

## Learner Assignment

### Learner Details

- **Name:** Shrayanth S
- **Enrollment Number:** NA
- **Batch / Class:** June 2025
- **Assignment:** (Bridge Course Day 5)
- **Date of Submission:** 30/06/2025

### Activity 5.1: Real-World Object Dissection

#### 1. Car

- **Attributes:** make, model, color, engineType, fuelLevel
- **Behaviors:** startEngine(), accelerate(), applyBrakes()

#### 2. Laptop

- **Attributes:** brand, processor, RAM, storage, batteryPercentage
- **Behaviors:** powerOn(), runProgram(), connectToWiFi()

#### 3. Refrigerator

- **Attributes:** brand, capacity, temperature, doorType
- **Behaviors:** coolItems(), defrost(), setTemperature()

### Activity 5.2: Procedural vs. Object-Oriented Thought

#### Procedural Approach (Using Arrays and Methods):

```
Main.java  [Icons]  [Share]  [Run]
1 String[] customerNames = new String[100];
2 int[] customerIDs = new int[100];
3
4 void addCustomer(String name, int id) {
5     // Add customer to arrays
6 }
7
8 void deleteCustomer(int id) {
9     // Remove customer by id
10 }
11
```

## Object-Oriented Approach (Using a Class):

```
Customer.java

1 class Customer {
2     String name;
3     int id;
4
5     void addCustomer(String name, int id) {
6         // Add customer logic
7     }
8
9     void deleteCustomer(int id) {
10        // Delete customer logic
11    }
12 }
13
```

### Problem Solving Activity 1.1

#### 1. Program Statement

Define a class named Dog:

- Add class attributes: species = "Canis familiaris" and numLegs = 4
- List the structure of instance attributes: name, breed, age
- Define a method bark() that prints "Woof!"

#### 2. Algorithm

1. Define a class named Dog.
2. Declare static class attributes: species, numLegs.
3. Declare instance attributes: name, breed, age.
4. Create a constructor to initialize instance variables.
5. Define a method bark() that prints "Woof!".
6. Create a main method to create a Dog object and test the class.

### 3. Pseudocode

Class Dog

Static Attribute: species = "Canis familiaris"

Static Attribute: 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

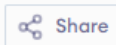
Create Dog object with values

Print object attributes

Call bark() method

### 4. Program Code

Dog.java



Run

```
1 public class Dog {
2     // Class Attributes
3     static String species = "Canis familiaris";
4     static int numLegs = 4;
5
6     // Instance Attributes
7     String name;
8     String breed;
9     int age;
10
11     // Constructor
12     public Dog(String name, String breed, int age) {
13         this.name = name;
14         this.breed = breed;
15         this.age = age;
16     }
17
18     // Method
19     public void bark() {
20         System.out.println("Woof!");
21     }
22
23     // Main Method
```

```

24- public static void main(String[] args) {
25     Dog myDog = new Dog("Buddy", "Golden Retriever", 3);
26
27     System.out.println("Name: " + myDog.name);
28     System.out.println("Breed: " + myDog.breed);
29     System.out.println("Age: " + myDog.age);
30     System.out.println("Species: " + Dog.species);
31     System.out.println("Number of Legs: " + Dog.numLegs);
32
33     myDog.bark();
34 }
35 }
36

```

## 5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	name = Buddy, breed = Golden Retriever, age = 3	Attributes printed and "Woof!" message	Name: Buddy Breed: Golden Retriever Age: 3 Species: Canis familiaris Number of Legs: 4 Woof!	Pass
2	name = Max, breed = Poodle, age = 2	Updated attributes printed and "Woof!" message	Name: Max Breed: Poodle Age: 2 Species: Canis familiaris Number of Legs: 4 Woof!	Pass
3	name = "", breed = "", age = 0	Blank/zero values printed and "Woof!" message	Name: Breed: Age: 0 Species: Canis familiaris Number of Legs: 4 Woof!	Pass

