Lecture - 1

What is OOP?

Part-1

Object Oriented Programming

What is object-oriented programming?

1. It is one of the most popular programming paradigm.

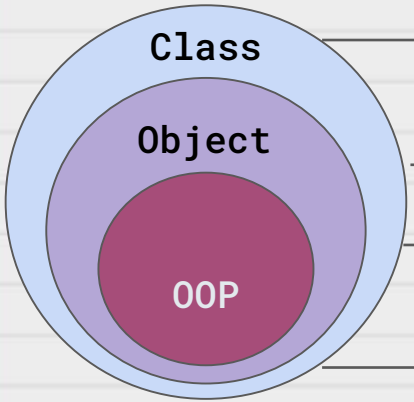
1. In simple words, it is a programming pattern that resolves around objects or entities.

1. Object-oriented programming teaches us how to build software by closely imitating the real world objects or entities.

Popular programming approaches

- Procedure Oriented Programming
- Object Oriented Programming
- Functional Programming
- Aspect Oriented Programming

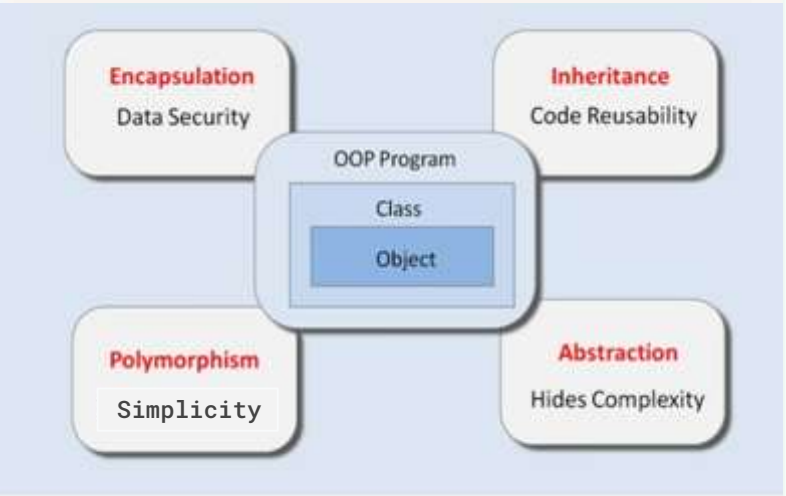
Advantages of object-oriented Programming



- Better Representation of Real World Scenario
- Enhanced Security
- Code Reusability & Easy Code Modification
- Better Software Maintenance

Key Concepts Of Object-Oriented Programming

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



Lecture - 2

Class & Object

Part-2

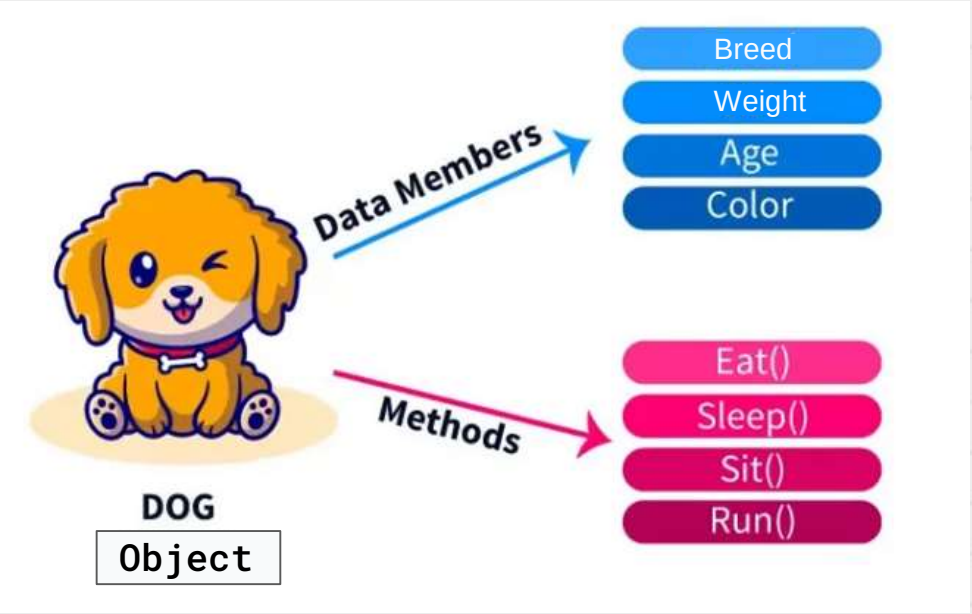
Class & Objects

What is an Object ?

1. In programming any real world entity is an Object.

Components of an Object ?

1. Every Object has two main components :- Property and Behaviours



Question -

Are you an Object ?

Yes,we humans are objects because:

We have attributes as name, height, age etc.

We also can show behaviors like walking, talking, running, eating etc.

Question -

Is your phone an Object ?

Yes, it is an object because:

It has attributes like model, price, color, weight etc.

It can also perform actions/behaviors like calling, recording, photo clicking etc.

How to create an object?

1. Now to create/represent objects we first have to write all their attributes and methods under a single group.

1. This group is called a class

What is Class ?

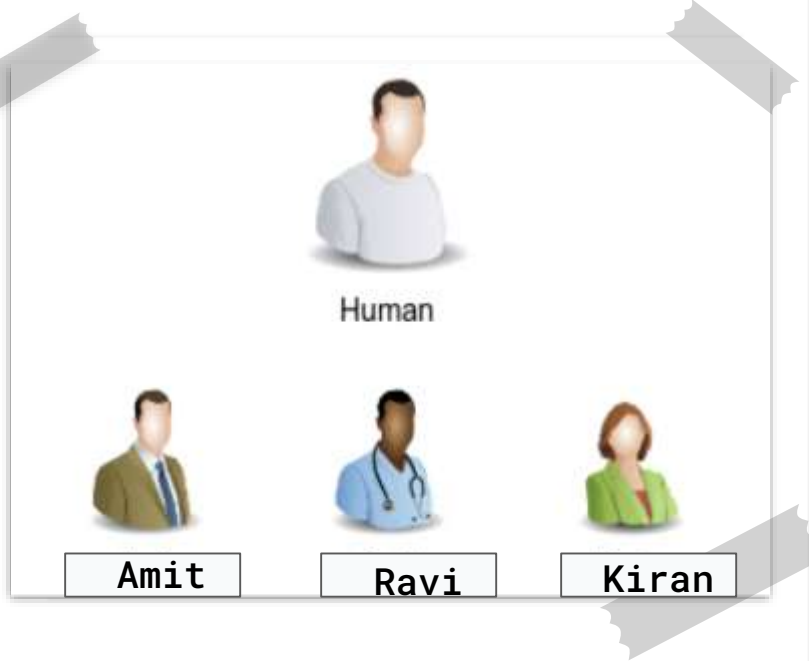
1. A class is used to specify the basic structure of an object.

1. It contains
    a. Data members/Attributes
    b. Methods/Behaviour

Example -

Each person collectively comes under a class called Human.

