Introduction to Java

Third Year Students
Integrative Programming

Dr Athraa Juhi Jani

Introduction

- 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.

Introduction

- Software developers create software with the help of powerful tools called **programming languages.**

- 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.

Which Programming Language is the Best?

- 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.

The Key for programming

- 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.

1. 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.

1.1 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 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

1.2 Assembly Language

- 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.

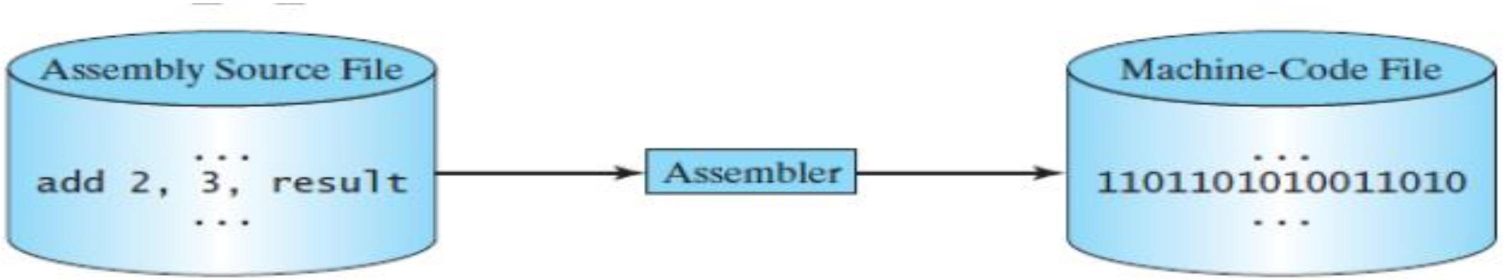


Figure 1.1: An assembler translates assembly-language instructions into machine code.

1.2 Assembly Language

- 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.

1.3 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;

