OOP Project Report

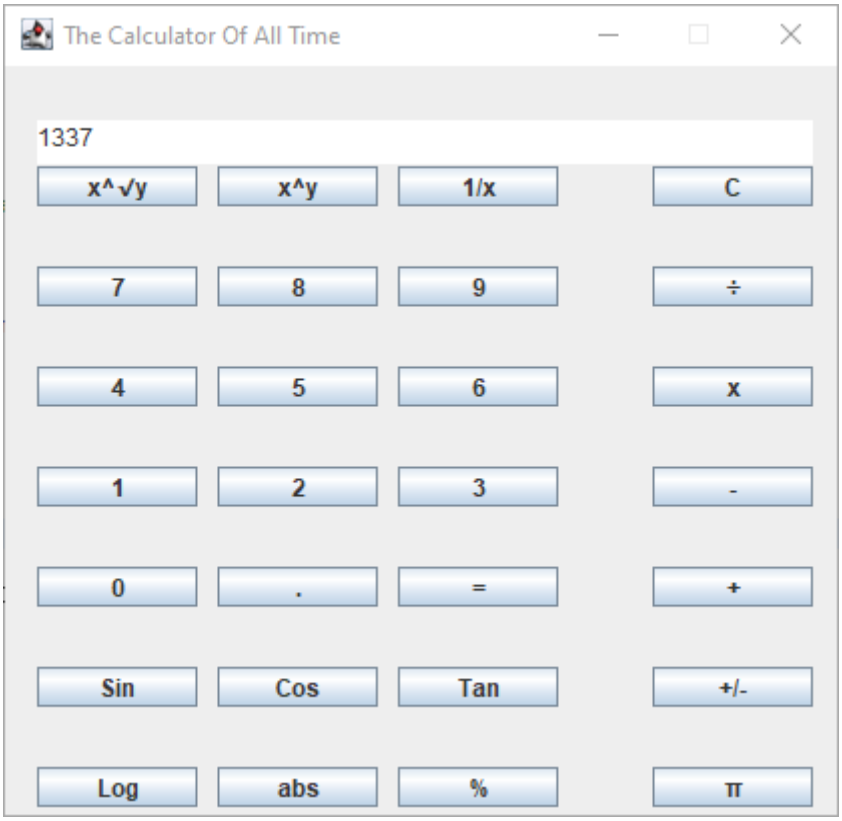


Table of Contents

Overview

This project was pretty enjoyable to do and also pretty challenging. I guess I could have made it a bit easier on myself by not doing a calculator but I just had my mind set on it from the beginning. So, because of that, I think it would be personally harder to do one of the other options since my mind would be on this one the whole time. Plus, this is the fourth of five projects that I am doing and, by the time of finishing this project, I should have nine days left for the last project. So, time wise, I'm doing well, even picking this harder option. I think I made some pretty cool decisions for how to implement some of the logic for this calculator. It was much more of an ordeal than doing a text-based calculator like we have done in the past.

I had some struggles but I got through them and I wrote many more methods than I ever expected I was going to do. It really felt like it was never ending for a time but it did make me learn a little bit more about a better way to go through the process of coding something (for me personally, anyway). I changed things up, doing all of the code and then doing the comments afterwards and I'm sure that would be a horrible idea for some. But, for me, it was really useful. I was forced to look at the code without actually coding, just writing comments. Because of that, I came up with more ideas to improve my code after I already finished it. I had all of the actual logic down and all of the changes I made didn't have any actual effect on the end user experience, but they made the code much more efficient. This method of writing code may not work for everyone but I'd say at least give it a try, there's no harm in it. So, with all of that said, onto the code itself.

