

Model-View Controller

Advanced GUI concepts

Background

The model-view controller (MVC) paradigm was developed at the Xerox Palo Alto Research Center (PARC). MVC was central to the architecture of the multi-windowed Smalltalk environment used to create the first graphical user interfaces. The approach taken was borrowed by the developers of the Apple Macintosh and many imitators in the years since. In such an interface, input is primarily by the mouse and keyboard; output is a mix of graphics and textual components as appropriate. MVC is elegant and simple, but rather unlike the approach of traditional application programs.

Required Reading

<http://java.sun.com/products/jfc/tsc/articles/architecture/index.html>



Sun Developer Network (SDN) » search tips

APIs Downloads Products Support Training Participate

Developers Home > Products & Technologies > Java Technology > Java Platform, Standard Edition (J2SE) > Desktop Java > JFC > News & Updates > The Swing Connection > Reference > Technical Articles and Tips >

Article

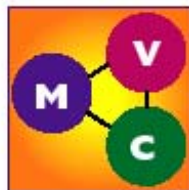
A Swing Architecture Overview



 Print-friendly Version

The Inside Story on JFC Component Design

By Amy Fowler



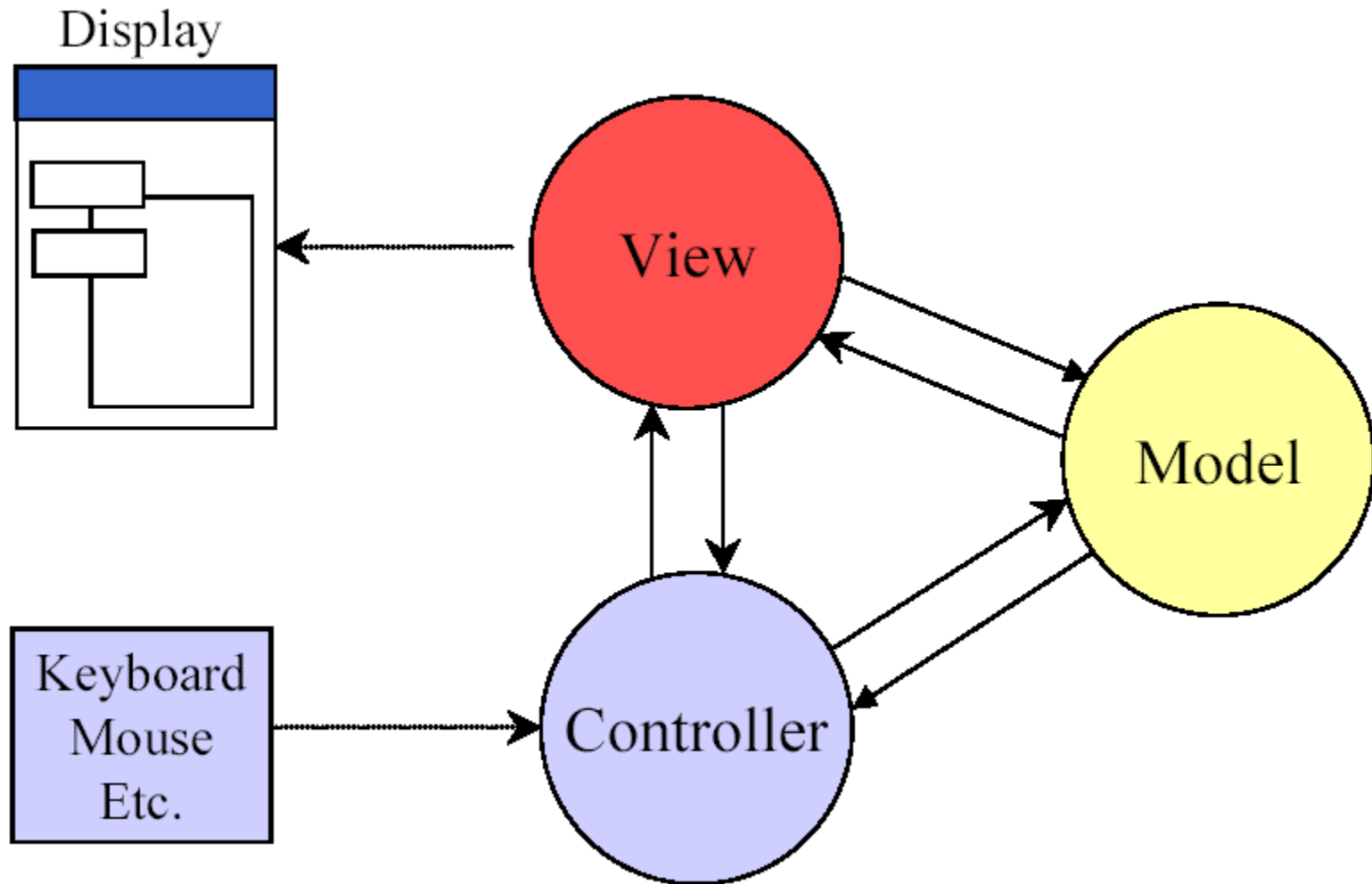
Most Swing developers know by now that Swing components have a separable model-and-view design. And many Swing users have run across articles saying that Swing is based on something called a "modified

MVC (model-view-controller) architecture."

