# JAVA OOP CHEAT SHEET

Learn **JAVA** from experts at https://www.edureka.co

## Object Oriented Programming in Java

Java is an **Object Oriented Programming** language that produces software for multiple platforms. An object-based application in Java is concerned with declaring classes, creating objects from them and interacting between these objects.

### Java Class

```
class Test {
        // class body
        member variables
        methods
    }
```

### Java Object

```
//Declaring and Initializing an object
Test t = new Test();
```

## Constructors

### Default Constructor

```
class Test{
 /* Added by the Java Compiler at the Run Time
 public Test(){
 }
 */
 public static void main(String args[]) {
   Test testObj = new Test();
    }
}
```

### Parameterized Constructor

```
public class Test {
    int appId;
    String appName;
//parameterized constructor with two parameters
Test(int id, String name){
    this.appId = id;
    this.appName = name;
}
void info(){
    System.out.println("Id: "+appId+" Name: "+appName);
}

public static void main(String args[]){
    Test obj1 = new Test(11001,"Facebook");
    Test obj2 = new Test(23003,"Instagram");
    obj1.info();
    obj2.info();
  }
}
```

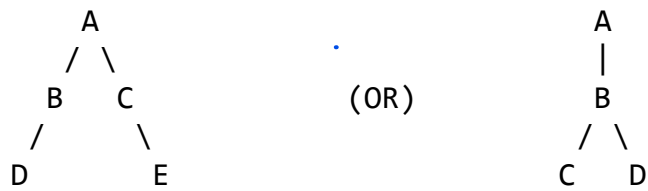### JAVA CERTIFICATION TRAINING

## Inheritance

### Single Inheritance

```
Class A {
    //your parent class code
}
Class B extends A {
    //your child class code
}
```

### Multi Level Inheritance

```
Class A {
    //your parent class code
}
Class B extends A {
   //your code
}
Class C extends B {
    //your code
}
```

### Hybrid Inheritance

```
            A                        A
          /  \                       |
         B    C        (OR)          B
        /  \                        /  \
       D    E                      C    D
```

### Hierarchical Inheritance

```
Class A {
    //your parent class code
}

Class B extends A {
    //your child class code
}

Class C extends A {
    //your child class code
}
```

### Multiple Inheritance

```
Class A {
    //your parent class code
}
Class B {
    //your parent class code
}
Class C extends A,B {      C(-,B)
    //your child class code
}
```

## Polymorphism

### Compile Time Polymorphism

```
class Calculator {
    static int add(int a, int b){
    return a+b;
    }
static double add( double a, double b){
    return a+b;
    }
public static void main(String args[]){
    System.out.println(Calculator.add(123,17));
    System.out.println(Calculator.add(18.3,1.9));
  }
}
```

### Run Time Polymorphism

```
public class Mobile{
void sms(){System.out.println("Mobile class");}
}

//Extending the Mobile class
public class OnePlus extends Mobile{
//Overriding sms() of Mobile class
 void sms(){
   System.out.println(" OnePlus class");
   }

public static void main(String[] args) {
   OnePlus smsObj= new OnePlus();
   smsObj.sms();
   }
}
```

## Abstraction

### Abstract Class

```
public abstract class MyAbstractClass
 {
    public abstract void abstractMethod();
    public void display(){
     System.out.println("Concrete method");
  }
}
```

### Interface

```
//Creating an Interface
public interface Bike { public void start(); }
//Creating classes to implement Bike interface
class Honda implements Bike{
   public void start() {
     System.out.println("Honda Bike");
} }
class Apache implements Bike{
   public void start() {
     System.out.println("Apache Bike");
} }
class Rider{
   public static void main(String args[]){
     Bike b1=new Honda();
     b1.start();
     Bike b2=new Apache();
     b2.start();
} }
```

## Encapsulation

```
public class Artist {
    private String name;
    //getter method
    public String getName() { return name; }
    //setter method
    public void setName(String name) { this.name = name; }
    }
public class Show{
    public static void main(String[] args){
      //creating instance of the encapsulated class
      Artist s=new Artist();
      //setting value in the name member
      s.setName("BTS");
      //getting value of the name member
      System.out.println(s.getName());
  }
}
```

## Modifiers in Java

### Access Modifiers

| Scope | Private | Default | Protected | Public |
|---|---|---|---|---|
| Same class | Yes | Yes | Yes | Yes |
| Same package subclass | No | Yes | Yes | Yes |
| Same package non-subclass | No | Yes | Yes | Yes |
| Different package subclass | No | No | Yes | Yes |
| Different package non-subclass | No | No | No | Yes |

### Non - Access Modifiers

| Type | Scope |
|---|---|
| Static | Makes the attribute dependent on a class |
| Final | Once defined, doesn't allow any changes |
| Abstract | Makes the classes and methods abstract |
| Synchronized | Used to synchronize the threads |