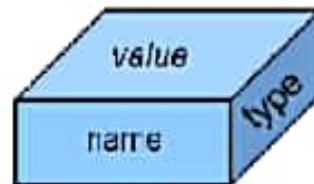# CHAPTER 4-OPERATORS AND VARIABLES IN JAVA

4.1 Objective

- Understand Java Variables
- Understand numeric data types, basic operations and expressions and operator precedence.
- Perform simple and complex numeric computations
- Ability to choose right numeric data types and order of computation
- Understanding this keyword

4.2 Course Content

4.2.1 Java Variables

- ✓ In Java, the attributes in classes are called as Variables.
- ✓ A variable is a container that holds values that are used in a Java program.
- ✓ Variable is declared with a type of data it stores.
- ✓ All variables should be declared before they are used in the program.
- ✓ As shown below, variables have a name, value and type.

Variable Diagram

4.2.2 Variable declaration

The basic form of a variable declaration is shown:

<data type> <variable name> = <initial value>

Example:

int age =20;

Here age is a variable of type integer with initial value of 20.

TCS Internal

```
Product.java ⌐

    public class Product {

        int id;
        float price;
        String name;


    }
```

Note:Product.java example which is been given that can not be executed as main method is not there.
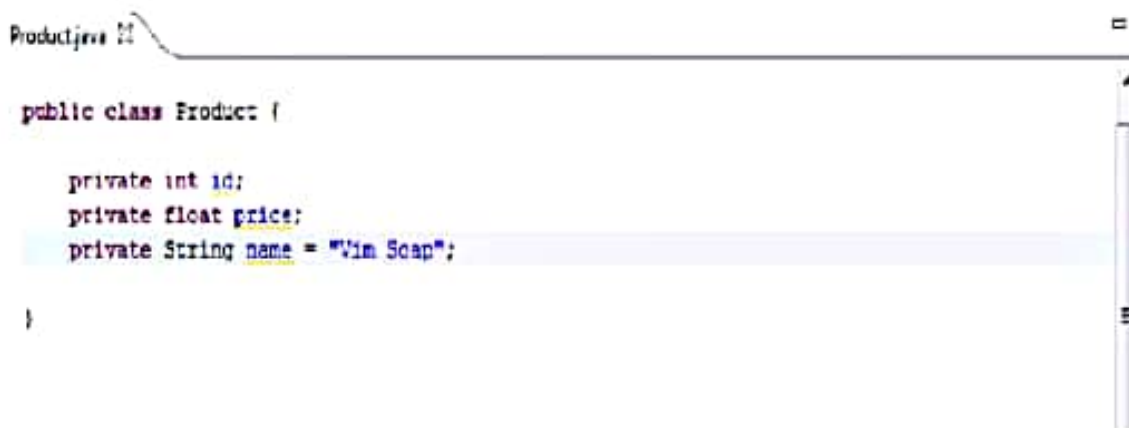
4.2.3 Types of Variables

Member / Instance variables:

- Instance variables are declared in a class, but outside a method, constructor or any block.

- Access modifiers such as private, public, protected can be specified for instance variables.

- They are visible for all methods, constructors and blocks in the class.

Values can be assigned during the declaration or within the constructor.

Example 1:
In this example we can see how to declare and assign initial value to the variables. We can also see the access modifiers used with the variable declaration. Here we have used private access modifier.
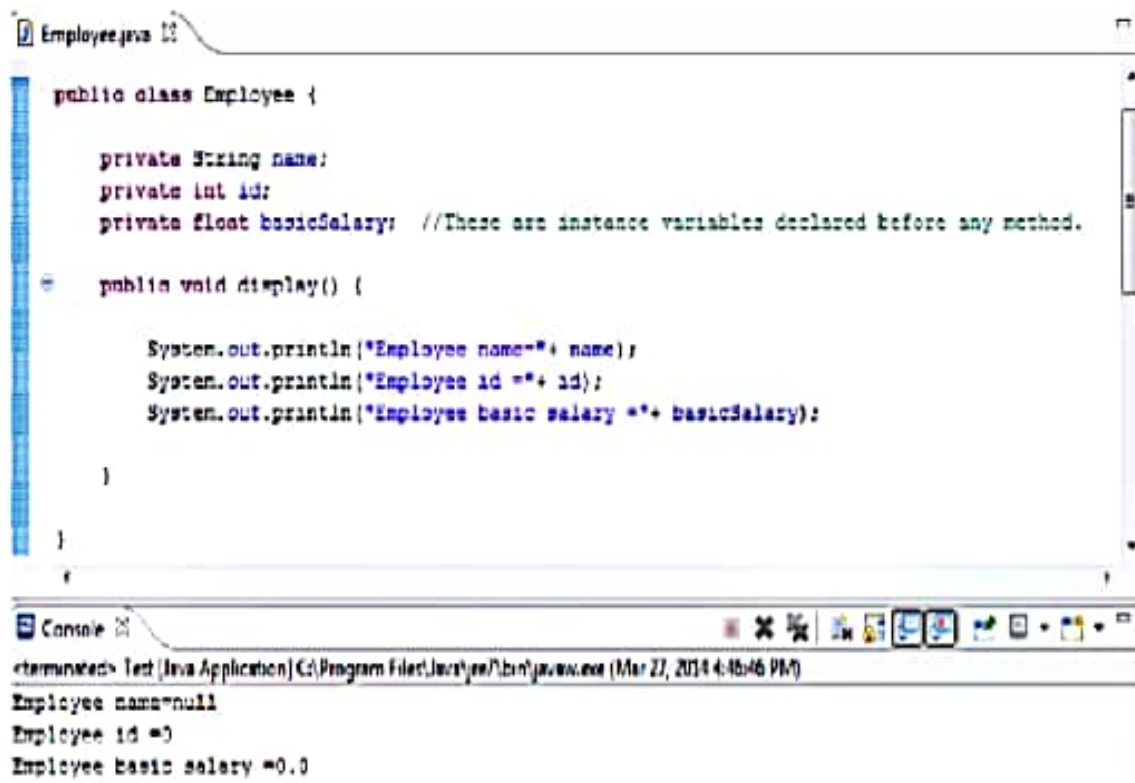
```
Product.java ⌐

    public class Product {

        private int id;
        private float price;
        private String name = "Vim Soap";


    }
```

