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 & JAVA

Topic: Features of OOP

Dr.P.Poonkodi

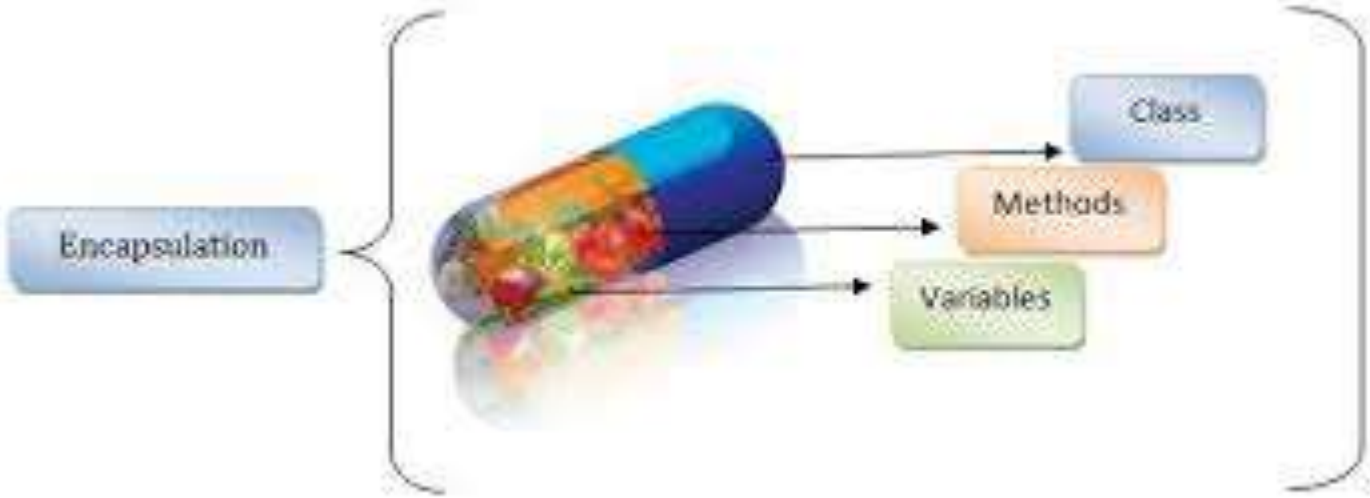
Assistant Professor(SG)

Department of Computer Science and Technology

Features of OOP

- Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects, which encapsulate data and behavior.
- The key features of OOP include
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
  - Class and Object
  - Message Passing
  - Dynamic Binding (Late Binding)
  - Modularity

