Name: Sariya Mazhar

Enrollment Number: mern02

Batch / Class: Batch-01

Assignment: (Bridge Course Day 5)

Date of Submission: 30/06/2025

Problem Solving Activity 1

1. Program Statement: Real-World Object Dissection

- I want to Identify three real-world objects and describe 3-5 attributes and 2-3 behaviors

---

2. Algorithm

Step1: Declare and initialise variables

Step2: Add method

Step3: Call the non static method with object creation

Step4: Call them using reference variable

Step5: Result is seen.

---

3. Pseudocode

```
Class Television:
  Attributes:
    brandName
    numberOfButtons
    currentVolume
  Method displayWelcome():
    Print "Welcome to " + brandName + " Digiworld"
  Method changeChannel():
    Print "Changing channels using " + numberOfButtons + " buttons..."
  Method adjustVolume():
    Print "Volume up... " + currentVolume + " ...Volume down"
```

Main:

Create object tv of class Television

Call tv.displayWelcome()

Call tv.changeChannel()

Call tv.adjustVolume()

4. Program Code

4.1: Example 1

```
public class Code1 {
   String Television="Samsung";
   int Button=1;
   int Volume=23;
   public  void Television(){
      System.out.println("Welcome to "+Television+" Digiworld");
   }
   public void Button(){
      System.out.println("Changing channels using "+Button+" buttons...");
   }
   public void Volume(){
      System.out.println("Volume up... "+Volume+" ...Volume down");
   }
   public static void main(String[] args) {
      Code1 TV=new Code1();
      TV.Television();
      TV.Button();
      TV.Volume();
   }
}
```

4.2 Example 2

```
public class Code1 {
   public  void Whatsapp(){
      System.out.println("Welcome to Whatsapp");
   }
   public void Send(){
      System.out.println("Sending a message..");
   }
   public void Call(){
      System.out.println("Calling my friend....");
   }
   public static void main(String[] args) {
      Code1 Wsp=new Code1();
      Wsp.Whatsapp();
      Wsp.Send();
      Wsp.Call();
   }
}
```

4.3 Example 3

```
public class Code1 {
   public  void Instagram(){
      System.out.println("Welcome to Instagram");
   }
   public void Send(){
      System.out.println("Sending a message..");
   }
   public void Call(){
      System.out.println("Calling my friend....");
   }
   public static void main(String[] args) {
```

```
        Code1 Wsp=new Code1();
        Wsp.Instagram();
        Wsp.Send();
        Wsp.Call();
    }
}
```

5. Output

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  /usr/bin/env /L
tailsInExceptionMessages -cp /Users/sariyamazhar/Library/Application\
dgeCourse_994dd6f4/bin Code1
Welcome to Samsung Digiworld
Changing channels using 1 buttons...
Volume up... 23 ...Volume down
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  cd /Users/sariy
ontents/Home/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptior
3020e2428b4528a3fc774c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6
Welcome to Whatsapp
Sending a message..
Calling my friend....
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  cd /Users/sariy
ontents/Home/bin/java --enable-preview -XX:+ShowCodeDetailsInExceptior
3020e2428b4528a3fc774c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6
Welcome to Instagram
Sending a message..
Calling my friend....
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

6. Observation / Reflection

- This program involves creation of object as we fetch for non-static methods. We simply implement the program using method calling in main function.

Problem Solving Activity 2

1. Program Statement: Procedural vs. Object-Oriented Thought

- To write a model list of customers:Procedurally using arrays and methods OOP using a Customer class with attributes (name, id) and behaviors (addCustomer(), deleteCustomer())

---

2. Algorithm

Step 1: Start

Step 2: Create a Customer class with attributes name and id, and a method display() to show customer details

Step 3: Create a CustomerManager class with an ArrayList to store Customer objects

Step 4: Define a method addCustomer(name, id) to add a customer to the list

Step 5: Define a method deleteCustomer(id) to search and remove a customer from the list

