Calculator Assignment

Object oriented Programming- Project Ca

Michael Norman

Table of Contents

[Introduction 1](#_Toc478766832)

[Requirements 1](#_Toc478766833)

[Project Development 1](#_Toc478766834)

[Graphic User Interface 1](#_Toc478766835)

[Stage one: 2](#_Toc478766836)

[Stage two: 2](#_Toc478766837)

[Final look: 3](#_Toc478766838)

[Calculation Logic: 4](#_Toc478766839)

[Basic operands: 5](#_Toc478766840)

[Memory functions: 6](#_Toc478766841)

[Memory add: 6](#_Toc478766842)

[Memory Delete: 7](#_Toc478766843)

[Memory Recall: 8](#_Toc478766844)

[Memory Clear: 8](#_Toc478766845)

[Other functions: 8](#_Toc478766846)

[Decimal point: 8](#_Toc478766847)

[Clear: 9](#_Toc478766848)

[Equals: 9](#_Toc478766849)

[Testing: 9](#_Toc478766850)

[Error logs and available improvements: 11](#_Toc478766851)

[Conclusion 11](#_Toc478766852)

[Testing documentation 11](#_Toc478766853)

[Specification: 13](#_Toc478766854)

[Pseudocode: 13](#_Toc478766855)

[Constructor pseudocode: 13](#_Toc478766856)

[Action listener pseudocode: 13](#_Toc478766857)

[Add digit method pseudocode: 14](#_Toc478766858)

[Add operands method pseudocode: 14](#_Toc478766859)

[Memory add method: 14](#_Toc478766860)

[Memory remove method: 15](#_Toc478766861)

[Memory clear method: 15](#_Toc478766862)

[Memory recall method: 15](#_Toc478766863)

# **Introduction**

This assignment combined all the aspects of programming that we learned this year combined, we were assigned to design and develop a calculator in java that would allow basic operations along with more complex operations like a memory function.

Use of correct programming practices and java conventions are essential to create the final application along with other requirements listed below.

# **Requirements**

The main requirements are to develop a program using java GUI components, the following requirements need to be met to achieve full marks:

* Addition
* Subtraction
* Multiplication
* Division
* Memory functions
  + Add to Memory
  + Delete from memory
  + Clear memory
  + Recall from memory
* A button to clear the screen
* The use of programmer defined exception classes

