Fundamentals of Programming
STUDY GUIDE FOR MODULE NO. 3
Variables and Constant

Module Overview

Welcome to this module! By the time that you are reading this, you already know the basic Java program structure and have written and build your first Java program. Further, the importance of documenting your program by including comments was also discussed to you in the previous module. In order to expand your knowledge in writing programs that perform useful tasks that really save you work, we introduce the concept of variables and named constants.

LEARNING CONTENTS (Identifiers and Keywords)

Programming involves the manipulation and processing of data. Thus, there has to be a way for the programmer to store this data in the computer memory. The computer memory is composed of memory locations, with each memory location having a unique numeric address. However, rather than using a cryptic numeric address, a programmer uses descriptive words that are easier to remember to refer to a memory location. Every memory location that a programmer uses in his program must be declared using a Java instruction that assigns a name, a data type, and (optionally) an initial value to the location. There are two types of memory locations that a programmer can declare: variable and named constants. In this unit, you will learn about identifiers and how to select a name for a memory location. Also you will learn about keywords (reserved words in Java programming that are part of the syntax).

1.1 Identifiers

Every memory location that a programmer uses in the program must be assigned a name. The name, also called the **identifier**, should describe the meaning of the value stored therein. An identifier is a descriptive name to a variable and named constant that the program will use and should help you remember the purpose of the memory location. However, there are specific rules that a programmer must follow in declaring identifiers in Java programs. A *valid identifier* must follow the following rules:

- A memory location's name must begin with a letter and it can include only letters, numbers, and underscore (_)
- No punctuation characters or spaces are allowed in a memory location's name
- The Java compiler you are using determines the maximum number of characters in a name.
- The name also cannot be the same as the keyword because keyword has a special meaning in a programming language.

Note – The Java language is a "case sensitive" language. That means that an identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example, the RESULT variable is not the same as the result variable or the Result variable. These are three different identifiers identifying three different variables.

You can choose any name as an identifier if you follow the above rules. However, you should give meaningful names to the identifier that makes sense.

Examples of identifiers:

Invalid Identifier	Bad Identifier	Good Identifier
`Total points`	`T_points`	`totalPoint`
`1list`	`list_1`	`list1`
`float`	`n_float`	`floatNumber`

LEARNING CONTENTS (Identifiers)

1.1.1 Identifier naming conventions

Programmers often follow naming conventions for variables and named constants to make their programs easy to read and maintain.

In Java it is a convention that variable names should begin with a lowercase letter. If the variable name is one word, the whole thing should be written in lowercase. Let's take a look at the following variable declarations:

```
1 int value; // correct
2
3 int Value; // incorrect (should start with lower case letter)
4 int VALUE; // incorrect (should start with lower case letter)
5 int VaLuE; // incorrect (see your psychiatrist) ;)
```

However, for a variable or function name which is multi-word, there are two common conventions:

1. words separated by underscores, called snake_case
2. intercapped (sometimes called camelCase, since the capital letters stick up like the humps on a camel).

Let's look at the following examples of correct and invalid variable declarations for multi-word variables:

```
1  int my_variable_name; // correct (separated by underscores/snake_case)
2  int my_function_name(); // correct (separated by underscores/snake_case)
3
4  int myVariableName; // correct (intercapped/CamelCase)
5  int myFunctionName(); // correct (intercapped/CamelCase)
6
7  int my variable name; // invalid (whitespace not allowed)
8  int my function name(); // invalid (whitespace not allowed)
9
10 int MyVariableName; // valid but incorrect (should start with lower case letter)
11 int MyFunctionName(); // valid but incorrect (should start with lower case letter)
```

You can follow either conventions for multi-word identifier names. It is common for programmers to use either and sometimes they use a mix of these two.

1.1.2 Best practices in naming identifiers

1. When working in an existing program, use the conventions of that program (even if they don't conform to modern best practices). However, if you are writing new programs, use modern best practices.
2. Avoid naming your identifiers starting with an underscore since these names are usually reserved for OS, library, and/or compiler use.
3. Make sure that your identifiers clearly state what value they are holding. This would make other programmers and non-programmers figure out as quickly as possible what your code does.
4. The length of the identifier must be proportional to how widely it is used.
   - An identifier with a trivial use can have a short name (e.g. such as i)
   - An identifier that is used more broadly (e.g. a function that is called from many different places in a program) should have longer and more descriptive name (e.g. instead of open, try openFileOnDisk)

The table below shows examples of identifiers that may or may not follow the identifier naming practices enumerated above.

int ccount	Bad	What does the c before “count” stand for?
int customerCount	Good	Clear what we’re counting
int i	Either	Okay if use is trivial, bad otherwise
int index	Either	Okay if obvious what we’re indexing
int totalScore	Either	Okay if there’s only one thing being scored, otherwise too ambiguous
int _count	Bad	Do not start names with underscore
int count	Either	Okay if obvious what we’re counting
int data	Bad	What kind of data?
int time	Bad	Is this in seconds, minutes, or hours?
int minutesElapsed	Good	Descriptive
int value1, value2	Either	Can be hard to differentiate between the two
int numApples	Good	Descriptive
int monstersKilled	Good	Descriptive
int x, y	Either	Okay if use is trivial, bad otherwise

5. In any case, avoid abbreviations. Use your editor’s auto-complete feature if you want to write code faster. Although using abbreviations reduce the time you need to write your code, they make your code harder to read. Code is read more often than it is written, the time you saved while writing the code is time that every reader, including the future you, wastes when reading it.