MVC paradigm

- Traditional paradigm...
 - Input → processing → output
- MVC paradigm...
 - Controller → model → view

MVC Schematic



Controller tasks

1. Receive user inputs from mouse and keyboard
2. Map these into commands that are sent to the model and/or viewport to effect changes in the view
 - E.g., detect that a button has been pressed and inform the model that the button state has changed

Model tasks

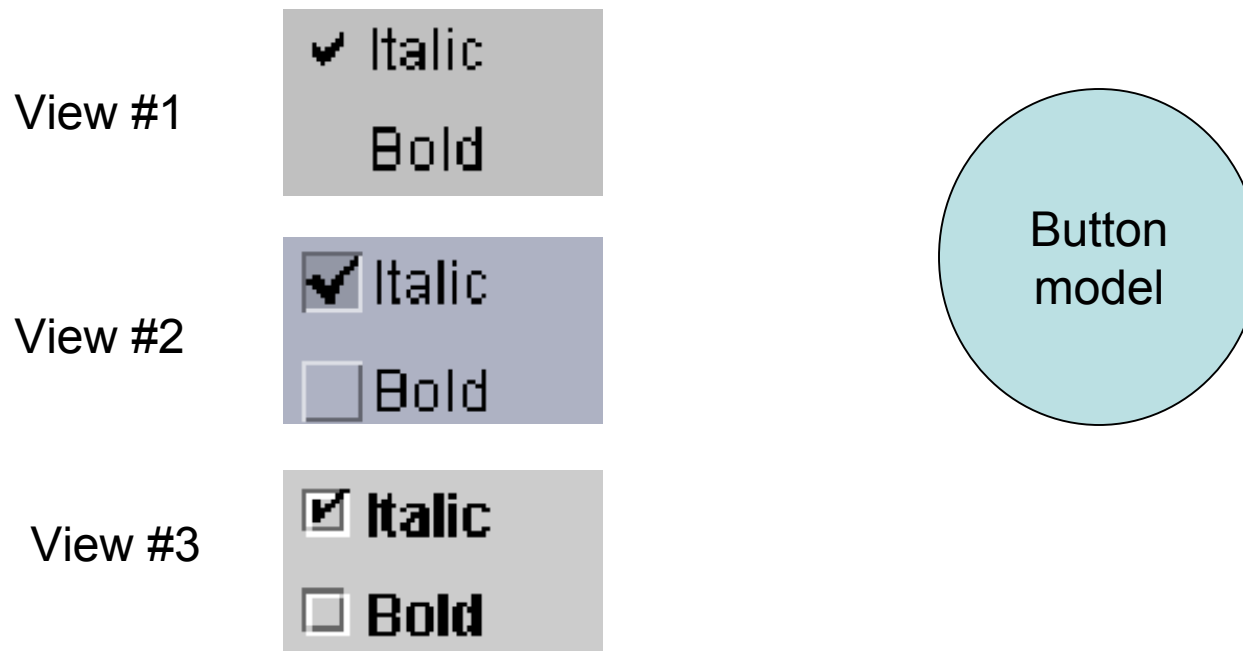
1. Store and manage data elements, such as state information
2. Respond to queries about its state
3. Respond to instructions to change its state
 - E.g., the model for a radio button can be queried to determine if the button is pressed

View task

1. Implements a visual display of the model
 - E.g., a button has a colored background, appears in a raised perspective, and contains an icon and text; the text is rendered in a certain font in a certain color

MVC Concepts – *multiple views*

- Any number of views can subscribe to the model



MVC Concepts - *model changes*

- What happens when the model changes?
- E.g., a button is pressed (the state of the button has changed!)
- The model must notify the view
- The view changes the visual presentation of the model on the screen

