# UNIT 2 BASICS OF JAVA

**Structure** **Page No**



## 2.0 INTRODUCTION

Java is an object-oriented, general-purpose programming language. The popularity of Java programming language is more because of its features. Basic features of Java include security, robustness, distributed, multithreading, and a rich library for Graphical User Interface (GUI), but one of its main characteristic of it is : being Architecture-Neutral and platform- independent, which we have studied in the previous unit of this Block. To write a program in any programming language, first of all, we need to know the basic facilities that the programming language provides. For example, support of different data types, operators, basic programming constructs for decision making or constructs of looping etc., are the basic facilities Java provides. In this unit, we will learn to use basic building blocks of Java, such as data types, java keywords, variables, literals, operators, statements and expressions. To improve your programming concepts, you can move towards learning of declaration and initiations of different data types and variables. An operator is a symbol that informs the compiler to perform some operation on the given data. Java provides several operators. Java expression is built from variables using different operators. In addition to expression, you will learn statements in Java, which are equivalent to sentences in natural languages. A statement is a complete unit of execution. At the end of this unit, you will be able to write java programs using the scanner class.

## 2.1 OBJECTIVES

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

- describe the different data types in Java,
- declare and initialize variables in Java,
- know reserved words in Java,
- define Variables and Literals,
- use appropriate Operators in writing programs,
- use statements and expressions in program, and
- write Java programs using Scanner class.

## 2.2 DATA TYPE

In programming, we use variables to store values. A variable stores values according to its declared type. Every programming language provides some primitive data types for storing values and evaluating expressions. Java provides primitive data types(basic types), which are **statically-typed**, which means you need to declare all variables before using them.

There are eight basic data types in Java, which include four types of integer values in 'int' data type, two types of fractional values in 'float' data type, character data type in 'char' and true or false values in 'boolean' data type. The int, float, char and boolean are keywords that can not be used as variable names. Java primitive types are given in Table 1.1

**Table 1.1: Java Primitive Data Types**

| Data Type | Size in byte | Description | Example | Groups | Range |
|---|---|---|---|---|---|
| byte | 1 | byte-length integer | 15, 25 | Integer | -128 to 127 |
| short | 2 | short integer | 550, 986 | | -32,768 to 32,767 |
| int | 4 | Integer | 9, 95, 378, 465 | | -2^31 to 2^31-1 |
| long | 8 | long integer | 989L, -545L | | -2^63 to 2^63-1 |
| float | 4 | Single-precision floating point | 809.5f, -232.2f, | Floating point numbers | -3.4e38 to 3.4e38 |
| double | 8 | Double-precision floating point | 2289.5,-4535.5, 27.6E5 | | -1.7e308 to 1.7e308 |
| char | 1 | A single character | 'a', '9', 't' 'A' | Characters | true or false |
| boolean | 1 bit | A boolean value (0 or 1) | true or false | Boolean | '\u0000' to '\uffff' |

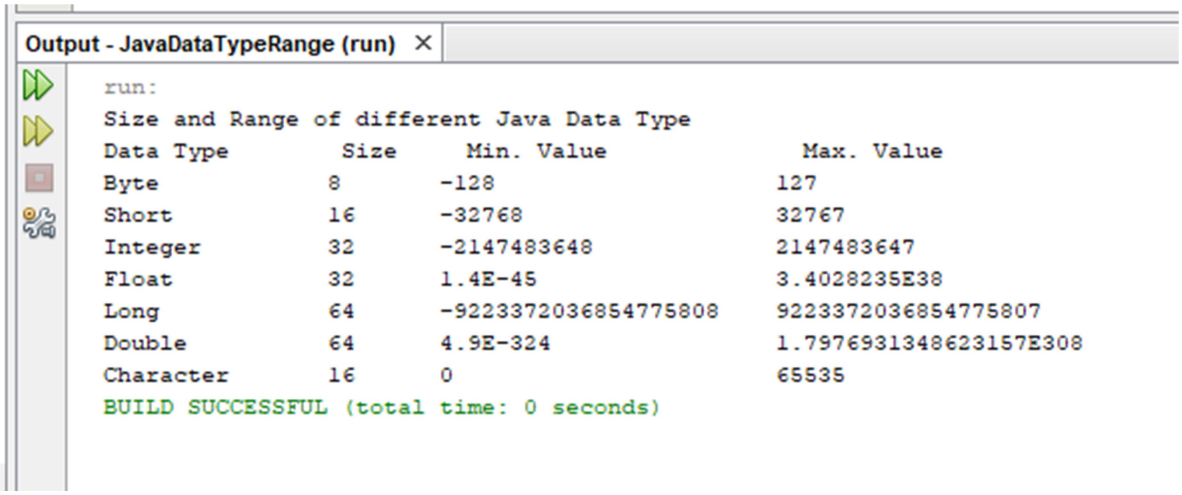
Following is the Java program to display the size and range of Java primitive data types.

**Example Program:**

```
package javadatatyperange;
public class JavaDataTypeRange
{
public static void main(String args[])
{
System.out.println("Size and Range of different Java Data Type ");
System.out.println("Data Type\t Size\t  Min. Value\t\t  Max. Value\t");
System.out.println("Byte\t\t" + Byte.SIZE + "\t" + Byte.MIN_VALUE
                    + "\t\t\t" + Byte.MAX_VALUE);
System.out.println("Short\t\t" + Short.SIZE + "\t" + Short.MIN_VALUE
                    + "\t\t\t" + Short.MAX_VALUE);
System.out.println("Integer\t\t" + Integer.SIZE + "\t" + Integer.MIN_VALUE
                    + "\t\t" + Integer.MAX_VALUE);
System.out.println("Float\t\t" + Float.SIZE + "\t" + Float.MIN_VALUE
                    + "\t\t\t" + Float.MAX_VALUE);
System.out.println("Long\t\t" + Long.SIZE + "\t" + Long.MIN_VALUE + "\t"
                    + Long.MAX_VALUE);
System.out.println("Double\t\t" + Double.SIZE + "\t" + Double.MIN_VALUE
                    + "\t\t" + Double.MAX_VALUE);
System.out.println("Character\t" + Character.SIZE +"\t" + (int)Character.MIN_VALUE
                    + "\t\t\t" + (int)Character.MAX_VALUE);
 }
}
```

If you write and run(execute) the above Java program , it give the following output.

**Output:**




In the above programe you can see that to print the **size** of a data type ***(name of datatype).SIZE*** is used. To print the minimum value, a data type can store (***Name of data type). MIN_VALUE*** is used. Similarly, to print the maximum value, a data type can store ***(Name of data type). MAX_VALUE*** is used. You can notice that the first word of the data type is written in capital. Also Character.MIN_VALUE and Character.MAX_VALUE is converted into an integer ( here, type casting is done). "/t"(tab) is used for proper alignments of output, the use of "/t" is similar to pressing the tab on our keyboard.

In most of programming languages, the format and size of primitive data types depend on the platform on which program code is running. But in Java , the size and format of its primitive data types is specified, and it frees programmers from the system dependencies while writing program and using the data types. For example, an **int** is always 32 bits in Java, regardless of the platform on which the Java program runs. This allows programs to be written with the guarantee to run *without porting* on any machine architecture. All the primitive data types can be divided into four major groups, as shown in Table-1.1. Now let us learn the use of these primitive data types in detail.

