Thanks for downloading this chapter from Murach’s Java Programming (5th Edition). 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 the previous editions

“Finally there is a Java book for serious programmers doing real life business applications.”

Donna Dean, IS Trainer, Chicago, Illinois

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

Richard Cooper, Programmer

“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

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

Posted at Amazon.com

“Murach’s Java is now my go-to source for reference and learning and brushing up, well above and beyond the other books in my collection.”

Jeff Salter, Sacramento Java Users Group (SacJUG)

“I love the 2-page style that you use in your Java book. It is such a great way to organize the lessons.”

Dave Merton, Director of Development, Connecticut

“If I’d seen this book first, I would not have wasted money (and time) on 6 other books! This one is highly organized, clear, and very effective as a learning tool.”

Posted at Amazon.com

“The style is clean, very user friendly, and simple. The narrative is totally accurate and focuses on important issues; it is not dumbed down. This book is a winner!”

Dr. Richard Wiener, Journal of Object Technology
## Contents

Introduction xvii

### Section 1 Essential skills

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>An introduction to Java</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>How to write your first Java applications</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>How to work with the primitive data types</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>How to code control statements</td>
<td>119</td>
</tr>
<tr>
<td>5</td>
<td>How to code methods, handle exceptions, and validate data</td>
<td>151</td>
</tr>
<tr>
<td>6</td>
<td>How to test, debug, and deploy an application</td>
<td>179</td>
</tr>
</tbody>
</table>

### Section 2 Object-oriented programming

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>How to define and use classes</td>
<td>207</td>
</tr>
<tr>
<td>8</td>
<td>How to work with inheritance</td>
<td>259</td>
</tr>
<tr>
<td>9</td>
<td>How to define and use interfaces</td>
<td>291</td>
</tr>
<tr>
<td>10</td>
<td>More object-oriented programming skills</td>
<td>323</td>
</tr>
</tbody>
</table>

### Section 3 More essential skills

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>How to work with arrays</td>
<td>353</td>
</tr>
<tr>
<td>12</td>
<td>How to work with collections and generics</td>
<td>379</td>
</tr>
<tr>
<td>13</td>
<td>How to work with strings</td>
<td>419</td>
</tr>
<tr>
<td>14</td>
<td>How to work with dates and times</td>
<td>439</td>
</tr>
<tr>
<td>15</td>
<td>How to work with file I/O</td>
<td>459</td>
</tr>
<tr>
<td>16</td>
<td>How to work with exceptions</td>
<td>501</td>
</tr>
</tbody>
</table>

### Section 4 GUI programming

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>How to get started with JavaFX</td>
<td>531</td>
</tr>
<tr>
<td>18</td>
<td>How to get started with Swing</td>
<td>573</td>
</tr>
<tr>
<td>19</td>
<td>More Swing controls</td>
<td>619</td>
</tr>
</tbody>
</table>

### Section 5 Database programming

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>An introduction to databases with SQLite</td>
<td>649</td>
</tr>
<tr>
<td>21</td>
<td>How to use JDBC to work with a database</td>
<td>671</td>
</tr>
</tbody>
</table>

### Section 6 Advanced skills

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>How to work with lambda expressions and streams</td>
<td>699</td>
</tr>
<tr>
<td>23</td>
<td>How to work with threads</td>
<td>725</td>
</tr>
</tbody>
</table>

### Reference Aids

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>How to set up Windows for this book</td>
<td>743</td>
</tr>
<tr>
<td>B</td>
<td>How to set up Mac OS X for this book</td>
<td>753</td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>763</td>
</tr>
</tbody>
</table>
Section 1

Essential skills

This section gets you started quickly with Java programming. First, chapter 1 introduces you to Java applications and shows you how to use an IDE to work with Java projects. Then, chapter 2 shows you how to write your first 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 the eight primitive data types. Chapter 4 presents the details for coding control statements. Chapter 5 shows how to code methods, handle exceptions, and validate data. And chapter 6 shows how to thoroughly test and debug an application. In addition, it shows how to deploy an application.

