Homework #2 - Part 2

Diljot Singh

Connect with me! https://www.linkedin.com/in/DiljotSinghSJSU/

Starting Off

We're told that we need to create four classes:

1. DataModel.java
2. GraphView.java
3. TextView.java
4. Hw2P2.java

1. **DataModel.java**
   - This class is the Model in the MVC Structure, meaning it stores all the data and notifies its Observers (the views, in this case GraphView and TextView) whenever that data is modified.
   - For this assignment, the data we are storing are integers - specifically a collection of them. The homework mentions using an Array to store the integers, but I just used an ArrayList to simplify the code and not worry about resizing/deletion.

- This means that the DataModel should have an ArrayList or an Array instance variable of type Integer that we can add integer elements to and modify.
- Whenever an element is added to the list or an existing element is modified, the homework mentions we should also call setChanged() and notifyObservers() to let the Observers (TextView and GraphView) know that the model has changed. What this does, as we'll see later, is just trigger the update() method that we will define in the TextView and GraphView classes.
- One of the methods that I defined in my DataModel class was "public void addValue(Integer value). What this method does is add the parameter value to the list, call setChanged(), then call notifyObservers(). The setChanged() and notifyObservers() methods are part of the Observable class, so since DataModel extends Observable, we do not need to define them ourselves — only call them.
- I also defined a method for overwriting an existing element (using list.set(int index, int value) for ArrayLists) and a getter for the list of data, but I'll leave this for you. The main thing to remember is to call setChanged() and notifyObservers() whenever we add something to the model or modify something inside it.

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

2. GraphView.java

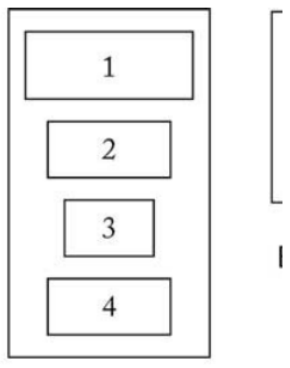
- This class is one of the Observers (views), in the MVC structure. This class represents the integers in the DataModel as rectangles whose widths depend on the corresponding integer values.
- In GraphView's constructor, I take in a DataModel object as a parameter, and save it to an instance variable.

- In terms of drawing all the rectangles in the paintComponent() method, we can loop through all the Integers in the DataModel instance variable's list and create a Rectangle based on each one.
- Remember that one of the Rectangle class's constructors lets us take in four parameters: Rectangle r = new Rectangle(int xCoordinate, int yCoordinate, int width, int height).
- In our case, the xCoordinate will always be 0. The yCoordinate should get incremented by each previous Rectangle's height in each loop iteration, which the homework says is always 20. So that way we can have one rectangle at (0, 0, width, height), then a second one at (0, 20, width, height), then a third one at (0, 40, width, height), and so on. The width of the Rectangle depends on the current Integer value in the for loop. Once we've constructed a Rectangle inside the for loop, don't forget to draw it (example: g2.draw(rectangle)).
- Since the GraphView implements the Observer interface, we have to define the update() method. In the update() method, all we're outlining is how the current view should update itself whenever the model changes. Since in our paintComponent() method we drew all the rectangles based on the Integer values, what do you think we should do when the model changes? (Hint: There's a method that we can call that rhymes with "re-faint")

```
public class GraphView extends JPanel implements Observer
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
      . . .
   }
   . . .
}
```

3. TextView

- This class is another Observer (view), in the MVC structure. This class represents the integers in the DataModel in terms of JTextField components.
- It extends the "Box" container which automatically uses a BoxLayout to format all of its components. We define the format of the BoxLayout components to be vertically aligned when we have "super(BoxLayout.Y_AXIS)" in the constructor. What this means is that all the components (in this case JTextFields) that we add to this class will automatically be formatted like this:



BoxLayout (vertical)

- In TextView's constructor, I also take in a DataModel object as a parameter, and save it to an instance variable.
- Again, this class implements the Observer interface, so we have to define what its update() method should do.
- In this case, what we can do is update every text field in the TextView whose current text doesn't match the integer stored in the model. We can accomplish this by looping through all the Components in the TextView (for Component comp : this.getComponents()) and start updating the text fields accordingly. You may need to cast the component as a "JTextField" to access the methods such as getText() and setText().
- **Note:** Remember that each TextField has a yCoordinate and a height. Let's assume for example that each text field's height is 10 pixels. Then the first text field might have parameters (0, 0, 10, width), the second would have (0, 10, 10,

width), the third would have (0, 20, 10, width) and so on (since each Textfield is right below the previous). The first text field (0, 0, 10, width) would display the first element in the DataModel's list of data (index 0), the second text field would display the second element in the list (index 1), the third text field would display the element at index 2, and so on.