1.3 High-Level Language

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

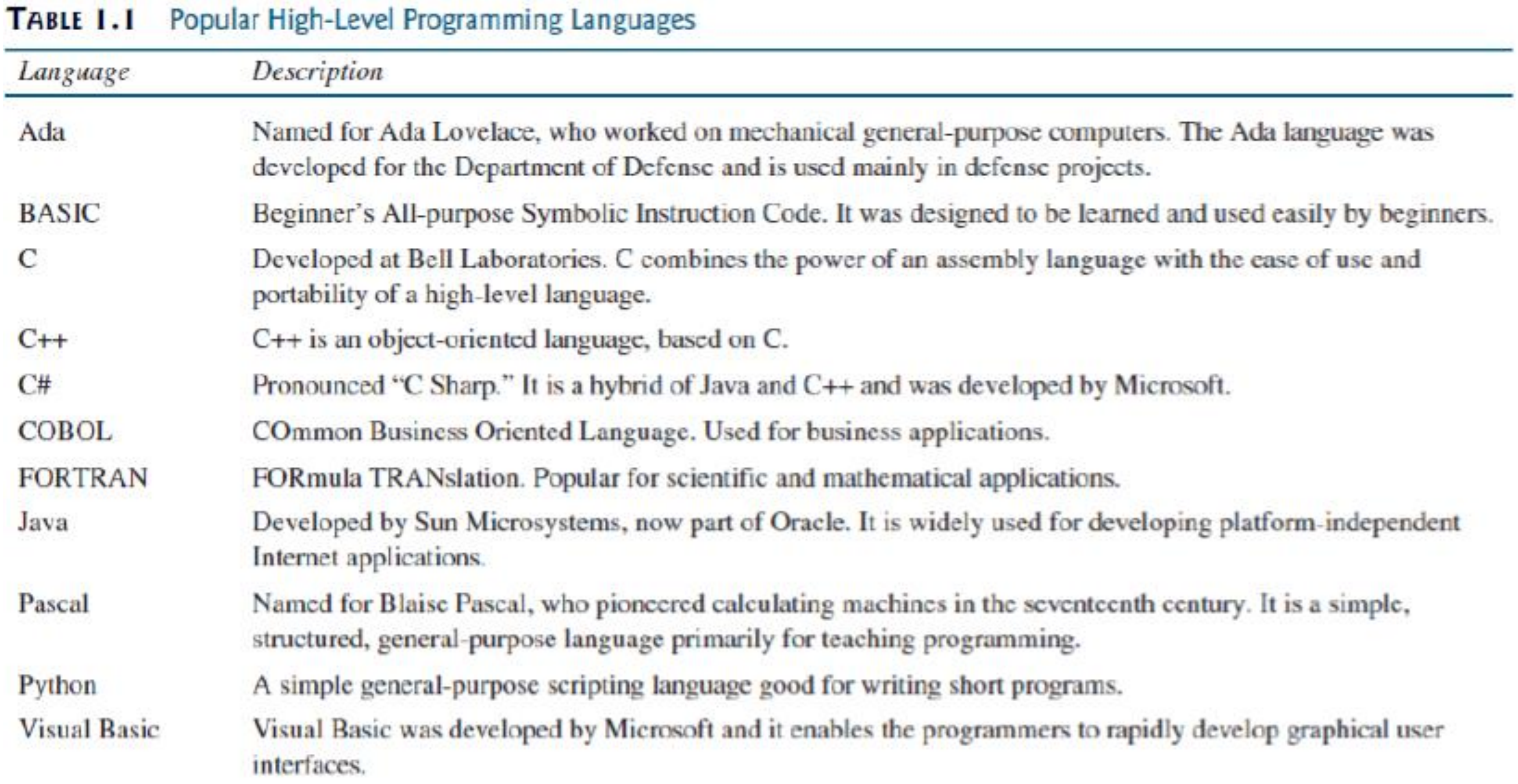


TABLE 1.1 Popular High-Level Programming Languages

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.

1.3 High-Level Language

- 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***.

1.3 High-Level Language

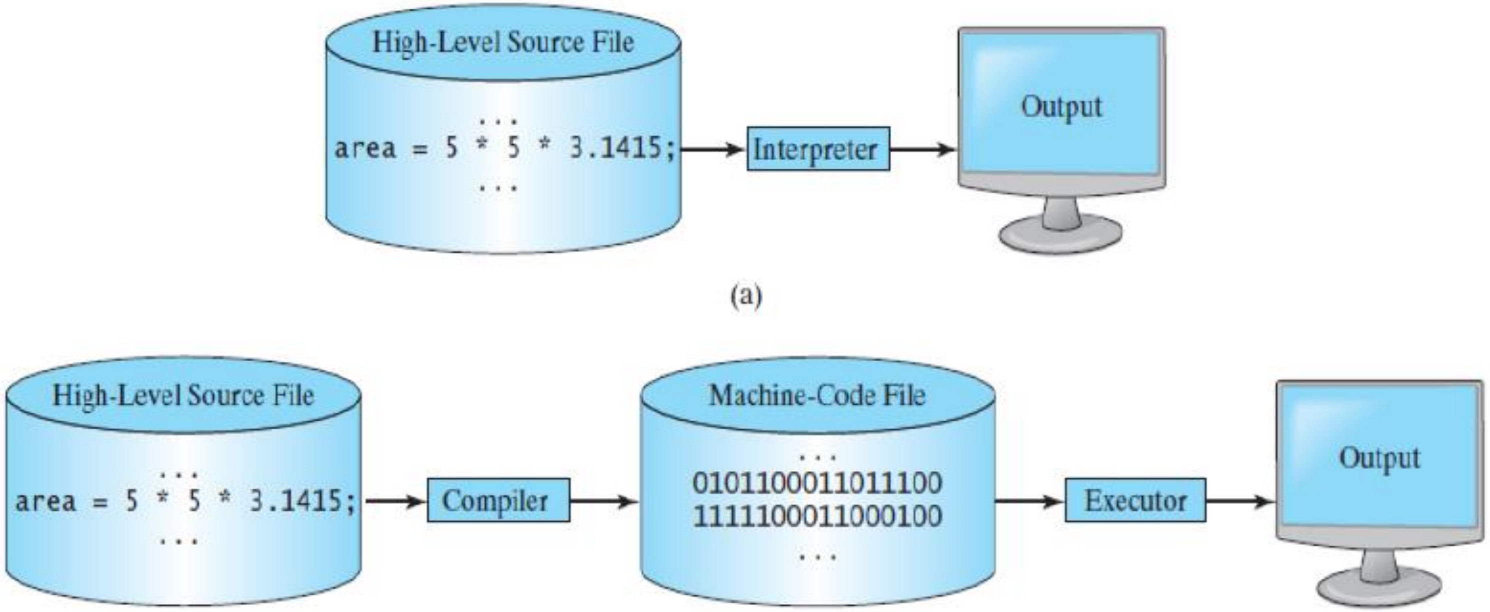
■ An ***interpreter*** reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away,

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.

1.3 High-Level Language

- **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.



(a)

(b)

2. 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.

- In 1995, Java 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.

Java as stated by its designer:

Java *is simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, and dynamic.*

Java

- 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.

Java

- 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.

Java and the World Wide Web

- 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 colourful World Wide Web and sophisticated Web browsers are the major reason for the Internet’s popularity.

Java and the World Wide Web

- 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.

Java and the World Wide 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.

Java and the World Wide Web

- 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.

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.

4. The Java Language Specification, API, JDK, and IDE

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.

4. The Java Language Specification, API, JDK, and IDE

- 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).

4. The Java Language Specification, API, JDK, and IDE

- 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.

