Java Class 17

LA Air Traffic Control

On September 14th, 2004, the LA air traffic control center suddenly lost voice control with 400 planes that it was tracking. The backup system crashed a minute after it was activated. Officially the incident was blamed on human error, but the root cause was traced to a software problem that had to do with an internal timer that determined when tests would be run. The timer was allowed to run down to zero, which then shut the system down to run the tests.

AND THE Lassie awards GO TO …

GUI Design

We must remember that the goal of software is to help the user solve the problem

To that end, the GUI designer should:

- Know the user
- Prevent user errors
- Optimize user abilities
- Be consistent

Each of these will be looked out in more detail

Know the User

Knowing the user implies an understanding of:

- the user's true needs
- the user's common activities
- the user's level of expertise in the problem domain and in computer processing

We should also realize these issues may differ for different users

Remember, to the user, the interface is the program

Prevent User Errors

Whenever possible, we should design user interfaces that minimize possible user mistakes

We should choose the best GUI components for each task

For example, in a situation where there are only a few valid options, using a menu or radio buttons would be better than an open text field

Error messages should guide the user appropriately

Optimize User Abilities

Not all users are alike – some may be more familiar with the system than others

Knowledgeable users are sometimes called *power users*

We should provide multiple ways to accomplish a task whenever reasonable

- "wizards" to walk a user through a process
- short cuts for power users

Help facilities should be available but not intrusive

Be Consistent

Consistency is important – users get used to things appearing and working in certain ways

Colors should be used consistently to indicate similar types of information or processing

Screen layout should be consistent from one part of a system to another

For example, error messages should appear in consistent locations

Mouse Events

JavaFX nodes can generate several types of mouse-based events:

Event	Description
mouse pressed	mouse button is pressed
mouse released	mouse button is released
mouse clicked	mouse button is pressed and released
mouse entered	mouse pointer is moved onto a node
mouse exited	mouse is moved off of a node
mouse moved	mouse is moved
mouse dragged	mouse is moved while holding the mouse button down

- The `MouseEvent` object representing the event can be used to obtain the mouse position
- There are convenience methods for setting the handler for each type of mouse event (such as `setOnMousePressed`)

```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.stage.Stage;

//************************************************************************
//  ClickDistance.java       Author: Lewis/Loftus
//
//  Demonstrates the handling of a mouse click event.
//************************************************************************

public class ClickDistance extends Application
{
    private Line line;
    private Text distanceText;

    //--------------------------------------------------------------------
    //  Shows the distance between the origin (0, 0) and the point where
    //  the mouse is clicked.
    //--------------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        line = new Line(0, 0, 0, 0);
        distanceText = new Text(150, 30, "Distance:  --");
```