Note:Product.java example which is been given that can not be executed as main method is not there.
Example 2:

This example demonstrates the default values of instance variables.

TCS Internal

```
Employee.java

public class Employee {

    private String name;
    private int id;
    private float basicSalary;   //These are instance variables declared before any method.

    public void display() {

        System.out.println("Employee name="+ name);
        System.out.println("Employee id ="+ id);
        System.out.println("Employee basic salary ="+ basicSalary);

    }

}

Console

<terminated> Test [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 27, 2014 4:46:46 PM)
Employee name=null
Employee id =0
Employee basic salary =0.0
```

Note:Employee.java example which is been given that can not be executed as main method is not there.

The default initial values of the data members of Employee class are as shown below:
  name is null (as it is a String object reference)
  id is 0(as it is primitive type : integer)
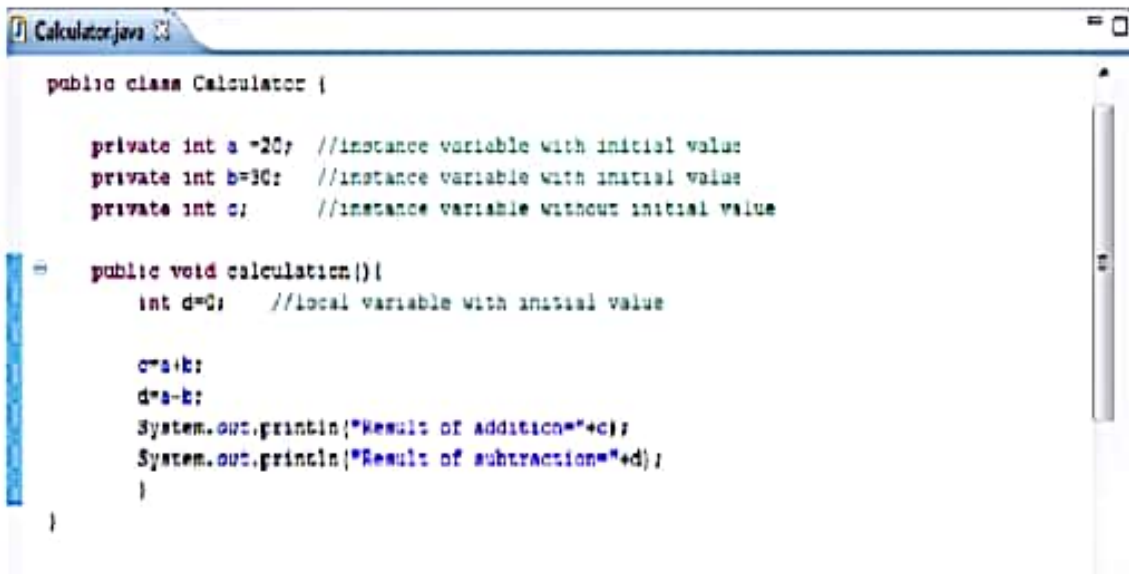 basicSalary is 0.0 (as it is primitive type : float)

Local variables

- Local variables are declared inside methods, constructors or blocks.

- Access modifiers cannot be used for local variables.

- They are visible only within the declared method, constructor or block.

- There is no default value for local variables. So local variables should be declared and an initialized before its first use.

Example:

This example demonstrates how to declare and use local variables.

TCS Internal

```
Calculator.java

public class Calculator {

    private int a = 20;    //instance variable with initial value
    private int b=30;      //instance variable with initial value
    private int c;         //instance variable without initial value

    public void calculation(){
        int d=0;    //local variable with initial value

        c=a+b;
        d=a-b;
        System.out.println("Result of addition="+c);
        System.out.println("Result of subtraction="+d);

    }

}
```

Note:Calculator.java example which is been given that can not be executed as main method is not there.