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.
An introduction to Java

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.

After the background information, this chapter shows how to use the NetBeans IDE (Integrated Development Environment) to work with a Java application. For this book, we recommend using NetBeans because we think it’s the best and most intuitive IDE for getting started with Java.

However, Eclipse is another great IDE for working with Java applications, and many programmers prefer it. As a result, the download for this book includes a PDF file that shows how to use Eclipse with this book instead of NetBeans. So, if you want to use Eclipse, you can use this PDF file whenever you need to learn how to perform a task with Eclipse.
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 1.9. 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 9 and Java 1.9 both refer to the ninth 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 9. However, Java is backwards compatible, so future versions of Java should work with this book too. In addition, most of the skills described in this book have been a part of Java since its earliest versions. As a result, earlier versions of Java work with most of the skills described in this book.

Java editions

This figure also describes the three most common editions of Java. To start, the Standard Edition 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 ones presented in section 4.

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>Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>January</td>
<td>JDK 1.0</td>
</tr>
<tr>
<td>1997</td>
<td>February</td>
<td>JDK 1.1</td>
</tr>
<tr>
<td>1998</td>
<td>December</td>
<td>SDK 1.2</td>
</tr>
<tr>
<td>1999</td>
<td>August</td>
<td>Java 2 Platform, Standard Edition (J2SE)</td>
</tr>
<tr>
<td></td>
<td>December</td>
<td>Java 2 Platform, Enterprise Edition (J2EE)</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>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>
<tr>
<td>2017</td>
<td>July</td>
<td>Java SE 9 with JDK 1.9</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).
- Java was originally developed and released by Sun Microsystems. However, Oracle bought Sun Microsystems in April 2010.
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# handles memory operations automatically.

C# can run on any platform that has a runtime environment for it. However, Windows is the only operating system that fully supports a 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 all Java 7 language features and some Java 8 features 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>

Figure 1-2  How Java compares to C++ and C#
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, I 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 section 4. Until then, this book uses console applications to teach the basics of Java.
A console application

A console application uses the console to interact with the user.

A GUI application

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

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.
Web applications and mobile apps

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. An applet could be stored in an HTML page and run inside a Java-enabled browser. With tightening security restrictions in recent years, applets are effectively obsolete. As a result, we don’t cover them in this book.

However, many web applications still rely on servlets. A servlet is a special type of Java application that runs on the server and can be called by a client such as a web browser. This is illustrated by the first screen in figure 1-4. To start, when the user clicks the Calculate button, 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.

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, this figure shows a mobile app that was developed with Java.

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 all features of Java 7 and some features of Java 8. 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 web application

A mobile app

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, so 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.

- 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 all features of Java 7 and some features of Java 8.
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 1-5 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 class named FutureValueApp that corresponds with a file named FutureValueApp.java. This class begins with an opening brace ( { ) and ends with a closing brace ( } ).

Within this class, two methods are defined. These methods also begin with an opening brace and end with a closing brace, and they are indented to clearly show that they are contained within the class.

The first method, named main(), is the main method for the application. The code within this method is executed automatically when you run the application. In this case, the code prints data to the console to prompt the user, accepts the data the user enters at the console, and calculates the future value. To do that, the main() method calls a second method, named calculateFutureValue(). This method calculates the future value and returns the result to the main() method.
The code for a console application

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

public class FutureValueApp {

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

        // get a Scanner object to scan for user input
        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 = sc.nextDouble();

            System.out.print("Enter yearly interest rate: ");
            double interestRate = sc.nextDouble();

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

            // convert all input values to months
            double monthlyInterestRate = interestRate / 12 / 100;
            int months = years * 12;

            // call method to calculate future value
            double futureValue = calculateFutureValue(
                monthlyInvestment, monthlyInterestRate, months);

