CHAPTER1: Introduction to Java

OBJECTIVES

After studying this chapter, the student will be able to

- To describe the relationship between Java and the World Wide Web
- To understand the meaning of Java language specification, API, JDK and IDE
- To write a simple Java program.
- To display output on the console.
- To explain the basic syntax of a Java program.
- To create, compile, and run Java programs.
- To use sound Java programming style and document programs properly.
- To explain the differences between syntax errors, runtime errors and logic errors.
- To develop Java programs using Eclipse.

1.1 What is programming?

The term programming means to create (or develop) software, which is also called a program. In basic terms, software contains the instructions that tell a computer—or a computerized device—what to do. Software is all around you, even in devices that you might not think would need it. Of course, you expect to find and use software on a personal computer, but software also plays a role in running airplanes, cars, cell phones, and even toasters. On a personal computer, you use word processors to write documents, Web browsers to explore the Internet, and e-mail programs to send and receive messages. These programs are all examples of software. Software developers create software with the help of powerful tools called programming languages.

This course teaches you how to create programs by using the Java programming language. There are many programming languages, some of which are decades old. Each language was invented for a specific purpose—to build on the strengths of a previous language, for example, or to give the programmer a new and unique set of tools. Knowing that there are so many programming languages available, it would be natural for you to wonder which one is best. But, in truth, there is no "best" language. Each one has its own strengths and weaknesses. Experienced programmers know that one language might work well in some situations, whereas a different language may be more appropriate in other situations. For this reason, seasoned programmers try to master as many different programming languages as they can, giving them access to a vast arsenal of software-development tools.

If you learn to program using one language, you should find it easy to pick up other languages. The key is to learn how to solve problems using a programming approach. That is the main theme of this course.

1.2 Programming Languages

Computer programs, known as software, are instructions that tell a computer what to do. Computers do not understand human languages, so programs must be written in a language a computer can use. There are hundreds of programming languages, and they were developed to make the programming process easier for people. However, all programs must be converted into the instructions the computer can execute.

1.2.1 Machine Language

A computer's native language, which differs among different types of computers, is its ***machine language***—a set of built-in primitive instructions. These instructions are in the form of binary code, so if you want to give a computer an instruction in its native language, you have to enter the instruction as binary code.

For example, to add two numbers, you might have to write an instruction in binary code, like this:

1101101010011010

1.2.2 Assembly Language

Programming in machine language is a tedious process. Moreover, programs written in machine language are very difficult to read and modify. For this reason, ***assembly language*** was created in the early days of computing as an alternative to machine languages. Assembly language uses a short descriptive word, known as a mnemonic, to represent each of the machine-language instructions. For example, the mnemonic add typically means to add numbers and sub means to subtract numbers. To add the numbers ***2*** and ***3*** and get the ***result***, you might write an instruction in assembly code like this:

add 2, 3, result

Assembly languages were developed to make programming easier. However, because the computer cannot execute assembly language, another program—called an ***assembler***—is used to translate assembly-language programs into machine code, as shown in Figure 1.1.

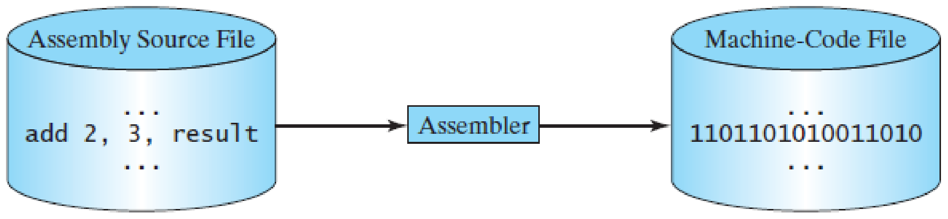


Figure 1.1: An assembler translates assembly-language instructions into machine code.

Writing code in assembly language is easier than in machine language. However, it is still tedious to write code in assembly language. An instruction in assembly language essentially corresponds to an instruction in machine code. Writing in assembly requires that you know how the CPU works. Assembly language is referred to as a ***low-level language***, because assembly language is close in nature to machine language and is machine dependent.