Step 6: Define a method showAllCustomers() to print details of all customers

Step 7: In the main method, create a CustomerManager object

Step 8: Add two customers (e.g., Alice and Bob) using addCustomer()

Step 9: Display all customers using showAllCustomers()

Step 10: Delete a customer by ID using deleteCustomer()

Step 11: Display all remaining customers again

Step 12: End

---

3. Pseudocode

```
Class Customer:
    Declare attributes name, id
    Constructor(name, id):
        Set name and id
    Method display():
        Print name and id


Class CustomerManager:
    Create customerList as an empty list of Customers
    Method addCustomer(name, id):
        Create new Customer object
```

```
        Add to customerList
        Print confirmation

    Method deleteCustomer(id):
        For each customer in customerList:
            If customer.id equals id:
                Remove customer from list
                Print confirmation
                Exit method
        Print not found message

    Method showAllCustomers():
        For each customer in customerList:
            Call display() method of customer

Main method:
    Create object manager of CustomerManager
    Call manager.addCustomer("Alice", 101)
    Call manager.addCustomer("Bob", 102)
    Call manager.showAllCustomers()
    Call manager.deleteCustomer(101)
    Call manager.showAllCustomers()
```

---

4. **Code:**

```
import java.util.ArrayList;
class Customer {
    // Attributes
    String name;
    int id;
    // Constructor
    Customer(String name, int id) {
```

```
        this.name = name;
        this.id = id;
    }
    void display() {
        System.out.println("Customer: " + name + ", ID: " + id);
    }
}
class CustomerManager {
    // List to store Customer objects
    ArrayList<Customer> customerList = new ArrayList<>();

    // Behavior to add a customer
    void addCustomer(String name, int id) {
        customerList.add(new Customer(name, id));
        System.out.println("Customer added: " + name + ", ID: " + id);
    }

    // Behavior to delete a customer
    void deleteCustomer(int id) {
        for (Customer c : customerList) {
            if (c.id == id) {
                customerList.remove(c);
                System.out.println("Customer deleted: ID " + id);
                return;
            }
        }
        System.out.println("Customer with ID " + id + " not found.");
    }
    void showAllCustomers() {
        for (Customer c : customerList) {
            c.display();
```

```
        }
    }

    public static void main(String[] args) {
        CustomerManager manager = new CustomerManager();
        manager.addCustomer("Alice", 101);
        manager.addCustomer("Bob", 102);
        manager.showAllCustomers();
        manager.deleteCustomer(101);
        manager.showAllCustomers();
    }
}
```

---

5. Output

```
import java.util.ArrayList;

class Customer {
    // Attributes
    String name;
    int id;

    // Constructor
    Customer(String name, int id) {
        this.name = name;
        this.id = id;
    }

    void display() {
        System.out.println("Customer: " + name + ", ID: " + id);
    }
}
```

```
class CustomerManager {
    // List to store Customer objects
    ArrayList<Customer> customerList = new ArrayList<>();

    // Behavior to add a customer
    void addCustomer(String name, int id) {
        customerList.add(new Customer(name, id));
        System.out.println("Customer added: " + name + ", ID: " + id);
    }

    // Behavior to delete a customer
    void deleteCustomer(int id) {
        for (Customer c : customerList) {
            if (c.id == id) {
                customerList.remove(c);
                System.out.println("Customer deleted: ID " + id);
                return;
            }
        }
        System.out.println("Customer with ID " + id + " not found.");
    }

    void showAllCustomers() {
        for (Customer c : customerList) {
            c.display();
        }
    }

    public static void main(String[] args) {
        CustomerManager manager = new CustomerManager();
```

```
        manager.addCustomer("Alice", 101);
        manager.addCustomer("Bob", 102);
        manager.showAllCustomers();
        manager.deleteCustomer(101);
        manager.showAllCustomers();
    }
}
```

---

6. Observation:

- In procedural programming, data is handled using parallel arrays and behaviors through static methods, making it harder to maintain. In contrast, object-oriented programming uses classes with attributes and methods, offering better modularity, reusability, and flexibility.