### 2.2.1 Integers and Floating Point Data Type

Integers and Floating-point numbers are used to store numeric data values. Java has six numeric data types, as shown in Table 1.1. They differ in size and precision of the numbers they can hold. The size of a data type means how many bits are needed to represent that data type. The range of a data type represents how small/big value can be stored in that data type with the precision of numbers.

**Integers**

To store an integer value, you need to declare a variable of integer type, and *int* keyword is used for it. The integer variable can be declared as follows:

int age;

The data type of the variable declared above is *int,* and the name of the variable is age. The declaration ends with a semicolon. You can store only an integer value in variable age. There are

four types available in Java for integer values. These four types are signed variables that can store either a positive or negative value, as shown in Table 2.2.

**Table 2.2: Java integer types**

| Sl. No. | Data Type | Data Range | Bytes |
|---|---|---|---|
| 1. | byte | -128 to + 128 | 1 |
| 2. | short | -32768 to + 32767 | 2 |
| 3. | int | -2147483648 to 2147483647 | 4 |
| 4. | long | -9223372036854775808 to +9223372036854775808 | 8 |

As a programmer, you need to choose a variable of any of the four types according to your need in the application. You may use byte if you are expecting a very small integer value for your variable. For example, to store student age, you may use a variable of type byte. The *int* is mostly used to store an integer value; for a bigger range of integer values, the long integer type is used. Let us see the four types of integers declared below:

```
byte age;
short year;
int No_of_Smart_City;
long AccoutNo;
```

The above declarations of variables of integer types are according to the range of values they can store.

**Floating Point Numbers**

Those numbers containing fractional parts are called real numbers or floating point numbers. Table 2.3 shows that real numbers can also be divided into two categories in Java: float and double. The double type is used where more accuracy is required for the fractional part.

As integral values are stored in integer variables, non-integral values are stored in floating-point variables. In other words, fractional values or values with a decimal point are stored in floating-point variables. If you have to store a value 16.045 you can be stored in a floating-point variable. The number of decimal places after the decimal point is called the precision. There are two primitive data types, *float* and *double,* for floating-point data.

**Table 2.3: Two categories of real numbers**

| Sl. No. | Data Type | Data Range | Bytes |
|---|---|---|---|
| 1. | Float | $-3.4 \times 10^{38}$ to $+ 3.4 \times 10^{38}$ (OR) -3.4 E38 to + 3.4 E38 | 4 |
| 2. | double | $-1.7 \times 10^{308}$ to + 3.4 E38 (OR) -1.7 E308 to 1.7 E308 | 8 |

As an example, the notation of $10^{35}$ can also be presented as E35. In table 2.3 , $10^{38}$ is presented as E38 in the table. The declaration of a floating-point variable is as follows:

float price = 156. 50F;

This statement declares a variable *price* by initializing it with a floating point constant 156.50F. if F is not appended to the constant, then the value is, by default, double type.

The declaration for the double type is the same as that for the float type but will not have the F towards the end of the constant. Following is a variable earning_ratio declared as double.

double earning_ratio = 52225.55557;

### 2.2.2 Character and Boolean Types

**Characters**

In Java, the character data type- *char* holds a single character. A char is a single character: a letter, a digit, a punctuation mark, a tab, a space or something similar. Each character is a number or character code belonging to a character set, which is an indexed list of symbols. For example, **ASCII** (American Standard Code for Information Interchange) is a character set. The ASCII character set ranges from **0 to 127 and** needs 8 bits( 1 byte) to represent a character. In Java, character variables are stored as **Unicode,** which uses 2 bytes of memory. The character variable is declared and initialized as follows:

Char MyChar = 'P';

The above statement initializes the variable MyChar with a Unicode character representing P. The character constant assigned to a character variable must be enclosed within single quotes. "The character enclosed within single quotes is treated as a character constant and that without single quotes is treated as a variable by the compiler".

Apart from character variables and constants, Java also provides Escape Sequences character constants. All the character constants are specified inside a pair of single quotes. Some characters cannot be entered directly using the pair of quotes. A backslash followed by a character is used to indicate to the compiler that it has a different meaning. Such sequences of characters are called Escape Sequences or Escape Characters. The sequence is typed as two characters with no blank in between the characters. The use of backslash suppresses the meaning of an escape sequence character. To display double quotes inside a string, \" is used so that it will display double quotes and not the end of the string. Though the escape sequence is of two characters, it is treated as a single character. A list of Escape Sequences is shown in Table 2.4.

**Table 2.4: Escape Sequences**

| Sl. No. | Escape Sequence | Meaning |
|---|---|---|
| 1. | \' | Single quote |
| 2. | \" | Double quote |
| 3. | \\ | Backslash |

| | | |
|---|---|---|
| 4. | \b | Backspace |
| 5. | \f | Form feed |
| 6. | \n | New line |
| 7. | \r | Carriage return |
| 8. | \t | Horizontal Tab |

**Booleans**

In Java, Boolean variables are used as logical variables with either true or false values. The Boolean variable has only two values: true or false. These two values are Boolean constants. Different attributes of the boolean data type are given in *Table2.5*.

**Table 2.5: Boolean Data Type**

| Type | Values | Default | Size |
|---|---|---|---|
| boolean | true, false | false | 1 bit used in 32 bit integer |

In Java a boolean variable can be declared as follows:
*boolean aTeacher = true;*

In the above statement, a boolean variable aTeacher is declared and initialized with the boolean value true.

## 2.2.3 Enumerated Types

The constants are those variables whose values are unchangeable. An enumerated data type is a special data type in Java . It is a user-defined data type which consists of a set of predefined values.

Enumerations: An enum is a special "class" used for representing a group of **constants**, were added to the Java language in JDK5. In Java, enumeration defines a class type. It is created using the enum keyword. Each enumeration constant is *public, static* and *final* by default. In Java, final variables are also constant variables. About the final variable, you will learn later in this course. The enumeration variables can be used and declared in much the same way as a primitive variable.

An enumeration can be defined simply by creating a list of *enum* variables. You may create an enum using the enum keyword, and you have to separate the constants using a comma. The general format of an enumerated type declaration:

enumTypeName { One or more enum constants }

Example of creating enumeration :

enum Size { LOW, MEDIUM, HIGH }

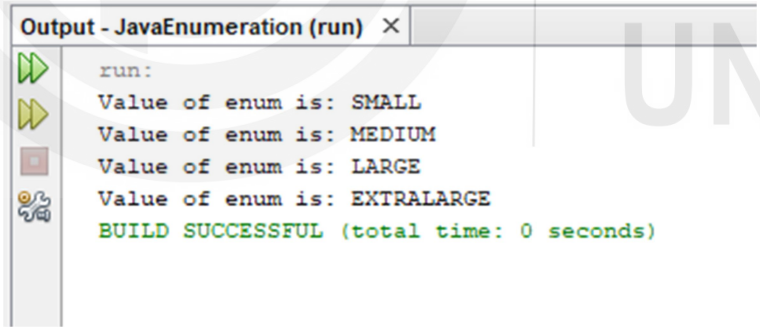
You can access enum constants with the **dot(.)** syntax:

Level myVar = Size.MEDIUM;

Program:

```
package javaenumeration;
enum Size {   SMALL, MEDIUM, LARGE, EXTRALARGE }
public class JavaEnumeration
{
  public static void main(String[] args)
  {
    System.out.println("Value of enum is: " + Size.SMALL);
    System.out.println("Value of enum is: " + Size.MEDIUM);
    System.out.println("Value of enum is: " + Size.LARGE);
    System.out.println("Value of enum is: " + Size.EXTRALARGE);
  }
}
```

Output:




Another Example:

Let us try to define PGDCA_NEW Ist semester Courses as enumeration.

enum PGDCA_NEW_Courses-I
{ MCS-201, MCS-202, MCS-203, MCS-204, MCSL-205}

Here identifiers such as MCS-201, MCS-202, MCS-203, MCS-204, and MCSL-205 are called enumeration constants. These are public, static, and final by default.

