OOPs concepts in Java Part 1

## **What is OOPs?**

Basically, oop’s is a programing design some sort of rule and regulation or we can say a structure of a program. Using this oops concept in our program we can write more robust and scalable code.in oops basically we play with class and objects

Opp’s contain the flowing characteristics

* Object
* Class
* Abstraction
* Inheritance
* Polymorphism
* Encapsulation

## **What are Objects?**

In java objects are real life entities.in java we can consider everything as object but object must have state and behaviors (state are represent as variables and behaviors are represent as methods). object is called instance of the class also

We can treat animal as an object which have states like color,breed,name and also have behavior like canEat(), runFast() etc.

We use *new* keyword to create an object from a class.

## **What are Classes?**

Classes are blue print of an object. A class will say how an object will look like. What will be the characteristic of the object?

When we create class, it doesn’t contain any memory but when we create an object of that class the that object occupy memory in heap.

Class keyword used to create a class and first letter of the class should ne capital and class name and file name should be same.

The variable and data and the methods define inside the class called instance variable and they have scope throughout the class.

## **What is Abstraction**

Abstraction or we can say data hiding.it hide the internal implementation details and show only functionality part. Abstraction inly discloses the top-level information and only necessary information and hide all the internal details.

Ex- to drive a car we don’t need mechanical degree. We just need to know how to handle the steering wheel that all. We don’t care all the complex mechanism of the car. This is abstraction. It only displays the relevant part which is steering wheel and hide all the complex under the hood.

There are two way we can achieve abstraction in java

1. Using Abstract class
2. Using interface

We can declare a class as an abstract class using abstract keyword. An abstract call can contain an abstract method or it may contain concrete method. We cannot create object from an abstract class but can be subclass. Another class must extend that abstract class and also must implement all the abstract methos. If not, so then next child class must do that and so on.

An abstract methos don’t have body don’t have any implementation. But it can have return type and argument or argument list. We use abstract method when two or more child class do the same task but different way.so that they can perform their own implementations.

We use the abstraction because it forces the child class or below class to do something. It is like a contract with base class if you extend me then you have to implement this thing.

abstract class Animal

{

public abstract void sound();

}

public class dog extends Animal

{

public void sound()

{

System.out.println("bark"):

}

}

public static void main(String[] args)

{

Dog d=new Dog();

d.sound();

Animal a=**new** Dog(); //called upcasting

a.sound();

}

Using interface, we can achieve 100% abstraction.

Interface just like a class but not a full class. interface can have variables and methods just like class but all the method inside interface by default abstract and all the variable inside interface by default public static and final. And of course, we can’t create an object form an interface.

**public** **interface** Bank

{

**int** ***min***=5;

**void** rate\_of\_intrest(**int** a, **int** b, **int** c);

}

**public** **class** SBI **implements** Bank

{

**public** **void** rate\_of\_intrest(**int** p, **int** t, **int** r)

{

**int** ptr=(p\*t\*r)/100;

System.***out***.println(ptr);

}

}

**public** **class** Axis **implements** Bank

{

**public** **void** rate\_of\_intrest(**int** p, **int** t, **int** r)

{

**int** ptr=(p\*t\*r)\*2/100;

System.***out***.println(ptr);

}

}

**public** **class** Main

{

**public** **static** **void** main(String[] args)

{

SBI sb=**new** SBI();

sb.rate\_of\_intrest(100, 2, 2);

System.***out***.println(sb.***min***);

Bank b=**new** Axis();

b.rate\_of\_intrest(100, 2, 2);

//accessing the static variable from interface

System.***out***.println(Axis.***min***);

}

}

## **What is Inheritance**

Inheritance means accessing the characteristic from parent class to child class

* **Single level**

Here a child class accessing the property from a parent class

Class a {

…

}

Class b **extends** **class** a {

…

}

* **Multilevel**

Here a parent class can have more than one child class

Class a {

….

}

Class b **extends** **class** a {

….

}

Class c **extends** **class** b {

…

}

* **Hierarchical level**

Here one parent class can be inherited by many child class

Class a {

…

}

Class b **extends** **class** a {

..

}

Class c **extends** **class** a {

..

}

* **Hybrid inheritance**

This is the combination of multiple and multilevel

Java don’t support multiple inheritance which means class C cannot inherit class A and class B at the same time. But it can be done by runnable interface.

## **What is Polymorphism**

Polymorphism refer to many forms. For example, a woman at the same time can consider as housewife, a teacher, a mother and a developer.

In java we can achieve polymorphism in two ways.

1. run time polymorphism or dynamic polymorphism
2. compile time polymorphism or static polymorphism

Run time polymorphism can be done by method overriding. Where same method shares by the parent and as well as child class. Using this child class can give its own implementation to the method which is already present in the parent class. This can be useful when several child classes inherit property from one parent class and so child can give their own implementation.

It is only happened when we have IS-A relation. Here both the method must have same name same sets of parameters. And at the run time JVM decide which method should called. Here we can use the super keyword to call parent version of the method

Using method overloading we can achieve compile time polymorphism. Where same set of method present in a class only the arguments are different.

## **What is Encapsulation**

Encapsulation is one of the concepts in OOPs concepts; it is the process that binds together the data and code into a single unit and keeps both from being safe from outside interference and misuse

In this process, the data is hidden from other classes and can be accessed only through the current class’s methods. Hence, it is also known as data hiding

Using getter and setters we can achieve encapsulation.

# static keyword

# static keyword we can use with variables, nested class and methods. Static field for class level which means we don’t need object to access them. We can use the class name to access them

#### **Final Method in Java**

# A method that is declared final is called a final method. We cannot override a final method. This means the child class can still call the final method of parent class without any problem, but it cannot override it. This is because the main purpose of making a method final is to stop the modification of the method by the sub-class. Simply this is final sub class can have it but cannot override it or change it.

# finally, keyword

this block of code will execute no matter what happen.

The finally keyword is **used to create a block of code that follows a try block**. A finally block of code always executes, whether or not an exception has occurred. Using a finally block allows you to run any cleanup-type statements that you just wish to execute, despite what happens within the protected code

## **Advantages of OOPs Concept**

* **Re-usability**
* **Data redundancy**
* **Code maintenance**
* **Security**
* **Easy troubleshooting**