GUI Programming with Java

More Layout Managers, MVC and Advanced Components

- We will look at…
  - Card and Box Layout
  - MVC
  - JSlider and JProgressBar

More Layouts available in SWING:

- We will revisit layouts and examine two more layout managers available in SWING:
    - BoxLayout
    - CardLayout
- Note: In the examples here we use methods to initialize the components\panels; it is good practice to use methods in this way for your programs

BoxLayout

- This layout simply stacks the components in a row or side-by-side
- It is a lot like FlowLayout but the programmer gets more control on the layout

BoxLayout

- The programmer simply chooses the orientation of the layout and adds the components

- There are final static variables to represent the orientation including:

  BoxLayout.X_AXIS (left to right in ‘x’ direction)

  BoxLayout.Y_AXIS (top to bottom, i.e., ‘y’ direction)

BoxLayout

- The sample program called BoxLayoutExample.java shows an example of BoxLayout being used with a set of labels (city names)

- The following lines of code are the key lines of code for setting the BoxLayout:

```
citiesPanel.setLayout(new BoxLayout(citiesPanel,BoxLayout.Y_AXIS));
```

- The above line of code sets the layout to BoxLayout and the orientation to vertical on panel citiesPanel

BoxLayout

- If you want to change the orientation for the layout you change the final static variable to:

  BoxLayout.X_AXIS

- Changing this setting will orient the components horizontally instead of vertically, then the components will fill the space required (BoxLayout.Y_AXIS)

- Components can be further spaced using fillers, e.g. *Box.createHorizontalGlue()* and *Box.createVerticalGlue()*

CardLayout

- This layout can be used when two or more components need to share the same space on the GUI (at different times – swapping in and out displays)
- The programmer sets ‘cards’ that can be switched in and out of the layout based on interactions
- The programmer can then switch from one card to another (the cards tend to be panels with various components)

CardLayout

- The example called CardLayoutExample.java shows an example of two panels (both at North) but with only one showing at a time (red text label or black text label)

- The two JPanels are created using two methods

- In the actionPerformed(ActionEvent e) the next card is displayed using the next() method of the CardLayout

```
CardLayout card = (CardLayout)displayPanel.getLayout();
card.next(displayPanel);
```

CardLayout

- The CardLayout next() method moves to the next card held in the card layout (will return to first card if end is reached)

- The panels were added to the card layout

```
displayPanel.setLayout(new CardLayout());
displayPanel.add(textPanel); //could add ID String “Red Text”
displayPanel.add(imagePanel); //could add ID String “Black Text”
```

- Once the cards are added the available switching methods can be invoked, e.g., next(), previous(), show(), first(), last() etc.

CardLayout

- first (Container parent) Flips to the first card of the container.
- next (Container parent) Flips to the next card of the container. If the currently visible card is the last one, this method flips to the first card in the layout.
- previous (Container parent) Flips to the previous card of the container. If the currently visible card is the first one, this method flips to the last card in the layout.
- last (Container parent) Flips to the last card of the container.
- show (Container parent, String name) Flips to the component that was added to this layout with the specified name

