Model View Controller Architecture with Java SWING

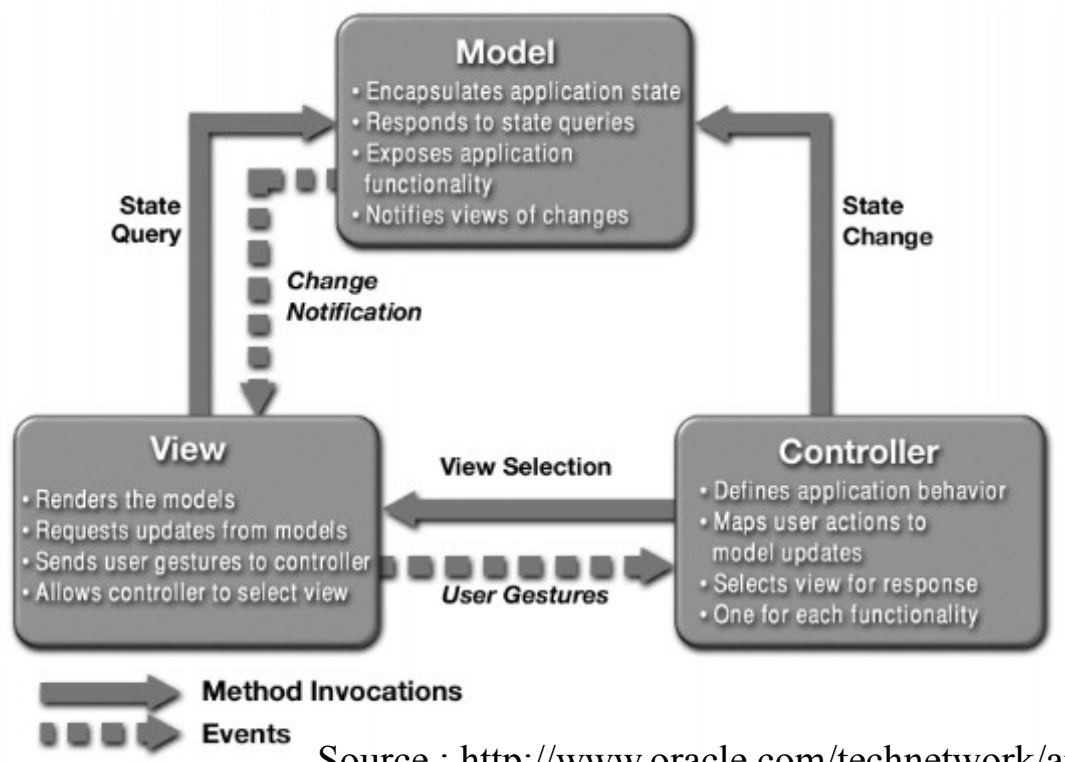
Jérôme David
L2 MIASHS
2021-2022
jerome.david@univ-grenoble-alpes.fr
Université Grenoble Alpes
France

Outline

- MVC overview
  - What are the model, the view and the controller
  - How they interact each other ?
- Notification mechanism
  - Observer/Observable and extensions
- Java Swing GUI toolkit
  - Overview, common widgets, layout managers
  - ActionListener and Action (Command pattern)
  - Swing and MVC

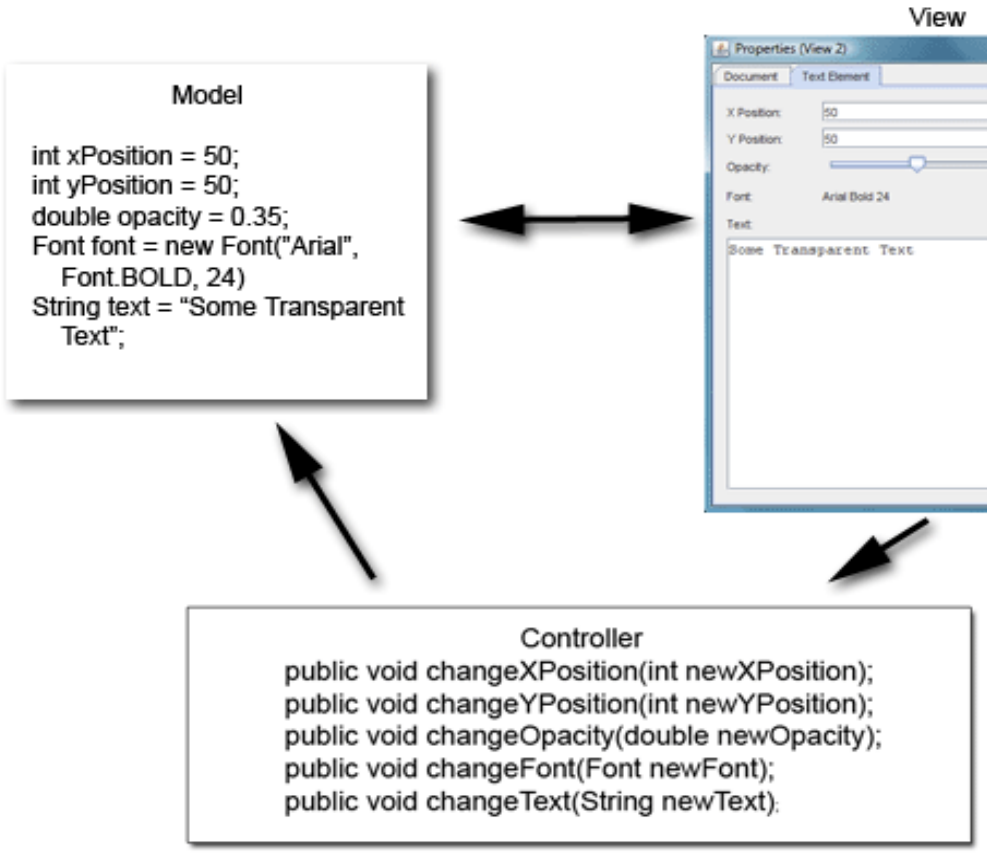
Model-View-Controller Design Pattern

- First introduced by a Smalltalk developer at the Xerox Palo Alto Research Center in 1979
- Goal : separate the program logic from the manner it is displayed to the user



Source : http://www.oracle.com/technetwork/articles/javase/mvc-136693.html

MVC Example



1- A change opacity value on spinner notifies the controller(the method changeFont of the controller will be called)

3- The controller change the value on the model (and deals with eventual problems)

4- the model notifies its change to all his views, and these views update themselves

Source : http://www.oracle.com/technetwork/articles/javase/mvc-136693.html

The Model

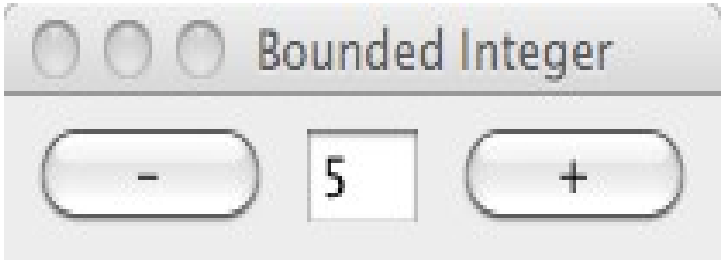
- It encapsulates the application data and logic
  - It has no reference to the view
  - It controls access to and updates of its data

```
public class BoundedInteger {

      private int lowerBound;
      private int upperBound;
      private int currentValue;

      public BoundedInteger(int currentVal, int min, int max) {...}

      public int increment() {...}
      public int decrement() {...}
      public void setValue(int newVal) {...}
      public void setUpperBound(int newMax) {...}
      public void setLowerBound(int newMin) {…}

      public int getCurrentValue() {...}
      public int getLowerBound() {...}
      public int getUpperBound() {...}

}
```

  - It also notifies its changes... but we will see this later

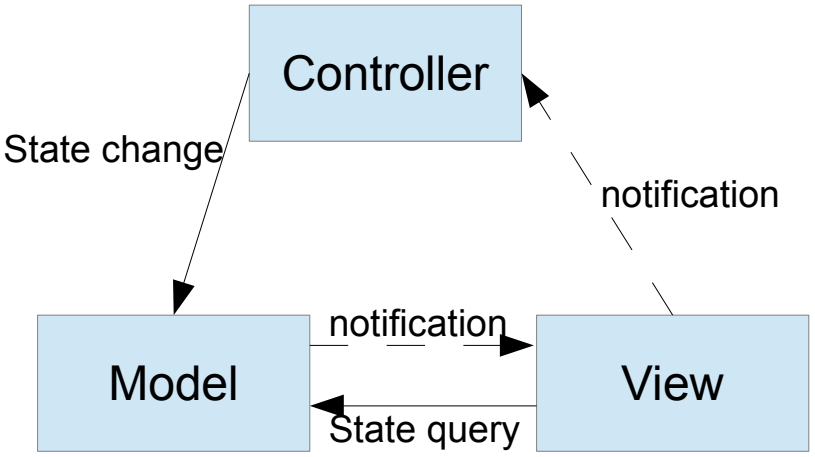
The view

- Goal: to present information to the user
  - It is agnostic of the application logic
  - It does not store application data
  - It contains a mechanism to be notified of the model changes
  - It notifies its changes (user interaction) to the controller

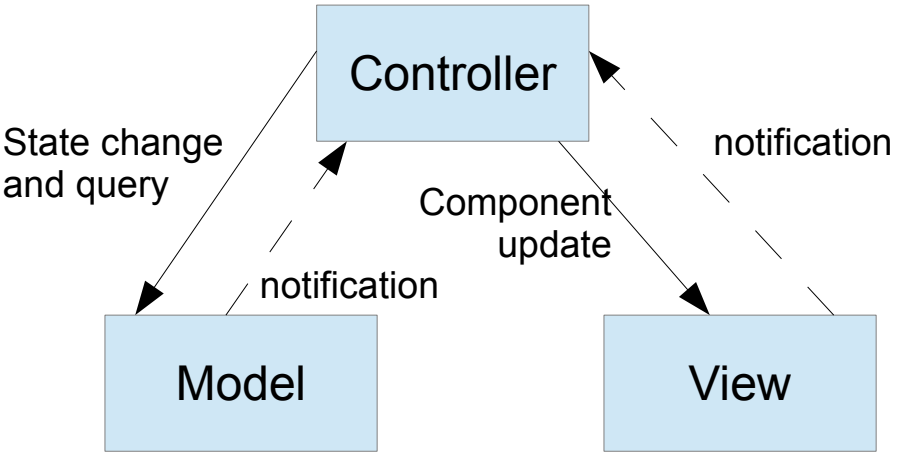


The Controller

- It is the link between the view and the model
  - It receives notifications from the view (user interaction) and modifies the model
- There are two ways to think a controller



Classical MVC



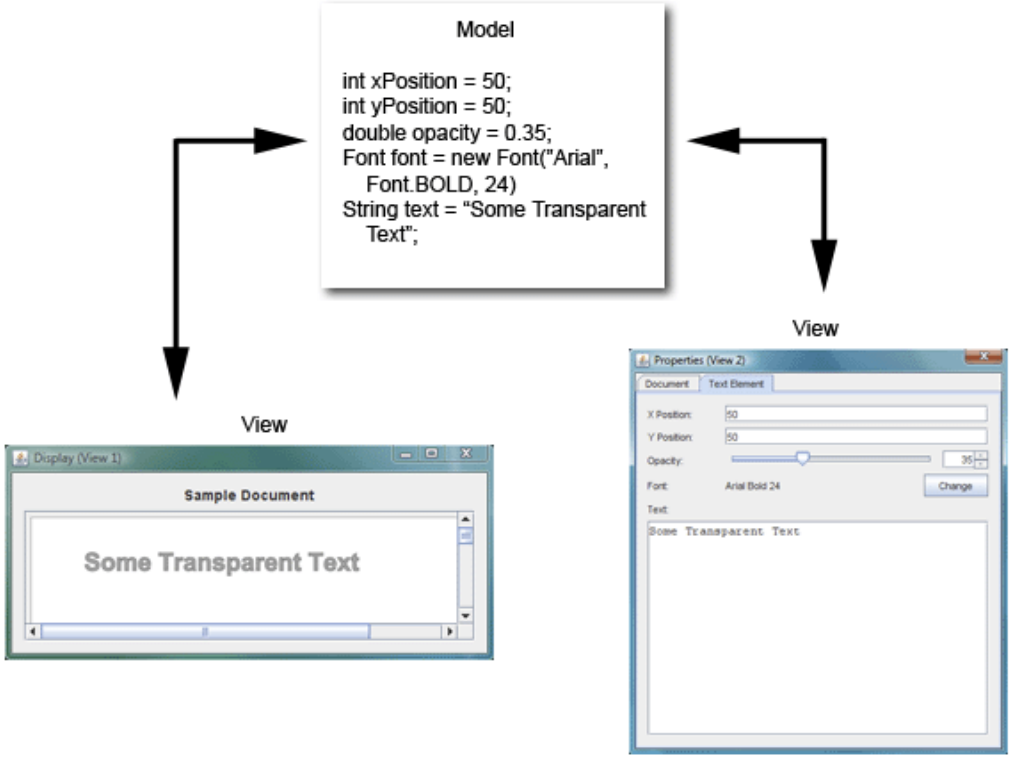
“Apple Cocoa style”
(or PAC)

MVC advantages

- It allows the development of software by several teams in parallel
  - Some developers will work on the model, other on the view. They only have to decide before the model interface
- It facilitates the maintenance of software
  - We can change the view without re-developing the model
- It allows multiples views on the same model...

