

- **Name: Afreen Ahmed**
  - **Enrollment Number: 1KI21CS005**
  - **Batch / Class: 2025**
  - **Assignment: (Bridge Course Day 4)**
  - **Date of Submission: 27/06/2025**
- 

## **Problem Solving Activity 5.1**

### **1. Program Statement:**

This program is about three things we use in daily life: **makeup**, **skincare product**, and a **dress**. We first create a lipstick using the Makeup class and do actions like apply, blend, and remove it. Then we create a moisturizer using the SkinCareProduct class and do actions like apply it, read how to use it, and check its expiry date. Finally, we create a party dress using the Dress class and do actions like wear, wash, and iron it. Each class has some details (like name, type, color) and some actions that we can do with them.

---

### **2. Algorithm**

1. Start
  2. Create a Makeup class with attributes like brand, type, shade, price, and waterproof status.
  3. Create methods to apply(), blend(), and remove() the makeup.
  4. Create a SkinCareProduct class with product details like name, skin type, ingredients, expiry date, and usage.
  5. Add methods to applyToSkin(), readInstructions(), and checkExpiry().
  6. Create a Dress class with color, size, fabric, style, and brand.
  7. Add methods to wear(), wash(), and iron() the dress.
  8. In main(), create one object for each class and call their methods to show actions.
  9. End
-

### 3. Pseudocode

Class Makeup:

brand, type, shade, price, isWaterproof

Method apply()

Method blend()

Method remove()

Class SkinCareProduct:

productName, skinType, ingredients, expiryDate, usageFrequency

Method applyToSkin()

Method readInstructions()

Method checkExpiry()

Class Dress:

color, size, fabric, style, brand

Method wear()

Method wash()

Method iron()

Main Method:

Create Makeup object

Call apply(), blend(), remove()

Create SkinCareProduct object

Call applyToSkin(), readInstructions(), checkExpiry()

Create Dress object

Call wear(), wash(), iron()

---

### 4. Program Code

```
package day5;
```

```
public class BeautyCareDemo {
```

```
public static void main(String[] args) {  
    // Makeup object  
    Makeup lipstick = new Makeup("Maybelline", "Lipstick", "Red", 499.0, true);  
    lipstick.apply();  
    lipstick.blend();  
    lipstick.remove();  
    System.out.println();  
    // SkinCareProduct object  
    SkinCareProduct moisturizer = new SkinCareProduct(  
        "HydraGlow Moisturizer",  
        "Dry",  
        "Aloe Vera, Vitamin E",  
        "2025-12-31",  
        "twice a day"  
    );  
    moisturizer.applyToSkin();  
    moisturizer.readInstructions();  
    moisturizer.checkExpiry();  
    System.out.println();  
    // Dress object  
    Dress partyDress = new Dress("Black", "M", "Silk", "Party", "Zara");  
    partyDress.wear();  
    partyDress.wash();  
    partyDress.iron();  
}  
}
```

```
// Makeup class  
class Makeup {
```

```
String brand;
String type;
String shade;
double price;
boolean isWaterproof;
public Makeup(String brand, String type, String shade, double price, boolean isWaterproof) {
    this.brand = brand;
    this.type = type;
    this.shade = shade;
    this.price = price;
    this.isWaterproof = isWaterproof;
}
void apply() {
    System.out.println("Applying " + type + " in shade " + shade);
}
void remove() {
    System.out.println("Removing " + type);
}
void blend() {
    System.out.println("Blending the " + type);
}
}
// SkinCareProduct class
class SkinCareProduct {
    String productName;
    String skinType;
    String ingredients;
    String expiryDate;
    String usageFrequency;
```

```
public SkinCareProduct(String productName, String skinType, String ingredients, String
expiryDate, String usageFrequency) {
    this.productName = productName;
    this.skinType = skinType;
    this.ingredients = ingredients;
    this.expiryDate = expiryDate;
    this.usageFrequency = usageFrequency;
}
void applyToSkin() {
    System.out.println("Applying " + productName + " for " + skinType + " skin.");
}
void checkExpiry() {
    System.out.println(productName + " expires on " + expiryDate);
}
void readInstructions() {
    System.out.println("Use " + productName + " " + usageFrequency);
}
}
// Dress class
class Dress {
    String color;
    String size;
    String fabric;
    String style;
    String brand;
    public Dress(String color, String size, String fabric, String style, String brand) {
        this.color = color;
        this.size = size;
        this.fabric = fabric;
        this.style = style;
    }
}
```

```

    this.brand = brand;
}
void wear() {
    System.out.println("Wearing a " + style + " dress in " + color + " color.");
}
void wash() {
    System.out.println("Washing the " + fabric + " dress.");
}
void iron() {
    System.out.println("Ironing the dress.");
}
}

```

## 5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Makeup: "Maybelline", "Lipstick", "Red", 499.0, true	Applying Lipstick in shade Red  Blending the Lipstick  Removing Lipstick	Applying Lipstick in shade Red  Blending the Lipstick  Removing Lipstick	Pass
2	SkinCareProduct: "HydraGlow Moisturizer", "Dry", "Aloe Vera, Vitamin E", "2025-	Applying HydraGlow Moisturizer for Dry skin.  Use HydraGlow Moisturizer twice a day  HydraGlow Moisturizer expires on 2025-12-31	Applying HydraGlow Moisturizer for Dry skin.  Use HydraGlow Moisturizer twice a day  HydraGlow Moisturizer expires on 2025-12-31	Pass