continue

continue

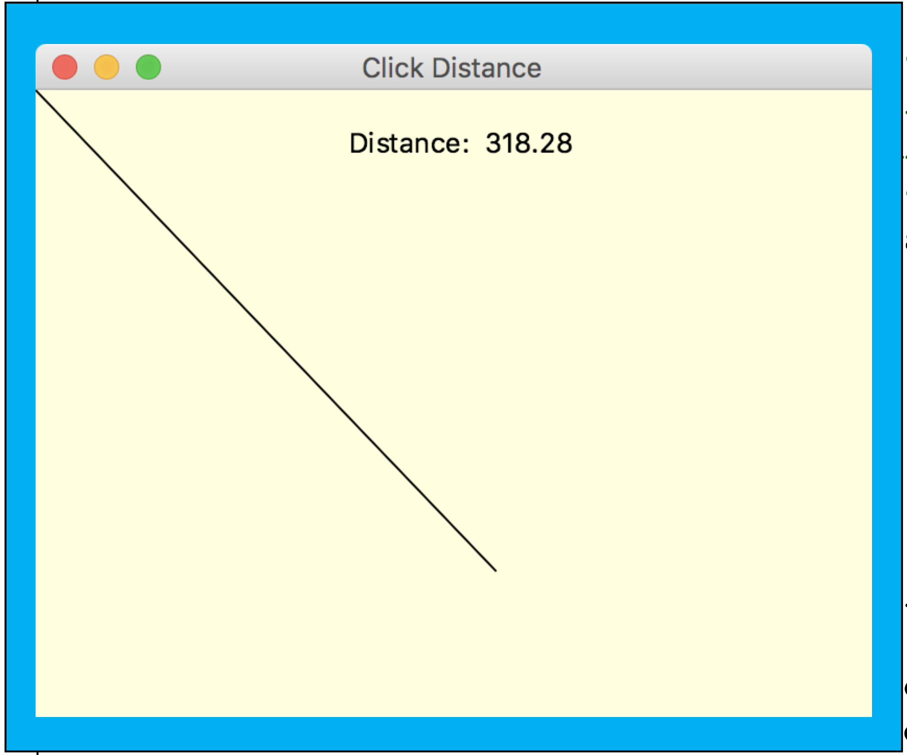
```
        Group root = new Group(distanceText, line);

        Scene scene = new Scene(root, 400, 300, Color.LIGHTYELLOW);

        scene.setOnMouseClicked(this::processMouseClick);

        primaryStage.setTitle("Click Distance");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

continue

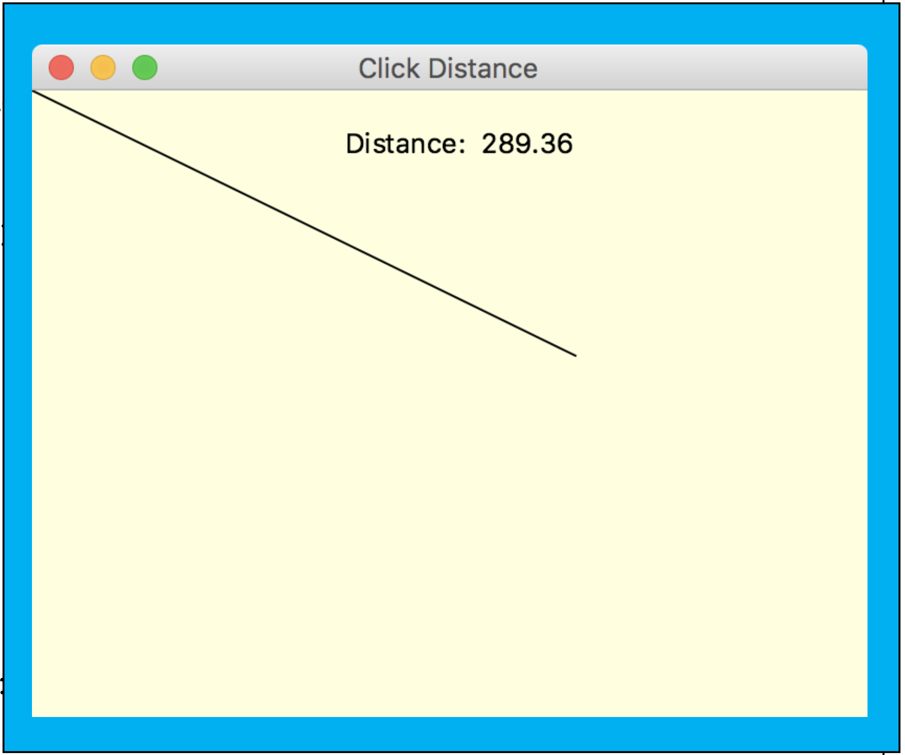
continue

```
    //--------------------------------------------------------------------
    //   Resets the end point of the line to the location of the mouse
    //   click event and updates the distance displayed.
    //--------------------------------------------------------------------
    public void processMouseClick(MouseEvent event)
    {
        double clickX = event.getX();
        double clickY = event.getY();

        line.setEndX(clickX);
        line.setEndY(clickY);

        double distance = Math.sqrt(clickX * clickX + clickY * clickY);

        String distanceStr = String.format("%.2f", distance);
        distanceText.setText("Distance:   " + distanceStr);
    }
}
```

continue


Click Distance

Distance: 318.28


Click Distance

Distance: 289.36

```
--
in
is
--
se




li

or
e:
    }
}
```

Mouse Events

A stream of mouse moved or mouse dragged events occur while the mouse is in motion

This essentially allows the program to track the movement in real time

Using the mouse to "draw" a shape into place is called *rubberbanding*

```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.input.MouseEvent;
import javafx.scene.paint.Color;
import javafx.scene.shape.Line;
import javafx.stage.Stage;