6. Finally, a clarifying comment can go a long way. As what have been discussed in the previous module, comments make your code easy to understand. For example, say we’ve declared a variable named numberOfChars that is supposed to store the number of characters in a piece of text. Does the text “Hello World!” have 10, 11, or 12 characters? It depends on whether we’re including whitespace or punctuation. Rather than naming the variable numberOfCharsIncludingWhitespaceAndPunctuation, which is rather lengthy, a well-placed comment on the declaration line should help the user figure it out:

```
1 /* holds number of chars in a piece of text -- including whitespace and
2 punctuation!*/
3 int numberOfChars;
```

LEARNING CONTENTS (Keywords)

Each line of a Java program is composed of several words. The compiler classifies these words into different categories: reserved words, standard identifiers, and user-defined identifiers. Section 1.1 focused on the discussion of user-defined identifiers. In this section, we will discuss the other two categories.

A **Reserved word** is a word with predefined meaning in Java and cannot be changed. It is always written in lowercase letters and can only be used for the purpose, which has been

defined to it by the Java compiler. You have already came across some of the reserved words in the previous modules.

For example,

```
int value;
```

Keywords are predefined words that have special meanings to the compiler. They refer to both reserved words and library identifiers

Here is a list of all the Java keywords

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
double	do	else	enum	extends	false
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	

LEARNING CONTENTS (Data Types)

We recall that every memory location that a programmer uses in his program must be declared using a Java instruction that assigns a name, a data type, and (optionally) an initial value to the location. We've already covered the assignment of name or identifier to the memory location. In this section, we will learn about the types of data a memory location can store.

In Java, a **data type** (more commonly just called a **type**) are declarations for variables. This determines the type and size of data associated with variables. For example,

```
int age;
```

In the above example, our variable age was given type int, which means variable age will represent an integer value. This means that the variable can only store integers or whole numbers of either 2 or 4 bytes.

Java fundamental data types are basic types implemented directly by the language that represent the basic storage units supported natively by most systems. The table below shows these fundamental data types, their meaning, and their sizes (in bytes):

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

2.1 Java int

The int keyword is used to indicate integers. This is used to represent a data type that stores whole numbers. The size of an integer type of data is usually 4 bytes. This means that it can store values from - 2147483648 to 214748647.

Example: int salary = 24000;

Here, salary is a variable of type int and holds an initial value of 24000.

Rules that must be observed when using integers are:

- Plus (+) signs do not have to be written before a positive integer, although they may be.
- Minus (-) signs must be written when using a negative number
- Decimal points cannot be used when writing integers. Although 14 and 14.0 have the same value, 14.0 is not of type int
- Commas cannot be used when writing integers; hence 2,176 is not allowed; it must be written as 2176
- Leading zeros should be avoided. If you use leading zeros, the compiler will interpret the number as an octal (base 8) number.

2.2 Java float and double

The float and double data types are used to store floating-point numbers or real numbers (decimals and exponentials). The size of float is 4 bytes while the size of double is 8 bytes. Therefore, double has two times the precision of float.

For example,

```
float area = 54.90;
double volume = 13457.64534;
```

As mentioned above, these two data types are also used for exponentials. For example,

```
double distance = 25E14 // 25E14 is equal to 25*10^14
```

Rules that must be observed when using integers are:

- Plus (+) and minus (-) signs for data of type double are treated exactly as with integers.
- When working with real numbers, however, trailing zeros are ignored.

As with integers, leading zeros should be avoided. Thus, +23.45, 23.45 and 23.450 have the same value, but 023.45 may be interpreted by some Java compilers as an octal followed by a decimal point which is a syntax error.

2.3 Java char

The type char is used to represent character data. In standard Java, data type char can be only a single character. Character constants of type char must be enclosed in single quotation marks when used in a program. Otherwise, they are treated as variables and subsequent use may cause compilation error. Thus, to use the letter A as a constant, you would type ‘A’.

Example: `char letter = ‘A’;`

2.4 Java wchar_t

Wide character wchar_t is similar to the char data type, except its size is 2 bytes instead of 1. It is used to represent characters that require more memory to represent them than a single char.

Example: `wchar_t test = L'ם '// storing Hebrew character`

Notice the letter L before the quotation marks.