Multiple views on the same model

- MVC facilitates the development of multiples views on the same model



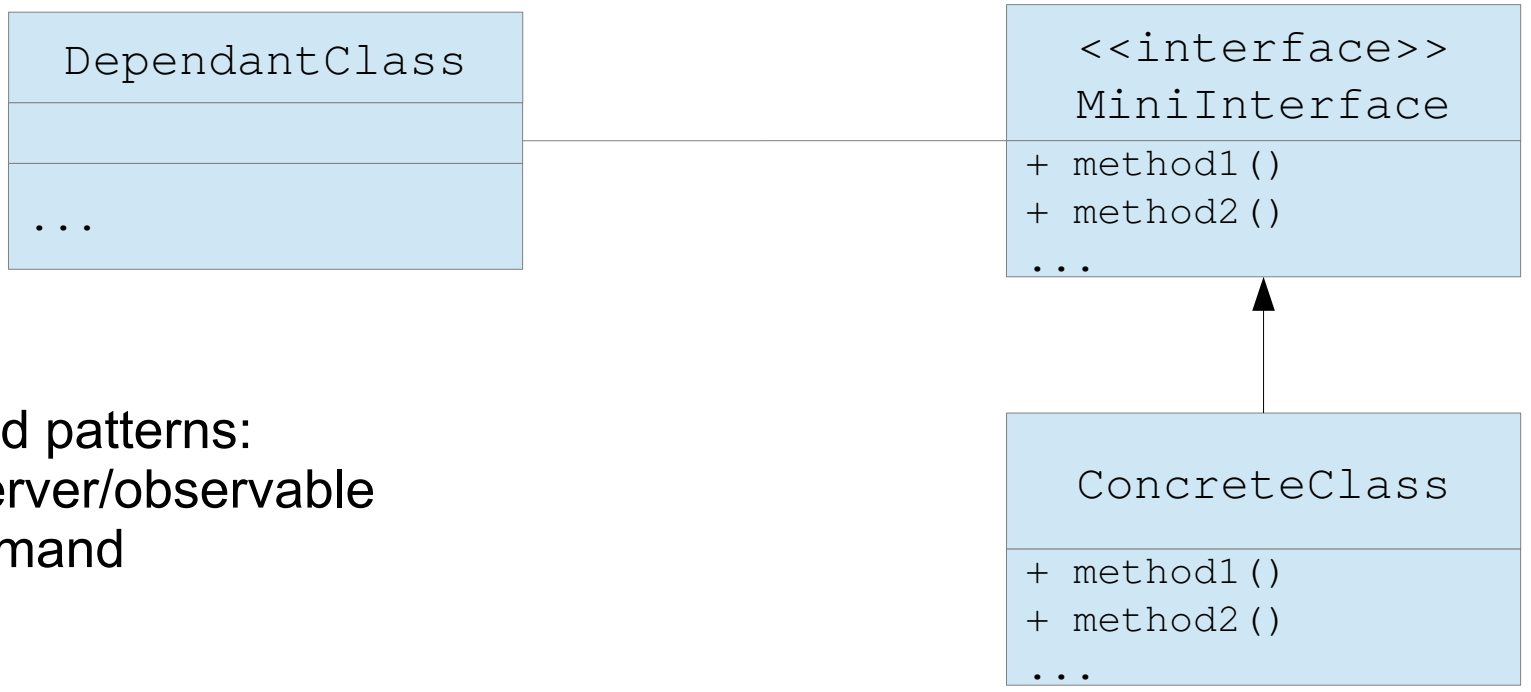
Source : http://www.oracle.com/technetwork/articles/javase/mvc-136693.html

The notifications

- MVC aims at separating the Model and the View
  - But there are still some dependancy between them
    - The model have to notify its changes (either to the controller or the view)
    - The view has to notify the user interaction (clic, etc.) to the controller
- Solution: use loose coupling for notifications
  - components has little or no knowledge of the definitions of other separate components
  - How to do that in Object Oriented programming ?

Loosely coupling architectures

- Limit the dependency between two classes
  - Solution: dependant class has to know only a limited interface about the component it depends on



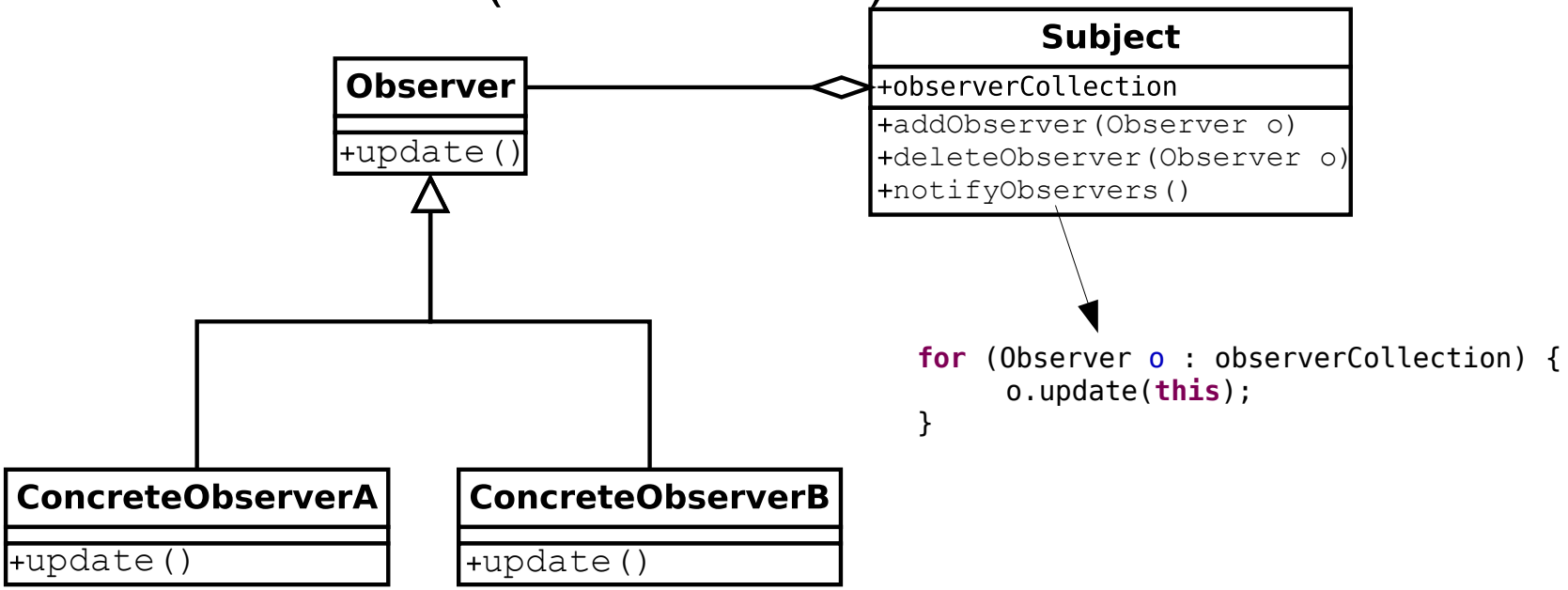
Related patterns:
- Observer/observable
- Command
...

Observer / Observable pattern

- The model has to notify its changes...

  → Use the pattern Observer/Observable

  - Allows to define a loose coupling dependency between the model (the observable) and the view and/or controllers (the observers)



Source: http://en.wikipedia.org/wiki/Fichier:Observer.svg

Observer / Observable
A basic observer

```
public class BasicBIObserver implements Observer {

      private BoundedInteger observedModel;
      private String name;

      public BasicBIObserver(BoundedInteger model, String name) {
            this.observedModel=model;
            this.name=name;
            model.addObserver(this);
      }

      public void update(Observable obs, Object evt) {
            if (obs==observedModel) {
                  System.out.println("I am observer "+name+". I have been notified of these changes: ");
                  System.out.println("\tlower bound"+observedModel.getLowerBound());
                  System.out.println("\tcurrent value"+observedModel.getCurrentValue());
                  System.out.println("\tupper bound"+observedModel.getUpperBound());
            }
      }

}
```

1 - implement interface java.util.Observer

2 - register the observer against the model

3 - implement the inherited update method

```
public class MainProgram {

      public static void main(String[] args) {
            BoundedInteger model = new BoundedInteger(1,0,10);
            BasicBIObserver anObserver = new BasicBIObserver(model,"Observer 1");

            model.increment();
            model.decrement();
            model.setLowerBound(20);
      }
}
```

TRY !!!

Observer / Observable Limitations

- The model has to extend the Observable class
  - And if the model has to extend another class ?
    - Solution : To encapsulate an instance of observable

```
public class BoundedInteger2 extends AnotherClass {
      private final Observable obs = new Observable();


      public void addObserver(Observer o) {
            obs.addObserver(o);
      }

      public void deleteObserver(Observer o) {
            obs.deleteObserver(o);
      }

      //other model methods ...
}
```

Add an encapsulated Observable instance

Write encapsulated methods which are need

But a new problem: the model is not the source anymore of the notifications

Observer / Observable Limitations

- How to manage several kinds of observers ?
  - Use the optional argument in method notify

```
public class MyEvents {
	public static final MyEvents LOWER_BOUND_CHANGE=new MyEvents();
	public static final MyEvents UPPER_BOUND_CHANGE=new MyEvents();
	public static final MyEvents CURRENT_VALUE_CHANGE=new MyEvents();

	private MyEvents(){}
}
```

1 - define a class Event

```
public class BoundedInteger extends Observable {
	...
	public int increment() {
		...
		this.setChanged();
		this.notifyObservers(MyEvents.CURRENT_VALUE_CHANGE);
		return currentValue;
	}
```

2 – send the relevant event when notify

```
public class ObserverCurrentValueChange implements Observer {

	private BoundedInteger observedModel;

	public ObserverCurrentValueChange(BoundedInteger model) {
		this.observedModel=model;
		model.addObserver(this);
	}

	public void update(Observable o, Object arg) {
		if (   o==observedModel &&
			arg ==  MyEvents.CURRENT_VALUE_CHANGE) {
		...
		}
	}
}
```

