Class Fundamentals & Inheritance

Dr. Partha Pratim Sarangi

School of Computer Engineering

Classes and Objects

Class

- A class is a user defined blueprint or template from which objects are created.
- It has set of attributes and methods that describe the behavior of the class.

Object

- It is a basic unit of Object Oriented Programming and represents the real life entities.
- When an object of a class is created, the class is said to be instantiated. All the instances share the attributes and the behavior of the class.

Class

In general, class declarations can include these components, in order:

- Modifiers: A class can be public or has default access.
- class keyword: class keyword is used to create a class.
- Class name: The name should begin with an initial letter (capitalized by convention).
- Superclass (if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
- Body: The class body surrounded by braces, { }.

Class Components

`public`	Class is publicly accessible.
`abstract`	Class cannot be instantiated.
`final`	Class cannot be subclassed.
`class NameOfClass`	***Name of the Class.***
`extends` *`Super`*	Superclass of the class.
`implements` *`Interfaces`*	Interfaces implemented by the class.

`{`
`  `*`ClassBody`*
`}`	

Some Important points about a class:

- A class is a non-primitive or user-defined data type in Java, while an object is an instance of a class.
- A class is a group of objects that share common properties and behavior.
- Class is a logical entity that does not occupy any space/memory. Memory is allocated when objects are created of a same class type.
- In Java, we can not declare a top-level (Outer) class as **private**. Java allows only **public and default** access specifiers for top-level classes. We can declare inner classes as private.

- There can be only **one public class** in a single program and its name should be the same as the name of the Java file. There can be **more than one non-public classes** in a single Java file.
- A public class is visible to all classes from all the packages.
- A class with default access is visible only to the classes within the same package.
- We can also use the non-access modifiers for the class such as **final, abstract and strictfp**.
- We cannot create an object or instance of an abstract class.
- No subclasses or child class can be created from a class that is declared as final.
- A class cannot be declared both as **final and abstract** at the same time.

Continued

A class can have any of the following three variables in Java – local, instance, and static variables.

- **Local variables** − Variables defined inside methods, constructors or blocks are called local variables. These variables are destroyed after execution of the method has completed.
- **Instance variables** − Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.
- **Class variables** − Class variables are variables declared within a class, outside any method, with the static keyword.

Object

An object consists of:

- **State**: It is represented by attributes of an object. It also reflects the properties of an object.
- **Behavior**: It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

- Note: **Size of an object is equal to the size of instance variables present in the corresponding class.**

Creating an Object

In Java, the new keyword is used to create new objects.

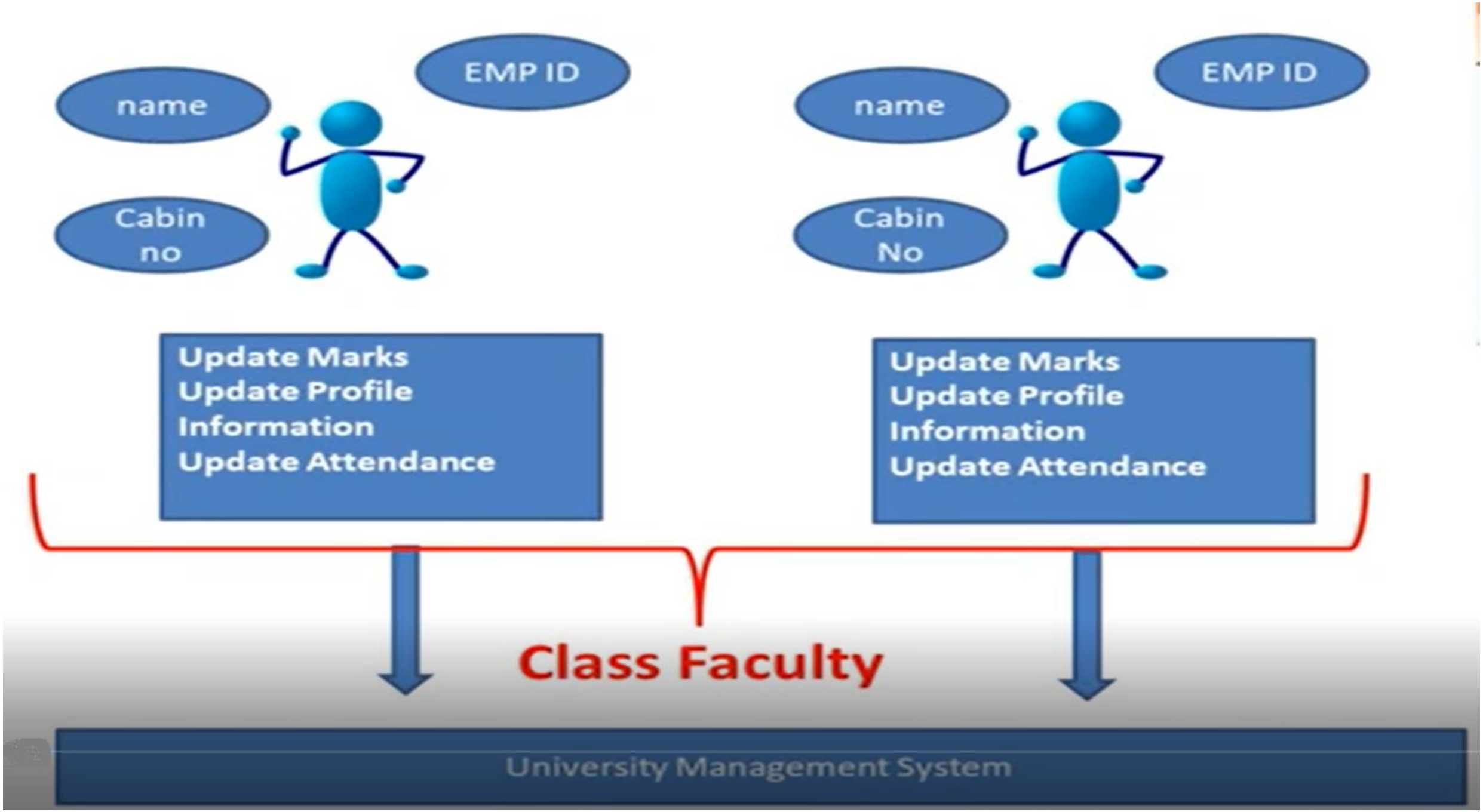
- There are three steps when creating an object from a class:
  - **Declaration** − A variable declaration with a variable name with an object type.
  - **Instantiation** − The 'new' keyword is used to create the object.
  - **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.
- Student s1 = new Student();

Add Profile Info
View Marks
View Profile Information
View Attendance

View Marks
View Profile Information
View Attendance

Class Students

University Management System


name
EMP ID
Cabin no
name
EMP ID
Cabin No
Update Marks
Update Profile
Information
Update Attendance
Update Marks
Update Profile
Information
Update Attendance
Class Faculty
University Management System

```
import java.util.*;

public class Inheritance {
   public static void main(String[] args) {
      Student s1 = new Student();
      s1.addInfo();
      s1.viewInfo();
   }

}

   class Student {
   String name = "";
   String regno = "";
   public void addInfo()
   {
      Scanner input = new Scanner(System.in);
      System.out.println("Enter your name: ");
      name = input.nextLine();
      System.out.println("Enter regno: ");
      regno = input.nextLine();
   }

   public void viewInfo()
   {
      System.out.println(name+" "+regno);
   }
}
```

Creating array of objects

- Create an array of students objects
- Read name and regno and display the data for 10 student objects using the concept of array of objects.

```
public static void main()
{
        Student  s[ ] = new Student[10];
        for(int i=0; i<10; i++)
        {
            s[i] = new Student();
        }
}
```

Classes: Student and Faculty

Class Student

Data members

- Name
- Reg no
- Dob
- Address

Methods/Member Functions

- View Profile()
- View Attendance()
- View Marks()

Class Faculty

Data members

- Name
- Emp id
- Cabin no

Methods/Member Functions

- View Profile()
- Update Attendance()
- Update Marks()

Constructor in Java

- Constructor in java is used to create the instance of the class.
- Constructors are almost similar to methods except for two things - its name is the same as the class name and it has no return type.
- Sometimes constructors are also referred to as special methods to initialize an object.
- Whenever we use **new** keyword to create an instance of a class, the constructor is invoked and the object of the class is returned by Java runtime.
- This is the way java runtime distinguish between a normal method and a constructor.

- A Constructor is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object.
- It can be used to initialize the objects to desired values or default values at the time of object creation. It is not mandatory for the coder to write a constructor for a class.
- It should not return a value not even void.
- A class can have multiple constructors, but they must have different signatures because the Java compiler differentiates the constructors based on the number and the type of the arguments.
- All classes have at least one constructor. If a class explicitly does not declare any constructor, then the Java compiler automatically provides a default constructor, i.e., the no argument constructor.

