Object Oriented Programming

Unit 2

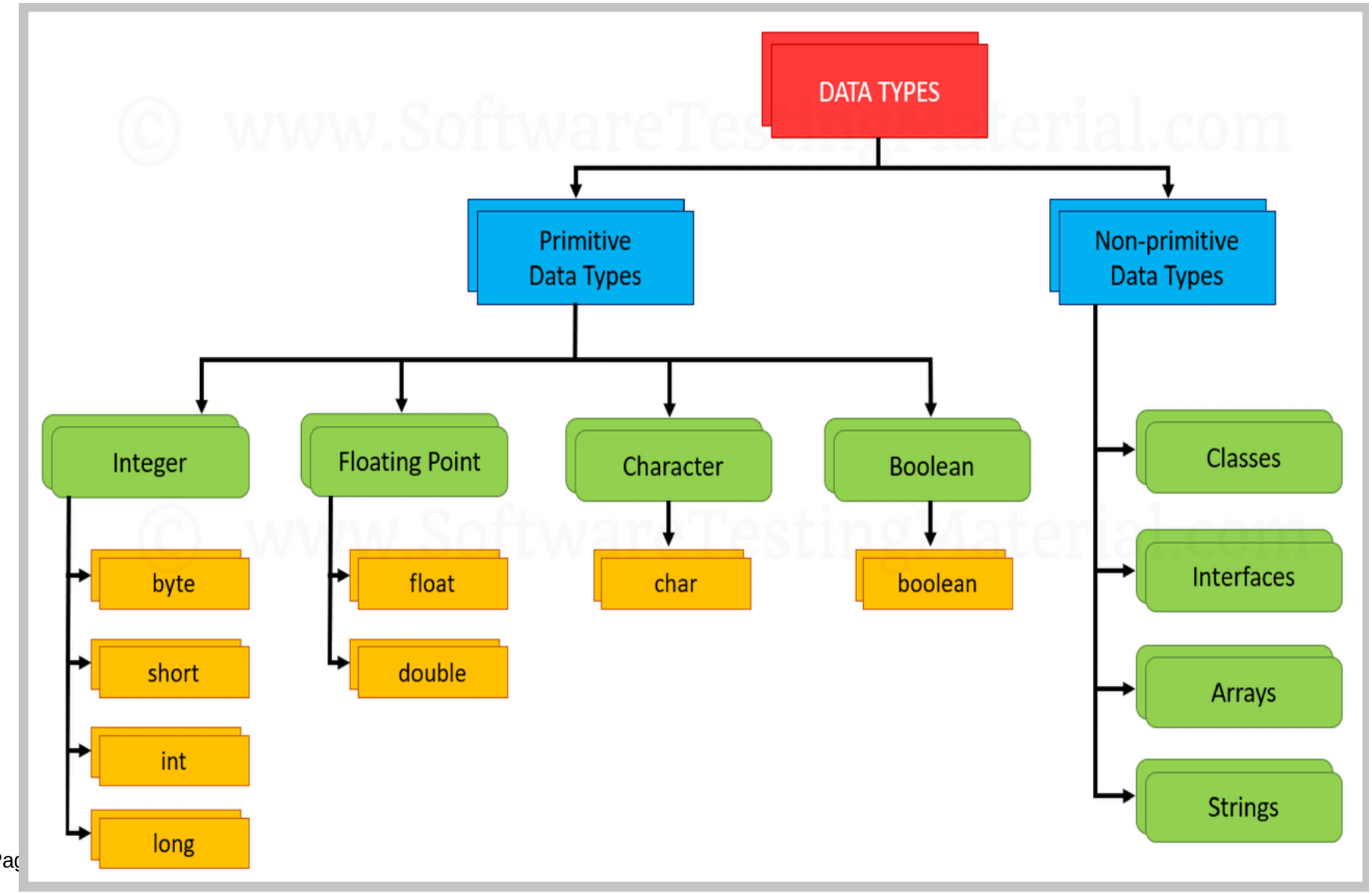
Java Language and operators:

Data types, variables and operators: data type, type casting, variable, operators, scanner class, implementation scanner class

Programming Constructs : what are programming constructs, if statements, implementation ternary operator, case statements, implementation switch case, looping

Arrays : Arrays , implement single dimensional array, single dimensional array using integer, multidimensional array

Datatypes in Java



Datatypes in Java(Primitive Data type)

Type	Description	Default	Size	Example Literals
boolean	true or false	false	1 bit	true, false
byte	twos complement integer	0	8 bits	(none)
char	Unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'', '\n', 'ß'
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d