Here"d" is the local variable created inside the calculation method.

Parameters

Parameters are the list of arguments that come into the function. Parameters are used to pass values to functions and to receive values in a function.

Example: This example demonstrates to how to use parameters to receive values.

```
Calculator.java

public class Calculator {

    public void add(int a, int b){
        int c= a+b;
        System.out.println("Value of addition="+c);

    }

}
```
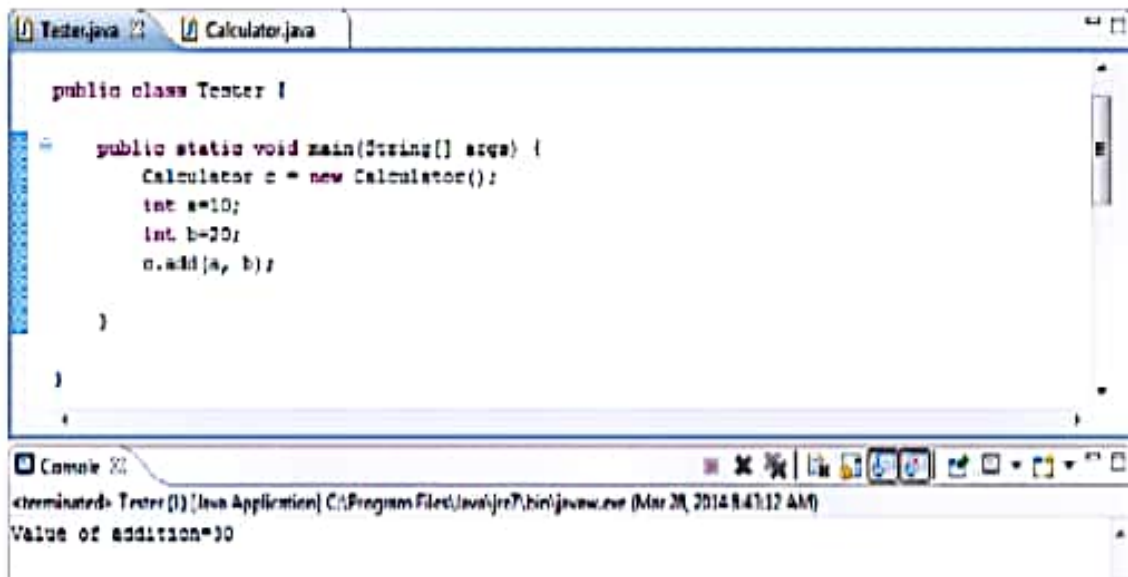
Here we are using `int a, int b` as parameters to receive values sent to add method.
Note:Calculator.java example which is been given that can not be executed as main method is not there.
Example 2:

This example demonstrates to how to use parameters to send values to functions.

TCS Internal

```
┌─────────────────────────────────────────────────────────────────────────┐
│ [J] Tester.java ⊠   [J] Calculator.java                              ↔ □ │
│                                                                           │
│   public class Tester {                                                  │
│                                                                           │
│       public static void main(String[] args) {                          │
│           Calculator c = new Calculator();                              │
│           int a=10;                                                      │
│           int b=20;                                                      │
│           c.add(a, b);                                                   │
│                                                                           │
│       }                                                                  │
│                                                                           │
│   }                                                                      │
│                                                                           │
├─────────────────────────────────────────────────────────────────────────┤
│ □ Console ⊠                          ■ ✕ ✖ | ░ ░ ░ ░ | ☐ □ ▾ ☐ ▾ ▀ □    │
│ <terminated> Tester (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 28, 2014 8:43:12 AM) │
│ Value of addition=30                                                     │
└─────────────────────────────────────────────────────────────────────────┘
```

In c.add(a,b) a and b act as parameters, which are used to send values to add method.
Note: Until, you will not include import statement or if both programs (Tester.java, Calculator.java) are not in same package that program will not run.

4.2.4 Static Variables and Functions

Static Variables

   i.   Class variables also known as static variables, are declared with the static keyword in a class.
  ii. These are declared outside a method or constructor.

Eg: public static int count = 0;

 iii. The basic concept behind static is, there would only be one copy of each class variable per class, regardless of how many objects are created from it.
 iv. That means, for a static field, regardless of the number of instances created, will hold the same data across all objects.
  v. If the value in static variable is changed, it is reflected in all objects. On the other hand, non-static fields of objects store their individual state. The value of a non-static field (also called instance variables) is unique for every instance / object.

 vi. Static variables are created when the program starts and destroyed when the program stops.
vii. Default values are same as instance variables. For numbers, the default value is 0; for boolean, it is false; and for object references, it is null.
viii.   Values can be assigned during the declaration or within the constructor.

 ix. Static variables in a class can be accessed from a second class by calling with the class name of the first. ClassName.variableName. Thus we can see that the static variables are not called by objects. It is directly called with class name.

