**OVERVIEW**

This calculator program will prompt for user input in form of two doubles separated by an operator and will return the sum of the most recent calculation along with a running total of all operations. Operators accepted are addition, subtraction, multiplication and division. The calculator will handle functions such as exit, undo, and clear. The undo function will remove the last state from the running total, clear will remove all previous operations and set the running total to zero, and exit will end the program.

**SCOPE**

This program will accept user input in form of real-literal, supported operator, real-literal.

The program will accept functions such as undo, exit, clear and handle the running total accordingly. The program will output in the console the sum of the last operation, and the running total of all operations.

The program supports three languages based on computers locale setting English, Russian, and Chinese.

**PROCESSING**

This process requires user input each line and will prompt the user after every operation or function except for exit which ends the program.

The application will obtain the users locale setting and set language variables for the program to use. The process of this application will accept user input, check if the user input is correct in the format desired and begin performing the functions required. The calculator functions required are undo, exit, clear, and operations. The undo function will delete the last state item from the Memento stack data type and update the running total. The exit function will end the program. The clear function will delete all Memento stack elements and set running total to zero. Operations in the form of real – literal operator real – literal where operator is addition, subtraction, multiplication, and division will be calculated and return the sum value of the most recent operation, and the running total of all previous operations.

After each operation the program will prompt for user input until exit function is called. After each operation the program will return a sum and running total.

**Memento Stack Example**

|  |
| --- |
| newMemento – E total |
| newMemento – E total |
| newMemento – E total |
| newMemento – E total |
| newMemento – E total |
| newMemento – E total |

Processing Outline Main

1. Initialize data structures and variables

a. Double a

b. Double b

2. Main Method

1. Call calculator class instance
2. Call calculator setVariables method

c. Print initial user input prompt

d. Initialize scanner

e. While scanner scan has next line

1. user input equals scanner line
2. regex pattern
3. matcher compares regex and user input
4. If user input contains “clear”

i. Call calculator clear message

g. Else if user input contains “undo”

1. Call calculator undo method

h.Else if user input contains “exit”

1. Call calculator exit method

I.Else

1. If matcher is true and first is false
2. Double a set to matcher group
3. First set to true

ii. If matcher is true and first is true

1. Double b set to matcher group

iii. Else

1. Print error message
2. Set double a to 0.0
3. Set double b to 0.0

j. If user input contains “+”

1. Call calculator add method

k. Else if user input contains “- “

i. Call calculator subtract method

l. Else if user input contains “\*”

1. Call calculator multiply method

m. Else if user input contains “/”

1. Call calculator divide method

n. Else

1. Print error message

o. Prompt for user input

Processing outline Calculator

1. Initialize variables
2. runningTotal
3. Stack result
4. Sixteen language variables
5. setVariables method
6. Get default locale and set locale
7. If local equals Chinese set language variables to Chinese language
8. Else if locale equals Russian set language variables to Russian language
9. Else set language variables to English language
10. Clear method
11. While result is not empty remove top stack memento, set running total to zero, print clear message
12. Undo method
13. If result is empty throw error message
14. Else running total equals running total subtract top stack memento, remove the last stack memento
15. Add method accepts double a and b
16. Creates new memento object
17. double total equals double a plus double b

iii. Running total + total

1. Set new memento’s total to double total
2. push new memento to stack of memento’s
3. Print total and running total
4. Subtract method accepts double a and b
5. Creates new memento object
6. double total equals double a subtract double b
7. Running total + total
8. Push new memento to stack of memento’s
9. Print total and running total
10. Multiply method accepts double a and b
11. Creates new memento object
12. Double total equals double a plus double b
13. Set new memento’s total to double total
14. push new memento to stack of memento’s
15. Print total and running total
16. Divide method accepts double a and b
17. Creates new memento object
18. Double total equals double a subtract double b
19. Running total + total
20. Push new memento to stack of memento’s
21. Print total and running total

**DATA**

***User Input***

The user input will be in the format of: real – literal, operator, real – literal. Each new line scanned will store user input in a String data type to be analyzed for the program use. The user will also have input options of: undo, clear, and exit. After each function or operation, the program will return a total of last operation, and a running total of all operations.

