SNS COLLEGE OF ENGINEERING

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

COURSE NAME : 23CSB101 & Object Oriented Programming

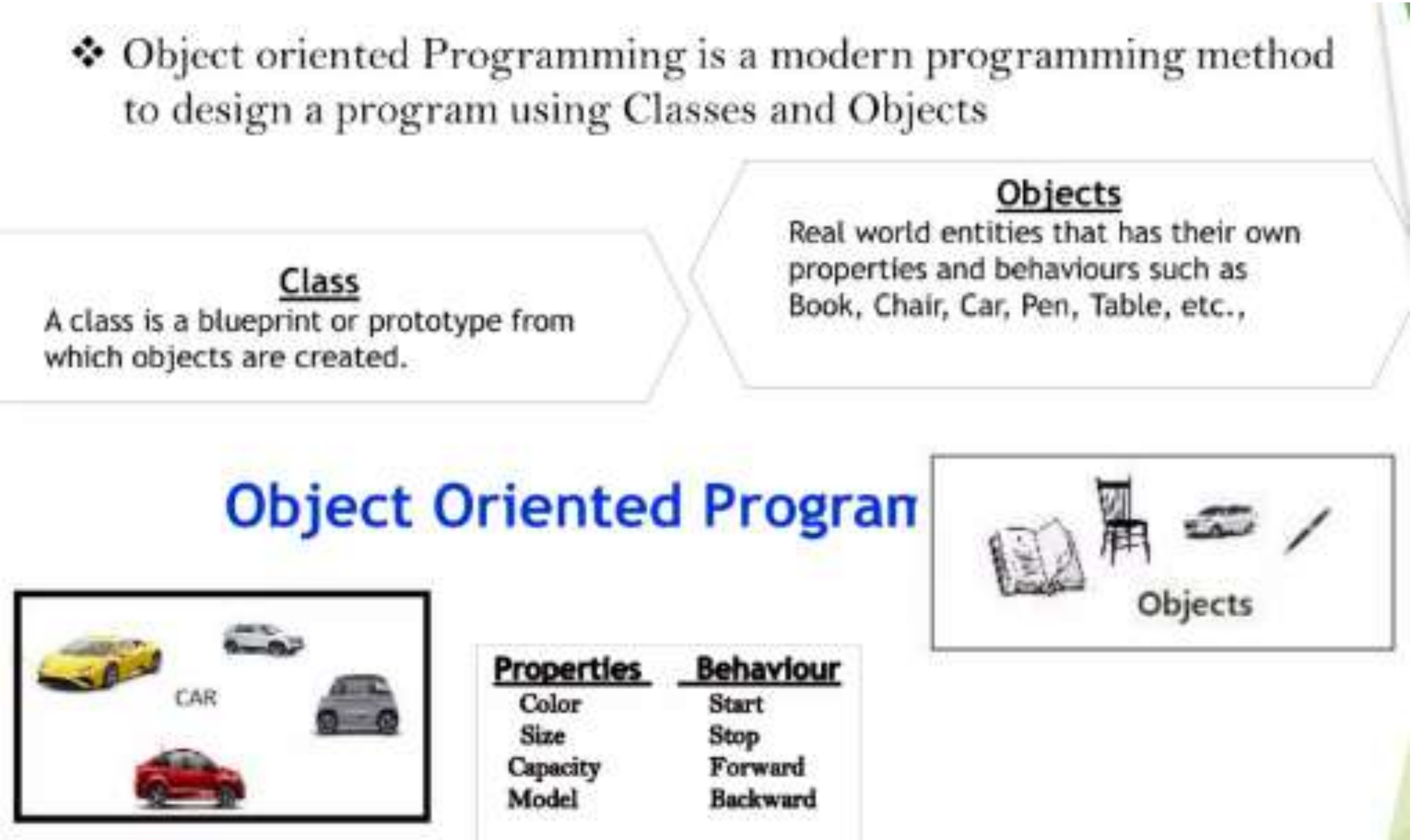
I YEAR/ II SEMESTER

UNIT – I INTRODUCTION TO OOP

Topic: Object Oriented Programming Concepts

Dr.P.Poonkodi
Assistant Professor(SG)
Department of Computer Science and Technology

OOP Concepts



❖ Object oriented Programming is a modern programming method to design a program using Classes and Objects

Class
A class is a blueprint or prototype from which objects are created.

