CHAPTER 1

Getting started

LEARNING OBJECTIVES

By the end of this chapter you will understand the following:

- The main activities involved in creating and maintaining programs.
- What source code is, and how we create it.
- What the basic components of a Java program are.
- How to compile the Java source code into an executable program.
- How to run a compiled Java program.

INTRODUCTION

This chapter illustrates some important programming concepts by way of an example. We will look at how to write, build and run programs. Later chapters will provide a more thorough explanation of the concepts introduced here.

1.1 Programming

A *program* is a set of instructions that can be executed on a computer to accomplish a specific task. Modern computers execute sequences of instructions quickly, accurately and reliably. Most people who use computers today are *end users*, i.e. they do not write their own programs. They mainly use off-the-shelf programs (*software*) for many purposes: word processors to write documents, drawing programs to make illustrations and spreadsheets to perform calculations.

The task of writing new programs is called *programming*. We write programs using the language constructs of a programming language. The program in this form is called *source code* and is stored in *text files*. The source code contains human readable descriptions of

the tasks the computer should perform, and usually needs to be translated before it can be executed on the computer.

FIGURE 1.1 Main activities in writing programs

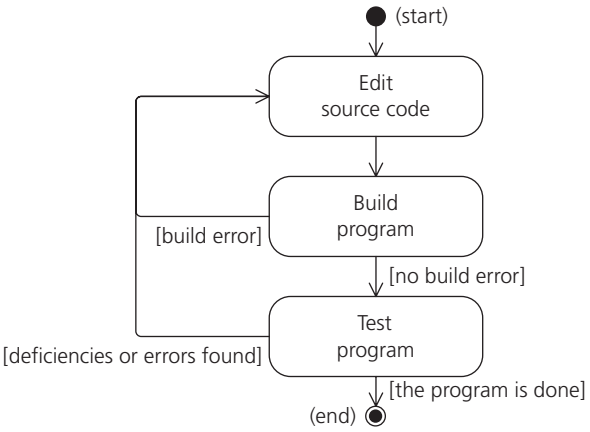


Figure 1.1 shows the main activities involved in creating programs. First we write the source code in text files, commonly called *source code files*. The source code describes exactly what tasks the computer should perform. Then we build an executable program from these source code files. At this point we need to correct any errors in the source code that prevent the program from being built. After building the program, we usually need to test it to make sure that it actually behaves the way we intended. If we detect any deficiencies or errors in the program, we need to go back and improve the source code.

Many other activities are involved when writing large programs, but the activities described in this chapter will be sufficient to get you started with programming.

There are many programming languages, but in this book we focus on the popular programming language, Java. Many of the concepts and programming techniques shown in this book are also applicable to other programming languages. The Java *compiler* is the program we use to build new programs from files containing Java source code.

Let's jump right in and take a quick look at some source code. Program 1.1 is the source code for a small program that calculates and reports the number of characters in a particular proverb. We will dissect this source code line by line later in the chapter, but for now it is sufficient to keep in mind that source code like this is stored in text files and is used to build executable programs. Before we examine the contents of Program 1.1 any further, we will look at how to enter the source code, build Java programs and run Java programs.

PROGRAM 1.1 The source code for a simple Java program

```
// (1) This source code file is called SimpleProgram.java
public class SimpleProgram {
  // Print a proverb, and the number of characters in the proverb.
  public static void main(String[] args) {                    // (2)
```

```
    System.out.println("A proverb:");                              // (3)

    String proverb = "Practice makes perfect!";                   // (4)
    System.out.println(proverb);                                  // (5)

    int characterCount = proverb.length();                        // (6)
    System.out.println("The proverb has " + characterCount + " characters.");
  }
}
```


1.2 Editing source code

The source code files contain only characters that constitute the actual text of the source code, and no text formatting information. Word processors such as Microsoft Word are not suited for writing source code, because their primary function is creating formatted documents. Many text editors for editing source code exist (e.g. JEdit), but any application (e.g. Microsoft Notepad) that can edit and save plain text can be used.

BEST PRACTICES

Choose a good editor and spend a few hours learning its features. In the long run, this effort will pay off handsomely in terms of productivity.