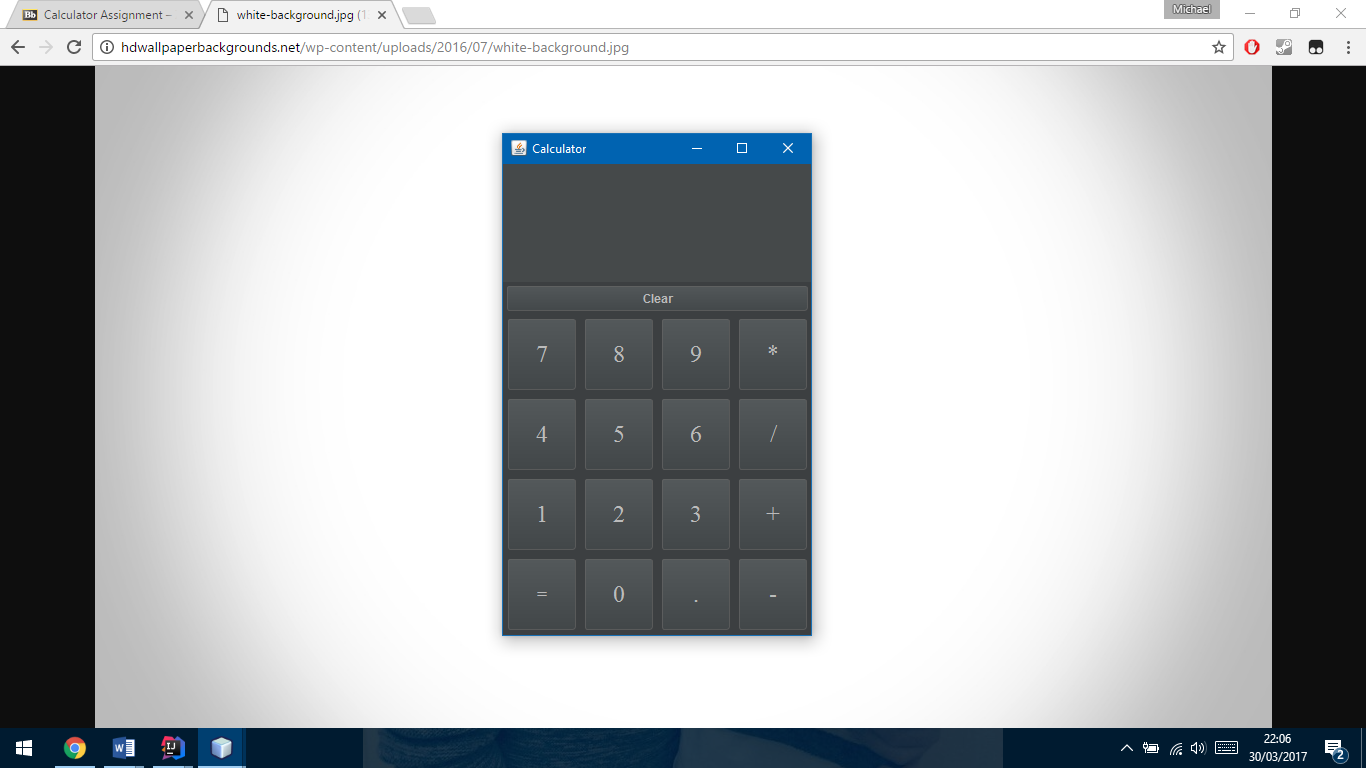
# **Project Development**

For the development of this project I used two different IDE’s Netbeans IDE 8.2 and IntelliJ IDEA 2017.1, Netbeans is a free software that has built in GUI development tools for easy use which is what I used to design the layout