Data Types and Operators in Java

OBJECTIVES OF THIS CHAPTER

5.1 DATA TYPES

Data type defines which type of data will be stored in the program elements, such as variables, arrays etc. Data types determine a specific type, range of values and the operations that can be performed on data.Java is a strongly typed language therefore data type of all the variables must be declared during declaration.

Java provides various data types and each data type is represented differently within computer's memory. The Data types provided by Java can be categorized broadly into two types:

- Primitive Data Types
- Non-Primitive Data Types

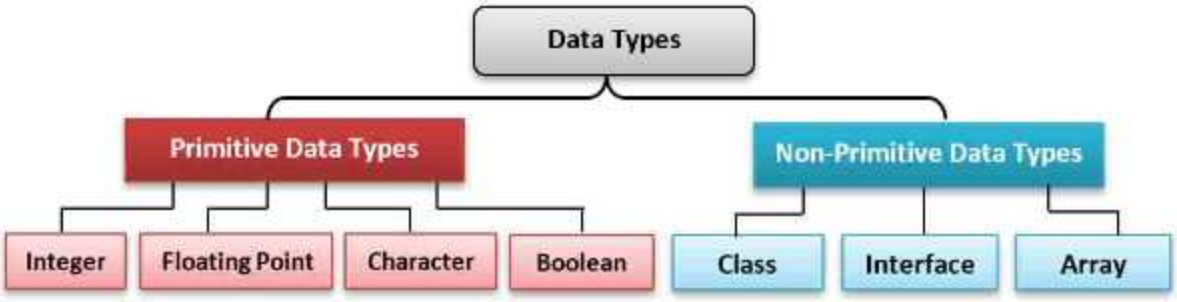


Fig 5.1: Classification of Data Types in Java

5.1.1 Primitive Data Types : Primitive data types are also known as built-in data types. Java has four main primitive data types built into the language.

1. ***Integer:*** byte, short, int and long

II. ***Floating Point:*** float and double

III. ***Character:*** char

IV. ***Boolean:*** boolean variable with a value of true or false

These data types have been described as following in detail.

I. Integer Type: Integer data type is used to store integer values (numbers without fractional part) like 49, 174, +2901, -54896 etc. Java support four types of integer data type: byte, short, int and long. All of these data types are signed as positive or negative. There is no concept of unsigned integer in java. The default value for these types is 0. These types have different storage capacities.

Following table shows the description about the memory requirements and range of values for the integer type data types:

Data Type	Storage Size	Range of Data	
byte	1 byte	-2^7 to $+2^7 - 1$	OR (-127 to +128)
short	2 bytes	-2^{15} to $+2^{15} - 1$	OR (-32,768 to +32,767)
int	4 bytes	-2^{31} to $+2^{31} - 1$	OR (-2,147,483,648 to (+2,147,483,647)
long	8 bytes	-2^{63} to $+2^{63} - 1$	OR (-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807)

Table: 5.1 Summary of Integer Data Types

We can select an appropriate data type for a variable depending on the type and range of integer value required.

II. Floating-Point Type : Floating-point data type is used to store real numbers such as 3.14, 74.5884569, +29.05, -524836.458 etc. Java supports two floating point data types: float and double.Following table shows the description about the memory requirements and range of values for the floating-point type data types:

ਡਾਟਾ ਟਾਈਪ	ਸਟੋਰੇਜ਼ ਸਾਈਜ਼	ਡਾਟਾ ਦੀ ਰੇਂਜ਼
float	4 bytes	3. 4E–38 to 3. 4E+38
double	8 bytes	1.7E–308 to 1. 7E+308

Table: 5.2 Summary of Floating-Point Data Types

Data type float represents the a single-precision number while double type represents the double-precision number. Single-precision number occupies lesser space than double precision. Single-precision value becomes inaccurate when values are large. Therefore, double type floating-point is the best choice when we need to store large values. For example: If we want to store the value of marks obtained by students, float data type can be used. Similarly, when we are working with mathematical functions like sin(), cos(), sqrt() etc, we should use double data type. The default value of float data type is 0.0f and double data type is 0.0d.

III. Character Data Type : Character data type is used to store a single Unicode

character enclosed in single quotes. To represent the character data type, char keyword is used. It takes 2 bytes (16 bits) of memory space. The range of char data type is 0 to 65535. The default value of char data type is '\u0000'.

