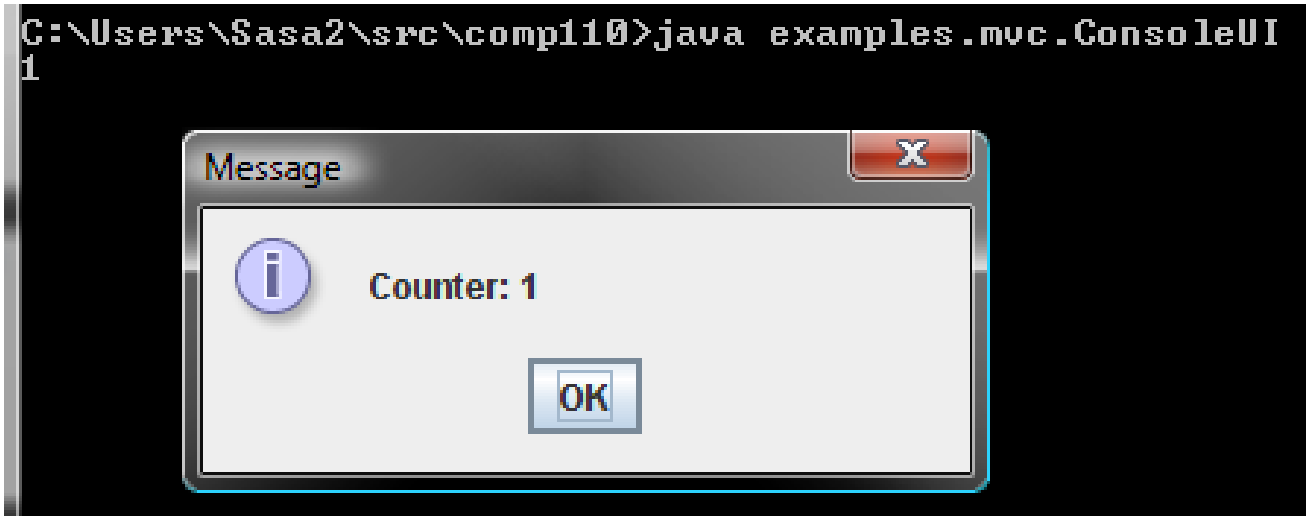
Comp 401
MVC and User-Interface Toolkits

Instructor: Prasun Dewan

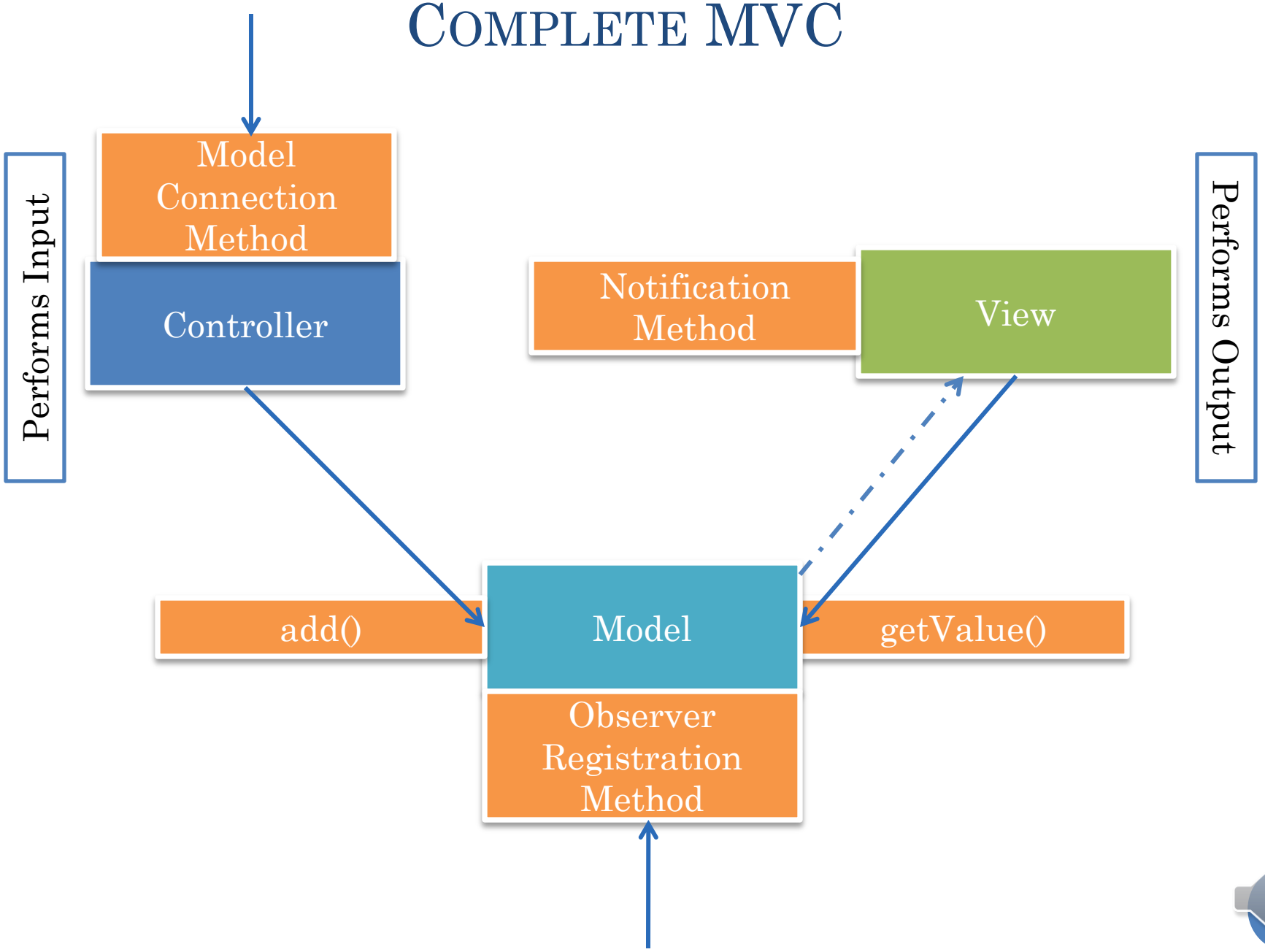
Prerequisites

- MVC

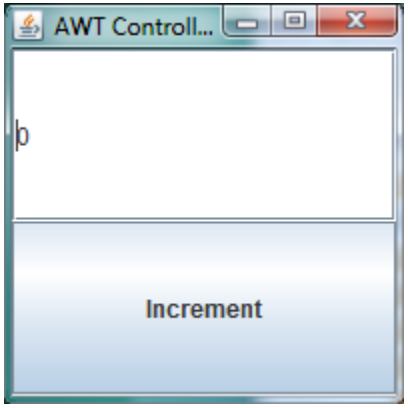
Counter Application

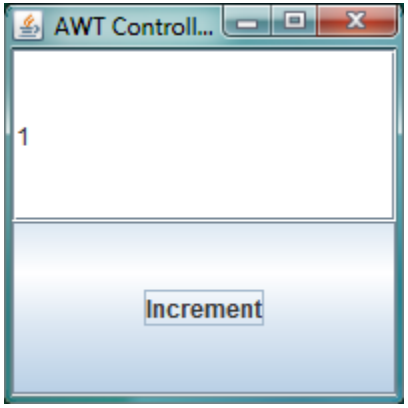


Complete MVC

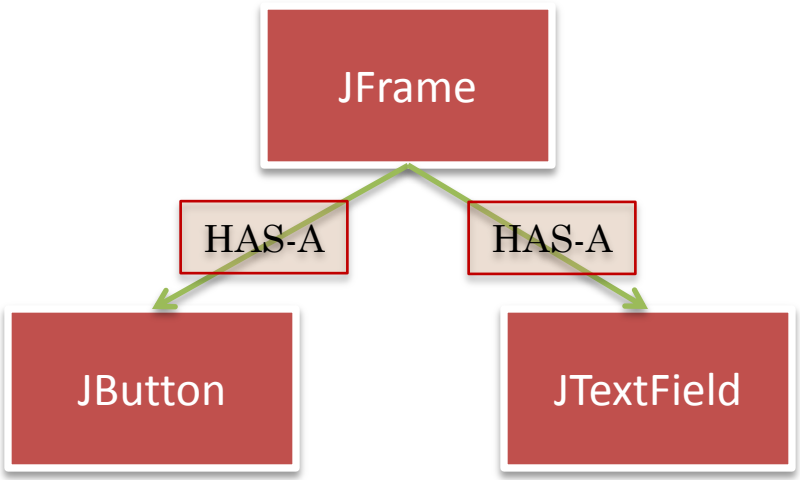


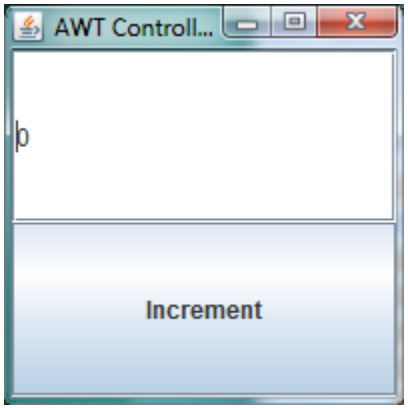
More Sophisticated AWT/Swing Counter User Interface





Widgets and Display Logical Structure





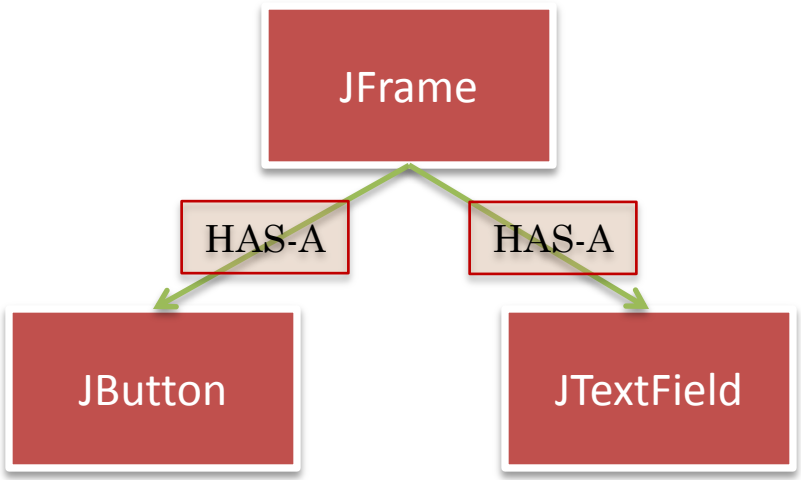
Layout?

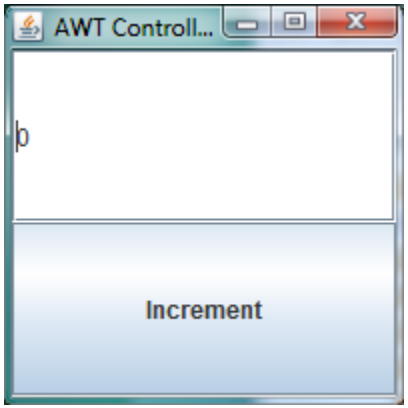
AWT Single Frame Composer

```
public class SingleFrameAWTComposer {
  public static void main (String args[]) {
  // compose AWT components
    JFrame frame = new JFrame("AWT Controller and View");
    JButton button = new JButton("Increment");
    JTextField textField = new JTextField();
    frame.setLayout(new GridLayout(2,1));
    frame.add(textField);
    frame.add(button);
    frame.setSize(200, 200);
    frame.setVisible(true);
    //compose model view and controller
    ObservableCounter model = new AnObservableCounter();
    …
   }
}
```