Objects
Real world entities that has their own properties and behaviours such as Book, Chair, Car, Pen, Table, etc.,

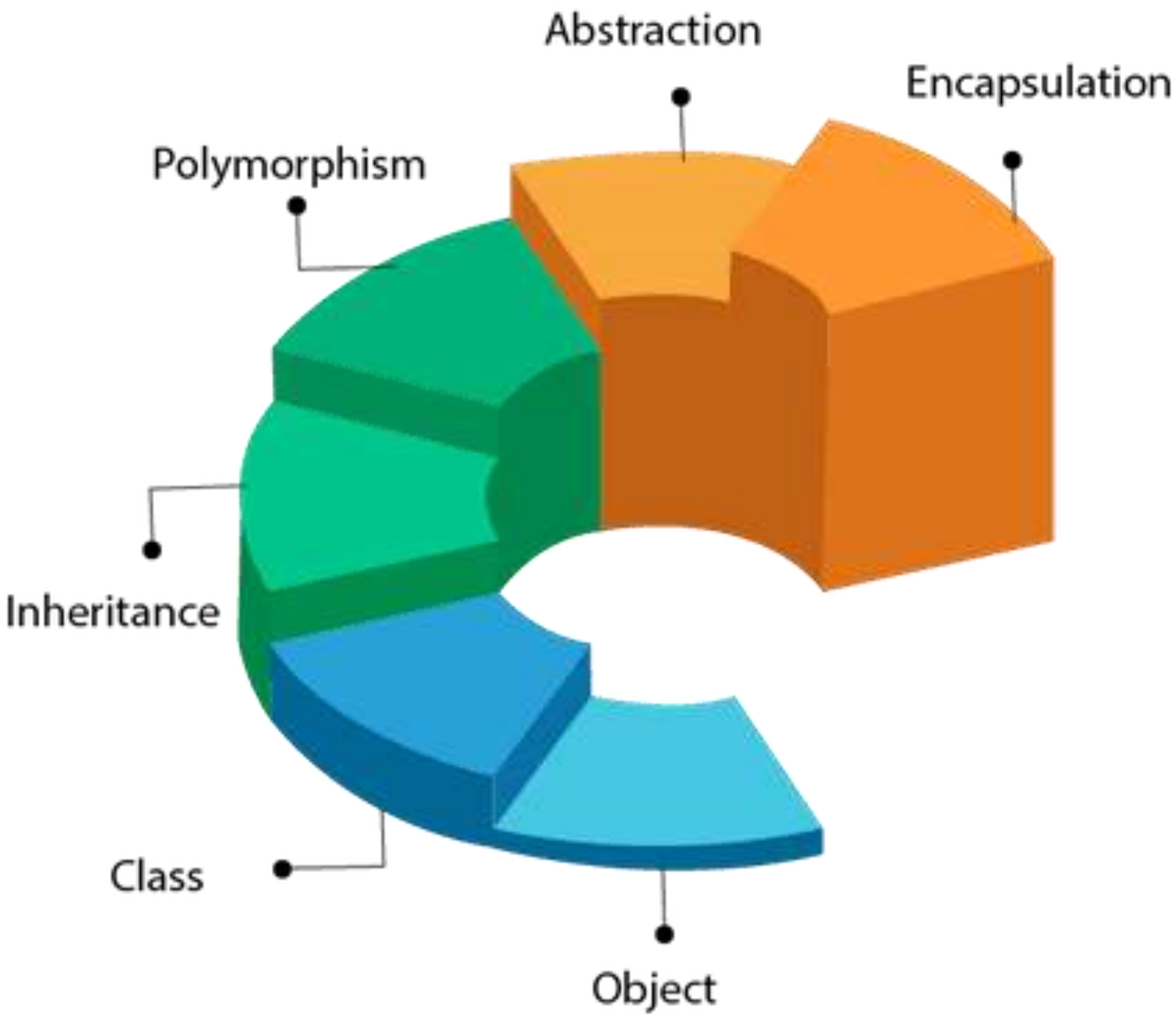
Object Oriented Progran

Properties	Behaviour
Color	Start
Size	Stop
Capacity	Forward
Model	Backward

OOP Concepts

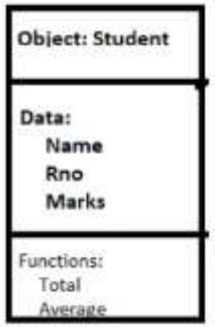
- Concepts (generic programming)
  - a description of supported operations on a type, including syntax and semantics
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts

OOP Concepts



Object

- Any entity that has state and behavior is known as an object. For example


Object: Student

Data:
Name
Rno
Marks

Functions:
Total
Average

Syntax:

Class-name object-name = new class-name();

Example :

Box mybox=new Box();