Encapsulation

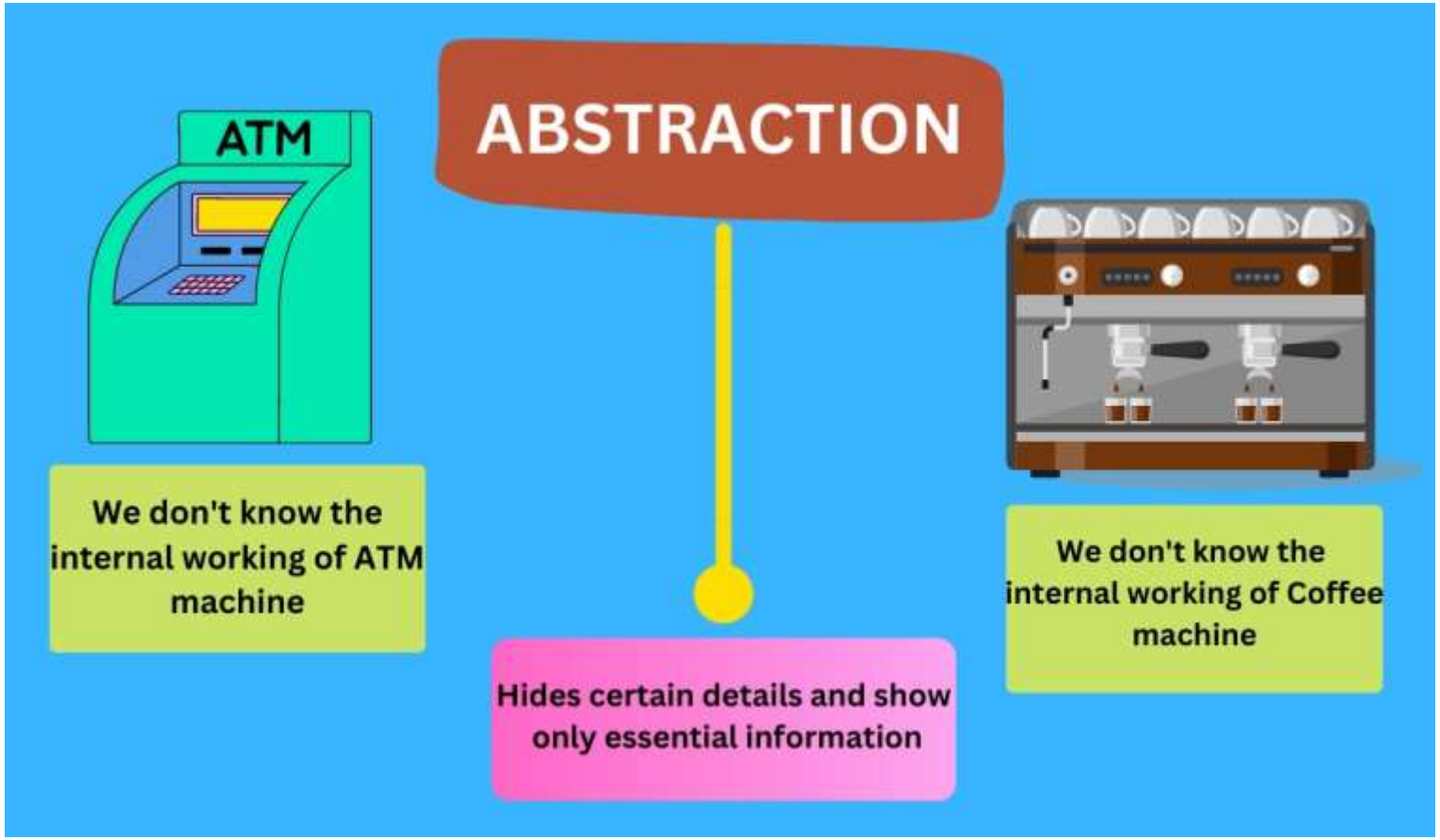


Encapsulation

Bundling data (variables) and methods (functions) that operate on the data into a single unit (class). It also restricts direct access to some components, which helps in data hiding.

```
class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double balance) {
        this.accountNumber = accountNumber;
        this.balance = balance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insufficient funds");
        }
    }

    public double getBalance() {
        return balance;
    }
}
```

```
// Usage
public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("123456", 5000);
        account.deposit(1000);
        System.out.println("Balance: " + account.getBalance());  // Output: 6000
    }
}
```


ABSTRACTION
ATM
We don't know the internal working of ATM machine
Hides certain details and show only essential information
We don't know the internal working of Coffee machine

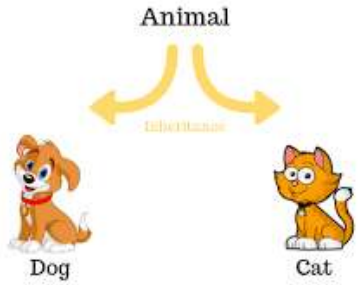
Abstraction

Hiding the complex implementation details and exposing only the necessary parts of an object's functionality to simplify usage.

```
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
    }
}
```