Separate layout object

Widgets and Display Logical Structure





HAS-A == is parent of in logical structure

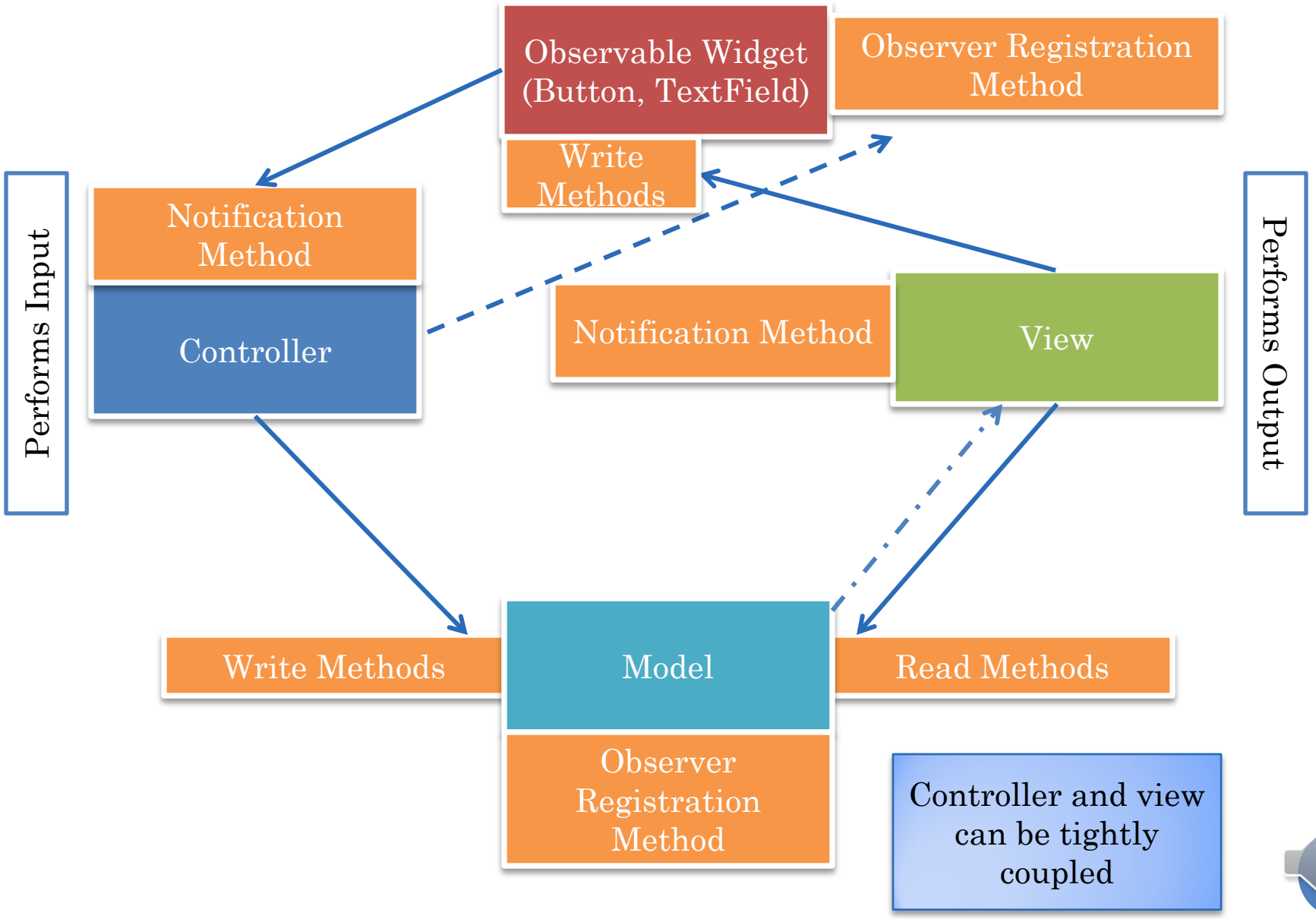
Has a reference to

Output vs. Input widgets?

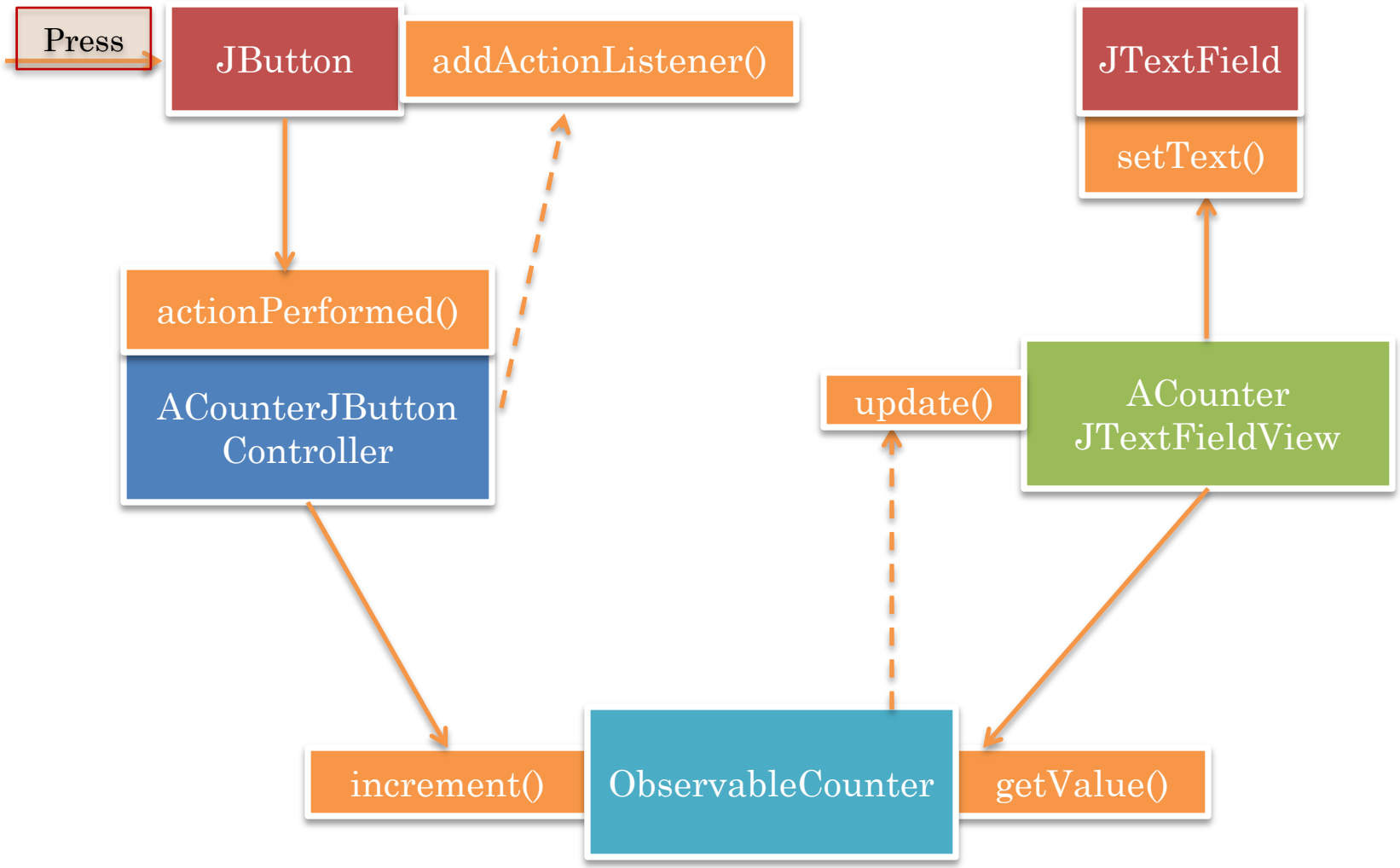
How should the view tell the text field to display a new String.

How should the button tell a controller that it has been pressed so it can increment the counter?

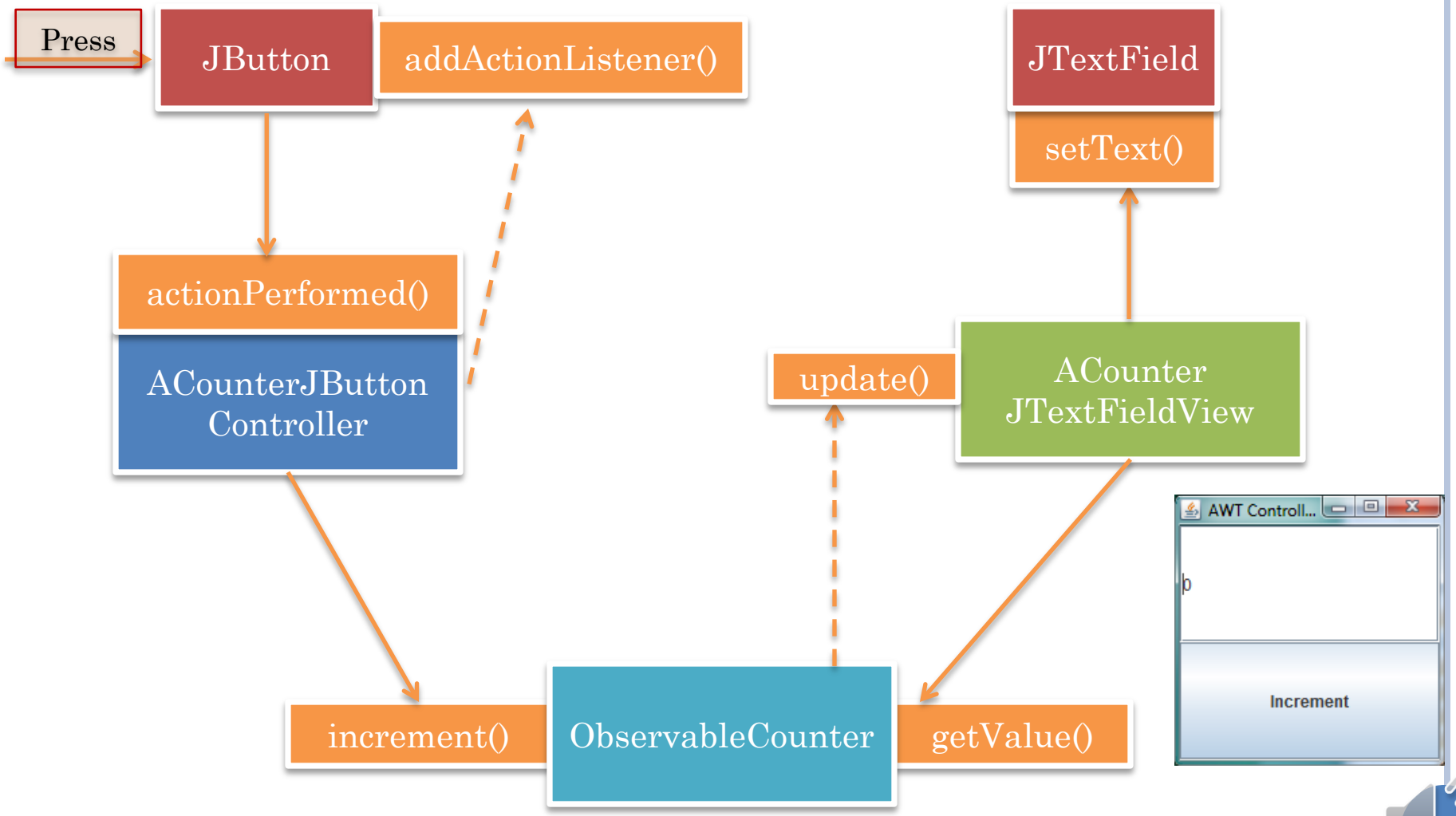
MVC and Toolkits



MVC in Example Application



MVC in Example Application (Review)



Observer Pattern in Java Toolkit

```
import  java.awt.ActionListener;
public void addActionListener (ActionListener actionListener)
```

```
import java.awt.ActionEvent;
public interface ActionListener {
        public void actionPerformed(ActionEvent e);
}
```

When you edit text and hit return this event sent by JTextField, TextField widget to its listeners such as ObjectEditor

When you press a button, this event sent by Button/Jbutton to its listeners such as ObjectEditor

Counter Observable

```
public class AnObservableCounter implements ObservableCounter {
        int counter = 0;
        ObserverList observers = new AnObserverList();
        public void add (int amount) {
                counter += amount;
                notifyAllObservers();
        }
        public int getValue() {
                return counter;
        }
        public void addObserver(CounterObserver observer) {
                observers.addElement(observer);
                observer.update(this);
        }
        public void removeObserver(CounterObserver observer) {
                observers.removeElement(observer);
        }
        void notifyAllObservers() {
                for (int observerNum = 0; observerNum < observers.size();
                        observerNum++)
                observers.elementAt(observerNum).update(this);
        }
}
```

Button Controller also an Observer

```
public class ACounterJButtonController implements
CounterController, ActionListener {
  JButton button;
  ObservableCounter counter;
  public ACounterJButtonController(JButton theButton) {
    button = theButton;
    button.addActionListener(this);
  }
  public void setModel (ObservableCounter theCounter) {
    counter = theCounter;
  }
  // do not need to implement anything in this controller
  public void processInput() {

  }
  //ActionListener method
  public void actionPerformed(ActionEvent arg0) {
    counter.add(1);
  }
}
```

TextField View Invoking Widget Write Methods

```
public class ACounterJTextFieldView implements
CounterObserver {
   JTextField textField;
   public ACounterJTextFieldView(JTextField theTextField){
     textField = theTextField;
   }
   public void update(ObservableCounter counter) {
     textField.setText("" + counter.getValue());
   }
}
```

