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

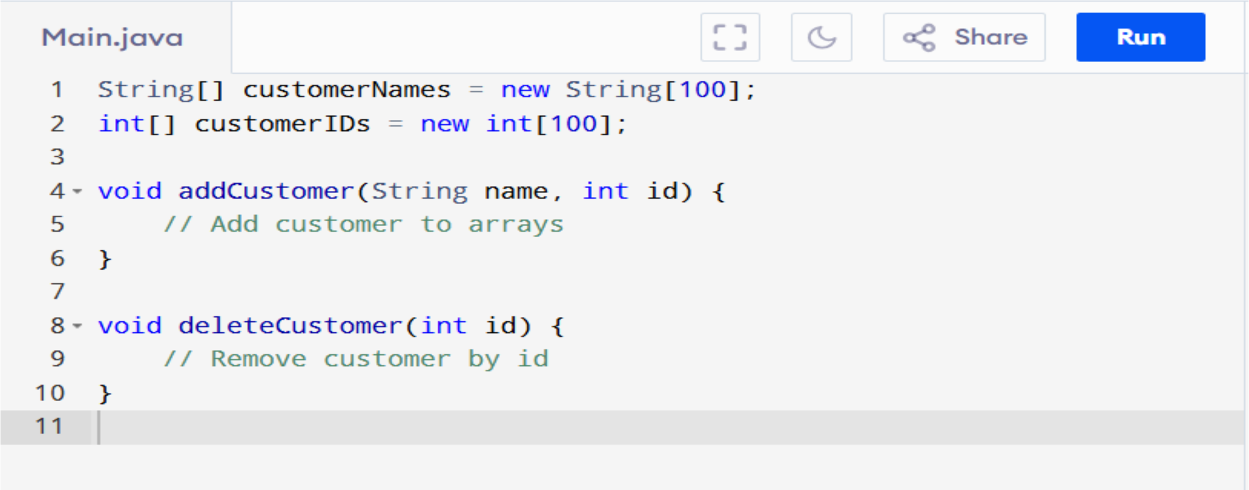
- Attributes: brand, processor, RAM, storage, batteryPercentage
- Behaviors: powerOn(), runProgram(), connectToWiFi()

3. Refrigerator

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

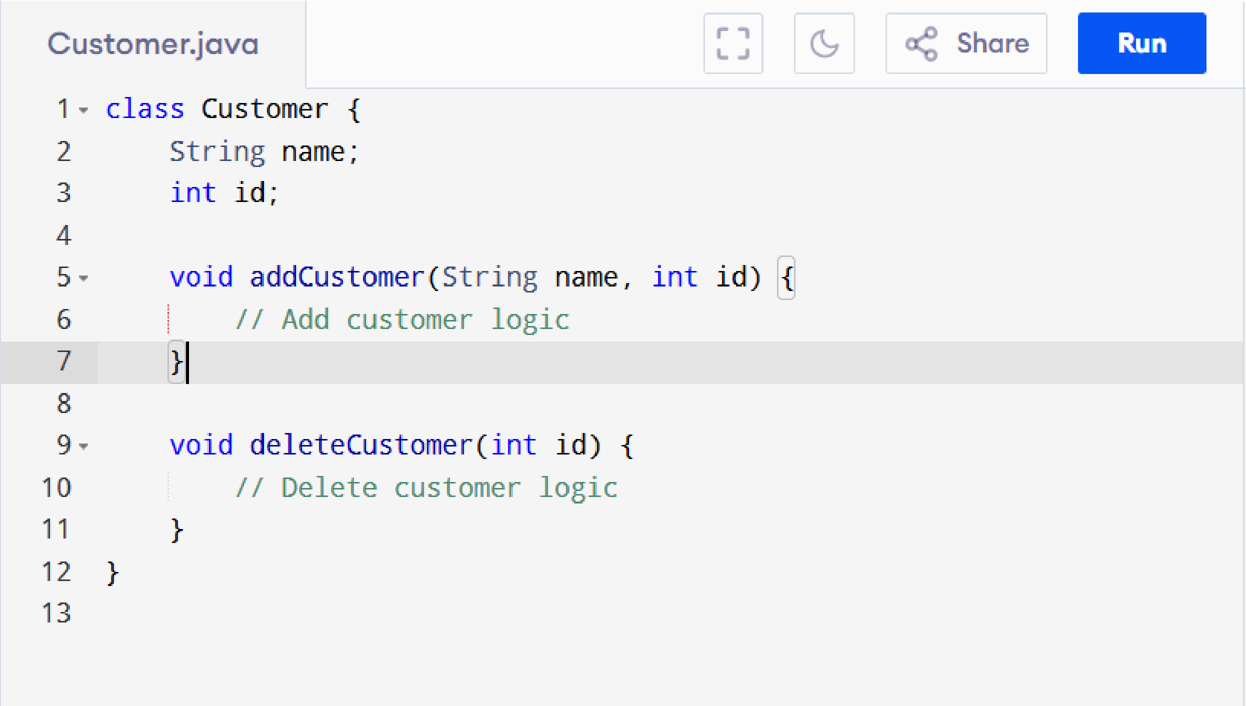
Activity 5.2: Procedural vs. Object-Oriented Thought

Procedural Approach (Using Arrays and Methods):



```java
String[] customerNames = new String[100];
int[] customerIDs = new int[100];

void addCustomer(String name, int id) {
    // Add customer to arrays
}

void deleteCustomer(int id) {
    // Remove customer by id
}

```

Object-Oriented Approach (Using a Class):



```java
class Customer {
    String name;
    int id;

    void addCustomer(String name, int id) {
        // Add customer logic
    }

    void deleteCustomer(int id) {
        // Delete customer logic
    }
}

```

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!"