Data Type	Storage Size	Range of Data
char	2 bytes	0 to 65535

Table: 5.3Summary of Character Type Data Types

IV. Boolean Data Type : Boolean data type is used to store boolean values for the program variables. This data type accepts only two values: true or false. The keyword boolean is used to denote the boolean data type. It specifies 1 bit of information but its size can't be defined precisely. The default value of boolean data type is false.

5.1.2 Non-Primitive Data Types : Non-Primitive data types are also known as user-defined data types or reference types. These data types are derived from the primitive data types. Classes, Interfaces and Arrays are the Non-Primitive data types in Java.

5.2VARIABLES

A Variable is an identifier that represents a memory location. This memory location is used to store data/value. Data/value stored in these locations can be accessed using the variable name. Variables allow us to change their values during execution. Every variable must have a data type, which specifies the size and type of value that can be stored in the variable. To use the variables in the program, they must be declared.

5.2.1 Variable Declaration : Java is a strongly typed language. It means we must declare variables before using them in the program. If we use variables without declaring them, compiler will generate a syntax error. A variable declaration consists of two parts: data type and variable name (identifier). Following syntax is used to declare variable in the java program:

data_type variable_name;

Here, **data_type** tells the compiler what type of value is going to be stored in the variable, and **variable_ name** is the valid identifier which tells the compiler about the name of the variable which will be used to refer to the value stored in the variable. Consider the following example of variable declaration:

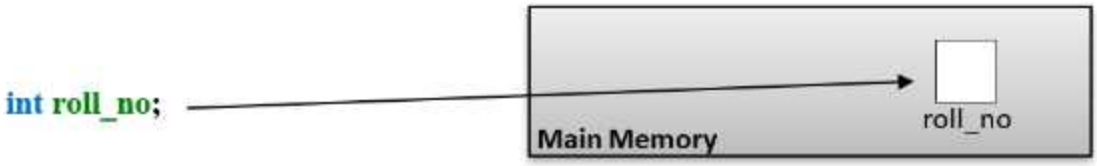


Fig: 5.2 Memory allocation to the Variable

Here, **int** represents the integer data type which allocated 4 bytes of memory to identifier **roll_no**. This identifier name is used to access the value stored in the variable,

whenever required. Here, the variable **roll_no** can hold only integer value. We can also declare multiple variables of same type by using the comma separator. For example:

```
int a, b, c;
```

5.2.2 Variable Initialization: Declaration of a variable only allocates memory to it but it does not store any data at the time of declaration.We can assign a value to a variable during its declaration, which is called variable initialization. For example:

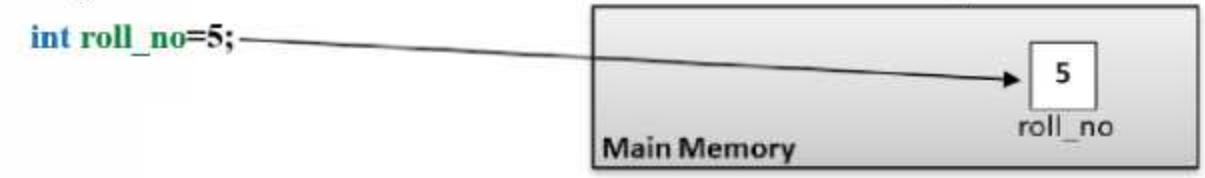
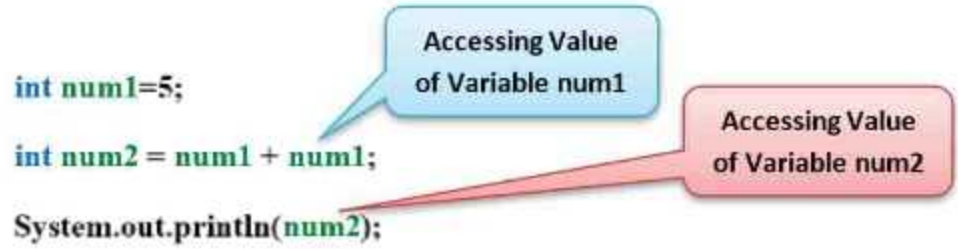


Table: 5.3 Value stored in the Variable

This assigned value can be changed later at any time because a variable allows us to change its values at any time during execution.

5.2.3 Accessing Value of a Variable: To access/use the value stored in the variable, we have to use its name. For example:



In above example, we declared and initialized two variables num1 and num2. Value of num1 variable has been used for calculating the value of num2, i.e. int num2=num1+num1. To show the value of num2, we use println() method of java.