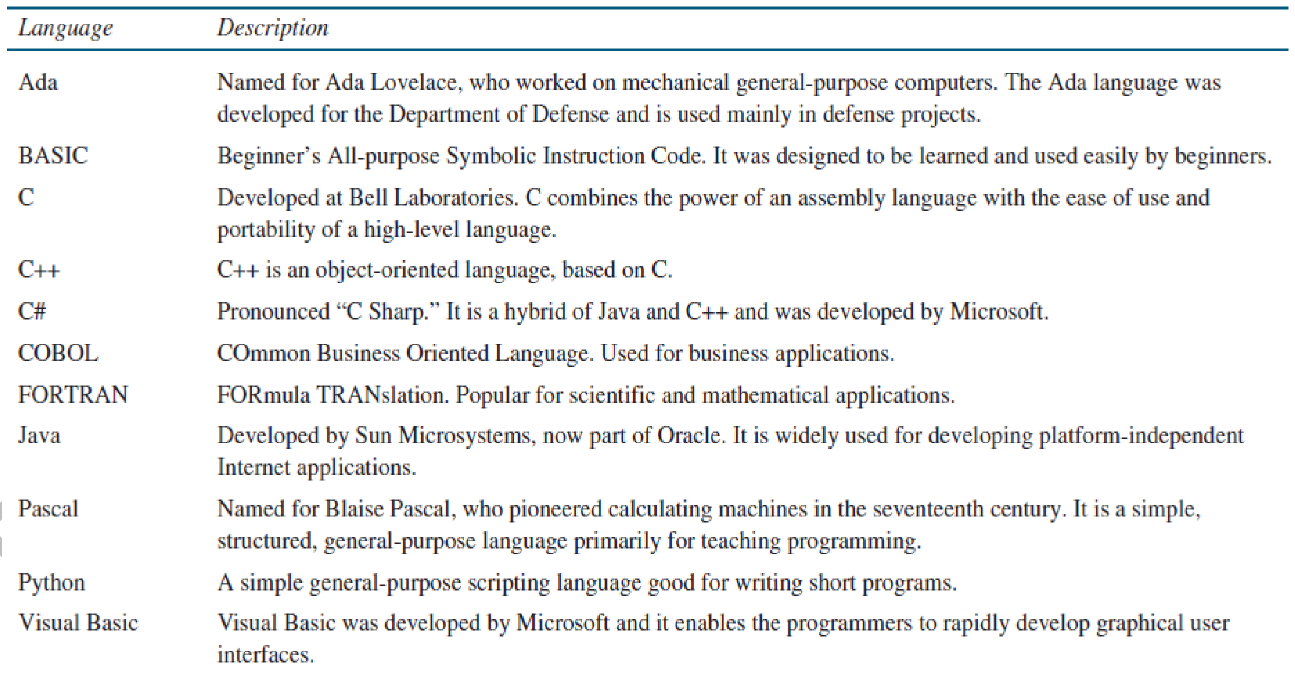
1.2.3 High-Level Language

In the 1950s, a new generation of programming languages known as ***high-level languages*** emerged. They are platform independent, which means that you can write a program in a high-level language and run it in different types of machines. High-level languages are English-like and easy to learn and use. The instructions in a high-level programming language are called ***statements***. Here, for example, is a high-level language statement that computes the area of a circle with a radius of 5:

area = 5 * 5 * 3.14159;

There are many high-level programming languages, and each was designed for a specific purpose. Table 1.1 lists some popular ones.

TABLE 1.1 Popular High-Level Programming Languages

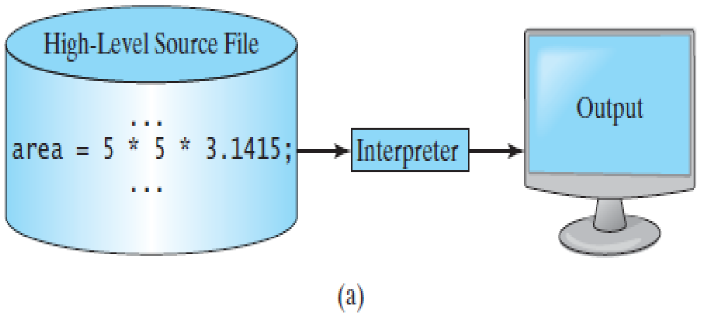


Language	*Description*
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	C++ is an object-oriented language, based on C.
C#	Pronounced "C Sharp." It is a hybrid of Java and C++ and was developed by Microsoft.
COBOL	COmmon Business Oriented Language. Used for business applications.
FORTRAN	FORmula TRANslation. Popular for scientific and mathematical applications.
Java	Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications.
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming.
Python	A simple general-purpose scripting language good for writing short programs.
Visual Basic	Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop graphical user interfaces.

A program written in a high-level language is called a ***source program or source code***. Because a computer cannot execute a source program, a source program must be translated into machine code for execution. The translation can be done using another programming tool called an ***interpreter or a compiler***.

■ An ***interpreter*** reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away, as shown in Figure 1.2a. Note that a statement from the source code may be translated into several machine instructions.

■ A ***compiler*** translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in Figure 1.2b.



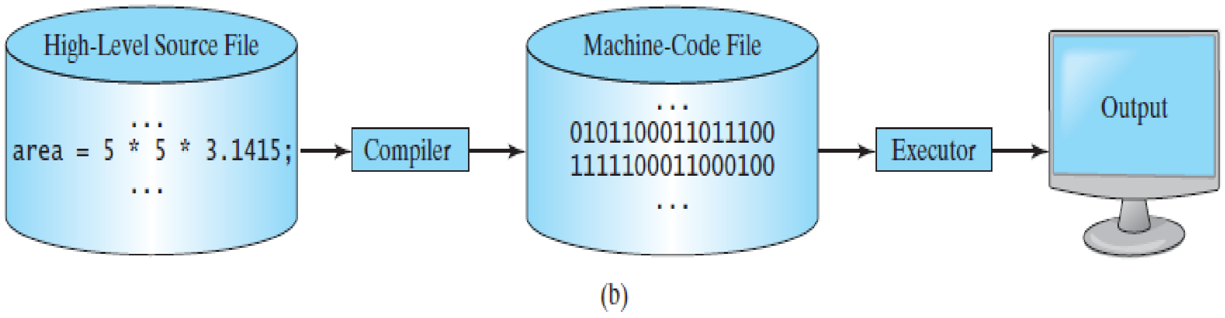


Figure 1.2: (a) An interpreter translates and executes a program one statement at a time. (b) A compiler translates the entire source program into a machine-language file for execution.

1.3 Java, the World Wide Web, and Beyond

Java is a powerful and versatile programming language for developing software running on mobile devices, desktop computers, and servers. Java was developed by a team led by James Gosling at Sun Microsystems. Sun Microsystems was purchased by Oracle in 2010. Originally called Oak, Java was designed in 1991 for use in embedded chips in consumer electronic appliances. In 1995, renamed Java, it was redesigned for developing Web applications.

Java has become enormously popular. Its rapid rise and wide acceptance can be traced to its design characteristics, particularly its promise that you can write a program once and run it anywhere. As stated by its designer, **Java *is simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, and dynamic.***

Java is a full-featured, general-purpose programming language that can be used to develop robust mission-critical applications. Today, it is employed not only for Web programming but also for developing standalone applications across platforms on servers, desktop computers, and mobile devices. It was used to develop the code to communicate with and control the robotic rover on Mars. Many companies that once considered Java to be more hype than substance are now using it to create distributed applications accessed by customers and partners across the Internet. For every new project being developed today, companies are asking how they can use Java to make their work easier.

The World Wide Web is an electronic information repository that can be accessed on the Internet from anywhere in the world. The Internet, the Web's infrastructure, has been around for more than forty years. The colorful World Wide Web and sophisticated Web browsers are the major reason for the Internet's popularity.