//************************************************************************
//  RubberLines.java       Author: Lewis/Loftus
//
//  Demonstrates the handling of mouse press and mouse drag events.
//************************************************************************

public class RubberLines extends Application
{
    private Line currentLine;
    private Group root;

    //--------------------------------------------------------------------
    //  Displays an initially empty scene, waiting for the user to
    //  draw lines with the mouse.
    //--------------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        root = new Group();

        Scene scene = new Scene(root, 500, 300, Color.BLACK);
```

continue

continue

```
        scene.setOnMousePressed(this::processMousePress);
        scene.setOnMouseDragged(this::processMouseDrag);

        primaryStage.setTitle("Rubber Lines");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    //-----------------------------------------------------------------
    //  Adds a new line to the scene when the mouse button is pressed.
    //-----------------------------------------------------------------
    public void processMousePress(MouseEvent event)
    {
        currentLine = new Line(event.getX(), event.getY(), event.getX(),
            event.getY());
        currentLine.setStroke(Color.CYAN);
        currentLine.setStrokeWidth(3);
        root.getChildren().add(currentLine);
    }
```

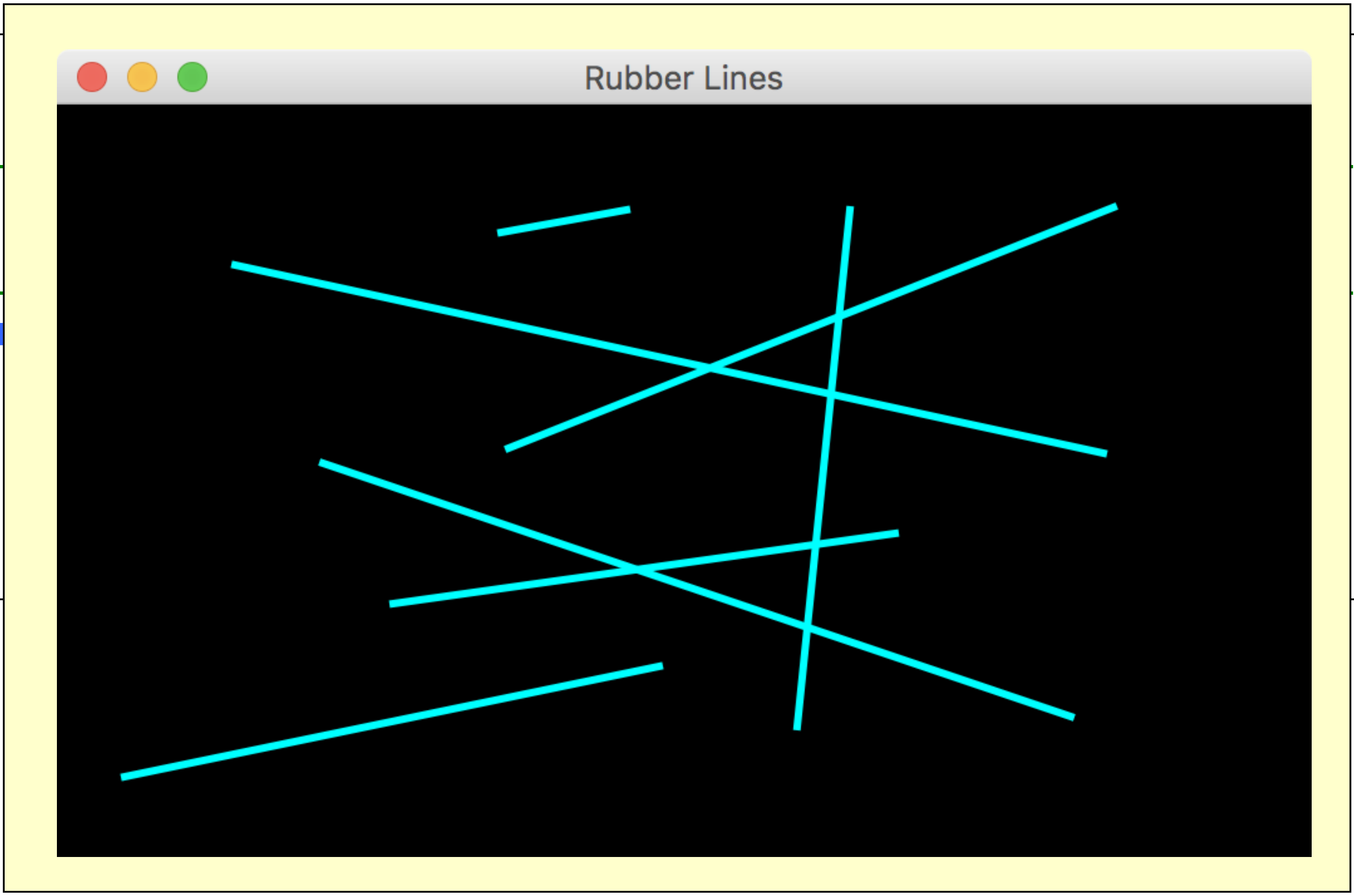
continue

continue