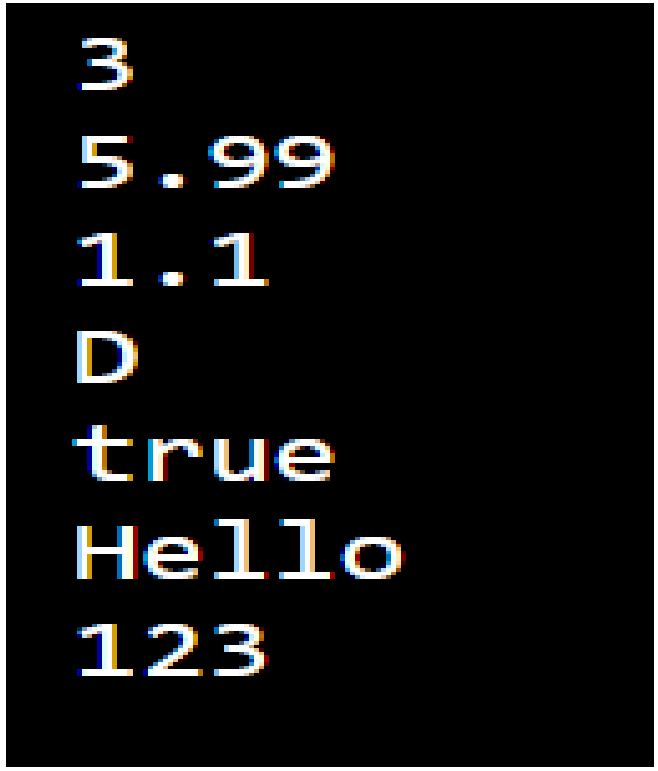
Datatypes in Java(Example)

```
public class Main
{
  public static void main(String[] args)
  {
      int num = 3;                        // integer (whole number)
      float floatNum = 5.99f;             // floating point number
      double doubleNum = 1.1;             // double point number
      char letter = 'D';                  // character
      boolean bool = true;                // boolean
      String text = "Hello";              // String
      byte by=123;                        //byte
```

Datatypes in Java(Example)

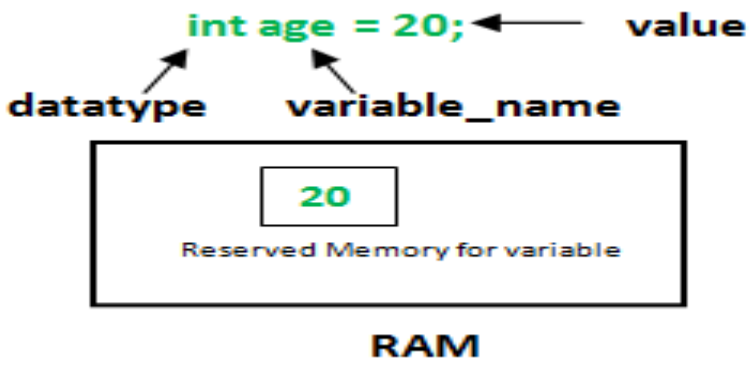
```
    System.out.println(num);
    System.out.println(floatNum);
    System.out.println(doubleNum);
    System.out.println(letter);
    System.out.println(bool);
    System.out.println(text);
    System.out.println(by);
  }
}
```

Output:

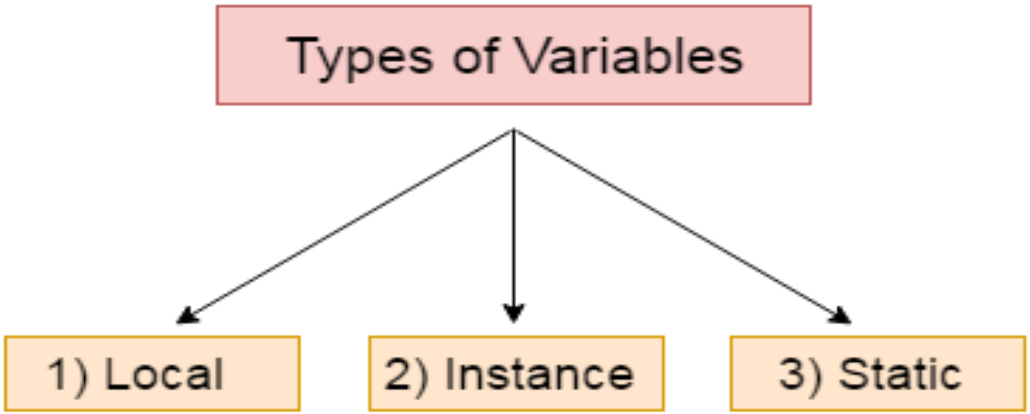


Variables in Java

- A variable is the **name given to a memory location.** It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.
- It is a combination of "**vary + able**" that means its value can be changed.



Types of Variable



1) Local Variable

A variable which is **declared inside the method** is called local variable.

2) Instance Variable

A variable which is **declared inside the class but outside the method**, is called instance variable . It is not declared as static.

3) Static variable

A variable that is **declared as static** is called static variable. It cannot be local.

Example: Types of variables

```
import java.io.*;

public class Demo
{
    int data=10;  //instance variable
    static int num1=90;  //static variable

    void getData()
   {
       int num2=120;  //local variable
    }
}
```

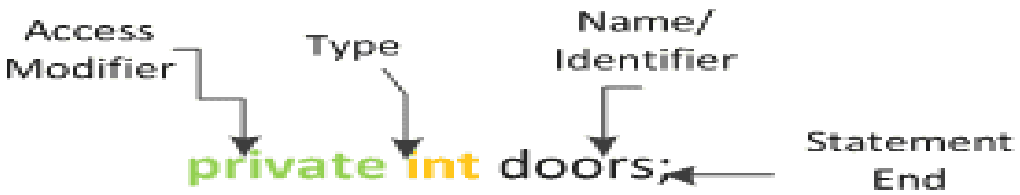
Local Variables

- Local variables are **declared in methods, constructors, or blocks.**
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- **Example:**

```
float getDiscount(int price)
{
  float discount;
  discount=price*(20/100);
  return discount;
}
```