Dual Role Composer

```
public class SingleFrameAWTComposer {
  public static void main (String args[]) {
  // compose AWT components
    JFrame frame = new JFrame("AWT Controller and View");
    JButton button = new JButton("Increment");
    JTextField textField = new JTextField();
    frame.setLayout(new GridLayout(2,1));
    frame.add(textField);
    frame.add(button);
    frame.setSize(200, 200);
    frame.setVisible(true);
    //compose model view and controller
    ObservableCounter model = new AnObservableCounter();
    CounterObserver view =
                    new ACounterJTextFieldView(textField);
    model.addObserver(view);
    CounterController controller =
                    new ACounterJButtonController(button);
    controller.setModel(model);
  }
}
```

Window Systems and UI Toolkits

- Window system
  - Provides hierarchical rectangular areas called windows
  - Windows have no associated behavior
- User Interface toolkit
  - Provides widgets: windows with some behavior
    - E.g: JTextField, Jslider, JMenuItem, JMenu
  - Widgets are observables and the ones we have seen
    - Support action listeners through the addActionListener method
    - Notify listeners about ActionEvent
  - Widgets provide writemethods to change their state
    - setText(), setValue()
- Window Toolkit
  - Window system + user interface toolkit

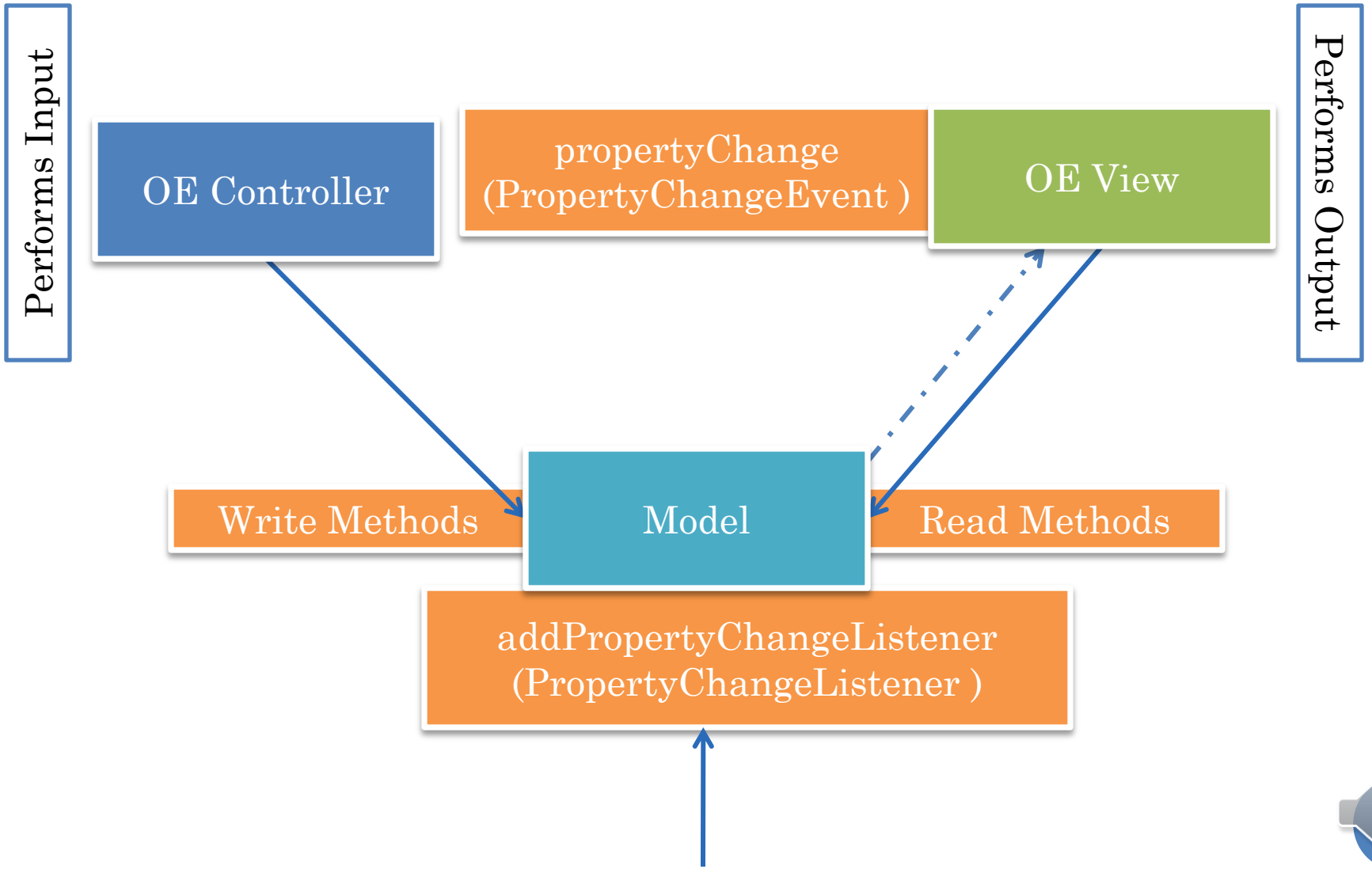
MVC with Toolkit Widgets for I/O

- Like regular MVC:
  - Controller for input, View for output, and Model for semantics
  - Main program instantiates model, views and controllers and connects these objects together
- Additional twists;
  - The main program also builds a widget/window tree
  - A controller
    - gets in its constructor (or through a setter method) a set of input widgets such JTextField, JMenuItem and registers itself as a listener of these widgets
    - On receiving event from input widgets, calls write methods in the model
  - A view
    - gets in its constructor a set of output widgets such as JTextField
    - calls write methods in them when it receives notifications from the model

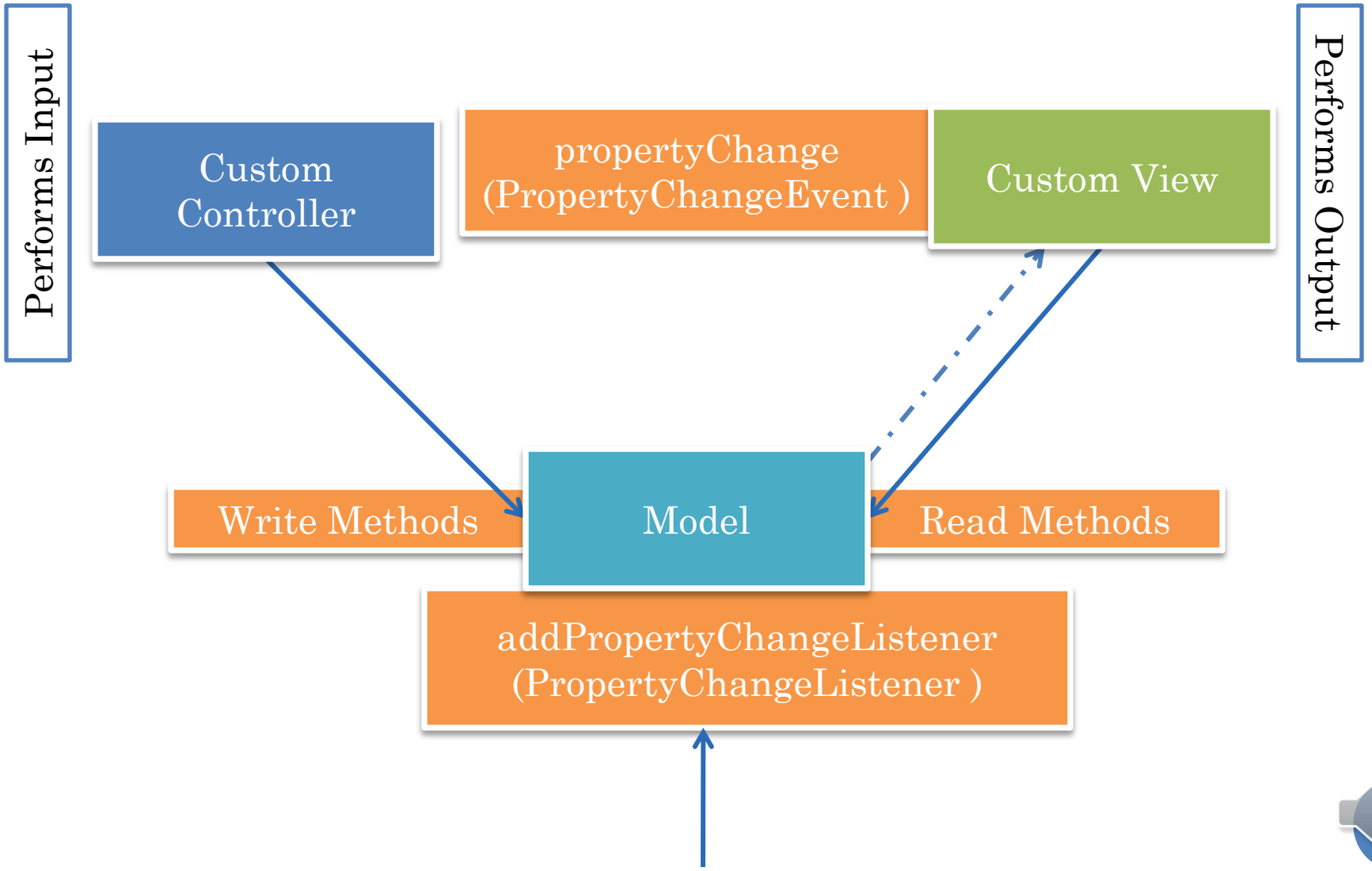
Controller/View Roles with widgets

- Controller:
  - Receives events from widgets and calls **read methods** in the widgets
  - Calls **write methods** in the model
  - Essentially reads widget state and writes model state: widget state → model state)
- View
  - Receives notifications in the model
  - Calls **read methods** in the model
  - Calls **write methods** in the widgets
  - Essentially reads model state and writes widget state: model state → widget state

Writing Only the Observable



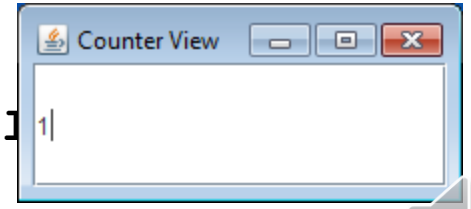
The Full App


Performs Input
Performs Output
Custom Controller
propertyChange (PropertyChangeEvent)
Custom View
Write Methods
Model
Read Methods
addPropertyChangeListener (PropertyChangeListener)

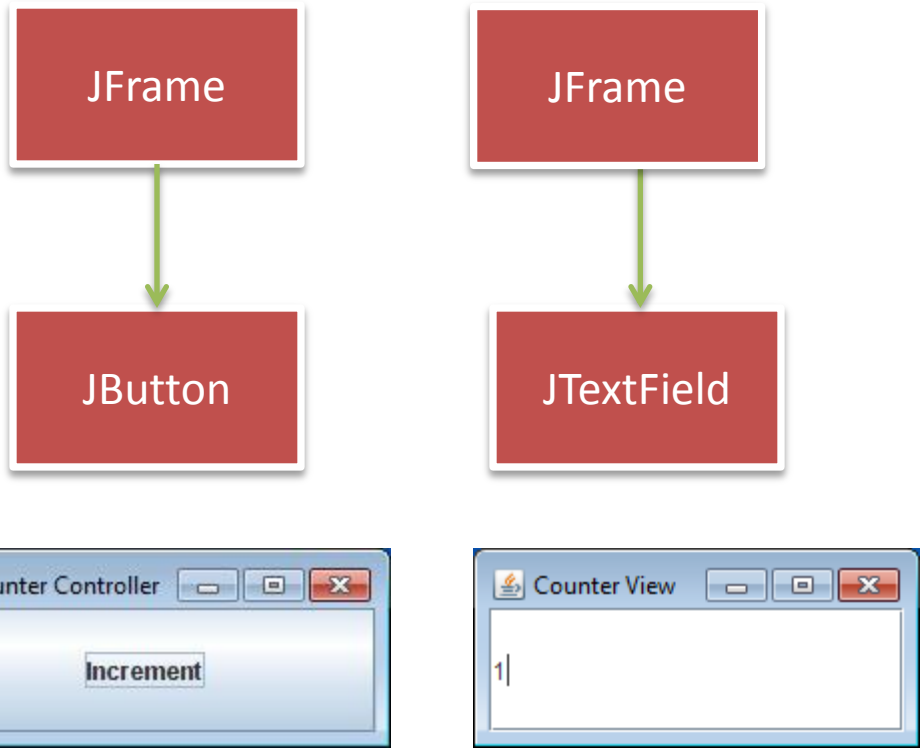
AWT Single Frame Composer

```
public class SingleFrameAWTComposer {
  public static void main (String args[]) {
  // compose AWT components
    JFrame frame = new JFrame("AWT Controller and View");
    JButton button = new JButton("Increment");
    JTextField textField = new JTextField();
    frame.setLayout(new GridLayout(2,1));
    frame.add(textField);
    frame.add(button);
    frame.setSize(200, 200);
    frame.setVisible(true);
    //compose model view and controller
    ObservableCounter model = new AnObservabl
    CounterObserver view =
                    new ACounterJTextFieldView
    model.addObserver(view);
    CounterController controller =
                    new ACounterJButtonControll
    controller.setModel(model);
  }
}
```

Separate button and text field into different frames?



