disk.

How to install the JDK

- Run the exe file and respond to the resulting dialog boxes. When you're prompted for the JDK directory, use the default directory. For most Windows systems, the default directory is C:\Program Files\Java\jdk1.5.0 for version 5 of the JDK.

Notes

- For more information about installing the JDK, you can refer to the Java web site.
- If you are installing the Windows version of the JDK, you can perform either an offline or an online installation. An online installation is faster because only a small setup file is downloaded. However, you must remain online during the entire installation so the required files can be installed.
- You can click the "How long will it take" link to determine how long it will take to download the setup program based on its size and the speed of your modem. For a 56K modem, for example, it will take about 2 hours and 13 minutes to download the Windows version of JDK 5.0 (about 44MB).

A summary of the directories and files of the JDK

Figure 1-5 shows the directories and files that are created when you install the JDK. Here, the JDK is stored in the C:\Program Files\Java\jdk1.5.0 directory. This directory has multiple subdirectories, but the *bin*, *lib*, and *docs* directories are the most important.

The *bin* directory holds all the tools necessary for developing and testing a program, including the Java compiler. Later in this chapter, you'll learn how to use these tools to compile and run Java applications. The *lib* directory contains libraries and support files required by the development tools.

The *jre* directory contains the Java interpreter, or *Java Runtime Environment (JRE)*, that's needed to run Java applications once they've been compiled. Although the JDK uses this internal version of the JRE, you can also download a standalone version of the JRE from the Java web site. Once you're done developing a Java application, for example, you can distribute the standalone JRE to other computers so they can run your application.

The *docs* directory can be used to store the Java documentation. Later in this chapter, you'll learn how to download and install this documentation.

In the JDK directory, you can find an HTML *readme* file that contains much of the information that's presented in this figure as well as more technical and detailed information about the JDK. You can view the HTML file with a web browser.

The JDK directory also contains the *src.zip* file. This is a compressed file that holds the source code for the JDK. If you want to view the source code, you can extract the source files from this zip file. If you're curious to see the Java code of the JDK, you may want to do that once you understand Java better.

When you work with Windows, you'll find that it sometimes uses the terms *folder* and *subfolder* to refer to directories and subdirectories. For consistency, though, we use the term *directory* throughout this book. In practice, these terms are often used interchangeably.

The default directory for the JDK on a Windows machine

C:\Program Files\Java\jdk1.5.0\bin

Four important subdirectories of the JDK

Directory	Description
bin	The Java development tools and commands
jre	The root directory of the Java Runtime Environment (JRE)
lib	Additional libraries of code that are required by the development tools
docs (optional)	The on-line documentation that you can download (see figure 1-15)

Two important files stored in the JDK directory

File	Description
readme.html	An HTML page that provides information on Java 2, including system requirements, features, and documentation links.
src.zip	A zip file containing the source code for the J2SE API. If you use a zip tool such as WinZip to extract these directories and files, you can view the source code for the JDK.

Description

- The *Java Runtime Environment (JRE)* is the Java interpreter that allows you to run compiled programs in Java. The jre directory is an internal copy of the runtime environment that works with the JDK. You can also download a standalone version of the JRE for computers that don't have the JDK installed on them.

How to set the command path

Figure 1-6 shows you how to configure Windows to make it easier to work with the JDK. If you're not using Windows, you can refer to the Java web site to see what you need to do to configure Java for your system.

To configure Windows to work with the JDK, you need to add the bin directory to the *command path*. That way, Windows will know where to look to find the Java commands that you use.

If you're using a recent version of Windows like Windows 2000, NT, or XP, you can use the first procedure in this figure to set the command path. When you find the Path variable in the Environment Variables, you can usually add the path for the Java bin directory to the end of the list of paths. To do that, you type a semicolon and the complete path (shaded at the top of this figure). However, if you've installed previous versions of Java on your system, you need to make sure that the path for Java 5.0 is in front of the path for earlier versions.

If you're using an older version of Windows like Windows 95, 98, or ME, you can use the second procedure in this figure to edit the Path or Set Path command in the *autoexec.bat* file. This is the file that is automatically executed every time you start your computer. After you edit the file, you can restart your computer to run the *autoexec.bat* file and establish the new path. Then, you can enter *path* at the command prompt to make sure that the bin subdirectory of the JDK directory is now in the command path.

Whenever you edit the command path, be careful! Since the command path may affect the operation of other programs on your PC, you don't want to delete or modify any of the other paths. You only want to add one directory to the command path. If that doesn't work, be sure that you're able to restore the command path to its original condition.

If you don't configure Windows in this way, you can still compile and run Java programs, but it's more difficult. For instance, instead of typing a command like this to compile a Java class

```
javac
```

you may need to type this command:

```
\Program Files\Java\jdk1.5.0\bin\javac
```

As you can see, then, setting the command path makes this much simpler. That's why it's so important to get this set right.

A typical Path command

```
path=C:\Windows;C:\Program Files\Java\jdk1.5.0\bin
```

How to set the path for Windows 2000/NT/XP

1. Depending on how your system is set up, (1) go to the Start menu, and select the Control Panel, or (2) go to the Start menu, select Settings, and select the Control Panel.
2. Edit the environment variables for your system.
 - For 2000, select the System icon, the Advanced tab, and the Environment Variables button.
 - For NT, select the System icon and the Environment tab.
 - For XP, select the Performance and Maintenance icon, the System icon, the Advanced tab, and the Environment Variables button.
3. If you haven't installed earlier versions of Java, type a semicolon and the path for the bin subdirectory of JDK 1.5 to the far right of the list of paths. Otherwise, add the 1.5 path followed by a semicolon before the paths for earlier JDK versions.

How to set the path for Windows 95/98/ME

1. Start the Windows Explorer and navigate to the autoexec.bat file, which should be stored in the root drive (c:\autoexec.bat).
2. Right-click on the autoexec.bat file and select the Edit command. This should open the file in a text editor.
3. If the file doesn't contain a Path or Set Path command, enter "path=" at the beginning of the file followed by the bin subdirectory of JDK 1.5. If the file does contain a Path or Set Path command and you haven't installed earlier versions of Java, type a semicolon at the end of the command followed by the path of the bin subdirectory of JDK 1.5. However, if you have installed earlier Java versions, add the 1.5 path followed by a semicolon before the paths of any earlier versions.
4. Save the file and exit the text editor.
5. To have the new path take effect, run the autoexec.bat file by restarting your computer.

How to check the current path

- Start a Command Prompt or DOS Prompt as described in figure 1-12. Then, enter the Path command by typing the word path. This will display the current path statement.

Description

- The *command path* on a Windows system tells Windows where to look for the commands that it is told to execute. When you use Java tools, you need to add the path for the jdk1.5.0\bin directory that's shown above.

Notes

- For more information about setting the path for Windows or for information about configuring non-Windows operating systems, you can refer to the Java web site.

How to set the class path

The *class path* is used to tell the operating system where to look for the Java files that you create. When you compile a class, for example, the operating system needs to know where to find the source code (.java files) for that class. Similarly, when you run a program, the operating system needs to know where to find the bytecodes (.class files) for that class.

By default, the class path for a Windows system includes the current directory, which is usually all you need. Then, before you compile or run a program from a command prompt, you simply change to the directory that contains the .java or .class files. You'll learn more about how to display and work from the command prompt later in this chapter.

The operating system also uses the class path when you compile or run a program using some Java tools such as TextPad. In that case, the tool automatically changes the current directory to the directory where the source code and class files for the program you're compiling or running are stored.

So as long as the class path for your system includes the current directory, the tool will work correctly. If the class path doesn't include the current directory, however, you'll need to add it before the tool will work correctly. Figure 1-7 describes how you do that.

