Programming Report – Calculator Program

CI404 Introduction to Programming

By James Robert Maile

Contents

Introduction

Figure 1. Google Calculator

Source: CNET.com [1]

For this module, I have decided to work on the Calculator source code that was provided. The first thing i am going to address in this report is the design element. Since the calculator already has the simple gray colour pallet this will not be touched as it is similar to many other calculators and blends in well. See reference of google calculator as an example. [1]

My first point of modification within the codebase is to add in the rest of the functions as the buttons are there just not actually working. This is just a simple feature to add first. Once again this is spoken in detail later on down in the report.

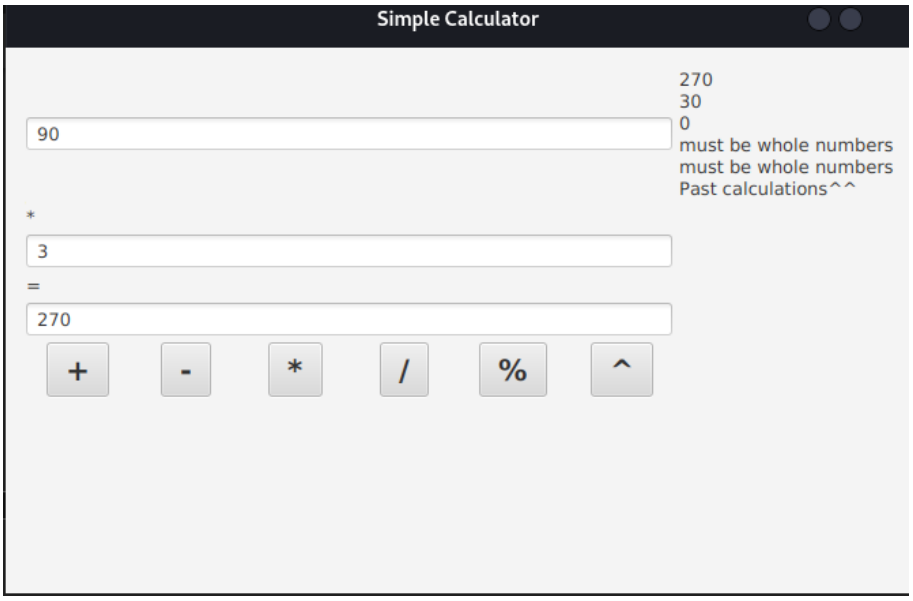
For new features, I will add more buttons, these new buttons would add extra functions like power, percentage and even square root. Another feature I will add is History, this is important to have so you can view your past calculations.

Calculator and development

Design:

In the section, we will talk about the design of the calculator. The source code that was provided gave a very simple calculator design that is all gray and consists of 3 text boxes, a gray background and a basic button layout.

In this case, i have kept the simple design as it is easy to modify and add more features.



The picture above shows my calculator whilst in development. As you can see from the picture, it has kept that very simple look. My justification for this is that there isn't much you can do in terms of adding colour without making it look weird. This bland colour scheme is on par with the calculator theme just like what the google calculator looks like.

This also uses Grids to make up the interface. This is important for later on as it also causes a bug.

Development:

Note: All classes have javadoc notes stating what the class is and does.

Feature 1: Restored basic calculator functionality

The first feature of mine is that I added back the calculator's functionality, as the source code was unfinished. The Buttons were there but had no functionality whatsoever. below i will show my code that i used to regain functionality.