MVC

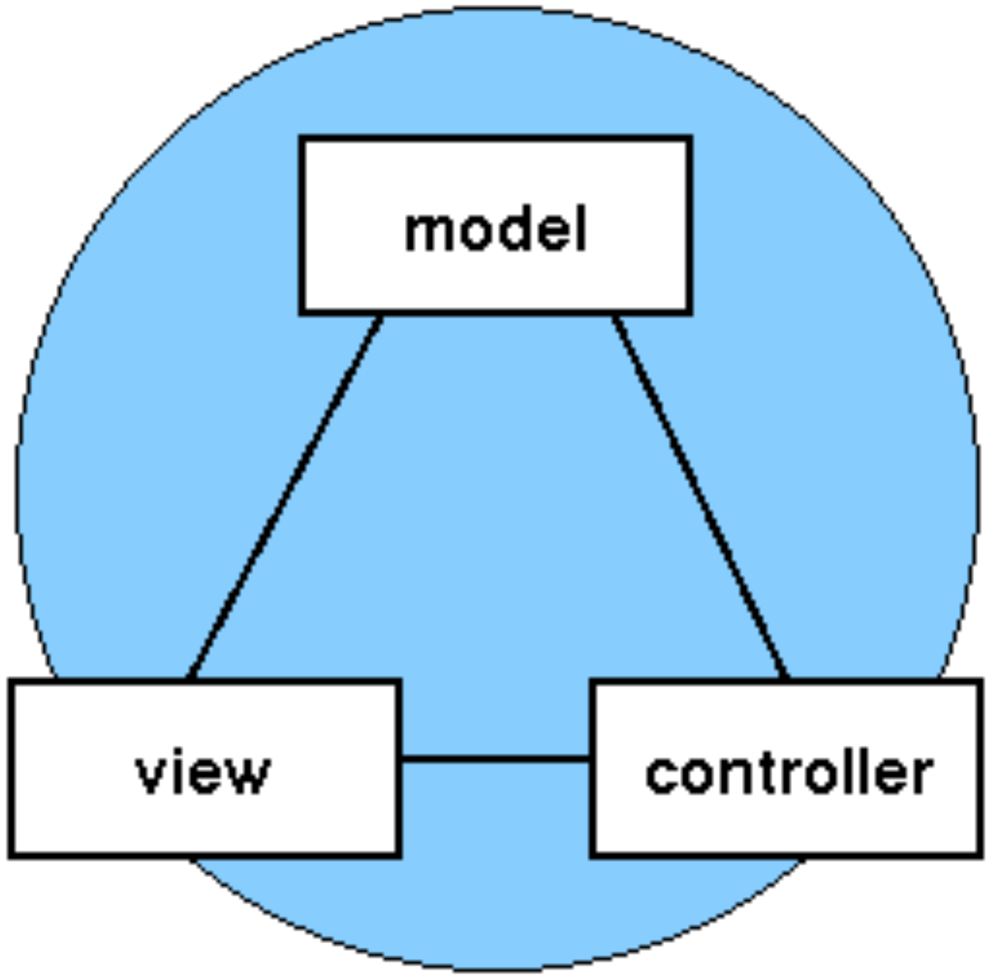
The Model-View-Controller Architecture

- The Model-View-Controller (MVC) design pattern was developed using the Smalltalk programming environment for the creation of user interfaces. It is a popular OO pattern

- The goal of the MVC design pattern is to separate the application object (model) from the way it is represented to the user (view) from the way in which the user controls it (controller).

- Before MVC, user interface designs tended to lump such objects together. MVC decouples them thus allowing greater flexibility and possibility for re-use. MVC also provides a powerful way to organise systems that support multiple presentations of the same information.

The Model-View-Controller Architecture



What is a design pattern?

- One of the main reasons computer science researchers began to recognize design patterns was to satisfy the need for good, simple, and reusable solutions.
- The term *design pattern* can sound a little bit formal to the beginner, but in fact a design pattern is just a convenient way of reusing object-oriented code between projects and between programmers.
- The idea behind design patterns is simple:

To catalog common characteristics between objects that programmers have often found useful.

