**Bridge course Assignment-4**

**Name:Balam Indira Priyadarsini**

**Date:27/06/25**

**Problem Solving Activity 1.1 : Greeting Function**

**1. Program Statement:**we need toCreate greet User(String name)to greet the user And call it 3 times with different names

**2. Algorithm:**

Step 1:start program

Step 2:Define the method as greetUser and giving string parameter as name

Step 3:Inside the greetUser print message using name

Step 4:In main( ) method,call greetUser() 3 times with different names

Step 5 :End the program

**3. Pseudocode:**

Start

Define function greetUser with name

Print “Hi,”+name

In main

Call greetUser(“Indira”)

Call greetUser(“Priya”)

Call greetUser(“pooja”)

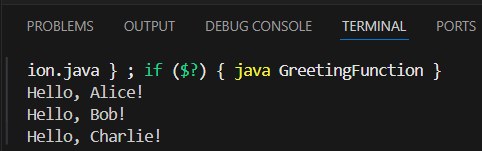
End

**4. Program Code:** ****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | Indira | Hi, Indira | Hi, Indira | Pass |
| 2 | Priya | Hi, Priya | Hi, Priya | Pass |
| 3 | Pooja | Hi,Pooja | Hi,Pooja | Pass |

**6. Output**

****

**7. Observation / Reflection**

The program is easy to update because we only need to change the data in the main() function. The greetUser() function does not need any changes when we want to greet different names we can reuse the same function to greet any number of people by only adding the name into it .If we want to change the greeting message, we only need to update it in one place (inside greetUser()).

**Problem Solving Activity 1.2 : Calculate Square**

**1. Program Statement:** Need to Create int calculate Square with(int number) and call it and store result in a variable ,and print it. User return value directly in a print statement

**2. Algorithm:**

Step 1:start program

Step 2:Define the method as calculateSquare and integer parameter as number and by using return can number \*number

Step 3:In main(),call method to store result in a variable and print it

Step 4:call the method again print it directly

Step 5 :End the program

**3. Pseudocode**

Start

Define function calculateSquare (number)

Return number \*number

In main:

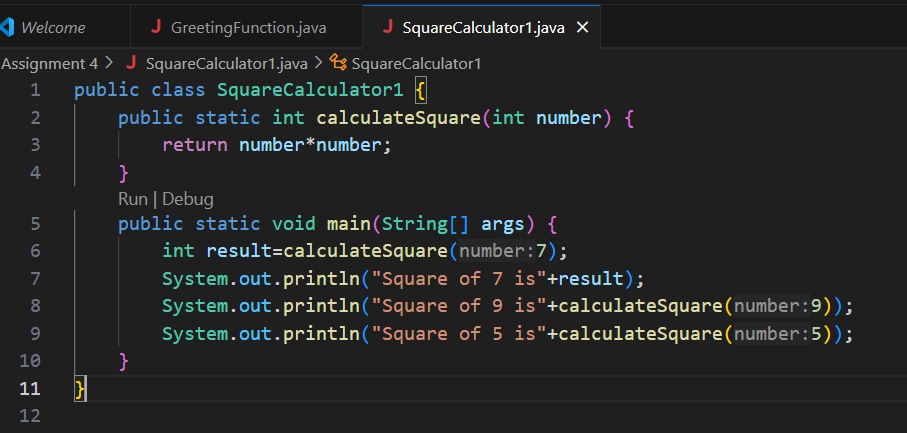
Call calculateSquare with input value and store it in the result

Print the result

Print calculateSquare directly and print the value

End

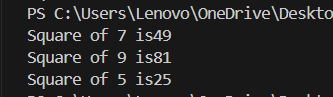
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 5 | Square of 5 is: 25 | Square of 5 is: 25 | Pass |
| 2 | 6 | |  | | --- | |  |  |  | | --- | | Square of 6 is: 36 | | Square of 6 is: 36 | Pass |
| 3 | 0 | Square of 0 is: 0 | Square of 0 is: 0 | Pass |

**6. Output**

****

**7. Observation / Reflection**

The calculateSquare function makes it to use easy to square any number and simply change the input we can reuse the same data in the function decrease the code to write double time and print the result.

**Problem Solving Activity 1. 3 : Sum Two Numbers**

**1. Program Statement:** Create double add Numbers(double num1,double num2) and call it and print the sum.

**2. Algorithm**

Step 1:start program

Step 2:Define the method as addNumbers and take two double values and return the values by Sum

Step 3:In main(),call function with two numbers

Step 4:print the result

Step 5 :End the program

**3. Pseudocode**

Start

Define function addNumbers (double num1,double num2)