Source code file naming rules

To build an executable program, the compiler requires the source code files to be named according to specific rules. A Java source code file usually contains a language construct called a *class*. The name of this class is important when naming source code files. When saving the source code of a Java program, you need to make sure that:

- The name of the source code file is the same as the name of the class it contains, followed by the extension "`.java`".
- Use of lower and uppercase letters is the same as in the class name.

According to these rules, the source code file for Program 1.1 must be named "`SimpleProgram.java`", as it contains a class named "`SimpleProgram`".

If a source code file contains more than one class, only one class is designated as the *primary class*. The primary class is declared with the keyword `public`, and the source code file is named after this class.

The desktop environment of some operating systems hides the extension at the end of the file name by default. The compiler will refuse to compile source code files whose names

do not have the correct file extension. A common mistake is to store the source code file as either "<*name*>`.java.doc`" or "<*name*>`.java.txt`". Such file names will not be accepted by the compiler, even if the operating system shows them as simply "<*name*>`.java`". Microsoft Windows also provides alternative *short names* for files, e.g. "`MainCl~1.java`". Such file names are also not accepted by the compiler.

1.3 Development tools for Java

There are several ways to compile and run programs, depending on the development tools you are using. The following sections show how to compile and run programs using the standard software development tools for Java.

Sun Microsystems provides a package of tools called the Java Development Kit (JDK). This package contains the basic tools needed to compile and run programs written in Java. Appendix G provides more information about this development kit.

This book will show how to compile and run programs in a *terminal window*. Most operating systems have some sort of a terminal window with a command line where you can enter commands you want to run. Read through the documentation for your operating system if you need information on how to run commands from the command line in a terminal window.

1.4 Compiling Java programs

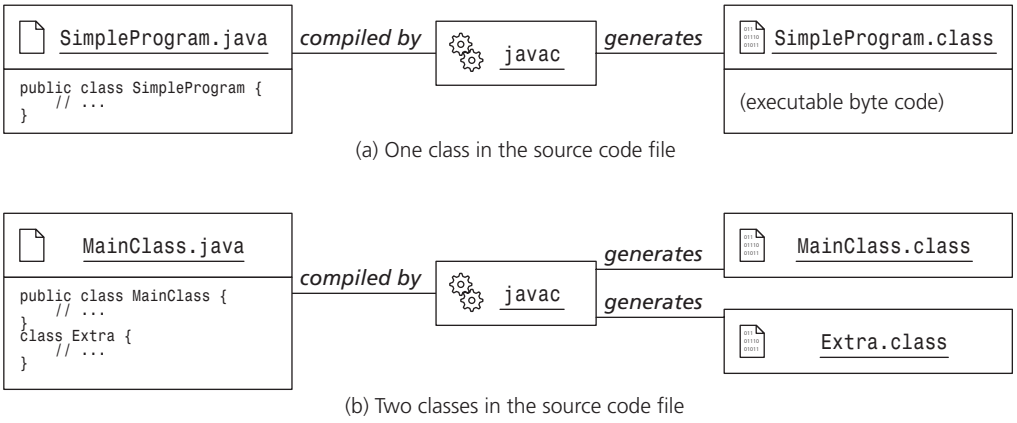
The syntax for running the Java compiler from the command line is:

```
> javac SimpleProgram.java
```

The `javac` compiler reads the source code from the specified Java source code file, and translates each class in the source code into a compiled form known as *Java byte code* (see Figure 1.2). The compiler creates files named "<*name*>`.class`" that contain the byte code for each class. Section 1.8 on page 11 explains byte code.

The compiler may detect errors in the source code when translating it to byte code. The compiler will report any errors and terminate the compilation. The errors must be corrected in the source code and the compiler run again to compile the program. With a little practice, you will be able to interpret the most common errors reported by the compiler, and identify the cause of the errors in the source code.

