Java Programs

Lecture 5

By
Dr. Radwa Rady & Dr. Ghada Fathy

Objectives

- **Overview for Main OOP Concepts:**
  - ✓ **Class.**
  - ✓ **Object.**
  - ✓ **Encapsulation.**
  - ✓ **Inheritance.**
  - ✓ **Polymorphism.**
  - ✓ **Abstraction.**

Object Oriented Programming) Concepts

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects that contain data (fields) and behavior (methods). It focuses on designing software that closely represents real-world entities. It is used to:

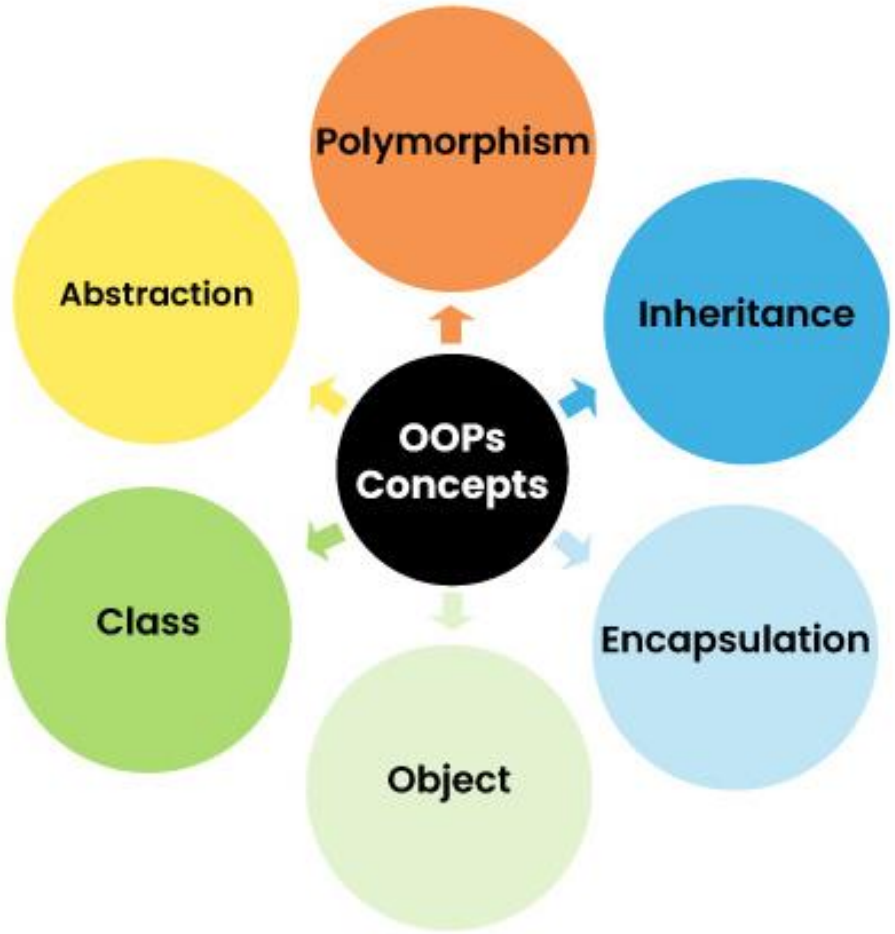
- Improves code reusability
- Enhances maintainability and scalability
- Makes programs easier to understand and manage
- Closely models real-world entities

Object-Oriented Programming (OOP)

Object-oriented Programming (OOPs) means we organize our software as a combination of different types of objects that incorporate both data and behavior.

