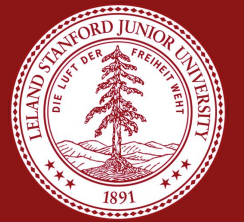
Interactors

Lecture 23

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

With inspiration from slides created by Keith Schwarz, Mehran Sahami, Eric Roberts, Stuart Reges, Chris Piech and others.



Announcements

- Assignment 5 was due at 10am
- Assignment 6 will be released after lecture
  - Note: No late days may be used on Assignment 6

- We are no longer accepting regrade requests for the Midterm.

Plan for Today

- Review: Classes
- Interacting with Interactors
- JLabels, JTextFields, JButtons
- Example: MadLibs
- Example: MyDrawingProgram

Learning Goal for Today

Know how to add and cause changes in your program with

`Interactors`

What do we know about classes?

A class defines a new variable type.

Review: Making a Class ~ 3 Ingredients

1. Define the **variables** each instance stores (state)
2. Define the **constructor** used to make a new instance
3. Define the **methods** you can call on an instance (behaviors)

* all class methods and constructors have access to a `this` reference

Review: BankAccount

```
public class BankAccount {
	// Step 1: the data inside a BankAccount
	private String name;
	private double balance;

	// Step 2: how to create a new BankAccount
	public BankAcount(String name) {
		this.name = name;
		this.balance = 0;
	}
	// Step 3: the things a BankAccount can do
	public void deposit(double amount) {
		this.balance += amount;
	}
	public boolean withdraw(double amount) {
		if (this.balance >= amount) {
			this.balance -= amount;
			return true;
		}
		return false;
	}
}
```

Review: Getters & Setters

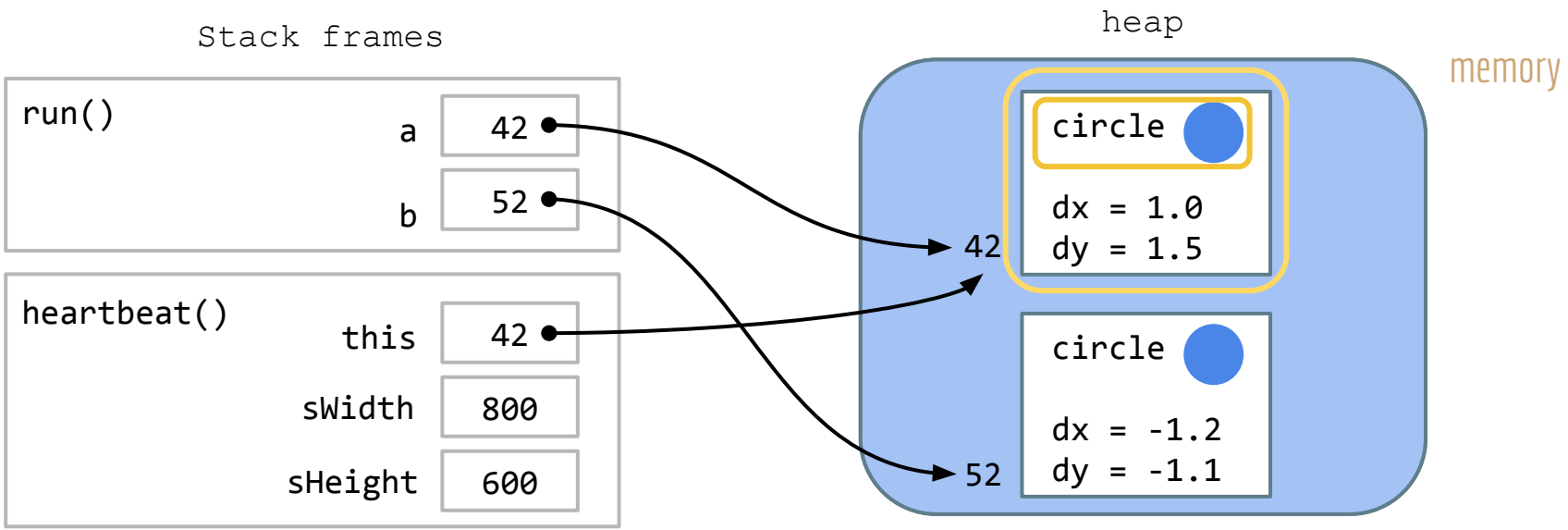
```
public class BankAccount {
     private String name;
     private double balance;
     …
     // “setter”
     public void setName(String newName) {
          if (newName.length() > 0) {
               this.name = newName;
          }
     }
     // “getters”
     public String getName() {
          return this.name;
     }
     public double getBalance() {
          return this.balance;
     }
}
```

```
public class BouncingBall extends GraphicsProgram {
        public void run() {
                // make a few new bouncing balls
                Ball a = new Ball();
                Ball b = new Ball();

                // call a method on one of the balls
                a.heartbeat(getWidth(), getHeight());
        }
}
```

```
public void heartbeat(int sWidth, int sHeight) {
        this.circle.move();
        reflectOffWalls(sWidth, sHeight);
}
```

heartbeat() was called on ball **a**
⇒ So, **this** refers to **a**

code



A Note About Classes

`extends` is the word we use when a Class is based off of another class!

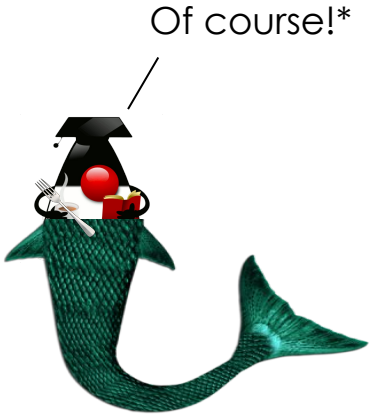
`implements` is the word we use when we're promising our Class will define certain things. Can also be used to share constants between Classes!

We'll see this in an example later.