2.5 Java bool

The bool data type has one of two possible values: true or false. Booleans are used in conditional statements and loops (which we will learn in the next modules).

Example: `bool cond = false;`

2.6 Java void

The void keyword indicates an absence of data. It means “nothing” or “no value”. The void type will be discussed in more details in the succeeding modules.

Note – We cannot declare variables of the void type.

2.7 Java Type Modifiers

C++ type modifiers are used to further modify some of the fundamental data types. There are 4 types modifiers in Java. These are:

1. signed
2. unsigned
3. short
4. long

We can modify the following data types with the above modifiers:

- int
- double
- char

LEARNING CONTENTS (Variables)

Introduction

In the previous sections of this module, we covered how to select appropriate name and data type of a memory location. There are two types of memory location that a programmer can declare: variables and named constants. A variable is a memory location whose value can change during runtime. Most of the memory locations declared in a program are variables. On the other hand, a named constant is a memory location whose value cannot be changed during runtime.

In a program that inputs the radius of any circle and then calculates and outputs the circle's area, a programmer would declare variables to store the values of the radius and area; doing this allows the values of radius and area to change while the program is executing. However, he or she would declared a named constant to store the value of pi (π), which is used in the formula for calculating the area of a circle. (The formula is $A = \pi r 2$). A named constant is appropriate in this case since the value of pi (3.141593 when rounded to six decimal places) will always be the same. Examine the Java code below:

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // This program calculates the area of a circle given its radius
        double radius, area;
        final double PI = 3.141593;

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the radius of a circle: ");
        radius = scanner.nextDouble();

        area = PI * radius * radius; // Fixed area calculation
        System.out.println("The area of the circle is: " + area);
    }
}
```

[Sample Output]

```
Enter the radius of a circle: 7.5
The area of the circle is: 23.5619
```

The statement double radius, area; is a declaration of variables named radius and area that can hold exponential values. While the statement const double PI = 3.141593; is a declaration of named constant PI that holds a constant value of 3.141593. You will encounter more statements like these in this lesson.

3.1 Variable Declaration

The variable declarations in a Java program communicate to the Java compiler the names of all variables used in a program. Further, they tell the compiler what kind of information will be stored in each variable. Java a strongly-typed language, and requires every variable to be declared with its type before its use. This means that the type of a variable must be known at compile-time (when the program is compiled), and that type cannot be changed without recompiling the program.

You declare a variable using a statement, which is a Java instruction that causes the computer to perform some action after being executed by the computer. A statement that declares a variable causes the computer to reserve a memory location with the name, data type, and initial value you provide. A variable declaration statement is one of many different types of statements in Java. The declaration of variables is done after the opening brace of main().

The syntax to declare a new variable in Java is straightforward: we simply write the type followed by the variable name (i.e., its identifier). The term syntax refers to the rules of a programming language. For example:

```
1 int total;
2 float price;
```

These are two valid declarations of variables. The first one declares a variable of type int with the identifier total. The second one declares a variable of type float with the identifier price. Once declared, the variables total and price can be used within the rest of their scope in the program.

If declaring more than one variable of the same type, they can all be declared in a single statement by separating their identifiers with commas. For example:

```
int num1, num2, num3;
```

This declares three variables (num1, num2, and num3), all of them of type int, and has exactly the same meaning as:

```
int num1;
int num2;
int num3;
```

There are two common mistakes that new programmers tend to make (neither serious, since the compiler will catch these and ask you to fix them) when defining multiple variables of the same type in a single statement:

The first mistake is giving each variable a type when defining variables in sequence

```
int num1, int num2, int num3;     //wrong (compiler error)
```

The second mistake is to try to define variables of different types in the same statement, which is not allowed. Variables of different types must be defined in separate statements.

```
int num1, double num2, float num3;     //wrong (compiler error)

int num1; double num2; float num3;     //correct (but not recommended)

//correct and recommended (easier to read)
int num1;
double num2;
float num3;
```

To see what variable declarations look like in action within a program, let's have a look at the entire C++ code of the example below:

```
public class Main {
    public static void main(String[] args) {
        // declaring variables:
        int num1, num2;
        int result;

        // process:
        num1 = 5;
        num2 = 2;
        num1 = num1 + 1;
        result = num1 - num2;

        // print out the result:
        System.out.println(result);
    }
}
```

```
[Sample Output]
4
```

Don't be worried if something else than the variable declarations themselves look a bit strange to you. Most of it will be explained in more detail in the next modules.

The declaration of a named constant is quite similar to the declaration of variables. To reserve a named constant, you must follow this syntax:

const datatype constantName = value;

In the syntax, datatype is the type of data the named constant will store, constantName is the identifier of the constant data, and value is the literal constant that represents the value you want to be stored in a named constant. The keyword const is used to indicate that the variables value cannot be changed.

Examples:

```
const int  AGE  =  65;
const float MAXPAY  =  15.75;
const char  YES  =  'Y';
```