Q1. Write in brief about OOPS Concept in java with Examples. (In your own words)

Object Oriented programming is a programming style which is associated with the concepts like class, object, Inheritance, Encapsulation, Abstraction, Polymorphism. Most popular programming languages like Java, C++, C#, Ruby, etc. follow an object-oriented programming.

What is Object Oriented Programming?

Object Oriented programming (OOP) refers to a type of programming in which programmers define the data type of a data structure and the type of operations that can be applied to the data structure.

What are the four basic principles/ building blocks of OOP (object oriented programming)?

The building blocks of object-oriented programming are Inheritance, Encapsulation, Abstraction, and Polymorphism.

Let's understand more about each of them in the following sequence:

1.Inheritance

2.Encapsulation

3.Abstraction

4.Polymorphism

What are the benefits of Object Oriented Programming?

1.Improved productivity during software development

2.Improved software maintainability

3.Faster development sprints

4.Lower cost of development

5.Higher quality software

However, there are a few challenges associated with OOP, namely:

1.Steep learning curve

2.Larger program size

3.Slower program execution

4. Its not a one-size fits all solution

Let's get started with the first Java OOPs concept with Example, i.e. Inheritance.

Object Oriented Programming : Inheritance

In OOP, computer programs are designed in such a way where everything is an object that interact with one another. Inheritance is one such concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes.

Inheritance - object oriented programming

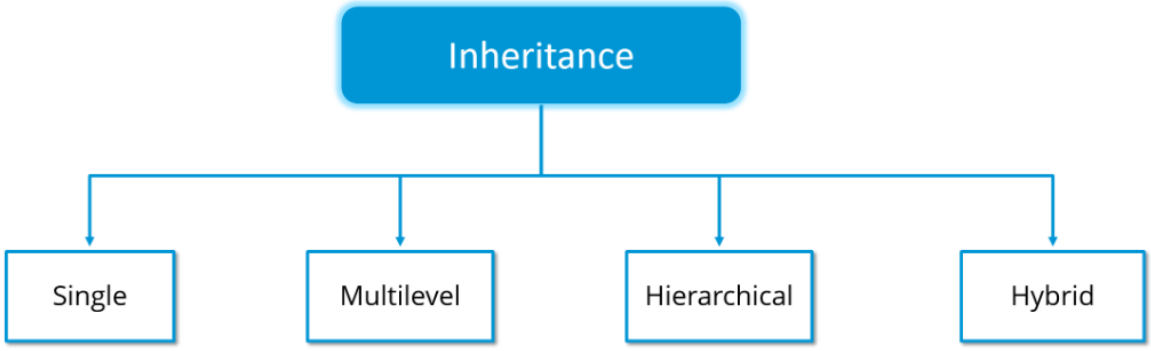
There are two classes:

1. Parent class (Super or Base class)

2. Child class (Subclass or Derived class)

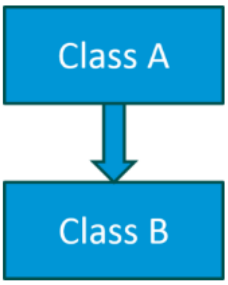
A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

Inheritance is further classified into 4 types:



So let's begin with the first type of inheritance i.e. Single Inheritance:

Single Inheritance:



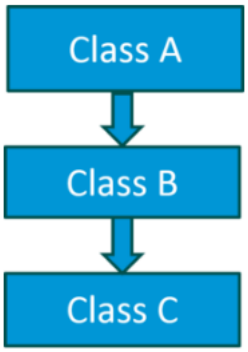
In single inheritance, one class inherits the properties of another. It enables a derived class to inherit the properties and behaviour from a single parent class. This will in turn enable code reusability as well as add new features to the existing code.

Here, Class A is your parent class and Class B is your child class which inherits the properties and behaviour of the parent class.