MORE Interaction

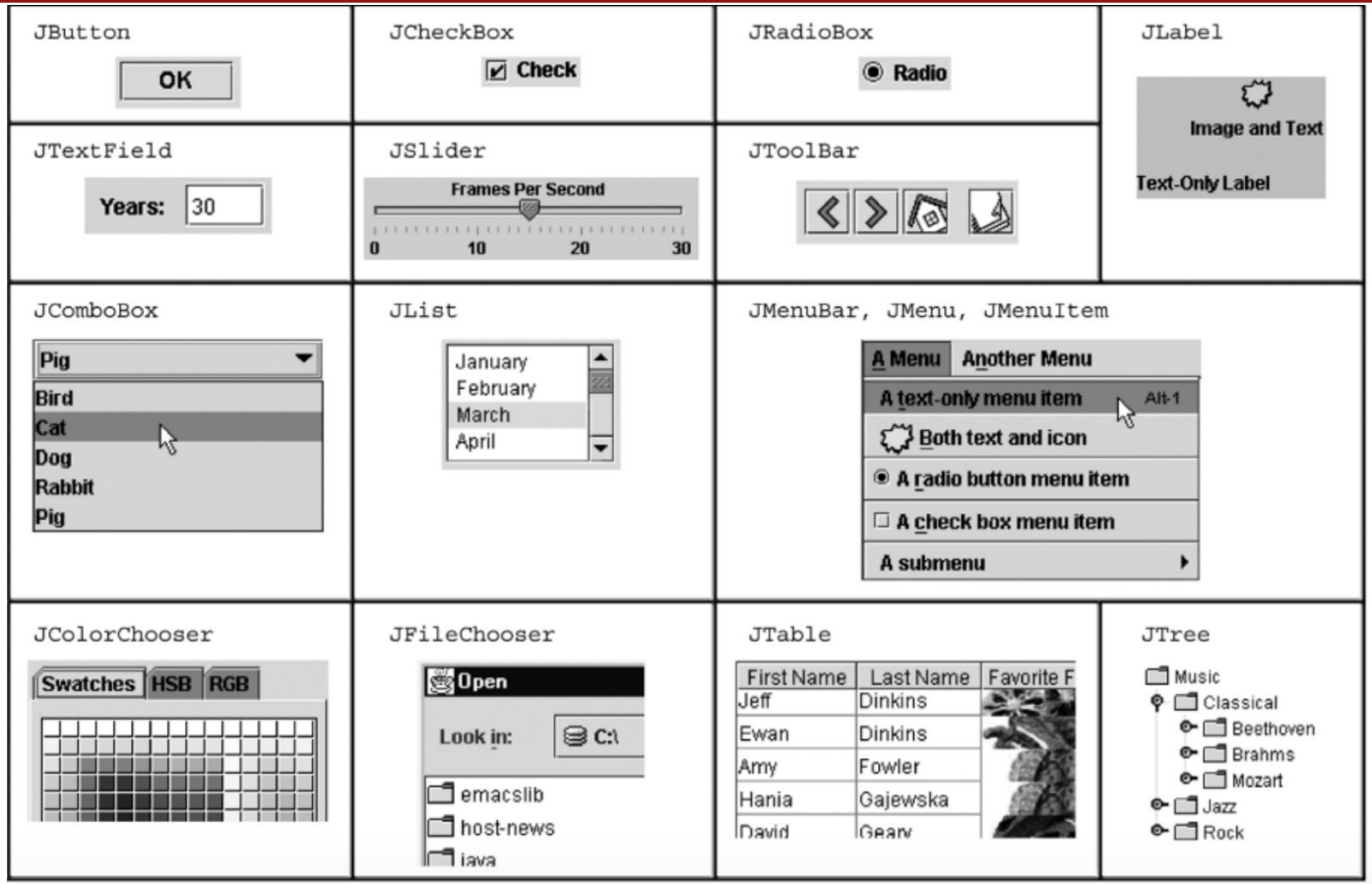
Right now, we know how to read input from mouse clicks, movement, and typed user input.

Can we detect even *more* interactions?



* How many times can I use this joke before we get sick of this...

Of Course, We Can!

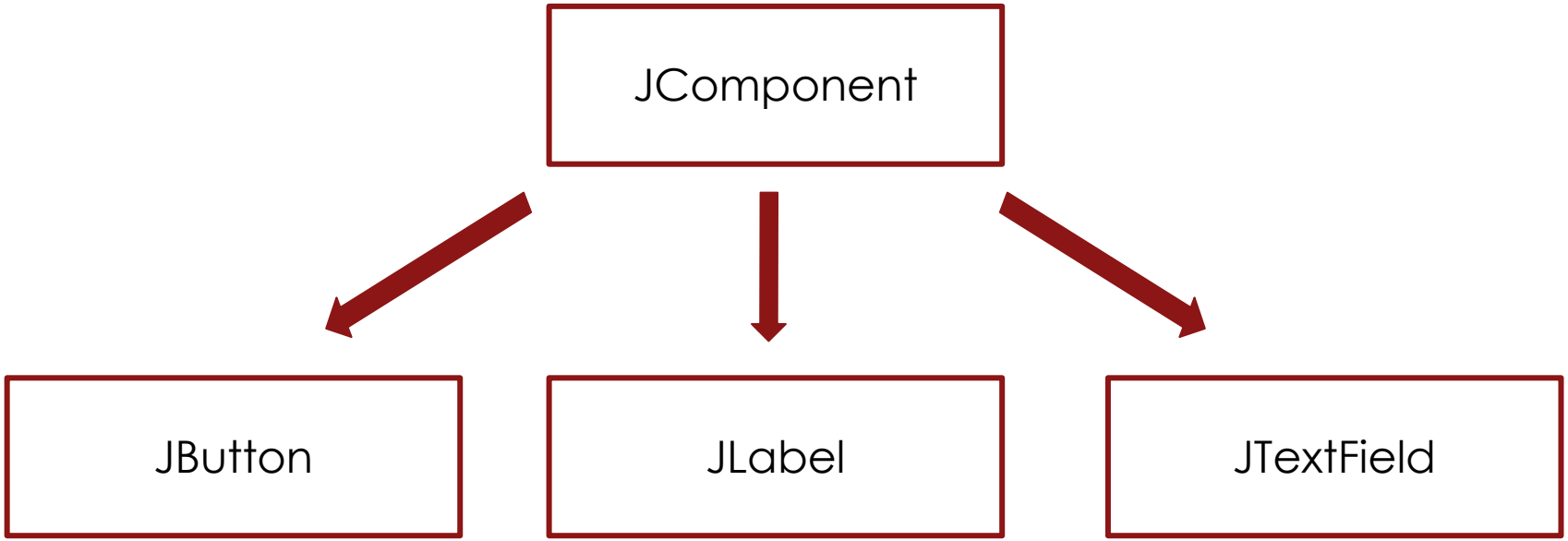


Interactors

We can use Interactors to detect more complex interactions from our users.