For example, a user enters 4.5+5.5 the output should return

*Sum: 10.0 Running Total: 10.0*

*Enter (CLEAR,UNDO,EXIT,Expression)*

**User Messages**

These messages are displayed to the user if English is locale language.

|  |  |
| --- | --- |
| **Message Number** | **Message Text** |
| 1 | Enter (CLEAR,UNDO,EXIT,Expression) |
| 2 | All calculations have been cleared |
| 3 | You must enter an input value in the example form of: 4.4+5.5 |
| 4 | Sum: |
| 5 | Running Total: |
| 6 | undo is unavailable |
| 7 | You must use the following operands between real-literal inputs: (+,-,\*,/) |
| 8 | product |
| 9 | difference |
| 10 | Quotient |
| 11 | product |
| 12 | undo |
| 13 | clear |
| 14 | exit |
| 15 | Cannot divide by 0 |

***Data Structures***

A stack which holds Memento objects. The Memento object has a single constructor element called Total which is the total of every operation the user has performed.

Therefore, each Memento object is initialized with the following:

 The totals of all operations the user has performed during the program.

**Access Methods**

To access the last stack object instance in the stack, the access method that is used is a peek () which looks at the last Memento object total without removing it from the stack. This method avoids a linear search on an object.

**Removal Methods**

To remove the last stack object instance, the removal method that is used is pop (). Pop () removes the last Memento instance and avoids linear search on an object.

**Clearing Methods**

To clear the stack data structure, the clearing method used is pop () while the stack is not empty. This method removes all Memento instances in the stack and is an O(1) operation for each instance in the stack. This method is linear based on number of stack objects.

**COMPONENTS**

A Memento class will be initialized to contain the operation totals that occurs in the program. The UML class diagram is shown below:

|  |
| --- |
| Memento |
| -total : E |
| +getTotal() : E total  +setTotal(in total : E) |

|  |
| --- |
| Calculator |
| -Stack : Memento  -runningTotal : double  +Locale : locale  +String promptOne, promptTwo, promptThree, promptFour, promptFive, promptSix, promptSeven, promptEight, promptNine, promptTen, promptEleven, promptTwelve, promptThirteen, promptFourteen, promptFifteen, promptSixteen |
| +Undo()  +setVariables()  +clear()  +add()  +subtract()  +multiply()  +divide() |

|  |
| --- |
| Main |
| -a : double  -b : double |
| +boolean first |

***Properties***:

**total:** This property represents the total of each operation

***Methods*:**

Memento (): The default constructor will initialize the properties as follows.

E Total;

**getTotal()**: This method returns the value of **total.**

**setTotal()**: This method assigns a value to **total.**

**TESTING**

The table below presents an overview of the testing scenarios and the status of running the scenarios.

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1 | Test for correct sum, running total, add, subtract, multiply, divide, Undo, Clear, and Exit functions | PASS |
| 2 | Test for user input invalid responses allowance | PASS |
| 3 | Test for proper language being used | PASS |

Each scenario is presented along with the steps required to execute the scenario.

*Scenario #1-* Test for correct sum, running total, and Undo, Clear, and Exit functions.

