An Introduction to Model View Controllers and Swing

Reading Assignment:

Java APIs and tutorials:

http://java.sun.com

Swing

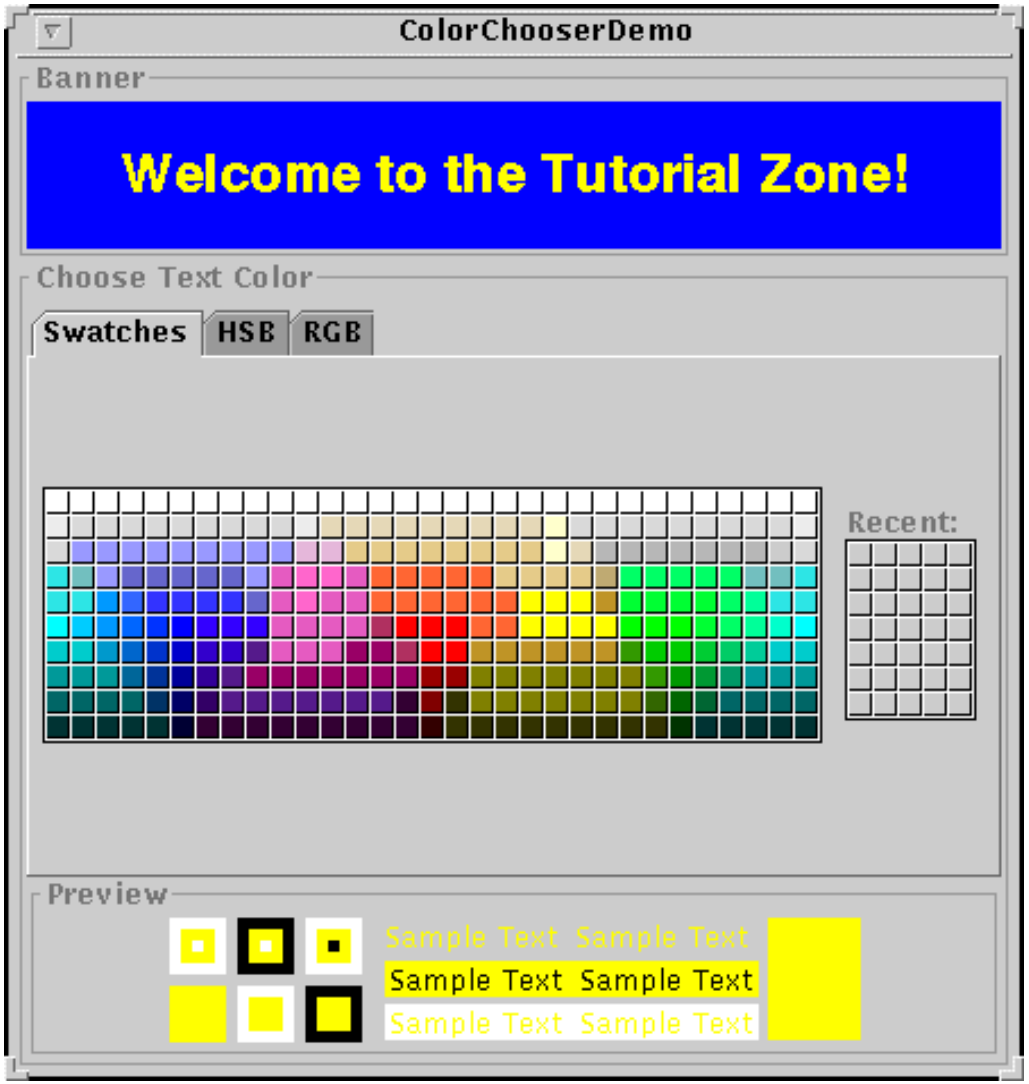
- The Java Foundation Classes is a collection of APIs for developing graphical user interfaces
- They include the following:
  - Abstract Window Toolkit (AWT)
  - 2D API
  - Swing Components
  - Accessibility API
- AWT was the original Java toolkit for developing user interfaces
  - Only provides a single pen size for graphical operations
  - Lacks many components you would expect
  - They don't scale very well…. Heavy weight components
- 2D API provides other graphical rendering capabilities
- Swing is a set of mostly lightweight components build on top of the AWT
- Swing can take on the look and feel of components on different platforms

Lightweight Components

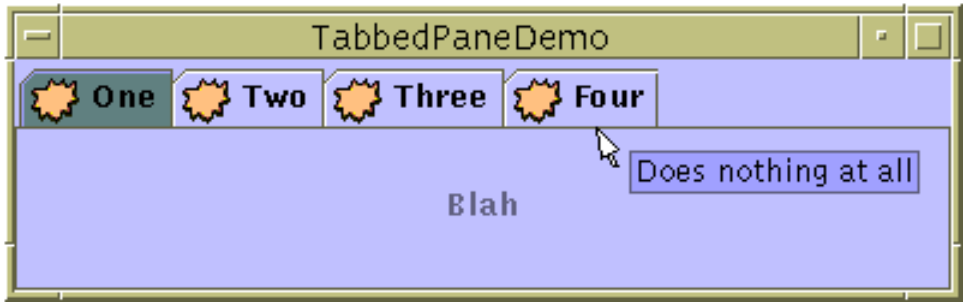
- Can be any shape and be transparent!
- They must be rendered in a heavyweight component – frames, applets, windows and dialogs
- Swing has over 250 classes – mixture of components and classes and over 40 components
- Swing components have class names that begin with the letter 'J'
  - ✍ E.g. JTextArea (hint for Ass#4!)