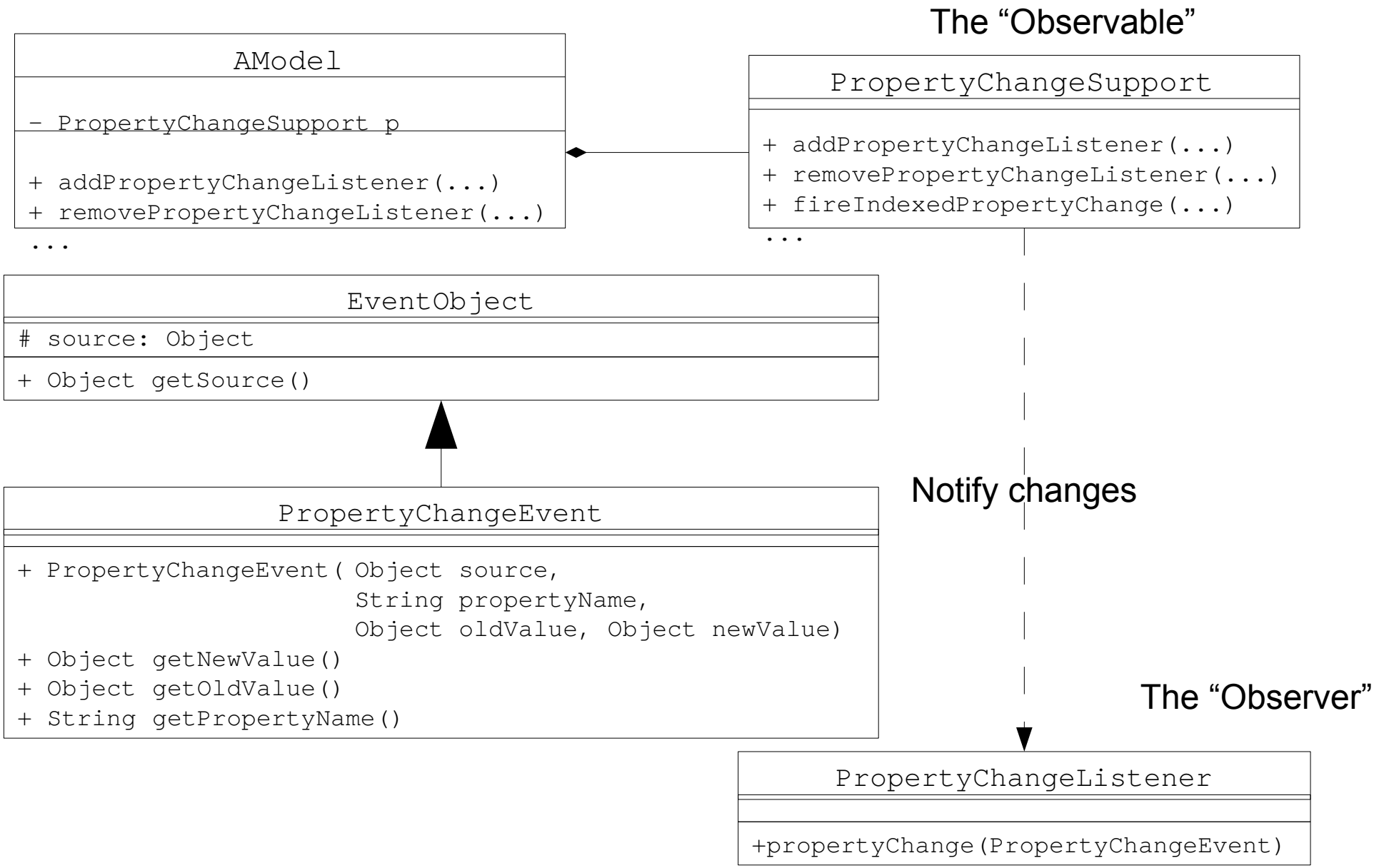
3 – test the event before doing action

BUT IT EXISTS A READY SOLUTION !

An extension of Observer/Observable pattern

- A model usually contains several different properties for which we want to notify changes
  - We saw that it is possible to do that with Observer/Observable classes from Java
- Java SDK gives a nice solution for this:
  - The Bounded Property, i.e. property that notifies listeners of its changes
  - Requirement: the class has to be a java bean
    - Each bound property has a get and set method

Bound property architecture



PropertyChangeSupport

- Utility class used for delegating listener registration and property changes notification
  - The model has to have an instance of this class
- Some useful methods:
  - Register/de-register change listener for an attribute having the given name

```
addPropertyChangeListener(String propertyName, PropertyChangeListener listener)

removePropertyChangeListener(PropertyChangeListener listener)
```

  - Notify the registered listeners that a change on some property occurred

```
firePropertyChange(String propertyName, Object oldValue, Object newValue)

firePropertyChange(PropertyChangeEvent  evt)
```

PropertyChangeSupport

```
public class PCModel {
	private final PropertyChangeSupport pcs = new PropertyChangeSupport(this);

	private int prop1;
	private String prop2;

	public PCModel() {setProp1(1);setProp2("Hello");}

	public void setProp1(int newValue) {
		int oldValue=this.prop1;
		this.prop1 = newValue;
		pcs.firePropertyChange("prop1", oldValue, newValue);
	}

	public int getProp1() {return prop1;}

	// do the same for prop2


	public void addPropertyChangeListener(String propertyName, PropertyChangeListener listener) {
		pcs.addPropertyChangeListener(propertyName,listener);
	}

	public void removePropertyChangeListener(String propertyName, PropertyChangeListener listener) {
		pcs.removePropertyChangeListener(propertyName,listener);
	}
}
```

Add a PropertyChangeSupport instance

Fire event when there is a change

Expose the registering methods needed and delegate them to the pcs

PropertyChangeListener

- Interface that must implement a property change listener (the observer)
- The method called is
  - **`void propertyChange(PropertyChangeEvent evt)`**
    - The parameter event represents the change (source object, old value, new value)

```
public class PCListener implements PropertyChangeListener {

	public void propertyChange(PropertyChangeEvent event) {
		System.out.println(event.getSource());
		System.out.println(event.getPropertyName());
		System.out.println(event.getOldValue());
		System.out.println(event.getNewValue());
	}
}
```

```
public class PCMainProg {
	public static void main(String[] args) {
		PCModel model = new PCModel();
		PCListener listener = new PCListener();
		model.addPropertyChangeListener(PCModel.PROP_1, listener);
		model.setProp1(3);
	}
}
```

PropertyChangeEvent

- Represents an update of a property
- Event sent to registered listeners
- Methods
  - `getNewValue()`
    - The new property value
  - `getOldValue()`
    - The old property value
  - `getPropertyName()`
    - The name of the updated property (model attribute)

Exercise

1 – Change the BoundedInteger model in order it notifies changes with the bound property mechanism. It has to manage the different types of changes (currentValue, lowerBound, upperBound)

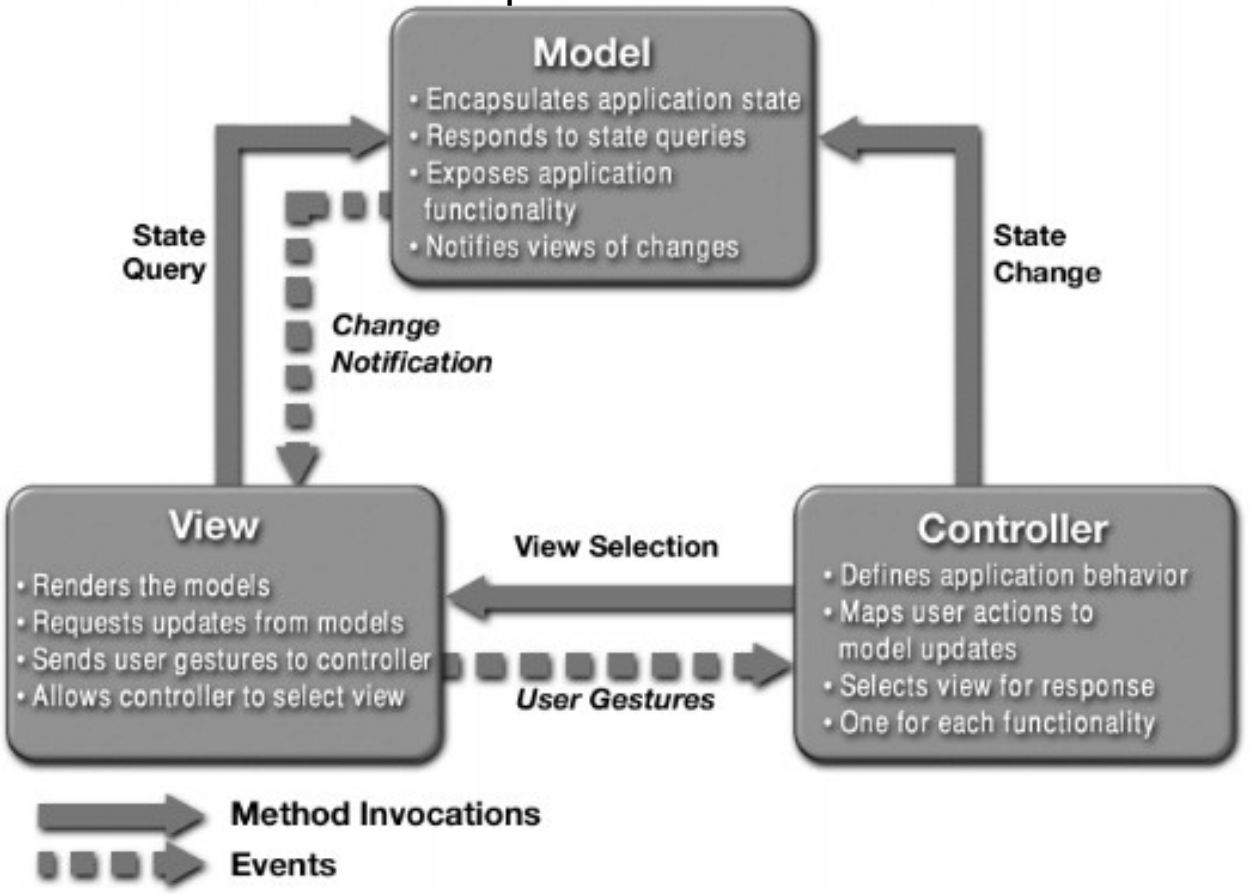
2 – Create two property change listeners, the first one will be notified only for current the currentValue changes, and the second one will be notified for lowerBound changes

3 – Create a program to test these listeners.