To set the class path, you use the same techniques that you use for setting the command path (see figure 1-6), but you add a path to the Classpath list instead of the Path list. Specifically, you add a dot (.) for the current directory followed by a semicolon at the start of the list of paths. This is illustrated in the class path at the top of this figure.

To determine what the current class path is, you can enter *set* at the prompt. This runs the Set command, which displays a variety of settings, including the class path. This will also show you whether you've successfully added the current directory.

A typical Classpath command

```
classpath=.;c:\java\classes;
```

When and how to modify the class path

- If your system doesn't have a Classpath command, the default class path will allow you to run all of the programs described in this book. As a result, you won't need to modify the class path.
- If your system has a Classpath command that doesn't include the current directory, you need to add the current directory to the class path.
- To modify the class path, follow the procedure shown in figure 1-6 for modifying the command path, but modify the Classpath command instead. To include the current directory in the class path, just code a period for the directory followed by a semicolon at the start of the list of paths as shown above.

How to check the current class path

- Start a Command Prompt or DOS Prompt as described in figure 1-12, and enter the Set command by typing the word *set*. This will display a variety of settings including the current class path.

Description

- The *class path* tells the JDK where to find the .java and .class files you create. The class path is used when you compile or run a program. By default, the class path is the current directory.

Note

- For more information about configuring the class path or for information about configuring non-Windows operating systems, you can refer to the Java web site.

How to use TextPad to work with Java

Once the JDK is installed and configured for your operating system, you're ready to create your first application. Since most Java development is still done under Windows, this topic shows how to install and use a free trial version of TextPad, one of the most popular *text editors* that's designed for Java development.

Unfortunately, TextPad only runs under Windows. As a result, if you're using a non-Windows computer, you'll need to search the web to find a text editor for Java development that runs on the operating system that you're using. Fortunately, you can find a variety of these types of text editors on the web, and many of them are available for free (freeware) or for a small fee (shareware).

How to install TextPad

Figure 1-8 shows how to download and install a free trial version of TextPad. Once you save the exe for the setup file to your hard disk, you simply run this file and respond to the resulting dialog boxes. Since this version of TextPad is a trial version, you should pay for TextPad if you decide to use it beyond the initial trial period. Fortunately, this program is relatively inexpensive (about \$30), especially when you consider how much time and effort it can save you.

The TextPad web site

www.textpad.com

How to download TextPad

1. Go to the TextPad web site.
2. Find the Free Trial version of TextPad and save the exe file for the setup program to your hard disk. This should take just a few minutes.

How to install TextPad

- Run the exe file and respond to the resulting dialog boxes.

Notes

- The trial version of TextPad is free, but if you like TextPad and continue to use it, you can pay the small fee of approximately \$30 to purchase it.
- The examples in this chapter were created with version 4.7 of TextPad.
- Although TextPad is a popular text editor for doing Java development under Windows, it doesn't run on Linux, Macintosh, or Solaris. As a result, if you're using a non-Windows operating system, you'll need to find a text editor for Java development that runs on your operating system. For example, VI and Emacs are two popular text editors for Linux.

How to use TextPad to save and edit source code

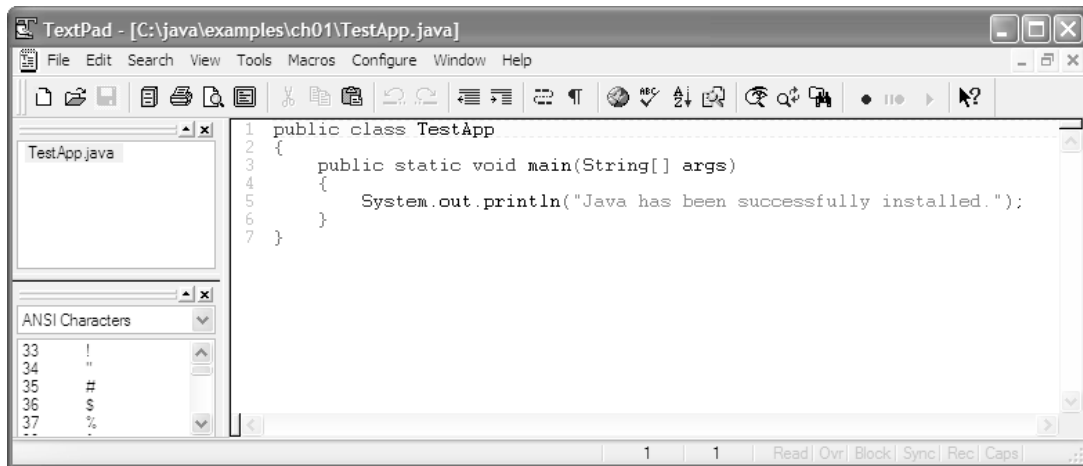
Figure 1-9 shows how to use TextPad to save and edit source code. In short, you can use the standard Windows shortcut keystrokes and menus to enter, edit, and save your code. You can use the File menu to open and close files. You can use the Edit menu to cut, copy, and paste text. And you can use the Search menu to find and replace text. In addition, TextPad color codes the code in the source files so it's easier to recognize the Java syntax.

When you save Java source code, you must give the file the same name as the class name. Since Java is a *case-sensitive* language, you must also use the same capitalization in the file and class names. If you don't, you'll get an error message when you try to compile the code. In this figure, for example, you can see that "TestApp" is used for both the class name and the file name.

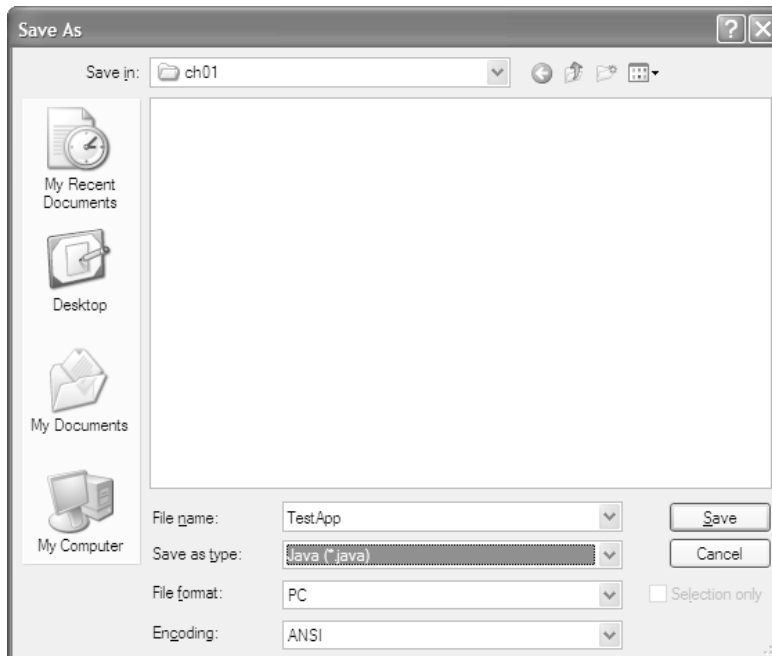
You must also save a Java source file with the four-letter *java* extension. When you use TextPad to save your source code, you can use the Save As Type drop-down list in the Save As dialog box to apply this extension. However, if you're using a text editor that isn't designed for working with Java, you may need to add the *java* extension to the end of the file name and enclose the file name in quotation marks like this: "TestApp.java". Otherwise, the text editor may truncate the extension to *jav* or change the capitalization in the file name. This will lead to errors when you try to compile the source code.

You must also save Java source code in a standard text-only format such as the *ASCII format* or the *ANSI format*. By default, TextPad saves code in the ANSI format and that's usually what you want. If, however, you need to save a file in another format such as Unicode, TextPad can do that too.

