Thanks for downloading this chapter from Murach’s Beginning Java with Eclipse. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its “how-to” headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our website. 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 other books on related topics.

Thanks for your interest in our books!
What programmers have said about our Java books

“If you’re a Java virgin like I was before reading this book, then you’re going to love it. I had a development environment up & running within half an hour and was dabbling with code 15 minutes after that!”

Andy Bonner, vbcity.com

“I bought your Java book a week ago and I am already writing useful programs, not ‘toys!’”

Richard Cooper, Programmer

“One of the things I really like is that this book uses NetBeans as the IDE to teach Java to the next generation of programmers. A lot of Java books focus on the language itself, [but] any professional programmer will tell you that an IDE is absolutely essential in making you more productive.”

John Yeary, Java Evangelist

“Terrific – Fantastic – Superlative! WELL worth several times the purchase price.”

Posted at Amazon.com

“I love your Java book. It cuts right to the essential information, providing the perfect balance between too many details and too little information. Example apps are incredible. Keep up the good work.”

Steve, Programmer, Denver, Colorado

“Another thing I like is the exercises at the end of each chapter. They’re a great way to reinforce the main points and force you to get your hands dirty.”

Hien Luu, SD Forum/Java SIG

“The absolute best teaching text for Java among the gazillions of self-hyped tutorials and skills books on the market today.”

Posted at Amazon.com
Get started right

This section gets you started quickly with Java programming. First, chapter 1 introduces you to some concepts and terms that apply to Java development. In addition, it shows you how to use the Eclipse IDE to import and run existing projects.

After that, chapter 2 shows you how to use Eclipse to start writing Java code. Chapter 3 shows how to develop two procedural applications by using classes and methods that are available from the Java API. Chapter 4 shows how to convert a procedural application to an object-oriented application by creating your own classes and methods. Chapter 5 shows how to structure an object-oriented application. And chapter 6 shows how to thoroughly test and debug this application. When you complete these chapters, you’ll be able to write, test, and debug simple object-oriented applications of your own.
An introduction to Java programming

This chapter starts by presenting some background information about Java. This information isn’t essential to developing Java applications, so you can skim it if you want. However, it does show how Java works and how it compares to other languages.

This chapter finishes by showing how to use the Eclipse IDE to work with an existing project. This gives you some hands-on experience using Eclipse to work with projects such as the projects for this book that you can download from our website.
Section 1  Get started right

An overview of Java

In 1996, Sun Microsystems released a new programming language called Java. Today, Java is owned by Oracle and is one of the most widely used programming languages in the world.

Java timeline

Figure 1-1 starts by describing all major releases of Java starting with version 1.0 and ending with version 1.8. Throughout Java’s history, the terms Java Development Kit (JDK) and Software Development Kit (SDK) have been used to describe the Java toolkit. In this book, we’ll use the term JDK since it’s the most current and commonly used term.

In addition, different numbering schemes have been used to indicate the version of Java. For example, Java SE 8 or Java 1.8 both refer to the eighth major version of Java. Similarly, Java SE 7 and Java 1.7 both refer to the seventh major version of Java. The documentation for the Java API uses the 1.x style of numbering. As a result, you should be familiar with it. However, it’s also common to only use a single number such as Java 6.

This book shows how to use Java 8. However, Java is backwards compatible, so it should also work for future versions of Java. In addition, most of the skills described in this book have been a part of Java since its earliest versions. As a result, those skills should work with earlier versions of Java.

Java editions

This figure also describes the three most common editions of Java. To start, there is the Standard Edition, which is known as Java SE. It’s designed for general purpose use on desktop computers and servers, and it’s the edition that you’ll learn how to work with in this book. For example, you can use Java SE to create a desktop application like the one presented at the end of chapter 22.

The Enterprise Edition is known as Java EE. It’s designed to develop distributed applications that run on an intranet or the Internet. You can use Java EE to create web applications.

The Micro Edition is known as Java ME. It’s designed to run on devices that have limited resources such as mobile devices, TV set-top boxes, printers, smart cards, hotel room key cards, and so on.