Important points to remember about constructor

- Constructor name is same as that of the class name.
- A constructor does not have any return type.
- A constructor cannot use any modifier.
- A constructor is of two types:
  - Parameterize constructor
  - Non-parameterize or default constructor
- If no constructor is defined in the program, then JVM automatically recognizes the default constructor.
- If the parameterized constructor is defined and the default constructor is called, then compile time error occurs.
- A constructor cannot be overridden but can be overloaded.

Types of Constructors

- Default Constructors
- No-Arguments Constructors
- Parameterized Constructors

Default Constructor

- It’s not required to always provide a constructor implementation in the class code.
- If we don’t provide a constructor, then java provides default constructor implementation for us to use.
- Default constructor only role is to initialize the object with default values for the instance variable and return it to the calling code.
- Default constructor is always without argument and provided by java compiler only when there is no existing constructor defined.

No-Arguments Constructors

- Constructor without any argument is called a no-args constructor.
- It's like overriding the default constructor and used to do initialization of object explicitly by program codes.
- Whenever we create object by new operator, then our no-args constructor will be called.

```
class Rectangle
{
        double width;
        double height;
        Rectangle()
        {
                width = 10.5;
                height = 5.5;
        }
        double area()
        {
                return width*height;
        }
}
```

```
class RectangleDemo
{
        public static void main(String[] args)
        {
            Rectangle r = new Rectangle();
            System.out.println(r);
            double area = r.area();
            System.out.println("Area of rectangle
is "+area);
        }
}
```

Parameterized Constructor

- Constructor with arguments is called parameterized constructor.
- The parameterized constructor is used to provide different values to distinct objects.

```
public class Data {

        private String name;

        public Data(String n) {

        System.out.println("Parameterized
Constructor");
                this.name = n;
        }
```

```
public String getName() {
                return name;
        }
        public static void main(String[] args) {
          Data d = new Data("Java");

        System.out.println(d.getName());
      }
}
```

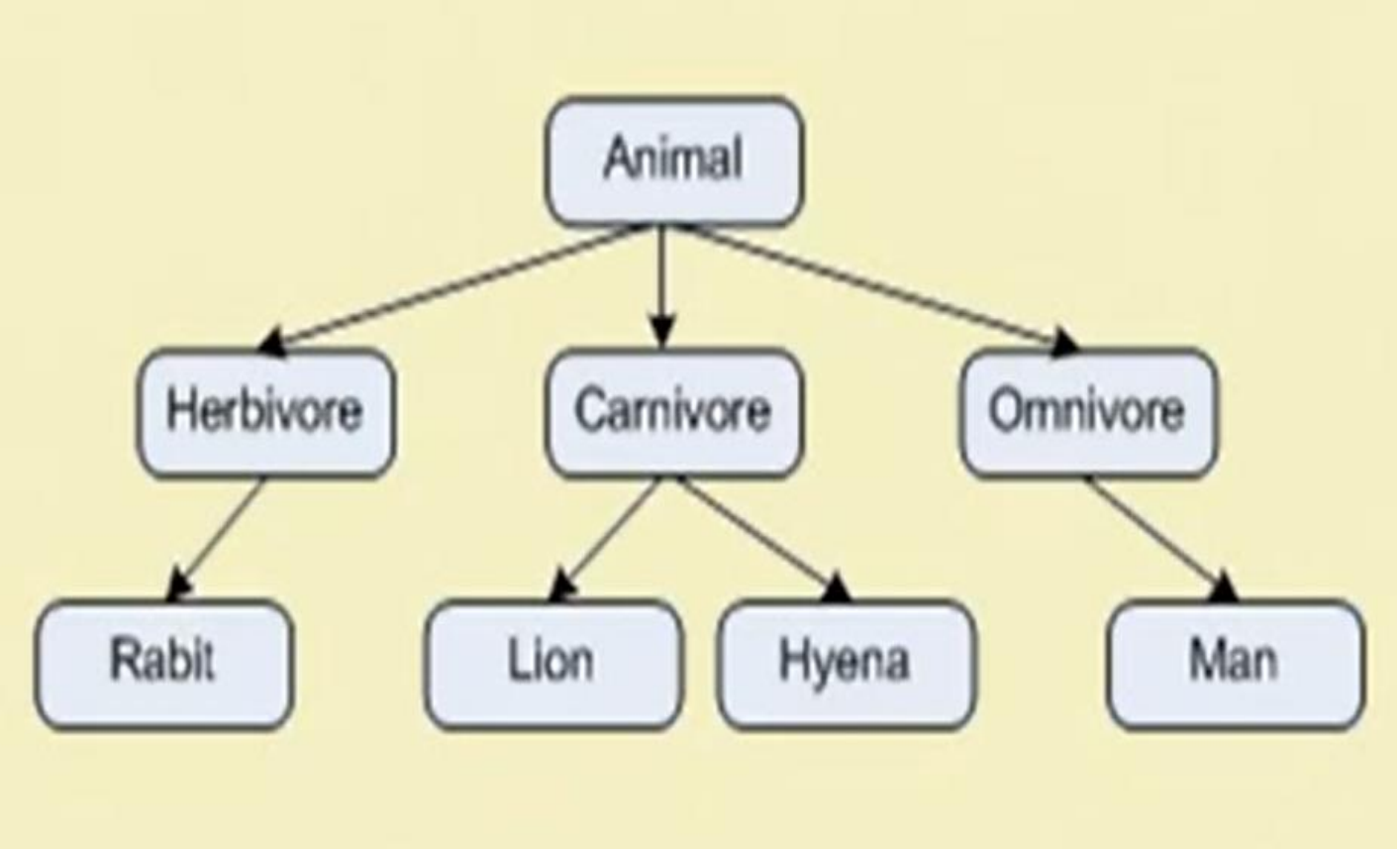
Inheritance

- It is an important Object Oriented Programming Concept.
- A Class inherits the data members (attributes) and methods from another class.
- New classes are built using the existing classes.
- It is used for code reusability.

Continued

- One object type is defined as being a special version of some other object type.
  - a *specialization*.
- The more general class is called:
  - base class, super class, parent class.
- The more specific class is called:
  - derived class, subclass, child class.

Continued



Single multi-level inheritance

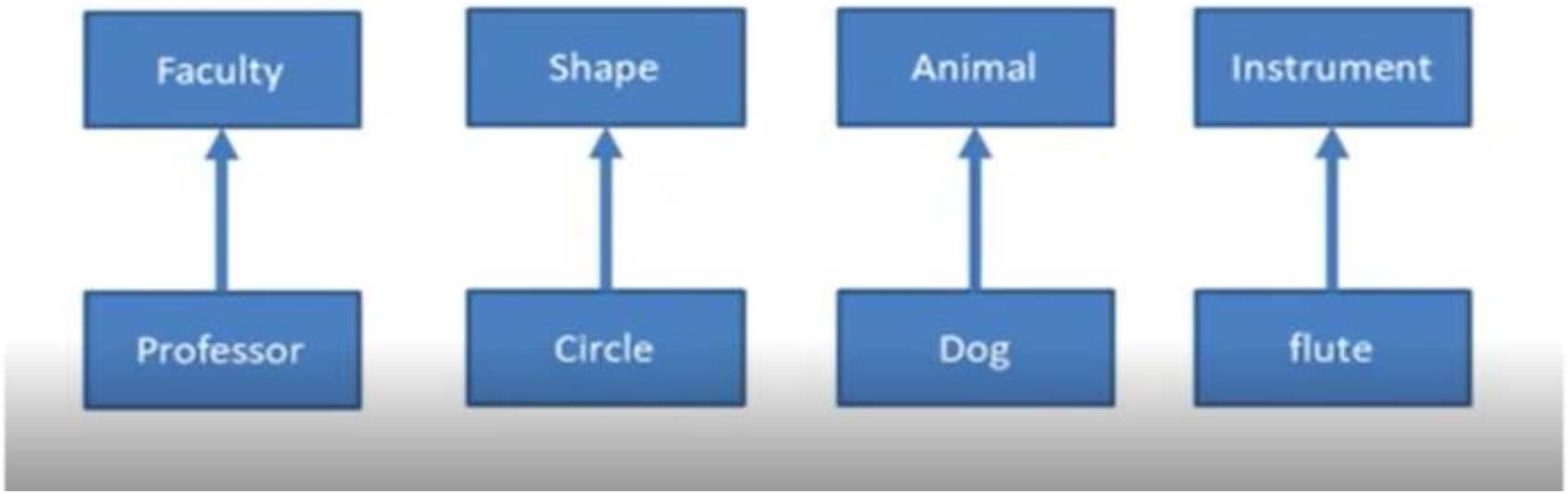
Inheritance in Java

- Inheritance in one of the important concept of object-oriented programming because it allows the creation of hierarchical classification.
- Using inheritance, one can create a general class that include some common set of items.
- This class then can be used to create more specific classes which has all the items from the base class, in addition to some items of its own.
- It is a “is-a” relationship, for example, Lion is an animal or car is a vehicle.

