

1st week discussion

ICS 52: Introduction to Software Engineering

Instructor: Prof. Dan Frost

TA: Derek Pfister

Oct. 1, 2008





Questions from last session

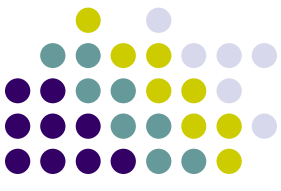
- Discussion slides are on course web
- Enhanced for loop

```
void cancelAll(Collection<TimerTask> c)
{ for (TimerTask t : c)
    t.cancel();
}
```

“for each TimerTask t in c”

- It hides iterators, so it is not usable to remove or replace elements
- <http://java.sun.com/j2se/1.5.0/docs/guide/language/foreach.html>

Algorithm- & Event-driven Application



algorithm-driven

- Application is active
- Application determines exactly what information it needs from environment, and when to get it.
- The text-based interfaces are algorithm driven.

event-driven

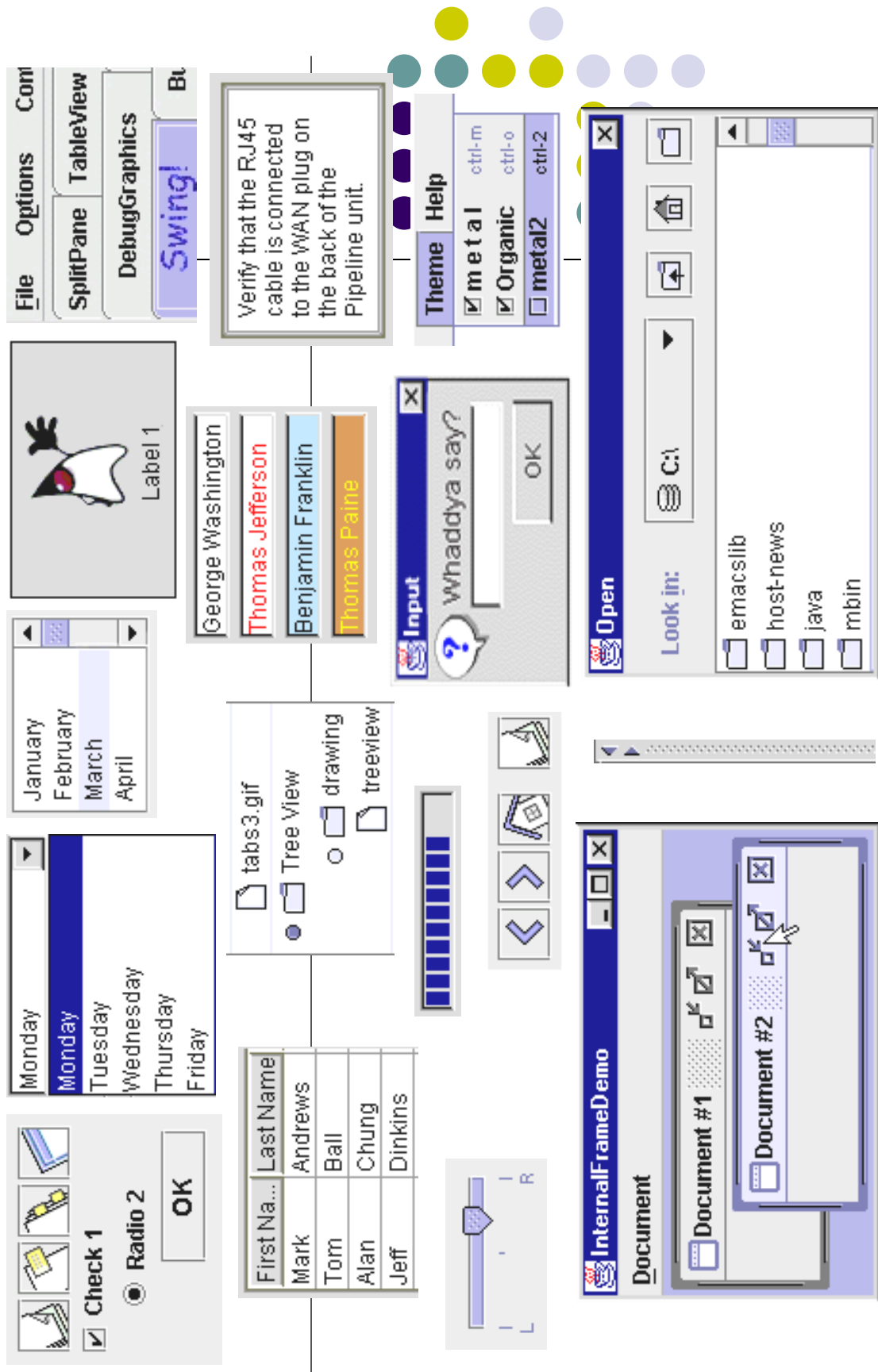
- Application is passive.
- Application waits for an event to happen in the environment
- When an event occurs, the application responds to the event, and then waits for the next event.
- Applications with a graphical, window-based user interface are almost always event driven

