UNIT 2 : PROGRAMMING BASICS

UNIT STRUCTURE

2.1 Learning Objectives

2.2 Introduction

2.3 Java Tokens

2.4 Variables

2.4.1 The Scope of Variables

2.5 Constants

2.6 Data Types

2.7 Operators and Expressions

2.8 Input/Output in Java

2.9 Let Us Sum Up

2.10 Further Readings

2.11 Answers to Check Your Progress

2.12 Model Questions

2.1 LEARNING OBJECTIVES

After going through this unit, you will be able to :

- define Java tokens
- define the variables, constants and data types in Java
- declare and define variables in Java
- define the various operators used in Java programming
- describe and use control flow statements in Java.

2.2 INTRODUCTION

The previous unit is an introductory unit where you have been acquainted with the object-oriented features of the programming language Java. The installantion procedure of Java SDK is also described in the unit.

You have learnt how to write, save, compile and execute programs in Java from the previous unit.

In this unit we will discuss the basics of Java programming language which include tokens, variables, data types, constants etc. This might be a review for learners who have learnt the languages like C/C++ earlier. We extend this discussion by adding some new concepts associated with Java. Different control flow statements like if, if-else, while, for, break, continue etc. will also be covered in this unit.

2.3 JAVA TOKENS

The smallest individual units in a program are known as ***tokens***. A Java program is basically a collection of tokens. There are five types of tokens in Java language. They are: ***Keywords***, ***Identifiers***, ***Literals***, ***Operators*** and ***Separators***.

- **Keywords :** Keywords are some reserved words which have some definite meaning. Java language has 60 reserved words as keywords. They cannot be used as variable name and they are written in lower-case letter. Since Java is case-sensitive, one can use thse word as identifiers by changing one or more letters to upper-case. But generally it should be avoided. Java does not use many keywords of C/C++ language but it has some new keywords which are not present in C/C++. A list of Java keywords are given table 2.1.

Table 2.1 : Java Keywords

abstract	boolean	break	byte	byvalue
case	cast	catch	char	class
const	continue	default	do	double
else	extends	false	final	finally
float	for	future	generic	goto
if	implements	import	inner	instanceof
int	interface	long	native	new

null	operator	outer	package	private
protected	public	rest	return	short
static	super	switch	syncrhronized	this
threadsasafe	throw	throws	transient	true
try	var	void	volatile	while

- **Identifiers :** Java Identifers are used for naming classes, methods, variables, objects, labels in a program. These are actually tokens designed by programmers. There are a few rules for naming the identifiers. These are:
- Identifier name may consists of alphabets, digits, dolar ($) character, underscore(_).
- Identifier name must not begin with a digit.
- Upper case and lowercase letters are distinct.
- Blank space is not allowed in a identifier name.
- They can be of any length.

While writing Java programs, the following naming conventions should be followed by programmers :

- All local and private variables use only lower-case letters. Underscore is used to combine words if required. For example,

  total_marks

  average
- When more than one word are used in a name, the second and subsequent words are marked with a leading upper-case letter. For example,

  Date of Birth,

  Total Marks,

  Student Name
- Names of all public methods and interface variables start with a leading lower-case letters. For example: total, average.
- All classes and interfaces start with a leading upper-case letter. For example,

HelloJava

Employee

ComplexNumber

- Variables that represent constant values use all upper-case letters and underscore between word if required. For example,

  PI

  RATE

  MAX_VALUE

- **Literals :** Literals in Java are a sequence of characters such as digits, letters and other characters that represent constant values to be stored in a variable.
- **Operators :** An operator is a symbol that takes one or more arguments and opeates on them to produce a result. We will explain various operators in *section 2.7*.
- **Separators :** Separators are symbols used to indicate where groups of code are arranged and divided. Java separators are as follows:

  { } Braces

  ( ) Parentheses

  [ ] Brackets

  ; Semicolon

  , Comma

  . Period

2.4 VARIABLES

A variable is an identifier that denotes a storage location where a value of data can be stored. The value of a variable in a particular program may change during the execution of the program.