Let's see the syntax for single inheritance:

```
Class A
{
---
}
Class B extends A {
---
}
```

2. Multilevel Inheritance:



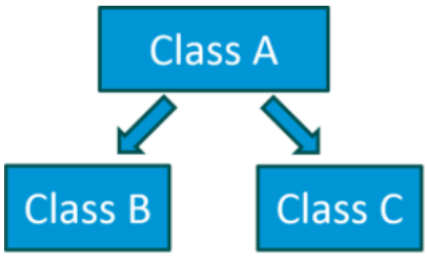
When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.

If we talk about the flowchart, class B inherits the properties and behavior of class A and class C inherits the properties of class B. Here A is the parent class for B and class B is the parent class for C. So in this case class C implicitly inherits the properties and methods of class A along with Class B. That's what is multilevel inheritance.

Let's see the syntax for multilevel inheritance in Java:

```
Class A{
---
}
Class B extends A{
---
}
Class C extends B{
---
}
```

3. Hierarchical Inheritance:



When a class has more than one child classes (sub classes) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as hierarchical.

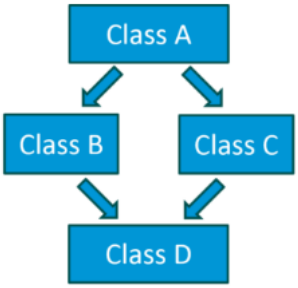
If we talk about the flowchart, Class B and C are the child classes which are inheriting from the parent class i.e Class A.

Let's see the syntax for hierarchical inheritance in Java:

```
Class A{
---
}
Class B extends A{
---
```

}
Class C extends A{

}
4.Hybrid Inheritance:



Hybrid inheritance is a combination of multiple inheritance and multilevel inheritance. Since multiple inheritance is not supported in Java as it leads to ambiguity, so this type of inheritance can only be achieved through the use of the interfaces.

If we talk about the flowchart, class A is a parent class for class B and C, whereas Class B and C are the parent class of D which is the only child class of B and C.

Object Oriented Programming : Encapsulation

Encapsulation is a mechanism where you bind your data and code together as a single unit. It also means to hide your data in order to make it safe from any modification. What does this mean? The best way to understand encapsulation is to look at the example of a medical capsule, where the drug is always safe inside the capsule. Similarly, through encapsulation the methods and variables of a class are well hidden and safe.

We can achieve encapsulation in Java by:

Declaring the variables of a class as private.

Providing public setter and getter methods to modify and view the variables values.

Let us look at the code below to get a better understanding of encapsulation:

```
public class Employee {
 private String name;
 public String getName() {
 return name;
 }
 public void setName(String name) {
 this.name = name;
 }
 public static void main(String[] args) {
```

```
 }

}
```

Let us try to understand the above code. I have created a class Employee which has a private variable name. We have then created a getter and setter methods through which we can get and set the name of an employee. Through these methods, any class which wishes to access the name variable has to do it using these getter and setter methods.

Object Oriented Programming : Abstraction

Abstraction refers to the quality of dealing with ideas rather than events. It basically deals with hiding the details and showing the essential things to the user. If you look at the image here, whenever we get a call, we get an option to either pick it up or just reject it. But in reality, there is a lot of code that runs in the background. So, you don't know the internal processing of how a call is generated, that's the beauty of abstraction. Therefore, abstraction helps to reduce complexity. You can achieve abstraction in two ways:

a) Abstract Class

b) Interface

Abstract class:

Abstract class in Java contains the 'abstract' keyword. Now what does the abstract keyword mean? If a class is declared abstract, it cannot be instantiated, which means you cannot create an object of an abstract class. Also, an abstract class can contain abstract as well as concrete methods.

Note: You can achieve 0-100% abstraction using abstract class.

To use an abstract class, you have to inherit it from another class where you have to provide implementations for the abstract methods there itself, else it will also become an abstract class.

