

Java Day 1 & 2 Assignment

Task 1: Data Types/Variables

Write a program that declares two integer variables, swaps their values without using a third variable, and prints the result.

Java program that swaps the values of two integer variables without using a third variable:

```
public class SwapWithoutThirdVariable {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = 10;  
        System.out.println("Before swapping: a = " + a + ", b = " + b);  
        a = a + b;  
        b = a - b;  
        a = a - b;  
        System.out.println("After swapping: a = " + a + ", b = " + b);  
    }  
}
```

In this Java program:

- a and b are the two integer variables that we want to swap.
- We print the values of a and b before swapping.
- To swap the values without using a third variable:
 - `a = a + b;` adds a and b and stores the result in a.
 - `b = a - b;` subtracts b from a (the new value of a) and stores the result in b.
 - `a = a - b;` subtracts the new value of b from a and stores the result in a.
- Finally, we print the values of a and b after swapping.
- When you run this Java program, you'll see the output:

Output:

Before swapping: a = 5, b = 10

After swapping: a = 10, b = 5

Task 2: Operators

Create a program that simulates a simple calculator using command-line arguments to perform and print the result of addition, subtraction, multiplication, and division.

Java program that simulates a simple calculator using command-line arguments to perform addition, subtraction, multiplication, and division operations:

```
public class SimpleCalculator {  
    public static void main(String[] args) {  
        if (args.length < 3) {  
            System.out.println("Usage: java SimpleCalculator <number1> <operator>  
<number2>");  
            System.out.println("Supported operators: +, -, *, /");  
            return;  
        }  
        double num1 = Double.parseDouble(args[0]);  
        String operator = args[1];  
        double num2 = Double.parseDouble(args[2]);  
  
        double result = 0.0;  
        switch (operator) {  
            case "+":  
                result = num1 + num2;  
                break;  
            case "-":  
                result = num1 - num2;  
                break;  
            case "*":  
                result = num1 * num2;  
                break;  
            case "/":  
                if (num2 == 0) {
```

```

        System.out.println("Error: Division by zero is not allowed.");
        return;
    }
    result = num1 / num2;
    break;
default:
    System.out.println("Error: Invalid operator. Please use one of +, -, *, /");
    return;
}
System.out.println("Result: " + result);
}
}

```

Here's how the program works:

- It checks if there are exactly three arguments provided (<number1>, <operator>, <number2>). If not, it displays the usage information and exits.
- It parses the first and third arguments (<number1> and <number2>) as double values.
- It checks the operator provided (<operator>) and performs the corresponding arithmetic operation.
- If the operator is division (/), it checks for division by zero.
- Finally, it prints the result of the operation.

OutPut:

\$ java SimpleCalculator 5 + 3

Result: 8.0

\$ java SimpleCalculator 10 - 7

Result: 3.0

```
$ java SimpleCalculator 4 * 6
```

Result: 24.0

```
$ java SimpleCalculator 8 / 2
```

Result: 4.0

Task 3: Control Flow

Write a Java program that reads an integer and prints whether it is a prime number using a for loop and if statements.

```
import java.util.Scanner;
```

```
public class Prime {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter the number:");  
        int n = sc.nextInt();  
        int count = 0;  
        for (int i = 1; i <= n; i++) {  
            if (n % i == 0) {  
                count++;  
            }  
        }  
  
        if (count == 2) {  
            System.out.println(n + " is a prime number");  
        } else {  
            System.out.println(n + " is not a prime number");  
        }  
    }  
}
```

- It initializes a count variable to 0.
- It uses a for loop that runs from 1 to n to check how many times n is divisible by i without a remainder (i.e., $n \% i == 0$).
- Each time n is divisible by i, it increments the count.
- Determining Prime Status:
- After the loop, if count is equal to 2, it means n has exactly two divisors (1 and n itself), so it prints that n is a prime number.
- Otherwise, it prints that n is not a prime number.