Declaration of Instance variable

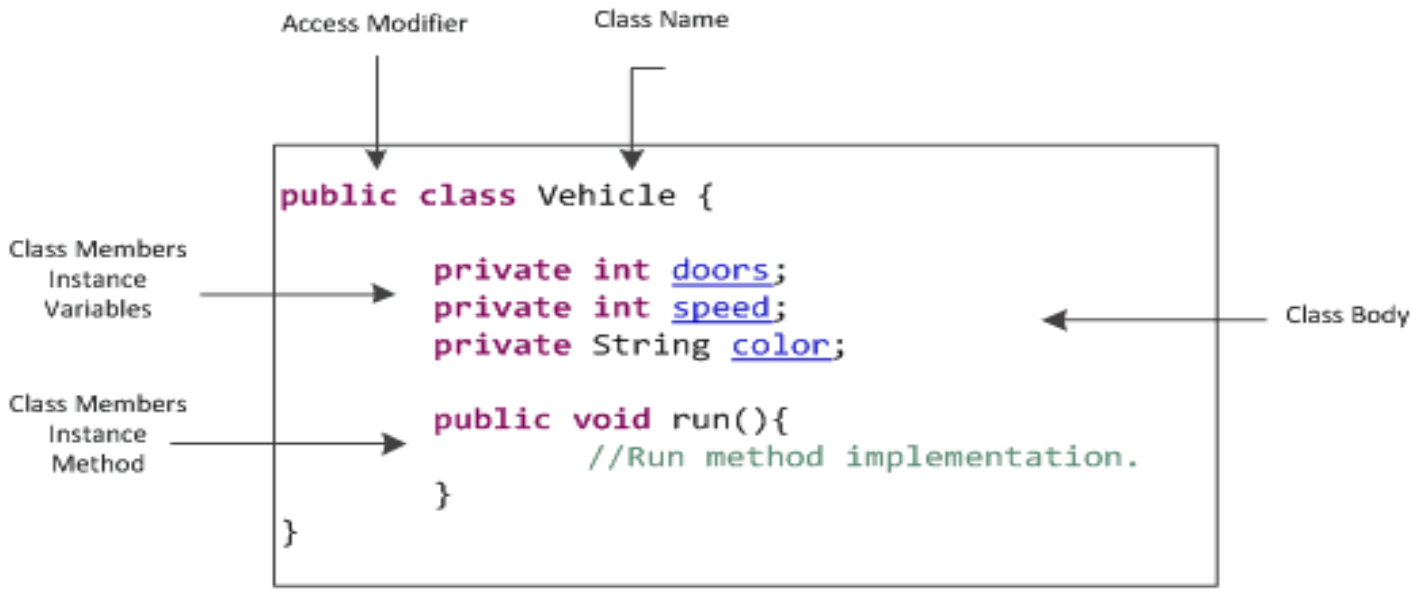
- **Variables defined within a class are called instance variables** because each instance of the class (that is, each object of the class) contains its own copy of these variables.
- Thus, the data for one object is separate and unique from the data for another.
- An instance variable can be declared **public or private or default (no modifier).**
- When we do not want our variable's value to be changed outside our class we should declare them private.
- public variables can be accessed and changed from outside of the class



Instance variable

Declaration of Class :

A class is declared by use of the class keyword. The class body is enclosed between curly braces { and }. The data or variables, defined within a class are called instance variables. The code is contained within methods. Collectively, the methods and variables defined within a class are called members of the class.



- Instance variable is the variable declared inside a class, but outside a method something like:

```
class IronMan{

    /** These are all instance variables **/
    public String realName;
    public String[] superPowers;
    public int age;

    /** Getters / setters here **/
}
```

- Now this IronMan Class can be instantiated in other class to use these variables, something like:

```
class Avengers{
        public static void main(String[] a){
                IronMan ironman = new IronMan();
                ironman.realName = "Tony Stark";
                // or
                ironman.setAge(30);
         }

}
```

Static Variables

- **Class variables also known as static variables** are declared with the **static keyword** in a class, but outside a method, constructor or a block.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static.
- Static variables are stored in the static memory.

Example

```
import java.io.*;
public class Employee {

   // salary  variable is a private static variable
   private static double salary;

   // DEPARTMENT is a constant
   public static final String DEPARTMENT = "Development ";

   public static void main(String args[]) {
      salary = 1000;
      System.out.println(DEPARTMENT + "average salary:" + salary);
   }
}
```

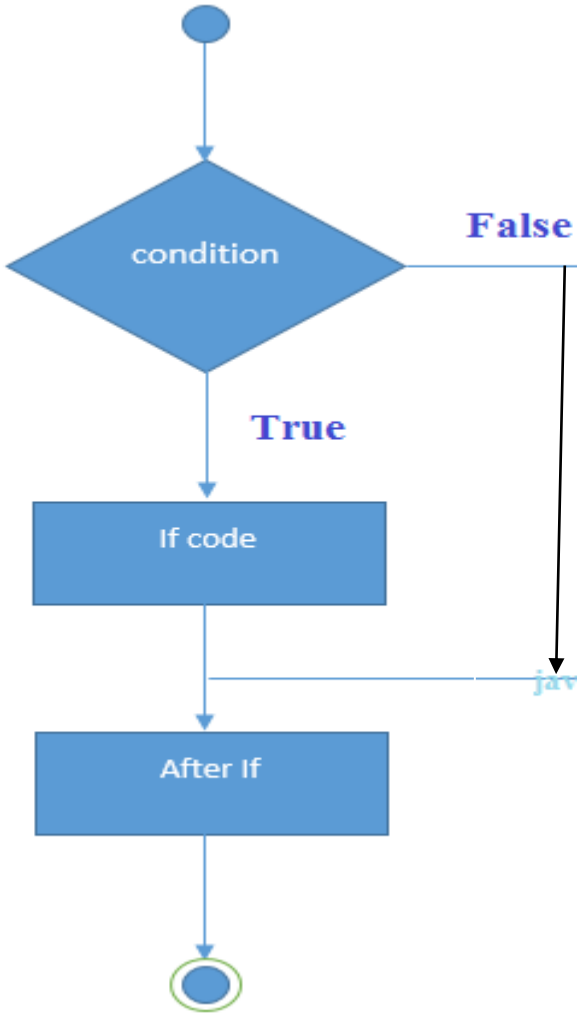
Programming Constructs : Control Flow Statements