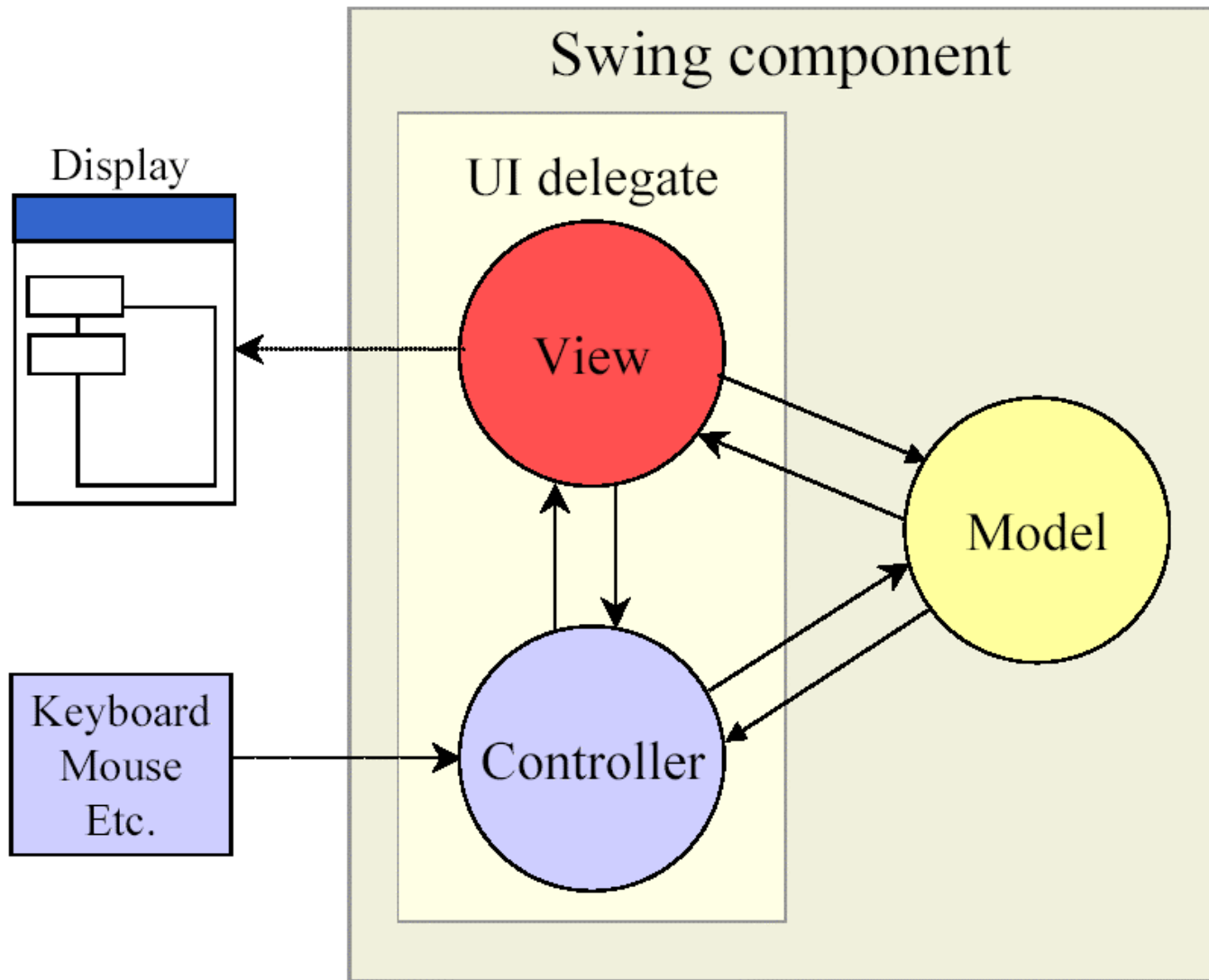
Benefits of MVC Architecture

- Improved maintainability
 - Due to modularity of software components
- Promotes code reuse
 - Due to OO approach (e.g., subclassing, inheritance)
- Model independence
 - Designers can enhance and/or optimize model without changing the view or controller
- Pluggable look and feel
 - New L&F without changing model
 - Multiple views use the same data

MVC and Swing

- Swing designers found it difficult to write a generic controller that didn't know the specifics about the view
- So, they collapsed the view and controller into a single UI (user interface) object known as a delegate (the UI is *delegated* to this object)
- This object is known as a UI delegate

MVC and Swing (2)