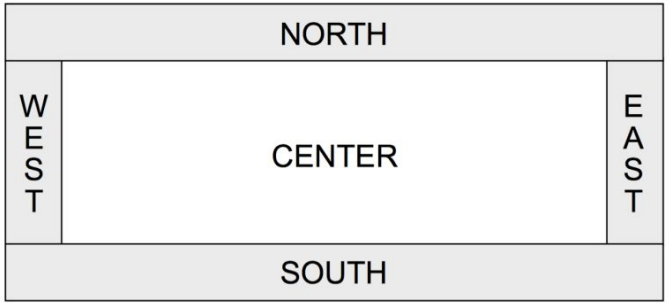
Today, we'll be looking at JLabels, JTextFields, and JButtons.

JComponent is the SuperClass!



Regions

Interactors can be placed in 5 regions on the screen.



- The center is usually where things happen!
  - The ConsoleProgram adds the Console there.
  - The GraphicsProgram add the Canvas there.
- We only see the other regions of the screen if we add interactors there using `add(component, REGION)`
- Interactors are automatically centered in their region.

Which Libraries to Import

```
import javax.swing.*;
import java.awt.event*;
```

Our First Interactor

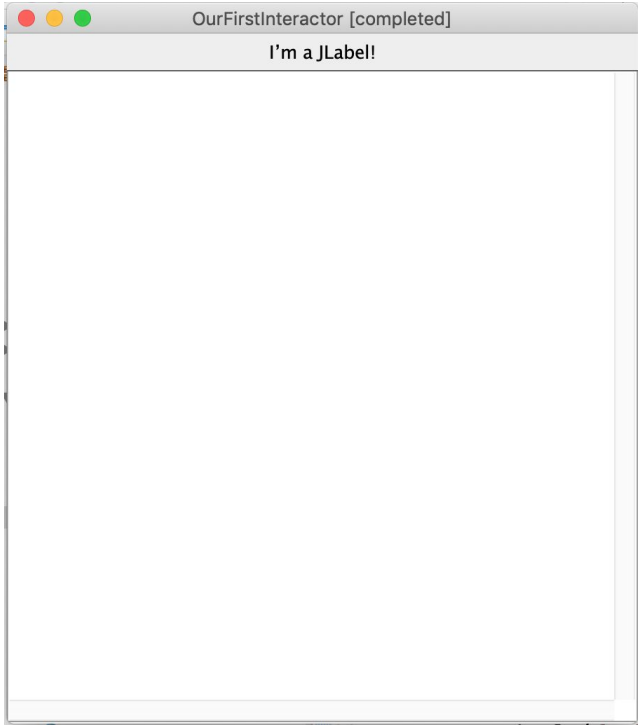
```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {









}
```

Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {



     public void init(){
          // add interactors here!



     }

}
```

Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {



     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);



     }

}
```



Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);

     }

}
```

The text of the label

The region where we're creating the label.

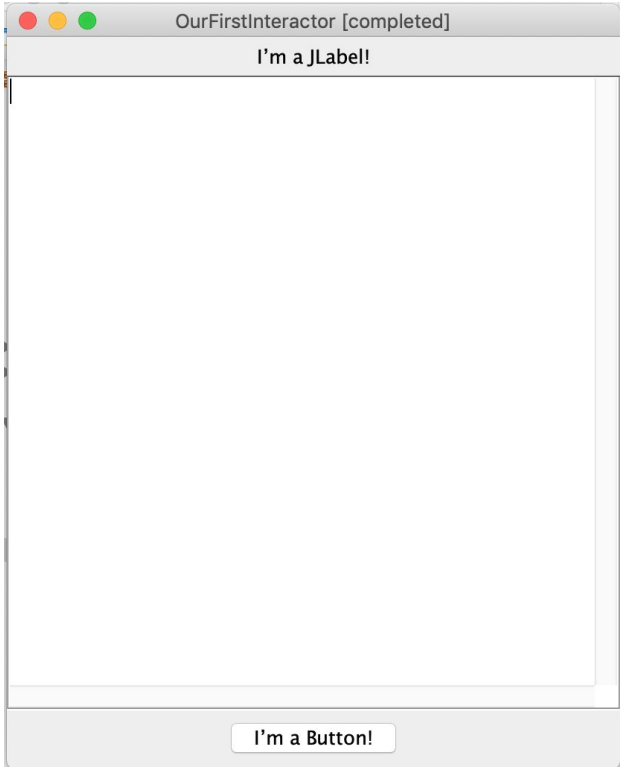
Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {


     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);



     }

}
```

We didn't create a new variable for the label. We created it and added it in this one step.

Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {



     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);
          add(new JButton("I'm a Button!"), SOUTH);


     }

}
```



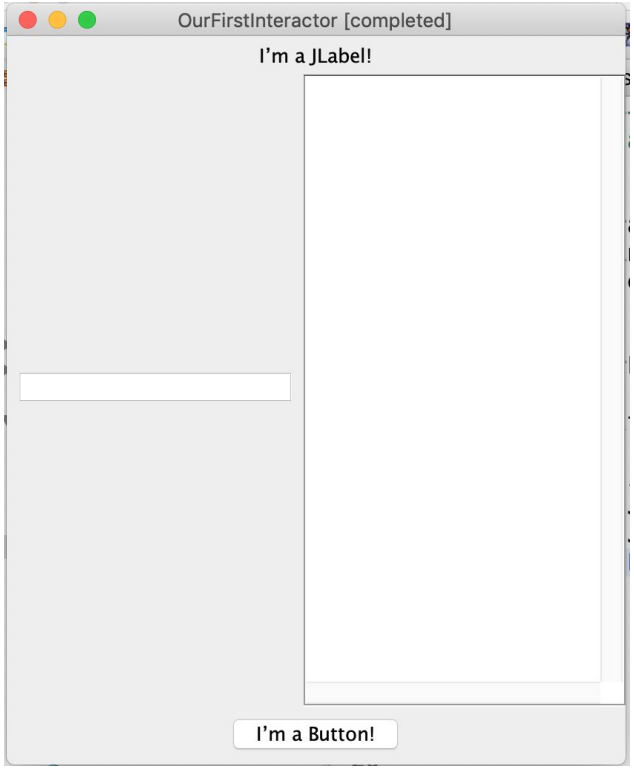
Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     private JTextField textField = new JTextField(15);

     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);
          add(new JButton("I'm a Button!"), SOUTH);


     }

}
```

We want to access our JTextField later, so we're making it an instance variable.