- Java Control statements **control the order of execution in a java program**, based on data values and conditional logic. There are three main categories of control flow statements;
  - **Selection statements:**
    - Selection statement causes the program control to be transferred to a specific flow based upon whether a certain condition is true or false. These are called as conditional statements.
    - if, if-else, multiple-if, nested-if, if-else ladder(multiple if-else), else-if and switch.
  - **Loop statements:** while, do-while and for.
  - **Transfer statements:** break, continue, return, try-catch-finally

If Statement

- The Java if statement tests the condition. It executes the *if block* if condition is true.

```
if (condition)
{
        statement;
}
```



If Statement Example

```
public class IfExample
{
public static void main(String[] args)
{
    int marks=90;
    if(marks>75)
    {
        System.out.print("Grade A");
    }
}
}
```

Compile by: javac IfExample.java

Run by: java IfExample

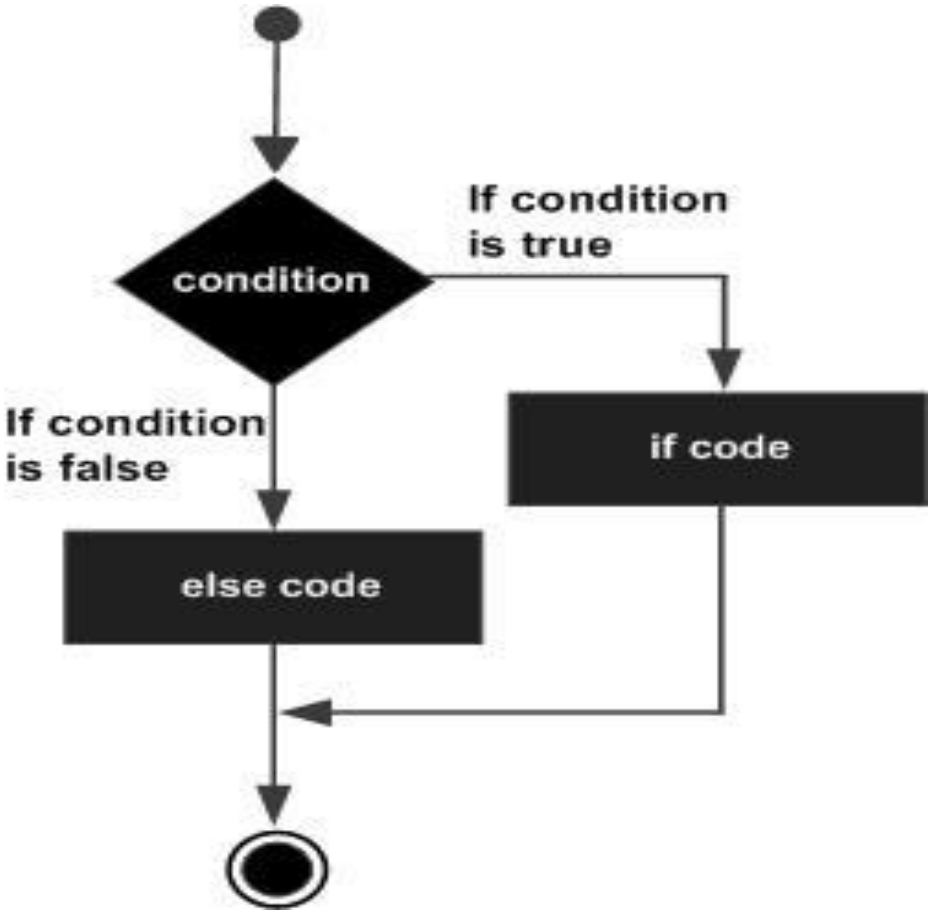
```
Grade A
```

If…else Statement

- The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

- Syntax:

```
if(condition)
{
//code if condition is true
}
Else
{
//code if condition is false
}
```



If…else Statement Example

```
public class Test {


   public static void main(String args[]) {
      int x = 30;
      if( x < 20 ) {
         System.out.print("This is if statement");
      }else {
         System.out.print("This is else statement");
      }
   }
}
```

If…else if Ladder Statement

- An if statement can be followed by an optional *else if...else* statement, which is very useful to test various conditions using single if...else if statement.
  - When using if, else if, else statements there are a few points to keep in mind.
  - An if can have zero or one else's and it must come after any else if's.
  - An if can have zero to many else if's and they must come before the else.
  - Once an else if succeeds, none of the remaining else if's or else's will be tested.