- Example : A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

- Collection of objects is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

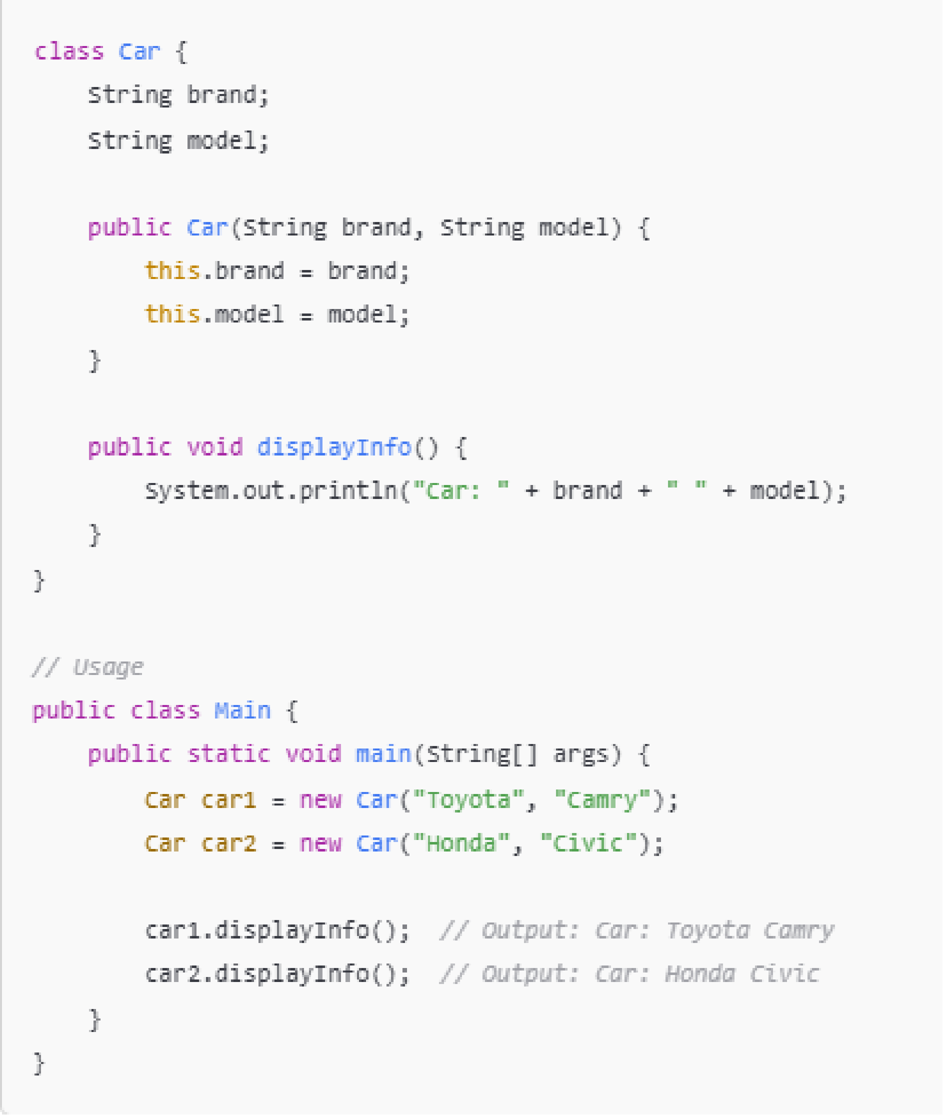
Syntax:

```
Class class-name
{
            Members;
}
```