	12-31", "twice a day"			
3	Dress: "Black", "M", "Silk", "Party", "Zara"	Wearing a Party dress in Black color. Washing the Silk dress. Ironing the dress.	Wearing a Party dress in Black color. Washing the Silk dress. Ironing the dress.	Pass

## 1. Screenshots of Output

```

● Applying Lipstick in shade Red
  Blending the Lipstick
  Removing Lipstick

Applying HydraGlow Moisturizer for Dry skin.
Use HydraGlow Moisturizer twice a day
HydraGlow Moisturizer expires on 2025-12-31

Wearing a Party dress in Black color.
Washing the Silk dress.
Ironing the dress.
○ PS C:\stemupbridge>

```

## 7. Observation / Reflection

The code provides a real-life example by using familiar daily objects like makeup, moisturizer, and a dress, making it easy to relate and understand. It demonstrates how to organize related data using classes and perform specific actions through methods, reflecting real-world behavior. The program also helps in understanding the basics of object-oriented programming in Java, such as how to create objects, use constructors to initialize them, and call methods to perform tasks. Each object in the code performs realistic actions, like applying makeup or wearing a dress, which makes the learning experience more practical and meaningful.

## Problem Solving Activity 1.1: Simple Dog Class

We are going to make a class called Dog that tells us about dogs. Every dog has the same species and number of legs, so we add those as class attributes. Each dog also has its own name, breed, and age, which we add as instance attributes. We also create a method called bark() that makes the dog bark by printing “Woof!”.

---

### 2. Algorithm

1. Start
  2. Create a class called Dog
  3. Add class attributes: species = "Canis familiaris" and numLegs = 4
  4. Define a constructor to set name, breed, and age
  5. Create a method bark() that prints “Woof!”
  6. In the main part, create dog objects with different details
  7. Call the bark() method
  8. End
- 

### 3. Pseudocode

Class Dog:

Class Attributes:

species = "Canis familiaris"

numLegs = 4

Instance Attributes:

name

breed

age

Constructor(name, breed, age):

Set this.name ← name

Set this.breed ← breed



Set this.age ← age

Method bark():

Print "Woof!"

Main:

Create Dog object

Call bark() method

---

#### 4. Program Code

```
package day5;

public class Dog {

    // Class attributes

    static String species = "Canis familiaris";

    static int numLegs = 4;

    // Instance attributes

    String name;

    String breed;

    int age;

    // Constructor

    public Dog(String name, String breed, int age) {

        this.name = name;

        this.breed = breed;

        this.age = age;

    }

    // Method to make the dog bark

    public void bark() {

        System.out.println("Woof!");

    }

}
```

```
// Main method to test the Dog class
public static void main(String[] args) {
    Dog myDog = new Dog("Buddy", "Labrador", 3);
    System.out.println("Name: " + myDog.name);
    System.out.println("Breed: " + myDog.breed);
    System.out.println("Age: " + myDog.age);
    System.out.println("Species: " + Dog.species);
    System.out.println("Legs: " + Dog.numLegs);
    myDog.bark();
}
}
```

---

## 5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"Rocky", breed = "Labrador", age = 3	Woof!	Woof!	Pass
2	"Tommy", breed = "Pug", age = 5	Woof!	Woof!	Pass
3	"Bruno", breed = "German Shepherd", age = 2	Woof!	Woof!	Pass

---

## 6 Screenshots of Output

### Case 1:

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS Run: Dog + - [ ] [ ] ... ^ X
• PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppData\Roaming\Code\User\workspaceStorage\937abdb2243a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Dog'
Name: Rocky
Breed: Labrador
Age: 3
Species: Canis familiaris
Legs: 4
Woof!
○ PS C:\stemupbridge>
```

### Case 2:

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS Run: Dog + - [ ] [ ] ... ^ X
• PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppData\Roaming\Code\User\workspaceStorage\937abdb2243a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Dog'
Name: Tommy
Breed: Pug
Age: 3
Species: Canis familiaris
Legs: 4
Woof!
○ PS C:\stemupbridge>
```

### Case 3:

```
PROBLEMS 34 OUTPUT DEBUG CONSOLE TERMINAL PORTS Run: Dog + - [ ] [ ] ... ^ X
• PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppData\Roaming\Code\User\workspaceStorage\937abdb2243a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Dog'
Name: Bruno
Breed: German Shepherd
Age: 3
Species: Canis familiaris
Legs: 4
Woof!
○ PS C:\stemupbridge>
```

---

## **7. Observation / Reflection**

This program shows how we can create a simple class for dogs using both class-level and object-level data. It teaches how to use shared information like species and legs for all dogs, and also store different names, breeds, and ages for each dog. The method bark() gives the dog an action, making it more like a real object. This is a basic example of object-oriented programming, where data and behavior are grouped inside a class.

### 3. Program Statement 1.2: Basic Book Class

We need to create a class named Book that shows the details of a book. Each book will have a title, the name of the author, the number of pages, and whether the book is open or closed. We will also add two actions: one to open the book and one to close it. Opening sets isOpen to true, and closing sets it to false.