The TextPad text editor with source code in it



TextPad's Save As dialog box



How to enter, edit, and save source code

- To enter and edit source code, you can use the same techniques that you use for working with any other Windows text editor.
- To save the source code, select the Save command from the File menu (Ctrl+S). Then, enter the file name so it's exactly the same as the class name, and select the Java option from the Save As Type list so TextPad adds the four-letter java extension to the file name.

Figure 1-9 How to use TextPad to save and edit source code

How to use TextPad to compile source code

Figure 1-10 shows how to use TextPad to compile the source code for a Java application. The quickest way to do that is to press Ctrl+1 to execute the Compile Java command of the Tools menu. If the source code compiles cleanly, TextPad will generate a Command Results window and return you to the original source code window.

However, if the source code doesn't compile cleanly, TextPad will leave you at a Command Results window like the one shown in this figure. In this case, you can read the error message, switch to the source code window, correct the error, and compile the source code again. Since each error message identifies the line number of the error, you can make it easier to find the error by selecting the Line Number option from the View menu. That way, TextPad will display line numbers as shown in this figure.

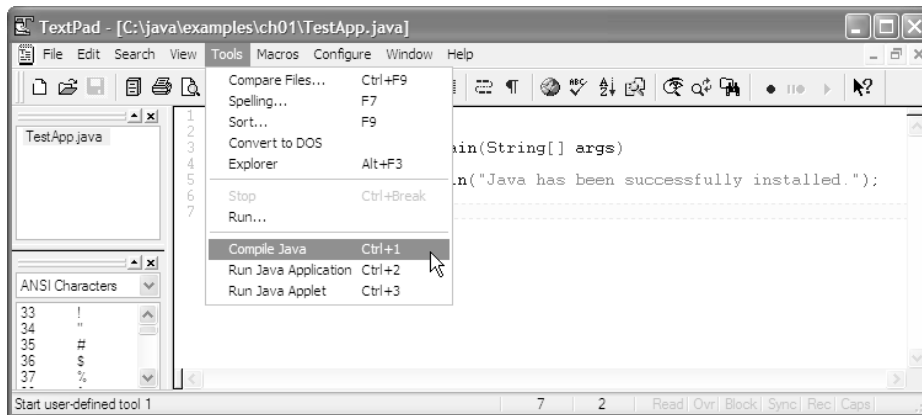
When you have several Java files open at once, you can use the Document Selector pane to switch between files. In this figure, only two documents are open (TestApp and Command Results), but you can open as many files as you like. You can also use the Window menu and standard Windows keystrokes (Ctrl+F6 and Ctrl+Shift+F6) to switch between windows.

To edit as efficiently as possible, you can use the View menu to set the editing options for a single file. And you can use the Configure→Preferences command to set the options for all files. In particular, you may want to turn the Line Number option on, and you may want to set the tab settings so you can easily align the code in an application.

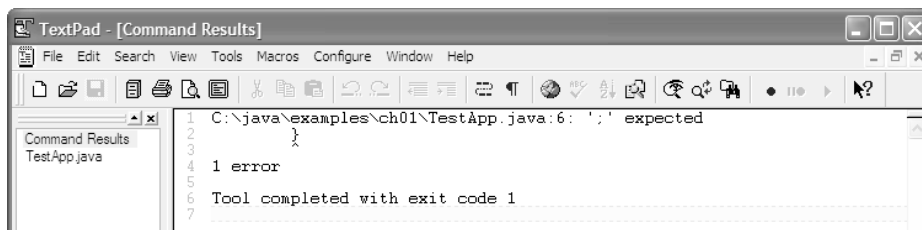
How to use TextPad to run an application

Once you've compiled the source code for an application, you can run that application by pressing Ctrl+2. In this figure, the application prints text to the *console*. As a result, TextPad starts a console window like the one shown in this figure. Then, you can usually press any key to end the application and close the window. Sometimes, however, you may need to click on the Close button in the upper right corner of the window to close it.

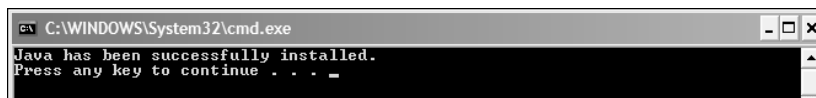
The Tools menu



A compile-time error



Text printed to the console



How to compile and run an application

- To compile the current source code, press Ctrl+1 or select Tools→Compile Java.
- To run the current application, press Ctrl+2 or select Tools→Run Java Application.
- If you encounter compile-time errors, TextPad will print them to a window named Command Results. To switch between this window and the source code window, press Ctrl+F6 or use the Document Selector pane that's on the left side of the TextPad window.
- When you print to the *console*, a DOS window like the one above is displayed. To close the window, press any key or click the Close button in the upper right corner.

How to display line numbers and set options for one source file

- To display the line numbers for the source code, select View→Line Numbers.
- To set formatting options like tab settings, select View→Document Properties.

How to display line numbers and set the options for all source files

- Select Configure→Preferences. Then, in the Preferences dialog box, click on the type of default that you would like to set, and check or uncheck the appropriate options. For instance, to display line numbers for all files, click on View and click on Line Numbers.

Figure 1-10 How to use TextPad to compile and run an application

Common error messages and solutions

Figure 1-11 summarizes some common error messages that you may encounter when you try to compile or run a Java application. The first two errors illustrate *compile-time errors*. These are errors that occur when the Java compiler attempts to compile the program. In contrast, the third error illustrates a *runtime error*. That is an error that occurs when the Java interpreter attempts to run the program but can't do it.

The first error message in this figure involves a syntax error. When the compiler encounters a syntax error, it prints two lines for each error. The first line prints the name of the *.java file, followed by a colon, the line number for the error, another colon, and a brief description of the error. The second line prints the code that caused the error, including a caret character that tries to identify the location where the syntax error occurred. In this example, the caret points to the right brace, but the problem is that the previous line didn't end with a semicolon.

The second error message in this figure involves a problem defining the *public class* for the file. The compiler displays an error message like this when the file name for the *.java file doesn't match the name of the public class defined in the source code. For example, the TextApp.java file must contain this code

```
public class TestApp{
```

If the name of the file doesn't match the name of the public class (including capitalization), the compiler will give you an error like the one shown in this figure. You'll learn more about the syntax for defining a public class in the next chapter.

The third error message in this figure occurs if you try to run an application that doesn't have a main method. You'll learn how to code a main method in the next chapter. For now, just realize that every application must contain a main method. To correct the error shown in this figure, then, you must add a main method to the class or run another class that has a main method.

Most of the time, the information displayed by an error message will give you an idea of how to fix the problem. Sometimes, though, the compiler gets confused and doesn't give you accurate error messages. In that case, you'll need to double-check all of your code. You'll learn more about debugging error messages like these as you progress through this book.

A common compile-time error message and solution

Error: `C:\java\examples\ch01\TestApp.java:6: ';' expected`

```

    }
    ^

```

Description: The first line in this error message displays the file name of the *.java file, a number indicating the line where the error occurred, and a brief description of the error. The second line displays the line of code that may have caused the error with a caret symbol (^) below the location where there may be improper syntax.

Solution: Edit the source code to correct the problem and compile again.

Another common compile-time error message and solution

Error: `C:\java\examples\ch01\TestApp.java:1: class testapp is public, should be declared in a file named testapp.java`

```

public class testapp
    ^

```

Description: The *.java file name doesn't match the name of the public class. You must save the file with the same name as the name that's coded after the words "public class". In addition, you must add the java extension to the file name.

Solution: Edit the class name so it matches the file name (including capitalization), or change the file name so it matches the class name. Then, compile again.