Example:

```
Class Box
{
     double Height;
     double width;
}
```

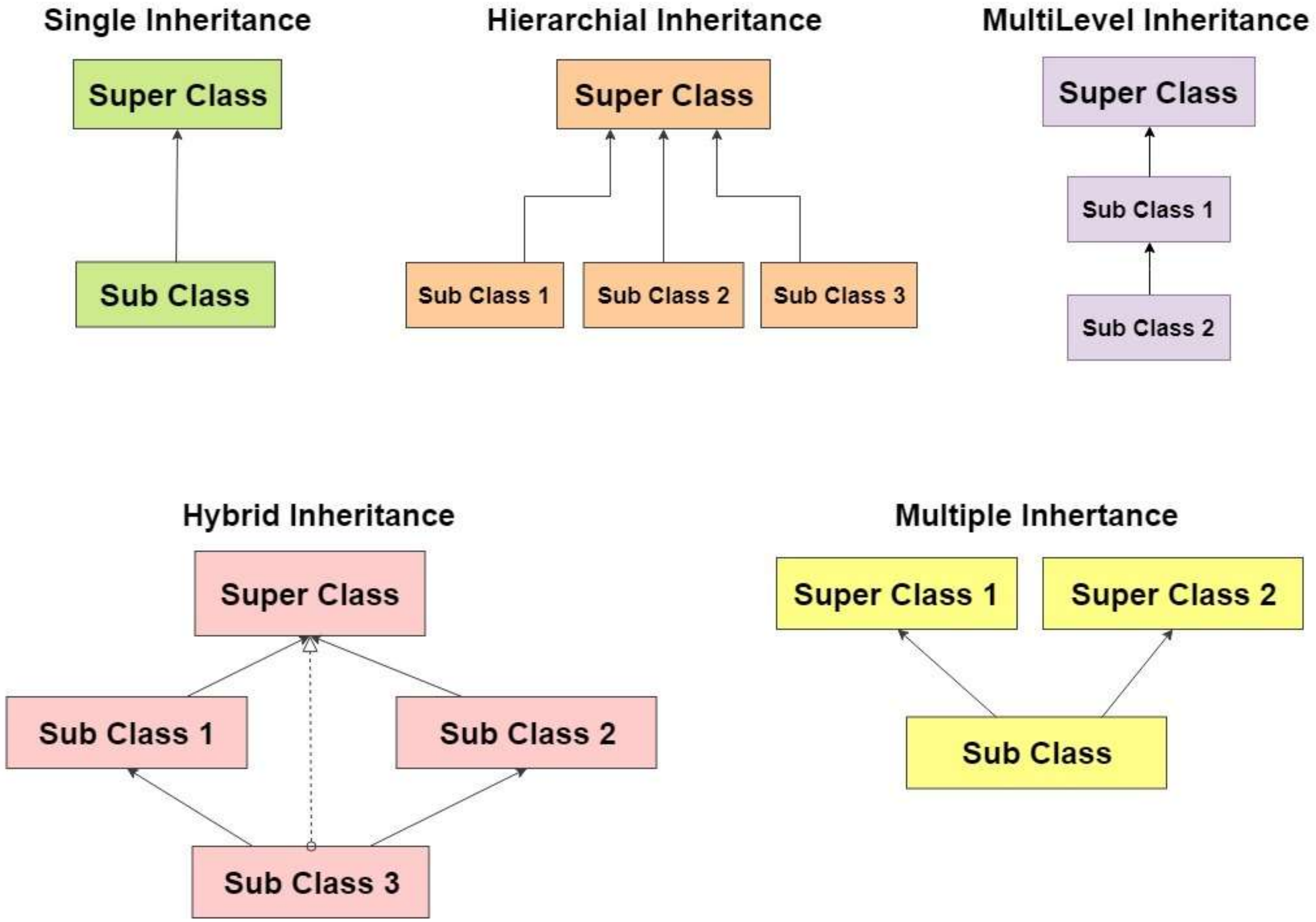
Example



```java
class Car {
    String brand;
    String model;

    public Car(String brand, String model) {
        this.brand = brand;
        this.model = model;
    }

    public void displayInfo() {
        System.out.println("Car: " + brand + " " + model);
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Car car1 = new Car("Toyota", "Camry");
        Car car2 = new Car("Honda", "Civic");

        car1.displayInfo();  // Output: Car: Toyota Camry
        car2.displayInfo();  // Output: Car: Honda Civic
    }
}
```

Inheritance

- a mechanism in which one class acquires the property of another class
- Types of inheritance