Example 1:
Following example demonstrates how to declare and use static variables.

TCS Internal

```java
class StaticExample {

    public static int num;      //Here num is a static variable.

    public void add(int n, int m){
        num = n + m;
        System.out.println("Result ="+num);
    }

}
```

Note:StaticExample.java example which is been given that can not be executed as main method is not there.

```java
public class TestStatic {

    public static void main(String[] args) {

        System.out.println("Value of num="+StaticExample.num); // Use of static variable.

        StaticExample s = new StaticExample();
        s.add(10,15);

        System.out.println("Value of num="+StaticExample.num); // Use of static variable.

    }

}
```

<terminated> TestStatic [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 28, 2014 9:30:00 AM)
```
Value of num=0
Result =25
Value of num=25
```

In StaticExample class we have declared a static variable called num.
In TestStatic class we are accessing that variable.

We have created object of StaticExample class in TestStatic class to access the attributes and methods of StaticExample class.
Since num is a **static** variable, we can access it directly with the **class** name, before creating object of that **class**.

Static Functions
Through these courses we have already come across a static function. Can you guess? Yes, it's nothing but the main method which we used to test the classes and functions which we write.
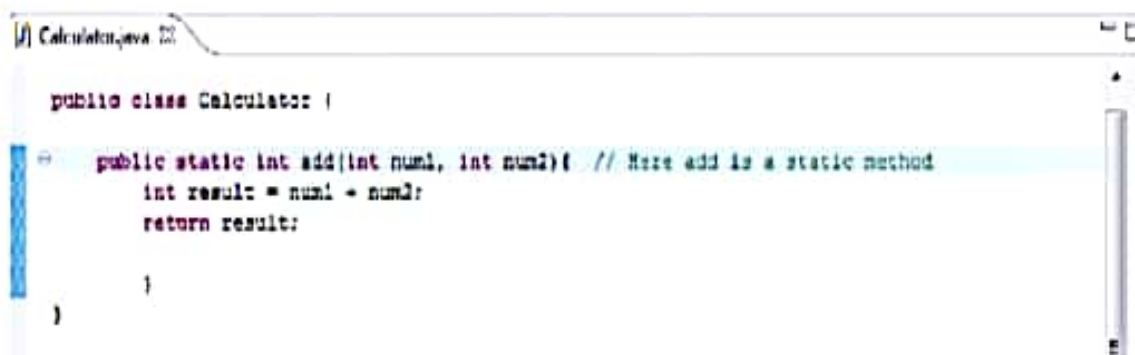
TCS Internal

Static methods are conceptually the same as static variables, thus the reasons to use or not use them are similar. They belong to the class, not specific objects of that class. There is no need of a logical instance variable to call these methods. That is, if a method needs to be in a class, but not tied to an object, then it make it static. If the method is more logically part of an object, then it should not be static.

Main is static because JVM can call main without creating an object first. It probably simplified the design of the JVM.
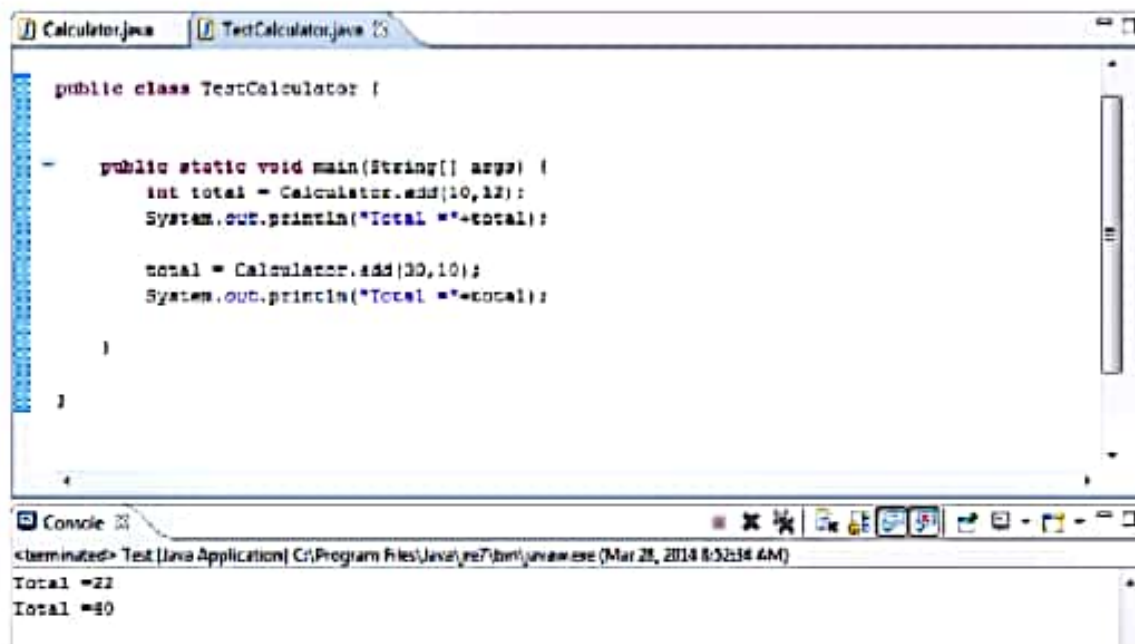
Static methods are also accessed by calling the method with the class name. ClassName.methodname.