Widgets and Display Logical Structure

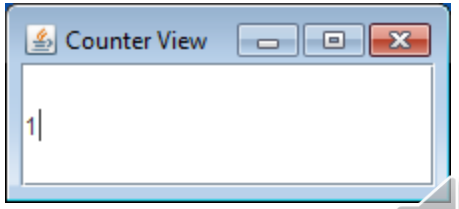


AWT Multi Frame Composer

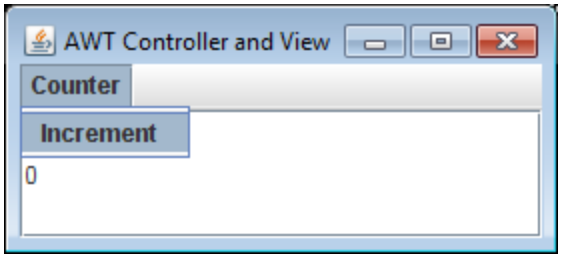
```
public class MultiFrameButtonTextFieldAWTComposer {
  public static void main (String args[]) {
    ObservableCounter model = new AnObservableCounter();
    CounterController controller = createControllerAndControllerFrame();
    CounterObserver view = createViewAndViewFrame();
    model.addObserver(view);
    controller.setModel(model);
  }
  public static CounterController createControllerAndControllerFrame() {
    JFrame frame = new JFrame("Counter Controller");
    JButton button = new JButton("Increment");
    frame.add(button);
    frame.setSize(200, 100);
    frame.setVisible(true);
    return new ACounterJButtonController(button);
  }
  public static CounterObserver createViewAndViewFrame() {
    JFrame frame = new JFrame("Counter View");
    JTextField textField = new JTextField();
    frame.add(textField);
    frame.setSize(200, 100);
    frame.setVisible(true);
    return new ACounterJTextFieldView(textField);
   }
}
```

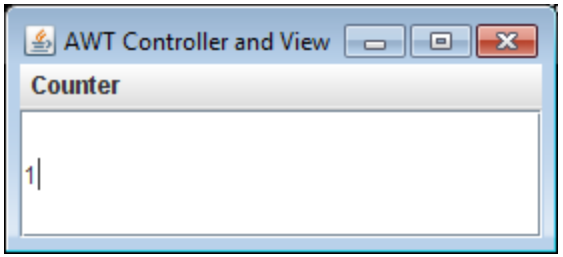
Separation of controller and view allows reuse by writing a different composer

If two pieces can change independently, separate them

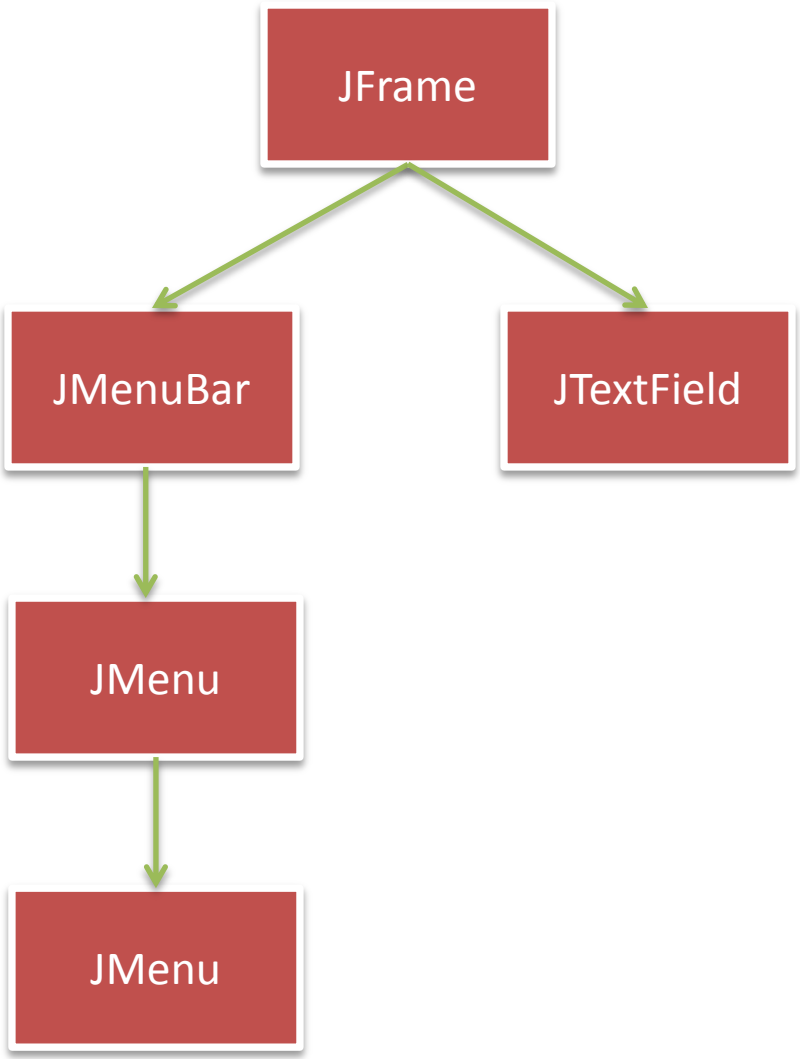


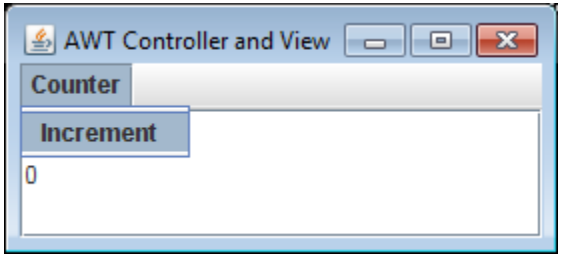
Reusing View with Different Input Widget



