CSCI 125 Lecture 3

Java Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

```
public static int methodName(int a, int b) { // body}
```

- public static − modifier
- int − return type
- methodName − name of the method
- a, b − formal parameters
- int a, int b − list of parameters

```
modifier returnType nameOfMethod (Parameter List) {
   // method body
}
```

The syntax shown above includes −

- modifier − It defines the access type of the method and it is optional to use.
- returnType − Method may return a value.
- nameOfMethod − This is the method name. The method signature consists of the method name and the parameter list.
- Parameter List − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.
- method body − The method body defines what the method does with the statements.

Method Calling

For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).

The process of method calling is simple. When a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when −

- the return statement is executed.
- it reaches the method ending closing brace.

The void Keyword

The void keyword allows us to create methods which do not return a value. Here, in the following example we're considering a void method *methodRankPoints*. This method is a void method, which does not return any value. Call to a void method must be a statement i.e. *methodRankPoints(255.7);*. It is a Java statement which ends with a semicolon as shown in the following example.

```
  public static void methodRankPoints(double points) {return points;}

  public static void main(String[] args) {

      methodRankPoints(255.7);

   }
```

Passing Parameters by Value

While working under calling process, arguments is to be passed. These should be in the same order as their respective parameters in the method specification. Parameters can be passed by value or by reference.

Passing Parameters by Value means calling a method with a parameter. Through this, the argument value is passed to the parameter.

```
  public static void swapFunction(int a, int b) {}
public static void main(String[] args) {

      int a = 30;
      int b = 45;
      System.out.println("Before swapping, a = " + a + " and b = " + b);
      // Invoke the swap method
      swapFunction(a, b);
      System.out.println("\n**Now, Before and After swapping values will be same here**:)");
      System.out.println("After swapping, a = " + a + " and b is " + b);
   }
```

```
public class ExampleMinNumber {



   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }
   /** returns the minimum of two numbers */
   public static int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      Else
         min = n1;
      return min;
   }

}
```

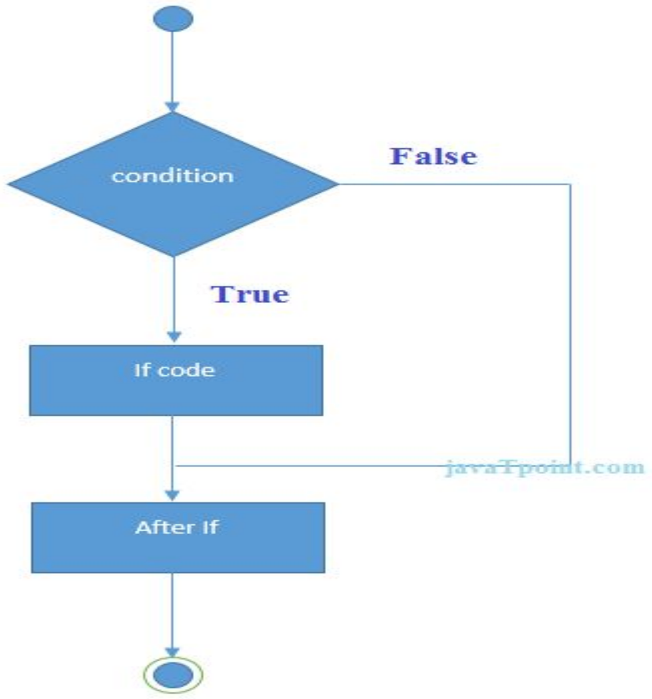
Java If-else Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in Java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

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

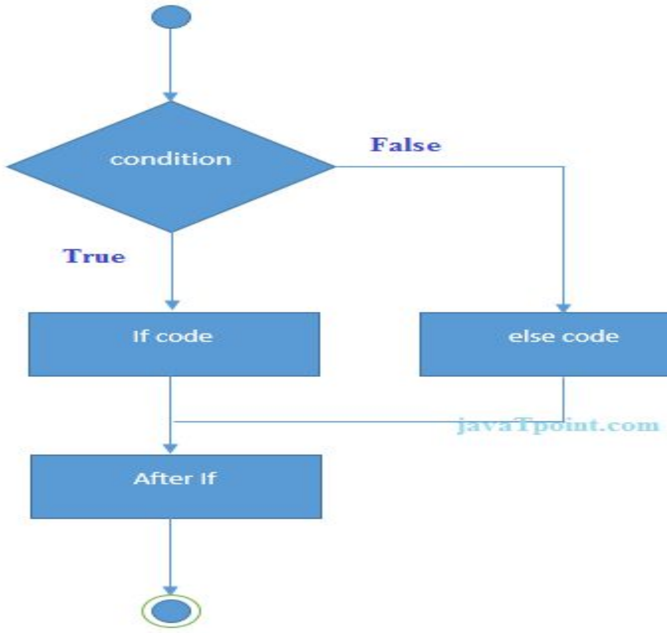
.



```
public class IfExample {

public static void main(String[] args) {

    //defining an 'age' variable

    int age=20;

    //checking the age

    if(age>18){

        System.out.print("Age is greater than 18");

    }

}

}
```

