W1+2+3

Java Exercises: Data Types, Input/Output, Increment/Decrement Operators

Grading criteria:

1. **Correctness (70%):**
    - The program compiles and runs without errors.
    - The program produces the correct output for given inputs.
    - All specified functionalities are implemented correctly.

2. **Commenting the Logic (30%):**
    - The code includes comments explaining the purpose of major code sections.
    - The comments describe the logic behind key operations and decisions.
    - The comments help readers understand how the code works.

Exercise 1: Simple Calculator
Objective: Practice using arithmetic operators, input/output, and basic data types.

1. Write a Java program that prompts the user to enter two integers.
2. Perform the addition, subtraction, multiplication, and division of these two numbers.
3. Display the results to the user.

Expected Input:
```
Enter the first integer: 10
Enter the second integer: 5
```

Expected Output:
```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
```

Exercise 2: Temperature Converter
Objective: Practice using arithmetic operators, input/output, and data types.

1. Write a Java program that prompts the user to enter a temperature in Celsius.
2. Convert the temperature to Fahrenheit using the formula: F = (C * 9/5) + 32.
3. Display the temperature in Fahrenheit.

Expected Input:
```
Enter temperature in Celsius: 25
```

Expected Output:
```
Temperature in Fahrenheit: 77.0
```

Exercise 3: BMI Calculator
Objective: Practice using arithmetic operators, input/output, and data types.

1. Write a Java program that prompts the user to enter their weight (in kilograms) and height (in meters).
2. Calculate the Body Mass Index (BMI) using the formula: BMI = weight / (height * height).
3. Display the BMI to the user.

Expected Input:
```
Enter your weight in kilograms: 70
Enter your height in meters: 1.75
```

Expected Output:
```
Your BMI is: 22.857142857142858
```

Exercise 4: Counter Program
Objective: Practice using increment and decrement operators, input/output, and basic data types.

1. Write a Java program that prompts the user to enter an integer. The program should then increment and decrement the integer, displaying the results.

Requirements:
1. Prompt the user to enter an integer.
2. Increment the integer using the ++ operator and display the result.
3. Decrement the integer using the -- operator and display the result.

Expected Input:
```
Enter an integer: 10
```

Expected Output:
```
After incrementing: 11
After decrementing: 10
```

Exercise 5: Leap Year Checker
Objective: Practice using logic conditions and input/output.

1. Write a Java program that prompts the user to enter a year.
2. Check if the year is a leap year using the following conditions:
3. A year is a leap year if it is divisible by 4, except for years divisible by 100, but these centurial years are leap years if they are divisible by 400.
4. Display whether the year is a leap year or not.

Expected Input:

```
Enter a year: 2024
```

Expected Output:

```
2024 is a leap year.
```

W4 + 5

Java Exercises: Switch, For Loop

Grading criteria:

1. **Correctness (70%):**
    - The program compiles and runs without errors.
    - The program produces the correct output for given inputs.
    - All specified functionalities are implemented correctly.

2. **Commenting the Logic (30%):**
    - The code includes comments explaining the purpose of major code sections.
    - The comments describe the logic behind key operations and decisions.
    - The comments help readers understand how the code works.

Exercise 1: FizzBuzz Variation with Switch and For Loop

Write a program that prints numbers from 1 to 100. However, for multiples of 3, print **"Fizz"**, for multiples of 5, print **"Buzz"**, and for numbers that are multiples of both 3 and 5, print **"FizzBuzz"**. Use a **for** loop and a **switch** statement to handle this.

Expected Output:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
…
```

Exercise 2: Grade Calculation Using Switch

Write a program that calculates the grade based on the percentage entered by the user. The program should categorize the grade using a **switch** statement. The grades should be categorized as follows:

- **A** for 90-100%
- **B** for 80-89%
- **C** for 70-79%
- **D** for 60-69%
- **F** for 0-59%

Expected Input:

```
85
```

Expected Output:

```
Your grade is B
```

Exercise 3: Guess the Number Game

Write a program that randomly generates a number between 1 and 10. The user needs to guess the generated number. After each guess, the program will provide feedback, informing the user whether their guess is too high, too low, or correct. The game will continue until the user correctly guesses the number. Once the correct number is guessed, the program will display the total number of attempts it took the user to guess correctly.

Expected Input & Output:

```
Welcome to the Number Guessing Game!
I have selected a number between 1 and 10. Try to guess it!

Enter your guess: 5
Too low! Try again.

Enter your guess: 8
Too high! Try again.

Enter your guess: 6
Correct! The number was 6.
It took you 3 attempts.
```

Challenge Extension:

- **Difficulty Levels**: Add difficulty levels (e.g., Easy: Range 1-50, Medium: Range 1-100, Hard: Range 1-1000).

Expected Input & Output:

```
Welcome to the Guess the Number Game!