4. The Java Language Specification, API, JDK, and IDE

To sum up:

- Java syntax is defined in the ***Java language specification.***

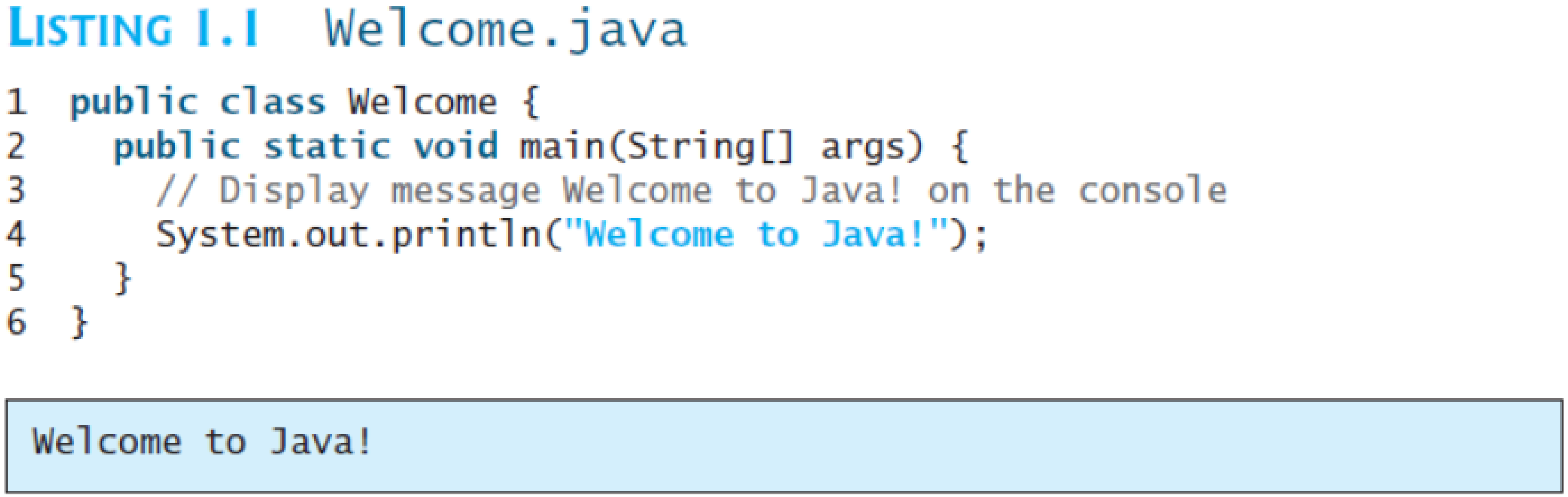
- Java library is defined in the ***Java API***.

- The ***JDK*** is the software for developing and running Java programs.

- An ***IDE*** is an integrated development environment for rapidly developing programs.

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

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

```
Welcome to Java!
```

5. A Simple Java Program

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

- **Line 1** 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**.

- **Line 2** 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.

5. A Simple Java Program

- 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***.

5. A Simple Java Program

- ***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**.

5. A Simple Java Program

- **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 ***/**.