FIGURE 1.2 Compiling source code



(a) One class in the source code file

(b) Two classes in the source code file

1.5 Running Java programs

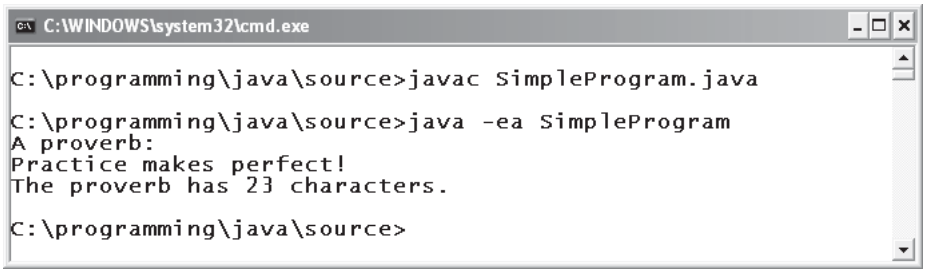
The command for running a compiled Java program is "`java`". This command should not be confused with the command "`javac`" used to compile the source code.

The syntax for running a Java program from the command line is:

```
> java -ea SimpleProgram
```

The `java` command starts the *Java virtual machine* (see Section 1.8 on page 11) that provides the runtime environment for executing the byte code of a Java program. The virtual machine starts the execution in the `main()` method of the class specified on the command line. More information on the `java` command is provided in Appendix G.

FIGURE 1.3 Terminal window showing program execution



The output from running the `SimpleProgram` example is shown in Figure 1.3. The length of a string includes all characters between the double quotes, including any spaces.

The `java` command requires that the exact name of the class containing the `main()` method is specified. Here are some guidelines in case there are problems running the `java` command:

- Specify the exact class name, without any "`.class`" or "`.java`" extensions.
- Check the use of upper and lowercase letters in the class name.
- Make sure that the source code has been compiled, so that there is a "`.class`" file for each class in the program.

1.6 The components of a program

This section introduces terminology that we will use in the next section to identify the language constructs used in the source code of Program 1.1. Later chapters will elaborate on the concepts mentioned here.

Operations

When creating a new program, we break down the given problem into smaller tasks that can be accomplished by performing one or more operations. Each operation is realised by a sequence of actions that describe in detail what the computer should do. These actions are written in the chosen programming language.

Programming with objects

The operations that need to be performed to complete a task often involve several types of objects. To start thinking in terms of objects and operations, let's consider the operations needed to make an omelette:

1. Open the refrigerator
2. Take out an egg carton
3. Open the egg carton
4. Take out two eggs
5. Close the egg carton
6. Place the egg carton back in the refrigerator
7. Close the refrigerator
8. Turn on the stove
9. ...

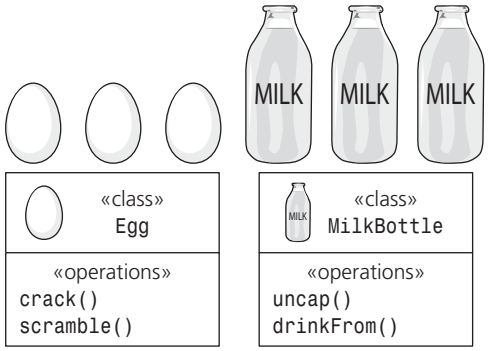
In this context we can consider the refrigerator, the egg carton, the eggs and the stove to be different types of objects. The operations that are performed are inherently connected to these objects. The type of an object determines the operations that can be performed on it. For example, it is possible to open an egg carton, but it is not possible to open a frying pan.

Object-based programming

Object-based programming (OBP) involves describing tasks in terms of operations that are executed on objects. What the objects represent depends on what the program is trying to accomplish. A program to keep track of library loans, for example, might use objects to represent tangible items such as books, journals, audio tapes, etc., and also objects to represent non-tangible concepts such as the lending date and personal information about library users.

Programs usually have more than one object of the same type. Back in the kitchen we have several objects representing bottles of milk. Each bottle can be handled independently, but they all have certain common properties, e.g. they can be uncapped and drunk from. Objects that share a set of common properties can be considered to belong to the same *class* of objects. Every egg object is a concrete *instance* of a particular `Egg` class, just as every milk bottle is a concrete instance of a particular `MilkBottle` class. This distinction is illustrated in Figure 1.4.