for the calculator. I then switched to IntelliJ as it is the usual IDE that I use and I continued the rest of the project using this software.

# Graphic User Interface

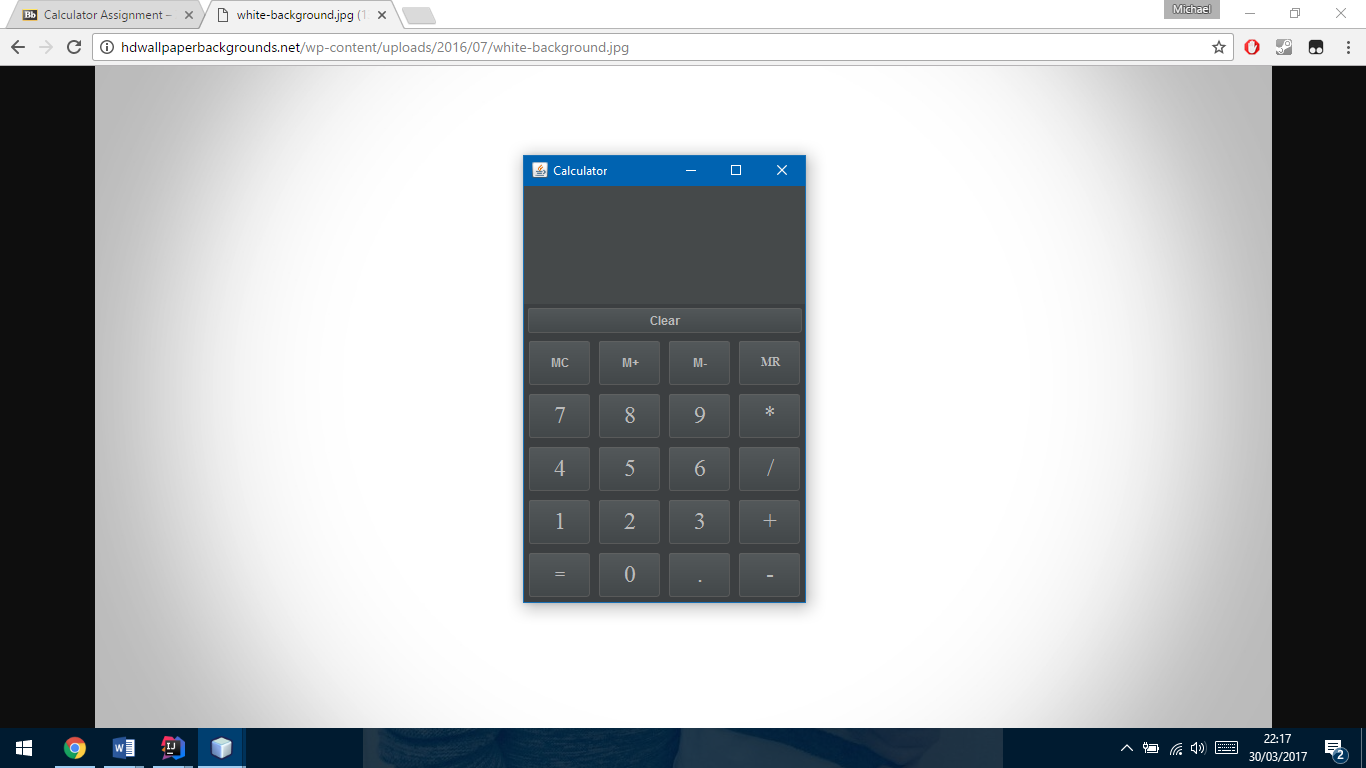
I done Two stages for the design of this to make it easier to program at the start.

