CS 2300 Lab Week 12

Basic MVC GUI – Part 4

Size Panels

The goal is to write a set of basic classes implementing a minimal MVC (Model View Controller) GUI (Graphical User Interface). You can use this project as a template for full featured GUI projects.

1. Do some preliminary reading on topics for today's lab; `ActionListeners`, `TextField`, `Integer.parse`, `JOptionPane` and `showMessageDialogue`.

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

3. Now we will add some stuff so we can adjust the size of the object we are working with. Go to the `SizeXPanel` class and do the following.

3.1. Below the c-tor put in this temporary method:

```
private void updateXSize() {       // TEMP STUB METHOD!!

    System.out.println("TEMP METHOD!");

}
```

This will prevent an annoying error message from coming up. We will fill out this method presently.

3.2. At the top of the class, above the c-tor, add the fields

```
private final JLabel jlXSize = new JLabel("Set X Size :  ");

private final int TEXTFIELD_SIZE = 10;       // adjust as needed or desired

private final JTextField jtfXSize = new JTextField(TEXTFIELD_SIZE);
```

Fix imports to make the error messages go away.

4. A key part of GUI programming is using **ActionListener**s. Through Java 7, this required a lot of "boilerplate code" (code that is almost exactly the same each time). Java 8 has added **functional programming** concepts. The key functional programming idea is a **lambda expression**. The two approaches will be shown next.

Inside the c-tor, add a statement to update the text on the textfield:

```
jtfXSize.setText("" + model.getxSize());
```

5. Next we have to define what will happen to any text that gets written into the textfield. Most Java Swing controls do this by associating an **action listener** with the control. This has just gotten a lot easier (in Java 8 and 9) by using "lambda expressions", as we will do next. Add this statement after the setText statement:

```
jtfXSize.addActionListener(ae -> updateXSize());
```

This basically says: If anything "fires" this control, do what ever is in the `updateXSize` method, which we will soon write. Finally, finish the c-tor by "adding" the label and textfield to the panel with these statements:

```
        add(jlXSize);

        add(jtfXSize);
```

Run the program and notice that the second control panel (on the left edge) now has something in it. Change the 66 to some other number, and hit enter. Notice that the output window now shows the output: `TEMP METHOD!`. That is what we put in the `updateXSize` method to remind us to finish the method.

6. The last step is to read in the input, translate it to a number, and update the Size value in the model. We can do this with the `Integer.parse` method. (This is a **static** method called `parse` of the `Integer` class).

Here is a complication: The entered number might be something like `5WS9-ww`, not an actual number, in which case the `parse` method will **throw an exception**. To deal with this possibility, we will put the entire thing inside a **try block**. A try block is a construction that surrounds some code that might throw an exception. If an exception is thrown, we can catch it and attempt to deal with is. If an exception is not caught, then the exception is thrown to the method that called the method you are working on where it might be caught. Likewise, if the calling method does not catch the exception, it is **rethrown** up another level. If the exception gets repeatedly rethrown all the way to the top (the `main` method that started the program), the program exits and prints out a bunch of info about the path the exception took to the top. This is to help the programmer figure out what happened.

In this case, we can deal with the exception by asking the user to try again. Replace the "stub" method with this:

```
    void updateXSize(){

        String stringNewSize = jtfXSize.getText();     // get the string entered

        try {

            int newXSize = Integer.parseInt(stringNewSize);       // parse to int

            model.setxSize(newXSize);                             // update model

            model.getView().repaint();                           // draw again

        } catch (Exception e) {                                  // deal with WTF

            jtfXSize.setText("BAD INPUT!");                       // msg to jtf

            JOptionPane.showMessageDialog(                       // GUI msg

                    null,

                    "Invalid input. Not a number.",               // error message

                    "CS 2300 Error Message:",                     // error title

                    JOptionPane.ERROR_MESSAGE);                   // error symbol

           jtfXSize.setText("" + model.getXSize());               // msg to jtf

        }
    }
```

7. Here is what is going on:

7.1 First off we capture whatever was entered into the `JTextField` into a `String` called `stringNewSize`.

7.2. Converting a string (which is a sequence of characters) into something else is called **parsing** the string. To parse a string into a primitive data type, such as an `int`, we use the parse method of the wrapper class (because the primitive data types themselves are not classes, so they don't have any methods.). In this case, we can use the `Integer.parseInt()` method to translate the `String` (called `stringNewSize`) into an `Integer`, which will automatically translate itself into the `int` called `newXSize`.

But, we must be prepared in case some nimrod entered bad input. For this we use a **try block**. A try block allows you to try to do something that may or may not work, and recover nicely if it does not work.

7.3. If the string is successfully translated into an `int`, then the `setXSize` method uses this `int` to update the model. Then we call `model.getView().repaint();` to let the JVM know that it is time to repaint the view with the new data.

7.4. But, if the string cannot be translated into an integer, an **exception** will be **thrown**, and "control" will immediately jump to the **catch block** at the bottom.

7.5. We will only get into the catch block if the input was bad. Here we will write an error message to the output, and also pop a **Message Dialogue** on top of the monitor. It will stay there until the user clicks "OK".

8. Run the program. Enter some smaller and larger numbers.

9. Try entering a bad number, perhaps something like "kkjkjkj". Note the pop up message box, and how the stuff used to define it shows up. The `JOptionPane` is a cool way to inform users of various stuff.

10. Go to the `SizeYPanel` class and write the code following the pattern used for the `SizeXPanel` class. It should be exactly the same except for changing some `X`'s into `Y`'s. You can either rewrite it from scratch, or use the copy – paste – change method. Run the program and be both the X and Y size controls work.

11. Show the lab instructor that you can change the size with good input, and you get the popup for bad input.