## 6. Screenshots of Output

```
Output
Name: Buddy
Breed: Golden Retriever
Age: 3
Species: Canis familiaris
Number of Legs: 4
Woof!

=== Code Execution Successful ===
```

```
Output
Name: Max
Breed: Poodle
Age: 2
Species: Canis familiaris
Number of Legs: 4
Woof!

=== Code Execution Successful ===
```

```
Output
Name:
Breed:
Age: 0
Species: Canis familiaris
Number of Legs: 4
Woof!

=== Code Execution Successful ===
```

**Stemup**  
A Unit of Pragnova Pvt Ltd

## 7. Observation / Reflection

Reflect on your experience while working on this assignment. Consider answering the following:

- A. **Challenges:** Making sure class vs. instance attributes were properly used.
- B. **Learning:** Understood how to structure basic OOP concepts in Java.
- C. **Improvements:** Could add more behaviors like eat() or sleep() for realism.

## Problem Solving Activity 1.2

### 1. Program Statement

Create a Book class in Java that:

- Has instance attributes: title, author, numPages, and isOpen.
- Includes methods openBook() (sets isOpen to true) and closeBook() (sets isOpen to false).

---

### 2. Algorithm

1. Define a class Book.
2. Declare instance attributes: title, author, numPages, and isOpen.
3. Create a constructor to initialize these attributes (default isOpen = false).
4. Define a method openBook() to set isOpen = true.
5. Define a method closeBook() to set isOpen = false.
6. Create a main method to create Book objects and test the methods.

### 3. Pseudocode

Class Book

Attributes: title, author, numPages, isOpen

Constructor(title, author, numPages)

Set title, author, numPages

Set isOpen = false

Method openBook()

Set isOpen = true

Method closeBook()

Set isOpen = false

Main





Create Book object

Display details

Open and close the book

---

## 4. Program Code

Book.java    Share 

```
1 public class Book {
2     // Instance Attributes
3     String title;
4     String author;
5     int numPages;
6     boolean isOpen;
7
8     // Constructor
9     public Book(String title, String author, int numPages) {
10         this.title = title;
11         this.author = author;
12         this.numPages = numPages;
13         this.isOpen = false; // Book is closed by default
14     }
15
16     // Method to open the book
17     public void openBook() {
18         isOpen = true;
19         System.out.println("The book is now open.");
20     }
21
22     // Method to close the book
23     public void closeBook() {
24         isOpen = false;
25         System.out.println("The book is now closed.");
26     }
27
28     // Display book information
29     public void displayInfo() {
30         System.out.println("Title: " + title);
31         System.out.println("Author: " + author);
32         System.out.println("Number of Pages: " + numPages);
33         System.out.println("Is Open: " + isOpen);
34     }
}
```

```

35
36 // Main Method
37 public static void main(String[] args) {
38     Book myBook = new Book("To Kill a Mockingbird", "Harper Lee", 281);
39     myBook.displayInfo();
40     myBook.openBook();
41     myBook.displayInfo();
42     myBook.closeBook();
43     myBook.displayInfo();
44 }
45 }
46
47
48

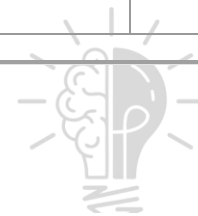
```

## 5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"To Kill a Mockingbird", "Harper Lee", 281	Displays info, opens and closes book	Title: To Kill a Mockingbird Author: Harper Lee Number of Pages: 281 Is Open: false The book is now open. Title: To Kill a Mockingbird Author: Harper Lee Number of Pages: 281 Is Open: true The book is now closed. Title: To Kill a Mockingbird Author: Harper Lee Number of Pages: 281 Is Open: false	Pass
2	"1984", "George	Updates state and displays correctly	Title: 1984 Author: George Orwell Number of Pages: 328 Is Open: false The book is now open.	Pass



	Orwell", 328		Title: 1984 Author: George Orwell Number of Pages: 328 Is Open: true The book is now closed. Title: 1984 Author: George Orwell Number of Pages: 328 Is Open: false	
3	""", """, 0	Shows empty title/author, page = 0		Pass