## 2.3 UNICODE

In Java to fully understand the char type, you need to know about the Unicode encoding scheme. Before Unicode, many other coding standards, such as ASCII , ISO 8859-1, KOI-8, GB18030 etc. were used in different regions of the world. Having the use of different coding standards caused two main problems:

1. A code value corresponds to different letters in the different encoding schemes.

2. The encodings in the case of those languages which have large character sets have variable lengths.

To solve these problems, Unicode was designed to solve these problems. With the attempt of the unification effort started in the 1980s, it was decided to have a fixed 2-byte code to encode all characters used in all languages in the world. The "Unicode 1.0 was released, using slightly less than half of the available 65,536 code values. Java was designed from the ground up to use 16-bit Unicode characters, which was a major advance over other programming languages that used 8-bit characters".

Unicode is **a universal character encoding standard**. It is "an international encoding standard for use with different languages and scripts, by which each letter, digit, or symbol is assigned a unique numeric value that applies across different platforms and programs." This standard includes roughly about 100000 characters to represent characters of different languages. While ASCII uses only 1 byte Unicode uses 4 bytes to represent characters. Hence, it provides a very wide variety of encoding. "Unicode covers **all the characters for all the writing systems of the world, modern and ancient"**. It also includes technical symbols, punctuations, and many other characters used in writing text. The most widely used forms of Unicode are:

- UTF-32, is most appropriate for single-encoding characters. It uses 32-bit code units, each of which is used for storing a single code point.
- UTF-16 uses one or two 16-bit code units for each code point. It is the default encoding for Unicode.
- UTF-8, with one to four 8-bit code units (bytes) for each code point.

## 2.4 VARIABLES

In Java there are some predefined keywords; they are particular words that act as a key to a code. These keywords are also known as reserved words by Java, so you can not use them as a variable or object name or class name. As you know, a variable is a titled portion of the memory used to store data that is required in any program, perhaps even in other than Java programming language also. A variable can store only one type of data. In this section, you will learn more about java keywords, variables, and literals.

### 2.4.1 Java Keywords

A keyword or reserved word in Java are predefined word with a special meaning and cannot be used as a user-defined variable or identifier. The keywords are used by the compiler, and if you

use them as variable names, the compiler will generate errors. The following Table 2.6 gives a list of Java keywords and their defined purpose. Remember that these keywords cannot be used as variable names or function names in any Java program.

**Table 2.6: List of Java Keywords**

| Keyword | Purpose |
|---|---|
| abstract | Used to declare that a class or method is abstract |
| assert | Used to specify assertions |
| boolean | Declares that variable is boolean type |
| break | Used to exit a loop before the end of the loop is reached |
| byte | Tossed to declares that variable is byte type |
| case | for representing different conditions in a switch statement |
| catch | Used for handling exceptions |
| char | To declare that variable is character type |
| class | For a class definition |
| const | It is a keyword reserved by Java, but now it is not in use |
| continue | Use for prematurely returning to the beginning of a loop |
| default | For defining default action in a switch statement |
| do | For beginning a do while loop |
| double | To declare a variable of double type |
| else | Used for indicating the code to be executed when if condition executes to false |
| extends | Used to Specify the class base from which the correct class is inherited |
| final | It declares that a class may not be extended or that a field or method may not be overridden |
| finally | Used to declare a block of code guaranteed to be executed |
| float | For defining a floating point variable |
| for | To start a for loop |
| goto | This keyword is reserved by Java but now it is not in use |
| if | Used to represent a conditional statement |

| | |
|---|---|
| implements | Declares that this class implements the given interface |
| import | permits access to a class or group of classes in a package |
| instance of | tests whether an object is an instance of a class |
| int | For declaring an integer variable |
| interface | signals the beginning of an interface definition |
| long | For declaring a long integer variable |
| native | It declares that a method that is implemented in native code |
| new | To allocates memory to an object dynamically |
| package | For defining the package to which this source code file belongs |
| private | Used for declaring a method or member variable to be private |
| protected | To declare a class, method or member variable to be protected |
| public | To declare a class, method or member variable to be public |
| return | To return a value from a method |
| short | For declaring a short integer variable |
| static | To declare that a field or a method belongs to a class rather than an object |
| strictfp | For declaring a method or class must be run with exact IEEE 754 semantics |
| super | Used to refer to the parent of the current object |
| switch | Used to test for the truth of various possible cases |
| synchronized | Used to indicate that a section of code is not thread-safe |
| this | Use as a reference to the current object |
| throws | It declares the exceptions thrown by a method |
| transient | Used to make data non-serialized |
| try | Used in exception handling. It attempts an operation that may throw an exception |
| void | It declares that a method does not return a value |

| | |
|---|---|
| volatile | Indicates to the compiler that a variable changes asynchronously |
| while | For beging of a while loop |

Java is case-sensitive, so even though the break is a keyword, Break is not a keyword at all. At the time of designing Java, its designers decided to **make it case-sensitive because,** first of all, this enhances the readability of code; secondly, it reduces the compilation time and increases the efficiency of the compiler. All the Java technology keywords are in lowercase. The words true, false, and null are reserved words. Java keeps on adding new keywords as per need. **Java 1.4** adds the **assert** keyword to specify assertions. As we have seen above in table 2.6, each keyword has a specific and well-defined purpose.

## 2.4.2 VARIABLES AND LITERALS

### Variables

In programming, we use variables to store data that is used in any program. Like other languages, Java also expects the variables to be declared and used. A variable is a titled portion of the memory. You can use a variable to store only one type of data. For example, if a variable is declared to store an integer, it can store only integer data. Java predefines the type of data stored in a variable. Java variables are declared by a name and their type. The name given to a variable is called an identifier. There are a few guiding rules for naming a variable, which is as follows:

- The name of a variable may be of any length.
- The name of a variable must start with an alphabet, underscore ( _ ) or a dollar symbol ($).
- The remaining characters of the variable name can be alphabetic or numeric.
- Special characters such as @,% etc., are not allowed as part of the name of variables.
- Underscore ( _ ) can be used to connect two words such as first _ name, student_marks etc.
- As Java is case-sensitive hence the variables Age and age are treated as two different variables.
- Blanks or tabs may not be used in variable names.

As a programmer, you are free to give any names to the variables, and only you cannot use the keywords defined by Java as the name of the variable. Also, it is advisable that naming a variable should reflect the purpose for which it is declared in a program. The variable names should be meaningful such as book_tiltle, name, username and password, age, marks etc. This makes your program more readable. In Java, the naming convention followed is:

- the variable name starts with a lowercase alphabet
- the remaining words, the first alphabet is an uppercase alphabet

**Declaring and Initialization Variables**

As you are aware, before using any variable, you have to declare it. After it is declared, you can then assign values to it. Also you can declare and assign a value to a variable at the same time.

