CS 2300 Lab MVC GUI – Part 5 Week 12

Size Panels

Additional Tasks

This lab develops classes to enable entering text and converting it to numbers to define the size and shape of the object in the view panel.

1. Do some preliminary reading on topics for today's lab: `JTextField`, `Integer.parse`, `JOptionPane`, `showMessageDialogue`, and throwing exceptions.

2. Continue on with the project from the last lab. This should closely parallel the classroom development.

3. Background.

3.1: **JTextField**. The `JTextField` control allows a `String` to be obtained from the user.

3.2: **Parse**. Converting a string into a particular data type (such as an `int` or a `double`) is called **parsing** the string. This is actually a very complicated task. Fortunately, the **wrapper classes** for primitive data types (such as **`Integer`** and **`Double`**) have parsing methods written by experts. For example, the `Integer` class has a method `parseInt` to do the job of converting a string into an `int`. The hard part is dealing with everything that can go wrong.

3.3: **Exceptions**. There is a potential problem whenever parsing a string into a number. If the `String` is something like `12xy8z`, it does not represent any number. When a string does not represent an integer, the `parse` method **throws an exception**. Thus we have to be prepared to deal with this possibility.

3.4: **Try / Catch Block**. You can deal with methods that might throw an exception by putting them inside a **try / catch block**. This allows for code to deal with the bad event to be used. Throwing exceptions is a major strategy for dealing with "bad events" in modern programming languages such as Java and C++.

3.5: **JOptionPane**. The `JOptionPane` control has several methods for alerting users that something needs attention. We will use the `showMessageDialogue` method to tell the user that a bad string has been entered for a number.

4. It would be nice if the `JTextField` class had a method called something like `getInt` that would do all the work; but it does not. Furthermore, we cannot add one, because Java follows certain OOP principals, one of which is the **open / closed principal**. This means classes are **OPEN for extension**, but **CLOSED for modification**. The bad news (because the class is "closed for modification") is that we cannot modify the

`JTextField` class. The good news (because the class is "open for extension") is that we can extend the class by creating a subclass with an additional method.

We will **extend** the `JTextField` class to `JTextFieldInt` by adding a `getInt` method.

4.1. Add the following class to the project:

```
public final class JTextFieldInt extends JTextField { … }
```

4.2. Have NetBeans generate a constructor. Choose the third option (an `int` argument). It should look like this:

```
public JTextFieldInt(int columns) {
    super(columns);
}
```

Be sure this is what you get. If you clicked the wrong box, you will get something else, and you better fix it now, or you will be confused later.

4.3. The `JTextFieldInt` class already can do everything that the `JTextField` class can, because it is a child class. We will extend the class by adding the following function (right after the c-tor):

```
public int getInt() {
    String newTFString;
    int newTFInt = 0;                                   // default value
    try {
        newTFString = getText();                        // String entered in TF
        newTFInt = Integer
                .parseInt(newTFString);                 // parse txt to int
        System.out.println("good new int " + newTFInt);
    } catch (NumberFormatException                      // deal with bad number
            | NullPointerException nfe) {               // deal with WTF?
        setText("BAD INPUT!");                          // replace bad input
        JOptionPane.showMessageDialog(                  // popup GUI msg
                null,
```

```
                "Invalid input. Not a number.",    // error message
                "CS 2300 Error Message:",          // error title
                JOptionPane.ERROR_MESSAGE);        // error symbol
        System.out.println("bad new int " + newTFInt);
    } catch (Exception ex){                         // check for bad WTF?
        System.out.println("Bad error. Should not occur");
    }
    return newTFInt;
}
```

A lot of interesting stuff is going on in this method that will be discussed in classroom discussion. The `JTextFieldInt` class will be used below.

5. We have already created `PnlSizeX` class like this:

```
public final class PnlSizeX extends PnlAbsCtrl {
    public PnlSizeX(Model model) {
        super(model);
    }
}
```

Be sure this is right.