Our First Interactor

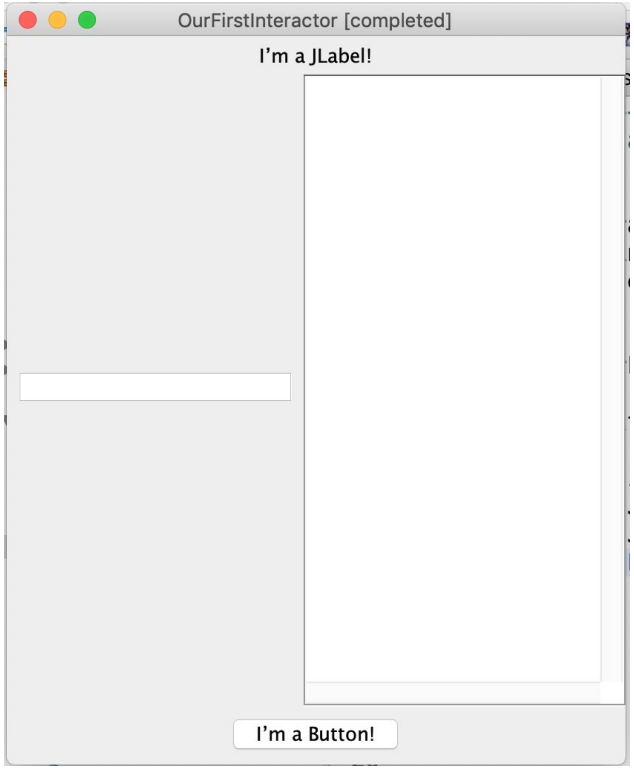
```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     private JTextField textField = new JTextField(15);

     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);
          add(new JButton("I'm a Button!"), SOUTH);
          add(textField, WEST);

     }

}
```



Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     private JTextField textField = new JTextField(15);

     public void init(){
          add(new JLabel("I'm a JLabel!"), NORTH);
          add(new JButton("I'm a Button!"), SOUTH);
          add(textField, WEST);
          addActionListeners();
     }

}
```

In order to detect actions in these fields, we must addActionListeners()



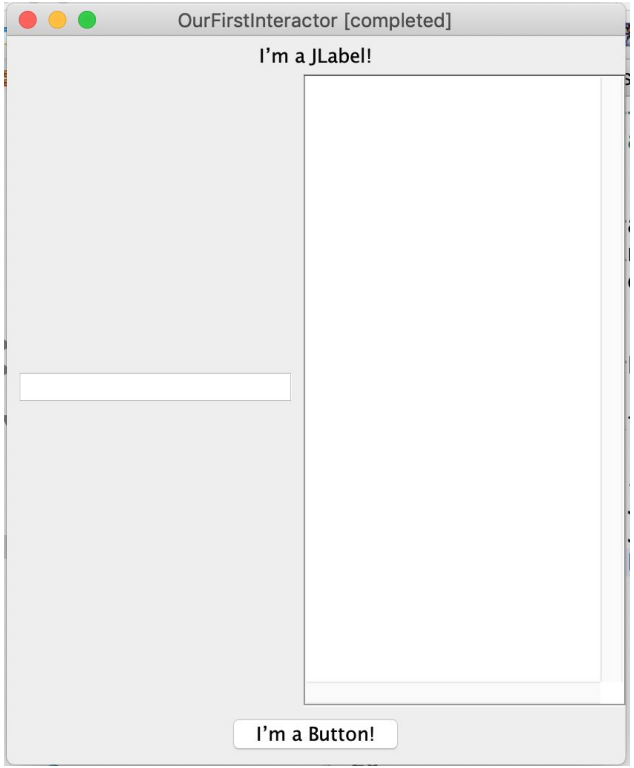
actionPerformed

Using `addActionListeners()` allows us to detect actions. *This line of code must occur **after** we've added our JComponents.*

This allows us to use `actionPerformed` to do actions when someone interacts with our interactors.

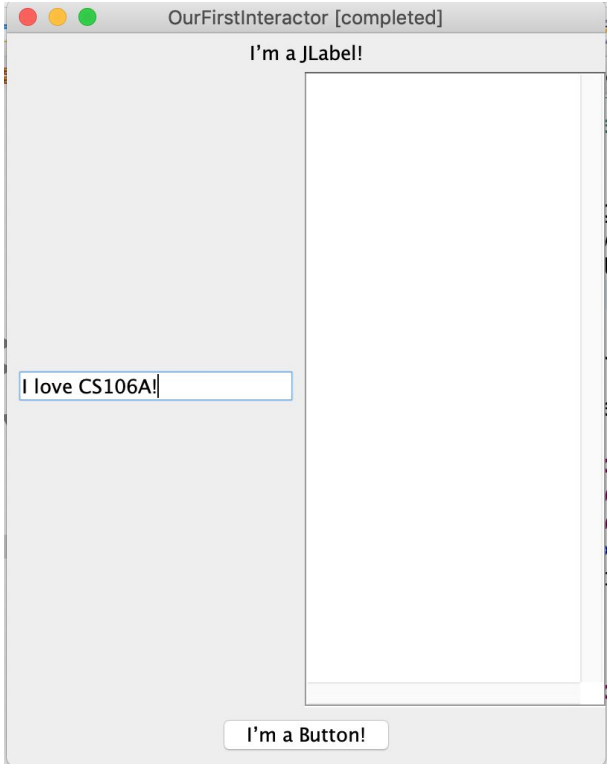
Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     ... // added JComponents, etc here

     public void actionPerformed(ActionEvent e){
          // Similar to mouseMoved, mouseClicked, etc.
          // Only occurs when an action is performed
          // e is the event that was detected


     }
}
```