MVC - Interaction

- Observer / Observable allows notifications between model and view

The model extends or encapsulate Observable

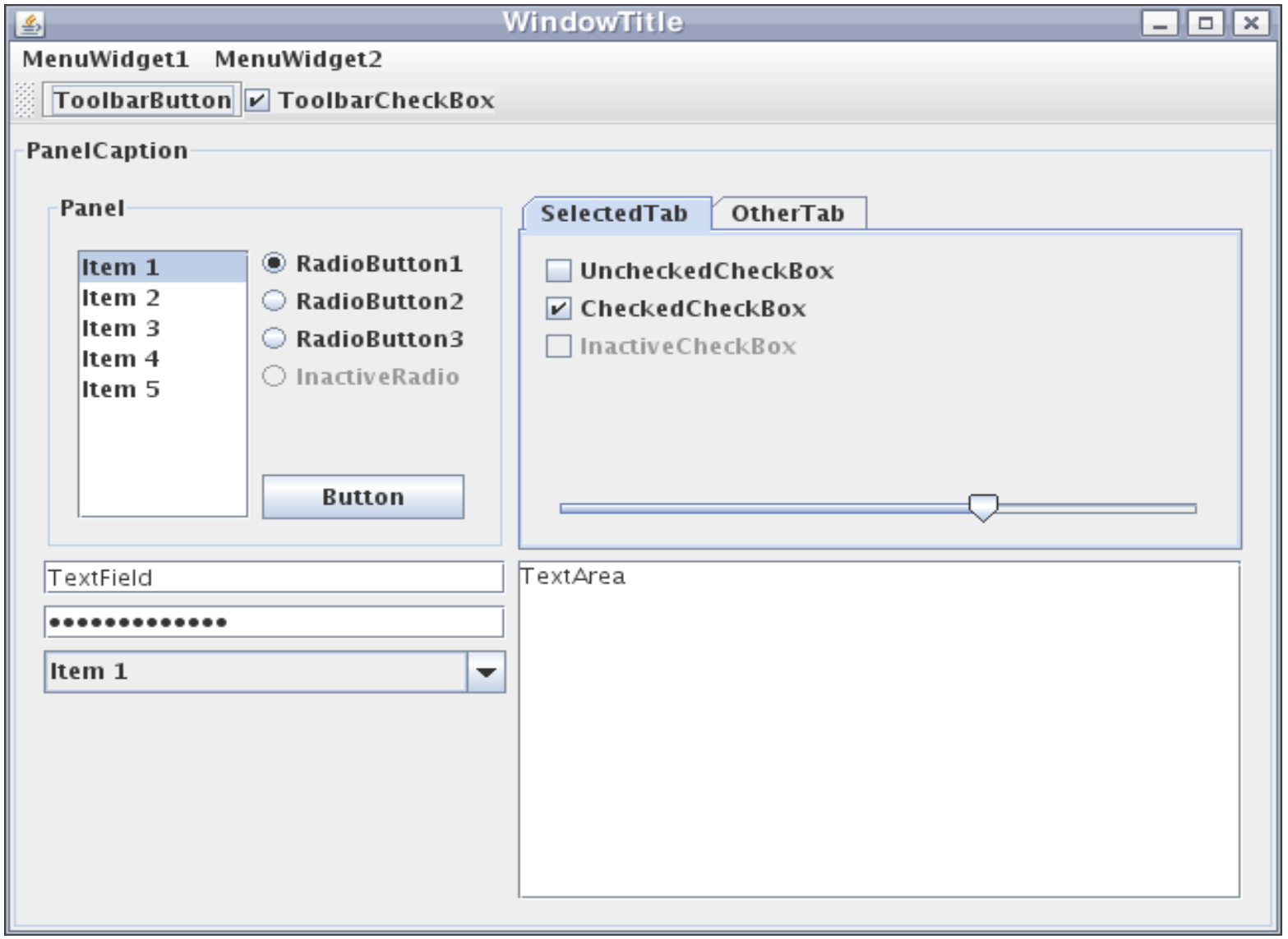


The view implements Observer

Swing basics

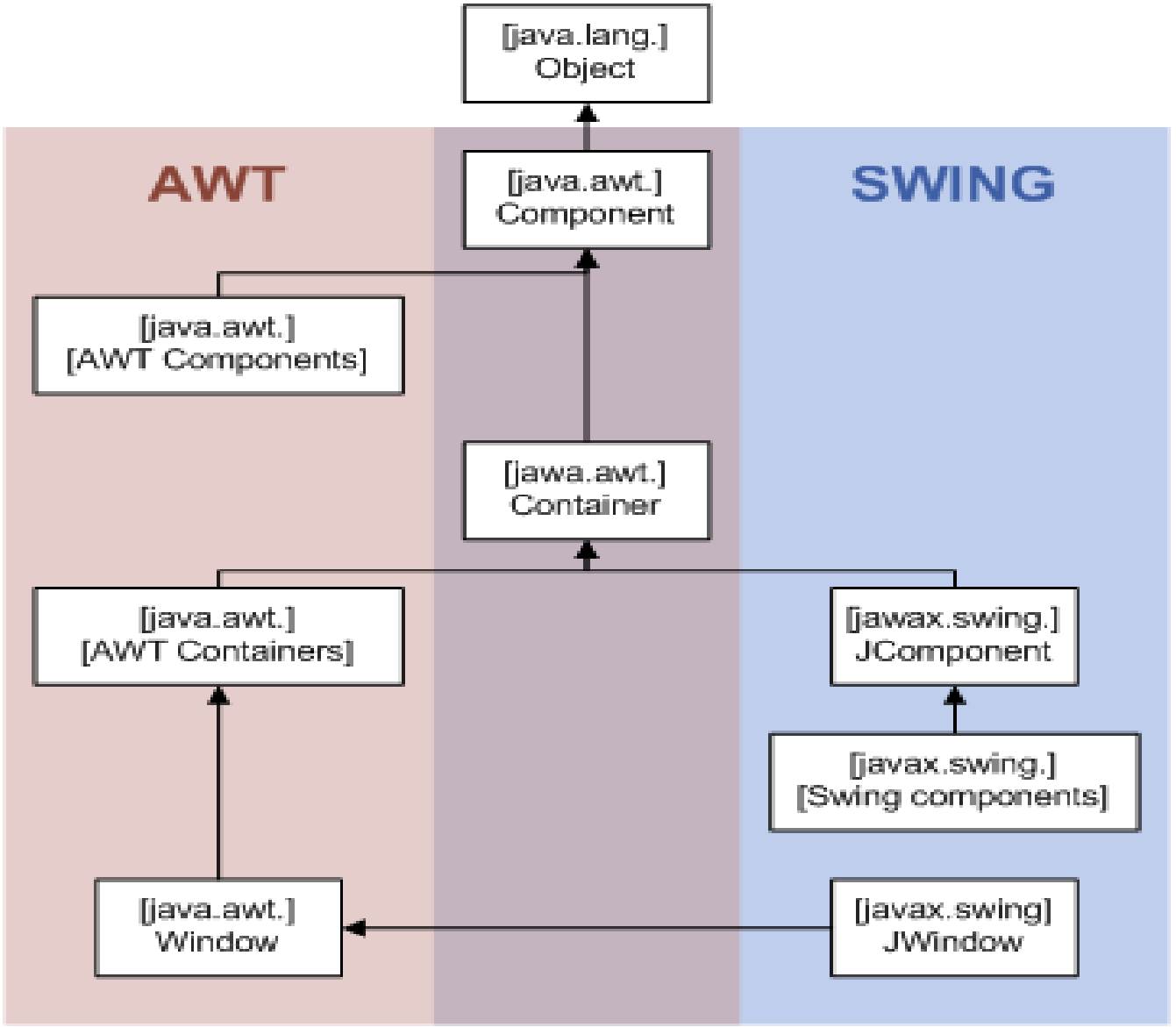
- The primary Java GUI widget toolkit
  - Most sophisticated than previous AWT
  - Appearance independent from the OS
    - Native look and feel & pluggable look and feel
  - Purely written in JAVA (AWT widgets are native)
  - Extensible
  - Make uses of MVC (or more precisely M-VC)
    - Controller and view are somehow mixed

Swing UI example



Source: http://en.wikipedia.org/wiki/File:Gui-widgets.png

Swing API architecture



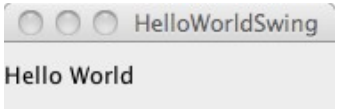
Source: http://en.wikipedia.org/wiki/File:AWTSwingClassHierarchy.png

Components and Containers

- Any UI widget (button, windows, menu, etc.) is a **component**
- A **container** is special component which can contains components or containers
- A container is usually associated with a **Layout Manager**, which define how to lay out component inside the container (in a grid, in a line, etc.)

A first Swing program

```
import javax.swing.*;

public class HelloWorldSwing {
    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("HelloWorldSwing");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Add the ubiquitous "Hello World" label.
        JLabel label = new JLabel("Hello World");
        frame.getContentPane().add(label);

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```



Source: http://docs.oracle.com/javase/tutorial/uiswing/examples/start/HelloWorldSwingProject/src/start/HelloWorldSwing.java

JLabel

- Displays unmodifiable text and/or image
- No user interaction


```
JLabel lb = new JLabel("Name :");
lb.setText("Name :");
lb.setIcon(new
ImageIcon("picture.gif"));
```

JTextField and others

- Basic text control which allow user to edit small amount of text


```
JTextField tf = new JTextField("some text");
tf.setText("other text");
tf.setColumns(15);
```

- Some JTextField extensions
  - JFormattedTextField: allows to specify the legal set of characters that the user can enter.
  - JPasswordField: does not not show the characters that the user types.

Subclasses of AbstractButton

- Buttons in Swing extends AbstractButton
  - JButton: 


```
JButton bt = new JButton("Info", new ImageIcon("picture.gif"));
bt.setMnemonic('n');
bt.setBorderPainted(false);
```

  - JCheckBox: 


```
ButtonGroup group = new ButtonGroup();
JRadioButton rb1 = new JRadioButton("Rabbit");
JRadioButton rb2 = new JRadioButton("Pig");
rb2.setSelected(true);
group.add(rb1);
group.add(rb2);
```

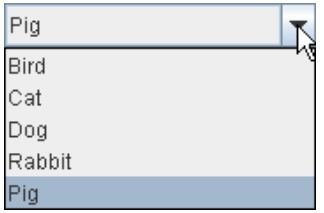
  - JRadioButton: 

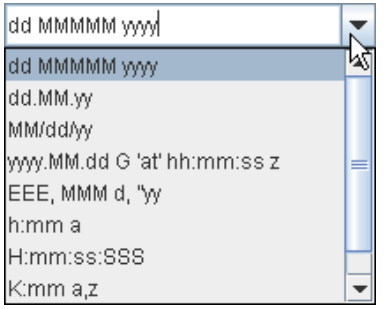
  -
  - JToggleButton: two states button

```
JToggleButton bt = new JToggleButton(new ImageIcon("image.gif"));
bt.setSelectedIcon(new ImageIcon("Image2.gif"));
```

JComboBox

- Widget which lets the user choose one of several choices
- Two kinds:
  - Non editable (like button)
  - Editable (like text field)



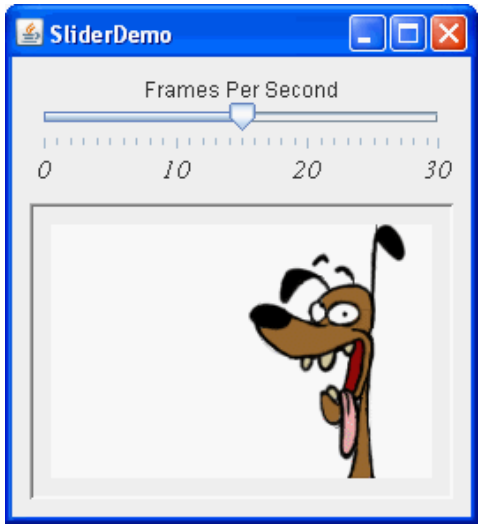



```
String[] petStrings = { "Bird", "Cat", "Dog", "Rabbit", "Pig" };

//Create the combo box, select item at index 4.
//Indices start at 0, so 4 specifies the pig.
JComboBox petList = new JComboBox(petStrings);
petList.setSelectedIndex(4);
petList.addActionListener(this);
```

