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

Problem Solving Activity 4.1

Program Statement:

Write a program that calculates the area of three rectangles. First, do this by repeating the same lines of code for each rectangle. After writing the program, observe and identify which lines are repeated. Then, improve the program by creating a function called calculateArea(int length, int width) that takes the length and width of a rectangle and returns its area. Use this function to calculate the area of all three rectangles, so the code becomes cleaner and easier to manage.

---

2. Algorithm

1. Start the program
2. Define a method Area(w, h) that:

   ➤ Multiplies w * h to get area

   ➤ Prints the result
3. In main(), call Area() three times with different width and height values
4. End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION Area(w, h)
    res ← w × h
  PRINT "The area of rectangle is " + res
 END FUNCTION
 MAIN
  CALL Area(4, 2)
  CALL Area(3, 4)
  CALL Area(5, 3)
 END MAIN
END
```

4. Program Code

```
public class d41{
  static void Area(int w,int h){
        int res=(w*h);
    System.err.println("The area of rectangle is "+res);
  }
  public static void main(String[] args) {
    Area(4,2);
    Area(3,4);
    Area(5,3);
  }
}
```

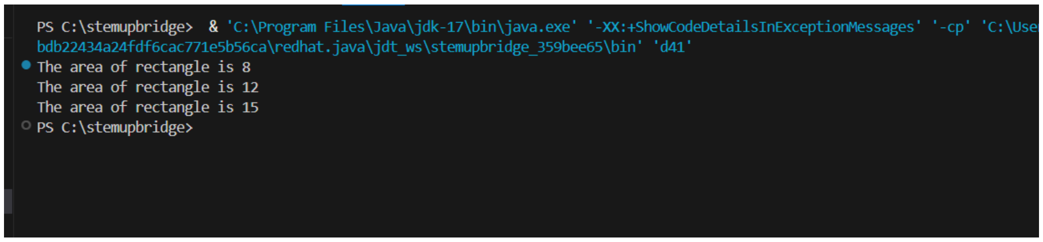
5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	4,2	8	8	Pass
2	3,4	12	12	Pass

3	5,3	15	15	Pass

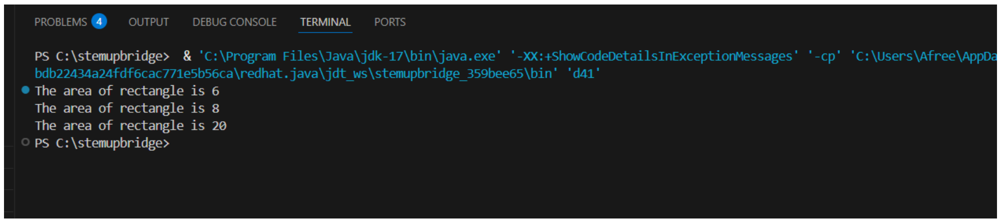
5. Screenshots of Output

Case 1:



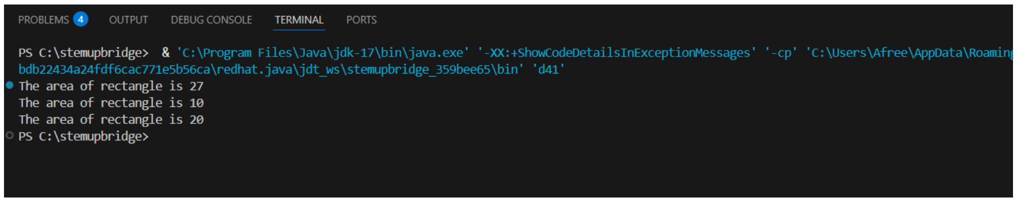
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Use
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'd41'
The area of rectangle is 8
The area of rectangle is 12
The area of rectangle is 15
PS C:\stemupbridge>
```

Case : 2



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppDa
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'd41'
The area of rectangle is 6
The area of rectangle is 8
The area of rectangle is 20
PS C:\stemupbridge>
```

Case : 3



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppData\Roaming
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'd41'
The area of rectangle is 27
The area of rectangle is 10
The area of rectangle is 20
PS C:\stemupbridge>
```

7. Observation / Reflection

It was a little hard to understand how to use functions and call them correctly at first. Through practice, writing simple methods and reusing them in the program became clearer. The concept of passing values to functions and getting results back was understood better. This also highlighted the importance of keeping the code neat and modular. In the future, the program can be improved by taking input from the user instead of using fixed values. Adding basic error handling will also help make the program more robust and reliable

Problem Solving Activity 4.2

Write a simple ATM program where a user can check their account balance, deposit money, and withdraw money. The program should be written in a clean way by using separate functions for each task like checking the balance, adding money, and withdrawing money.

---

2. Algorithm

1. Start the program
2. Set initial balance to ₹1000
3. Show menu with options:

- Check Balance
- Deposit
- Withdraw
- Exit

4. Based on user choice, call the respective function
5. Repeat the menu until the user chooses to exit
6. End the program

---

3. Pseudocode

```
BEGIN
 SET balance ← 1000
 FUNCTION checkBalance()
  PRINT balance
 END FUNCTION
 FUNCTION depositMoney(amount)
  balance ← balance + amount

  PRINT deposit message
 END FUNCTION
 FUNCTION withdrawMoney(amount)
  IF amount $\leq$ balance THEN
   balance $\leftarrow$ balance - amount
   PRINT withdraw message
  ELSE
   PRINT "Insufficient balance"
  END IF
 END FUNCTION
 DO
  DISPLAY menu
  GET user choice
  SWITCH choice
   CASE 1: CALL checkBalance()
   CASE 2: GET amount $\rightarrow$ CALL depositMoney(amount)
   CASE 3: GET amount $\rightarrow$ CALL withdrawMoney(amount)
   CASE 4: PRINT "Thank you"
   DEFAULT: PRINT "Invalid choice"
  END SWITCH
 WHILE choice $\neq$ 4
END
```

---

4. Program Code

```
import java.util.Scanner;
public class atm {
    static double balance = 1000.0;
    public static void checkBalance() {
        System.out.println("Current Balance: ₹" + balance);
```

```
}
public static void depositMoney(double amount) {
    balance += amount;
    System.out.println("Deposited: ₹" + amount);
}
public static void withdrawMoney(double amount) {
    if (amount <= balance) {
        balance -= amount;
        System.out.println("Withdrawn: ₹" + amount);
    } else {
        System.out.println("Insufficient balance!");
    }
}
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int choice;
    do {
        System.out.println("\n===== ATM Menu =====");
        System.out.println("1. Check Balance");
        System.out.println("2. Deposit Money");
        System.out.println("3. Withdraw Money");
        System.out.println("4. Exit");
        System.out.print("Enter choice: ");
        choice = sc.nextInt();
        switch (choice) {
            case 1:
                checkBalance();
                break;
            case 2:
                System.out.print("Enter amount to deposit: ");
                double deposit = sc.nextDouble();
                depositMoney(deposit);
                break;
            case 3:
                System.out.print("Enter amount to withdraw: ");
                double withdraw = sc.nextDouble();
                withdrawMoney(withdraw);
                break;
            case 4:
                System.out.println("Thank you for using the ATM.");
                break;
            default:
                System.out.println("Invalid choice.");
          }
        } while (choice != 4);
        sc.close();
      }
    }
    }
```

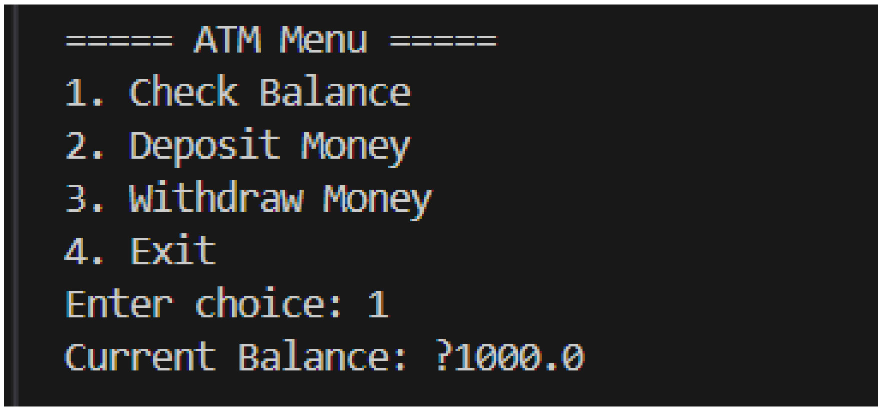
5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	Enter choice: 1	Current Balance	Current Balance: ?1000.0	Pass
2	Enter choice: 2	Enter amount to deposit: 3	Enter amount to deposit: 3 Deposited: ?3.0	Pass
3	Enter choice: 4	Exit	Enter choice: 4 Thank you for using the ATM.	Pass