Example: This example demonstrates use of static methods.

```
public class Calculator {

    public static int add(int num1, int num2){  // Here add is a static method
        int result = num1 + num2;
        return result;

    }

}
```

Note:Calculator.java example which is been given that can not be executed as main method is not there.

```
public class TestCalculator {

    public static void main(String[] args) {
        int total = Calculator.add(10,12);
        System.out.println("Total ="+total);

        total = Calculator.add(30,10);
        System.out.println("Total ="+total);

    }

}
```

Console
<terminated> Test [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 28, 2014 8:52:34 AM)
Total =22
Total =40

TCS Internal

Here add is a static method written in calculator class. We are accessing that in TestCalculator class. Since it is static method we can access it without creating object of that class, directly with the classname.

Points to remember

1. Static methods can't use non-static, instance variables directly.
2. Static methods can't use non-static, methods directly.
That means, if a static method need to access a non-static variable or method, then first create an object and using that object you need to access the method or variable.
This is because; static members which are related directly to class cannot see instance variable state and behaviour.

4.2.5 Numeric Data types

Numbers are so important in Java that 6 of the 8 primitive data types are numeric types which are shown in tabular format below.
There are both integer and floating point primitive types. Integer types have no fractional part whereas floating point types have a fractional part.
Description of each type is given after the table.

Type	Size	Range	Example
byte	8 bits	-128 to +127	byte b = 120;
short	16 bits	-32768 to +32767	short s = 1200;
int	32 bits	-2,147,483,648 to +2,147,483,647	int num = 120000;
long	64 bits	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	long d = 12345678998 l;
float	32 bits	1.40129846432481707e-45 to 3.402823466385288 60e+38 (positive or negative).	float f = 123.45f;
double	64 bits	4.940656458412465 44e-324d to 1.797693134862315 70e+308d (positive or negative).	double d = 12345.7856;

TCS Internal

Each of these types uses a fixed number of bits. i.e. same number of bits will be used no matter what value is represented.

For example, all values represented using the short data type use 16 bits. The value zero (as a short) uses 16 bits and the value thirty thousand also uses 16 bits.

1] byte:
- byte data type is an 8-bit signed two's complement integer.
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- byte data type is used to save space in large arrays, mainly in place of integers,

since a byte is four times smaller than an int.
- Example: byte a = 100 , byte b = -50

2] short:
- short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- short data type can also be used to save memory as byte data type.

A short is 2 times smaller than an int.
- Default value is 0.
- Example: short s = 10000, short r = -20000

3] Int:
- int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^31)
- Maximum value is 2,147,483,647(inclusive).(2^31 -1)
- int is generally used as the default data type for integral values unless there is a concern about

memory.
- The default value is 0.
- Example: int a = 100000, int b = -200000

4] long:
- long data type is a 64 bit signed two's complement integer.
- Minimum value is -9,223,372,036,854,775,808.(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive). (2^63 -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example: long a = 100000L, int b = -200000L

5] float:
- float data type is a single-precision 32-bit IEEE 754 floating point.
- float is mainly used to save memory in large arrays of floating point numbers.