Problem Solving Activity 3

1. Program statement:Simple Dog Class, Define a class named Dog

- To Design a class named Dog that includes class attributes species = "Canis familiaris" and numLegs = 4, instance attributes name, breed, and age, and a method bark() that prints "Woof!" to simulate the dog's barking behavior.

---

2. Algorithm

Step 1: Define a class Code3 with static variables: species and numLegs.

Step 2: Declare instance variables: name, breed, and age.

Step 3: Create a constructor to initialize the instance variables.

Step 4: Define a method bark() that prints "Woof!".

Step 5: In the main() method, create a Code3 object with sample data.

Step 6: Print all attributes and call the bark() method.

Step 7: End.

---

3. Pseudocode

```
Class Dog:
    Static attributes:
        species = "Canis familiaris"
        numLegs = 4
    Instance attributes:
        name, breed, age
    Constructor(name, breed, age):
        Set name, breed, age
    Method bark():
        Print "Woof!"
    Create dog object with "Buddy", "Labrador", 3
    Print name, breed, age
    Print species and number of legs
    Call bark()
```

---

4. Code

```
public class Code3{
    static String species = "Canis familiaris";
    static int numLegs = 4;
    String name;
    String breed;
    int age;
    public Code3(String name, String breed, int age) {
        this.name = name;
        this.breed = breed;
        this.age = age;
    }
```

```
    public void bark() {
        System.out.println("Woof Woof!!!!!!");
    }
    public static void main(String[] args) {
        Code3 myDog = new Code3("Thor", "Greatdane", 2);
        System.out.println("Name: " + myDog.name);
        System.out.println("Breed: " + myDog.breed);
        System.out.println("Age: " + myDog.age);
        System.out.println("Species: " + Code3.species);
        System.out.println("Number of Legs: " + Code3.numLegs);
        myDog.bark();
    }
}
```

5. Output:

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  /usr/bin/
w -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sariyamazha
4c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6f4/bin Code3
Name: Thor
Breed: Greatdane
Age: 2
Species: Canis familiaris
Number of Legs: 4
Woof Woof!!!!!!
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

6. Observation

This program defines a Dog class with both class and instance attributes, and simulates behavior using the bark() method. I used concepts like constructors, static variables, and method.

Problem Solving Activity 4

1. Program Statement: Basic book class

- I want to create a class named Book that contains instance attributes: title, author, numPages, and isOpen.My code should implement methods like openBook() to set isOpen to true and closeBook() to set isOpen to false, simulating the opening and closing of a book

---

2. Algorithm

Step 1: Define a Book class with attributes: title, author, numPages, and isOpen; initialize them using a constructor.

Step 2: Create methods openBook() and closeBook() to change and display the open/closed status.

Step 3: Define displayInfo() to print the book's title, author, and number of pages.

Step 4: In main(), create a book object and call all methods to test its behavior.

---

3. Pseudocode

```
Class Book:
  Attributes: title, author, numPages, isOpen
  Constructor(title, author, numPages):
  Method openBook():
    Set isOpen to true and print "The book is now open"
  Method closeBook():
    Set isOpen to false and print "The book is now closed"
  Method displayInfo():
    Print title, author, and numPages
  Create book object with "The Alchemist", "Paulo Coelho", 208
  Call displayInfo(), openBook(), closeBook() as needed
```

---

4. Code

```
public class Code4 {
  String title;
  String author;
  int numPages;
```

```
    boolean isOpen;
     public Code4(String title, String author, int numPages){
        this.title = title;
        this.author = author;
        this.numPages = numPages;
        this.isOpen = true;
    }
    public void closeBook() {
        isOpen = false;
        System.out.println("The book is now closed.");
    }
    public void openBook() {
        isOpen = true;
        System.out.println("The book is now open.");
    }
    public void displayInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Number of Pages: " + numPages);
    }
    public static void main(String[] args) {
        Code4 myBook = new Code4("The Alchemist", "Paulo Coelho", 208);
        myBook.displayInfo();
        myBook.openBook();
        myBook.displayInfo();
        myBook.closeBook();
        myBook.displayInfo();
    }
}
```