With some older versions of Java, Java SE was known as J2SE (Java 2 Platform, Standard Edition). Similarly, Java EE was known as J2EE (Java 2 Platform, Enterprise Edition). If you are searching for information about Java on the Internet, you may come across these terms. However, they aren’t commonly used anymore.
Java timeline

<table>
<thead>
<tr>
<th>Year</th>
<th>Month</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>January</td>
<td>JDK 1.0 released.</td>
</tr>
<tr>
<td>1997</td>
<td>February</td>
<td>JDK 1.1 released.</td>
</tr>
<tr>
<td>1998</td>
<td>December</td>
<td>SDK 1.2 released.</td>
</tr>
<tr>
<td></td>
<td>December</td>
<td>Java 2 Platform, Enterprise Edition (J2EE) released.</td>
</tr>
<tr>
<td>2000</td>
<td>May</td>
<td>J2SE with SDK 1.3.</td>
</tr>
<tr>
<td>2002</td>
<td>February</td>
<td>J2SE with SDK 1.4.</td>
</tr>
<tr>
<td>2004</td>
<td>September</td>
<td>J2SE 5.0 with JDK 1.5.</td>
</tr>
<tr>
<td>2006</td>
<td>December</td>
<td>Java SE 6 with JDK 1.6.</td>
</tr>
<tr>
<td>2010</td>
<td>April</td>
<td>Oracle buys Sun.</td>
</tr>
<tr>
<td>2011</td>
<td>July</td>
<td>Java SE 7 with JDK 1.7.</td>
</tr>
<tr>
<td>2014</td>
<td>March</td>
<td>Java SE 8 with JDK 1.8.</td>
</tr>
</tbody>
</table>

Java editions

<table>
<thead>
<tr>
<th>Platform</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java SE (Standard Edition)</td>
<td>For general purpose use on desktop computers and servers. Some early versions were called J2SE (Java 2 Platform, Standard Edition).</td>
</tr>
<tr>
<td>Java EE (Enterprise Edition)</td>
<td>For developing distributed applications that run on an intranet or the Internet. Some early versions were called J2EE (Java 2 Platform, Enterprise Edition).</td>
</tr>
<tr>
<td>Java ME (Micro Edition)</td>
<td>For devices with limited resources such as mobile devices, TV set-top boxes, printers, and smart cards.</td>
</tr>
</tbody>
</table>

Description

- The Java Development Kit (JDK) includes a compiler, a runtime environment, and other tools that you can use to develop Java applications. Some early versions were called the Software Development Kit (SDK).
How Java compares to C++ and C# 

Figure 1-2 compares Java to C++ and C#. As you can see, Java has some similarities and some differences with these languages.

When Sun’s developers created Java, they tried to keep the syntax for Java similar to the syntax for C++. That way, 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 without needing to be compiled for each platform. In contrast, C++ needs to be compiled for each platform.

Java was also designed to automatically handle many operations involving the allocation and de-allocation of memory. This is a key reason why it’s easier to develop programs and write bug-free code with Java than with C++.

To provide these features, early versions of Java sacrificed some speed (or performance) when compared to C++. However, improvements in later versions of Java have greatly improved Java’s speed. Now, Java runs faster than C++ in some contexts, and its performance is adequate in most contexts.

When Microsoft’s developers created C#, they used many of the best ideas of Java. Like Java, C# uses a syntax that’s similar to C++. In addition, C# automatically handles memory operations.

C# can run on any platform that has a runtime environment for it. However, Windows is the only operating system that fully supports runtime environment for C#. As a result, C# is primarily used for developing applications that only need to run on Windows.

Java runs faster than C# in most contexts. However, the performance of C# is adequate in most contexts.
Operating systems that support Java

- Windows
- Mac OS X
- Linux
- Most versions of UNIX
- Most other modern operating systems

A note about Android

- The Android operating system doesn’t support Java in the same way as most operating systems. However, you can use most features of Java 6 and 7 to write the code for Android apps.