Java actually has three kinds of variables:

- Instance variables
- Class variables
- Local Variables.

You will learn more about instance variables and class variables in unit 4 of this Block. Instance variables are those variables which are used to define the attributes of a particular object. Class variables are similar to instance variables, except their values is the same for all the objects of a class rather than having different values for each object of the class. Local variables are declared for the local consumption of the methods; for example, for index counters in a for loop, temporary variables to keep some values or to hold values that you need only inside the method definition itself.

**Variable Declarations**

In Java, a general variable declaration looks like the following:

datatype identifier [= default value] {, identifier [= defaultvalue] };

Let us consider the following variable declarations:

byte b;

int age;

boolean male;

char c;

Also, it is possible to declare multiple variables of one type in one expression, as given below:

int age, enrollnum, accountno;

Variable declarations can be anywhere in your code as long as they precede the first use of the variable. However, it is common practice to place the declarations at the top of each block of code. Variable names in Java can only 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 letters, numbers, or a combination of both.

As you know, the Java language is case sensitive, implying that the variable x in lowercase is different from variable X in uppercase. In Java, a variable name cannot start with a digit or hyphen. The names of the variables should be kept in such a way which may help you understand what is happening in your program. Therefore you should keep the name of your variables intelligently or according to their role in the program. Your variable's name cannot contain any white space. There is no limit to the length of a variable name in Java.

Suppose you want to begin a variable name with a digit, prefix underscore, with the name, e.g. _1Name. You can also use the underscore in the variable name to provide better readability and meaning, e.g. Student_age.

### Variable Assignment and Initialization

Once you have declared the type of a variable, you can initialize it with some value.
**variable name = some value**

For example, consider the following example:
int Student_age;
Student_age = 21;  // here the value 21 is assigned to the variable 'Student_age'

You will learn more about assignment operators later in this unit.

### Literals

The constants or data values used in a program are called literals. Each literal is of a particular type. For example, the percentage of marks for a student is 78.25, which is a literal and fractional value, and its type is elaborated on in the subsequent section.

Primitive types in Java are called **literals**. A literal is the source code representation of a fixed value in memory. Literals are nothing but pieces of Java code that indicate explicit values. For example, "Hello IGNOU!" is a String literal. The double quote marks indicate to the compiler that this is a string literal. The quotes indicate the start and the end of the string, but remember that the quotation marks themselves are not a part of the string. Similarly, Character Literals are enclosed in single quotes, and it must have exactly one character. TRUE and FALSE are boolean literals that mean true and false. Number, double, long and float literals also exist there. See examples of all types of literals in the following *Table 2.7*.

**Table 2.7: Examples of literals**

| Types of literal | Example |
|---|---|
| Number Literals | 145, 104L, 777, 0XFF, 2.56F, 15e45, .36E-2 |
| Boolean Literals | TRUE, FALSE |
| Character Literals | 'c', '#', '5', \n, \\, \" |
| String Literals | "Java Programming" |
| Double Literals | 16.5, 85.6, 96.4E8 |
| Long Literals | 54L |
| Float Literals | 35.6f, 86.4E, 8F, 1.5F |

### Constants

Constants are used for fixed values. Their value never changes during the program execution. To declare constants in Java keyword final is used. The following statement defines an integer constant x containing a value of 25.

final int x =25;

### Check Your Progress- 1

1) What is literal in Java?

……

……

……

……

2) Write a program in Java to calculate the Area of a circle. Show the use of the 'final' keyword in your program.

……

……

……

……

3) What is boolean data type in Java?

……

……

……

……

In the next section, we will explore the different types of operators in Java with their meaning and uses in the programs.

## 2.5 OPERATORS

An operator is a symbol that informs the compiler to perform some operation on the given data. Java provides several operators. The operations can be classified into four categories: arithmetic, relations, logical and bitwise. Each category is explained briefly in the subsequent sections.

### Arithmetic operators

You must be friendly with arithmetic expressions in mathematics like 'A – B'. In this expression A and B are operands and the subtraction sign ' – ' is the Operator. The same terminology is also used here in the programming language. The following table lists the basic arithmetic operators provided by the Java programming language, along with their descriptions and uses. Descriptions of arithmetic operators are defined in Table 2.8(a). Operators are divided into two categories **Binary and Unary.** Addition, subtraction, multiplication etc. are binary operators and applied only on two operands. Increment and decrement are unary operators and are applied on a single operand.

**Table 2.8(a): Description of arithmetic operators(**Binary Operators)

| Operator | Use | Description |
|---|---|---|
| + | A + B | Adds A and B |
| - | A – B | Subtracts B from A |
| * | A * B | Multiplies A by B |
| / | A / B | Divides A by B |
| % | A % B | Computes the remainder of dividing A by B |

**Table 2.8(b): Description of arithmetic operators(Unary Operator)**

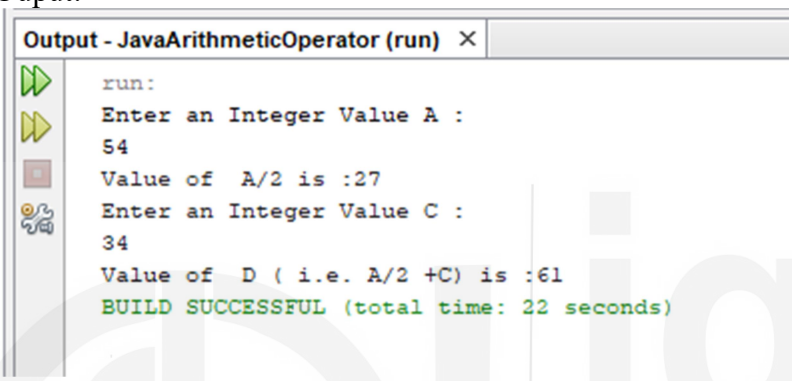
| Operator | Use | Description |
|---|---|---|
| ++ "Post-increment" | A++ | Post-increment: The value is assigned before the increment is made, e.g.<br>A = 5;<br>B = A++;<br>Then B will hold 5, and A will hold 6 |
| -- "Post-decrement" | A-- | Post-decrement: The value is assigned before the decrement is made, e.g. :<br>A = 2;<br>B = A--;<br>Then B will hold 2, and A will hold 1. |
| ++ "Pre-increment" | ++A | Pre-increment: The value is assigned after the increment is made, e.g.<br>A = 1;<br>B = ++A;<br>Then B will hold 2, and A will hold 2. |
| -- "Pre-decrement" | --A | Pre-decrement: The value is assigned after the decrement is made, e.g.<br>A = 5;<br>B = --A;<br>Then B will hold 4, and A will hold 4. |

Example Program: Use of Arithmetic Operators

```
package javaarithmeticoperator;
import java.util.Scanner;
public class JavaArithmeticOperator
{
public static void main(String[] args)
 {
int A,B,C,D,E;
Scanner s=new Scanner (System.in);
System.out.println ("Enter an Integer Value A : ");
A=s.nextInt ( ) ;
```

```
B= A/2;
System.out.println ("Value of  A/2 is :" + B);
System.out.println ("Enter an Integer Value C : ");
C=s.nextInt ( ) ;
D= B+C;
System.out.println ("Value of  D ( i.e. A/2 +C) is :" + D);
}
}
```