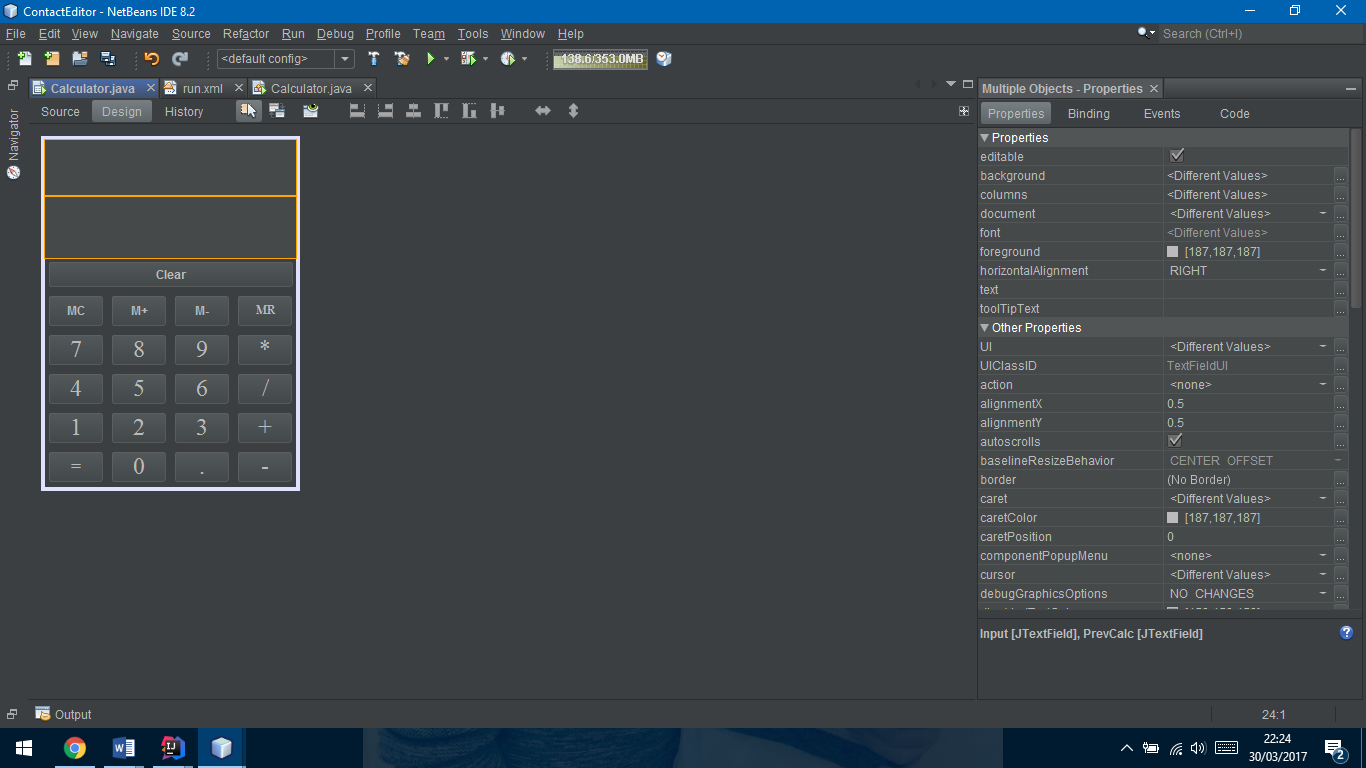
## **Stage one:**

I started with the basic functions add subtract Etc.

This made programming the basic commands first easier and allowed me to not worry about more operations than I needed to and made it feel easier by splitting the work up a bit.

## **Stage two:**

This left only the memory functions to add which I designed like this.

I also added a secon output field that can only be seen in the editor this was added to show previous calculations. Please note the style of the calculators is due to the editor the final version will look different.

## **Final look:**

# **Calculation Logic:**

Enter first digit using keypad

Enter second number from numpad

Move Digit and operand to top screen

If +-/\*

Press operator button [+ , - , \*, / , clear]

If memory button

If MR

Add number to memory

If M+

Iterate through memory array and display to screen

If M-

Delete this number from the memory array

Press equals, answer displayed, proceed with next number

## **Basic operands:**

The logic used for basic calculations is mainly through a switch case function and a Boolean to see if there’s a number beforehand.

This looks as follows

String operand = ((JButton)button.getSource()).getText();  
switch (operand)  
{  
 case "+":  
 if (Input.getText()!="")  
 {  
 if (newCycle){  
 PrevCalc.setText(nf.format(answer));  
 newCycle = false;  
 }  
 num1 = Double.*parseDouble*(Input.getText());  
 operands='+';  
 PrevCalc.setText(Input.getText() + operand);  
 Input.setText("");  
 }  
 break;  
  
 case "-":  
 if (Input.getText()!=""){  
 if (newCycle){  
 PrevCalc.setText(nf.format(answer));  
 newCycle = false;  
 }  
 num1 = Double.*parseDouble*(Input.getText());  
 operands='-';  
 PrevCalc.setText(Input.getText() + operand);  
 Input.setText("");  
 }  
 break;

This code checks if the screen has numbers if there is a new cycle ie if you done a calculation before and want to continue using the answer.

This code uses a lot of repeated conditions so it makes sense to tidy it up by creating a method to do this for us, the new code looks like this..

String operand = ((JButton)button.getSource()).getText();  
switch (operand)  
{  
 case "+":  
 variableSet(operand);  
 break;  
  
 case "-":  
 variableSet(operand);  
 break;  
  
 case "\*":  
 variableSet(operand);  
 break;  
  
 case "/":  
 variableSet(operand);  
 break;

This significantly reduces the code needed for the same process the method created looks like this..

private void variableSet(String operand){  
 //checks if the text field isn’t empty  
 if (Input.getText()!="")  
 {  
 //if you’re not continuing a calculation  
 if (newCycle){  
 PrevCalc.setText(nf.format(answer));  
 newCycle = false;  
 }  
 //setting number 1 to the input for calculation later   
 num1 = Double.*parseDouble*(Input.getText());  
 //set operand for future calculation   
 operands=operand.charAt(0);  
 //moving the number and operand to the top screen  
 PrevCalc.setText(Input.getText() + operand);  
 //clearing the main text field  
 Input.setText("");  
 }  
}

## **Memory functions:**

Other functions like the memory buttons go through the same operation when there called through the switch call, their corresponding methods are called to do their functions.

case "MR":  
 memoryRecall();  
 break;  
  
case "M+":  
 memoryAdd();  
 break;  
  
case "M-":  
 memRemove();  
 break;  
  
case "MC":  
 memoryClear();  
 break;

Each method has their own test and validations shown below.