5. A Simple Java Program

- 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! */
```

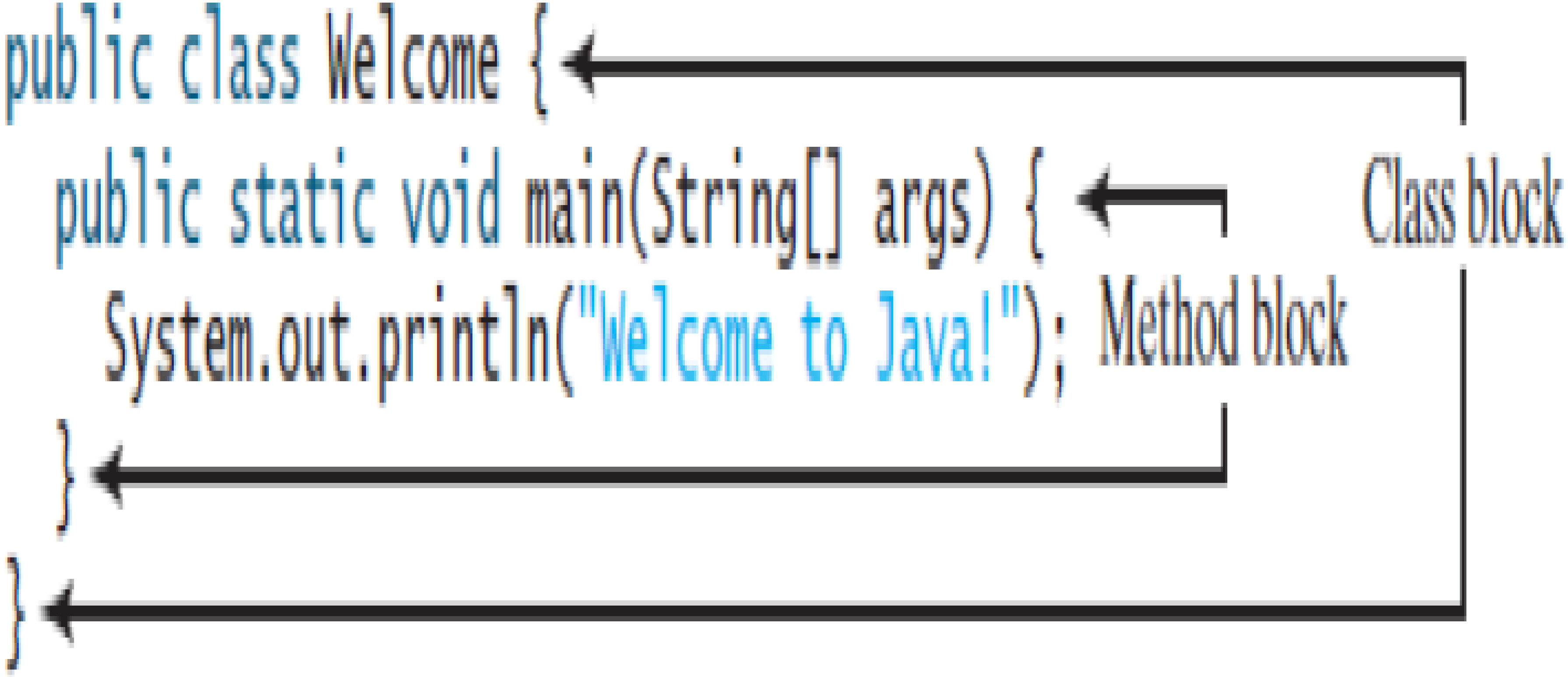
5. A Simple Java Program

- 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.

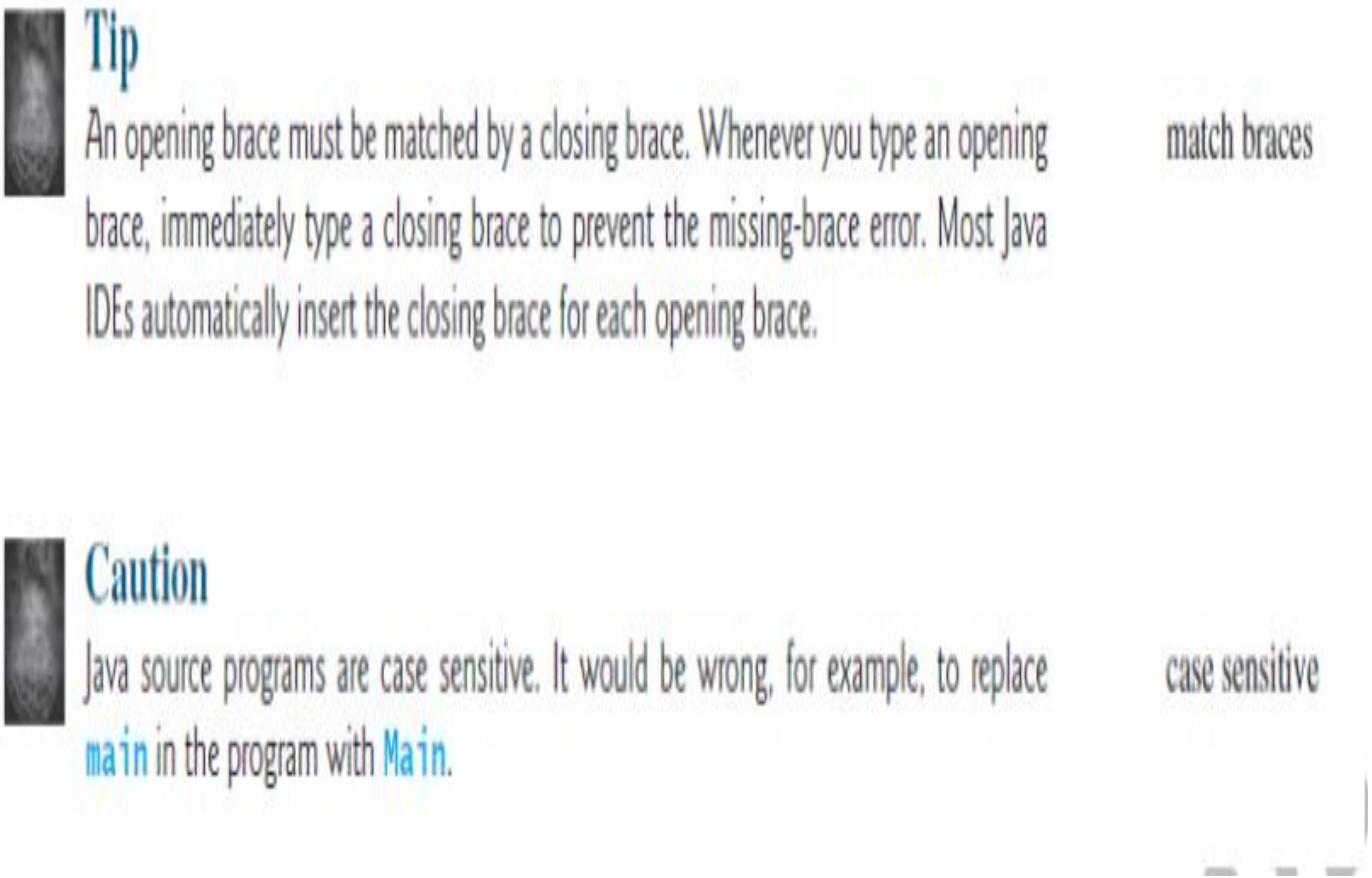
- Blocks can be ***nested***, meaning that one block can be placed within another, as shown in the following code.