5.3 OPERATORS

Operators are the symbols which are used to perform some specific type of operation on data. For example: + symbol is used to perform addition, * is used to perform multiplication, >= is used to perform comparison etc. Here +, * and >= are the operators to perform different types of operations. After performing the operations, all operators produce a value.

To perform any type of operation, we require Operands. **Operands** are the data items on which operators can perform operations. These operands can be either variables or constant values. Consider the following example:

```
a + 5 * 10
```

In this example, + and * are the operators which perform the operation on variable 'a' and constant values 5 and 10. Here the variable 'a' and constant values 5 and 10 are called the Operands. A valid combination of operators and operands is known as **Expression**.

Depending on the function performed by the operator, Java operators can be classified into various categories:Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operators, Bitwise Operators, Increment & Decrement Operators, Conditional Operators, and Special Operators.

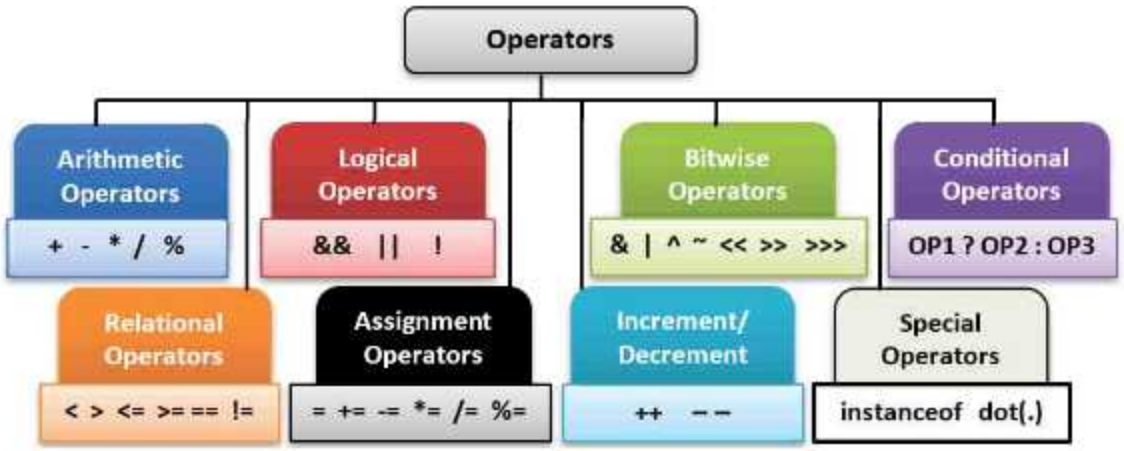


Fig: 5.4 Classification of Operators in Java

5.3.1 Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations such as: addition, subtraction, multiplication, division etc. There are five arithmetic operators in 'Java'. All these operators are binary operators because all these operators require two operands to perform their operations. Following table shows the list and working of all these operators:

Name	Operator	Description	Examplesof Operators on	
			Integer Values	Real Values
Add	+	Used to perform Addition of numbers.	2+4 → 6	2.0+4.0 → 6.0
Subtract	-	Used to perform subtraction or used as any unary minus.	6-2 → 4	6.0-4.0 → 2.0
Multiply	*	Used to perform multiplication of numbers.	7*2 → 14	7.0*2.0 → 14.0
Divide	/	Used to perform division of numbers.	5/2 → 2	5.0/2.0 → 2.5
Modulus	%	Used to get remainder value after division of numbers.	7%4 → 3	5.0%2.0 →1.0

Fig: 5.4 Arithmetic Operators

Following program shows how to use arithmetic operators in Java Programming:

Program 5.1: Use of Arithmetic Operators in Java

```
1  class Test1
2  {
3      public static void main(String args[])
4      {
5          float num1=5.0f, num2=2.0f, result;
6          result=num1+num2;
7          System.out.println("Result of Addition is "+result);
8          result=num1-num2;
9          System.out.println("Result of Subtraction is "+result);
10         result=num1*num2;
11         System.out.println("Result of Multiplication is "+result);
12         result=num1/num2;
13         System.out.println("Result of Division is "+result);
14         result=num1%num2;
15         System.out.println("Result of Modulus is "+result);
16     }
17 }
```

Compilation, Execution and Output of Program 5.1:

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac Test1.java

D:\Java Programs>java Test1
Result of Addition is 7.0
Result of Subtraction is 3.0
Result of Multiplication is 10.0
Result of Division is 2.5
Result of Modulus is 1.0

D:\Java Programs>
```

In the program 5.1, we declare three variables: num1, num2, and result of floating type. Variables num1 and num2 are initialized with float values 5.0f and 2.0f respectively. Then the calculation result of all arithmetic operators is stored in the result variable which has been displayed using the System.out.println() method of java. We can apply these operators on integer and floating-point operands.

5.3.2 Relational Operators

Relational operators are also called comparison operators. These operators are used to test the relationship between operands. In other words, these operators are used to compare values. After comparison, these operators return boolean value either **true** or **false**. All the relational operators present in Java language are of binary type. It means these operators require two operands to perform their operation. There are 6 relational operators in Java which are given below in the table with examples:

Name	Operator	Description	Example	Result
Equals to	==	Used to check whether two values are equal	4==5 5==5	False True
Not Equal to	!=	Used to check whether two values are not equal	4! =5 4! =4	True False
Greater than	>	Used to check whether the first value is greater than second	4>5 5>4	False True
Less than	<	Used to check whether the first value is less than second	4<5 5<4	True False
Greater than or equal to	>=	Used to check whether first value is greater than or equal to second value	5>=5 6>=8 10>=5	True False True
Less than or equal to	<=	Used to check whether first value is lesser than or equal to second value	4<=5 4<=2 4<=4	True False True

Table 5.5 Relational Operators

Program 5.2: Use of Relational Operators in Java

```
1  class Test2
2  {
3      public static void main(String args[])
4      {
5          int num1=5, num2=2;
6          boolean result;
7          System.out.println("Value of num1 is "+num1);
8          System.out.println("Value of num2 is "+num2);
9          result=num1==num2;
10         System.out.println("Result of num1==num2 is "+result);
11         result=num1!=num2;
12         System.out.println("Result of num1!=num2 is "+result);
13         result=num1>num2;
14         System.out.println("Result of num1>num2 is "+result);
15         result=num1<num2;
16         System.out.println("Result of num1<num2 is "+result);
17         result=num1>=num2;
18         System.out.println("Result of num1>=num2 is "+result);
19         result=num1<=num2;
20         System.out.println("Result of num1<=num2 is "+result);
21     }
22 }
```

Compilation, Execution and Output of Program 5.2:

```
C:\Windows\System32\cmd.exe
D:\Java Programs>javac Test2.java

D:\Java Programs>java Test2
Value of num1 is 5
Value of num2 is 2
Result of num1==num2 is false
Result of num1!=num2 is true
Result of num1>num2 is true
Result of num1<num2 is false
Result of num1>=num2 is true
Result of num1<=num2 is false