FIGURE 1.4 Classes and instances



A class is defined by describing the properties specific to the objects of the class, and the operations that can be performed on these objects. A program can consist of user-defined classes as well as classes from other sources. The Java language comes bundled with a large collection of ready-to-use classes called the *Java standard library*. These classes contain program code that can readily be used for solving a wide range of programming problems.

1.7 The Java programming language

Classes and methods

The language constructs of the Java programming language have a prescribed structure. We call this structure the *syntax* of the language. A *class declaration* is a language construct to define classes. Operations in a class are defined by *method declarations* containing sequences of *statements* describing the actions that need to be performed.

Figure 1.5 on page 8 shows the language constructs used in the source code of Program 1.1. The program has a class declaration that specifies a class called `SimpleProgram`, and this class has a method called `main`.

Comments and indentation

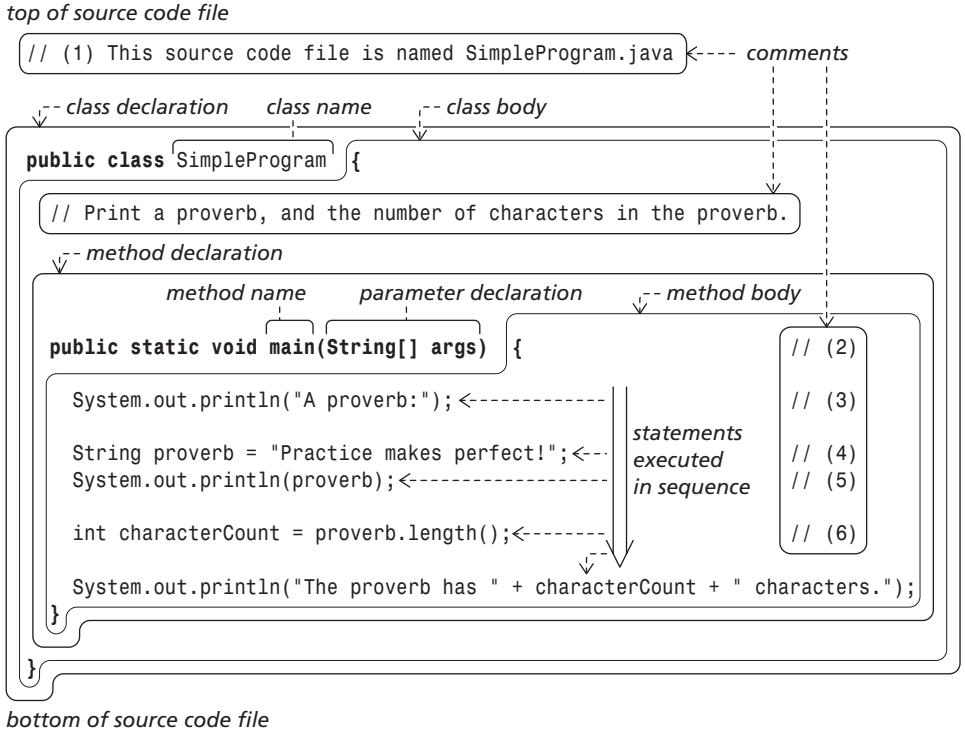
The first line of source code in Program 1.1 is:

```
// (1) This source code file is called SimpleProgram.java
```

This is a source code comment, and is ignored by the compiler. We write comments in the source code to aid programmers in understanding the inner workings of the program. Everything following the characters `//` on a line of source code is a comment.

When this book wants to draw attention to specific lines of code from the code examples, the code line have a comment of the form `// (`*n*`)`, and the text refers to the lines using markers such as (*n*), where *n* is an integer.

FIGURE 1.5 Class and method declarations



The compiler completely ignores all comments, as well as some special characters such as spaces and end-of-line characters. As far as the compiler is concerned, the `main()` method declaration starting at (2) could just as well have been written like this:

```
public static void main(String[]args){System.out.println("A proverb:");
String proverb="Practice makes perfect!";System.out.println(proverb);int
characterCount=proverb.length();System.out.println("The proverb has "+
characterCount+" characters.");}
```

The layout of the source code has no bearing on program execution. It is a common practice to write one statement per line, and use indentation from the left margin to show

the nesting of the language constructs. This book uses two spaces for each indentation step to conserve the available page width.