If…else if Ladder Statement

```
if(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2)
{
//code to be executed if condition2 is true
}
else if(condition3)
{
//code to be executed if condition3 is true
}
else{
//code to be executed if all the conditions are false
}
```

If…else if Ladder Statement Example

```
import java.util.Scanner;
class Practical1a
{
	public static void main(String args[])
	{
		int b;
		Scanner sc=new Scanner(System.in);
		b=sc.nextInt();
		if(b>0)
		{
			System.out.println(b+" is positive");
		}
		else if(b<0)
		{
			System.out.println(b+" is negative");
		}
		else
		{
			System.out.println(b+" is zero");
		}
	}
}
```

Nested if Statement

- The nested if statement represents the if block within another if block. Here, the inner if block condition executes only when outer if block condition is true.
- **Syntax:**

```
if(condition)
{
    //code to be executed
   if(condition)
   {
          //code to be executed
    }
}
```

Nested if Statement Example

```
public class JavaNestedIfExample
{
   public static void main(String[] args)
{
        int age=20;
        int weight=80;
        if(age>=18)
        {
                if(weight>50)
                {
                        System.out.println("You are eligible to donate blood");
                }
        }
}
}
```

switch Statement

- The Java switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement. In other words, the switch statement tests the equality of a variable against multiple values.
- Syntax:

```
switch(expression)
{
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......
default:
 code to be executed if all cases are not matched;
}
```

switch Statement Example

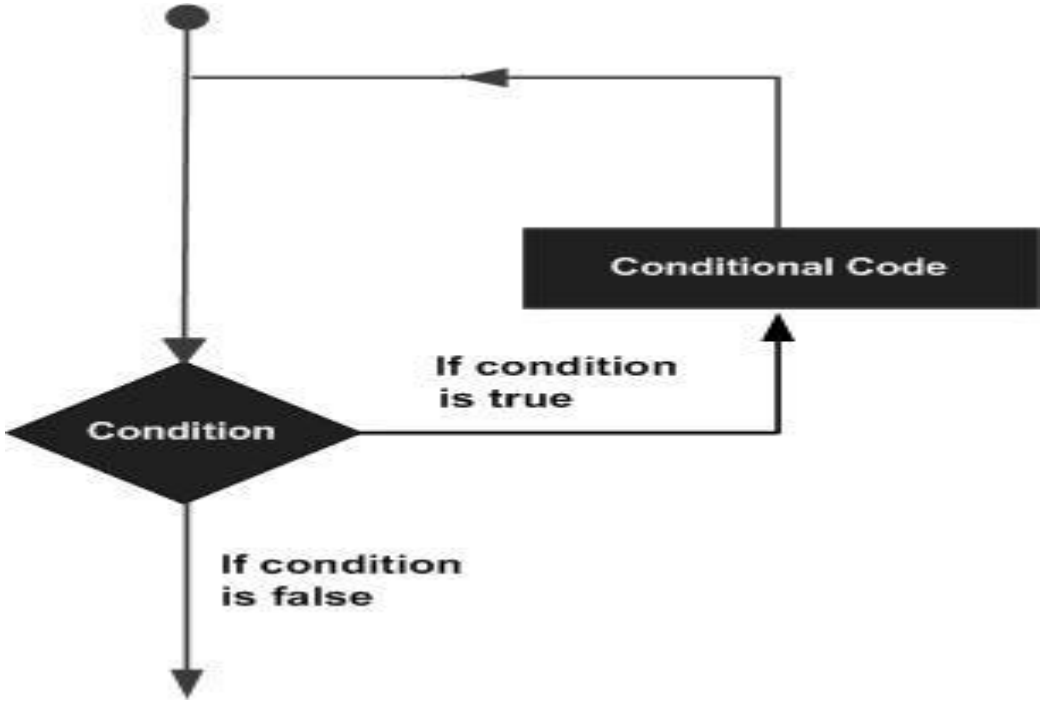
```
public class SwitchExample {
public static void main(String[] args) {
    int month=20;
    switch(month){
    case 1:
            System.out.println(“January");
            break;
    case 2:
            System.out.println(“February");
            break;
    case 3:
            System.out.println(“March");
            break;
```

switch Statement Example

```
    case 4:
            System.out.println(“March");
            break;
     default:
            System.out.println(“Invalid Month");
    }
}
}
```

Iterations

- Programming languages provide various control structures that allow for more complicated execution paths.
- A **loop statement allows us to execute a statement or group of statements multiple times** and following is the general form of a loop statement in most of the programming languages.