The Code

```
JButton root = new JButton("x^\u221Ay");
root.setBounds(16, 50, 80, 20);
frame.getContentPane().add(root);

JButton plusMinus = new JButton("+/-");
plusMinus.setBounds(323, 300, 80, 20);
frame.getContentPane().add(plusMinus);

JButton sin = new JButton("Sin");
sin.setBounds(16, 300, 80, 20);
frame.getContentPane().add(sin);

JButton cos = new JButton("Cos");
cos.setBounds(106, 300, 80, 20);
frame.getContentPane().add(cos);

JButton tan = new JButton("Tan");
tan.setBounds(196, 300, 80, 20);
frame.getContentPane().add(tan);

JButton absolute = new JButton("abs");
absolute.setBounds(106, 350, 80, 20);
frame.getContentPane().add(absolute);

JButton x1 = new JButton("1/x");
x1.setBounds(196, 50, 80, 20);
frame.getContentPane().add(x1);

JButton log = new JButton("Log");
log.setBounds(16, 350, 80, 20);
frame.getContentPane().add(log);

seven.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        numbers.addButton("7", operations, display);
    }
});

eight.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        numbers.addButton("8", operations, display);
    }
});

nine.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        numbers.addButton("9", operations, display);
    }
});

four.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        numbers.addButton("4", operations, display);
    }
});
```

First is the main code file, WindowCalc. The image above is part of it but once you've seen part of it, you've seen all of it. There's no real logic in this class. This is all in a function named initialise which is run with a try and catch in the main method on startup. And, as you can see, there's not much to it. Just button instantiations, adding action listeners and method calls. It's a very good example of abstraction. Take the commonly used tv and tv remote example for example. You just press a button and something happens on your tv, you're likely not completely aware of how that works and what's going on in the tv or the remote. The same goes here, the user presses, say, the 7 button on the calculator. They press it and a 7 is displayed to a JTextArea on the calculator. But they don't know a function is being called which takes three different parameters, checks against them with if statements and methods before finally running another method to append text to the JTextArea. To the programmer it's complicated but the user knows nothing about it, that is encapsulation.

```
public void addButton(String i, OperationsManager operations, JTextArea display) {
    if (operations.getAddition() || operations.getDivision() || operations.getSubtraction() || operations.getMultiplication() || operations.getPower() || operations.getRoot()) {
        if(!operations.getCleared()) {
            display.setText("");
            operations.setCleared(true);
        }
        display.append(i);
    }
    else {
        display.append(i);
        operations.setCleared(false);
    }
}

 * Similar in concept to the addButton method, this has an added bit of logic to check if the JTextArea
public void addZero(OperationsManager operations, JTextArea display) {
    if (display.getText().equals("")) {

    }
    else {
        if (operations.getAddition() || operations.getDivision() || operations.getSubtraction() || operations.getMultiplication() || operations.getPower() || operations.getRoot()) {
            if(!operations.getCleared()) {
                display.setText("");
                operations.setCleared(true);
            }
            display.append("0");
        }
        else {
            display.append("0");
            operations.setCleared(false);
        }
    }
}

 * Similar in concept to the addZero method, this checks if there is already a decimal (period) in the
public void addDecimal(OperationsManager operations, JTextArea display) {
    if (display.getText().contains(".")) {

    }
    else {
        if (operations.getAddition() || operations.getDivision() || operations.getSubtraction() || operations.getMultiplication() || operations.getPower() || operations.getRoot()) {
            if(!operations.getCleared()) {
                display.setText("");
                operations.setCleared(true);
            }
            display.append(".");
        }
        else {
            display.append(".");
            operations.setCleared(false);
        }
    }
}
```

Next is the NumberManager class. It's kind of a lot so I'll try to explain it the best that I can. First things first, I understand that I could have used inheritance for this class (as well as the next I will be showing off) but there's four methods here and over ten in the other class so it gets to the point where that would be harder to manage rather than easier. Also, I understand there is some repetitive code but I prefer it this way for this method, there's only four methods so it's not *too* repetitive and in the other class I will present, I have shown my skills in not repeating myself so it's certainly not something I'm incapable of.