---

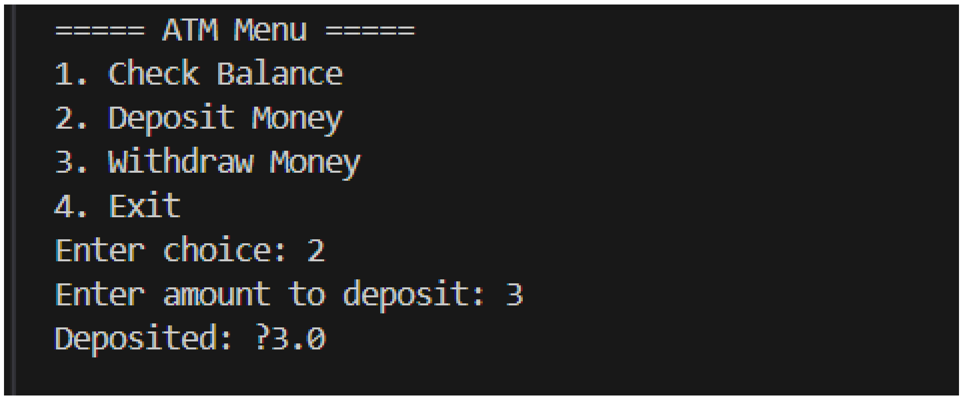
6 Screenshots of Output

Case 1:



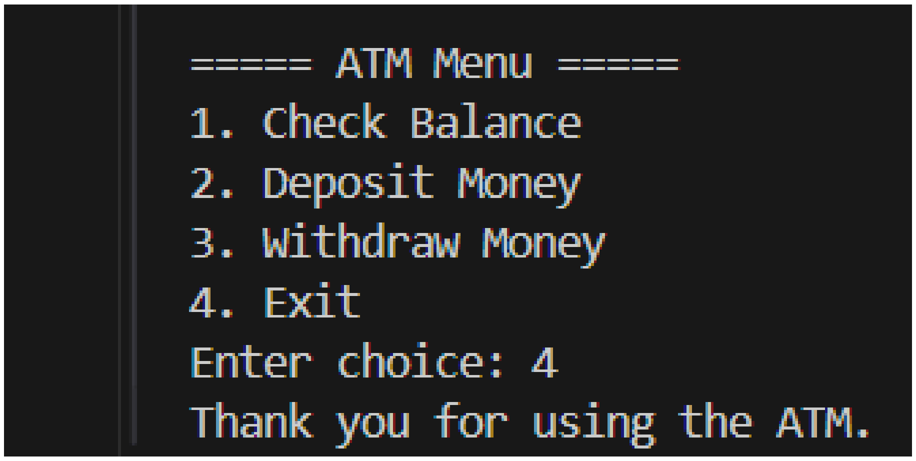
```
===== ATM Menu =====
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter choice: 1
Current Balance: ?1000.0
```

Case 2:



```
===== ATM Menu =====
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter choice: 2
Enter amount to deposit: 3
Deposited: ?3.0
```

Case 3:



```
===== ATM Menu =====
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter choice: 4
Thank you for using the ATM.
```

---

7. Observation / Reflection

The program is well-organized and divided into three clear functions: one for checking the balance, one for depositing money, and one for withdrawing money. It uses a loop to keep showing the menu again and again until the user chooses to exit, making it user-friendly. By using functions, the code becomes clean, reusable, and easy to manage. Before allowing any withdrawal, the program also checks if the user has enough balance, which makes it work like a real ATM. Overall, the program is simple to understand and effectively performs basic ATM operations.

3. Program Statement: Greeting Function

- Make a function called greetUser(String name) that says hello to the user using their name. Call this function three times using three different names.

---

2. Algorithm

1. Start the program
2. Define a method greetUser(name) that prints "Hello, name!"
3. In the main() method, call greetUser() three times with different names
4. End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION greetUser(name)
    PRINT "Hello, " + name + "!"
  END FUNCTION
  MAIN
    CALL greetUser("Afreen")
    CALL greetUser("Riya")
    CALL greetUser("Ankit")
  END MAIN
END
```

---

4. Program Code

```
public class greet1  {
    public static void greetUser(String name) {
        System.out.println("Hello, " + name + "!");
    }
    public static void main(String[] args) {
        greetUser("Afreen");
        greetUser("Riya");
        greetUser("Ankit");
    }
}
```

---

5. Test Cases

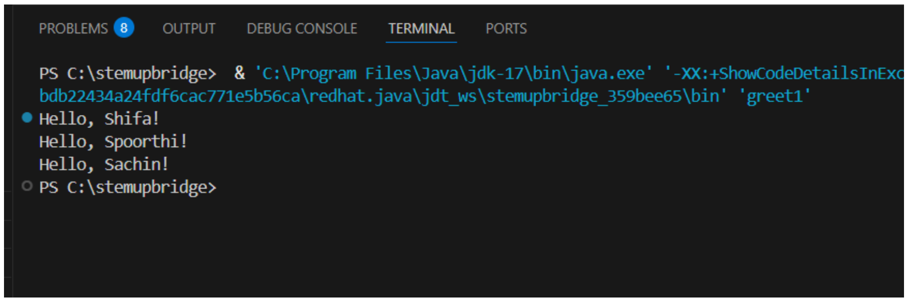
Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	**Input**	**Expected Output**	**Actual Output**	**Status (Pass/Fail)**
1	------------	Shifa	Shifa	Pass
2	-------------	Spoorthi	Spoorthi	Pass
3	-------------	Sachin	Sachin	Pass

---

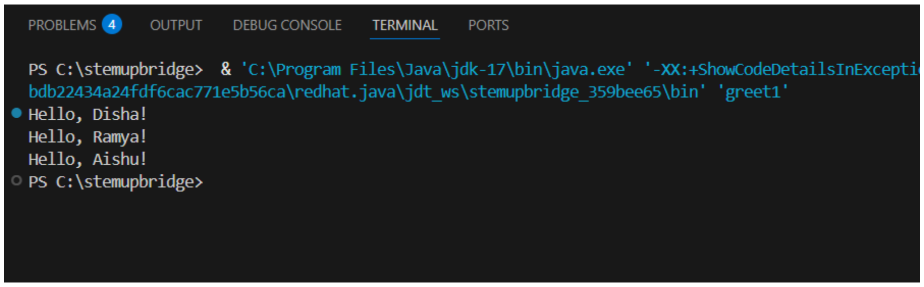
5. Screenshots of Output

Case 1:



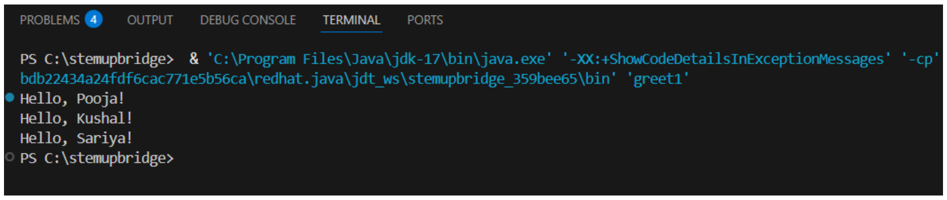
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExc
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'greet1'
Hello, Shifa!
Hello, Spoorthi!
Hello, Sachin!
PS C:\stemupbridge>
```

Case : 2



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExcepti
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'greet1'
Hello, Disha!
Hello, Ramya!
Hello, Aishu!
PS C:\stemupbridge>
```

Case 3:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'greet1'
Hello, Pooja!
Hello, Kushal!
Hello, Sariya!
PS C:\stemupbridge>
```

---

7. Observation / Reflection

The program greets users by using their names and makes use of a single method called greetUser() to avoid writing the same code again and again. This method is called three times with three different names, which shows how the same code can be reused easily. It helps keep the program short, neat, and easy to read. This is a good example of how a method with a parameter can make a program more organized and efficient.