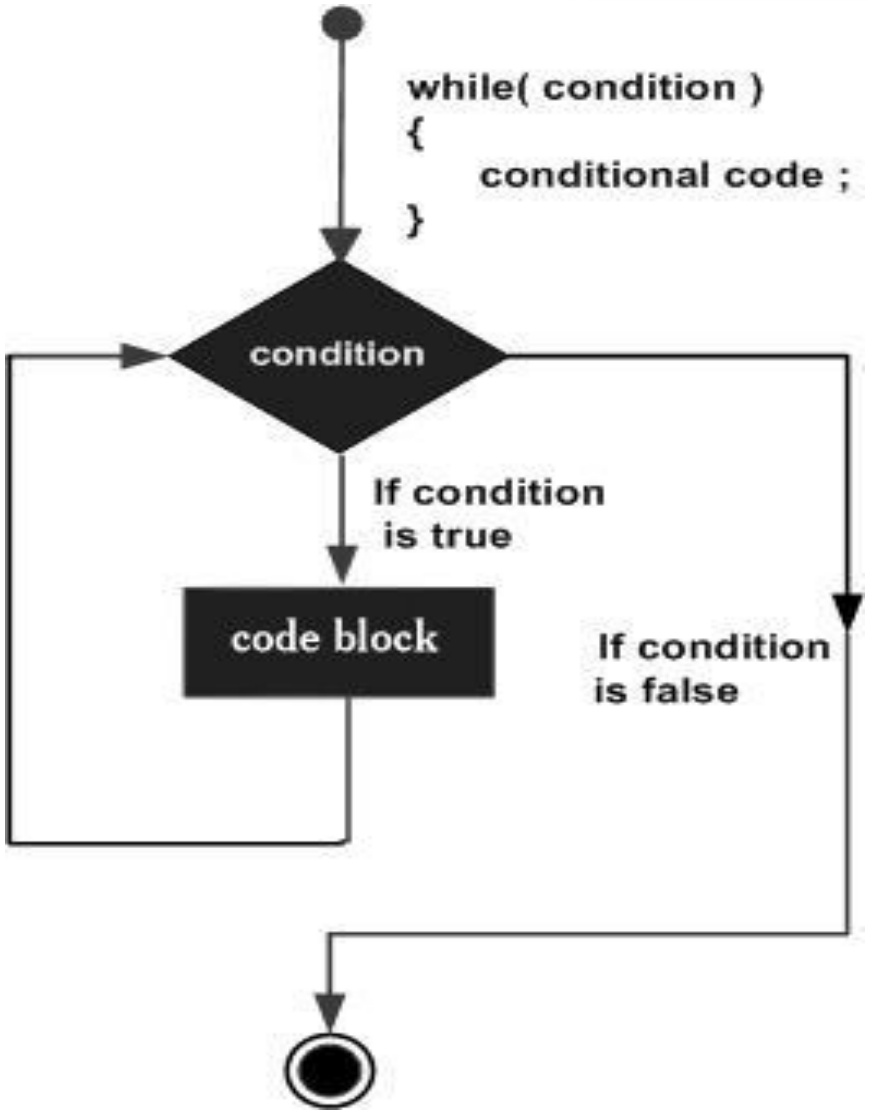


Iterations: while loop

- A while loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.
- The syntax of a while loop is −

```
while(Boolean_expression)
{
        // Statements
}
```

- Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non zero value.



Iterations: while loop example

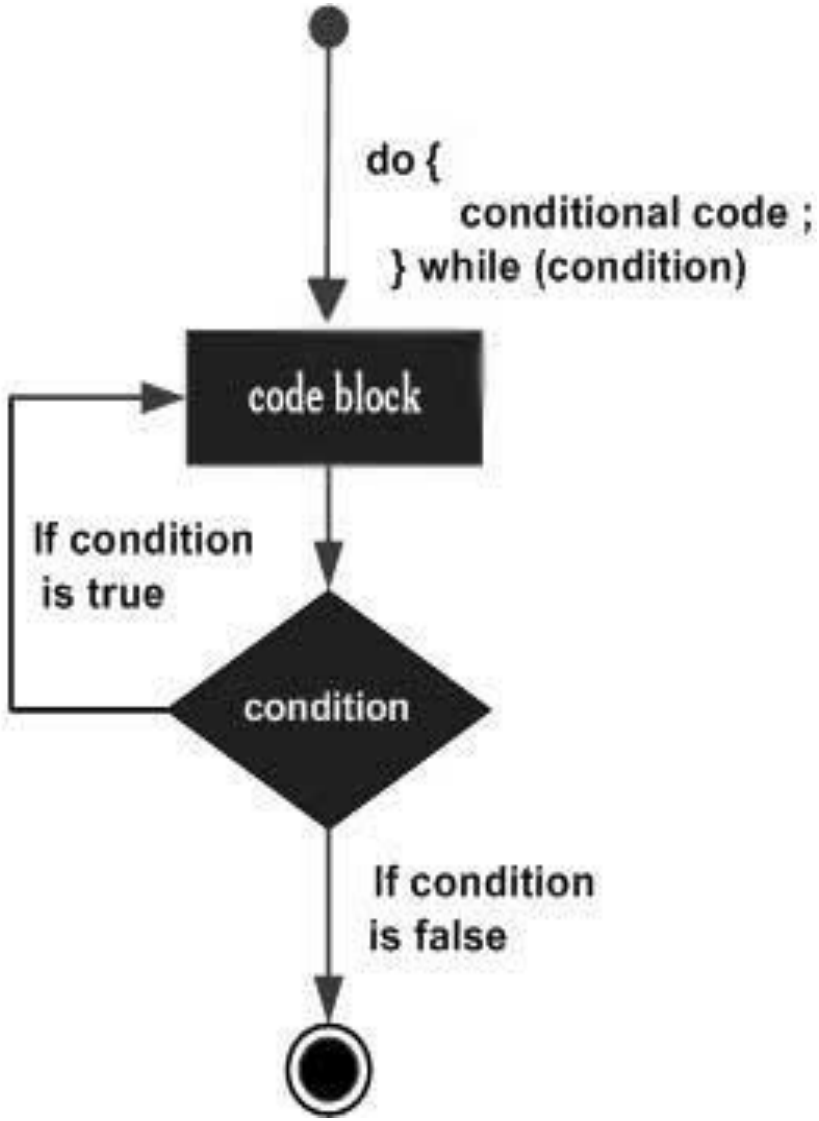
```
public class Test
{
   public static void main(String args[])
  {
      int x = 10;
      while( x < 20 )
     {
         System.out.print("value of x : " + x );
         x++;
         System.out.print("\n");
      }
   }
}
```

Output:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Iterations: do….while loop

- A do...while loop is similar to a while loop, except that a do...while loop is **guaranteed to execute at least one time**.