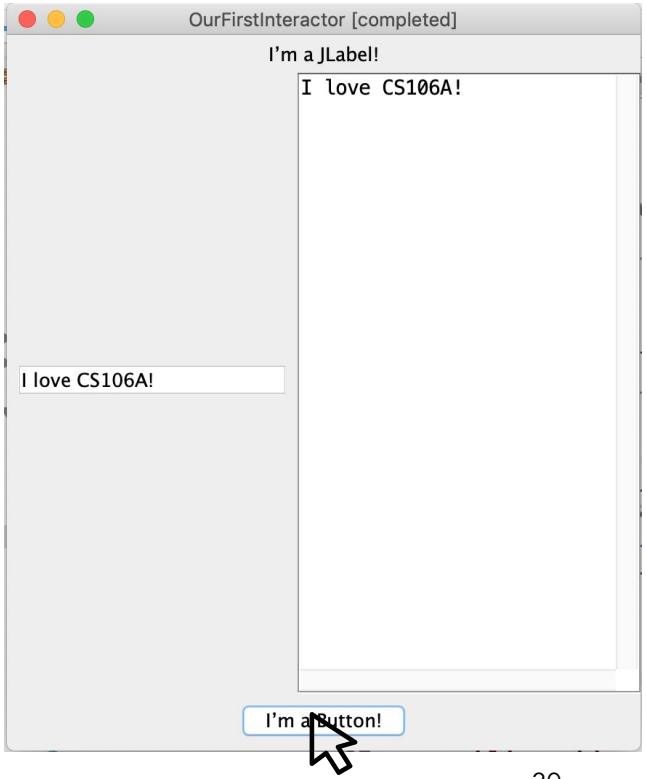
Our First Interactor

```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     ... // added JComponents, etc here

     public void actionPerformed(ActionEvent e){
          // If we click the button, this will trigger
          String text = textField.getText();
          println(text);


     }
}
```



Our First Interactor

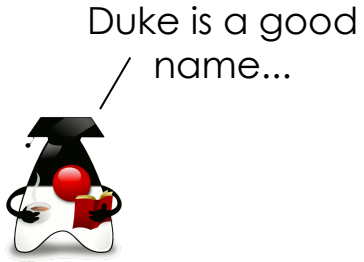
```
import javax.swing.*;
import java.awt.event*;

public class ourFirstInteractor extends ConsoleProgram {

     ... // added JComponents, etc here

     public void actionPerformed(ActionEvent e){
          // If we click the button, this will trigger
          String text = textField.getText();
          println(text);


     }
}
```



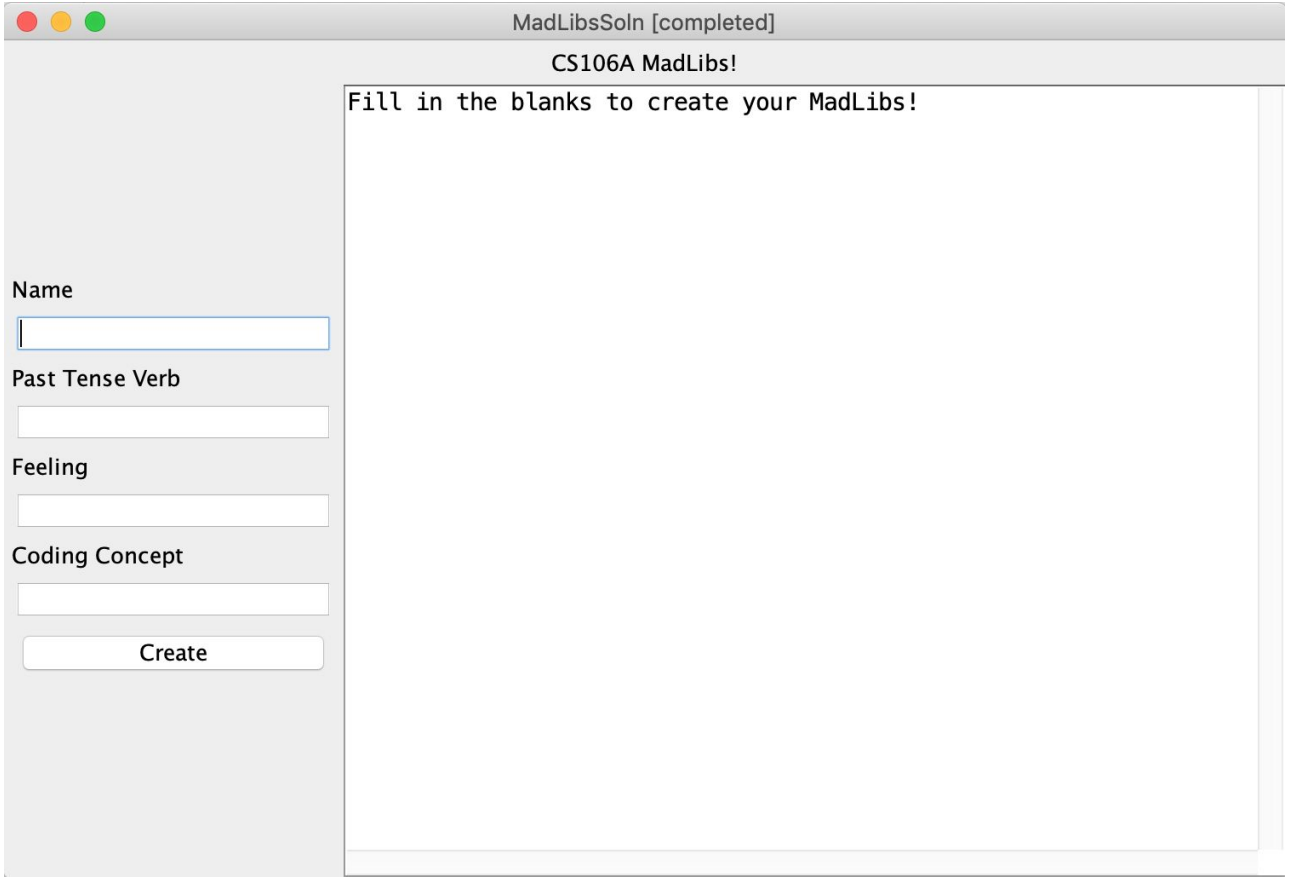
We've Learned Enough to Create a Game!