Inheritance : is-a relationship

[is a] relationship - Examples

- [is a] relationship between Objects is known as inheritance



Terms used in inheritance

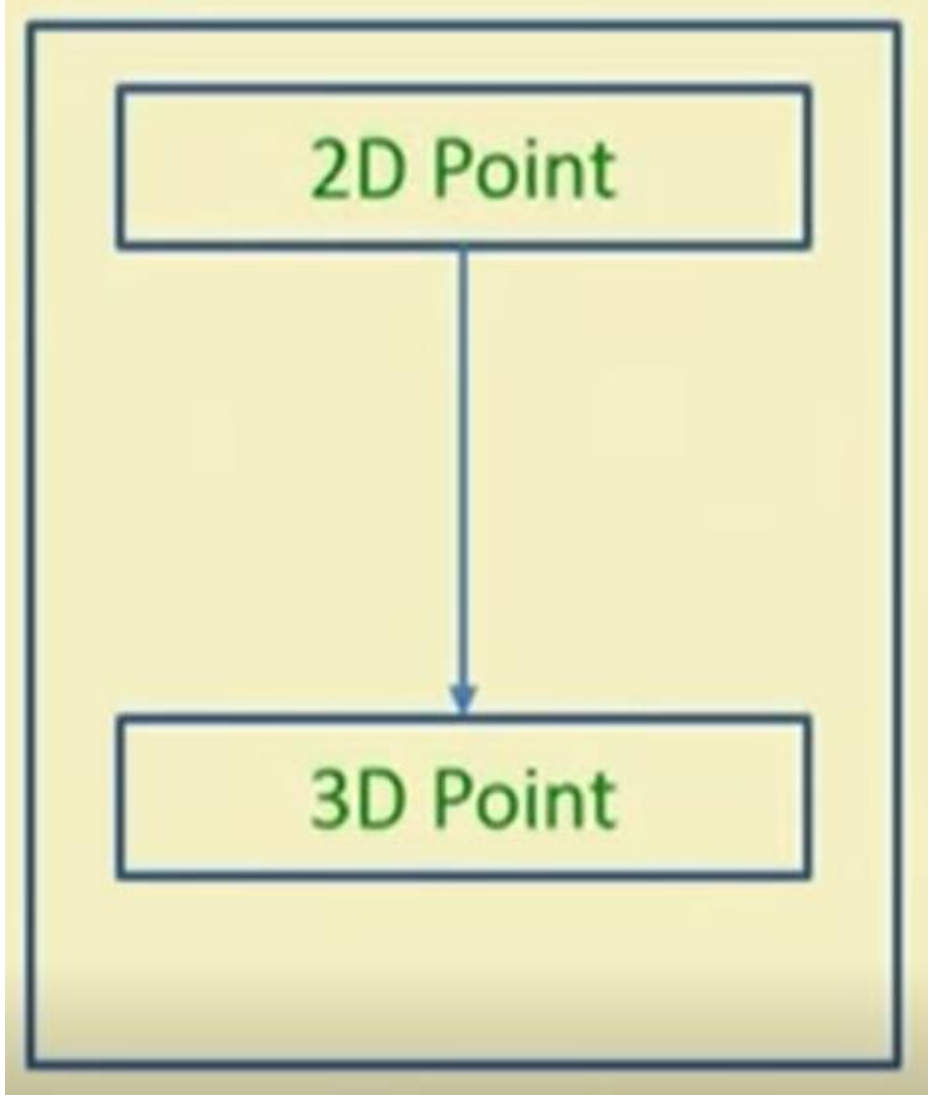
- Superclass: A class that is inherited is called a superclass.
- Subclass: The class that does inheriting is called a subclass.
  - ➢ A subclass is a specialized version of a superclass.
  - ➢ It inherits all of the instance variables and methods defined by the superclass and add its own unique items.
- Reusability: It is a mechanism which facilitates you to reuse the data and methods of the existing class when one creates a new class.
  - ➢ One can use the same data and methods already defined in the previous class.

Inheritance syntax

- The **extends** keyword is used to define a new class that derives from an existing class. The meaning of “extends” is to increase the functionality of new class.

```
Class <subclass-name> extends <superclass-name> {
        // data and methods in this subclass
}
```

Example of a simple inheritance

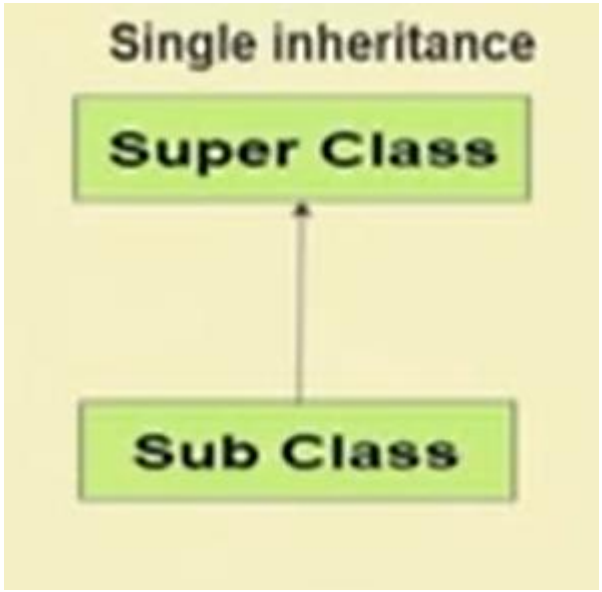


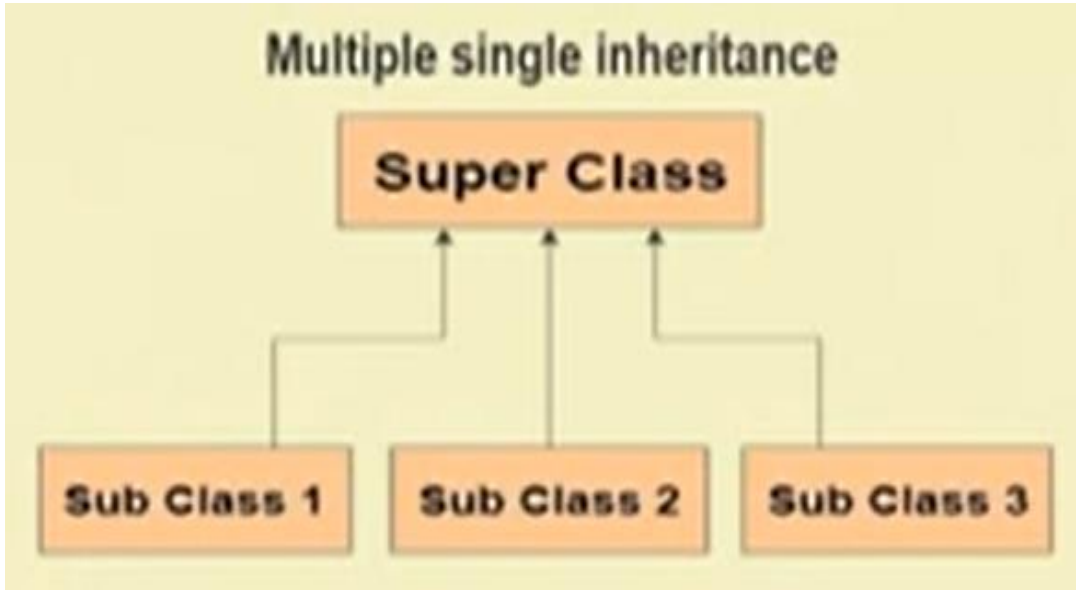

```
class Point2D {
        int x;
        int y;
        void display( ) {
                System.out.println(“x = ”+x+“y = ”+y);
        }
```

