Week 5 Assignment

1. In the mystical land of Stringia, you have been chosen as the ***Alphabet Guardian***. Your task is to maintain a running total of all alphabetic characters passed to you, ensuring that only valid alphabetic characters enter the sacred vault. If a non-alphabetic character attempts to breach the vault, you must raise an alarm by throwing an exception called ***NonAlphabeticCharacterException***.

   Your mission:

   Write a class AlphabetGuardian that keeps a running total of all alphabetic characters passed to it.

   If a non-alphabetic character is passed, throw a custom exception NonAlphabeticCharacterException.

   Provide a method addCharacter(char c) that adds a valid character to the total or throws the exception if it's invalid.

   Instructions:

- Define the custom exception ***NonAlphabeticCharacterException.***
- Implement the ***AlphabetGuardian*** class with a method addCharacter(char c) that checks if the character is alphabetic.
- In case of a non-alphabetic character, throw the custom exception.
- Provide a method to retrieve the total number of valid characters stored so far.

   Output Format:

- If all characters are valid, print the running total.
- If an invalid character is passed, print a custom error message.

2. As the guardian of numbers, you have been tasked with calculating the factorial of any given number, storing the results for future reuse. However, there's a catch: the power of the factorial is so immense that only numbers within a certain range can be computed safely. Numbers too large will overflow the array, and negative numbers are forbidden.

   Your mission:

- Write a program called Factorial.java that computes the factorial of a given number x and stores the result in an array of type long.
- If x is less than 0, throw an ***IllegalArgumentException*** with the message "Value of x must be positive".
- If x exceeds the array limit (factorial of numbers greater than 20 will overflow a long), throw an ***IllegalArgumentException*** with the message "Result will overflow".

Instructions:

- Precompute and store factorials for numbers from 0 to 20 in an array of type long for fast lookup.
- Ensure proper exception handling for invalid values of x.

Output Format:

- Print the factorial of x if valid, or print the exception message if the input is invalid.

3. In the land of Codeopolis, you are assigned the role of the **Character Collector**, responsible for gathering only alphabetic characters into the sacred vault. Your challenge is to maintain a running total of all valid characters passed to you. However, if a non-alphabetic character attempts to enter the vault, you must immediately raise an alarm by throwing a custom exception called ***InvalidCharacterException***.

   Your mission:

   - Write a class CharacterCollector that stores a running total of alphabetic characters passed to it, one at a time.
   - If a non-alphabetic character is passed, throw a custom exception ***InvalidCharacterException***.
   - Provide a method addCharacter(char c) to add valid characters or throw the exception if invalid.

   Instructions:

   - Define a custom exception ***InvalidCharacterException.***
   - Implement the CharacterCollector class with a method addCharacter(char c) to process characters.
   - Keep a count of the total number of valid characters and display the total when asked.

   Output Format:

   - Print the running total of valid characters.
   - Print an error message if an invalid character is passed

4. In the Kingdom of Geografia, knowing the capital of a country is a prized skill. You are tasked with building a program that takes the name of a country from the command line and returns its capital. However, if the country is unknown, an ancient exception called ***NoMatchFoundException*** must be thrown.

   Your mission:

- Write a program that takes the name of a country as input from the command line.
- The program should print the capital of the country if it matches a known country.
- If the country is unknown, throw a custom exception ***NoMatchFoundException*** with a message indicating the country was not found.

Instructions:

- Define the custom exception ***NoMatchFoundException.***
- Store a predefined list of countries and their capitals.
- Write a method getCapital(String country) that checks the list for the country's capital.
- If the country is not found, throw the ***NoMatchFoundException.***

Output Format:

- If the country is known, print its capital.
- If the country is unknown, print exception message.

5. In the realm of number crunchers, calculating factorials is a sacred task. You've been assigned to write a program that computes the factorial of a number provided at the command line. However, three potential pitfalls could derail your mission: A. The user may forget to provide a number, resulting in an ***ArrayIndexOutOfBoundsException***.
   B. The user might input a non-integer value (like a float or double), causing a ***NumberFormatException***.C. If the user inputs 0, the program should manually throw an ***IllegalArgumentException*** since factorial for 0 is not allowed in this quest.

   Your mission:

   - Write a program that computes the factorial of a number passed via the command line.
   - Handle three potential user input errors:
     No input provided (*ArrayIndexOutOfBoundsException*).
     Non-integer input (*NumberFormatException*).
     If the input is 0, manually throw an *IllegalArgumentException*.

   Instructions:

   - Implement appropriate try-catch blocks to handle each error scenario.
   - If the input is valid, print the factorial of the number.

   Output Format:

   - If valid input is given, print the factorial of the number.
   - If an error occurs, print a meaningful error message for each case.