Java initially became attractive because Java programs can be run from a Web browser. Such programs are called ***applets***. Applets employ a modern graphical interface with buttons, text fields, text areas, radio buttons, and so on, to interact with users on the Web and process their requests. Applets make the Web responsive, interactive, and fun to use. Applets are embedded in an HTML file. ***HTML (Hypertext Markup Language)*** is a simple scripting language for laying out documents, linking documents on the Internet, and bringing images, sound, and video alive on the Web. Today, you can use Java to develop rich Internet applications. A ***rich Internet application (RIA)*** is a Web application designed to deliver the same features and functions normally associated with desktop applications.

Java is now very popular for developing applications on Web servers. These applications process data, perform computations, and generate dynamic Web pages. Many commercial Websites are developed using Java on the backend.

Java is a versatile programming language: you can use it to develop applications for desktop computers, servers, and small handheld devices. The software for Android cell phones is developed using Java.

1.4 The Java Language Specification, API, JDK, and IDE

Computer languages have strict rules of usage. If you do not follow the rules when writing a program, the computer will not be able to understand it. The Java language specification and the Java API define the Java standards.

The ***Java language specification*** is a technical definition of the Java programming language's syntax and semantics.

The ***application program interface (API)***, also known as library, contains predefined classes and interfaces for developing Java programs. The API is still expanding.

Java is a full-fledged and powerful language that can be used in many ways. It comes in three editions:

- ***Java Standard Edition (Java SE)*** to develop client-side applications. The applications can run standalone or as applets running from a Web browser.
- ***Java Enterprise Edition (Java EE)*** to develop server-side applications, such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
- ***Java Micro Edition (Java ME)*** to develop applications for mobile devices, such as cell phones.

This course uses Java SE to introduce Java programming. Java SE is the foundation upon which all other Java technology is based. There are many versions of Java SE. Oracle releases each version with a ***Java Development Toolkit (JDK)***. For example, for Java SE 8, the Java Development Toolkit is called JDK 1.8 (also known as Java 8 or JDK 8).

The ***JDK*** consists of a set of separate programs, each invoked from a command line, for developing and testing Java programs. Instead of using the JDK, you can use a ***Java development tool*** (e.g. Eclipse, NetBeans and TextPad)—software that provides an ***integrated development environment (IDE)*** for developing Java programs quickly. Editing, compiling, building, debugging, and online help are integrated in one graphical user interface. You simply enter source code in one window or open an existing file in a window, and then click a button or menu item or press a function key to compile and run the program.

To sum up, Java syntax is defined in the ***Java language specification***, and the Java library is defined in the ***Java API***. The ***JDK*** is the software for developing and running Java programs. An ***IDE*** is an integrated development environment for rapidly developing programs.

1.5 A Simple Java Program

Let us begin with a simple Java program that displays the message ***Welcome to Java!*** on the ***console***. (The word console is an old computer term that refers to the text entry and display device of a computer. Console input means to receive input from the keyboard, and console output means to display output on the monitor.) The program is shown in Listing 1.1.

LISTING 1.1 Welcome.java

```
1  public class Welcome {
2    public static void main(String[] args) {
3      // Display message Welcome to Java! on the console
4      System.out.println("Welcome to Java!");
5    }
6  }
```

```
Welcome to Java!
```

Note that the ***<u>line numbers</u>*** are for reference purposes only; they are ***<u>not part</u>*** of the program. So, ***<u>do not</u>*** type line numbers in your program.

<u>Line 1</u> defines a ***class***. Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In this example, the class name is **Welcome**.

<u>Line 2</u> defines the ***main*** method. The program is executed from the **main** method. A class may contain several methods. The **main** method is the entry point where the program begins execution.

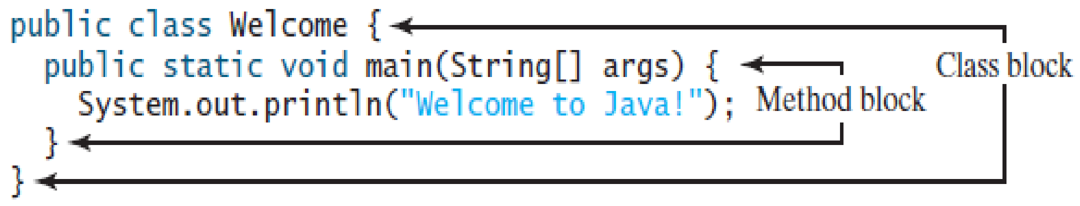
A ***method*** is a construct that contains statements. The **main** method in this program contains the **System.out.println** statement. This statement displays the string **Welcome to Java!** on the console (line 4). ***String*** is a programming term meaning a sequence of characters. A string must be enclosed in double quotation marks. Every statement in Java ends with a semicolon (**;**), known as the ***statement terminator***.

Reserved words*, or *keywords, have a specific meaning to the compiler and cannot be used for other purposes in the program. For example, when the compiler sees the word **class**, it understands that the word after **class** is the name for the class. Other reserved words in this program are **public**, **static**, and **void**.

