Study Unit 3: Java Basics

Introduction

So far, we have learned about Java programming in very broad terms-what a Java program and an executable look like, and how to create simple classes. In this study unit we look at the basics of Java as a programming language mand basically we going to focus on Java statements, expressions, Variables, data types and Comments

Learning Outcomes of Study Unit 3

Upon completion of this study unit, you should be able to

3.1 Explain what exactly a Java statement is

3.2 Evaluate expressions

3.3 Declare and use variables in java

3.4 Explain the different tata types in java as a programming language

3.1 Statements

3.1 Statements and Expressions

Technical Note
Java looks a lot like C++, and-by extension-like C. Much of the syntax will be very familiar to you if you are used to working in these languages. If you are an experienced C or C++ programmer, you may want to pay special attention to the technical notes (such as this one), because they provide information about the specific differences between these and other traditional languages and Java.

A statement indicates the simplest tasks you can accomplish in Java; a statement forms a single Java operation. All the following are simple Java statements:

```
int i = 1;
import java.awt.Font;
System.out.println("This motorcycle is a "
    + color + " " + make);
```

```
m.engineState = true;
```

Statements sometimes return values-for example, when you add two numbers together or test to see whether one value is equal to another. Such statements are called expressions.

White space in Java statements, as with C, is unimportant. A statement can be contained on a single line or on multiple lines, and the Java compiler will be able to read it just fine. The most important thing to remember about Java statements is that each one ends with a semicolon (`;`). Forget the semicolon, and your Java program won't compile.

Java also has compound statements, or blocks, which can be placed wherever a single statement can. Block statements are surrounded by braces (`{}`). You'll learn more about blocks when we are looking at Arrays, Conditionals, and Loops.

3.2 Expressions

3.2.1 simple expressions

expression: A value or operation that computes a value.

Examples: 1 + 4 * 5
(7 + 2) * 6 / 3
42

The simplest expression is a literal value.

A complex expression can use operators and parentheses.

3.2.2 Arithmetic operators

- **operator**: Combines multiple values or expressions.
  - `+` addition
  - `-` subtraction (or negation)
  - `*` multiplication
  - `/` division
  - `%` modulus (a.k.a. remainder)
- As a program runs, its expressions are *evaluated*.

  - 1 + 1 evaluates to 2
  - System.out.println(3 * 4); prints 12
    - How would we print the text 3 * 4 ?
- Applications of % operator:
  - Obtain last digit of a number: 230857 % 10 is 7
  - Obtain last 4 digits: 658236489 % 10000 is 6489
  - See whether a number is odd: 7 % 2 is 1, 42 % 2 is 0

3.2.3 Precedence

- **precedence**: Order in which operators are evaluated.
  - Generally operators evaluate left-to-right.

    1 - 2 - 3 is (1 - 2) - 3 which is -4
  - But * / % have a higher level of precedence than + -

    1 + **3 * 4** is 13

    6 + **8 / 2** * 3

    6 + **4 * 3**

    6 + 12 is 18
  - Parentheses can force a certain order of evaluation:

    (1 + 3) * 4 is 16
  - Spacing does not affect order of evaluation

    1+3 * 4-2 is 11
- What values result from the following expressions?

  9 / 5

  695 % 20

  7 + 6 * 5

  7 * 6 + 5

  248 % 100 / 5

  6 * 3 - 9 / 4

  (5 - 7) * 4

  6 + (18 % (17 - 12))

3.2.4 String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

  "hello" + 42 is "hello42"

  1 + "abc" + 2 is "1abc2"

  "abc" + 1 + 2 is "abc12"

  1 + 2 + "abc" is "3abc"

  "abc" + 9 * 3 is "abc27"

  "1" + 1 is "11"

  4 - 1 + "abc" is "3abc"

- Use + to print a string and an expression's value together.
  - System.out.println(**"Grade: " +** (95.1 + 71.9) / 2);

  Output: Grade: 83.5

3.3 Variables and Data Types

Variables are locations in memory in which values can be stored. Each one has a name, a type, and a value. Before you can use a variable, you have to declare it. After it is declared, you can then assign values to it (you can also declare and assign a value to a variable at the same time, as you'll learn in this section).