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

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

    private static double calculateFutureValue(double monthlyInvestment,
        double monthlyInterestRate, int months) {
        double futureValue = 0;
        for (int i = 1; i <= months; i++) {
            futureValue = (futureValue + monthlyInvestment) * (1 + monthlyInterestRate);
        }
        return futureValue;
    }
}
```

Figure 1-5 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 bytecode as shown in figure 1-6. At this point, the bytecode 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 bytecode. Among other things, this includes an implementation of a Java virtual machine (JVM). This JVM includes a Java interpreter to translate the Java bytecode 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 bytecode 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.
How Java compiles and interprets code

**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 *bytecode*. Files that contain Java bytecode have the .class extension.
- A *Java virtual machine (JVM)* includes a *Java interpreter* that executes Java bytecode. 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 bytecode including a JVM. Since JREs are available for most operating systems, Java bytecode can be run on most operating systems.
An introduction to Java IDEs

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 1-7 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, NetBeans and Eclipse, 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 NetBeans and Eclipse. 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 is now the official IDE for developing Android apps.
Popular Java IDEs

<table>
<thead>
<tr>
<th>IDE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBeans</td>
<td>A free, open-source IDE that runs on most modern operating systems. NetBeans is commonly used for developing most types of Java applications, but not for Android apps.</td>
</tr>
<tr>
<td>Eclipse</td>
<td>A free, open-source IDE that runs on most modern operating systems. Eclipse is commonly used for developing 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.
How to use NetBeans to work with existing projects

Now that you have some background information about Java, you’re ready to start working with existing NetBeans projects. In particular, you’re ready to learn how to open and run any of the projects for this book. You can download these projects as described in the appendixes for this book.

An introduction to NetBeans

Figure 1-8 shows the NetBeans IDE with an open Java project. In NetBeans, a project is a folder that contains all the files for an application. In this example, the project is named ch01_FutureValueConsole.

In the Projects window, the ch01_FutureValueConsole folder contains two subfolders: (1) Source Packages and (2) Libraries. The Source Packages folder contains the source files for the application, and the Libraries folder contains the Java libraries that are used by the application. In this case, the application uses just the JDK 1.8 libraries, but you can add others.

Within the Source Packages folder, the source files can be organized into packages. In this case, there’s only one file, and it’s stored in the default package. For simple applications like this, that’s acceptable. However, as you develop more complex applications, it’s considered a best practice to store your source files in packages as described in chapter 10.

In this figure, the application consists of a single source file named FutureValueApp.java, and this file is open in the code editor. Because this class contains the main() method for the application, it’s called the main class. When you run an application, the main() method in the main class is executed by default.

As of this writing, Java 9 has not yet been released. That’s why the JDK 1.8 library is used by this application, and that’s why NetBeans 8.1 is used to work with this project. When Java 9 is released in July of 2017, though, you’ll be able to use JDK 1.9 and NetBeans 9.0.
NetBeans with a Java project

Description

- A NetBeans project consists of a top-level folder that contains the subfolders and files for an application.
- The Source Packages subfolder contains the .java files that make up the project. These files define classes that are later compiled into .class files.
- At a minimum, a project consists of a single class that contains the main() method. The main() method is the starting point for the application. The class that contains the main() method can be referred to as the main class.
- The .java files that make up a project can be organized into one or more packages. It’s generally considered a best practice to organize your classes by storing each class within a package.
- The Libraries subfolder contains the libraries that are available to your project. These libraries contain the Java classes that you can use in your projects. By default, you can use the classes in the JDK libraries.
- The folders, files, and libraries that make up a Java project are listed in the Projects window. If this window isn’t visible, you can display it by using the Window→Projects command. Then, you can expand and collapse the nodes in this window by clicking on the plus and minus signs.
- When Java 9 is released in July of 2017, you’ll be able to use JDK 1.9 instead of 1.8 and NetBeans 9.0 instead of 8.1.
How to open, close, and delete a project

To open a project in NetBeans, you use the Open Project dialog box shown in figure 1-9. This dialog box lets you navigate to the folder that contains the project you want to open. In this figure, for example, the Open Project dialog box shows all of the existing NetBeans projects in this folder:

```
C:\murach\java\netbeans\book_apps
```

To clearly indicate when a folder contains a Java project, the Open Project dialog box displays a small coffee cup icon to the left of the folder name. Then, you select the project you want to open and click the Open Project button.

When you're done working with a project, you can close it to remove it from the Projects window. To do that, you can use one of the techniques described in this figure.

You can also delete a project if you decide that you no longer want to work with it in NetBeans. Before the project is deleted, NetBeans will prompt you to confirm the deletion. Then, by default, NetBeans deletes all of the files for the project except for the source files. That way, you can work with those files outside of NetBeans if you want to. If you want to delete the source files as well, you can select the “Also Delete Sources” option in the dialog box that's displayed.

How to compile and run a project

Figure 1-9 also describes how to compile and run a project. An easy way to run a project is to press F6. Then, if the project has been modified since the last time it was compiled, NetBeans automatically compiles the project and runs the main() method in the main class.

If you want to compile a project without running it, you can use the Build command as described in this figure. You can also use the Clean and Build command to compile the project and remove any files that are no longer needed. This sometimes helps to get a project to work correctly after you have copied, moved, or renamed some of its files.
The dialog box for opening a project

![Open Project Dialog Box](image)

How to open, close, and delete a project

- To open a project, click the Open Project button in the toolbar or select the File→Open Project command. Then, use the Open Project dialog box that’s displayed to locate and select the project and click the Open Project button.
- You can also open a project by using the File→Open Recent Project command and then selecting the project from the list that’s displayed.
- To close a project, right-click on the project in the Projects window and select the Close command, or select the project and then use the File→Close Project command.
- To delete a project, right-click on the project in the Projects window and select the Delete command. When you do, you’ll have the option of deleting just the files that NetBeans uses to manage the project or deleting all the folders and files for the project.

How to compile and run a project

- To run a project, press F6 or click the Run Project button in the toolbar.
- When you run a project, NetBeans automatically compiles it. As a result, you usually don’t need to compile a project separately.
- To compile a project without running it, you can right-click on the project in the Projects window and select the Build command.
- To delete all compiled files for a project and compile them again, you can right-click on the project and select the Clean and Build command. This removes files that are no longer needed and compiles the entire project.

Mac OS X note

- To enable right-clicking with Mac OS X, you can edit the system preferences for the mouse.
How to use the Output window with a console application

When you run a console application in NetBeans, any data that’s written to the console is displayed in the Output window. In addition, the Output window can accept input. This is illustrated in figure 1-10.

In this figure, the application prompts the user to enter the monthly investment amount, the yearly interest rate, and the number of years. Then, it calculates and displays the future value of this series of investments.

This application started by displaying a welcome message. Then, it displayed a prompt indicating that the user should enter a monthly investment. In response, the user typed “100” and pressed Enter. After that, the application prompted the user for two more values, and the user entered “3” for those values. Then, the application displayed the future value and asked the user if it should 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 three sections of this book show how to code console applications. Then, section 4 of this book teaches you how to create applications that use a graphical user interface (GUI).
An application that uses the Output window for input and output

Description

- When you run an application that prints data to the console, that data is displayed in the Output window.
- When you run an application that requests input from the console, the Output window pauses to accept the input. Then, you can click in the Output window, type the input, and press the Enter key.
- In addition to displaying output and accepting input, the Output window can display other information. For example, it can display messages when the application is compiled, and it can display errors that are encountered when an application is run.
How to work with two or more projects

Up to this point, you’ve seen how to work with a single project in NetBeans. However, NetBeans lets you open and work with two or more projects at the same time. If, for example, you want to run some of the projects from the download for this book before you start creating your own projects, you can open those projects in NetBeans at the same time. You’ll get a chance to do that in the first exercise for this chapter.