A common runtime error message and solution

Error: `Exception in thread "main" java.lang.NoSuchMethodError: main`

Description: The class doesn't contain a main method.

Solution: Run a different class that does have a main method, or enter a main method for the current class.

Description

- When you compile an application that has some statements that aren't coded correctly, the compiler cancels the compilation and displays messages that describe the *compile-time errors*. Then, you can fix the causes of these errors and compile again.
- When the application compiles without errors (a "clean compile") and you run the application, a *runtime error* occurs when the Java Virtual Machine can't execute a compiled statement. Then, the application is cancelled and an error message is displayed.

How to use the command prompt to work with Java

If you're using TextPad, you can use its commands to compile and run most of your Java applications. Even so, there may be times when you will need to compile and run Java applications from the *command prompt*. And if you're using another text editor that doesn't provide compile and run commands, you can use the command prompt to compile and run all of your Java applications.

How to compile source code

Figure 1-12 shows how to use a command prompt to compile and run applications. In particular, this figure shows how to use the Windows command prompt, sometimes called the *DOS prompt*.

To start, you enter the change directory (`cd`) command to change the current directory to the directory that holds the application. In this figure, for example, you can see that the directory has been changed to `c:\java\examples\ch01` because that's the directory that contains the `TestApp.java` file. Then, to compile the application, you use the *javac command* to start the Java compiler. When you enter the `javac` command, you follow it by a space and the complete name of the `*.java` file that you want to compile. Since Java is case-sensitive, you need to use the same capitalization that you used when you saved the `*.java` file.

If the application doesn't compile successfully, you can use your text editor to correct and resave the `*.java` file, and you can compile the program again. Since this means that you'll be switching back and forth between the text editor and the command prompt, you'll want to leave both windows open.

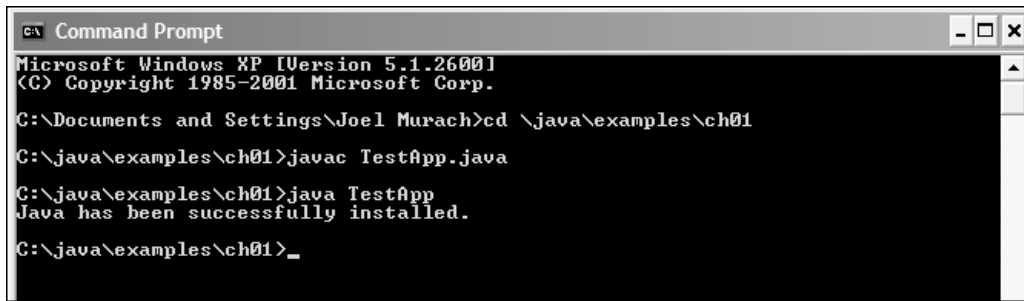
When you compile an application successfully, the Java compiler will create a `*.class` file that has the same file name as the `*.java` file. For example, a successful compilation of the `TestApp.java` file will create the `TestApp.class` file. This `*.class` file is the file that contains the Java bytecodes that can be run by the Java interpreter.

How to run an application

To run a program from the command prompt, you use the *java command* to start the Java interpreter. Although you need to use the proper capitalization when you use the `java` command, you don't need to include an extension for the file. When you enter the `java` command correctly, the Java interpreter will run the `*.class` file for the application.

Running a Java program often displays a graphical user interface like the one shown in figure 1-2. However, you can also print information to the console and get input from the console. For example, the `TestApp` file in this figure prints a single line of text to the console.

The commands for compiling and running an application



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Joel Murach>cd \java\examples\ch01
C:\java\examples\ch01>javac TestApp.java
C:\java\examples\ch01>java TestApp
Java has been successfully installed.
C:\java\examples\ch01>_
```

Syntax to compile an application

```
javac ProgramName.java
```

Syntax to run an application

```
java ProgramName
```

Description

- The *command prompt* is the prompt that indicates that the operating system is waiting for the next command. This prompt usually shows the current directory, and it always ends with >.
- In Windows, the command prompt is sometimes referred to as the *DOS prompt* or the *DOS window*. You can enter DOS commands at the DOS prompt.

Operation

- To open a command prompt in Windows, click on the Start button, find MS-DOS Prompt or Command Prompt, and select it. If its location isn't obvious, try looking in Accessories.
- To change the current directory to the directory that contains the file with your source code, use the change directory command (cd) as shown above.
- To compile the source code, enter the Java compile command (javac), followed by the file name (including the java extension). Since this is a case-sensitive command, make sure to use the same capitalization that you used when naming the file.
- If the code compiles successfully, the compiler generates another file with the same name, but with class as the extension. This file contains the bytecodes.
- If the code doesn't compile successfully, the java compiler generates error messages for the compile-time errors. Then, you must switch back to your text editor, fix the errors, save your changes, and compile the program again.
- To run the compiled version of your source code, enter the Java command (java), followed by the program name (without any extension). Since this is a case-sensitive command, make sure to use the same capitalization that you used when naming the file.

Note

- The code shown in the command prompt above will only work if the bin subdirectory of the JDK 5.0 directory has been added to the command path as shown in figure 1-6.

Figure 1-12 How to use the command prompt to compile and run an application

How to compile source code with a switch

Most of the time, you can use TextPad's compile command or the plain `javac` command shown in the last figure to compile Java source code. Sometimes, however, you need to supply a *switch* to use certain features of the `javac` command as summarized in figure 1-13.

Note in the syntax summary in this figure that the brackets around the switch name indicate that it's optional. In addition, the ellipsis (...) indicates that you can code as many switches as you need. You'll see this notation used throughout this book.

When new versions of Java become available, the designers of Java mark some older features as *deprecated*. When a feature is deprecated, it means that its use is not recommended and it may not be supported in future versions of Java. As a result, you should try to avoid using deprecated features whenever possible.

The first example in this figure shows how to use the deprecation switch to get more information about code that uses deprecated features of the JDK. As you can see, if you don't use this switch, you'll get a brief message that indicates that your code uses deprecated features. On the other hand, if you use this switch, you'll get detailed information about the code that uses deprecated features including the line number for that code. That makes it easier to modify your code so it doesn't use these features.

The second example shows how to use the source switch to compile code so it can run on another computer that's using an older Java interpreter. In particular, this example shows how to compile code so it can run on a computer with version 1.4 of the JRE. When you use this switch, though, you won't be able to use the new features of Java 5. As a result, you shouldn't use this switch if you want to use the new Java 5 features that are presented in this book.

This figure also shows how to use a switch with TextPad. To do that, you begin by selecting the `Configure→Preferences` command, which displays the Preferences dialog box. Then, you expand the Tools group and select `Compile Java`, which displays the options for compiling Java. At that point, you can add a switch by typing it at the end of the parameters in the Parameters text box. In this figure, for example, the deprecation switch has been added to the end of the Parameters text box. As a result, the deprecation feature will be on for future TextPad sessions until you remove it from this dialog box.

Keep in mind, though, that you don't need to use either of the two switches shown in this figure as you use this book. Since the book is designed to teach you Java 5, you shouldn't need the source switch to specify the use of an earlier version. Since this book doesn't teach the deprecated features, you shouldn't need more information about them. However, you may occasionally need to use one or both of these switches.

Syntax to compile source code with a switch

```
javac ProgramName.java -switch1Name [-switch2Name]...
```

Two javac switches

Name	Description
deprecation	You can use this switch to get additional information about any deprecated features that your source code uses.
source	You can use this switch followed by the version number (1.3 or 1.4) to compile code that works with older versions of the Java interpreter.

How to compile code that uses deprecated features

Without the deprecation switch (you get notes)

```
C:\java\ch01>javac LoanCalculatorFrame.java
Note: LoanCalculatorFrame.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