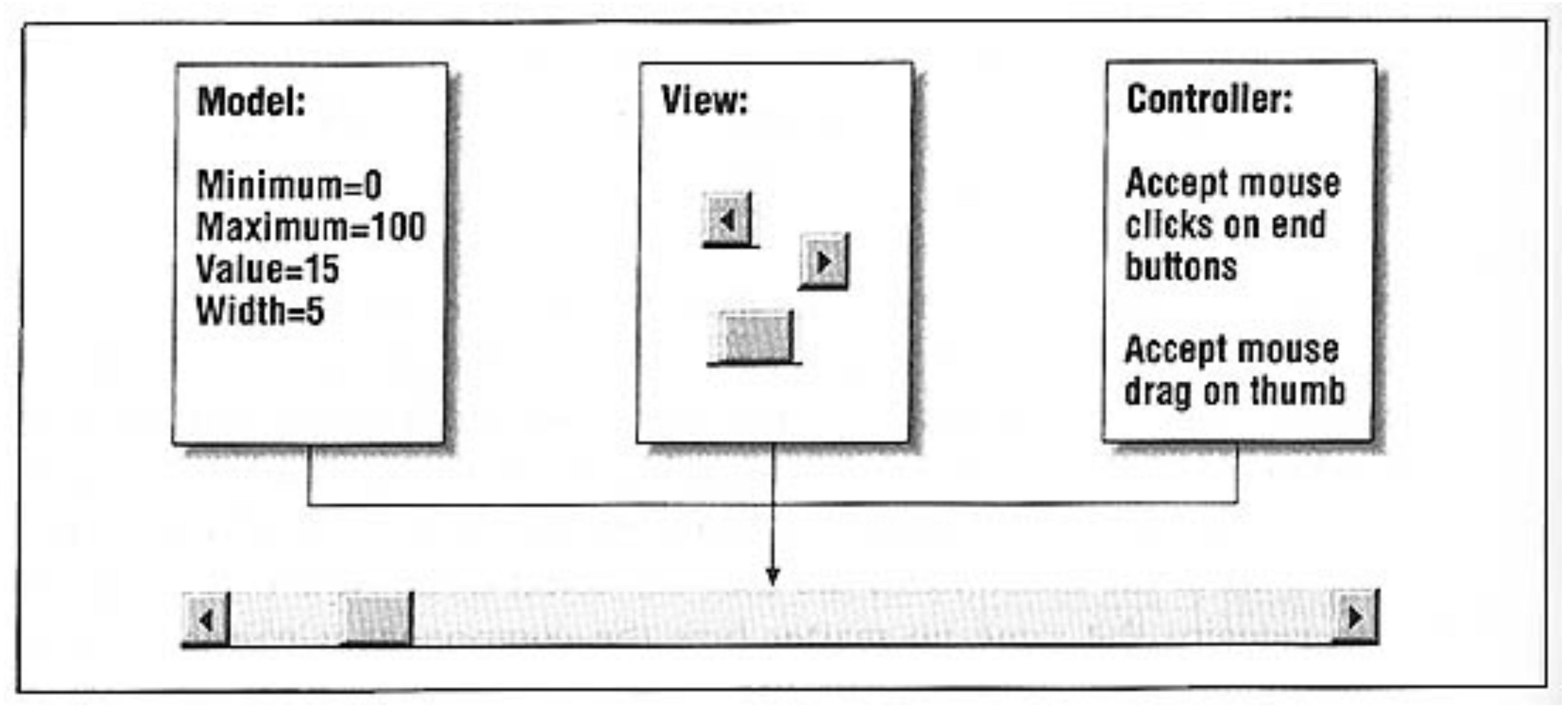
What is a design pattern?

- Some useful definitions of design patterns have emerged as the literature in this field has expanded:
  - "Design patterns are recurring solutions to design problems you see over and over." *[Smalltalk Companion]*
  - "Design patterns focus on the reuse of architectural design themes." [*Coplien and Schmidt, 1995*]
  - "A pattern addresses a recurring design problem that arises in specific design situations and presents a solution to it." *[Buschmann and Meunier, et al., 1996]*
  - "Patterns identify and specify abstractions that are above the level of single classes and instances, or of components." [Gamma, Helm, Johnson, and Vlissides, 1993]