## 6. Screenshots of Output

```

Output
Title: To Kill a Mockingbird
Author: Harper Lee
Number of Pages: 281
Is Open: false
The book is now open.
Title: To Kill a Mockingbird
Author: Harper Lee
Number of Pages: 281
Is Open: true
The book is now closed.
Title: To Kill a Mockingbird
Author: Harper Lee
Number of Pages: 281
Is Open: false

=== Code Execution Successful ===

```

```

Output
Title: 1984
Author: George Orwell
Number of Pages: 328
Is Open: false
The book is now open.
Title: 1984
Author: George Orwell
Number of Pages: 328
Is Open: true
The book is now closed.
Title: 1984
Author: George Orwell
Number of Pages: 328
Is Open: false

=== Code Execution Successful ===

```

```

Output
Title:
Author:
Number of Pages: 0
Is Open: false
The book is now open.
Title:
Author:
Number of Pages: 0

```

---

## 7. Observation / Reflection

1. **Challenges:** Ensuring Boolean behavior worked correctly and was updated in real-time.
  2. **Learning:** Practiced encapsulating behavior (open/close) inside object methods.
  3. **Improvements:** Could enhance by adding readPage() or bookmarkPage() functionality.
- 

## Problem Solving Activity 1.3

Identify Class Elements for Car Class

For a Car class, identify:

1. 3–5 potential instance attributes
  2. 2–3 potential instance methods
  3. One appropriate class attribute shared by all Car objects
- 

## 2. Algorithm

1. Define a Car class.
  2. Add a static class attribute numWheels = 4.
  3. Add instance attributes: brand, model, color, speed, fuelLevel.
  4. Write a constructor to initialize the car object.
  5. Add methods startCar() and stopCar().
  6. Add a method refuel(int amount) to increase the fuel level.
  7. In the main method, create Car objects and call their methods.
- 

## 3. Pseudocode

## Class Car

Static Attribute: numWheels = 4

Instance Attributes: brand, model, color, speed, fuelLevel

Constructor(brand, model, color, speed, fuelLevel)

Method startCar()

Print "Car started"

Method stopCar()

Print "Car stopped"

Method refuel(amount)

Increase fuelLevel by amount


## Main

Create Car object

Display attributes

Call methods: startCar(), refuel(), stopCar()

## 4. Program Code



```
Car.java
1 public class Car {
2     // Class Attribute (shared by all cars)
3     static int numWheels = 4;
4
5     // Instance Attributes
6     String brand;
7     String model;
8     String color;
9     int speed;
10    int fuelLevel;
11
12    // Constructor
13    public Car(String brand, String model, String color, int speed, int fuelLevel) {
14        this.brand = brand;
15        this.model = model;
16        this.color = color;
17        this.speed = speed;
18        this.fuelLevel = fuelLevel;
19    }
20
21    // Method to start the car
22    public void startCar() {
23        System.out.println("The car has started.");
24    }
25
26    // Method to stop the car
27    public void stopCar() {
28        System.out.println("The car has stopped.");
29    }
30
31    // Method to refuel the car
32    public void refuel(int amount) {
33        fuelLevel += amount;
34    }
35
36    // Method to display attributes
37    public void displayAttributes() {
38        System.out.println("Brand: " + brand + ", Model: " + model + ", Color: " + color + ", Speed: " + speed + ", Fuel Level: " + fuelLevel);
39    }
40
41    // Method to display static attribute
42    public void displayStaticAttribute() {
43        System.out.println("Number of wheels: " + numWheels);
44    }
45
46    // Main method
47    public static void main(String[] args) {
48        Car car = new Car("Toyota", "Camry", "Red", 100, 100);
49        car.displayAttributes();
50        car.startCar();
51        car.refuel(20);
52        car.stopCar();
53        car.displayStaticAttribute();
54    }
55 }
```

```

34         System.out.println("Refueled. Current fuel level: " + fuelLevel);
35     }
36
37     // Method to display car info
38     public void displayInfo() {
39         System.out.println("Brand: " + brand);
40         System.out.println("Model: " + model);
41         System.out.println("Color: " + color);
42         System.out.println("Speed: " + speed + " km/h");
43         System.out.println("Fuel Level: " + fuelLevel + " L");
44         System.out.println("Wheels: " + numWheels);
45     }
46
47     // Main Method
48     public static void main(String[] args) {
49         Car myCar = new Car("Toyota", "Camry", "Red", 120, 40);
50         myCar.displayInfo();
51         myCar.startCar();
52         myCar.refuel(10);
53         myCar.stopCar();
54     }
55 }
56