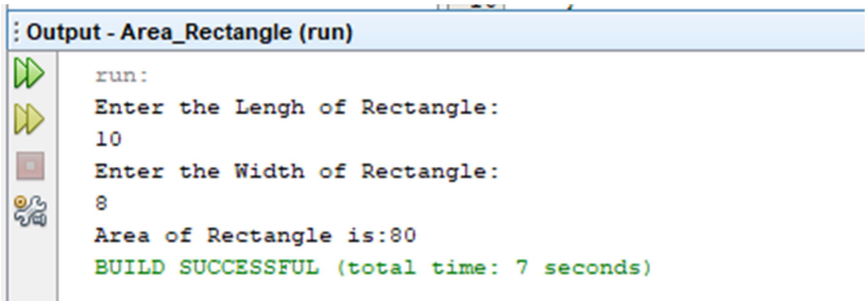
Ouput:




Example Proram: Find Area of Rectangle

```
package area_rectangle;
import java.util.Scanner;
public class Area_Rectangle
{
public static void main(String[] args)
 {
int Width, Length,Area;
Scanner s=new Scanner (System.in);
System.out.println ("Enter the Lengh of Rectangle: ");
Length = s.nextInt ( ) ;
System.out.println ("Enter the Width of Rectangle: ");
Width = s.nextInt ( ) ;
Area = Length * Width;
System.out.println ("Area of Rectangle is:" + Area);
}
}
```

Output:




## Relational Operators

Relational operators, also called comparison operators, are used to determine the relationship between two values in an expression. As given in *Table 2.9*, the return value depends on the operation.

**Table 2.9: Use of relational operators**

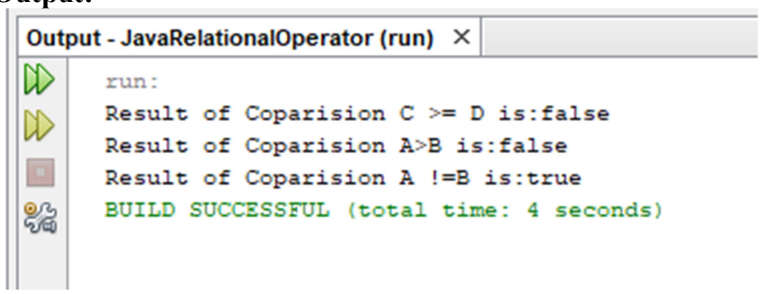
| Operator | Use | Returns true if |
|---|---|---|
| > | A > B | A is greater than B |
| >= | A >= B | A is greater than or equal to B |
| < | A < B | A is less than B |
| <= | A <= B | A is less than or equal to B |
| == | A == B | A and B are equal |
| != | A != B | A and B are not equal |

Program: Use of Relational Operators

```
package javarelationaloperator;
public class JavaRelationalOperator
{
   public static void main(String[] args)
 {

int A, B, C,D,E;
boolean i,j,k;
A =6;
B=4;
C=5
D=6
i = C>=D;
j = A<B;
k = A !=B;
System.out.println("Result of Comparision C >= D is:"+i);
System.out.println("Result of Comparision A>B is:"+j);
System.out.println("Result of Comparision A !=B is:"+k);
}
}
```

**Output:**




## Boolean Operators

Boolean operators allow you to combine the results of multiple expressions to return a single value that evaluates to either true or false. *Table 2.10* describes the use of each boolean Operator.

**Table 2.10: Description of Boolean operators**

| Operator | Use | Description |
|---|---|---|
| && | A && B | Conditional AND :If both A and B are true, result is true. If either A or B are false, the result is false. But if A is false, B will not be evaluated.<br>For example, (A > 5 && B <= 10) will evaluate to true if A is greater than 5, and B is less than or equal to 10. |
| \|\| | A \|\| B | Conditional OR: If either A or B are true, the result is true. But if A is true, B will not be evaluated.<br>For example, (A > 10 \|\| B > 5) will evaluate to true if either A is greater than 10 or B is greater than 5. |
| ! | ! A | Boolean NOT: If A is true, the result is false. If A is false, the result is true. |
| & | A & B | Boolean AND: If both A and B are true, the result is true. If either A or B are false, the result is false, and both A and B are evaluated before the test. |
| \| | A \| B | Boolean OR: If either A or B are true, the result is true. Both A & B are evaluated before the test. |
| ^ | A ^ B | Boolean XOR: If A is true and B is false, the result is true. If A is false and B is true, the result is true. Otherwise, the result is false. Both A and B are evaluated before the test. |

## Bitwise operators

As you know, in computers, data is represented in binary (1's and 0's) form. The binary representation of the number 43 is 0101011. The first bit from right to left in the binary representation is the least significant, i.e., the value is 1 here. Each Bitwise Operator allows you

to manipulate integer variables at bit level. *Table 2.11* given below, describes the use of the bitwise Operator with the help of a suitable example for each.

**Table 2.11: Description and use of Bitwise operators**

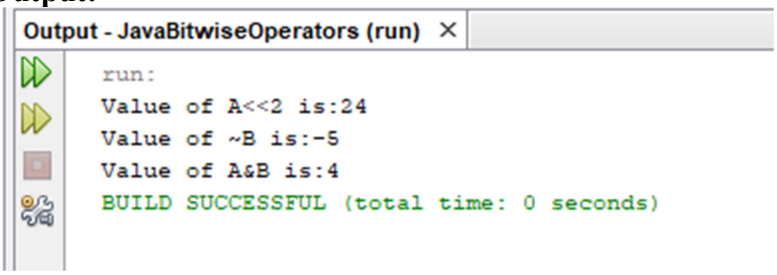
| Operator | Use | Description |
|---|---|---|
| >> | A >> B | Right shift: Shift bits of A right by distance B, 0 is introduced to the vacated most significant bits, and the vacated least significant bits are lost. |
| << | A << B | Left shift: Shift bits of A left by distance B , the most significant bits are lost as the number moves left, and the vacated least significant bits are 0. |
| >>> | A >>> B | Right shift unsigned: Shift A to the right by B bits. Low-order bits are lost. Zeros fill in left bits regardless of sign example. |
| ~ | ~B | Bitwise complement: The bitwise Complement changes the bits. 1, into 0, and bit 0, to 1. |
| & | A & B | Bitwise AND: AND is 1 if both the bits are 1; otherwise AND is 0. The bitwise AND is true only if both bits are set. |
| \| | A \| B | Bitwise OR: The bitwise OR is true if either bits are set. Or if the bit is 1 if either of the two bits is 1; otherwise, it is 0. |
| ^ | A ^ B | Bitwise Exclusive OR: The bitwise Exclusive OR is true if either bits are set, but not both. XOR of the bits is 1 if one bit is 1 & other bit is 0; otherwise, it is 0. |

Example Proram: Use of Bitwise Oprators

```
package javabitwiseoperators;
public class JavaBitwiseOperators
{
 public static void main(String[] args)
 {

    int A, B, C,D,E;
    A =6;
    B=4;
    C= A<<2;
    D = ~B;
    E = A&B;
    System.out.println("Value of A<<2 is:"+ C);
    System.out.println("Value of ~B is:"+ D);
    System.out.println("Value of A&B is:"+E);
 }
}
```

**Output:**




## Assignment Operators

The basic assignment operator '=' is used to assign value to the variables. For example, I = J; in which we assign the value of J to I. Similarly, let us see how to assign values to different data types.

```
int  IntVal = 50;
float FloatVal = 15.5;
char CharacrcterVal = 'P';
Boolean BooleanVal = true;
```

Along with basic assignment operators, Java has defined shortcut assignment operators that allow you to perform arithmetic, shift, or bitwise operation with one Operator. For example, if you want to add a number to a variable and assign the result back into the same variable, then you will write something like i = i + 2;

but using shortcut operator '+=' , you can shorten this statement, like
i += 2
Here remember, i = i + 2; and i += 2; both statements are the same for the compiler.

