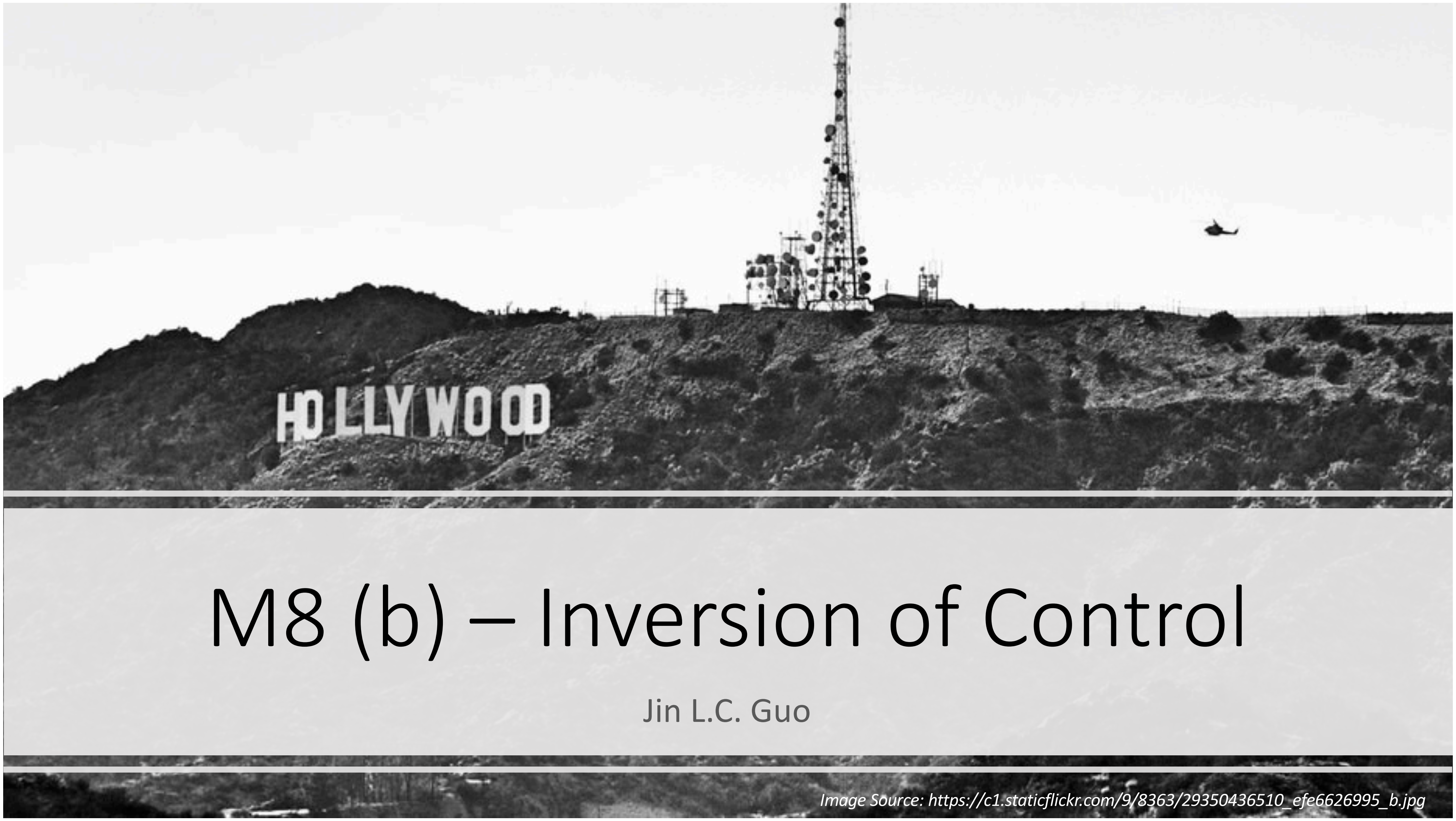


M8 (b) – Inversion of Control

Jin L.C. Guo

Image Source: https://c1.staticflickr.com/9/8363/29350436510_efe6626995_b.jpg

Objective

- Be able to Use Callback to achieve decoupling
- Be able to use the Observer design pattern effectively;
- Event Handling in GUI applications
- Understand the concept of an application framework;
- Understand the Model-View-Controller Decomposition

Objective

- Be able to Use Callback to achieve decoupling
- Be able to use the Observer design pattern effectively;
- Event Handling in GUI applications
- Understand the concept of an application framework;
- Understand the Model-View-Controller Decomposition

Event

- A notification that something interesting has happened.
- Examples in Graphic Interface?

Move a mouse

User click a button

Press a key

Mouse press and drag

Menu item is selected

Window is closed

Popup window is hidden

How to capture event and act accordingly

- Define an event handler

implement

Interface EventHandler<T extends Event>

WindowEvent

InputEvent

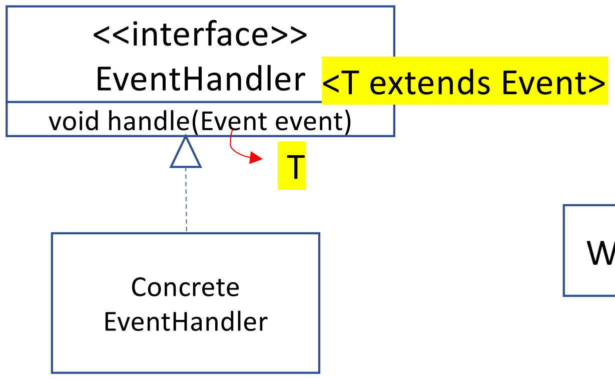
ActionEvent

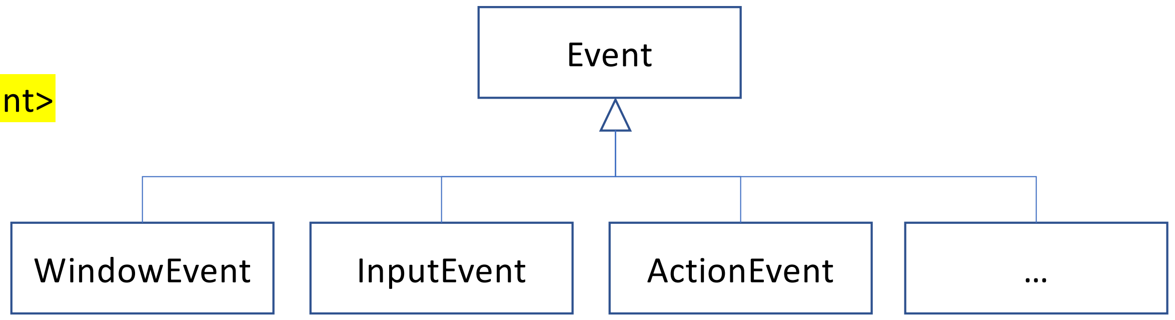
...

void handle(T event) **<= Callback method**

Invoked when a specific event of the type for which this handler is registered happens.