TCS Internal

- Default value is 0.0
- float data type is never used for precise values such as currency.
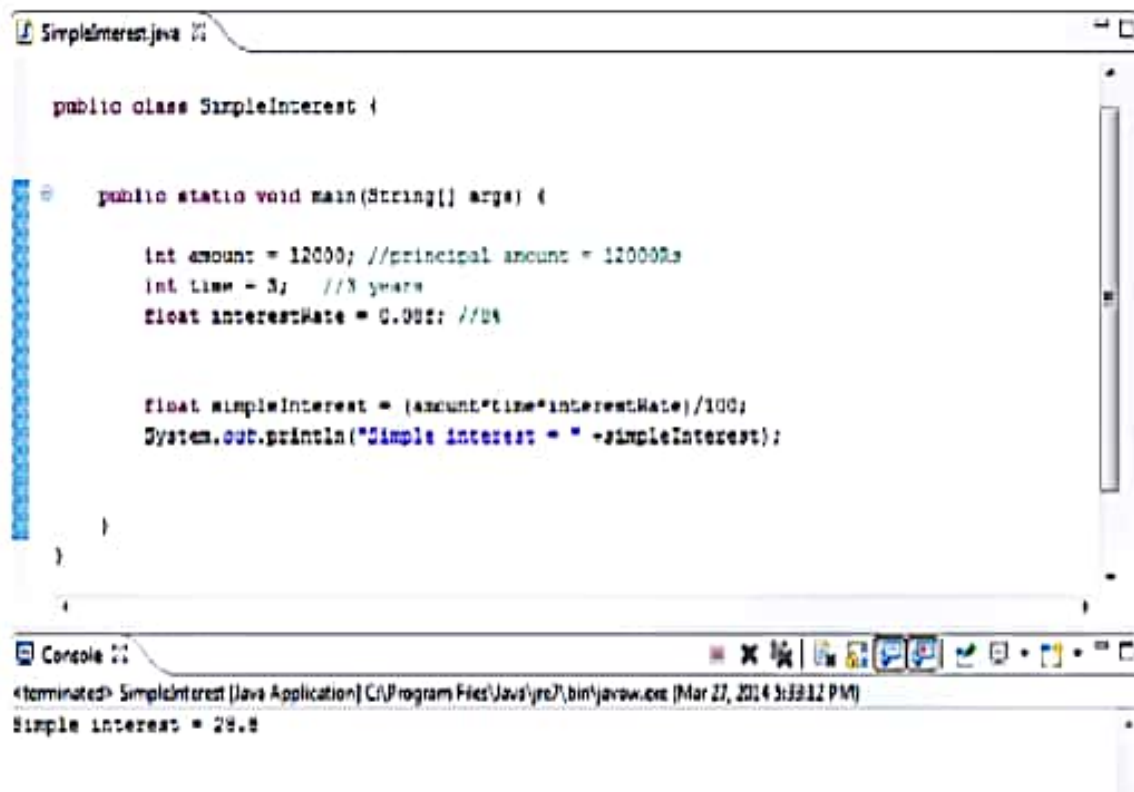- Example: float f1 = 234.5f

6] double:

- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values, generally the default choice.
- double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example: double d1 = 123.4

Fig: This digram demonstrates use of different datatypes.

TYPE	NAME	VALUE	
int	number →	1	Stored only Integer
int	sum →	500500	Stored only Integer
double	radius →	5.5	Stored only floating-point number
double	area →	95.0334	Stored only floating-point number
String	greeting →	Hello	Stored only texts
String	statusMsg →	Game Over	Stored only texts

A *variable* has a *name*, stores a *value* of the declared *type*.

Example : This program demonstrate the use of data types.

```
public class SimpleInterest {

    public static void main(String[] args) {

        int amount = 12000; //principal amount = 12000Rs
        int time = 3;    //3 years
        float interestRate = 0.08f; //8%


        float simpleInterest = (amount*time*interestRate)/100;
        System.out.println("Simple interest = " +simpleInterest);


    }

}
```

```
<terminated> SimpleInterest [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 27, 2014 3:33:12 PM)
Simple interest = 28.8
```

The result has fractional value, so we are using float variable to store it.

4.2.6 Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Misc Operators

1] The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
+	Addition – Adds values on either side of the operator	A + B = 30

TCS Internal

Operator	Description	Example
-	Subtraction – Subtracts right hand operand from left hand operand	A - B = -10
*	Multiplication – Multiplies values on either side of the operator	A * B = 200
/	Division – Divides left hand operand by right hand operand	B / A will give 2
%	Modulus – Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment – Increases the value of operand by 1	B++ will give 21
--	Decrement – Decreases the value of operand by 1	B-- will give 19

2] The Relational Operators:

There are following relational operators supported by Java language
Assume variable A holds 10 and variable B holds 20, then:

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If equal then condition becomes true.	A == B is not true
!=	Checks if the values of two operands are equal or not. If not equal then condition becomes true.	A != B is true
>	Checks if the value of left operand is greater than right operand. If yes then condition becomes true.	A > B Is not true
>=	Checks if the value of left operand is greater than or	A >= B Is not true

TCS Internal

Operator	Description	Example
	equal to right operand. If yes then condition becomes true.	
<	Checks if the value of left operand is smaller than right operand. If yes then condition becomes true.	A < B is true
<=	Checks if the value of left operand is smaller than or equal to right operand. If yes then condition becomes true.	A<=B is true

3] The Logical Operators:

➢ Logical operators return a true or false value based on the state of the Variables.

➢ There are six logical, or boolean, operators. They are AND, conditional AND, OR, conditional OR, exclusive OR, and NOT.

➢ Each argument to a logical operator must be a boolean data type, and the result is always a boolean data type.

➢ An example program is shown below that demonstrates the different Logical operators in Java.