Figure 1-11 presents the skills for working with two or more projects. When you open two or more projects, all of the open projects appear in the Projects window. 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 F6 or clicking on the Run Project button in the toolbar. Or, if you want to run a different project, you can select the project in the Projects window and then press F6 or click on the Run Project button.
NetBeans with two open projects

Description

- NetBeans lets you open and work with two or more projects at the same time.
- When you open multiple projects, they all appear in the Projects window.
- When you open a file for a project, NetBeans opens the file in a tab in the main window.
- To run the currently selected project, press F6 or click the Run Project button in the toolbar.
- To change the currently selected project, click on the project in the Projects window.

Figure 1-11 How to work with two or more projects
How to use NetBeans to develop new projects

Now that you know how to work with existing Java projects in NetBeans, you’re ready to learn how to develop new Java projects. That’s what you’ll learn in the remainder of this chapter.

How to create a new project

Figure 1-12 presents the dialog boxes for creating a Java application. You use the New Project dialog box to choose the type of project you want to create. In most cases, you’ll create a Java Application project as shown here. Then, when you click the Next button, NetBeans displays a New Java Application dialog box like the second one in this figure.

The New Java Application dialog box lets you enter a name and location for the project. In this figure, for example, the project name is “ch01_Test” and it will be stored in this folder:

C:\murach\java\netbeans\book_apps

If you install the source code for this book as described in the appendix, all of the applications presented in this book should be stored within this folder.

By default, when you create a Java application, NetBeans generates a main class with a main() method. If that’s not what you want, you can remove the check mark from the “Create Main Class” option. In most cases, though, you’ll leave this option checked. Then, you can enter a name for the main class. In addition, you typically want to enter a package for this class.

For the project in this figure, for example, NetBeans suggested “ch01_test.Ch01_Test”. Here, “ch01_test” would be the name of the package and “Ch01_Test” would be the name of the class. However, I deleted the name of the package, and I changed the name of the class to TestApp. As a result, NetBeans created a project named ch01_Test that contains a main class named TestApp in the default package.

When this dialog box is complete, you can click the Finish button to create the project and the class that contains the main() method. Then, NetBeans creates a folder that corresponds with the project name, and it creates some additional files that it uses to configure the project.
The dialog boxes for creating a new project

Description

- To create a new project, use the File→New Project command or click the New Project button in the toolbar to display the New Project dialog box. Then, select a project type, click the Next button, and complete the dialog box that’s displayed.
- To create a Java Application project, enter the project name and location and the name you want to use for the main class. It’s generally considered a best practice to enter a package name for each class.
**How to work with Java source code and files**

When you create a new project that contains a class with a main() method, the class is typically opened in a new code editor window as shown in figure 1-13. To make it easier for you to recognize the Java syntax, the code editor uses different colors for different language elements. In addition, NetBeans provides standard File and Edit menus and keystroke shortcuts that let you save and edit the source code. For example, you can press Ctrl+S to save your source code, and you can use standard commands to cut, copy, and paste code.

When you create a project with a main class, NetBeans generates some code for you. In this figure, for example, NetBeans generated the code that declares the class, the code that declares the main() method, and comments that describe the class and method. Although you can delete or modify the class and method declarations, you won’t usually do that. However, you may want to delete or modify some or all of the comments.

If the source code you want to work with isn’t displayed in a code editor window, you can use the Projects window to navigate to the .java file and then double-click on it to open it in a code editor window. In this figure, for example, you could double-click on the TestApp.java file in the Projects window to open it in the code editor window.

You can also rename or delete a .java file from the Projects window. To do that, just right-click on the file and select the appropriate command. If you rename a file, NetBeans automatically changes both the name of the .java file and the name of the class. Since the name of the .java file must match the name of the class, this is usually what you want.
NetBean’s code editor with the starting source code for a project

![NetBean's code editor](image)

**Description**

- To open a .java file in the *code editor*, double-click on it in the Projects window. Then, you can use normal editing techniques to work with the source code.