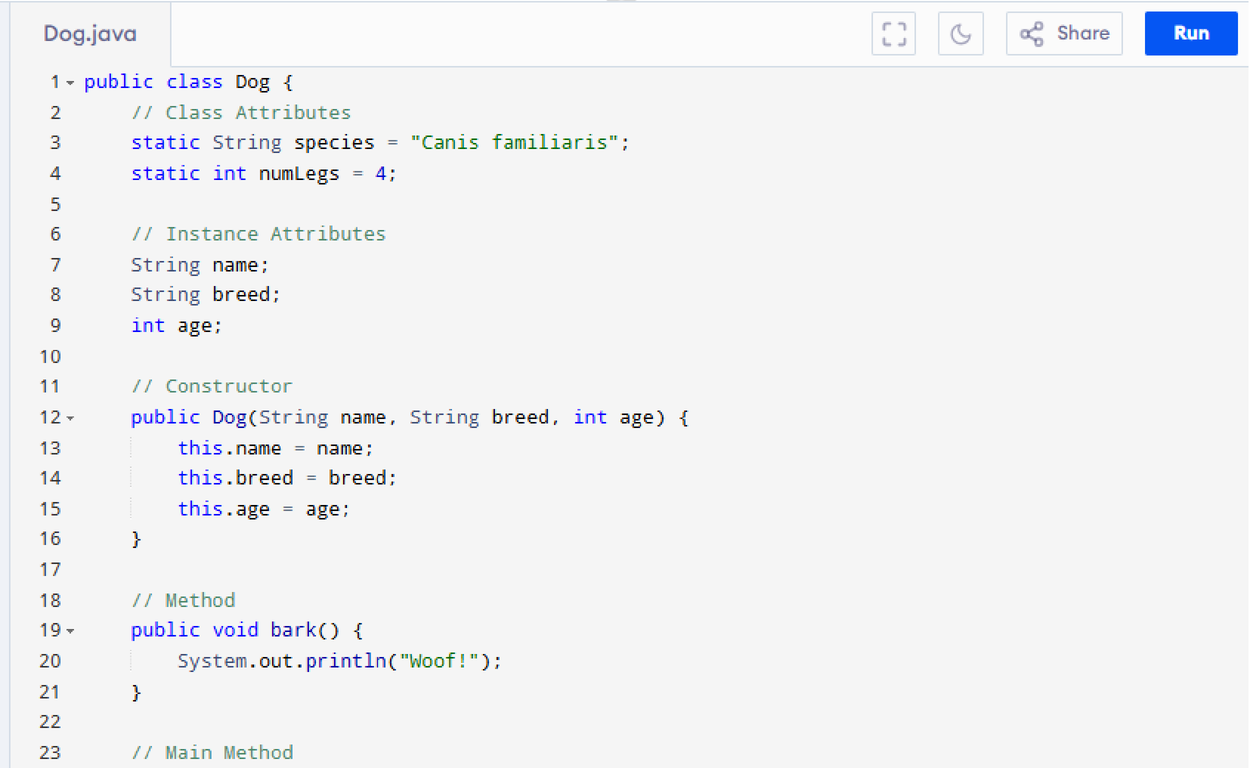
2. Algorithm

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.

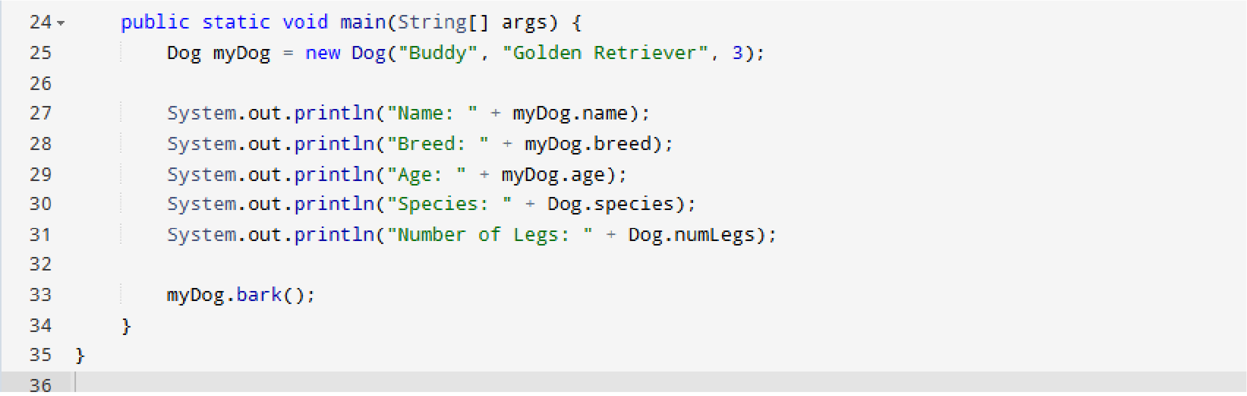
3. Pseudocode

```
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



```java
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
    public void bark() {
        System.out.println("Woof!");
    }

    // Main Method
```

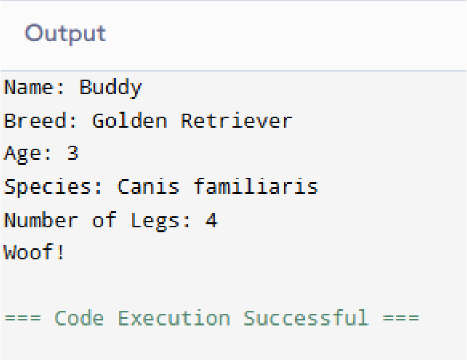


```java
    public static void main(String[] args) {
        Dog myDog = new Dog("Buddy", "Golden Retriever", 3);

        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("Number of Legs: " + Dog.numLegs);

        myDog.bark();
    }
}
```

5. Test Cases

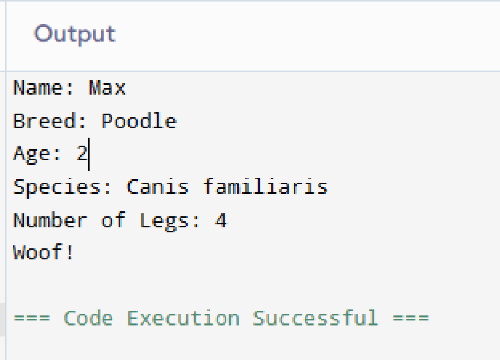
Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	name = Buddy, breed = Golden Retriever, age = 3	Attributes printed and "Woof!" message	Name: Buddy Breed: Golden Retriever Age: 3 Species: Canis familiaris Number of Legs: 4 Woof!	Pass
2	name = Max, breed = Poodle, age = 2	Updated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



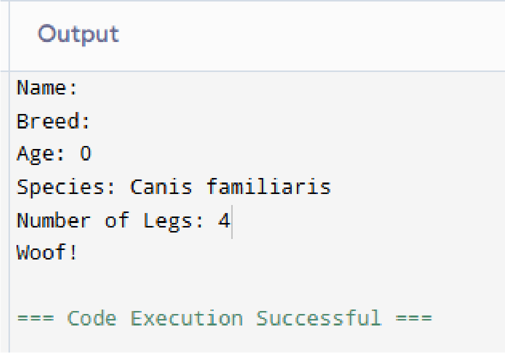
Output

```
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 ===
```

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
```