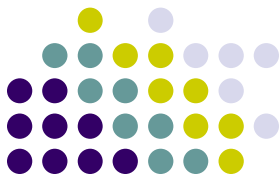
Main Steps



- To make any graphic program work we must be able to create windows and add content to them.
- To make this happen we must:
 1. Import the swing packages.
 2. Set up a top-level container.
 3. Fill the container with GUI components.
 4. Install listeners for GUI Components.
 5. Display the container.

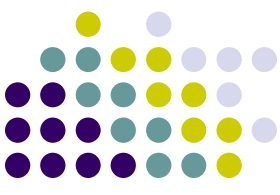
Using SWING to build user interface





Essential steps

- Import pertinent packages
 - `import javax.swing.*;`
 - `import java.awt.*;`
 - `import java.awt.event.*;`
- Set up top level container
 - This is the main window within which the GUI components will be displayed
- Set up intermediate container
 - This controls how items are laid out
- Add GUI items to the container
- Display the container
- Handle events



Containment Hierarchy

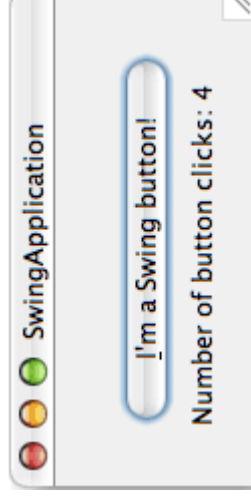
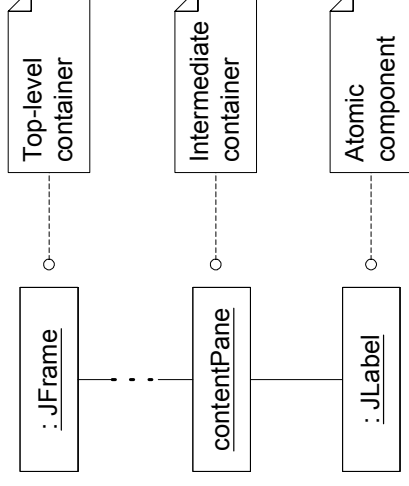
- **Top-level container**
 - contains all of the visual components of a GUI
 - provides the screen real estate used by application.
 - Define look and feel
 - Example: JFrame, JDialog

Intermediate containers

- used to organize and position GUI components
- Layout management
- Example: JPanel

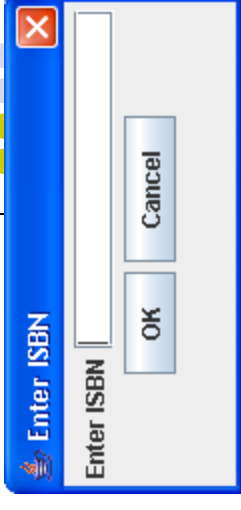
Basic components

- present information to or get information from user.
- Examples: button, label, text field, editor pane, combo box.



JDialog [EnterISBNDialog]

- Like a frame, a dialog box is a separate window.
- Unlike a frame, however, a dialog box is not completely independent.
- Every dialog box is associated with a frame, which is called its parent.