Java compared to C++

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Java syntax is similar to C++ syntax.</td>
</tr>
<tr>
<td>Platforms</td>
<td>Compiled Java code can run on any platform that has a Java runtime environment. C++ code must be compiled once for each type of system that it is going to be run on.</td>
</tr>
<tr>
<td>Speed</td>
<td>C++ runs faster than Java in some contexts, but Java runs faster in other contexts.</td>
</tr>
<tr>
<td>Memory</td>
<td>Java handles most memory operations automatically, but C++ programmers must write code that manages memory.</td>
</tr>
</tbody>
</table>

Java compared to C#

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Java syntax is similar to C# syntax.</td>
</tr>
<tr>
<td>Platforms</td>
<td>Like Java, compiled C# code can run on any platform that has a runtime environment for it.</td>
</tr>
<tr>
<td>Speed</td>
<td>Java runs faster than C# in most contexts.</td>
</tr>
<tr>
<td>Memory</td>
<td>Like Java, C# handles most memory operations automatically.</td>
</tr>
</tbody>
</table>
Types of Java applications

You can use Java to write almost any type of application (also known as an app or a program). In this book, you’ll learn how to develop desktop applications. However, you can also use Java to develop web applications and mobile apps.

Two types of desktop applications

Figure 1-3 shows two types of desktop applications that you can create with Java. This type of application runs directly on your computer.

The easiest type of desktop application to create is known as a console application. This type of application runs in the console, or command prompt, that’s available from your operating system. The console provides an easy way to get input from the user and to display output to the user. In this figure, for example, the user has entered three values in the console application, and the application has performed a calculation and displayed the result. When you’re learning Java, it’s common to work with console applications until you have a solid understanding of the Java language.

Once you have a solid understanding of the Java language, you can create a desktop application that uses a graphical user interface (GUI). In this figure, for example, the GUI application performs the same tasks as the console application. In other words, it gets the same input from the user, performs the same calculation, and displays the same result. However, the GUI application is more user-friendly and intuitive.

Since developing the GUI for an application requires some significant Java coding skills, this book doesn’t present a GUI application until the last two chapters of this book. Until then, this book uses console applications to teach the basics of Java.
A console application

A GUI application

Description

- A console application uses the console to interact with the user.
- A GUI application uses a graphical user interface to interact with the user.

Figure 1-3 Two types of desktop applications
Two types of web applications

Figure 1-4 shows two types of web applications that you can develop with Java. In the early days of Java, which were also the early days of the Internet, one of the most exciting features of Java was that you could use it to create a special type of web-based application known as an applet like the one shown in this figure. This applet works like the GUI application presented in the previous figure. However, unlike that GUI application, an applet could be stored in an HTML page and run inside a Java-enabled browser. As a result, it was once possible to distribute applets via the Internet or an intranet.

In recent years, with tightening security restrictions, fewer and fewer browsers support applets, even if you install the plug-in that was designed to allow applets to run in browsers. That’s why the applet in this figure is shown in the Applet Viewer, which comes as part of the JDK, instead of being shown in a web browser. Due to these security restrictions, applets are effectively obsolete. They are still included as part of Java SE for backwards compatibility, but they aren’t used much anymore, and we don’t cover them in this book.

In addition, since applets run within a browser on the client, they were never an ideal way to work with resources that run on the server, such as enterprise databases. A better way to provide access to enterprise databases is to use Java EE to create web applications that run on the server. These applications are often 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 like the applet. However, the servlet runs on the server, not the client.

To start, the web browser on the client sends a request to the servlet that’s running on the server. This request includes the user input. When the servlet receives this request, it performs the calculation and returns the result to the browser, typically in the form of an HTML page.

In this figure, the servlet doesn’t access a database. However, it’s common for servlets to work with a database. 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, CSS, and JavaScript 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.

To make it easy to store the results of a servlet within an HTML page, the Java EE specification provides for JavaServer Pages (JSPs). As a result, it’s common to use JSPs with servlets. Although servlets and JSPs aren’t presented in this book, they are present in a companion book, Murach’s Java Servlets and JSP. For more information about this book, please visit our website at www.murach.com.
An applet

An applet is a type of Java application that runs within a web browser. In the past, it was possible to run applets in most web browsers. Today, fewer and fewer web browsers support applets. As a result, they are effectively obsolete.

A servlet

A servlet is a type of Java application that runs on a web server. A servlet accepts requests from clients and returns responses to them. Typically, the clients are web browsers.