With the deprecation switch (you get details)

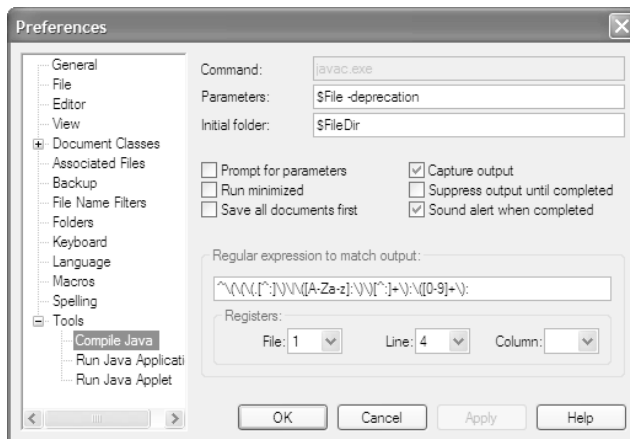
```
C:\java\ch01>javac LoanCalculatorFrame.java -deprecation
LoanCalculatorFrame.java:34: warning: [deprecation] show() in
java.awt.Window has been deprecated
        frame.show();
                ^
1 warning
```

1 warning

How to compile code with the source and deprecation switches

```
C:\java1.5\ch01>javac TestApp.java -source 1.4 -deprecation
```

The Preferences dialog box for TextPad with the deprecation switch on



Description

- If you want to use the Java 1.5 features and you don't want to get the extra information for *deprecated features*, you don't need to set any switches.
- To use a switch with TextPad, select **Configure**→**Preferences**, expand the **Tools** group, select the **Compile Java** group, and add the switch to the end of the **Parameters** text box.

Figure 1-13 How to compile source code with a switch

Essential DOS skills for working with Java

Figure 1-14 summarizes some of the most useful commands and keystrokes for working with DOS. In addition, it shows how to install and use a DOS program called DOSKey, which makes entering and editing DOS commands easier. If you're going to use DOS to work with Java, you should review these DOS commands and keystrokes, and you will probably want to turn on the DOSKey program if it isn't already on. If you aren't going to use DOS, of course, you can skip this figure.

At the top of this figure, you can see a DOS window that shows two DOS commands and a directory listing. Here, the first command changes the current directory to `c:\java\exercises\ch01`. The next command displays a directory listing. If you study this listing, you can see that this directory contains two files with one line of information for each file. At the right side of each line, you can see the complete file names for these two files (`TestApp.class` and `TestApp.java`), and you can see the capitalization for these files as well.

If you master the DOS commands summarized in this figure, you should be able to use DOS to work with Java. To switch to another drive, type the letter of the drive followed by a colon. To change the current directory to another directory, use the `cd` command. To display a directory listing for the current directory, use the `dir` command. Although DOS provides many more commands that let you create directories, move files, copy files, and rename files, you can also use the Windows Explorer to perform those types of tasks.

Although you don't need to use the DOSKey program, it can save you a lot of typing and frustration. If, for example, you compile a program and you encounter a syntax error, you will need to use a text editor to fix the error in the source code. Then, you will need to compile the program again. If you're using DOSKey, you can do that by pressing the up-arrow key to display the last command that was executed and then pressing the Enter key to execute the command. And if you make a mistake when entering a command, you can use the left- and right-arrow keys to edit the command instead of having to enter the entire command again.

A directory listing

```

C:\Documents and Settings\Joel Murach>cd \java\exercises\ch01
C:\java\exercises\ch01>dir
Volume in drive C has no label.
Volume Serial Number is A0E5-29E9

Directory of C:\java\exercises\ch01

07/08/2004  05:11 PM    <DIR>          -
07/08/2004  05:11 PM    <DIR>          -
07/08/2004  05:12 PM                445 TestApp.class
07/08/2004  04:09 PM                139 TestApp.java
                2 File(s)          584 bytes
                2 Dir(s)    21,394,849,792 bytes free

C:\java\exercises\ch01>_

```

A review of DOS commands and keystrokes

Command	Description
dir	Displays a directory listing.
dir /p	Displays a directory listing that pauses if the listing is too long to fit on one screen.
cd \	Changes the current directory to the root directory for the current drive.
cd ..	Changes the current directory to the parent directory.
cd <i>directory name</i>	Changes the current directory to the subdirectory with the specified name.
<i>letter:</i>	Changes the current drive to the drive specified by the letter. For example, entering d: changes the current drive to the d drive.

How to start the DOSKey program

- To start the DOSKey program, enter “doskey /insert” at the command prompt.
- To automatically start the DOSKey program for all future sessions, add the “doskey/insert” statement after the “path” statement in the autoexec.bat file. For help on editing the autoexec.bat file, see the second procedure in figure 1-6.

How to use the DOSKey program

Key	Description
Up or down arrow	Cycles through previous DOS commands in the current session.
Left or right arrow	Moves cursor to allow normal editing of the text on the command prompt.

Figure 1-14 Essential DOS skills for working with Java

How to use the documentation for the J2SE API

When you write Java code, you'll often need to look up information about the *Application Program Interface (API)* for the J2SE. The API provides the Java classes that you can use as you build your Java applications. Since Sun provides HTML-based documentation for the J2SE API, you can browse this documentation with any web browser to get the detailed information you need about any Java class.

How to install the API documentation

Although you can use your browser to view the documentation for the J2SE API from the Java web site, you will want to install this documentation on your hard drive instead. That way, you can browse the documentation more quickly, and you can browse it even if you aren't connected to the Internet.

To download and install this documentation, you can use the procedure shown in figure 1-15. Since the documentation comes in a compressed format called a *zip file*, you need to use an unzip tool to extract the HTML pages from the zip file. When you use this tool, it creates a docs directory that contains many files and subdirectories. Although you can store the docs directory anywhere you like, it's common to store it as a subdirectory of the JDK directory.

If your operating system doesn't include a tool for working with zip files, you can download one from the web. For example, WinZip is a popular program for working with zip files. You can download a free evaluation copy from www.winzip.com.

How to download the API documentation

1. Go to the Java web site (java.sun.com).
2. Go to the download page for the version of the JDK that you're using (see figure 1-4) and find the hyperlink for the API documentation download.
3. Follow the instructions for the download and save the zip file to your hard drive.

How to install the API documentation

- Extract all files in the zip file to your hard drive. This will create a directory named docs that contains several files and subdirectories.
- Although it's common to store the API documentation in the JDK directory (usually C:\Program Files\Java\jdk1.5.0 for version 5), you can store it anywhere you like.

Description

- The *Application Programming Interface*, or *API*, provides all the classes that are included as part of the JDK. To learn about these classes, you can use the API documentation.
- You can view the API documentation from the Java web site, or you can download and install it on your system.

Notes

- You can click the “How long will it take” link to determine how long it will take to download the zip file based on its size and the speed of your modem. For a 56K modem, for example, it will take about 2 hours and 16 minutes to download 46MB.
- If you're using an older operating system that doesn't automatically work with zip files, you may need to use a tool such as WinZip to extract the files from the zip file. To download a free evaluation copy of WinZip, go to www.winzip.com.

