CORE JAVA

JAVA:

Java is **a pure object-oriented programming language that produces software for multiple platforms**. When a programmer writes a Java application, the compiled code (known as bytecode) runs on most operating systems (OS), including Windows, Linux and Mac OS.

COMMENTS:

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

PRINT STATEMENT:

- System: It is a final class that belongs to the java.lang.package.
- out: The out is an instance of the System class and is of type PrintStream. It is a public and static member field. It is an instance of java.io.PrintStream. When we call the member, a PrintStream class object creates internally.
- println(): It is the method of PrintStream class that is used to print statements on the console.

PROGRAM 1:

```
public class MyClass
{
   public static void main(String args[])
  {
     //This is a single line comment
     System.out.println("Hello");
    /*This is a multi line comment and the above code will print the word that is
Hello*/
   }
}
```

EXERCISE :

1. Create a Java program to Print your Name
2. Create a Java program to print your name and address with comments

DATA TYPES:

Data type specifies the size and type of variable values, and it has no additional methods.

VARIABLES:

Variables are containers for storing data values.

PROGRAM 2:

```
public class Main
{
  public static void main(String[] args)
  {
    String name = "John";
    System.out.println(name);
  }
}
```

PROGRAM 3:

```
public class Main
{
  public static void main(String[] args)
  {
    int myNum;
    myNum = 15;
    System.out.println(myNum);
  }
}
```

PROGRAM 3:

```
public class Main
{
  public static void main(String[] args)
  {
    int myNum = 15;
    myNum = 20;  // myNum is now 20
    System.out.println(myNum);
  }
}
```

PROGRAM 4:

```
public class Main
{
  public static void main(String[] args)
  {
    String name = "abc";
    System.out.println("Hello " + name);
  }
}
```

PROGRAM 5:

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5;
    int y = 6;
    System.out.println(x + y); // Print the value of x + y
  }
}
```

PROGRAM 6:

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5, y = 6, z = 50;
    System.out.println(x + y + z);
  }
}
```

EXERCISE :

1. Create a java program for declaring and initializing 5 variables and print them

2. Create a java program with variables of different types and print them

CONSTANTS:

The final keyword is added if you don't want others (or yourself) to overwrite existing values (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

PROGRAM 7:

```
public class Main
{
  public static void main(String[] args)
  {
    final int myNum = 15;
    myNum = 20; // will generate an error
    System.out.println(myNum);
  }
}
```

EXERCISE:

1. Create a java program to declare Pi value and print the PI value

OPERATORS:

Operators are used to perform operations on variables and values.

PROGRAM 8:

```
public class MyClass
{
    public static void main(String args[])
{
      int x=10;
      int y=25;
      int z=x+y;
      System.out.println("Sum of x+y = " + z);
      int z=x-y;
      System.out.println("Subtraction of x-y = " + z);
      int z=x*y;
      System.out.println("Multiplication of x*y = " + z);
      int z=x/y;
      System.out.println("Division of x/y = " + z);
      int z=x%y;
      System.out.println("Remainder of x%y = " + z);
    }
}
```

PROGRAM 9:

```
public class Main
{
  public static void main(String[] args)
{
    int x = 5;
    ++x;
    System.out.println(x);
    --x;
    System.out.println(x);
  }
}
```

PROGRAM 9:

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5;
    System.out.println(x);
    x += 3;
    System.out.println(x);
    x -= 3;
    System.out.println(x);
    x *= 3;
    System.out.println(x);
    x /= 3;
    System.out.println(x);
    x %= 3;
    System.out.println(x);
  }
}
```

PROGRAM 10:

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5;
    int y = 3;
    System.out.println(x == y);
    System.out.println(x != y);
    System.out.println(x > y);
    System.out.println(x < y);
    System.out.println(x >= y);
    System.out.println(x <= y);
  }
}
```

PROGRAM 11:

```
public class Main
{
  public static void main(String[] args)
  {
    int x = 5;
```

System.out.println(x > 3 && x < 10); // returns true because 5 is greater than 3 AND 5 is less than 10

System.out.println(x > 3 || x < 4); // returns true because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)

System.out.println(!(x > 3 && x < 10)); // returns false because ! (not) is used to reverse the result

}

}

EXERCISE :

1. Create a java program to Calculate Area of Circle,Rectangle and Triangle
2. Create a java program to find Greatest and smallest among three numbers
3. Create a java program to find the average of two numbers
4. Create a java program to get marks of a student and calculate sum as well as average of a student

DECISION MAKING STATEMENT:

Decision Making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled.
A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

IF CONDITION:

if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

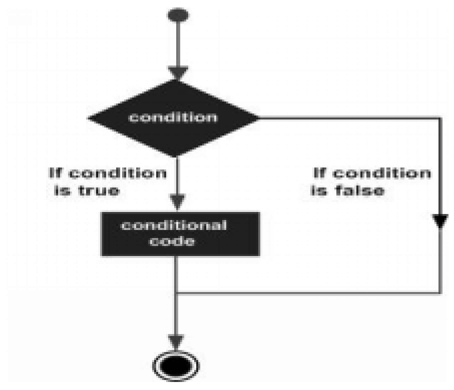
SIMPLE IF:

SYNTAX:

if(condition)

```
{
        /*Statements to execute if
        condition is true*/
}
```

FLOW CHART:



PROGRAM 12:

```
public class Test
{
  public static void main(String args[])
  {
     int x = 10;
     if( x < 20 )
     {
        System.out.print("This is if statement");
     }
```

```
   }
}
```

PROGRAM 13:

```
public class Logical
{
        public static void main(String[] args)
        {
                int a = 10, b = 20, c = 20, d = 0;
                System.out.println("Var1 = " + a);
                System.out.println("Var2 = " + b);
                System.out.println("Var3 = " + c);
                if ((a < b) && (b == c))
               {
                        d = a + b + c;
                        System.out.println("The sum is: " + d);
                }
        }
}
```

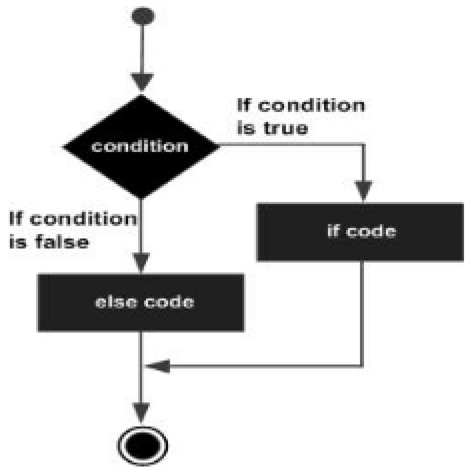
IF ELSE CONDITION:

SYNTAX:

```
if(condition)
{
        /*Executes this block if
```

```
    condition is true */
}
else
{
    /*Executes this block if
    condition is false*/
}
```

FLOW CHART:



PROGRAM 14:

```
public class Logical
{
    public static void main(String[] args)
    {
        int a = 10, b = 1, c = 10, d = 30;
         System.out.println("Var1 = " + a);
        System.out.println("Var2 = " + b);
```

```
                System.out.println("Var3 = " + c);
                System.out.println("Var4 = " + d);
                if (a > b || c == d)
                        System.out.println("One or both" + " the conditions are true");
                else
                        System.out.println("Both the" + " conditions are false");
                if (!(a > b || c == d))
                        System.out.println("One or both" + " the conditions are true");
                else
                        System.out.println("Both the" + " conditions are false");
        }
}
```

PROGRAM 15:

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

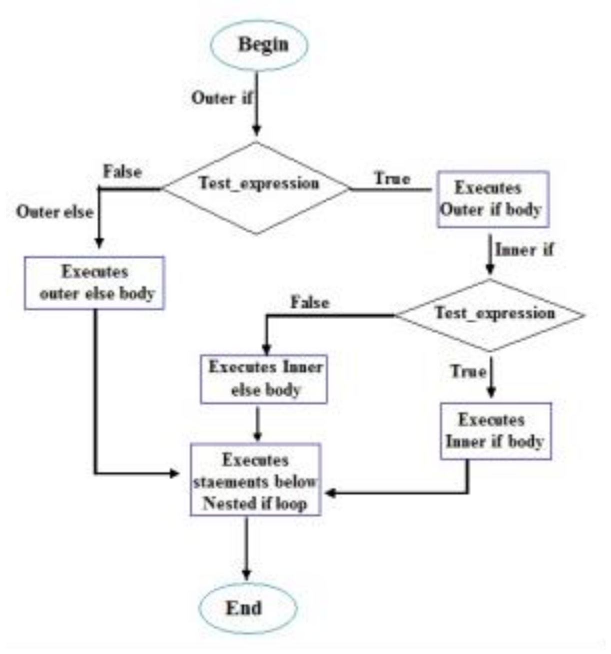
```
      {
         System.out.print("This is else statement");
      }
   }
}
```

NESTED IF CONDITION:

SYNTAX:

```
if(condition1)
{
          //executes if the condition 1 is true
          if(condition2)
          {
                    //Executes if the condition 2 is true
          }
}
```

FLOW CHART:



PROGRAM 16:

```
public class Test
{
   public static void main(String args[])
   {
      int x = 30;
      int y = 10;
      if( x == 30 )
      {
         if( y == 10 )
         {
            System.out.print("X = 30 and Y = 10");
         }
```