5. Output

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  cd
ines/jdk-24.jdk/Contents/Home/bin/java --enable-preview -X
de/User/workspaceStorage/470f142dc3020e2428b4528a3fc774c2/
Title: Varity
Author: Colleen Hoover
Number of Pages: 350
The book is now open.
Title: Varity
Author: Colleen Hoover
Number of Pages: 350
The book is now closed.
Title: Varity
Author: Colleen Hoover
Number of Pages: 350
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

---

6. Observation

- This program defines a Book class with attributes and methods to simulate opening and closing a book.It demonstrates object-oriented concepts like constructors, instance variables, and method behavior control.

Problem Solving Activity 5

1. Program Statement: Identify Class Elements for Car Class.

- A Car class can have instance attributes like model, color, Type and price, and methods like speed(), outlook(), and price().

---

2. Algorithm

Step 1: Define a class Code5 with attributes: model_name, cost, type, and color.

Step 2: Create a constructor with four parameters: a, b, c, d to initialize the attributes.

Step 3: Define method start() to display the start message using the model name.

Step 4: Define method Speed() to display the fuel type of the car.

Step 5: Define method Outlook() to display the color of the car.

Step 6: Define method Price() to display the model name and cost of the car.

Step 7: In the main() method, create an object c1 using the constructor with specific values.

Step 8: Call all four methods: start(), Speed(), Outlook(), and Price() using c1.

---

3. Pseudocode:

```
Class Code5:
    Attributes: model_name, cost, type, color
    Constructor(a, b, c, d):
        model_name = a
        cost = b
        type = c
        color = d
    Method start():
        Print "My " + model_name + " starts with tremendous exhaust"
    Method Speed():
        Print "My car runs on " + type
    Method Outlook():
        Print "My car has fancy deep " + color + " color"
    Method Price():
        Print "My car " + model_name + " costs " + cost
Main:
    Create object c1 with values: "Lamborghini", 30000000, "Diesel", "Black"
    Call c1.start()
    Call c1.Speed()
    Call c1.Outlook()
    Call c1.Price()
```

4. **Code:**

```
public class Code5 {
    String model_name;
    int cost;
    String type;
    String color;
    Code5(String a, int b, String c, String d){
        model_name=a;
        cost=b;
        type=c;
        color=d;
    }
    public void start(){
        System.out.println("My "+model_name+" starts with tremendous exhaust");
    }
    public void Speed(){
        System.out.println("My car runs on "+type);
    }
    public void Outlook(){
        System.out.println("My car has fancy deep "+color+ " color");
    }
    public void Price(){
        System.out.println("My car "+model_name+"costs "+cost);
    }
    public static void main(String[] args) {
        Code5 c1=new Code5("Lamborghini",30000000,"Diesel","Black");
        c1.start();
        c1.Speed();
        c1.Outlook();
        c1.Price();
     }}
```

5. Output

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  cd /Us
ines/jdk-24.jdk/Contents/Home/bin/java --enable-preview -XX:+
de/User/workspaceStorage/470f142dc3020e2428b4528a3fc774c2/red
My Lamborghini starts with tremendous exhaust
My car runs on Diesel
My car has fancy deep Black color
My car Lamborghinicosts 30000000
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

---

6. **Observation**

- My code showcases object creation and method invocation to display car details like start sound, fuel type, color, and cost.

Problem Solving Activity 6

1. Program Statement:Create Dogs

- To Create a Dog class with instance attributes for name, breed, and age, and a method bark() that prints a bark sound. Then I want to create two Dog objects with given details, call their bark() method, and print their names and ages.

---

2. Algorithm

Step 1: Define a class Dog with attributes: name, breed, and age.

Step 2: Create a constructor to initialize the attributes.

Step 3: Define a method bark() to print a barking sound with the dog's name.

Step 4: Define a method displayInfo() to print name and age.