MadLibs is a fill in the blank game where we ask for names, adjectives, verbs, etc and use the words given to us by the user to create a humorous blurb.

Let's create a simple MadLibs game!



MadLibs


MadLibsSoln [completed]
CS106A MadLibs!
Fill in the blanks to create your MadLibs!
Name
Past Tense Verb
Feeling
Coding Concept
Create

Let’s Code It!

MadLibs Code

```
public class MadLibsSoln extends ConsoleProgram {

    private JTextField nameField = new JTextField(15);
    private JTextField verbField = new JTextField(15);
    private JTextField feelingField = new JTextField(15);
    private JTextField codeField = new JTextField(15);

    public void init() {
        add(new JLabel("CS106A MadLibs!"), NORTH);
        add(new JLabel("Name"), WEST);
        add(nameField, WEST);
        add(new JLabel("Past Tense Verb"), WEST);
        add(verbField, WEST);
        add(new JLabel("Feeling"), WEST);
        add(feelingField, WEST);
        add(new JLabel("Coding Concept"), WEST);
        add(codeField, WEST);
        add(new JButton("Create"), WEST);
        addActionListeners();
    }

    public void run(){
        println("Fill in the blanks to create your MadLibs! \n");
    }

    public void actionPerformed(ActionEvent e) {
        String name = nameField.getText();
        String verb = verbField.getText();
        String feeling = feelingField.getText();
        String code = codeField.getText();

        createMadLib(name, verb, feeling, code);
    }

    private void createMadLib(String name, String verb, String feeling, String code) {
        println("On " + name + "'s first day of CS106A they accidentally woke up");
        println("late for class! They " + verb + " to class, but when " + name);
        println("arrived at Bishop Auditorium, no one was there! " + name + " was ");
        println(feeling + ". Well, good thing no one uses " + code + " anyways.");
    }
}
```

What If We Have TWO Buttons?

`actionPerformed` triggers if a button is pressed.

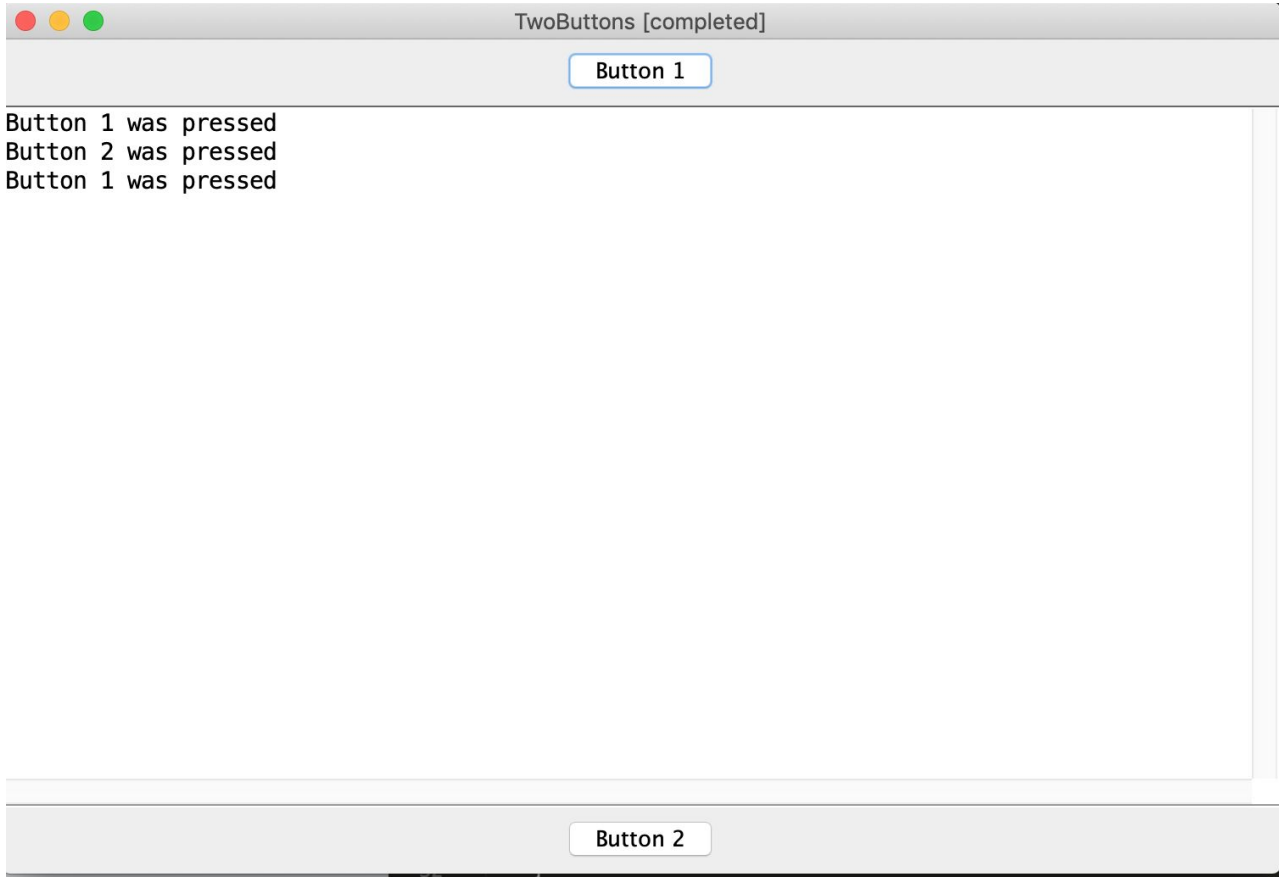
If we press two buttons, how does it know which button was pressed?

What If We Have TWO Buttons?

Method	Description
`e.getActionCommand()`	a text description of the event *(e.g., the text of the button clicked)*
`e.getSource()`	the interactor that generated the event

```
public void actionPerformed(ActionEvent e){

     String command = e.getActionCommand();
     if(command.equals("Button 1")){
          println("Button 1 was pressed");
     } else if (command.equals("Button 2")){
          println("Button 2 was pressed");
     }

}
```

Two Button Example 1



What If We Have TWO Buttons?

