OOP
Object-oriented programming

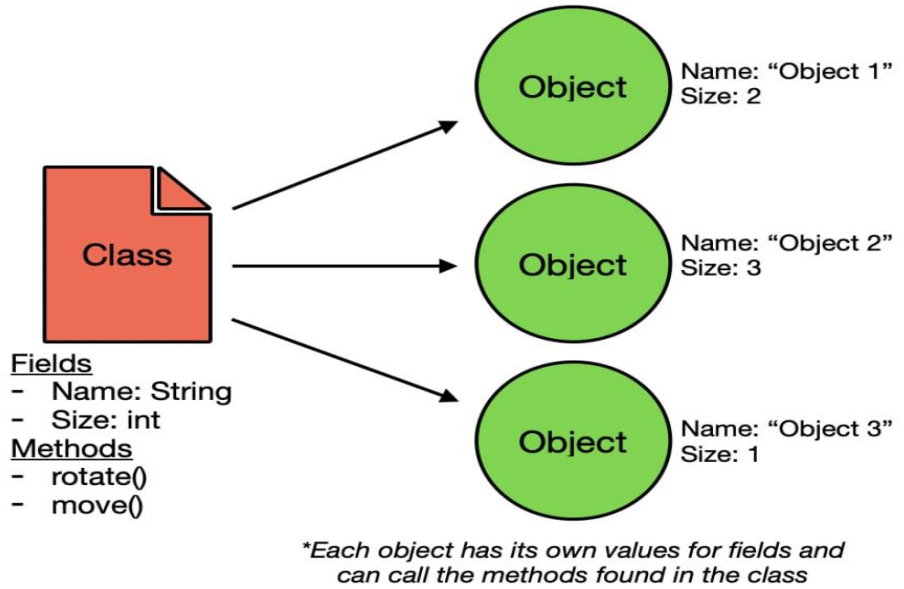
Object Oriented Programming in Java

Created By : Gayathri ramadurgum

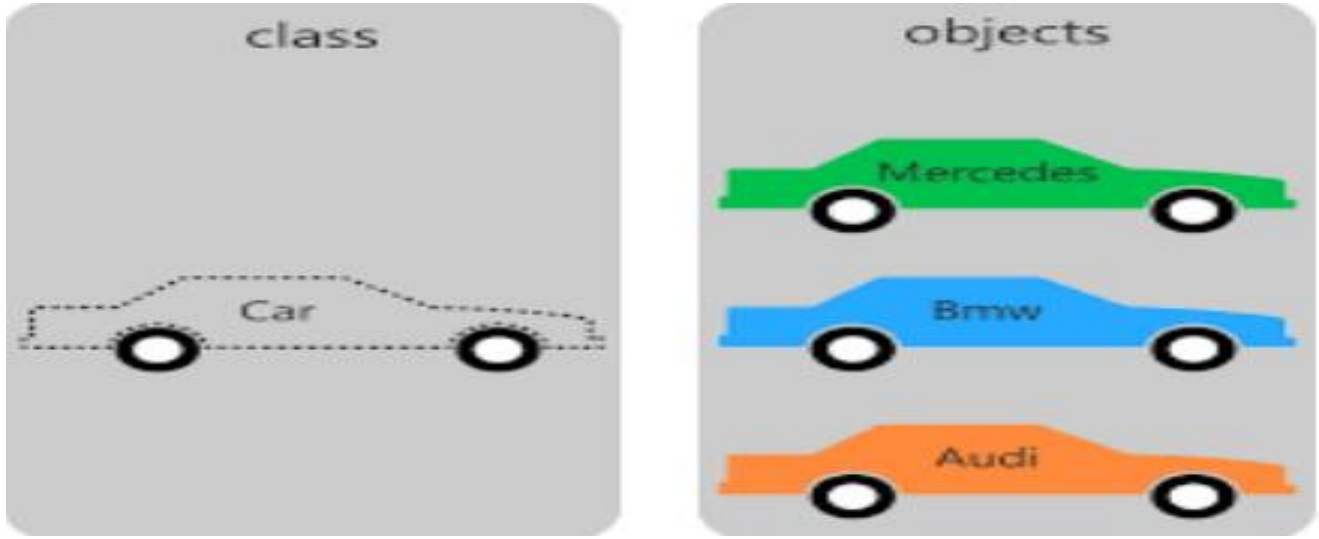
Date : 14 September, 2023

INTRODUCTION

- **Object Oriented** is an approach to software development that models application around real-world objects. A class defines the properties and methods of a real-world object. An object is an instance of a class.
- A **class** is an user defined blueprint or prototype from which objects are created. It represents the set of properties that are common to all objects of one type.
- An **object** has a state (represents the value of the object) and some behavior (represents the functionality of the object)