- To collapse the code for a method or comment, click the minus sign (−) to its left. Then, a plus sign (+) appears to the left of the method or comment, and you can click the plus sign to display the code again.

- To save the source code for a file, use the File ➔ Save command (Ctrl+S) or click the Save All Files button in the toolbar. This automatically compiles the file so it doesn’t have to be compiled when the project is run.

- To rename a file, right-click on it, select the Refactor ➔ Rename command, and enter the new name in the resulting dialog box.

- To delete a file, you can right-click on it, select the Delete command, and confirm the deletion in the resulting dialog box.

---

*Figure 1-13 How to work with Java source code and files*
How to use the code completion feature

Figure 1-14 shows how to use the code completion feature. This feature prevents you from making typing mistakes, and it allows you to discover what fields and methods are available from various classes and objects. In this figure, for example, I started to enter a statement that prints text to the console.

First, I entered “sys” and pressed Ctrl+Spacebar (both keys at the same time). This displayed a list with the System class as the only option. Then, I pressed the Enter key to automatically enter the rest of the class name.

Next, I typed a period. This displayed a list of fields and methods available from the System class. Then, I used the arrow keys to select the field named out and pressed the Enter key to automatically enter that field name.

Finally, I typed another period. This displayed a long list of method names. Then, I typed “pr” to scroll down the list to the methods that start with “pr”, and I used the arrow keys to select one of the println() methods. At this point, I could press Enter to have NetBeans enter the method into the editor for me.

When you use code completion, it automatically enters opening and closing parentheses and arguments whenever they’re needed. In this figure, for example, the selected println() method is followed by a set of parentheses that contains a string argument. When I inserted this method into the code editor, NetBeans inserted the parentheses and highlighted the argument so I could enter a value for it.

The code completion feature can also make it easy for you to enter values for string variables. If you type a quotation mark to identify a string value, the code completion feature automatically adds the closing quotation mark and places the cursor between the two quotes. At this point, you can enter the text for the string.

If you experiment with the code completion feature, you’ll quickly see when it helps you enter code more quickly and when it makes sense to enter the code yourself. In addition, you’ll see that it helps you discover the fields and methods that are available to the various classes and objects that you’re working with. This will make more sense as you learn more about Java in the next few chapters.
The code editor with a code completion list

![Image of code editor with code completion feature]

**Description**

- You can use the *code completion feature* to help you enter the names of classes and objects and select from the methods and fields that are available for a class or object.

- To activate the code completion feature for entering a class or object name, press Ctrl+Spacebar after entering one or more letters. Then, a list of all the classes and objects that start with those letters is displayed.

- To insert an item from a code completion list, select the item and then press the Enter key.

- If you enter the opening quote for a string value, the code completion feature automatically adds the closing quote and places the cursor between the two quotes.

*Figure 1-14  How to use the code completion feature*
How to detect and correct syntax errors

A syntax error is a statement that won’t compile. As you enter text into the code editor, NetBeans displays syntax errors whenever it detects them. In figure 1-15, for example, NetBeans displays an error that indicates that a semicolon needs to be entered to complete the statement. This error is marked with a red icon to the left of the statement. In addition, the statement that contains the error is marked with a wavy red underline.

If you position the mouse pointer over the red error icon or over the statement itself, NetBeans displays a description of the error. In this figure, for example, the description indicates that NetBeans expects a semicolon at the end of the statement. As a result, you can fix the error by typing the semicolon.
The code editor with an error displayed

![Image of code editor with an error]

**Description**

- NetBeans often detects *syntax errors* as you enter code into the code editor.
- When NetBeans detects a syntax error, it displays a red error icon to the left of the statement in error, and it places a red wavy line under the statement.
- To get more information about a syntax error, you can position the mouse pointer over the error icon. Or, you can move the cursor to the line that contains the error and press Alt+Enter.
**Perspective**

In this chapter, you were introduced to Java, and you learned how to use an IDE such as NetBeans to create and run a Java application. With that as background, you’re ready to learn how to write your own Java applications. But first, I recommend that you familiarize yourself with an IDE such as NetBeans by doing the exercises at the end of this chapter.