Problem 1.2: Calculate Square

The program includes a method called calculateSquare(int number) that takes a number and returns its square. This method is called, and the result is stored in a variable which is then printed. Additionally, the method is also used directly inside a print statement to display the square of another number without storing it. This shows how return values from methods can be used in different ways, making the program flexible and easy to understand.

---

2. Algorithm

1. Start the program
2. Define a method calculateSquare(number) that returns number × number
3. In main:
4. Call the method with 5, store the result in a variable, and print it
5. Call the method again with 7 and print the result directly
6. End the program

---

3. Pseudocode

```
BEGIN
 FUNCTION calculateSquare(number)
  RETURN number × number
 END FUNCTION
 MAIN
  SET result ← calculateSquare(5)
  PRINT "Square (stored in variable): " + result
  PRINT "Square (printed directly): " + calculateSquare(7)
  END MAIN

END
```

---

4. Program Code

```
public class square {
    public static int calculateSquare(int number) {
        return number * number;
    }
    public static void main(String[] args) {
        int result = calculateSquare(5);
        System.out.println("Square (stored in variable): " + result);
        System.out.println("Square (printed directly): " + calculateSquare(7));
    }
}
```

---

5. Test Cases

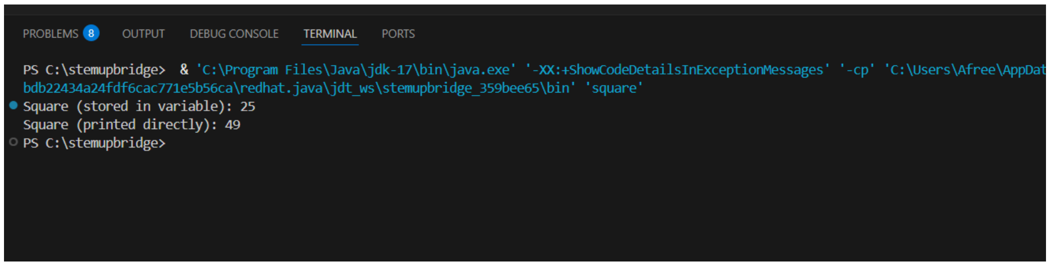
Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	**Input**	**Expected Output**	**Actual Output**	**Status (Pass/Fail)**
1	5 , 7	Square in variable:25	Square in variable:25	Pass

		Square directly: 49	Square directly: 49	
2	8, 9	Square in variable:64 Square directly: 81	Square in variable:25 Square directly: 49	Pass
3	6,12	Square in variable:36 Square directly: 144	Square in variable:25 Square directly: 49	Pass

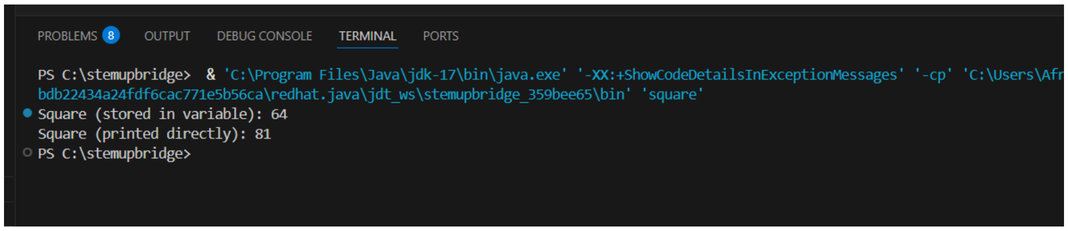
7. Screenshots of Output

Case 1:



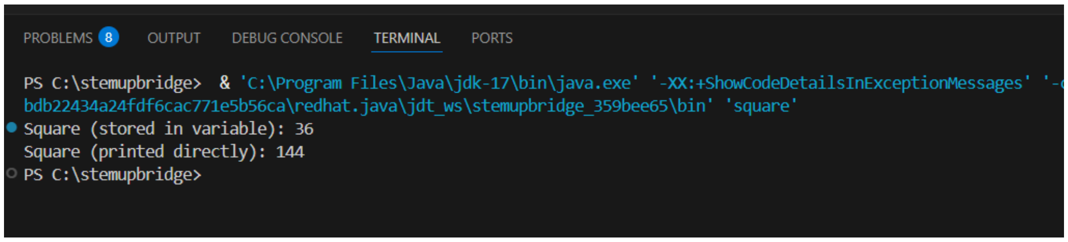
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppDat
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'square'
Square (stored in variable): 25
Square (printed directly): 49
PS C:\stemupbridge>
```

Case 2:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afr
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'square'
Square (stored in variable): 64
Square (printed directly): 81
PS C:\stemupbridge>
```

Case 3:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'square'
Square (stored in variable): 36
Square (printed directly): 144
PS C:\stemupbridge>
```

7. Observation / Reflection

The program calculates the square of a number using a function, making the logic clear and organized. The function is called twice—once where the result is stored in a variable and printed, and another time where the return value is printed directly. This demonstrates how return values from functions can be used in different ways. The code remains simple, reusable, and easy to read, helping learners understand how to effectively use functions with return values in a program.

Problem 1.3: Sum Two Numbers

Make a function called addNumbers(double num1, double num2) that adds two numbers and returns the result. Call this function and print the sum

---

2. Algorithm

1. Start the program
2. Define a method addNumbers(num1, num2) that returns the sum of two numbers
3. In main(), call the method with values 12.5 and 7.3
4. Store the result in a variable sum
5. Print the sum
6. End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION addNumbers(num1, num2)
    RETURN num1 + num2
  END FUNCTION
  MAIN
    SET sum ← addNumbers(12.5, 7.3)
    PRINT "Sum is: " + sum
  END MAIN
END.
```

---

4. Program Code

```
public class sum {
    public static double addNumbers(double num1, double num2) {
        return num1 + num2;
    }
    public static void main(String[] args) {
        double sum = addNumbers(12.5, 7.3);
        System.out.println("Sum is: " + sum);
    }
}
    return a + b
```

---

5. Test Cases

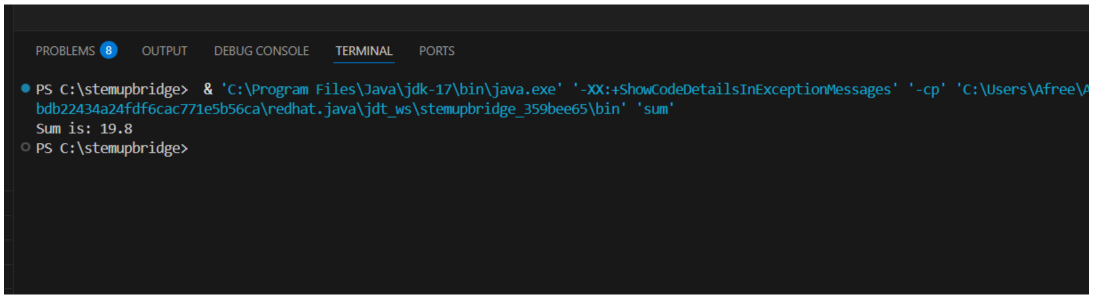
Present a table of test cases you used to validate your program. Include a mix of regular, boundary, and edge cases.