How to navigate the API documentation

Figure 1-16 shows how to navigate the documentation for the J2SE API. To start, you point your web browser to the index page that's stored in the docs\api directory. To do that for the first time, use the Windows Explorer to navigate to the docs\api directory and double-click on the index.html file, or enter the location of this page into the address area of your web browser:

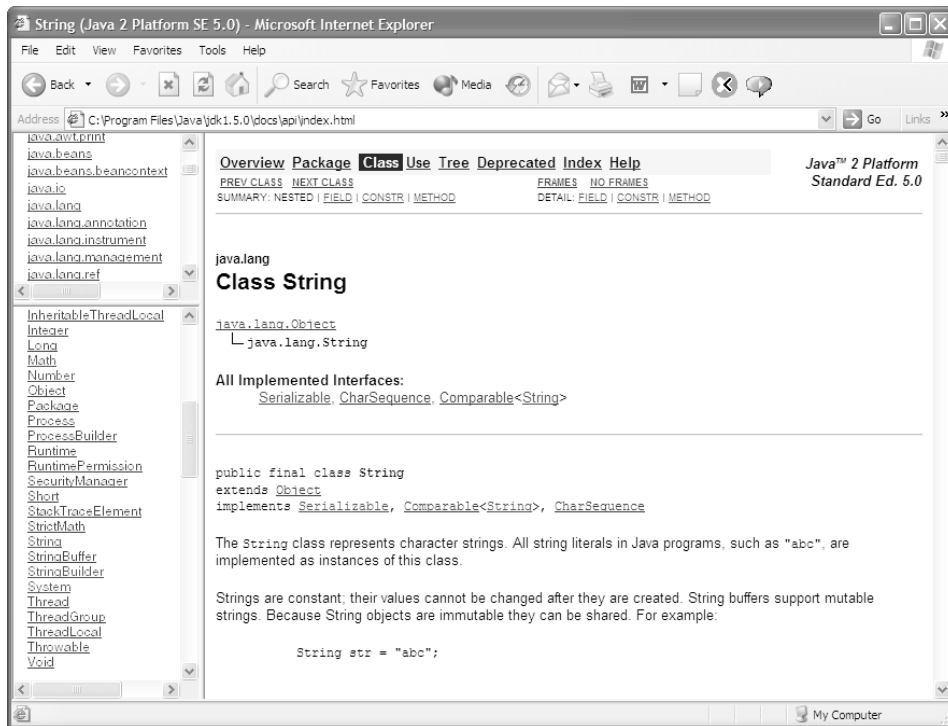
```
C:\Program Files\Java\jdk1.5.0\docs\api\index.html
```

Since you'll need to access this page often as you learn about Java, you should use your web browser's favorites or bookmark feature to mark this page. Then, the next time you need to use this page, you can go right to it.

To browse the Java documentation, you need to know that all the classes in the Java API are organized into *packages*. As a result, the index page for the documentation provides a way to navigate through packages and classes. To start, you can select a package from the upper left frame. When you do, the lower left frame displays all the classes in that package. Then, you can select a class to display information about that class in the right frame.

As you progress through this book, you'll learn about many different packages and classes, and you'll learn much more about how packages and classes work. Along the way, you can always use the documentation for the Java API to help clarify the discussion or further your knowledge.

The index for the documentation



Description

- If you've installed the API documentation on your hard drive, you can display an index like the one shown above by using your web browser to go to the index.html file in the docs\api directory. If you haven't installed the documentation, you can browse through it on the Java web site.
- Related classes in the Java API are organized into *packages*, which are listed in the upper left frame of the documentation page. When you select a package, all the classes for that package are listed in the lower left frame.
- You can select a class from the lower left frame to display the documentation for that class in the right frame. You can also select the name of the package at the top of the lower left frame to display information about the package and the classes it contains.
- Once you display the documentation for a class, you can scroll through it or click on a hyperlink to get more information.
- The documentation for a class usually provides a wide range of information, including a summary of all of its methods. You'll learn more about methods and how they're used throughout this book.
- To make it easier to access the API documentation, you should bookmark the index page. To do that with the Internet Explorer, select the Add To Favorites command from the Favorites menu and accept the default name for the page or assign your own name to it. Then, you can redisplay this page later by selecting it from the Favorites menu.

Figure 1-16 How to navigate the API documentation

Introduction to Java IDEs

Many *Integrated Development Environments (IDEs)* are available for working with Java. A typical IDE provides not only a text editor, but also tools for compiling, running, and debugging code as well as tools for designing user interfaces. To illustrate the range of Java IDEs available, I'll present an overview of two IDEs. Keep in mind, however, that a search of the web will show that dozens of other IDEs are also available for working with Java.

Overview of Eclipse

Eclipse is a popular IDE that's available for free from www.eclipse.org. The first screen in figure 1-17 shows the window you use to edit code using Eclipse. As you work with the code editor, Eclipse can help you complete your code and notify you of potential compile-time errors. In addition, Eclipse will automatically compile your programs for you when you run them. And when you run a program that prints text to the console, Eclipse displays that text in the Console window.

The second screen in this figure shows the windows that are displayed when you use Eclipse to debug a program. Here, you can use the code editor to set breakpoints, and you can use the buttons in the toolbar to step through your code. As you do, you can use the top-right window to view the values of all of the active variables. If you've used any modern debugging tool, you should understand how this works.

Eclipse also provides many other advanced features that aren't available from a simple tool like TextPad. For example, you can use Eclipse with JUnit, which helps you test each part of an application, and also with Ant, which helps you document, package, and deploy your applications.

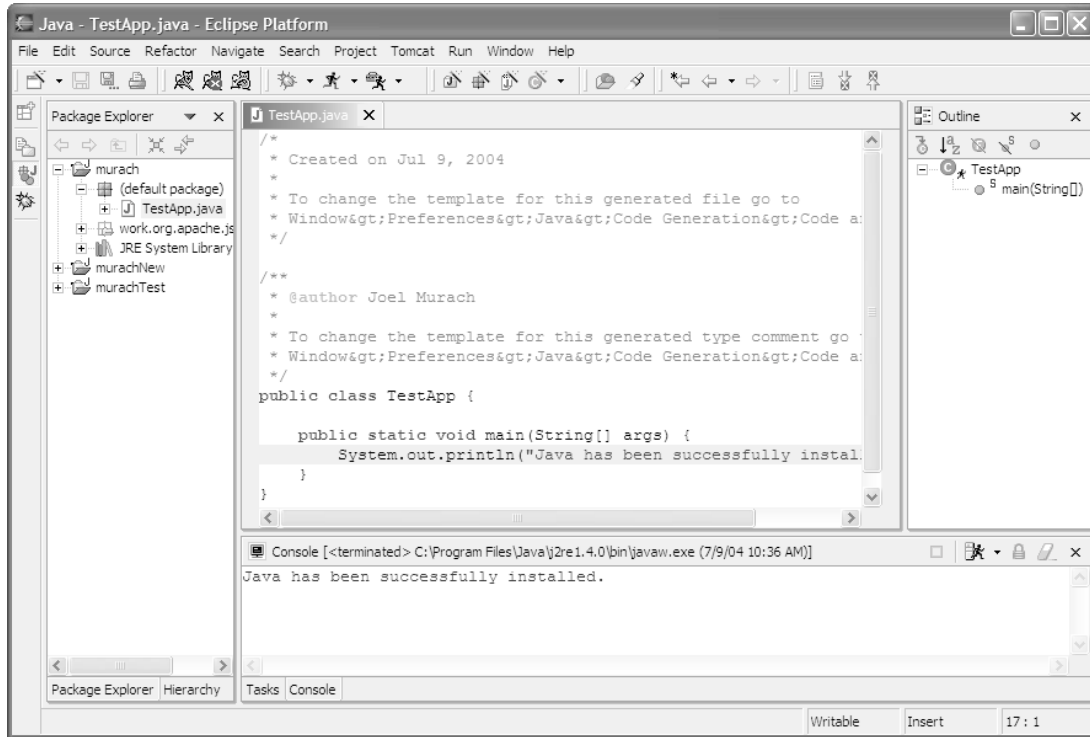
When we recommend using a professional IDE

Although a professional IDE like Eclipse can make it easier to develop Java applications, we don't recommend that you use one when you're learning Java for two reasons. First, this type of IDE will often generate code for you, which won't help you learn how to write the code for yourself. Second, an IDE like this is a complex tool with operational details that are themselves difficult to learn.