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

Class block

Method block



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.

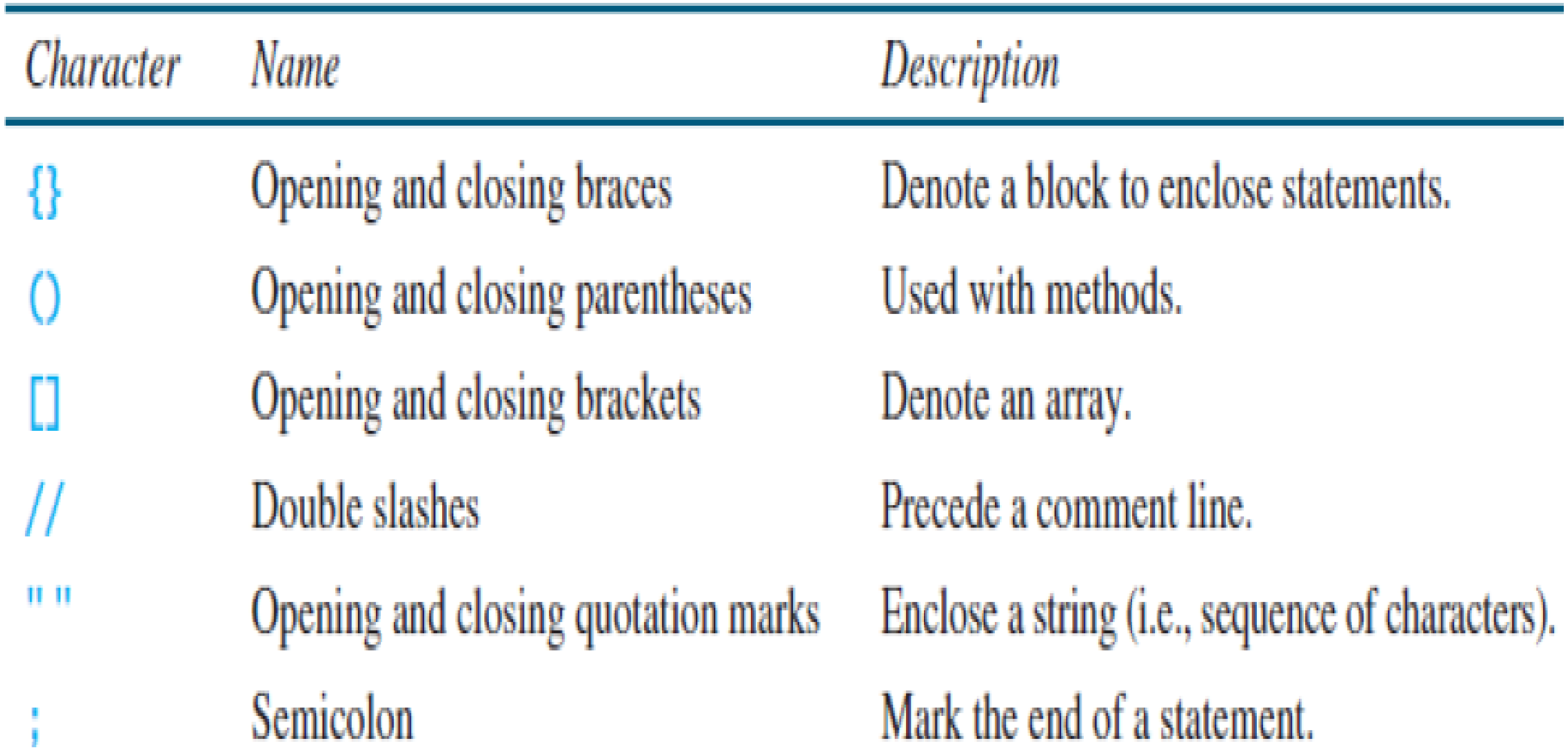
match braces

Caution

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

case sensitive

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.

Most Common Errors