- **Syntax:**

Following is the syntax of a do...while loop −

```
do
{
   // Statements
}
while(Boolean_expression);
```

Iterations: do….while loop example

```
public class Test
{
   public static void main(String args[])
   {
      int x = 10;
      do
     {
         System.out.println("value of x : " + x );
         x++;
     }
        while( x < 20 );
   }
}
```

Output:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Iterations: do….while loop example

```
public class Test
{
   public static void main(String args[])
   {
      int x = 10;
      do
     {
         System.out.println("value of x : " + x );
         x++;
     }
        while( x > 20 );
   }
}
```

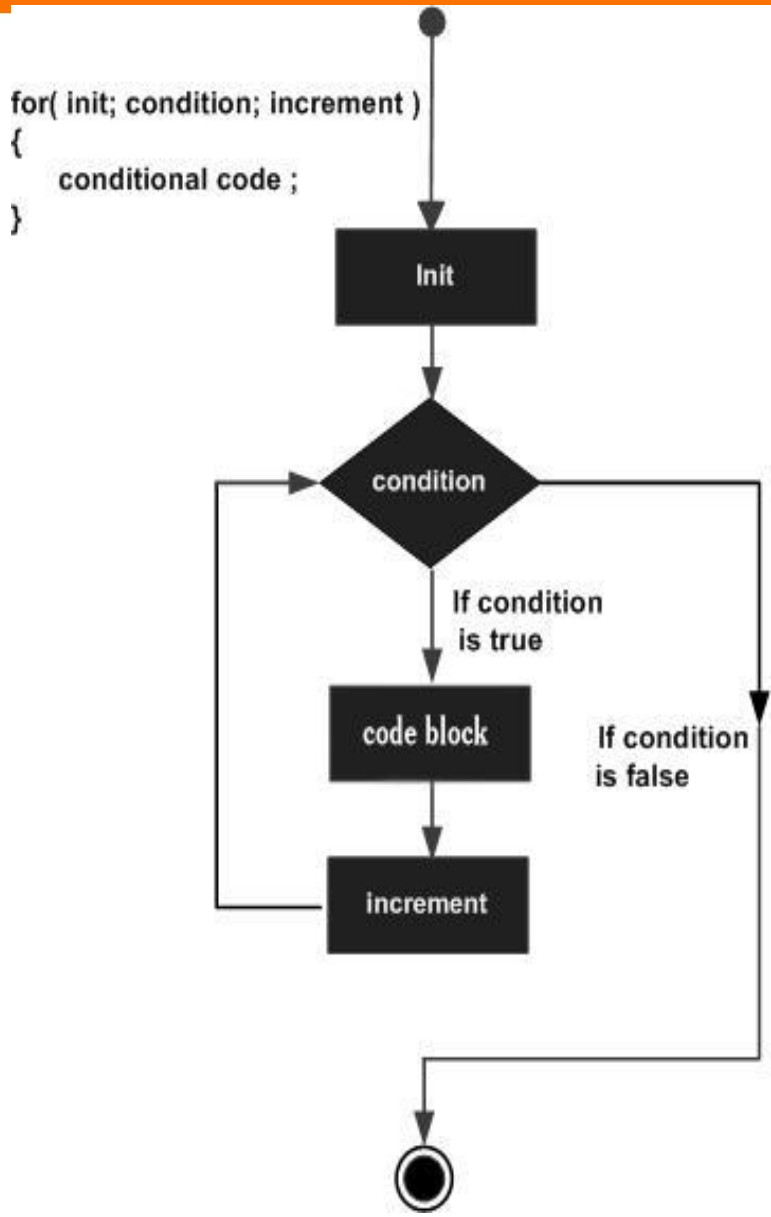
Output:

value of x : 10

Iterations: for loop

- A for loop is a **repetition control structure** that allows you to efficiently write a loop that needs to be executed a specific number of times.
- A for loop is useful when you know **how many times a task is to be repeated**.
- The syntax of a for loop is:

```
for(initialization; condition; incr/decr)
{
  // Statements
}
```



Iterations: for loop example

```
public class Test
{
  public static void main(String args[])
  {

    for(int x = 10; x < 20; x ++)
    {
      System.out.println("value of x : " + x );
     }
  }
}
```

Output:

```
value of x : 10
value of x : 11
value of x : 12
value of x : 13
value of x : 14
value of x : 15
value of x : 16
value of x : 17
value of x : 18
value of x : 19
```

Iterations: for-each loop

- The for-each loop is used to traverse array or collection in java.
- It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
- It works on elements basis not index. It returns element one by one in the defined variable.
- **Syntax:**

```
for(type var:array)
{
   //code to be executed
}
```

Iterations: for-each loop example

```
public class ForEachExample
{
public static void main(String[] args)
  {
     int arr[]={12,23,44,56,78};
     for(int i:arr)
    {
       System.out.println(i);
    }
  }
}
```