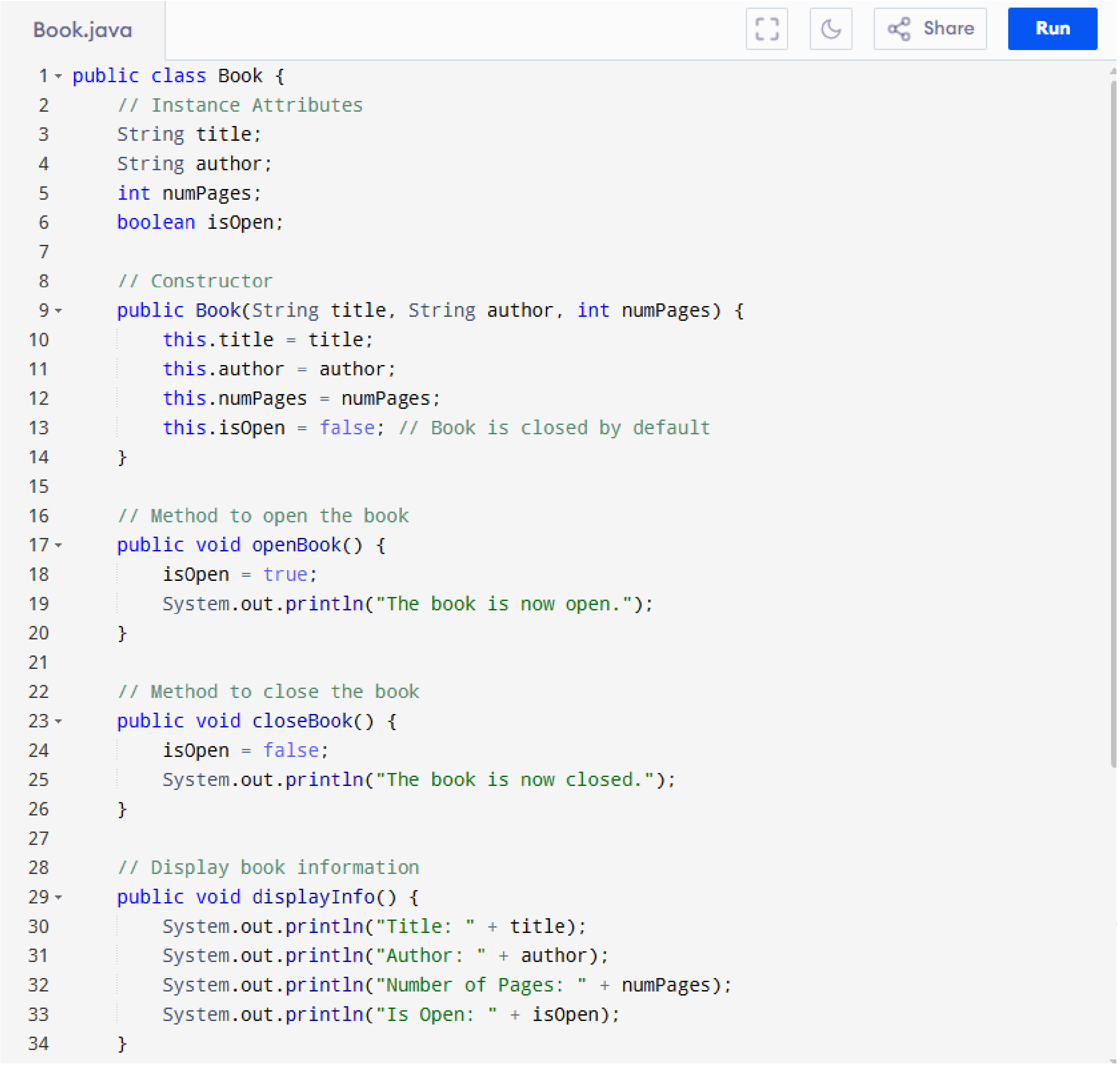
Main

Create Book object

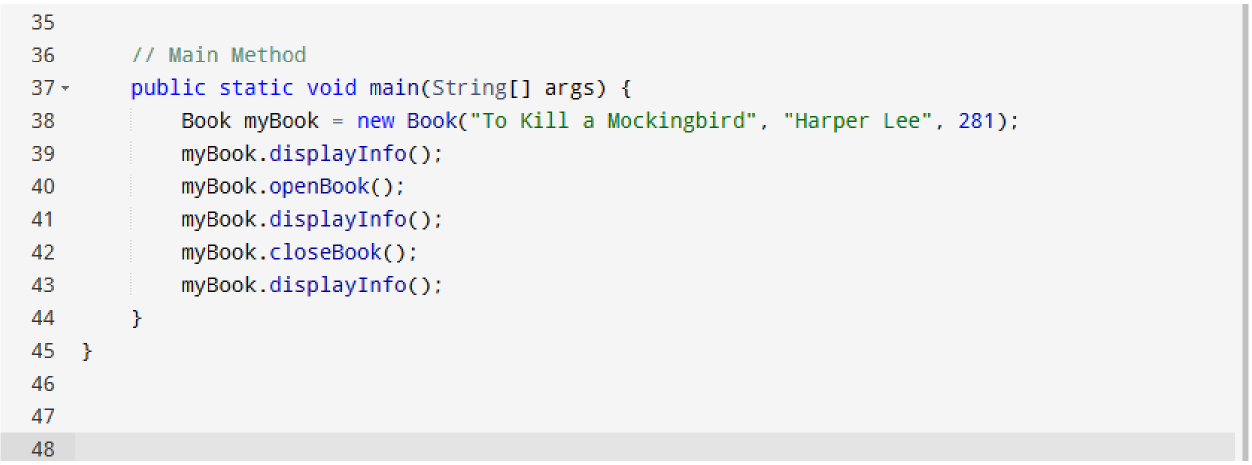
Display details

Open and close the book

4. Program Code



```java
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 closed by default
    }

    // 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.");
    }

    // Display book information
    public void displayInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Number of Pages: " + numPages);
        System.out.println("Is Open: " + isOpen);
    }
```



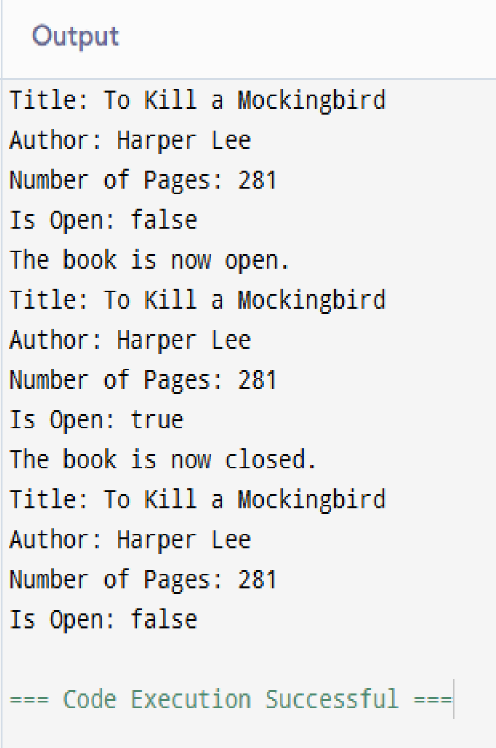
```
    // Main Method
    public static void main(String[] args) {
        Book myBook = new Book("To Kill a Mockingbird", "Harper Lee", 281);
        myBook.displayInfo();
        myBook.openBook();
        myBook.displayInfo();
        myBook.closeBook();
        myBook.displayInfo();
    }
}
```

5. Test Cases

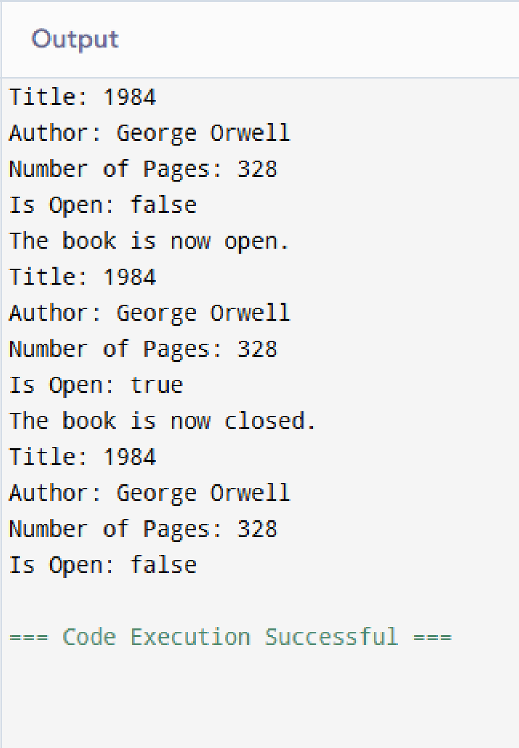
Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	"To Kill a Mockingbird", "Harper Lee", 281	Displays info, opens and closes book	Title: To Kill a Mockingbird Author: Harper Lee Number of Pages: 281 Is Open: false The book is now open. Title: To Kill a Mockingbird Author: Harper 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

