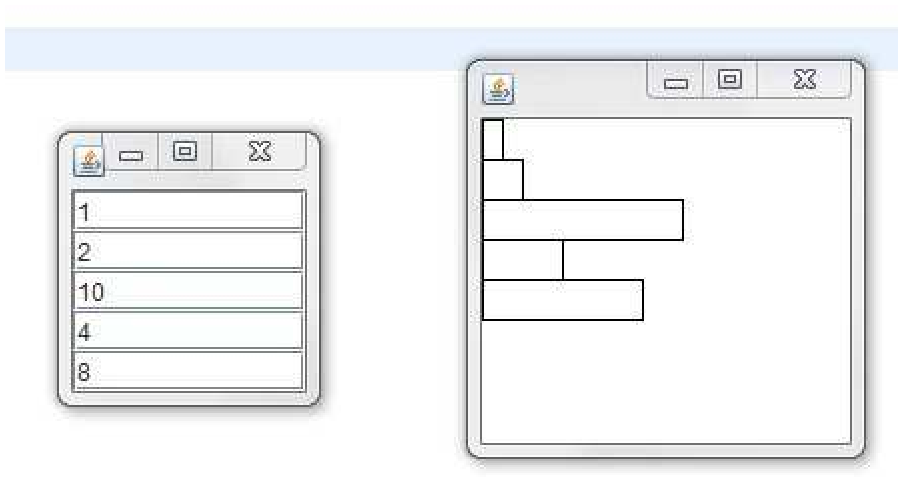
Lab 3 — Observer Pattern

CS 205

Lab objectives:

- Implement the observer pattern within a Model/View/Controller architecture.
- Make use of the Observable class and the Observer interface.

You will implement the classical example of the observer pattern — a data set and two views, one text view and one graphics view.



1
2
10
4
8

The data model consists of an array of integers. For simplicity, you may assume that the integers are in a range that is suitable for plotting in the graphical view. I used the range $[0, 20]$ and set the graphical view's content pane to a width of 200 and a height of 20 times the array's length. The text view is a collection of `JTextField` components. Editing a text field updates the model and hence the other view. The graphical view is a simple bar chart. Clicking anywhere next to a bar moves the bar to the mouse position.

You *must* use the Observer interface and the Observable class as follows:

```
public class DataModel extends Observable
{
    . . .
} // DataModel class

public class TextView extends Box implements Observer
{
    public TextView(DataModel aModel)
    {
        super(BoxLayout.Y_AXIS);
        . . .
    } // TextView constructor
    . . .
} // TextView class

public class GraphView extends JPanel implements Observer, MouseListener
{
    public void paintComponent(Graphics g)
    {
        super.setBackground(Color.white);
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        . . .
    } // paintComponent
    . . .
} // GraphView class
```

Hints:

1. Your main method will create an array of integers which can be either read-in or hard-coded. You can place the main method in the `DataModel` class. You will create `DataModel`, `TextView`, and `GraphView` objects and register listeners and observers. Make two `JFrame` objects. In the first frame, set its content pane to the text view and pack it. In the second frame, set the content pane to the graph view. Use `setPreferredSize` to set the graph's content pane to an appropriate size. You can use `setLocation` to set each window's location.

2. The `Box` class takes care of packing the `JTextField` components vertically. Just use the `add` method.

3. Each `JTextField` object should have an `actionListener` which will take the appropriate action when the text value is changed. Implement the `actionListener`s with a lambda expression.

4. The graph view should have a `MouseListener` which takes the appropriate action when the mouse is clicked. You can have empty methods for the other mouse actions.

5. The `DataModel` class will need to use the `setChanged` and `notifyObservers` methods inherited from the `Observable` class after updating a data value.

6. Give each bar in the graph view a fixed height. I used

   ```
   public static final int HEIGHT = 20;
   public static final int SCALE = 10;
   ```

   Then when a user clicks on the point `(x,y)`, the graphical view can tell the model to set the data value with index `y / HEIGHT` to `x / SCALE`. You can find out which point was clicked from the `MouseEvent`. When re-drawing the bars, multiply each data value by `SCALE` to obtain the bar's width.

Add Javadoc comments to document the code that you write for the lab. Export your lab into a ZIP archive and submit it in Canvas.