How to capture event and act accordingly





```java
Public class MyEventHandler implements EventHandler<ActionEvent>
{
   @Override
   public void handle(ActionEvent event)
   {
       //Event Handling steps
   }
}
```

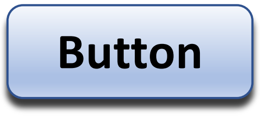
How to capture event and act accordingly

- Instantiate and register the event handler

```
MyEventHandler eventHandler = new MyEventHandler();
Button btn = new Button();
btn.setOnAction(eventHandler);
```

How to capture event and act accordingly

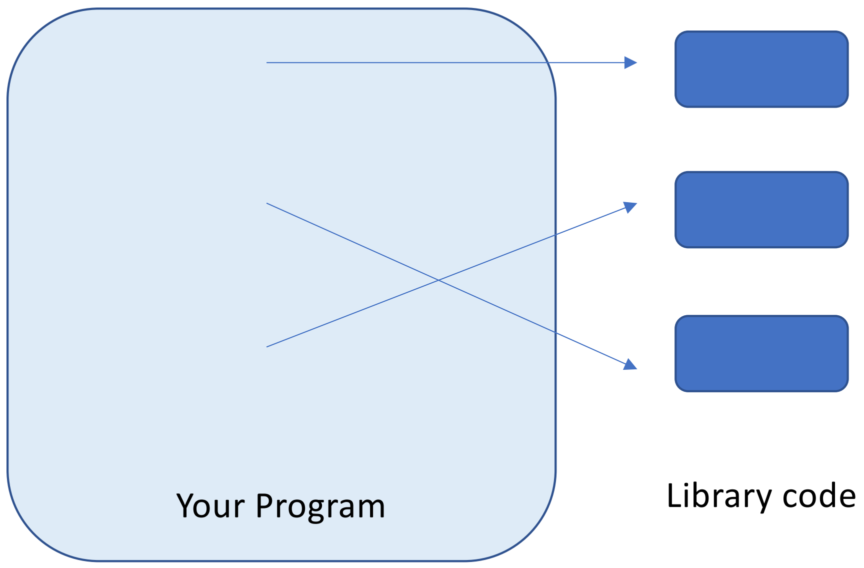
- Instantiate and register the event handler



anonymous class

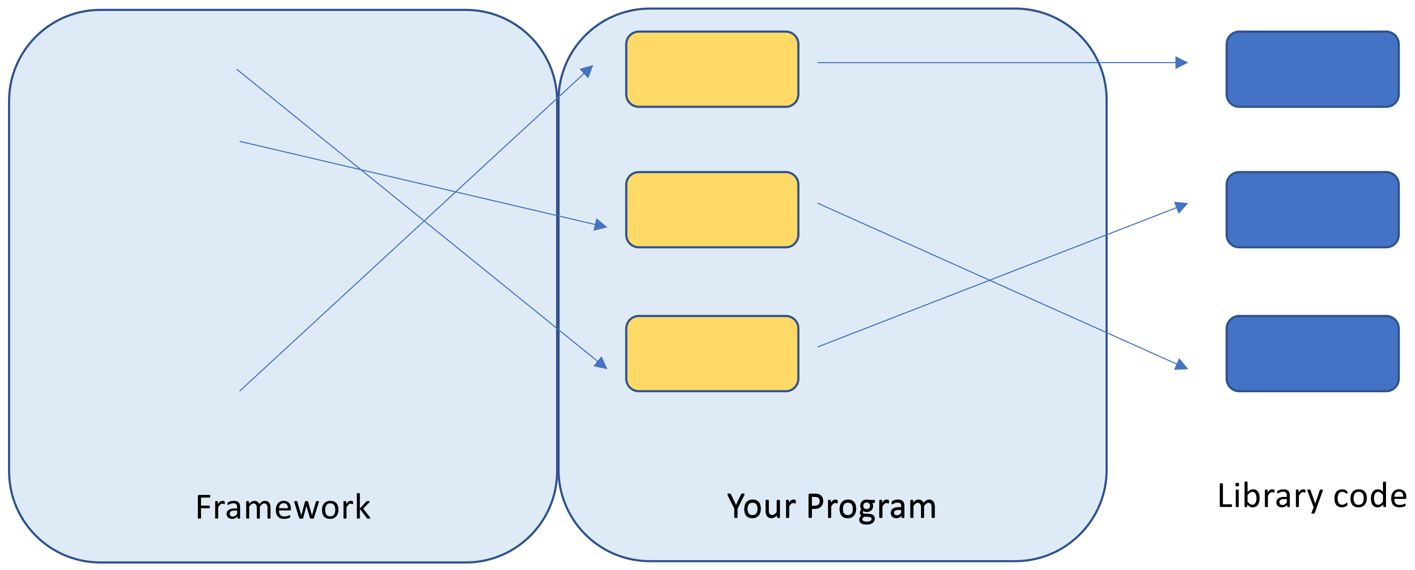
```
Button btn = new Button();

btn.setOnAction(new EventHandler<ActionEvent>() {
          public void handle(ActionEvent event) {
          //Event Handling steps
          }
      });
```

Library vs Framework



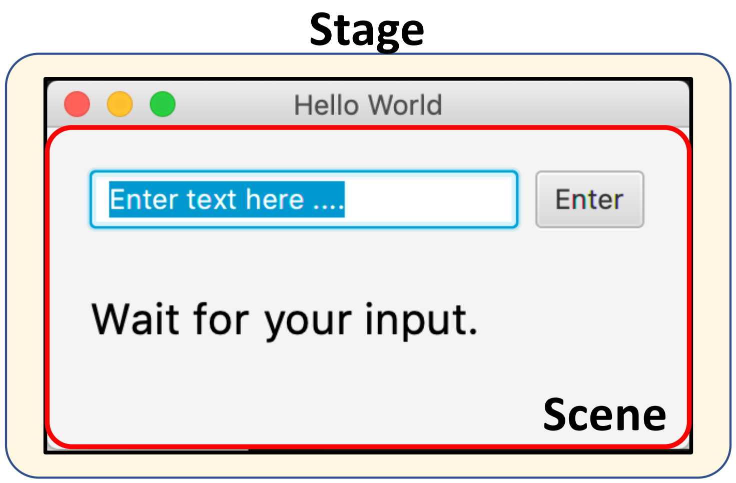
```
public static void main(String[] arg)
```