Added code:

```
//Subtraction added here
case "-":
    model.doSub();
    break;

//Multiplication added here
case "*":
    model.doMul();
    break;

//dividing added here
case "/":
    model.doDiv();
    break;
```

```
    }
    public void doAdd(){
        String resultStr = null;
        try{
            int num1 = Integer.parseInt(view.textNum1.getText());
            int num2 = Integer.parseInt(view.textNum2.getText());
            int result = calculator.add(num1,num2);
            resultStr = result + "";
        }
        catch(Exception e){
            resultStr = "Whole numbers please :)";
            view.update("","", resultStr);
        }
        String operatorIcon = "+";
        String equation = "= ";
        view.update(operatorIcon,equation, resultStr);
    }
    public void doSub(){
        String resultStr = null;
        try{
            int num1 = Integer.parseInt(view.textNum1.getText());
            int num2 = Integer.parseInt(view.textNum2.getText());
            //Pulled subtraction method from calculator
            int result = calculator.sub(num1,num2);
            resultStr = result + "";
        }
        catch(Exception e){
            resultStr = "Whole numbers please :)";
            view.update("","", resultStr);
        }
        String operatorIcon = "-";
        String equation = "= ";
        view.update(operatorIcon,equation, resultStr);


        }

    public void doMul(){
        String resultStr = null;
        try{
            int num1 = Integer.parseInt(view.textNum1.getText());
            int num2 = Integer.parseInt(view.textNum2.getText());
            //pulled muliplication method from Calculator class
            int result = calculator.mul(num1,num2);
            resultStr = result + "";
        }
        catch(Exception e){
            resultStr = "Whole numbers please :)";
            view.update("","", resultStr);
        }
        String operatorIcon = "*";
        String equation = "= ";
        view.update(operatorIcon,equation, resultStr);
```

```
public void doDiv(){
    String resultStr = null;
    try{
        int num1 = Integer.parseInt(view.textNum1.getText());
        int num2 = Integer.parseInt(view.textNum2.getText());
        //pulled dividing method from Calculator class
        int result = calculator.div(num1,num2);
        resultStr = result + "";
    }
    catch(Exception e){
        resultStr = "Whole numbers please :)";
        view.update("","", resultStr);
    }
    String operatorIcon = "/";
    String equation = "= ";
    view.update(operatorIcon,equation, resultStr);
```

The top screenshot of code shows the controller, it has the simple purpose of allowing the program to perform it's task. In this instance it is multiplying, subtracting and dividing. The added code in that section allows it to do just that.

The model which is shown in the 2nd screenshot, processes the data inputted (numbers in this case) and processed them using the calculator class into an Output which is the "Result".

Update: The buttons used to have the proper divide and times symbols when I first edited the codebase but due to some minor complications, they have been replaced with their respective counterparts which are * and /.

Feature 2: added power calculation functionality

Here, I have added an extra Operator called Power. Here is the code for it below.

```
String[] buttonText = { "+","-","*","/","%","^","√"};
```

Added button is "^" for power

```
public int pow(int a, int b) {
    return (int) Math.pow(a, b);
}
```

Actual calculation for power, used w3 schools for help finding the right bit of code to use [2]

```
public void doPow(){
    String resultStr = null;
    try{
        int num1 = Integer.parseInt(view.textNum1.getText());
        int num2 = Integer.parseInt(view.textNum2.getText());
        //Pulled power method from calculator
        int result = calculator.pow(num1,num2);
        resultStr = result + "";
    }
    catch(Exception e){
        resultStr = "Whole numbers please :)";
        view.update("","", resultStr);
    }
    String operatorIcon = "^";
    String equation = "= ";
    view.update(operatorIcon,equation, resultStr);


    }
```

How power is processed.

```
        //power added here
        case "^":
            model.doPow();
            break;
```

So the calculation can be preformed

Above, you can see multiple different bits of code have been added to this so the calculation can actually work. The screenshot is code used to make buttons. The button we are interested in here is the power key (“^”).

The 2nd screenshot is the code used to make sure the calculator could process the calculation correctly. Here in the code, we also used the Math.pow [2] java code instead of just using a symbol like most of the others. This is because it didn’t work as well with just the ^ symbol. I have put simple notes just describing what is happening but some of those may not be visible here. Looking at the source code is suggested for a full overview of the notes I have left.

The 3rd screenshot is just how the calculation is once again processed. Same as the other calculations. It just collects the inputs and uses the Calculator class to do the calculation and then spits out the output.

The 4th and last screenshot shows the code to make the button for power to work.

Feature 3: Added history

In this part, I will show how I added history to my calculator.

Added code:

```
    //components of the user interface
    Label      operator, equal,resultMeaasge, chistory;          //
    TextField  textNum1,textNum2, textResult;
      // result area, where number appear
    GridPane   gridPane;      // main layout manager grid
    TilePane   buttonPane;    // tiled area for buttons

    //calchis (calchistory), appears as a single string
    private String Calchis = "";
```

```
//chistory - Label added to show history of calculations

chistory = new Label("Past calculations^^ ");
gridPane.add(chistory, 2, 1);
```

```
}
public void update(String operIcon, String equalIcon, String result)
{
    if (model != null){
        operator.setText(operIcon);
        equal.setText(equalIcon);
        textResult.setText(result);
        chistory.setText(String.valueOf(result)+ "\n" +chistory.getText());

    }

}
```

The top pictures shows an added label called chistory and a new private string called Calchis. These are used to display the results in a container. This is what is shown in use on the right-hand side in the design segment of this document.