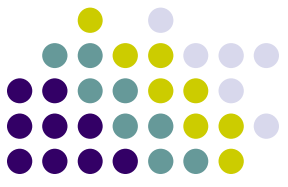


```
cancelButton.addActionListener(buttonListener); // 1
okButton.addActionListener(buttonListener);    // 2
.....
Object object = event.getSource();             // 3
if (object == okButton)                        // 4
{
.....
```

- **Create dialog boxes for the other box searches (title, author, keyword)**

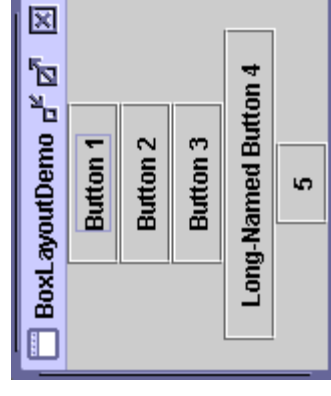
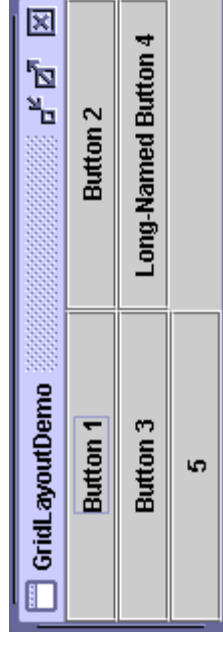
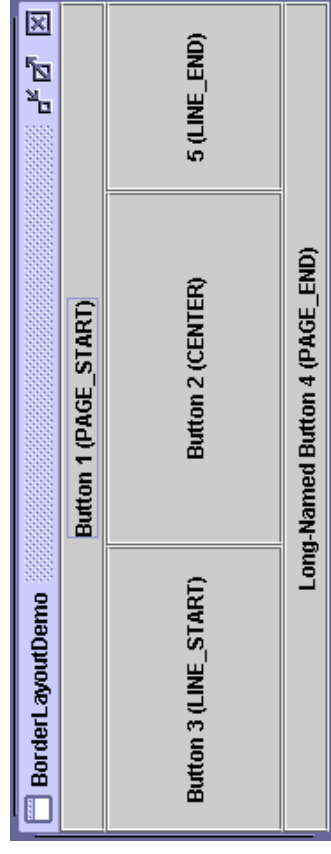
JFrame

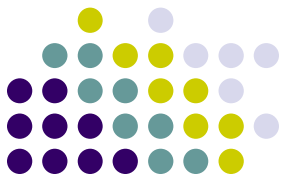
- It's a window with title, border, (optional) menu bar and user-specified components.
- It can be moved, resized, minimized.
- Handles layout management
- *The LibFrame class in the assignment extends JFrame*



Layout Management

- Responsible for positioning and sizing components added to a container.
- Each container is associated with a `LayoutManager`
 - **FlowLayout** lays out components left to right, top to bottom.
 - **BorderLayout** lays out up to five components, positioned “north,” “south,” “east,” “west,” and “center.”
 - **GridLayout** lays out components in a two-dimensional grid.
 - **BoxLayout** lays out components in either a single horizontal row or single vertical column