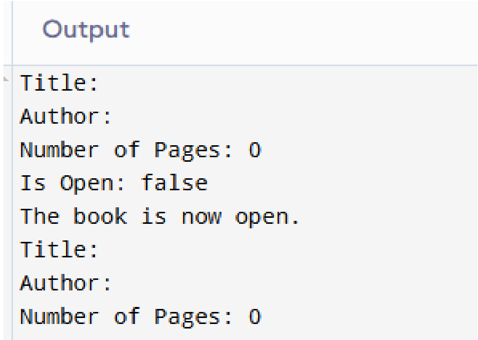
6. Screenshots of Output



```
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 ===
```



```
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 ===
```



```
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

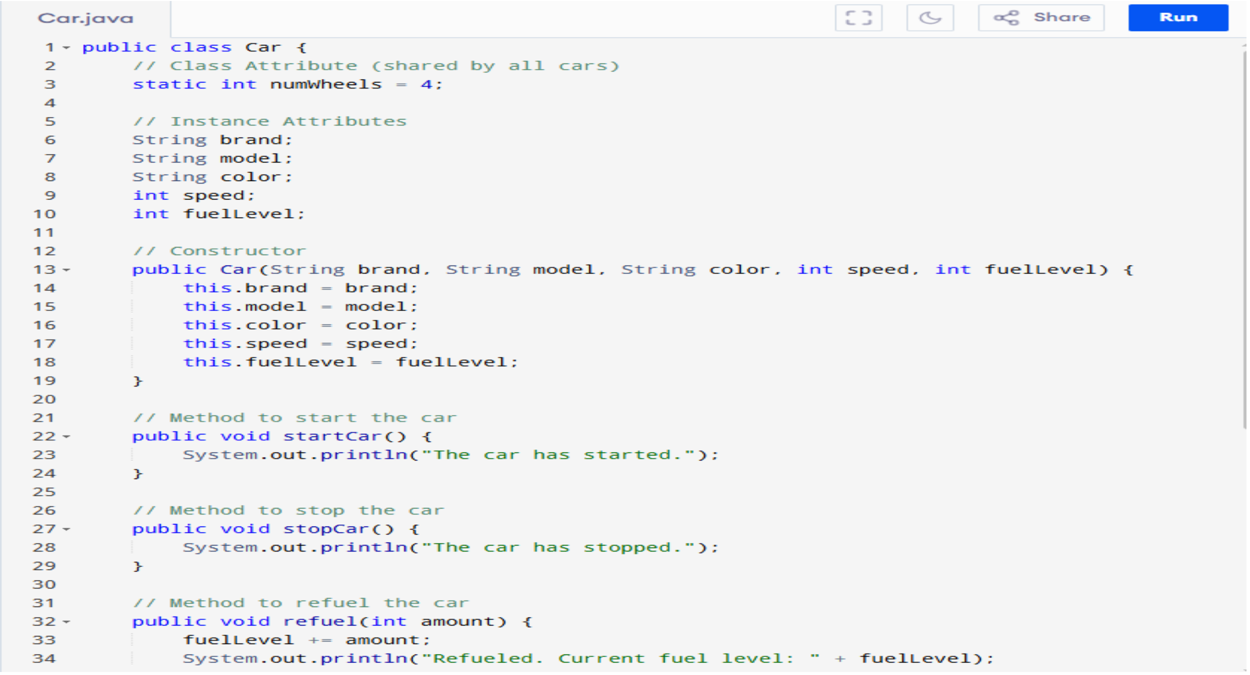
2. Algorithm

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.

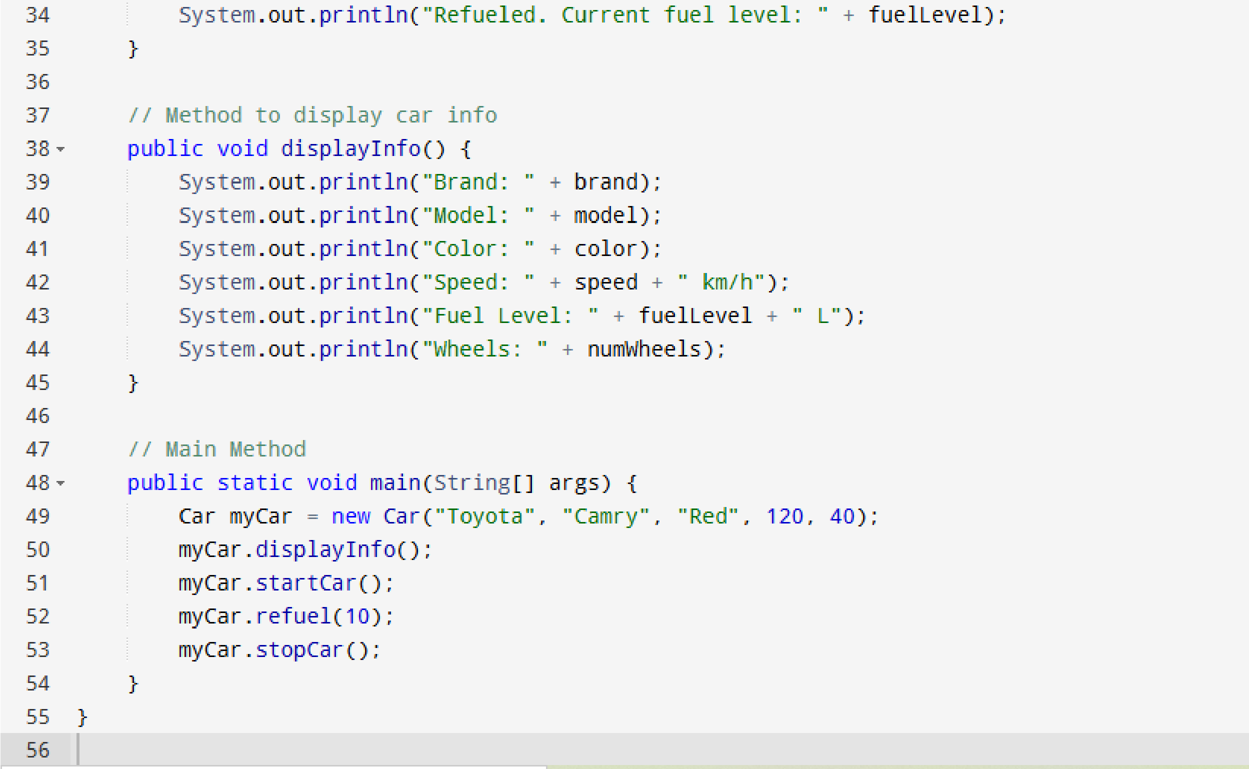
3. Pseudocode

```
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



```java
public class Car {
    // Class Attribute (shared by all cars)
    static int numWheels = 4;

    // Instance Attributes
    String brand;
    String model;
    String color;
    int speed;
    int fuelLevel;

    // Constructor
    public Car(String brand, String model, String color, int speed, int fuelLevel) {
        this.brand = brand;
        this.model = model;
        this.color = color;
        this.speed = speed;
        this.fuelLevel = fuelLevel;
    }

    // Method to start the car
    public void startCar() {
        System.out.println("The car has started.");
    }

    // Method to stop the car
    public void stopCar() {
        System.out.println("The car has stopped.");
    }

    // Method to refuel the car
    public void refuel(int amount) {
        fuelLevel += amount;
        System.out.println("Refueled. Current fuel level: " + fuelLevel);
```



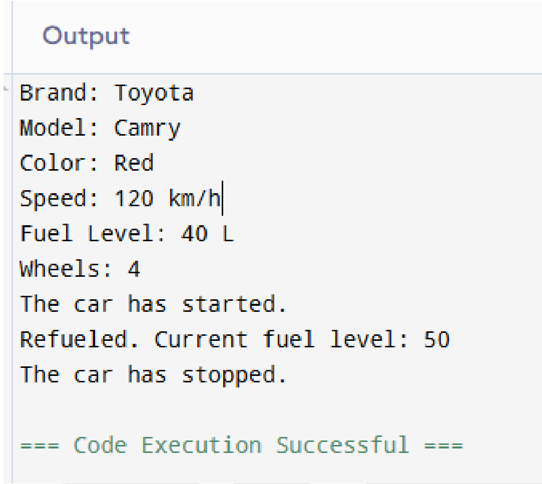
```java
        System.out.println("Refueled. Current fuel level: " + fuelLevel);
    }

    // Method to display car info
    public void displayInfo() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Color: " + color);
        System.out.println("Speed: " + speed + " km/h");
        System.out.println("Fuel Level: " + fuelLevel + " L");
        System.out.println("Wheels: " + numWheels);
    }

    // Main Method
    public static void main(String[] args) {
        Car myCar = new Car("Toyota", "Camry", "Red", 120, 40);
        myCar.displayInfo();
        myCar.startCar();
        myCar.refuel(10);
        myCar.stopCar();
    }
}