Let's look at the syntax of an abstract class:

```
Abstract class Mobile {   // abstract class mobile
Abstract void run();      // abstract method
```

Interface: Interface in Java is a blueprint of a class or you can say it is a collection of abstract methods and static constants. In an interface, each method is public and abstract but it does not contain any constructor. Along with abstraction, interface also helps to achieve multiple inheritance in Java.

Note: You can achieve 100% abstraction using interfaces.

So an interface basically is a group of related methods with empty bodies. Let us understand interfaces better by taking an example of a 'ParentCar' interface with its related methods.

```
public interface ParentCar
{
public void changeGear( int newValue);
public void speedUp(int increment);
public void applyBrakes(int decrement);
}
public class Audi implements ParentCar {
int speed=0;
```

```
int gear=1;
public void changeGear( int value){
gear=value;
}
public void speedUp( int increment)
{
speed=speed+increment;
}
public void applyBrakes(int decrement)
{
speed=speed-decrement;
}
void printStates(){
System.out.println("speed:"+speed+"gear:"+gear);
}
public static void main(String[] args) {
// TODO Auto-generated method stub
Audi A6= new Audi();
A6.speedUp(50);
A6.printStates();
A6.changeGear(4);
A6.SpeedUp(100);
A6.printStates();
}
}
```

Object Oriented Programming : Polymorphism

Polymorphism means taking many forms, where 'poly' means many and 'morph' means forms. It is the ability of a variable, function or object to take on multiple forms. In other words, polymorphism allows you define one interface or method and have multiple implementations.

Polymorphism in Java is of two types:

1.Run time polymorphism

2.Compile time polymorphism

Run time polymorphism:

In Java, runtime polymorphism refers to a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this, a reference variable is used to call an overridden method of a superclass at run time. Method overriding is an example of run time polymorphism. Let us look the following code to understand how the method overriding works:

```
public Class BowlerClass{
void bowlingMethod()
{
System.out.println(" bowler ");
}
public Class FastPacer{
void bowlingMethod()
{
```

```
System.out.println(" fast bowler ");
}
Public static void main(String[] args)
{
FastPacer obj= new FastPacer();
obj.bowlingMethod();
}
}
```

Compile time polymorphism:

In Java, compile time polymorphism refers to a process in which a call to an overloaded method is resolved at compile time rather than at run time. Method overloading is an example of compile time polymorphism. Method Overloading is a feature that allows a class to have two or more methods having the same name but the arguments passed to the methods are different. Unlike method overriding, arguments can differ in:

Number of parameters passed to a method

Datatype of parameters

Sequence of datatypes when passed to a method.

Let us look at the following code to understand how the method overloading works:

```
class Adder {
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(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}
}
```

Q2.Write simple programs(wherever applicable) for every example given in Answer 2.

Inheritance

Inheritance – Single, Multilevel, Hierarchical, and Multiple
Inheritance is the process by which one class inherits the functions and properties of another class. The main function of inheritance is the reusability of code. Each subclass only has to define its features. The rest of the features can be derived directly from the parent class.

Single Inheritance – Refers to a parent-child relationship where a child class extends the parent class features. Class Y extends Class X.

Multilevel Inheritance – Refers to a parent-child relationship where a child class extends another child's class. Class Y extends Class X. Class Z extends Class Y.

Hierarchical Inheritance – This refers to a parent-child relationship where several child classes extend one class. Class Y extends Class X, and Class Z extends Class X.

Multiple Inheritance – Refers to a parent-child relationship where one child class is extending from two or more parent classes. JAVA does not support this inheritance.

Example Program of Inheritance in Java

```
class Animal
{ void habit()
{ System.out.println("I am nocturnal!! ");
}
}
class Mammal extends Animal
{
void nature()
{
System.out.println("I hang upside down!! ");
}
}
class Bat extends Mammal
{
void hobby()
{
System.out.println("I fly !! ");
}
}
public class Inheritance
{
public static void main(String args[])
{
Bat b = new Bat();
b.habit();
b.nature();
b.hobby();
}
}
```

Encapsulation

Encapsulation is a means of binding data variables and methods together in a class. Only objects of the class can then be allowed to access these entities. This is known as data hiding and helps in the insulation of data.