As given in *Table 2.12*, Java programming language provides different *shortcut assignment operators*:

**Combined Assignment Operators**

In an assignment operator, the same variable may appear on either side. To simplify the assignment, Java has provided a set of arithmetic assignment operators which are listed in Table 2.12.

**Table 2.12: Assignment arithmetic operators**

| Sl. No. | Operator | Meaning |
|---|---|---|
| 1. | += | Addition Assignment |
| 2. | -= | Subtraction Assignment |
| 3. | *= | Multiplication Assignment |
| 4. | /= | Division Assignment |
| 5. | %= | Modulus Assignment |

One variable named *var* is initialized as
int *var* = 50;
to realize the assignment arithmetic operation. Here the variable *var* has the value of 50
.
When the following statements make use of the assignment arithmetic operators
*var* = *var* + 1 (or) *var* + = 1;

The above notations are equal, and the usage of these notations is left to the discretion of the programmer. The value of *var* is 50; it is added with 1, and the results of the above statements assign 51.

The next operator is -= which is used as follows: *var* - = 5;

The value of the variable *var* is 51 is subtracted by 5, and the result 46 is assigned to *var*.

The next Operator *= is used in the following statement
*var* * = 3 ;

The value of the variable *var* is 46 multiplied by 3, and the result 138 is assigned to a.

The next operator /= is used in the following statement
*var* / = 4 ;

The value of the variable *var* is 138 which is divided by 4; the result is 34.5, and the integer part of the result, that is, 34 is assigned to *var* . Therefore *var* will have a value 34.

The next operator %= is then applied in the following statement

a % = 5;;
The value of the variable *var* is 34, and modulus operation with 5 actually divides the 34 by 5 and gives the remainder 4, which is assigned to *var*.

## Multiple Assignments

Multiple assignments are possible in the following format:
IdentifierA= Identifier B= Identifier C=…….= Expression
In this case, all the Identifiers are assigned the value of Expression.

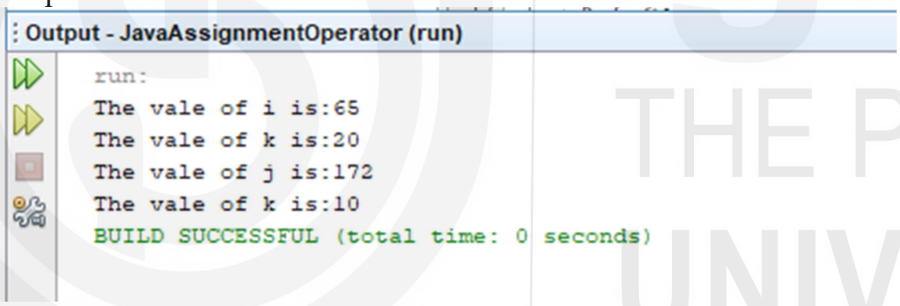
Let's see one example:

P=R=501+101+x+y+z; this is the same if you write:
P=501+101+x+y+z;
Q=501+101+x+y+z;
R=501+101+x+y+z;

Example Program

```
package javaassignmentoperator;
public class JavaAssignmentOperator
{
 public static void main(String[] args)
 {
 int i,j,k,l;
 float p,q,r;
 i= 45;
 k=20;
 i +=20;
 System.out.println("The vale of i is:"+i);
 j = ++i +k;
 System.out.println("The vale of k is:"+k);
 j *=2;
 System.out.println("The vale of j is:"+j);
 k /=2;
 System.out.println("The vale of k is:"+k);
 }
 }
```

Output:




## Class and Object Operators

The use of class and object operators is given in the following Table 2.13. Here these operators are listed for the sake of completeness of the topic. Use of some of these operators you will see in unit 4 of this Block.

**Table 2.13: Class and Object Operators**

| Operator | Name | Description |
|---|---|---|
| instance of | Class Test Operator | The first operand must be an object reference. For example, 'A instance of B', Returns true if A is an instance of B. Otherwise, it return false. |
| new | Class Instantiation | Creates a new object. For example, new A, in this A is either a call to a constructor or an array specification. |

| | | |
|---|---|---|
| "." | Class Member Access | It accesses a method or field of a class or object. For example, A.B is used for 'field access for object A', and A.B() is used for 'method access for object A' |
| ( ) | Method Invocation | For example, A(parameters) , Declares or calls the method named A with the specified parameters. |
| (type) | Object Cast | (type) A, in this example () operator Cost (convert) A to a specific type. An exception will be thrown if the type of A is incompatible with the specified type. In this type can be an object or any primitive data type. |

### Check Your Progress- 2

1) What is the use of relational operators? Explain with the help of an example.
……
……
……

2) What is a combined assignment operator? Explain with the help of an example.
……
……
……
……

3) Explain how to use & and | operators with the help of a program.
……
……
……
……

## 2.6 EXPRESSIONS AND STATEMENTS

Here we will learn about the statements and expressions. Expressions are segments of code that perform some computations and return some values. Expressions are formed by combining literals, variables and operators. Certain expressions can be made into statements, which are complete units of execution. In a program, statements are normally executed in the sequence in which they appear in the program. But there are number of situations where you may have to change the order of execution of statements based on certain conditions, or some tie you need to

repeat a group of statements until the certain specified condition(s) are met. You may find more details on statements in unit 3 of this Block.

### 2.6.1 Expressions

An expression is a series of variables, operators and method calls that evaluates to a single value. Also, sometimes you can write compound expressions by combining expressions as long as the types required by all of the operators involved in the compound expression are compatible/ correct. The Java programming language allows programmers to construct compound expressions and statements from smaller expressions. In Java, it is allowed as long as the data types required by one part of the expression match the data types of the other. Let us consider the following example of a compound expression:

P = A *B / C;

In this example, the order in which the expression is evaluated is unimportant because the results of multiplication and division are independent of order. However, it is not true for all expressions. For example, where addition/subtraction and division/multiplication operations are involved. You will get different results depending on the order of operation ( i.e. whether you perform the addition or the division operation first). Such expressions are called ambiguous expression.

For example

R= A + B / 50; //ambiguous expression

As a programmer you can specify exactly how you want an expression to be evaluated by using balanced parentheses '(' and ')'.

For example, you could write:

R= (A + B)/ 50;  //unambiguous and recommended way

Suppose as a programmer; you don't explicitly indicate the order of the operations in a compound expression to be performed. In that case, it is determined by the precedence assigned to the operators and the operators which have higher precedence get evaluated first.