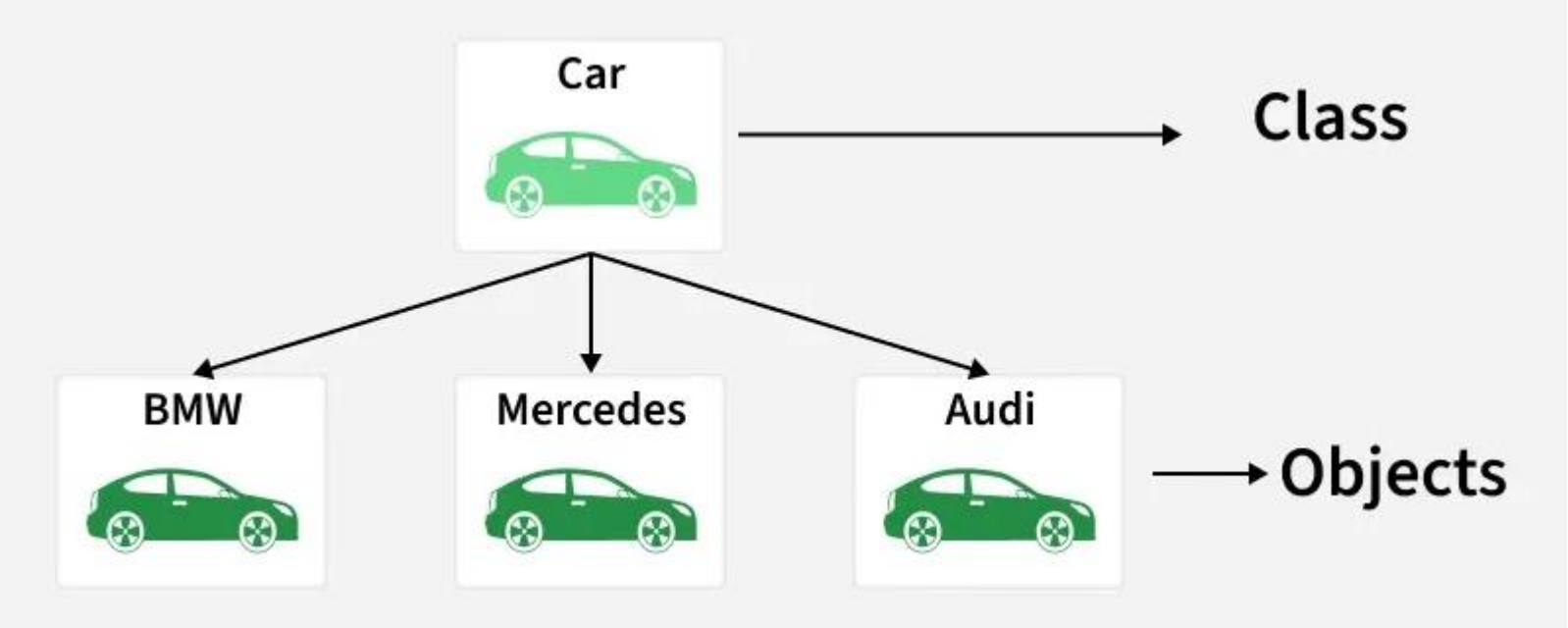
OOPs is a methodology that simplifies software development and maintenance by providing some rules.

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



1- Class

- A class is a blueprint or template for creating objects. It defines the properties (fields) and behaviors (methods) that the objects created from the class can have. In Java, a class is defined using the class keyword.
- A Car represents a class (blueprint), while BMW, Mercedes, and Audi represent objects (instances) created from that class.



Class in Java

```java
public class Car {

        // Fields (properties)
        String color;
        String model;

        // Methods (behaviors)
        void drive() {
            System.out.println("The car is driving.");
        }

        void stop() {
            System.out.println("The car has stopped.");
        }

}
```