I just wanted to clear the air about that since this is for a project (and, since this is a project, I wanted to not spend all my time on it since I still have one left). Anyway, there are four methods (only three pictured) that each have their own purpose despite their similarities. The purpose of addButton is to add the functionalities of the numbered buttons 1-9 since they all work the same aside from the number they set to the JTextArea. It takes a string which is parsed as a double, OperationsManager which is used to check if an operator button has been pressed or if the JTextArea is cleared (used as a way to let the user input more than one number after an operator button has been pressed). addZero is the same but it checks if the JTextArea is empty since most calculators do not allow for a leading zero. addDecimal is the same as that but it checks if there's already a decimal in the JTextArea for obvious reasons. Finally, the unseen method is addPi which is similar to addDecimal but instead of doing nothing if there's a decimal already in the JTextArea, it adds a version of Pi with no decimal (3141592).

Honestly the most complicated part of this section was to add the functionality to check for different operator buttons being pressed and if the screen had already been cleared before. It was quite hard to get all of the pieces to properly move in sync so to speak but I got it up and running properly. That will especially be shown in the next class which I am just about to show to you now.

```
 * This is the logic for the root button, aside from what booleans are triggered, it's the same as
public void root(JTextArea display) {
    if (checkEmpty(display)) {

    }
    else if (temp != 0) {
        temp = Double.parseDouble(display.getText());
        applyAdd(display);
        applyDivide(display);
        applyMultiply(display);
        applySubtract(display);
        applyRoot(display);
        applyPower(display);
        r = true;
        a = false;
        m = false;
        s = false;
        d = false;
        p = false;
    }
    else {
        temp = Double.parseDouble(display.getText());
        r = true;
        a = false;
        m = false;
        s = false;
        d = false;
        p = false;
    }
}

 * Checks if the display is empty or has just a decimal (period), if not, sets the text in the JTextArea
public void percent(JTextArea display) {
    if (checkEmpty(display)) {

    }
    else {
        display.setText("" + Double.parseDouble(display.getText()) / 100);
    }
}

 * Checks if the display is empty or has just a decimal (period), if not, sets the text in the JTextArea
public void plusMinus(JTextArea display) {
    if (checkEmpty(display)) {

    }
    else {
        display.setText("" + Double.parseDouble(display.getText()) * -1);
    }
}
```

This one has a lot more going on and there's a whole bunch of methods but some are similar. First, there's 6 methods to return each of the booleans for the two-input operators (add, subtract, multiply, divide, power and root). You saw these methods earlier in the previous class and I explained their purpose so no need to do that again. The next set of methods are the ones for the operator buttons. If the display is empty, nothing happens. I could have done it without an empty if statement and just had temp !0 || !checkEmpty(display) instead but I did it a completely different way before so it was easier to change this way. It's a little bit verbose but it still works and I understand that it could be done a better way, I just opted not to so as to save myself some time.

If there is something in temp (aka, the user previously pressed a number button), the operator button is also treated similarly to an equals button, each of the apply methods I created for the 6 main operators. This does go against the open-closed solid principle a little bit but there should never be a need for more than these 6 operators anyway so it's fine. Also, I'm not sure of a natural place to fit this in so I'll say it here, encapsulation is used since these methods are public but the fields are private and some can't be changed directly. Also, the open-closed principle is followed everywhere where it's important in the code. I think that's everything I personally have to say on the SOLID and OOP principles for this. I didn't use every single one but it'd be hard to and, as mentioned earlier, using inheritance and the other principles that go in turn with it, everything would be harder to manage.