- What we can do is divide the TextField's yCoordinate by the TextField's height to get the corresponding integer's index in the list. Then we can get the value at that index and see if it matches the text stored in the current text field. If it does not, we can set the text field's text to that integer.
- **Important:** Before looping through all the components, I made sure that the model's list size was less than or equal to the number of components in the TextView. This helps us avoid some runtime exceptions where not all the corresponding components have been added yet and we're trying to access something that doesn't exist.

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

---

4. Hw2P2.java

The implementation of Hw2P2.java has different parts to it, so if you're stuck on implementing a specific portion of the class, you can just scroll to the appropriate section:

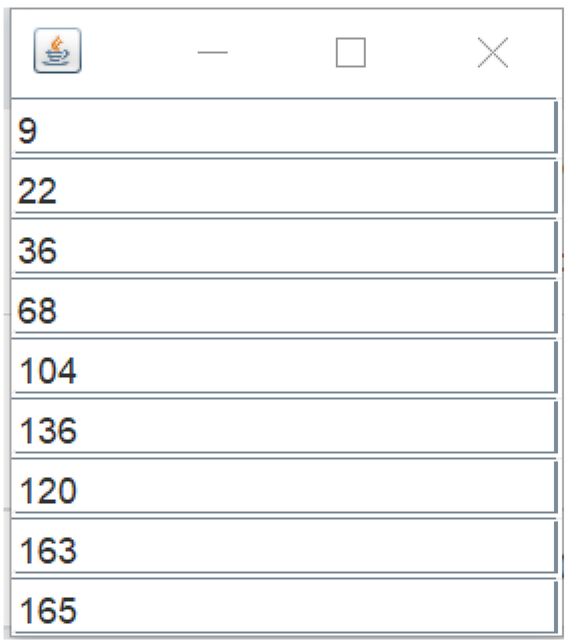
Instantiating the classes and setting up the MVC Structure

- First we're going to create a DataModel object, let's call it "model"
- Then we're going to create a TextView object and pass in the model in the constructor's parameter
- Same for GraphView: create an instance of the object and pass in the model
- **Important:** Don't forget to add the two views as Observers to the model. This function is already built into DataModel since it extends Observable, so we can just do something like "model.addObserver(name_of_text_view_object)" and same for GraphView

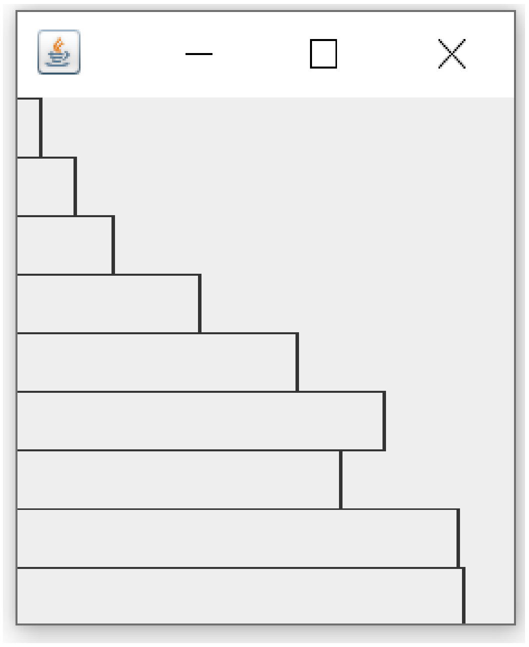
Creating the two frames

- We can just create two JFrame objects and as the homework mentions, set each frame's content pane (frame.setContentPane(view_object)) to the corresponding view
- At the bottom of the main method, don't forget to add the standard operations to the frames and the window listeners to save the data when any frame is closed:
    - For the text view frame:
        - Pack the frame
        - Make it visible
        - Add a window listener to write all the current data from the model back into the model (we will talk about this later)
    - For the graph view frame:
        - Make it visible
        - Set some bounds for it so that it doesn't overlap the text view frame
        - Add a window listener to write all the current data from the model back into the model (we will talk about this later)