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1 | Run program | Prompt for user input |
| 2 | Insert a valid operation (Repeat for \* and / operators) | 4.5+5.5 |
| 3 | Check calculations returned for accuracy | Sum: 10.0 Running Total: 10.0 |
| 4 | Insert another valid operation | 8.0-5.0 |
| 5 | Check calculations returned for accuracy | Difference: 3.0 Running Total: 13.0 |
| 6 | Enter undo as user input | undo |
| 7 | Check calculations returned for accuracy | Running Total: 10.0 |
| 8 | Enter clear as user input | clear |
| 9 | Check calculations returned for accuracy | Running Total: 0.0 |
| 10 | Insert exit as user input | exit |
| EXPECTED OUTPUT | | Sum  Running Total  User input prompt |
| ACTUAL OUTPUT | | Sum  Running Total  User input prompt |
| RESULTS | | PASS |

*Scenario #2-* Test for user input invalid responses allowance

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/output |
| 1 | Run program | Prompt for user input |
| 2 | Insert invalid number format | Number format error message |
| 3 | Insert invalid operator | Invalid operator error message |
| 4 | Insert more than real literals | 4.5+5.5+6.5  Sum: 10 Running Total: 10.0 |
| EXPECTED OUTPUT | | -Number format error message  - Invalid operator error message  - Sum: 10 Running Total: 10.0 |
| ACTUAL OUTPUT | | -Number format error message  - Invalid operator error message  - Sum: 10 Running Total: 10.0 |
| RESULTS | | PASS |

*Scenario #3-* Test for proper language use

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/output |
| 1 | Run the program | Prompt for user input |
| 2 | Create string locale set to Russian and Chinese | String locale = “zh\_cn”; |
| 3 | Insert valid input | 4.5+5.5 |
| 4 | Copy and paste output in translator online | 小计： 10.0 累计： 10.0  输入（清除，撤消，退出，表达式 |
| EXPECTED OUTPUT | | 小计： 10.0 累计： 10.0  输入（清除，撤消，退出，表达式 |
| ACTUAL OUTPUT | | 小计： 10.0 累计： 10.0  输入（清除，撤消，退出，表达式 |
| RESULTS | | PASS |

**Sample Output for Test Scenario #1**

Output from the sample program to compare with the actual program.

Sample program:

Enter (CLEAR,UNDO,EXIT,Expression)

4.5 + 5.5

Sum: 10.0 Running Total: 10.0

Actual program:

Enter (CLEAR,UNDO,EXIT,Expression)

4.5+5.5

Sum: 10.0 Running Total: 10.0

**Analysis & Justification of Language/Dialect**

Localization in this program is demonstrated to accommodate North America, Europe, and Asia geographical locations. For the application to be as widely accepted as possible research was performed to find which languages the program should utilize. North America is an English dominant language with both Canadian and U.S. English versions being the most common. Both Canadian and U.S. English are very similar and the word variety/spelling in the program did not change from one type to another. Not only is English a dominant language in North America, but it is also the second most spoken languages in the world. English is the default language for this program. After researching Asia’s most dominant languages Chinese as expected is the most dominant language which influenced the second choice for the programs supported languages. Chinese not only is the most popular language in Asia, but it is also the most spoken language in the world. Finally, for the third language choice Europe was researched to find the most popular language spoken. Europe’s most popular language was found to be Russian and for the purpose of trying to accommodate for Europe’s geographical area, Russian is chosen for the third supported language of the program. The languages chosen for the program were done so without any bias. Choosing the supported languages was done using research of the geographical locations the program will be deployed and is focused on accommodating the general populations of each geographical area. By using the most popular languages of each geographical region and basing the decision making on that specified criteria this avoided bias as to which languages are supported by the program.

Cultural biases are often a factor when building programs as to which languages to use. Converting one language to another can prove to have some difficulties as keywords used in the program may not match up exactly to keywords in another language, and formatting may be different across languages with regards to punctuation, and numerical values. These complications can often influence which languages are supported by a program due to their transferability from one another. The languages in this program avoid that by using research-based decision making rather than easiest to implement. Other biases may be a factor including programmers known languages, and cultural background. To avoid issues with language support and accommodating the greatest population where the program is to be deployed this avoids cultural bias and provides the best success for the program. For a program to be successful globally the greatest population must be accommodated for unless a certain ethnicity is the target of the application for example an English to French translation program would likely support only English and French regardless of deployed location.