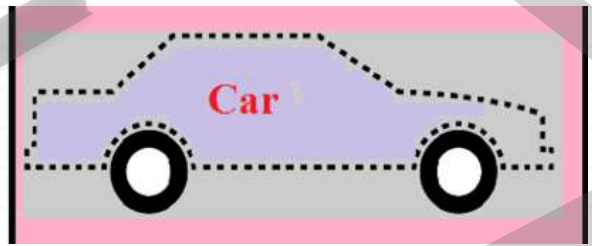
So we belong to the class Human

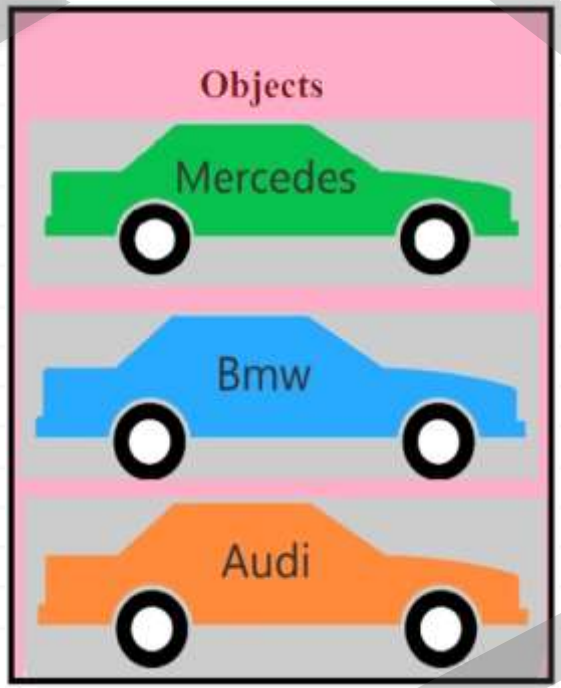


Example -

Each Type of car collectively comes under a class called Car .

Class





Syntax of Object Oriented Programming

```
<access modifier> class <class Name>
{
        // Declaration of Data Types
        <access modifier> <data_Type> <variable_Name> = value;
        ..............    .........   ............   = .....;
        ..............    .........   ............   = .....;

        // Declaration of Methods
    <access modifier> <return_Type> <method_Name>( <argument_dataType> argument )
    {
        // Method Body
    }
}
```

Sample code of Object Oriented programming

Entity Class

```
class Student
{
        int roll;
        String name;
        double per;

}
```

Driver Class

```
class UseStudent
{
        public static void main(String [] args)
    {
        Student s;
        s=new Student();
        S.roll = 101;
        S.name = “Amit”;
        S.per = 82.9;
        System.out.println(“Roll:” + s.roll);
        System.out.println(“Name:” + s.name);
        System.out.println(“Per:” + s.per);
    }
}
```

Lecture - 3

Encapsulation

Part-3

Object Oriented Programming

What Is Encapsulation ?

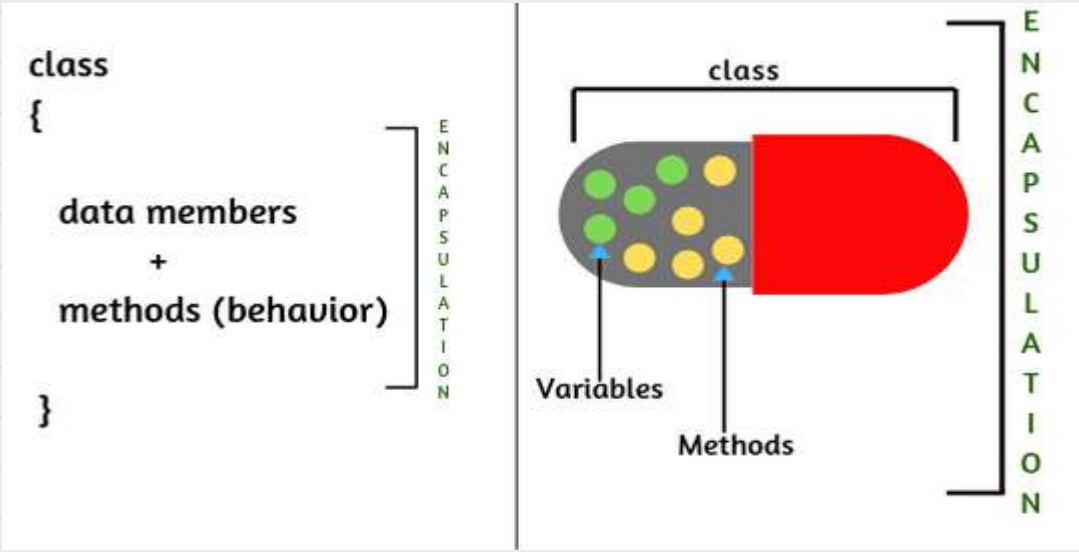
1. Encapsulation is the process of binding or wrapping the data and the codes that operates on the data into a single entity

1. Another way to think about encapsulation is, that it is a protective shield that prevents the data from being accessed by the code outside this shield.

According To Encapsulation. . .

- **Data** :- Data members must be kept **private**.

- **Methods** :- Methods only can access these data members and these methods can be made **public**.



Real world example of Encapsulation

- Suppose you have an account in the bank. Is it possible for anyone obtain your account details like your account balance, transactions done etc.? Obviously No.

- This means that your bank details are private, so that anyone cannot see your account details.

- Only you will have access to your account details and that too using method defined inside that class and this method will ask your account id and password for authentication.

How to achieve or implement Encapsulation in Java

- There are two steps required to achieve or implement encapsulation in Java program.

- Declaring the instance variable of the class as private so that it cannot be accessed directly by anyone from outside the class.

- Provide the public setter and getter methods in the class to set/modify the values of the variable/fields.

Entity Class (Without Encapsulation)

```
class Student {

    int roll;
    String name;
    double per;

}
```

Driver Class

```
class UseStudent
{
    public static void main(String [] args)
    {
        Student s;
        s=new Student();
        s.roll = 101;
        s.name = "Amit";
        s.per = 82.9;
        System.out.println("Roll:" + s.roll);
        System.out.println("Name:" + s.name);
        System.out.println("Per:"  + s.per);
    }
}
```

Correction In Previous Code

```
class Student{

    private int roll;
    private String name;
    private double per;


public void setData(int r, String n, double p){
     roll=r;
     name=n;
     per=p;
}
```

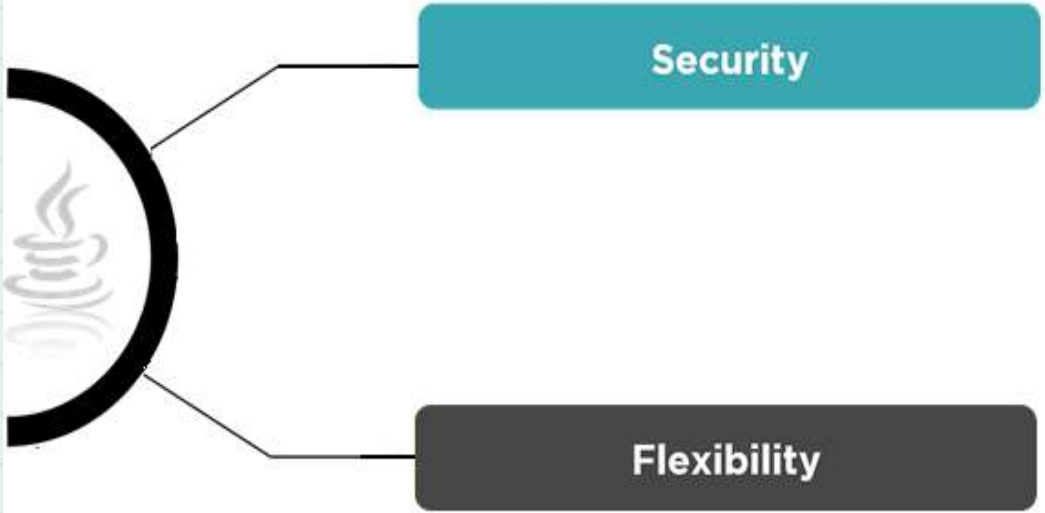
Correction In Previous Code

```
public void showData( ) {

    System.out.println("Roll no is "   +roll);
    System.out.println("Name is "      +name);
    System.out.println("Percentage is "+per);
  }

}
```

Correction In Previous Code

```
class UseStudent
{
    public static void main(String [] args)
    {
        Student s;
        s=new Student();
        s.setData(101, "Amit", 82.9);
        s.showData();
    }
}
```