Java 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.



```
public class IfElseExample {

public static void main(String[] args) {

    //defining a variable

    int number=13;

    //Check if the number is divisible by 2 or not

    if(number%2==0){

        System.out.println("even number");

    }else{

        System.out.println("odd number");

    }

}

}
```

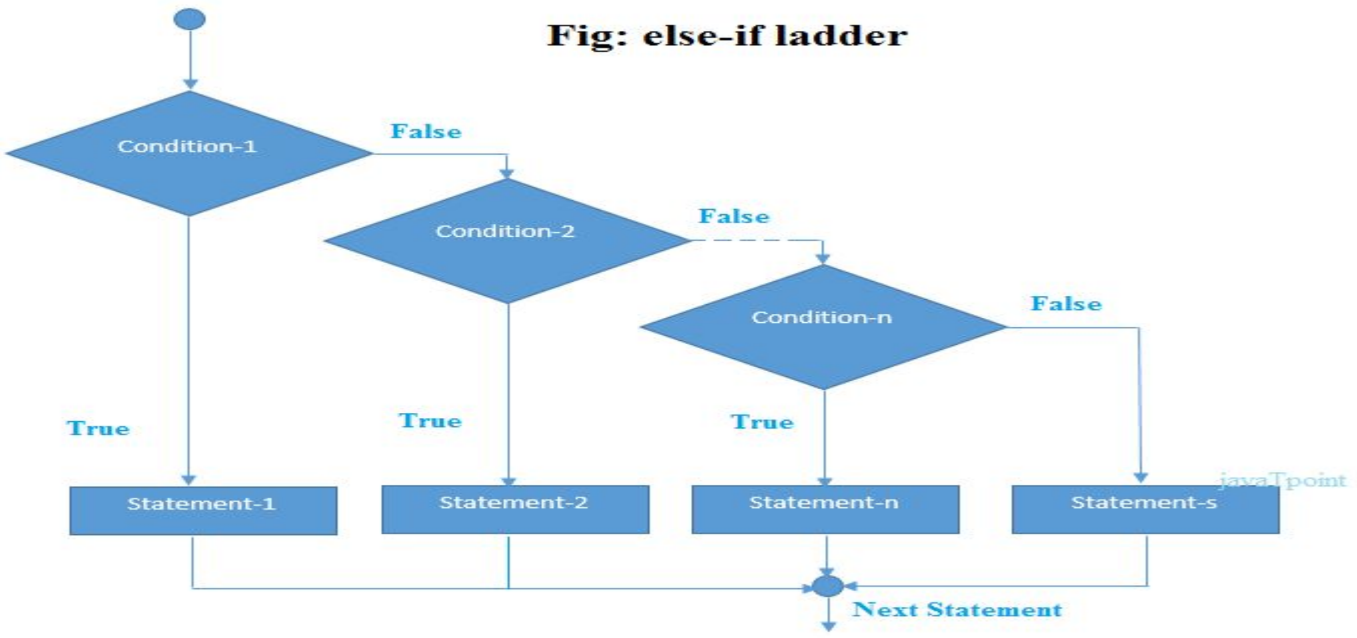
Using Ternary Operator

We can also use ternary operator (? :) to perform the task of if...else statement. It is a shorthand way to check the condition. If the condition is true, the result of ? is returned. But, if the condition is false, the result of : is returned.

```
public class IfElseTernaryExample {

public static void main(String[] args) {

    int number=13;

    //Using ternary operator

    String output=(number%2==0)?"even number":"odd number";

    System.out.println(output);

}

}
```

