

```
}
```

However, the first one's style (with whitespace) is preferred, as the readability is higher. (The method body is easier to distinguish from the head, even at a higher reading speed.)

Statements

Now, that we have the Java platform on our systems and have run the first program successfully, we are geared towards understanding how programs are actually made. As we have already discussed. A program is a set of instructions – very simple tasks provided to a computer. These instructions are called **statements** in Java. Statements can be anything from a single line of code to a complex mathematical equation. This section helps you understand what statements are and how they work.

What exactly are statements?

Statements are a single instruction in a program – a single unit of code. Consider the following line:

Listing 1.1: A simple assignment statement.

```
int age = 24;
```

This line is a simple instruction that tells the system to initialize a variable and set its value as 24. Within that simple definition, we talked about initialization of a variable and setting its value. This all might sound a bit too technical, but it will make sense as you read ahead.

Where do you find statements

Java, in the same style as C and C++, places statements within functions (or methods). The function in turn is placed within a class declaration. If the above statement was the only one in the program, it would look similar to this:

```
public class MyProgram
{
    public static void main (String[] args)
    {
        int age = 24;
    }
}
```

The class declaration and function declaration will be described in the upcoming chapters.

Variables

Christmas are usually exciting, birthdays too. And the one thing that makes them exciting are: you get presents. Think of a present you've ever gotten. A tiny (or big, if you're lucky) box with your name on it. Now think of variables as something similar. They are tiny little boxes in the computer's memory that save something within themselves. This something within them is called a value.

Data types

Take a look at the code in Listing 1.1. Here the variable we have just created is age. The word int tells us what is inside the age variable. int actually stands for integer – a number. On the right to this variable is the value of the variable which is the number 24. Just like int, we can use byte, short, long, double, float, boolean, char or String. All

these tell us what type of data is within a variable. These are hence called **data types**.

We explored the statement in Listing 1.1, where the variable held an integer within in. Let's put another type of data within it. Let's say, the number 10.5. The code would look something like this:

Listing 1.2: Putting a number with decimal point inside an integer variable.

```
int age = 10.5;
```

This is actually wrong. By definition (if you have been awake throughout your mathematics lectures) you'd know that integers are whole numbers: 0, 1, 2, all the way up to infinity. Anything with a decimal point is not an integer, hence the statement by virtue of it is wrong.

What would make it right is when you assign a certain type to the variable that would accept numbers with decimal points – numbers with decimal points are called **floating points**.

Whole numbers and floating point numbers

The data types that one can use for whole numbers are byte, short, int and long but when it comes to floating point numbers, we use float or double. Now that we know that, we can modify the code in Listing 1.2 as:

Listing 1.3: The correct way to assign a type to floating point variables

```
double age = 10.5;
```

Why not float, you say? Well, there are several reasons why not. 10.5 can be anything – a float or a double but by a certain rule, it is given a certain type. This can be explained further by looking at the table below.

Data type	Values accepted	Declaration
byte	Any number between -128 and 127.	byte b = 123;
short	Any number between -32,768 and 32,767.	short s = 12345;
int	Any number between -2,147,483,648 and 2,147,483,647.	int i = 1234567;
long	Any number between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.	long l = 1234567890L;
float	Extremely large numbers beyond the scope of discussion here.	float f = 123.4567f;
double	Extremely large numbers beyond the scope of discussion here. The only difference between double and float is the addition of an f as a suffix after the float value.	double d = 1234.56789;

The above table only list the number data types. We will look at the others as we go on. So, did you notice why we used a double in listing 1.3, and not a float? The answer is pretty simple. If we'd used a float, we would have to append the number with a f as a suffix, so 10.5 should be 10.5f as in:

Listing 1.4: The correct way to define floating point numbers of type float.

```
float age = 10.5f;
```

Assignment statements

Up until now, we've assumed the creation of variables as a single statement. In essence, we assign a value to those variables, and that's just what it is called. When you assign a value to a variable in a statement, that statement is called an **assignment statement**. Did you notice one more thing? The semicolon (;). It's at the end of each statement. A clear indicator that a line of code is a statement is its termination with an ending semicolon. If one was to write multiple statements, it is usually done on each separate line ending with a semicolon. Consider the example below:

Listing 1.5: Multiple assignment statements.

```
int a = 10;
int b = 20;
int c = 30;
```

You do not necessarily have to use a new line to write each statement. Just like English, you can begin writing the next statement where you ended the first one as depicted below:

Listing 1.6: Multiple assignment statement on the same line.

```
int a = 10; int b = 20; int c = 30;
```

However, the only problem with writing such code is, it's very difficult to read it back. It doesn't look that intimidating at first, but once you've got a significant amount of code, it's usually better to organize it in a way that makes sense. It would look more complex and incomprehensible written as it is in Listing 1.6.

Now that we have looked into the anatomy of a simple assignment statement, we can look back at what we've achieved. We know that...

- A statement is a unit of code in programming.
- If we are assigning a variable a value, the statement is called an assignment statement.
- An assignment statement includes three parts: a data type, variable name (also called an identifier) and the value of a variable. We will look more into the nature of identifiers and values in the section titled *Identifiers, literals and expressions* later.

Now, before we move on to the next topic, you need to try and understand what the code below does.

Listing 1.7: Multiple assignment statements with expressions

```
int firstNumber = 10;
int secondNumber = 20;
int result = firstNumber + secondNumber;
```

The first two statements are pretty much similar to those in Listing 1.5 but with different variable names. The third however is a bit interesting. We've already talked of variables as being similar to gift boxes. Think of your computer's memory as a shelf where you put all those boxes. Whenever you need a box (or variable), you call its identifier (that's the name of the variable). So calling the variable identifier `firstNumber` gives you the number 10, calling `secondNumber` would give you 20 hence when you add the two up, the answer should be 30. That's what the value of the last variable `result` would be. The part of the third statement where you add the numbers, i.e., `firstNumber + secondNumber` is called an **expression** and the expression is what decides what the value is to be. If it's just a plain value, nothing fancy, then it's called a **literal**.

With the information you have just attained, you can actually write a decent Java program that can sum up values. To learn more, continue reading.

Classes, Objects and Types

Navigate **Language Fundamentals** topic:

Objects and Classes

An **object** is composed of **members** and **methods**. The members, also called *data members*, *characteristics*, *attributes*, or *properties*, describe the object. The methods generally describe the actions associated with a particular object. Think of an object as a noun, its members as adjectives describing that noun, and its methods as the verbs that can be performed by or on that noun.