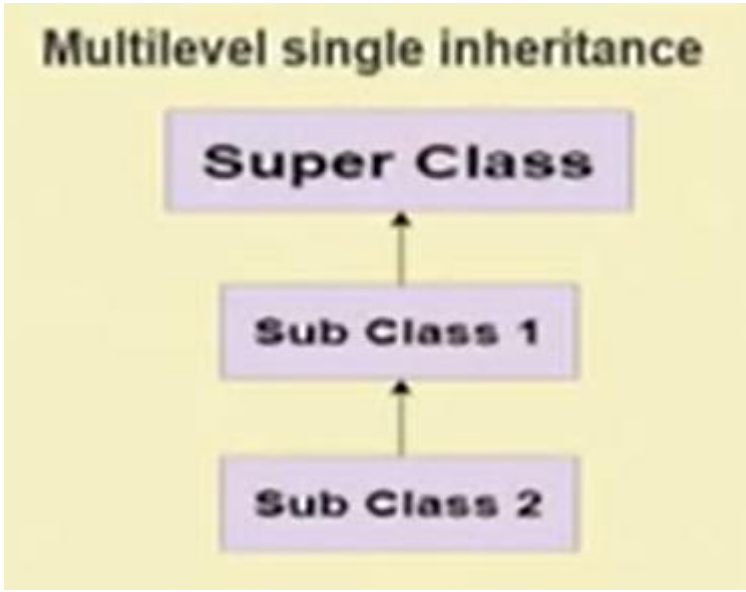
```
class Point3D extends point2D {
        int z;
        void display( ) {
                System.out.println(“x = ”+x+“y = ”+y+“z = ”+z);
        }
```

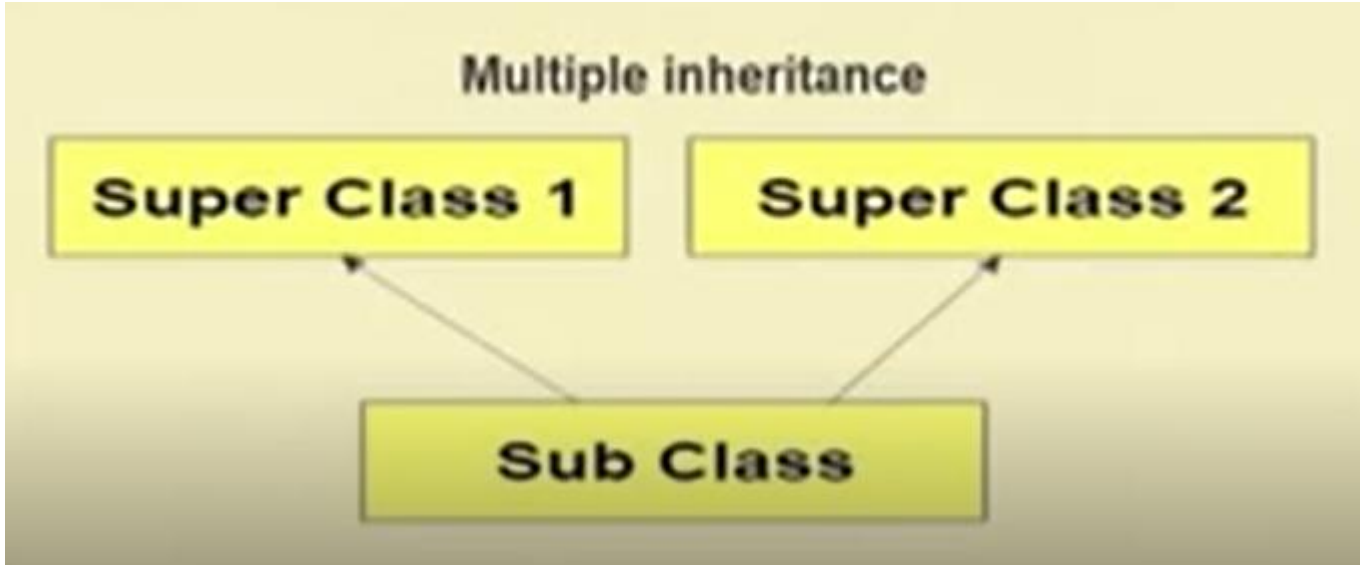
```
class SimpleSingleInheritance {
        public static void main(String args[ ]) {
         Point2D P1 = new Point2D( );
         Point3D P2 = new Point3D( );
         P1.x = 10;
         P1.y = 20;
         System.out.println("Point2D P1 is "+P1.display( ));
         P2.x = 5;
         P2.y = 6;
         P2.z = 15;
         System.out.println("Point3D P2 is "+P2.display( ));
      }
}
```

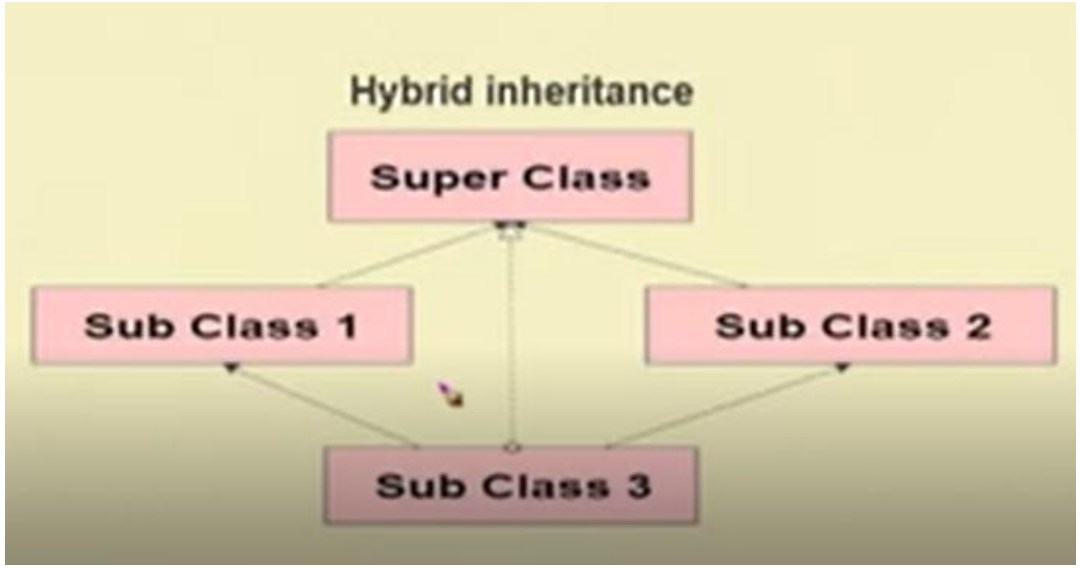
Types of inheritances







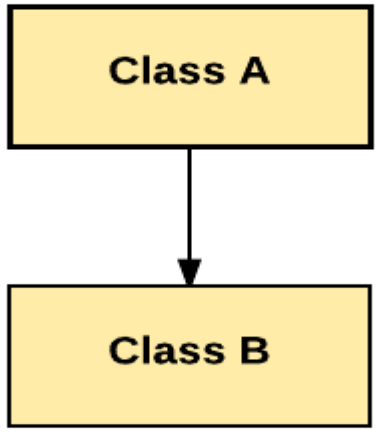




Types of Inheritance

Single Inheritance:

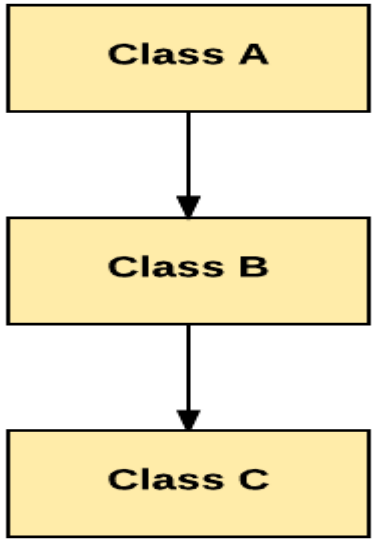
- In Single Inheritance one class extends another class (one class only).
- In above diagram, Class B extends only Class A. Class A is a super class and Class B is a Sub-class.