Library vs Framework



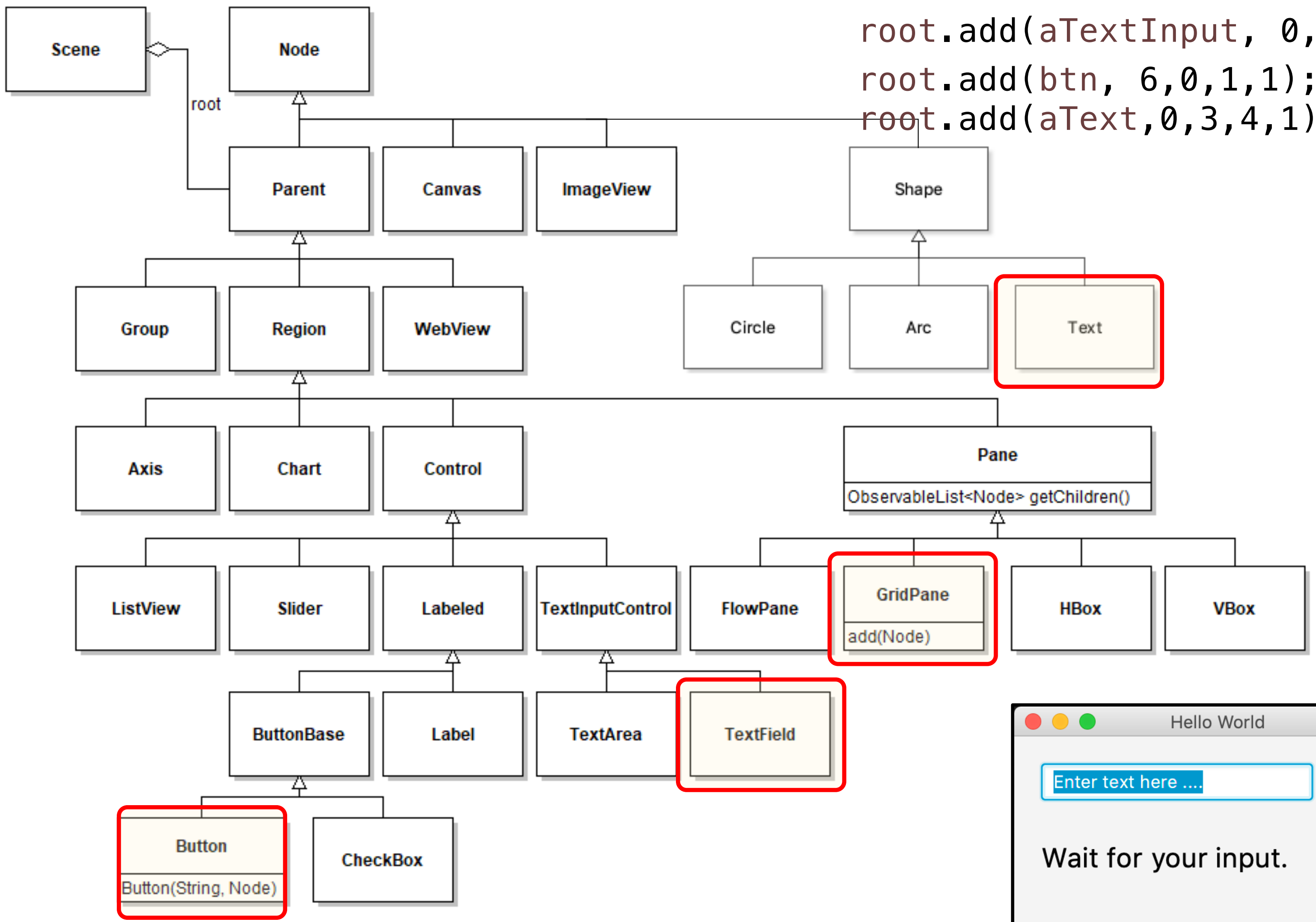
Launch JavaFX framework

```
public class MyApplication extends Application
{
    /**
    * Launches the application.
    * @param pArgs This program takes no argument.
    */
    public static void main(String[] pArgs)
    {
        launch(pArgs);
    }

    @Override
     public void start(Stage pPrimaryStage)
     {
         //Setup the stage
         pPrimaryStage.show();
     }
}
```

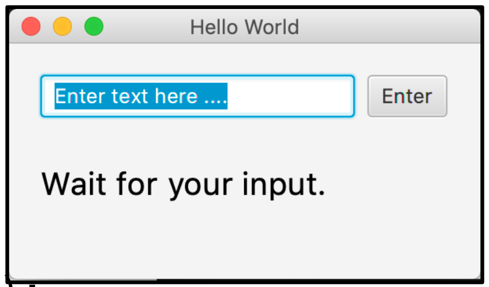


Wait for your input.

```
GridPane root = new GridPane();
root.add(aTextInput, 0,0,2,1);
root.add(btn, 6,0,1,1);
root.add(aText,0,3,4,1);

 Scene scene = new Scene(root, Width, Height)

 primaryStage.setScene(scene);
```