Step 5: In the main() method, create two Dog objects: dog1 and dog2.

Step 6: Call bark() and displayInfo() methods for both dogs.

Step 7: End the program.

3. Pseudocode

```
Class Dog:
    Attributes: name, breed, age
    Constructor(n, b, a):
        name = n
        breed = b
        age = a
    Method bark():
        Print name + " says: Woof woof!"
    Method displayInfo():
        Print name and age
Main:
    Create dog1 with values: "Buddy", "Golden Retriever", 5
    Create dog2 with values: "Lucy", "Poodle", 2
    Call dog1.bark()
    Call dog1.displayInfo()
    Call dog2.bark()
    Call dog2.displayInfo()
```

---

4. Code:

```
public class Code6 {
   String name;
   String breed;
   int age;
    Code6(String n, String b, int a) {
      name = n;
      breed = b;
      age = a;
   }
   void bark() {
      System.out.println(name + " says: Woof woof!");
```

```
    }
    void displayInfo() {
        System.out.println("Name: " + name + ", Age: " + age);
    }
    public static void main(String[] args) {
        Code6 dog1 = new Code6("Buddy", "Golden Retriever", 5);
        Code6 dog2 = new Code6("Lucy", "Poodle", 2);
        dog1.bark();
        dog1.displayInfo();
        dog2.bark();
        dog2.displayInfo();
    }
}
```

5. Output

```
 /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-24.
ariyamazhar/Library/Application\ Support/Code/User/work
in Code6
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
w -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sa
4c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6f4/bi
Buddy says: Woof woof!
Name: Buddy, Age: 5
Lucy says: Woof woof!
Name: Lucy, Age: 2
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

6. Observation

- This coding problem is met for the program statement. It simply used constructors and methods to call via reference variable.

Problem Solving Activity 7

1. Program Statement:Manage Books

- To Create a Book class with attributes title, author, and isOpen, and use a constructor to initialize them. I want to Implement a method displayStatus() that prints whether the book is "Open" or "Closed" based on the isOpen value.

---

2. Algorithm

Step 1: Define a class named Book with attributes: title, author, and isOpen.

Step 2: Create a constructor to initialize the book's title, author, and isOpen values.

Step 3: Define a method displayStatus() to display the book's status.

Step 4: In displayStatus(), use an if-else condition:

If isOpen is true, set status to "Open".

Else, set status to "Closed".

Step 5: Print the book's title, author, and current status.

Step 6: In the main() method, create two book objects with different values.

Step 7: Call the displayStatus() method for both objects.

Step 8: End.

---

3. **Pseudocode**

```
Class Book:
 Attributes: title, author, isOpen
 Constructor(t, a, o):
    title = t
    author = a
    isOpen = o
 Method displayStatus():
    Declare status as String
    If isOpen is true
       Set status = "Open"
    Else
```

```
        Set status = "Closed"
    Print title + " by " + author + " is " + status
Main:
  Create object book1 with "abc", "def", true
  Create object book2 with "xyz", "ghi", false
  Call book1.displayStatus()
  Call book2.displayStatus()
```

---

4. Code

```
public class Code7 {
  String title;
  String author;
  boolean isOpen;
  String status;
  Code7(String t, String a, boolean o) {
    title = t;
    author = a;
    isOpen = o;
  }
  void displayStatus() {
    if (isOpen) {
      status = "Open";
    } else {
      status = "Closed";
    }
    System.out.println(title + " by " + author + " is " + status);
  }
  public static void main(String[] args) {
    Code7 book1 = new Code7("The Alchemist", "Paulo Coelho", true);
    Code7 book2 = new Code7("Atomic Habits", "James Clear", false);
    book1.displayStatus();
```

```
        book2.displayStatus();
    }
}
```

---

5. Output

```
 /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-24.jdk/C
ariyamazhar/Library/Application\ Support/Code/Use%
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  /usr/
w -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sariyar
4c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6f4/bin Coc
abc by def is Open
xyz by ghi is Closed
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

---

6. Observation:

- Instead of using the ternary operator, I used an if-else block for clarity and simplicity.Each book object has its own state (isOpen), and displayStatus() reflects that state accurately.