Java if-else-if ladder Statement

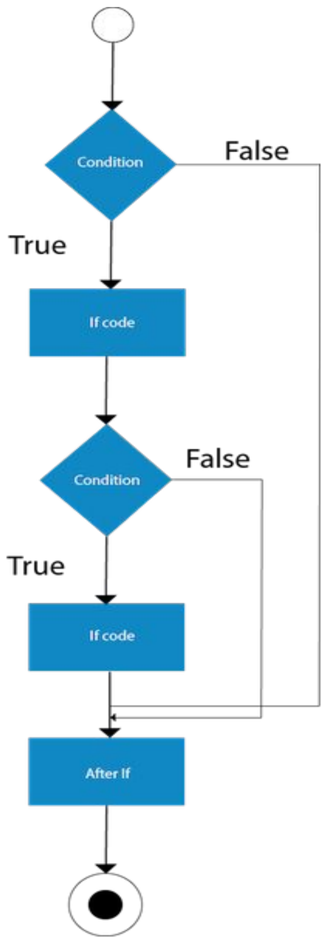
The if-else-if ladder statement executes one condition from multiple statements.



```
public class IfElseIfExample {
public static void main(String[] args) {
  int marks=65;
 if(marks<50){
 System.out.println("fail");   }

    else if(marks>=50 && marks<60){   System.out.println("D grade");   }

    else if(marks>=60 && marks<70){  System.out.println("C grade");  }

    else if(marks>=70 && marks<80){  System.out.println("B grade");  }

    else if(marks>=80 && marks<90){    System.out.println("A grade");

    }else if(marks>=90 && marks<100){

        System.out.println("A+ grade");   }else{

        System.out.println("Invalid!");

    }

}

}
```

Java Nested if statement

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

```



```
public class JavaNestedIfExample {

public static void main(String[] args) {

    //Creating two variables for age and weight

    int age=20;

    int weight=80;

    //applying condition on age and weight

    if(age>=18){

        if(weight>50){

            System.out.println("You are eligible to donate blood");

        }

    }

}}
```

Java Switch Statement

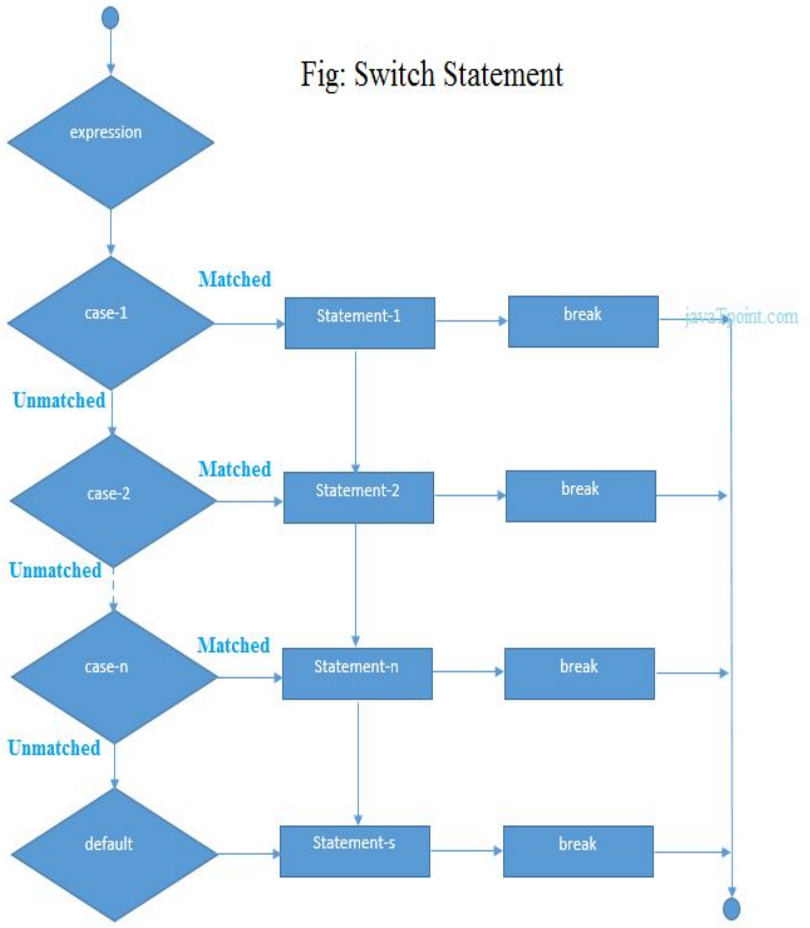
The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.

In other words, the switch statement tests the equality of a variable against multiple values.

Points to Remember

- There can be *one or N number of case values* for a switch expression.
- The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.
- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type),* *enums* *and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

```
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;