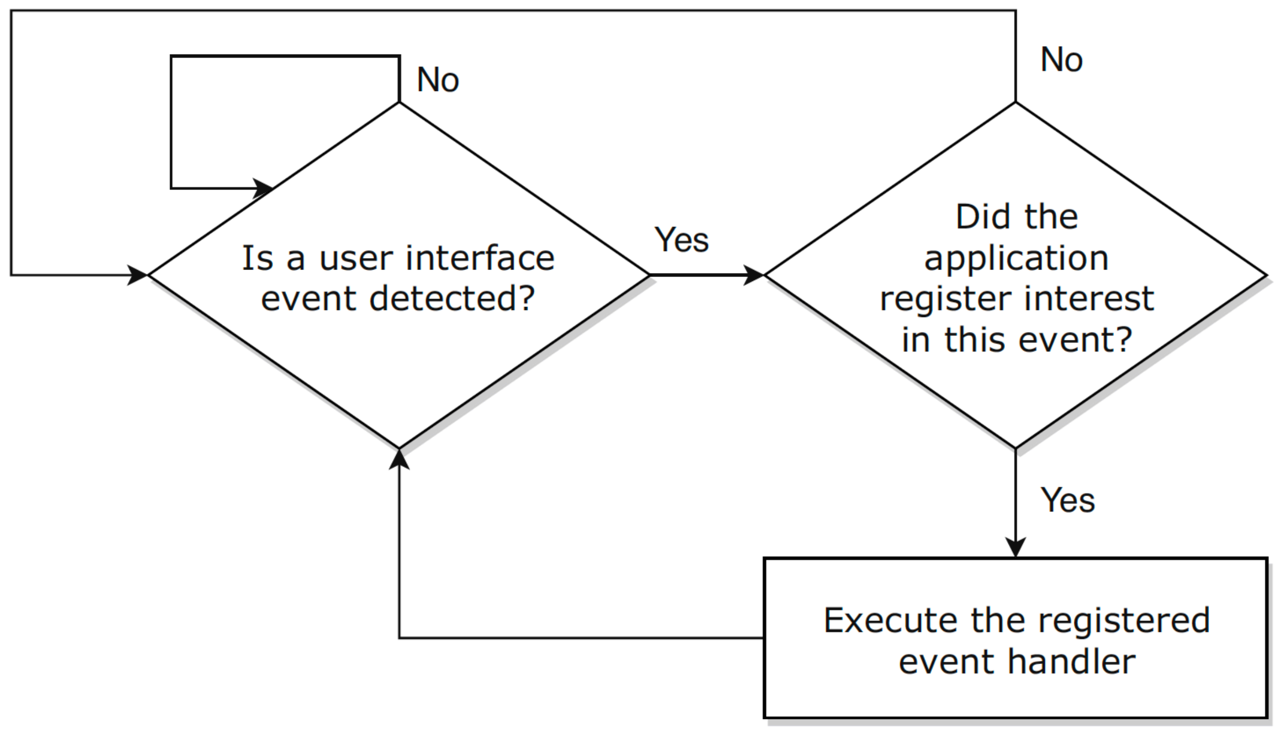
```
root.add(aTextInput, 0,0,2,1);
root.add(btn, 6,0,1,1);
root.add(aText,0,3,4,1);
```



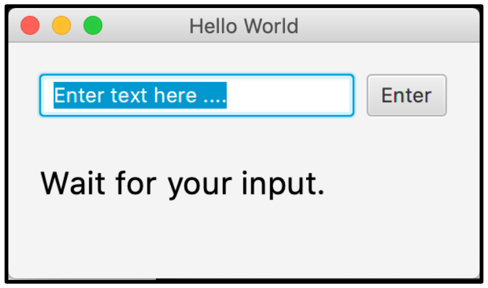
Wait for your input.

```
Scene scene = new Scene(root, Width, Height);
```

When does event handling happen?


No
No
Is a user interface event detected?
Yes
Did the application register interest in this event?
Yes
Execute the registered event handler

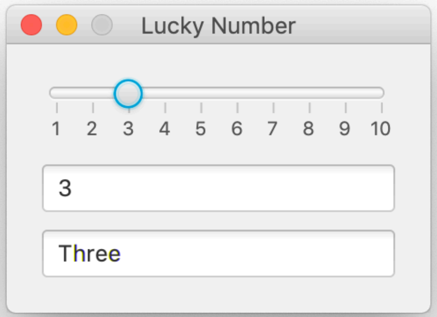
Text Display Demo



Wait for your input.

```
Text aText = new Text();
TextField aTextInput = new TextField();


aTextInput.setOnAction((actionEvent) -> aText.setText(aTextInput.getText()));

Button btn = new Button();
btn.setOnAction((actionEvent) -> aText.setText(aTextInput.getText()));
```

Lucky Number Example



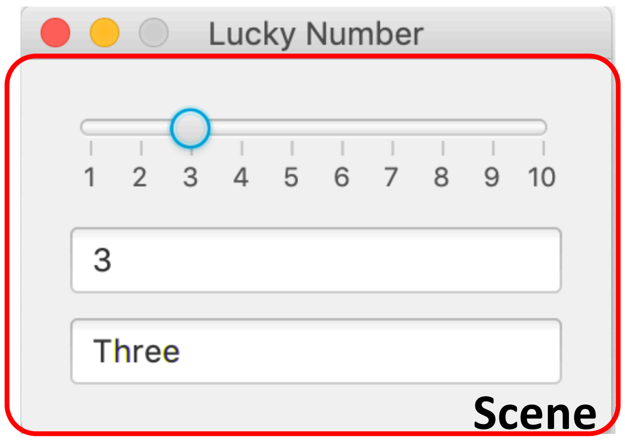
3

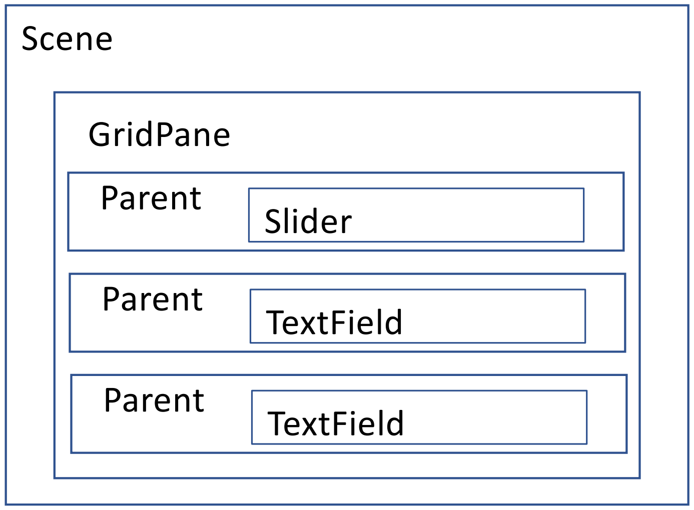
Three

The user should be able to select a number between 1 and 10 inclusively.