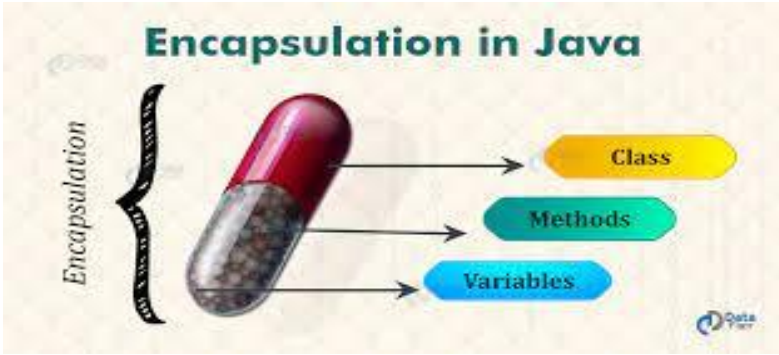
EXAMPLE



- In the above example, the class ‘Car’ would define properties of a car such as color,price,type,etc. and it would also define the methods of a car like driving(), breaks(), accelerating(),etc.
- BMW, Audi, and Mercedes are the instances of the class

ENCAPSULATION

- Encapsulation in Java is a process of wrapping code and data together into a single unit. We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.



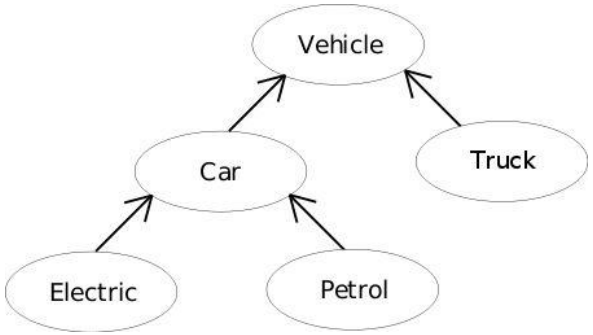
- Encapsulation provides :
  - ✓Control over the data
  - ✓Data hiding since other classes will not be able to access private data members
  - ✓Makes testing of code easier

```java
public class Student {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
public class AnotherClass {
    public static void main(String[] args) {
        // Create an instance of the Student class
        Student student = new Student();

        // Use the setter method to set the name
        student.setName("John Doe");

        // Use the getter method to retrieve the name
        String studentName = student.getName();

        // Print the name
        System.out.println("Student name: " + studentName);
    }
}
```

INHERITANCE

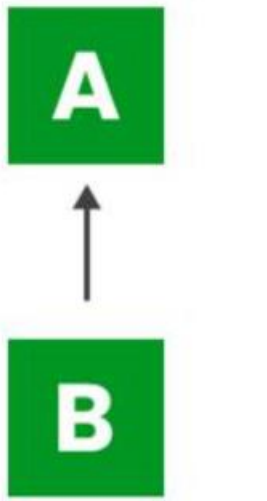
- Inheritance is a mechanism of driving a new class from an existing class. The existing (old) class is known as base class or super class or parent class. A class whose properties are inherited is known as parent class and a class that inherits the properties of the parent class is known as child class.
- Points to remember :
  - Constructor cannot be inherited in Java.
  - Private members do not get inherited in Java.
  - Multiple Inheritance is not supported in Java (But it can be achieved through the help of interfaces)



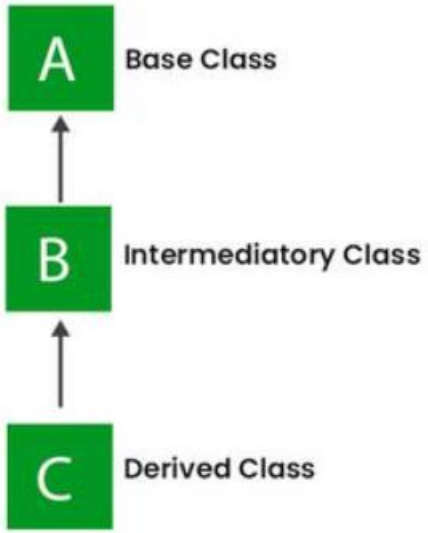
In java inheritance can be achieved using the ‘extends’ keyword as shown below :

```java
class Animal {
    void eat() {
        System.out.println("The animal eats.");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

public class InheritanceExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.bark();
    }
}
```