Advantages Of Encapsulation



Lecture - 4

Abstraction

Part-4

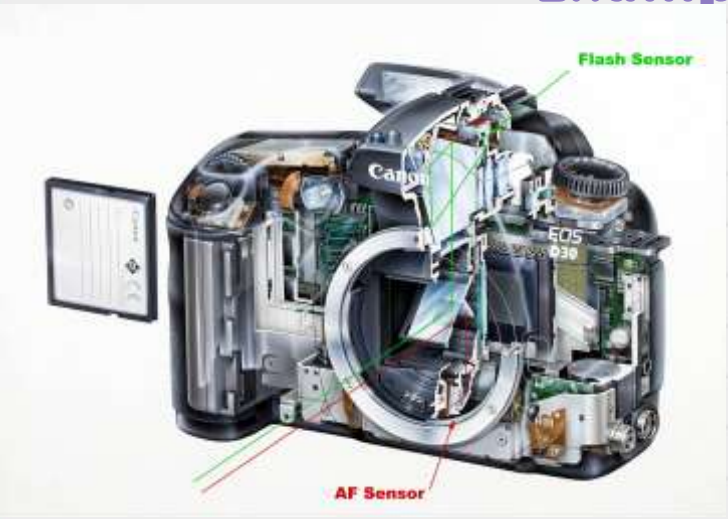
Object Oriented Programming

What Is Abstraction ?

1. **Abstraction** is one of the key concepts of object-oriented programming that **“shows”** only essential attributes and **“hides”** unnecessary information.

2. The main purpose of **abstraction** is to make the interaction with the application/product simple.

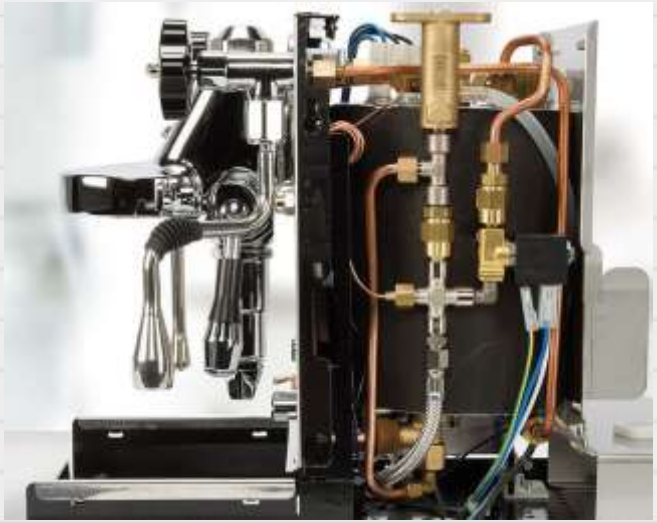
Let's Understand Abstraction With Example



Without Abstraction

Abstraction

Another example to Understand Abstraction



Without Abstraction

Abstraction

Let's Understand Abstraction With Code

```
public class Car {
    private void moveBreakPads() {
    }
    private void changePistonSpeed(){
    }
    private void createSpark(){
    }

    public void turnOnCar(){
        createSpark();
    }
    public void accelerate(){
        changePistonSpeed();
    }
    public void brake(){
        moveBreakPads();
    }
}
```

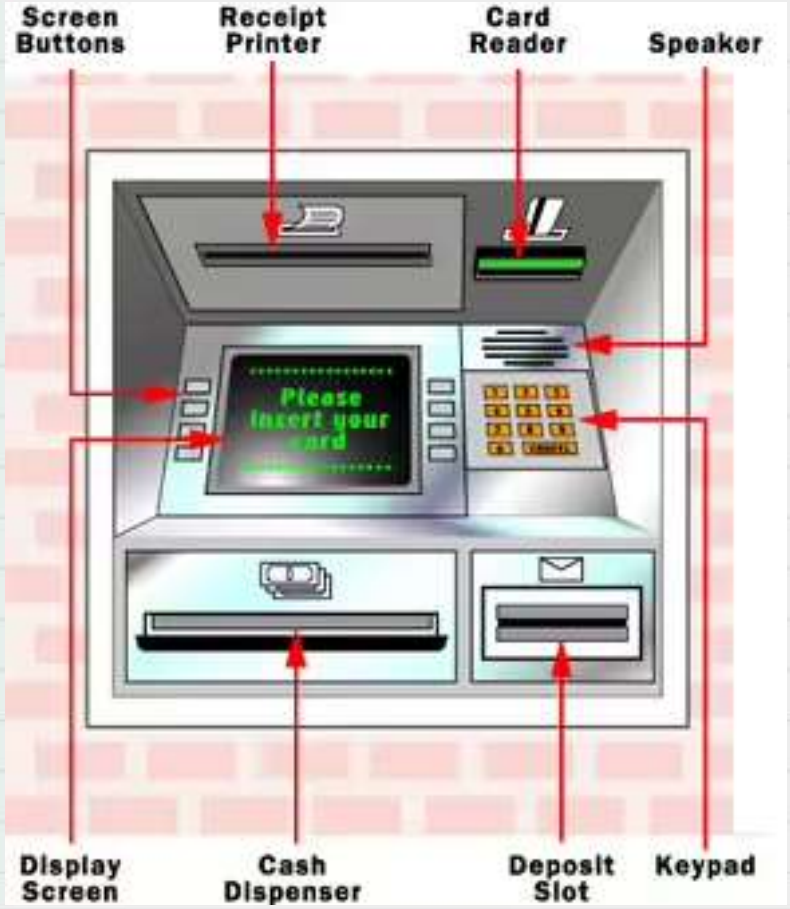
Let's Understand Abstraction With Code

```
//Driver Class

public class UseCar {
    public static void main(String[] args) {
        Car C=new Car(); //Create Object

        C.turnOnCar();
                C.accelerate();
         C.brake();

    }
}
```

Difference Between

Abstraction And Encapsulation

Abstraction	Encapsulation
Abstraction hides the implementation details and shows only the functionality to the user	Encapsulation binds and wraps the data and methods together into a single unit
Hides the implementation details to reduce the code complexity	Hides data for the purpose of data protection
Abstraction can be achieved by using abstract class and interfaces	It can be achieved by making data members private and accessing them through public methods



Abstraction Vs Encapsulation

Lecture - 5

Constructor

Part-5

Object Oriented Programming

What Is Constructor?

A constructor is a special method of a class in object-oriented programming that initializes a newly created object of that class.

Constructors are methods which...

1. Have the same name as that of the class.

1. Don't have any return type.

1. Automatically called as soon as object is created

What is Default Constructor

- Default constructor is automatically inserted by java compiler, if we have not created any constructor explicitly .

```
class Account {
   private int accid;
   private String name;
   private double balance;
}
```

Our Code

```
class Account {
   private int accid;
   private String name;
   private double balance;

   public Account() {
   }
}
```

Compiler's Code

Entity Class

```
class Account {
    private int accid;
    private String name;
    private double balance;

    public Account() {
        accid=101;
        name="AMIT"
        balance=50000.0;
    }

    public void show() {
        System.out.println(accid);
        System.out.println(name);

System.out.println(balance);
    }
}
```

Non Parameterized Constructor

Driver Class

```
class CreateAccount

{

   public static void main(String [] args)

{

        Account A;

         A=new Account();

         A.show();

    }

}
```

Entity Class

```
class Account
{
    private int accid;
    private String name;
    private double balance;
    public Account(int id, String s, double b) {
        accid=id;
        name=n;
        balance=b;
    }
    public void show() {
        System.out.println(accid);
        System.out.println(name);
        System.out.println(balance);
    }
}
```

Parametrized
Constructor

Driver Class

```
class CreateAccount

{

    public static void main(String [] args)

    {

        Account A;

        A=new Account(101,”Ravi”,56000);

        A.show();

    }

}
```

Lecture - 6

Getter & Setter

Part-6

Object Oriented Programming

What is Getter and Setter

1. Getter and Setter are methods defined to work upon instance variables of the class.

1. For each instance variable, a getter method returns its value

1. Setter method sets or updates instance variable value.

Let's Understand With Example

Entity Class

```
class Vehicle {
   private String color;


   public String getColor() {
       return color;
   }


   public void setColor(String c) {
       this.color = c;
   }
}
```

Driver Class

```
class UseVehicle {

   public static void main(String[] args) {

       Vehicle v1;

       v1=new Vehicle();

       v1.setColor("Red");

       System.out.println(v1.getColor());
   }
}
```

Advantages Of Getter and Setter

1. It helps us achieve encapsulation

1. Better control over setting the value of the property correctly

1. Achieve immutability by declaring the fields as private and using only getters

Lecture - 7

Method Overloading

Part-7

Object Oriented Programming

What is Method Overloading ?