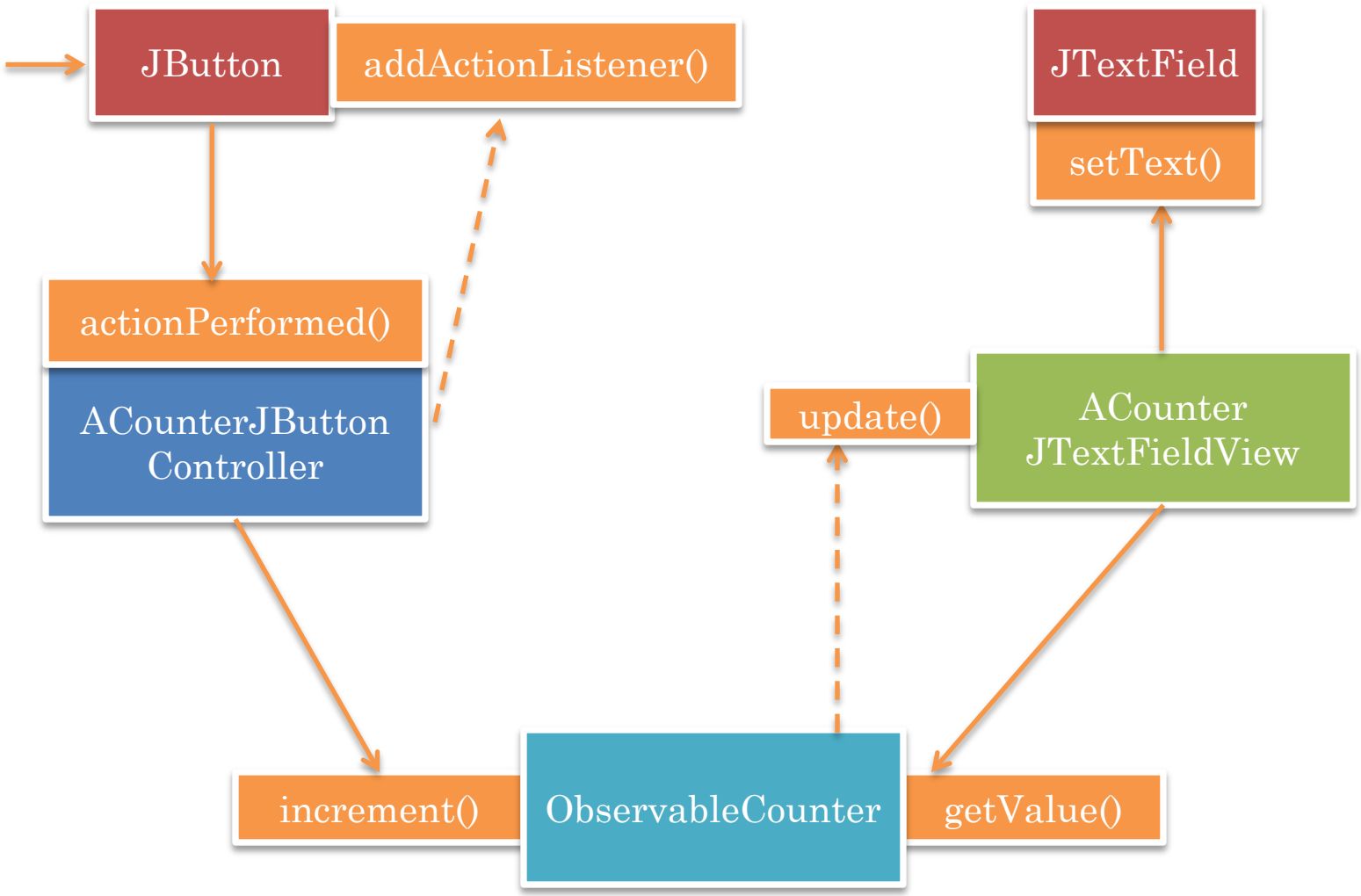


Widgets and Display Logical Structure

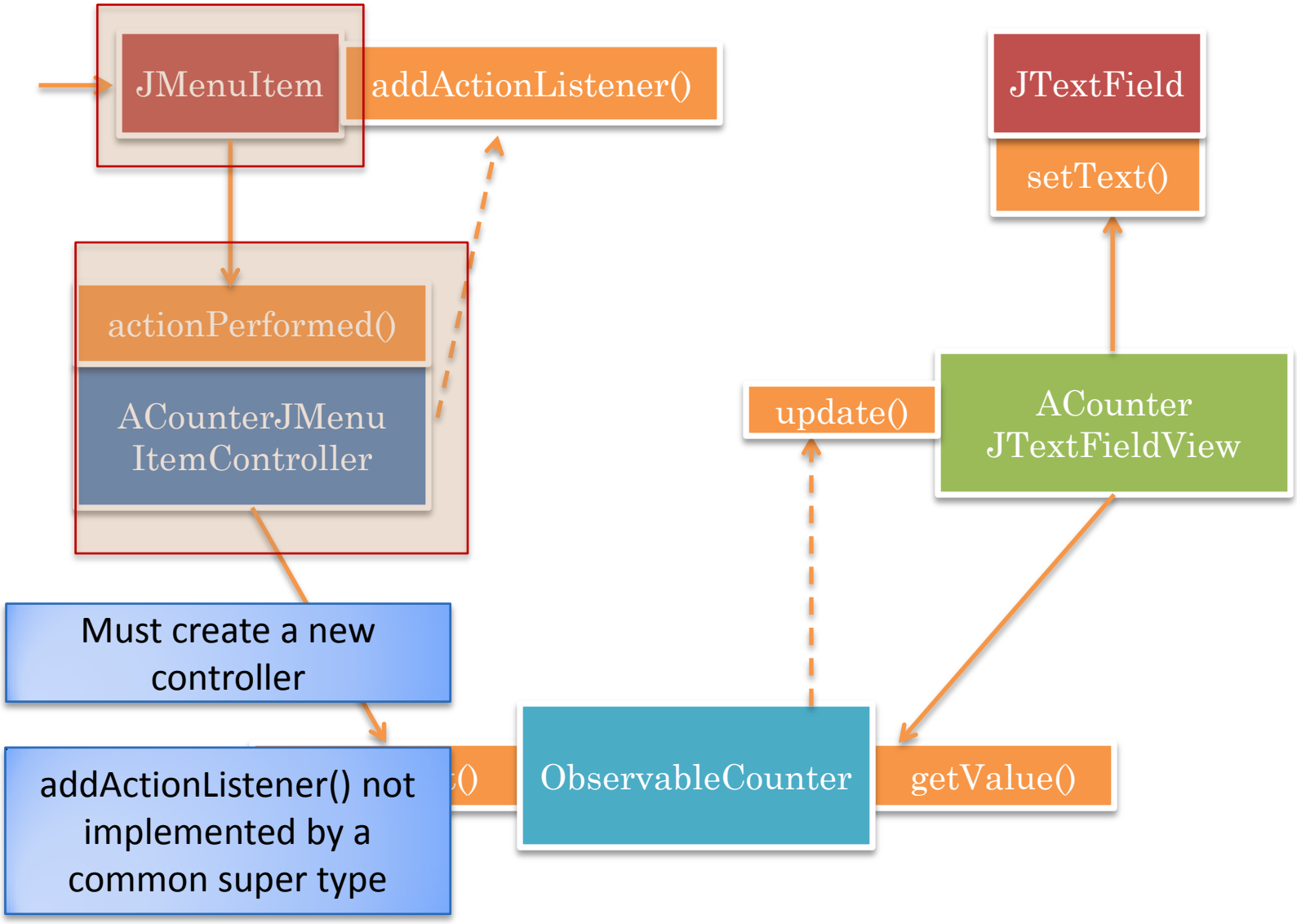




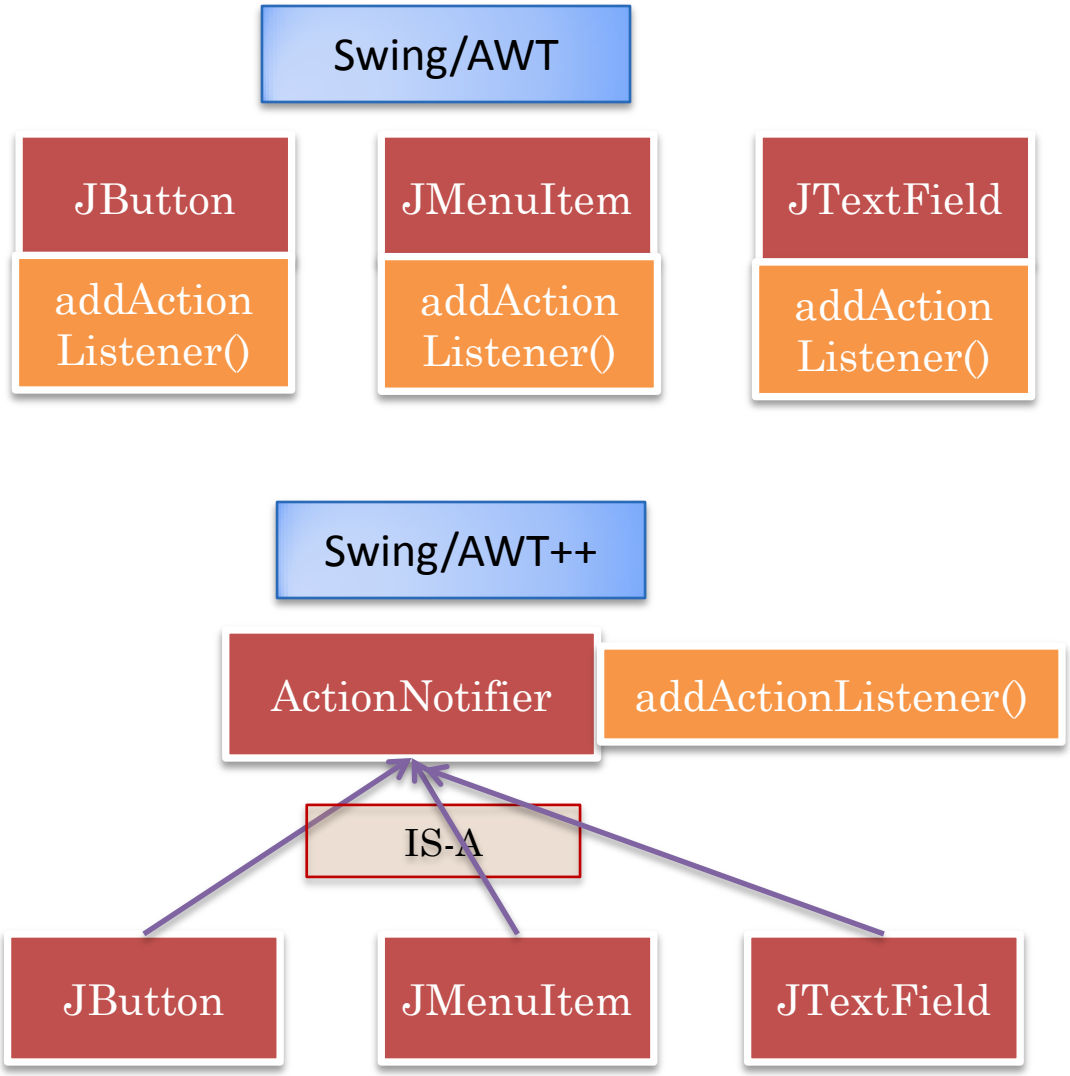
Original Controller and Input Widget



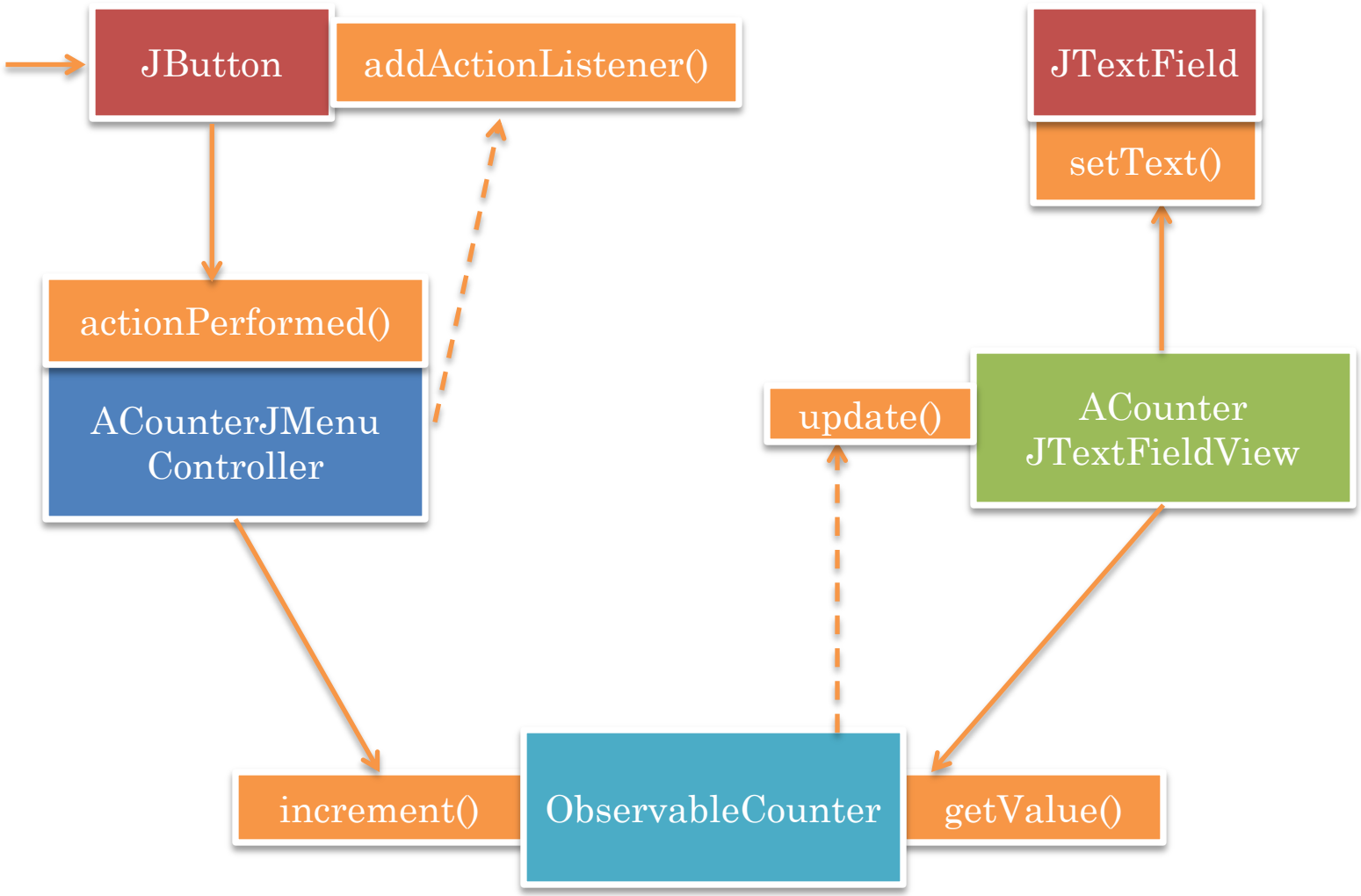
New Controller and Input Widget



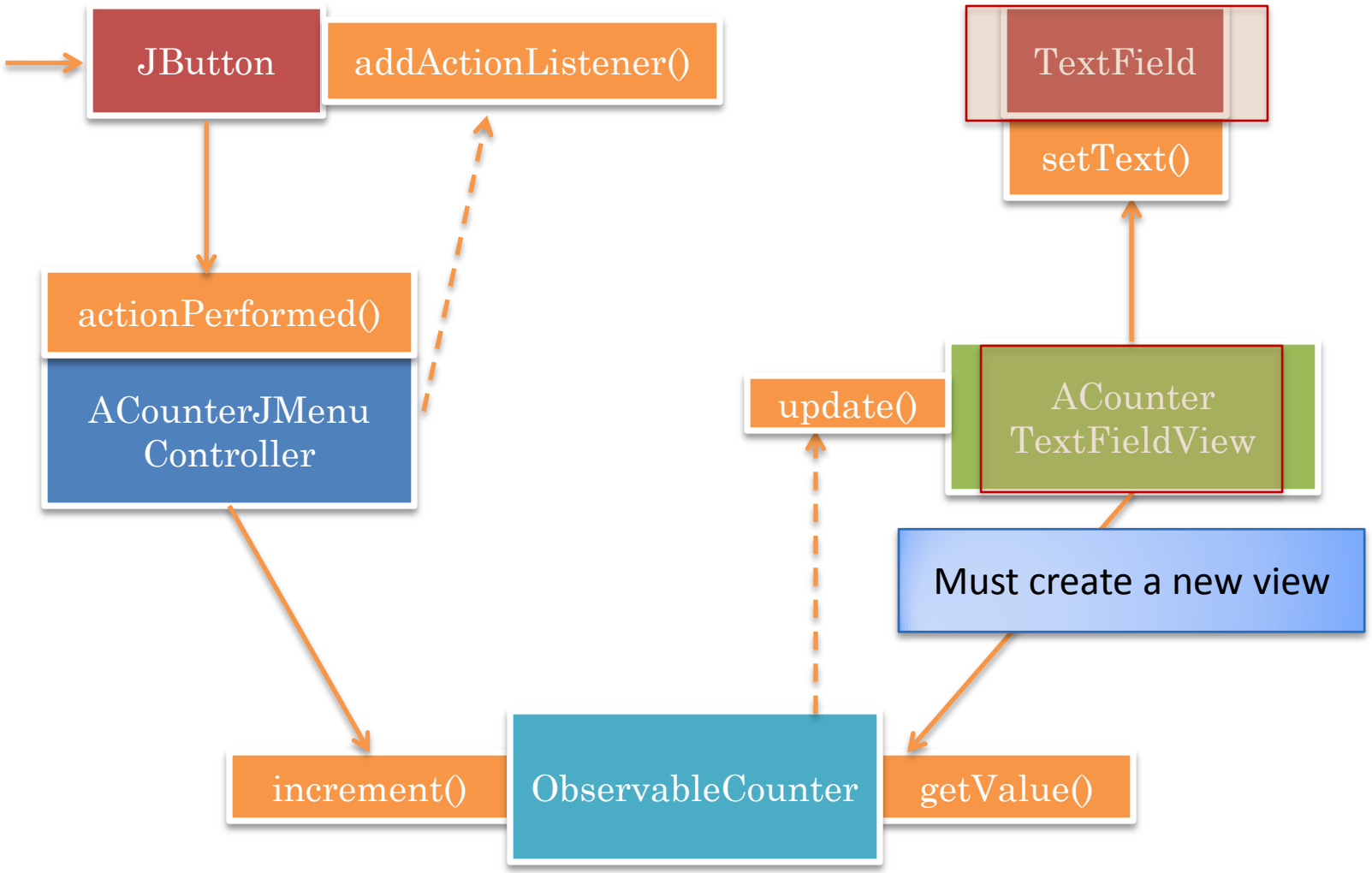
Exists vs. Ideal



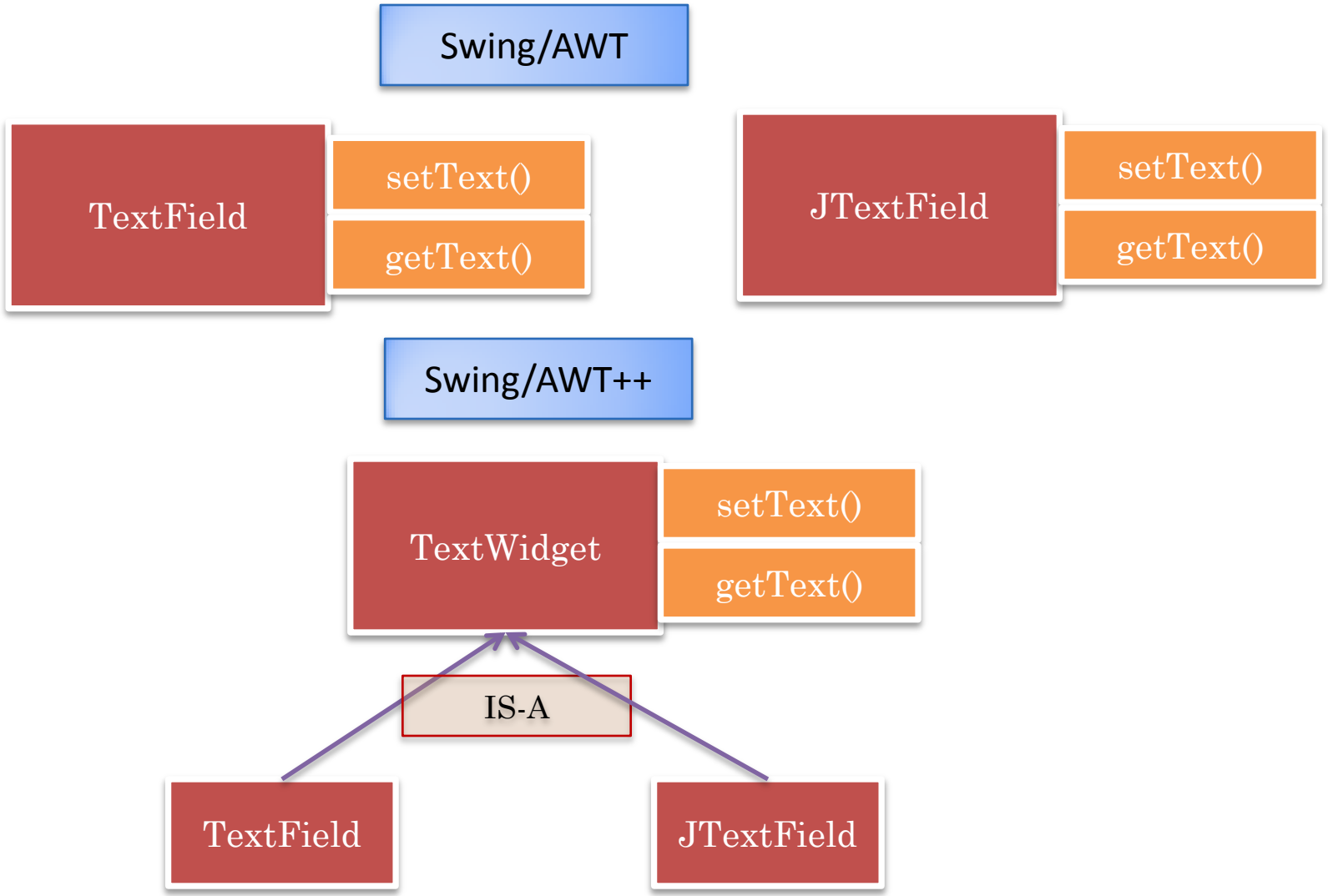
Using JTextField



Using TextField



Exists vs. Ideal



Menu Item Controller

```
public class ACounterJMenuItemController
                implements CounterController, ActionListener {
  JMenuItem menuItem;
  ObservableCounter counter;
  public ACounterJMenuItemController(JMenuItem theMenuItem) {
    menuItem = theMenuItem;
    menuItem.addActionListener(this);
  }
  public void setModel (ObservableCounter theCounter) {
    counter = theCounter;
  }
  public void processInput() {

  }
  public void actionPerformed(ActionEvent arg0) {
    counter.add(1);
  }
}
```

Menu Item Composer

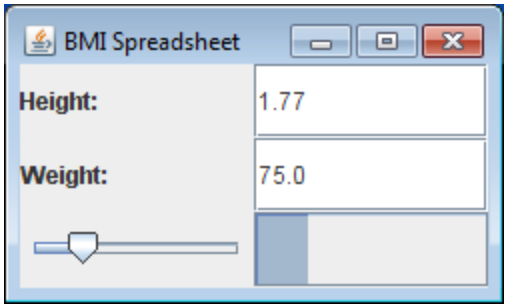
```
public class SingleFrameAWTMenuItemTextFieldComposer {
  public static void main (String args[]) {
  // compose AWT components
    JFrame frame = new JFrame("AWT Controller and View");
    JMenuItem menuItem = new JMenuItem("Increment");
    JMenu menu = new JMenu("Counter");
    JMenuBar menuBar = new JMenuBar();
    JTextField textField = new JTextField();
    frame.add(textField);
    menu.add(menuItem);
    menuBar.add(menu);