}
```



```
public class SwitchExample {
public static void main(String[] args) {
    //Declaring a variable for switch expression
    int number=20;
    //Switch expression
    switch(number){
    //Case statements
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    //Default case statement
    default:System.out.println("Not in 10, 20 or 30");
    }
}
}
```

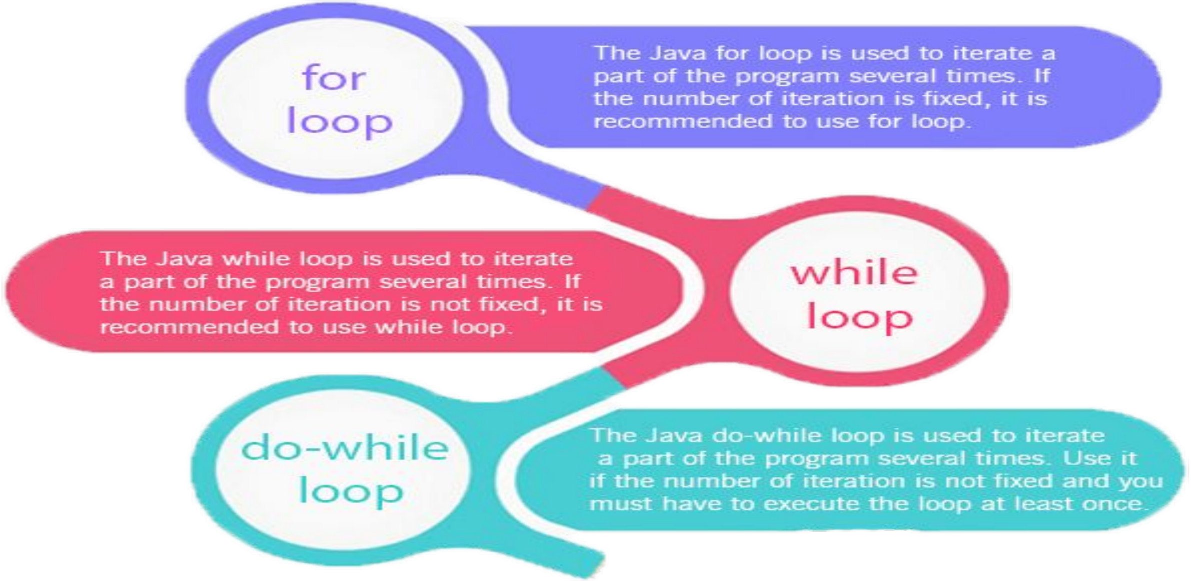
```java
public class SwitchMonthExample {
public static void main(String[] args) {
   //Specifying month number
   int month=7;
   String monthString="";
   //Switch statement
   switch(month){
   //case statements within the switch block
   case 1: monthString="1 - January";
   break;
   case 2: monthString="2 - February";
   break;
   case 3: monthString="3 - March";
   break;
   case 4: monthString="4 - April";
   break;
   case 5: monthString="5 - May";
   break;
   case 6: monthString="6 - June";
   break;
   case 7: monthString="7 - July";
   break;
      default:System.out.println("Invalid Month!");
   }
   //Printing month of the given number
   System.out.println(monthString);
}
}
```

Loop

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

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 −


for loop
The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
while loop
The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.
do-while loop
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

Java For Loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loops in java.

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop

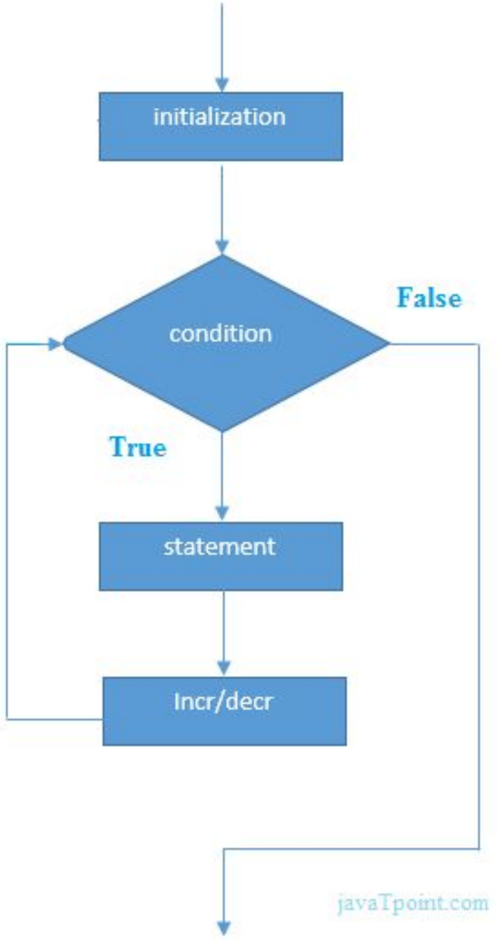
Java Simple For Loop

A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement**: The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

```java
//Java Program to demonstrate the example of for loop
//which prints table of 1
public class ForExample {
public static void main(String[] args) {
    //Code of Java for loop
    for(int i=1;i<=10;i++){
        System.out.println(i);
    }
}
}
```



Java Nested For Loop

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);
}//end of i
}//end of j
}
}
```

Java 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.

```
for(Type var:array){

//code to be executed

}

public class ForEachExample {

public static void main(String[] args) {

    //Declaring an array

    int arr[]={12,23,44,56,78};

    //Printing array using for-each loop

    for(int i:arr){

        System.out.println(i);

    }

}

}
```