Java actually has three kinds of variables: instance variables, class variables, and local variables.

Instance variables, as you learned in chapter two, are used to define the attributes of a particular object. Class variables are similar to instance variables, except their values apply to all that class's instances (and to the class itself) rather than having different values for each object.

Local variables are declared and used inside method definitions, for example, for index counters in loops, as temporary variables, or to hold values that you need only inside the method definition itself. They can also be used inside blocks. Once the method (or block) finishes executing, the variable definition and its value cease to exist. Use local variables to store information needed by a single method and instance variables to store information needed by multiple methods in the object.

Although all three kinds of variables are declared in much the same ways, class and instance variables are accessed and assigned in slightly different ways from local variables. In this chapter, you'll focus on variables as used within method definitions; in the next chapter you'll learn how to deal with instance and class variables.

Note

Unlike other languages, Java does not have global variables-that is, variables that are global to all parts of a program. Instance and class variables can be used to communicate global information between and among objects. Remember that Java is an object-oriented language, so you should think in terms of objects and how they interact, rather than in terms of programs.

3.3.1 Declaring Variables

To use any variable in a Java program, you must first declare it. Variable declarations consist of a type and a variable name:

```
int myAge;
String myName;
boolean isTired;
```

Variable definitions can go anywhere in a method definition (that is, anywhere a regular Java statement can go), although they are most commonly declared at the beginning of the definition before they are used:

```
public static void main (String args[]) {
    int count;
    String title;
    boolean isAsleep;
...
}
```

You can string together variable names with the same type on one line:

```
int x, y, z;
```

```
String firstName, LastName;
```

You can also give each variable an initial value when you declare it:

```
int myAge, mySize, numShoes = 28;
String myName = "Laura";
boolean isTired = true;
int a = 4, b = 5, c = 6;
```

If there are multiple variables on the same line with only one initializer (as in the first of the previous examples), the initial value applies to only the last variable in a declaration. You can also group individual variables and initializers on the same line using commas, as with the last example.

Local variables must be given values before they can be used (your Java program will not compile if you try to use an unassigned local variable). For this reason, it's a good idea always to give local variables initial values. Instance and class variable definitions do not have this restriction. (Their initial value depends on the type of the variable: `null` for instances of classes, `0` for numeric variables, `'\0'` for characters, and `false` for booleans.)

3.3.2 Variable Names

Variable names in Java can start with a letter, an underscore (_), or a dollar sign (`$`). They cannot start with a number. After the first character, your variable names can include any letter or number. Symbols, such as `%`, `*`, `@`, and so on, are often reserved for operators in Java, so be careful when using symbols in variable names.

In addition, the Java language uses the Unicode character set. Unicode is a character set definition that not only offers characters in the standard ASCII character set, but also includes several thousand other characters for representing most international alphabets. This means that you can use accented characters and other glyphs as legal characters in variable names, as long as they have a Unicode character number above `00C0`.

Warning

The Unicode specification is a two-volume set of lists of thousands of characters. If you don't understand Unicode, or don't think you have a use for it, it's safest just to use plain numbers and letters in your variable names. You'll learn a little more about Unicode later.

Finally, note that the Java language is case sensitive, which means that uppercase letters are different from lowercase letters. This means that the variable `X` is different from the variable `x`, and a `rose` is not a `Rose` is not a `ROSE`. Keep this in mind as you write your own Java programs and as you read Java code other people have written.