    frame.setJMenuBar(menuBar);
    frame.setSize(250, 125);
    frame.setVisible(true);

    //compose model view and controller
    ObservableCounter model = new AnObservableCounter();
    CounterObserver view = new ACounterJTextFieldView(textField);
    model.addObserver(view);
    CounterController controller = new ACounterJMenuItemController(menuItem);
    controller.setModel(model);
   }
 }
```

MVC, Toolkits and Beans

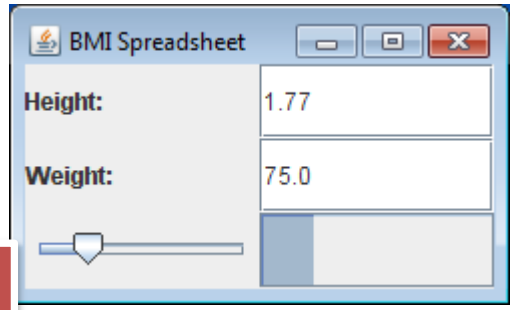
```
import java.beans.PropertyChangeEvent;
import java.beans.PropertyChangeListener;
import util.annotations.ObserverRegisterer;
import util.annotations.ObserverTypes;
import util.models.PropertyListenerRegisterer;
public class AnObservableBMISpreadsheet
         implements  BMISpreadsheet, PropertyListenerRegisterer {
  PropertyListenerSupport propertyListenerSupport =
         new APropertyListenerSupport();

  @ObserverRegisterer(ObserverTypes.PROPERTY_LISTENER)
  public void addPropertyChangeListener(
          PropertyChangeListener listener) {
    propertyListenerSupport.addElement(listener);
  }
  …
  public void setWeight(double newWeight) {
    double oldWeight = weight;
    double oldBMI = getBMI();
    weight = newWeight;
    propertyListenerSupport.notifyAllListeners(
     new PropertyChangeEvent(this, "weight", oldWeight, newWeight));
    propertyListenerSupport.notifyAllListeners(
     new PropertyChangeEvent(this, "bmi", oldBMI, getBMI()));
  }
}
```