Example:
In this example you can see the use of logical operators.

```
LogicalOPeratorsDemo.java ×

    public class LogicalOPeratorsDemo {

        public static void main(String args[]) {

            boolean a = true;
            boolean b = false;

            System.out.println("\na && b = " + (a&&b));

            System.out.println("a || b = " + (a||b) );

            System.out.println("!(a && b) = " + !(a && b));
        }
    }
```

```
Console ×
<terminated> LogicalOPeratorsDemo [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 28, 2014 9:27:37 AM)

a && b = false
a || b = true
!(a && b) = true
```

Fig: Logical operators

x	y	!x	!y	x&y x&&y	x|y x||y	x^y
TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	TRUE	FALSE	FALSE	FALSE

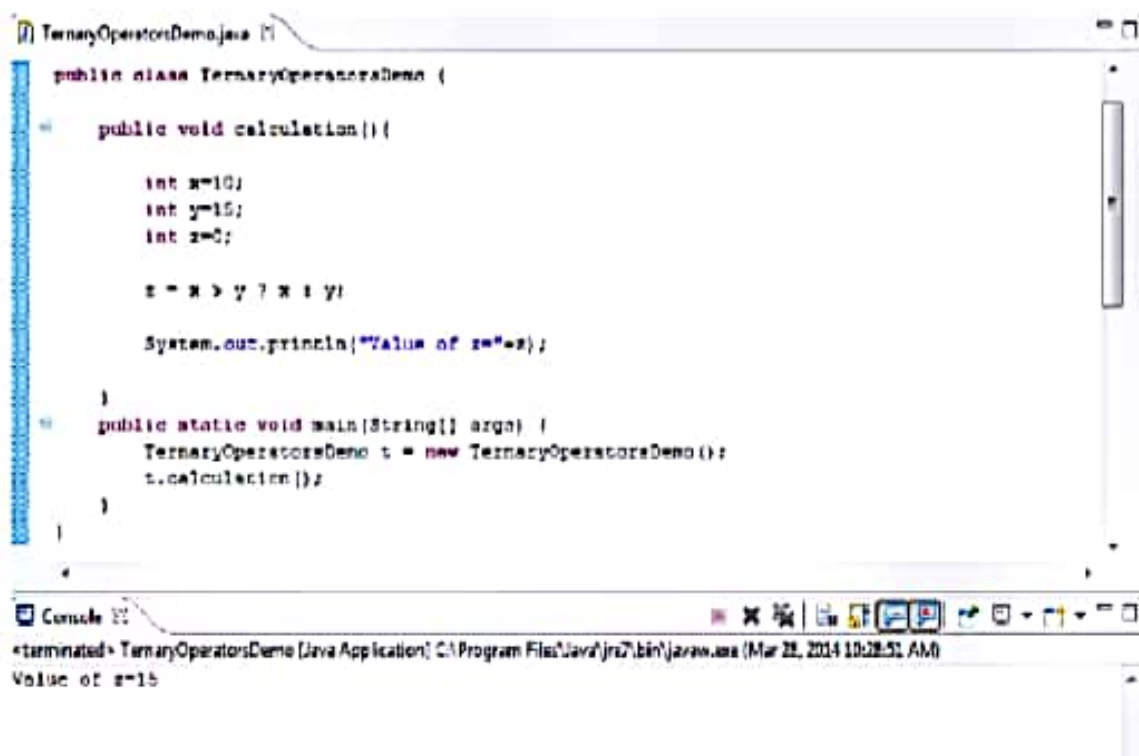## 5] The Assignment Operator:

- ✓ Assignment operator is used to assign values to variables.
- ✓ " = " is used as assignment operator.
  Eg: int i=10;
- ✓ The variable are always on the left-hand side of the assignment operator and the value to be assigned is always on the right-hand side of the assignment operator.
- ✓ The assignment operator is evaluated from **right to left**, so a = b = c = 0; would assign 0 to c, then c to b then b to a.
- ✓ i = i + 2; // same as i=10+2
- ✓ Here we say that we are assigning i's value to the new value which is i+2.

TCS Internal

✓ A short cut way to write assignments like this is to use the += operator. It's one operator symbol so don't put blanks between the + and =. So it is also called as short hand operator.
✓ i += 2; // Same as "i = i + 2

6] The Conditional Operator or Misc operator:

➤ The Conditional operator is the only ternary (operator takes three arguments) operator in Java. The operator evaluates the first argument and, if true, evaluates the second argument. If the first argument evaluates to false, then the third argument is evaluated.
➤ The conditional operator is the expression equivalent of the if-else statement.
➤ An example program is shown below that demonstrates the Ternary operator in Java.
➤ This example demonstrates how conditional operator can be used to find grater of 2 numbers.

Operators can also be classified as follows:

Unary Operators
The unary operators require only one operand. They work with single operands.

Example:
    count ++; // Existing value in variable count is incremented by 1.
    count --; // Existing value in variable count is decremented by 1.

TCS Internal

Binary Operators

Binary operators work with two operands or variables.

Example:

    int age = 35; // value 35 assigned to variable age.

    int x = 5; int y = 3;
    int z = x + y; // adding values of two variables and assigning result to another variable.

Ternary Operators

In ternary operator, takes 3 components: condition, true result, false result. This operator is having a ? and :. It looks like result = condition ? value 1 : value 2

Example:

    int a = 10;
    int x = a > 0 ? 1: 0; // checking a > 0, if yes returns 1, if no returns 0.