We must provide a name and a type for each variable we want to use in our program. The variable's name must be a legal identifier. The variable's type determines what values it can hold and what operations can be performed on it. Some valid variable names are: *sum*, *student_age*, *totalMarks* etc. The rules for naming a variable are same as for the identifiers

which are already discussed in the previous section. Blank space is not allowed in a variable name. Like other language C, C++ , the general syntax of the variable declaration in Java looks like this:

type name ;

2.4.1 The Scope of Variables

In addition to the name and the type that we explicitly give to a variable, a variable has scope. The section of code where the variable's name can be used is the variable's scope. The variable's scope is determined implicitly by the location of the variable declaration, that is, where the declaration appears in relation to other code elements. We will learn more about the scope of Java variables in this section.



Method : Method is a subprogram that perform a specific task

Java variables are categorized into three groups according to their scope. These are: ***local variable***, ***instance variable*** *and* ***class variable***.

- ***Local variables*** are variables which are declared and used inside methods.Outside the method definition they are not available for use and so they are called local variables. Local variables can also be declared inside program blocks that are defined between an opening { and a closing brace }. These variables are visible to the program only.
- ***Instance variables*** are created when the *class* objects are instantiated. They can take different values for each object.
- ***Class variables*** are global to a class and belong to the entire set of object that class creates. Only one memory location is reserved for each *class* variable. *Instance* and *class* variables are declared inside a class.

2.5 CONSTANTS

While a program is running, different values may be assigned to a variable at different times (thus the name *variable*, since the values it contains can vary), but in some cases we donot want this to happen. If we want a value to remain fixed, then we use a constant. ***Constants*** in Java refer to fixed values that donot change during the execution of a program. A constant is declared in a manner similar to a variable but with additional reserved word ***final***. A constant must be assigned a value at the time of its declaration. Thus, the general syntax is:

final type name = value ;

Here's an example of declaring some constants:

final double PI = 3.14159;

final int PASS_MARK = 200 ;

Java supports several types of contants such as *integer constants*, *real constants*, *character constants*, *string constants*, *backslash character constants*.

- **Integer constant :** An integer constant refers to a sequence of digits. For example, Decimal integer constants: 5, -5, 0, 254321

  Octal: 0, 025, 0125

  Hexadecimal: 0X2, 0X9F, Ox etc.
- **Real constant :** Real or floating point constants are represented by numbers containing fractional parts. For example,

  0.125, -.25, 124.75, .8, 450.,-85 etc.

  Exponential notation: 1.5e+4, 0.55e4 etc.
- **Character constant :** Character constant contains a single character enclosed within a pair of single quote marks. For example,

  '1' , 'A', 'c' , ' ' , ' ; ' etc.
- **String constant :** It is a sequence of characters enclosed between double quotes. Characters may be alphabets, digits, special characters, blank spaces. For example,

  "2018", "KKSHOU", "3+4", "Hello Java", "$5", "A"

- **Backslash Character constant :** Java supports some backslash character constants that are used in output methods. These are also known as *escape sequences*. For example,

' \b '	back space
' \n '	new line
' \f '	form feed
' \r '	carriage return
' \t '	horizontal tab
' \' '	single quote
' \ " " '	double quote
' \\ '	backslash

2.6 DATA TYPES

Every variable must have a data type. A data type determines the values that the variable can contain and the operations that can be performed on it. A variable's type also determined how its value is stored in the computer's memory. The JAVA programming language has the following categories of data types as shown in figure 2.1.