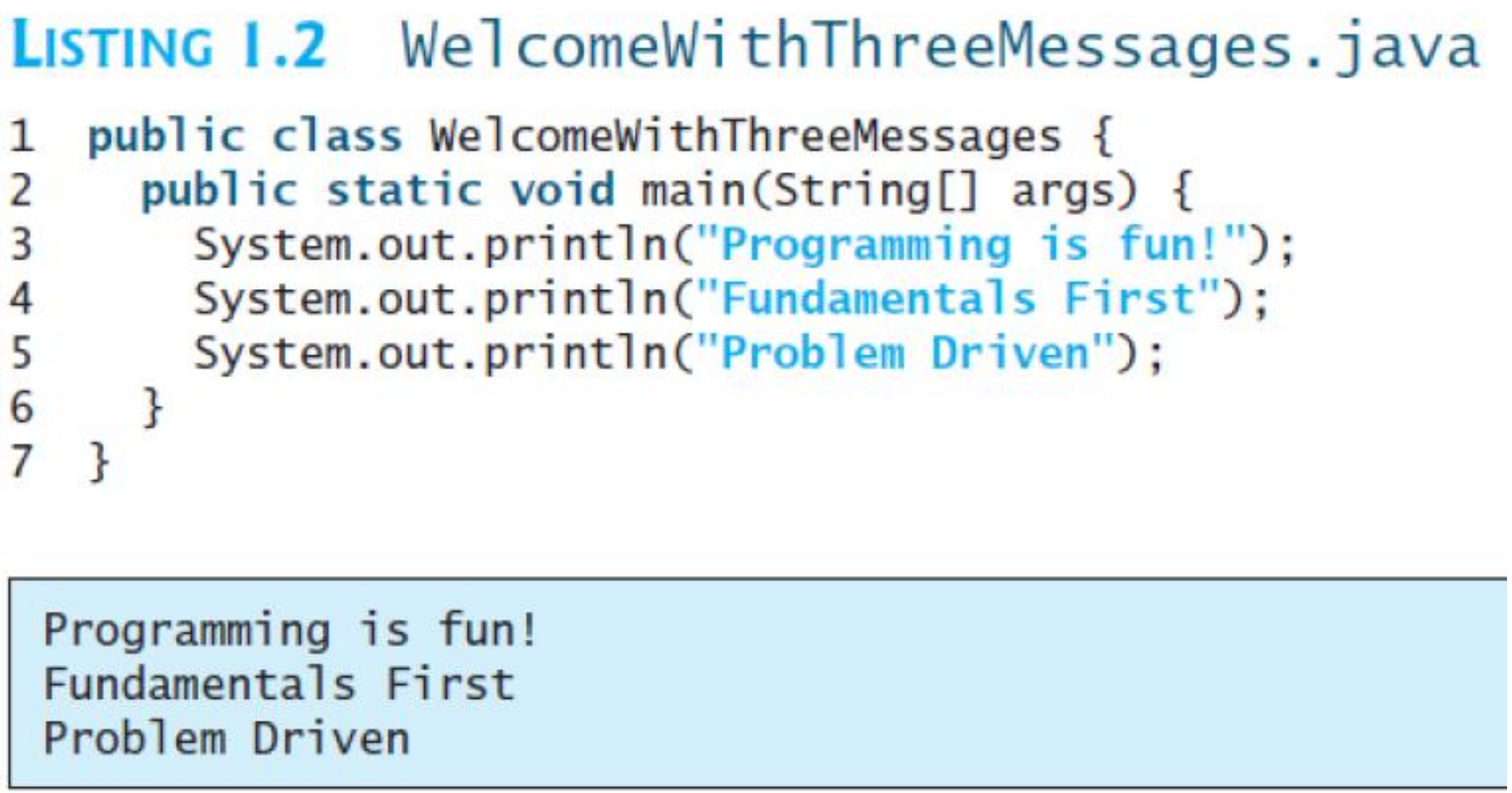
- The most ***common errors*** you will make as you learn to program will be syntax errors.

- 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.

Example 2

- You can rewrite the program to display three messages, as shown in Listing 1.2:



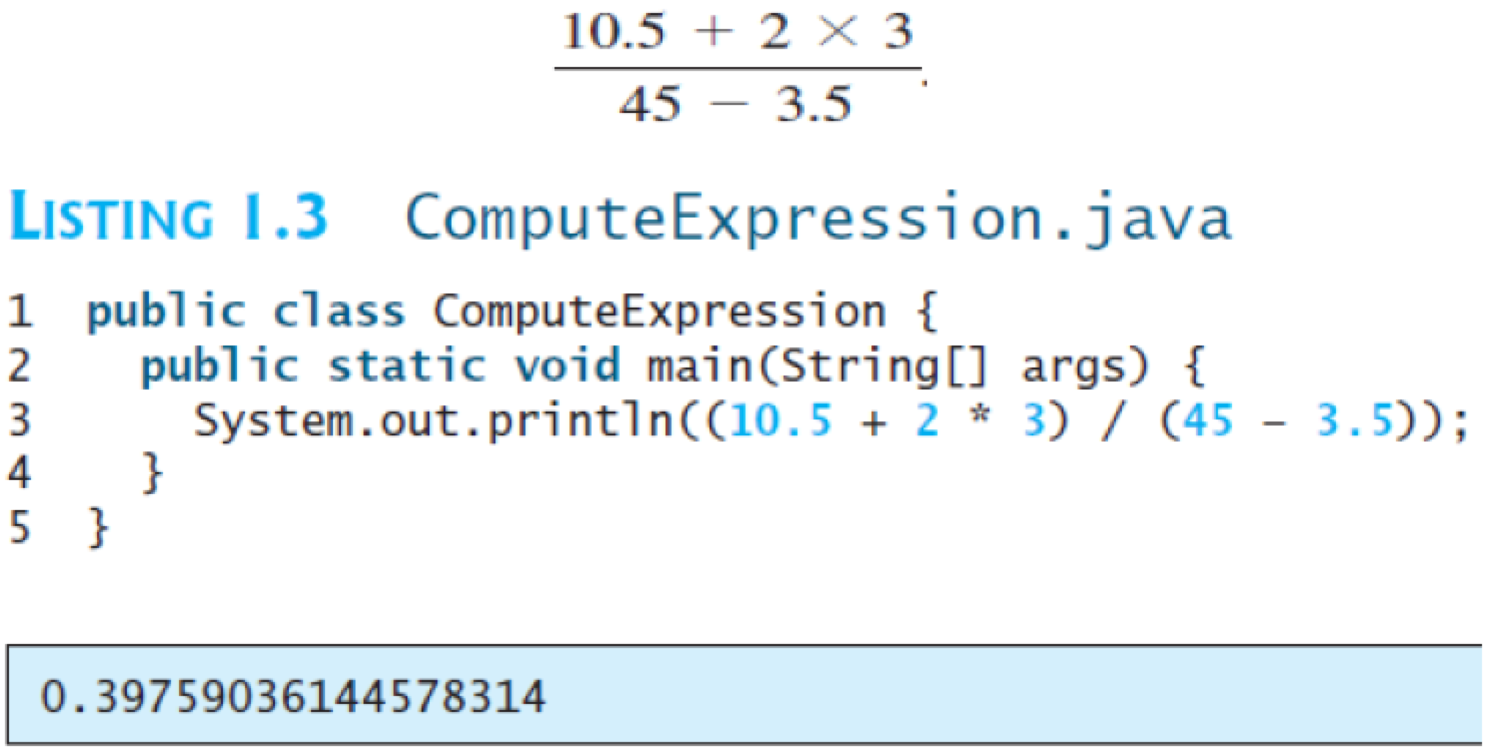
LISTING 1.2 WelcomeWithThreeMessages.java