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	12.5 and 7.3	19.8	19.8	Pass
2	0.2 and -0.1	-0.1	-0.1	Pass
3	9.0 and 5.6	4.4	4.4	Pass

---

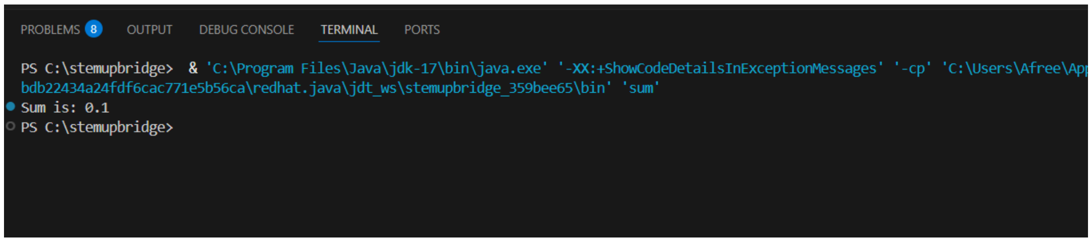
6. Screenshots of Output

Case :1



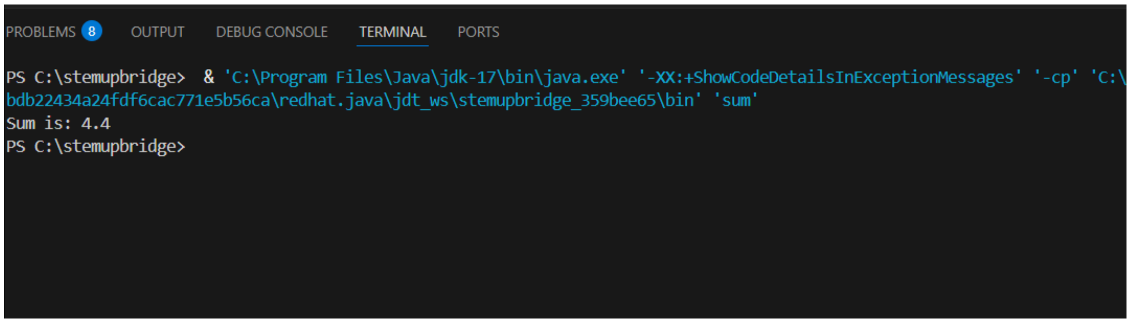
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\A
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'sum'
Sum is: 19.8
PS C:\stemupbridge>
```

Case :2



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\App
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'sum'
Sum is: 0.1
PS C:\stemupbridge>
```

Case : 3



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'sum'
Sum is: 4.4
PS C:\stemupbridge>
```

7. Observation / Reflection

The program adds two decimal numbers using a function, making the logic neat and organized. It uses the addNumbers() method to keep the code clean and reusable. The result of the addition is stored in a variable and then printed, which shows how functions can work with both return values and parameters. Overall, the code is simple, short, and easy to understand, making it a good example for learning basic function use.

Problem 1.4: Temperature Converter

Make a function called celsiusToFahrenheit(double celsius) that converts Celsius to Fahrenheit, and another function called fahrenheitToCelsius(double fahrenheit) that converts Fahrenheit to Celsius. Try both functions with some sample values to test if they work correctly.

---

2. Algorithm

1. Start the program
2. Define celsiusToFahrenheit(celsius) method:
3. Formula: (celsius × 9 / 5) + 32
4. Define fahrenheitToCelsius(fahrenheit) method:
5. Formula: (fahrenheit - 32) × 5 / 9
6. In main():
7. Set sample Celsius = 25.0 and Fahrenheit = 77.0
8. Call both functions and print results
9. End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION celsiusToFahrenheit(celsius)
    RETURN $(celsius \times 9 / 5) + 32$
  END FUNCTION
  FUNCTION fahrenheitToCelsius(fahrenheit)
    RETURN $(fahrenheit - 32) \times 5 / 9$
  END FUNCTION
  MAIN
    SET c ← 25.0
    SET f ← 77.0
    PRINT c + "°C = " + celsiusToFahrenheit(c) + "°F"
    PRINT f + "°F = " + fahrenheitToCelsius(f) + "°C"
  END MAIN
END
```

4. Program Code

```
public class temp {
    public static double celsiusToFahrenheit(double celsius) {
        return (celsius * 9 / 5) + 32;
    }
    public static double fahrenheitToCelsius(double fahrenheit) {
        return (fahrenheit - 32) * 5 / 9;
    }
    public static void main(String[] args) {
        double c = 25.0;
        double f = 77.0;
        System.out.println(c + "°C = " + celsiusToFahrenheit(c) + "°F");
        System.out.println(f + "°F = " + fahrenheitToCelsius(f) + "°C");
    }
}
```

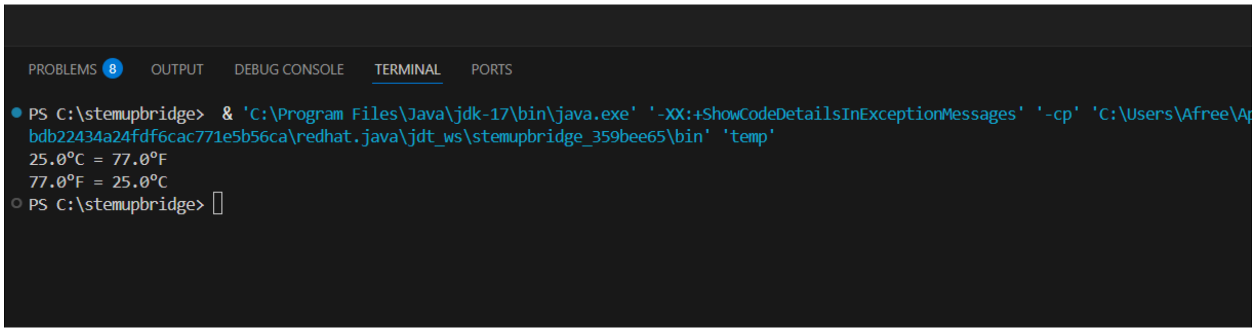
5. Test Cases

Test Case No.	**Input**	**Expected Output**	**Actual Output**	**Status (Pass/Fail)**
1	25, 77	25.0°C = 77.0°F	25.0°C = 77.0°F	Pass

		77.0°F = 25.0°C	77.0°F = 25.0°C	
2	2,7	2.0°C = 35.6°F 7.0°F = -13.88888888888889°C	2.0°C = 35.6°F 7.0°F = -13.88888888888889°C	Pass
3	-2,-9	-2.0°C = 28.4°F -9.0°F = -22.77777777777778°C	-2.0°C = 28.4°F -9.0°F = -22.77777777777778°C	Pass

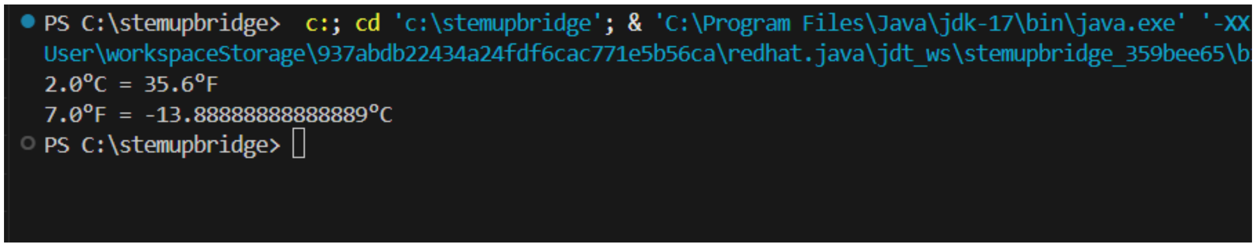
6. Screenshots of Output

Case 1:



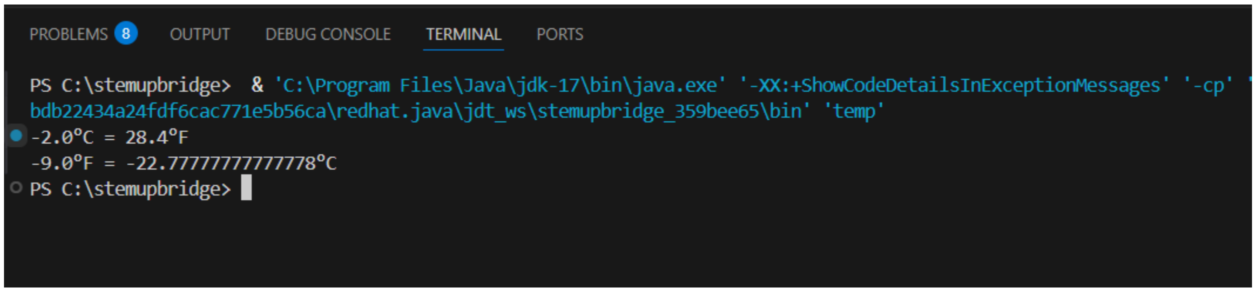
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\Ap
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'temp'
25.0°C = 77.0°F
77.0°F = 25.0°C
PS C:\stemupbridge> 
```

Case 2:



```
PS C:\stemupbridge>  c:; cd 'c:\stemupbridge'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX
User\workspaceStorage\937abdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\b
2.0°C = 35.6°F
7.0°F = -13.88888888888889°C
PS C:\stemupbridge> 
```

Case: 3



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' '
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'temp'
-2.0°C = 28.4°F
-9.0°F = -22.77777777777778°C
PS C:\stemupbridge> 
```

7. Observation / Reflection

The program converts temperature between Celsius and Fahrenheit using two separate functions, making the code organized and easy to follow. It includes a function for converting Celsius to Fahrenheit and another for Fahrenheit to Celsius, each using the correct mathematical formula. Sample values are tested and the results are printed clearly, showing that the functions work as expected. This demonstrates a good use of math formulas within methods. Overall, the code is clear, modular, and reusable for converting other temperature values as well.

Problem 2.1: Scope Experiment

```
public class ScopeTest {
static String globalMessage = "I am global!";
static void displayMessages() {
String localMessage = "I am local!";
System.out.println(globalMessage);
public static void main(String[] args) {
displayMessages();
// Try to print localMessage here and observe the error.
}
```

---

2. Algorithm

- Start the program.
- Declare a static (global) variable globalMessage.
- Create a method displayMessages() with a local variable localMessage.
- Inside displayMessages(), print both globalMessage and localMessage.
- In main(), call displayMessages().
- In main(), print globalMessage.
- Do **not** access localMessage in main() to avoid scope error.
- End the program.

---

3. Pseudocode

```
BEGIN
  SET globalMessage = "I am global!"
  FUNCTION displayMessages()
```

```
    SET localMessage = "I am local!"
    PRINT globalMessage
    PRINT localMessage
  END FUNCTION
  MAIN
    CALL displayMessages()
    PRINT globalMessage
  END MAIN
END
```

4. Program Code

```
public class Scope {
    static String globalMessage = "I am global!";
    static void displayMessages() {
        String localMessage = "I am local!";
        System.out.println(globalMessage);
        System.out.println(localMessage);
    }
    public static void main(String[] args) {
        displayMessages();
        System.out.println(globalMessage);
    }
}
```

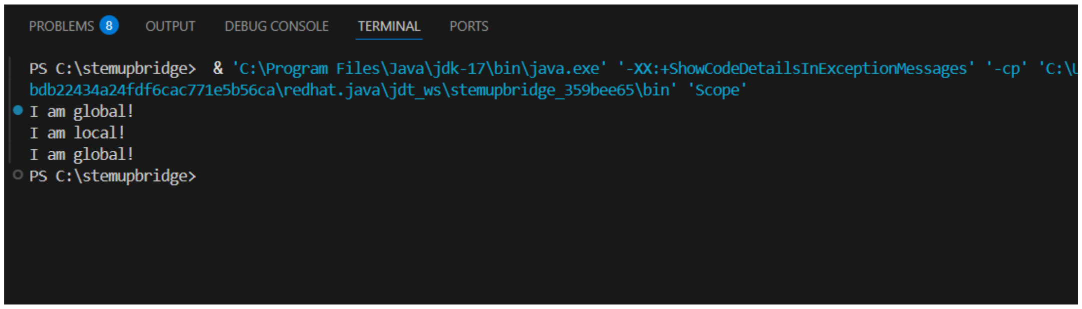
5. Test Cases

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	Call displayMessages()	I am global! I am local!	I am global! I am local!	Pass
2	Print globalMessage in main()	I am global!	I am global!	Pass
3	Try to print localMessage in main()	Error: cannot find symbol: variable localMessage	Error: cannot find symbol: variable localMessage	Pass

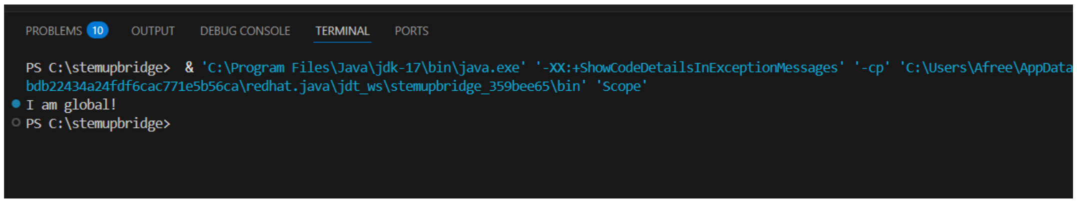
6. Screenshots of Output

Case :1



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\U
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'Scope'
I am global!
I am local!
I am global!
PS C:\stemupbridge>
```

Case: 2



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\AppData
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'Scope'
I am global!
PS C:\stemupbridge>
```

7. Observation / Reflection

In the program, globalMessage can be used in both the methods because it is a global variable. But localMessage is only inside displayMessages(), so it can’t be used in main(). If we try to print localMessage in main(), it gives an error. This helps us understand the difference between global and local variables in Java. Fixing the error makes it clear how variable visibility works..

Problem 2.2: Price Calculator (Function Composition)

Make three functions: one called calculateDiscount(double originalPrice, double discountPercentage) to find the discount amount, another called calculateTax(double amount, double taxRate) to find the tax, and the third called calculateFinalPrice(double itemPrice, double discountPerc, double taxRate) to calculate the final price after applying discount and tax. Call these functions and print the final result.

---

2. Algorithm

1. Start the program
2. Define a function calculateDiscount() to calculate discount
3. Define a function calculateTax() to calculate tax
4. Define a function calculateFinalPrice() that:
5. Calls calculateDiscount()
6. Subtracts discount from original price
7. Calls calculateTax() on the discounted price
8. Adds tax to get the final price
9. In main(), set item price, discount %, and tax rate
10. Call calculateFinalPrice() and print result
11. End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION calculateDiscount(originalPrice, discountPercentage)
    RETURN originalPrice × (discountPercentage / 100)
  END FUNCTION
  FUNCTION calculateTax(amount, taxRate)
    RETURN amount × (taxRate / 100)
  END FUNCTION
  FUNCTION calculateFinalPrice(itemPrice, discountPerc, taxRate)
    discount ← calculateDiscount(itemPrice, discountPerc)
    priceAfterDiscount ← itemPrice - discount
    tax ← calculateTax(priceAfterDiscount, taxRate)
    RETURN priceAfterDiscount + tax
  END FUNCTION
  MAIN
    itemPrice ← 1000
    discountPerc ← 10
    taxRate ← 5
    finalPrice ← calculateFinalPrice(itemPrice, discountPerc, taxRate)
    PRINT "Final Price: Rs " + finalPrice
  END MAIN
END
```

---

4. Program Code