Method	Description
e.getActionCommand()	a text description of the event *(e.g., the text of the button clicked)*
e.getSource()	the interactor that generated the event

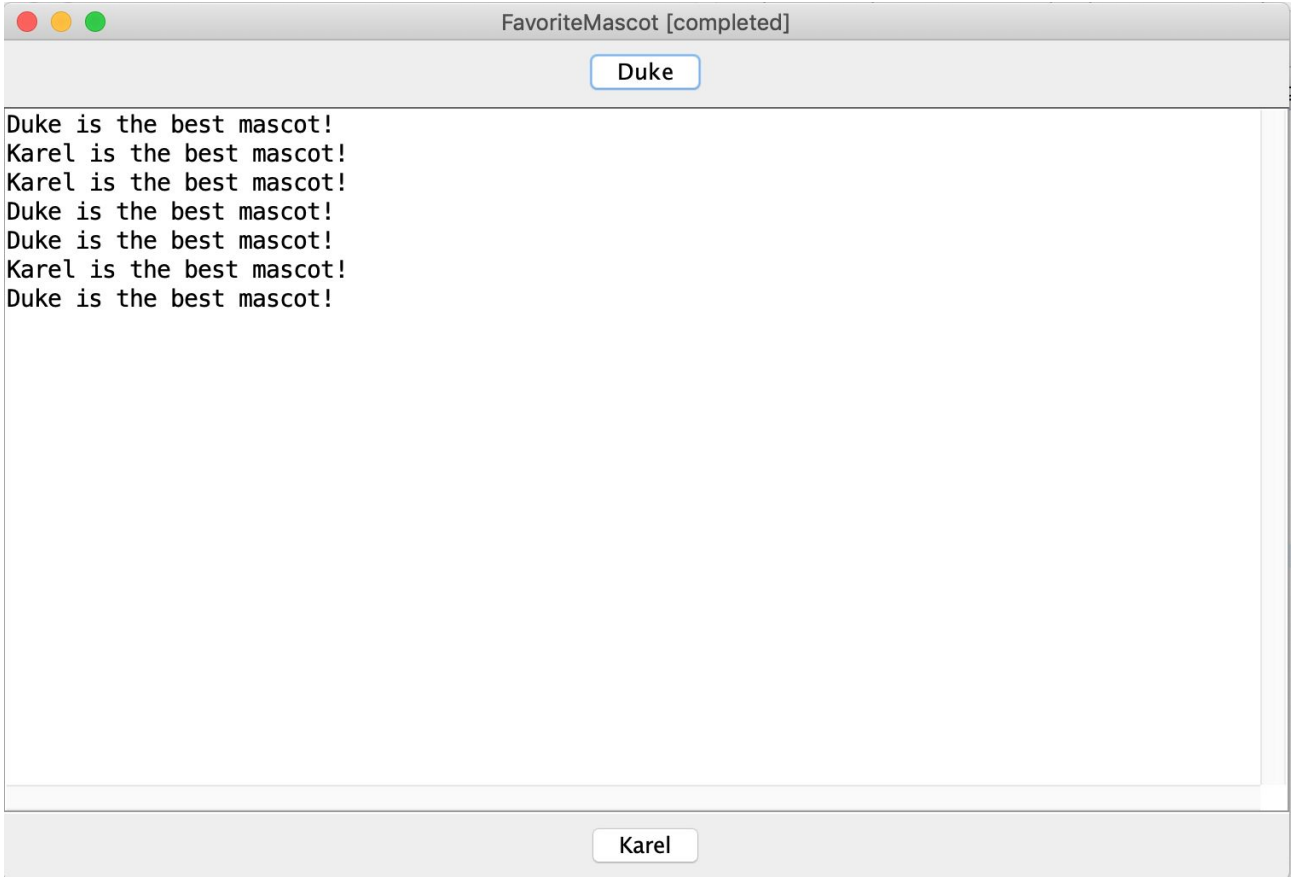
```
private JButton duke = new JButton("Duke");
private JButton karel = new JButton("Karel");

... // added JComponents, etc here

public void actionPerformed(ActionEvent e){
     if(e.getSource() == duke){
          println("Duke is the best mascot!");
     } else if (e.getSource() == karel){
          println("Karel is the best mascot!");
     }
}
```

What If We Have TWO Buttons?

Method	Description
e.getActionCommand()	a text description of the event *(e.g., the text of the button clicked)*
e.getSource()	the interactor that generated the event

```
private JButton duke = new JButton("Duke");
private JButton karel = new JButton("Karel");

... // added JComponents, etc here

public void actionPerformed(ActionEvent e){
     if(e.getActionCommand().equals(“Duke”)){
          println("Duke is the best mascot!");
     } else if (e.getActionCommand().equals(“Karel”)){
          println("Karel is the best mascot!");
     }
}
```

Two Button Example 2



Pressing Enter

Do we need a button? What if I just press “enter” in a text box?

Pressing Enter

```
import javax.swing.*;
import java.awt.event*;

public class PressingEnter extends ConsoleProgram {

     private JTextField textField = new JTextField(15);

     public void init(){
          // the next two lines allows us to detect if we press “Enter” in this
          // textField
          textField.addActionListener(this);
          textField.setActionCommand(“Go”);

          add(textField, NORTH);
     }
}
```

Pressing Enter

```
import javax.swing.*;
import java.awt.event*;

public class PressingEnter extends ConsoleProgram {

     ... // added JComponents, etc here

     public void actionPerformed(ActionEvent e){
          if(e.getActionCommand().equals("Go")){
               println("You pressed enter in the textField!");
          }
     }
}
```

Pressing Enter

Does this mean we can either press enter OR use a button? Why can't we do both? 😢

Pressing Enter

A lot of times, a text field has a button that "goes with it". If you set the text fields action command to the name of the button, we can detect pressing "enter" and pressing the button!