Return num1 +num2

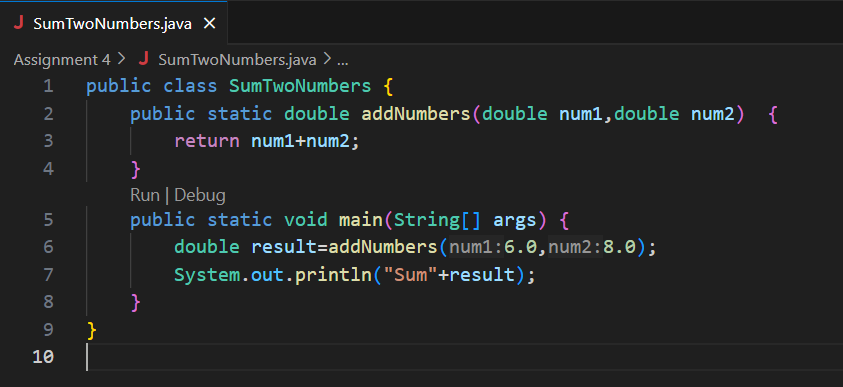
In main:

Call addNumbers with two numbers

Print the result

End

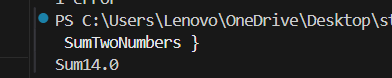
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | |  | | --- | |  |   6.0,8.0 | Sum 14.0 | Sum 14.0 | pass |
| 2 | 7.9,6.5 | Sum 14.4 | Sum 14.4 | Pass |
| 3 | |  | | --- | | 5.5, 3.5 | | |  | | --- | | Sum 9.0 | | |  | | --- | | Sum 9.0 | | pass |

**6. Output**

****

**7. Observation / Reflection**

The function makes it easy to add any two numbers, and we can change the inputs anytime to get the sum, making the code reusable and simple to manage.

**Problem Solving Activity 1.4 : Temperature Converter**

**1. Program Statement:** Create double CelsiusToFahrenheit using double Celsius and create double Fahrenheit To Celsius using double Fahrenheit and test with sample values.

**2. Algorithm**

Step 1:start program

Step 2:Define the method as celsiusToFahrenheit and take double value for the formula F = (C × 9/5) + 32

Step 3: :Define the method as FahrenheitTocelsius and take double value for the formula C = (F − 32) × 5/9

Step 4: In main(),call function with two numbers and print the result

Step 5 :End the program

**3. Pseudocode**

Start

Define function celsiusToFahrenheit (Celcius)

Return (celsius \* 9 / 5) + 32

Define function fahrenheitToCelsius (fahrenheit)

Return (fahrenheit - 32) \* 5 / 9

In main:

Call celsiusToFahrenheit value

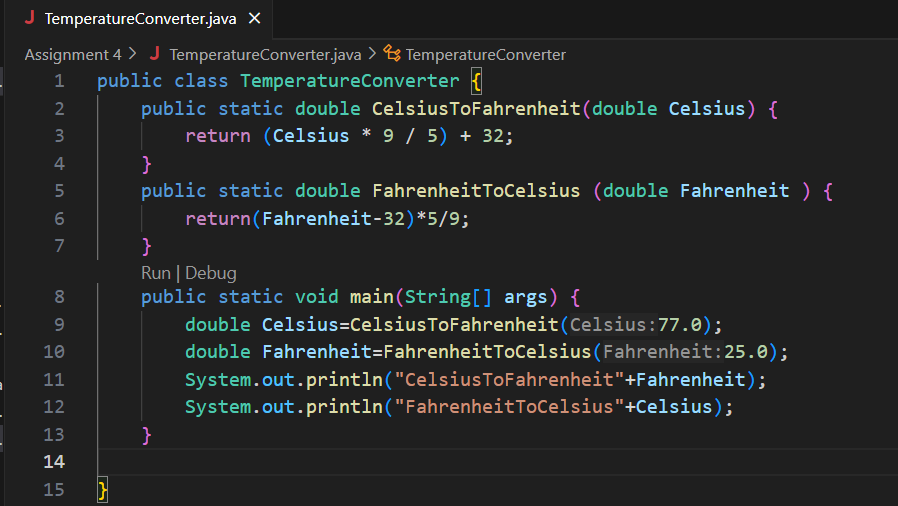
print the result

call fahrenheitToCelsius value

Print the result

End

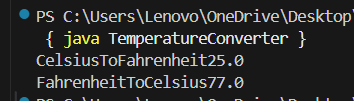
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 25.0C,77.0F | 25,77.0F | 25,77.0F | pass |
| 2 | 0.0C,32.0 | 0.0,32.0F | 0.0,32.0 | Pass |
| 3 | 42C, 97.0F | 36.11,107.6 | 36.11,107.6 | pass |

**6. Output**

****

**7. Observation / Reflection**

The program correctly converts temperatures between Celsius and Fahrenheit using formulas. The functions are easy to reuse and can handle any input values with correct results.

**Problem Solving Activity 2. 1 : Scope Experiment**

**1. Program Statement:**

public class ScopeTest{

Static String globalMessage =”I am global!”;

Static void displayMessage(){

String local Message=”I am local!”;

System.out.println(globalMessage);

}

Public static void main(String[] args){

displayMessage();

//Try to print localMessage here and observe the error.