Example of Encapsulation

```
class Encapsulate
{

   private String Name;
   private int Height;
```

```
   private int Weight;

   public int getHeight()
   {
    return Height;
   }


   public String getName()
   {
    return Name;
   }


   public int getWeight()
   {
     return Weight;
   }


   public void setWeight( int newWeight)
   {
    Weight = newWeight;
   }


   public void setName(String newName)
   {
    Name = newName;
   }

   public void setHeight( int newHeight)
   {
    Height = newHeight;
   }
}

public class TestEncapsulation
{
   public static void main (String[] args)
   {
      Encapsulate obj = new Encapsulate();

      obj.setName("Abi");
      obj.setWeight(70);
      obj.setHeight(178);

      System.out.println("My name: " + obj.getName());
      System.out.println("My height: " + obj.getWeight());
      System.out.println("My weight " + obj.getHeight());
           }
```

```
}
```

Abstraction

Abstraction means showing only the relevant details to the end-user and hiding the irrelevant features that serve as a distraction. For example, during an ATM operation, we only answer a series of questions to process the transaction without any knowledge about what happens in the background between the bank and the ATM.

Example Program of Abstraction in Java

```
abstract class Bike
{
 Bike()
 {
  System.out.println("The Street Bob. ");
 }
 abstract void drive();
 void weight()
 {
  System.out.println("Light on its feet with a hefty : 630 lbs.");
 }
}

class HarleyDavidson extends Bike
 {
  void drive()
  {
   System.out.println("Old-school yet relevant.");
  }
 }

public class Abstraction
 {
  public static void main (String args[])
  {
   Bike obj = new HarleyDavidson();
   obj.drive();
   obj. weight();
  }
 }
```

Polymorphism – Static and Dynamic

It is an object-oriented approach that allows the developer to assign and perform several actions using a single function. For example, “+” can be used for addition as well as string concatenation. Static Polymorphism is based on Method Overloading, and Dynamic Polymorphism is based on Method Overriding.

Example Program of Static Polymorphism with Method Overloading

Method Overloading

```
class CubeArea
{
```

```
        double area(int x)
        {
                return 6 * x * x;
        }
}

class SphereArea
{
        double area(int x)
{
                return 4 * 3.14 * x * x;
        }
}

class CylinderArea
{
        double area(int x, int y)
        {
                 return x * y;
 }
}

public class Overloading
{

 public static void main(String []args)
 {
                CubeArea ca = new CubeArea();
                SphereArea sa = new SphereArea();
                CylinderArea cia = new CylinderArea();

                System.out.println("Surface area of cube = "+ ca.area(1));
                System.out.println("Surface area of sphere= "+ sa.area(2));
                System.out.println("Surface area of cylinder= "+ cia.area(3,4));
          }
}
```

Example Program of Dynamic Polymorphism with Method Overriding

```
class Shape
{
   void draw()
   {
      System.out.println("Your favorite shape");
   }

   void  numberOfSides()
   {
       System.out.println("side = 0");
   }
}
```

```
class Square extends Shape
{
   void draw()
   {
      System.out.println("SQUARE ");
   }

   void numberOfSides()
   {
      System.out.println("side = 4 ");
   }
}

class Pentagon extends Shape
{
   void draw()
   {
      System.out.println("PENTAGON ");
}

   void numberOfSides()
   {
      System.out.println("side= 5");
   }
}

class Hexagon extends Shape
{
   void draw()
   {
      System.out.println("HEXAGON ");
   }

   void numberOfSides()
   {
      System.out.println("side = 6 ");
   }
}

public class Overriding{

   public static void main(String []args){
      Square s = new Square();
      s.draw();
      s.numberOfSides();

      Pentagon p = new Pentagon();
      p.draw();
      p.numberOfSides();
```

```
        Hexagon h = new Hexagon();
        h.draw();
        h.numberOfSides();
    }
}
```