### **Memory add:**

This makes sure the memory Array List isn’t empty and it doesn’t contain the same number to stop repeating numbers or accidental double-clicks filling up the array, it also sets a limit to the Array List to 6 and if the limit is met it replaces the last one in the array and shifts the down by one placing the newest number at the start.

It then sets all the buttons to enabled as by default they are disabled as have no use while there are no numbers in the array list.

private void memoryAdd() {  
 newCycle = true;  
 //while the text screen isn't empty  
 // and the memory arrayList doesn't contain the desired number  
 while (!Input.getText().equals("") && !memory.contains(Input.getText()))  
 {  
 //setting the max amount of numbers in the arrayList to 6  
 //if the array is full remove the last one and push all down by one  
 // and add number to the start of the array  
 if (memory.size() == 6) {  
 memory.add(0, Input.getText());  
 memory.remove(5);  
 }  
 //else just add to the start of the array push others down one  
 else {  
 memory.add(0, Input.getText());  
 }  
 //while the array isnt empty enable all other memory buttons  
 ButMemRecall.setEnabled(true);  
 ButMemClear.setEnabled(true);  
 ButMemSub.setEnabled(true);  
 }  
}

### **Memory Delete:**

This checks if the array contains the number on the screen if it is find its location and removes it from the memory array list.

It also sets the other buttons to disabled if it removes the only number in the array list.

private void memRemove(){  
 //if the array contains the number  
 if (memory.contains(Input.getText()))  
 {  
 //sets the location of the number for deletion   
 int location = memory.indexOf(Input.getText());  
 //removes the number at the location from earlier   
 memory.remove(location);  
 }  
 //if the array list is empty set all the buttons to disabled   
 if (memory.isEmpty()){  
 ButMemClear.setEnabled(false);  
 ButMemRecall.setEnabled(false);  
 ButMemSub.setEnabled(false);  
 }  
}

### **Memory Recall:**

This uses error handling to catch the out of bounds exception if the button count tries to match the index of the memory array and it doesn’t exist the catch then sets the button count to the size of the array and iterates again.

private void memoryRecall()throws IndexOutOfBoundsException {  
 //button count set to 5 to match array limit  
 //try to set the corresponding button count  
 //to index number and print to screen  
 //iterate down through the array  
 try {  
 Input.setText(memory.get(buttonCount));  
 buttonCount--;  
 }  
 //if it goes out of bounds   
 catch (IndexOutOfBoundsException e){  
 //if the array is empty set the screen to blank  
 if (memory.isEmpty())  
 {  
 Input.setText("");  
 }  
 //else set the buttoncount back to 5  
 //iterate again   
 else {  
 buttonCount = memory.size() - 1;  
 Input.setText(memory.get(buttonCount));  
 buttonCount--;  
 }  
 }  
}

### **Memory Clear:**

This clears all items from the array, sets the output the blank and disables all memory buttons except the memory add button.

private void memoryClear(){  
 //clears all items in the array list   
 memory.clear();  
 //sets the screen to blank   
 Input.setText("");  
 //sets all the memory buttons to disabled  
 ButMemRecall.setEnabled(false);  
 ButMemClear.setEnabled(false);  
 ButMemSub.setEnabled(false);  
}

## **Other functions:**

Other functions such as equals, decimal point and clear buttons are accessed through the same switch case operation and have their own validations.