Inheritance



Enabling a new class (child/subclass) to acquire the properties and behavior of an existing class (parent/superclass), promoting code reuse and hierarchical relationships.

```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

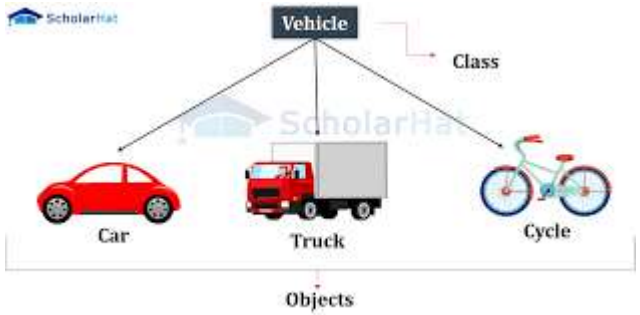
Allowing objects to be treated as instances of their parent class while maintaining their own unique behaviors. It enables method overloading (same function name, different parameters) and method overriding (redefining a method in a subclass).

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

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

```java
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
    }
}
```

Class & Object

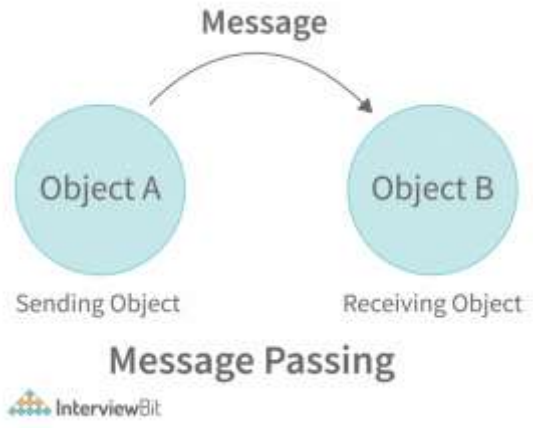


A class is a blueprint for creating objects, and an object is an instance of a class with its own state and behavior.

```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);
    }
}
```

```java
// 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
    }
}
```

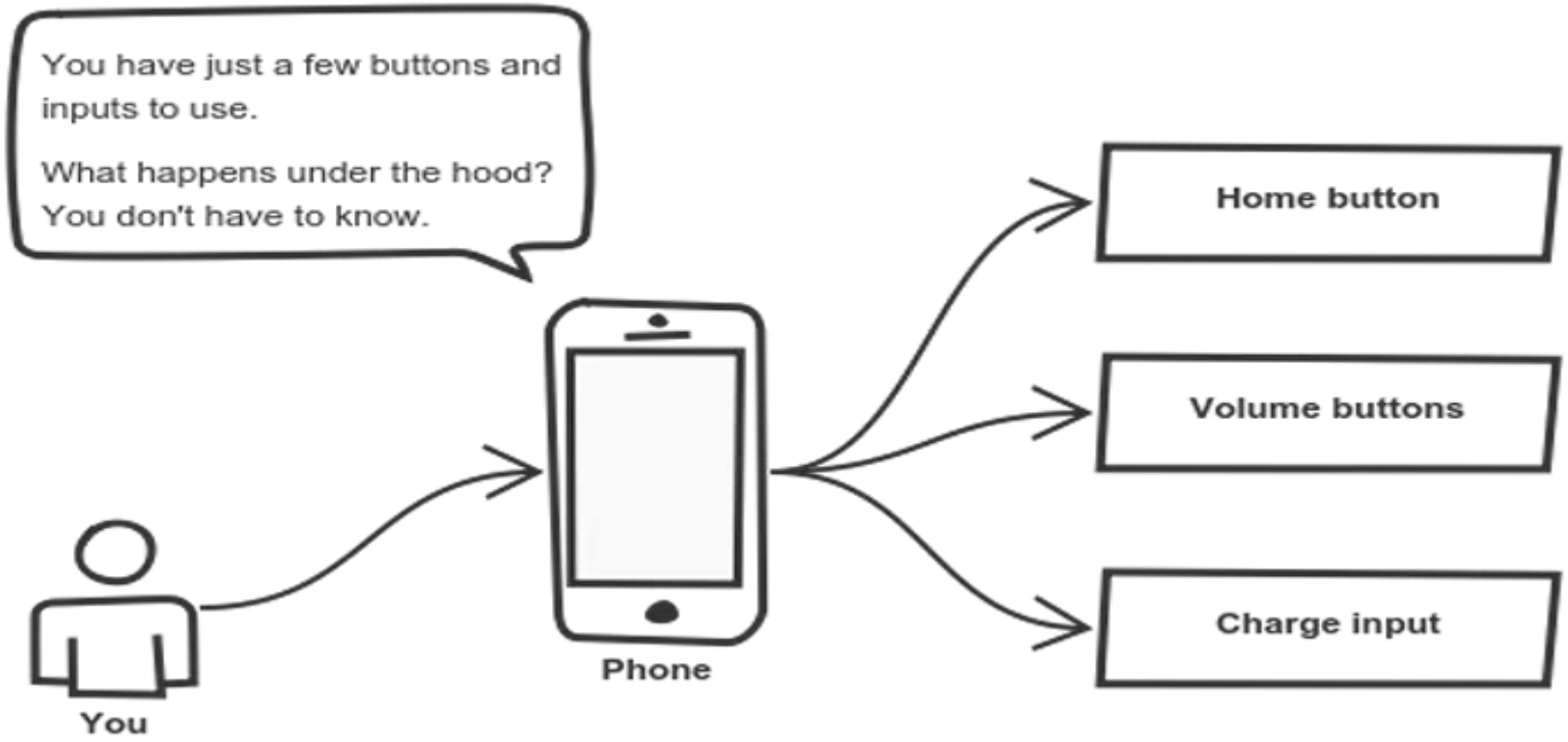
Message Passing



Objects interact with each other by calling methods and exchanging data, allowing modular and structured program design.