Single Inheritance

Super Class

Sub Class

Hierarchial Inheritance

Super Class

Sub Class 1

Sub Class 2

Sub Class 3

MultiLevel Inheritance

Super Class

Sub Class 1

Sub Class 2

Hybrid Inheritance

Super Class

Sub Class 1

Sub Class 2

Sub Class 3

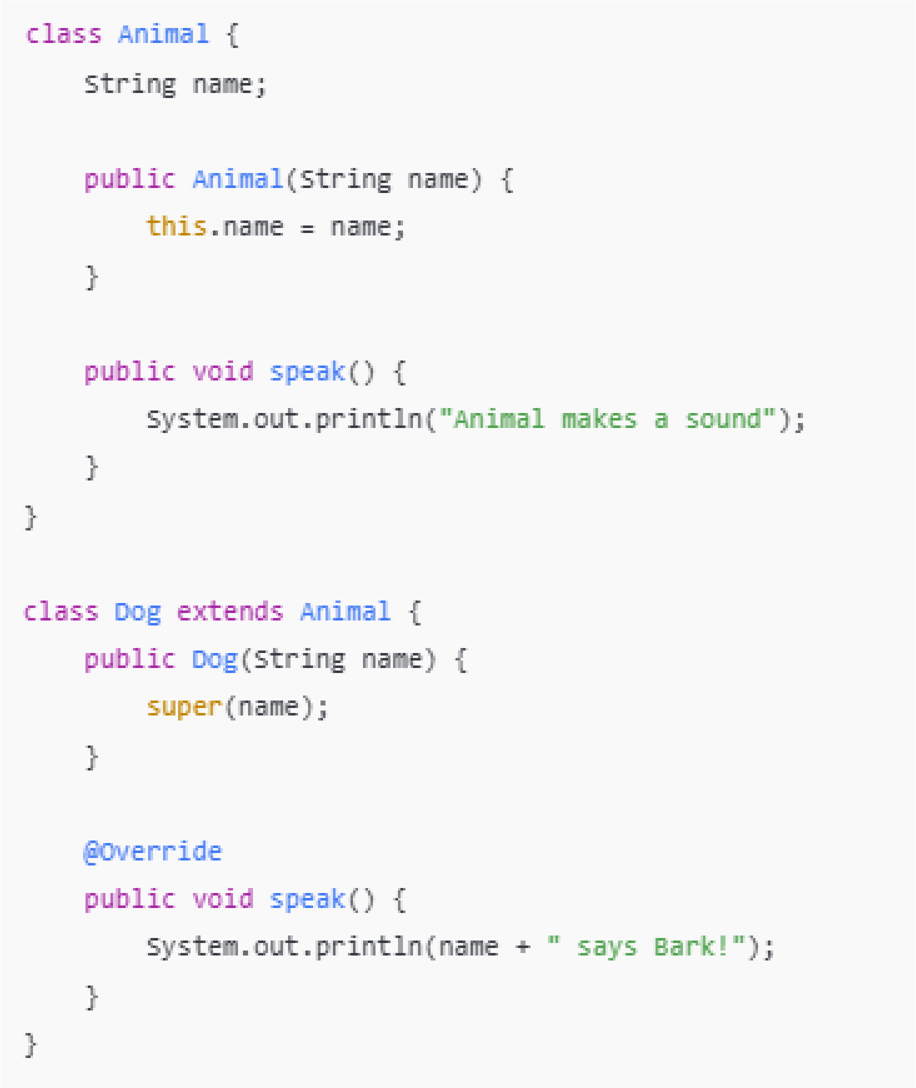
Multiple Inhertance

Super Class 1

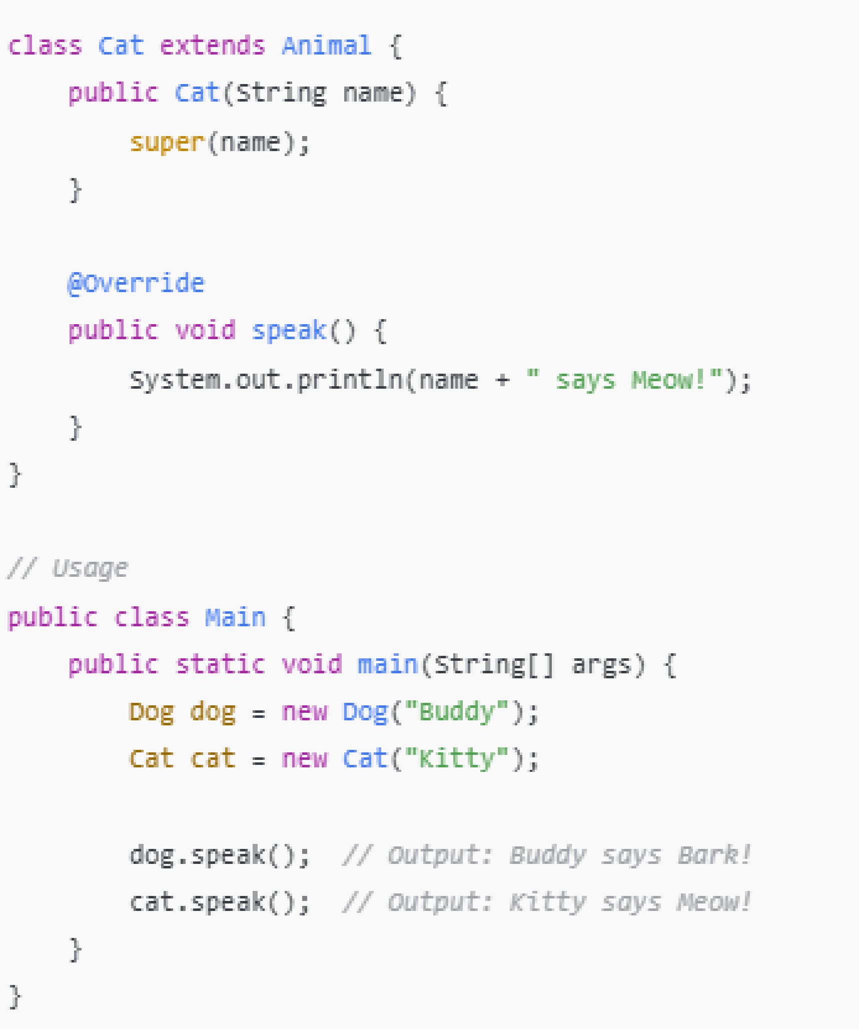
Super Class 2

Sub Class

Example



```java
class Animal {
    String name;

    public Animal(String name) {
        this.name = name;
    }

    public void speak() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }

    @Override
    public void speak() {
        System.out.println(name + " says Bark!");
    }
}
```