Output:

```
12
23
44
56
78
```

Labelled Statement

- Unlike C/C++, Java does not have goto statement, but Java supports label.
- Mostly, the only place where a label is useful in java is just above nested loop statements.
- We can specify a label name with break to terminate specified outer loop.
- Similarly, label name can be specified with continue also.
- **Syntax:**

```
break label_name;

        OR

continue label_name;
```

- **Syntax:**

```
label:
{
  statement1;
  statement2;
  statement3;
  .
  .
}
```

Labelled Statement Example

```
public class Tester
 {
  public static void main(String args[])
  {
     second:
          for (int i = 0; i < 3; i++)
          {
             for (int j = 0; j < 3; j++)
            {
               if(i == 1)
               {
                  break second;
               }
              System.out.print(" [i = " + i + ", j = " + j + "] ");
            }
          }
    }
}
```

Output:

[i = 0, j = 0] [i = 0, j = 1] [i = 0, j = 2]

Break Statement

- When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- **The Java break is used to break loop or switch statement.** It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
- We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.
- **Syntax:**

```
break;
```

Break Statement Example

```
public class BreakExample
{
public static void main(String[] args)
 {
    for(int i=1;i<=10;i++)
    {
        if(i==5)
       {
            break;
        }
        System.out.println(i);
    }
}
}
```

Output:

1
2
3
4

Break Statement with inner loop Example

```
public class BreakExample2
{
public static void main(String[] args)
{
          for(int i=1;i<=3;i++)
          {
                  for(int j=1;j<=3;j++)
                  {
                     if(i==2&&j==2)
                     {
                      break;
                     }
                     System.out.println(i+" "+j);
                  }
          }
}
}
```

Output:

```
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Continue Statement

- The continue statement is used in loop control structure when you need to **jump to the next iteration of the loop immediately**. It can be used with for loop or while loop.

- The Java continue statement is used to **continue the loop**. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.

- We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

- **Syntax:**

```
continue;
```

Continue Statement Example

```
public class ContinueExample
{
public static void main(String[] args)
{
   for(int i=1;i<=10;i++)
   {
      if(i==5)
      {
         continue;
       }
       System.out.println(i);
   }
}
}
```

Output:

1
2
3
4
6
7
8
9
10

Return Statement

- To return a value from function, we use the keyword **return.**
- We can return a direct console value of variable value. But, a function can return only one value.
- **return statement should be the last statement of a function, because it also returns the controls back to the calling function.**
- The datatype of returned value will be the return-type of that function.
- If a function does not return anything, then return-type will be void.
- **Syntax:**

  return value/variable_name;

Return Statement Example

```
class A
{
        double RR(double a, double b)
        {
                double sum = 0;
                sum = (a + b) / 2.0;
                return sum;
        }
        public static void main(String[] args)
        {
                A obj=new A();
                System.out.println(obj.RR(5.5, 6.5));
        }
}
```

Output:
6.0

Array

- An array is a **collection of similar type** of elements that have a contiguous memory location.
- Java **array is an object** which contains elements of a similar data type. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.
- Array in java **is index-based**, the first element of the array is stored at the 0 index.

Syntax:

```
datatype var_name[]={value1,value2,value3,…,valueN};  //Array

datatype[] var_name=new datatype[size];

datatype[] var_name=new datatype[size]{values};
```

Array

Declaration, Instantiation and Initialization of Java Array

```
int a[]={33,3,4,5};  //Array
class TestArray
{
  public static void main(String args[])
 {
   int a[]={33,3,4,5};
   for(int i=0;i<a.length;i++)
   System.out.println(a[i]);
 }
}
```

Multidimensional Arrays

- A multidimensional array is an **array containing one or more arrays.**
- To create a two-dimensional array, add each array within its own set of curly braces:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements.

- To access the elements of the myNumbers array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

```
public class MyClass
{
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    int x = myNumbers[1][2];
    System.out.println(x);
  }
}
```

Multidimensional Arrays

- **dataType[][] arrayRefVar; (or)**
- **dataType[][] arrayRefVar; (or)**
- **dataType arrayRefVar[][]; (or)**
- **dataType[] arrayRefVar[];**
- **Example to instantiate Multidimensional Array in Java**
- int[][] arr=new int[3][3]; //3 row and 3 column
- **Example to initialize Multidimensional Array in Java**

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;

arr[2][0]=7;

arr[2][1]=8;

arr[2][2]=9;

Multidimensional Arrays

```
public class MyClass
{
  public static void main(String[] args)
{
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length; i++)
{
      for(int j = 0; j < myNumbers[i].length; j++)
{
        System.out.println(myNumbers[i][j]);
    }
    }
  }
}
```

```
1
2
3
4
5
6
7
```

Multidimensional Arrays

```
class TestArray
 {
public static void main(String args[])
{
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
for(int i=0;i<3;i++)
{
 for(int j=0;j<3;j++)
{
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}
}
```

```
1 2 3
2 4 5
4 4 5
```

Multidimensional Arrays

Arr.java - Notepad

File Edit Format View Help

```
class Arr
{
static void min(int arr[])
{
                int min=arr[0];
                for(int i=1;i<arr.length;i++)
                {
                  if(min>arr[i])
                  {
                     min=arr[i];
                  }
                  }
                    System.out.println(min);
}

public static void main(String args[])
{
                int a[]={33,3,4,5};
                min(a);
}
}
```