Output:

Enter the number:

17

17 is a prime number

Enter the number:

15

15 is not a prime number

Task 4: Constructors

Implement a Matrix class that has a constructor which initializes the dimensions of a matrix and a method to fill the matrix with values.

```
public class Matrix {
    private int rows;
    private int cols;
    private int[][] matrix;

    public Matrix(int rows, int cols) {
        this.rows = rows;
        this.cols = cols;
        this.matrix = new int[rows][cols];
    }
}
```

```

public void fillMatrix(int[][] values) {
    if (values.length != rows || values[0].length != cols) {
        throw new IllegalArgumentException("Matrix dimensions do not match.");
    }
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            matrix[i][j] = values[i][j];
        }
    }
}

public void printMatrix() {
    System.out.println("Matrix:");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    Matrix mat = new Matrix(3, 3);
    int[][] values = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
}

```

```

        mat.fillMatrix(values);
        mat.printMatrix();
    }
}

```

Explanation:

- **Matrix Class:** This class represents a matrix with a specified number of rows and columns (rows and cols).
- **Constructor (Matrix(int rows, int cols)):** Initializes the matrix with the given dimensions.
- **Method (fillMatrix(int[][] values)):** Fills the matrix with the provided values. It checks if the dimensions of the provided values match the dimensions of the matrix.
- **Method (printMatrix()):** Prints the matrix to the console.
- **Main Method:** Includes an example of how to create a Matrix object, fill it with values, and print the matrix.

Example Output:

Matrix:

1 2 3

4 5 6

7 8 9

Task 5: Inheritance

Create a Shape class with a method area() and extend it with Circle and Rectangle classes overriding the area() method appropriately.

```

abstract class Shape {
    // Abstract method to calculate area
    public abstract double area();
}

```

```

// Circle class extending Shape
class Circle extends Shape {
    private double radius;
}

```

```
// Constructor

public Circle(double radius) {

    this.radius = radius;

}


// Override area method to calculate circle area

@Override

public double area() {

    return Math.PI * radius * radius;

}

}


// Rectangle class extending Shape

class Rectangle extends Shape {

    private double length;

    private double width;


// Constructor

public Rectangle(double length, double width) {

    this.length = length;

    this.width = width;

}


// Override area method to calculate rectangle area

@Override

public double area() {

    return length * width;

}

}
```



```
}
```

```
// Main class for testing
```

```
public class ShapeTest {
```

```
    public static void main(String[] args) {
```

```
        // Create a Circle object and calculate its area
```

```
        Circle circle = new Circle(5.0);
```

```
        double circleArea = circle.area();
```

```
        System.out.println("Circle Area: " + circleArea);
```

```
        // Create a Rectangle object and calculate its area
```

```
        Rectangle rectangle = new Rectangle(4.0, 6.0);
```

```
        double rectangleArea = rectangle.area();
```

```
        System.out.println("Rectangle Area: " + rectangleArea);
```

```
    }
```

```
}
```

Explanation:

- **Shape Class:** This is an abstract class with an abstract method `area()`. Every subclass (Circle and Rectangle) must provide an implementation for this method.
- **Circle Class:** Extends Shape and overrides the `area()` method to calculate the area of a circle using the formula
- **Rectangle Class:** Also extends Shape and overrides the `area()` method to calculate the area of a rectangle using the formula $\pi \times r^2$, where r is the radius.
- **Rectangle Class:** Also extends Shape and overrides the `area()` method to calculate the area of a rectangle using the formula **length×width**.
- **ShapeTest Class (Main Class):** Provides an example of how to create objects of Circle and Rectangle, calculate their areas, and print the results.

Output:

Circle Area: 78.53981633974483

Rectangle Area: 24.0

Task 6: Packages/Classpath

Create a package com.math.operations and include classes for various arithmetic operations. Demonstrate how to compile and run these using the classpath.

Step 1: Create the Package and Classes

Create a directory structure to reflect the package structure:

- Create a directory named com in your working directory.
- Inside the com directory, create a directory named math.
- Inside the math directory, create a directory named operations.

```
package com.math.operations;
```

```
public class AddOperation {  
    public static int add(int a, int b) {  
        return a + b;  
    }  
}
```

```
public class SubtractOperation {  
    public static int subtract(int a, int b) {  
        return a - b;  
    }  
}
```

```
public class MultiplyOperation {  
    public static int multiply(int a, int b) {  
        return a * b;  
    }  
}
```

```
public class DivideOperation {  
    public static double divide(double a, double b) {
```

```

    if (b == 0) {
        throw new IllegalArgumentException("Cannot divide by zero");
    }
    return a / b;
}
}

```

Step 2: Compile the Classes

To compile these classes, navigate to the directory containing the **com** directory (which is your working directory) and use the **javac** command with the classpath option (**-cp** or **-classpath**) to include the current directory (.):

```
javac -cp . com/math/operations/*.java
```

This command compiles all Java files inside the com/math/operations directory and its subdirectories.