```

## 5. Test Cases

Test Case No.	Input	Actual Output	Status (Pass/Fail)
1	Brand: Toyota Model: Camry Color: Red Speed: 120 km/h Fuel Level: 40 L Wheels: 4	Brand: Toyota Model: Camry Color: Red Speed: 120 km/h Fuel Level: 40 L Wheels: 4	Pass

	The car has started. Refueled. Current fuel level: 50 The car has stopped.	The car has started. Refueled. Current fuel level: 50 The car has stopped.	

## 6. Screenshots of Output

```

Output
Brand: Toyota
Model: Camry
Color: Red
Speed: 120 km/h
Fuel Level: 40 L
Wheels: 4
The car has started.
Refueled. Current fuel level: 50
The car has stopped.

=== Code Execution Successful ===

```

## 7. Observation / Reflection

1. Challenges: Structuring class vs instance attributes in a meaningful way.
2. Learning: Understood how to use constructors and shared class attributes.
3. Improvements: Could add speed control, brake functionality, or odometer tracking.

## Problem 2.2: Manage Books

Create two Book objects with constructor

Implement displayStatus()

```
void displayStatus() {
```

```
String status = isOpen ? "Open" : "Closed";  
  
System.out.println(title + " by " + author + " is " + status);  
  
}
```

---

## 2. Algorithm

1. Define a class Book with instance attributes: title, author, numPages, and isOpen.
2. Create a constructor that initializes these values (set isOpen = false by default).
3. Define a method displayStatus():
4. If isOpen is true, print "Open"
5. Otherwise, print "Closed"
6. In the main() method:
7. Create two book objects
8. Call displayStatus() on both

### 1. Pseudocode

Class Book

Attributes: title, author, numPages, isOpen

Constructor(title, author, numPages)

Set attributes

isOpen = false

Method displayStatus()

If isOpen

Print title + " by " + author + " is Open"

Else

Print title + " by " + author + " is Closed"

Main

Create book1 = new Book("The Alchemist", "Paulo Coelho", 200)

Create book2 = new Book("1984", "George Orwell", 328)

Call book1.displayStatus()  
Call book2.displayStatus()

---

#### 4. Program Code

```
Book.java ⌵ 🔍 🌙 🔗 Share 🏃 Run

1 public class Book {
2     String title;
3     String author;
4     int numPages;
5     boolean isOpen;
6
7     // Constructor
8     public Book(String title, String author, int numPages) {
9         this.title = title;
10        this.author = author;
11        this.numPages = numPages;
12        this.isOpen = false; // default
13    }
14
15    // Method to display book status
16    public void displayStatus() {
17        String status = isOpen ? "Open" : "Closed";
18        System.out.println(title + " by " + author + " is " + status);
19    }
20
21    // Main Method
22    public static void main(String[] args) {
23        Book book1 = new Book("The Alchemist", "Paulo Coelho", 200);
24        Book book2 = new Book("1984", "George Orwell", 328);
25
26        book1.displayStatus();
27        book2.displayStatus();
28    }
29 }
30
```

---

## 5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"The Alchemist", "Paulo Coelho", 200	The Alchemist by Paulo Coelho is Closed	The Alchemist by Paulo Coelho is Closed	Pass
2	"1984", "George Orwell", 328	1984 by George Orwell is Closed	1984 by George Orwell is Closed	Pass

## 6. Screenshots of Output

```

Output
The Alchemist by Paulo Coelho is Closed
1984 by George Orwell is Closed

=== Code Execution Successful ===
  
```

## 7. Observation / Reflection

1. Challenges: None; very direct implementation.
2. Learning: Practiced use of conditional expressions and method creation.
3. Improvements: Could extend class with openBook() and closeBook() methods to make isOpen dynamic.