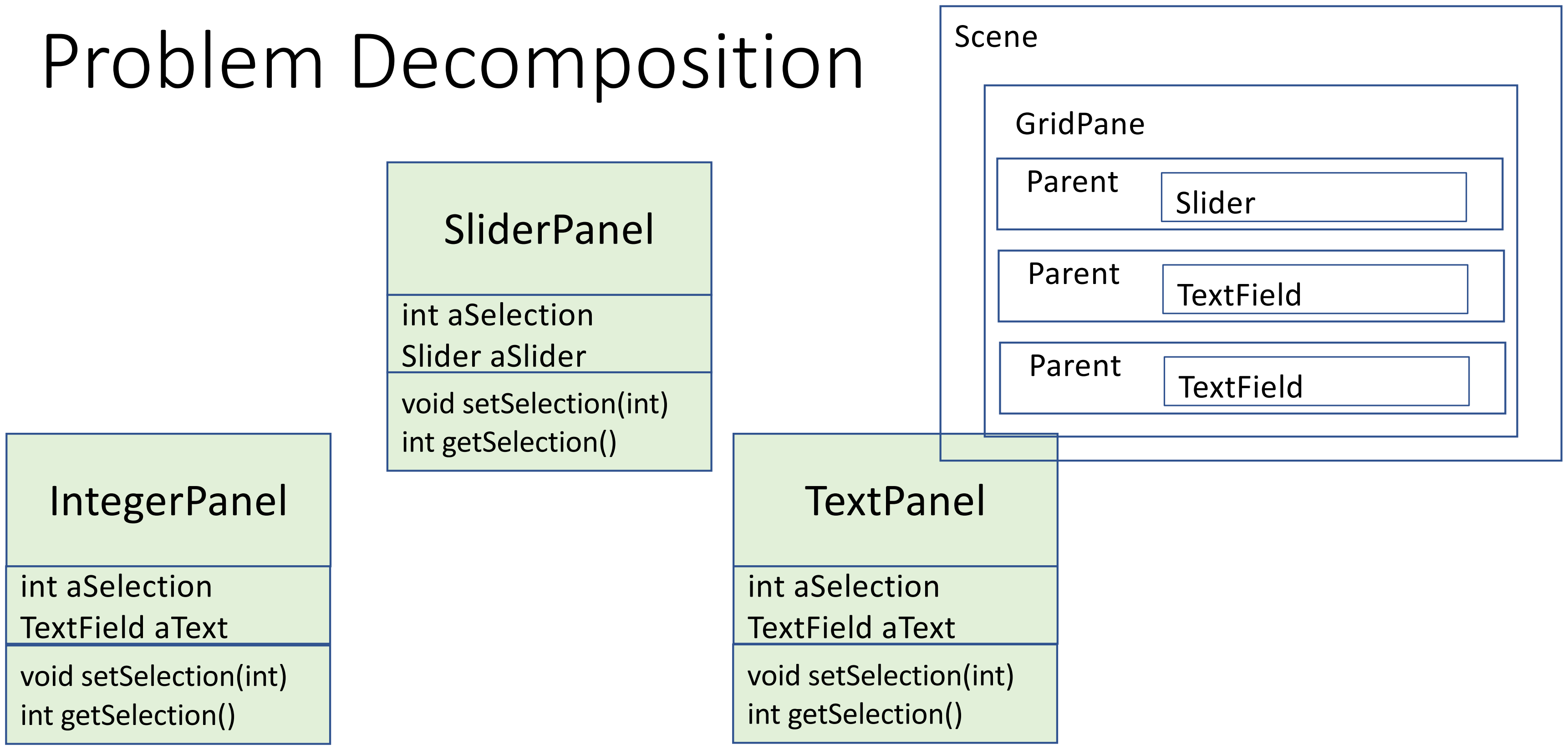
The selection should be performed through either typing it, writing it out in the corresponding fields, or selecting it from a slider.

The current selection should also be able to viewed in the integer and text fields and the slider.