Step 3: Create a Main Class to Test the Operations

Create a new Java file named Main.java in your working directory (the directory containing com) with the following content:

```

import com.math.operations.*;

public class Main {

    public static void main(String[] args) {

        int a = 10;

        int b = 5;


        int sum = AddOperation.add(a, b);
        System.out.println("Sum of " + a + " and " + b + " is: " + sum);


        int difference = SubtractOperation.subtract(a, b);
        System.out.println("Difference between " + a + " and " + b + " is: " + difference);


        int product = MultiplyOperation.multiply(a, b);
        System.out.println("Product of " + a + " and " + b + " is: " + product);
    }
}

```

```

try {
    double quotient = DivideOperation.divide(a, b);
    System.out.println("Quotient of " + a + " and " + b + " is: " + quotient);
} catch (IllegalArgumentException e) {
    System.out.println(e.getMessage());
}
}
}

```

Step 4: Compile and Run the Main Class

Compile the Main.java class along with the classpath option to include the current directory (.):

```
javac -cp . Main.java
```

Run the Main class with the classpath option to include the current directory (.):

```
java -cp . Main
```

Expected Output:

- Sum of 10 and 5 is: 15
- Difference between 10 and 5 is: 5
- Product of 10 and 5 is: 50
- Quotient of 10 and 5 is: 2.0

Explanation:

- **Package Structure:** The classes **AddOperation**, **SubtractOperation**, **MultiplyOperation**, and **DivideOperation** are placed inside the **com.math.operations** package.
- **Compilation:** We compile all classes in the **com.math.operations** package using the **javac** command with the **classpath (-cp .)**.
- **Main Class:** The Main class demonstrates the usage of these arithmetic operations.
- **Compilation and Execution:** We compile the **Main.java** file and run the **Main** class with the **classpath (-cp .)**.

Task 7: Basic Exception Handling

Write a program that attempts to divide by zero, catches the `ArithmeticException`, and provides a custom error message.

```
public class DivideByZeroExample {  
    public static void main(String[] args) {  
        int dividend = 10;  
        int divisor = 0;  
  
        try {  
            int quotient = divide(dividend, divisor);  
            System.out.println("Quotient: " + quotient);  
        } catch (ArithmeticException e) {  
            System.out.println("Error: Division by zero is not allowed.");  
        } finally {  
            System.out.println("Program execution completed.");  
        }  
    }  
}  
  
public static int divide(int dividend, int divisor) {  
    if (divisor == 0) {  
        throw new ArithmeticException("Attempt to divide by zero");  
    }  
    return dividend / divisor;  
}
```

Explanation:

Main Method:

- It initializes dividend to 10 and divisor to 0.
- It tries to compute the quotient by calling the divide method.
- It catches any **ArithmeticException** that might occur during the division and prints a custom error message.
- It uses a finally block to print a message indicating that the program execution is complete

divide Method:

- This method performs integer division of dividend by divisor.
- If **divisor** is 0, the method throws an **ArithmeticException** with a custom error message

Output:

- If **divisor** is not 0, the program prints the quotient.
- If **divisor** is 0, the program catches the **ArithmeticException** and prints "**Error: Division by zero is not allowed.**"
- The finally block ensures that the "**Program execution completed.**" message is always printed, regardless of whether an exception was thrown

Compile the program using the following command:

- **javac DivideByZeroExample.java**

Run the compiled program using:

java DivideByZeroExample

Expected Output:

Error: Division by zero is not allowed.

Program execution completed.

- The try-catch block in the main method demonstrates basic exception handling in Java.
- The divide method throws an **ArithmeticException** explicitly when attempting to divide by zero.
- The finally block is executed after the try block completes, ensuring that cleanup actions are performed regardless of whether an exception occurred.