4.2.7 Precedence of Operators

Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

Precedence Order:

When two operators share an operand the operator with the higher precedence goes first. For example, 1 + 2 * 3 is treated as 1 + (2 * 3), whereas 1 * 2 + 3 is treated as (1 * 2) + 3 since multiplication has a higher precedence than addition.

Associativity:

When two operators with the same precedence the expression is evaluated according to its associativity. For example x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17, since the = operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side). On the other hand, 72 / 2 / 3 is treated as (72 / 2) / 3 since the / operator has left-to-right associativity.

Precedence and associativity of Java operators:

The table below shows all Java operators from highest to lowest precedence, along with their associativity.
Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

For more information on precedence and associativity you can visit the following link.
TCS Internal

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

4.2.8 String

Strings are an incredibly common type of data in computers. Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

Java string is a series of characters gathered together, like the word "Hello", or the phrase "practice makes perfect". Create a string in the code by writing its chars out between double quotes.

Following are some of the ways to instantiate a java String
String str1 = "Hello";
String str2 = new String("Apple");
String str3 = new String(char []);

Methods:

There are methods for:

- concatenating two strings - **concat() ,+**

```
public class StringConcatWays {

    public static void main(String a[]) {
        String str1 = "TATA ";
        String str2 = "Consultancy Services";

        String str3 = "TATA ";
        String str4 = "Group of companies";
        /** We can do string concatination by two ways. * One is by using '+' operator, shown below.*/
        String d = str1 + str2;
        System.out.println(d);
        /** Another way is by using concat() method, * which appends the specified string at the end. */
        d = str3.concat(str4);
        System.out.println(d);

        /** You can also use the concat() method with string literals, as in: */
        String str5 = "My name is ".concat("Rumple");

        System.out.println(str5);
        /** The + operator is widely used in print statements.  */
        String string1 = "saw I was ";
        System.out.println("Dot " + string1 + "Tod");

    }
}
```

TCS Internal

Result:

```
TATA Consultancy Services
TATA Group of companies
My name is Rumple
Dot saw I was Tod
```

- getting the character at a given position within the string -- charAt() .By using indexOf() method you get get the position of the sepcified string or char from the given string. You can also get the index strting from a specified position of the string.
ex:

charAt(int index)

@Returns the char value at the specified index

```
public class StringChatAt {
    public static void main(String args[]) {
        String s = new String("TATA Consultancy Services!");
        char result = s.charAt(6);
        System.out.println(result);
    }
}
```

will result as

```
a
```

Combining charAt() with another String method, we can get the final character of any String, regardless of length:

```
public class StringChatAt {
    public static void main(String args[]) {
        String s = new String("What a long, strange trip it's been.");
        char result = s.charAt(s.length()-1);

        System.out.println(result);


    }
}
```

will result as

```
.
```

T

- seeing if a character or string exists within a string -- indexOf()

By using indexOf() method you get get the position of the sepcified string   or   char   from the given string. We can also get the index strting from a   specified position of the string.

```
public class StringIndexOf {
    public static void main(String[] a) {
        String str = "Use this string for testing this";
        System.out.println("Basic indexOf() example");
        System.out.println("Char 's' at first occurance: " + str.indexOf('s'));
        System.out.println("String \"this\" at first occurance: "
            + str.indexOf("this"));
        /** * Returns the first occurance from specified start index */
        System.out
            .println("First occurance of char 's' from 4th index onwards : "
                + str.indexOf('s', 4));
        System.out
            .println("first occurance of String \"this\" from 6th index onwards: "
                + str.indexOf("this", 6));
    }
}
```

Output is

```
Basic indexOf() example
Char 's' at first occurance: 1
String "this" at first occurance: 4
First occurance of char 's' from 4th index onwards : 7
First occurance of String "this" from 6th index onwards: 28
```

- getting the number of characters in a string -- length()

Example:

Output:

```
Length:17
```

- extracting a substring from a string – substring()
The substring begins with the character at the specified index and extends   to the end of this string or up to endIndex - 1 if second argument is given.

Syntax of this method:

public String substring(int beginIndex)

or

public String substring(int beginIndex, int endIndex)

Note:

beginIndex -- the begin index, inclusive.

TCS Internal

endIndex -- the end index, exclusive.

Example:

```
public class SubstringExample {

    public static void main(String args[]) {
        String str = new String("TATA Consultancy Services");

        System.out.println("Initial string is: " + str);

        System.out.println("Start position=4 and no end position specified: "
                + str.substring(4));

        System.out.println("Start position=2 and end position=11: "
                + str.substring(2, 11));

        // if start = end, then the extracted string will be empty
        System.out.println("Start position=3 and end position=3: "
                + str.substring(3, 3).isEmpty());

    }

}
```

Output:

```
Initial string is: TATA Consultancy Services
Start position=4 and no end position specified:  Consultancy Services
Start position=2 and end position=11: TA Consul
Start position=3 and end position=3: true
```