Description

- An applet is a type of Java application that runs within a web browser.
- In the past, it was possible to run applets in most web browsers. Today, fewer and fewer web browsers support applets. As a result, they are effectively obsolete.
- A servlet is a type of Java application that runs on a web server.
- A servlet accepts requests from clients and returns responses to them. Typically, the clients are web browsers.
Mobile apps

You can also use Java to develop mobile apps, which are applications that run on a mobile device such as a smartphone or tablet. In particular, Java is commonly used to write the code for apps that run on Android devices. For example, figure 1-5 shows a mobile app that was developed with Java. This app performs the same task as the applications presented in the previous figures.

An app works much like a traditional application. However, the user interface has to be modified so that it’s appropriate for a mobile device. In this figure, for example, the user interface has been modified to work with a touch-screen device that has a small screen and no keyboard. As a result, the user can use the keypad that’s displayed onscreen to enter numbers and can press the Done button on this keypad to perform the calculation.

The Android operating system includes its own virtual machine that supports a subset of Java, including most features of Java 6 and 7. As a result, if you use Java to develop Android apps, you can’t use all of the features of Java, especially the newest ones. That’s because the Android virtual machine is not a Java virtual machine. In other words, the Android virtual machine can’t run compiled Java code, and a Java virtual machine can’t run compiled Android code. Still, you can use most features of Java to write code for Android apps, and it’s easy enough to compile that code so the Android virtual machine can run it.
A mobile app

Description

- A mobile app uses a mobile device such as a smartphone or tablet to interface with the user.
- The Android operating system supports a subset of Java, including most features of Java 6 and 7.
An introduction to Java development

At this point, you’re ready to see the source code for an application. You’re ready to learn how Java compiles and interprets this code. And you’re ready to be introduced to some of the IDEs that you can use to develop this type of code.

The code for a console application

When you develop a Java application, you start by entering and editing the source code. To give you an idea of how the source code for a Java application works, figure 1-6 presents the code for the console version of the Future Value application shown in figure 1-3.

If you have experience with other programming languages, you may be able to understand much of this code already. If not, don’t worry! You’ll learn how all of this code works in the next few chapters. For now, here’s a brief explanation of this code.

Most of the code for this application is stored in a package named murach.fv that corresponds with the murach/fv folder. Within this package, there is a class named Main that corresponds with a file named Main.java. This class begins with an opening brace ( { ) and ends with a closing brace ( } ).

Within this class, the code defines one method named main. This method also begins with an opening brace ( { ) and ends with a closing brace ( } ). These braces are indented to clearly show that they are contained within the class.

This is a special type of method known as the main method for the application. The code within this method is executed automatically when you run the application. In this case, the code prompts the user for input, gets input from the user, converts the input to appropriate data types, calculates the future value based on the user input, and displays the results to the user.

To do that, this code uses two while loops. Each loop begins with an opening brace ( { ) and ends with a closing brace ( } ). The code within these braces is also indented. This shows that the first loop is coded within the main method and the second loop is coded within the first loop.

The first while loop allows the user to continue making calculations by entering “y” or to exit by entering “n”. The second while loop is executed once for each month and contains the statements that calculate the future value. To do that, these statements add the monthly investment and the monthly interest amount to the current future value.
The code for a console application

```java
package murach.fv;

import java.text.NumberFormat;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        // displayLine a welcome message
        System.out.println("Welcome to the Future Value Calculator");
        System.out.println();

        Scanner sc = new Scanner(System.in);
        String choice = "y";
        while (choice.equalsIgnoreCase("y")) {

            // get input from user
            System.out.print("Enter monthly investment:   ");
            double monthlyInvestment = Double.parseDouble(sc.nextLine());

            System.out.print("Enter yearly interest rate: ");
            double yearlyInterestRate = Double.parseDouble(sc.nextLine());

            System.out.print("Enter number of years:      ");
            int years = Integer.parseInt(sc.nextLine());

            // convert yearly values to monthly values
            double monthlyInterestRate = yearlyInterestRate / 12 / 100;
            int months = years * 12;

            // calculate the future value
            double futureValue = 0;
            int i = 1;
            while (i <= months) {
                futureValue = futureValue + monthlyInvestment;
                double monthlyInterestAmount =
                        futureValue * monthlyInterestRate;
                futureValue = futureValue + monthlyInterestAmount;
                i = i + 1;
            }

            // format and display the result
            System.out.println("Future value:               
                        " +
                        NumberFormat.getCurrencyInstance().format(futureValue));
            System.out.println();

            // see if the user wants to continue
            System.out.print("Continue? (y/n): ");
            choice = sc.nextLine();
            System.out.println();
        }

        sc.close();
        System.out.println("Bye!");
    }
}
```