1. Overloading is a concept of OOPS in which we can create multiple versions of the same entity in the same scope.

1. So , if a class has multiple methods having same name but different parameters, it is known as Method Overloading.

Benefits of using Method Overloading. . .

1. Method overloading minimises the complexity of the code.

1. It increases the readability of the program.

How to do Method Overloading?

In Java, we do method overloading in three ways:

1. By changing data types of parameters.
2. By changing the number of parameters.
3. By changing order of parameters.

By changing data types.

Entity Class

```
class Addition{
    public int add(int a, int b){
        return a+b;
    }
    public double add(double a, double b){
        return a+b;
    }

}
```

Driver Class

```
class UseAddition{
    public static void main(String[] args){
        Addition obj;
        obj=new Addition();
        System.out.println(obj.add(2,4));

System.out.println(obj.add(22.6,28.7));
    }
}
```

By changing the number of parameters.

Entity Class

```
class Addition{
   public int add(int a, int b){
       return a+b;
   }
   public  int add(int a,int b,int c){
       return a+b+c;
    }
}
```

Driver Class

```
class UseAddition{
   public static void main(String[] args){
       Addition obj;
       obj=new Addition();
       System.out.println(obj.add(2,11));
       System.out.println(obj.add(6,4,12));
   }
}
```

By Changing Order Of Paramaters

Entity Class

```
class Addition{
   private int a;
   private String b;

   public String add(int a, String b){
       return a+b;
   }
   public String  add(String a, int b){
       return a+b;
   }

}
```

Driver Class

```
class UseAddition{
   public static void main(String[] args){
    Addition A;
    A=new Addition();
    System.out.println(A.add(2,"Ravi"));

System.out.println(A.add("Ajay",28));
   }
}
```

Lecture - 10

this Keyword

Part-10

Object Oriented Programming

What is this keyword ?

- “this” is a special object reference, which is also a keyword.
- “this” is automatically created by Java in a method’s argument, as soon we call that method.
- “this” stores the memory address of the CURRENT OBJECT (the object on which the method has been called)

Benefits Of Using “this”. . .

➔ By using “this” we can resolve the overlapping of instance variable of a class done by the local variables of a method with the same name.

➔ 2. By using “this” we can perform CONSTRUCTOR CHAINING

Lecture -11

Inheritance

Part-11

Object Oriented Programming

What Is Inheritance?

➔ Inheritance is a technique using which we can acquire the features of an existing class/object in a newly created class/object

Let's Understand With An Example

```
class Fruit
{
        //data
        ...
        ..
        .
        //method
}
class Orange
{
        ..
}
```

As per OOP , the class Orange must inherit the class Fruit so that all the features (data & methods) of the Fruit class get inherited in Orange class also

Benefits of Inheritance. . .

➔ Reusability: The child class programmer is not required to rewrite those methods or data again which have already been designed by the parent class programmer

➔ Maintainability: Easy to incorporate changes in the code

Terminologies Used In Inheritance

1. The class which gets inherited is knows by 3 names

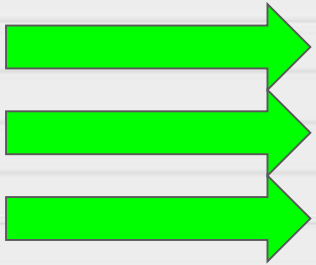
a. Parent class

b. Base class

C++

c. Super class

Java/Python



OOP

Terminologies Used In Inheritance

2. The class which inherits is known by 3 Names

a. Child class

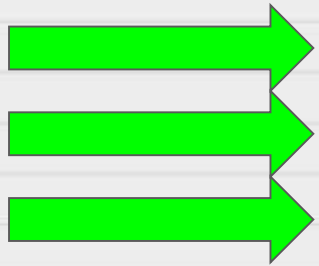
b. Derived class

C++

c. Sub class

Java/Python

OOP

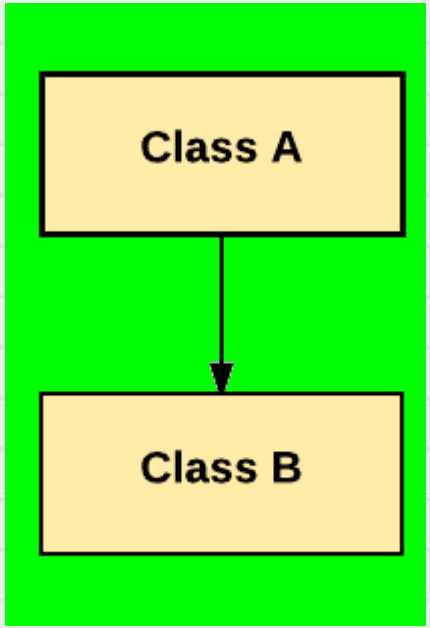


Types Of Inheritance

- Single
- Multilevel
- Multiple
- Hierarchical
- Hybrid
- Note: Java does not support multiple inheritance and thus it also does not support hybrid inheritance

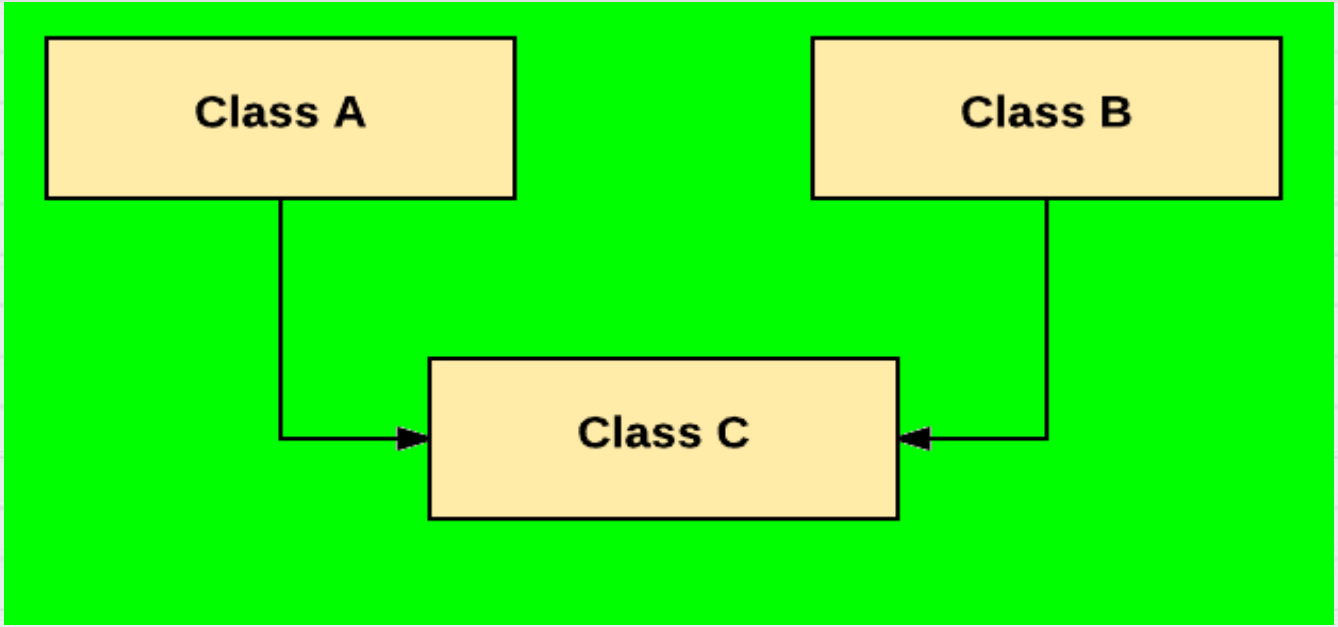
Single Inheritance:

In Single Inheritance one class extends another class (one class only).