```
    //-----------------------------------------------------------------
    //  Updates the end point of the current line as the mouse is
    //  dragged, creating the rubber band effect.
    //-----------------------------------------------------------------
    public void processMouseDrag(MouseEvent event)
    {
        currentLine.setEndX(event.getX());
        currentLine.setEndY(event.getY());
    }
}
```

continue

```
   //--
   //
   //
   //--
   publ
   {


   }
}
```



Key Events

There are three JavaFX events related to the user typing at the keyboard:

Event	Description
key pressed	a keyboard key is pressed down
key released	a keyboard key is released
key typed	a keyboard key that generates a character is typed (pressed and released)

The `getCode` method of the event object returns a code that represents the key that was pressed

```java
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

//************************************************************************
//  AlienDirection.java       Author: Lewis/Loftus
//
//  Demonstrates the handling of keyboard events.
//************************************************************************

public class AlienDirection extends Application
{
    public final static int JUMP = 10;

    private ImageView imageView;
```

continue

continue

```
//--------------------------------------------------------------------
//  Displays an image that can be moved using the arrow keys.
//--------------------------------------------------------------------
public void start(Stage primaryStage)
{
    Image alien = new Image("alien.png");

    imageView = new ImageView(alien);
    imageView.setX(20);
    imageView.setY(20);

    Group root = new Group(imageView);

    Scene scene = new Scene(root, 400, 200, Color.BLACK);
    scene.setOnKeyPressed(this::processKeyPress);

    primaryStage.setTitle("Alien Direction");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