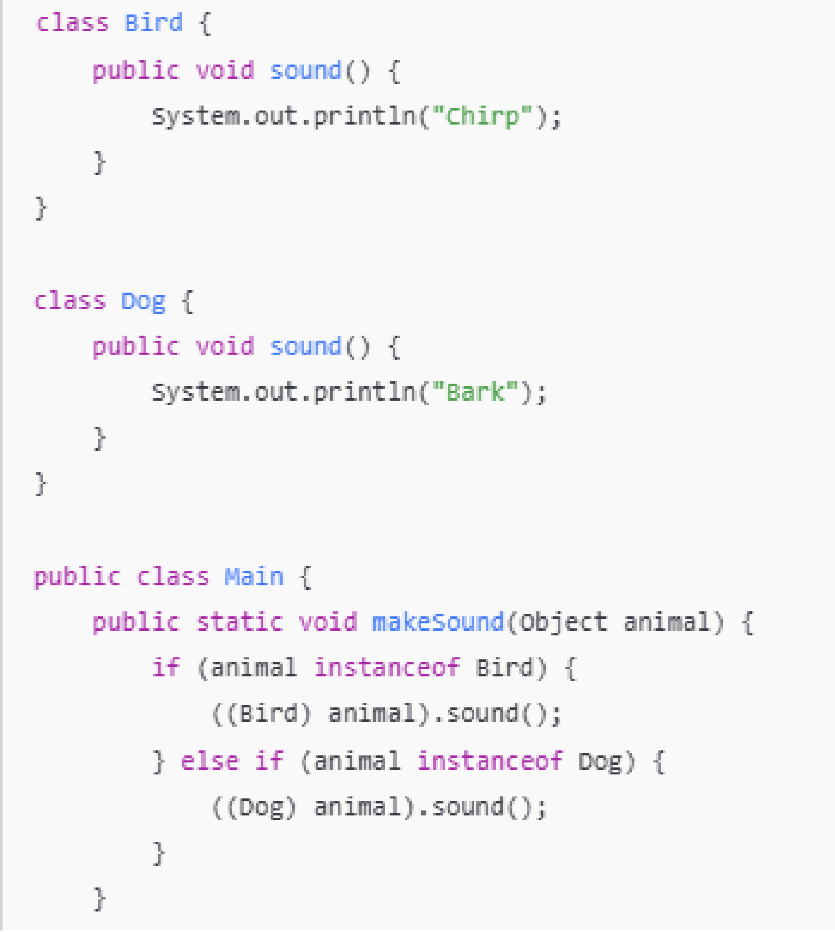


```java
class Cat extends Animal {
    public Cat(String name) {
        super(name);
    }

    @Override
    public void speak() {
        System.out.println(name + " says Meow!");
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog("Buddy");
        Cat cat = new Cat("Kitty");

        dog.speak();  // Output: Buddy says Bark!
        cat.speak();  // Output: Kitty says Meow!
    }
}
```

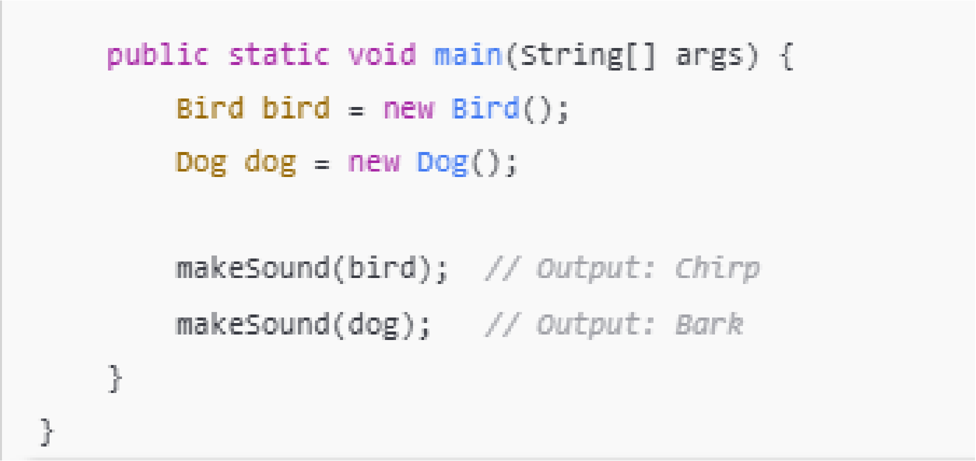
Polymorphism

- one task is performed in different ways
- Derived from two different words
  - Poly - many
  - Morphs – forms
- Two types
  - Compile time polymorphism
  - Run time polymorphism