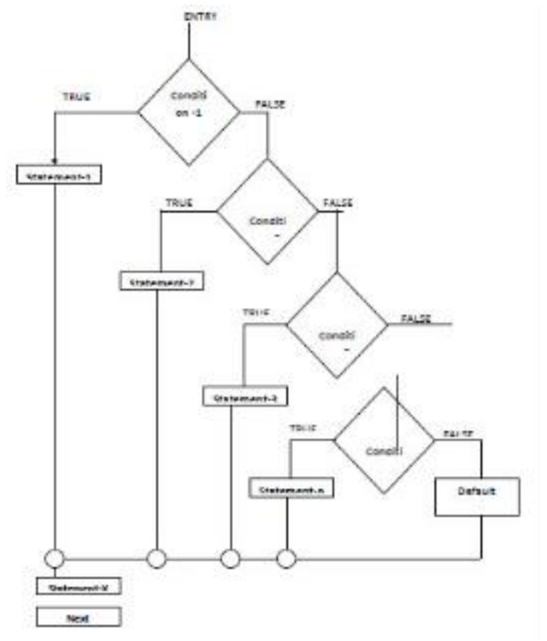
```
    }
  }
}
```

ELSE IF LADDER:

SYNTAX:

```
if (condition)
        statement;
else if(condition)
        statements;
.
.
.
else
        statement;
```

FLOW CHART:



PROGRAM 17:

```
public class Main
{
  public static void main(String[] args)
{
    int number = 0;
    if (number > 0)
   {
      System.out.println("The number is positive.");
    }
     else if (number < 0)
    {
      System.out.println("The number is negative.");
    }
     else
    {
      System.out.println("The number is 0.");
    }
  }
}
```

EXERCISE:

1. Create a java program to find whether the given number is odd or even
2. Create a java program to find whether the given number is positive or negative
3. Create a java program that asks the user to enter three inputs and find the largest among three values
4. Create a java to get the employee name , salary and bonus. And print the salary of an employee with the help of Else if ladder condition

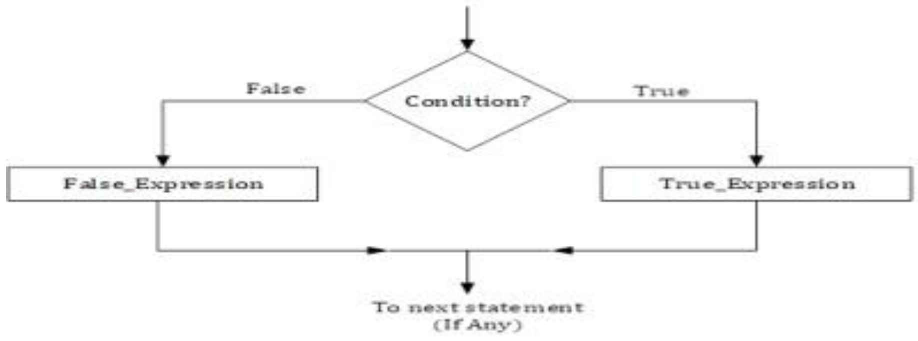
TERNARY OPERATOR:

The **ternary operator** consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

SYNTAX:

Expression1? expression2: expression3;

FLOW CHART:



PROGRAM 18:

```
public class Test
{
   public static void main(String args[])
  {
      int a, b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );
      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

EXERCISE:

1. Create a java program to find whether the given year is leap year or not

SWITCH STATEMENTS:

SIMPLE SWITCH:

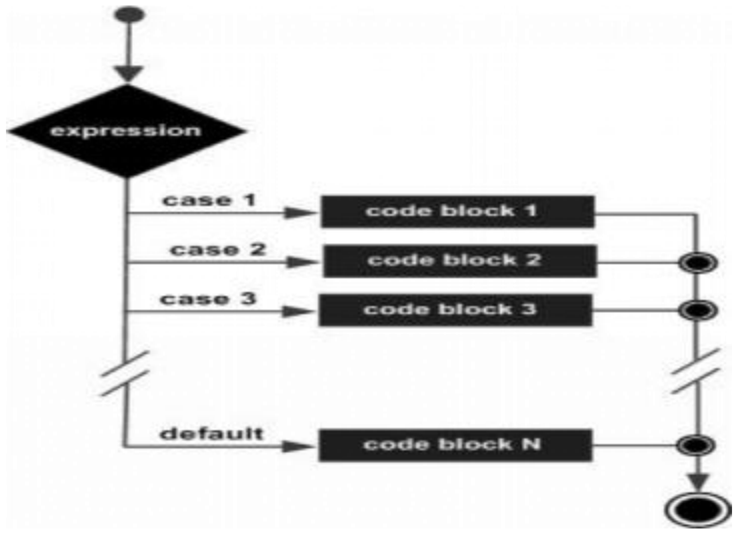
Use the switch statement to select one of many code blocks to be executed.

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and default keywords are optional
- When Java reaches a break keyword, it breaks out of the switch block.
- The default keyword specifies some code to run if there is no case match:

SYNTAX:

```
switch(expression)
{
        case constant-expression:
                statement1;
                break; //optional
        case constant-expression:
                statement2;
                break; //optional
                        //you can have any number of case statements
        default: //optional
                statement n;
}
```

FLOW CHART:



PROGRAM 19:

```
public class Main
{
  public static void main(String[] args)
{
    int day = 4;
    switch (day)
   {
      case 1:
        System.out.println("Monday");
        break;
      case 2:
        System.out.println("Tuesday");
        break;
      case 3:
        System.out.println("Wednesday");
        break;
      case 4:
        System.out.println("Thursday");
        break;
      case 5:
        System.out.println("Friday");
        break;
      case 6:
        System.out.println("Saturday");
        break;
      case 7:
        System.out.println("Sunday");
        break;
      default:
          System.out.println("Invalid input");
    }
  }
}
```

NESTED SWITCH:

SYNTAX:

```
switch(condition)
{
          case constant-expression:
                    statement1;
                    break; //optional
          switch(condition)
          {
                    case constant-expression:
                              statement1;
                              break; //optional
                    case constant-expression:
                              statement1;
                              break; //optional
          }
          case constant-expression:
                    statement1;
                    break; //optional
}
```

PROGRAM 20:

```
public class NestedSwitchExample
{
    public static void main(String args[])
      {
        char branch = 'C';
        int collegeYear = 2;
        switch( collegeYear )
        {
            case 1:
                System.out.println("English, Maths, Science");
                break;
            case 2:
                switch( branch )
                {
```

```
            case 'C':
                System.out.println("Operating System, Java, Data Structure");
                break;
            case 'E':
                System.out.println("Micro processors, Logic switching theory");
                break;
            case 'M':
                System.out.println("Drawing, Manufacturing Machines");
                break;
        }
        break;
    case 3:
        switch( branch )
        {
            case 'C':
                System.out.println("Data Communication and Networks,
MultiMedia");
                break;
            case 'E':
                System.out.println("Embedded System, Image Processing");
                break;
            case 'M':
                System.out.println("Production Technology, Thermal Engineering");
                break;
            }
            break;
        }
    }
}
```

EXERCISE:

1. Create a java program to perform all the arithmetic operations with the help of switch statement
2. Create a java program to perform all the relational operations with the help of switch statement
3. Create a java program to find the area of different shapes with the help of switch statement
4. Create a java program to fing the grade of a student with the help of switch statement

LOOPING STATEMENTS:

Loops can execute a block of code as long as a specified condition is reached.

Loops are handy because they save time, reduce errors, and they make code more readable.

WHILE LOOP:

SYNTAX:

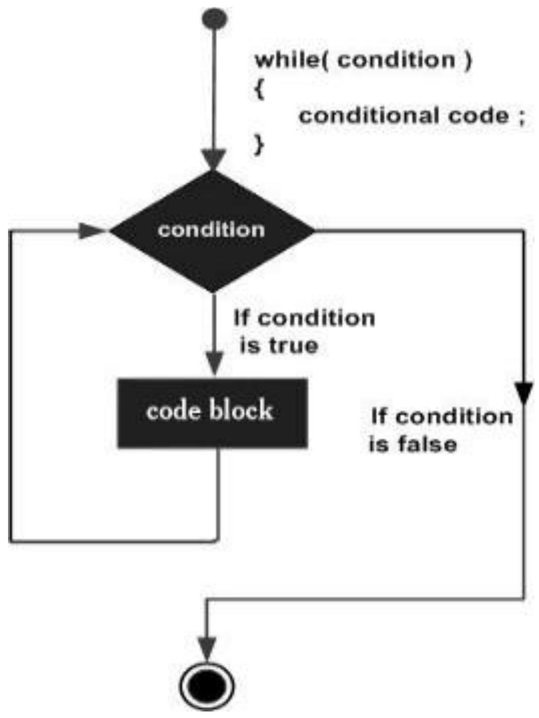
initialization;

while (**condition**)

{

```
  // statements
  Increment/decrement;
}
```

FLOW CHART:



PROGRAM 21:

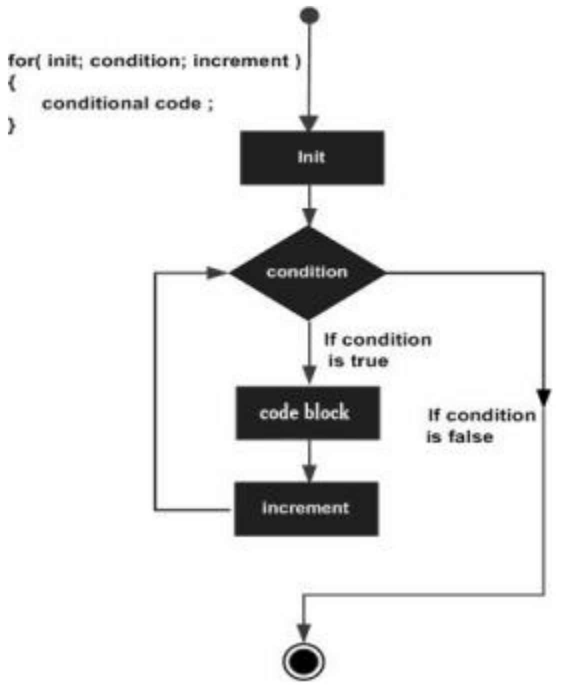
```
public class Test
{
  public static void main(String args[])
  {
    int x = 10;
    while( x < 20 )
    {
        System.out.print("value of x : " + x );

        x++;