D:\Java Programs>
```

Usually relational operators are used in the expression that consists of control statements such as branching, looping and jumping statements. These statements have been explained in the next chapter of this book.

5.3.3 Logical Operators:

Logical operators are also called Boolean Operators. These operators are used to make compound relational expressions. In other words, we can say that these operators are used when we want to test more than one condition at a time. There are 3 Logical operators in Java: 'Logical AND', 'Logical OR' and 'Logical NOT'. Here, 'Logical AND' and 'Logical OR' are the binary operators whereas 'Logical NOT' is unary operator. Two operands are required for 'Logical AND' and 'Logical OR' to perform their operations while one operand is required for 'Logical NOT' to perform its operation.

All Logical Operators return boolean value either true or false. The result of a logical AND operator will return true only if both operands are true, whereas the result of a logical OR operator will be true if either operand is true or if both operands are true. Logical NOT operator returns true only when its operand is false. Following table shows the list and working of all Logical Operators used in Java language with appropriate examples:

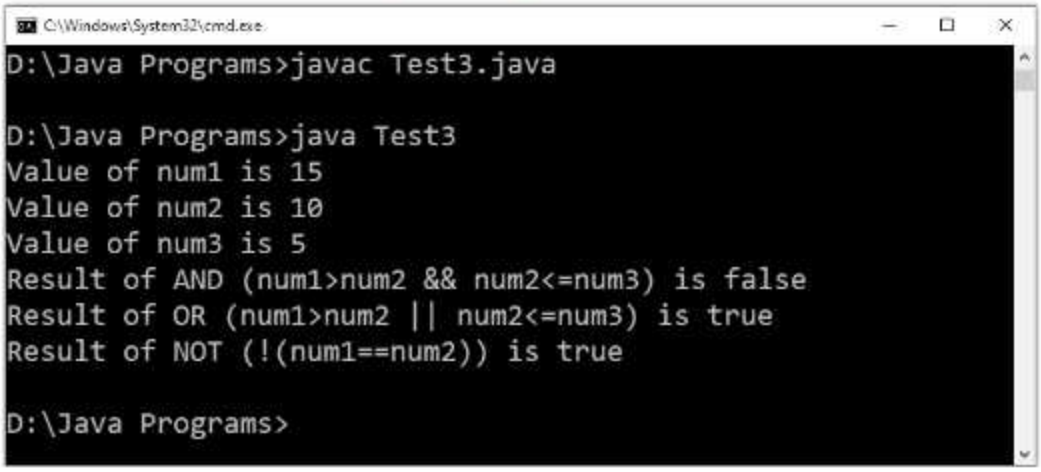
Name	Operator	Description	Example	Explanation and Result
AND	&&	Returns true only if both operands are true otherwise it returns false	3>5 && 4>5 3>5 && 4<5 3<5 && 4>5 3<5 && 4<5	False && False → False False && True → False True && False → False True && True → True
OR	||	Returns true if at least one of its operands is true otherwise it returns false	3>5 || 4>5 3>5 || 4<5 3<5 || 4>5 3<5 || 4<5	False || False → False False || True → True True || False → True True || True → True
NOT	!	Returns true only when its operand is false otherwise it returns false	!(3<5) !(3>5)	!(True) → False !(False) → True

Table 5.6 Logical Operators

Program 5.3 Use of Logical Operators in java

```
class Test3
{
    public static void main(String args[])
    {
        int num1=15, num2=10, num3=5;
        boolean result;
        System.out.println("Value of num1 is "+num1);
        System.out.println("Value of num2 is "+num2);
        System.out.println("Value of num3 is "+num3);
        result=num1>num2 && num2<=num3;
        System.out.println("Result of AND (num1>num2 && num2<=num3) is "+result);
        result=num1>num2 || num2<=num3;
        System.out.println("Result of OR (num1>num2 || num2<=num3) is "+result);
        result=!(num1==num2);
        System.out.println("Result of NOT !(num1==num2) is "+result);
    }
}
```

Compilation, Execution and Output of Program 5.3:



5.3.4 Assignment operators:

These Operators are used to assign or store values in variables etc. The symbol of assignment operator is =. Consider the following examples which show how to use assignment operator in Java programs:

```
int a = -2;          // assigns -ve value (-2) to the variable.
int b = 5;           // assigns value (5) to the variable.
int c = a + b;       // assigns the result of expression to the variable.
int a = a + 10;      // self-assignment of a variable.
```

Left hand side of assignment operator must be a valid identifier which represents a memory location. We cannot put the expression on the left side of the assignment operator. For example, following assignment statement would be invalid:

```
a +b=c;              //Invalid assignment statement
```

With the help of assignment operator, several variables can be assigned a common value. Consider the following example:

```
a = b = c = 5;       // value 5 will be assigned to three variables a, b and c
```

Assignment operators can also be used as **compound or shorthand assignment operators**. Shorthand assignment operators are useful for self-assignment statements. Following table shows the examples of shorthand assignment with arithmetic operators:

Let's assume `int a=5;`

Shorthand Operator	Example for Shorthand Assignment	Equivalent Self-Assignment	Result
+=	a+ =2	a = a + 2	a=7
- =	a- =2	a = a – 2	a=3
*=	a* =2	a = a * 2	a=10
/=	a / =2	a = a / 2	a=2
%=	a%=2	a = a % 2	a=1

Table 5.7: Shorthand Assignment for Arithmetic Operators

Assignment operator = and the equality (equal to) operator ==both are different types of operators. The assignment operator is used to assign a value to an identifier, whereas the equality operator is used to determine if two operands have the same value. These two operators cannot be used in place of each other.

5.3.5 Bitwise Operators:

Bitwise operators are used for performing bit level operations. They can be used with any integral type (char, short, int, etc.). Following table shows a summary of bitwise operators:

Assume **int A=60** and **int B=13** then:

Operator	Description	Example
& (Bitwise AND)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (Bitwise OR)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (Bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (Bitwise Compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (Left Shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (Right Shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (Zero Fill Right Shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Table 5.8: Bitwise Operators

5.3.6 Increment and Decrement Operators:

These are the unary operators. The symbol ++ is used for increment operator while symbol - - used for decrement operator. The increment operator causes its operand to increase by one whereas the decrement operator causes its operand to decrease by one. The operand used with each of these operators must be a single variable. These operators cannot be applied directly on the constant values.

For example:

```
int x=10;        (x is an integer variable that has been assigned a value of 10)
```

Following expression causes the value of x to be increased to 11

++x; (which is equivalent to x= x+1)

Similarly, following expression causes the original value of x to be decreased to 9:

--x, (which is equivalent to x=x-1)

If we use 10++ or --10, it will be a wrong statement as we have already studied that these cannot be applied directly on the constant values.

The increment and decrement operators can each be utilized in two different ways. It depends on whether the operator is written before or after the operand:

- Prefix increment and decrement
- Postfix increment and decrement

If the operator precedes the operand, then the value of operand will be altered before it is used for its intended purpose within the program. This is called **pre-increment/decrement**. If, however the operator follows the operand then the value of the operand will be changed after it is used. This is called **post-increment/ decrement**.

For example:

If the value of x is initially 10, it can be increased by two methods:

y = ++x; (pre-increment)

Here,at first, the value of x will be incremented to 11 and then this incremented value of x will be assigned to variable y, i.e. y will also get value 11 (i.e. x=11 and y=11)

y = x++; (post-increment)

Here, at first, the value of x will be assigned to variable y (i.e. y will get value 10) then value of x will be incremented to 11 (i.e. y=10 and x=11)

Similarly, decrement operator can be used. For example:

y = --x; (pre-decrement)

Here,first of all, the value of x will be decremented to 9 and then this decremented value of x will be assigned to variable y, i.e. y will also get value 9 (i.e. x=9 and y=9)

y = x--; (post-decrement)

Here, first of all, the value of x will be assigned to variable y (i.e. y will get value 10) then value of x will be decremented to 9 (i.e. y=10 and x=9)

5.3.7 Conditional Operator (? :)

It is a ternary operator. There is one and only one ternary operator in 'Java' language. This operator requires three operands to perform its operation. Ternary operator in Java is represented using **? :** symbols. The syntax for using this operator is given below:

*exp1***?***exp2***:***exp3*;

Here, exp1 must be a conditional expression which produces a result either true or false. If the value of exp1 is true then the exp2 will perform its function otherwise exp3 will perform its function. Consider the following example:

```
a=5;
b=10;
c =a > b ? a : b;
```

Here, the expression a>b will produce false (0) result (Operand1/exp1), therefore value of b (Operand3/exp3) will be stored in variable c. Variable a (Operand2/exp2) will not do any function because it will perform its function only if exp1 is true (1).

Program 5.4 Use of Conditional Operator in java

Test4.java

```
1  class Test4
2  {
3      public static void main(String args[])
4      {
5          int num1=15, num2=10;
6          int result;
7          System.out.println("Value of num1 is "+num1);
8          System.out.println("Value of num2 is "+num2);
9          result = num1>num2 ? num1 : num2;
10         System.out.println("Largest Number is "+result);
11     }
12 }
```

Compilation, Execution and Output of Program 5.4:

```
D:\Java Programs>javac Test4.java

D:\Java Programs>javac Test4.java

D:\Java Programs>java Test4
Value of num1 is 15
Value of num2 is 10
Largest Number is 15

D:\Java Programs>
```

5.3.8 Special Operators:

Java provides some special operators such as instance of and member selection operators:

- **instanceof operator:** This operator is used to check whether the object belongs o a particular class or not. It returns either true or false value depending on whether the object on left side of expression is an instance of the object on the right side. For example:
  mango **instanceof** fruit
  This statement returns true of the object mango belongs to class fruit, otherwise it returns false.
- **Member Selection Operator:** It is also known as dot(.) operator. This operator is used to access the variables and methods of a class using its objects:

```
Object.variable;        //accessing variable of a class through object
Object.method();        //accessing method of a class through object
```

5.4 EXPRESSIONS

An expression is like a formula in mathematics. An **expression** can be any valid combination of operators and operands. A valid combination is such a combination that confirms to the syntax rules of Java language. A valid expression is also known as well-formed-expression. An expression after evaluation always returns a single value. This result can be used in the Java programs. Expressions can be as simple as a single value and as complex as a large calculation. Consider the following examples:

```
x=2.9;
```

It is a simple expression where = operator is used with operands x and 2.9

```
x=2.9*y+3.6>z-(3.4/z);
```

It is a somewhat complex expression which consists of many operators and operands. Here, =, *, +, >, - and / are the operators and x, 2.9, y, 3.6, 3.4 and z are the operands.

Now, consider the following combination of operators and operands which do not form a valid combination to be an expression:

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

The above combination of operators and operands do not form a valid expression, although we are using valid operators and operands in the above example. But this combination of operators and operands do not follow the syntax rules of Java language to be a valid expression. Left side of = operator must represent a valid memory location (identifier) to store value of z. Here, x+y cannot be a valid identifier, because a valid identifier cannot have special character other than underscore (as we learnt in the previous chapter).

All Java expressions can be categorized broadly into following types:

5.4.1 Numerical Expressions:

These expressions are used to perform numerical calculations. These expressions always return a numerical value after evaluating operators and operands. Consider the following examples:

```
4+3*2
3.2-7.8
```

After evaluation of above given numerical expression, a numerical value 10 and -4.6 will be produced.

5.4.2 Logical or Conditional Expressions:

These expressions are used to perform logical or conditional operations. These expressions always return one of two possible values: either true or false. Consider the following examples:

```
14>6
15<=6
```

After evaluating above conditional expressions, we receive true for the first expression and false for the second expression.

5.5 PRECEDENCE/HIERARCHY AND ASSOCIATIVITY OF OPERATORS

The order or priority in which various operators in an expression are evaluated is known as **precedence**. Operators with a higher precedence are carried out before operators having a lower precedence. In simple words, the sequence of evaluation of operators in which they are applied on the operands in an expression is called the **Precedence of Operators**. For example, division is performed before the subtraction as division operator has higher precedence than the subtraction operator. The natural order can be altered by making use of parentheses ( ).

Precedence of commonly used operators in decreasing order and their Associativity is given below:

S.N.	Common Operators	Associativity
1.	. ( ) [ ]	Left to Right
2.	- ++ - - ! ~ (type) casting	Right to Left
3.	* / %	Left to Right
4.	+ -	Left to Right
5.	< <= > >= instanceof	Left to Right
6.	== !=	Left to Right
7.	&&	Left to Right
8.	||	Left to Right
9.	? :	Right to Left
10.	= *= /= %- += -=	Right to Left

Consider the following example which illustrates how arithmetic expressions are evaluated using operator's precedence:

```
a = 5 * 4 / 4 + 8 – 9 / 3;        (* is evaluated)
a = 20 / 4 + 8 – 9 / 3;           (/ is evaluated)
a = 5 + 8 – 9 / 3;                (/ is evaluated)
a = 5 + 8 - 3;                    (+ is evaluated)
a = 13 - 3;                       (- is evaluated)
a = 10;                           result of expression
```

The order in which operators of the same precedence are evaluated is known as **Associativity**. For example: addition and subtraction operators have the same precedence. However, addition and subtraction may be performed on the expression depending upon the order of their occurrence. The associativity of an operator can be

either from left to right or from right to left. The operators with left to right associativity are evaluated from the left end of the expression while the operators with right to left associativity are evaluated from right end of the expression. Consider the following examples:

For example:

```
a = b = c = 8;
```

The assignment operator has right to left associativity. It means that the value 8 is assigned to c, then c is assigned to b, and at last b is assigned to a.

5.5 TYPE CONVERSION

The value of an expression can be converted to a different data type in JAVA, if desired. When value of one type is converted into some other type, it is called Type Conversion.Operands that differ in type may undergo type conversion before the expression takes on its final value. There are two ways of type conversions in JAVA:

1. Implicit Conversion or Widening Conversion
2. Explicit Conversion or Narrowing Conversion

5.5.1 Implicit Conversion or Widening Conversion:

This type of conversion is **automatic**. For this type of conversion, we use assignment (=) operator. This type of conversion is used when operand having lower data type is converted into higher data type. If both the data types are compatible and the data type of target variable is large enough to store the value of the source variable, java automatically converts the source type into target type. There is no loss of information in this type of conversion. For example, value of int data type can be assigned to a variable of double data type since double type is larger than int type. Consider the following example for automatic conversion:

```
double n;
n = 5;
```

In this example, implicit conversion takes place, as integer type value (5) will automatically be converted into double type value so that it can be stored in the double type variable n i.e. value of variable n will become 5.0d

5.5.2 Explicit Conversion or Narrowing Conversion or Type Casting:

If the target type is smaller than the source type, conversion cannot be performed automatically. For example, value of double type cannot be stored in the int type variable. For such conversion, java provides a mechanism known as **type casting**. This is forceful conversion. For this type of conversion, we use **caste operator**. There may or may not be any loss of information in this type of conversion.

The syntax for this type of casting is:

```
(data type) variable or expression
```

The name of data type into which the conversion is to be made is enclosed in parentheses and placed directly to the left of the value to be converted.

For example:

```
double n=45.5;
int a=(int) n;
```

In this example, type casting is performed to convert double data type to int data type. Hence, value of integer type variable 'a' becomes 45, i.e. a=45

1. Data types determine a specific type, range of values and the operations that can be performed on data.
2. Primitive data types are also known as built-in data types.
3. Java support four types of integer data type: byte, short, int and long.
4. Floating-point data type is used to store real numbers such as 3.14, 74.5884569, +29.05, -524836.458 etc.
5. Character data type is used to store a single Unicode character enclosed in single quotes.
6. Boolean data type is used to store boolean values for the program variables. This data type accepts only two values: true or false.
7. Non-Primitive data types are also known as user-defined data types or reference types.
8. A Variable is an identifier that represents a memory location.
9. We can assign a value to a variable during its declaration, which is called variable initialization.
10. **Operands** are the data items on which operators can perform operations.
11. A valid combination of operators and operands is known as **Expression**.
12. Arithmetic operators are used to perform arithmetic operations such as: addition, subtraction, multiplication, division etc.
13. Relational operators are also called comparison operators. These operators are used to test the relationship between operands.
14. Logical operators are also called Boolean Operators.
15. Shorthand assignment operators are useful for self-assignment statements.
16. Increment (++) and Decrement (--) Operators are the unary operators.
17. Conditional Operator is a ternary operator.

18. The sequence of evaluation of operators in which they are applied on the operands in an expression is called the **Precedence of Operators**.
19. The order in which operators of the same precedence are evaluated is known as **Associativity**.
20. When value of one type is converted into some other type, it is called Type Conversion.

Exercise

Q:1 Multiple Choice Questions

i. _____ determine a specific type, range of values and the operations that can be performed on data.

a. Variable
b. Expression
c. Operand
d. Data Type

ii. Floating-point data type is used to store _____

a. Integers
b. Character
b. Real numbers
d. Boolean

iii. A _____ is an identifier that represents a memory location.

a. Variable
b. Literal
c. Integer
c. Operator

iv. _____ are the data items on which operators can perform operations

a. Operands
b. Literal
c. Data Types
c. Operator

v. _____ operators are used to test the relationship between operands.

a. Arithmetic
b. Unary
c. Relational
c. Ternary

vi. Increment (++) and Decrement (--) Operators are the _____ operators.

a. Unary
b. Binary
c. Ternary
c. None of these

vii. When value of one type is converted into some other type, it is called _____.

a. Data Conversion
b. Type Conversion
c. Value Conversion
c. Operator Conversion

Q:2 Fill in the Blanks

I. Java supports four types of integer data type: byte, short, int and _____.
ii. Character data type is used to store a single Unicode character enclosed in _____ quotes.
iii. A valid combination of operators and operands is known as _____.

iv. Logical operators are also called _____ Operators.
v. Conditional Operator is a_____ operator.
vi. The order in which operators of the same precedence are evaluated is known as _____.

Q:3 Short Answer Type Questions

i. Define Data types.
ii. Write the name of different types of Primitive Data Types used in Java.
iii. What is a Variable?
iv. What are Operands?
v. Write about the Arithmetic Operators of Java.
vi. What are assignment Operators?
vii. What do you know about Increment and Decrement Operators?
viii. What do you mean by precedence of Operators in Java?
ix. What is Type Conversion?

Q:4 Long Answer Type Questions

i. What is Expression? Explain different types of Expressions in Java.
ii. What are Relational Operators? Explain with suitable examples.
iii. Define Logical Operators. Explain with suitable examples.
iv. Write a program in Java which shows the use of Arithmetic Operators in Java.
v. How will we use Conditional Operator in Java? Explain with suitable example.
vi. What is Type Conversion? Explain different types of Type Conversion.