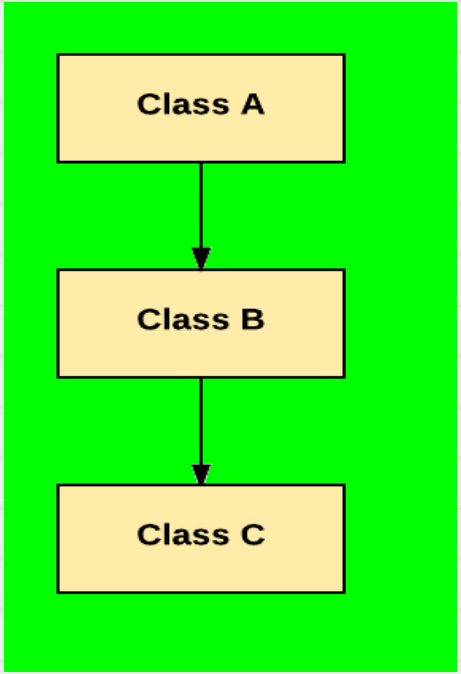
Multiple Inheritance:

Multiple Inheritance is one of the inheritance in OOP where one class extends more than one class. Java does not support multiple inheritance.



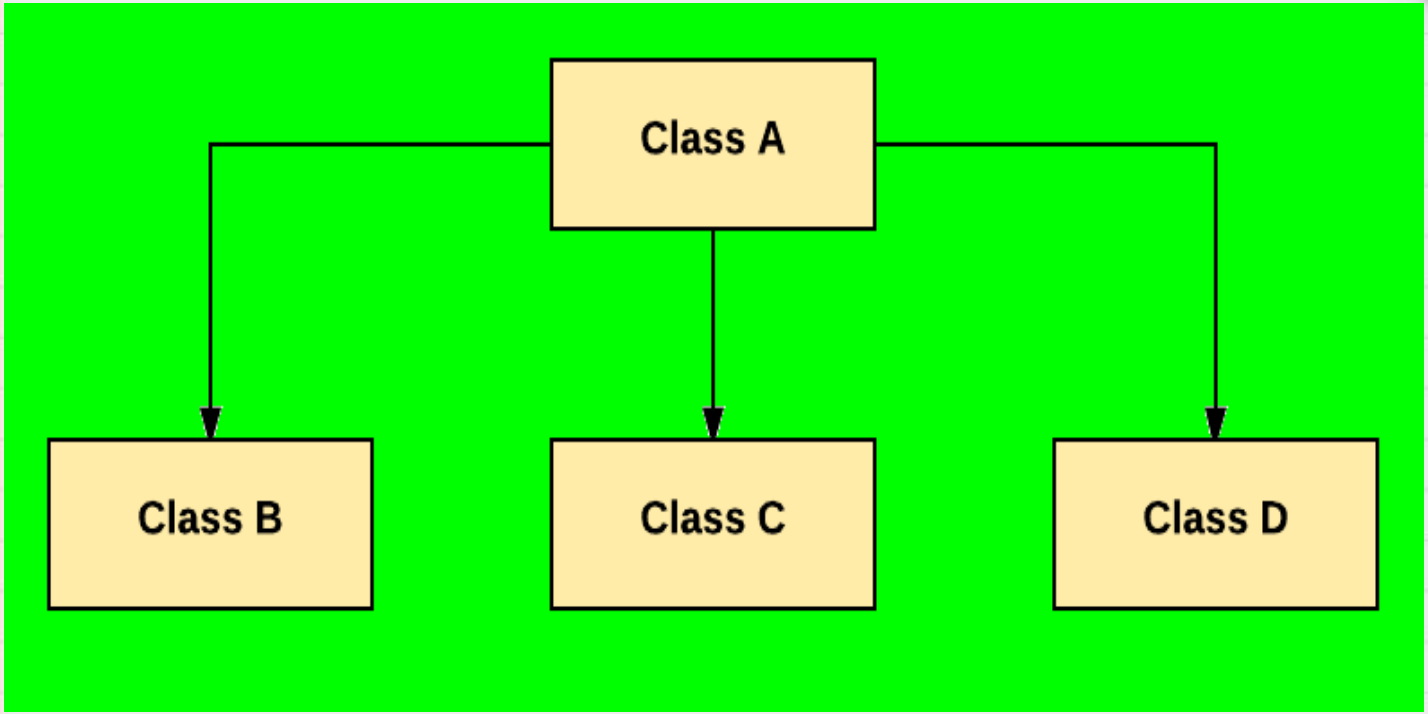
Multilevel Inheritance:

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.



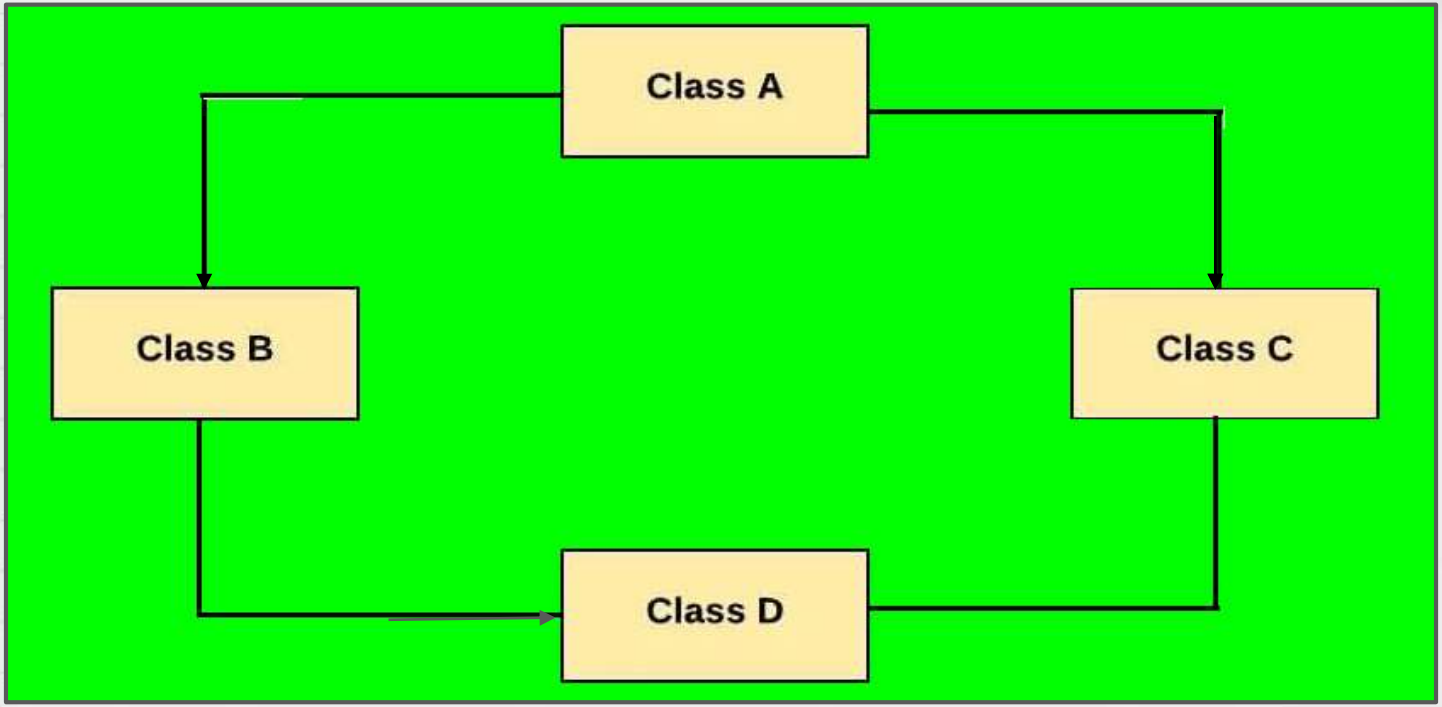
Hierarchical Inheritance:

In Hierarchical Inheritance, one class is inherited by many sub classes.