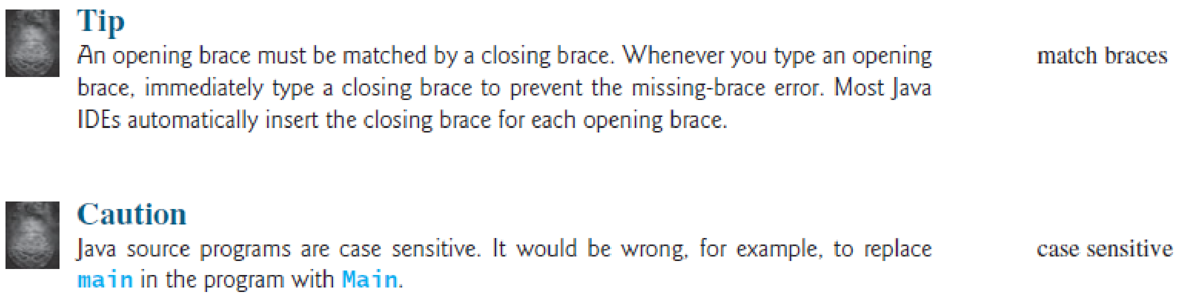
Line 3 is a ***comment*** that documents what the program is and how it is constructed. Comments help programmers to communicate and understand the program. They are not programming statements and thus are ignored by the compiler. In Java, comments are preceded by two slashes (//) on a line, called a ***line comment***, or enclosed between **/*** and ***/** on one or several lines, called a ***block comment* or *paragraph comment***. When the compiler sees **//**, it ignores all text after // on the same line. When it sees **/***, it scans for the next ***/** and ignores any text between **/*** and ***/**. Here are examples of comments:

```
// This application program displays Welcome to Java!
/* This application program displays Welcome to Java! */
/* This application program
displays Welcome to Java! */
```

A pair of curly braces in a program forms a ***block*** that groups the program's components. In Java, each block begins with an opening brace (**{**) and ends with a closing brace (**}**). Every class has a ***class block*** that groups the data and methods of the class. Similarly, every method has a ***method block*** that groups the statements in the method. Blocks can be ***nested***, meaning that one block can be placed within another, as shown in the following code.



```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```



Tip

An opening brace must be matched by a closing brace. Whenever you type an opening brace, immediately type a closing brace to prevent the missing-brace error. Most Java IDEs automatically insert the closing brace for each opening brace.

match braces

Caution

Java source programs are case sensitive. It would be wrong, for example, to replace `main` in the program with `Main`.

case sensitive

You have seen several ***special characters*** (e.g., { }, //, ;) in the program. They are used in almost every program. Table 1.2 summarizes their uses.

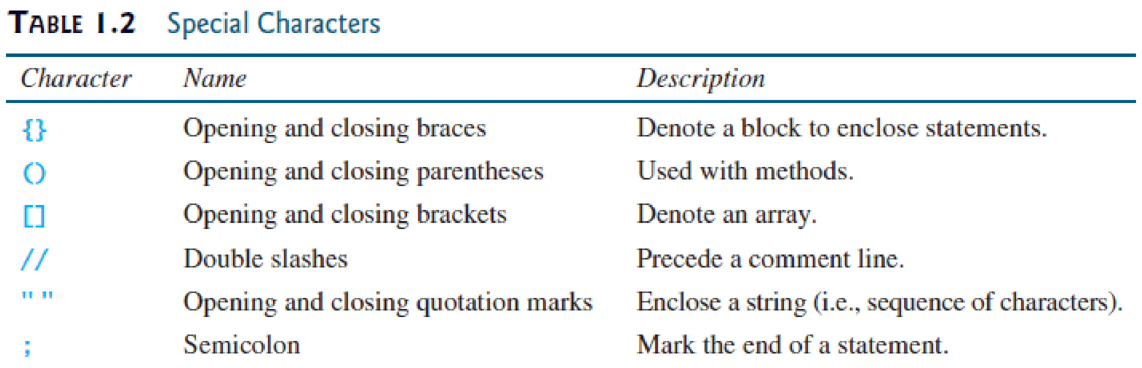


TABLE 1.2 Special Characters

Character	*Name*	*Description*
`{}`	Opening and closing braces	Denote a block to enclose statements.
`()`	Opening and closing parentheses	Used with methods.
`[]`	Opening and closing brackets	Denote an array.
`//`	Double slashes	Precede a comment line.
`" "`	Opening and closing quotation marks	Enclose a string (i.e., sequence of characters).
`;`	Semicolon	Mark the end of a statement.

The most ***common errors*** you will make as you learn to program will be syntax errors. Like any programming language, Java has its own syntax, and you need to write code that conforms to the ***syntax rules***. If your program violates a rule—for example, if the semicolon is missing, a brace is missing, a quotation mark is missing, or a word is misspelled—the Java compiler will report syntax errors. Try to compile the program with these errors and see what the compiler reports.

The program in Listing 1.1 displays one message. Once you understand the program, it is easy to extend it to display more messages. For example, you can rewrite the program to display three messages, as shown in Listing 1.2.

LISTING 1.2 WelcomeWithThreeMessages.java

```java
public class WelcomeWithThreeMessages {
  public static void main(String[] args) {
    System.out.println("Programming is fun!");
    System.out.println("Fundamentals First");
    System.out.println("Problem Driven");
  }
}
```