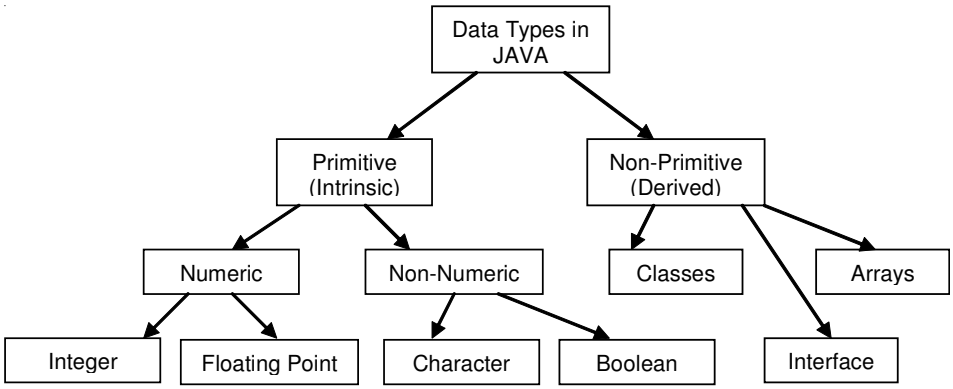


Figure 2.1 : Data types in Java

A variable of ***primitive*** type contains a single value of the appropriate size and format for its type: a number, a character, or a boolean value. Primitive types are also termed as *intrinsic* or *built-in* types. The primitive types are described below:

- **Integer type :** Integer type can hold whole numbers like 1,2, 3,.... -4, 1996 etc. Java supports four types of integer types: ***byte***, ***short***,

int and ***long***. It does not support *unsigned* types and therefore all Java values are *signed* types. This means that they can be positive or negative. For example, the ***int*** value 1996 is actually stored as the bit pattern 00000000000000000000011111001100 as the binary equivalent of 1996 is 11111001100. Similary, we must use a ***byte*** type variable for storing a number like 20 instead of an ***int*** type. It is because that smaller data types require less time for manipulation. We can specify a long integer by putting an 'L' or 'l' after the number. 'L' is preferred, as it cannot be confused with the digit '1'.

- **Floating Point type :** Floating point type can hold numbers contaning fractorial parts such as 2.5, 5.75, -2.358. i.e., a series of digits with a decimal point is of type floating point. There are two kinds of floating point storage in Java. They are: ***float*** (Single-precision floating point) and ***double*** (Double-precision floating point). In general, floating point numbers are treated as double-precision quantities. To force them to be in single-precision mode, we must append 'f' or 'F' to the numbers. For example, 5.23F, 2.25f.

- **Character type :** Java provides a character data type to store character constants in memory. It is denoted by the keyword ***char***. The size of char type is 2 bytes.

- **Boolean type :** Boolean type can take only two values: ***true*** or ***false***. It is used when we want to test a particular condition during the execution of the program. It is denoted by the keyword ***boolean*** and it uses 1 byte of storage.

  The memory size and range of all eight primitive data types are given in the following table 2.2 :

Table 2.2 : Size and Range of Primitive type

Type	Size	Minimum Value	Maximum Value
byte	1 byte	–128	127
short	2 bytes	–32, 768	32, 767
int	4 bytes	–2, 147, 483, 648	2, 147, 483, 647
long	8 bytes	–9, 223, 372, 036, 854, 775, 808	9, 223, 372, 036, 854, 775, 807
float	4 bytes	3.4e–038	3.4e+038
double	8 bytes	1.7e–308	1.7e+308
char	2 bytes	a single Unicode character	
boolean	1 byte	a boolean value (true or false)	

In addition to eight primitive types, there are also three kinds of ***non-primitive*** types in JAVA. They are also termed as *reference* or *derived* types. The non-primitive types are: ***arrays***, ***classes*** and ***interfaces***. The value of a non-primitive type variable, in contrast to that of a primitive type, is a reference to (an address of) the value or set of values represented by the variable.These are discussed later as and when they are encountered.



CHECK YOUR PROGRESS

Q1. Which of the following are valid variable names?
char, float, anInt, p, 4, total-marks, total_matks, sum10digit, num2, MARKS, super .

Q2. What are the eight Java primitive types?

Q3. Write true or false :

i) A constant is declared in a manner similar to a variable but with additional reserved word *final.*

ii) *Boolean* and it uses 2 bytes of storage.

iii) Array is an example of non-primitive type.

iv) Identifier name must not begin with a digit.

v) Variable name may consists of only alphabets and digits.