Single Inheritance

Multilevel Inheritance:

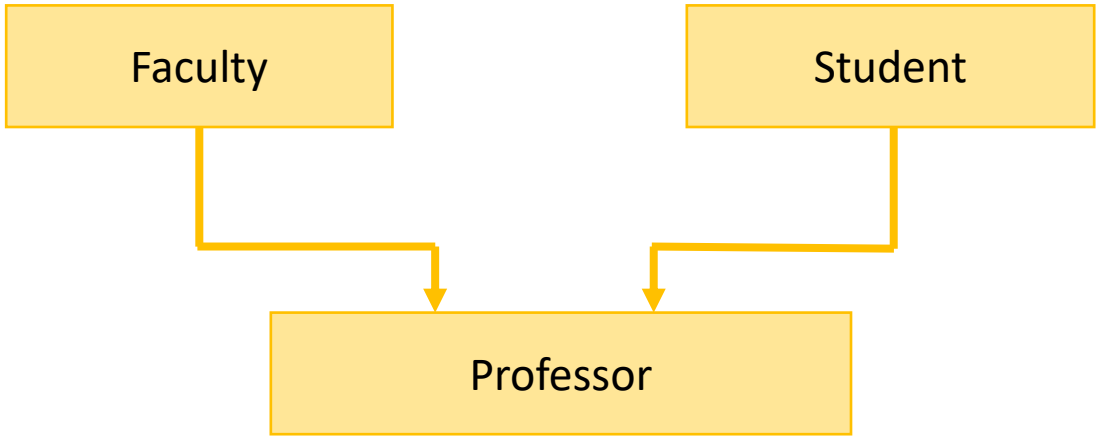
- **In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.**
- **As per shown in diagram Class C is subclass of B and B is a of subclass Class A.**



Multilevel Inheritance

Multiple inheritance:

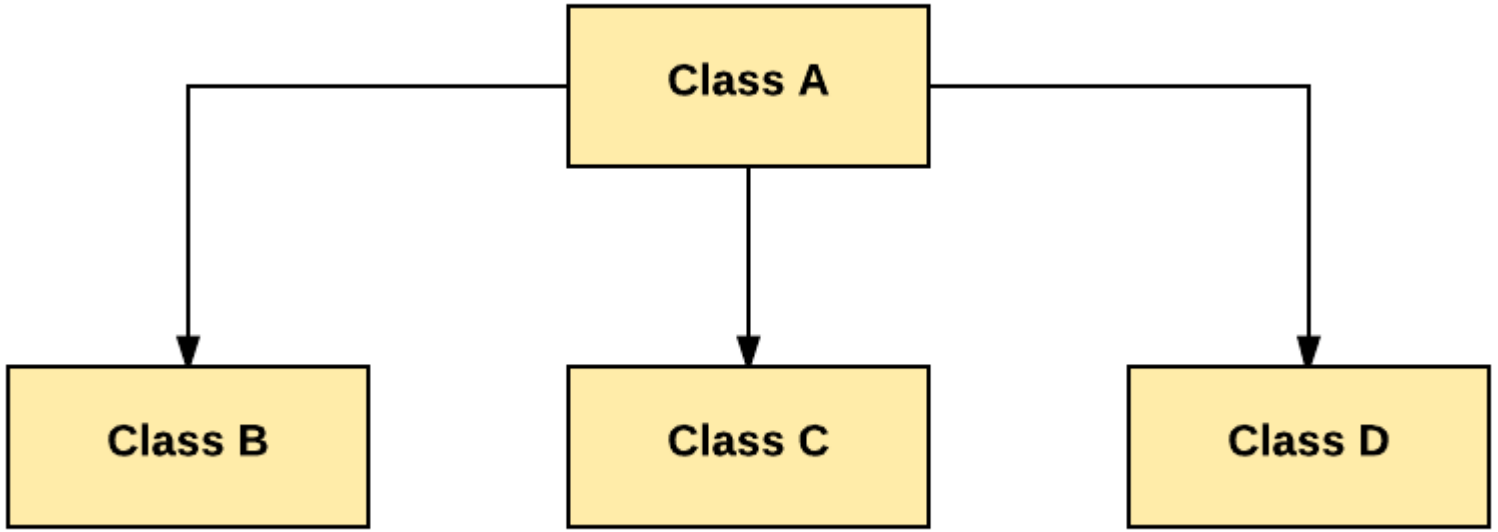
- In multiple inheritance, the child class is derived from more than one parent classes.
- The child class inherits the all attributes and methods from its super classes.
- Note: Java doesn't support Multiple inheritance.



Multiple inheritance

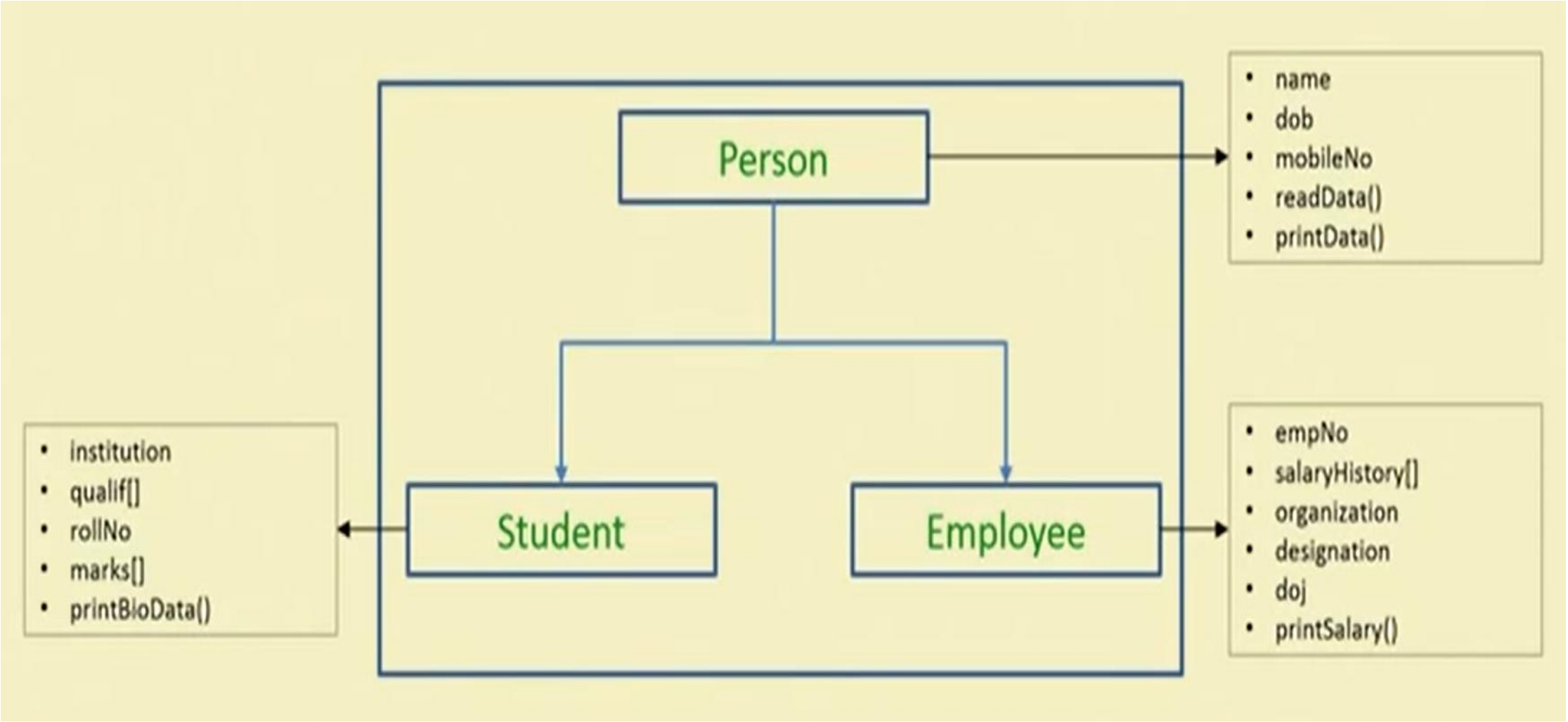
Hierarchical Inheritance:

- In Hierarchical Inheritance, one class is inherited by many sub classes.
- As per example, Class B, C, and D inherit the same class A.



Hierarchical Inheritance

Single inheritance in Java



```
class Person{
        String name;
        Date dob;
        int mobileNo;
        void readData(String n, Date d, int m){
            name = n;
            dob = d;
            mobileNo = m;
        }
        void printData(){
            System.out.println("Name : "+ name);
            dob.printDate();
            System.out.println("Mobile : "+ mobileNo);
        }
}
```

```
class Student extends Person{
        String institution;
        int[] qualif = new int[20];
        int rollNo;
        int[] marks = new int[5];

        void printBioData(){
            printData();
            System.out.println("Institution : "+ institution);
            System.out.println("Roll : "+ rollNo);
            for(int q=0; q<qualif.length;q++){
               System.out.println("Marks "+q+": "+ qualif[q]);
            }
            for(int m=0; m<marks.length;m++){
               System.out.print("Result "+m+": "+marks[m]);
            }
        }
}
```

```
class Employee extends Person{
        int empNo;
        int[] salaryHistory = new int[12];
        String organization;
        String designation;
        Date doj;
        void printSalary(){
            for(int s=0; s<salaryHistory.length;s++){
               System.out.println("Salary "+s+": "+salaryHistory[s]);
            }
        }
}
```

```
class inheritanceDemo1{
      public static void main(String args[]){
          Person p = new Person();
          //Code with the objects p…
          Student s = new Student [100];
          //Code with the objects s…
          Employee e = new Employee[50];
          //Code with the objects e…

      }
}
```

Method overriding in Java

Usage of Java method overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its super class.
- Method overriding is used for runtime polymorphism.