Figure 1-6 The code for a console application
How Java compiles and interprets code

Once the source code has been written, you use the Java compiler to compile the source code into a format known as Java bytecodes as shown in figure 1-7. At this point, the bytecodes can be run on any platform that has a Java runtime environment (JRE) installed on it. A JRE includes all of the software needed to run bytecodes. Among other things, this includes an implementation of a Java virtual machine (JVM). This JVM includes a Java interpreter to translate the Java bytecodes into native code that can be understood by the underlying operating system.

Most modern implementations of the JVM have replaced the Java interpreter with a just-in-time compiler (JIT compiler). A JIT compiler is similar to an interpreter in some ways, but it actually compiles the most used parts of the Java bytecodes into native code and stores this code in a cache. This improves performance significantly.

Since JREs 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 platform.
Description

- When you develop a Java application, you typically use a code editor to work with the *source code* for the application. Files that contain source code 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.

- A *Java virtual machine (JVM)* includes a *Java interpreter* that executes Java bytecodes. Most modern implementations of the JVM have replaced the Java interpreter with a *just-in-time compiler (JIT compiler)*. A JIT compiler is similar to an interpreter in some ways, but it improves performance significantly.

- A *Java runtime environment (JRE)* has all of the components necessary to run bytecodes including a JVM. Since JREs are available for most operating systems, Java bytecodes can be run on most operating systems.
Introduction to IDEs for Java development

To develop Java applications, you typically use an Integrated Development Environment (IDE). Although you can use a simple text editor with command-line tools, an IDE provides features that can make developing Java applications considerably easier. Figure 1-8 describes some of the features of the most popular IDEs.

All of the IDEs listed in this figure are either free or have a free edition. That makes them particularly attractive to students as well as programmers who are learning on their own. Most of these IDEs also run on all modern operating systems.

The first two IDEs listed in this figure, Eclipse and NetBeans, are two of the most popular Java IDEs. Both of these IDEs provide all of the features listed in this figure. For example, both of these IDEs help you complete your code and notify you of potential compile-time errors. They both automatically compile your code before you run it. And they both include a debugger that lets you perform standard debugging functions like setting breakpoints, stepping through code, and viewing the values of variables.

The third IDE listed in this figure, IntelliJ IDEA, isn’t as popular as Eclipse and NetBeans. However, we have included it here to give you an idea of the range of IDE choices that are available for Java. In addition, other Java IDEs are available that aren’t included here.

The fourth IDE listed in this figure, Android Studio, is designed for developing Android apps. It was developed by Google and IntelliJ, and version 1.0 was released in December 2014.

This book shows how to use Eclipse because we think it’s a great IDE. However, we’ve also published a NetBeans version of this book because we think it’s a great IDE too.
Popular Java IDEs

<table>
<thead>
<tr>
<th>IDE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>A free, open-source IDE that runs on most modern operating systems. Eclipse is commonly used for developing all types of Java applications including Android apps.</td>
</tr>
<tr>
<td>NetBeans</td>
<td>A free, open-source IDE that runs on most modern operating systems. NetBeans is commonly used for most types of Java applications, but not for Android apps.</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>The Community Edition of this IDE is a free, open-source IDE that runs on most modern operating systems.</td>
</tr>
<tr>
<td>Android Studio</td>
<td>An IDE specifically designed for Android development that’s based on IntelliJ IDEA and backed by Google.</td>
</tr>
</tbody>
</table>

Features provided by most IDEs

- A code editor with code completion and error detection.
- Automatic compilation of classes when you run the application.
- A debugger that lets you set breakpoints, step through code, and view the values of active variables.

Description

- To develop Java applications, you typically use an Integrated Development Environment (IDE) like those listed above. All of these IDEs are either free or have free editions.
An introduction to Eclipse

In Eclipse, a *project* is a folder that contains all the folders and files for an application. The easiest way to get started with Eclipse is to import an existing project. For example, you can import the projects for any of the applications presented in this book after you download them from our website as described in the appendixes.