```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
```

Example 3

- 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
```

6. Creating, Compiling, and Executing a Java Program

- You save a Java program in a ***.java file***.

- Then compile it into a ***.class file***.

- The ***.class file*** is executed by the ***Java Virtual Machine***.

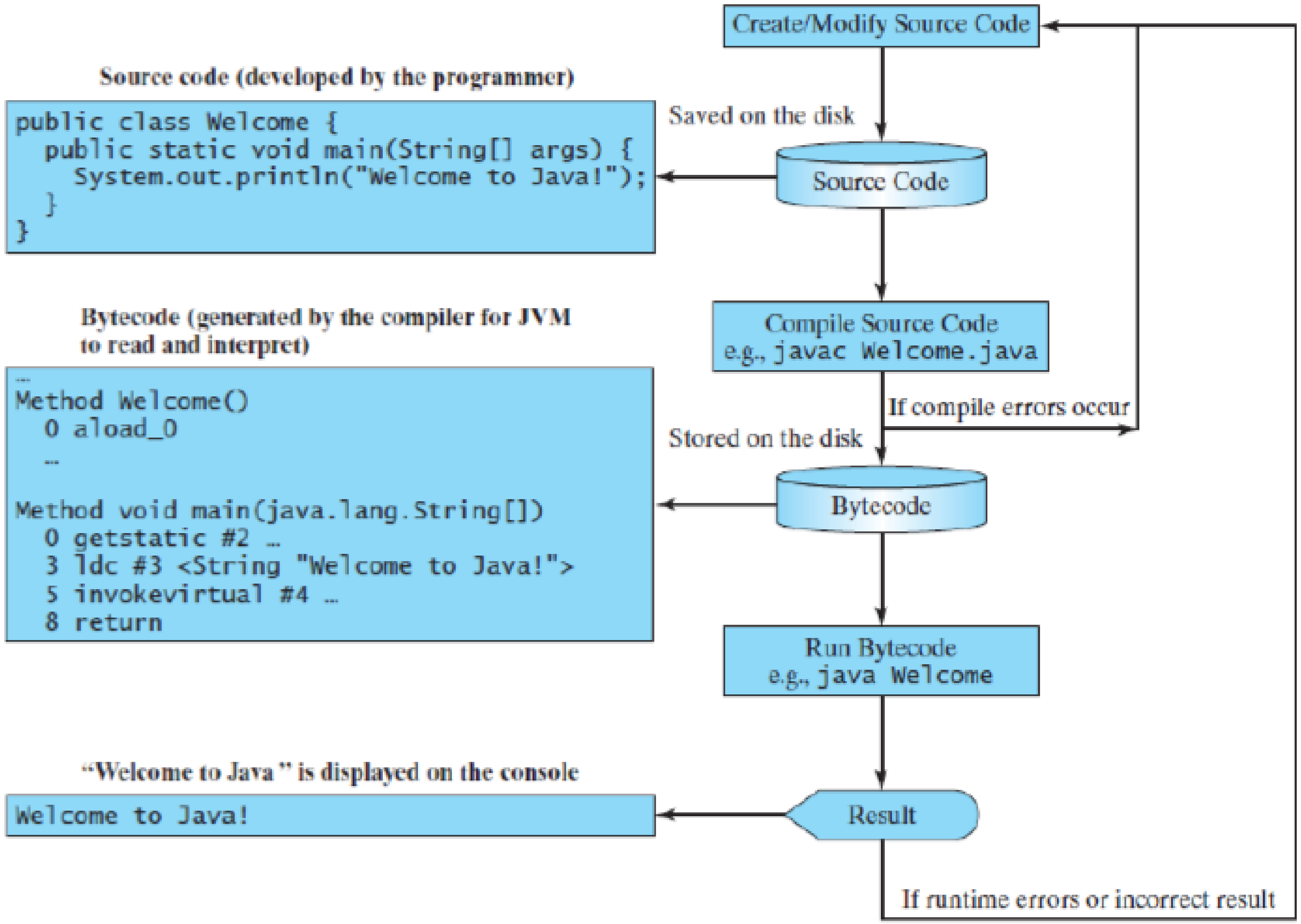
6. Creating, Compiling, and Executing a Java Program