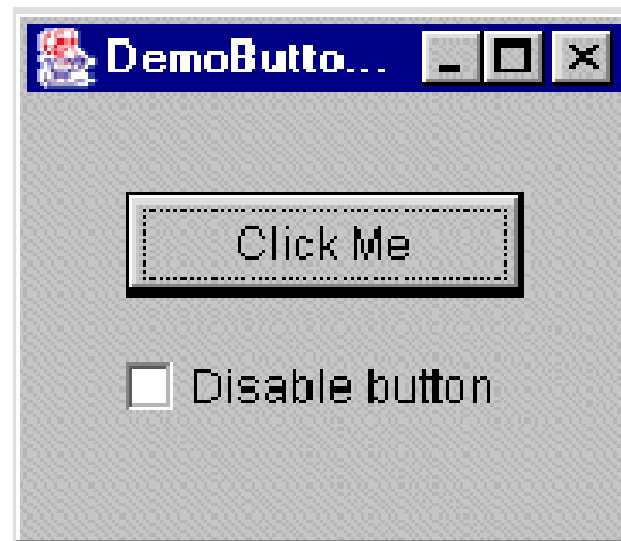


M - Swing Models

- In Swing, many models exist as interfaces
 - Eg., ButtonModel, BoundedRangeModel, ComboBoxModel, ListModel, ListSelectionModel, TableModel, Document
- The interface is implemented in model classes
- Usually there is a default model class that is automatically associated with a component (whew!)
 - E.g., DefaultButtonModel implements ButtonModel
 - E.g., AbstractDocument implements Document (PlainDocument is a subclass of AbstractDocument)

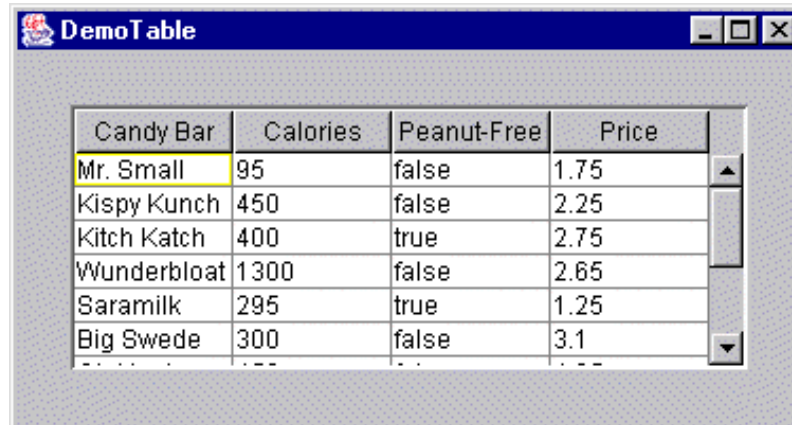
Example Program

DemoButtonModel.java



Example Program

DemoTableModel.java

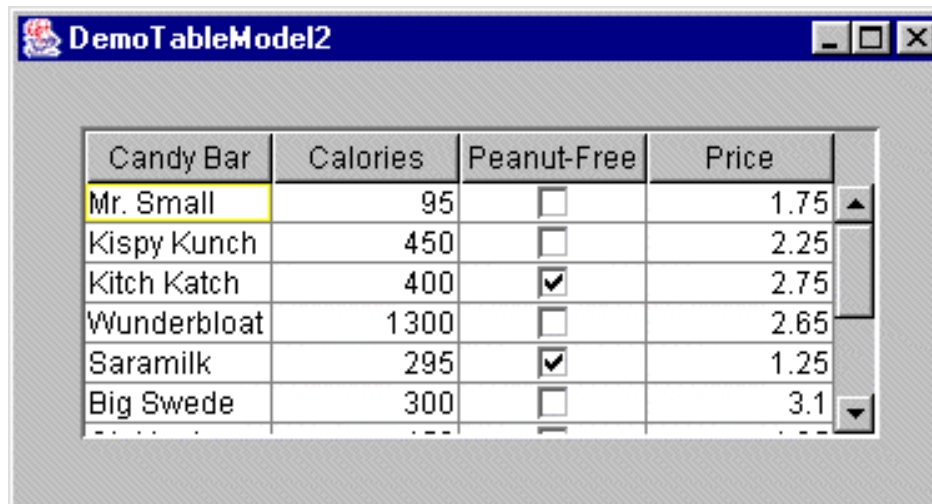


Candy Bar	Calories	Peanut-Free	Price
Mr. Small	95	false	1.75
Kippy Kunch	450	false	2.25
Kitch Katch	400	true	2.75
Wunderbloat	1300	false	2.65
Saramilk	295	true	1.25
Big Swede	300	false	3.1

Instead of passing the data directly to the `JTable` object, we create a data model, pass the data to the model, then pass the model to the `JTable` object.