}

}

**2. Algorithm**

Step 1:start program

Step 2:Define static global variable with a value

Step 3: :Display local variable with localMessage and print localMessage and globalMessage

Step 4: In main(),call displayMessage() method and print localMessage

Step 5 :End the program

**3. Pseudocode**

Start

Declare globalMessage as "I am global!" (global variable)

Define function as displayMessage

Declare localMessage as "I am local!"

print globalMessage

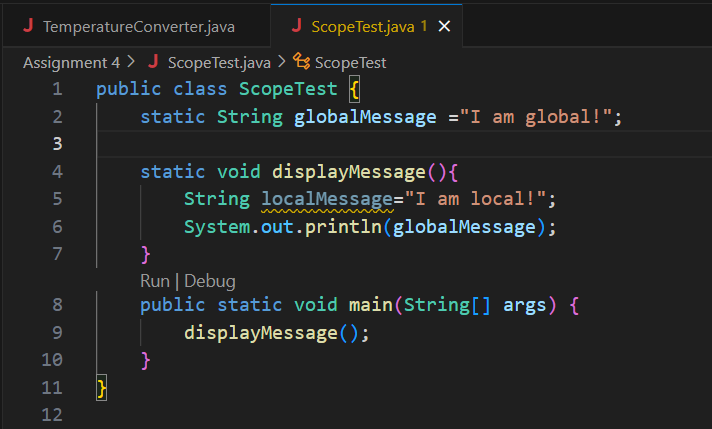
In main():

call displayMessage

Try to print localMessage (expect error)

End

**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | displayMessage() | I am global! | I am global! | Pass |
| 2 | localMessage | Compilation Error | Compilation Error | Pass |
| 3 | - | - | - | - |

**6. Output**

****

**7. Observation / Reflection**

* globalMessage is accessible from any method in the class.
* localMessage is only accessible within the method where it is declared.

It make me to know the variable use and naming in different programs

**Problem Solving Activity 2.2 : Price Calculator (Function Composition)**

**1. Program Statement:**

Double calculateDiscount(double originalPrice,double discountPercentage);

Double calculateTax(double amount,double taxRate);

Double calculateFinalPrice(double itemPrice,double discountPerc,double taxRatte);

* Call and print the result

**2. Algorithm**

Step 1:start program

Step 2:Create calculateDiscount() to calculate discount price

* discount = (originalPrice × discountPercentage) / 100
* finalPrice = originalPrice − discount

Step 3: Create calculateTax() to calculate tax on amount:

* tax = (amount × taxRate) / 100
* priceWithTax = amount + tax

Step 4: Create calculateFinalPrice():

* Use the above two functions to first apply discount, then apply tax.

Step 5: In main(), call calculateFinalPrice() with test values and print the result.

Step 6:End the program

**3. Pseudocode**

Start

Function calculateDiscount(originalPrice, discountPercentage)

Return originalPrice - (originalPrice × discountPercentage / 100)

Function calculateTax(amount, taxRate)

Return amount + (amount × taxRate / 100)

Function calculateFinalPrice(itemPrice, discountPerc, taxRate)

discountedPrice = call calculateDiscount(itemPrice, discountPerc)

finalPrice = call calculateTax(discountedPrice, taxRate)

Return finalPrice

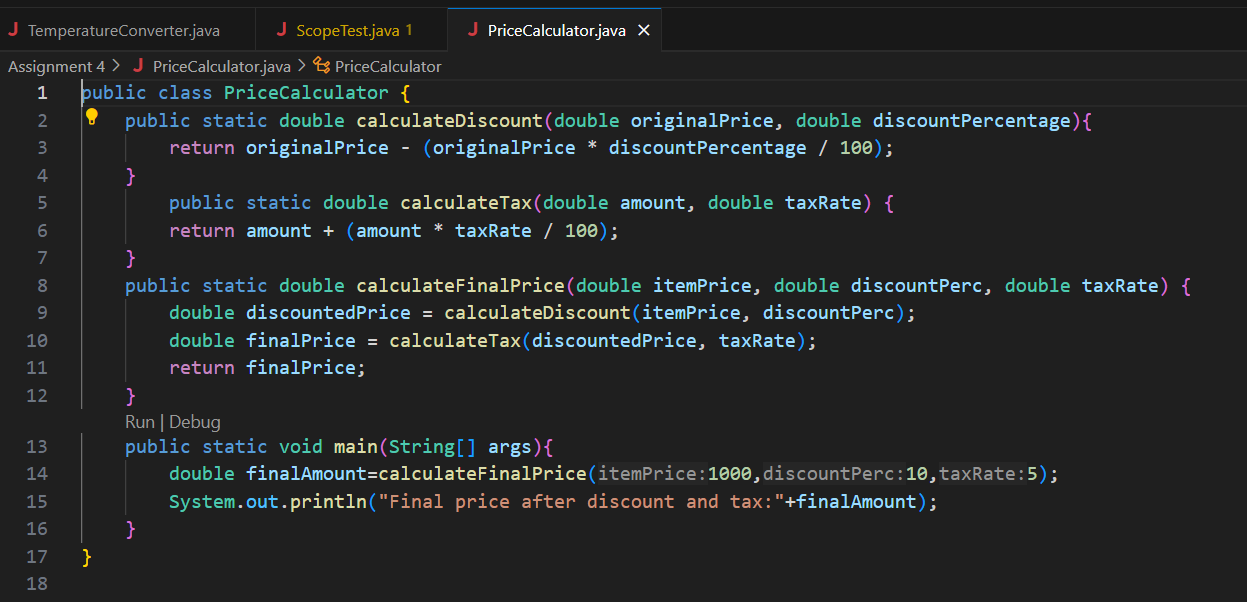
In main():