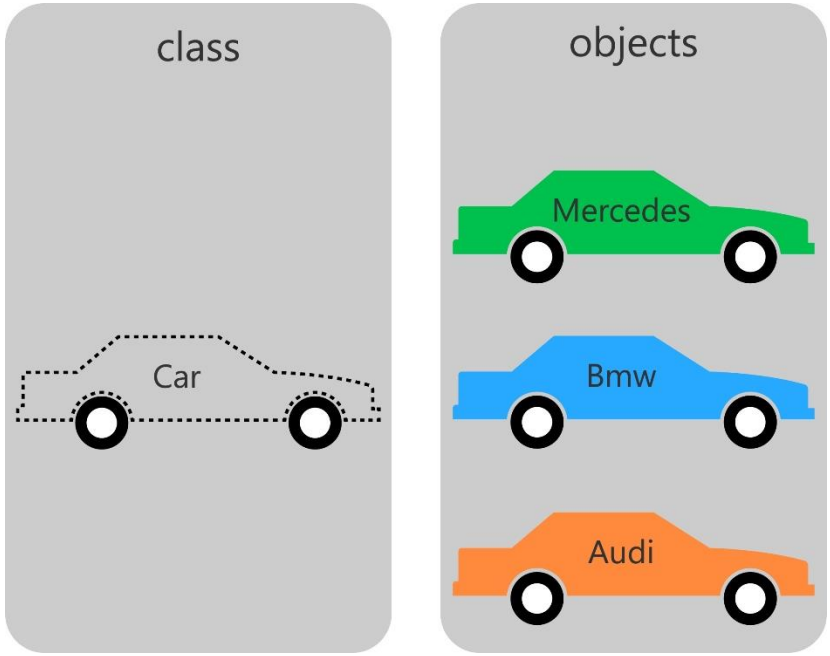
2- Object

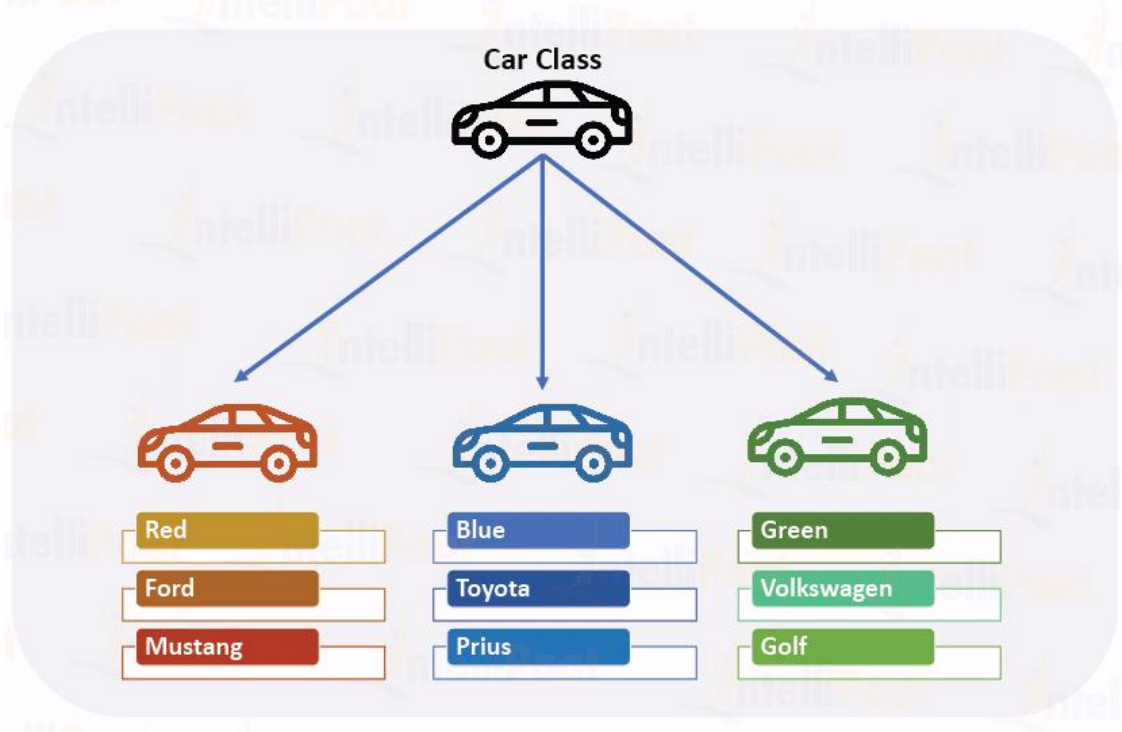
- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc.

- An object is an instance of a class. It is created from a class and has its own set of values for the properties defined in the class. In Java, an object is created using the new keyword.

What are Classes and Objects?

- **Class** is a template for objects, and an **object** is an instance of a class.
- When the individual **objects** are created, they inherit all the variables and methods from the **class**.





Object in Java

```java
public class Main {
    public static void main(String[] args) {
        // Creating an object of the Car class
        Car myCar = new Car();
        myCar.color = "Red";
        myCar.model = "Tesla Model S";

        // Calling methods on the object
        myCar.drive(); // Output: The car is driving.
        myCar.stop();  // Output: The car has stopped.

        System.out.println("Car Model: " + myCar.model); // Output: Car Model: Tesla Model S
        System.out.println("Car Color: " + myCar.color); // Output: Car Color: Red
    }

}
```

Example: Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other.