BEST PRACTICES

In Java it is customary to use four spaces for each indentation step, and we recommend that you use do the same when you write source code yourself. Proper indentation makes the source code easier to read and modify.

Program entry point

Line (2) of Program 1.1 is the start of a method declaration that defines a method called `main`:

```
public static void main(String[] args) {                    // (2)
```

The method being defined here has the *parameter declaration* `String[] args`, as shown in Figure 1.5. A parameter declaration specifies the information a method is given when it is executed. In this particular program we are not interested in giving any information to the method, so we will just ignore the parameter declaration for now.

Throughout this book we follow the convention of writing "`()`" at the end of names to indicate that the names refer to methods. This does not necessarily mean that the methods do not have any parameters.

For a Java program to be executable, it must define exactly one `main()` method that is declared with `public static void main(String[] args) { ... }`, which is the method where program execution will start. The statements within the method body obviously vary from program to program, reflecting the task each program is trying to accomplish. The significance of using a particular method declaration to designate the *entry point* where execution of the program starts becomes clear when writing programs that have more than one method declaration.

Each line in the `main()` method body in Program 1.1 is a separate statement. When the program starts to run, the statements in the `main()` method are executed one by one, starting with the first statement. Figure 1.6 on page 10 shows the statements annotated with the language constructs they use.

BEST PRACTICES

For very small programs it is normal to have only one source code file, and to define the primary class in the file that contains the `main()` method. For larger programs, it is more practical to split the source code into several files. In such cases, it is customary to declare only one class in each file.

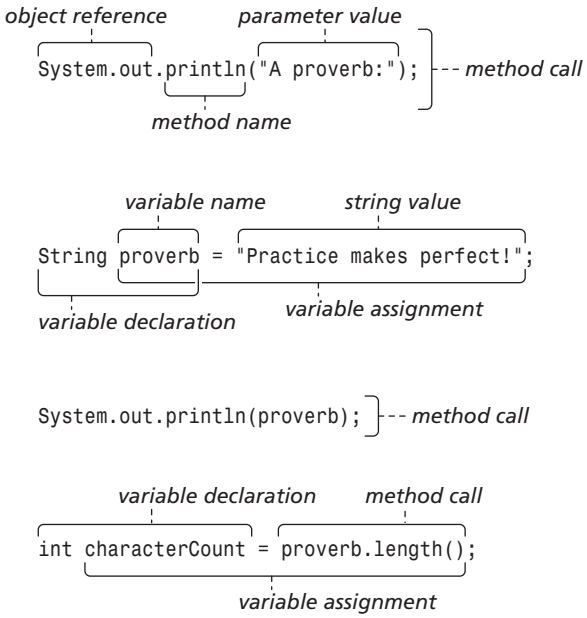
Method calls

The first statement that is executed when the program starts is (3):

```
System.out.println("A proverb:");                    // (3)
```

This statement *calls* the method `println` in the object known as `System.out`. The method `println()` in this object is responsible for printing a line of text. The statement calling the `println()` method passes it the sequence of characters to be printed. Such a sequence of characters is called a *string*. The statement at (3) passes the string `"A proverb:"` as a parameter to the `println()` method. Figure 1.3 on page 5 shows a *terminal window* where Program 1.1 has been compiled and run. As we can see, the strings given in the `println()` method calls are printed to the terminal window.

FIGURE 1.6 Syntax of statements in Program 1.1



Variables

Variables are named locations in the computer's internal storage (memory) where values can be held during program execution. Methods often use variables to hold intermediate results. The `main()` method of Program 1.1 uses two such variables (`proverb` and `characterCount`) to store values it subsequently uses in later statements. Numeric values are very common, but as we will see later, there are other *types* of values. When we *assign* a value to a variable, we store the value in the variable so that the value can later be used by referring to the variable.