```
class Student {
    String name;

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

    public void greet() {
        System.out.println("Hello, my name is " + name);
    }
}

class Teacher {
    String name;

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

    public void introduceStudent(Student student) {
        System.out.println("I am " + name + ", and this is my student.");
        student.greet();
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Student student = new Student("Alice");
        Teacher teacher = new Teacher("Mr. John");

        teacher.introduceStudent(student);
    }
}
```

Dynamic Binding



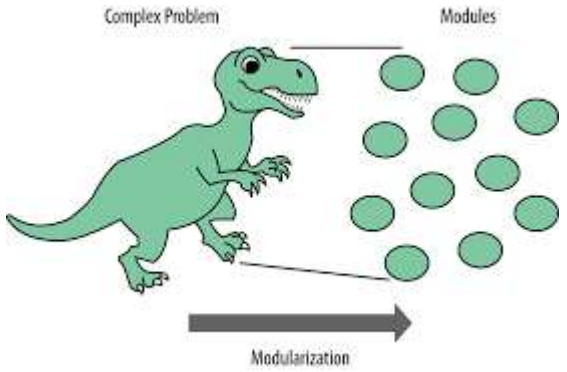
Dynamic Binding

The code to be executed for a function call is determined at runtime, which enables flexibility in method overriding.

```
class Shape {
    public void area() {
        System.out.println("Calculating area...");
    }
}

class Circle extends Shape {
    double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public void area() {
        System.out.println("Circle area: " + (3.14 * radius * radius));
    }
}
```

```
class Rectangle extends Shape {
    double length, width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    @Override
    public void area() {
        System.out.println("Rectangle area: " + (length * width));
    }
}

// Usage
public class Main {
    public static void main(String[] args) {
        Shape shape1 = new Circle(5);
        Shape shape2 = new Rectangle(4, 6);

        shape1.area();  // Output: Circle area: 78.5
        shape2.area();  // Output: Rectangle area: 24
    }
}
```

Modularity



Breaking down a program into smaller, manageable, and reusable components, making development and maintenance easier

Person.java (Module)

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

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void greet() {
        System.out.println("Hi, I am " + name + " and I am " + age + " years old.");
    }
}
```

Main.java (Main Program)

```java
public class Main {
    public static void main(String[] args) {
        Person p1 = new Person("John", 30);
        p1.greet();  // Output: Hi, I am John and I am 30 years old.
    }
}
```

References

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


Thank You!