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

```
// Output
5
25
```

Example: Multiple Attributes

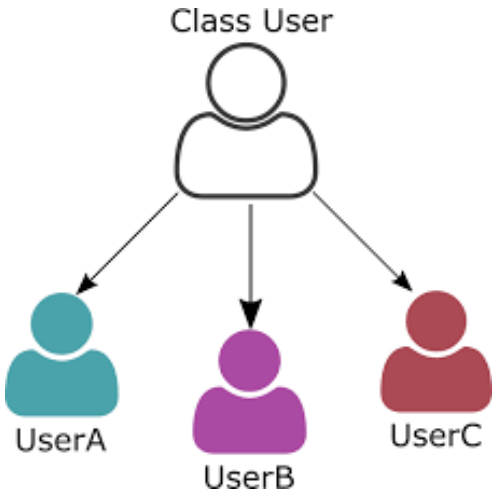
You can specify as many attributes as you want.

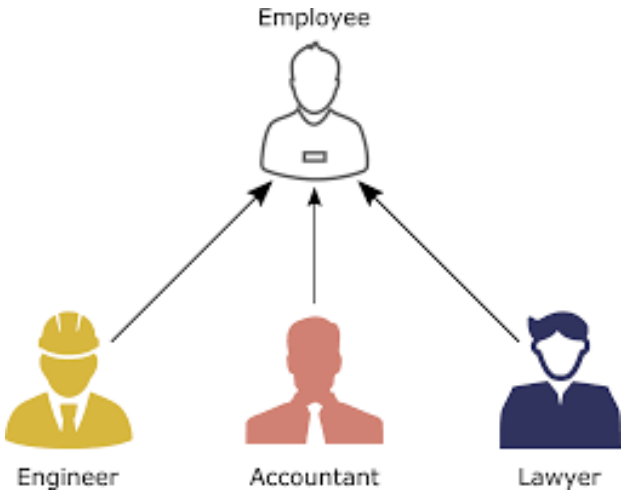
```
public class Main {
  String fname = "John";
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

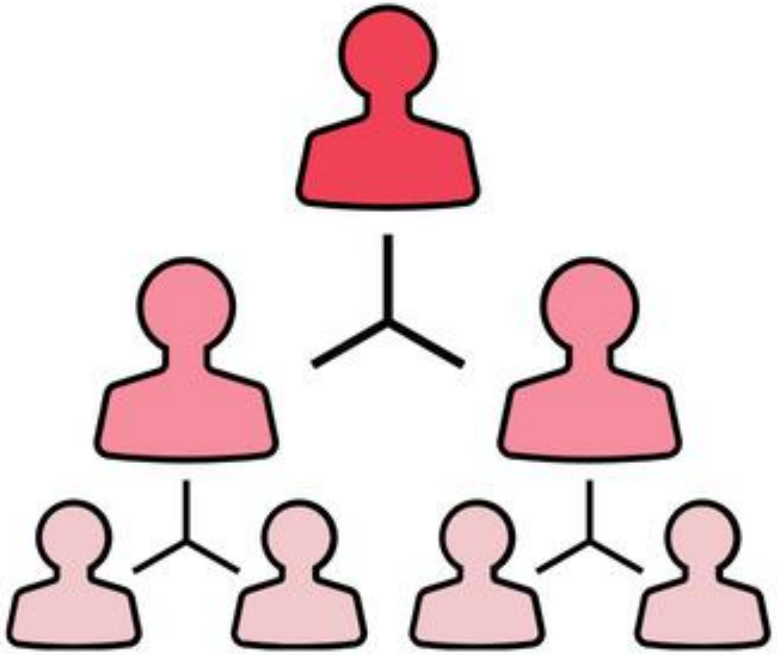
```
//Output
Name: John Doe
Age: 24
```

Inheritance

- Inheritance is a mechanism wherein a new class (subclass) inherits the properties and behavior (methods) of an existing class (superclass). This allows for hierarchical classification and promotes code reusability.
- When one **object acquires** all the properties and behaviors of a **parent object**, it is known as **inheritance**.