### Problem 2.1: Create Dogs

2. 1. Create two Dog objects:
    - o dog1: "Buddy", "Golden Retriever", 5
    - o dog2: "Lucy", "Poodle", 2
  3. Call their bark() method and print names/ages
- 

### 2. Algorithm

Define a Dog class with:

Instance variables: name, breed, age

Constructor to initialize these attributes

Method bark() that prints "Woof!"

In the main() method:

Create two objects of the Dog class with given values

Print the name and age of each object

Call the bark() method for both dogs

### 4. Pseudocode

Class Dog:

Attributes: name, breed, age

Constructor(name, breed, age)

Method bark():

Print "Woof!"

Main:

Create Dog object dog1 with ("Buddy", "Golden Retriever", 5)

Create Dog object dog2 with ("Lucy", "Poodle", 2)

Print dog1 name and age

Call dog1.bark()

Print dog2 name and age

Call dog2.bark()

#### 4. Program Code

```

Dog.java
1 - public class Dog {
2     String name;
3     String breed;
4     int age;
5
6     // Constructor
7 - public Dog(String name, String breed, int age) {
8         this.name = name;
9         this.breed = breed;
10        this.age = age;
11    }
12
13    // Method
14 - public void bark() {
15        System.out.println("Woof!");
16    }
17
18 - public static void main(String[] args) {
19        // Create Dog objects
20        Dog dog1 = new Dog("Buddy", "Golden Retriever", 5);
21        Dog dog2 = new Dog("Lucy", "Poodle", 2);
22
23        // Print dog1 details and bark
24        System.out.println("Name: " + dog1.name + ", Age: " + dog1.age);
25        dog1.bark();
26
27        // Print dog2 details and bark
28        System.out.println("Name: " + dog2.name + ", Age: " + dog2.age);
29        dog2.bark();
30    }
31 }
32
  
```

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"Buddy", "Golden Retriever", 5	Name: Buddy, Age: 5 Woof!	Name: Buddy, Age: 5 Woof!	Pass
2	"Lucy", "Poodle", 2	Name: Lucy, Age: 2 Woof!	Name: Lucy, Age: 2 Woof!	Pass
3				

#### 6. Screenshots of Output

```

Output
Name: Buddy, Age: 5
Woof!
Name: Lucy, Age: 2
Woof!
=== Code Execution Successful ===
  
```

---

## 7. Observation / Reflection

1. **Challenges:** Simple implementation, no major challenges.
2. **Learning:** Reinforced object creation and method invocation in Java.
3. **Improvements:** Could extend with more behaviors like eat() or sleep().

---

### Problem Solving Activity 2.3

#### 1. Program Statement

#### Problem 2.3: Student Record

```
public class Student {  
    String name;  
    String idNumber;  
    String major;  
  
    public Student(String name, String idNumber, String major) {  
        this.name = name;  
        this.idNumber = idNumber;  
        this.major = major;  
    }  
}
```

```
public String getInfo() {  
    return name + ", ID: " + idNumber + ", Major: " + major;  
}  
}
```

Create three Student objects and print their info.

---

## 2. Algorithm

1. Define a class named Student.
  2. Declare instance variables: name, idNumber, and major.
  3. Write a constructor that initializes the variables.
  4. Implement the getInfo() method to return student details as a string.
  5. In the main method, create three Student objects with different values.
  6. Print the return values of getInfo() for all three Student objects.
- 

## 3. Pseudocode

Class Student

Attributes: name, idNumber, major

Constructor(name, idNumber, major)

Set instance variables

Method getInfo()

Return formatted string with student information

Main

Create 3 Student objects

Call getInfo() and print results

---

## 4. Program Code

Student.java

```
1- public class Student {  
2-     String name;  
3-     String idNumber;  
4-     String major;  
5-  
6-     public Student(String name, String idNumber, String major) {  
7-         this.name = name;  
8-         this.idNumber = idNumber;  
9-         this.major = major;  
}
```