Hybrid Inheritance:

Hybrid inheritance is one of the inheritance types in OOP which is a combination of Single and Multiple inheritance.



SYNTAX OF INHERITANCE IN JAVA

```
class <class_name>
{

        //body of class

}

class <class_name> extends <class_name>
{

                //body of class

}
```

Let's Understand With Example

Entity Class

```
class Doctor {
 public void qualification() {
  System.out.println("Qual Details...");
 }
}

class Surgeon extends Doctor {
 public void specialist() {
  System.out.println("Surgeon Detail...");
 }
}
```

Driver Class

```
public class Hospital {
 public static void main(String args[]) {
  Surgeon s = new Surgeon();
  s.qualification();
  s.specialist();
 }
```

Lecture -12

super Keyword

Part-12

Object Oriented Programming

What Is “super”?

- In Java, the keyword “super” is used by a CHILD CLASS for EXPLICITLY referring (Using dot operator) the members of its PARENT CLASS.

Let's Understand With Example

Driver Class

```
B obj=new B();
obj.show();
```

Entity Class

```
class A
{
    public void display() {
        ...
    }
    public void print() {
        ...
    }

}
class B extends A
{
    public void show()
    {
        display();
        print();
    }
}
```

Uses of “super”

- For calling constructor of PARENT class from CHILD class

- For resolving method overriding

Lecture -13

Method Overriding

Part-13

Object Oriented Programming

What is Method Overriding

- Whenever a child class contains a method with the same prototype as the parent class, then we say that the method of child class has OVERRIDDEN the method of parent class

Let's Understand With Example

Driver Class

```
class UseDemo {

  public static void main(String
[]args){

      DemoB obj=new DemoB();
           obj.show();

   }
}
```

Entity Class

```
class DemoA {

   public void show(){
      //Some code
   }
 }

class DemoB extends DemoA {

    public void show(){
      //Some other code
    }
}
```

Question ?

Why we override a super class method ?

Answer :-

To provide better/correct implementation of that method in child class

Let us understand this with an example

Let's Understand With Example

Driver Class

```
class UseFruit{

  public static void main(String []args){

        Grapes G=new Grapes();
        G.taste();

   }
}
```

Entity Class

```
class Fruit{

   public void taste(){

       System.out.println("Sweet");
   }
}

class Grapes extends Fruits{

    public void taste(){

       System.out.println("Citrus");
    }
}
```

Role of super in Overriding

- If the child class has overriding a method of parent class, but due to some reason now the programmer of child class also wants to execute the version of parent class of the overridden method.

- In this case he will use super and the syntax will be:

- super.<method_name>(<list_of_arg>);

Method Overriding V/s Method Overloading

Method Overloading

- Overloading can be done either within the same or between methods of PARENT & CHILD class
- Overloading says methods must have same, but COMPULSORILY different arguments

Method Overriding

Overriding can never be done within a single class and it always requires INHERITANCE i.e overriding can be done between between of PARENT & CHILD only

Overriding says methods MUST COMPULSORILY have exactly same prototype. Although overriding allows CO-VARIANT return types

Lecture -16

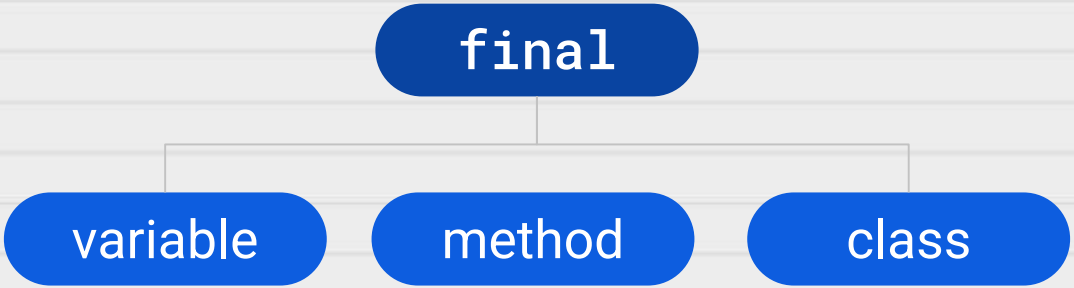
final keyword

Part-16

Object Oriented Programming

What is “final” keyword in java

➔ final is a non access modifier in java and can be used in three ways



“final” variable

- **final variable:** Once we declare a variable as a final we can’t perform re-assignment.

Let's Understand With Example

```
class Circle {
        private int radius;
        private final double pi = 3.14; //Now this value is
fixed

        public Circle(int radius) {
                this.radius = radius;
        }
}
```

Another Way To Initialize final Data

```
class Circle {
        private int radius;
        private final double pi;

        public Circle(int radius) {
                this.radius = radius;
                pi = 3.14; //Now this value is fixed
        }
}
```

Let's Understand With Example

```
class Circle {
        private int radius;
        private final double pi = 3.14;

        public Circle(int radius) {
                this.radius = radius;
                pi = 5.0; //Error: final cannot be changed
        }


}
```

“final” method

- final method: Whenever we declare a method as a final it can’t be overridden in the child class.
- Let us understand with an Example

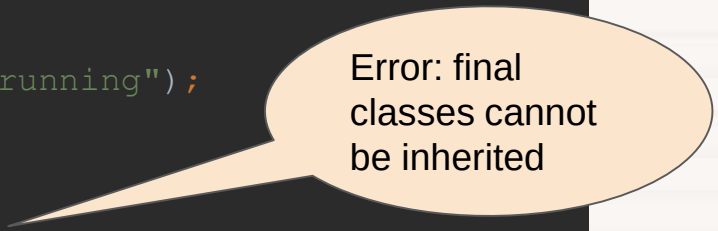
```java
//Entity Class
class Bike {
    final public void run(){
        System.out.println("running");
    }
}

class Honda extends Bike {
    public void run() {
        System.out.println("running safely with 100kmph");
    }
}

// Driver Class
class UseBike {
    public static void main(String args[]){
        Honda honda= new Honda();
        honda.run();
    }
}
```

Error: Overriding a final method is not allowed

“final” class

- final class: Whenever we declare a class as final it can’t be extended or inherited by sub classes.

```java
//Entity Class
final class Bike {
   public void run(){
        System.out.println("running");
   }
}

class Honda extends Bike {

}
```



Some predefined final classes provide by java

- Math
- String
- All Wrapper Classes like
  - Integer
  - Double
  - Character
  - etc.