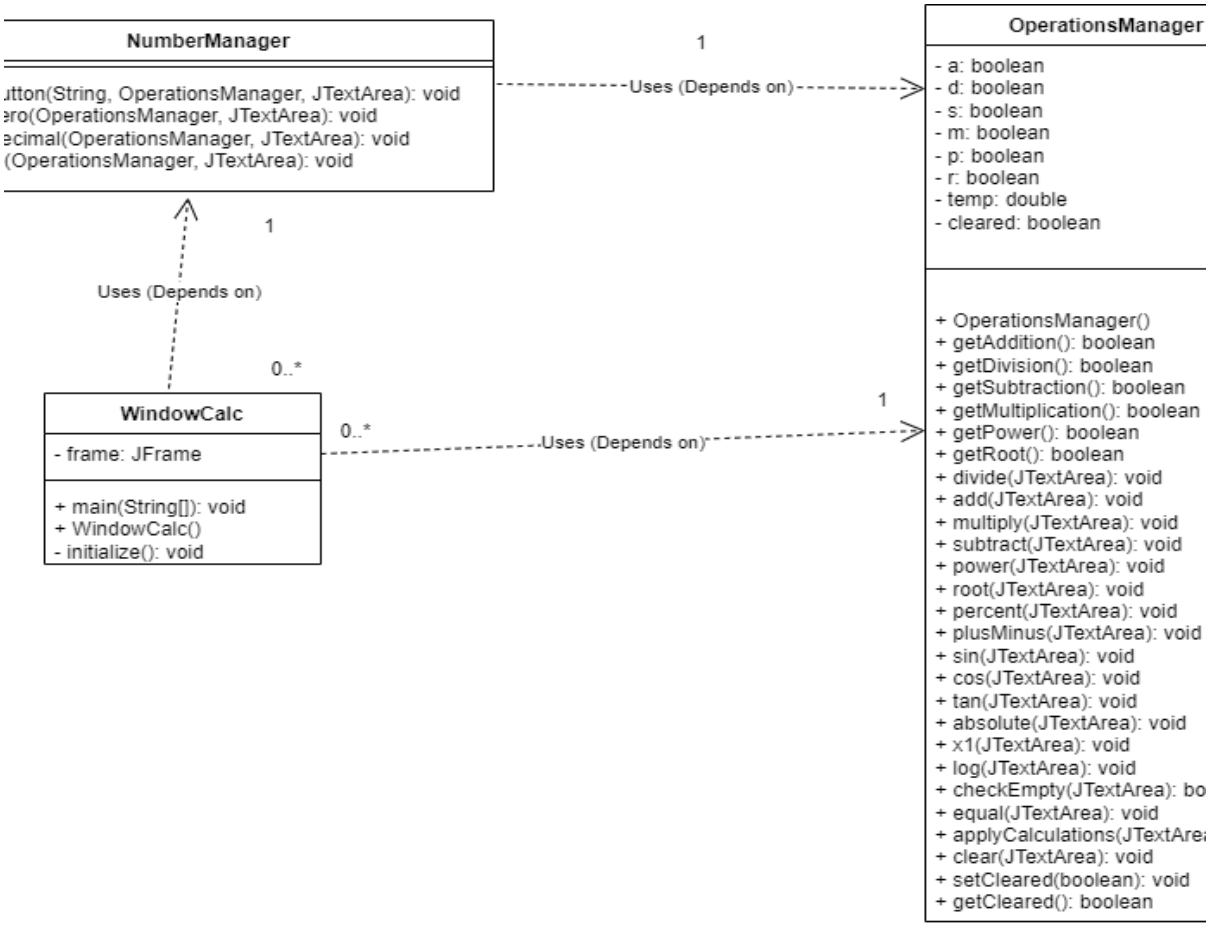
Back to the code, if there is nothing in temp, the same code runs, making the appropriate boolean true depending on the method but doesn't treat it like an equals button. The next group of methods are ones that take one input and change it in a way. For example, plusMinus multiplies the value currently in the JTextArea by -1. sin, cos, and tan run the appropriate Math methods, etc. checkEmpty returns true or false depending on the state of the JTextArea. equal changes all of the booleans to false and runs the methods for the operators. The apply methods apply their named operators. Clear sets temp to 0, makes all booleans false and sets the JTextArea's text to nothing. Finally, set and getCleared do as one would expect. So yeah, a lot of methods and maybe if I did this and had more time I would have split this class and used extensions and implementations but without more time and having to do other things to do, the works perfectly fine. Even with the code changed, the functionality would be the exact same. Either way, onto the (much shorter) Javadoc section.