The 2nd screenshot shows me adding a bit of text to the Label, this allowed me to show where past calculations actually were. This just helps with identifying the in the main program window.

The last screenshot here shows how past calculations are actually displayed. The chistory label grabs the last calculation stored in the Result label aka grabbing the value of the Results. This allows it to actually display the last known calculation made in the container (chistory label).

This was the easiest way possible to add history. Any other attempt would of taken longer due to having a slightly novice skill set.

Feature 4: Added percentage calculation functionality

Adding percentage was quite easy as it followed the same rules as the calculations in in feature 1.

```
//calculation for Percentage
public int per(int a, int b) {
    return a % b;
}
```

Actual calculation method for percentage.

```
public void doPer(){
    String resultStr = null;
    try{
        int num1 = Integer.parseInt(view.textNum1.getText());
        int num2 = Integer.parseInt(view.textNum2.getText());
        //Pulled percentage method from calculator
        int result = calculator.per(num1,num2);
        resultStr = result + "";
    }
    catch(Exception e){
        resultStr = "Whole numbers please :)";
        view.update("","", resultStr);
    }
    String operatorIcon = "%";
    String equation = "= ";
    view.update(operatorIcon,equation, resultStr);
```

This is once again how it's processed and outputted.

```
//percentage added here
case "%":
    model.doPer();
    break;
```

Allows calculator to recognize the button.

```
// Buttons - these are laid out on a tiled pane, then
// the whole pane is added to the main gridPane
// Button text - empty strings are for blank spaces

String[] buttonText = { "+","-","*","/","%","^","√"};
```

Added in the percentage key as “%”.

The first screen shot shows the calculation for percentage. This just allows it to do the calculation but without it’s model which is in the 2nd screen shot. The model allows the calculator to process the inputs, run it through the calculator class and then spit out the output. The Percentage key as stated above is the “%” symbol and in the 3rd picture is how the button actually works, what it does it it detects when the text/button was activated then sends a request to the model which then does the requested calculation and then spits it out into the view.

Update: I am not sure how well the percentage is actually working, it definitely works but probably not to its full capacity. Solution is in the bug area of this report.

Feature 5: added square root calculation functionality

Adding square root was the most complicated, I will describe it all after I show the code

Added code:

```
public double sqrt(double a) {
    if (a < 0) throw new IllegalArgumentException("Square root of negative number is undefined by calculator");
    return Math.sqrt(a);
}
```

Added calculation using java Math.sqrt to get the calculation to work. Also added new exception message. [3] [4]

```
public void doSqrt() {
    String resultStr = null;
    try {
        double num = Double.parseDouble(view.textNum1.getText());
        //Pulled Sqrt method from calculator class
        double result = calculator.sqrt(num);
        resultStr = String.valueOf(result);
    } catch (NumberFormatException e) {
      resultStr = "Please Input a Number :)";
    } catch (IllegalArgumentException e) {
      resultStr = e.getMessage();
    }
    String operatorIcon = "√";
    String equation = "= ";
    view.update(operatorIcon, equation, resultStr);
}
```

Added the method with some changes to declare the output as a double instead of integer, and to accept only 1 input.

```
    //square root added here
    case "√":
        model.doSqrt();
        break;
```

Button functionality added

```
// Buttons - these are laid out on a tiled pane, then
// the whole pane is added to the main gridPane
// Button text - empty strings are for blank spaces

String[] buttonText = { "+","-","*","/","%","^","√"};
```

Button added with "√" symbol

This was a difficult function to get working due to my limited knowledge of java. I used lectures and online websites like W3Schools to understand what i needed to do to get this functioning. [3]

Due to the square root needing to be a double I had to declare the result as a double so it would output. This declaration is in the 2nd screenshot which shows the model. In the first

image i used math.sqr [3] as it was the only way to get this function to work. I also had to figure out how to add Exceptions in the model.