2.7 OPERATORS AND EXPRESSIONS

Operators are special symbols that are commonly used in expressions. An operator performs a function on one, two, or three operands. An operator that requires one operand is called a *unary operator*. For example, ++ is a unary operator that increments the value of its operand by 1. An operator that requires two operands is a *binary operator*. For example, = is a binary operator that assigns the value from its right-hand operand to its left-hand operand. Finally, a *ternary* operator is one that requires three operands. The Java programming language has one ternary operator (**?:**).

Expressions are the simplest form of statement in Java that actually accomplishes same operations.

Many Java operators are similar to those in other programming languages. Java supports most C++ operators. In addition, it supports a few that are unique to it. Operators in Java include ***arithmetic***, ***assignment***, ***increment*** and ***decrement***, ***Relational*** and ***logical*** operations. This section describes all these operations.

Arithmetic Operators : Java has the following operators for basic arithmetic as shown in table 2.3.

Table 2.3: Arithmetic operators

Operator	Meaning	Example
+	Addition	2 + 3
–	Subtraction	5 – 2
*	Multiplication	2 * 3
/	Division	14 / 4
%	Modulus	14 % 4

In the table, each operator takes two operands, one on either side of the operator. Integer division results in an integer. Because integers do not have decimal fractions, any remainder is ignored. The expression 14/ 3, for example, results in 3. The remainder 2 is ignored in this case. Modulus (%) gives the remainder once the operands have been evenly divided. For

example, 14 % 4 results in 2 because 4 goes into 14 three times, with 2 left over. A sample program for arithmetic operation is given below:

*//**Program 2.1 :** OperatorDemo.java*

```
public class Operator Demo
{
      public static void main(String[ ] args){
            int a = 5, b =3;
            double x = 25.50, y = 5.25;
            System.out.println("Variable values are :\n");
            System.out.println("    a = " + a);
            System.out.println("    b = " + b);
            System.out.println("    x = " + x);
            System.out.println("    y = " + y);
            //Addition
            System.out.println("\nAddition...");
            System.out.println("    a + b = " + (a + b));
            System.out.println("    x + y = " + (x + y));
            //Subtraction
            System.out.println("\nSubtraction...");
            System.out.println("    a - b = " + (a - b));
            System.out.println("    x - y = " + (x - y));
            //Multiplication
            System.out.println("\nMultiplication...");
            System.out.println("    a * b = " + (a * b));
            System.out.println("    x * y = " + (x * y));
            //Division operation
            System.out.println("\nDivision...");
            System.out.println("    a / b = " + (a / b));
            System.out.println("    x / y = " + (x / y));
            //Modulus operation
            System.out.println("\nModulus...");
            System.out.println("   a % b = " + (a % b));
            System.out.println("   x % y = " + (x % y));
```

```
        }
}
```

The output of the above program will be:

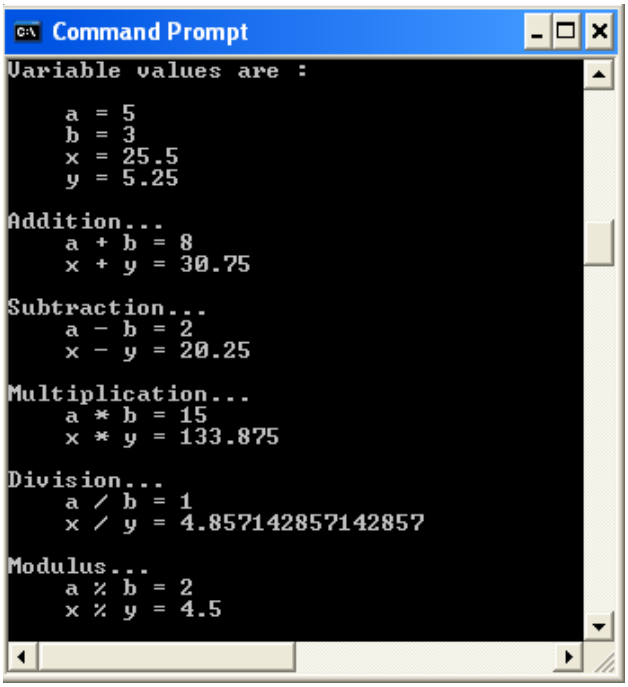


Figure 2.2 : Output of Program 2:1

Assignment Operators : Assignment operators are used to assign the value of an expression to a variable. The usual assignment operator is '='. The general syntax is:

variableName = value;

For example, sum = 0; // 0 is assigned to the variable sum

x = x + 1;

Like C/C++, Java also supports the shorthand form of assignments. For example, the statement x = x + 1; can be written as x += 1; in shorthand form.

Increment and Decrement Operators : The unary **increment** and **decrement** operators ++ and -- comes in two forms, *prefix* and *postfix*. They perform two *operations*. They *increment* (or *decrement*) their operand, and *return* a value for use in some larger expression.

In ***prefix*** form, they modify their operand and then produce the new

value. In ***postfix*** form, they produce their operand's original value, but modify the operand in the background. For example, let us take the following two expressions:

```
y = x++;
y = ++x;
```

These two expressions give very different results because of the difference between prefix and postfix. When we use postfix operators (x++ or x--), y gets the value of x before before x is incremented; using prefix, the value of x is assigned to y after the increment has occurred.

Relational Operators : Java has several expressions for testing equality and magnitude. All of these expressions return a boolean value (that is, true or false). Table 2.4 shows the relational operators:

Table 2.4 : Relational operators

Operator	Meaning	Example
==	Equal	x == 5
!=	Not equal	x != 10
<	Less than	x < 7
>	Greater than	x > 4
<=	Less than or equal to	y <= 8
>=	Greater than or equal to	z >= 15

Logical Operators : Expressions that result in boolean values (for example, the Relational operators) can be combined by using logical operators that represent the logical combinations **AND**, **OR**, **XOR**, and logical **NOT**.

For **AND** operation, the && symbol is used. The expression will be true only if both operands tests are also true; if either expression is false, the entire expression is false.

For **OR** expressions, the || symbol is used. OR expressions result in true if either or both of the operands is also true; if both operands are false, the expression is false.

In addition, there is the **XOR** operator ^, which returns true only if its

operands are different (one true and one false, or vice versa) and false otherwise (even if both are true).

For **NOT**, the ! symbol with a single expression argument is used. The value of the NOT expression is the negation of the expression; if x is true, !x is false.

Bitwise Operators : Bitwise operators are used to perform operations on individual bits in integers. Table 2.5 summarizes the bitwise operators available in the JAVA programming language. When both operands are boolean, the bitwise AND operator (**&**) performs the same operation as logical AND (&&). However, & always evaluates both of its operands and returns true if both are true. Likewise, when the operands are boolean, the bitwise OR (|) performs the same operation as is similar to logical OR (||). The | operator always evaluates both of its operands and returns true if at least one of its operands is true. When their operands are numbers, & and | perform bitwise manipulations.

Table 2.5 : Bitwise operators in Java

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Left shift
>>	Right shift
>>>	Zero fill right shift
~	Bitwise complement
<<=	Left shift assignment (x = x << y)
>>=	Right shift assignment (x = x >> y)
>>>=	Zero fill right shift assignment (x = x >>> y)
x&=y	AND assignment (x = x & y)
x|=y	OR assignment (x + x | y)
x^=y	NOT assignment (x = x ^ y)

A shift operator performs bit manipulation on data by shifting the bits of its first operand right or left. For example if op1 and op2 are two operands, then the statement

op1 << op2;

shift bits of op1 left by distance op2; fills with zero bits on the right-hand side and op1 >> op2; shift bits of op1 right by distance op2; fills with highest (sign) bit on the left-hand side.

op1 >>> op2;

shift bits of op1 right by distance op2; fills with zero bits on the left-hand side. Each operator shifts the bits of the left-hand operand over by the number of positions indicated by the right-hand operand. The shift occurs in the direction indicated by the operator itself.