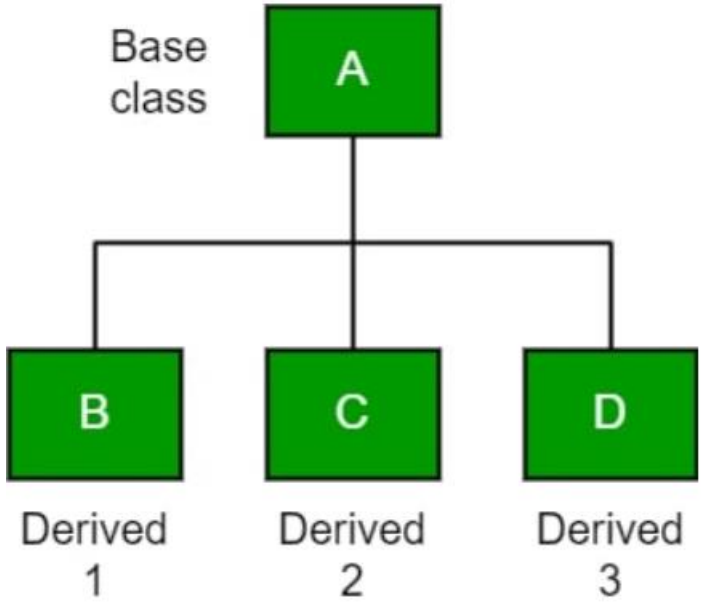
TYPES OF INHERITANCE



Single Inheritance

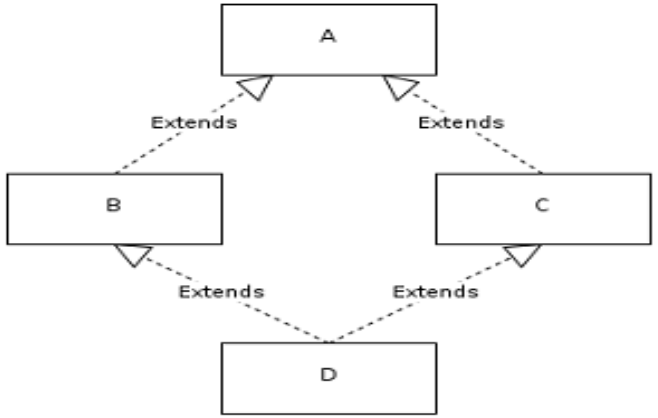


Multilevel Inheritance



Why multiple inheritance is not supported?

- One of the primary reasons for avoiding multiple inheritance is the "diamond problem.“



Here suppose class A has a method sum(). Now since class B and class C extend A, suppose they have overridden the sum() methods in their respective classes. If class D was to extend both B and C then there would exist an ambiguity which sum() method to call.

POLYMORPHISM

- Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways.
- Types of Polymorphism :
  - Compile Time Polymorphism
  - Run Time Polymorphism
- Compile-time polymorphism can be achieved by method overloading, and Runtime polymorphism can be achieved by method overriding.

METHOD OVERLOADING

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading. Method overloading increases the readability of the program. One way to overload your methods is by changing the number of parameter that the method accepts and another way to overload your methods is by changing the data types of the parameters:

```
class Adder {
    static int add(int a, int b) {
        return a + b;
    }

    static int add(int a, int b, int c) {
        return a + b + c;
    }
}

class TestOverloading1 {
    public static void main(String[] args) {
        System.out.println(Adder.add(11, 11));
        System.out.println(Adder.add(11, 11, 11));
    }
}
```

```
class Adder {
    static int add(int a, int b) {
        return a + b;
    }

    static double add(double a, double b) {
        return a + b;
    }
}

class TestOverloading2 {
    public static void main(String[] args) {
        System.out.println(Adder.add(11, 11));
        System.out.println(Adder.add(12.3, 12.6));
    }
}
```

METHOD OVERRIDING

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java. In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.
- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.

```
class Vehicle {
    void run() {
        System.out.println("Vehicle is running");
    }
}

class Bike2 extends Vehicle {
    void run() {
        System.out.println("Bike is running safely");
    }

    public static void main(String args[]) {
        Bike2 obj = new Bike2();
        obj.run();
    }
}
```

Overloading Vs. Overriding

Method Overloading:

- Method overloading occurs within the same class
- It involves defining multiple methods in the same class with the same name but different parameters (different number, types, or order of parameters).
- Overloaded methods must have the same name but different parameter lists.
- The return type of the overloaded methods may or may not be the same.
- Method overloading is determined at compile time

Method Overriding:

- Method overriding occurs in a superclass-subclass relationship (inheritance).
- It involves defining a method in a subclass with the same name, return type, and parameter list as a method in the superclass.
- Overridden methods must have the same name, return type, and parameter list (method signature).
- Method overriding is used to provide a specific implementation of a method in the subclass, replacing the implementation in the superclass.
- Method overriding is determined at runtime