        System.out.print("\n");

      }

   }

}
```

FOR LOOPS:

SYNTAX:

```
for(initialization,condition,increment)

{

         //statements;

}
```

FLOW CHART:



PROGRAM 22:

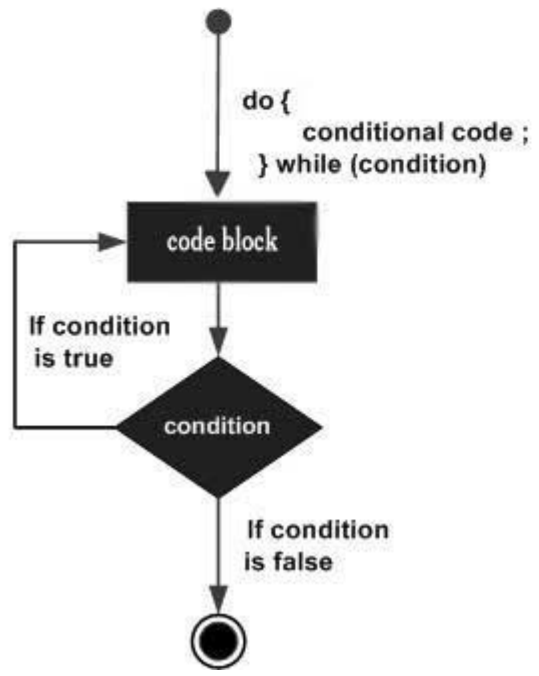
```
public class Test
{
   public static void main(String args[])
  {
      for(int x = 10; x < 20; x = x + 1)
     {
         System.out.print("value of x : " + x );
         System.out.print("\n");
      }
   }
}
```

DO WHILE LOOP:

SYNTAX:

```
initialization;
do
{
   // statements
   Increment/decrement;
} while (condition);
```

FLOW CHART:



PROGRAM 3:

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

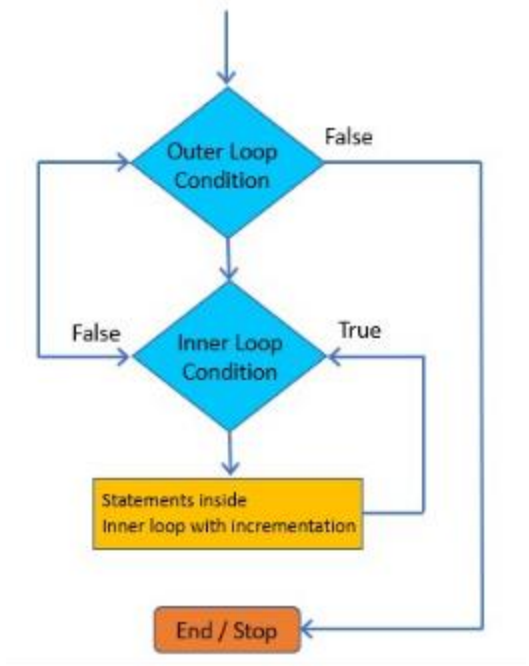
NESTED LOOPS:

SYNTAX:

```
do
{
  while(condition)
{
    for ( initialization; condition; increment )
    {
          // statement of inside for loop
    }
    // statement of inside while loop
  }
  // statement of outer do-while loop
}while(condition);
```

FLOW CHART:

}



PROGRAM 24:

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

EXERCISE:

1. Create a java program to calculate the sum of 10 natural numbers with loops
2. Create a java program to calculate the factorial value of a given number
3. Create a java program to calculate the multiplication table with a given number
4. Create a java program to find the Fibonacci series of given values(Possible output : 0 1 1 2 3 5 8 13 21 34)

JUMPING STATEMENTS:

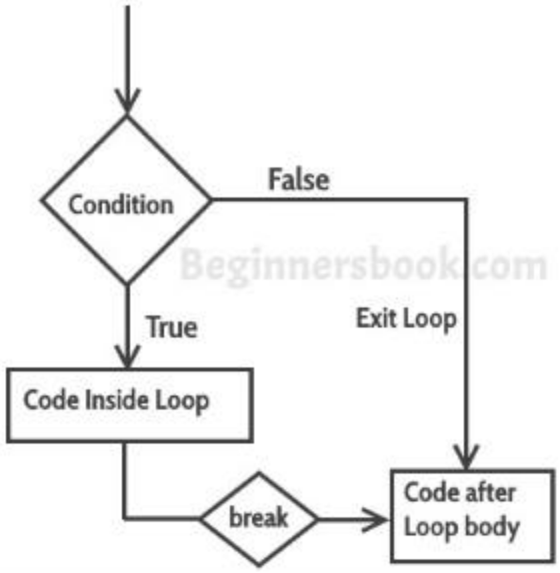
There are four types of jump statements in java

- Break
- Goto
- Continue
- Return

BREAK:

In java the break statement has 3 uses

- Terminates a statement sequence in a switch statement
- It can be used to exit loop
- It can be used more civilized form of goto statement
  - **SYNTAX:**
  - break;
  -
  - **FLOW CHART:**

- 

PROGRAM 25:

```
public class BreakDemo
{
        public static void main(String args[])
        {
                for (int i = 0; i < 10; i++)
                {
                                if (i == 5)
                                break;
                        System.out.println("i: " + i);
                }
                System.out.println("Loop complete.");
        }
}
```

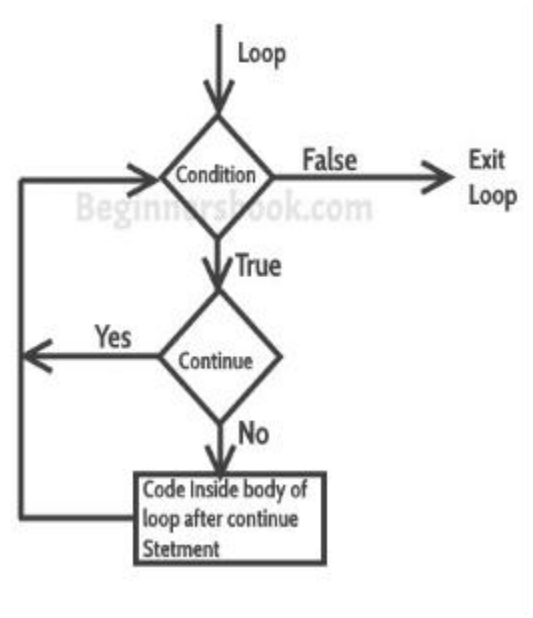
CONTINUE:

A continue statement causes control to be transferred directly to the conditional expression that controls the loop

SYNTAX:

continue;

FLOW CHART:



PROGRAM 26:

```
public class ContinueDemo
{
        public static void main(String args[])
        {
                for (int i = 0; i < 10; i++)
                {
                        if (i%2 == 0)
```

```
                                    continue;
            System.out.print(i + " ");
                    }
          }
}
```

RETURN:

The return statement is used to explicitly return from a method.That is,it causes program control to transfer back to the caller of the method.At any time in a method the return statement can be used to cause execution to branch back to the caller of the method.Thus,the return statement can be used to cause execution to branch back to the caller to the method.Thus, the return statement immediately terminates the method in which it is executed.

PROGRAM 27:

```
public class Return
{
          public static void main(String args[])
          {
                    boolean t = true;
                    System.out.println("Before the return.");
                    if (t)
                              return;
          System.out.println("This won't execute.");
          }
}
```

STRINGS:

Strings are used for storing text.

A String variable contains a collection of characters surrounded by double quotes:

PROGRAM 28:

```
public class StringExample

{

public static void main(String args[])

{

String s1="java";

char ch[]={'s','t','r','i','n','g','s'};

String s2=new String(ch);

String s3=new String("example");

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}

}
```

PROGRAM 29:

```
public class Comparison

{

 public static void main(String args[])

{
   String s1="Sachin";
   String s2="SACHIN";
   String s3=new String("Sachin");
   System.out.println(s1.equals(s2));
   System.out.println(s1.equalsIgnoreCase(s2));
   System.out.println(s1==s2);
   System.out.println(s1==s3);
   System.out.println(s1.compareTo(s2));
   System.out.println(s1.compareTo(s3));
   System.out.println(s3.compareTo(s1));
 }
}
```

PROGRAM 30:

```
public class Concatenation
{
 public static void main(String args[])
{
   String s1="Hai";
   String s2=" Welcome";
   String s3=s1.concat(s2);
   System.out.println(s3);
   String s="Sachin"+" Tendulkar";
   System.out.println(s);
}
}
```

PROGRAM 31:

```
public class Substring
{
 public static void main(String args[])
{
 String s="SachinTendulkar";
 System.out.println("Original String: " + s);
 System.out.println("Substring starting from index 6: " +s.substring(6));
 System.out.println("Substring starting from index 0 to 6: "+s.substring(0,6));
 }
}
```

PROGRAM 32:

```
public class Stringoperation
{
public static void main(String args[])
{
String s="Sachin";
System.out.println(s.toUpperCase());
System.out.println(s.toLowerCase());
System.out.println(s);
}
}
```

PROGRAM 33:

```
public class Stringoperation
{
public static void main(String args[])
{
String s="  Sachin  ";
System.out.println(s);
System.out.println(s.trim());
}
}
```

PROGRAM 34:

```
public class Stringoperation
{
public static void main(String args[])
{
String s="Sachin";
 System.out.println(s.startsWith("Sa"));
 System.out.println(s.endsWith("n"));
}
}
```

PROGRAM 35:

```
public class Stringoperation
{
public static void main(String ar[])
{
String s="Sachin";
System.out.println(s.charAt(0));
System.out.println(s.charAt(3));
System.out.println(s.length());
}
}
```

PROGRAM 36:

```
public class Stringoperation
{
public static void main(String ar[])
{
int a=10;
String s=String.valueOf(a);
System.out.println(s+10);
}
}
```

PROGRAM 37:

```
public class Stringoperation
{
public static void main(String ar[])
{
String s1="Java is a programming language. Java is a platform. Java is an Island."
String replaceString=s1.replace("Java","Kava");
System.out.println(replaceString);
}
}
```

ARRAYS:

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

PROGRAM 38:

```
public class A
{
        public static void main (String[] args)
        {
        int[] arr;
        arr = new int[5];
        arr[0] = 10;
        arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;

        arr[4] = 50;

        for (int i = 0; i < arr.length; i++)

                System.out.println("Element at index " + i +" : "+ arr[i]);

        }

}
```

PROGRAM 39:

```
public class Array1

{

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]);

}

}
```

PROGRAM 40:

```
public class multiDimensional

{

        public static void main(String args[])

        {

                int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };

                for (int i=0; i< 3 ; i++)
                {
                        for (int j=0; j < 3 ; j++)
                                System.out.print(arr[i][j] + " ");
                        System.out.println();
                }
        }
}
```

PROGRAM 41:

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

}

}
```

FUNCTIONS/ METHODS:

A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

The main advantage of methods is reusability of code.

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**.

PROGRAM 42:

```
public class Main

{

  static void myMethod()

  {

    System.out.println("I just got executed!");

  }

  public static void main(String[] args)

  {

    myMethod();

    myMethod();

    myMethod();

  }
}
```

PROGRAM 43:

```
public class Main
{
  static void myMethod(String fname)
  {
    System.out.println(fname + "ABC");
  }
  public static void main(String[] args)
  {
    myMethod("PQR");
    myMethod("XYZ");
    myMethod("UVW");
  }
}
```

PROGRAM 44:

```
import java.util.*;
public class Main
{
  static int myMethod(int x,int y)
  {
    return x + y;
  }
  public static void main(String[] args)
  {
    Scanner sc=new Scanner(System.in);
    System.out.print("Enter first number- ");
    int a= sc.nextInt();
    System.out.print("Enter second number- ");
    int b= sc.nextInt();
    int c=myMethod(a,b);
    System.out.println(c);
  }
}
```

PROGRAM 45:

```
public class Main
{
  static int plusMethod(int x, int y)
  {
    return x + y;
  }
  static double plusMethod(double x, double y)
  {
    return x + y;
  }
  public static void main(String[] args)
  {
    int myNum1 = plusMethod(8, 5);
    double myNum2 = plusMethod(4.3, 6.26);
    System.out.println("int: " + myNum1);
    System.out.println("double: " + myNum2);
  }
}
```

PROGRAM 46:

```
public class Main
{
  public static void main(String[] args)
  {
    int result = sum(10);
    System.out.println(result);
  }
  public static int sum(int k)
  {
    if (k > 0)
    {
      return k + sum(k - 1);
    }
    else
    {
      return 0;
    }
  }
}
```

OOPS CONCEPT:

OOP stands for **Object-Oriented Programming**.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

CLASS AND OBJECT:

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

To create a class, use the keyword class

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the class name, followed by the object name, and use the keyword new

Variables are usually called as attributes of class and functions are class methods

PROGRAM 47:

```
public class Main
{
  int x = 5;
  public static void main(String[] args)
  {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

PROGRAM 48:

```
public class Main
{
  int x = 5;
  public static void main(String[] args)
  {
    Main myObj1 = new Main();
    Main myObj2 = new Main();
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

PROGRAM 49:

```
public class Main
{
  static void myMethod()
  {
    System.out.println("Hello World!");
  }
  public static void main(String[] args)
  {
    myMethod();
  }
}
```

POLYMORPHISM:

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

PROGRAM 50:

```
public class Sum
{
          public int sum(int x, int y)
          {
                    return (x + y);
          }
```

```
        public int sum(int x, int y, int z)

        {

                return (x + y + z);

        }

        public double sum(double x, double y)

        {

                return (x + y);

        }

        public static void main(String args[])

        {

                Sum s = new Sum();

                System.out.println(s.sum(10, 20));

                System.out.println(s.sum(10, 20, 30));

                System.out.println(s.sum(10.5, 20.5));

        }

}
```

ENCAPSULATION:

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as private
- provide public **get** and **set** methods to access and update the value of a private variable

PROGRAM 51:

```
class Area
{
  int length;
  int breadth;
  Area(int length, int breadth)
  {
    this.length = length;
    this.breadth = breadth;
  }
public void getArea()
{
    int area = length * breadth;
    System.out.println("Area: " + area);
  }
}
public class Main
{
  public static void main(String[] args)
  {
    Area rectangle = new Area(5, 6);
    rectangle.getArea();
  }

}
```

DATA ABSTRACTION:

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods

PROGRAM 52:

```
abstract class Shape

{

abstract void draw();

}

class Rectangle extends Shape

{

void draw()

{

   System.out.println("drawing rectangle");

}

}
```

```
class Circle1 extends Shape

{

void draw()

{

   System.out.println("drawing circle");

}

}

public class Test

{

public static void main(String args[])

{

Shape s=new Circle1();

s.draw();

}

}
```

INHERITANCE:

In Java, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories:

- **subclass** (child) - the class that inherits from another class
- **superclass** (parent) - the class being inherited from
- **Reusability-**As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

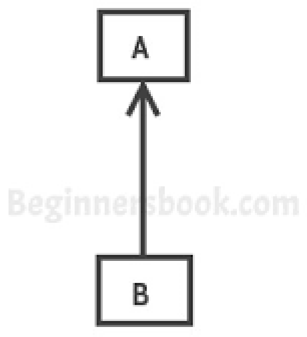
To inherit from a class, use the extends keyword.