Lecture -17

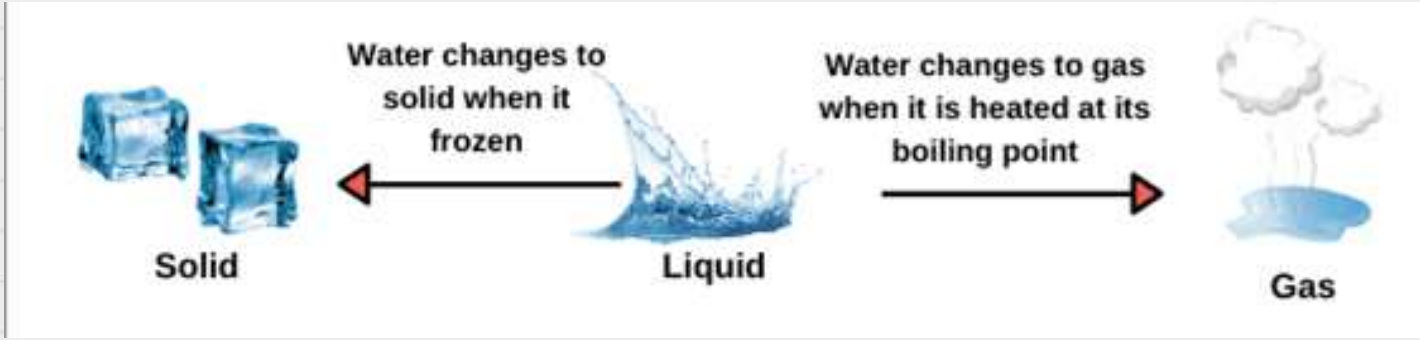
Polymorphism

Part-17

Object Oriented Programming

What is Polymorphism

- The derivation of the word Polymorphism is from two different Greek words- poly and morph.
- “Poly” means multiple, and “Morph” means forms.
- So, polymorphism means ability to acquire multiple forms.



Real Life Example of Polymorphism

Another Example of Polymorphism

In Shopping malls behave like

CUSTOMER

In Bus behave like

PASSENGER

In School behave like

STUDENT

At Home bheave like

SON

Types of polymorphism in Java

➔ Compile time polymorphism

➔ Runtime polymorphism

Compile time polymorphism

- Compile-time polymorphism is also known as static polymorphism.
- To achieve compile time polymorphism we must have different implementations of the same method in a class.
- And this is done using method overloading

Runtime Polymorphism

Part-18

Object Oriented Programming

Runtime Polymorphism

- Runtime polymorphism is also called as Dynamic Method Dispatch in java.
- To achieve Runtime polymorphism in java we must be able to call different versions of the same method using the same reference variable.
- These methods should be present in **child class** and the reference must be of **parent class.**

Entity Class

```
class Rectangle
{
    public void area()
    {
        ...
    }
}
class Circle
{
    public void area()
    {
        ...
    }
}
class Triangle
{
    public void area()
    {
        ...
    }
}
```

How to call area() of Every Class?

Driver Class

```
public class UseShape
{
   public static void main(String[] args)
   {
                Rectangle r = new
Rectangle();
                r.area();

                Circle c = new Circle();
                c.area();

                Triangle t = new Triangle();
                t.area();
    }
}
```

Although it works, but it is not runtime polymorphism

Runtime Polymorphism

- But to Understand how this concepts works we must first understand the concept of binding.

What Is Binding ?

★ The word binding is the mechanism using the compiler decides that which function call which function body will get executed.

In Java we have 2 types of binding

a. Early binding Or Compile time binding or static binding

a. Late binding or runtime binding or dynamic binding

Types of Binding

Part-19

Object Oriented Programming

Early Binding

- If the method being called is a static method, then java selects the method by looking at the (class) of the reference used in the call and not the type of object to which the reference is pointing.

Entity Class

```
class Parent
{
    public static void show()
    {
        ...
    }
}
class Child extends Parent
{
    public static void show()
    {
        ...
    }
}
```

Which method will execute ?

Let's Understand With Example

Driver Class

```
public class UseShape
{
public static void main(String[] args)
{
                        Parent p = new Child
();
            p.show();

            }
}
```

Late Binding

- If the method being called is a non-static method i.e instance method, then Java selects the method by considering the type of object, pointed by the reference and not considering the type of reference itself

Entity Class

```
class Parent
{
    public void show()
    {
        ...
    }
}
class Child extends Parent
{
    public void show()
    {
        ...
    }
}
```

Which method will execute ?

Let's Understand With Example

Driver Class

```
public class UseShape
{
public static void main(String[] args)
{
                        Parent p=new
Child();
            p.show();

            }
}
```

Abstract
Method & class

Part-21

Object Oriented Programming

Find The Problem In This Code?

```
class Language{

        public void greetings(){

   }
}
```

- The problem with the code is that we have left the body of the method `greetings()` empty

- This is because we don't have any suitable logic for the method `greetings()` in the class Language

Can we improve it ?

- Yes we can improve it by removing the body of the method and just declaring it in the class Language
- But to do this we must prefix the method as well as the class with the keyword abstract

For ex:-

```
abstract class Language
{

        abstract public void greetings();
}
```

When we should declare a method as abstract ?

- There are situations when we have a method in super class which cannot be implemented properly due to lack of information in super class.

- But we need this method for achieving runtime polymorphism and thus in such cases we must declare a method as abstract.

For ex:-

- Suppose we have class called Instrument and the class has a method called sound ().

- What will be the implementation of sound () in the Instrument ?

Entity Class

```
class Instrument{

    public void sound(){

    }
}
class Violin extends Instrument{

    public void sound(){
        ...
    }
}
class Guitar extends Instrument{

    public void sound(){
        ...
    }
}
```

Let's Understand With Example

Driver Class

```
class UseInstrument{
    public static void main(String args[]){
        Instrument I1;
        I1=new Violin();
        I1.sound();
        I1=new Guitar();
        I1.sound();
    }
}
```

What is the implementation ?

Entity Class

```
abstract class Instrument{

   abstract public void sound();

}
class Violin extends Instrument{

   public void sound(){
      ...
   }
}
class Guitar extends Instrument{

   public void sound(){
      ...
   }
}
```

Improved version

Driver Class

```
class UseInstrument{
   public static void main(String args[]){
        Instrument I1;
        I1=new Violin();
        I1.sound();
        I1=new Guitar();
        I1.sound();
   }
}
```

Abstract
Method & class

Object Oriented Programming

Some important points

★ To make a method abstract it is compulsory to use the keyword abstract in the method prototype.

★ An abstract method should never have any implementation in the class where it is being declared.

★ If a method has been declared as “abstract” in the class, then the class itself must be prefixed with the keyword abstract

For:- example

```
class A
{
    public abstract void show();
}
```

Error !

```
                    abstract class A
                    {

p                   ct void show();
                    }
```

Now it is OK

Some important points

★ If a class is abstract then we are not allowed to create its “object” .

★ Although we can create reference of the class.

```
A obj;// Ok
obj=new A();// Error
```

Some important points

★ An abstract class can CONTAIN concrete methods as well as Constructors.

★ This is because abstract classes can be inherited and these methods can be inherited and these methods can be accessed by the objects of child classes.

Some important points

★ If a class inherits an abstract class then

- Either it must compulsorily override all the abstract methods inherited from the super class.

  OR

- The child class itself will have to be prefix with the keyword abstract and we won't be allowed to create its object also.

Abstract

Part-23

Object Oriented Programming

Which methods cannot be declared as abstract ?

- Static Methods
- Constructors
- Private methods
- Final methods

Interface

Part-24

Object Oriented Programming

What is an interface?

➔ An interface is almost same as a pure abstract class.

➔ Just like a class or abstract class an interface also can contain data members.

What is an interface?

- But every data declared in an interface is automatically converted to “public” “static” and “final” by java.
- An interface can also contain methods but every method by default is “public” and “abstract”
- However from java 8, an interface can also contain “default” and “static” methods

What is an interface?

- Just like we cannot instantiate an abstract class, similarly we also cannot instantiate an interface

- However we can create a reference of an interface.

What is an interface?

- Just like an abstract class an interface also can be inherited by child classes but the keyword used for this inheritance is implements.
- The reference of an interface can point to the object of its implementation class or child class
- And this forms the basis of runtime polymorphism

What is an interface?

➔ If a class inherits an interface then it is compulsory to override every abstract method inherited from the interface.

➔ Otherwise, the derived class also will have to be declared as abstract.

