Thomas Coll

A calculator build to meet the specifications set by Jason Barron, lecturer at IT Carlow

Calculator Project

Object Orientated Programming

Table of Contents

Introduction 2

Requirements 2

Project development 2

Graphic User Interface 3

Calculation logic 4

Testing 12

Error log and available improvements 15

Conclusion 16

# Introduction

This project was given in 2016 by Jason Barron with the expected due date of 3rd April 2017. The project’s requirement was to create a calculator, using the Java language and to use Graphical User Interface (GUI) components to give it an appealing look and functionality. Main requirements are listed below.

# Requirements

The main requirements as stipulated by the assignment is to create a calculator which can perform the following actions via GUI:

* Addition
* Subtraction
* Multiplication
* Division
* Clear function
* Memory function

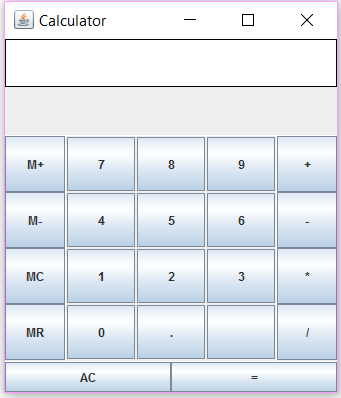
These are the most important function that must be included in application. Acceptable user interface is an advantage.

# Project development

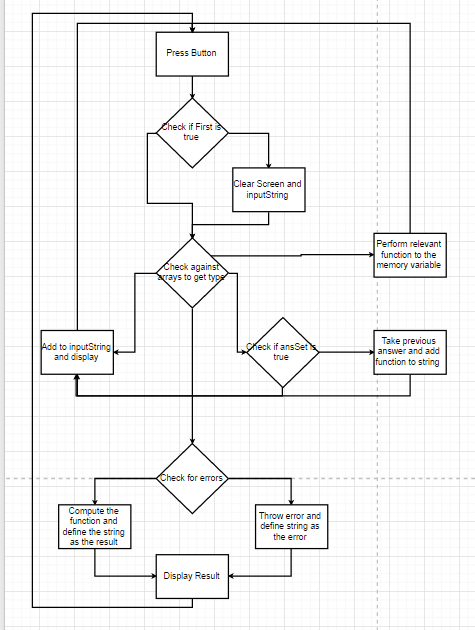
For development process, I chose to use Eclipse, a java based project development app which allows the creation of Javadoc’s and executable .jar files. This coupled with its excellent UI and GitLab connectivity makes it an ideal choice for project development.

## Graphic User Interface

GUI development was very simple process as it is just selecting items from menu and placing them on screen in appropriate way. NetBeans software automatically generates code such as variable name, and it’s constant position on screen. Graphic user interface look like:



## Calculation logic



Logic used for this calculator is based on is relatively simple but uses a more complicated implementation. Firstly, arrays are used to create buttons with the ability to interact with the mouse. Upon clicking a button if it is a number or a function, it shall add it to a string behind the scenes. The script engine manager will treat this string like a mathematical equation and upon the entering of calculate() will attempt to output the result of said equation, in the event it cannot or there is a mistake in the string, the system will throw an error message. The memory function is based on a simple double value stored in the calculator and is set to 0 by default. It can be added to or subtracted from, it can be cleared or pasted into the calculator. The AC button will clear the entire string and set the calculator back into its default state.