}

SINGLE INHERITANCE:

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

FLOW CHART:



Single Inheritance

PROGRAM 53:

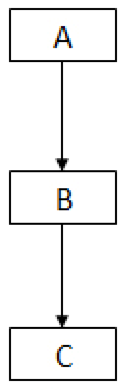
```
class one
{
        public void print1()
        {
                System.out.println("ABC");
        }
}
class two extends one
{
        public void print2() { System.out.println("XYZ"); }
}
```

```
public class Main
{
        public static void main(String[] args)
        {
                two a = new two();
                a.print1();
                a.print2();
        }
}
```

MULTILEVEL INHERITANCE:

When there is a chain of inheritance, it is known as *multilevel inheritance*

FLOW CHART:



Multilevel Inheritance

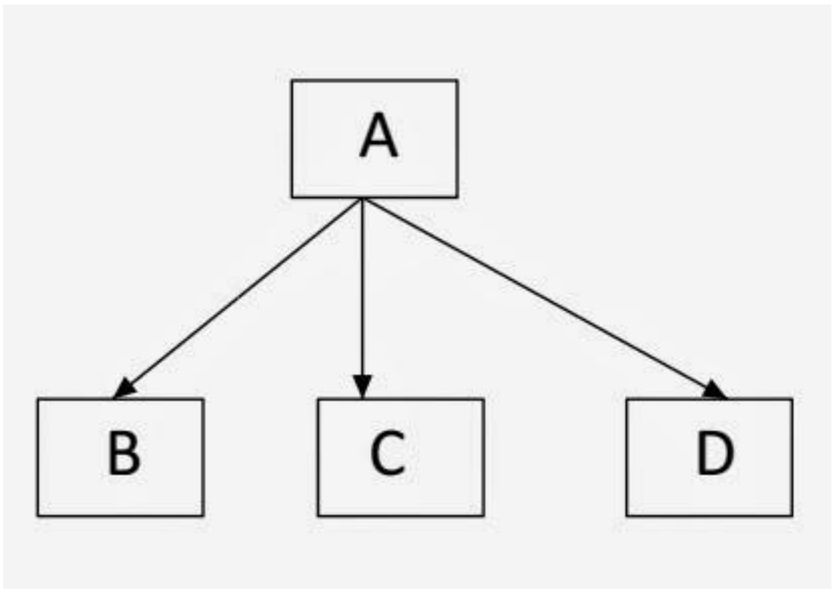
PROGRAM 54:

```
class one
{
        public void print1()
        {
                System.out.println("abc");
        }
}
class two extends one
{
        public void print2()
        {
         System.out.println("XYZ");
        }
}
class three extends two
{
        public void print3()
        {
                System.out.println("PQR");
        }
}
public class Main
{
        public static void main(String[] args)
        {
                three a = new three();
                a.print1();

                a.print2();

                a.print3();

        }

}
```

HIERARCHICAL INHERITANCE:

When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

FLOW CHART:



PROGRAM 55:

```
class A

{

        public void print_A()

        {

           System.out.println("Class A");
```

```
          }
}
class B extends A
{
          public void print_B()
          {
              System.out.println("Class B");
          }
}
class C extends A
{
          public void print_C()
          {
              System.out.println("Class C");
          }
}
class D extends A
{
          public void print_D()
          {
              System.out.println("Class D");
          }
}
```

```
public class Test
{
        public static void main(String[] args)
        {
                B obj_B = new B();
                obj_B.print_A();
                obj_B.print_B();
                C obj_C = new C();
                obj_C.print_A();
                obj_C.print_C();
                D obj_D = new D();
                obj_D.print_A();
                obj_D.print_D();
        }
}
```

MULTIPLE INHERITANCE:

Multiple Inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with same signature in both the super classes and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

But java doesn’t support this multiple inheritance. So to obtain this facility we can include interface.

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

PROGRAM 56:

```
interface Drawable

{

void draw();

}

class Rectangle implements Drawable

{

public void draw()

{

   System.out.println("drawing rectangle");

}

}

class Circle implements Drawable{

public void draw()

{

   System.out.println("drawing circle");

}

}

public class Test

{

public static void main(String args[])
{
Drawable d=new Circle();
Drawable d1=new Rectangle();
d.draw();
d1.draw();
}
}
```