How to select a workspace

When you start Eclipse for the first time, it displays a dialog box like the one shown in figure 1-9 that allows you to select the workspace. A *workspace* is a folder that stores information about how to configure Eclipse including which projects to display. In this figure, the dialog box selects a workspace that you can use with this book. However, you can use any workspace you want, including the default workspace that’s selected the first time you start Eclipse.

After you select a workspace, Eclipse may display a Welcome page. If so, you can click the Workbench icon to go to a *workbench* like the one shown in figure 1-11. This is where you work on code.

Once you select a workspace, you can view all of the projects available from that workspace as described in the next few figures. If you want to switch to a different workspace, you can select the File → Switch Workspace command. Then, you can use the dialog box shown in this figure to select the new workspace. This restarts Eclipse, selects the new workspace, and displays its projects.

Once you have selected a workspace, it appears in the Workspace menu that’s available from the dialog box shown in this figure. This makes it easy to switch between workspaces.
The dialog box for selecting a workspace

![Workspace Launcher dialog box](image)

Description

- In Eclipse, a *project* contains the folders and files for a Java application.
- When you start Eclipse, you typically select the workspace you want to use. A *workspace* stores information about how to configure Eclipse including which projects to display.
- When Eclipse is running, you can switch to a different workspace by selecting the *File → Switch Workspace* command. This restarts Eclipse and selects the new workspace.
- Once you have selected a workspace, it appears in the Workspace menu. This makes it easy to switch between workspaces.
- After you select a workspace, Eclipse may display a Welcome page. If so, you can click the Workbench icon to go to the *workbench*, which is where you work on code.
How to import a project into a workspace

When you select a workspace for the first time, it doesn’t display any projects. To change that, you can import an existing project into the workspace as shown in figure 1-10. All of the projects for this book are located within this folder:

murach/java_eclipse

In this figure, I’ve selected the folder that contains the project named ch01_FutureValueConsole. As a result, that’s the only project that’s shown in the Projects section of the Import Projects dialog.

If there are multiple projects in the directory you select, all of the projects are displayed in the Projects section of this dialog. Then, you can import the ones you want by selecting them. For example, if you select the book_apps folder, the Projects section displays all projects in that folder. Then, you can select the ones you want to import.

When you import projects, it’s possible to copy the project into to the existing workspace. However, for the purposes of this book, you don’t need to do that.

It’s also possible to import other types of projects. For example, it’s possible to import projects that are stored in archive files. However, for the purposes of this book, the only types of projects you need to import are existing projects as shown in this figure.
The dialog boxes for importing a project

How to import a project

- Select the File → Import command from the menu system.
- In the first dialog box, select the General → Existing Projects into Workspace option and click on the Next button.
- In the second dialog box, click the Browse button and navigate to the folder that contains the project you want to import. This should display the project in the Projects pane and select it.
- Click the Finish button to import the selected project or projects.

How to import all projects in the book_apps folder

- To import all projects in the book_apps folder, navigate to the book_apps in step 3. This should select all projects in the book_apps folder.

Figure 1-10   How to import a project into a workspace
How to open a file in the code editor

Figure 1-11 shows a workspace that contains a project for a Java application. In this example, the project is named ch01_FutureValueConsole.

The Package Explorer shows that the folder for the project contains a subfolder named src. This folder contains the source code for the application. The second icon that looks like a stack of books is called a library, and it contains the libraries of code that your application uses. In this case, the application uses just the default libraries for Java 8, as indicated by the JRE System Library [JavaSE-1.8] label. However, you can add more libraries later if necessary.

Within the src folder, the source files can be organized into packages. In this figure, the project contains a single package named murach.fv. When you develop small applications like the one shown here, one package is usually acceptable. For larger applications, though, you’ll want to use two or more packages as described later in this book.

The application shown here consists of a single source code file named Main.java. In this figure, this file is open in the code editor. In the next chapter, you’ll learn more about working with this code editor. For now, all you need to know is that you can open a source code file in the code editor by double-clicking on it in the Package Explorer.
The Package Explorer and the code editor

How to navigate through the Package Explorer

- The Package Explorer displays the projects, folders, files, and libraries that make up an Eclipse workspace.
- To expand or collapse the nodes in the Package Explorer, click on the arrow signs to the left of its folders and files.
- The src (source) folder typically contains one or more packages, which are folders that store the .java files that contain the code for your application.
- If it isn’t visible, you can display the Package Explorer by selecting the Window→Show View→Package Explorer command.