Example Program

DemoTableModel2.java

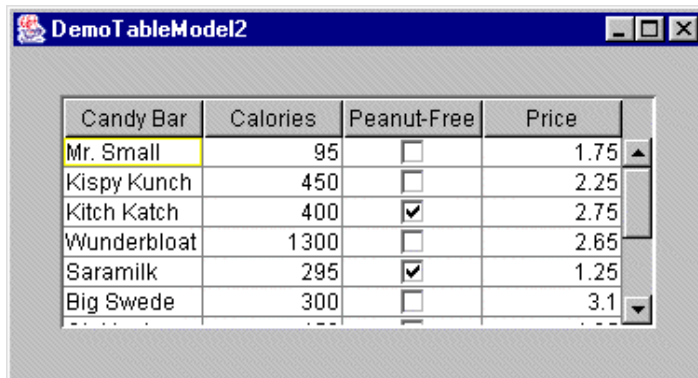


Candy Bar	Calories	Peanut-Free	Price
Mr. Small	95	<input type="checkbox"/>	1.75
Kippy Kunch	450	<input type="checkbox"/>	2.25
Kitch Katch	400	<input checked="" type="checkbox"/>	2.75
Wunderbloat	1300	<input type="checkbox"/>	2.65
Saramilk	295	<input checked="" type="checkbox"/>	1.25
Big Swede	300	<input type="checkbox"/>	3.1

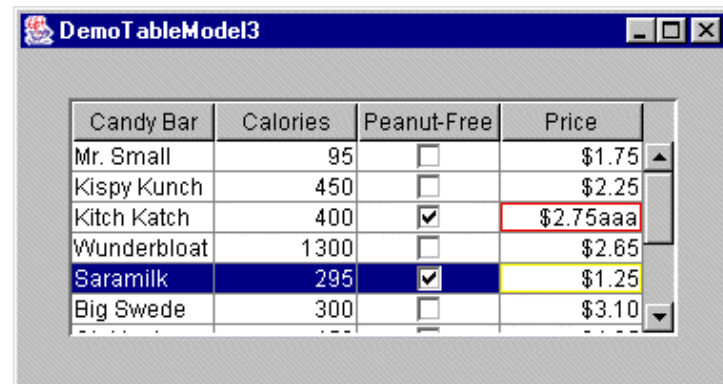
Using a custom table model
(See source code and javadoc comments for details)

Example Program

DemoTableModel3.java



Candy Bar	Calories	Peanut-Free	Price
Mr. Small	95	<input type="checkbox"/>	1.75
Kисpy Kunch	450	<input type="checkbox"/>	2.25
Kitch Katch	400	<input checked="" type="checkbox"/>	2.75
Wunderbloat	1300	<input type="checkbox"/>	2.65
Saramilk	295	<input checked="" type="checkbox"/>	1.25
Big Swede	300	<input type="checkbox"/>	3.1



Candy Bar	Calories	Peanut-Free	Price
Mr. Small	95	<input type="checkbox"/>	\$1.75
Kисpy Kunch	450	<input type="checkbox"/>	\$2.25
Kitch Katch	400	<input checked="" type="checkbox"/>	\$2.75aaa
Wunderbloat	1300	<input type="checkbox"/>	\$2.65
Saramilk	295	<input checked="" type="checkbox"/>	\$1.25
Big Swede	300	<input type="checkbox"/>	\$3.10

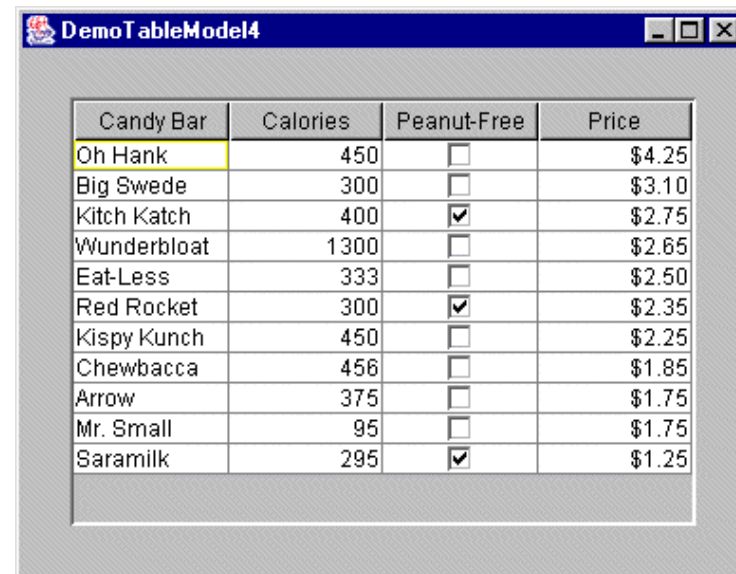
Using a custom cell renderer and custom cell editor
(See source code and javadoc comments for details)

Example Program

DemoTableModel4.java



Candy Bar	Calories	Peanut-Free	Price
Saramilk	295	<input checked="" type="checkbox"/>	\$1.25
Mr. Small	95	<input type="checkbox"/>	\$1.75
Arrow	375	<input type="checkbox"/>	\$1.75
Chewbacca	456	<input type="checkbox"/>	\$1.85
Kispy Kunch	450	<input type="checkbox"/>	\$2.25
Red Rocket	300	<input checked="" type="checkbox"/>	\$2.35
Eat-Less	333	<input type="checkbox"/>	\$2.50
Wunderbloat	1300	<input type="checkbox"/>	\$2.65
Kitch Katch	400	<input checked="" type="checkbox"/>	\$2.75
Big Swede	300	<input type="checkbox"/>	\$3.10
Oh Hank	450	<input type="checkbox"/>	\$4.25