Comments and Javadoc

Modifier and Type	Method	Description
void	absolute(JTextArea display)	Checks if the display is empty or has just a decimal (period), if not, sets the text in the JTextArea to what it was under the Math.abs method.
void	add(JTextArea display)	This is the logic for the add button, aside from what booleans are triggered, it's the same as the other 5, two-input style operators.
void	applyAdd(JTextArea display)	Applies the proper calculation to temp and what's currently in the JTextArea depending on what variable is true.
void	applyDivide(JTextArea display)	Applies the proper calculation to temp and what's currently in the JTextArea depending on what variable is true.
void	applyMultiply(JTextArea display)	Applies the proper calculation to temp and what's currently in the JTextArea depending on what variable is true.
void	applyPower(JTextArea display)	Applies the proper calculation to temp and what's currently in the JTextArea depending on what variable is true.
void	applyRoot(JTextArea display)	Applies the proper calculation to temp and what's currently in the JTextArea depending on what variable is true.
void	applySubtract(JTextArea display)	Applies the proper calculation to temp and what's currently in the JTextArea depending on what variable is true.
boolean	checkEmpty(JTextArea display)	Checks if a JTextArea is completely empty or only has a decimal (period) using the getText and equals methods and returns a corresponding true or false value.
void	clear(JTextArea display)	Sets every boolean to false, temp to zero and sets the text of the JTextArea to nothing; it's exactly as you would expect from a real calculator.
void	cos(JTextArea display)	Checks if the display is empty or has just a decimal (period), if not, sets the text in the JTextArea to what it was under the Math.cos method.
void	divide(JTextArea display)	This is the logic for the division button, aside from what booleans are triggered, it's the same as the other 5, two-input style operators.
void	equal(JTextArea display)	Runs the applyCalculations method and then sets every boolean in this class to false; it's exactly as you would expect from a standard calculator.
boolean	getAddition()	This returns the value of the a variable.
boolean	getCleared()	Returns the value of the cleared variable.
boolean	getDivision()	This returns the value of the d variable.
boolean	getMultiplication()	This returns the value of the m variable.
boolean	getPower()	This returns the value of the p variable.

Considering that I've already talked a lot about this project and, at the time I'm writing this, I've already completed the SAAD project, I've said more than enough about Javadoc. It's super neat how all you have to do is write comments and have an entire website generated about it is super neat. It's an interesting way to solve the problem of having documentation being unified. But even saying that much is still repeating myself. I just think that it's important enough to have a section on, however short that section may be. Javadoc is important and cool! Spread the word! Use Javadoc for everything! It will make your life easier and better!

Diagram



Another similarity to the SAAD project, this one also includes a diagram. The connections are also similar too. There's no standard connection like that diagram but since WindowCalc uses NumberManager and NumberManager uses OperationsManager. NumberManager only uses one OperationsManager and OperationsManager is only used by one NumberManager. WindowCalc can use more than one NumberManager but not the other way around and same goes for WindowCalc and OperationsManager. Not much else to say, the methods are there, the connections are there and the visibility is there. It's a simple yet effective diagram.

Conclusion

So, that’s everything. Even a simple calculator using java.swing elements can be an entire ordeal. It was certainly a learning experience, both from the using a new Java library perspective and from a general coding experience as well. It was pretty challenging but that made it enjoyable. Nothing’s good without a little challenge after all. I think I’ll take a screenshot of the other options for this project and do them in my spare time at some point. I may even redo this specific one again and see if I can do it better with the same functionality. To add to that, this project really made me realise that you can’t do everything in the way you might want to with the time you have. It’s better to get everything working than not, after all. Either way, to wrap up, this was enjoyable, perfectly challenging and I can see where there’s room to grow. From that point of view, I’d say this was a perfect project.