Choose a difficulty level:

1. Easy (1-50)

2. Medium (1-100)

3. Hard (1-1000)

Enter your choice (1/2/3): 2

I have selected a number between 1 and 100. Try to guess it!

Enter your guess: 45

Too low! Try again.

Enter your guess: 75

Too high! Try again.
```

```
Enter your guess: 60
Correct! The number was 60.
It took you 3 attempts.
```

W6 + 7

Exercise: Library Management System with Access Modifiers

Objective: Create a simple library management system using OOP principles with arrays and access modifiers.

Requirements:

1. **Classes and Objects:**

- Create a Book class with the following attributes:
    - title (private String)
    - author (private String)
    - isbn (private String)
    - isAvailable (private boolean)
- Create a Library class with the following attributes:
    - books (private Book[])
    - bookCount (private int)

2. **Methods:**

- In the Book class, add methods to:
    - Display book details (public).
    - Check availability (public).
    - Getters for title, author, isbn, and isAvailable (public).
- In the Library class, add methods to:
    - Add a new book (public).
    - Remove a book by ISBN (public).
    - Search for a book by title or author (public).
    - Display all books (public).
    - Borrow a book (public).
    - Return a book (public).

3. **Main Method:**

- Create a Main class with a main method to:
    - Instantiate the Library class.
    - Add some books to the library.
    - Demonstrate borrowing and returning books.
    - Display the list of all books in the library.

W8

Banking System: Using Abstract Classes, Interfaces, Overloading, Overriding, and Inheritance

Problem Statement:

Enhance the banking system by incorporating **abstract classes** and **interfaces**.

- Use an **abstract class** BankAccount as a blueprint for different types of accounts.
- Use an **interface** Transaction to define common operations like deposit and withdraw.
- Implement **overloading** for deposits and **overriding** for withdrawals.

Requirements:

1. **Create an abstract class** BankAccount that includes:
    - Common attributes: accountHolder and balance.
    - An **abstract method** withdraw(double amount).
    - A **concrete method** displayBalance() to show the balance.
2. **Create an interface** Transaction that defines:
    - A deposit(double amount) method.
    - A withdraw(double amount) method (to be overridden in subclasses).
3. **Create subclasses** SavingsAccount **and** CurrentAccount that:
    - Extend BankAccount (inheritance).
    - Implement Transaction (interface).
    - Override the withdraw() method with custom rules.
4. **Use Overloading:**
    - deposit(double amount): Deposits a given amount.
    - deposit(double amount, String currency): Deposits with currency conversion (simplified).

W12

Exercise: Utilizing Selenium for writing automation test cases.

Grading criteria:

1. **Correctness (70%):**
    - The program compiles and runs without errors.
    - The program produces the correct output for given inputs.
    - All specified functionalities are implemented correctly.

2. **Commenting the Logic (30%):**
    - The code includes comments explaining the purpose of major code sections.
    - The comments describe the logic behind key operations and decisions.
    - The comments help readers understand how the code works.

Exercise: Write at least 2 test cases for each of the below scenarios

1. Navigate to **saucedemo.com**.
2. Create test cases for the login page.
3. Develop test cases to list all products on the inventory page.
4. Formulate test cases to verify the checkout process.

W13

Exercise: Utilizing Selenium for writing automation test cases.

Grading criteria:

1. **Correctness (70%):**
    - The program compiles and runs without errors.
    - The program produces the correct output for given inputs.
    - All specified functionalities are implemented correctly.

2. **Commenting the Logic (30%):**
    - The code includes comments explaining the purpose of major code sections.
    - The comments describe the logic behind key operations and decisions.
    - The comments help readers understand how the code works.

Exercise: Write at least 2 test cases for each of the below scenarios in the demo website: https://the-internet.herokuapp.com/

1. Dynamic Controls
2. Dynamic Loading
3. Dropdown
4. Disappearing Elements
5. Javascript Alerts
6. Add/Remove Elements

Tab 7

Exercise: Utilizing Selenium for writing automation test cases.

Grading criteria:

1. **Correctness (70%):**
    - The program compiles and runs without errors.
    - The program produces the correct output for given inputs.
    - All specified functionalities are implemented correctly.

2. **Commenting the Logic (30%):**
    - The code includes comments explaining the purpose of major code sections.
    - The comments describe the logic behind key operations and decisions.
    - The comments help readers understand how the code works.

Exercise: Design the codebase for each page in saucedemo.com following the POM (Page Object Model) architecture.

- Refactor the existing test cases for saucedemo.com by implementing the POM (Page Object Model) architecture to create streamlined, modular, and reusable test cases.

Tab 8