```
  String[] patternExamples = {
         "dd MMMMM yyyy",
         "dd.MM.yy",
         "MM/dd/yy"
};
///. . .
JComboBox patternList = new JComboBox(patternExamples);
patternList.setEditable(true);
patternList.addActionListener(this);
```

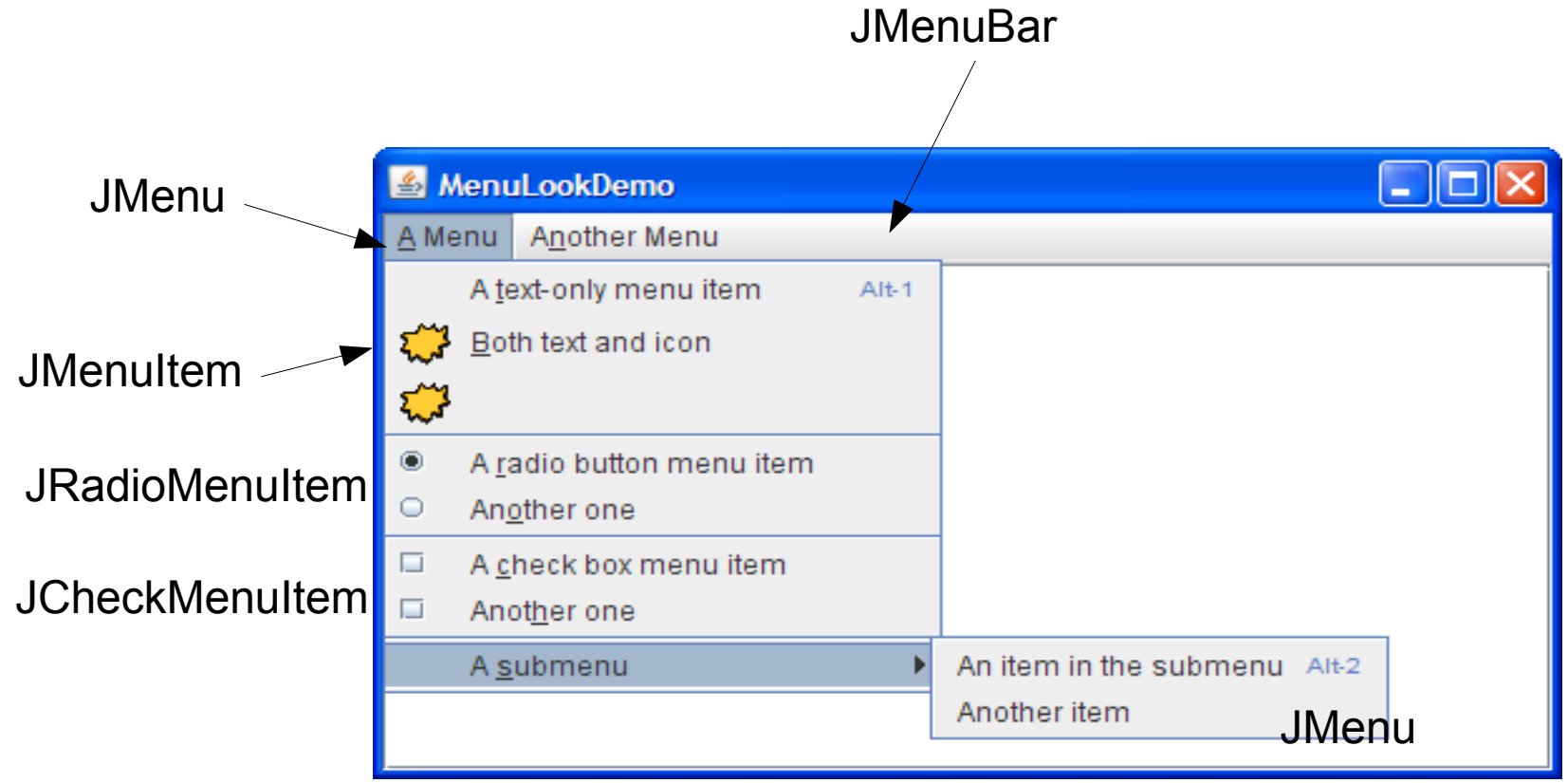
JSlider

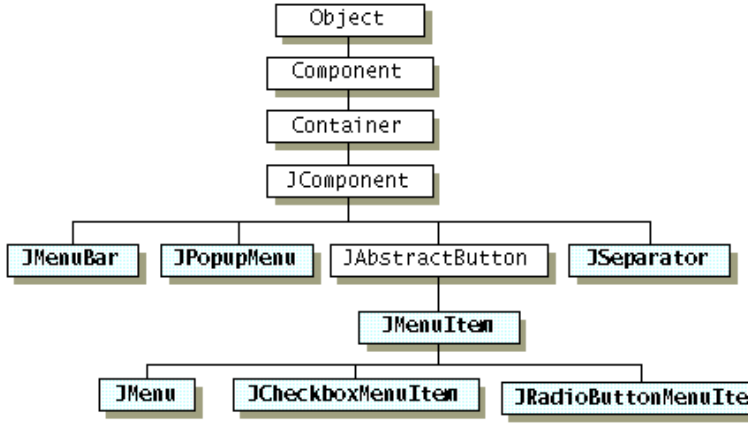
- Easily allow the user to enter a numeric value bounded by a minimum and maximum value



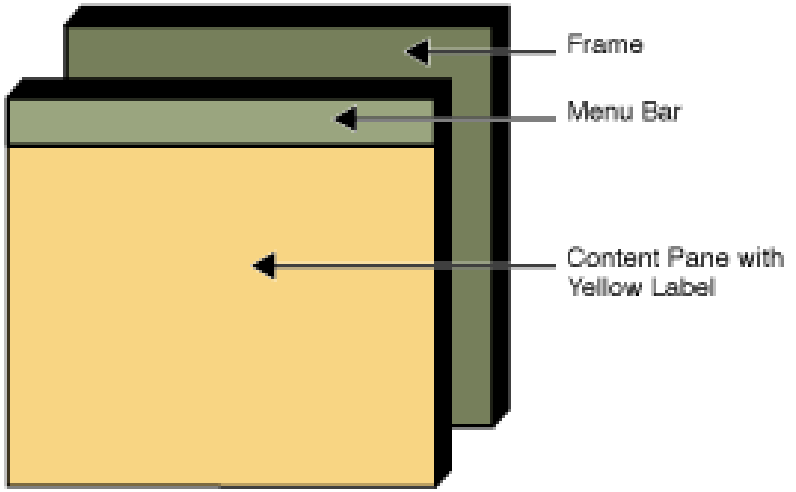

```
static final int FPS_MIN = 0;
static final int FPS_MAX = 30;
static final int FPS_INIT = 15;    //initial frames per second
. . .
JSlider framesPerSecond = new JSlider(JSlider.HORIZONTAL,
                                      FPS_MIN, FPS_MAX, FPS_INIT);

//Turn on labels at major tick marks.
framesPerSecond.setMajorTickSpacing(10);
framesPerSecond.setMinorTickSpacing(1);
framesPerSecond.setPaintTicks(true);
framesPerSecond.setPaintLabels(true);
```

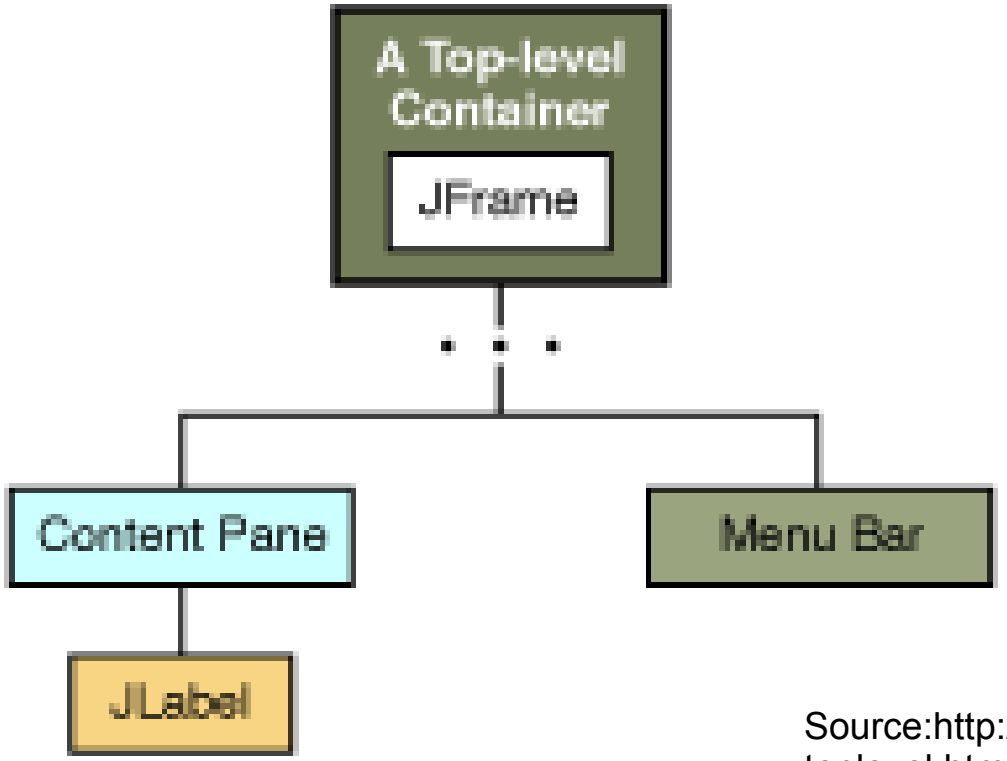
JMenuBar, JMenu, etc.





JFrame structure

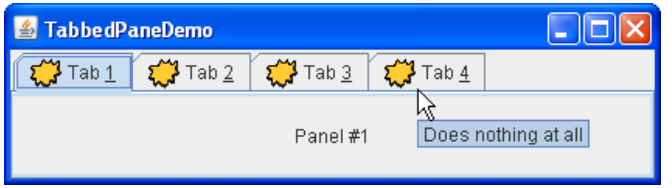


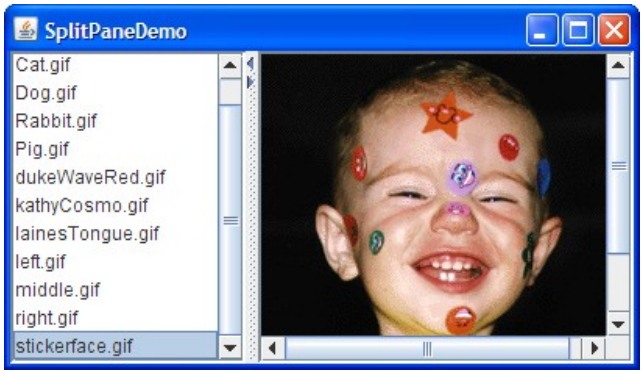


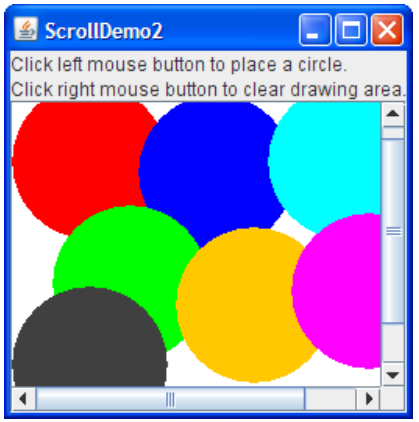
Source:http://docs.oracle.com/javase/tutorial/uiswing/components/toplevel.html

Some containers

- Containers
  - JPanel
  - JTabbedPane (multiples panels)
  - JSplitPane (with a separation)
  - JScrollPane (with scroll bars)

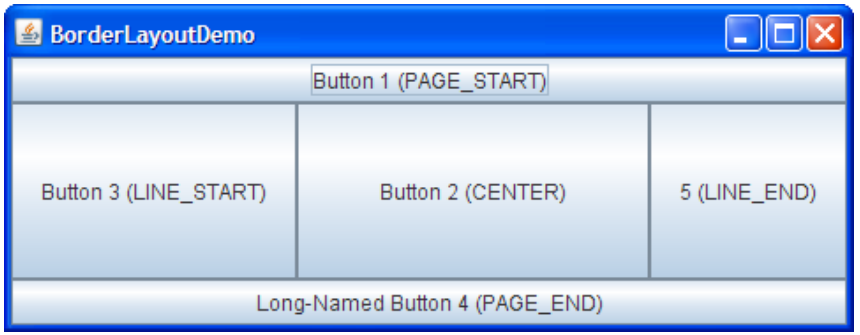






Basic Layout Managers

- BorderLayout



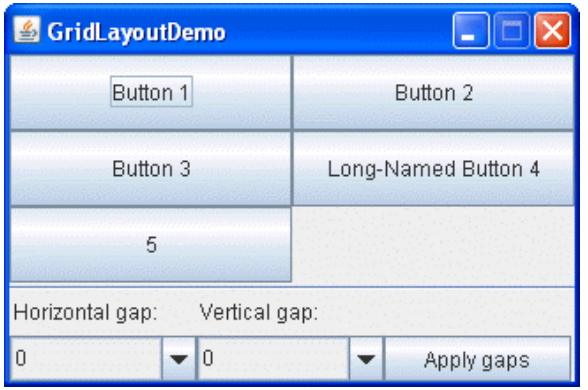

```
JPanel pane = new JPanel()

pane.setLayout(new BorderLayput());

JButton button = new JButton("Button 1 (PAGE_START)");
pane.add(button, BorderLayout.PAGE_START);