The statement at (4) declares a variable called `proverb`, and assigns the string `"Practice makes perfect!"` to it:

```
String proverb = "Practice makes perfect!";          // (4)
```

The program can now refer to this text string using the name `proverb`. The statement at (5) prints the string referred by the variable `proverb` to the terminal window:

```
System.out.println(proverb);                                    // (5)
```

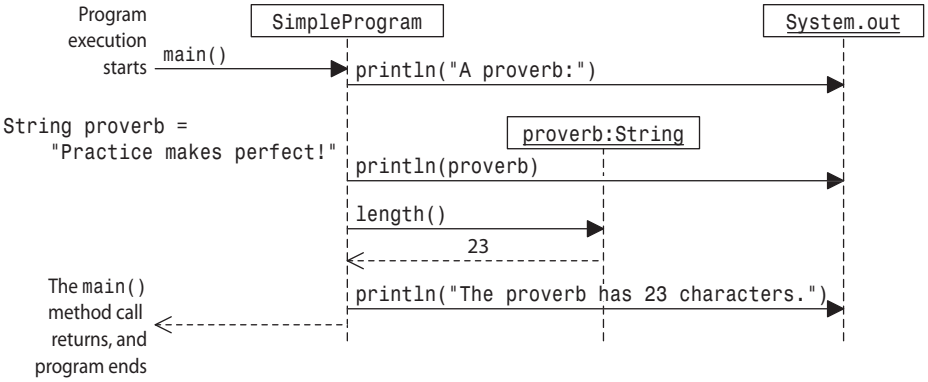
A string of text is an object in Java, and statement (6) calls the method `length()` on the string object to find out how many characters there are in the string:

```
int characterCount = proverb.length();                          // (6)
```

The character count returned by the `length()` method is stored in the variable `characterCount` that is declared in the same line of code. The class of the `proverb` string objects is `String`, which is one of the many classes provided by the Java standard library. Later chapters will explore many classes from this library, including the `String` class.

When a method is called, as in statement (3), the body of the called method will be executed before program execution continues past the method call statement. Figure 1.7 shows the method calls that are executed when Program 1.1 is run. The *flow of execution* in the program can be traced by reading the method calls in Figure 1.7 from top to bottom.

FIGURE 1.7 Sequence of method calls during program execution



1.8 The Java Virtual Machine

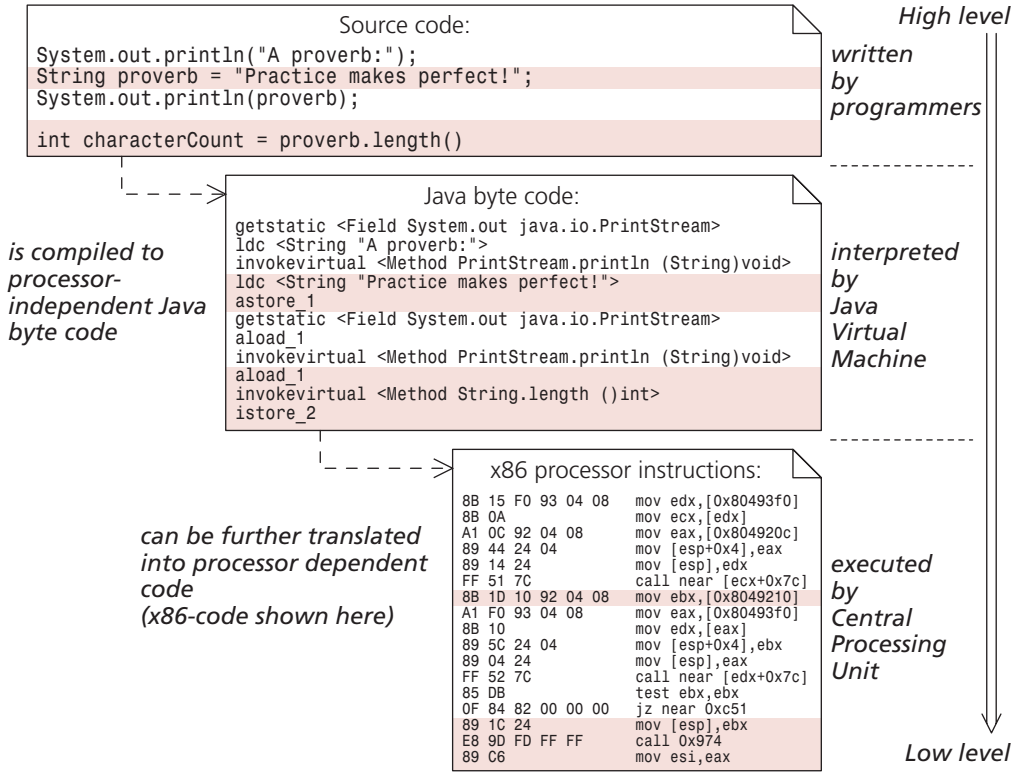
The Java programming language provides a rich set of language constructs that allow us to express program behaviour in a way that is natural for humans. On the other hand, Java byte code is a small set of basic instructions suited for execution by machines. The Java language is considered a *high-level language*, while Java byte code is considered a *low-level language*.