Multiple Choice Questions:

1: Answer (A)
Explanation: Making atleast one member function as pure virtual function is the method to make abstract class.
2. Answer: (A)
Explanation: The instance of an interface can't be created because it acts as an abstract class.
3. Answer: (B)
Explanation: Overloading is determined at compile time. Hence, it is also known as compile time polymorphism.
4. Answer: (A)
Explanation: A default constructor does not require any parameters for object creation that's why sometimes we declare an object without any parameters.
5. Answer: (A)
Explanation: The data members can never be called directly. Dot operator is used to access the members with help of object of class.
6. Answer: C) Class
Explanation:Objects are the variables of the type Class. Once the class has been defined, we can create any number of objects belonging to that class.
7. Answer: (A)
Explanation: A member function can access private data of the class but a non-member function cannot do that.
8. Answer: (B)
Explanation: In Java, fields of classes and objects that do not have an explicit initializer and elements of arrays are automatically initialized with the default value for their type (false for boolean, 0 for all numerical types, null for all reference types). Local variables in Java must be definitely assigned to before they are accessed, or it is a compile error.
9. Answer: (A)
10. Output : **Derived::show() called**
Explanation: In the above program, b is a reference of Base type and refers to an abject of Derived class. In Java, functions are virtual by default. So the run time polymorphism happens and derived fun() is called.
11. Output: Compiler Error
Explanation: Final methods cannot be overridden.
12. Output : **Base::show() called**
13. Output : Test class
Explanation: super keyword is used to invoke the overridden method from a child class explicitly.
14. Output : Compilation error
Explanation: The overriding method must have same signature, which includes, the argument list and the return type.
15. Output: Adding to 100, x = 104
Adding to 0, y = 3 3 3
Explanation: Properties of static are shown in this example. When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.
16. No Output

Explanation:
error-1: missing method body, or declare abstract
public void m1(float f,int i);
error-2: reference to m1 is ambiguous s.m1(20,20);
17.No Output
Explanation:
error: incompatible types: <null> cannot be converted to int
int temp = null;
18: Output:0 0
In Java, a protected member is accessible in all classes of the same package and in inherited classes of other packages. Since Test and Main are in the same package, no access-related problems in the above program. Also, the default constructors initialize integral variables as 0 in Java. That is why we get output as 0 0.
19. Output: Constructor called 10
Constructor called 5
Explanation: First t2 object is instantiated in the main method. As the order of initialization of local variables comes first then the constructor, first the instance variable (t1), in the class Test2 is allocated to the memory. In this line a new Test1 object is created, the constructor is called in class Test1 and 'Constructor called 10' is printed. Next, the constructor of Test2 is called and again a new object of the class Test1 is created and 'Constructor called 5' is printed.
20.Output :7
21. output:
$ javac Dynamic_dispatch.java
$ java Dynamic_dispatch2
Explanation :r is reference of type A, the program assigns a reference of object obj2 to r and uses that reference to call function display() of class B.
22. output:
$ javac inheritance_demo.java
$ java inheritance_demo
2
Explanation :class A & class B both contain display() method, class B inherits class A, when display() method is called by object of class B, display() method of class B is executed rather than that of Class A.
23. output:
$ javac Output.java
$ java Output
1 2
Explanation :Both class A & B have member with same name that is j, member of class B will be called by default if no specifier is used. I contains 1 & j contains 2, printing 1 2.
24. output:
$ javac super_use.java
$ java super_use
1 2
Explanation: Keyword super is used to call constructor of class A by constructor of class B. Constructor of a initializes i & j to 1 & 2 respectively.
25.output:
obj1.a = 4 obj1.b = 3
obj2.a = 4 obj2.b = 3
Explanation:
obj1 and obj2 refer to same memory address.