---

### 2. Algorithm

1. Start
  2. Create a class called Book
  3. Add instance variables: title, author, numPages, isOpen
  4. Create a constructor to set title, author, and numPages, and set isOpen to false by default
  5. Add a method openBook() that sets isOpen to true
  6. Add another method closeBook() that sets isOpen to false
  7. Create book objects and test by calling open and close methods
  8. End
- 

### 3. Pseudocode

Class Book:

Attributes:

title

author

numPages

isOpen  $\leftarrow$  false

Constructor(title, author, numPages):

Set this.title  $\leftarrow$  title

Set this.author ← author

Set this.numPages ← numPages

Set this.isOpen ← false

Method openBook():

Set isOpen ← true

Method closeBook():

Set isOpen ← false

Main:

Create Book object

Call openBook()

Call closeBook()

---

#### 4. Program Code

```
package day5;
```

```
public class Book {
```

```
    // Instance attributes
```

```
    String title;
```

```
    String author;
```

```
    int numPages;
```

```
    boolean isOpen;
```

```
    // Constructor
```

```
    public Book(String title, String author, int numPages) {
```

```
this.title = title;

this.author = author;

this.numPages = numPages;

this.isOpen = false; // Book is initially closed
}

// Method to open the book

public void openBook() {

    isOpen = true;

    System.out.println("The book is now open.");
}

// Method to close the book

public void closeBook() {

    isOpen = false;

    System.out.println("The book is now closed.");
}

// Main method to test the Book class

public static void main(String[] args) {

    Book myBook = new Book("The Alchemist", "Paulo Coelho", 197);

    System.out.println("Title: " + myBook.title);

    System.out.println("Author: " + myBook.author);

    System.out.println("Pages: " + myBook.numPages);

    System.out.println("Is Open: " + myBook.isOpen);
}
```

```

myBook.openBook(); // Sets isOpen to true

System.out.println("Is Open: " + myBook.isOpen);

myBook.closeBook(); // Sets isOpen to false

System.out.println("Is Open: " + myBook.isOpen);

}

}

```

---

## 5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"The Alchemist", author = "Paulo Coelho", pages = 197	isOpen = true	isOpen = true	Pass
2	"Wings of Fire", author = "A.P.J. Abdul Kalam", pages = 180	isOpen = false	isOpen = false	Pass
3	"1984", author = "George Orwell", pages = 328	isOpen = false	isOpen = false	Pass





## **7. Observation / Reflection**

This program helps to understand how to store details of a book using instance variables and perform actions like opening or closing it. It shows how to use a class to model a real-world object and manage its state using methods. The `isOpen` variable tells whether the book is open or not, which can change through the methods. This is a simple and useful way to learn object-oriented concepts like class, object, attributes, and behavior.

### Problem 1.3: Identify Class Elements for Car Class

This program helps to understand how to store details of a book using instance variables and perform actions like opening or closing it. It shows how to use a class to model a real-world object and manage its state using methods. The isOpen variable tells whether the book is open or not, which can change through the methods. This is a simple and useful way to learn object-oriented concepts like class, object, attributes, and behavior.

---

## 2. Algorithm

1. Start
  2. Create a class named Car
  3. Add a **class attribute** numWheels and set it to 4 (same for all cars)
  4. Add **instance attributes**: color, model, brand, year, and mileage
  5. Create a constructor to set these values
  6. Create methods:
  7. startEngine() to print engine started
  8. drive() to print car is driving
  9. stopEngine() to print engine stopped
  10. In the main() method:
  11. Create a car object with some value
  12. Print its details
  13. Call the methods to show actions
  14. End
- 

## 3. Pseudocode

Class Car:

Class Attribute:

numWheels ← 4

Instance Attributes:

color

model

brand

year

mileage

Constructor(color, model, brand, year, mileage):

Set this.color ← color

Set this.model ← model

Set this.brand ← brand

Set this.year ← year

Set this.mileage ← mileage

Method startEngine():

Print "Engine started."

Method drive():

Print "The car is driving."

Method stopEngine():

Print "Engine stopped."

Main Method:

Create Car object with sample values

Print car details

Call startEngine()

Call drive()

Call stopEngine()

---

#### 4. Program Code

```
package day5;

public class Car {

    // Class attribute (shared by all cars)

    static int numWheels = 4;

    // Instance attributes

    String color;

    String model;

    String brand;

    int year;

    double mileage;

    // Constructor

    public Car(String color, String model, String brand, int year, double mileage) {

        this.color = color;

        this.model = model;

        this.brand = brand;

        this.year = year;

        this.mileage = mileage;

    }
```

```
// Instance method to start the engine

public void startEngine() {

    System.out.println("Engine started.");

}

// Instance method to drive the car

public void drive() {

    System.out.println("The car is driving.");

}

// Instance method to stop the engine

public void stopEngine() {

    System.out.println("Engine stopped.");

}

// Main method to test the Car class

public static void main(String[] args) {

    Car myCar = new Car("Red", "Civic", "Honda", 2020, 15000.5);

    System.out.println("Brand: " + myCar.brand);

    System.out.println("Model: " + myCar.model);

    System.out.println("Year: " + myCar.year);

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

    System.out.println("Mileage: " + myCar.mileage);

    System.out.println("Wheels: " + Car.numWheels);

}
```

```

    myCar.startEngine();

    myCar.drive();

    myCar.stopEngine();

  }

}