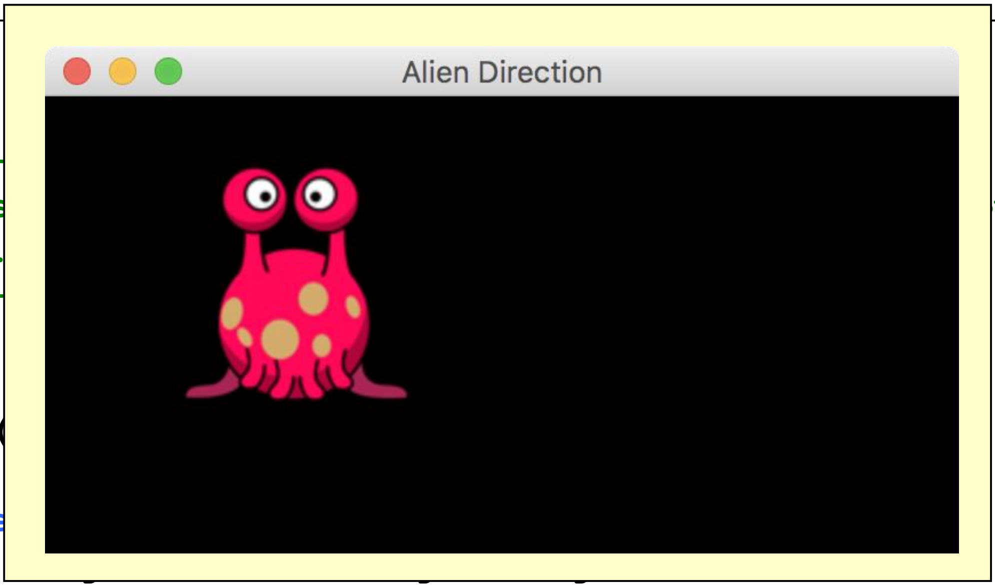
continue

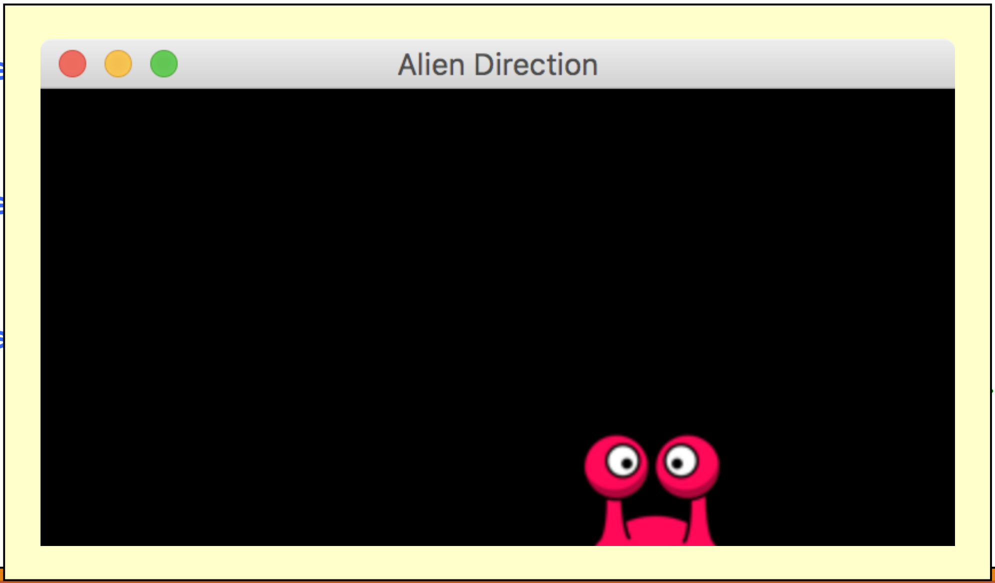
continue

```
    //-----------------------------------------------------------------
    //  Modifies the position of the image view when an arrow key is
    //  pressed.
    //-----------------------------------------------------------------
    public void processKeyPress(KeyEvent event)
    {
        switch (event.getCode())
        {
            case UP:
                imageView.setY(imageView.getY() - JUMP);
                break;
            case DOWN:
                imageView.setY(imageView.getY() + JUMP);
                break;
            case RIGHT:
                imageView.setX(imageView.getX() + JUMP);
                break;
            case LEFT:
                imageView.setX(imageView.getX() - JUMP);
                break;
            default:
                break;  // do nothing if it's not an arrow key
        }
    }
}
```

continue

```
    //----------                                                     --------------
    //   Modifies                                                    w key is
    //   pressed.
    //----------                                                     --------------
    public void 
    {
        switch (
        {
            case
                break;
            case DOWN:
                imageView.setY(imageView.getY() + JUMP);

            case

            case

            defa
                                                                   ey
        }
    }
}
```