By convention, Java variables have meaningful names, often made up of several words combined. The first word is lowercase, but all following words have an initial uppercase letter:

```
Button theButton;
long reallyBigNumber;
boolean currentWeatherStateOfPlanetXShortVersion;
```

3.3.3 Variable Types

In addition to the variable name, each variable declaration must have a type, which defines what values that variable can hold. The variable type can be one of three things:

1. One of the eight primitive data types
2. The name of a class or interface
3. An array

You'll learn about how to declare and use array variables later in the course; this chapter lesson focuses on the primitive and class types.

3.3.4 Primitive Types

The eight primitive data types handle common types for integers, floating-point numbers, characters, and boolean values (`true` or `false`). They're called primitive because they're built into the system and are not actual objects, which makes them more efficient to use. Note that these data

types are machine-independent, which means that you can rely on their sizes and characteristics to be consistent across your Java programs.

There are four Java integer types, each with a different range of values (as listed in Table 3.1). All are signed, which means they can hold either positive or negative numbers. Which type you choose for your variables depends on the range of values you expect that variable to hold; if a value becomes too big for the variable type, it is silently truncated.

Table 3.1. Integer types.

Type	Size	Range
`byte`	8 bits	-128 to 127
`short`	16 bits	-32,768 to 32,767
`int`	32 bits	-2,147,483,648 to 2,147,483,647
`long`	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Floating-point numbers are used for numbers with a decimal part. Java floating-point numbers are compliant with IEEE 754 (an international standard for defining floating-point numbers and arithmetic). There are two floating-point types: `float` (32 bits, single precision) and `double` (64 bits, double precision).

The `char` type is used for individual characters. Because Java uses the Unicode character set, the `char` type has 16 bits of precision, unsigned.

Finally, the boolean type can have one of two values, `true` or `false`. Note that unlike in other C-like languages, `boolean` is not a number, nor can it be treated as one. All tests of boolean variables should test for `true` or `false`.

Note that all the primitive types are in lowercase. Be careful when you use them in your programs that you do use the lowercase, because there are also classes with the same names (and an initial capital letter) that have different behavior-so, for example, the primitive type `boolean` is different from the `Boolean` class. You'll learn more about these special classes and what they're used for in the next chapter.

3.3.5 Class Types

In addition to the eight primitive data types, variables in Java can also be declared to hold an instance of a particular class:

```
String LastName;
Font basicFont;
OvalShape myOval;
```

Each of these variables can hold instances of the named class or of any of its subclasses. The latter is useful when you want a variable to be able to hold different instances of related classes. For example, let's say you had a set of fruit classes-`Apple`, `Pear`, `Strawberry`, and so on- all of which inherited from the general class `Fruit`. By declaring a variable of type `Fruit`, that variable can then hold instances of any of the `Fruit` classes. Declaring a variable of type `Object` means that variable can hold any object.

Technical Note
Java does not have a `typedef` statement (as in C and C++). To declare new types in Java, you declare a new class; then variables can be declared to be of that class's type.

Assigning Values to Variables

Once a variable has been declared, you can assign a value to that variable by using the assignment operator `=`, like this:

```
size = 14;
tooMuchCaffiene = true;
```

3.3.6 Comments

Java has three kinds of comments: two for regular comments in source code and one for the special documentation system `javadoc`.

The symbols `/*` and `*/` surround multiline comments, as in C or C++. All text between the two delimiters is ignored:

```
/* I don't know how I wrote this next part; I was working
    really late one night and it just sort of appeared. I
    suspect the code Elvis did it for me. It might be wise
    not to try and change it.
*/
```

These comments cannot be nested; that is, you cannot have a comment inside a comment.

Double-slashes (`//`) can be used for a single line of comment. All the text up to the end of the line is ignored:

```
int vices = 7; // are there really only 7 vices?
```

The final type of comment begins with `/**` and ends with `*/`. The contents of these special comments are used by the `javadoc` system, but are otherwise used identically to the first type of comment. `javadoc` is used to generate API documentation from the code. You can read more about javadoc on your own.