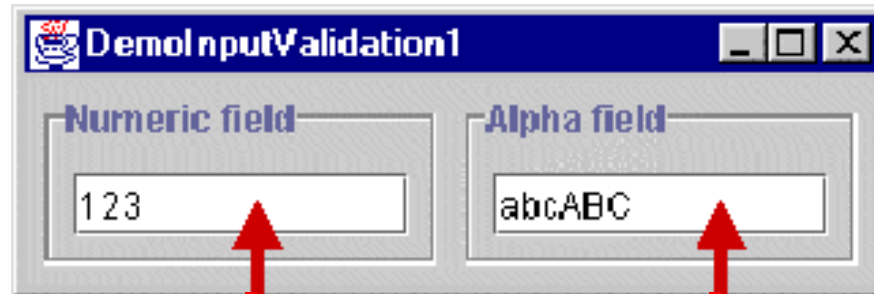


Candy Bar	Calories	Peanut-Free	Price
Oh Hank	450	<input type="checkbox"/>	\$4.25
Big Swede	300	<input type="checkbox"/>	\$3.10
Kitch Katch	400	<input checked="" type="checkbox"/>	\$2.75
Wunderbloat	1300	<input type="checkbox"/>	\$2.65
Eat-Less	333	<input type="checkbox"/>	\$2.50
Red Rocket	300	<input checked="" type="checkbox"/>	\$2.35
Kispy Kunch	450	<input type="checkbox"/>	\$2.25
Chewbacca	456	<input type="checkbox"/>	\$1.85
Arrow	375	<input type="checkbox"/>	\$1.75
Mr. Small	95	<input type="checkbox"/>	\$1.75
Saramilk	295	<input checked="" type="checkbox"/>	\$1.25

Demonstrates sorting the table by clicking on column header
(See source code and javadoc comments for details)

Example Program

DemoInputValidation3.java



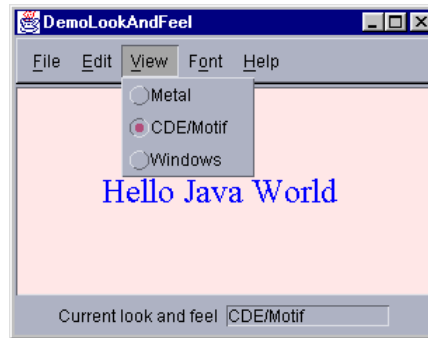
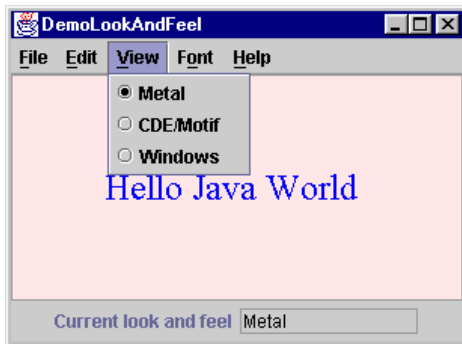
JTextField's default data model is PlainDocument. We can create a custom data model for a JTextField by creating our own data model and substituting it for PlainDocument

VC – Swing Views and Controllers

- In Swing, the term look and feel (L&F) is common
- The look is the view
- The feel is the controller
- In practice, the view and controller parts of MVC are very tightly connected
- Swing combines the view and controller into a single entity known as a UI delegate
- Advantage: combining the view and controller allows the appearance and behaviour (L&F) of a component to be treated as a single unit, thus facilitating changes to the UI are possible
- This is known as pluggable look and feel (next 3 slides)