```

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

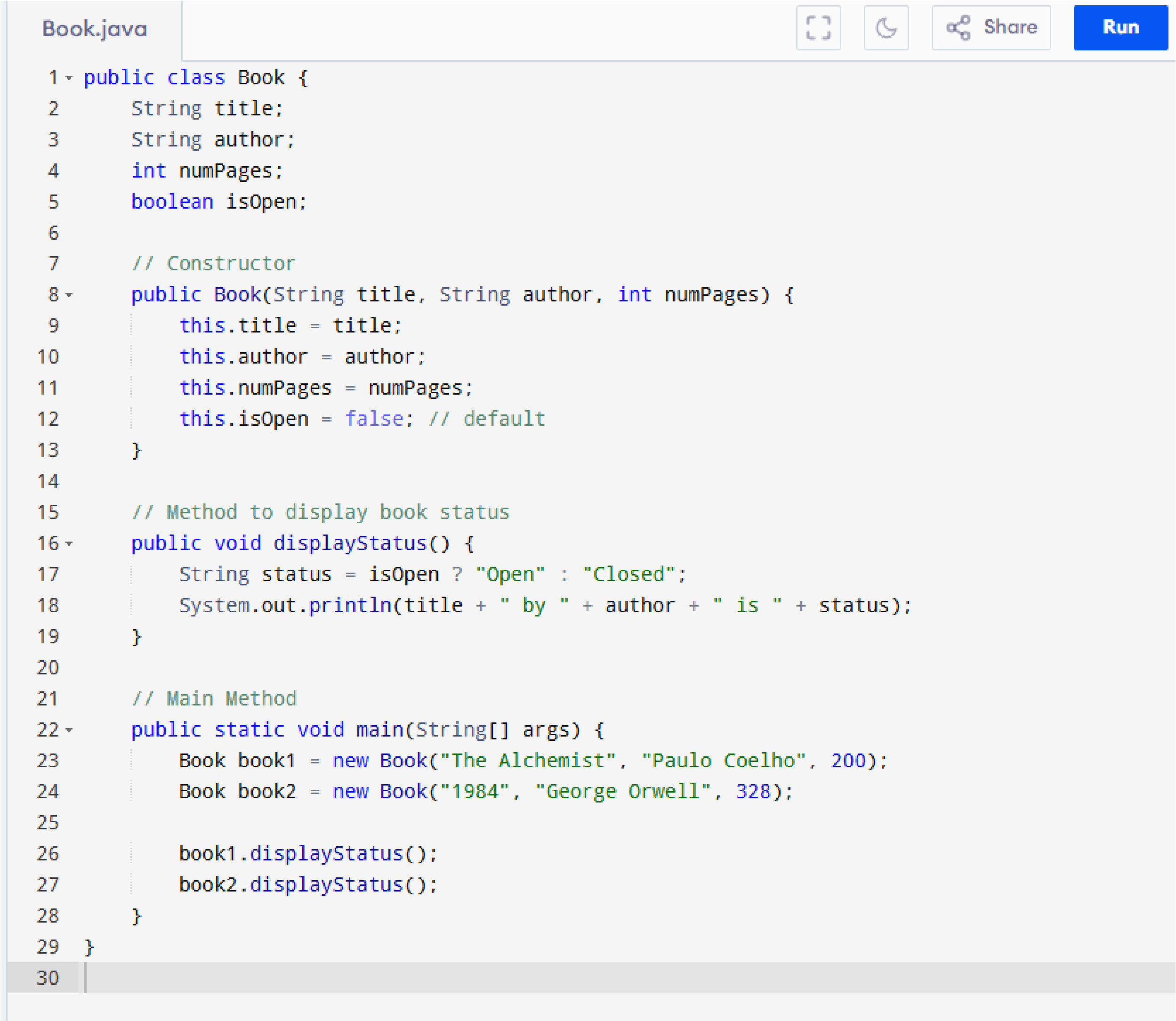
1. Define a class Book with instance attributes: title, author, numPages, and isOpen.
2. Create a constructor that initializes these values (set isOpen = false by default).
3. Define a method displayStatus():
4. If isOpen is true, print “Open”
5. Otherwise, print “Closed”
6. In the main() method:
7.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

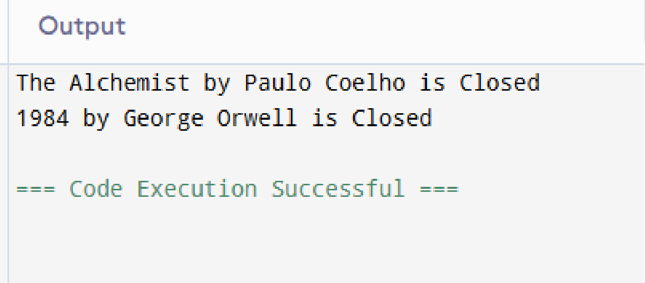


```java
public class Book {
    String title;
    String author;
    int numPages;
    boolean isOpen;

    // Constructor
    public Book(String title, String author, int numPages) {
        this.title = title;
        this.author = author;
        this.numPages = numPages;
        this.isOpen = false; // default
    }

    // Method to display book status
    public void displayStatus() {
        String status = isOpen ? "Open" : "Closed";
        System.out.println(title + " by " + author + " is " + status);
    }

    // Main Method
    public static void main(String[] args) {
        Book book1 = new Book("The Alchemist", "Paulo Coelho", 200);
        Book book2 = new Book("1984", "George Orwell", 328);

        book1.displayStatus();
        book2.displayStatus();
    }
}