```

## 5. Test Cases

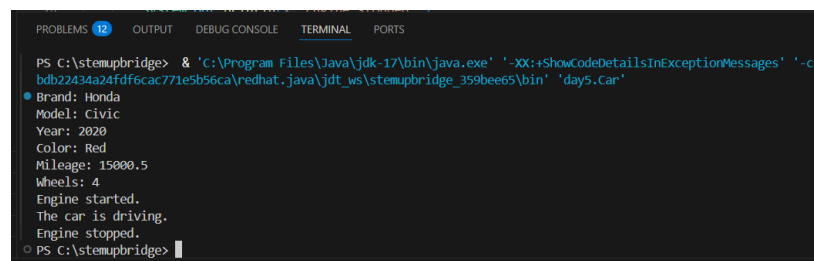
Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"Red", model = "Civic", brand = "Honda", year = 2020, mileage = 15000.5	Brand: Honda Model: Civic Year: 2020 Color: Red Mileage: 15000.5 Wheels: 4 Engine started. The car is driving. Engine stopped.	Brand: Honda Model: Civic Year: 2020 Color: Red Mileage: 15000.5 Wheels: 4 Engine started. The car is driving. Engine stopped.	Pass
2	color = "Blue", model = "Swift", brand = "Maruti",	Brand: Maruti Model: Swift Year: 2019	Brand: Maruti Model: Swift Year: 2019	Pass

	year = 2019, mileage = 20000	Color: Blue  Mileage: 20000.0  Wheels: 4  Engine started.  The car is driving.  Engine stopped.	Color: Blue  Mileage: 20000.0  Wheels: 4  Engine started.  The car is driving.  Engine stopped.	
3	color = "White", model = "i20", brand = "Hyundai", year = 2021, mileage = 9000	Brand: Hyundai  Model: i20  Year: 2021  Color: White  Mileage: 9000.0  Wheels: 4  Engine started.  The car is driving.  Engine stopped.	Brand: Hyundai  Model: i20  Year: 2021  Color: White  Mileage: 9000.0  Wheels: 4  Engine started.  The car is driving.  Engine stopped.	Pass

## 1. Screenshots of Output

### Case 1:



```

PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp bdb22434a24fd6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Car'
• Brand: Honda
  Model: Civic
  Year: 2020
  Color: Red
  Mileage: 15000.5
  Wheels: 4
  Engine started.
  The car is driving.
  Engine stopped.
○ PS C:\stemupbridge>
  
```



## Case 2:

```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Af
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Car'
Brand: Maruthi
Model: Swift
Year: 2020
Color: Blue
Mileage: 20000.5
Wheels: 4
Engine started.
The car is driving.
Engine stopped.
PS C:\stemupbridge>
```

## Case 3:

```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Car'
Brand: Maruthi
Model: Hyдай
Year: 2020
Color: White
Mileage: 90000.0
Wheels: 4
Engine started.
The car is driving.
Engine stopped.
PS C:\stemupbridge>
```

---

## 7. Observation / Reflection

This program models a real-life car using a Java class. It shows how to use class and instance attributes to store both common (like number of wheels) and unique (like color, brand, mileage) details of a car. The methods demonstrate behaviors like starting, driving, and stopping the engine. The code also explains the use of constructors to initialize objects and how to use both static and non-static members. It's a good example of object-oriented programming that connects real-world actions with class design.

## Problem 2.1: Create Dogs

In this problem, we need to create two dogs using the Dog class. The first dog is named Buddy, its breed is Golden Retriever, and it is 5 years old. The second dog is Lucy, a Poodle, and 2 years old. After creating them, we have to call their bark() method so they bark, and then print their names and ages to show their details.

---

## 2. Algorithm

1. Start
  2. Create a class PetDog
  3. Add **class attributes**: species = "Canis familiaris" and numLegs = 4
  4. Add **instance attributes**: name, breed, and age
  5. Create a constructor to set these values
  6. Create a method bark() that prints the dog's name followed by "says: Woof!"
  7. In the main() method:
  8. Create two PetDog objects: dog1 and dog2 with different values
  9. Call bark() method for both
  10. Print their names and ages
  11. End
- 

## 3. Pseudocode

Class PetDog:

Class Attributes:

species = "Canis familiaris"

numLegs = 4

Instance Attributes:

name

breed

age

Constructor(name, breed, age):

Set this.name  $\leftarrow$  name

Set this.breed  $\leftarrow$  breed

Set this.age  $\leftarrow$  age

Method bark():

Print name + " says: Woof!"

Main Method:

Create dog1  $\leftarrow$  PetDog("Buddy", "Golden Retriever", 5)

Create dog2  $\leftarrow$  PetDog("Lucy", "Poodle", 2)

Call dog1.bark()

Print dog1's name and age

Call dog2.bark()

Print dog2's name and age

---

#### 4. Program Code

```
package day5;

public class PetDog {

    // Class attributes

    static String species = "Canis familiaris";

    static int numLegs = 4;

    // Instance attributes

    String name;

    String breed;

    int age;

    // Constructor

    public PetDog(String name, String breed, int age) {

        this.name = name;
```

```

        this.breed = breed;

        this.age = age;
    }

    // Method to make the dog bark
    public void bark() {
        System.out.println(name + " says: Woof!");
    }

    // Main method
    public static void main(String[] args) {
        // Create two PetDog objects
        PetDog dog1 = new PetDog("Buddy", "Golden Retriever", 5);
        PetDog dog2 = new PetDog("Lucy", "Poodle", 2);

        // Call bark() method and print name and age
        dog1.bark();

        System.out.println("Name: " + dog1.name + ", Age: " + dog1.age);
        dog2.bark();

        System.out.println("Name: " + dog2.name + ", Age: " + dog2.age);
    }
}