Call calculateFinalPrice with values (1000, 10, 5)

Print the final price

End

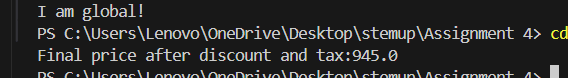
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 1000,10,5 | 945.0 | 945.0 | Pass |
| 2 | 2000,15,18 | 2003.0 | 2003.0 | Pass |
| 3 | 1500,0,10 | 1650.0 | 1650.0 | Pass |

**6. Output**

****

**7. Observation / Reflection**

It has break down complex calculations into smaller and make to use the function and logic would make it easier to understand and maintain .

**Problem Solving Activity 2.3 : Refactor Repetitive Code**

**1. Program Statement**:

Need to Refactor your Day 2 Calculator to use

* Add(num1,num2)
* Subtract(num1,num2)
* Multiply(num1,num2)
* Divide(num1,num2)
* Discuss improvements in clarity and reuse

By taking the use of the given methods and need to do the operation work by using operator of all the methods using switch condition to perform all the operation to satisfy.

**2. Algorithm**

Step 1:start program

Step 2: Define four methods: add(), subtract(), multiply(), and divide() to perform operations.

Step 3: Get input from the user as two numbers and an operator (+, -, \*, /).

Step 4: Use a switch or if-else to call the correct method.

Step 5: End the program

**3. Pseudocode**

start

Define add(a, b) → return a + b

Define subtract(a, b) → return a - b

Define multiply(a, b) → return a \* b

Define divide(a, b) → return a / b

Read num1

Read num2

Read operator

IF operator is '+'

Call add(num1, num2)

ELSE IF operator is '-'

Call subtract(num1, num2)

ELSE IF operator is '\*'

Call multiply(num1, num2)

ELSE IF operator is '/'

IF num2 == 0 → print "Division by zero error"

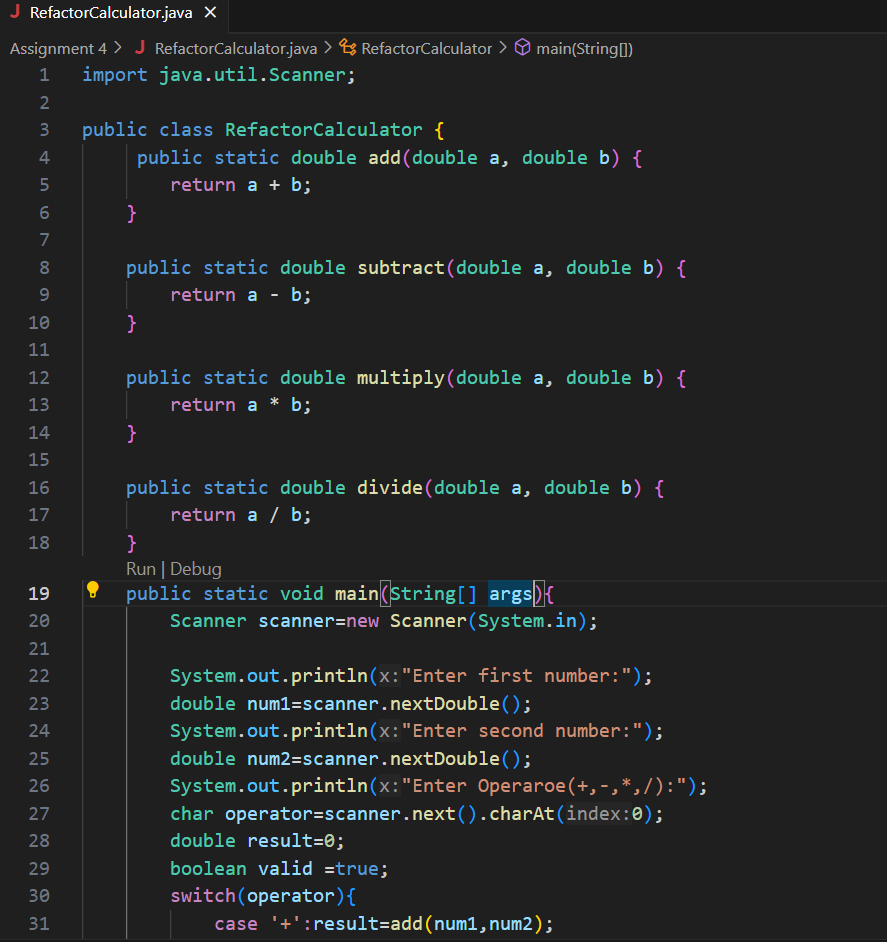
ELSE → Call divide(num1, num2)