Problem Solving Activity 8

1. Program Statement: Student Record

- Design a Java program to represent student records using an object-oriented approach. Create a Student class with attributes such as name, idNumber, and major.

---

2. Algorithm

Step 1: Start

Step 2: Define a class with attributes: name, idNumber, major

Step 3: Create a constructor to initialize the attributes

Step 4: Define getInfo() to return student details

Step 5: In main, create 3 student objects with values

Step 6: Call getInfo() and print for each student

3. Pseudocode:

```
Class Student:
    Declare name, idNumber, major

    Constructor(name, idNumber, major):
        Set attributes

    Method getInfo():
        Return name + ID + Major

Main:
    Create student s1 with values
    Create student s2 with values
    Create student s3 with values
    Print s1.getInfo()
    Print s2.getInfo()
    Print s3.getInfo()
```

---

4. Code:

```
public class Prog8{
    public static double calculateDiscount(double originalPrice, double discountPercentage) {
        return originalPrice * (discountPercentage / 100);
    }
    public static double calculateTax(double amount, double taxRate) {
        return amount * (taxRate / 100);
    }
    public static double calculateFinalPrice(double itemPrice, double discountPerc, double taxRate) {
        double discount = calculateDiscount(itemPrice, discountPerc);
        double priceAfterDiscount = itemPrice - discount;
        double tax = calculateTax(priceAfterDiscount, taxRate);
```

```
        return priceAfterDiscount + tax;
    }
    public static void main(String[] args) {


        double finalPrice = calculateFinalPrice(1000.0, 10.0, 5.0);
        System.out.println("Final Price: Rs " + finalPrice);
    }
}
```

5. Output

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  /usr/bin/env
w -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sariyamazhar/L
4c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6f4/bin Code8
Abc, ID: 101, Major: CSE
def, ID: 102, Major: CSE
xyz, ID: 103, Major: ECE
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

6. Observation

- The program successfully creates and displays student records using a class and object-oriented approach. Each student's information is neatly printed using the getInfo() method.

Problem Solving Activity 9

1. Program Statement: Bank Account

- To create a bank user details using encapsulation. To create and test a BankAccount class with getBalance(), deposit(), and withdraw() methods

---

2. Algorithm

Step 1: Start

Step 2: Create a class with an attribute balance

Step 3: Initialize balance in the constructor

Step 4: If initial balance is negative, set to 0

Step 5: Define getBalance() to return current balance

Step 6: Define deposit(amount) to add to balance if amount > 0

Step 7: Define withdraw(amount) to subtract if valid and funds are enough

Step 8: In main, create account with initial balance

Step 9: Call deposit and withdraw with various values

Step 10: Print final balance

---

3. Pseudocode

```
Class Account:
    Declare balance

    Constructor(initialBalance):
        If initialBalance >= 0
            Set balance
        Else
            Print warning, set balance = 0

    Method getBalance():
        Return balance
```

```
    Method deposit(amount):
        If amount > 0
            Add to balance
        Else
            Print error

    Method withdraw(amount):
        If amount <= 0
            Print error
        Else if amount > balance
            Print insufficient funds
        Else
            Subtract from balance

Main:
    Create account with 1000
    Print initial balance
    Call deposit(500)
    Call deposit(-100)
    Call withdraw(200)
    Call withdraw(2000)
    Call withdraw(-50)
    Print final balance
```