What the two frames will look like:

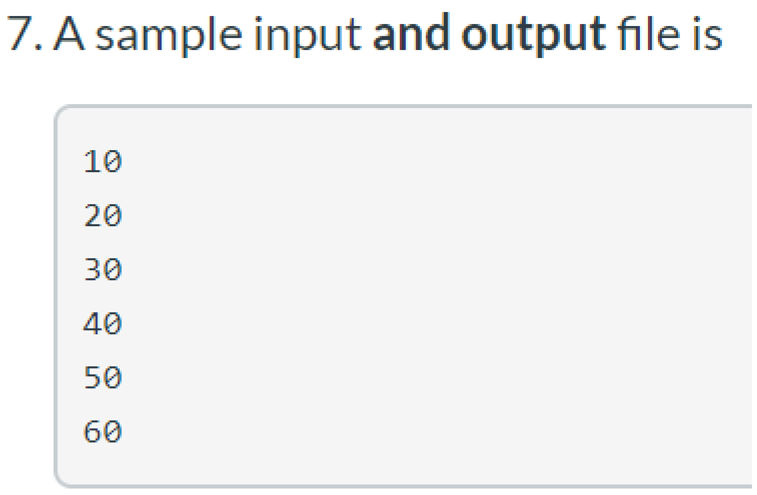


```
9
22
36
68
104
136
120
163
165
```



Parsing Data from the File

- We'll need a Buffered Reader or a Scanner to read lines from the given File name in args[0] and add each integer to our model's list of integers
- It's mentioned that the File's structure will look like this diagram below, with an integer on each line:



7. A sample input **and output** file is

```
10
20
30
40
50
60
```

- For submitting the assignment, we can accomplish this by creating the reader like this: BufferedReader br = new BufferedReader(new FileReader(new File(args[0])));

- But for the sake of testing and making sure our program works, we can create a text file in your project directory and put in some numbers to test your program, so the reader might instead look like this: BufferedReader br = new BufferedReader(new FileReader(new File("filename.txt")))
- Inside the while loop of parsing the data, we can add every number to the model using the "addValue" method we mentioned earlier (so that the views are notified) and create a JTextField to display the current number. <u>Don't forget to add each text field we create to the text view</u>

<u>DocumentListeners for JTextFields</u>

- To track the editing of each text field, we can assign them a document listener inside the while loop when we're parsing the data and creating the text fields
- There are three methods we'll need to implement:
  - **changedUpdate()**: this one does not get triggered much but it's for when the attributes of the text field have changed (such as the height or width, I believe)
  - **insertUpdate()**: gets triggered when something is added to the text field, such as typing a digit
  - **removeUpdate()**: gets triggered when something is removed from the text field (such as backspacing a digit)
- In each of these three methods, we do a similar task:
  - Calculate the index of the element the text field is showing by dividing the text field's <u>y position</u> by its <u>height</u> (similar to TextView's update() method)
  - Get the current value the text field is showing by getting the text
  - Use a try statement to set the element at <u>index</u> in the model's list to the <u>current value</u> of the text field (make sure this calls setChanged() and notifyObservers() in the DataModel method you use)
  - **For the removeUpdate() method, there is a slight modification to this:**
    - We still calculate the index as normal but in this case, we also have to consider the possibility that the user could backspace all the digits in a text field, leaving it empty. In this case, what should the new value be?

        - Otherwise if the text field still contains some digits, we can just get the text as normal
        - __Note:__ **Instead of catching a NumberFormatException as in the other two methods, we need to catch a different type of exception in this method for when the user erases a field completely. You'll see the type of exception in the console when testing erasing / adding digits**

__Adding a Mouse Listener to GraphView__

- For this feature, we want a bar of data to update to our mouse's X position when we click next to it
- We calculate the index in a similar fashion as in the document listeners (but instead of getting the y coordinate of a text field, what are you getting the y coordinate of?)
- What should the new value of the element at the index be set to?

__Loading Data back into the File__

- Similar to parsing the data from the original file, we now want to write all the data from the model's list back into the file whenever the user closes any frame
- This would go inside each frame's window listener
- Don't forget to write the data in the same format as you parsed it: one integer on each line