Example



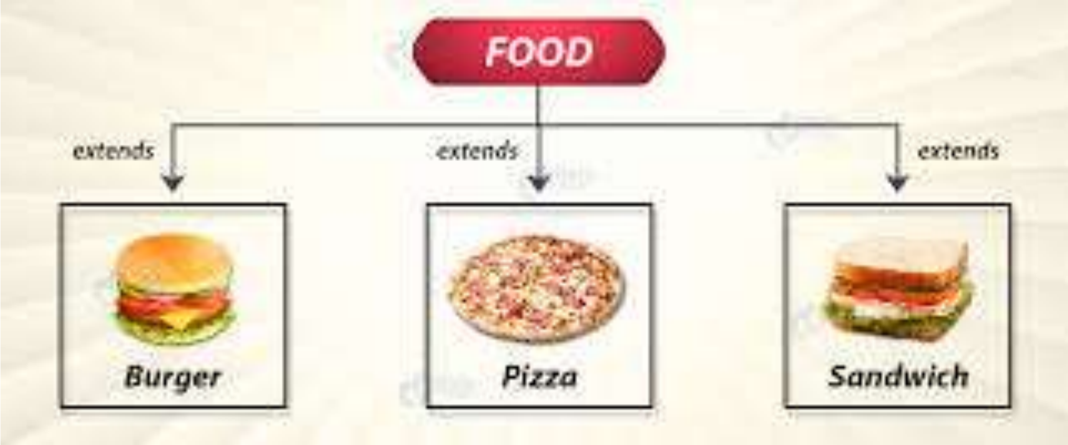
```java
class Bird {
    public void sound() {
        System.out.println("Chirp");
    }
}

class Dog {
    public void sound() {
        System.out.println("Bark");
    }
}

public class Main {
    public static void makeSound(Object animal) {
        if (animal instanceof Bird) {
            ((Bird) animal).sound();
        } else if (animal instanceof Dog) {
            ((Dog) animal).sound();
        }
    }
```



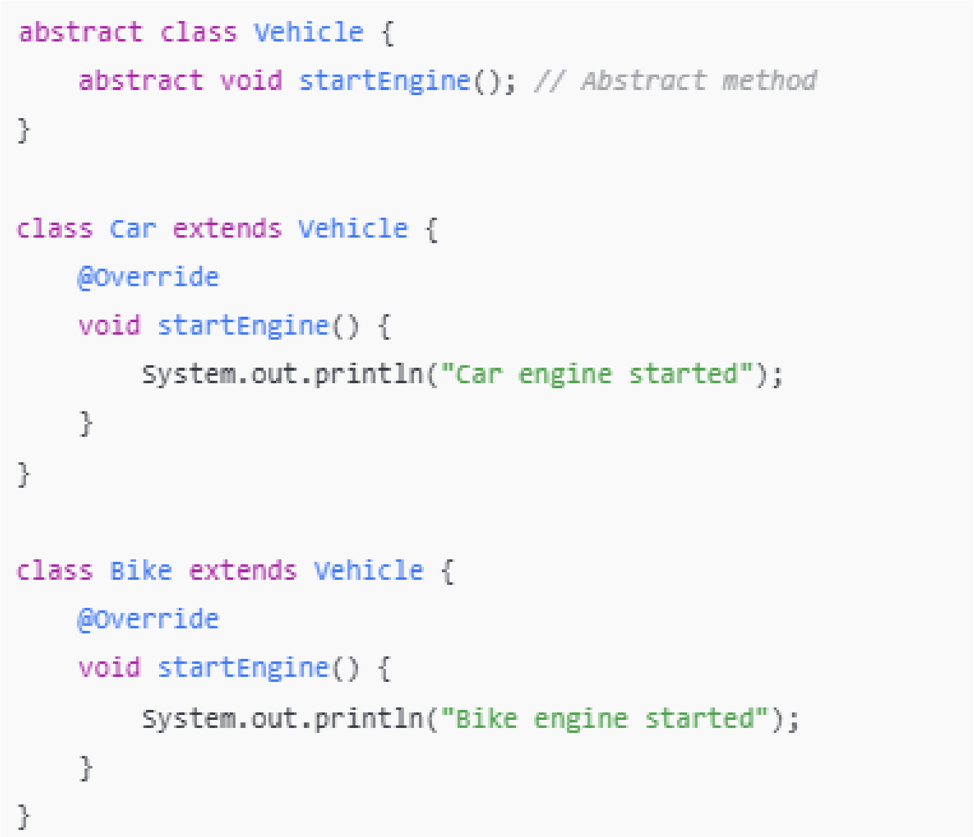
```java
    public static void main(String[] args) {
        Bird bird = new Bird();
        Dog dog = new Dog();

        makeSound(bird);  // Output: Chirp
        makeSound(dog);   // Output: Bark
    }
}
```