Java Labeled For Loop

We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop. Usually, break and continue keywords breaks/continues the innermost for loop only.

```
labelname:
for(initialization;condition;incr/decr){
//code to be executed
}
```

```
public class LabeledForExample {
public static void main(String[] args) {
    //Using Label for outer and for loop
    aa:
        for(int i=1;i<=3;i++){
            bb:
                for(int j=1;j<=3;j++){
                    if(i==2&&j==2){
                        break aa;
                    }
                    System.out.println(i+" "+j);
                }      } } }
```

Java Infinitive For Loop

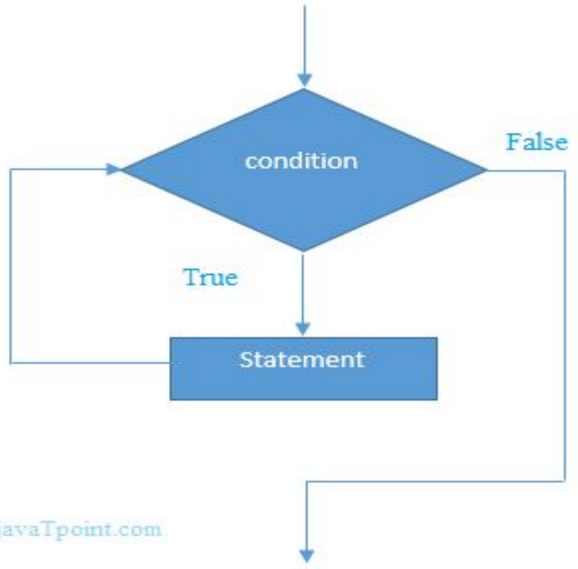
If you use two semicolons ;; in the for loop, it will be infinitive for loop.

```
for(;;){
//code to be executed
}
//Java program to demonstrate the use of infinite for loop
//which prints an statement
public class ForExample {
public static void main(String[] args) {
    //Using no condition in for loop
    for(;;){
        System.out.println("infinitive loop");
    }
}
}
```

Java While Loop

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

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



Java While Loop

```
public class WhileExample {
public static void main(String[] args) {
    int i=1;
    while(i<=10){
        System.out.println(i);
    i++;
    }
}
}
```

Java Infinitive While Loop

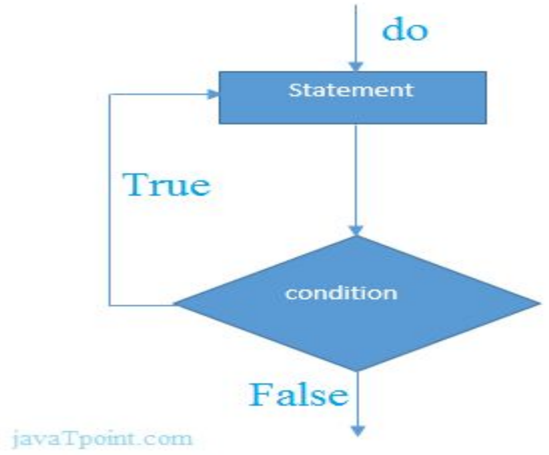
```
while(true){
//code to be executed
}
public class WhileExample2 {
public static void main(String[] args) {
    while(true){
        System.out.println("infinitive while loop");
    }
}
}
```

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

```
do{
//code to be executed
}while(condition);
```



Do While

```
public class DoWhileExample {
public static void main(String[] args) {
    int i=1;
    do{
        System.out.println(i);
    i++;
    }while(i<=10);
}
```

Java Infinitive do-while Loop

```
do{
//code to be executed
}while(true);
```

```
public class DoWhileExample2 {
public static void main(String[] args) {
    do{
        System.out.println("infinitive do while loop");
    }while(true);
}
}
```

Java 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* statement 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.

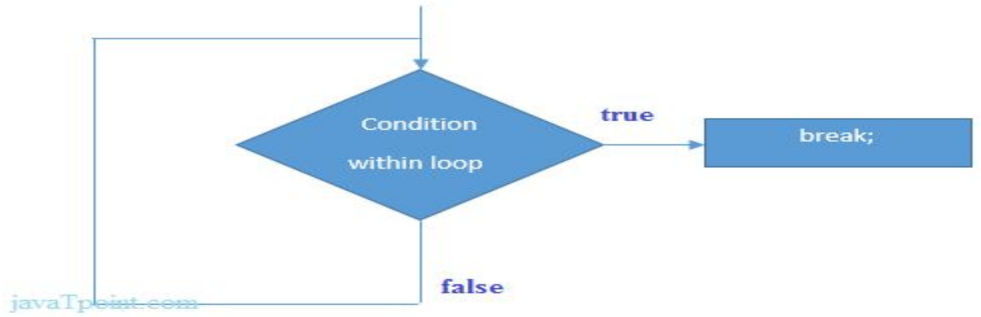


Figure: Flowchart of break statement

Java 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.