For example, the statement 25 >> 1; shifts the bits of the integer 25 to the right by one position. The binary representation of the number 25 is 11001. The result of the shift operation of 11001 shifted to the right by one position is 1100, or 12 in decimal.

The Java programming language also supports the following operators as shown in table 2.6.

Table 2.6 : Other Operators

Operator	Description
?:	Conditional operator (a ternary operator)
[ ]	Used to declare arrays, to create arrays, and to access array elements
.	Used to form qualified names
(params)	Delimits a comma-separated list of parameters
(type)	Casts (converts) a value to the specified type
new	Creates a new object or array
instanceof	Determines whether its first operand is an instance of its second operand

2.8 INPUT/OUTPUT IN JAVA

Java is rich in method and class for input/output from the key board and to the monitor. The Scanner class can be used for inputting integer data as wel as string data from the keyboard. *Program 2:2* is used to input two integer data from the keyboard and find the sum of the numbers.

//Program 2:2 **:** Read two numbers from the keyboard and find the sum

```
import java.util.Scanner;
class Add_number
{
	public static void main(String args[])
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the first Number :");
		int a=sc.nextInt();
		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();
		int c=a+b;
		System.out.println("The result of addition is :"+c);

	}
}
```

Here, Scanner is a class, sc is an object, a,b and c are three integer variables and nextInt() is a predefined method of Scaner class. The following program is used to read a string from the keyboard and display the content of the string.

//Program 2:3 : Read a string from the keyboard and display it.

```
import java.util.Scanner;
class String_read{
	public static void main(String args[]){
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the Name :");
		String name=sc.next();
		System.out.println("Name :"+name);
	}
}
```



CHECK YOUR PROGRESS

Q4. Write true or false :

i) Logical XOR operator returns true only if its operands are different.

ii) Logical AND operator returns true only if both operands tests are false.

iii) x+=7; has the same effect as x = x+7;

iv) && is the symbol of logical AND and & is the symbol of bitwise AND operator.

Q5. Determine the value of each of the following logical expressions if x = 5, y =10 and z = - 6

i) x > y && x < z ii) x == z || y > x

iii) x < y && x > z



2.9 LET US SUM UP

- In this unit we have discussed all the basic data types and operators in Java and their use in expressions.
- Java variables are categorized into three groups according to their scope: *local variable* , *instance variable and class variable*.
- In Java, constants are declared in the manner similar to variables but with additional reserved word *final.*



2.10 FURTHER READINGS

1) Balagurusamy, E. (2007). Programming With Java: A Primer 3E.

2) Schildt, H. (2000). C/C++ Programmer's Reference. McGraw-Hill, Inc..

3) Deitel, P., & Deitel, H. (2011). Java How to program. Prentice Hall Press.



2.11 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q. No. 1 : anInt, p, total_marks, sum10digit, num2, MARKS

Ans to Q. No. 2 : The eight Java primitive types are boolean, char, byte, short, int, long, float and double

Ans to Q. No. 3 : (i) True; (ii) False; (iii) True; (iv) True; (v)False

Ans to Q. No. 4 : (i) True; (ii) False; (ii)True; (iv) True

Ans to Q. No. 5 : (i) False; (ii) True; (iii) True



2.12 MODEL QUESTIONS

Q1. (a) Consider the following code :

```
int x = 10;
int n = x++%5;
```

What are the values of x and n after the code is executed? What are the final values of x and n if instead of using the postfix increment operator (x++), you use the prefix version (++x)?

(b) What is the value of i after the following code executes?

```
int i = 8;
i >>=2;
```

(c) What is the value of i after the following code executes?

```
int i = 17;
i >>=1;
```

Q2. What are the rules for naming a Java varibles?

Q3. What are the different types of constants in Java? Explain with examples.

Q4. What is scope of a variable?