Once you've got a solid grasp on the use of Java, though, a professional IDE can make working with Java easier. In particular, it can make it easier to develop graphical user interfaces and to debug your code. As a result, we recommend that you start using one of these tools after you've mastered the core Java skills.

Of course, we realize that this choice is a personal one. Some talented programmers prefer the simplicity and speed of a text editor to the complexity and sluggishness of a professional IDE. Conversely, some students prefer learning with an IDE despite the additional learning curve that it presents. With or without an IDE, though, this book will help you master the core Java skills.

The Eclipse code editor



The Eclipse debugger

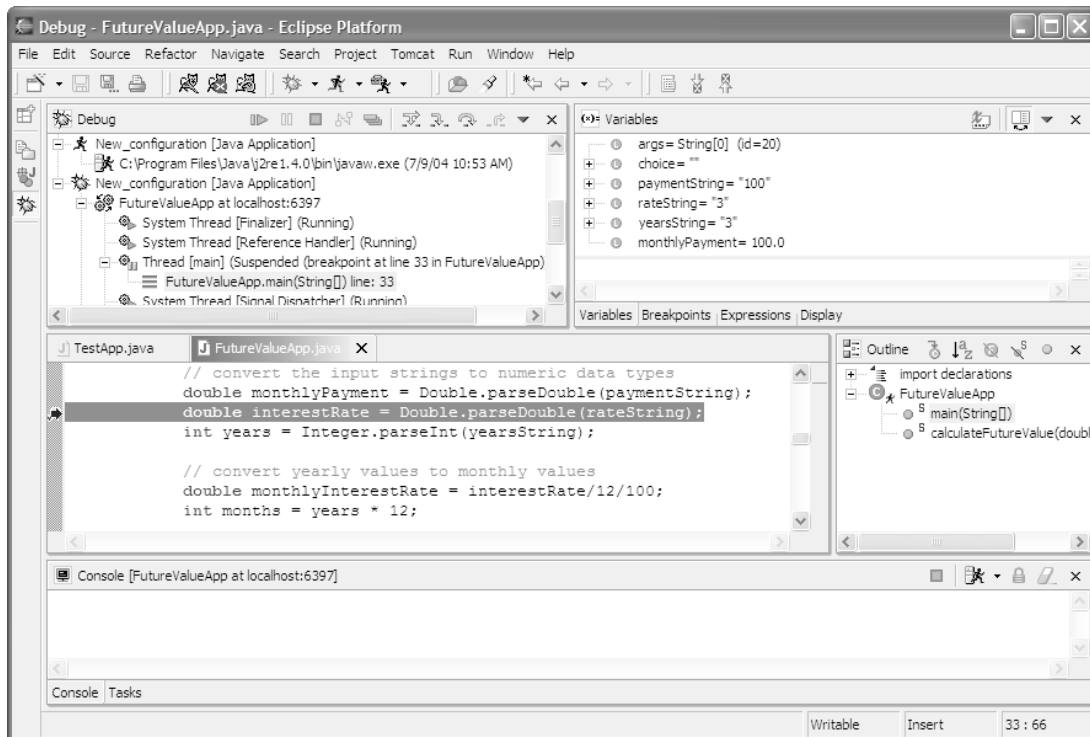


Figure 1-17 Two views of a professional IDE named Eclipse

Overview of BlueJ

BlueJ is a free IDE that was developed as part of a university research project to teach object-oriented programming to first year Java students. Like Eclipse, BlueJ is available for free, and you can download it from www.bluej.org. Although BlueJ doesn't provide as many features as a professional IDE like Eclipse, it provides some of the advantages of an IDE without some of the drawbacks. For example, BlueJ includes a debugger that's easy for students to learn. As a result, if you're new to object-oriented programming, you might want to try using BlueJ.

The first screen in figure 1-18 shows how BlueJ maintains a diagram of all classes in a project. This diagram can help students visualize how the classes in a project work together. In this diagram, the arrows with closed heads show that the Book and Software classes inherit the Product class. In contrast, the other arrows show that the ProductDB class uses the Product, Book, and Software classes and that the ProductApp class uses the Product and ProductDB classes.

Since the BlueJ diagram is interactive, it also encourages learning. For instance, you can right-click on any class to compile the code for a class, to edit or view the code for the class, to experiment with the class, and to explore the class. While this provides little benefit to an experienced programmer, it can help illustrate object-oriented concepts to beginners.

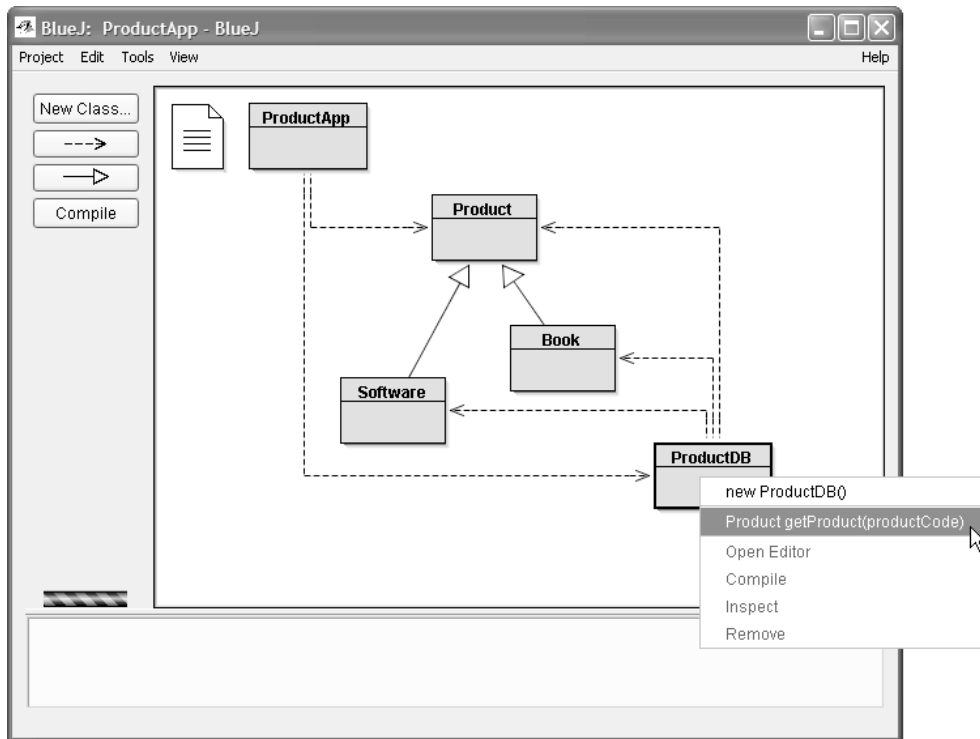
The second screen shows the BlueJ code editor. This editor provides all of the features that you would expect from a text editor that's designed for working with Java. In fact, it provides many of the same features as TextPad, including a Compile command that allows you to compile code directly from the code editor.

For more information about using BlueJ with this book, please visit our web site at:

www.murach.com/bluej

Here, you'll find a PDF file that shows how to download and install BlueJ. This file also shows how to use BlueJ to work with the applications presented in this book.