```
Programming is fun!
Fundamentals First
Problem Driven
```

Further, you can perform mathematical computations and display the result on the console. Listing 1.3 gives an example of evaluating:

$$\frac{10.5 + 2 \times 3}{45 - 3.5}.$$

LISTING 1.3 ComputeExpression.java

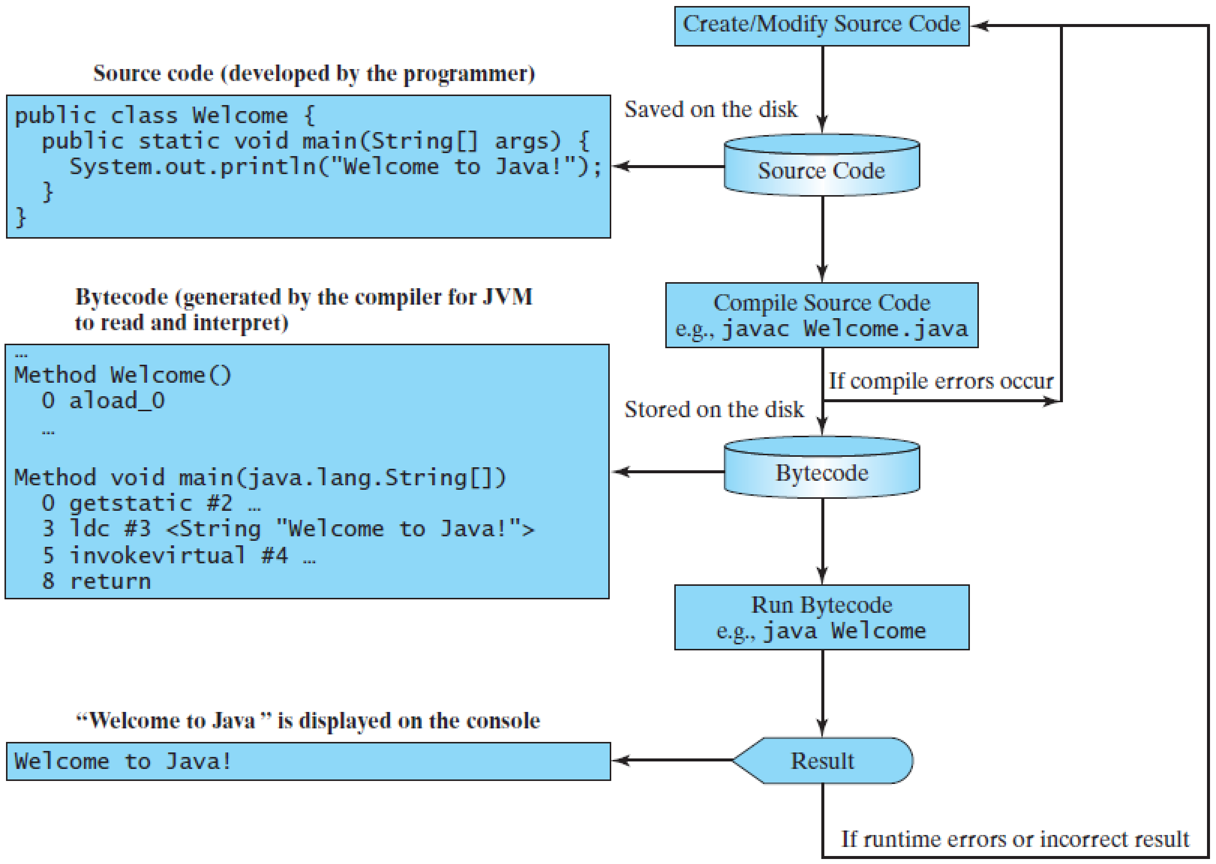
```java
public class ComputeExpression {
  public static void main(String[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

```
0.39759036144578314
```

1.6 Creating, Compiling, and Executing a Java Program

You save a Java program in a ***.java file*** and compile it into a ***.class file***. The ***.class file*** is executed by the ***Java Virtual Machine***.

You have to create your program and compile it before it can be executed. This process is repetitive, as shown in Figure 1.3. If your program has compile errors, you have to modify the program to fix them, and then recompile it. If your program has runtime errors or does not produce the correct result, you have to modify the program, recompile it, and execute it again.


Create/Modify Source Code
Saved on the disk
Source Code
Compile Source Code
e.g., javac Welcome.java
If compile errors occur
Stored on the disk
Bytecode
Run Bytecode
e.g., java Welcome
Result
If runtime errors or incorrect result
Source code (developed by the programmer)
Bytecode (generated by the compiler for JVM to read and interpret)
"Welcome to Java" is displayed on the console

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
…
Method Welcome()
  0 aload_0
  …

Method void main(java.lang.String[])
  0 getstatic #2 …
  3 ldc #3 <String "Welcome to Java!">
  5 invokevirtual #4 …
  8 return
```

```
Welcome to Java!
```

Figure 1.3: The Java program-development process consists of repeatedly creating/modifying source code, compiling and executing programs.

Note

The source file must end with the extension `.java` and must have the same exact name as the public class name. For example, the file for the source code in Listing 1.1 should be named **Welcome.java**, since the public class name is `Welcome`.

You can use any text editor or IDE to create and edit a Java source-code file. A ***Java compiler*** translates a ***Java source file*** into a ***Java bytecode file***.

If there are not any syntax errors, the compiler generates a ***bytecode file*** with a ***.class*** extension. Thus, the preceding command generates a file named ***Welcome.class***, as shown in Figure 1.4a.

The Java language is a high-level language, but Java bytecode is a low-level language. The bytecode is similar to machine instructions but is architecture neutral and can run on any platform that has a ***Java Virtual Machine (JVM)***, as shown in Figure 1.4b. Rather than a physical machine, the virtual machine is a program that interprets Java bytecode. This is one of Java's primary advantages: Java bytecode can run on a variety of hardware platforms and operating systems. Java source code is compiled into Java bytecode and Java bytecode is interpreted by the JVM. Your Java code may use the code in the Java library. The JVM executes your code along with the code in the library.

To execute a Java program is to run the program's bytecode. You can execute the bytecode on any platform with a JVM, which is an interpreter. It translates the individual instructions in the bytecode into the target machine language code one at a time rather than the whole program as a single unit. Each step is executed immediately after it is translated.

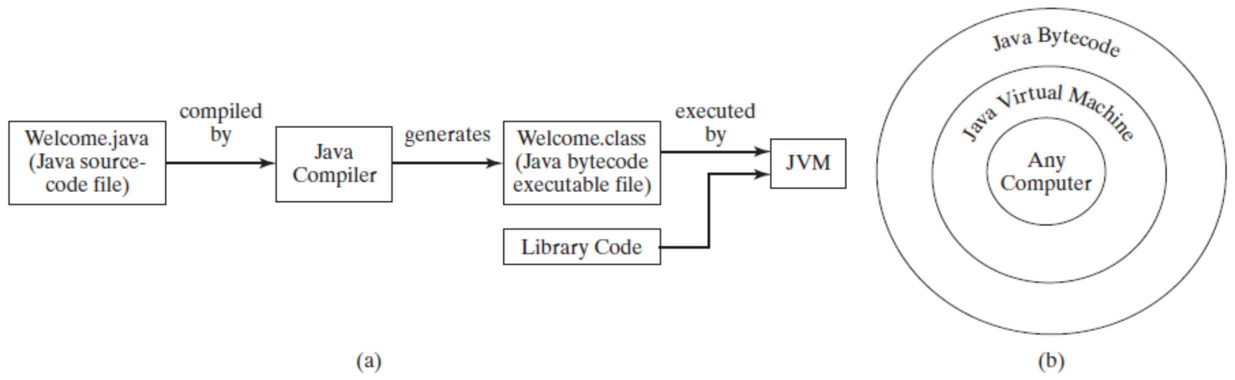


Figure 1.4: (a) Java source code is translated into bytecode. (b) Java bytecode can be executed on any computer with a Java Virtual Machine.

1.7 Programming Style and Documentation

Good programming style and proper documentation make a program easy to read and help programmers prevent errors. Programming style deals with what programs look like. A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read. Documentation is the body of explanatory remarks and comments pertaining to a program. Programming style and documentation are as important as coding. Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read.

1.7.1 Appropriate Comments and Comment Styles

Include a summary at the beginning of the program that explains what the program does, its key features, and any unique techniques it uses. In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read. It is important to make comments concise so that they do not crowd the program or make it difficult to read.

1.7.2 Proper Indentation and Spacing

A consistent indentation style makes programs clear and easy to read, debug, and maintain. Indentation is used to illustrate the structural relationships between a program's components or statements. Java can read the program even if all of the statements are on the same long line, but humans find it easier to read and maintain code that is aligned properly. Indent each subcomponent or statement at least two spaces more than the construct within which it is nested.

A single space should be added on both sides of a binary operator, as shown in the following statement:

```
System.out.println(3+4*4);            Bad style
System.out.println(3 + 4 * 4);        Good style
```

1.7.3 Block Styles

A block is a group of statements surrounded by braces. There are two popular styles, next-line style and end-of-line style, as shown below.

```java
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

Next-line style

```java
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

End-of-line style

The next-line style aligns braces vertically and makes programs easy to read, whereas the end-of-line style saves space and may help avoid some subtle programming errors. Both are acceptable block styles. The choice depends on personal or organizational preference. You should use a block style consistently—mixing styles is not recommended. This course uses the end-of-line style to be consistent with the Java API source code.

1.8 Programming Errors

Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.

1.8.1 Syntax Errors

Errors that are detected by the compiler are called syntax errors or compile errors. Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace.

These errors are usually easy to detect because the compiler tells you where they are and what caused them. For example, consider the program in Listing 1.4:

LISTING 1.4 ShowSyntaxErrors.java

```java
1 public class ShowSyntaxErrors {
2   public static main(String[] args) {
3     System.out.println("Welcome to Java);
4   }
5 }
```

Two errors can be noticed:

■ The keyword void is missing before main in line 2.

■ The string Welcome to Java should be closed with a closing quotation mark in line 3.

Since a single error will often display many lines of compile errors, it is a good practice to fix errors from the top line and work downward. Fixing errors that occur earlier in the program may also fix additional errors that occur later.

1.8.2 Runtime Errors

Runtime errors are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out. Input mistakes typically cause runtime errors. An input error occurs when the program is waiting for the user to enter a value, but the user enters a value that the program cannot handle. For instance, if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program.

Another example of runtime errors is division by zero. This happens when the divisor is zero for integer divisions. For instance, the program in Listing 1.5 would cause a runtime error.

LISTING 1.5 ShowRuntimeErrors.java

```java
public class ShowRuntimeErrors {
  public static void main(String[] args) {
    System.out.println(1 / 0);
  }
}
```

1.8.3 Logic Errors

Logic errors occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the program in Listing 1.6 to convert Celsius 35 degrees to a Fahrenheit degree:

LISTING 1.6 ShowLogicErrors.java

```
public class ShowLogicErrors {
  public static void main(String[] args) {
    System.out.println("Celsius 35 is Fahrenheit degree ");
    System.out.println((9 / 5) * 35 + 32);
  }
}
```

```
Celsius 35 is Fahrenheit degree
67
```

You will get Fahrenheit 67 degrees, which is wrong. It should be 95.0. In Java, the division for integers is the quotient—the fractional part is truncated—so in Java 9 / 5 is 1. To get the correct result, you need to use 9.0 / 5, which results in 1.8.

In general, syntax errors are easy to find and easy to correct because the compiler gives indications as to where the errors came from and why they are wrong. Runtime errors are not difficult to find, either, since the reasons and locations for the errors are displayed on the console when the program aborts. Finding logic errors, on the other hand, can be very challenging.

1.8.4 Common Errors

Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

Common Error 1: Missing Braces

The braces are used to denote a block in the program. Each opening brace must be matched by a closing brace. A common error is missing the closing brace. To avoid this error, type a closing brace whenever an opening brace is typed, as shown in the following example.

```
public class Welcome {

} ←——Type this closing brace right away to match the opening brace
```

If you use an IDE such as Eclipse, the IDE automatically inserts a closing brace for each opening brace typed.

Common Error 2: Missing Semicolons

Each statement ends with a statement terminator (;). Often, a new programmer forgets to place a statement terminator for the last statement in a block, as shown in the following example.

```
public static void main(String[] args) {
  System.out.println("Programming is fun!");
  System.out.println("Fundamentals First");
  System.out.println("Problem Driven")
}
```

Missing a semicolon

Common Error 3: Missing Quotation Marks

A string must be placed inside the quotation marks. Often, a new programmer forgets to place a quotation mark at the end of a string, as shown in the following example.

```
System.out.println("Problem Driven );
```

Missing a quotation mark

If you use an IDE such as Eclipse, the IDE automatically inserts a closing quotation mark for each opening quotation mark typed.

Common Error 4: Misspelling Names

Java is case sensitive. Misspelling names is a common error for new programmers. For example, the word main is misspelled as Main and String is misspelled as string in the following code.