```

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



```
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

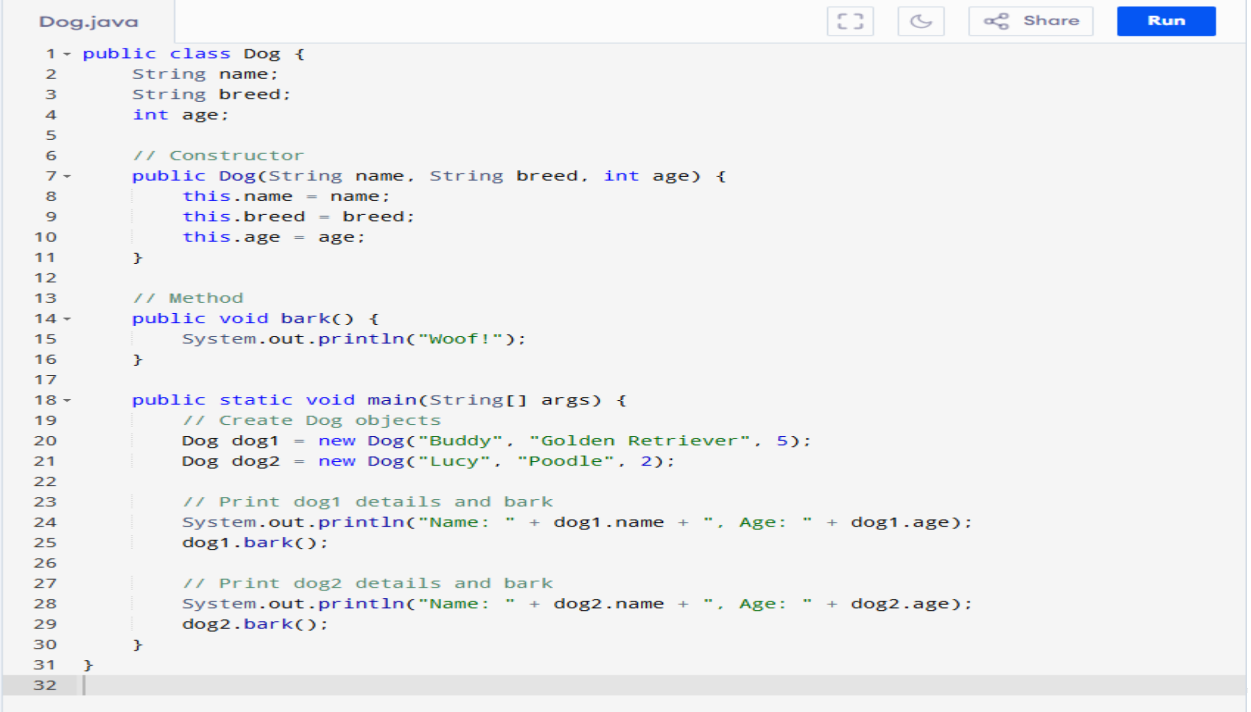
    Print the name and age of each object
    Call the bark() method for both dogs

---

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

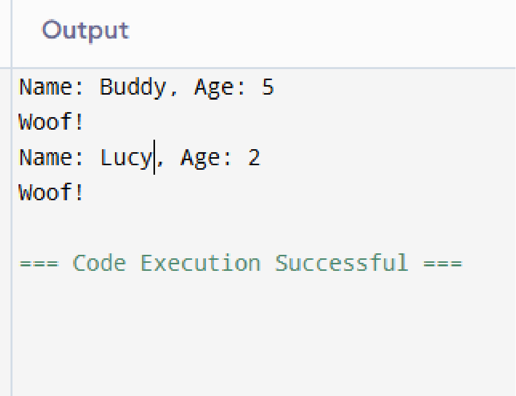


```java
public class Dog {
    String name;
    String breed;
    int age;

    // Constructor
    public Dog(String name, String breed, int age) {
        this.name = name;
        this.breed = breed;
        this.age = age;
    }

    // Method
    public void bark() {
        System.out.println("Woof!");
    }

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

        // Print dog1 details and bark
        System.out.println("Name: " + dog1.name + ", Age: " + dog1.age);
        dog1.bark();

        // Print dog2 details and bark
        System.out.println("Name: " + dog2.name + ", Age: " + dog2.age);
        dog2.bark();
    }
}
```

5. Test Cases

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



```
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.

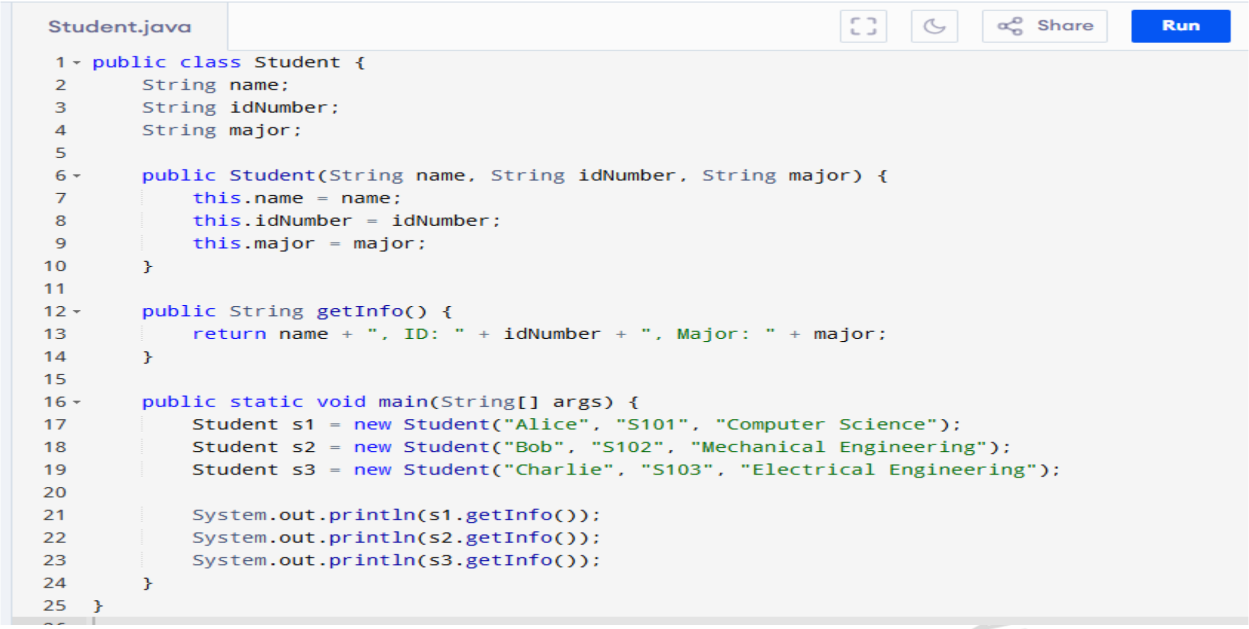
---

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



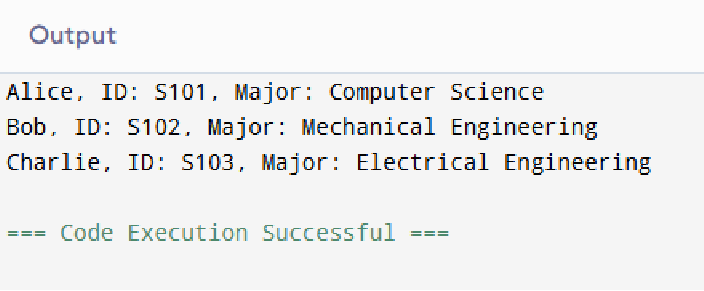
```java
public class Student {
    String name;
    String idNumber;
    String major;

    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;
    }

    public static void main(String[] args) {
        Student s1 = new Student("Alice", "S101", "Computer Science");
        Student s2 = new Student("Bob", "S102", "Mechanical Engineering");
        Student s3 = new Student("Charlie", "S103", "Electrical Engineering");

        System.out.println(s1.getInfo());
        System.out.println(s2.getInfo());
        System.out.println(s3.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	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



```
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 ≤ balance, subtract from balance.
- Else print "Insufficient funds".

In the main method:

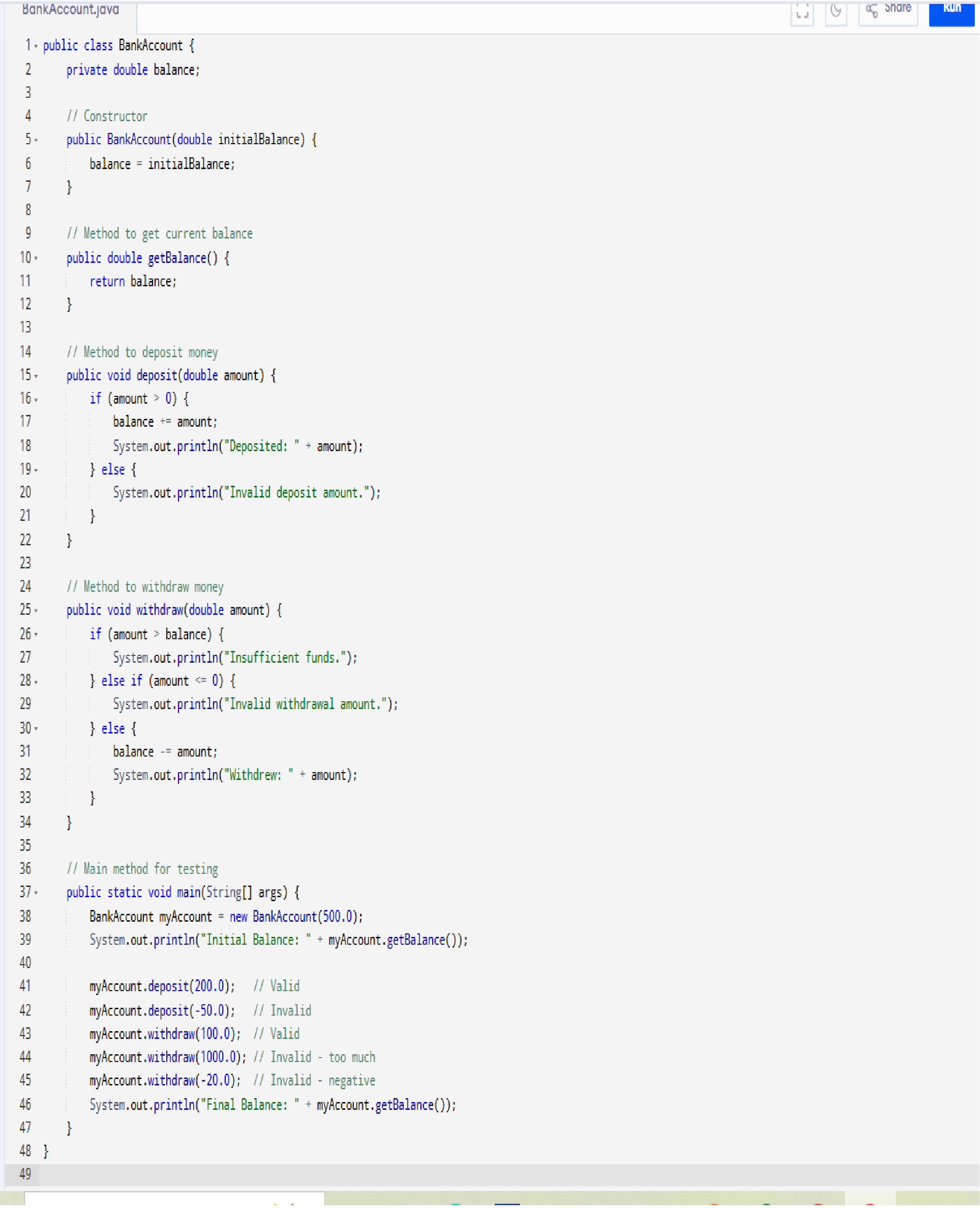
- Create a BankAccount object.
- Perform valid and invalid deposits and withdrawals.

---

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



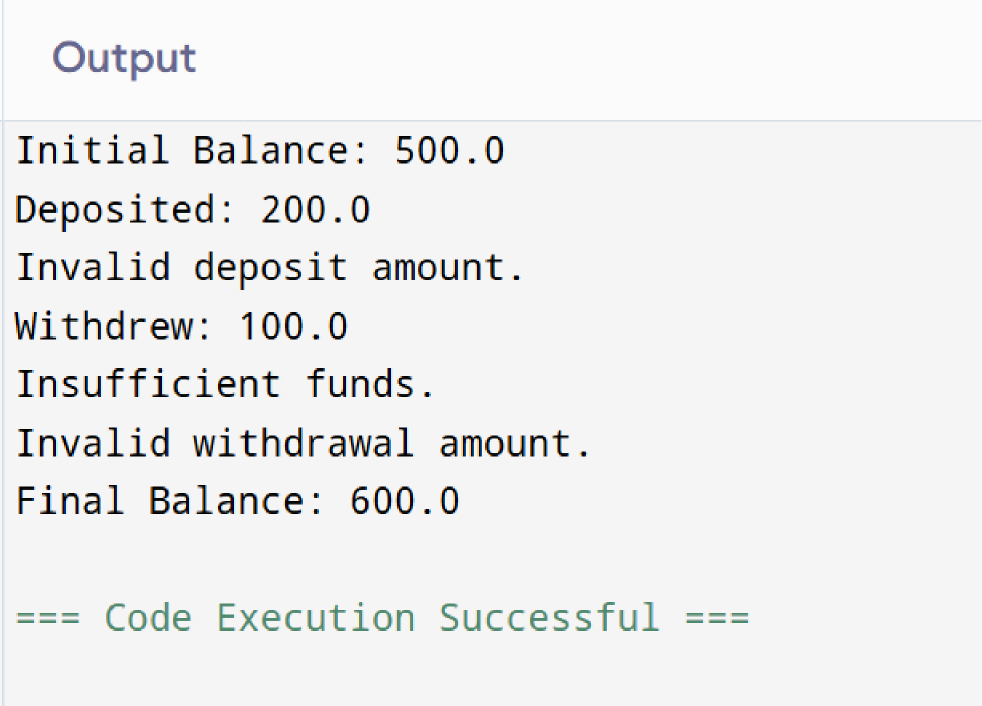
```java
public class BankAccount {
    private double balance;

    // Constructor
    public BankAccount(double initialBalance) {
        balance = initialBalance;
    }

    // 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 > balance) {
            System.out.println("Insufficient funds.");
        } else if (amount <= 0) {
            System.out.println("Invalid withdrawal amount.");
        } else {
            balance -= amount;
            System.out.println("Withdrew: " + amount);
        }
    }

    // Main method for testing
    public static void main(String[] args) {
        BankAccount myAccount = new BankAccount(500.0);
        System.out.println("Initial Balance: " + myAccount.getBalance());

        myAccount.deposit(200.0);    // Valid
        myAccount.deposit(-50.0);    // Invalid
        myAccount.withdraw(100.0);  // Valid
        myAccount.withdraw(1000.0); // Invalid - too much
        myAccount.withdraw(-20.0);  // Invalid - negative
        System.out.println("Final Balance: " + myAccount.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	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.
Final Balance: 600.0

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

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:

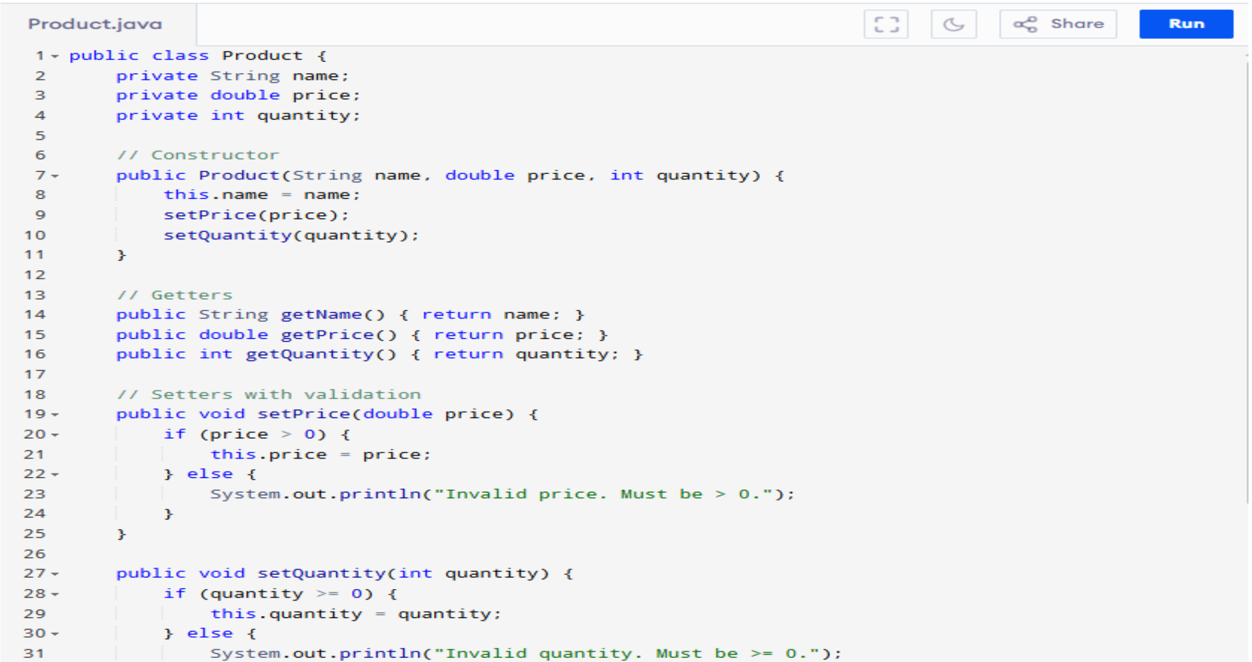
- Create a Product object
- Print its initial values
- Try setting invalid values
- Print final values and total inventory value

3. Pseudocode

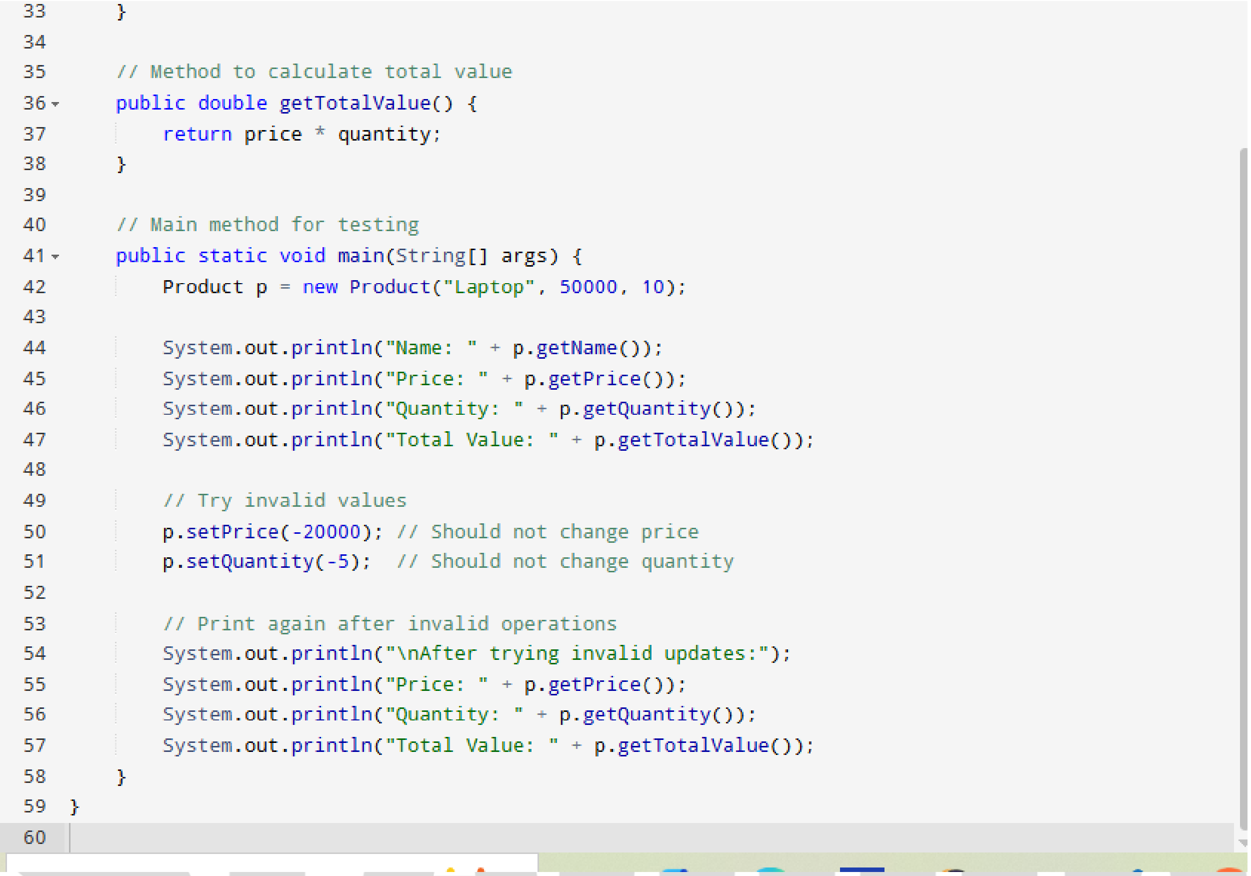
```
Class Product
```