5.1. Add the following fields to this class (at the top):

```
    private final JLabel jlXSize = new JLabel("X Size:");
    private final JLabel jlSpacer = new JLabel("          ");
    private final int JTFI_SIZE = 6;
    private final JTextFieldInt jtfiXSize
            = new JTextFieldInt(JTFI_SIZE);
```

5.2. Add the following statements to the constructor:

```
        super(model);
        jtfiXSize.setText("" + model.getxSize());
```

```
        jtfiXSize.addActionListener(ae -> update());
        add(jlXSize);
        add(jlSpacer);
        add(jtfiXSize);
```

5.3. This will cause an error until we implement the `update` method. Write this method:

```
    private void update(){
        int newXSize  = jtfiXSize.getInt();       // use new class
        if (newXSize == 0){                       // bad input
            jtfiXSize.setText("" + model.getxSize());  // reset number
        } else {
            model.setxSize(newXSize);              // update x size
            model.getView().repaint();             // show changes
        }
    }
```

We have already added the `PnlSizeX` object to the controller object, so things should be ready to go! Run the program to be sure everything is working: you can change the size with good input, and you get the popup for bad input.

Test everything: Run the program. Notice there is now an box to enter a new `X Size` in. Type in a new number, maybe 444. The view should be redrawn with the new size.

Next type in something like `XXXYYYZZZ` that is not a number. This should cause a pop up to tell you that this is not a number. Try again with a good number.

When everything is right, continue onto the next part.

6. Now make similar changes to the `PnlSizeY` class. Be **SURE** to change each occurrence of **x** with **y**.

7. Run the program and be sure you can enter new X and Y sizes, and they change the picture.

8. Go to the `View` class. In the `paintComponent` method, go to the `switch` statement.

8.1 Right after this code:

```
            case ELLIPSE:
```

```
                paintEllipse(g);
                break;
```

add the same thing, but with `ELLIPSE` replaced by `RECTANGLE`, and also `paintRectangle(g)`.

8.2. Add the `paintRectangle` method. It is exactly the same as the `paintEllipse` method, except it has `Rect` instead of `Oval` in two places.

8.3. Run the program and see if you can choose rectangle, and it paints one. (That was pretty easy!)

9. Go to the `View` class `paintComponent` method `switch` statement again.

9.1. Add another case:

```
            case INFORMATION:
                paintStudentInfo(g);
                break;
```

9.2. Create the `paintStudentInfo` method:

9.3. Put the following data **right before** this new method:

```
private final String studentTitle = "Programming Wizard";      // pick
favorite title
private final String studentName = "Your name here";           // use
actual name
private final String studentAffiliation = "Ohio University";
private final int MIN_LENGTH = 4;
private final double FONT_FACTOR = 0.3;
private final int INFO_X_START = 100;
private final double Y_FRACTION_1 = 0.3;
private final double Y_FRACTION_2 = 0.5;
private final double Y_FRACTION_3 = 0.7;
```

9.4. Use this code **for the method**:

```
private void paintStudentInfo(Graphics g) {
int maxStringLength = Math.max(studentTitle.length(),
     studentName.length());
    maxStringLength = Math.max(maxStringLength,
          studentAffiliation.length());
    maxStringLength = Math.min(maxStringLength, MIN_LENGTH);
    int fontSize = (int) (FONT_FACTOR * getWidth() / maxStringLength);
    g.setFont(new Font("Courier", Font.PLAIN, fontSize));
    g.setColor(model.getColor());
    g.drawString(studentTitle, INFO_X_START,
          (int) (Y_FRACTION_1 * getHeight()));
    g.drawString(studentName, INFO_X_START,
          (int) (Y_FRACTION_2 * getHeight()));
    g.drawString(studentAffiliation, INFO_X_START,
          (int) (Y_FRACTION_3 * getHeight()));
```

9.9. Run the program and change the task to Information. See what happens!

9.5. Toy around by changing the data in 9.3 and see how it affects the information.

How to turn in Project. This is slightly complicated, so follow these instructions for submitting exactly:

1. Open file explorer
2. Find your project (likely C:\Users\<user>\IdeaProjects\MVC22F)
3. Right click on your project folder (this should contain src/mvc22f with all your .java files)
4. Select **Send to > Compressed (zipped) folder**
5. Submit MVC22F.zip to Blackboard