~ Doubt - Create a Person class with basic details like name and age. Then, create two subclasses: Student and Teacher. Both should inherit from Person and add specific attributes (e.g., grade for Student and subject for Teacher). Write a program to display details of a Student and a Teacher, showing inheritance in action.

```
package que;

public class Person {
    //attributes
    String name;
    int age;

    // constructor
    Person(String name, int age){
        this.name=name;
        this.age=age;
    }
}
```

```
package que;

public class Student extends Person{
    String grade;

    Student(String name,int age, String grade){
        super(name,age); // Call the parent class constructor
        this.grade=grade;
    }

    void displayDetails(){
        System.out.println("Student : "+name + ", Age: "+ age
+", Grade: "+grade);
    }
}
```

```
package que;


public class Teacher extends Person {
    String subject;

    Teacher(String name, int age, String subject){
        super(name,age); // Call the parent class constructor
        this.subject=subject;
    }
```

```
    void displayDetails(){
        System.out.println("Teacher: "+name + ", Age: "+ age +
", Subject: "+ subject);
    }
}
```

```
package que;

public class Main {
    public static void main(String[] args) {
        Student student = new Student("Krishna", 24,"A");
        Teacher teacher = new Teacher("Ravi",35,"Python");

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

Abstraction:

Hiding the implementation detail and showing only the essential features

Example: TV remote(only buttons are visible, not the internal circuits)

Encapsulation:

Wrapping of data(variables) and methods in a single unit(class), restricting the direct access to data

Example: Medical Capsule(The contents are enclosed/ encapsulated in a capsule)

Real-Life analogy:

Car Interface:

1. Driver operates a car without knowing or understanding how the engine works (Abstraction)
2. The engine and other mechanical components are encapsulated within the car’ s body (Encapsulation)

Abstract Class:

A class that cannot be instantiated and may contain abstract methods(without body) and non-abstract methods.

```
package abstractExample;

//Abstract class
abstract class Vehicle {
    // Abstract Method(No Implementation)
    abstract void start();

    // Non-Abstract Method (With implementation)
    void stop(){
        System.out.println("Vehicle Stopped");
    }
}
```

```
package abstractExample;

//Subclass  Car extends the abstract class vehicle
public class Car extends Vehicle{
    //Providing the implementation of the abstract method
    void start(){
        System.out.println("Car Started");
    }
}
```

```
package abstractExample;

public class Main {
    public static void main(String[] args) {
        // Create an instance of car class
        Car car = new Car();
```

```
        //Call the start method (from Car class)
        car.start();

        // Call the stop method (Inherited from Vehicle class)
        car.stop();
    }
}
```

~ Interfaces

A completely abstract class with only abstract methods.

```
package interfaceExample;

public interface Animal {

    //Abstract Methods(No Implementation)
    void sound();
}
```

```
package interfaceExample;

//Dog class implements the Animal Interface
public class Dog implements Animal {
    //providing the implementation of the abstract method
    public void sound() {
        System.out.println("Dog barks");
    }
}
```

```
package interfaceExample;

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        // Call the sound method(Defined in Animal,
implementated in dog)
        myDog.sound();
    }
}
```