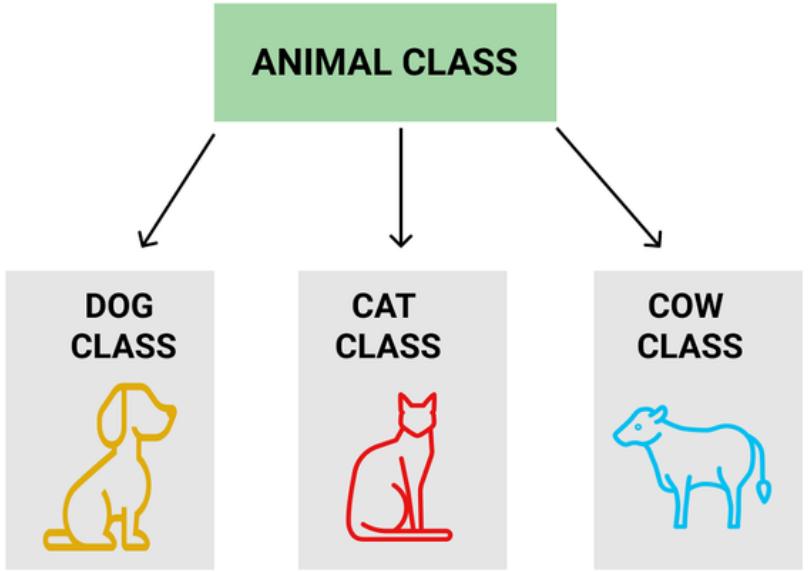






Key Benefits of Inheritance

- **Code Reusability:** Inherited methods and fields from the parent class can be reused in the child class.
- **Method Overriding:** Child classes can provide specific implementations of methods that are already defined in the parent class.
- **Extensibility:** New functionality can be added to existing classes without modifying them.

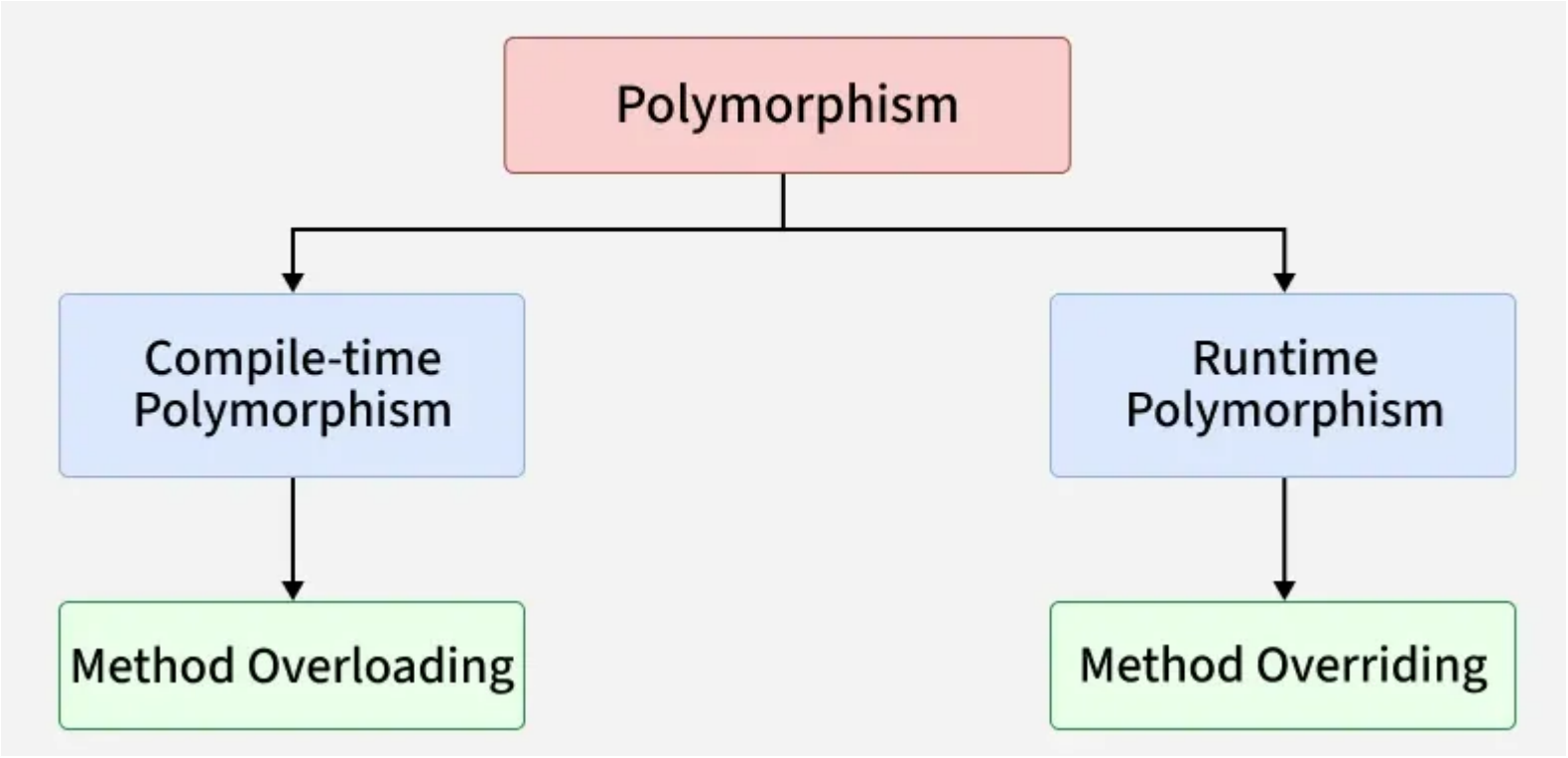


Inheritance in java

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

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

public class TestInheritance {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat(); // Inherited method from Animal
        dog.bark(); // Method of Dog
    }

}
```