//Make the center component big, since that's the
//typical usage of BorderLayout.
button = new JButton("Button 2 (CENTER)");
button.setPreferredSize(new Dimension(200, 100));
pane.add(button, BorderLayout.CENTER);

button = new JButton("Button 3 (LINE_START)");
pane.add(button, BorderLayout.LINE_START);

button = new JButton("Long-Named Button 4 (PAGE_END)");
pane.add(button, BorderLayout.PAGE_END);

button = new JButton("5 (LINE_END)");
pane.add(button, BorderLayout.LINE_END);
```

Basic Layout Managers

- FlowLayout: component are laid out in a line

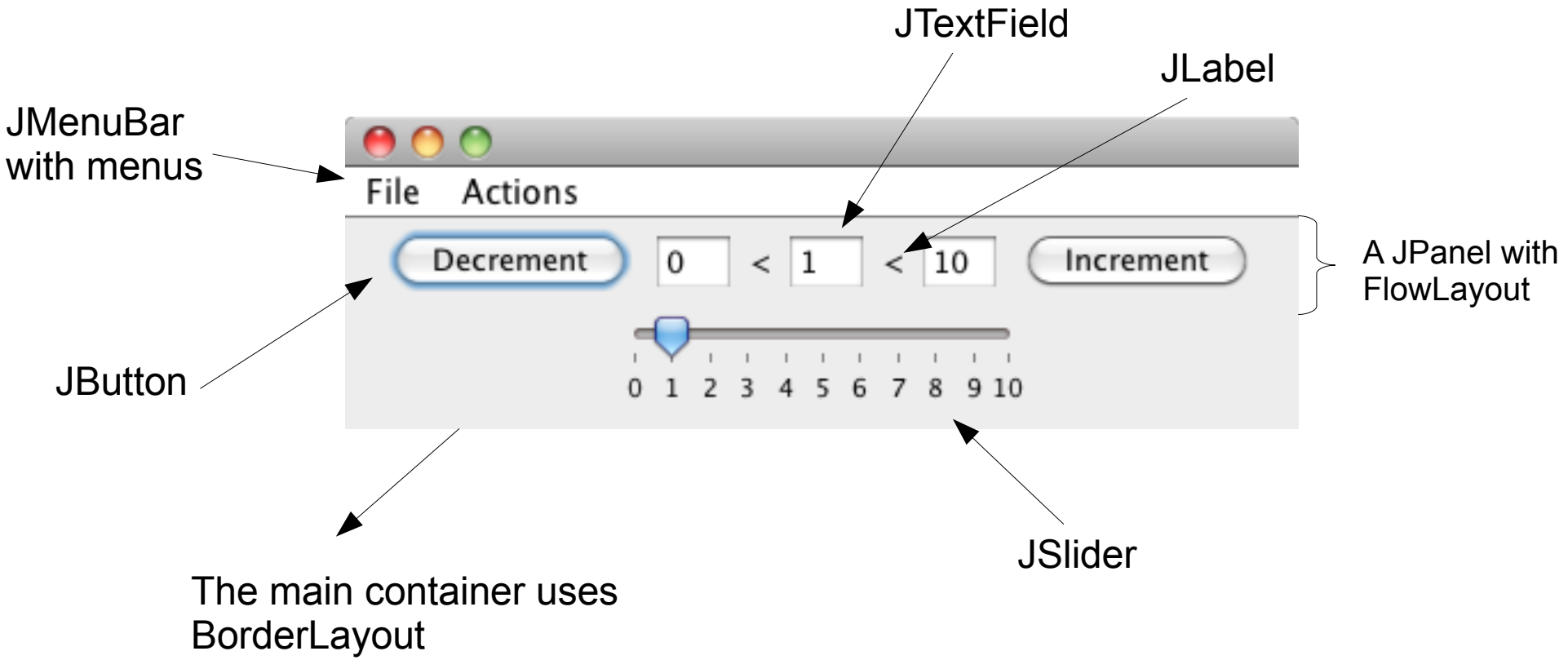
Other

- GridLayout: uses an uniform grid
- GridBagLayout: A grid with cells of different size based on constraints. One of the most powerful layout manager.



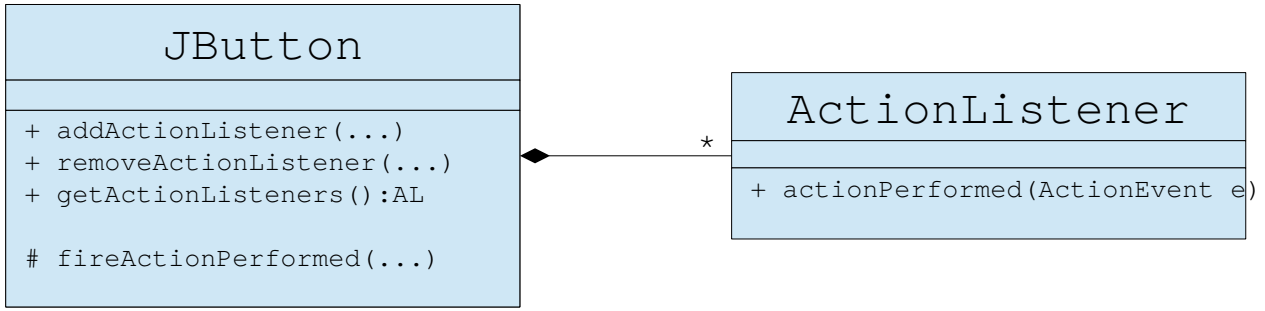
Exercise

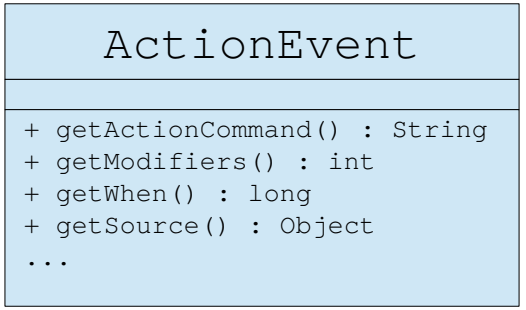
- Create a view for the bounded integer model
  - The view has to update itself when it is notified of the model changes



Classic action listener

- Buttons, textfields, menu items, etc. delegate the treatment of events to ActionListener
  - This is a kind of observer/observable pattern





Example

- Listener on the button “Increment”

```
public class ViewBI extends JPanel {

	private JButton btInc = new JButton("+");

	public ViewBI() {
		this.setLayout(new FlowLayout());
		this.add(btInc);
	}

	public void addActionIncrementListener(ActionListener al) {
		btInc.addActionListener(al);
	}
}
```

```
public class IncrementController implements ActionListener {
	private BoundedInteger3 model;
	public IncrementController(BoundedInteger3 model) {
		this.model=model;
	}
	public void actionPerformed(ActionEvent event) {
		try {
			model.increment();
		}
		catch (RuntimeException e) {
			JOptionPane.showMessageDialog(
		(Component) event.getSource(), e.getMessage());
		}
	}
}
```

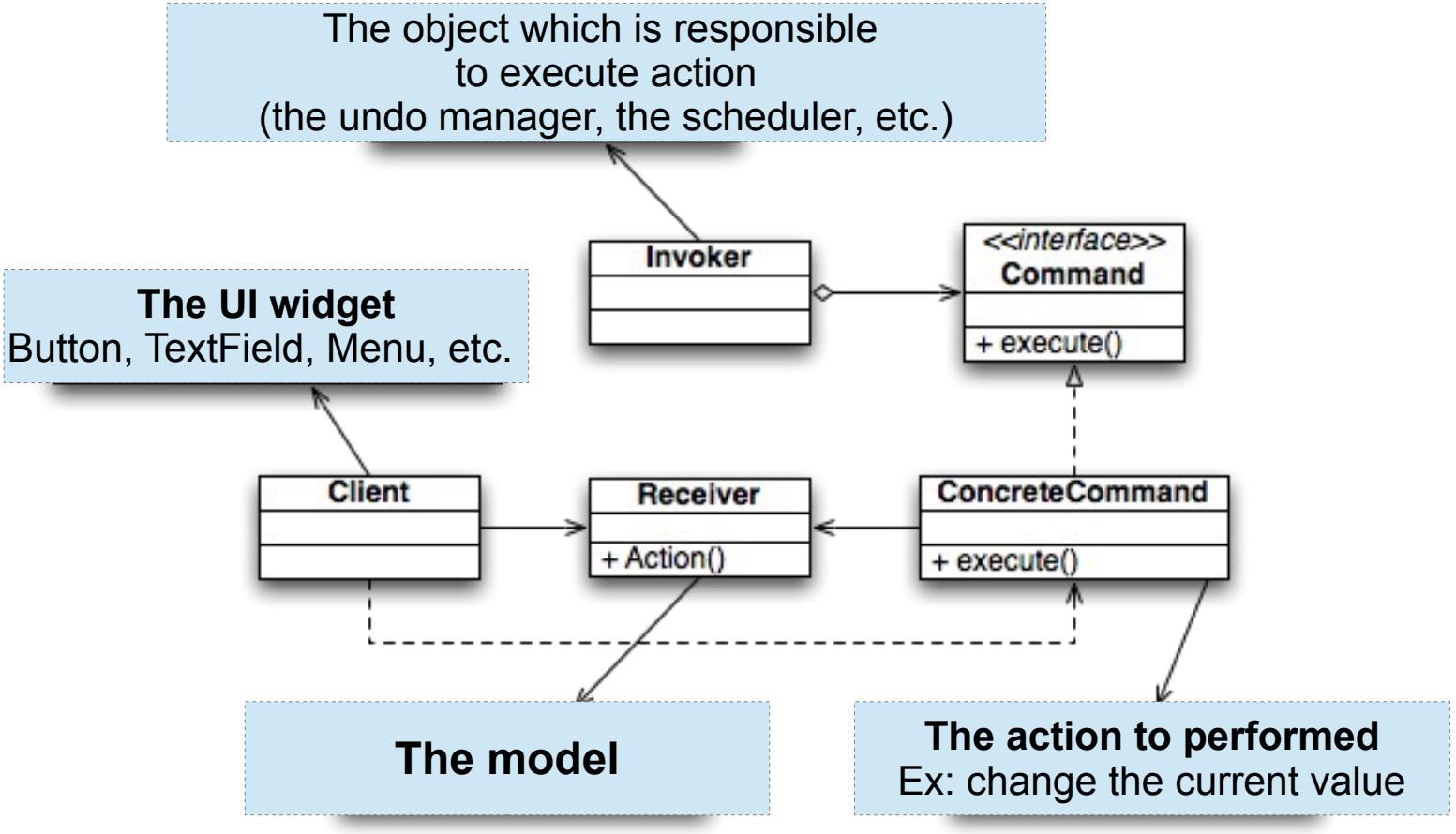
```
public static void main(String args[]) {
	final BoundedInteger3 model = new BoundedInteger3(1,0,10);
	final IncrementController aIncrement = new IncrementController(model);

	javax.swing.SwingUtilities.invokeLater(new Runnable() {
	   public void run() {
		ViewBI view = new ViewBI();
		view.addActionIncrementListener(aIncrement);
		JFrame frame =  new JFrame("Bounded Integer");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setContentPane(view);

		frame.pack();
		frame.setVisible(true);
	  }
	  });
}
```

Command Pattern

- Used to encapsulate a query into an object
  - This object is called the command
  - The command encapsulates the method to be called and several parameters
- To separate the description of an action from the object which execute this action
- Example:

  A print library use a command PrintJob. An instance of this class contains the number of copies, the name of the job, and the document to print.
  This command is then sent to the printer scheduler

Command Pattern



javax.swing.Action

- It is an extension of ActionListener
  - It extends ActionListener
  - Allows a same functionality to be accessed by several controls (UI widget)
    - Example : Action increment can accessible via menu and button
- Action are based on Command pattern
  - They encapsulates command infos
    - name, description, KeyStroke, mnemonic, an icon, etc.
    - A state: disable or enable

Action Interface

```
public interface Action extends ActionListener {
      public static final String DEFAULT = "Default";
      public static final String NAME = "Name";
      public static final String SHORT_DESCRIPTION = "ShortDescription";
      public static final String LONG_DESCRIPTION = "LongDescription";
      public static final String SMALL_ICON = "SmallIcon";
      public static final String ACTION_COMMAND_KEY = "ActionCommandKey";
      public static final String ACCELERATOR_KEY = "AcceleratorKey";
      public static final String MNEMONIC_KEY = "MnemonicKey";
      public Object getValue(String key);
      public void putValue(String key, Object value);
      public void setEnabled(boolean b);
      public boolean isEnabled();
      public void addPropertyChangeListener(PropertyChangeListener l);
      public void removePropertyChangeListener(PropertyChangeListener l);
}
```

The Action interface defines a set of keys, such as `Action.SHORT_DESCRIPTION`, which can be used to configure an Action.

Methods to access/update action properties. Example:

```
cutAction.putValue(Action.SHORT_DESCRIPTION, "Cut Command");
```

when an action is disabled, all the components which use the action will become disabled.

When a UI widget is linked to an action, it is PropertyListener of the action properties
If a property change, then the UI widget can change its appearance

How to use Actions ?

- Extends the class javax.swing.AbstractAction
- Set up the action properties in the constructor

```
public IncrementAction(BoundedInteger3 model) {
      super();
      this.model=model;
      putValue(Action.NAME, "Name of Action");
      putValue(Action.LONG_DESCRIPTION, "Increment the bounded integer");
      putValue(Action.ACCELERATOR_KEY,KeyStroke.getKeyStroke('+'));
}
```

- Implement actionPerformed method

```
  public void actionPerformed(ActionEvent event) {
            // do actions on the model
  }
```

Action myAction = new ….
myButton.setAction(myAction);
JMenu myMenu = new Jmenu(“blabla”);
myMenu.add(myAction);

Other Listener

- ChangeListener (used by JSlider)
  - similar to a property change listener