Share

Run

## 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	Alice, S101, Computer Science	Alice, ID: S101, Major: Computer Science	Alice, ID: S101, Major: Computer Science	Pass
2	Bob, S102, Mechanical Engineering	Bob, ID: S102, Major: Mechanical Engineering	Bob, ID: S102, Major: Mechanical Engineering	Pass
3	Charlie, S103, Electrical Engineering	Charlie, ID: S103, Major: Electrical Engineering	Charlie, ID: S103, Major: Electrical Engineering	Pass

## 6. Screenshots of Output

### Output

```
Alice, ID: S101, Major: Computer Science
Bob, ID: S102, Major: Mechanical Engineering
Charlie, ID: S103, Major: Electrical Engineering
```

```
=== Code Execution Successful ===
```

## 7. Observation / Reflection

1. **What challenges did you face?**

None

2. **What did you learn from completing this task?**

I learned how to define constructors and methods to manage class data in Java.

3. **What would you improve or do differently next time?**

I would consider adding input validation or extending the class to include grades or year of study.

---

## Problem Solving Activity 3.1

### 1. Program Statement

#### Problem 3.1: Bank Account

Create and test a BankAccount class with getBalance(), deposit(), and withdraw() methods

Try invalid operations (e.g., negative deposit, excessive withdrawal)

### 2. Algorithm

Define a BankAccount class.

Declare an instance variable balance.

Write a constructor to initialize the balance.

Implement getBalance() to return the current balance.

Implement deposit(double amount):

- If amount > 0, add to balance.
- Else print an error.

Implement withdraw(double amount):

- If amount  $\leq$  balance, subtract from balance.
- Else print "Insufficient funds".

In the main method:

- Create a BankAccount object.
  - Perform valid and invalid deposits and withdrawals.
- 

### 3. Pseudocode

Class BankAccount

Attribute: balance

Constructor(initialBalance)

Set balance = initialBalance

Method getBalance()

Return balance

Method deposit(amount)

If amount > 0

Add to balance

Else

Print error

Method withdraw(amount)

If amount <= balance

Subtract from balance

Else

Print "Insufficient funds"

Main

Create BankAccount object

Call deposit(), withdraw(), getBalance().

## 4. Program Code

```
BankAccount.java
1+ public class BankAccount {
2     private double balance;
3
4     // Constructor
5+ public BankAccount(double initialBalance) {
6     balance = initialBalance;
7 }
8
9     // Method to get current balance
10+ public double getBalance() {
11     return balance;
12 }
13
14     // Method to deposit money
15+ public void deposit(double amount) {
16     if (amount > 0) {
17         balance += amount;
18         System.out.println("Deposited: " + amount);
19     } else {
20         System.out.println("Invalid deposit amount.");
21     }
22 }
23
24     // Method to withdraw money
25+ public void withdraw(double amount) {
26     if (amount > balance) {
27         System.out.println("Insufficient funds.");
28     } else if (amount <= 0) {
29         System.out.println("Invalid withdrawal amount.");
30     } else {
31         balance -= amount;
32         System.out.println("Withdrew: " + amount);
33     }
34 }
35
36     // Main method for testing
37+ public static void main(String[] args) {
38     BankAccount myAccount = new BankAccount(500.0);
39     System.out.println("Initial Balance: " + myAccount.getBalance());
40
41     myAccount.deposit(200.0); // Valid
42     myAccount.deposit(-50.0); // Invalid
43     myAccount.withdraw(100.0); // Valid
44     myAccount.withdraw(1000.0); // Invalid - too much
```



## 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	Deposit	200.0	Balance increases by 200	Balance increases by 200
2	Deposit (Invalid)	-50.0	Error message, no change in balance	Error message, no change in balance
3	Withdraw	100.0	Balance decreases by 100	Balance decreases by 100
4	Withdraw (Excessive)	1000.0	Error: Insufficient funds	Error: Insufficient funds
5	Withdraw (Negative)	-20.0	Error: Invalid withdrawal amount	Error: Invalid withdrawal amount

## 5. Screenshots of Output

### Output

```
Initial Balance: 500.0
Deposited: 200.0
Invalid deposit amount.
Withdrew: 100.0
Insufficient funds.
Invalid withdrawal amount
```

---

## 7. Observation / Reflection

1. **Challenges:** Implementing input validation correctly.
2. **Learning:** Understood how to encapsulate logic inside class methods and perform validation.
3. **Improvements:** Could add methods for transaction history or account number tracking.