ELSE

PRINT "Invalid operator"

END

**4. Program Code**

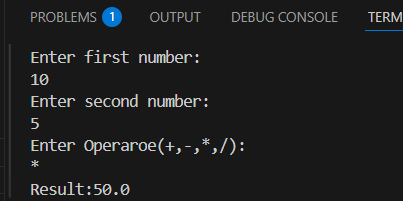
****

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 10,5 | + | 15.0 | Pass |
| 2 | 20,4 | - | 16.0 | Pass |
| 3 | 7,6 | / | 1.1666666 | Pass |

**6. Output**

****

**7. Observation / Reflection**

In this calculator program, we can enter any numbers and use different operators to do calculations. Each operation has its own function, so the code is not repeated. The same functions are used whenever needed, which makes the program neat and efficient.

This makes the program easier to test and fix the errors. It's also easy to read and change in the future. Using separate functions helps keep the code clean and simple.

**Problem Solving Activity 3.1 :Customizable Greeting (Overloading)**

**1. Program Statement:**

Void customGreet(String name,String greeting);

Void customGreet(String name);

Void customGreet();

* Demonstrate calling all variants

By using above methods need to call the different variants by using string name and string greeting in different types of entering both to a method and once need to add name only and after leave it empty.

**2. Algorithm**

Step 1:start program

Step 2: Define the method customGreet(name, greeting) – prints a custom greeting.

Step 3: Define the method customGreet(name) – prints a greeting using a default message.

Step 4: Define the method customGreet() - prints a general greeting. In main() method call all the three versions

Step 5: End the program

**3. Pseudocode**

Start

Function customGreet(name, greeting):

print greeting + ", " + name + "!"

Function customGreet(name):

print "Hello, " + name + "!"

Function customGreet():

print "Hello, Guest!"

In main():

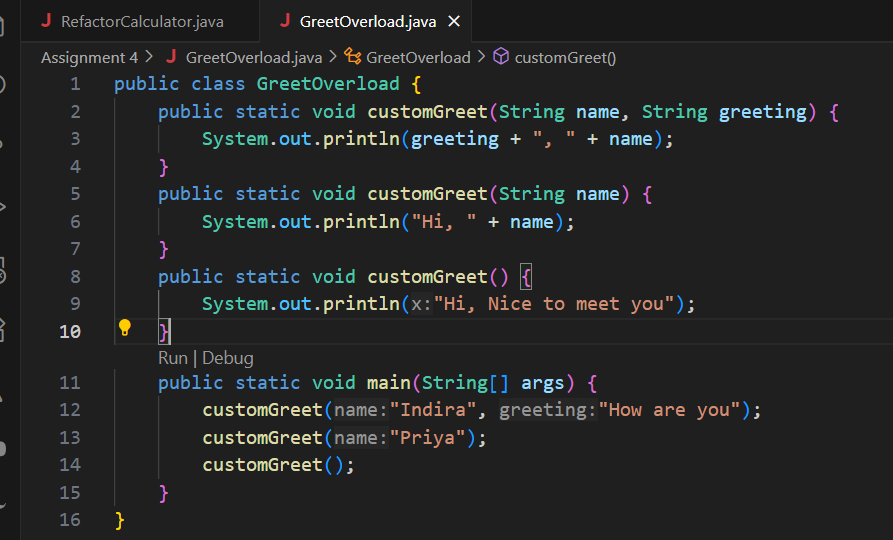
Call customGreet("Indira", "How are you")

Call customGreet("Priya")

Call customGreet()

End

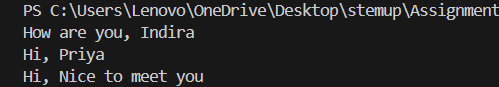
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | How are you,Indira | How are you,Indira | How are you,Indira | Pass |
| 2 | Hi,Priya | Hi,Priya | Hi,Priya | Pass |
| 3 | Hi,Nice to meet you | Hi,Nice to meet you | Hi,Nice to meet you | pass |

**6. Output**

****

**7. Observation / Reflection**

This program shows how **method overloading** allows us to use the same method name with different parameters. It makes the code more flexible and user-friendly by offering multiple ways to greet someone.