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

```
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.

6. Creating, Compiling, and Executing a Java Program

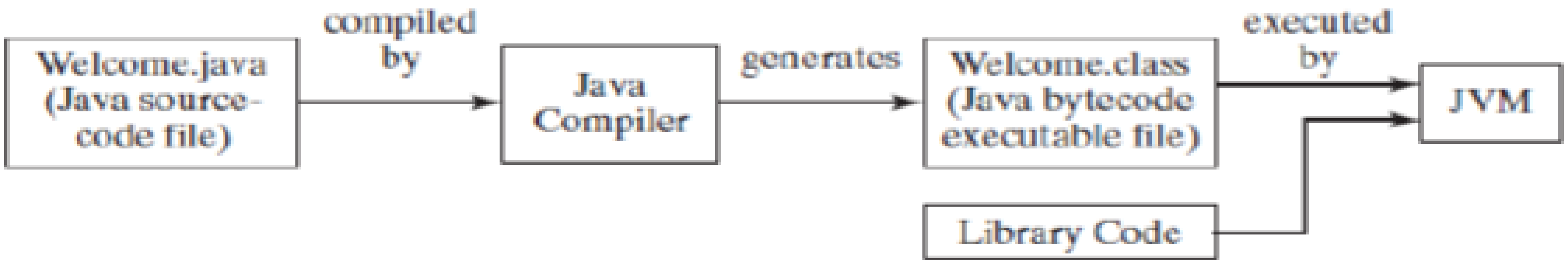
- You can use any text editor or IDE to create and edit a Java source-code file.
- A ***Java compiler*** translates a ***Java source file*** into a ***Java bytecode file***.

- 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.

6. Creating, Compiling, and Executing a Java Program

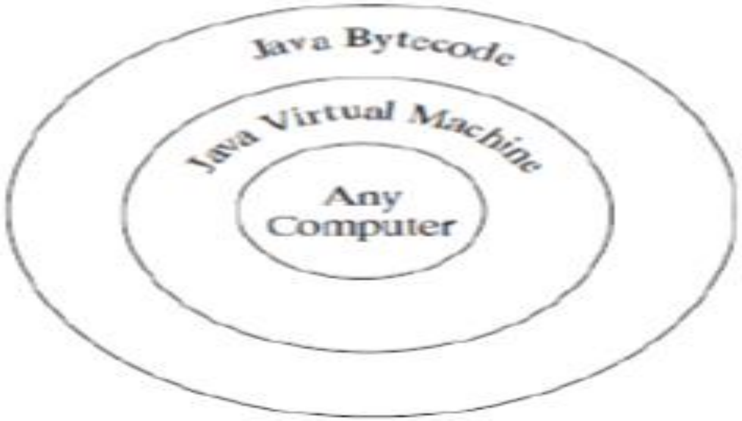
- Figure 1.4 (a) Java source code is translated into bytecode.


Welcome.java (Java source-code file)
compiled by
Java Compiler
generates
Welcome.class (Java bytecode executable file)
executed by
JVM
Library Code

(a)

6. Creating, Compiling, and Executing a Java Program

- 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.



(b)

- Figure 4.1(b) Java bytecode can be executed on any computer with a Java Virtual Machine.

6. Creating, Compiling, and Executing a Java Program

- 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.

6. Creating, Compiling, and Executing a Java Program

- 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.

Thank You