Other than the change in both the Model and Calculator class it was all the same as the others.

Blackbox testing

Here I will conduct Blackbox testing as with time constraints, unit testing wasn't a possibility for me.

Test no.	Description	Expected result	Actual Result
1	Input a number into first text box	Number is inputted into text box	As expected, Number was in Text box
2	Input a number into Second text box	Number is inputted into text box	As expected, Number was in Text box
3	Try out Addition button	The button should add both numbers in text boxes together and put them in the output box. Also should change operator icon into +	Output is correct and in the correct text box. Operator icon also changed correctly
4	Try out subtraction	The calculation should subtract both numbers in text boxes and put the result in the output box. Also should change operator icon from + to -	Output is correct and in the correct text box. Operator icon also changed correctly
5	Try out Division	The calculation should divide the numbers successfully and put it in the output box. Operator should also change to / symbol	Calculation successful and put into output box. Operator also changes correctly.
6	Try out Multiplication	The calculation should multiply the numbers successfully and put it in the output box. Operator should also change to * symbol	Calculation successful and put into output box. Operator also changes correctly.

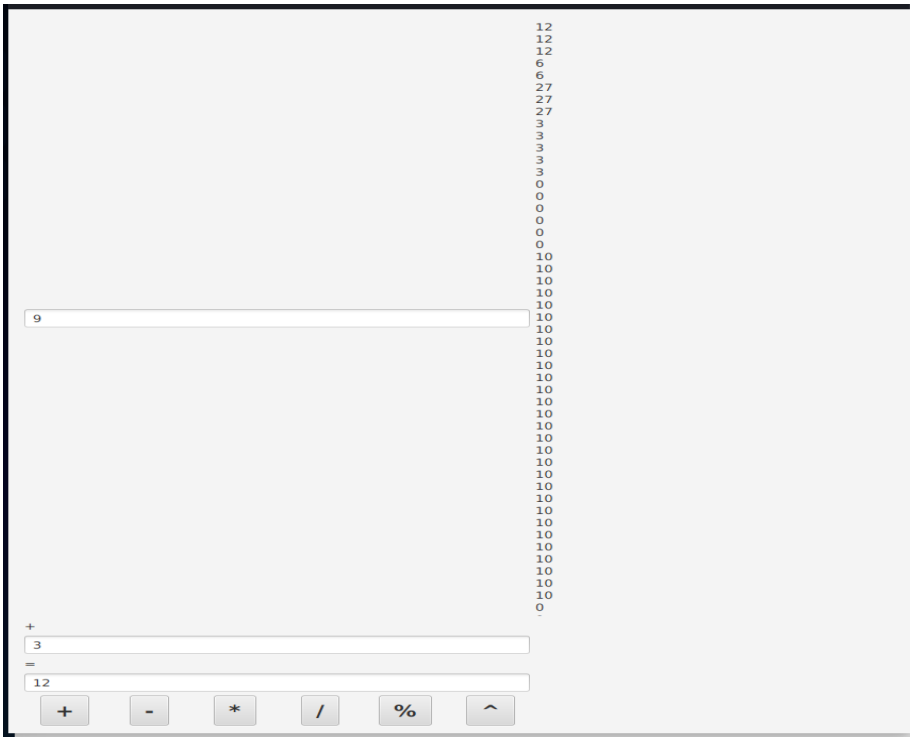
7	Try out Power	Power calculation should be correct and put into the output text box. Operator icon should change to ^ symbol	Calculation successful and put into output box. Operator also changes correctly.
8	Try out Percentage	The percentage calculation should be an accurate calculation of the percentage, Operator must also change to the % sign	Operator changes but the percentage function in my opinion doesn't work that well. It calculates to a point. Fix implemented, please check below.
9	Try out Square root and put a letter in to test the exception	The square root of the input number calculated successfully and put into the output box. Exception message also displays when anything but a number is put in.	Calculation is successful and put into output box. Exception message also works well.
10	Check if history displays all pervious calculations	Yes it displays all calculations done in the past.	Yes it does display all past calculations but it bugs out and extends the calculators grid causing the calculator to stretch.

Bugs

In this section of the report I will discuss a few bugs i have encountered during development and discuss if they have been fixed or if a fix hasn't been implemented yet.