### **Decimal point:**

Checks if the string contains a decimal point already, if not add it to the screen.

case ".":  
 //set check to the text on the screen  
 String check = Input.getText();  
 //tests to see if the string check already contains a decimal point  
 boolean test = check.contains(".");  
 //if test isnt true allow the decimal point to be used   
 if (!test){  
 Input.setText(Input.getText()+".");  
 }  
 break;

### **Clear:**

This clears both text screens of text.

case "Clear":  
 //stops newCycle  
 newCycle = false;  
 //clears both textfields  
 Input.setText("");  
 PrevCalc.setText("");  
 break;

### **Equals:**

This sets the second number to the inputted number, checks if it’s a new cycle and if it is sets the full equation to the top screen.

case "=":  
 //sets number two the input and parses to double  
 num2 = Double.*parseDouble*(Input.getText());  
 //if its a new cycle  
 if (newCycle) {  
 //set the equation to the top screen and changes new cycle to false  
 PrevCalc.setText(PrevCalc.getText()+ operands + nf.format(num1));  
 newCycle = false;  
 }  
 else {  
 //if newcycle is false just top text to the calculation  
 PrevCalc.setText(PrevCalc.getText() + Input.getText());  
 }  
 //get answer from calculation method  
 answer = calculation(num1,num2);  
 //set text to answer and format it by numberformat method   
 Input.setText(nf.format(answer));  
 newCycle = true;  
 break;

# **Testing:**

The testing helps decide whether the basic functionality is working properly and produces correct results. To test if the result is correct I used the built-in calculator in Microsoft windows.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test  Objectives/Scenarios | Test cases (Input Data) | Expected Results | Actual result | Test Date | Status |
| 1 | Testing Addition | Adding digits:  12 and 2.5 | 14.5 to be displayed | 14.5 displayed | 25/03/2017 | Successful |
| 2 | Testing Subtraction | Subtracting digits:  12 and 2.5 | 9.5 to be displayed | 9.5 displayed | 25/03/2017 | Successful |
| 3 | Testing division | Dividing digits:  12 and 2 | 6 to be displayed | 6 displayed | 25/03/2017 | Successful |
| 4 | Testing multiplication | Multiplying digits:  12 and 2 | 24 to be displayed | 24 displayed | 25/03/2017 | Successful |
| 5 | Test if decimal point can be inserted more than one time | Testing if decimal point can be added multiple times. | Functions as expected. | Functions as expected | 25/03/2017 | Successful |
| 6 | Try to perform any of action that requires two variables in sequence without clicking equals sign | Try to perform following sequence:  3+4+5=12 | Application should not allow to enter more than two variables without clicking equal | Top screen replaces the with the new operands | 25/03/2017 | Unsuccessful |
| 7 | Testing if memory add works | Adding 112 to memory. | Memory recall, clear and delete buttons to be usable and memory recall to show 112 | Buttons unusable works as expected | 25/03/2017 | Successful |
| 8 | Testing if memory delete works | Deleting 112 from memory. | Either entering in the number in the text box or finding it through memory recall and being removed | Works as expected | 25/03/2017 | Successful |
| 9 | Testing if memory recall works | Recall 112 from memory. | After pressing memory recall 112 prints to screen | 112 printed to the screen | 25/03/2017 | Successful |
| 10 | Testing if memory clear works | Check if you can press the other memory buttons | Functions as expected | Functions as expected | 25/03/2017 | Successful |

# **Error logs and available improvements:**

When entering multiple numbers and operands the newest number replaces the previously entered number ie. Entering 33 + 5 then pressing the + symbol again replaces 33 with 5.

This could be improved by allowing a running total for the answer to allow for continuous calculation without pressing the equals button.

# **Conclusion**

Output of this assignment is executable .jar file that is cross-platform compliant and do not need any addition software to run.