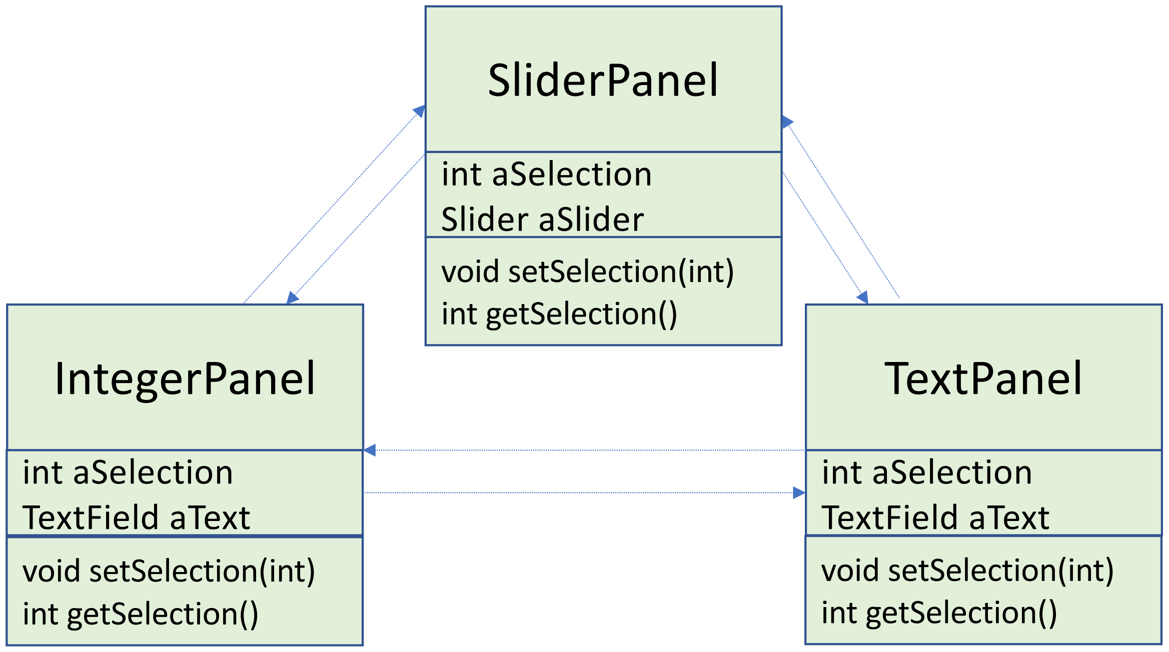

Lucky Number
1
2
3
4
5
6
7
8
9
10
3
Three
Scene


Scene
GridPane
Parent
Slider
Parent
TextField
Parent
TextField

Problem Decomposition



Problem Decomposition



High Coupling

Components are inter-dependent

Low Extensibility

hard to add/remove selection mechanism

MVC Decomposition

Model – View – Controller

Design pattern

Architectural pattern

Guideline to separate concerns

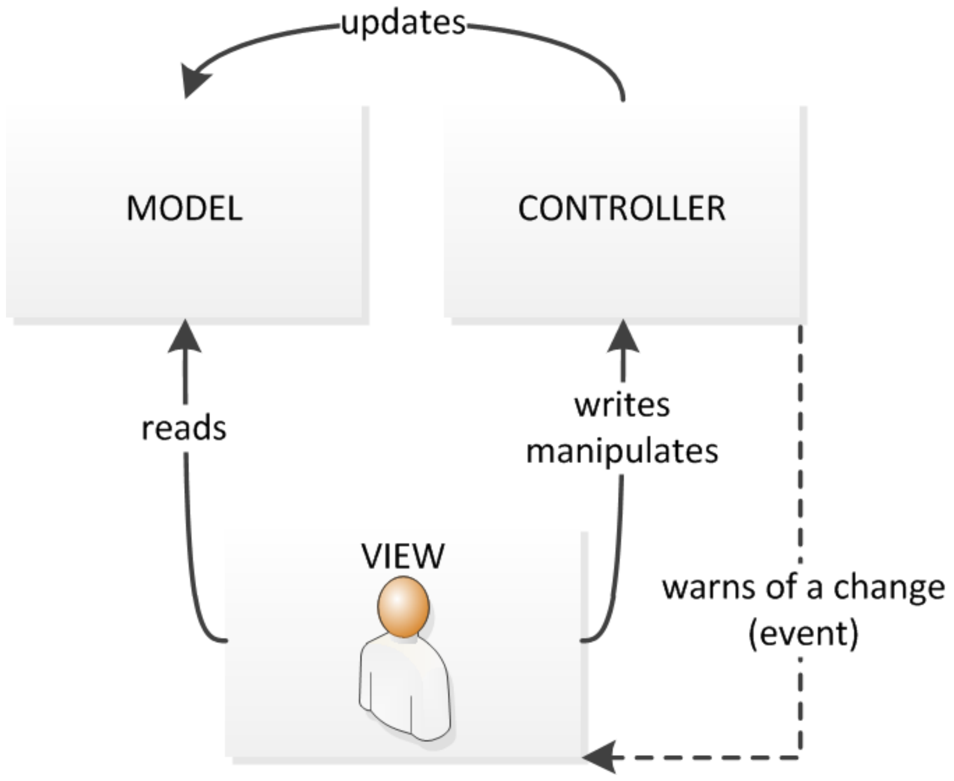
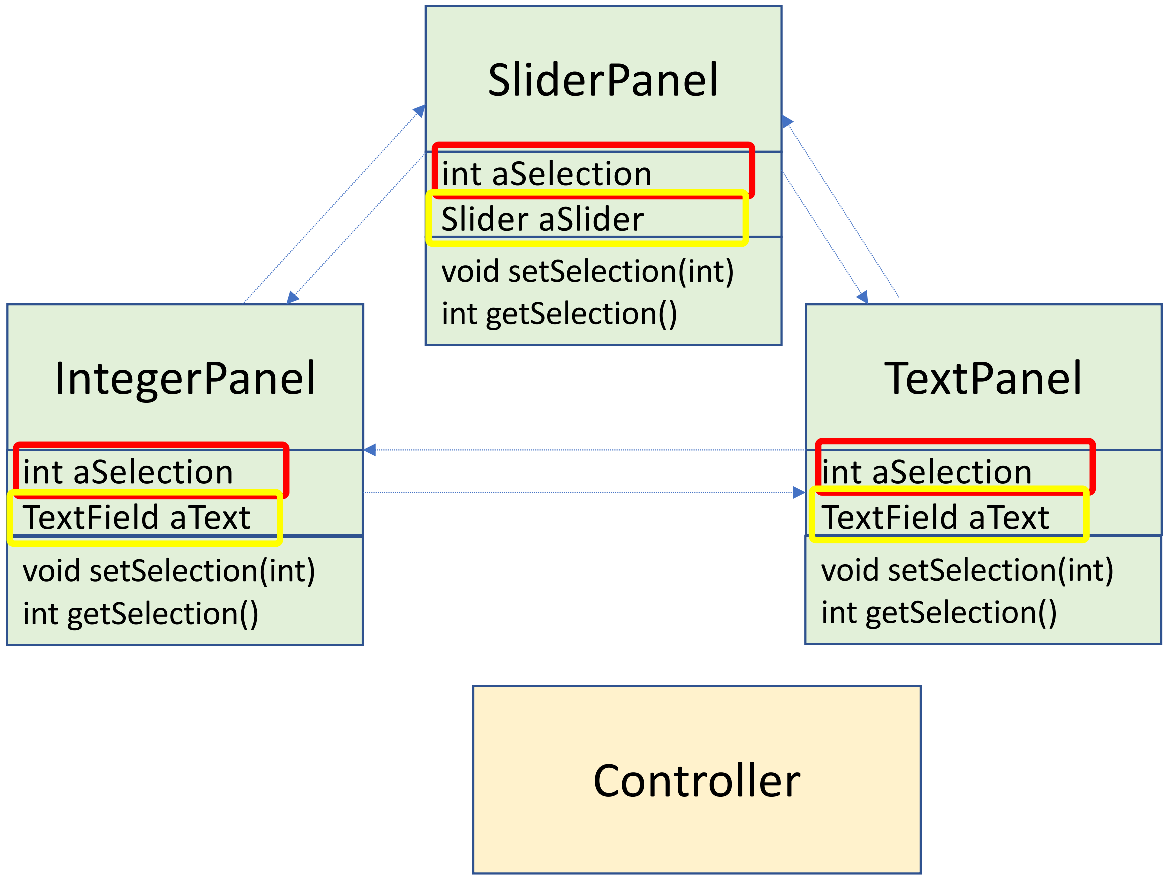
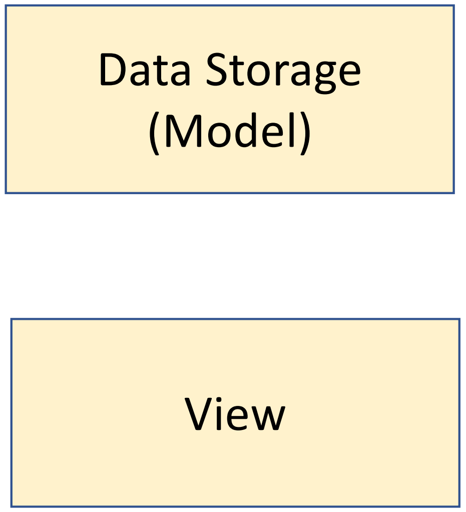