```

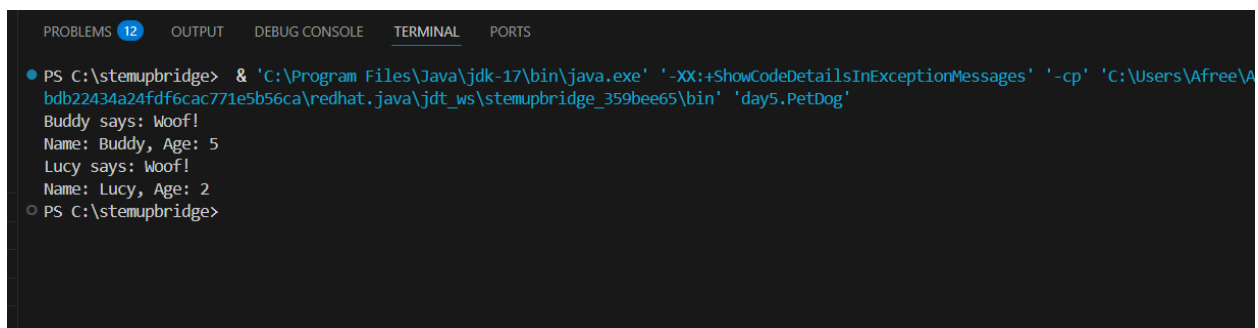
## 5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	name = "Buddy", breed = "Golden	Buddy says: Woof!	Buddy says: Woof!	Pass

	Retriever", age = 5	Name: Buddy, Age: 5	Name: Buddy, Age: 5	
2	name:"Lucy", breed = "Poodle", age = 2	Lucy says: Woof! Name: Lucy, Age: 2	Lucy says: Woof! Name: Lucy, Age: 2	Pass
3	name = "Max", breed = "Beagle", age = 3	Max says: Woof! Name: Max, Age: 3	Max says: Woof! Name: Max, Age: 3	Pass

## 6. Screenshots of Output



```

PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\A\bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.PetDog'
Buddy says: Woof!
Name: Buddy, Age: 5
Lucy says: Woof!
Name: Lucy, Age: 2
○ PS C:\stemupbridge>
  
```

## 7. Observation / Reflection

This program shows how to create and use multiple objects from the same class. It uses a constructor to set the values for each dog, and a method to make the dog bark. The bark() method prints a friendly message with the dog's name. The use of both class and instance attributes helps understand the difference between shared and unique data in object-oriented programming. This is a good basic example of how real-world things (like dogs) can be modeled in Java.

## Problem 2.2: Manage Books

In this problem, we need to create a class called Book and make two book objects using a constructor. Each book will have some details like title, author, number of pages, and whether the book is open or closed. We also need to create a method called displayStatus() that checks if the book is open or not. If the book is open, it should print "Open", otherwise it should print "Closed". The output should show the book's title, author, and current status.

---

## 2. Algorithm

1. Start
  2. Create a class named LibraryBook
  3. Add instance attributes: title, author, numPages, and isOpen
  4. Use a constructor to set title, author, and number of pages; set isOpen to false by default
  5. Create a method openBook() to set isOpen as true
  6. Create a method closeBook() to set isOpen as false
  7. Create a method displayStatus():
  8. Check if the book is open
  9. Print the book's title, author, and status (Open/Closed)
  10. In main(), create two book objects
  11. Open the first book
  12. Display status of both books
  13. End
- 

## 3. Pseudocode

Class LibraryBook:

Attributes:

title

author

numPages

isOpen ← false

Constructor(title, author, numPages):

Set this.title ← title

Set this.author ← author

Set this.numPages ← numPages

Set this.isOpen ← false

Method openBook():

Set isOpen ← true

Method closeBook():

Set isOpen ← false

Method displayStatus():

If isOpen is true:

status ← "Open"

Else:

status ← "Closed"

Print title + " by " + author + " is " + status

Main Method:

Create book1 ← LibraryBook("The Alchemist", "Paulo Coelho", 197)

Create book2 ← LibraryBook("1984", "George Orwell", 328)

Call book1.openBook()

Call book1.displayStatus()

Call book2.displayStatus()

---

#### 4. Program Code

package day5;

```
public class LibraryBook {  
    // Instance attributes  
    String title;  
    String author;  
    int numPages;  
    boolean isOpen;  
    // Constructor  
    public LibraryBook(String title, String author, int numPages) {  
        this.title = title;  
        this.author = author;  
        this.numPages = numPages;  
        this.isOpen = false; // Initially closed  
    }  
    // Method to open the book  
    public void openBook() {  
        isOpen = true;  
    }  
    // Method to close the book  
    public void closeBook() {  
        isOpen = false;  
    }  
    // Method to display the status of the book  
    void displayStatus() {  
        String status = isOpen ? "Open" : "Closed";  
        System.out.println(title + " by " + author + " is " + status);  
    }  
}
```



```
// Main method

public static void main(String[] args) {

    // Create two LibraryBook objects

    LibraryBook book1 = new LibraryBook("The Alchemist", "Paulo Coelho", 197);

    LibraryBook book2 = new LibraryBook("1984", "George Orwell", 328);

    // Open one book

    book1.openBook();

    // Display status of both

    book1.displayStatus();

    book2.displayStatus();

}

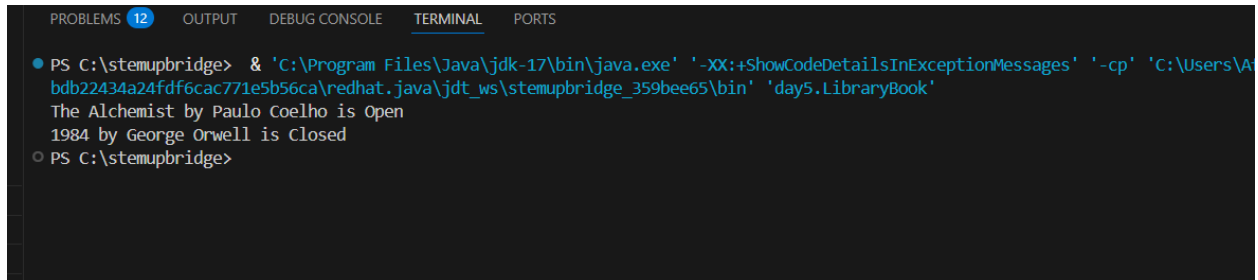
}
```