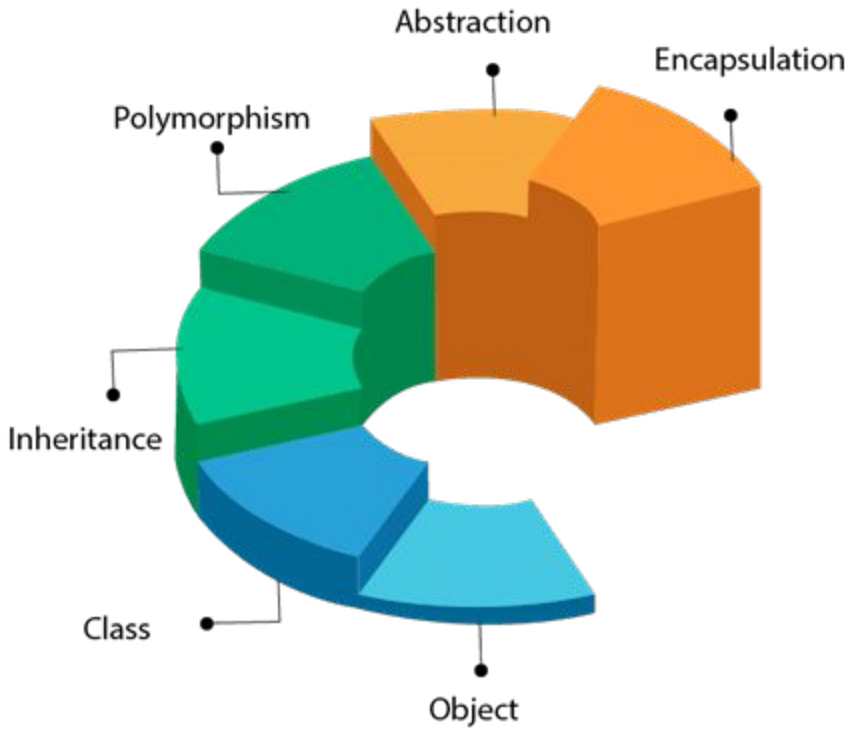
```
//Java Program to demonstrate the use of continue statement

//inside the for loop.

public class ContinueExample {

public static void main(String[] args) {

    //for loop

    for(int i=1;i<=10;i++){

        if(i==5){

            //using continue statement

            continue;//it will skip the rest statement

        }

        System.out.println(i);

    }  }

}
```

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

OOPs (Object-Oriented Programming System)



Objects in Java

Let us now look deep into what are objects. If we consider the real-world, we can find many objects around us, cars, dogs, humans, etc. All these objects have a state and a behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging the tail, running.

If you compare the software object with a real-world object, they have very similar characteristics.

Software objects also have a state and a behavior. A software object's state is stored in fields and behavior is shown via methods.

Class

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

1) OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.

More on Object

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

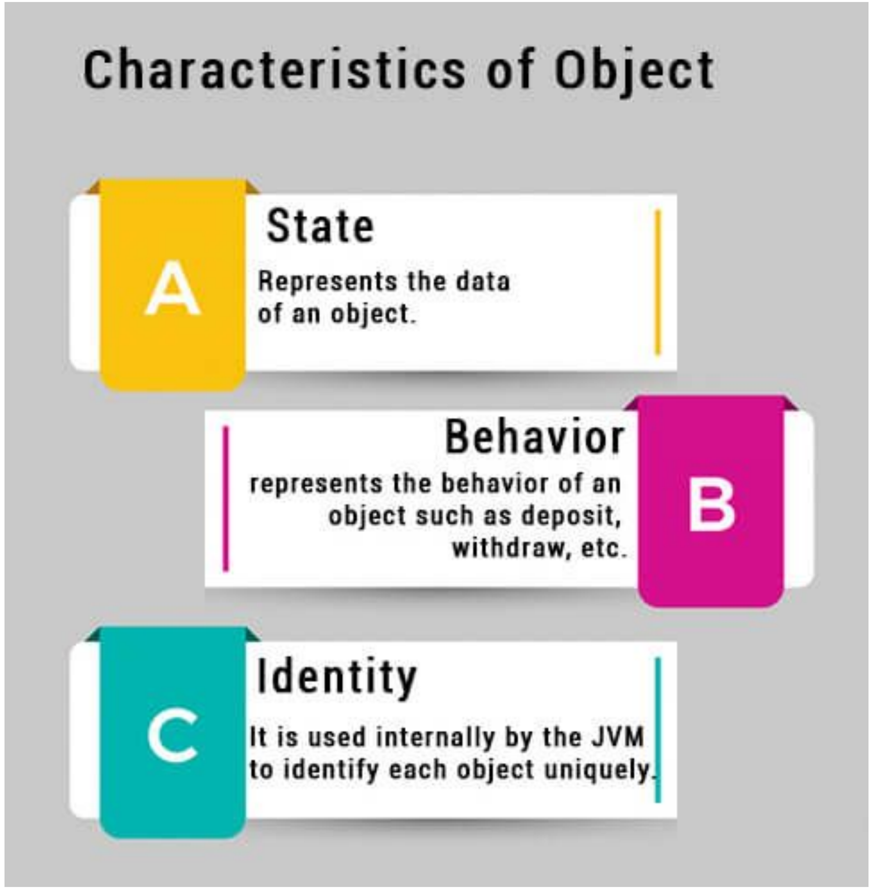
- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.