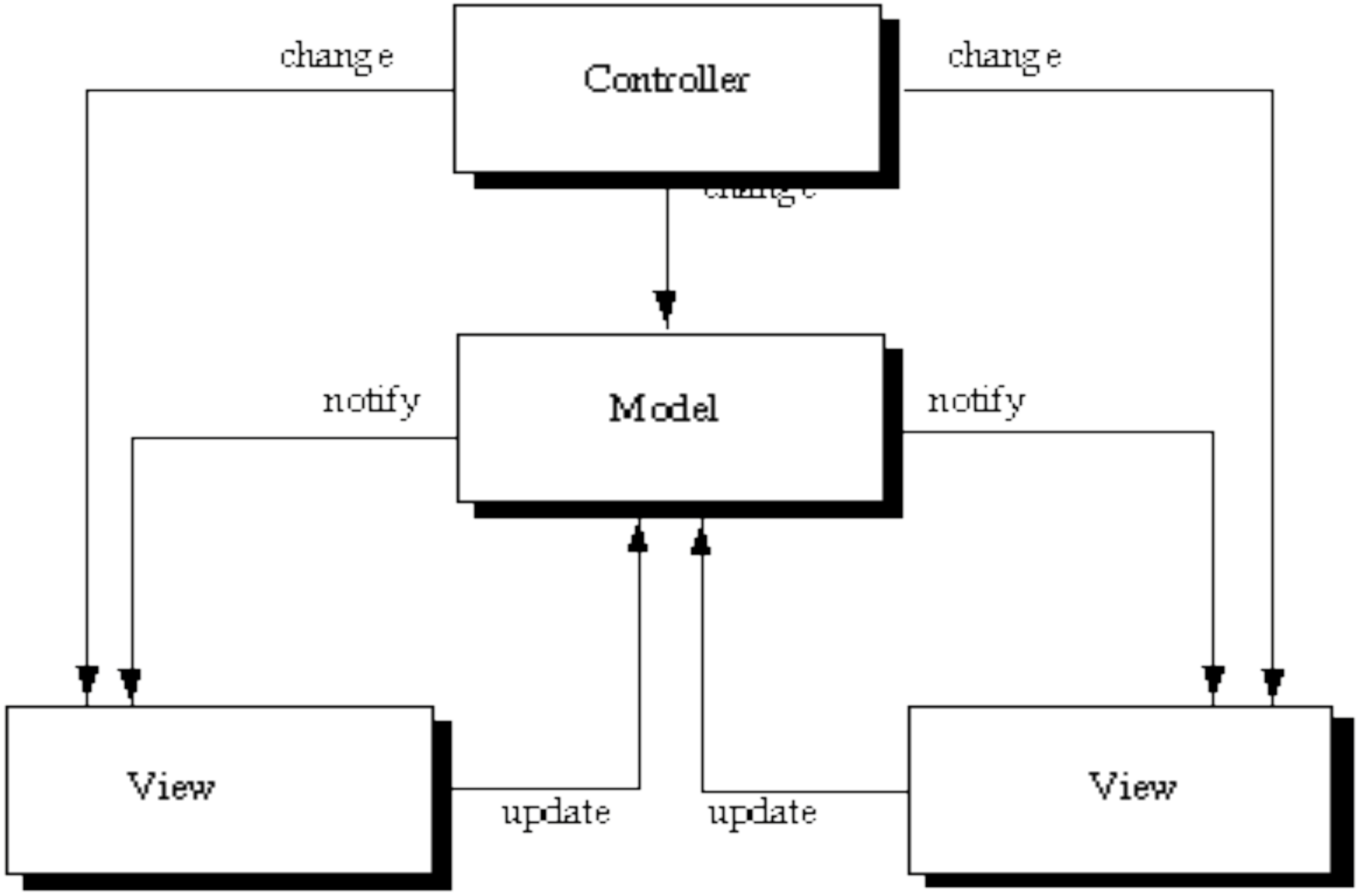
How is MVC a design pattern?

- In MVC, each aspect of the problem is a separate object, and each has its own rules for managing its data.
- Communication between the user, the graphical user interface, and the data should be carefully controlled; this separation of functions accomplishes that.
- Three objects talking to each other using this restrained set of connections is an example of a powerful design pattern.
- This architecture follows the goals of a design pattern, which describes how objects communicate without becoming entangled in each other’s data models and methods.

MVC Example


Model:
Minimum=0
Maximum=100
Value=15
Width=5
View:
Controller:
Accept mouse clicks on end buttons
Accept mouse drag on thumb

The Model-View-Controller Architecture



The Model

- The model object knows about all the data that need to be displayed.
- It also knows about all the operations that can be applied to transform that object.
- However, it knows nothing whatever about the GUI, the manner in which the data are to be displayed, nor the GUI actions that are used to manipulate the data.
- The data are accessed and manipulated through methods that are independent of the GUI.