## 5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Book: "The Alchemist", Author: "Paulo Coelho", Pages: 197	The Alchemist by Paulo Coelho is Open	The Alchemist by Paulo Coelho is Open	Pass
2	Book: "1984", Author: "George Orwell", Pages: 328	1984 by George Orwell is Closed	1984 by George Orwell is Closed	Pass
3	Book: "Wings of Fire", Author: "A.P.J. Abdul Kalam", Pages: 180	Wings of Fire by A.P.J. Abdul Kalam is Closed	Wings of Fire by A.P.J. Abdul Kalam is Closed	Pass

## 6. Screenshots of Output

### Case 1:



```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
• PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\A\
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.LibraryBook'
The Alchemist by Paulo Coelho is Open
1984 by George Orwell is Closed
○ PS C:\stemupbridge>
```

---

## 7. Observation / Reflection

This program shows how to create book objects using a constructor and check their status using a method. The use of a boolean variable `isOpen` helps keep track of whether a book is currently open or closed. The method `displayStatus()` uses a simple condition to print the correct status. This example is helpful to understand object-oriented concepts like encapsulation, object creation, and behavior through methods in Java. Code is clear, modular, and easy to reuse for other values.

### Problem 2.3: Student Record

In this problem, we are creating a class called Student to store student details. Each student will have a name, an ID number, and a major (subject they are studying). We use a constructor to set these details when a student object is created. There is also a method called getInfo() that gives back the student's information. We will then create three student objects with different details and print their info using the method.

---

### 2. Algorithm

1. Start
  2. Create a class named Student
  3. Add instance variables: name, idNumber, and major
  4. Create a constructor to initialize the above values
  5. Define a method getInfo() that returns student details as a formatted string
  6. In the main() method:
  7. Create three student objects with different data
  8. Print each student's info using the getInfo() method
  9. End
- 

### 3. Pseudocode

Class Student:

Attributes:

name

idNumber

major

Constructor(name, idNumber, major):

Set this.name  $\leftarrow$  name

Set this.idNumber  $\leftarrow$  idNumber

Set this.major  $\leftarrow$  major

Method getInfo():

```
Return "Name: " + name + "\nID: " + idNumber + "\nMajor: " + major + "\n"
```

Main Method:

```
Create student1 ← Student("Alice", "S101", "Computer Science")
```

```
Create student2 ← Student("Bob", "S102", "Mechanical Engineering")
```

```
Create student3 ← Student("Charlie", "S103", "Electrical Engineering")
```

```
Print student1.getInfo()
```

```
Print student2.getInfo()
```

```
Print student3.getInfo()
```

---

#### 4. Program Code

```
package day5;
```

```
public class Student {
```

```
    // Instance attributes
```

```
    String name;
```

```
    String idNumber;
```

```
    String major;
```

```
    // Constructor
```

```
    public Student(String name, String idNumber, String major) {
```

```
        this.name = name;
```

```
        this.idNumber = idNumber;
```

```
        this.major = major;
```

```
    }
```

```
    // Method to return student info
```

```
    public String getInfo() {
```

```
        return "Name: " + name + "\nID: " + idNumber + "\nMajor: " + major + "\n";
```

```

}

// Main method

public static void main(String[] args) {

    // Create three Student objects

    Student student1 = new Student("Alice", "S101", "Computer Science");

    Student student2 = new Student("Bob", "S102", "Mechanical Engineering");

    Student student3 = new Student("Charlie", "S103", "Electrical Engineering");

    // Print their info

    System.out.println(student1.getInfo());

    System.out.println(student2.getInfo());

    System.out.println(student3.getInfo());

}

}

```

## 5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"Afreem", "KIT005", "Computer Science"	Name: Afreem ID: KIT005 Major: Computer Science	Name: Afreem ID: KIT005 Major: Computer Science	Pass
2	"Shifa", "KIT102", "Mechanical Engineering"	Name: Shifa ID: KIT102	Name: Shifa ID: KIT102	Pass

		Major: Mechanical Engineering	Major: Mechanical Engineering	
3	" Spoorthi ", " KIT108", "Electrical Engineering"	Name: Spoorthi ID: KIT108 Major: Electrical Engineering	Name: Spoorthi ID: KIT108 Major: Electrical Engineering	Pass

## 6. Screenshots of Output

### Case :1

```
ID: KIT005
Major: Computer Science

Name: Shifa
ID: KIT102
Major: Mechanical Engineering

Name: Spoorthi
ID: KIT108
Major: Electrical Engineering
```

## 7. Observation / Reflection

This program helps us understand how to create a class to hold data for multiple students. Each student has unique values like name, ID, and major. The constructor initializes this data, and the getInfo() method helps return the details in a proper format. This is a good example of using classes and objects to handle structured information in Java. It also shows how to reuse methods and avoid repeating code.