```
public class SliderListener implements ChangeListener {
    public void stateChanged(ChangeEvent e) {
        JSlider source = (JSlider)e.getSource();
        if (!source.getValueIsAdjusting()) {
            int fps = (int)source.getValue();
            ...
        }
    }
}
```

- MouseListener, MouseMotionListener
- KeyListener
- …

Exercise

- Actions
  - Create Actions for the operations increment and decrement on the BoundedInteger model
  - Add these actions to your view
  - Test
- Actions listeners of the model
  - Disable or enable action according to the value of the model. To do this, actions have to be listener of the model.
  - Test
- Create menu in your application and add increment and decrement action to the menu.

Exercise

- Actions
  - Create Actions for the operations increment and decrement on the BoundedInteger model
  - Add these actions to your view
  - Test
- Actions listeners of the model
  - Disable or enable action according to the value of the model. To do this, actions have to be listener of the model.
  - Test
- Create menu in your application and add increment and decrement action to the menu.

Exercise

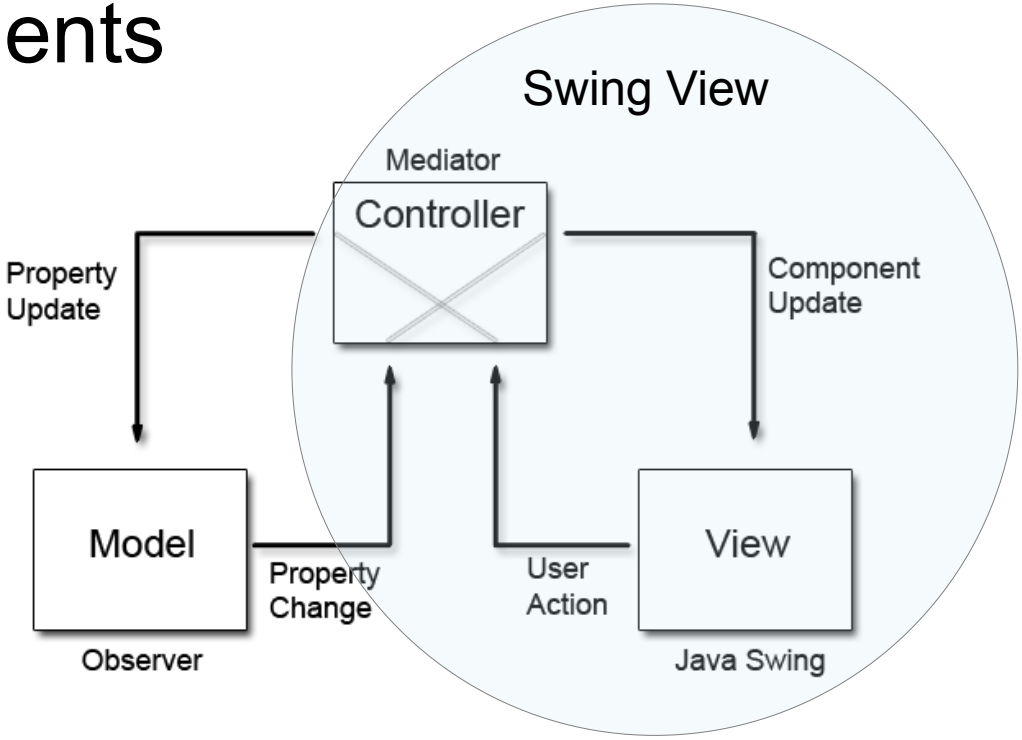
- Complete the interaction on the view by adding listeners for
  - The currentValue text field (ActionListener)
  - The bound text fields
  - The slider (Change Listener)
- Test
  - The application should be now complete
  - Try to create 2 views on the same model

Synchronized Model

- Create now two instances (and associated views) of your model
- Create a listener that guarantees that a first integer is always lower than or equals to the second one
  - What is the problem ?
  - How to solve it ?

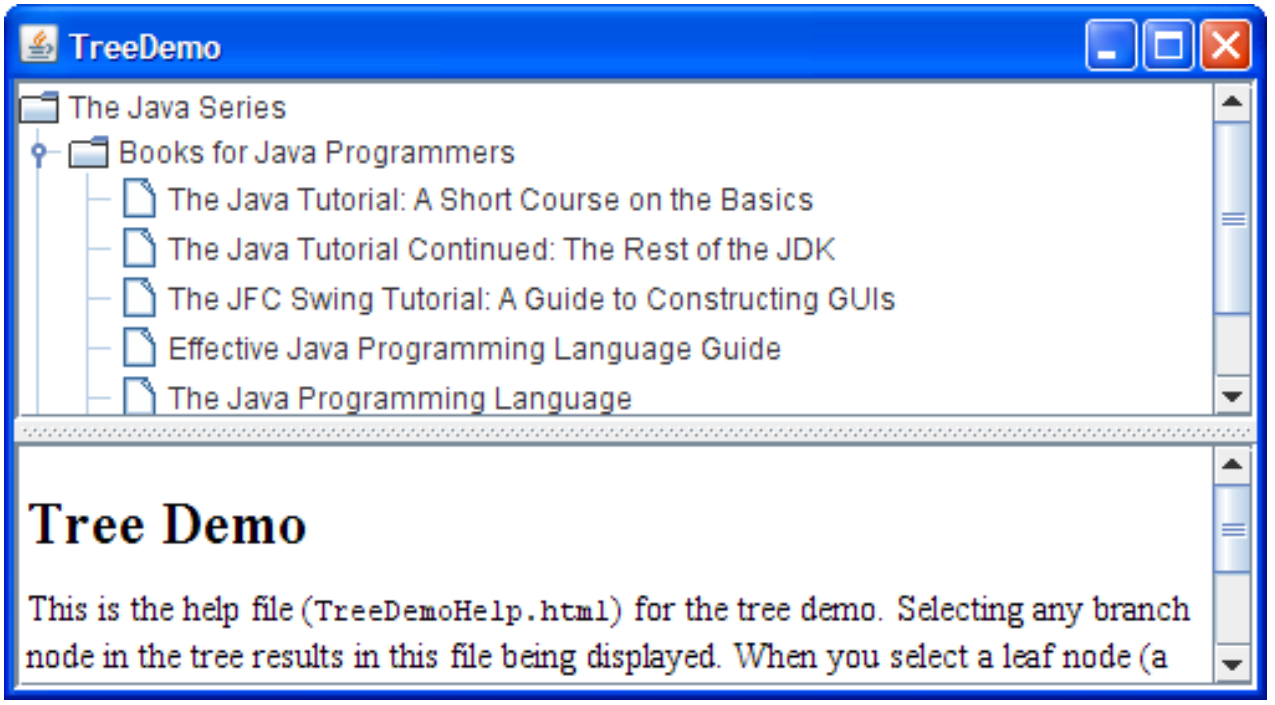
Swing and MVC

- Several high level component follows a M-VC design
  - The view and controller are somehow mixed
