**AC12001**

Name: …Miklos Mayer………………………………………………………………………..

Matric number: …190017778 …………………………………………………….……….

Lab Title: …AC12001 assignment 2: Sets………………………………..……….

Test number/date/version: 03/02/20 ……………………………………….……..

Test Notes: …Tests run on submitted assignment ……………………………

Summary:

A program is needed to calculate Reverse Polish Notation expressions.

Requirements:

* The program shall be able to solve Reverse Polish Notation expressions with the 4 basic binary operator (+, -, \*, /).
* The calculator shall use a Stack data structure for calculating the result
* The program shall include a Menu options for the following tasks:
  + Show help
  + Enter and calculate a Reverse Polish Notation Expression
  + Exit
* The program shall handle all the errors and exceptions which could occur during the calculation and inform the user about them.
* The program should transform the Reverse Polish Notation into Infix Notation

Class designs:

* RPN\_Calculator:
  + Handle user inputs
  + Calculate expressions
  + Informs the user about errors
  + Fields: none
  + Methods:
    - static main(String[]): void – Entry point of the program, operate the calculator and the user inputs.
    - private static printMenu(Scanner): String – Print the menu of the game to the user and return their commands.
    - calculate(String): double – Calculate the result of the given expression.
    - private operate(String, Stack): double – Executes one operation between two numbers and handle Exceptions. Returns the result of the operation
* ListNode:
  + A node in a Linked List data structure for double numbers
  + Fields:
    - number: double – The stored number in the node
    - next: ListNode – The next node in the list
  + Methods:
    - getNext(): ListNode
    - setNext(ListNode): void
    - getNumber(): double
    - setNumber(double): void
* LinkedList:
  + A Linked List data structure implementation for double numbers
  + Fields: head: The first node in the Linked List
  + Methods:
    - isListEmpty(): boolean
    - getHead(): ListNode
    - addToList(double): void – Add a new number to the beginning of the list
    - deleteFromStart(): void – Delete the first number from the list
    - findInList(double): ListNode – Find a number in the list. Return null if it does not contain the number.
* Stack:
  + A Stack data structure implementation.
  + Fields: stack: LinkedList – The stack uses a LinkedList as its core data structure
  + Methods:
    - push(double): void – Add a new number to the top of the stack
    - pop(): double – Return and delete the number from the top of the stack
    - isEmpty(): boolean
* EmptyStackExpression: Thrown when you try to pop an empty stack
* InvalidRPNExpressionException: Thrown when an invalid expression is provided to the RPN calculator.

Pseudocodes:

The calculator:

1. Split the expression into numbers and operators
2. Push numbers into the stack
3. When an operator reached execute it on the last two numbers from the stack
4. Push the result back into the stack
5. Continue from step 2. until the expression run out of numbers and/or operators
6. Return the end-result

Possible errors:

* The expression is ending with a number
* Too much number in the expression, there’s no operators left to operate with them during the calculation
* Too few numbers in the expression, there are unused operators left during the calculation
* Invalid operator provided by the user
* A division by zero occurs during the calculation
* The provided expression is empty
* A space is missing between a number and an operator

Test results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Description** | **Test Data** | **Expected result** | **Worked?** |
| Create a new ticket with 6 different numbers | 1, 2, 3, 4, 5, 6 | The new Ticket is created |  |
| Create new ticket with 5 different numbers | 1, 2, 3, 4, 5 | Exception’s thrown |  |
| Create a new ticket with 7 different numbers | 1, 2, 3, 4, 5, 6, 7 | Exception’s thrown |  |
| Create a new ticket with numbers outside the lottery range | -1, 1, 2, 3, 4, 5, lotteryMax + 1, 6 | The System will ask for numbers until a valid number entered |  |
| Try to create a ticket with a duplicate number | 1, 1, 2, 3, 4, 5, 6 | The system will ask for numbers until it gets 6 different numbers |  |
| Draw the lottery a hundred times | None | The System creates a random ticket each time |  |
| Run the lottery 10 times | 1, 2, 3, 4, 5, 6 | The system should display in the end how much the user has spent and won. |  |
| Run the lottery 100 times | 1, 2, 3, 4, 5, 6 | The system should display in the end how much the users have spent and won. |  |
| Run the lottery for 3 users 10 times | 1, 2, 3, 4, 5, 6 | The system should display in the end how much the users have spent and won. |  |
| Run the lottery for 3 users 100 times | 1, 2, 3, 4, 5, 6 | The system should display in the end how much the users have spent and won. |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Self-evaluation:

I enjoyed this assignment. I am pretty familiar with Linked Lists and Stacks, so my challenge for the assignment was to run the test in Junit5. I really enjoyed figuring out all the edge-cases and write a bunch of tests for them.