```
public class Test {
  public static void Main(string[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

1.9 Developing Java Programs Using Eclipse

You can edit, compile, run, and debug Java Programs using Eclipse. This section gives the essential instructions to guide new users to create a project, create a class, and compile/run a class in Eclipse.

1.9.1 Creating a Java Project

Before creating Java programs in Eclipse, you need first to create a project to hold all files.

Here are the steps to create a Java project in Eclipse:

1. Choose File, New, Java Project to display the New Project wizard, as shown in Figure 1.5.
2. Type demo in the Project name field. As you type, the Location field is automatically set by default. You may customize the location for your project.
3. Make sure that you selected the options Use project folder as root for sources and class files so that the .java and .class files are in the same folder for easy access.
4. Click Finish to create the project, as shown in Figure 1.6.

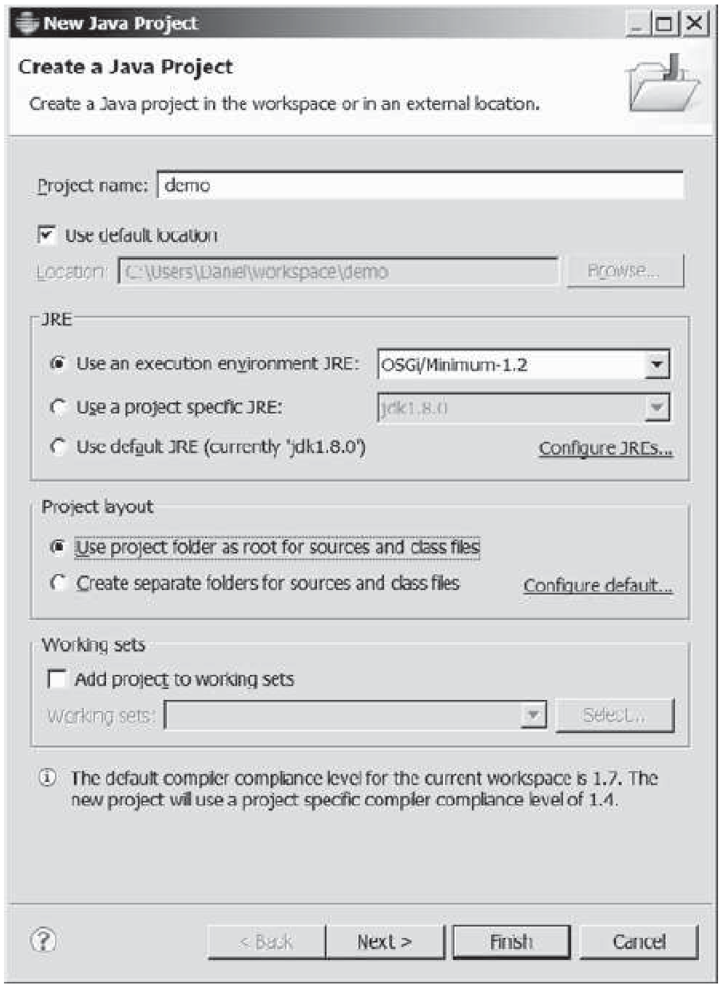

New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.
Project name: demo
☑ Use default location
Location: C:\Users\Daniel\workspace\demo
Browse...
JRE
● Use an execution environment JRE: OSGi/Minimum-1.2
○ Use a project specific JRE: jdk1.8.0
○ Use default JRE (currently 'jdk1.8.0')
Configure JREs...
Project layout
● Use project folder as root for sources and class files
○ Create separate folders for sources and class files
Configure default...
Working sets
☐ Add project to working sets
Working sets:
Select...
The default compiler compliance level for the current workspace is 1.7. The new project will use a project specific compiler compliance level of 1.4.
< Back
Next >
Finish
Cancel

Figure 1.5: The New Java Project dialog is for specifying a project name and properties.

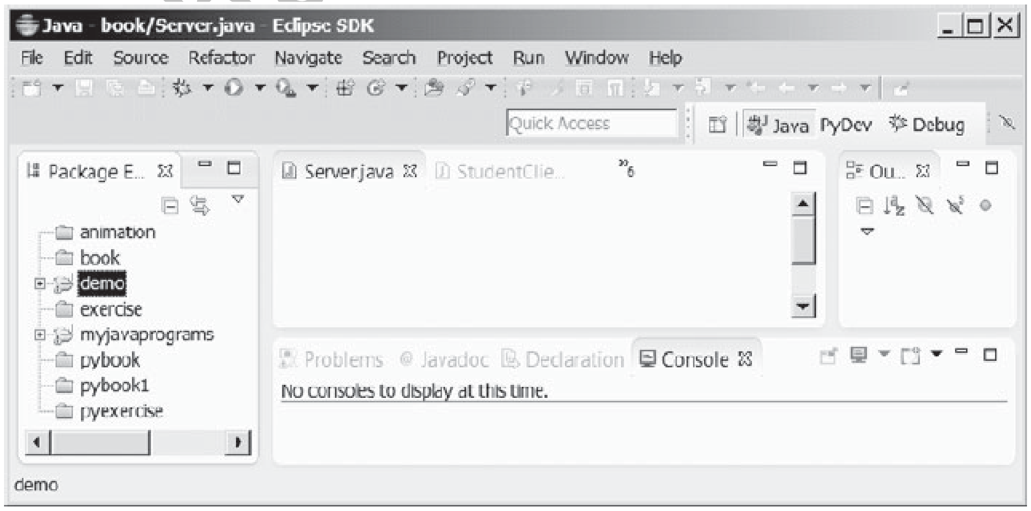
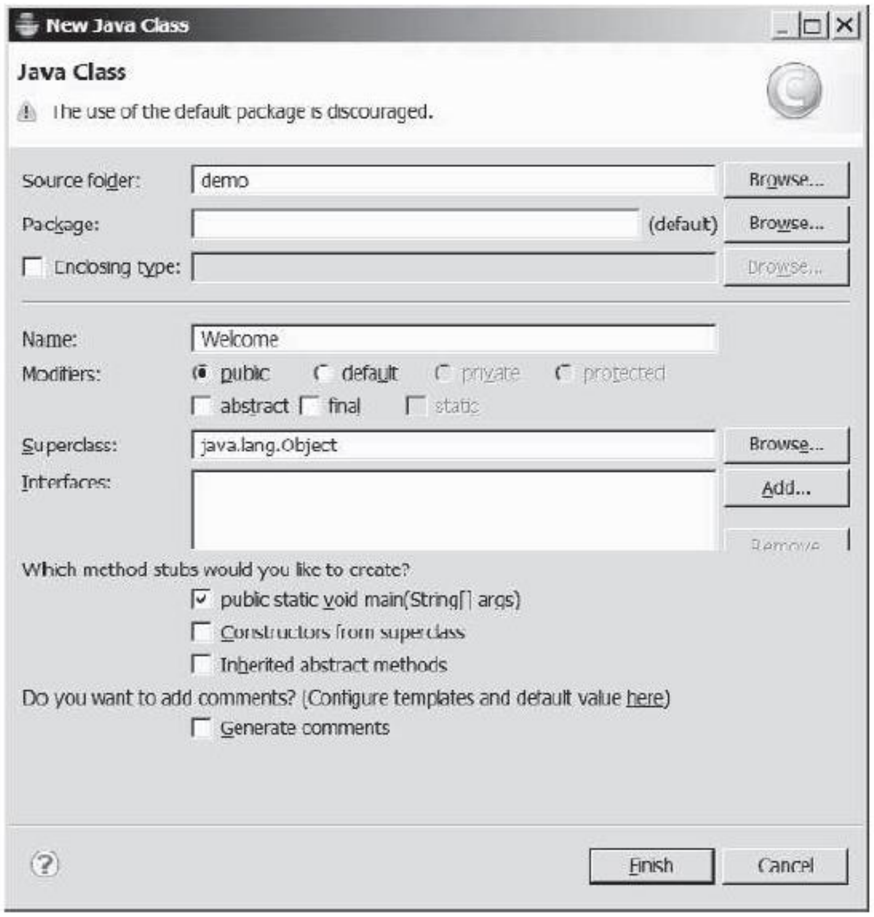