Example Program

DemoLookAndFeel.java

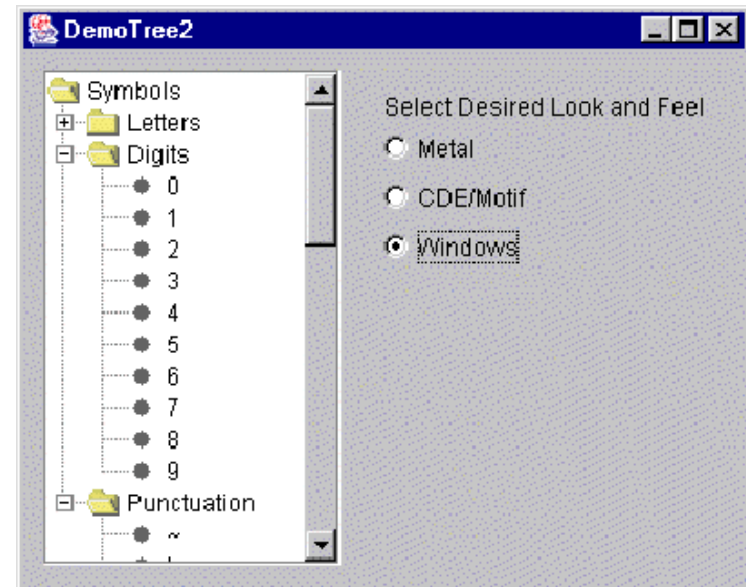
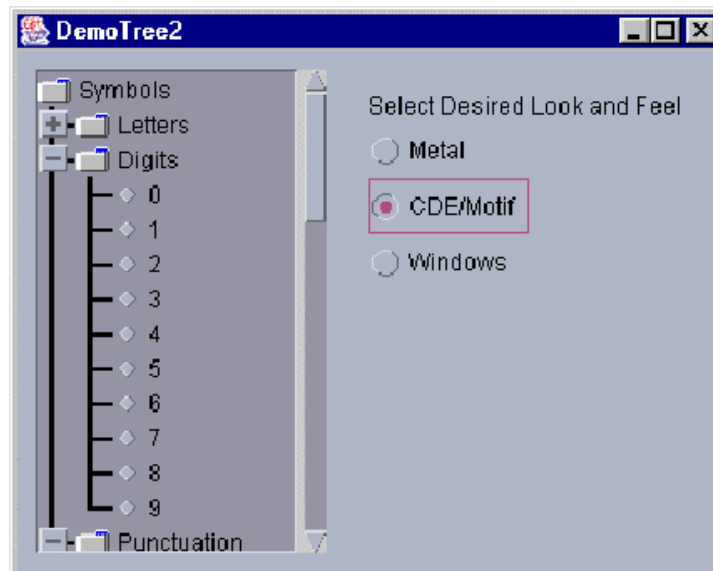
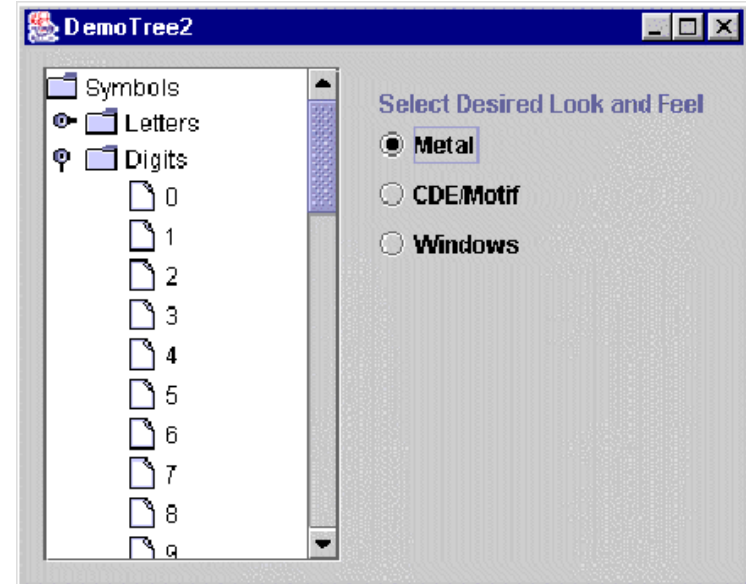