How to open a file in the code editor

- You can use the code editor to edit the code in a .java file.
- To open a .java file in the code editor, use the Package Explorer to expand the src folder, expand the package that contains the file, and double-click on the .java file.
How to compile and run a project

Figure 1-12 shows how to compile and run a project. An easy way to run a project is to press Ctrl+F11. Eclipse automatically compiles your project, so there’s no need to compile it before running it.

If you want to remove all of the compiled files and force Eclipse to rebuild the entire project, you can use the Clean command as shown in the figure. This sometimes helps to get a project to work correctly after you have copied, moved, or renamed some of its files.

How to enter input for a console application

When you run a console application in Eclipse, any data that’s written to the console is displayed in the Console window. In addition, the Console window can accept input.

In this figure, for example, the application started by displaying a welcome message. Then, it prompted the user to enter a monthly investment. At this prompt, the user typed “100” and pressed Enter. After that, the application prompted the user to enter a yearly interest rate and a number of years. At both of these prompts, the user typed “3” and pressed Enter. Then, the application performed the calculation, and displayed the result. After that, the application asked the user if he or she wanted to continue. At this point, the application is still running, and the user can enter “y” to perform another calculation or “n” to end the application.

When you’re learning Java, it’s common to create applications that use the console to display output and get input. Because of that, the first four sections of this book teach you Java using console applications. Then, section 5 of this book teaches you how to create a graphical user interface.
A project that uses the Console window for input and output

How to compile and run a project

- To run the current project, press Ctrl+F11 or click the Run button in the toolbar.
- Eclipse automatically compiles your projects. As a result, you usually don’t need to compile a project before you run it.
- To delete all compiled files for a project and compile them again, select the project in the Package Explorer. Then, select the Project→Clean command.

How to work with the Console window

- When you run an application that prints data to the console, that data is displayed in the Console window.
- When you run an application that requests input from the console, the Console window pauses to accept the input. Then, you can click in the Console window, type the input, and press the Enter key.
- The Console window can also display messages and errors when you run an application.
How to work with two or more projects

Up to this point, this chapter has shown a workspace that contains a single project. However, Eclipse lets you import multiple projects into a workspace.

Figure 1-13 presents the skills for working with a workspace that contains two or more projects. When you import multiple projects, all of the imported projects appear in the Package Explorer. Then, when you open any of the files for a project, they appear in separate tabs in the main window. After you open a file, you can run the project for that file by pressing Ctrl+F11 or clicking on the Run button in the toolbar. Or, if you want to run a different project, you can select the project in the Package Explorer and press Ctrl+F11 or click on the Run button.

How to remove a project from a workspace

To remove a project from the current workspace, you can right-click on the project in the Project Explorer and select the Delete command from the context menu. This displays a confirmation dialog that asks if you really want to delete the project from the workspace.

By default, the project is only deleted from the current workspace, but the folder and files for the project remain on disk. This allows you to import it again later if you want. However, if you want to delete the project from disk, you can select the “Delete project contents” checkbox from the confirmation dialog. This removes the project from the workspace and deletes all of its folders and files.

This book often instructs you to right-click because that’s common in Windows. However, on Mac OS X, right-clicking is not enabled by default. If you want, you can enable right-clicking by editing the system preferences for your mouse. Or, if you prefer, you can hold down the Control key and click instead of right-clicking.

Similarly, this book presents keystrokes that work for Windows. However, with Mac OS X, you may need to modify some of these keystrokes by holding down the Command or Function (Fn) keys. In general, the Mac OS X keys are clearly marked in the menus. As a result, you can look them up if necessary.
Eclipse with two projects in the workspace

Description

- Eclipse lets you import and work with two or more projects at the same time.

How to change the current project

- When you open a file for a project, Eclipse opens the file in a tab in the main window and makes the project associated with that file the current project.
- To change the current project, click on the project in the Package Explorer window.

How to remove a project from the workspace

- To remove a project from the workspace, right-click on the project in the Package Explorer, select the Delete command, make sure the “Delete project contents” check box is not selected, and click the OK button.
- To remove a project from the workspace and delete its folders and files, right-click on the project in the Package Explorer, select the Delete item, select the “Delete project contents” check box, and click the OK button.