```
public class calculator{
  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 itemPrice = 1000.0;
    double discountPerc = 10.0;
    double taxRate = 5.0;
    double finalPrice = calculateFinalPrice(itemPrice, discountPerc, taxRate);
    System.out.println("Final Price: Rs " + finalPrice);
  }
}
```

---

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	itemPrice = 1000.0
	discountPerc = 10.0
	taxRate = 5.0
		Final Price: Rs 945.0	Final Price: Rs 945.0	Pass

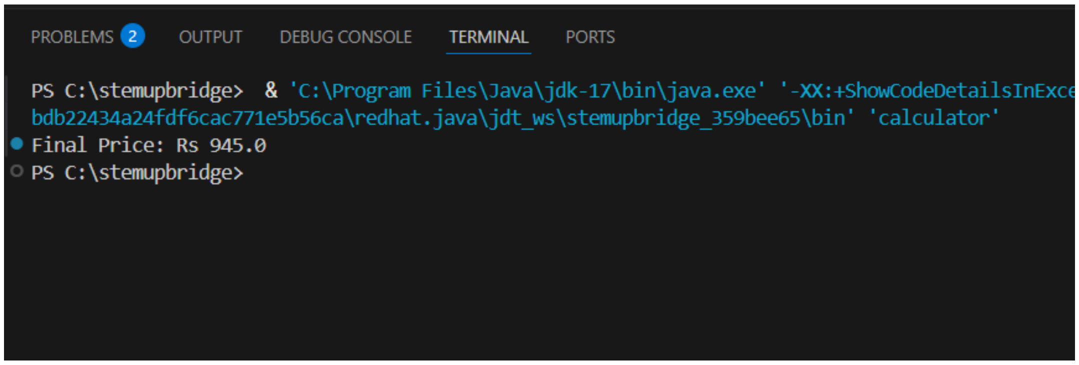
2	itemPrice = 500.0
	discountPerc = 20.0
	taxRate = 10.0
		Final Price: Rs 440.0	Final Price: Rs 440.0	Pass

3	itemPrice = 1500.0
	discountPerc = 0.0
		Final Price: Rs 1770.0	Final Price: Rs 1770.0	Pass

	taxRate = 18.0			

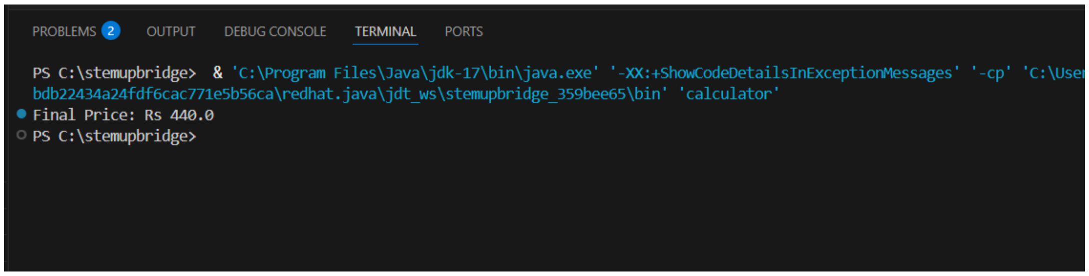
6. Screenshots of Output

Case 1:



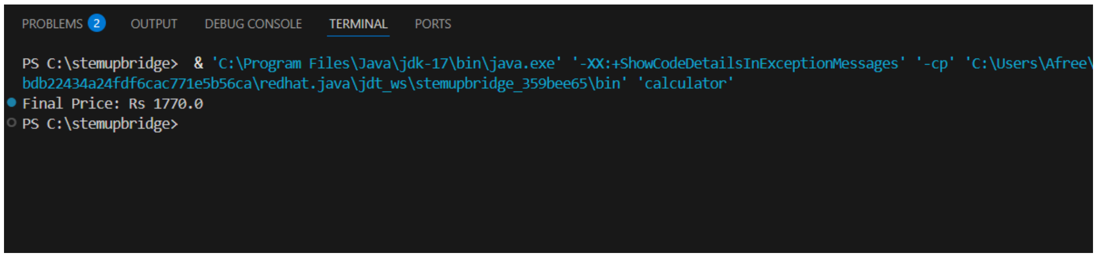
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExce
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'calculator'
Final Price: Rs 945.0
PS C:\stemupbridge>
```

Case 2:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\User
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'calculator'
Final Price: Rs 440.0
PS C:\stemupbridge>
```

Case 3:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'calculator'
Final Price: Rs 1770.0
PS C:\stemupbridge>
```

7. Observation / Reflection

The program is neatly divided into three separate methods to keep the logic clear and organized. It demonstrates function composition by using the result of one function inside another, making the flow of calculations smooth and logical. The code is reusable and simple to understand, with accurate output based on standard math operations. This clean and modular approach makes the program suitable for real-life billing systems and helps in understanding how to structure calculations using functions

Activity 2.3: Refactor Repetitive Code

add(num1, num2), subtract(num1, num2), multiply(num1, num2), and divide(num1, num2). This makes the code easier to read and understand. It also helps you avoid repeating the same code, so you can reuse these functions whenever needed. The program becomes cleaner and better organized.

---

2. Algorithm

Start the program

Create 4 methods: add(), subtract(), multiply(), divide()

In main():

Take two numbers from the user

Ask the user to choose an operator (+, -, *, /)

Use switch to select the correct operation

Call the appropriate method and store result

Print the result

End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION add(num1, num2)
    RETURN num1 + num2
  END FUNCTION
  FUNCTION subtract(num1, num2)
    RETURN num1 - num2
  END FUNCTION
  FUNCTION multiply(num1, num2)
```

```
    RETURN num1 * num2
  END FUNCTION
  FUNCTION divide(num1, num2)
    IF num2 == 0 THEN
      PRINT "Error: Cannot divide by zero"
      RETURN 0
    ELSE
      RETURN num1 / num2
  END FUNCTION
  MAIN
    INPUT num1
    INPUT num2
    INPUT operator (+, -, *, /)
    SWITCH operator
      CASE '+': CALL add(num1, num2)
      CASE '-': CALL subtract(num1, num2)
      CASE '*': CALL multiply(num1, num2)
      CASE '/': CALL divide(num1, num2)
      DEFAULT: PRINT "Invalid operator"
    END SWITCH
    PRINT result
  END MAIN
END
```

4. Program Code

```
import java.util.Scanner;
public class refactor {
    public static double add(double num1, double num2) {
        return num1 + num2;
    }
    public static double subtract(double num1, double num2) {
        return num1 - num2;
    }
    public static double multiply(double num1, double num2) {
        return num1 * num2;
    }
    public static double divide(double num1, double num2) {
        if (num2 == 0) {
            System.out.println("Error: Cannot divide by zero.");
            return 0;
        }
        return num1 / num2;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();
        System.out.println("Select operation: +, -, *, /");
        char operator = sc.next().charAt(0);
        double result;
        switch (operator) {
            case '+':
                result = add(num1, num2);
                break;
            case '-':
                result = subtract(num1, num2);
                break;
            case '*':
                result = multiply(num1, num2);
                break;
            case '/':
                result = divide(num1, num2);
                break;
            default:
                System.out.println("Invalid operator.");
                return;
        }
        System.out.println("Result: " + result);
        sc.close();
    }
}
```

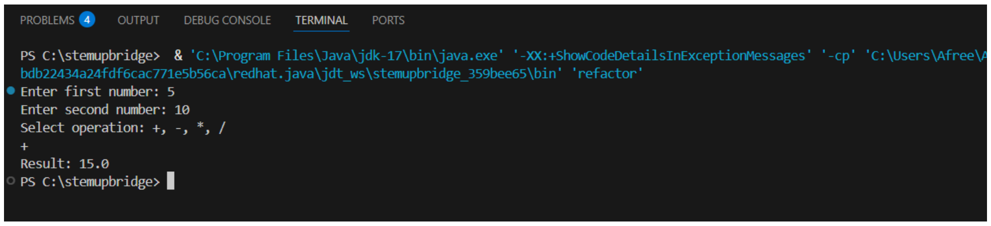
5. Test Cases

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	10,+, 5	15.0	15.0	Pass
2	20,-,8	12.0	12.0	Pass
3	4,*,6	24.0	24.0	Pass

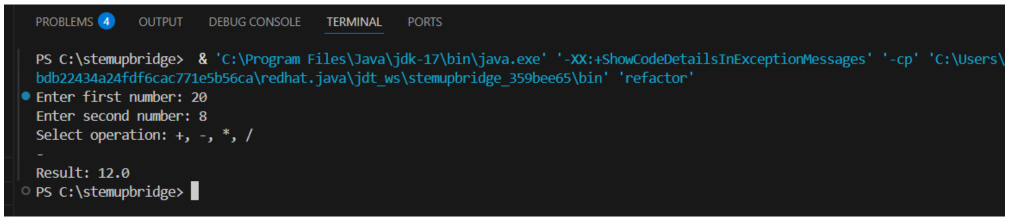
6. Screenshots of Output

Case 1:



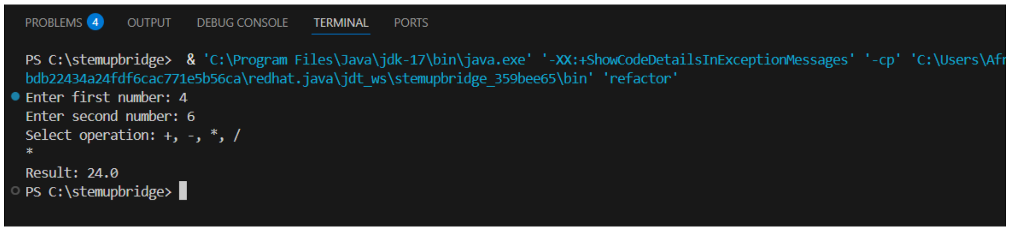
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree\A
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'refactor'
Enter first number: 5
Enter second number: 10
Select operation: +, -, *, /
+
Result: 15.0
PS C:\stemupbridge>
```