Shown earlier

Example Program

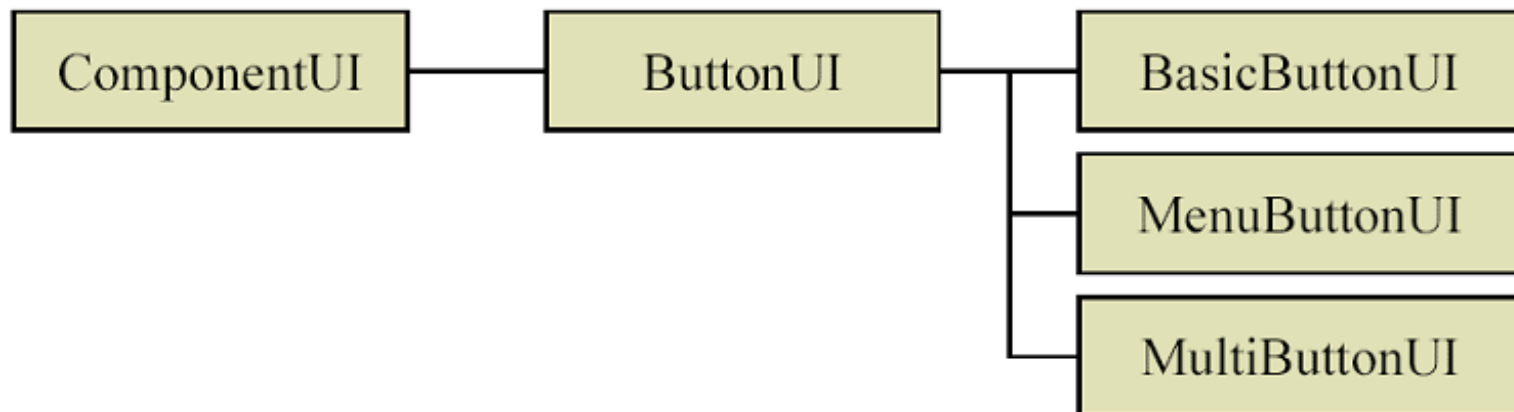
DemoTree.java

DemoTree2.java



ComponentUI Class

- The delegate part of a component is derived from an abstract class named ComponentUI
- Naming convention: remove the “J” from the component’s class name, then add “UI” to the end (e.g., JButton ButtonUI)



Design Challenge

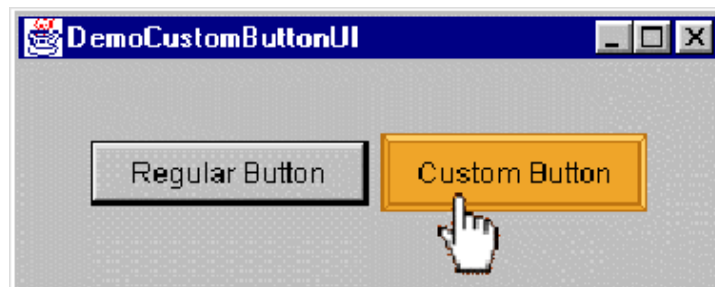
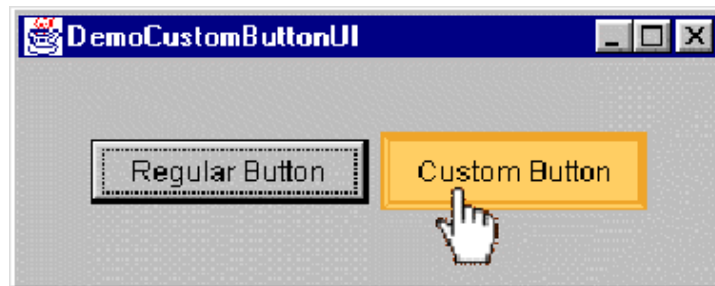
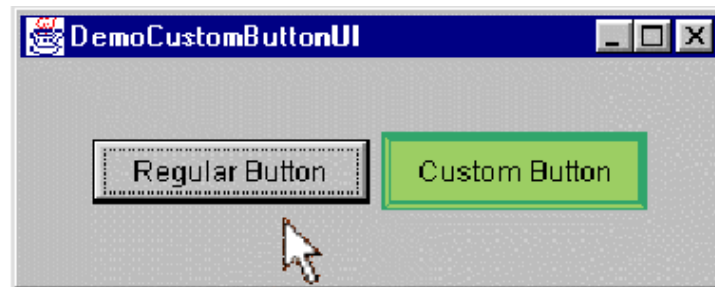
- A corporation specializing in children's games wishes to use a custom "corporate style" in all their applications
- As one example, they'd like the buttons for their applications to look as follows...



- Design a custom L&F for a JButton, as above

Example Program

DemoCustomButtonUI.java



Example Program

DemoCustomButtonUI2.java



Example Program

DemoCustomButtonUI3.java



Preferred Size Property

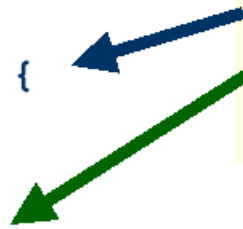
- An important task for a window manager is determining the size of widgets
- What happens when `getPreferredSize` is invoked on a `JButton`?
- `JButton`'s `getPreferredSize` method is inherited from `JComponent`
- Let's see...

getPreferredSize (from JComponent.java)

```
/**
 * If the preferredSize has been set to a non-null value
 * just returns it.  If the UI delegate's getPreferredSize()
 * method returns a non null value then return that; otherwise
 * defer to the component's layout manager.
 *
 * @return the value of the preferredSize property
 * @see #setPreferredSize
 */
public Dimension getPreferredSize()
{
    if (preferredSize != null) {
        return preferredSize;
    }
    Dimension size = null;
    if (ui != null) {
        size = ui.getPreferredSize(this);
    }
    return (size != null) ? size : super.getPreferredSize();
}
```

Returns either...

- Value set with setPreferredSize,
- Value from UI delegate, or
- Value from Container



Next Topic...