```
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



```java
public class Product {
    private String name;
    private double price;
    private int quantity;

    // Constructor
    public Product(String name, double price, int quantity) {
        this.name = name;
        setPrice(price);
        setQuantity(quantity);
    }

    // Getters
    public String getName() { return name; }
    public double getPrice() { return price; }
    public int getQuantity() { return quantity; }

    // Setters with validation
    public void setPrice(double price) {
        if (price > 0) {
            this.price = price;
        } else {
            System.out.println("Invalid price. Must be > 0.");
        }
    }

    public void setQuantity(int quantity) {
        if (quantity >= 0) {
            this.quantity = quantity;
        } else {
            System.out.println("Invalid quantity. Must be >= 0.");
```



```
    }

    // Method to calculate total value
    public double getTotalValue() {
        return price * quantity;
    }

    // Main method for testing
    public static void main(String[] args) {
        Product p = new Product("Laptop", 50000, 10);

        System.out.println("Name: " + p.getName());
        System.out.println("Price: " + p.getPrice());
        System.out.println("Quantity: " + p.getQuantity());
        System.out.println("Total Value: " + p.getTotalValue());

        // Try invalid values
        p.setPrice(-20000); // Should not change price
        p.setQuantity(-5);  // Should not change quantity

        // Print again after invalid operations
        System.out.println("\nAfter trying invalid updates:");
        System.out.println("Price: " + p.getPrice());
        System.out.println("Quantity: " + p.getQuantity());
        System.out.println("Total Value: " + p.getTotalValue());
    }
}
```

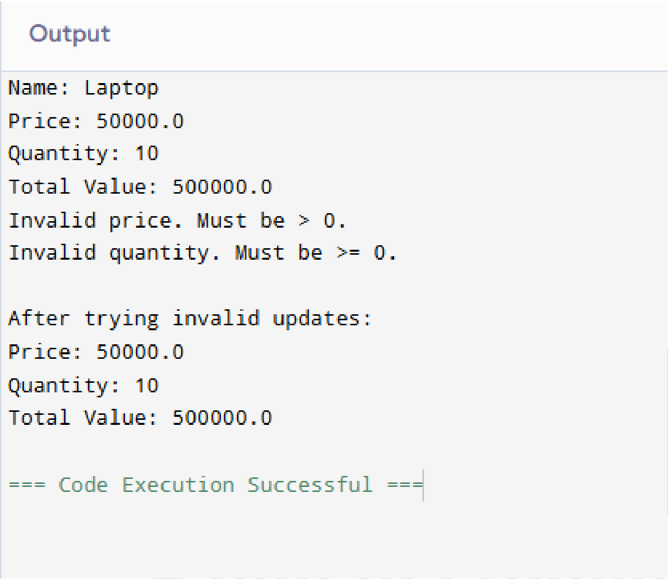
5. Test Cases

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



```
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.