REBUS PUZZLE

COM COM
COM COM

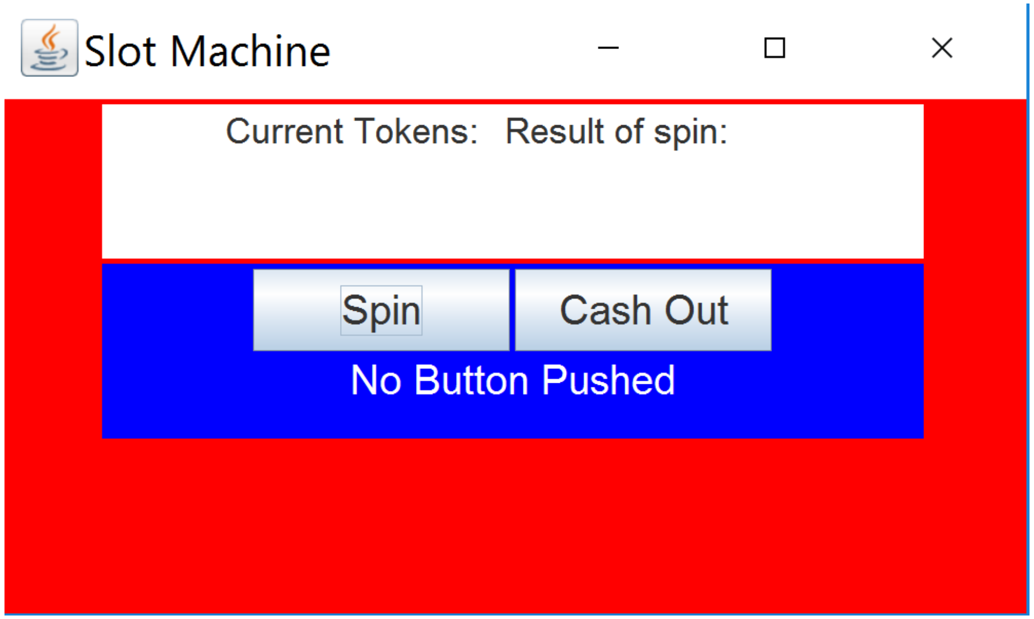
Individual Exercise

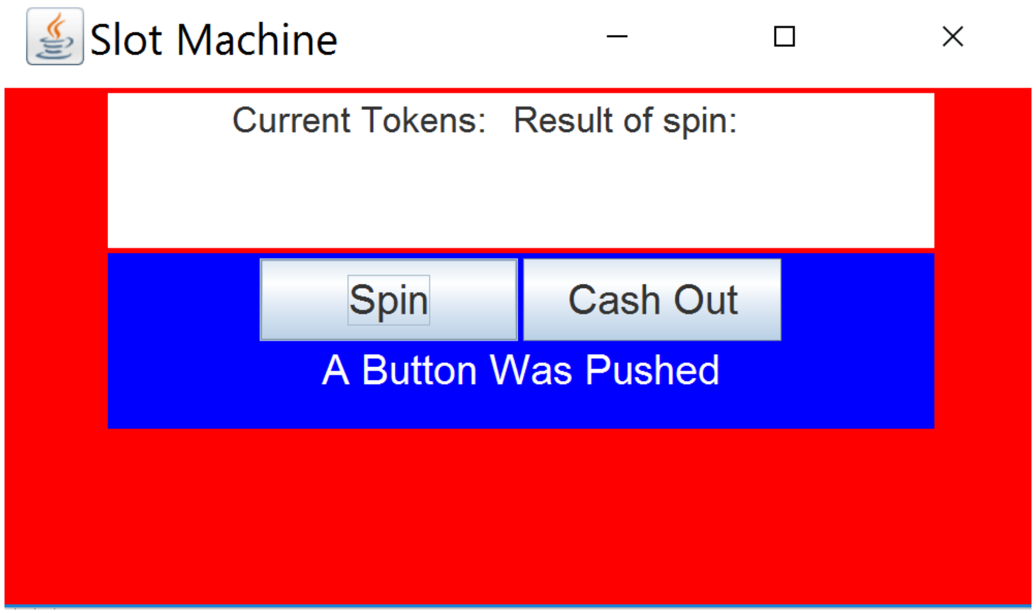
Note: this is an individual assignment!

Continue with Programming Assignment #2

On your SlotMachinePanel, add two labels to the upper panel for the current number of tokens and the spin result. Add two buttons to the lower panel labeled “Spin” and “Cash Out.” Add a button listener for the two buttons. As a first step, have the button listener print out a message that only indicates a button was pushed.

- *Add more instance variables for 3 labels and 2 buttons*
- *Create label and button objects*
- *Add 2 of the labels to slotsPanel and 2 buttons and label to buttonPanel*
- *Create ButtonListener class*


Slot Machine
Current Tokens:
Result of spin:
Spin
Cash Out
No Button Pushed


Slot Machine
Current Tokens:
Result of spin:
Spin
Cash Out
A Button Was Pushed

Some Useful Code

```
spinLabel = new JLabel ("Result of spin:       ");
spinLabel.setFont(new Font("Arial", Font.PLAIN, 35));
```

```
spinButton = new JButton ("Spin");
spinButton.setPreferredSize(new Dimension (250,80));
spinButton.setFont(new Font("Arial", Font.PLAIN, 40));
```

Assignment for Classes 18 and 20

Class 18

- Review ClickDistance, RubberLines, AlienDirection
- Study for Quiz #2

Class 20

- Read Chapter 8.1, 8.2, 8.3, 8.4, 8.6