The Model (2)

- A simple example of a model would be a clock object.
  - It has intrinsic behavior whereby it keeps track of time by updating an internal record of the time ever second.
  - The object would provide methods which allow view objects to query the current time.
  - It would also provide methods to allow a controller object to set the current time.

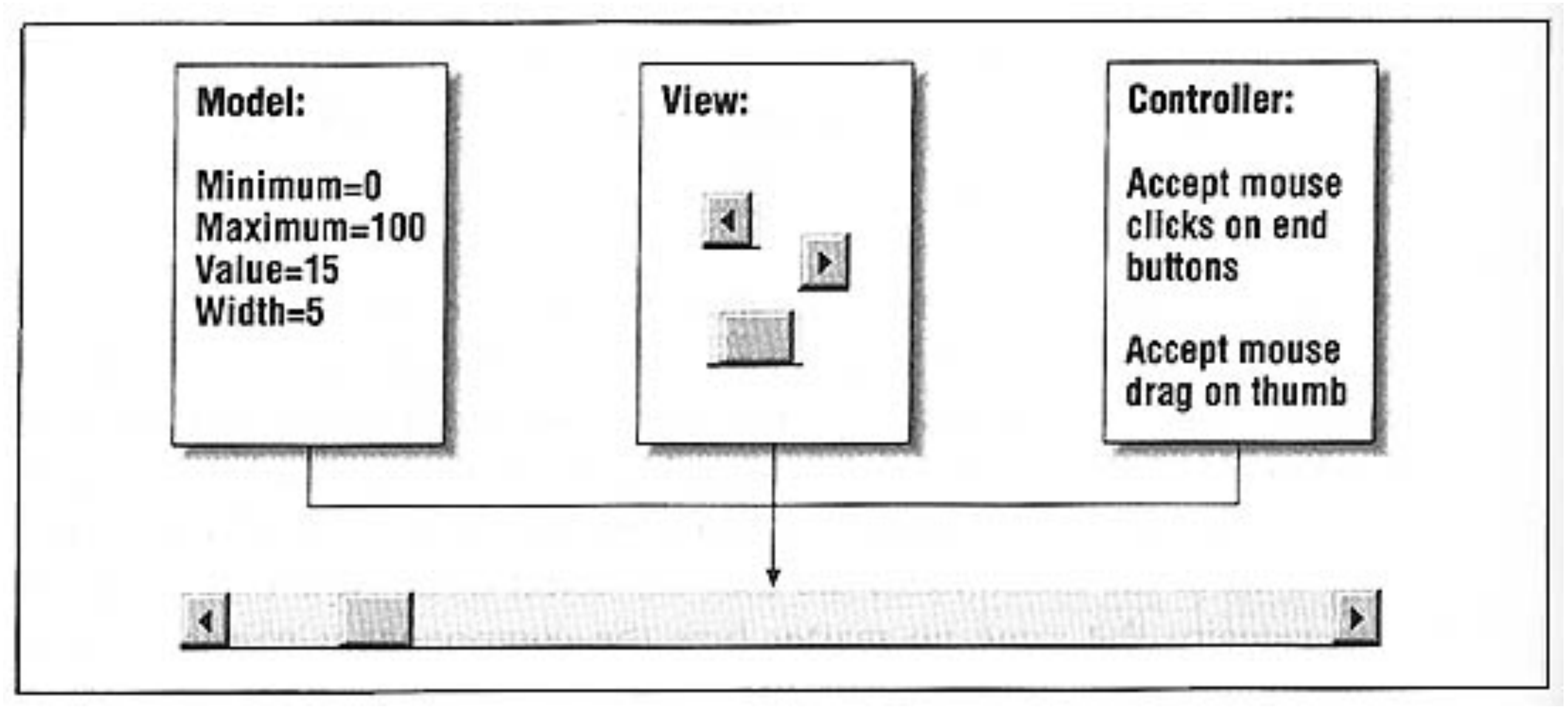
The View Object

- The view object refers to the model.
- It uses the query methods of the model to obtain data from the model and then displays the information.
- The display can take any form
  - in the clock example, one view object could display the time as an analogue clock
  - another could show it as a digital clock
- The different displays would have no bearing whatsoever on the intrinsic behavior of the clock.