Figure 1.6: A New Java project named demo is created.

1.9.2 Creating a Java Class

After a project is created, you can create Java programs in the project using the following steps:

1. Choose File, New, Class to display the New Java Class wizard.
2. Type **Welcome** in the Name field.
3. Check the option ***public static void main(String[] args)***.
4. Click **Finish** to generate the template for the source code **Welcome.java**, as shown in Figure 1.7.


New Java Class

Java Class

The use of the default package is discouraged.

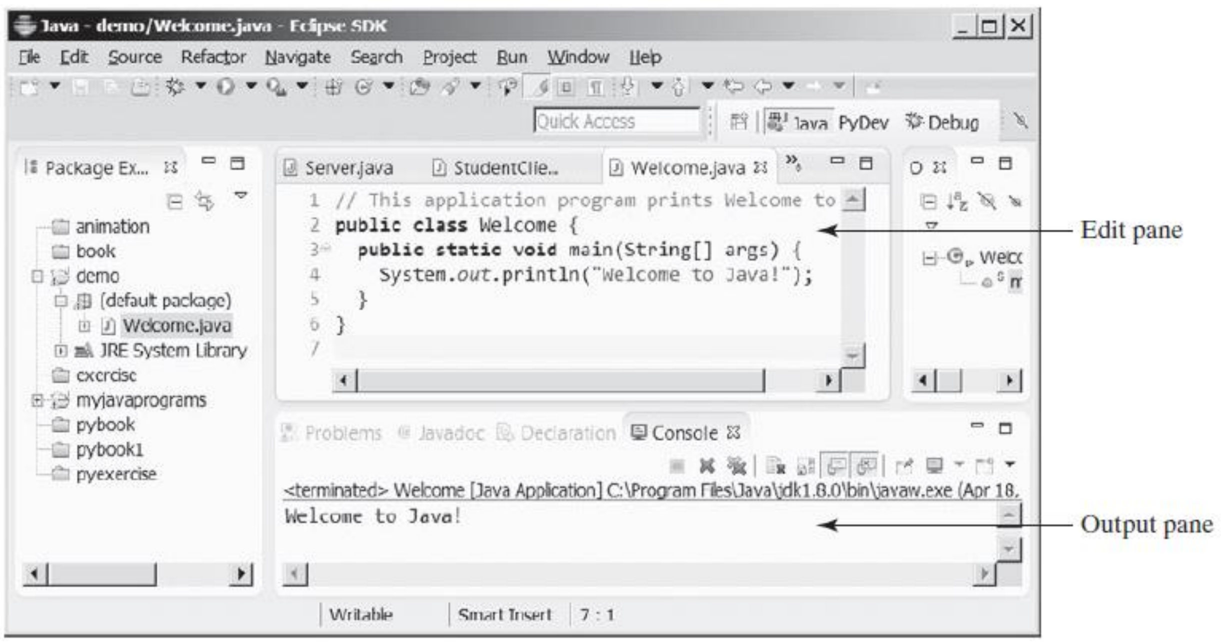
Source folder: demo
Package: (default)
[ ] Enclosing type:
Name: Welcome
Modifiers: [✓ public] default private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[✓] public static void main(String[] args)
[ ] Constructors from superclass
[ ] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

Figure 1.7: The New Java Class dialog box is used to create a new Java class.

1.9.3 Compiling and Running a Class

To run the program, right-click the class in the project to display a context menu. Choose Run, Java Application in the context menu to run the class. The output is displayed in the Console pane, as shown in Figure 1.8.



```
// This application program prints Welcome to
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

```
<terminated> Welcome [Java Application] C:\Program Files\Java\jdk1.8.0\bin\javaw.exe (Apr 18,
Welcome to Java!
```

Figure 1.8: You can edit a program and run it in Eclipse.