Rules for Java method overriding

- The method must have the same name as in the parent class.
- The method must have the same parameters as in the parent class.
- There must be an “is-a” relationship (inheritance).

Dynamic method dispatch concept

- Dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile time, it is called Runtime polymorphism.
- In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Continued…

- Dynamic method dispatch is a mechanism in which a call to an overridden method is resolved at runtime rather than compile time.
- This concept is called runtime polymorphism.
- In this mechanism a superclass reference variable can refer to a subclass object.
- When an overridden method is called through a superclass object (reference), Java determines which version of that method is called at the runtime.

Method overriding: An Example

```
class Point2D{
        int x;
        int y;
        Point2D(int a, int b){
            x = a;
            y = b;
        }
        void display(){
            System.out.println("x = "+x+"y = "+y);
        }
}
```

```
class Point3D extends Point3D{
        int z;
        Point3D(int c){
            z = c;
        }
        void display(){
            System.out.println("x="+x+"y="+y+"z="+z);
        }
}
```

```
class MethodOverridingTest{
    public static void main(String args[]){
        Point2D p = new Point2D(3.0, -4.0);
        p.display(); // Refers to the method in Point2D

        Point3D q = new Point3D(0.0);
        q.display(); // Refers to the method in Point3D

        Point2D x =(Point2D) q; // Cast q to an instance of class Point2D
        x.display();
        }
}
```

```java
class Bike
{
    void run()
    {
        System.out.println("Running");
    }
}
```

```java
class Splender extends Bike
{
    void run()
    {
        System.out.println("Running with safe speed");
    }
}
```

```java
class DynamicMethodDispatch
{
    public static void main(String args[])
    {
        Splender s = new Splender();
        s.run();
        Bike b = new Bike();
        b.run();
        /*Dynamic method dispatch in which overridden
          method is called through the reference variable
          of the super class*/
        Bike b1 = new Bike();
        b1.run();
        Bike b2 = new Splender(); //Up casting
        b2.run();
    }
}
```

super keyword concept in Java

- The super keyword in Java is a reference variable which is used to refer immediate parent class members.
- Whenever you create an instance of a subclass, an instance of its parent class is created implicitly, which is referenced by super keyword.

Usage of super keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super( ) can be used to invoke immediate parent class constructor.

super : Referring parent class instance variable

```
class Animal{
    String color="white";
}
class Dog extends Animal{
    String color = "black";
    void printColor(){
        System.out.println(color);
        System.out.println(super.color);
    }
}
class TestSuper1{
    public static void main(String args[]){
        Dog d = new Dog();
        d.printColor();
    }
}
```

super : Invoking parent class method

```
class Animal{
   void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
   void eat(){System.out.println("eating bread...");}
   void bark(){System.out.println("barking...");}
   void work(){
           super.eat();
           bark();
           eat();}
   }
class TestSuper2{
   public static void main(String args[]){
      Dog d = new Dog();
      d.work();
   }
}
```

super : invoking parent class constructor

```
class Animal{
  Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
    Dog(){
        super();
        System.out.println("dog is created");
    }
}

class TestSuper3{
   public static void main(String args[]){
       Dog d = new Dog();
   }
}
```

super : invoking parent class constructor

If there is a number of overloading constructors in the superclass then you have to define the super constructors matching with each constructor.

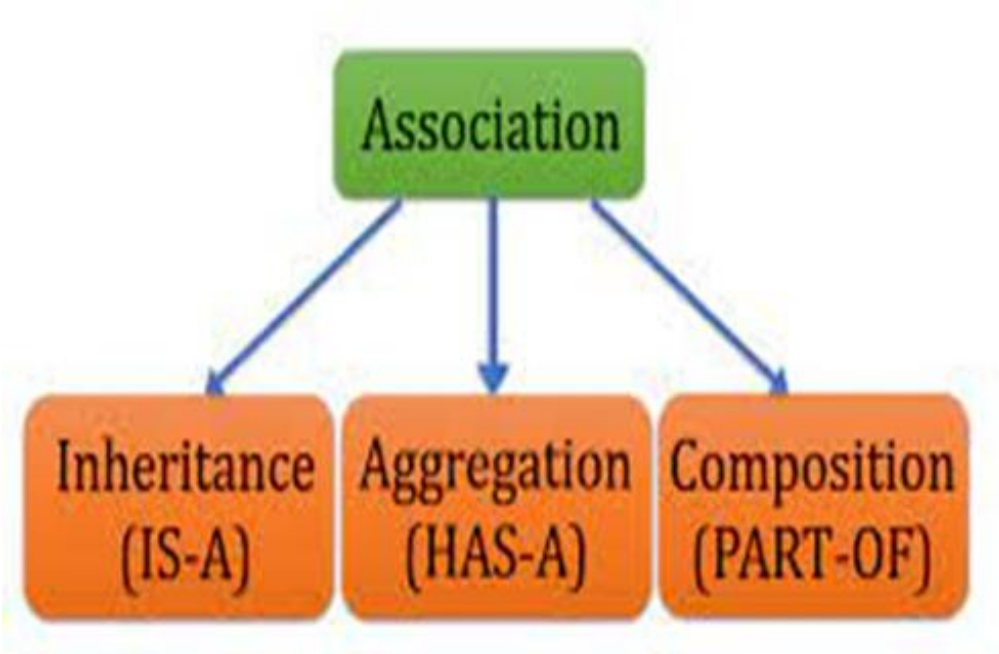
```
class Point2D{
   double x, y;
   Point2D(){x = 0.0; y = 0.0} //Default initialization
   Point2D(double x, double y){thix.x = x; this.y = y;}
}
class Point3D extends Point2D{
   double z;
   Point3D(){super(); z = 0.0} //Default initialization
   Point3D(double x, double y, double z){
   super(x, y);
   this.z = z; }
}
class TestSuper4{
   public static void main(String args[]){
       Point3D p = new Point3D(2.0, 3.0, 4.0);
   }
}
```