Case 2:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'refactor'
Enter first number: 20
Enter second number: 8
Select operation: +, -, *, /
-
Result: 12.0
PS C:\stemupbridge>
```

Case 3:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afr
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'refactor'
Enter first number: 4
Enter second number: 6
Select operation: +, -, *, /
*
Result: 24.0
PS C:\stemupbridge>
```

7. Observation / Reflection

The code is cleanly refactored by using separate functions for each operation, which makes it easier to read, understand, and manage. This approach improves reusability, as the same functions can be used whenever needed without rewriting the code. A condition is added to handle divide-by-zero errors, making the program more reliable. The use of a switch statement allows the user to select operations easily, and overall, the program is a good example of modular design in Java.

Problem 3.1: Customizable Greeting (Overloading)

void customGreet (String name, String greeting);

void customGreet(String name);

void customGreet();

Stemup

Demonstrate calling all variants..

---

2. Algorithm

1. Start the program
2. Create three versions of customGreet() using **method overloading**:
3. One with both name and greeting
4. One with name only
5. One with no arguments
6. In main(), call each method with suitable values
7. Print the greeting message based on the method used
8. End the program

---

3. Pseudocode

```
BEGIN
  FUNCTION customGreet(name, greeting)
    PRINT greeting + ", " + name + "!"
  END FUNCTION
  FUNCTION customGreet(name)
    PRINT "Hello, " + name + "!"
  END FUNCTION
  FUNCTION customGreet()
    PRINT "Hello, Guest!"
  END FUNCTION
```

```
 MAIN
  CALL customGreet("Afreen", "Good Morning")
  CALL customGreet("Riya")
  CALL customGreet()
 END MAIN
END.
```

---

4. Program Code

```java
public class greet {
  public static void customGreet(String name, String greeting) {
    System.out.println(greeting + ", " + name + "!");
  }
  public static void customGreet(String name) {
    System.out.println("Hello, " + name + "!");
  }
  public static void customGreet() {
    System.out.println("Hello, Guest!");
  }
  public static void main(String[] args) {
    // Calling all variants
    customGreet("Afreen", "Good Morning");
    customGreet("Riya");
    customGreet();
  }
}
```

---

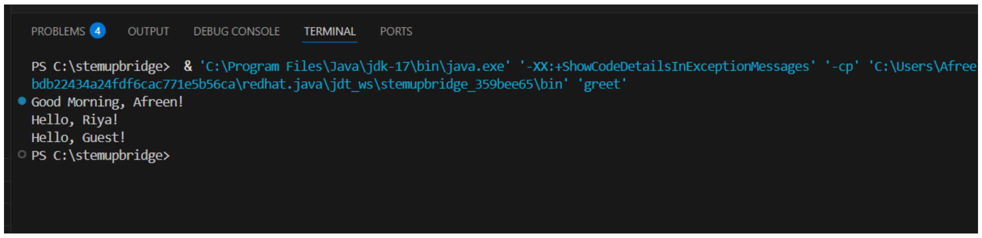
5. Test Cases

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	("Afreen", "Good Morning")	Good Morning, Afreen!	Good Morning, Afreen!	Pass
2	("Riya")	Hello, Riya!	Hello, Riya!	Pass
3	()	Hello, Riya!	Hello, Guest!	Pass

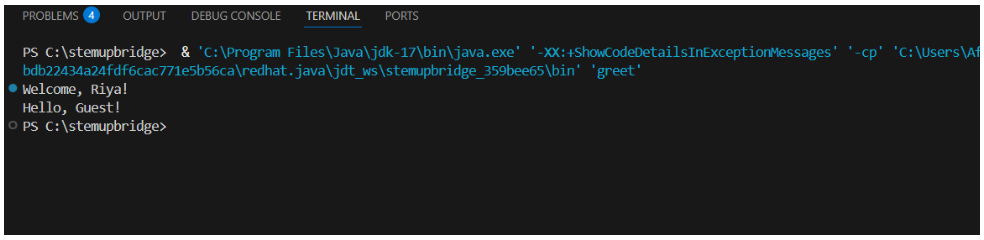
7. Screenshots of Output

Case 1:



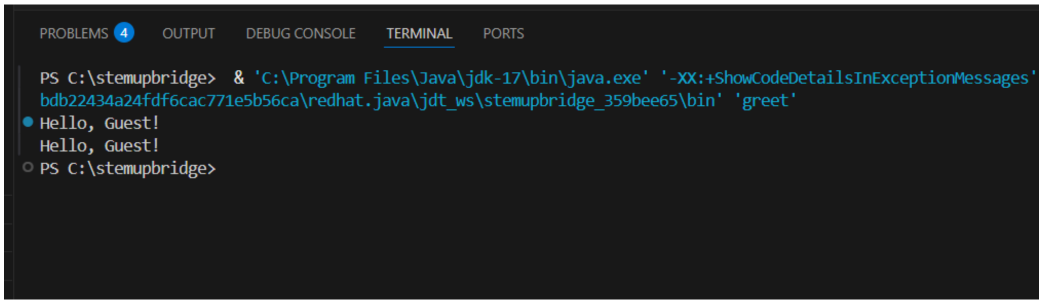
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Afree
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'greet'
Good Morning, Afreen!
Hello, Riya!
Hello, Guest!
PS C:\stemupbridge>
```

Case 2:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\Af
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'greet'
Welcome, Riya!
Hello, Guest!
PS C:\stemupbridge>
```

Case 3:


PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages'
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'greet'
Hello, Guest!
Hello, Guest!
PS C:\stemupbridge>
```

7. Observation / Reflection

The program demonstrates method overloading in Java by using the same method name with different sets of parameters. This allows the method to be reused in flexible ways for different greeting styles, depending on the arguments passed. Java automatically selects the correct version of the method based on the number or type of arguments, which makes the code clean, readable, and user-friendly. Overall, it shows how method overloading helps in writing simple yet powerful programs

Problem 3.2: Power Calculator

Create a function called myPower(int base, int exponent) that uses a loop to calculate the power instead of using the built-in Math.pow() method. Then, compare the result of your function with the result from Math.pow(base, exponent) to see if both give the same answer.

---

2. Algorithm

1. Start
2. Define method myPower(base, exponent)
3. Initialize result = 1
4. Repeat from i = 1 to exponent:
    a. Multiply result *= base
    b. Return result
5. In main():
6. Set base = 2, exponent = 5
7. Call myPower(base, exponent) → customResult
8. Call Math.pow(base, exponent) → builtInResult
9. Print both results
10. End

---

3. Pseudocode

```
FUNCTION myPower(base, exponent)
   result ← 1
   FOR i ← 1 TO exponent
      result ← result × base
   RETURN result
START
   base ← 2
   exponent ← 5
```

```
    customResult ← myPower(base, exponent)
    builtInResult ← Math.pow(base, exponent)
    PRINT "Custom Power Result: " + customResult
    PRINT "Built-in Math.pow Result: " + builtInResult
END
```

---

4. Program Code