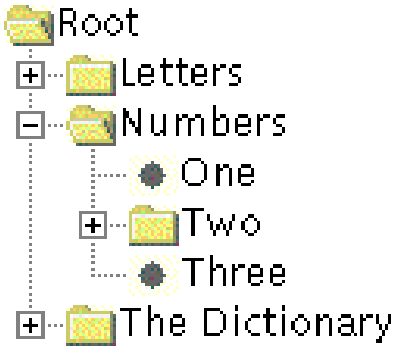
- Swing has many advanced components such as internal frames, tabbed panes, toolbars, color chooser, table, trees, dialog boxes…

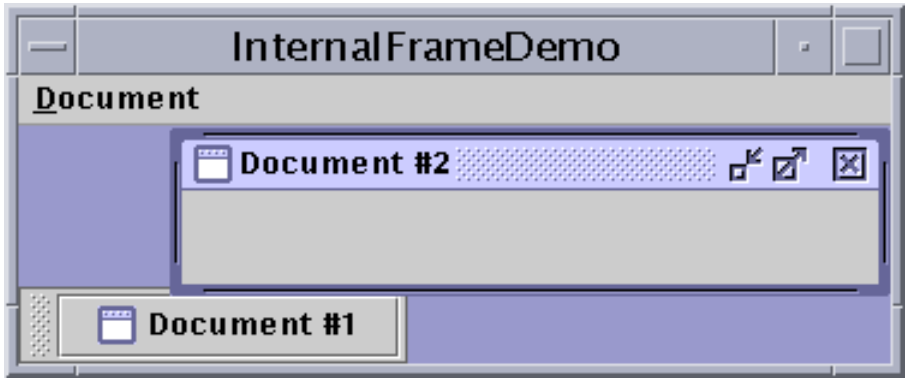
Advanced swing components



Advanced swing components







Pluggable look and feel

- Default look and feel can be set at runtime
  - Look – appearance of the components
  - Feel – interaction style of the widgets

Model View Controller Architecture

- Designed for applications that need to provide multiple views of the same data
- Models: maintain data and provide data accessor methods
- Views: paint a visual representation of some or all of a model's data
- Controllers: handle events

Controllers in Java

- AWT and Swing ***listeners*** are basically MVC controllers
- Swing events are handled by event listeners that register with an event source

Observer design pattern

- Listeners are an example of the ***Observer*** design pattern
  - where a single object can notify many observers when the observed object is modified
  - the observer knows very little about the objects, other than how and when to notify them
- The MVC architecture makes use of the observer pattern to notify views when the model changes
- Events are handled by controllers that typically make changes to the model and one or more views depending on the type of events
- A Model maintains a list of views that have registered with the model for change notification
- When a change occurs, the model updates the view but the view generally has to get extra details from the model to update itself

Advantages of MVC

- Can have multiple views and controllers plugged into a single model
- A model's views can be updated if the model changes but if the views change the model doesn't necessarily have to change
- We can also have updates to views cause updates in other views (as will be the case in Ass#4)
- Provides a way to build abstractions and encapsulate them in classes!
  - MVC encapsulates three general abstractions that are present in most graphical applications: views, models and controllers
  - Thus leading to more flexible and reusable application components

Swing Components, Models and Events

- Swing Components typically maintain a reference to their model and provide “model pass-through methods” to access the model
- For example, for Jslider.java provides a method call getMinimum() which internally calls getModel().getMinimum()
- When a property changes for a component, a property change event should be automatically fired by the model

- But who’s listening?
- When the model changes, the component will be automatically notified and will forward the event to other listeners that have registered with the component

Why is MVC so important?

- There are many applications where it is advantageous to have multiple views… for example, we may have both a graph view and a table view for some data that should be kept consistent when the data is changed

A closer look at the APIs for some Swing Components

- JComponent:
  - Ancestor class for all Swing lightweight components
  - Provides functionality that is essential and substantial
  - It can contain other AWT and Swing components

- JPanel:
  - Can be used for rendering both text and graphics
  - Can also be used as a generic container

A closer look at the APIs for some Swing Components -- 2

- JTree:
  - Displays hierarchical data using the well known paradigm of folders and leaf items
  - Trees are composed of nodes (either folders or leaves)
  - Tree nodes have a user object associated with each nodes
  - Other related classes include:
    - DefaultMutableTreeNode – a mutable node with one parent and possibly many children and a user object
    - Treepath – a path from one node to another node, often used to communicate selections
  - They can have a simple model – DefaultTreeModel (only really keeps track of the root node)

Swing examples….

- Goal: “Make simple things easy, and difficult things possible”

1) Button1.java

2) Button2.java

Events

- Clean separation of the *interface* (i.e. the graphical components) from the *implementation* (the code that you want to run when an event happens to a component)
- Each swing component can report all the events that might happen to it (e.g. mouse over, click etc)

Listeners

- We register our interest in the button press event by calling the Jbutton's addActionListener() method
- The addActionListener method expects an object of type ActionListener
- An object of type ActionListener implements the ActionListener interface which means it must provide an implementation for the actionPerformed() method
- This method will be called when the button is pressed (this type of behaviour is often called a "callback")

Anonymous inner classes

- We can have inner classes created as you need them without giving them a name
- They are used frequently in the Java tutorial examples

Button3.java

More advanced example

- Introduce: Swing trees, scroll panes, selecting nodes
- Review: 2D arrays, static variables

BTree.java

Student Exercise

- Bonus Mark!
- Pair up with a partner – read through the example handed out and document each couple of lines with what is going on – if you don't understand some of it – raise your hand
- Once you have progressed through all of the code, describe how and where you would add a button which clears the text screen when it is pushed

TreeDemo.java

Assignment #4

- Assignment 4 tips….. A look at the Dynamic Tree code examples I posted