The Controller Object

- The controller object knows about the physical means by which users manipulate data within the model.
- In a GUI for example, the controller object would receive mouse clicks or keyboard input which it would translate into the manipulator method which the model understands.
- For example, the clock could be reset directly by typing the current time into the digital clock display.
- The controller object associated with the view would know that a new time had been entered and would call the relevant "SetTime" method for the model object.

MVC Example


Model:
Minimum=0
Maximum=100
Value=15
Width=5
View:
Controller:
Accept mouse clicks on end buttons
Accept mouse drag on thumb

MVC Example (2)

- **Model**
  - Represents the state of the component
  - Data is independant of the component's visual representation
- **View**
  - How the component is represented on the screen (LnFs)
- **Controller**
  - Describes how the component interacts with the user

- The scrollbar uses the information in the model to determine how far into the scrollbar to render the thumb and how wide the thumb should be.

Advantages of the Model-View-Controller Architecture (1)

- The MVC architecture has the following benefits:

1) **Multiple views using the same model:** The separation of model and view allows multiple views to use the same enterprise model. Consequently, an enterprise application's model components are easier to implement, test, and maintain, since all access to the model goes through these components.

2) **Easier support for new types of clients:** To support a new type of client, you simply write a view and controller for it and wire them into the existing enterprise model.

Advantages of the Model-View-Controller Architecture (2)

3) **Clarity of design:** By glancing at the model's public method list, it should be easy to understand how to control the model's behavior. When designing the application, this trait makes the entire program easier to implement and maintain.

4) **Efficient modularity:** of the design allows any of the components to be swapped in and out as the user or programmer desires - even the model! Changes to one aspect of the program aren't coupled to other aspects, eliminating many nasty debugging situations. Also, development of the various components can progress in parallel, once the interface between the components is clearly defined.

Advantages of the Model-View-Controller Architecture (3)

5) **Ease of growth:** Controllers and views can grow as the model grows; and older versions of the views and controllers can still be used as long as a common interface is maintained.

6) **Distributable:** With a couple of proxies one can easily distribute any MVC application by only altering the startup method of the application.

How does MVC fit in with our study of SWING?

- When the JAVA development team where putting together SWING they had five basic development goals

1. *Be implemented entirely in Java* to promote cross-platform consistency and easier maintenance.
2. *Provide a single API capable of supporting multiple look-and-feels* so that developers and end-users would not be locked into a single look-and-feel.
3. *Enable the power of model-driven programming* without requiring it in the highest-level API.
4. *Adhere to JavaBeansTM design principles* to ensure that components behave well in IDEs and builder tools.
5. *Provide compatibility with AWT APIs where there is overlapping, to leverage the AWT knowledge base and ease porting*.

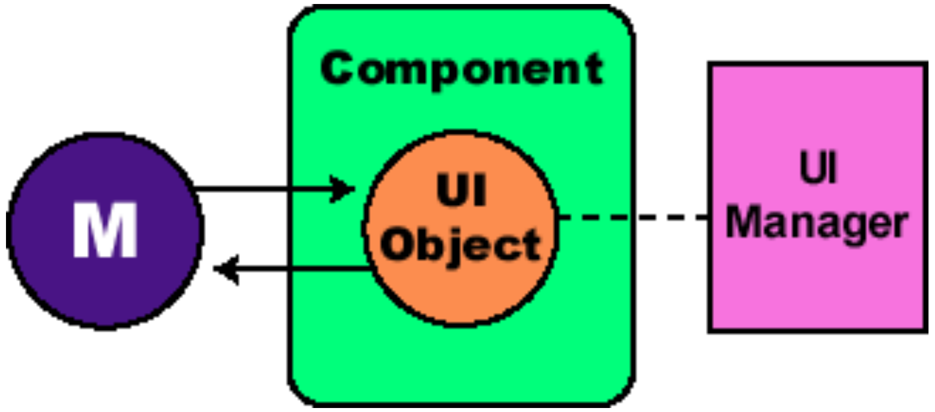
How does MVC fit in with our study of SWING?