Polymorphism

- The word polymorphism means having many forms.
- Polymorphism allows one interface to be used for a general class of actions.
- Polymorphism can be broadly classified into two types:



Compile-time Polymorphism

1. Compile-time Polymorphism(Method Overloading) : Achieved when multiple methods have the same name but different parameters. The method call is resolved at compile time.

```
class Product{

    public int prod(int a, int b, int c){
        return a * b * c;

    }
    public double prod(double a, double b, double c){
        return a * b * c;
    }
}

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

        Product p = new Product();
        System.out.println(p.prod(1, 2, 3));
        System.out.println(p.prod(1.0, 2.0, 3.0));
    }
}
```

Run-time Polymorphism

2. Runtime Polymorphism (Method Overriding): Achieved when a subclass provides a specific implementation of a method already defined in its superclass. The method call is resolved at runtime based on the object .

```
class Animal {

    void move(){
        System.out.println(
      "Animal is moving.");

    }
    void eat(){

        System.out.println(
      "Animal is eating.");

    }
}
```

```
class Dog extends Animal{

    @Override void move(){

      // move method from Base class is overriden
      // method
        System.out.println("Dog is running.");
    }
    void bark(){

        System.out.println("Dog is barking.");

    }
}
```

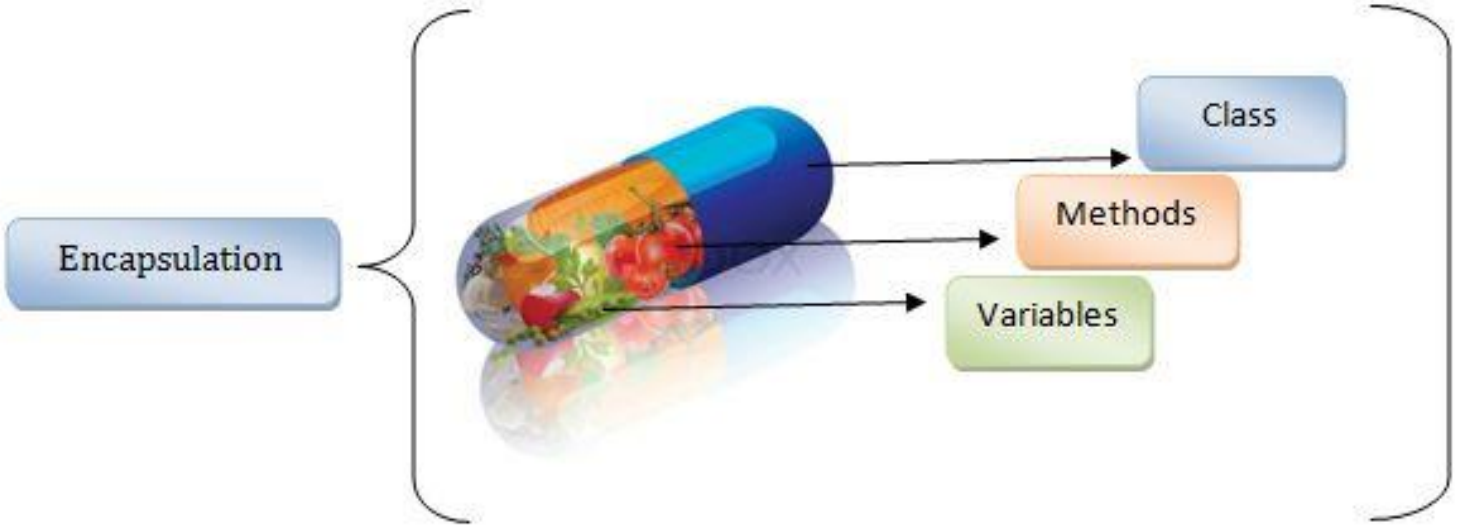
```
public class Geeks {
    public static void main(String[] args)
    {
        Dog d = new Dog();
        d.move();
        d.eat();
        d.bark();
    }
}
```