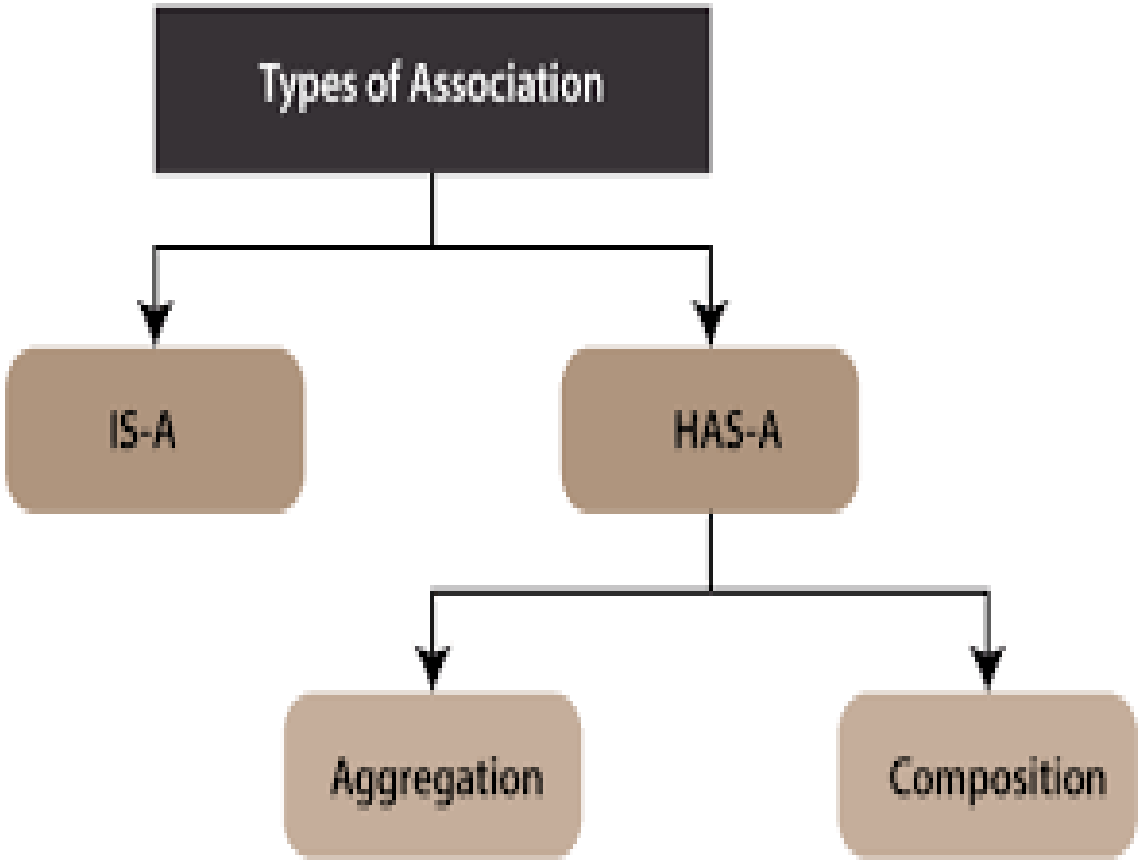
Types of relationship in Java

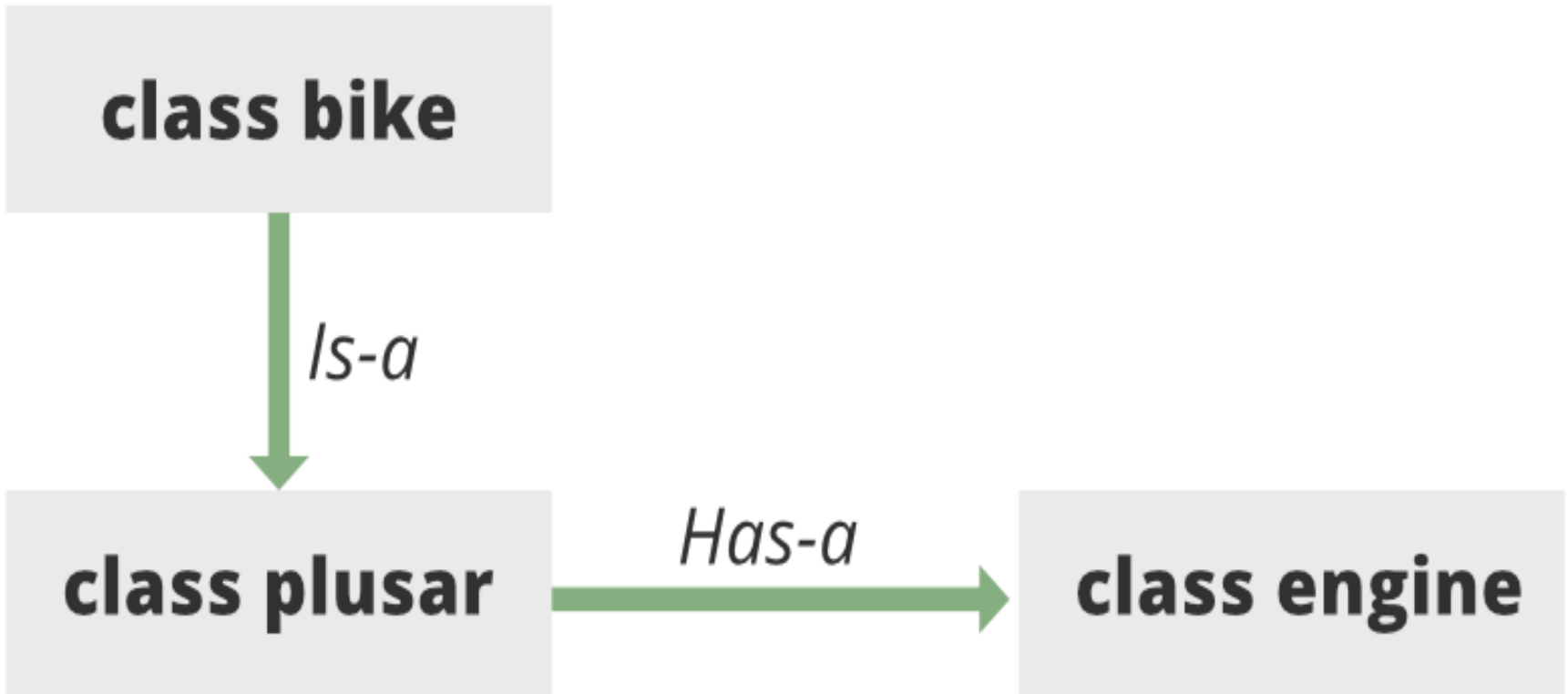
- When one object type depends on another, the relationship could be:
  - is-a (Inheritance)
  - has-a (Aggregation)
  - part-of (Composition)

- Association is a relationship between two separate classes which establishes through their objects. Has-a (Aggregation) and part-of (association) are the two forms of association.

Types of relationship in Java






class bike
Is-a
class plusar
Has-a
class engine

Composition/Aggregation

- One class has instance variables that refer to object of another.
- Sometimes we have a collection of objects, the class just provides the glue.
  - establishes the relationship between objects.
- When the reference of a class acts as a member in another class is known as aggregation.
- Aggregation represents [Has a] relationship.
- Aggregation “has-a” relation is used when we need to use property and behavior of a class without modifying it inside our class.

Has-a relationship (Aggregation)

- In Java, aggregation represents has-a relationship, which means when a class contains reference of another class known to have aggregation.
- In other wards, class A has-a relationship with class B, if class A has a reference to an instance of class B.
- The has-a relationship is based on usage, rather than inheritance.
- As an example, consider two classes Student and address. Each student has own address that makes has-a relationship.
- The main advantage of using aggregation is to maintain code re-usability.

Example: Aggregation

```
class Name
{
    String fname;
    String mname;
    String lname;
    Name(String fname, String mname, String lname)
    {
        this.fname = fname;
        this.mname = mname;
        this.lname = lname;
    }
}
```

```
class Student
{
    int age;
    Name n;
    Student(int age,Name n)
    {
        this.age = age;
        this.n = n;
    }
    public void display()
    {
        System.out.println("Student Name: "+n.fname+" "+n.mname+" "+n.lname);
        System.out.println("Age: "+age);
    }
}
```

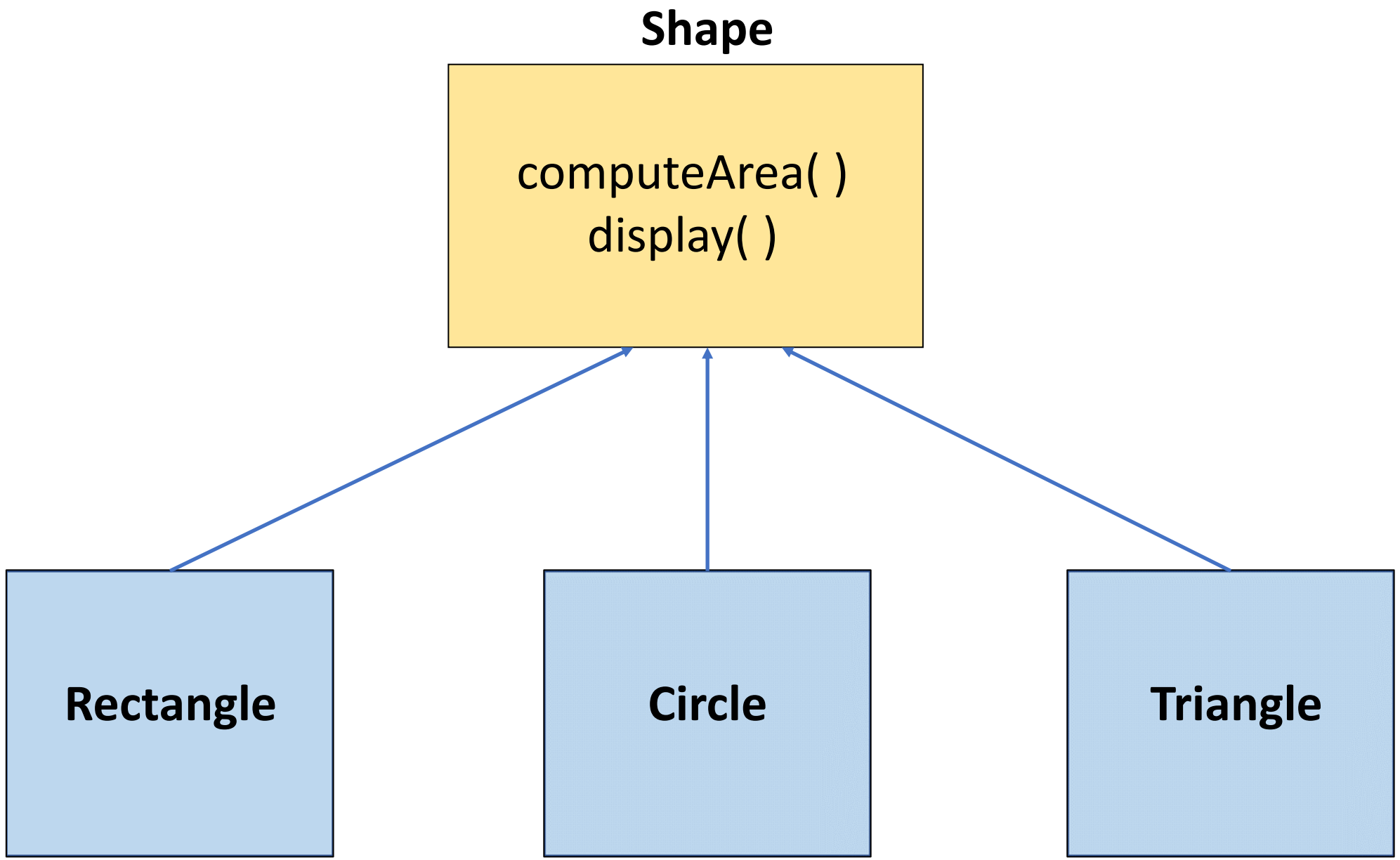
```
public class StudentInfo
{
    public static void main(String args[])
        {

            Name n = new Name("Ayan", "Kumar", "Pathak");
            Student s = new Student(20,n);
            s.display();
        }
}
```

Part-of relationship (Composition)

- Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.
- It represents part-of relationship.
- In composition, both entities are dependent on each other.
- When there is a composition between two entities, the composed object cannot exist without the other entity.
- For example: Engine is a part of Car, Books are part of Library, etc.