- Early on, MVC was a logical choice for Swing because it provided a basis for meeting the first three of the teams design goals within the bounds of the latter two.

- It was quickly discovered that this split didn't work well in practical terms because the view and controller parts of a component required a tight coupling (for example, it was very difficult to write a generic controller that didn't know specifics about the view).

- The development team then collapsed these two entities into a single UI (user-interface) object, as shown in the following diagram:

How does MVC fit in with our study of SWING?

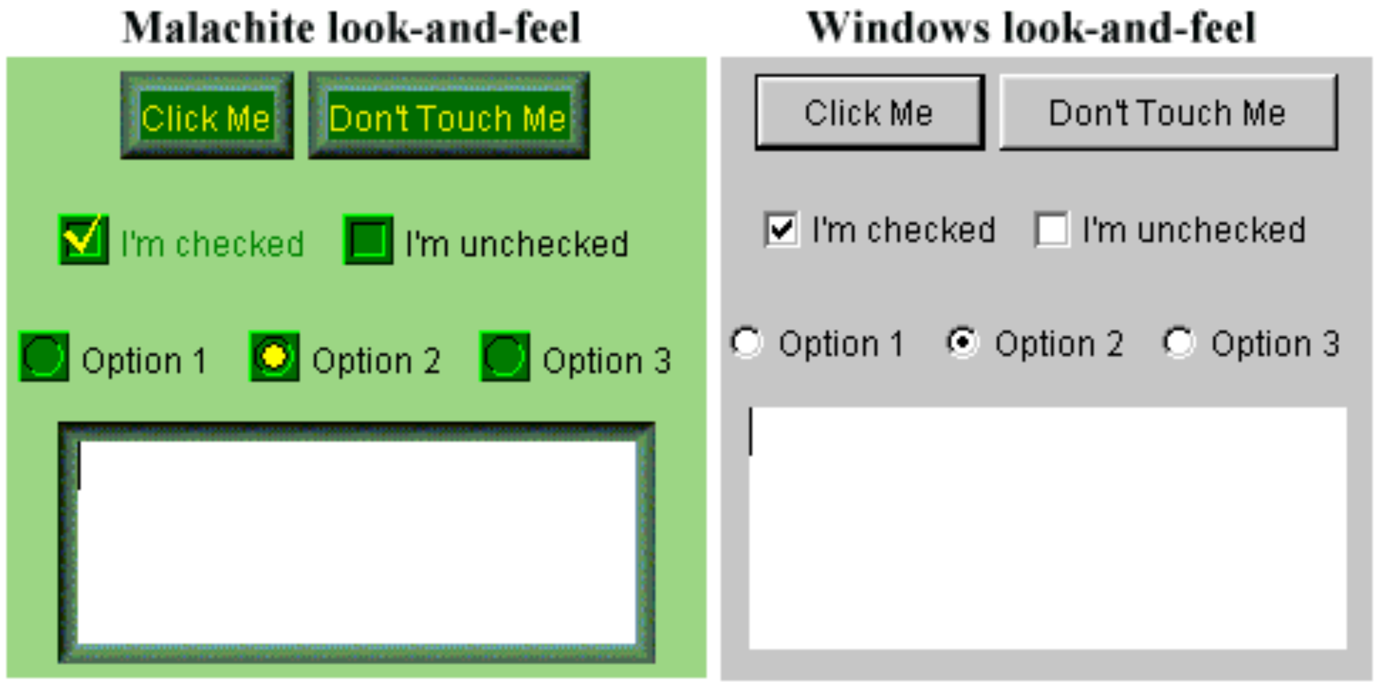


- **The UI delegate object shown in this picture is sometimes called a delegate object, or *UI delegate*.**
- **As the diagram illustrates, Swing architecture is loosely based -- but not *strictly* based -- on the traditional MVC design.**
- **In the world of Swing, this new quasi-MVC design is sometimes referred to a *separable model architecture*.**
- **Swing's separable model design treats the model part of a component as a separate element, just as the MVC design does. But Swing collapses the view and controller parts of each component into a single UI (user-interface) object.**

Pluggable Look and Feel?