Output

```
Dog is running.
Animal is eating.
Dog is barking.
```

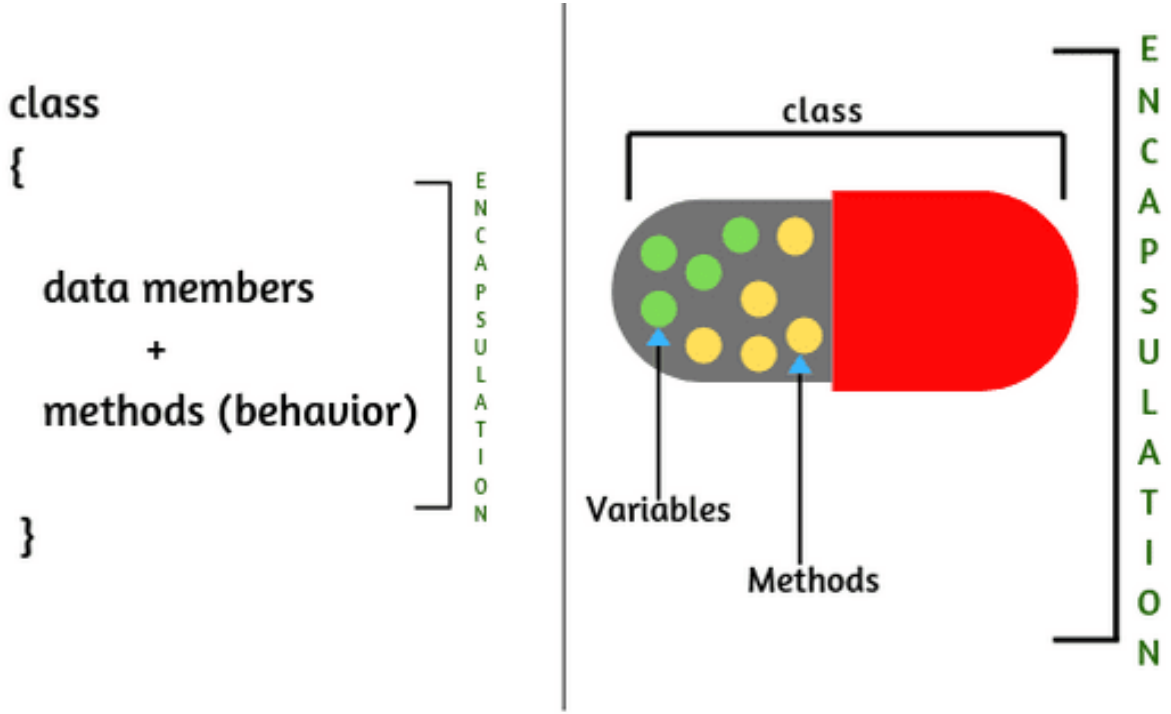
3- Encapsulation

- **Binding** (or wrapping) code and data together into a single unit are known as **encapsulation**.
- For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation.
- Java is fully encapsulated class because all the data members are private here.



Encapsulation

In Encapsulation, the variables or data of a class are hidden from any other class and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.