### Problem 3.1: Bank Account

In this problem, we need to create a BankAccount class to handle a simple bank account. The account should store the balance and have three actions: one to check the balance (getBalance()), one to add money (deposit()), and one to take out money (withdraw()). We also need to test some invalid cases, like trying to deposit a negative amount or trying to withdraw more money than what's in the account. The program should show messages for these invalid operations.

---

### 2. Algorithm

1. Start
  2. Create a class BankAccount
  3. Add a private attribute balance
  4. Create a constructor that sets the initial balance (0 if it's negative)
  5. Create a method getBalance() to return the current balance
  6. Create a method deposit(amount):
    7. If amount is positive, add it to balance
    8. Else, show invalid message
  9. Create a method withdraw(amount):
    10. If amount is  $\leq 0$ , show invalid message
    11. If amount  $>$  balance, show insufficient funds
    12. Else, subtract amount from balance
  13. In the main() method:
    14. Create a BankAccount object
    15. Test valid and invalid deposit and withdrawal operations
    16. Print final balance
  17. End
-

### 3. Pseudocode

Class BankAccount:

Attribute:

balance

Constructor(initialBalance):

If initialBalance  $\geq$  0:

Set balance  $\leftarrow$  initialBalance

Else:

Print "Initial balance can't be negative"

Set balance  $\leftarrow$  0

Method getBalance():

Return balance

Method deposit(amount):

If amount  $>$  0:

balance  $\leftarrow$  balance + amount

Print "Deposited: " + amount

Else:

Print "Invalid deposit amount"

Method withdraw(amount):

If amount  $\leq$  0:

Print "Invalid withdrawal amount"

Else If amount  $>$  balance:

Print "Insufficient funds. Withdrawal denied."

Else:

balance  $\leftarrow$  balance - amount

Print "Withdrawn: " + amount



Main Method:

Create BankAccount with initial balance 1000

Call getBalance()

Call deposit(500)

Call deposit(-100)

Call withdraw(200)

Call withdraw(2000)

Call withdraw(-50)

Call getBalance()

---

#### 4. Program Code

```
package day5;
```

```
public class BankAccount {
```

```
    // Instance attribute
```

```
    private double balance;
```

```
    // Constructor
```

```
    public BankAccount(double initialBalance) {
```

```
        if (initialBalance >= 0) {
```

```
            this.balance = initialBalance;
```

```
        } else {
```

```
            System.out.println("Initial balance can't be negative. Setting to 0.");
```

```
            this.balance = 0;
```

```
        }
```

```
    }
```

```
    // Method to get current balance
```

```
    public double getBalance() {
```

```
        return balance;
```

```
}  
  
// Method to deposit money  
public void deposit(double amount) {  
    if (amount > 0) {  
        balance += amount;  
        System.out.println("Deposited: " + amount);  
    } else {  
        System.out.println("Invalid deposit amount.");  
    }  
}  
  
// Method to withdraw money  
public void withdraw(double amount) {  
    if (amount <= 0) {  
        System.out.println("Invalid withdrawal amount.");  
    } else if (amount > balance) {  
        System.out.println("Insufficient funds. Withdrawal denied.");  
    } else {  
        balance -= amount;  
        System.out.println("Withdrawn: " + amount);  
    }  
}  
  
// Main method to test the class  
public static void main(String[] args) {  
    BankAccount account = new BankAccount(1000.0);  
    System.out.println("Initial Balance: " + account.getBalance());  
    account.deposit(500.0);    // Valid  
    account.deposit(-100.0);  // Invalid
```

```

account.withdraw(200.0);    // Valid

account.withdraw(2000.0);   // Invalid (excessive)

account.withdraw(-50.0);    // Invalid

System.out.println("Final Balance: " + account.getBalance());
}
}

```

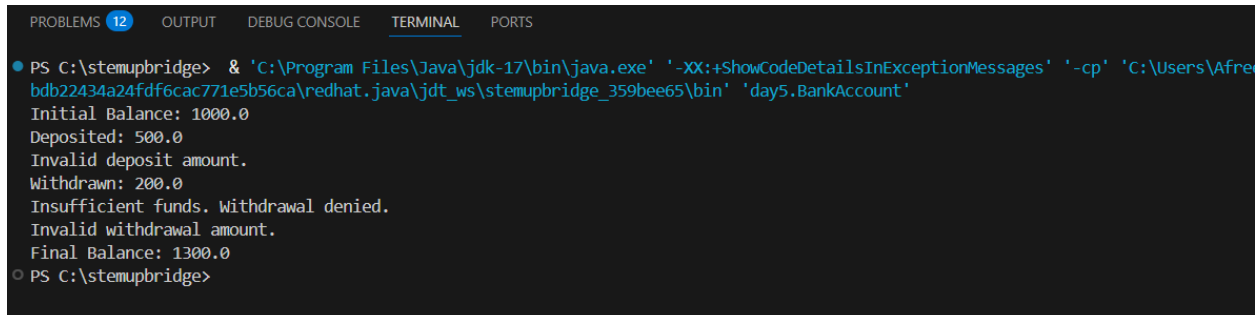
## 5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Valid Deposit and Withdrawal  Initial Balance: 1000 Actions: deposit(500), withdraw(200)	Deposited: 500  Withdrawn: 200  Final Balance: 1300.0	Deposited: 500  Withdrawn: 200  Final Balance: 1300.0	Pass
2	Invalid Deposit  Action: deposit(-100)	Invalid deposit amount.	Invalid deposit amount.	Pass
3	Excessive and Negative Withdrawal  Actions: withdraw(2000), withdraw(-50)	Insufficient funds. Withdrawal denied.  Invalid withdrawal amount.	Insufficient funds. Withdrawal denied.  Invalid withdrawal amount.	Pass

