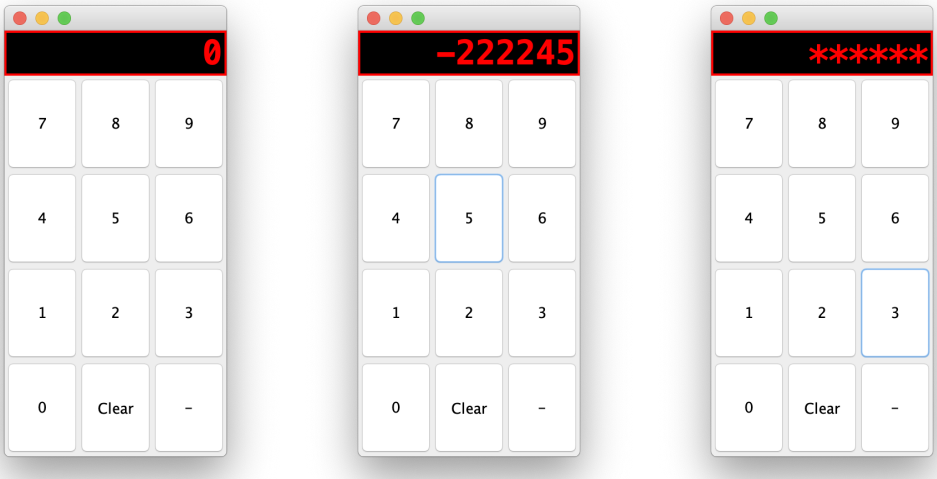

Figure 1: Calculator Screen shots

A Very Small Calculator

Due Date:

This assignment is due Friday 2022-03-25.

Purpose:

To become familiar with more components of hte the `javax.swing.*` architecture for creating GUI applications.

Goal

To create a very small "calculator" that looks like those shown in Figure **??** The initial calculator should look like the leftmost image. Pressing various buttons should normally produce an image like the middle one; too many digits should result in a row of stars, as shown on the right.

More precisely:

- Clicking Clear should reset the calculator to the leftmost image.

- Tapping the display screen with the mouse should also clear the calculator.
- Clicking a digit button normally adds that digit to the right end of the number. There are three exceptions.
- Clicking a `0` when the display is just `0` or `-0` should have no effect.
- When the display is "full" clicking any digit causes the display to go into overflow mode where only `*******`s are displayed.
- When the display has overflowed in `*******`s the only button that is effective is the clear button.
- Except when the display is overfull, clicking the − button should toggle the presence or absence of a leading - digit.

Method

The overall display is a `JFrame` or `JPanel` using `BorderLayout`. The display part is in the north component, the buttons are in the center

Display

The display part of the calculator should be a `javax.swing.JTextField`. Attributes that you likely want to set include

- the foreground and background colours,
- the font (the screenshot are from a program that uses

```
display.setFont(new Font("Menlo", Font.BOLD,30)) ;
```

  Feel free to pick your own font.)
- `.setEditable(false)`
- the left-right alignment (`.setHorizontalAlignment(JTextField.TRAILING)`)

Buttons

The individual buttons are `JButtons`. They are arraged in a `JPanel` with `GridLayout`.

Model

Building a separate model class `Register` (or your favourite name) is strongly recommended. The model might have public methods like those shown in Figure **??**. The behaviours of the model are designed to be convenient to attach to the various buttons.

The `getDisplayText()` method should return a `String` appropriate for use with the display's `setText(...)` method.

```
1     public boolean isNegated() ;
2     public int getDisplayDigitsMax(); max digits before stars appear
3     public String getDisplayText() ; // what to display in the text field
4
5     public void negate() ; // change the register's sign
6     public void setDisplayDigitsMax(int dMax) ;
7     public void clearDisplay();
8
9     public void addDigit(int k) ;
10    public void addDigit(char k) ;
```

Figure 2: Methods for a model class

You may want to add mthods to the `Register` class to help link a `Register` with the `JTextField` that is its view. This drags a bit on the cohesiveness and coupling of the `Register` class, but is simple to implement.

Linking

The buttons each need to be linked to something. The simplest approach is to use an `addActionListener` and a λ-expression. This might look something like

```
clearButton.addActionListener((ae)-> myRegister.clearDisplay()) ;
```

for each of the buttons.

The display needs to have a `MouseListener` attached so that clicking on the display causes the model object to clear.

Finally the model needs to be linked to the display `JTextField`, so that changes in the model cause the display to be updated.

Submission

Submit Lab 6 the same way that you submitted Lab 5.