6. You are tasked with a special mission as the **Argument Guardian**. You must ensure that exactly five numbers are passed through the command line to complete the magical summation. If fewer than five arguments are provided, you must raise the custom exception **CheckArgumentException** and halt the mission.

   Your mission:

   - Write a program that checks if exactly five numbers are passed via the command line.
   - If fewer than five arguments are provided, throw a custom exception ***CheckArgumentException*** with a message indicating the required number of arguments.
   - If five arguments are passed, print the sum of the five numbers.

   Instructions:

   - Define the custom exception ***CheckArgumentException***.
   - Parse the arguments as integers and compute their sum.
   - Throw the exception if the number of arguments is less than 5.

   Output Format:

   - If fewer than five arguments are passed, print the exception message.
   - If five arguments are passed, print their sum.

7. In the **Student Examination Database System**, you must ensure that the mark sheets are correctly generated for each student. Your task is to validate the marks input through the command line. Each student's marks must fall within the range of 0 to 50 for each of the six subjects. If any mark is out of this range, raise a custom exception ***RangeException***. If all marks are valid, compute the total marks and print the percentage.

   Your mission:

   - Write a program that accepts a student's name and marks in six subjects from the command line.
   - Validate that each mark is between 0 and 50. If any mark is out of range, throw a custom exception *RangeException* with an appropriate error message.
   - If all marks are valid, calculate and print the total marks and percentage of the student.

   Instructions:

- Define the custom exception *RangeException*.
- Check each mark to ensure it is within the valid range.
- Compute the total and percentage if all marks are valid, and print them.

Output Format:

- Print an error message if any mark is out of range.
- Print the total marks and percentage if all marks are valid.

8. In the realm of **MathLandia**, you are entrusted with calculating complex equations. Your task is to design a robust error handler that uses a custom exception to manage at least two kinds of arithmetic errors that may arise during the computation of an equation like $X+Y\times (P/Q)^Z -I$.

   Your mission:

   - Write a Java program that computes the result of a given equation.
   - Create a custom exception called *ArithmeticOperationException* to handle:

     Division by Zero – If any division operation in the equation results in division by zero.

     Overflow – If the computation results in a value that exceeds the limits of data types.
   - Catch these exceptions and handle them appropriately while performing the calculation.

   Instructions:

   - Define the custom exception ArithmeticOperationException.
   - Implement the equation calculation, handling division by zero and overflow errors.
   - Ensure the program prints meaningful error messages when exceptions occur.

   Output Format:

   - Print the result of the equation if no exceptions occur.
   - Print an error message if an arithmetic exception is caught.

9. The Student Score Validator - Ensuring Fairness in Grades: As part of a university grading system, you need to implement a program that tracks student scores. However, any invalid scores must be caught to ensure fairness in the grading process.

   Problem Statement:

Write a program that stores **student IDs** in an array and asks the user to enter a **test score** for each student. If the user enters an invalid score (not between 0 to 100), throw a custom exception called `ScoreException`.

Constraints:

- Catch invalid scores and replace them with a **0**.
- Display all student IDs and their scores at the end of the program.
- Handle the invalid input gracefully with the custom exception.

Can you build a system that ensures every student's score is entered correctly?

10. Number Validator - The Positive Number Challenge: You've been tasked with creating a number validation system for a scientific project. All the numbers passed to this system must be positive to ensure calculations proceed correctly.

Problem Statement:

Write a Java program that accepts a number. If the number is **not positive**, throw a custom exception called `NonPositiveNumberException`.

Constraints:

- Numbers must be greater than **0**.
- Handle edge cases, including **0** and negative values, by throwing the exception.
- Print **"Valid Number"** if the input passes the validation.

11. Security Breach - The Username Dilemma: In a top-tier tech firm, your task is to create a secure login system. The system has stringent rules for usernames and passwords, ensuring no unauthorized access.

Problem Statement:

Create a program that validates usernames and passwords. If the **username** is less than **6 characters** or the **password** does not match a predefined value, throw a custom exception `InvalidCredentialsException`.

Constraints:

- The password should be case-sensitive and match exactly to be considered valid.
- You must create a custom exception called InvalidCredentialsException.
- If both username and password meet the criteria, print **"Login Successful."**

Can you design a login system using java programming that prevents unauthorized access and ensures maximum security?

12. Authentication Challenge - The Password Protector: Your mission is to build the most secure password authentication system for an upcoming banking app. Hackers are constantly trying to guess passwords, so the system must catch any failure and alert the user.

    Problem Statement:

    Write a Java program that accepts a **password** from the user. If the password is incorrect, throw a custom exception called `AuthenticationFailureException`. If the password is correct, print **"Access Granted."**

    Constraints:

    - Password validation should be **case-sensitive**.
    - Create a custom exception called AuthenticationFailureException.
    - Handle empty or null password inputs and throw the same exception for such cases.

    Can your java program stand strong against attempts of password breaches?