Abstraction

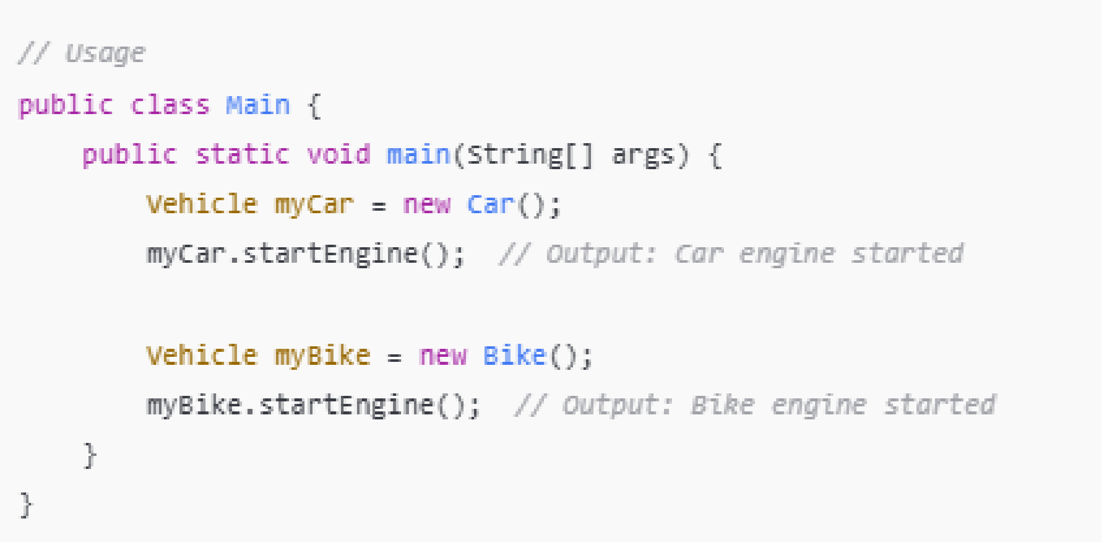
- Hiding internal details and showing functionality is known as abstraction
- Essential element
- Programmer can manage complexity
- Manage through use of hierarchical classification
- 2 types
  - Data abstraction
  - Process abstraction
- Example
  - car, we don't know the internal processing



Example



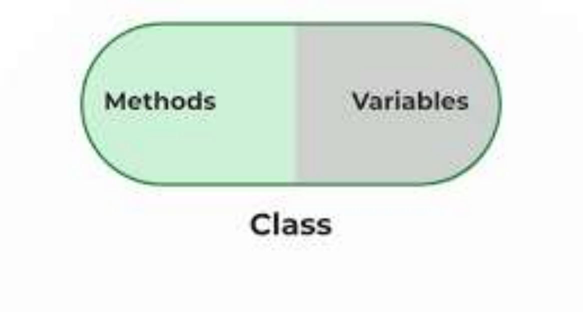
```
abstract class Vehicle {
    abstract void startEngine(); // Abstract method
}

class Car extends Vehicle {
    @Override
    void startEngine() {
        System.out.println("Car engine started");
    }
}

class Bike extends Vehicle {
    @Override
    void startEngine() {
        System.out.println("Bike engine started");
    }
}
```



```
// Usage
public class Main {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.startEngine();  // Output: Car engine started

        Vehicle myBike = new Bike();
        myBike.startEngine();  // Output: Bike engine started
    }
}
```

Encapsulation

- Wrapping up of data
- Mechanism that binds together code & data, manipulate & keep safe from outside interface & misuse
- Example : capsule



- Types of encapsulation
  - Member variable encapsulation
  - Function encapsulation
  - Class encapsulation

3 OOP Principles

- Encapsulation
- Inheritance
- Polymorphism

Abstraction	Encapsulation
Design level process	Implementation level process
Reduce complexity	Provide privacy & maintain control over transparency

References

- Java : the complete Reference (Eleventh Edition), Herbert Schildt, 2018.

Thank You!