Syntax of declaring an interface

```
interface <interface_name>
{
        <data_type> <var_name>=<value>;
         .
         .
         .
        <return_type> <method_name> (list_of_arg);
         .
         .
         .
}
```

Let's Understand With Example

Interface declaration

```
public interface Animal {
    void makeSound();
    void eat();
}
```

```
//Both the above methods are by default
public and abstract
```

Let's Understand With Example

Entity class

```
public class Elephant implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Trumpet!");
    }

    @Override
    public void eat() {
        System.out.println("Elephant eats grasses and leaves");
    }
}
```

Let's Understand With Example

Entity class

```
public class Lion implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Roar!");
    }

    @Override
    public void eat() {
        System.out.println("Lion eats flesh");
    }
}
```

Let's Understand With Example

Driver class

```
class UseAnimal{
   public static void main(Sring []args){
          Animal obj;

      obj= new Elephant();
      obj.makeSound();
      obj.eat();

      obj= new Lion();
      obj.makeSound();
      obj.eat();
   }
}
```

Abstract Class
vs
Interface

Part-25

Object Oriented Programming

Abstract Class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	1) Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	2) Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	3) Interface has **only static and final variables**.

Abstract Class	Interface
4) Abstract class **can provide the implementation of interface**.	4) Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	5) The **interface keyword** is used to declare interface. we
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	6) An **interface** can extend another interface only. However it can extend multiple interfaces.

Abstract Class	Interface
7) An **abstract class** can be extended using keyword "extends".	7) An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	8) Members of a Java **interface** are public by default. But from **java 9** onwards methods of an interface can also be declared as **private**

9) **Example:**
 	
public abstract class Shape{
public abstract void draw();
}
	9) **Example:**
	
	public interface Drawable{
	void draw();
	}

Association

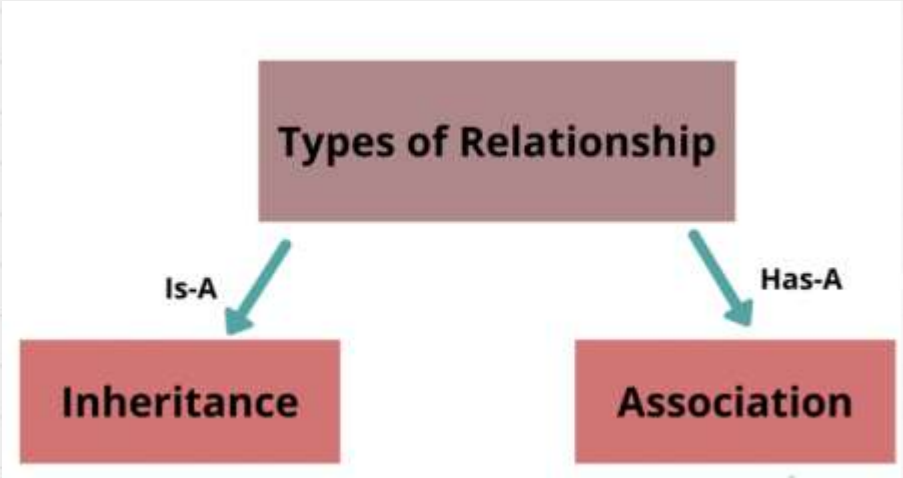
Part-26

Object Oriented Programming

Java has two types of relationship

1. Inheritance (IS-A)

2. Association (HAS-A)



What Is Inheritance ?

1. Inheritance (IS-A)

★ An IS-A relationship signifies that one object is a type of another.

★ For Example:

- CAR IS-A vehicle
- Apple IS-A fruit
- Circle IS-A shape

★ It is implemented using “extends” or “implements” keywords.

Let's Understand With Example

Entity Class

```
class Vehicle {

          //Some Code

}

class Car extends Vehicle{

          //Some Code
}
```

Driver Class

```
public class UseCar {

 public static void main(String
args[]){

          Car car = new Car();
          //Some More Code

   }
}
```

What Is Association ?

2. Association (HAS-A)

- ★ A HAS-A relationship signifies that a class is associated with that is it holds object(s) of another class in its body
- ★ For Example:
  - Car has Engine
  - College has Students
  - House has Rooms
- ★ For instance, class A holds class B's reference and can access all properties of class B.

Entity Class

```
class Engine {
            public void start() {
            //Some Code
            }
}

class MusicPlayer {
      public void start() {
            //Some Code
            }
}

class Car {
      Engine engine = new Engine();
      MusicPlayer player = new MusicPlayer();

      public void startEngine() {
            engine.start();
            }
            public void startMusicPlayer() {
            player.start();
            }
}
```

Let's Understand With Example

Driver Class

```
public class UseCar{

 public static void main(String args[]){

     Car car = new Car();
     car.startEngine();
     car.startMusicPlayer();

     //Some More Code
 }

}
```

Types of Relationship

Types of Association

Part-27

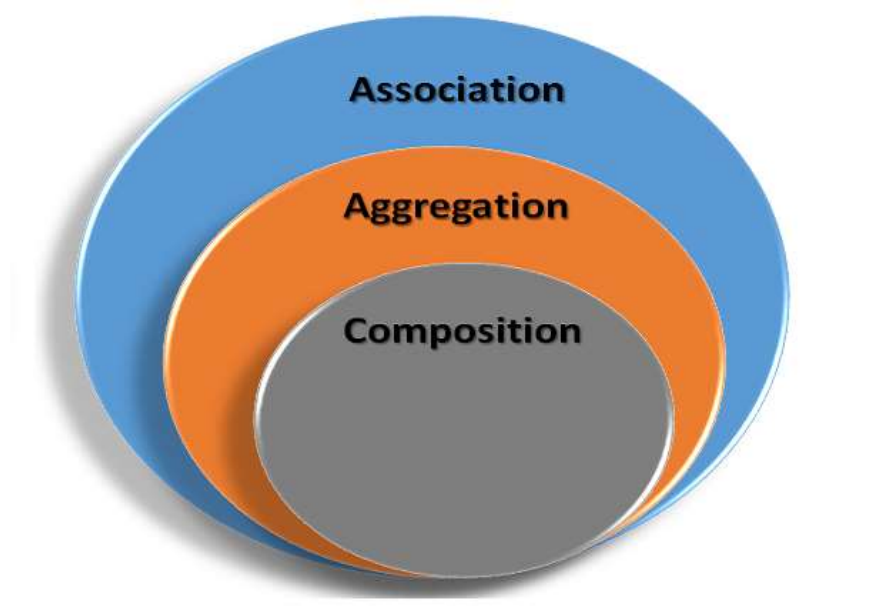
Object Oriented Programming

What Is Association ?

A HAS-A relationship signifies that a class is associated with that is it holds object(s) of another class in its body

- ★ For Example:
  - Car has Engine
  - College has Students
  - Building has Rooms
- ★ In simple words we can say that, class A holds class B's reference and can access methods of class B.

Types Of Association



Aggregation

- ★ Aggregation is often represented using a "has-a" relationship.
- ★ For example, a car object "has-a" music player object.
- ★ Similarly, a computer object "has-a" hard drive object and so on.
- ★ In aggregation, the component object can exist independently of the container object.

Let's Understand With Example

```
class MusicPlayer {
    public void start() {
        //Some Code
        }
}

class Car {
        private String name;
    private MusicPlayer player;
}
```

Types of Association

Part-28

Object Oriented Programming

Composition

- A composition in Java between two objects associated with each other exists when there is a strong relationship between one class and another.
- Means contained object cannot exist without the owner or container object.
- For example, A College is a composition of Department. If the College object doesn't exist or dies then Department object will also not exist
- Thus we can say that composition is a restricted form of Aggregation.

Implementation of Composition in Java

- The College and Department relationship is implemented using the concept of inner class in Java.

- Also we use, the “final” keyword while representing composition.

- This is because the ‘Owner’ object expects a part object to be available and function by making it “final”.


THANK
YOU