  - The model is separated
- Some of these components
  - JList
  - JTable
  - JTree

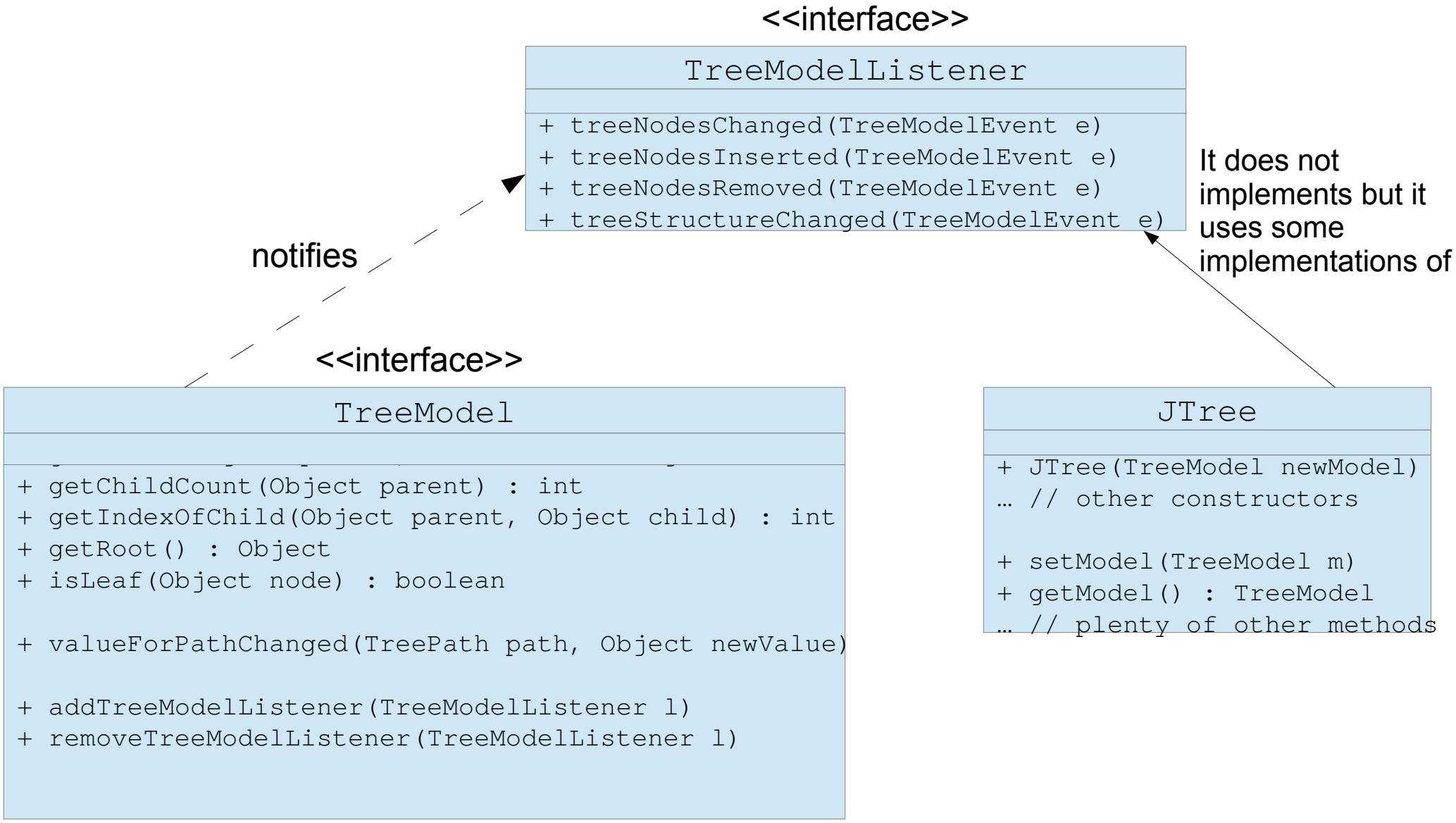


JTree

- Used to display hierarchical data
  - Model: have to implements TreeModel (or use DefaultTreeModel + TreeNode)
  - View: JTree (implements TreeModelListener)



JTree API & MVC



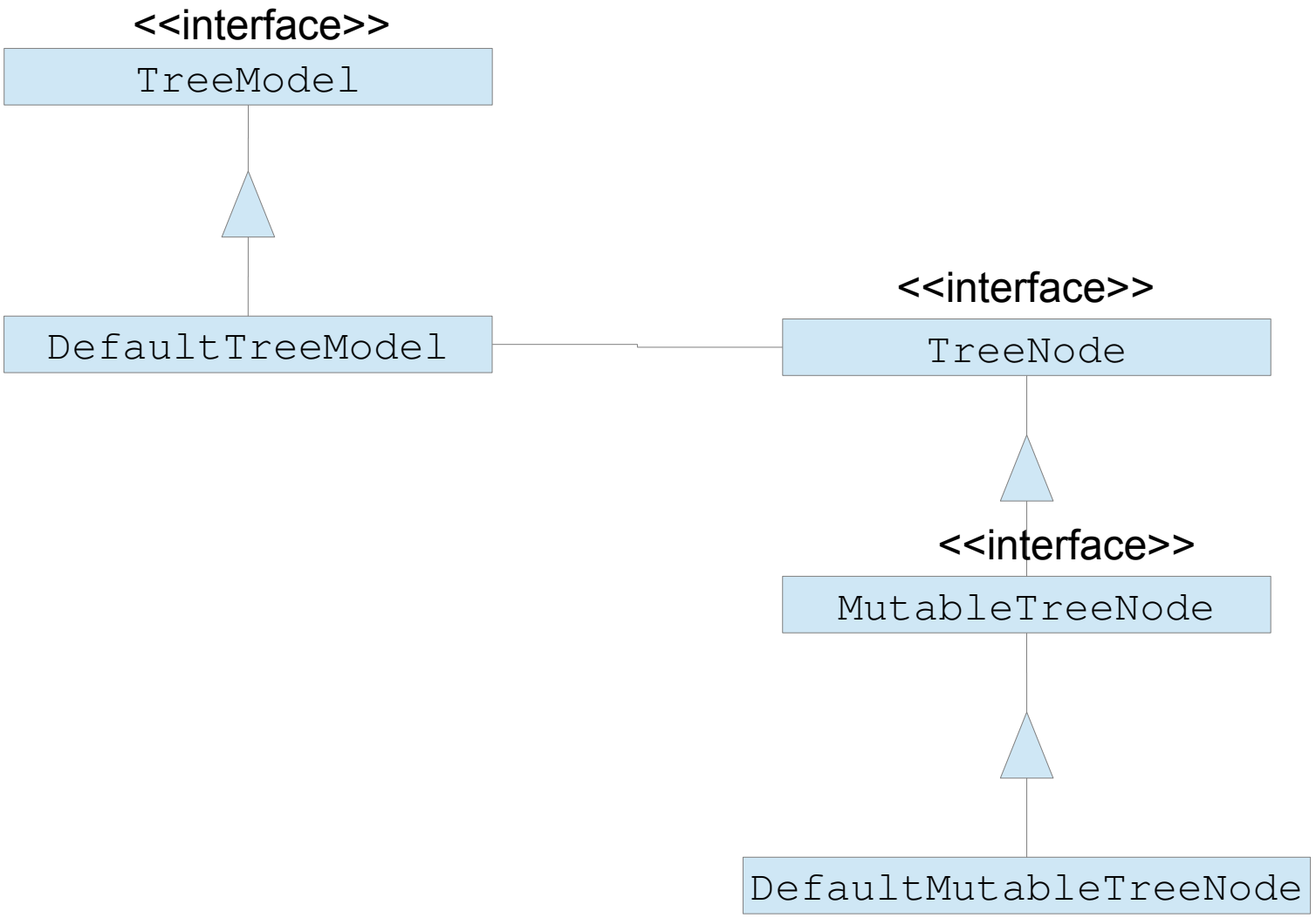
The model

The view + some controllers

TreeModelEvent

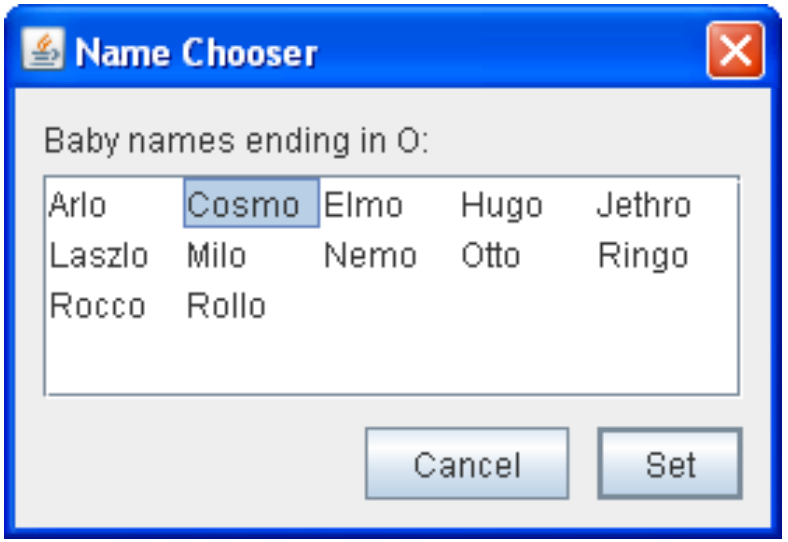
- Class used to describe a notification
  - It contains
    - the path from the root to the node from which the modification begin
    - The indexes and objects of children affected by the modif.
- Methods
  - `Object[] getPath() & TreePath getTreePath()`
  - `int[] getChildIndices() & Object[] getChildren()`
- Constructors
  - `TreeModelEvent(Object source, Object[] path)`
  - `TreeModelEvent(Object source, Object[] path, int[] childIndices, Object[] children)`
  - And the same with a `Treepath` instead for `Object[]`

The default impl. of TreeModel

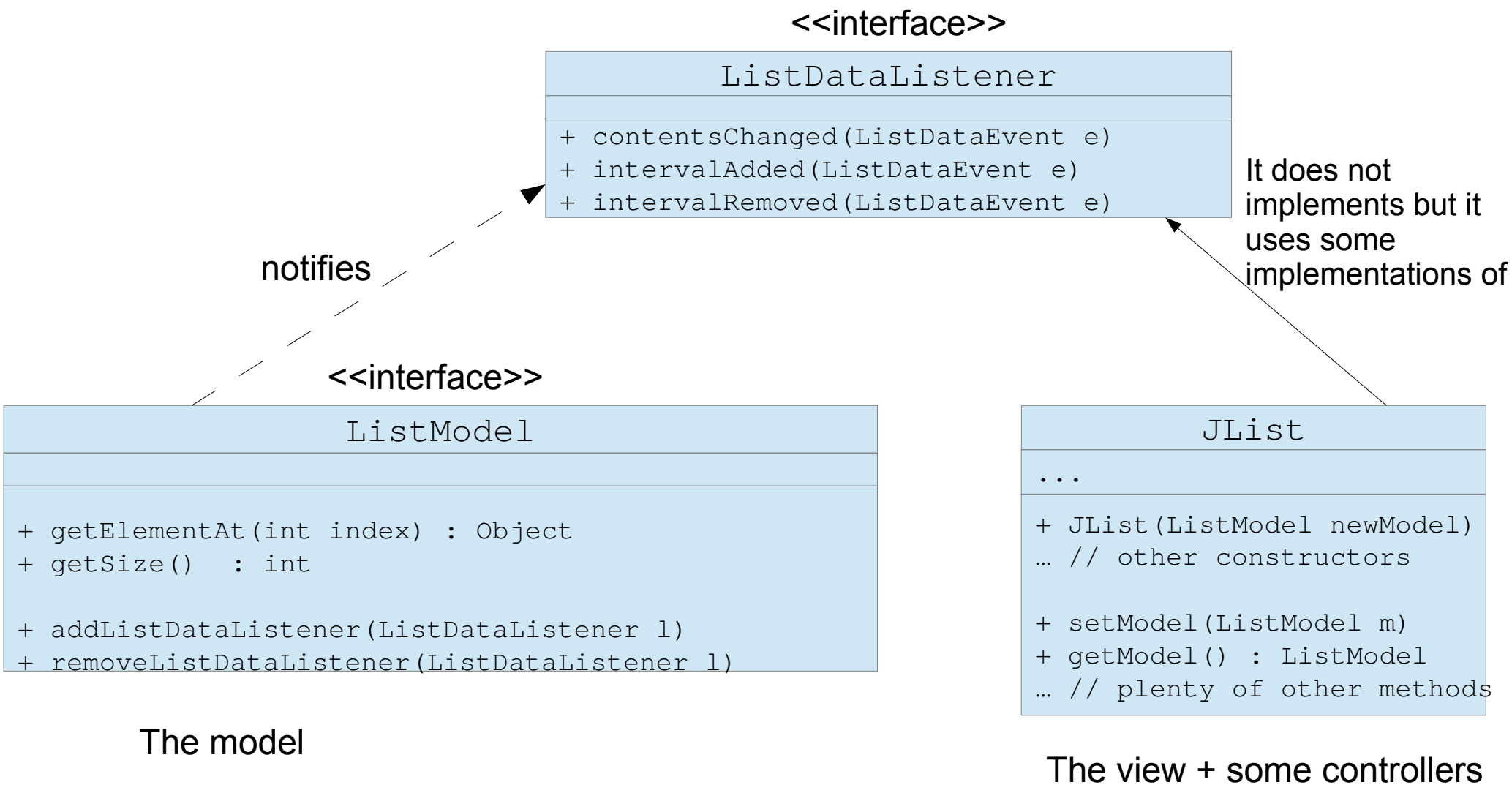


JList

- Presents a list of items in one or more columns to the user
  - Model: have to implements ListModel (or extends AbstractListModel)
  - View and controller: JList (implements ListDataListener)



JList API & MVC



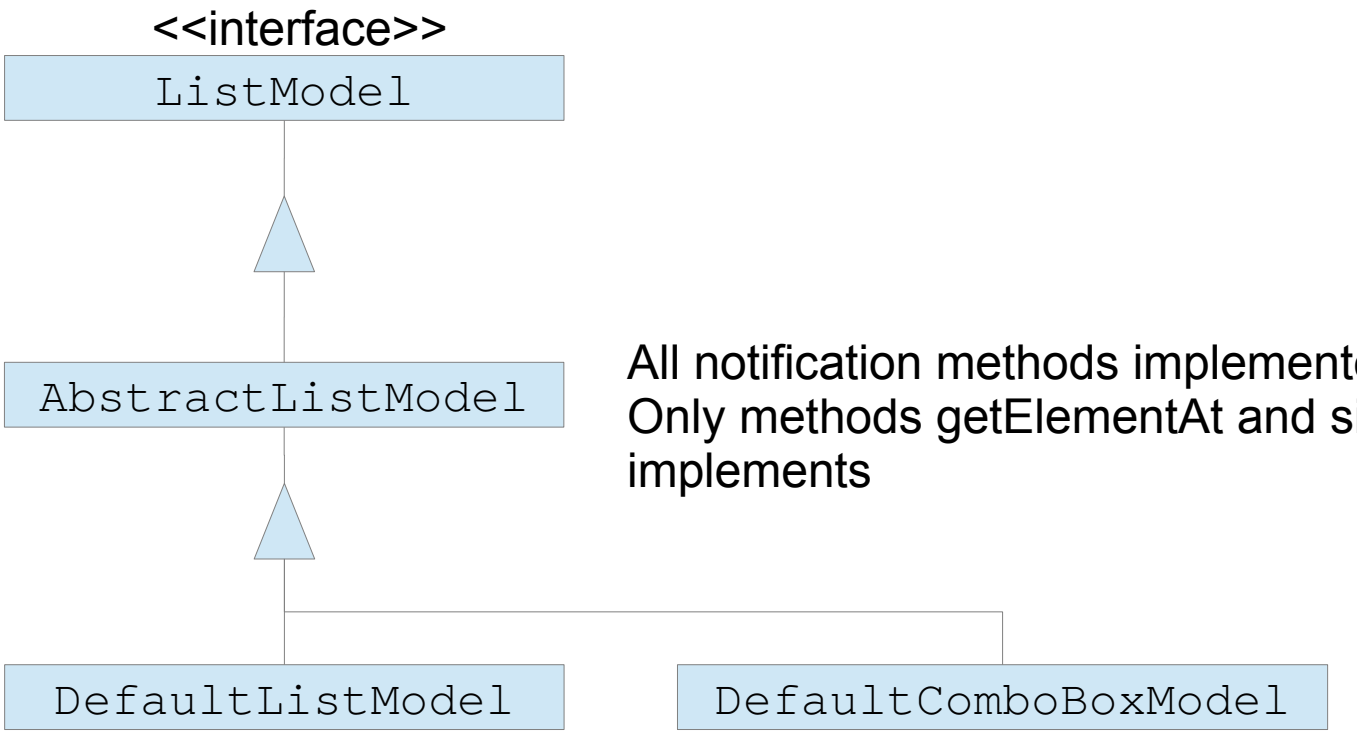
The model

The view + some controllers

ListDataEvent

- Class used for describing a notification
- Methods
  - `getIndex0()`: Returns the lower index of the range
  - `getIndex1()`: Returns the upper index of the range
  - `getType()`: can be
    - `ListEvent.CONTENTS_CHANGED`
    - `ListEvent.INTERVAL_ADDED`
    - `ListEvent.INTERVAL_REMOVED`
- Constructor
  - `ListDataEvent(Object source, int type, int index0, int index1)`

Default implementations of ListModel



All notification methods implemented
Only methods getElementAt and size to implements