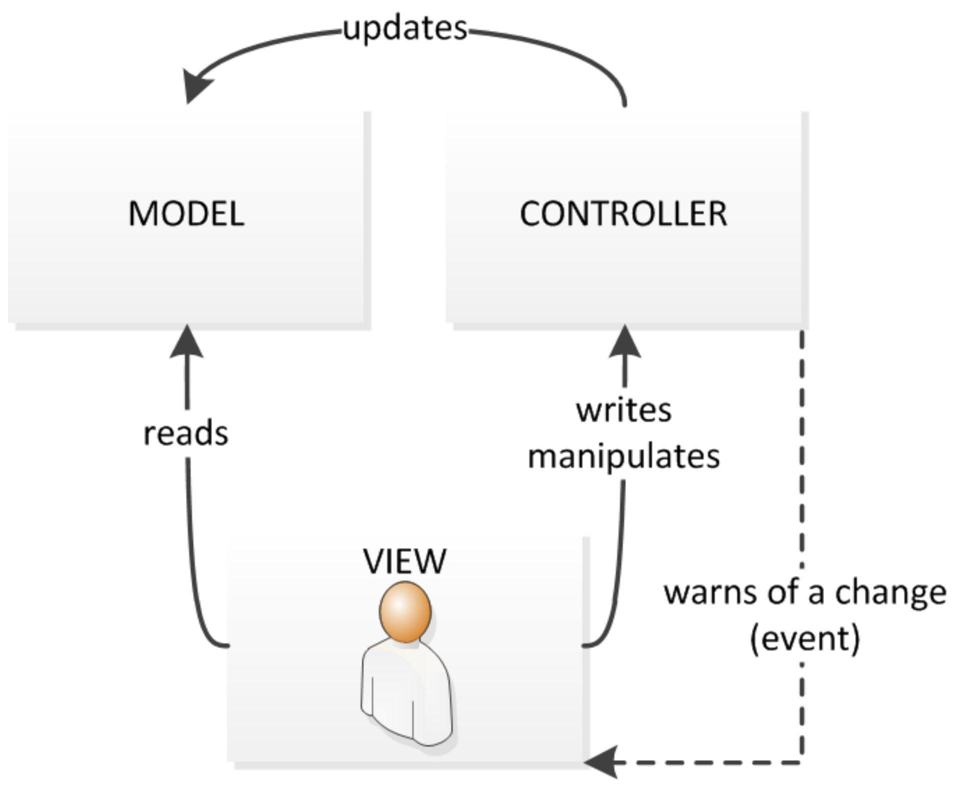
Image Source: https://upload.wikimedia.org/wikipedia/commons/6/63/ModeleMVC.png

Problem Decomposition





Problem Decomposition



Data Storage
(Model)