- One of the major advantages MVC architecture provides is the ability to customize the "look" and "feel"of a component without modifying the model.

- The graphic shows a group of components using two different user interfaces.



Pluggable Look and Feel?

- The important point to make about this figure is that the components shown are actually the same, but they are shown using two different *look-and-feel* implementations.
- Swing includes several sets of UI delegates. Each set contains ComponentUI implementations for most Swing components and we call each of these sets a *look-and-feel* or a *pluggable look-and-feel* (PLAF) implementation.
- The javax.swing.plaf package consists of abstract classes derived from ComponentUI, and the classes in the javax.swing.plaf.basic package extend these abstract classes to implement the Basic look-and-feel (our standard appearance for buttons etc).

Pluggable Look and Feel?

- This is the set of UI delegates that all other look-and-feel classes are expected to use as a base for building off of.
- There are three pluggable look-and-feel implementations derived from the Basic look-and-feel:
  - Windows: com.sun.java.swing.plaf.windows.WindowsLookAndFeel
  - CDE\Motif: com.sun.java.swing.plaf.motif.MotifLookAndFeel
  - Metal (default): javax.swing.plaf.metal.MetalLookAndFeel

Pluggable Look and Feel?

- There is also a MacLookAndFeel for simulating Macintosh user interfaces, but this does not ship with Java 2 and must be downloaded separately.
- The Windows and Macintosh pluggable look-and-feel libraries are only supported on the corresponding platform.
- To change the current look-and-feel of an application we can simply call the UIManager’s setLookAndFeel() method, passing it the fully qualified name of the LookAndFeel to use.

Pluggable Look and Feel?

```
try
    { UIManager.setLookAndFeel( "com.sun.java.swing.plaf.motif.MotifLookAnd
    Feel"); SwingUtilities.updateComponentTreeUI(myJFrame);
}

catch (Exception e) { System.err.println("Could not load LookAndFeel");
}
```

JSlider and JProgressBar

JSlider

- Slider bars as used to select some value (usually between a pre-defined max and min value)

- For example if a slider were to be used to select a percentage it might be set as minimum 0 and maximum 100

- Listeners and handlers can then be used to respond based on the user input

JSlider

- The example called JSliderExample.java shows how to create a simple slider, the slider is created as follows:

```
JSlider slider = new JSlider();
```

- The ChangeListener interface can be used to track changes on the slider and is added using addChangeListener method

- In this example when the value changes in the slider the stateChanged(ChangeEvent e) is called and the text area displays the current value

JSlider

- The minimum and maximum values are set for the slider:

```
slider.setMaximum(100);
slider.setMinimum(0);
```

- In the example you can see the text in the label change continuously as the slider is modified
- A button could be used to invoke the use of the current value set by getting the text from the field at a given moment

JProgressBar

- Progress bars are normally shown when a task is to be completed but may take some time

- The progress bar will display a percentage complete to the user so that the user is aware some activity is occurring (otherwise the user could think the system is locked up)

- In Java the JProgressBar is used for this purpose

JProgressBar

- In the example called ProgBarExample.java it shows the creation of a JProgressBar using:

  progBar = **new JProgressBar();**

- Once the progress bar is initialized there must be some value changing that progresses the bar (this could be a percentage download for example)

- In the example supplied a method called awaitProg() is used to add ten to the value in the bar every second

JProgressBar

- As the total of the value increases every second the progress bar changes value

- In ten seconds the progress bar completes and a beep occurs when the progress bar reaches the maximum value

- The current value of the progress bar is set using:

```
progBar.setValue(progBar.getValue()+10);
```