## 6. Screenshots of Output

Case 1:



```
PROBLEMS 12 OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\stemupbridge> & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afreen\OneDrive\Desktop\Stemup\src\day5\BankAccount\classes' 'day5.BankAccount'
Initial Balance: 1000.0
Deposited: 500.0
Invalid deposit amount.
Withdrawn: 200.0
Insufficient funds. Withdrawal denied.
Invalid withdrawal amount.
Final Balance: 1300.0
○ PS C:\stemupbridge>
```

---

## 7. Observation / Reflection

This program teaches how to create and use a simple banking system in Java. It includes good examples of **validation** (checking for invalid input) and **encapsulation** (using private variables with public methods). It helps understand how methods can control access and behavior based on user actions like deposit and withdrawal. This is useful for learning how real-life applications like banking apps work at a basic level.

### Problem 3.2: Product Inventory

We want to create a class `Product` that stores details like the name, price, and quantity of a product. We'll use **getters** to view values, **setters** to update values with rules (like price must be positive), and a method `getTotalValue()` to return the total price ( $\text{price} \times \text{quantity}$ ).

---

## 2. Algorithm

1. Start
  2. Create a `Product` class with attributes: name, price, and quantity
  3. Create a constructor to initialize values
  4. Add **getters** for all attributes
  5. Add **setters**:
  6. `setPrice()` allows only if  $\text{price} > 0$
  7. `setQuantity()` allows only if  $\text{quantity} \geq 0$
  8. Create a method `getTotalValue()` that returns  $\text{price} \times \text{quantity}$
  9. In `main()`, create a `Product` object, print details, try invalid values, update with valid values, and display final result
  10. End
- 

## 3. Pseudocode

Class `Product`:

Attributes:

name

price

quantity

Constructor(name, price, quantity):

Set name

Call `setPrice(price)`

Call `setQuantity(quantity)`

Method getName():

Return name

Method getPrice():

Return price

Method getQuantity():

Return quantity

Method setPrice(price):

If price > 0:

Set this.price  $\leftarrow$  price

Else:

Print "Invalid price"

Method setQuantity(quantity):

If quantity  $\geq$  0:

Set this.quantity  $\leftarrow$  quantity

Else:

Print "Invalid quantity"

Method getTotalValue():

Return price  $\times$  quantity

Main Method:

Create product  $\leftarrow$  Product("Laptop", 45000.0, 5)

Print product details using getters

Try setting invalid price and quantity

Set valid price and quantity

Print updated total value

---

#### 4. Program Code

```
package day5;
```

```
public class BankAccount {  
    // Instance attribute  
    private double balance;  
    // Constructor  
    public BankAccount(double initialBalance) {  
        if (initialBalance >= 0) {  
            this.balance = initialBalance;  
        } else {  
            System.out.println("Initial balance can't be negative. Setting to 0.");  
            this.balance = 0;  
        }  
    }  
    // Method to get current balance  
    public double getBalance() {  
        return balance;  
    }  
    // Method to deposit money  
    public void deposit(double amount) {  
        if (amount > 0) {  
            balance += amount;  
            System.out.println("Deposited: " + amount);  
        } else {  
            System.out.println("Invalid deposit amount.");  
        }  
    }  
}
```

```
// Method to withdraw money

public void withdraw(double amount) {
    if (amount <= 0) {
        System.out.println("Invalid withdrawal amount.");
    } else if (amount > balance) {
        System.out.println("Insufficient funds. Withdrawal denied.");
    } else {
        balance -= amount;
        System.out.println("Withdrawn: " + amount);
    }
}

// Main method to test the class

public static void main(String[] args) {
    BankAccount account = new BankAccount(1000.0);
    System.out.println("Initial Balance: " + account.getBalance());
    account.deposit(500.0);    // Valid
    account.deposit(-100.0);   // Invalid
    account.withdraw(200.0);   // Valid
    account.withdraw(2000.0);  // Invalid (excessive)
    account.withdraw(-50.0);   // Invalid
    System.out.println("Final Balance: " + account.getBalance());
}
}
```

---



## 5. Test Cases

Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	name = "Laptop", price = 45000, quantity = 5	Total Value = 225000	Total Value = 225000	Pass
2	price = -1000	Invalid price. Price remains unchanged.	Invalid price. Price remains unchanged.	Pass
3	quantity = -3	Invalid quantity. Quantity remains unchanged.	Invalid quantity. Quantity remains unchanged.	Pass

## 6. Screenshots of Output

### Case 1:

```

bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'day5.Product'
Product: Laptop
Price: 45000.0
Quantity: 5
Total Value: 225000.0
Invalid price. Must be positive.
Invalid quantity. Cannot be negative.

Updated Details:
Price: 50000.0
Quantity: 3
Total Value: 150000.0
PS C:\stemupbridge>
  
```

## 7. Observation / Reflection

This program teaches how to use **encapsulation** by using private variables and public getters/setters. The getTotalValue() method is a good example of combining data to get a result. The use of validations helps prevent wrong data (like negative price or quantity). It's a useful structure for creating shopping or billing systems where product data must be handled carefully.