**Problem Solving Activity 3.2 :Power Calculator**

**1. Program Statement:**

write myPower(int base,int exponent)using loop and compare with math.pow(base,exponent)

**2. Algorithm**

Step 1:start program

Step 2: Create a method mypower(int base, int exponent) and initialize result=1 and loop from 1 to the exponent

Step 3: Multiply the result base in each iteration

Step 4: In main() set the values for base and exponent and call mypower() and math.pow() for the input and print the results

Step 5: End the program

**3. Pseudocode**

Start

Function mypower(base,exponent):

Result=1

For I from 1 to exponent Do

Result is equal to result \*base

Return result

in main:

base=2

exponent=4

for a variable loopResult can Call mypower(base,exponent)

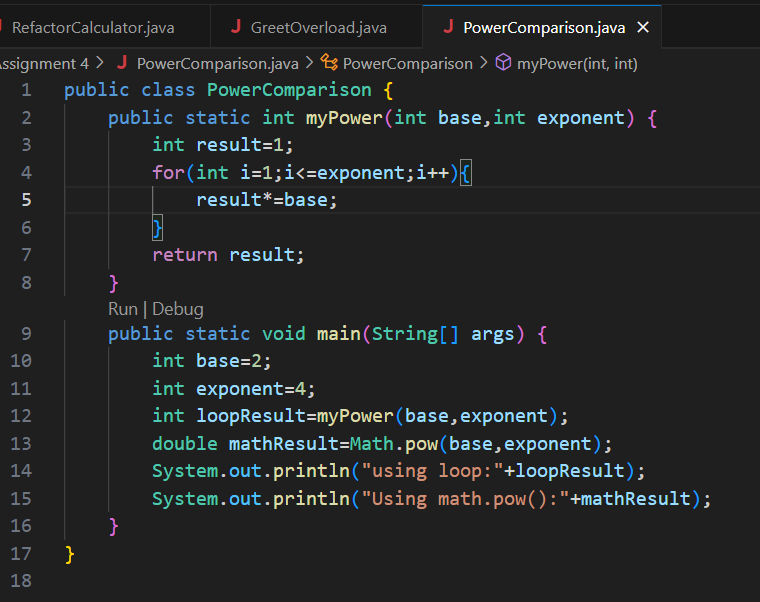
for a variable mathResult can Call Math.pow(base,exponent)

Then print the loopResult

And also print the mathResult

End

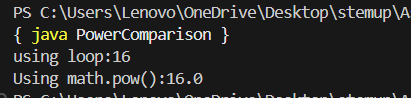
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 2,,4 | 16 | 16.0 | Pass |
| 2 | 3,3 | 27 | 27.0 | Pass |
| 3 | 6,0 | 1 | 1.0 | pass |

**6. Output**

****

**7. Observation / Reflection**

The program can be used for the power to calculate by using the loopby built-in -methods as mypower() function give the result as Math.pow()for positive integers.

**Problem Solving Activity 3.3 :Trace the Flow**

**1. Program Statement:**

Create3 functions A,B,C where:

* A calls B
* B returns a value used in A to call C
* C prints the result
* Manually trace the call order and explain execution

**2. Algorithm**

Step 1: Start the program.

Step 2:Call method A().

Step 3: Inside A(), call B() to get a value.

Step 4: Pass the returned value to C(int value).

Step 5: Method C() prints the value.

Step 6:End the program.

**3. Pseudocode**

start

Method A:

result = call B()

call C(result)

Method B:

return 10

Method C(value):

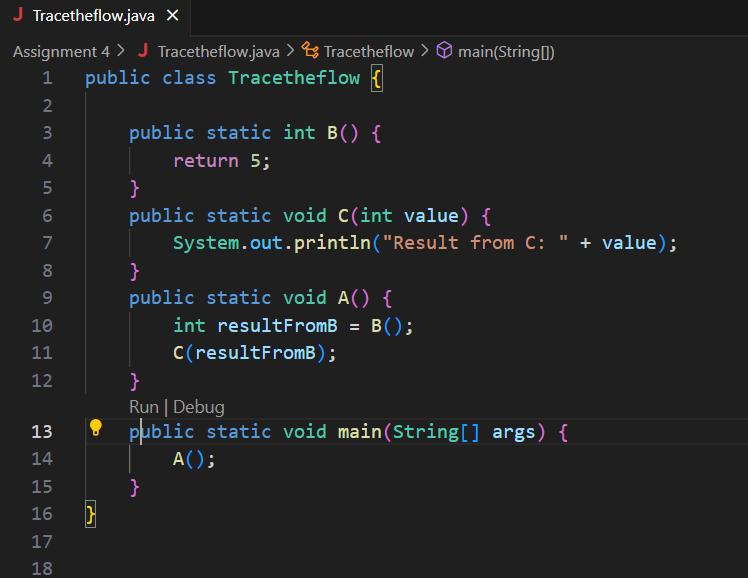
print value

main():

call A()

End

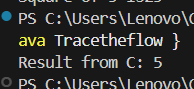
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 5 | Result from C: 5 | Result from C: 5 | pass |
| 2 | 10 | Result from C: 10 | Result from C: 10 | pass |
| 3 | -3 | Result from C: -3 | Result from C: -3 | pass |

**6. Output**

****

**7. Observation / Reflection**

This program demonstrates **function chaining** and **data flow** between methods.  
Tracing the execution order helps in understanding how values are passed and used between functions.  
It’s a clear example of how **modular programming** works and how **control flows** from one method to another.