Abstract class concept

- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Abstraction lets you focus on what the object does instead of how does it.
- A class which is declared with the abstract keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (i.e., method with the body only without its definition).


Shape
computeArea( )
display( )
Rectangle
Circle
Triangle

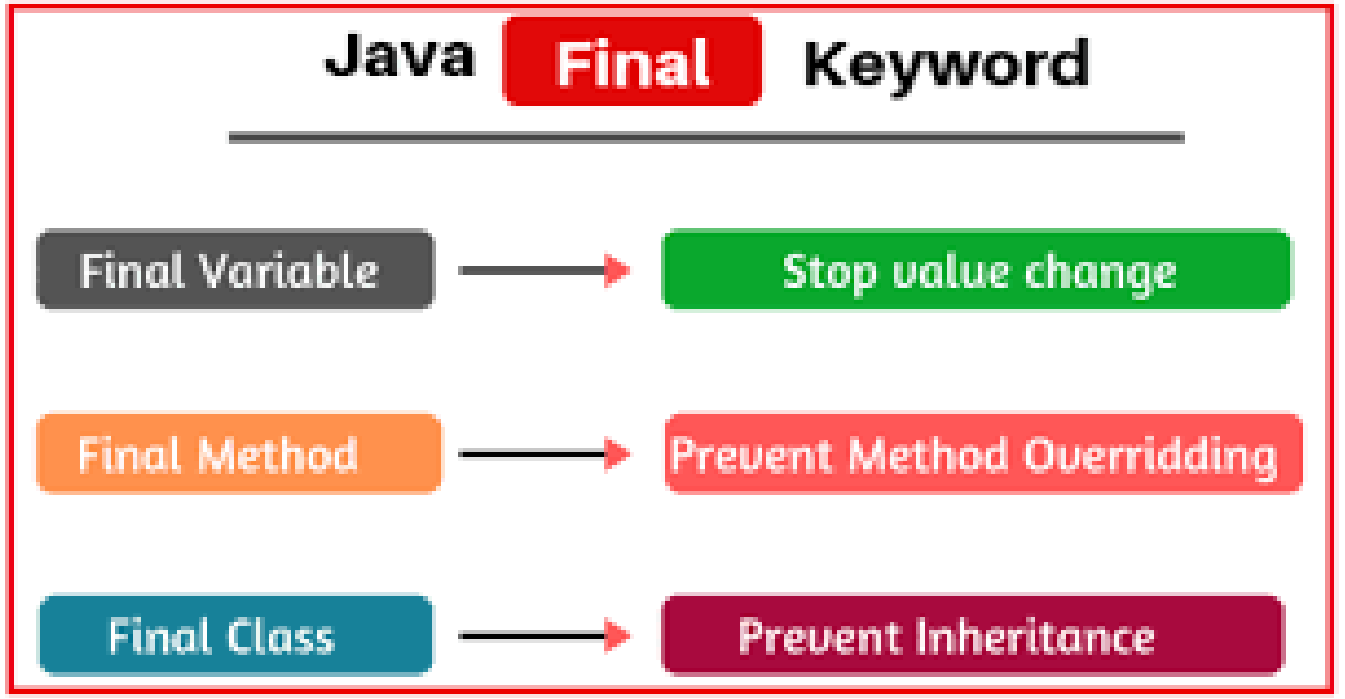
```
abstract class Bike
{
    abstract void run();
}

class Honda extends Bike
{
    void run()
    {
        System.out.println("Running Honda Byke");
    }
}
class AbstractClassDemo
{
    public static void main(String args[])
    {
        Honda obj = new Honda();
        obj.run();
    }
}
```

Abstraction important points

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the sub class not to change the body of the method.

final keyword concept

- The final keyword in Java is used to restrict the access of an item from its super class to a sub class. The Java final keyword can be used in many context in the program as follows.
- Variable: a variable cannot be changed in sub class.
- Method: a method cannot called from a sub class object.
- Class: a class cannot be extended to sub class.



Java final keyword

```
class A {
final int a;
void f(final int b) {
        a=2;b=5;
}}
```

final Variable
Can't be Modified

```
final class A{}
class B extends A{}
```

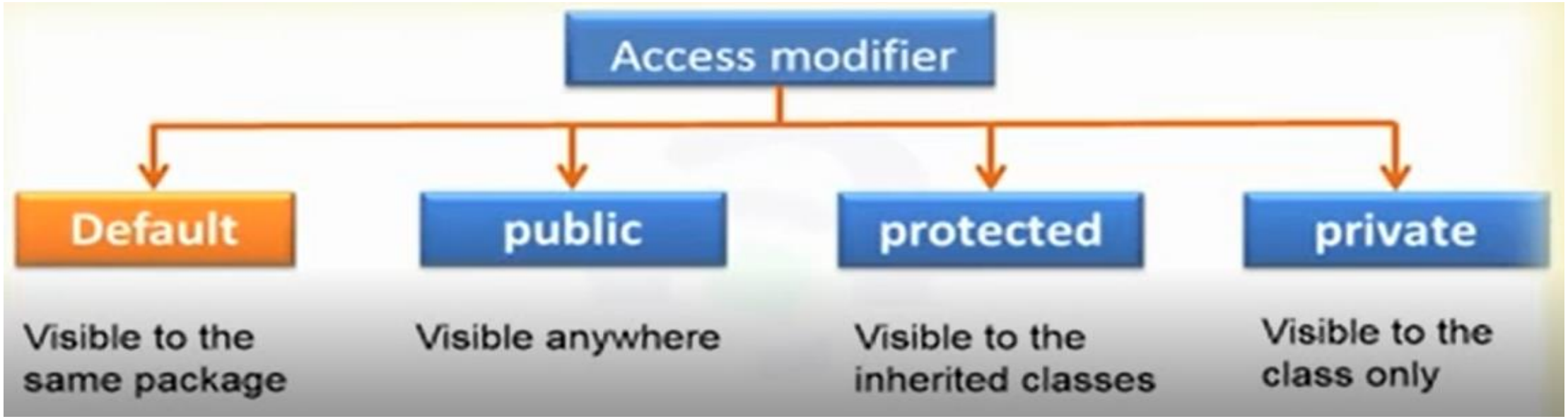
final class
Can't be Extended

```
class A{
        final void f() {}
}
class B extends A{
        void f() {}
}
```

final method
Can't be Overridden

Access Modifiers in Java

- The access modifiers in Java specify accessibility (scope) of a data member, method, constructor or class.
- We can change the access level of data members, constructors, methods, and class by applying the access modifier on it.
- There are four types of Java access modifiers such as default, public, protected, and private.



1. Default: The access level of a default modifier is only within the package. It cannot be assessed from outside the package. If you do not specify any access level, it will be the default.
2. Public: The access level of the public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.
3. Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Access levels of modifiers

Access levels / Modifier	Class	Package	Subclass	Everywhere
Default	Y	Y	N	N
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
Private	Y	N	N	N

Object class and its methods

- The Object class is the parent class of all the classes in Java by default.
- In other words, it is the topmost class of Java.
- The object class has been defined in java.lang package.
- If no inheritance is specified when a class is defined, the superclass of the class is object by default.
- Object class acts as a root of inheritance hierarchy in any Java Program.
- Hence, Object class methods are available to all Java classes.

- There are methods in the Object class:

1. toString(): The toString() provides a string representation of an object and is used to convert an object to String.
2. For example, A a = new A();
3. System.out.println(a.toString());
4. Invoking toString() on an object returns a string that describes the object.
5. By default, it returns a string consisting of a class name of which the object is an instance, an at sign (@), and the object's memory address in hexadecimal.
6. It is always recommended to override the toString() method to get our own String representation of Object.