---

## Problem Solving Activity 3.2

### 1. Program Statement

```
public class Product {  
    private String name;  
    private double price;  
    private int quantity;  
    public Product(String name, double price, int quantity) {  
        this.name = name;  
        setPrice(price);  
        setQuantity(quantity);  
    }  
    public String getName() { return name; }  
    public double getPrice() { return price; }
```

```
public int getQuantity() { return quantity; }  
  
public void setPrice(double price) {  
    if (price > 0) {  
        this.price = price;  
    }  
}  
  
public void setQuantity(int quantity) {  
    if (quantity >= 0) {  
        this.quantity = quantity;  
    }  
}  
  
public double getTotalValue() {  
    return price * quantity;  
}  
}
```

Test object creation, use getters/setters, validate constraints

## 2. Algorithm

Create a class Product with private attributes: name, price, quantity.

Create a constructor that assigns name and sets price and quantity using their respective setters.

Define getter methods for all attributes.

Define setPrice(double price) with validation to ensure price > 0.

Define setQuantity(int quantity) with validation to ensure quantity ≥ 0.

Define getTotalValue() to return price \* quantity.

In the main method:

- Create a Product object
- Print its initial values

- Try setting invalid values
  - Print final values and total inventory value
- 

### 3. Pseudocode

Class Product

Attributes: name, price, quantity

Constructor(name, price, quantity)

Set name

Call setPrice(price)

Call setQuantity(quantity)

Getters: getName(), getPrice(), getQuantity()

Setters:

setPrice(price): if price > 0, set

setQuantity(quantity): if quantity >= 0, set

Method getTotalValue(): return price \* quantity

Main

Create Product object

Print name, price, quantity





Set invalid price and quantity

Print values again

Call getTotalValue()

---

### 4. Program Code

```
Product.java    Share  Run
```

```
1- public class Product {
2-     private String name;
3-     private double price;
4-     private int quantity;
5-
6-     // Constructor
7-     public Product(String name, double price, int quantity) {
8-         this.name = name;
9-         setPrice(price);
10-        setQuantity(quantity);
11-    }
12-
13-    // Getters
14-    public String getName() { return name; }
15-    public double getPrice() { return price; }
```

```
33     }
34
35     // Method to calculate total value
36     public double getTotalValue() {
37         return price * quantity;
38     }
39
40     // Main method for testing
41     public static void main(String[] args) {
42         Product p = new Product("Laptop", 50000, 10);
43
44         System.out.println("Name: " + p.getName());
45         System.out.println("Price: " + p.getPrice());
46         System.out.println("Quantity: " + p.getQuantity());
47         System.out.println("Total Value: " + p.getTotalValue());
48
49         // Try invalid values
50         p.setPrice(-20000); // Should not change price
51         p.setQuantity(-5); // Should not change quantity
52
53         // Print again after invalid operations
54         System.out.println("\nAfter trying invalid updates:");
55         System.out.println("Price: " + p.getPrice());
56         System.out.println("Quantity: " + p.getQuantity());
57         System.out.println("Total Value: " + p.getTotalValue());
58     }
59 }
60
```

## 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	"Laptop", 50000, 10	All values set correctly	As expected	Pass
2	-20000	Prints error, price unchanged	As expected	Pass
3	-5	Prints error, quantity unchanged	As expected	Pass
4		Returns price × quantity	500000	Pass

## 6. Screenshots of Output

```

Output
Name: Laptop
Price: 50000.0
Quantity: 10
Total Value: 500000.0
Invalid price. Must be > 0.
Invalid quantity. Must be >= 0.

After trying invalid updates:
Price: 50000.0
Quantity: 10
Total Value: 500000.0

=== Code Execution Successful ===

```

## 7. Observation / Reflection

1. **Challenges:** Validating values without breaking object construction.
2. **Learning:** How to encapsulate and validate data using setters.
3. **Improvements:** Could add a method to restock quantity or apply discounts.