**Problem Solving Activity 4.1 :Identify Repetition**

1. **Program Statement:**

Write a program that calculate the area of 3 rectangles using repeated code and then, identify the lines that are repeated. Create a function calculate Area(int length,int width)and call it for each rectangle.

**2. Algorithm**

Step 1:start program

Step 2: Declare and initialize length and width for three rectangles.

Step 3: Calculate area using the formula area = length \* width

Step 4: Print the area for each rectangle.

Step 5: End the program

**3. Pseudocode**

Start

FUNCTION calculateArea(length, width):

area ← length × width

PRINT "Area: " + area

MAIN:

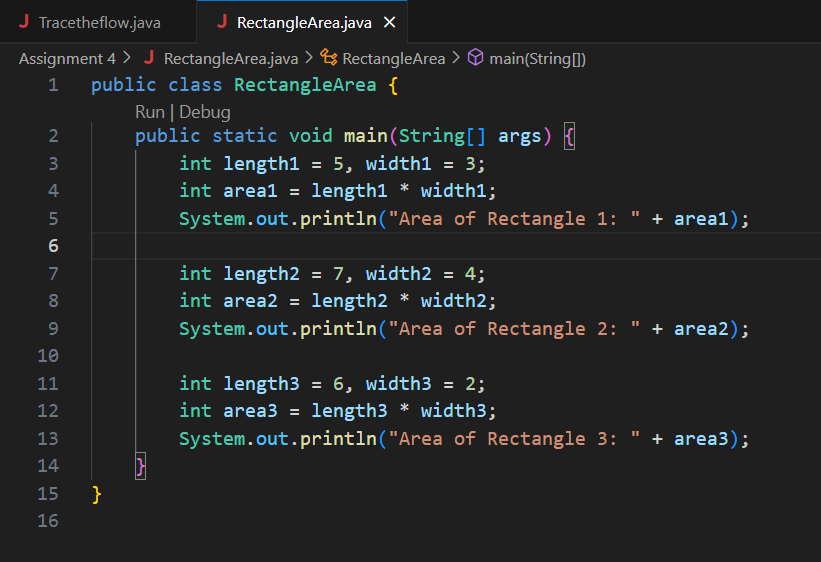
CALL calculateArea(5, 3)

CALL calculateArea(6, 4)

CALL calculateArea(7, 2)

END

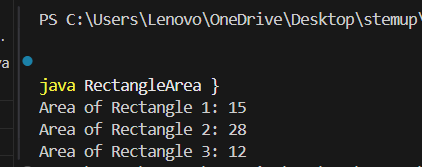
**4. Program Code**

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| 1 | 5,3 | Area of Rectangle 1:15 | Area of Rectangle 1:15 | pass |
| 2 | 7,4 | Area of Rectangle 2:28 | Area of Rectangle 2:15 | pass |
| 3 | 6,2 | Area of Rectangle 3:12 | Area of Rectangle 3:12 | pass |

**6. Output**

****

**7. Observation / Reflection**

Initially, the same lines were repeated for each rectangle, making the code longer.After creating a function, the code became **clean, short, and reusable**.This helps avoid errors and makes future changes easier.

**Problem Solving Activity 4.2 :Program Breakdown**

**1. Program Statement:**

Think of a simple ATM program with tasks like:

* Checking balance
* Depositing money
* Withdrawing money
* Define atleast 3 functions to modularize this program

**2. Algorithm**

Step 1:start program

Step 2: Initialize a balance

Step 3: Display menu options:

* + Check balance
  + Deposit money
  + Withdraw money
  + Exit

Step 4: Use a loop to allow repeated actions until exit.

Step 5: For each choice:

* If **1**, call checkBalance().
* If **2**, call depositMoney() and update the balance.
* If **3**, call withdrawMoney() and update the balance.
* If **4**, exit.

Step 6: End the program

**3. Pseudocode**

SET balance = 1000

FUNCTION checkBalance():

PRINT balance

FUNCTION depositMoney():

ASK user for amount

ADD amount to balance

PRINT updated balance

FUNCTION withdrawMoney():

ASK user for amount

IF amount > balance:

PRINT "Insufficient funds"

ELSE:

SUBTRACT amount from balance

PRINT updated balance

MAIN:

LOOP:

DISPLAY menu

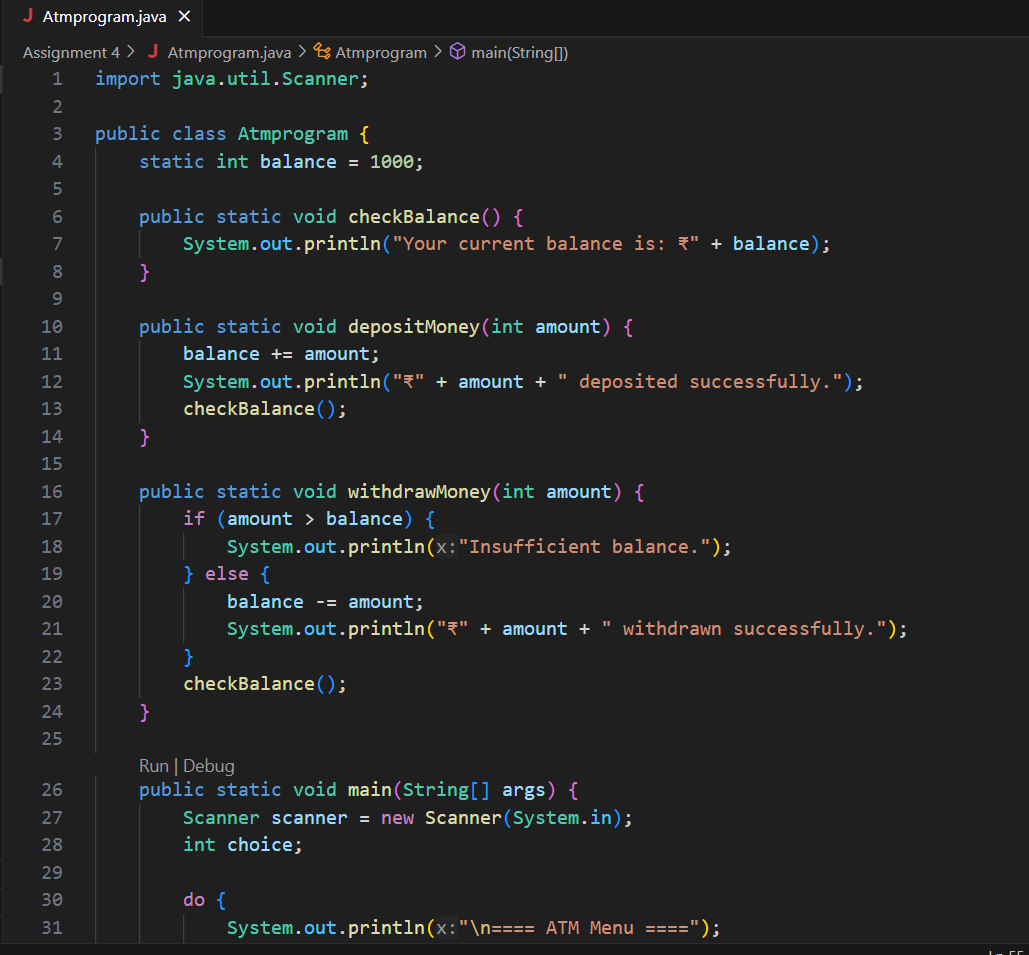
GET user choice

CALL respective function

EXIT if choice is 4

END

**4. Program Code**

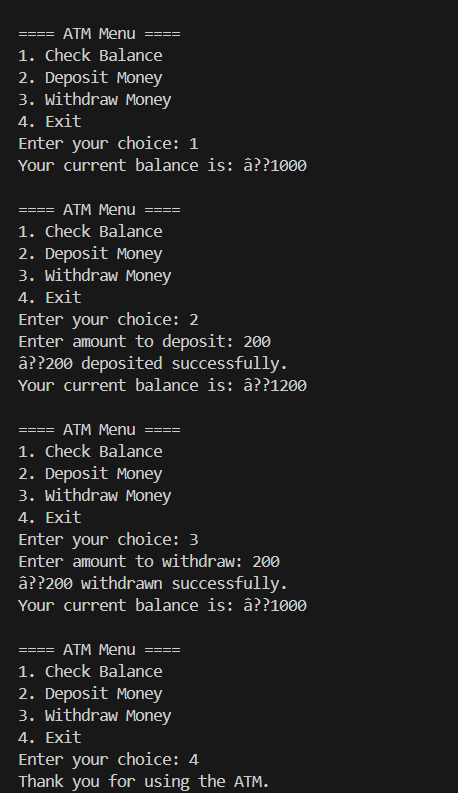
****

****

**5. Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case**  **No.** | **Input** | **Expected**  **Output** | **Actual**  **Output** | **Status (Pass/Fail)** |
| **1** | 1 | Balance = ₹1000 | ₹1000 | Pass |
| **2** | 2 → Deposit ₹200 | Balance updated to ₹1200 | ₹1200 | Pass |
| **3** | 3 → Withdraw ₹200 | Balance updated to ₹1000 | ₹1000 | Pass |

**6. Output**

****

**7. Observation / Reflection**

The ATM program was initially lengthy and a bit confusing. Using functions helped simplify the code, avoid repetition, and make it easier to manage. It also saved time during updates and made the program more reusable and efficient.