Characteristics of Object
A
State
Represents the data of an object.
Behavior
represents the behavior of an object such as deposit, withdraw, etc.
B
C
Identity
It is used internally by the JVM to identify each object uniquely.

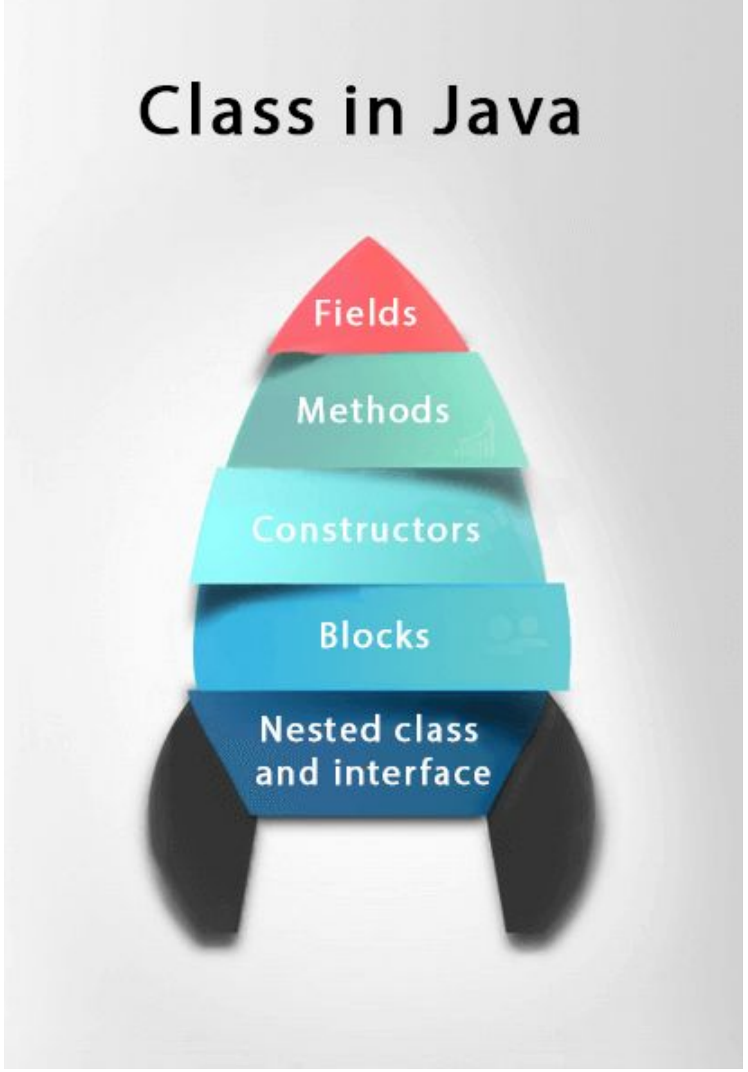
class

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

```
class <class_name>{
    field;
    method;
}
```



Object and Class Example: main within the class

```
//Java Program to illustrate how to define a class and fields
//Defining a Student class.
class Student{
 //defining fields
 int id;//field or data member or instance variable
 String name;
 //creating main method inside the Student class
 public static void main(String args[]){
  //Creating an object or instance
  Student s1=new Student();//creating an object of Student
  //Printing values of the object
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Object and Class Example: main outside the class

```
//Java Program to demonstrate having the main method in
//another class
//Creating Student class.
class Student{
 int id;
 String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
 public static void main(String args[]){
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
 }
}
```

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

3 Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

Initialization through reference

```
class Student{
 int id;
 String name;
}
class TestStudent2{
 public static void main(String args[]){
  Student s1=new Student();
  s1.id=101;
  s1.name="Sonoo";
  System.out.println(s1.id+" "+s1.name);//printing members with a white space
 }
}
```

Initialization through method