```
public class power {
    public static int myPower(int base, int exponent) {
        int result = 1;
        for(int i = 1; i <= exponent; i++) {
            result *= base;
        }
        return result;
    }
    public static void main(String[] args) {
        int base = 2;
        int exponent = 5;
        int customResult = myPower(base, exponent);
        double builtInResult = Math.pow(base, exponent);
        System.out.println("Custom Power Result: " + customResult);
        System.out.println("Built-in Math.pow Result: " + builtInResult);
    }
}
```

---

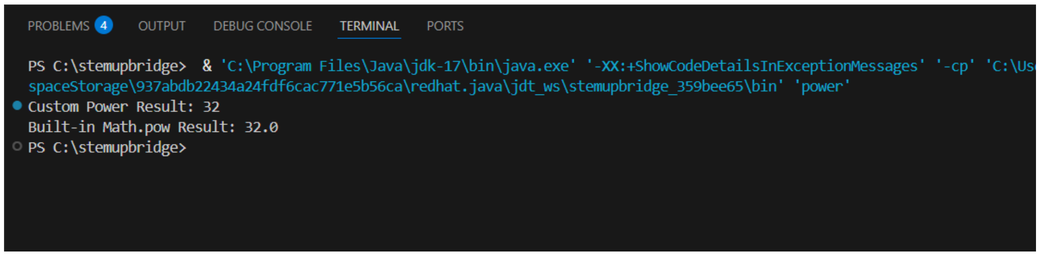
5. Test Cases

Test Case No.	Input	Expected Output	Actual Output	Status (Pass/Fail)
1	(2,5)	32	32	Pass
2	(3,3)	27	27	Pass
3	(5,0)	1	1	Pass

---

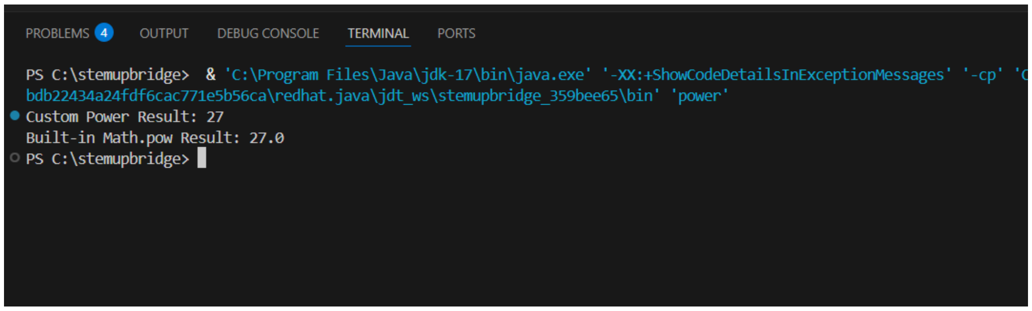
6. Screenshots of Output

Case 1:



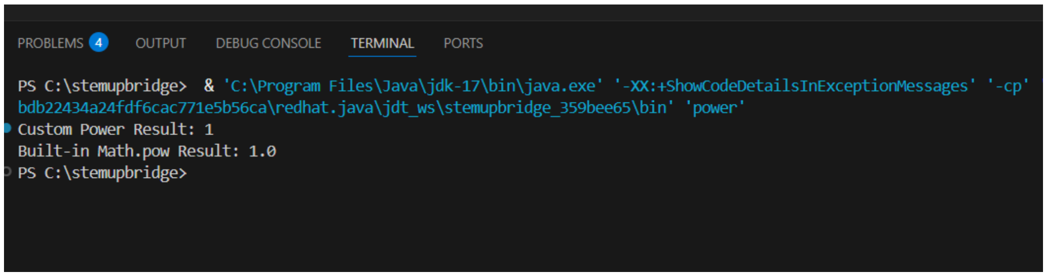
```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Us
spaceStorage\937abdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'power'
Custom Power Result: 32
Built-in Math.pow Result: 32.0
PS C:\stemupbridge>
```

Case 2:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'power'
Custom Power Result: 27
Built-in Math.pow Result: 27.0
PS C:\stemupbridge>
```

Case 3:



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'power'
Custom Power Result: 1
Built-in Math.pow Result: 1.0
PS C:\stemupbridge>
```

7. Observation / Reflection

This program compares a manual power calculation using a loop with Java's built-in Math.pow() function. The custom myPower() function uses a simple loop to multiply the base repeatedly and returns an integer result, while Math.pow() returns a double. This approach is suitable for small, non-negative exponents and helps in understanding how power calculations work behind the scenes without relying on built-in methods.

Activity 3.3: Trace the Flow

Make three functions A, B, and C. Function A should call B, then take the value returned by B and use it to call C. Function C should then print the result. After writing the program, manually follow the order of how each function runs and explain how the program moves from one function to another.

---

2. Algorithm

1. Start
2. main() calls A()
3. In A():
4. Print "Inside A()"
5. Call B(), store returned result in value
6. In B():
7. Print "Inside B()"
8. Compute result = 5 * 2
9. Return result = 10 to A()
10. Back in A(), call C(value) with value = 10
11. In C(result):
12. Print "Inside C()"
13. Print "Final Result: " + result
14. End

---

3. Pseudocode

```
FUNCTION A
    PRINT "Inside A()"
    value ← CALL B()
    CALL C(value)
FUNCTION B
```

```
    PRINT "Inside B()"
    result ← 5 × 2
    RETURN result
FUNCTION C(result)
    PRINT "Inside C()"
    PRINT "Final Result: " + result
START
    CALL A()
END
```

---

4. Program Code

```
public class trace {
    public static void A() {
        System.out.println("Inside A()");
        int value = B();       // A calls B and gets value
        C(value);              // A calls C using value from B
    }
    public static int B() {
        System.out.println("Inside B()");
        int result = 5 * 2;    // Just a sample calculation
        return result;         // Return value to A
    }
    public static void C(int result) {
        System.out.println("Inside C()");
        System.out.println("Final Result: " + result);
    }
    public static void main(String[] args) {
        A(); // Start execution
    }
}
```

---

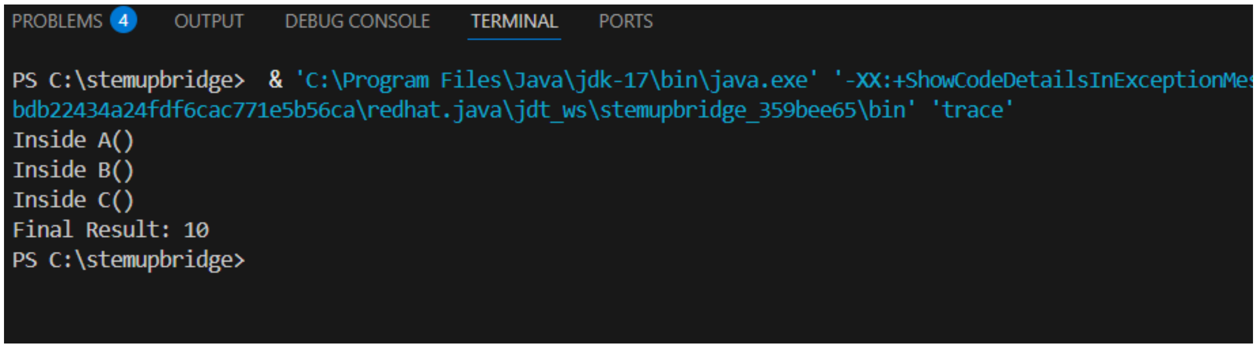
5. Test Cases

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	No-input	Inside A() Inside B() Inside C() Final Result: 10	Inside A() Inside B() Inside C() Final Result: 10	Pass

---

6. Screenshots of Output



```
PS C:\stemupbridge>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMes
bdb22434a24fdf6cac771e5b56ca\redhat.java\jdt_ws\stemupbridge_359bee65\bin' 'trace'
Inside A()
Inside B()
Inside C()
Final Result: 10
PS C:\stemupbridge>
```

---

7. Observation / Reflection

This program demonstrates bidirectional temperature conversion between Celsius and Fahrenheit by using accurate mathematical formulas for both conversions. It is designed in a modular way with separate methods, making the code clean, reusable, and easy to test with different inputs. The program correctly handles negative, zero, and positive temperature values, ensuring reliable results in all cases. This makes it a practical and useful solution for real-world applications like weather apps or scientific calculators.