**if**(evt.getSource() == number[0])

{

**if**(first)

{

inputString = "7";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "7";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[1])

{

**if**(first)

{

inputString = "8";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "8";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[2])

{

**if**(first)

{

inputString = "9";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "9";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[3])

{

**if**(first)

{

inputString = "4";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "4";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[4])

{

**if**(first)

{

inputString = "5";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "5";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[5])

{

**if**(first)

{

inputString = "6";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "6";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[6])

{

**if**(first)

{

inputString = "1";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "1";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[7])

{

**if**(first)

{

inputString = "2";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "2";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[8])

{

**if**(first)

{

inputString = "3";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "3";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[9])

{

**if**(first)

{

inputString = "0";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "0";

screen.setText(inputString);

}

}

**else** **if**(evt.getSource() == number[10])

{

**if**(first)

{

inputString = ".";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += ".";

screen.setText(inputString);

}

}

//FUNCTIONS

//Addition

**else** **if**(evt.getSource() == function[0])

{

**if**(ansSet && first)

{

inputString = inputString + "+";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "+";

screen.setText(inputString);

}

}

//Subtraction

**else** **if**(evt.getSource() == function[1])

{

**if**(ansSet && first)

{

inputString = inputString + "-";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "-";

screen.setText(inputString);

}

}

//Multiplication

**else** **if**(evt.getSource() == function[2])

{

**if**(ansSet && first)

{

inputString = inputString + "\*";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "\*";

screen.setText(inputString);

}

}

//Division

**else** **if**(evt.getSource() == function[3])

{

**if**(ansSet && first)

{

inputString = inputString + "/";

screen.setText(inputString);

first = **false**;

}

**else**

{

inputString += "/";

screen.setText(inputString);

}

}

//MEMORY

//Memory Add

**else** **if**(evt.getSource() == memory[0])

{

**try**

{

memoryNum += Double.*parseDouble*(inputString);

first = **true**;

inputString = "" + memoryNum;

screen.setText(inputString);

}

**catch**(NumberFormatException a)

{

inputString = "Error parsing int";

screen.setText(inputString);

first = **true**;

}

}

//Memory Subtract

**else** **if**(evt.getSource() == memory[1])

{

**try**

{

memoryNum -= Double.*parseDouble*(inputString);

first = **true**;

inputString = "" + memoryNum;

screen.setText(inputString);

}

**catch**(NumberFormatException a)

{

**if**(inputString == "")

{

screen.setText("ERROR: no input to read");

}

**else**

{

screen.setText("Error Parsing input");

}

}

}

//Memory Clear

**else** **if**(evt.getSource() == memory[2])

{

memoryNum = 0;

}

//Memory Recall

**else** **if**(evt.getSource() == memory[3])

{

**try**

{

inputString += memoryNum;

screen.setText(inputString);

}

**catch**(NumberFormatException a)

{

**if**(inputString == "")

{

screen.setText("No input to read");

}

**else**

{

screen.setText("Error Parsing input");

}

}

}

//COMPUTATIONS

//Clear

**else** **if**(evt.getSource() == computations[0])

{

first = **true**;

ansSet = **false**;

inputString = "";

screen.setText("");

}

//Equals

**else** **if**(evt.getSource() == computations[1])

{

//inputString = screen.getText().toString();

**try**

{

inputString = calculate();

screen.setText(inputString);

first = **true**;

ansSet = **true**;

}

**catch**(DivideByZeroException e)

{

inputString = "Tried to divide by Zero";

screen.setText(inputString);

inputString = "";

first = **true**;

ansSet = **false**;

}

}

The buttons 0-9 and . will add the respective symbols to the calculator and shall display them as such. There is a Boolean variable called “first” and one called “ansSet”, first is used to mark if the inputted value is the first in the equation. If so it shall wipe the input string and screen. ansSet is a more specific variable, it is defined as false at the start and is turned true in the event a calculation is performed. It allows the system to put a function button (+-\*/) as the first and will add on to the anser.

Memory function. We have three button associated with it. “M+” button will take digit that is displayed and store it as separate variable. The M- button shall subtract the number from the memory variable. The “MC” button is short for memory clear and shall reset the memory variable to 0. The final button is “MR” this button will paste the stored number directly into the string and will remain stored in memory.

If “AC” button is used, the string is cleared and set back to nothing along with the same effect for the screen. The first and ansSet buttons are reset and left in their original states. Note the memory variable is not affected by this button.