BMI value shown in two different widgets

Widgets and Model Structures



JFrame

HAS-A

JPanel

JPanel

JPanel

JLabel

JTextField

JLabel

JTextField

JSlider

JProgressBar

Causes change in

Causes change in

Causes change in

double

double

double

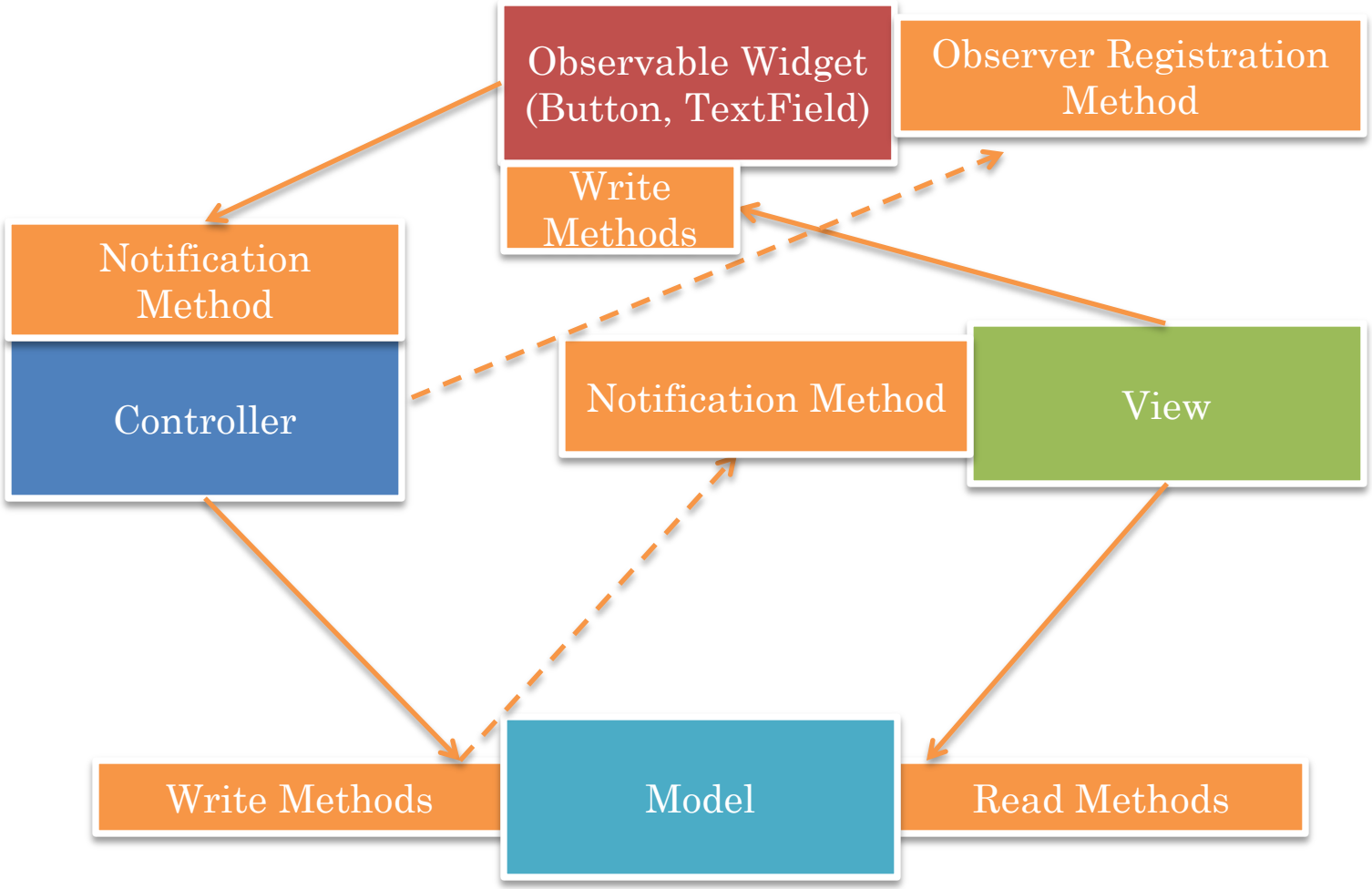
height

weight

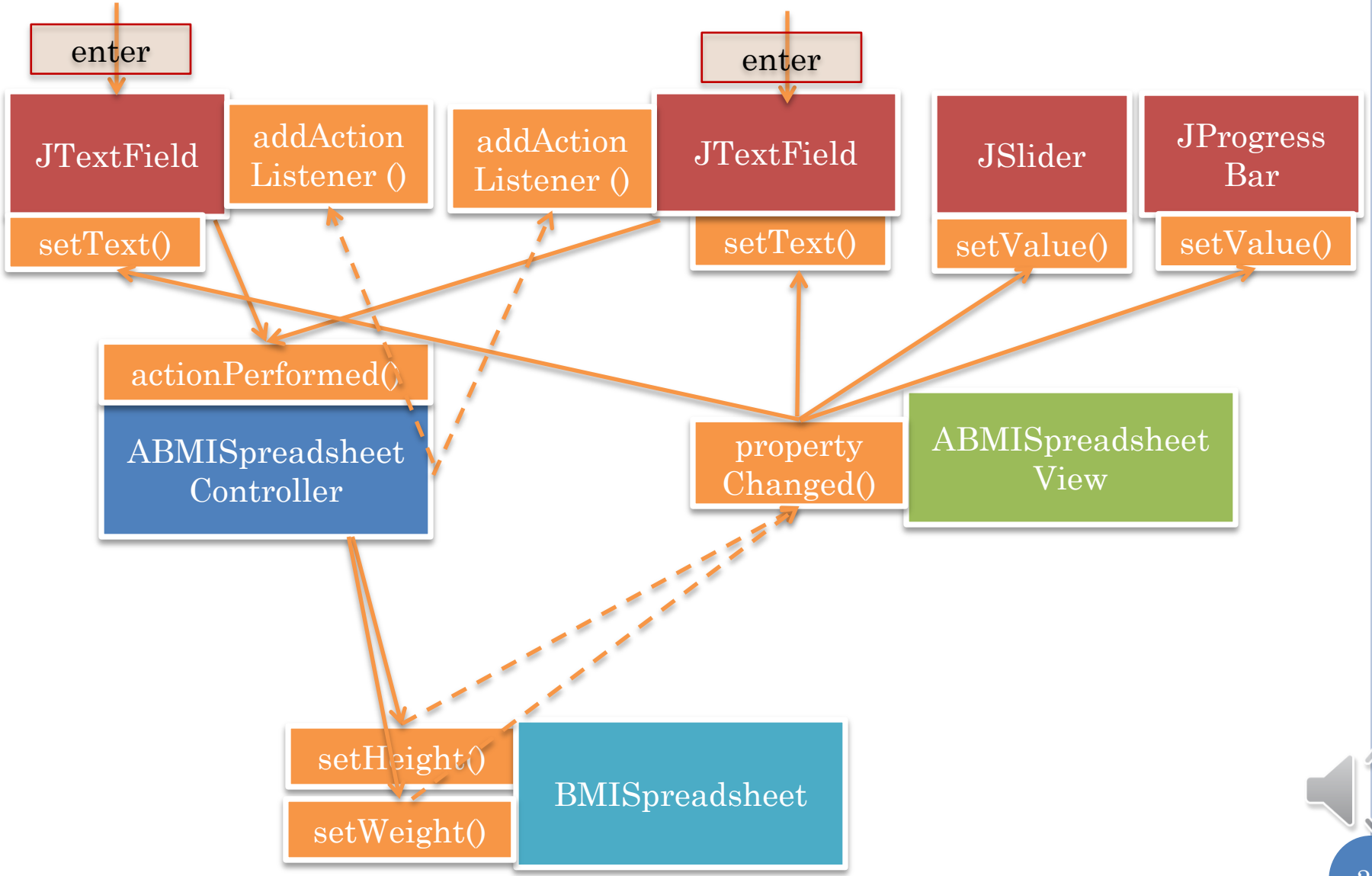
bmi

BMISpreadsheet

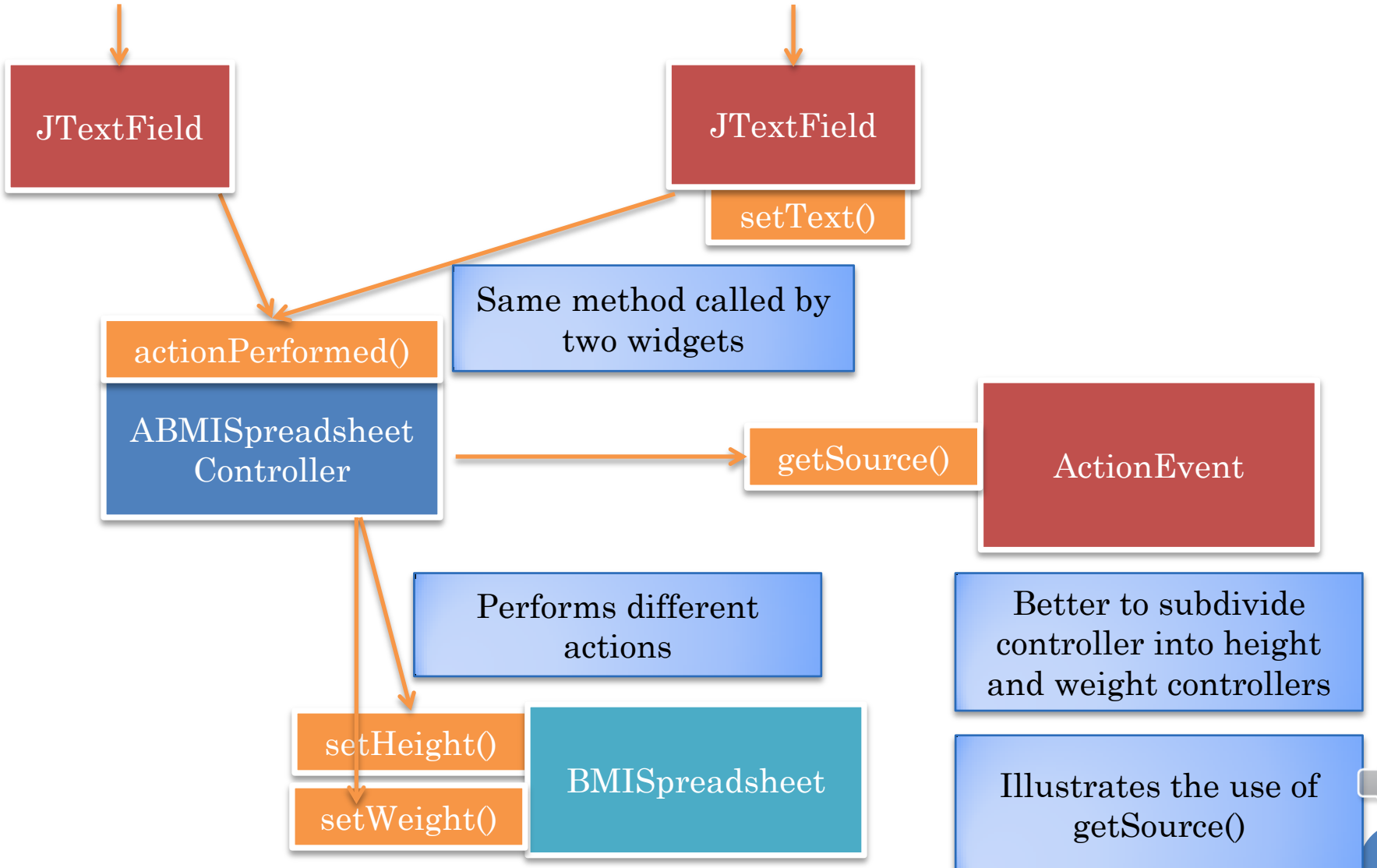
MVC and Toolkits: General Structure



MVC and Toolkits: BMI SPreadsheet

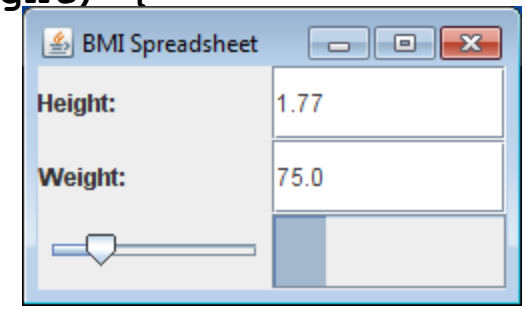


MVC and Toolkits: General Structure



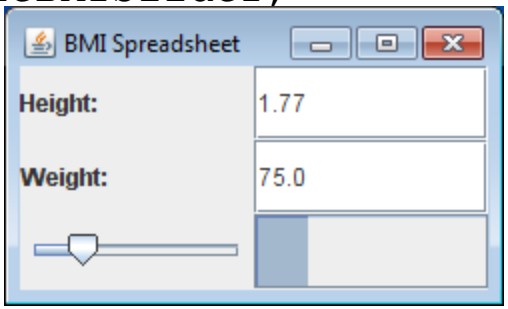
BMI Controller

```
public class ABMISpreadsheetController implements ActionListener {
  JTextField height, weight;
  BMISpreadsheet bmiSpreadsheet;
  public ABMISpreadsheetController (
        BMISpreadsheet theBMISpreadsheet,
        JTextField theHeight, JTextField theWeight) {
    height = theHeight;
    weight = theWeight;
    bmiSpreadsheet = theBMISpreadsheet;
    height.addActionListener(this);
    weight.addActionListener(this);
   }
  public void actionPerformed(ActionEvent event) {
    JTextField source = (JTextField) event.getSource();
    String text = source.getText();
    double val = Double.parseDouble(text);
    if (source == height) {
      bmiSpreadsheet.setHeight(val);
    } else {
      bmiSpreadsheet.setWeight(val);
    }
  }
}
```



BMI View

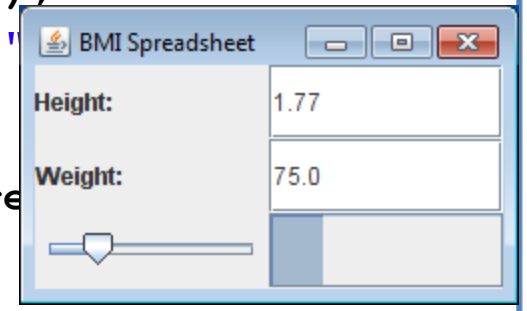
```
public class ABMISpreadsheetView implements PropertyChangeListener{
  JTextField heightField, weightField;
  JSlider bmiSlider;
  JProgressBar bmiProgressBar;
  public ABMISpreadsheetView (JTextField theHeightField,
          JTextField theWeightField, JSlider theBMISlider,
          JProgressBar theBMIProgressBar) {
    heightField = theHeightField;
    weightField = theWeightField;
    bmiSlider = theBMISlider;
    bmiProgressBar = theBMIProgressBar;
  }
  public void propertyChange(PropertyChangeEvent event) {
    String propertyName = event.getPropertyName();
    Double newValue = (Double) event.getNewValue();
    if (propertyName.equalsIgnoreCase("height")) {
      heightField.setText(newValue.toString());
    } else if (propertyName.equalsIgnoreCase("weight")) {
       weightField.setText(event.getNewValue().toString());
    } else if (propertyName.equalsIgnoreCase("bmi")) {
       double newBMI = newValue;
       bmiSlider.setValue((int) newBMI);
       bmiProgressBar.setValue((int) newBMI);
    }}}
```



BMI Composer

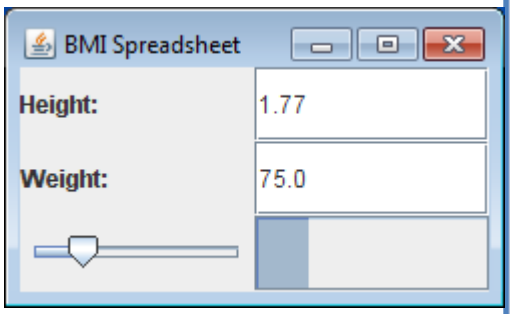
```
public class BMIFrameComposer {
  static JFrame frame = new JFrame("BMI Spreadsheet");
  static JTextField heightField = new JTextField();
  static JLabel heightLabel = new JLabel("Height:");
  static JPanel heightPanel = new JPanel();
  static JTextField weightField = new JTextField();
  static JLabel weightLabel = new JLabel("Weight:"
  static JPanel weightPanel = new JPanel();
  static JSlider bmiSlider = new JSlider();
  static JProgressBar bmiProgressBar = new JProgre
  static JPanel bmiPanel = new JPanel();

  public static void main (String args[]) {
    composeLabelledField(heightPanel, heightLabel, heightField);
    composeLabelledField(weightPanel, weightLabel, weightField);
    composeBMI();
    composeFrame();
    composeMVC();
  }
```