During execution, the byte code is interpreted by a virtual machine (the *Java Virtual Machine*, *JVM*). Rather than being a physical machine, the virtual machine is a program that interprets byte code instructions, in the same way that a central processing unit (CPU) in a computer would execute machine code instructions.

Figure 1.8 shows how a few lines of Java source code are translated to an executable form. The Java compiler translates the source code to Java byte code. The byte code can now be interpreted by a Java Virtual Machine installed on any computer. We call this *platform independence*, since it is not limited to a particular type of computer. Some virtual machines interpret the byte code directly, while others recompile it to a third form called *machine code* during execution. Machine code is at an even lower level than byte code, and will always be created specifically for the instruction set of the CPU in the computer. This means that machine code can only be executed on the specific type of computer for which it has been compiled. The machine code shown in Figure 1.8 is specific to the x86-processor series.

Some programming languages do not have an intermediate byte code representation, and lock the program to a specific platform as soon as the source code is compiled. Such programming languages allow the programmer to obtain speed improvements by programming directly at the machine code level, but this also increases the risk of programming errors. Most Java programmers will never need to examine the byte code or the machine code representation of programs.

FIGURE 1.8 Program code at several levels



1.9 Review questions

1. How many comments does Program 1.1 have?

2. A computer has a _____ that executes low-level platform-specific instructions, and makes the computer work.

3. _____ is a high-level description of the tasks the computer should perform, which are written in a high-level _____ and stored in text files.

4. Which of these components are software?

- **a** A compiler.
- **b** A keyboard.
- **c** A virtual machine.
- **d** The result of compiled source code.
- **e** A CPU.

5. We specify the set of common properties we want a group of objects to share by defining a _____.

6. All Java programs have a _____ called _____, where the execution of the program starts.

7. Which statement best describes Object-Based Programming (OBP)?

- **a** Making an omelette.
- **b** Defining classes of objects that have common properties and operations that can be performed on these objects.
- **c** Compiling source code to Java byte code.
- **d** Translating programs to processor-dependent code.

8. Program 1.1 has a class called _____ and a method named _____.

9. The command _____ can be executed on the command line to compile a source code file called `TestProgram.java`.

10. The command _____ can be executed on the command line to run a Java program consisting of a primary class called `TestProgram`.

11. In which form is program code usually written and edited?

- **a** Source code.
- **b** Processor-independent byte codes.
- **c** Processor-dependent instructions.

12. Which of these file names are valid for a Java source code file that defines a primary class called `Dog`?

- **a** `Cat.java`
- **b** `Dog.jav`

c `Dog.java`

d `Dog.java.doc`

e `DOG.JAVA`

1.10 Programming exercises

1. Use a text editor to write a file called `SimpleProgram.java` containing the source code from Program 1.1.

2. Compile the source code you wrote in the previous exercise. Fix any errors that the compiler finds in the source code.

3. Run the program that was compiled in the previous exercise.

4. Write a program that prints out the number of characters in your surname. Use the source code from Program 1.1 as a base for your own program.

5. Llanfairpwllgwyngyllgogerychwyrndrobwllllantysiliogogogoch is the name of a village in Wales. Taumatawhakatangihangakoauauotamateapokaiwhenuakitanatahu is the name of a hill in New Zealand. Write a program to calculate the number of characters the hill in New Zealand has in its name compared to that of the village in Wales. The result should be printed out like this, where `?` is replaced by the answer:

```
The hill has ? characters more than the village.
```