# 🧪 LAB PROBLEM 1: Fruit and Apple Classes (1;2 done)

**Topic: Basic Single Inheritance**

## Problem Statement:

Create a Fruit class with color and taste fields. Create an Apple class that extends

Fruit and adds variety field.

## Hints:

* Use extends keyword for inheritance
* Make fields protected in parent class
* Test by creating Apple object and accessing inherited fields

class Fruit {

protected String color;

protected String taste;

public Fruit(String color, String taste) {

this.color = color;

this.taste = taste;

}

}

class Apple extends Fruit {

String variety;

public Apple(String color, String taste, String variety) {

super(color, taste);

this.variety = variety;

}

public void display() {

System.out.println("Color: " + color);

System.out.println("Taste: " + taste);

System.out.println("Variety: " + variety);

}

}

public class FruitTest {

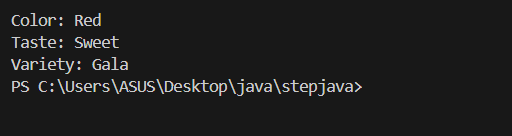
public static void main(String[] args) {

Apple apple = new Apple("Red", "Sweet", "Gala");

apple.display();

}

}



# 🧪 LAB PROBLEM 2: Phone and SmartPhone Constructors

**Topic: Constructor Chaining with super()**

## Problem Statement:

Create Phone class with brand and model. Create SmartPhone class extending Phone with

operatingSystem field. Use constructor chaining.

## Hints:

* Add print statements in constructors to see execution order
* Use super() in child constructor
* Create objects using different constructor combinations

class Phone {

String brand;

String model;

public Phone(String brand, String model) {

this.brand = brand;

this.model = model;

System.out.println("Phone Constructor called");

}

}

class SmartPhone extends Phone {

String operatingSystem;

public SmartPhone(String brand, String model, String operatingSystem) {

super(brand, model);

this.operatingSystem = operatingSystem;

System.out.println("SmartPhone Constructor called");

}

public void display() {

System.out.println("Brand: " + brand + ", Model: " + model + ", OS: " + operatingSystem);

}

}

public class PhoneTest {

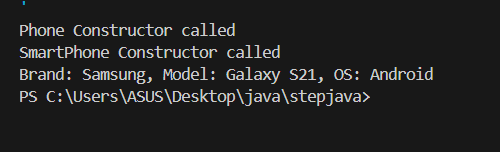
public static void main(String[] args) {

SmartPhone sp = new SmartPhone("Samsung", "Galaxy S21", "Android");

sp.display();

}

}



# 🧪 LAB PROBLEM 3: Bird Flying Behavior

**Topic: Method Overriding with @Override**

## Problem Statement:

Create Bird class with fly() method. Create Penguin and Eagle classes that override

fly() method differently.

## Hints:

* Use @Override annotation
* Make different implementations in each child class
* Test polymorphism with array of Bird references

# 🧪 LAB PROBLEM 4: Color Hierarchy Chain

**Topic: Multilevel Inheritance**

## Problem Statement:

Create inheritance chain: Color → PrimaryColor → RedColor. Each class adds specific properties and methods.

## Hints:

* Color has name field
* PrimaryColor adds intensity field
* RedColor adds shade field
* Show constructor chaining through all levels

# 🧪 LAB PROBLEM 5: Musical Instrument Family

**Topic: Hierarchical Inheritance**

## Problem Statement:

Create Instrument base class. Create Piano, Guitar, and Drum classes that all extend

Instrument.

## Hints:

* Base class has common fields like name, material
* Each child adds specific fields (strings, keys, etc.)
* Test using array of Instrument references

# 🧪 LAB PROBLEM 6: Box and Gift Box Enhancement

**Topic: Using super in overridden methods**

## Problem Statement:

Create Box class with pack() and unpack() methods. Create GiftBox that overrides these methods but still uses parent functionality.

## Hints:

* Call super.pack() in overridden method first
* Add gift-specific functionality after super call
* Show enhanced behavior while preserving original