Bug 1: Calculator extends when History builds up

This bug is caused by the history also being part of the grid which in turn stretches out the interface to weird proportions. currently due to my skill set I cannot find a way to fix this at the moment. Research is underway to find a solution. The picture below shows the issue.



Bug 2: Percentage function inaccurate.

The percentage function I believe is inaccurate, possible cause is the use of the Symbol in the calculation class, new fix has been implemented, and test results are below for the fix as well as a screenshot of updated code.

Fixed code:

```
public void doPer(){
    String resultStr = null;
    try{
        double num1 = Double.parseDouble(view.textNum1.getText());
        double num2 = Double.parseDouble(view.textNum2.getText());
        //Pulled percentage method from calculator
        double result = calculator.per(num1, num2);

        resultStr = result + "";
    }
    catch(Exception e){
        resultStr = "Whole numbers please :)";
        view.update("","", resultStr);
    }
    String operatorIcon = "%";
    String equation = "= ";
    view.update(operatorIcon,equation, resultStr);


    }
```

```
//calculation for Percentage
public double per(double a, double b) {
    return (a / 100) * b;
}
```

Testing:

Test no.	Test	Expected result	Actual result

1	Try out percentage calculation	To display the correct percentage calculation in the output text box	Result is correct and displayed in the correct box.

What I could add

Feature 1: decimals

We can add decimals to the code by making all int's into doubles. This was tried before but unsuccessful but now that i have learned a bit more i can add it into an updated version later down the road. This can enable us to do more complex calculations and add more functions.

Feature 2: More functions

More functions could be added like figuring out angles and other trigonometric and finding cube root, xth root and all that. This could be useful in the future to make the calculator more useful to older audiences who are doing more complicated calculations in college/univeristy.

Feature 3: Update interface

The interface does need an update and as I learn more I can completely update the interface and add number buttons, make the layout more pleasing or similar to other calculator applications. The interface is important to update eventually as also updating it can maybe solve the bug that the History function created.

Conclusions

Predicted grade: D

Reasoning: this report and codebase have decent improvements but due to my novice skillset there is improvements that could be made and maybe better implementation of

the code and this report could use quite a bit of polishing. This is why i marked myself at the passing grade.

What i learned: During the development i was able to learn how to use catch statements, learn how to use labels, make the classes interact with each other, use doubles and use other statements and use Math. Calculations. This has been a good challenge and hopefully i have done a good execution. The lectures from university helped me groqw my skills in this as well.

During this, i haven't touched the Main class at all due to not needing to, this is why the version number 1.00 instead of 1.03.

References

[1] Reisinger, D. (2012) *Go figure: Google adds calculator to search results*, *CNET*. Available at: https://www.cnet.com/tech/services-and-software/go-figure-google-adds-calculator-to-search-results/ (Accessed: 23 January 2025).

[2] *W3schools.com* (no date) *JavaScript Math pow() Method*. Available: https://www.w3schools.com/jsref/jsref_pow.asp (Accessed: 23 January 2025).

[3] *W3schools.com* (no date a) *JavaScript Math sqrt() Method*. Available at: https://www.w3schools.com/jsref/jsref_sqrt.asp (Accessed: 23 January 2025).

[4] Oracle (2024) *IllegalArgumentException (Java Platform SE 8)*. Available at: https://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html (Accessed: 23 January 2025).

Lectures that helped me as a whole [estimated dates of access to these lectures.]:

Shine, S., (2024) Methods – week 5 of CI401 introduction to Programming etc. [online] available at: https://brighton.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=0bf5b90e-a682-4a4d-9680-b21a00c6229a&start=0] accessed 15th January 2025

Shine, S., (2024) Testing and JUnit– week 10 of CI401 introduction to Programming etc. [online] available at: https://brighton.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=5d8f9033-f46b-4f5f-b869-b23d00ca791d&start=0] accessed 15th January 2025

These are the two lectures that has helped me a ton. After that it was just general knowledge gained from labs and simple knowledge from college.

LSP Disclaimer

I confirm that I have a Learning Support Plan for adjusted deadlines and spelling and grammar as recommended by the Disability and Learning Support Team, and agreed by the School. I understand that this should be taken into consideration when my assessment is marked/ graded.