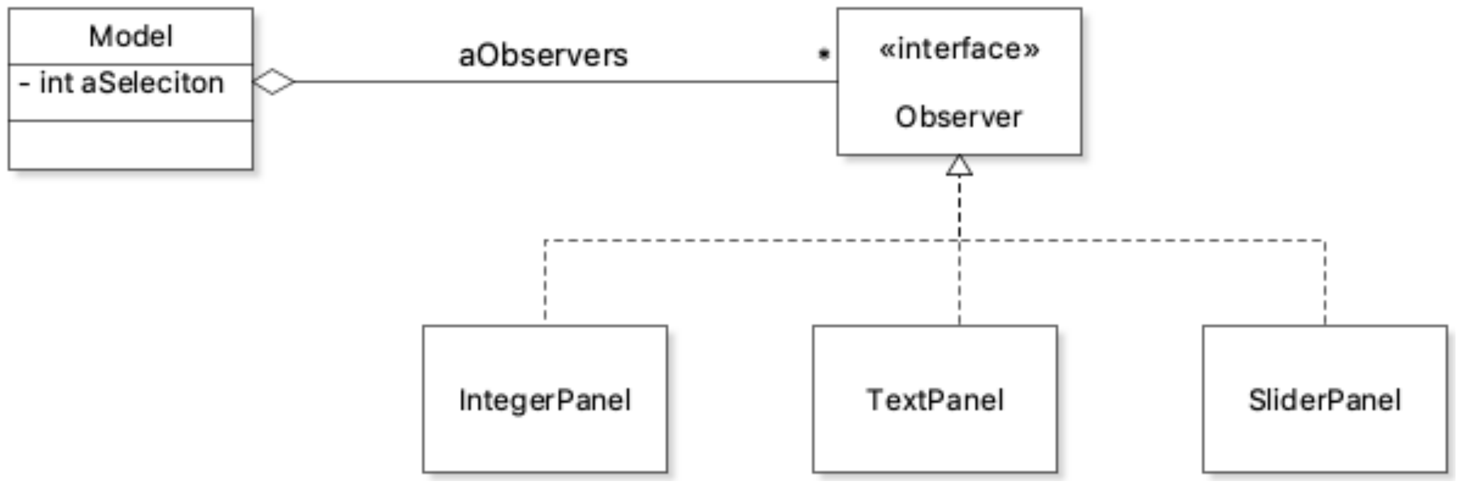
View/Controller

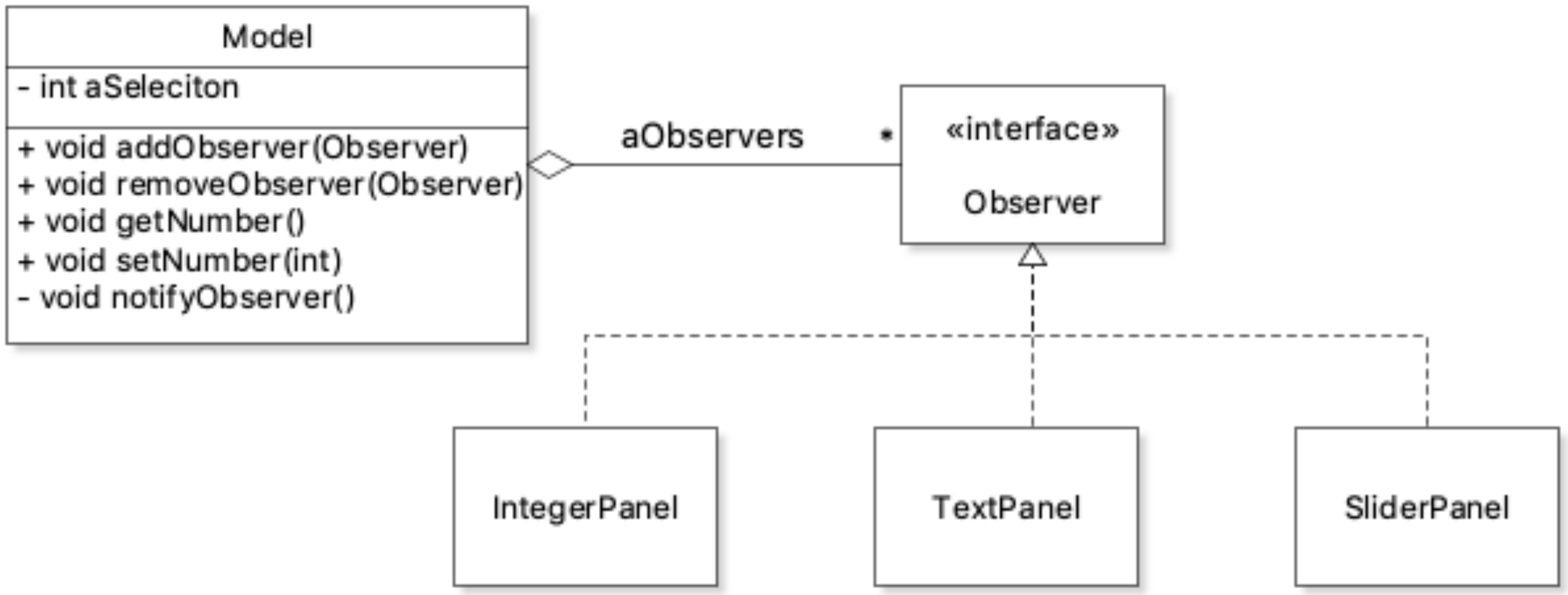
Activity

- Improve the design using Observer Pattern and MVC decomposition.

Activity: Applying Observer in MVC

- What methods should be included in Model?



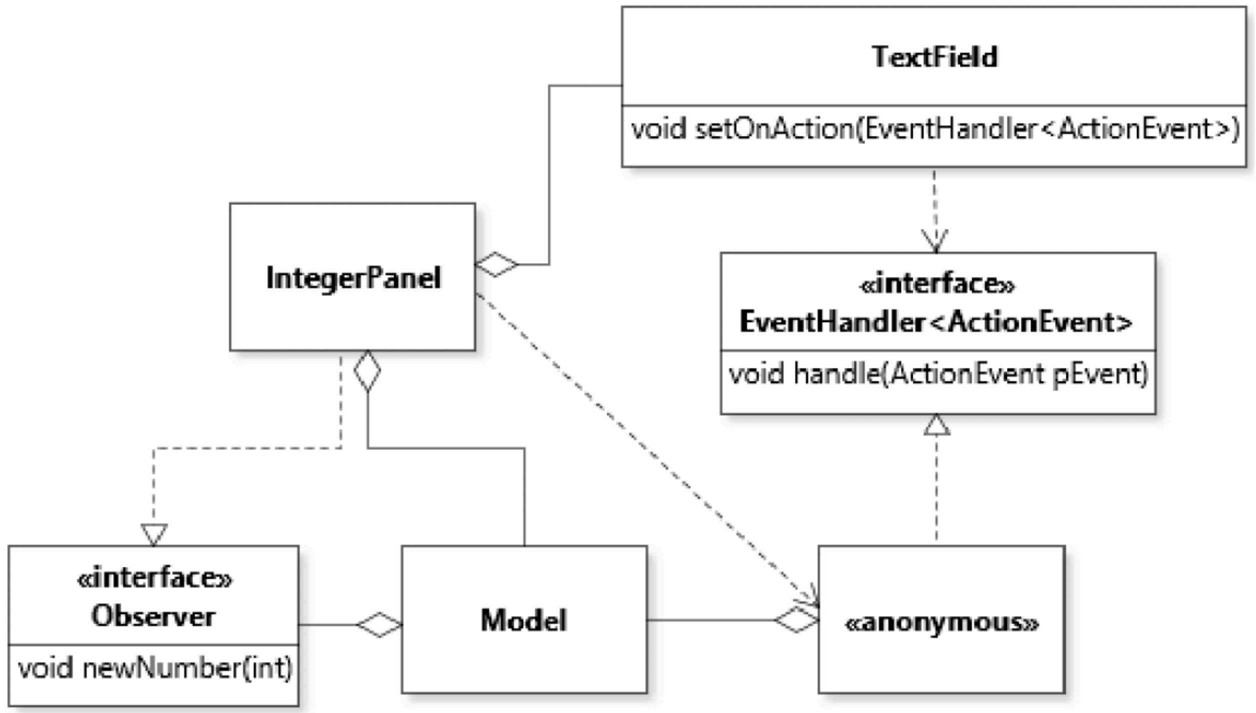

Model
- int aSeleciton
+ void addObserver(Observer)
+ void removeObserver(Observer)
+ void getNumber()
+ void setNumber(int)
- void notifyObserver()
aObservers
*
«interface»
Observer
IntegerPanel
TextPanel
SliderPanel

```java
/**
 * Abstract observer role for the model.
 */
interface Observer
{
   void newNumber(int pNumber);
}
```

```java
class IntegerPanel extends Parent implements Observer
{
   private TextField aText = new TextField();
   private Model aModel;

   … …
   @Override
   public void newNumber(int pNumber)
   {
      aText.setText(new Integer(pNumber).toString());
   }
}
```

Call `aModel.setNumber(lInteger);`

```
/**
 * Constructor.
 */
IntegerPanel(Model pModel)
{
   aModel = pModel;
   aModel.addObserver(this);
   aText.setMinWidth(LuckyNumber.WIDTH);
   aText.setText(new Integer(aModel.getNumber()).toString());
   getChildren().add(aText);

   aText.setOnAction(new EventHandler<ActionEvent>(){
      @Override
      public void handle(ActionEvent pEvent){
         int lInteger = 1;
         try{
            lInteger = Integer.parseInt(aText.getText());
         } catch(NumberFormatException pException ){
            //Code to handle exception
         }
         aModel.setNumber(lInteger);
      }
   });
}
```


TextField
void setOnAction(EventHandler<ActionEvent>)
IntegerPanel
«interface»
EventHandler<ActionEvent>
void handle(ActionEvent pEvent)
«interface»
Observer
void newNumber(int)
Model
«anonymous»

Objective

- Be able to Use Callback to achieve decoupling
- Be able to use the Observer design pattern effectively;
- Event Handling in GUI applications
- Understand the concept of an application framework;
- Understand the Model-View-Controller Decomposition