Mac OS X notes

- To enable right-clicking, you can edit the system preferences for the mouse.
- To use the Windows keys shown in this book, you may need to hold down the Command or Function (Fn) keys to modify those keys.
Perspective

In this chapter, you learned some background information about Java. In addition, you learned how to use Eclipse to import, compile, and run existing projects. With that as background, you’re ready to start learning how to write Java code and create new projects.

Summary

- You can use the Java Development Kit (JDK) to develop Java applications.
- The Standard Edition (SE) of Java is called Java SE.
- You can use Java SE to create desktop applications that run on your computer.
- A desktop application can use a graphical user interface (GUI) or a console to display output and get user input. Applications that use a console to interact with the user are known as console applications.
- You can use the Enterprise Edition of Java, which is known as Java EE, 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.
- A Java runtime environment (JRE) includes all of the software needed to run bytecodes.
- A JRE includes an implementation of a Java virtual machine (JVM).
- A JVM includes a Java interpreter to translate the Java bytecodes into native code that can be understood by the underlying operating system.
- Most modern JVMs have replaced the Java interpreter with a just-in-time compiler (JIT compiler). A JIT compiler is similar to an interpreter in some ways, but it improves performance significantly.
- An Integrated Development Environment (IDE) can make working with Java easier by providing code completion, error detection, automatic compilation, and a debugger.
- In Eclipse, a project is a folder that contains all of the folders and files that make up an application.
- In Eclipse, a workspace can hold one or more projects.
- If an application prints text to the console, Eclipse displays the text in the Console window. Eclipse also allows you to enter input into the Console window.
Before you do the exercises for this chapter

Before you do any of the exercises in this book, you need to install Eclipse and the JDK. In addition, you need to install the source code for this book from our website (www.murach.com). See appendix A (PC) or B (Mac) for details.

Exercise 1-1 Open and run two projects

This exercise guides you through the process of using Eclipse to open and run two applications.

Open and run the console version of the Future Value application
1. Start Eclipse. When you are prompted for a workspace, navigate to the murach/java_eclipse folder on your hard disk and select it.
2. Import the project named ch01_FutureValueConsole into the workspace.
3. Expand the murach.fv package.
4. Open the Main.java file in the code editor and review its code to get an idea of how this console application works.
5. Press Ctrl+F11 to run the application.
6. Click in the Console window to make it active and enter values for monthly investment, yearly interest rate, and years when you’re prompted for them. Then, when you’re asked if you want to continue, enter “n” to exit the application.

Open and run the GUI version of the Future Value application
7. Import the project named ch01_FutureValueGUI into the workspace.
8. Expand the murach.fv package.
9. Open the .java files in the code editor and review their code. This should give you an idea of what it takes to develop a simple GUI application. For now, don’t worry if you don’t understand this code! You’ll learn how to write code like this at the end of this book.
10. Click the Run button in the toolbar to run the application.
11. Enter values in the first three text boxes, click the Calculate button, and view the result of the calculation. Then, click the Exit button to exit the application.

Set the main project and run the applications again
12. Select the ch01_FutureValueConsole project in the Projects window. Then, press Ctrl+F11 to run this application.
13. Select the ch01_FutureValueGUI project in the Projects window. Then, click the Run button to run this application.
The easiest way is to let Murach’s Beginning Java with Eclipse be your guide! So if you’ve enjoyed this chapter, I hope you’ll get your own copy of the book today. And don’t miss its companion text for web programming, Murach’s Java Servlets and JSP (3rd Edition). You can use both books to:

- Teach yourself how to code desktop and web applications in Java
- Take advantage of all the time- and work-saving features that are offered in an IDE like Eclipse as you develop Java applications
- Understand how object-oriented programming really works to create your own 3-tier database applications, the way the pros do
- Pick up a new skill whenever you want or need to by focusing on material that’s new to you
- Look up coding details or refresh your memory on forgotten details when you’re in the middle of developing a Java application
- Loan to your colleagues who will be asking you more and more questions about Java programming

To get your copy, you can order online at www.murach.com or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

You must be satisfied. Each book you buy directly from us must outperform any competing book or course you’ve ever tried, or send it back within 60 days for a full refund…no questions asked.

Thanks for your interest in Murach books!

-- Mike