These simple functions will meet the basic requirements that are listed above.

# Testing

I began testing on a regular basis, conducting the same test multiple times to control for potential errors and to give me a fuller reading of my own designs to control for potential errors. This was done by using my personal calculator (Casio fx-GT), the calculator built into the iPhone SE, and the windows calculator (on Windows 7 and 10), in order to control for potential errors in each calculator.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Objective | Test Input | Expected Output | Actual Output | Test Date | Result  P = Pass  F = Fail |
| 01 | Add two number | 5 + 10 | 15 | 15 | 10/01/2017 | P |
| 02 | Add three numbers | 5 + 10 + 10 | 25 | 25 | 10/01/2017 | P |
| 03 | Add 2 numbers and subtract one in a single input | 5+5-5 | 10 | 10 | 10/01/2017 | P |
| 04 | Subtract three numbers | 5-3-2 | 0 | 0 | 11/012017 | P |
| 05 | Multiply two numbers | 5\*2 | 10 | 10 | 11/01/2017 | P |
| 06 | Divide Two numbers | 10/2 | 5 | 5 | 15/01/2017 | P |
| 07 | Add a number to memory | 5 | Add 5 | Added 5 | 20/01/2017 | P |
| 08 | Subtract a number from memory | 5 | -5 | Error | 20/01/2017 | F |
| 09 | Subtract a number from memory | 5 | -5 | -5 | 20/01/2017 | P |
| 10 | Clear memory button | 15 | 0 | 0 | 20/01/2017 | P |
| 11 | Throws divide by zero exception | 10/0 | Error | Infinite | 10/02/2017 | F |
| 12 | Throws divide by zero exception | 10/0 | Error | Error | 10/02/2017 | P |
| 13 | Pastes Memory var to screen | 10+MR(5) | 10+5 | 10+5 | 15/02/2017 | P |
| 14 | Use AC button to clear screen | 10, AC | Empty Screen and the inputString variable being null | Empty Screen and the inputString variable being null | 01/03/2017 | P |

# Error log and available improvements

Upon clicking the = button the calculate() function is called, it first checks to see if the string is empty, if so then it will stop a potential error and notify the user. If the string isn’t empty then they try to perform the calculation, calling the engine script to calculate the function, but If the inputNum result is infinite (Double.*longBitsToDouble*(0x7ff0000000000000L)), then throw an error and return the inputString, if there is an error in the calculation (double – etc.) then catch the ScriptException and display an error.

Object result = **null**;

**double** inputNum;

**if**(inputString != "")

{

**try**

{

result = engine.eval(inputString);

inputString = result.toString();

inputNum = Double.*parseDouble*(inputString);

**if**(inputNum == Double.*longBitsToDouble*(0x7ff0000000000000L))

{

**throw** **new** DivideByZeroException();

}

**return** inputString;

}

//divide by zero error, input error

**catch** (ScriptException e)

{

System.***out***.println("Error evaluating input string.");

screen.setText("Error Parsing input");

inputString = "";

screen.setText(inputString);

first = **true**;

}

}

**else**

{

System.***out***.println("ERROR: No input");

screen.setText("");

inputString = "ERROR: No input";

first = **true**;

}

**return** inputString;

Due to the time constraints of the project and pressure from other projects I was unable to add all the functionality I wanted into the project along with more errors, I wished to have more custom error checks but due to time constraints I was unable to add them. I would also have liked to be able to add additional functionality, such as the ability to use Squared and Square Root functions, I also added certain serial IDs, mostly to make Eclipse happy, although it wasn’t consistent, had I more time I would have done them properly.

# Conclusion

This assignment was completed and zipped on 02/04/2016 and done to the best of my ability considering the time frame, I have completed the Javadoc and will add it to the zipped file. Output of this assignment is executable .jar file that is cross-platform compliant and do not need any addition software to run.