Encapsulation in java

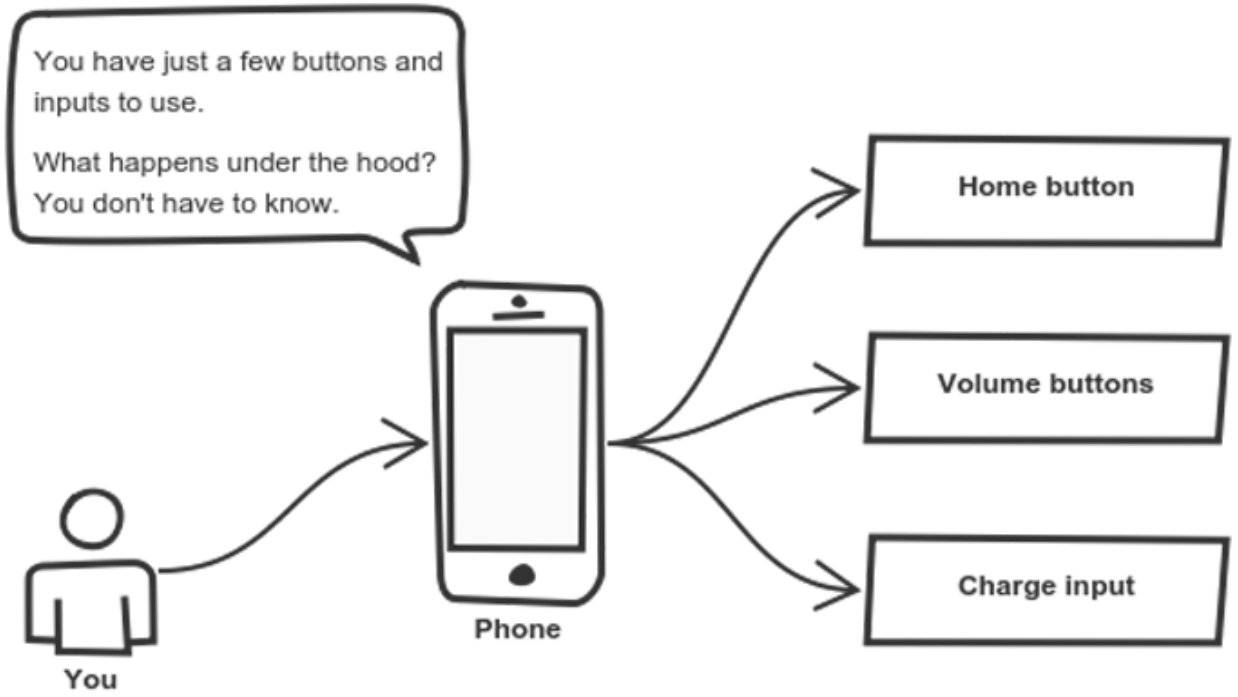
```java
public class Student {
    private String name;
    private int age;

    // Getter method for name
    public String getName() {
        return name;
    }

    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        if (age > 0) {
            this.age = age;
        }
    }
}
```

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

        Student student = new Student();
        student.setName("John");
        student.setAge(21);

        System.out.println("Name: " + student.getName());
        System.out.println("Age: " + student.getAge());

    }

}
```

Note: The name and age variables of the Student class are private and can only be accessed through the public getter and setter methods. This ensures that the internal state of the object is protected and can only be modified through controlled access points.

Abstraction

- Abstraction is the concept of hiding the complex implementation details and showing only the essential features of the object.
- For example, phone call, we don't know the internal processing.
- We use **abstract class** and **interface** to achieve abstraction.(hiding implementation and showing only functionality). I cant create an object from abstract class.



Abstract Class

```
abstract class Animal {

    abstract void sound();   // abstract method

    void eat() {             // normal method
        System.out.println("Animal eats");
    }
}

class Dog extends Animal {

    void sound() {
        System.out.println("Dog barks");
    }
}
```

An abstract class can have both abstract methods (methods without a body) and concrete methods (methods with a body).

Interface Class

```java
interface Animal {

    void sound();   // abstract by default
}

class Dog implements Animal {

    public void sound() {
        System.out.println("Dog barks");
    }
}
```

The Interface class have abstract methods by default (methods without a body). Using implements not extends

Difference Between Abstract Class and Interface

Feature	Abstract Class	Interface
Keyword	abstract class	interface
Methods	Can have **abstract and regular methods**	Methods are **abstract by default** (traditionally)
Variables	Can have **instance variables**	Variables are **public static final** (constants)
Constructors	✓ Allowed	✗ Not allowed
Access Modifiers	Can use private, protected, public	Methods are usually public
Inheritance	A class can extend **only one abstract class**	A class can implement **multiple interfaces**

Thanks

References: https://www.w3schools.com