Let us consider the above example again; the division operator has higher precedence than the addition operator. Thus, the two following statements are equivalent:

R= A + B / 50;

R= A + (B / 50);

Therefore, it is better to write compound expressions, using parentheses to specify which operators should be evaluated first. This also will make your code easier to read and maintain. The following table 2.14 shows Operator's precedence

**Table 2.14: Operators Precedence**

| | |
|---|---|
| Special operators | [ ] Array element reference<br>Member selection<br>(params) Function call |
| unary operators | ++, --, +, -, ~, ! |
| Creation or cast | new (type)expression |

| | |
|---|---|
| multiplicative | *, /, % |
| Additive | +, - |
| Shift | <<, >>, >>> |
| Relational | <, >, <=, >=, instance of |
| Equality | = =, != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | \| |
| logical AND | && |
| logical OR | \|\| |
| Conditional | ? : |
| Assignment | =, += ,-=, *= ,/=, %=, &=, ^=, \|=, <<= >>=, >>>= |

### 2.6.2 Statements

In a program a statement is a complete unit of execution. Statements in Java are equivalent to sentences in natural languages. It is an executable combination of tokens ending with a semicolon (;) mark. A token of expression can be any variable, constant, Operator or expression. By terminating the expression with a semicolon (;), the following types of expressions can be made into a statement:

- Assignment expressions
- Any use of ++ or --
- Object creation expression
- Method calls

Let us consider one example of expression statements:

```
float piValue = 3.141; //assignment statement

counter++; //increment statement

System.out.println(piValue); //method call statement

Integer integerObject = new Integer(4); //object creation statement
```

In addition to these kinds of expression statements, there are two other kinds of statements.

- A declaration statement declares a variable such as:

  ```
  int counter;

  double xValue = 55.24;
  ```

- A control statement regulates the flow of execution of statements based on the changes to the state of a program.

  The while loop, for loop and *if statement* is examples of control flow statements. In this category of statements, you will learn Unit 3 of this Block.

**Assignment Statement**

The assignment statement assigns the values to variables. The equal (=) sign is used as the assignment operator. The assignment statement has the following format:

Variable = constant (or) variable (or) expression;

The assignment operator has a right-hand side (RHS) and a left-hand side (LHS). The LHS must be a variable, and the RHS may have a constant or variable or expression.

If the assignment statement has a constant on the LHS the value is assigned to the RHS variable. If it has a variable on the RHS, the constant is assigned to the LHS variable. Suppose it has a variable on the RHS; the value of the variable is assigned to the LHS variable. If the RHS has an expression it is evaluated, and the result is assigned to the LHS variable. The following statement assigns values to the variables

- StudentName = "Mohit";
- AccountBalance = previousBalance;
- Area = pi * radius * radius;

The first statement is to assign a string constant to the variable StudentName, the second assigns the value of previousBalance to the variableAccountBalance, and the last statement calculates the result by multiplying the values of pi, and radius twice and assigns it to the variable Area.

Also, while writing a program sometimes the same variable may appear on both the RHS and LHS as follows :

count =20;
count = count + 1 ;
It is an assignment statement where the RHS is evaluated first by taking the present value of the variable count; for example, it is 20, and it is added with the constant 1, and the result 21 is assigned to the variable count on the LHS. Now the value of the count is 21.

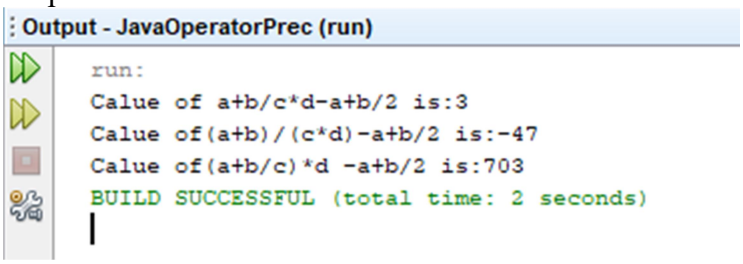
Example Program:

```
package javaoperatorprec;
public class JavaOperatorPrec
{
   public static void main(String[] args)
   {
      int a , b, c, d,,e;
      a= 50;
      b=6;
      c=12;
      d=15;
      e = a+b/c*d-a+b/2;
      System.out.println("Calue of a+b/c*d-a+b/2 is:"+e);
      e = (a+b)/(c*d)-a+b/2;
      System.out.println("Calue of(a+b)/(c*d)-a+b/2 is:"+e);
      e = (a+b/c)*d -a+b/2;
```

```
        System.out.println("Calue of(a+b/c)*d -a+b/2 is:"+e);
    }

}
```

Output:




## 2.7 EXAMPLE PROGRAM USING SCANNER CLASS

Java Scanner class provides many methods which a programmer may use to read inputs of different types. This class is part of import java.util package.

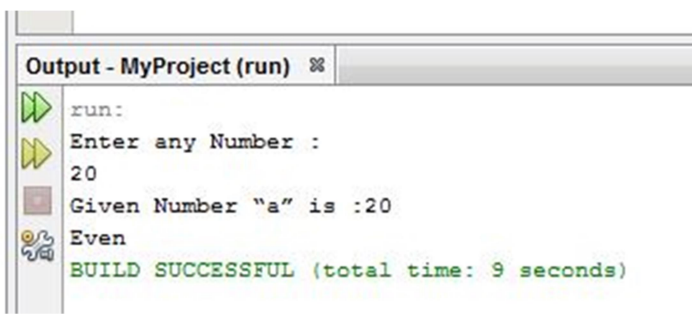
**Table 2.14: Methods of Scanner Class**

| Method | Description |
|---|---|
| nextInt() | Use to read an int value from the user |
| nextFloat() | It reads a float value given by the user |
| nextBoolean() | Used for reading a boolean value from the user |
| nextLine() | Used for reading a line of text (String) from the user |
| next() | It reads a word from the user |
| nextByte() | Used for reading a byte value from the user |
| nextDouble() | Used for reading a double value from the user |
| nextShort() | Used for reading a short value from the user |
| nextLong() | Used for reading a long value from the user |

The following Java Program print whether the given number is odd or even. Using method nextInt() input number is taken from the user.

```
//Program to print the odd or even numbers using objects
import java.util.Scanner;
class EvenOdd
{
int a;
public void get ( )
{
Scanner s=new Scanner (System.in);
System.out.println ("Enter any Number : ");
a=s.nextInt ( ) ;
System.out.println ("Given Number “a” is :" + a);
}
public void display ( )
{
if (a%2==0)
{
System.out.println ("Even");
}
else
{
System.out.println ("Odd");
}
}
public static void main (String[ ] args)
{
EvenOdd e=new EvenOdd();
e.get ( ) ;
e.display ( );
}
}
```

When you run the above java program, it will give the output as follows:



For a better understanding of the use of scanner class, some more examples are also given in the next unit of this Block.

### Check Your Progress- 3

1) What will be the output of the following Java program if 7 is given as input?

```
package circlearea;
import java.util.Scanner;
public class CircleArea
{
public static void main(String[] args)
 {
int radius;
float area, pi;
pi= 3.14F;
Scanner s=new Scanner (System.in);
System.out.println ("Enter the Radius of Circle: ");
radius = s.nextInt ( ) ;
area = pi * radius * radius;
System.out.println ("Area of Circle:" + area);
}
}
```

……

……

……

……

2) Write a Java program that takes the user's name, address and mobile number as input and displays it?

……

……

……

…….

## 2.8 SUMMARY

This unit has described to you the different basic data types in Java. Basic data types include four groups Integer, Floating Point Numbers, Characters and Boolean. This unit explained Unicode's use and different Java keywords that have special meanings for the compiler and cannot be used as a user-defined identifier. Before using the Java variables in your program, you should declare them and assign them value. You learnt how to declare variables and use them. You have learnt different types of operators in Java. You have also learnt about the use of expressions and

statements. Expression is a series of variables, operators, and method calls that evaluates to a single value. Statements in Java are equivalent to sentences in natural languages. A statement is a complete unit of execution. Toward the end of this unit, you learned to use some Scanner class methods.