BMI Composer (contd)

```
public static void composeBMI() {
  bmiPanel.setLayout(new GridLayout(1, 2));
  bmiPanel.add(bmiSlider);
  bmiPanel.add(bmiProgressBar);
}
public static void composeFrame() {
  frame.setLayout(new GridLayout(3, 1));
  frame.add(heightPanel);
  frame.add(weightPanel);
  frame.add(bmiPanel);
  frame.setSize(250, 150);
  frame.setVisible(true);
}
public static void composeMVC() {
  BMISpreadsheet bmiSpreadsheet =
       new AnObservableBMISpreadsheet();
  new ABMISpreadsheetController(bmiSpreadsheet,
       heightField, weightField);
  PropertyChangeListener bmiSpreadsheetView =
  new ABMISpreadsheetView(heightField, weightField,
       bmiSlider, bmiProgressBar);
    bmiSpreadsheet.
    addPropertyChangeListener(bmiSpreadsheetView);
}}
```



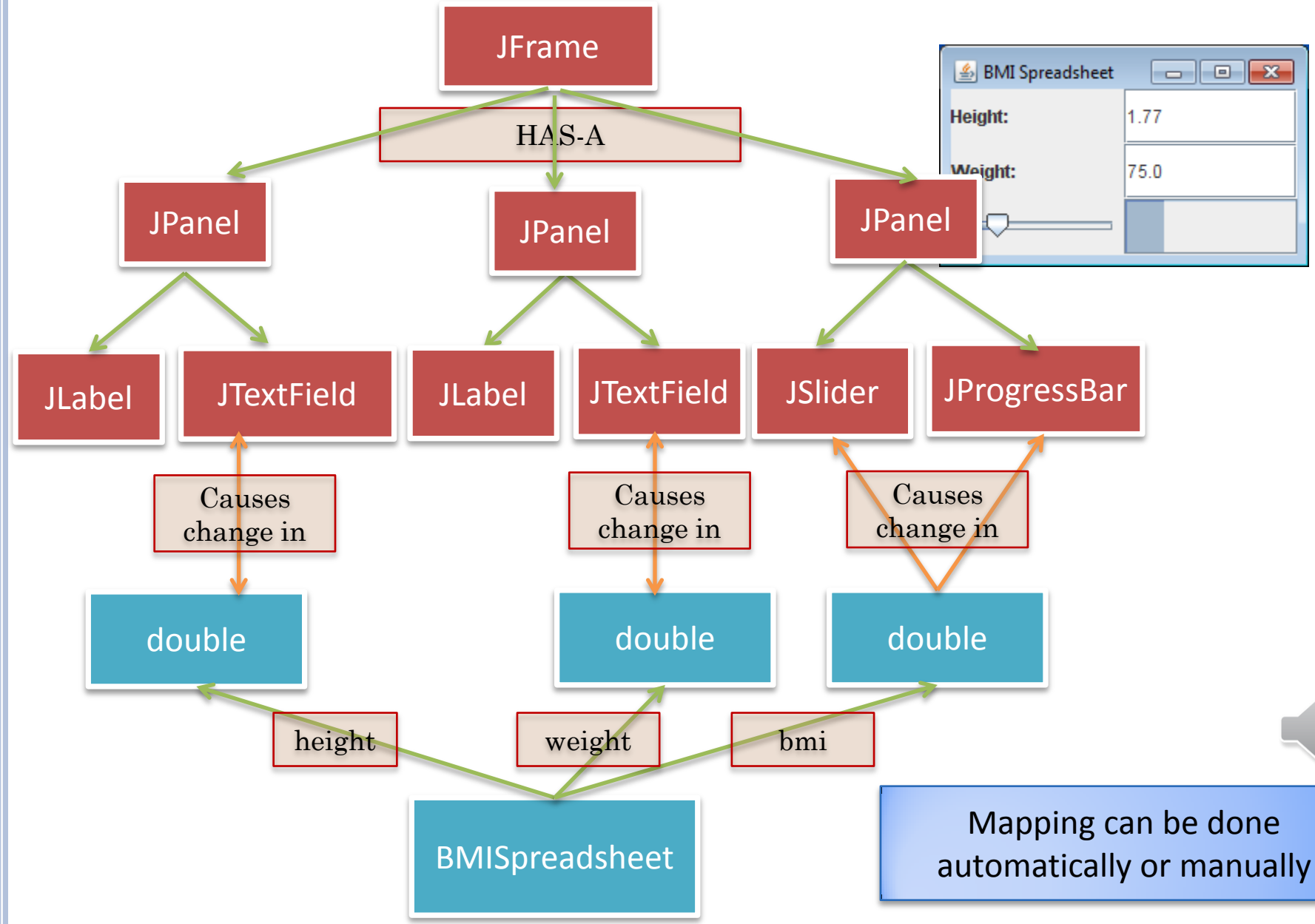
MVC, Toolkits + Beans

Like MVC with toolkits we saw earlier

Additional twists

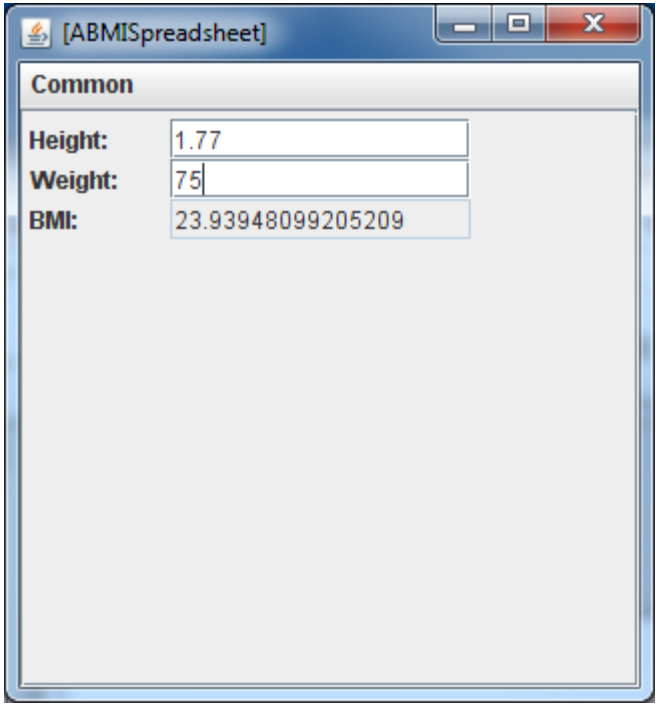
- Model now is divided into multiple properties
- User interface composed of multiple input/output widgets
- View must now implement the Java java.beans.PropertyChangeListener interface
  - Do not implement your interface that follows the same naming conventions
- View and controllers can get multiple widgets from main, usually one for each properties they change/display, to which they store references. Sometimes a view may display the same property using multiple widgets (e.g. slider and progress bar for bmi) or a controller may listen to multiple widgets for setting a property (e.g. menu item and button)

Widgets and Model Structures

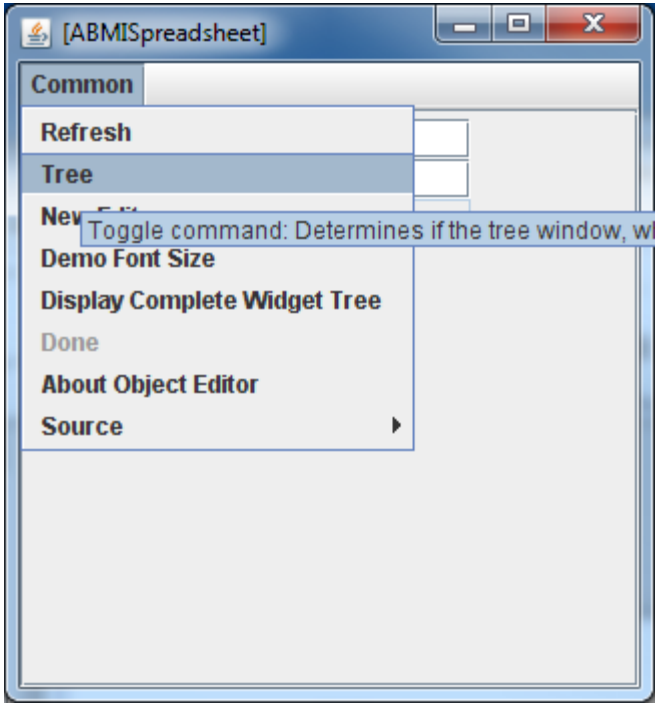


Mapping can be done automatically or manually

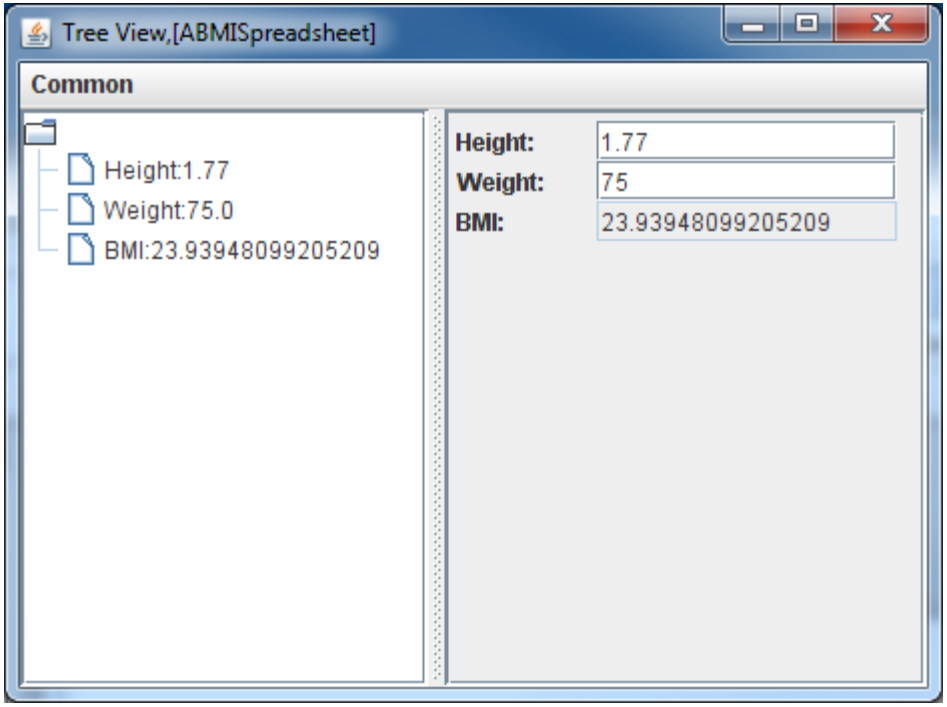
Visualizing Model Structure



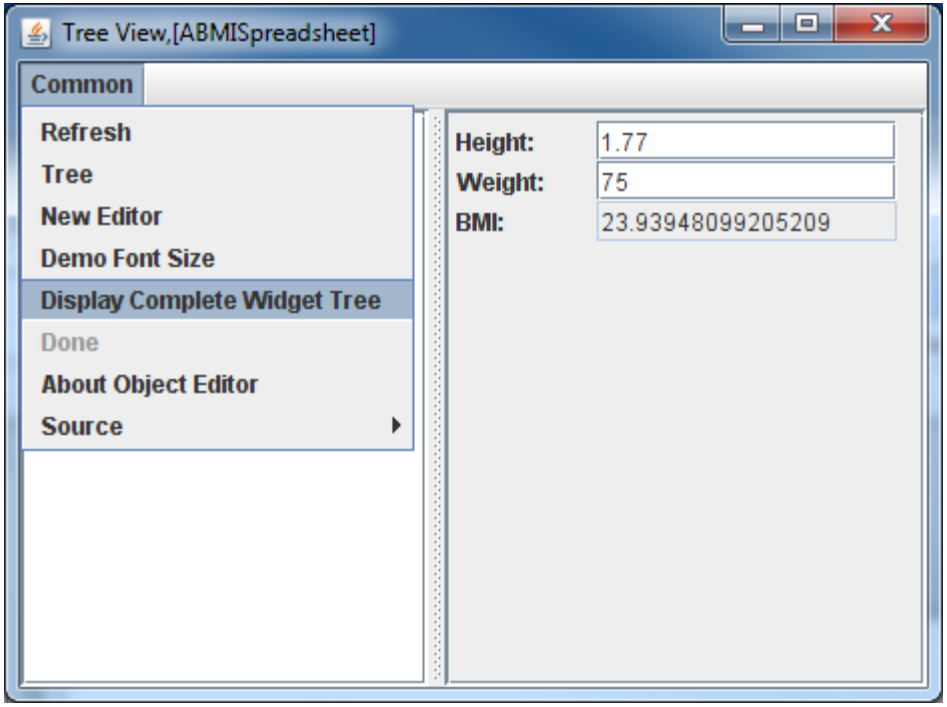
Visualizing Model Structure



Model Structure



Visualizing Widget Structure



Widget Structures