Pressing Enter

```
public void init(){
     JButton goButton = new JButton("Go");
     add(goButton, NORTH);

     JTextField textField = new JTextField(15);
     textField.addActionListener(this);
     textField.setActionCommand("Go");
     add(textField, NORTH);
     addActionListeners();
}

public void actionPerformed(ActionEvent e){
     if(e.getActionCommand().equals("Go")){
          println("You pressed enter OR you pressed the button!");
     }
}
```

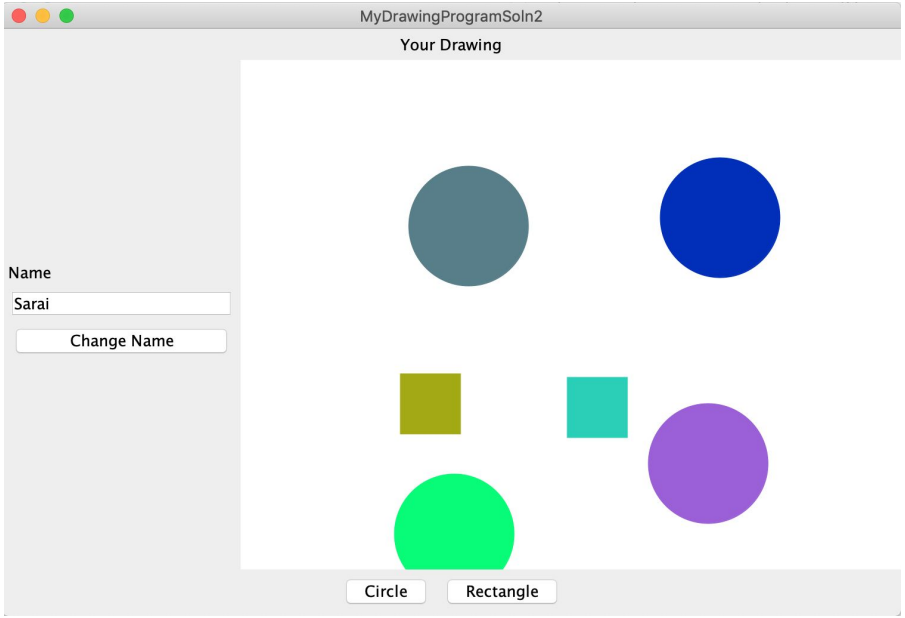
Pressing Enter

```
public void init(){
     JButton goButton = new JButton("Go");
     add(goButton, NORTH);

     JTextField textField = new JTextField(15);
     textField.addActionListener(this);
     textField.setActionCommand("Go");
     add(textField, NORTH);
     addActionListeners();
}

public void actionPerformed(ActionEvent e){
     if(e.getActionCommand().equals("Go")){
          println("You pressed enter OR you pressed the button!");
     }
}
```

Graphics and Interactors

Let's use our new knowledge to create a drawing game! We'll let the user choose which shapes to draw on the canvas and add their name to their drawing!

We'll even let the artist save the Drawing using their name!



Let’s Code It!

Drawing Class Code

```
import acm.graphics.*;
import java.util.*;

public class Drawing{

    private String artist;
    private ArrayList<GObject> shapes;

    public Drawing(String artist) {
        this.artist = artist;
        this.shapes = new ArrayList<>();
    }

    public String getArtist() {
        return this.artist;
    }

    public GObject getShapeAt(int index) {
        return this.shapes.get(index);
    }

    public void addShape(GObject shape) {
        this.shapes.add(shape);
    }

    public int numShapes() {
        return this.shapes.size();
    }

    public String toString(){
        return this.artist + ": shapes=" + this.shapes.toString();
    }


}
```

Drawing Program Code

```
import acm.program.*;
import acm.graphics.*;
import javax.swing.*;
import java.util.*;

import java.awt.Color;
import java.awt.event.*;
import acm.util.*;


public class MyDrawingProgramSoln2 extends GraphicsProgram implements MyDrawingProgramConstants {

    private JTextField nameField = new JTextField(15);
    private int currShapeType = CIRCLE;
    private HashMap<String, Drawing> drawings = new HashMap<>();
    private Drawing currDrawing;

    public void init() {
        add(new JLabel("Your Drawing"), NORTH);
        add(new JLabel("Name"), WEST);
        add(nameField, WEST);
        add(new JButton("Change Name"), WEST);
        add(new JButton("Circle"), SOUTH);
        add(new JButton("Rectangle"), SOUTH);
        addActionListeners();
    }

    public void actionPerformed(ActionEvent e) {
        if(e.getActionCommand().equals("Circle")){
            currShapeType = CIRCLE;
        } else if (e.getActionCommand().equals("Rectangle")){
            currShapeType = RECTANGLE;
        } else if(e.getActionCommand().equals("Change Name")) {
            String name = nameField.getText();
            removeAll();

            if(drawings.containsKey(name)) {
                currDrawing = drawings.get(name);
                for(int i = 0; i < currDrawing.numShapes(); i ++) {
                    add(currDrawing.getShapeAt(i));
                }
            } else {
                currDrawing = new Drawing(name);
            }
        }
    }

    public void mouseClicked(MouseEvent e) {
        int mouseX = e.getX();
        int mouseY = e.getY();

        if(currShapeType == CIRCLE) {
            GOval newCircle = randomColoredCircle();
            add(newCircle, mouseX, mouseY);
            if(currDrawing != null) {
                currDrawing.addShape(newCircle);
                drawings.put(nameField.getText(), currDrawing);
            }


        } else if (currShapeType == RECTANGLE) {
            GRect newRect = randomColoredRect();
            add(newRect, mouseX, mouseY);
            if(currDrawing != null) {
                currDrawing.addShape(newRect);
                drawings.put(nameField.getText(), currDrawing);
            }

        }
    }

/* The below code is given for you. */

    private Color getRandomColor() {
        RandomGenerator rg = new RandomGenerator();
        return rg.nextColor();
    }

    private GOval randomColoredCircle() {
        GOval newCircle = new GOval(CIRCLE_WIDTH, CIRCLE_WIDTH);
        newCircle.setFilled(true);
        newCircle.setColor(getRandomColor());
        return newCircle;

    }
    private GRect randomColoredRect() {
        GRect newRect = new GRect(RECTANGLE_WIDTH, RECTANGLE_WIDTH);
        newRect.setFilled(true);
        newRect.setColor(getRandomColor());
        return newRect;

    }
}
```

Plan for Today

- Review: Classes
- Interacting with Interactors
- JLabels, JTextFields, JButtons
- Example: MadLibs
- Example: MyDrawingProgram