A BlueJ project



The BlueJ code editor

```

1 import java.util.Scanner;
2
3 public class ProductApp
4 {
5     public static void main(String args[])
6     {
7         // welcome the user to the program
8         System.out.println("Welcome to the Product Selector");
9         System.out.println();
10
11        // perform 1 or more selections
12        String choice = "y";
13        while (choice.equalsIgnoreCase("y"))
14        {
15            // get the input scanner
16            Scanner sc = new Scanner(System.in);
17            System.out.print("Enter product code: ");
18            String productCode = sc.nextLine(); // read entire line
19
20            // use the ProductDB class to get the Product object
21            Product p = ProductDB.getProduct(productCode);
22
23            // display the output
24        }
25    }
26 }
  
```

The code editor window includes a menu bar (Class, Edit, Tools, Options) and a toolbar with buttons for Compile, Undo, Cut, Copy, Paste, Find..., Find Next, Close, and Implementation. A status bar at the bottom shows 'File saved' and a 'saved' button.

Figure 1-18 Two views of a student IDE named BlueJ

Perspective

In this chapter, you learned how to install and configure the JDK for developing Java programs. You learned how to use TextPad to enter, edit, compile, and run a program. You learned how to use the command prompt to compile and run programs. And you learned how to install and view the API documentation for the JDK. With that as background, you're ready to learn how to write your own Java applications.

Summary

- You use the *Java Development Kit (JDK)* to develop Java programs. This used to be called the *Software Development Kit (SDK)* for Java. Versions 1.2 through 5.0 of the JDK run under the *Java 2 Platform, Standard Edition (J2SE)* so they are sometimes referred to as *Java 2*.
- You can use the J2SE platform to create *applications* and a special type of Internet-based application known as an *applet*. In addition, you can use the *Java 2 Platform, Enterprise Edition (J2EE)* to create server-side applications using *servlets* and *JavaServer Pages (JSPs)*.
- The *Java compiler* translates *source code* into a *platform-independent* format known as *Java bytecodes*. Then, the *Java interpreter*, or *Java Runtime Environment (JRE)*, translates the bytecodes into instructions that can be run by a specific operating system. A Java interpreter is an implementation of a *Java virtual machine (JVM)*.
- When you use the JDK with Windows, you should add the bin directory (usually C:\Program Files\Java\jdk1.5.0\bin) to the *command path* and you should add the current directory to the *classpath*.
- A *text editor* that's designed for working with Java provides features that make it easier to enter, edit, and save Java code.
- Some text editors such as TextPad include commands for compiling and running Java applications. You can also use the *command prompt* to enter the commands for compiling and running an application.
- When you compile a program, you may get *compile-time errors*. When you run a program, you may get *runtime errors*.
- To compile code from the command prompt, you use the *javac command* to start the Java compiler. To run an application from the command prompt, you use the *java command* to start the Java interpreter.
- You can get detailed information about any class in the J2SE by using a web browser to browse the HTML-based documentation for its *Application Programming Interface (API)*.
- Once you've mastered the basics of Java, an *Integrated Development Environment (IDE)* can make working with Java easier.

Before you do the exercises for this chapter

Before you do any of the exercises in this book, you need to download the folders and files for this book from our web site (www.murach.com) and install them on your system. For complete instructions, please refer to appendix A. Then, you should follow the procedures shown in this chapter to install and configure the JDK (figure 1-4 through 1-7), TextPad (figure 1-8) or an equivalent text editor, and the documentation for the Java API (figure 1-15).

Exercise 1-1 Use TextPad to develop an application

This exercise will guide you through the process of using TextPad to enter, save, compile, and run a simple application.

Enter and save the source code

1. Start TextPad by clicking on the Start button and selecting Programs or All Programs→TextPad.
2. Enter this code (type carefully and use the same capitalization):

```
public class TestApp
{
    public static void main(String[] args)
    {
        System.out.println(
            "This Java application has run successfully");
    }
}
```

3. Use the Save command in the File menu to display the Save As dialog box. Next, navigate to the `c:\java1.5\ch01` directory and enter `TestApp` in the File name box. If necessary, select the Java option from the Save as Type combo box. Then, click on the Save button to save the file.

Compile the source code and run the application

4. Press `Ctrl+1` to compile the source code. If you get an error message, read the error message, edit the text file, save your changes, and compile the application again. Repeat this process until you get a clean compile (the code is displayed with no error messages).
5. Press `Ctrl+2` to run the application. This application should display a console window that says “This Java application has run successfully” followed by a line that reads “Press any key to continue...”.
6. Press any key. This should close the console window. If it doesn’t, click on the Close button in the upper right corner of the window to close it.

Introduce and correct a compile-time error

7. In the TextPad window, delete the semicolon at the end of the `System.out.println` statement. Then, press `Ctrl+1` to compile the source code. TextPad should display an error message that indicates that the semicolon is missing in the Command Results window.

8. In the Document Selector pane, click on the TestApp.java file to switch back to the source code, and press Ctrl+F6 twice to toggle back and forth between the Command Result window and the source code. Then, select View→Line Numbers to display the line numbers for the source code lines.
9. Correct the error and compile the file again (this automatically saves your changes). This time the file should compile cleanly, so you can run it again and make sure that it works correctly.
10. Select Configure→Preferences, click on View, and check Line Numbers. That will add line numbers to the source statements in all your applications. If you want to look through the other options and set any of them, do that now. When you're done, close the file and exit TextPad.

Exercise 1-2 Use any Java development tool to develop an application

If you aren't going to use TextPad to develop your Java programs, you can try whatever tools you are going to use with this generic exercise.

Use any text editor to enter and save the source code

1. Start the text editor and enter this code (type carefully and use the same capitalization):

```
public class TestApp
{
    public static void main(String[] args)
    {
        System.out.println(
            "This Java application has run successfully");
    }
}
```

2. Save this code in the c:\java1.5\ch01 directory in a file named "TestApp.java".

Compile the source code and run the application

3. Compile the source code. If you're using a text editor that has a compile command, use this command. Otherwise, use your command prompt to compile the source code. To do that, start your command prompt and use the cd command to change to the c:\java1.5\ch01 directory. Then, enter the javac command like this (make sure to use the same capitalization):

```
javac TestApp.java
```

4. Run the application. If you're using a text editor that has a run or execute command, use this command. Otherwise, use your command prompt to run the application. To do that, enter the java command like this (make sure to use the same capitalization):

```
java TestApp
```

5. When you enter this command, the application should print "This Java application has run successfully" to the console window.

Exercise 1-3 Use the command prompt to run any compiled application

This exercise shows how to use the command prompt to run any Java application.

1. Open the command prompt window. Then, change the current directory to `c:\java1.5\ch01`.
2. Use the `java` command to run the `LoanCalculatorApp` application. This application calculates the monthly payment for a loan amount at the interest rate and number of years that you specify. This shows how the JRE can run any application whether or not it has been compiled on that machine. When you're done, close the application to return to the command prompt.

Exercise 1-4 Navigate the API documentation

This exercise will give you some practice using the API documentation to look up information about a class.

1. Start a web browser and navigate to the index page that contains the API documentation for the JDK (usually `C:\Program Files\Java\jdk1.5.0\docs\api\index.htm`). This page should look like the one shown in figure 1-16.
2. Bookmark this page so you can easily access it later. To do that with the Internet Explorer, select the `Add To Favorites` item from the `Favorites` menu. Then, close your web browser.
3. Start your web browser again and use the bookmark to return to the API documentation for the JDK. To do that with the Internet Explorer, select the `Java 2 Platform SE` item from the `Favorites` menu.
4. Select the `java.lang` package in the upper left frame and notice that the links in the lower left frame change. Select the `System` class from this frame to display information about it in the right frame.
5. Scroll down to the `Field Summary` area in the right frame and click on the out link for the standard output stream. When you do, an HTML page that gives some information about how to use the standard output stream will be displayed. In the next chapter, you'll learn more about using the `out` field of the `System` class to print data to the console.
6. Continue experimenting with the documentation until you're comfortable with how it works. Then, close the browser.