JAVA PROGRAMMING

Tutorial 09

Getting Started

Before delving further into Java exceptions, it's essential to complete crucial preparatory tasks:

- Begin by establishing a package named "**tut09.utils**" to consolidate all necessary utilities and libraries for this week's tutorial. Within this package, develop a class called "**TextIO**" tailored to facilitate user input from the keyboard. This class employs exception handling to meticulously validate user input, ensuring that only after successful validation do methods return the intended values. The "**TextIO**" class should offer versatile methods to retrieve **integers**, **floats**, **words**, **emails**, and **other data types** in their *correct formats*.
- Furthermore, within the "**tut09.utils**" package, create a subpackage titled "**exceptions**" dedicated to housing ***custom exceptions***. Within this subpackage, implement an exception class named "**NotPossibleException**," designed to extend the **Java Runtime Exception**.

To enhance clarity and effectiveness, students should:

- Outline a detailed table of **domain constraints** that are pertinent to the attributes of the class in each exercise.
- Utilize the comprehensive **domain constraint table** as a foundational reference point for crafting **UML class diagrams**. Ensure that each constraint is accurately represented within the corresponding class diagram, maintaining consistency and alignment with the requirements and specifications.

Activity 01: Count Letters

In this exercise, students will create a Java program to analyze the **occurrence** of **each letter** in **a word** provided by the user. The program will adhere to the following requirements:

- **Input Validation**
  - Users must enter a single word without any spaces or punctuation.
  - The word should consist of only letters, including both uppercase and lowercase characters.
  - If the input violates these rules, the program should throw a custom exception, such as **InvalidInputException**, with an appropriate error message.
- **Word Length Restriction**
  - The word entered by the user should ***not be empty***, ***null***, or exceed ***a length of 45*** characters.
  - If the word fails to meet these criteria, the program should throw a custom exception, such as **WordLengthException**, with a descriptive error message.
- **Exception Handling**
  - Utilize Java's exception handling mechanisms to catch and handle exceptions gracefully.
  - Define custom exception classes like **InvalidInputException** and **WordLengthException** to encapsulate specific error conditions and provide meaningful feedback to the user.
- **Implementation**
  - Design a modular and well-structured program with clear separation of concerns.
  - Encapsulate the logic for reading user input, validating input constraints, and calculating letter occurrences into separate methods or classes for better maintainability and readability.
- **Output**
  - Display the occurrence of each letter in the word to the user in a clear and organized manner.

Activity 02: Integers Sum

In this exercise, students will develop a Java program to compute the **sum of a sequence of integers** entered by the user, **terminated by the number 0**. Here are the detailed requirements:

- **Input Validation:** Users must input a sequence of integers separated by single spaces, including both positive and negative numbers.
- **Exception Handling:** The program should handle exceptions for invalid input, such as non-numeric characters or multiple spaces between integers, using relevant exception classes like **NumberFormatException**.
- **Empty Line and Zero Check:** The input line must not be empty or contain only the number 0. Exceptions should be thrown for these cases to ensure valid input.
- **Calculation:** The program should calculate the sum of the integers in the sequence, excluding the terminating 0.
- **Output:** Display the computed sum to the user.

Students are encouraged to implement custom exception classes, such as **EmptyLineException**, for better code organization and error handling.

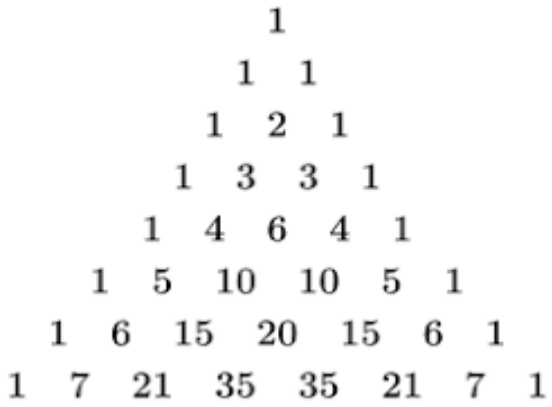
Activity 03: Pyramid

In this exercise, students will design and implement a Java program to print **various types of number pyramids** based on the **number of rows**. The program will utilize Java exceptions and implemented classes to ensure robust error handling and efficient execution. Here's a detailed description of the exercise:

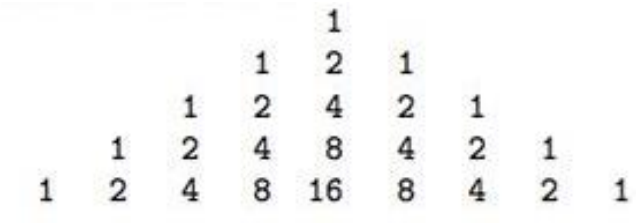
- **Input Validation:** Users will input an integer representing the number of rows for the pyramids. The program will throw relevant exceptions such as **NumberFormatException** if the input is not a valid integer or if it's negative.
- **Pyramid Printing Methods:**
  - Half-Right Pyramid
  - Half-Left Pyramid
  - Full Pyramid
  - Inverted Half-Right Pyramid
  - Inverted Half-Left Pyramid
  - Inverted Full Pyramid
  - Special Pyramids like Pascal's Pyramid, Floyd's Triangle, Pyramid with multiples of 2, etc.
- **Exception Handling:**

- Custom exception classes will handle invalid input and ensure proper error messaging to the user.

❖ **Output Formatting:** The pyramids will be printed using a specific format to ensure clarity and visual appeal.



Pascal Pyramid



Pyramid with multiple of 2

Activity 04: String Processing

In this exercise, students will design and implement a Java program that **processes a string** input from the user. The program will utilize Java exceptions and implemented classes to meet the following requirements:

❖ **Input Validation:** Implement a method to obtain user input. The string must not be empty, null, or contain only spaces. It should throw a custom exception for these errors.

❖ **String Manipulation Methods:**
- Reverse String
- Count Vowels and Consonants
- Display Characters at Odd Positions
- Display Characters at Even Positions
- Count Uppercase and Lowercase Letters
- Calculate Occurrence of Each Character
- Extract a Substring from a Specific Position with a Length
- Find Index of First Occurrence of a Substring
- Find Index of Last Occurrence of a Substring
- Divide String at a Specified Separator

❖ **Custom Exception Classes:** Introduce custom exception classes to handle relevant errors.

❖ **No Built-in Methods or Libraries:** Prohibit the use of built-in methods or libraries for string manipulation.

Submission

Submit a zip file containing all Java programs to this tutorial's submission box in the course website on FIT Portal.