```
class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Initialization through a constructor

```
class Employee{
    int id;
    String name;
    float salary;
    void insert(int i, String n, float s) {
        id=i;
        name=n;
        salary=s;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}
public class TestEmployee {
public static void main(String[] args) {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}
}
```

Constructors

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called. It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It contro

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

```
//Java Program to create and call a default constructor

class Bike1{

//creating a default constructor

Bike1(){System.out.println("Bike is created");}

//main method

public static void main(String args[]){

//calling a default constructor

Bike1 b=new Bike1();

}

}
```

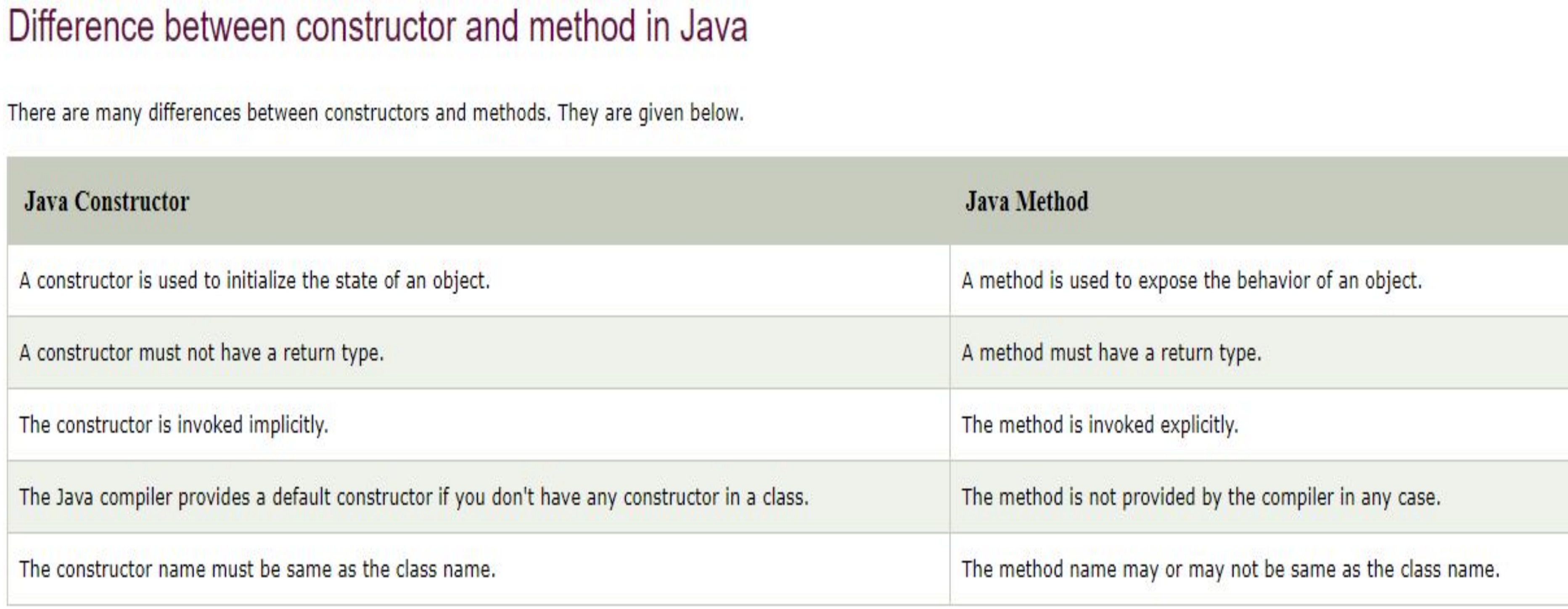
Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

```java
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
  int id;
  String name;
  //creating a parameterized constructor
  Student4(int i,String n){
  id = i;
  name = n;
  }
  //method to display the values
  void display(){System.out.println(id+" "+name);}

  public static void main(String args[]){
  //creating objects and passing values
  Student4 s1 = new Student4(111,"Karan");
  Student4 s2 = new Student4(222,"Aryan");
  //calling method to display the values of object
  s1.display();
  s2.display();
 }
}
```



Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.

Difference between constructor and method in Java

1 A method is used to expose the behavior of an object.

1 A constructor is used to initialize the state of an object.

2 A method must have a return type.

2 A constructor must not have a return type.

3 The method is invoked explicitly.

3 The constructor is invoked implicitly.

4 The method is not provided by the compiler in any case.

4 The Java compiler provides a default constructor if you don't have any constructor in a class.

5 The method name may or may not be same as class name.

5 The constructor name must be same as the class name.

Home Work

Write a Java program that contains a method to get a number from the user and print whether it is positive or negative

Write a Java program that contains a method to Take three numbers from the user and print the greatest number

Write a Java program that keeps a number from the user and generates an integer between 1 and 7 and displays the name of the weekday

```
•   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;
•   }
•
```