---

4. Code:

```
public class Code9 {
    // Instance attribute
    private double balance;

    // Constructor
    public Code9(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) {
    Code9 account = new Code9(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. Output

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  /usr/bin/en
w -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sariyamazhar/
4c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6f4/bin Code9
Initial Balance: 1000.0
Deposited: 500.0
Invalid deposit amount.
Withdrawn: 200.0
Insufficient funds. Withdrawal denied.
Invalid withdrawal amount.
Final Balance: 1300.0
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

6. Observation

- Each operation is handled in separate functions, The program became easier to read, maintain, and scale with better error handling.

Problem Solving Activity 10

1. **Program Statement:Product Inventory**

- I want to create void customGreet(String name, String greeting), void customGreet(String name) and call functions individually.

---

2. Algorithm

Step 1: Start

Step 2: Create a class with private attributes: name, price, quantity

Step 3: Use a constructor to initialize all attributes with validation for price and quantity

Step 4: Create getters for name, price, and quantity

Step 5: Create setPrice() to update price only if it is positive

Step 6: Create setQuantity() to update quantity only if it's zero or more

Step 7: Define getTotalValue() method to return price × quantity

Step 8: In main, create a product with initial values

Step 9: Display product details

Step 10: Try setting invalid price and quantity and observe error messages

Step 11: Set valid new price and quantity

Step 12: Display updated product details

---

3. **Pseudocode**

```
Class Product:
    Declare 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 price
    Else
        Print error
Method setQuantity(quantity):
    If quantity >= 0
        Set quantity
    Else
        Print error
Method getTotalValue():
    Return price * quantity:
Create product with name = "Laptop", price = 45000, quantity = 5
Print name, price, quantity, total value
Call setPrice(-1000) → invalid
Call setQuantity(-2) → invalid
Call setPrice(50000) → valid
Call setQuantity(3) → valid
Print updated price, quantity, and total value
```

4. Code

```
public class Code10 {
    // Private instance variables
    private String name;
    private double price;
    private int quantity;

    // Constructor
    public Code10(String name, double price, int quantity) {
        this.name = name;
        setPrice(price);
        setQuantity(quantity);
    }

    // Getter for name
    public String getName() {
        return name;
    }

    // Getter for price
    public double getPrice() {
        return price;
    }

    // Getter for quantity
    public int getQuantity() {
        return quantity;
    }

    // Setter for price with validation
```

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

// Setter for quantity with validation
public void setQuantity(int quantity) {
    if (quantity >= 0) {
        this.quantity = quantity;
    } else {
        System.out.println("Invalid quantity. Cannot be negative.");
    }
}

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

// Main method for testing
public static void main(String[] args) {
    // Create a product
   Code10 product = new Code10("Laptop", 45000.0, 5);

    // Print initial details
    System.out.println("Product: " + product.getName());
    System.out.println("Price: " + product.getPrice());
    System.out.println("Quantity: " + product.getQuantity());
```

```
    System.out.println("Total Value: " + product.getTotalValue());
    product.setPrice(-1000);     // Invalid
    product.setQuantity(-2);     // Invalid
    product.setPrice(50000);
    product.setQuantity(3);
    System.out.println("\nUpdated Details:");
    System.out.println("Price: " + product.getPrice());
    System.out.println("Quantity: " + product.getQuantity());
    System.out.println("Total Value: " + product.getTotalValue());
  }}
```

5. Output

```
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  /usr/bin/env
w -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/sariyamazhar/Li
4c2/redhat.java/jdt_ws/StemUp\ BridgeCourse_994dd6f4/bin Prog10
Good Morning, Sariya!
Hello, Sariya!
Hello, Guest!
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  cd /Users/sar
ines/jdk-24.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCod
de/User/workspaceStorage/470f142dc3020e2428b4528a3fc774c2/redhat.jav
Good Morning, Bruno!
Hello, Bruno!
Hello, Guest!
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %  cd /Users/sar
ines/jdk-24.jdk/Contents/Home/bin/java --enable-preview -XX:+ShowCod
de/User/workspaceStorage/470f142dc3020e2428b4528a3fc774c2/redhat.jav
Good Morning, Virat!
Hello, Virat!
Hello, Guest!
(base) sariyamazhar@SARIYAs-Air StemUp BridgeCourse %
```

6. Observation

The program demonstrates product creation with validation using encapsulation and setter methods. It ensures only valid price and quantity values are accepted, maintaining data integrity.