# **Testing documentation**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Number** | **Test scenario** | **Input Data** | **Expected result** | **Actual result** | **Test date** | **Status** |
| 1.1 | User Clicks a number button | Number | Number shows on screen | Number appears | 25/03/2017 | Complete |
| 1.2 | User enters a number, clicks on the addition button, enters another number and clicks the equals button | Numbers | Two numbers added and shown on screen | Addition correct | 25/03/2017 | Complete |
| 1.3 | User enters a number, clicks on the minus button, enters another number and clicks the equals button | Numbers | Two numbers subtracted and shown on screen | Subtraction correct | 25/03/2017 | Complete |
| 1.4 | User enters a number, clicks on the multiplication button, enters another number and clicks the equals button | Numbers | Two numbers multiplied and shown on screen | Multiplication correct | 25/03/2017 | Complete |
| 1.5 | User enters a number, clicks on the division button, enters another number and clicks the equals button | Numbers | Two numbers divided and shown on screen | Division correct | 25/03/2017 | Complete |
| 1.6 | User clicks the clear button to clear the text field | None | Text field cleared | Text field cleared | 25/03/2017 | Complete |
| 1.7 | User clicks on the “X” in the top right hand corner | None | Window closes | Window closed | 25/03/2017 | Complete |

# **Specification:**

Must develop a java calculator application with the following functionality:

1. Addition [+]
2. Subtraction [-]
3. Multiplication [\*]
4. Division [/]
5. Provide a way to clear the screen of numbers [ clear]
6. Provide memory functions to remember numbers [ M+, M-, MC, MR]
7. Use programmer defined Exception classes for error handling
8. Except integer and decimal numbers

# **Pseudocode:**

## Constructor pseudocode:

* Call initialising components method

Initialising components method

* Declare all the buttons needed
* Declare two text fields for top and bottom display
* Declare two panels one for button one for text fields
* Button panel
  + Set grid layout with 4 columns and 5 rows.
  + Set the min size so text doesn’t disappear.
  + Set font and size.
  + Set their values (1, 2, 3, MR, MC, etc)
  + Add them to the button panel
* Text field panel:
  + Set border layout
  + Add two text fields
  + Make top panel un-editable
* Add panels to the content pane
* Set all memory buttons to disabled except for memory add

## **Action listener pseudocode:**

* Get value of button pressed
* Check if it is a number
* If it’s a number go to add digit method sending the event
* Else go to add operands method sending the event

## **Add digit method pseudocode:**

* Get the value of the button pressed
* If this is a new calculation
  + Clear the screens of previous calculations
  + Add the new number to the display
  + Set the check for a previous calculation to false
* If this isn’t a new calculation
  + Add the number to the main display

## **Add operands method pseudocode:**

* Get the value of the button pressed
* Find which operand was pressed
  + If +, -, \*, /
    - Send to operand Processing method
  + If =
    - Set the second number to the value on the input screen
    - If it’s a new cycle move equations to top screen and set new cycle false
    - Else move the number one and operand to the top screen
    - Then send number 1 and 2 to calculation method to process and receive
    - Set the answer to the main text field
    - Set the new cycle to true
  + If clear
    - Set both text fields to blank
    - Set the new cycle to false
    - Set numbers to 0
  + If .
    - Get the current text in the main text field
    - Check the text to see if it already contains a decimal point
    - If it doesn’t contain a decimal point allow it to be put in
  + If M+
    - Call memory add method
  + If M-
    - Call memory remove method
  + If MC
    - Call memory clear method
  + If MR
    - Call Memory recall method

## **Memory add method:**

* Set new cycle to true
* While the text area isn’t empty and the number in the text
* Check if the array is full
* if it is add the new number to the start of the array
* Remove the last item in the array
* Move all items down one
* If its not full just add it to the start of the array
* Set all other memory buttons to enabled

## **Memory remove method:**

* Check if the number on screen is inside the memory array
* If it is get the location of the number in the array and remove it
* Then check if its empty
* If it is then set all memory buttons to disabled except the memory add button

## **Memory clear method:**

* Clear all items from the array
* Set the main text field to blank
* Set all memory buttons to disabled except memory add

## **Memory recall method:**

* Set a button count number to iterate through the array
* Set the main text field to the button count
* If the button count is more than the array size then reset it