**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 *web applications*.
- You can use Java to write the code for *mobile apps* such as Android apps.
- The *Java compiler* translates *source code* into a *platform-independent* format known as *Java bytecode*.
- A *Java runtime environment (JRE)* includes all of the software needed to run bytecode.
- A JRE includes an implementation of a *Java virtual machine (JVM)*.
- A JVM includes a *Java interpreter* to translate the Java bytecode into native code that can be understood by the underlying operating system.
- An *Integrated Development Environment (IDE)* can make working with Java easier by providing code completion, error detection, automatic compilation, and a debugger.
- NetBeans and Eclipse are two of the most commonly used IDEs for Java development.
- Java code is stored in *classes*. The *main class* of an application is the class that contains the *main method* (named main()), which is the starting point of the application.
Before you do the exercises for this chapter

Before you do any of the exercises in this book, you need to install the JDK and NetBeans. In addition, you need to install the source code for this book from our website (www.murach.com). For complete instructions, see the appendixes of this book.

If you want to use Eclipse instead of NetBeans, you can find an Eclipse version of these exercises in the PDF file that’s included in the download for this book.

Exercise 1-1  Use NetBeans to open and run two projects

This exercise shows you how to use NetBeans to open and run two console applications.

Open and run the Invoice application
2. Open the project named ch01_ex1_Invoice. On a Windows system, the project should be stored in this directory:
   C:\murach\java\netbeans\ex_starts
3. Open the InvoiceApp.java file in the code editor and review its code to get an idea of how this application works.
4. Press F6 to run the application. Enter a subtotal when you’re prompted, and then enter “n” when you’re asked if you want to continue.

Open and run the Test Score application
5. Open the project named ch01_ex2_TestScore. Then, open the TestScoreApp.java file in the code editor and review its code.
6. Click the Run Project button in the toolbar to run the application. Enter one or more grades when you’re prompted, and enter 999 to end the application.

Run the applications again
7. Select the Invoice application in the Projects window. Then, press F6 to run this application.
8. Select the Test Score application in the Projects window and click the Run Project button to run this application.
9. Close both projects.
Exercise 1-2  Use NetBeans to develop an application

This exercise guides you through the process of using NetBeans to enter, save, compile, and run a simple application.

Enter the source code and run the application
2. Select the File→New Project command from the NetBeans menu system. Then, use the resulting dialog boxes to create a Java Application project named ch01_Test that contains a main class named TestApp. On a Windows system, store the project in this directory:
   C:\murach\java\netbeans\ex_starts
3. Modify the generated code for the TestApp class so it looks like this (type carefully and use the same capitalization):
   ```java
   public class TestApp {
       public static void main(String[] args) {
           System.out.println("Success!");
       }
   }
   ```
4. Press F6 to compile and run the application. This should display “Success!” in the Output window.

Use the code completion feature
5. Enter the statement that starts with System.out again, right after the first statement. This time, type “sys” and then press Ctrl+Spacebar. Then, use the code completion feature to select the System class, and complete the statement.
6. Enter this statement a third time, right after the second statement. This time, type System, enter a period, and select out from the list that’s displayed. Then, enter another period, select the println() method, and complete the statement. You should now have the same statement three times in a row.
7. Run the application again. It should display the message three times in a row in the Output window.

Introduce and correct a syntax error
8. In the code editor window, delete the semicolon at the end of the first println() method. When you do, NetBeans should display an error icon to the left of the statement.
9. Correct the error. When you do, NetBeans should remove the error icon.
10. Use the File→Save command (Ctrl+S) to save the changes.
How to build your Java programming skills

The easiest way is to let Murach’s Java Programming (5th Edition) 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 texts for web and mobile programming, Murach’s Java Servlets and JSP (3rd Edition) and Murach’s Android Programming (2nd Edition). You can use these books to:

- Teach yourself how to code desktop, web, and mobile applications in Java
- Take advantage of all the time- and work-saving features offered by an IDE as you develop your applications
- Understand how object-oriented programming really works to create your own 3-tiered 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 copies of these books, 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!