## 2.9 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

### Check Your Progress 1

1)

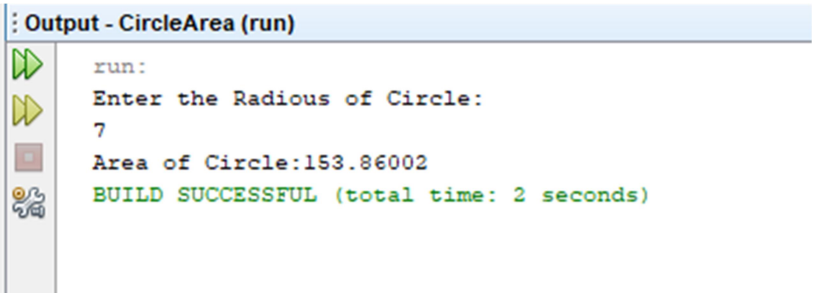
A literal represents a value of a certain type, and the type describes how that value behaves. Primitive types in Java are called **literals**. A literal is the source code representation of a fixed value in memory. There are different types of literals, namely number literals, character literals, boolean literals, and string literals.

2)

Program:

```
package circlearea;
import java.util.Scanner;
public class CircleArea
{
public static void main(String[] args)
 {
final float  pi = 3.14F;
int r;
float area ;
Scanner s=new Scanner (System.in);
System.out.println ("Enter the Radius of Circle: ");
r = s.nextInt ( ) ;
Area = pi * r * r;
System.out.println ("Area of Circle:" + area);
}
}
```

Output:



3)

In Java, boolean data type is used to define variables for use as logical variables. A boolean variable can have either true or false values. The boolean variable has only two values: true or false. These two values are boolean constants.

## Check Your Progress- 2

1)

Relational operators are used to find the relations between two variables(operands). For example, suppose you need to find the relation between the Marks of two students in MCS_206 you can use the relational operators.

To check whether the marks are equal you can use == Operator. Similarly, you may use other operators such as <, > etc.

2)

A combination of assignment operators is a combination of any arithmetic operator( +, -,* etc.) with the assignment operator(=).

For example, if you have an integer variable a and suppose it has a value of 20, then

a *=2;

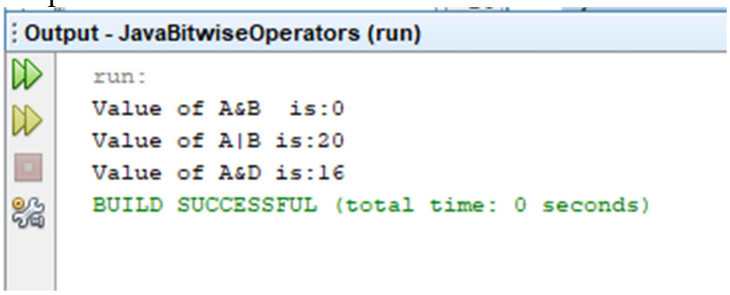
It will be similar to a = a*2;

3)

The following program show how to use & and | operators:

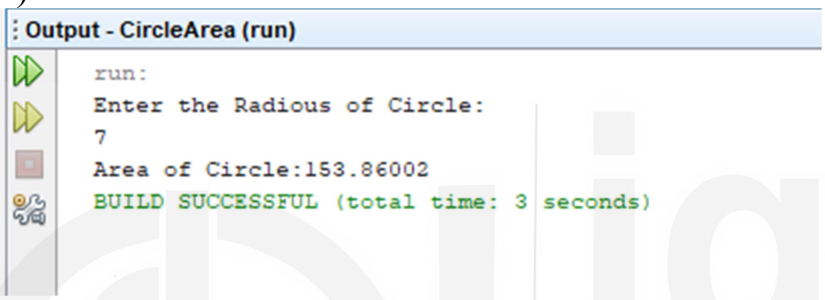
```
package javabitwiseoperators;
public class JavaBitwiseOperators
{
 public static void main(String[] args)
 {
int A, B, C,D,E;
A =16;
B=4;
C= A&B;
D = A|B;
E = A&D;
System.out.println("Value of A&B  is:"+ C);
System.out.println("Value of A|B is:"+ D);
System.out.println("Value of A&D is:"+E);
 }
}
```

Output:




## Check Your Progress- 3

1)



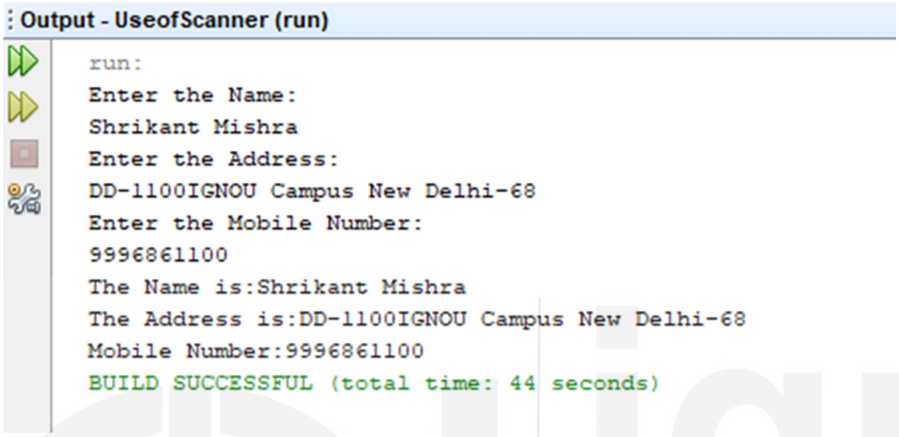

2)

Java Program

```
package useofscanner;
import java.util.Scanner;
public class UseofScanner
{
public static void main(String[] args)
 {
String Name,Address;
long Mobile_No;
Scanner s=new Scanner (System.in);
System.out.println ("Enter the Name: ");
Name= s.nextLine ( ) ;
System.out.println ("Enter the Address: ");
Address = s.nextLine ( ) ;
System.out.println ("Enter the Mobile Number: ");
Mobile_No = s.nextLong ( ) ;
```

```
System.out.println ("The Name is:" + Name +'\n' + "The Address is:"+ Address +'\n'+ "Mobile Number:"+Mobile_No);
}
}
```

Output:




## 2.10 REFERENCES/FURTHER READINGS


- S.Sagayaraj,R. Denis, P.Karthik and D.Gajalakshmi, "Java Programming for Core and Advanced Learners", University Press, 2018.
- Herbert Schildt, "Java The Complete Reference", McGraw-Hill,2017.
- Savitch, Walter, " Java: An introduction to problem solving & programming", Pearson Education Limited, 2019.
- CAY S.Horstmann, " Core Java Fundamentals", Pearson, Tenth Ed. , 2017.
- MCS-024 Course of IGNOU BCA(Revised ) Programme, Block - Unit-
- https://www.w3schools.com/java/
- http://www.cs.unc.edu/~weiss/COMP14/18-JavaKeyWords.html
- https://www.w3schools.com/java/java_ref_keywords.asp
- https://www.programiz.com/java-programming/scanner