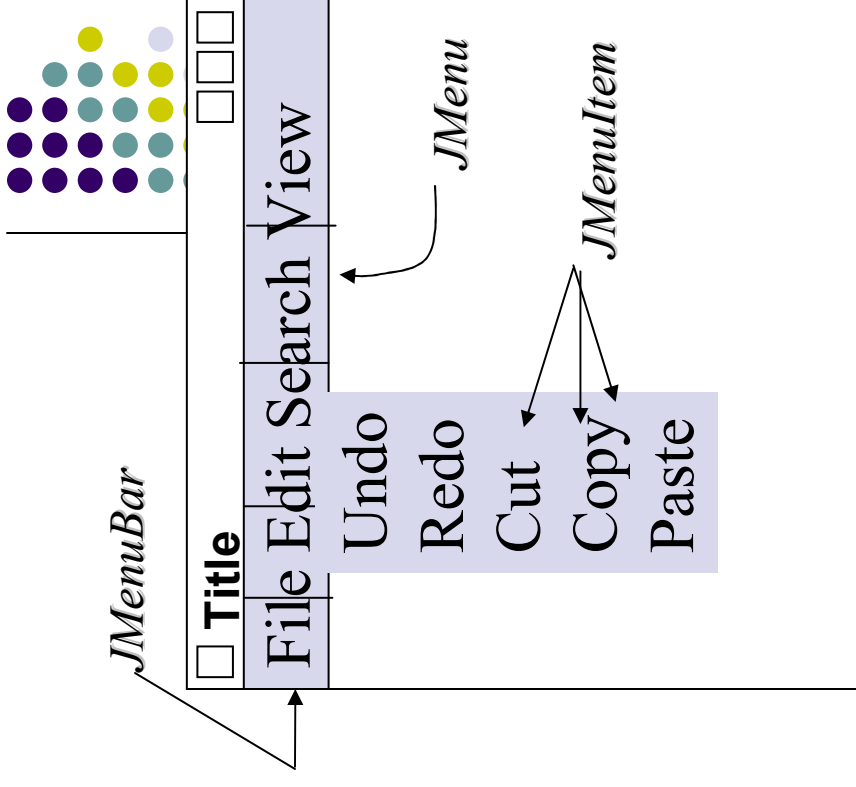
Using Layout Managers

- Set the layout manager
 - `JPanel panel = new JPanel(new BorderLayout());`
- Add components to container
 - `panel.add(aComponent, BorderLayout.PAGE_START)`
- Providing Size and Alignment Hints
- Spacing between components
- Setting container's orientation

JMenuBar

```
public class MenuFrame extends JFrame // 1
implements ActionListener // 2
{
    public MenuFrame() {
        // Set up frame itself – title,size,location

        JMenuBar menuBar = new JMenuBar( ); // 7
        setJMenuBar( menuBar ); // 8
        JMenu fileMenu = new JMenu( "File" ); // 9
        menuBar.add( fileMenu ); // 10
        JMenu editMenu = new JMenu( "Edit" ); // 11
        .....
        JMenuItem item; // 12
        item = new JMenuItem ( "Undo" ); // 13
        item.addActionListener( this ); // 14
        editMenu.add( item ); // 15
        item = new JMenuItem ( "Redo" ); // 16
        item.addActionListener( this ); // 17
```



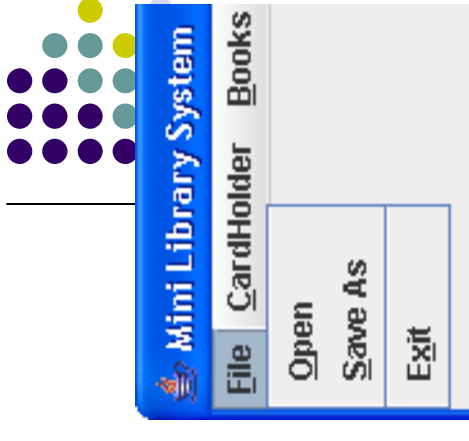
- Create a menu bar and add menus to it (lines 7, 10)
- Add the menu bar to the frame (8)
- Create menu objects and add items to them (lines 11, 15)
- Create menu items objects for your actions (lines 13, 16)

JMenuBar [LibMenuBar]

```
JMenu fileMenu, chMenu, bkMenu; // 1
JMenuItem menuItem; // 2
// Build the File menu. // 4
fileMenu = new JMenu("File"); // 5
fileMenu.setMnemonic(KeyEvent.VK_F); // 6
add(fileMenu);

// attach to it a group of JMenuItems
menuItem = new JMenuItem("Open", KeyEvent.VK_O); // 8
menuItem.addActionListener(new FileOpenListener()); // 9
fileMenu.add(menuItem); // 10
.....

class FileOpenListener implements ActionListener { // 11
    public void actionPerformed(ActionEvent e) { // 12
        parentFrame.loadData(); // 13
    } // 14
}
```



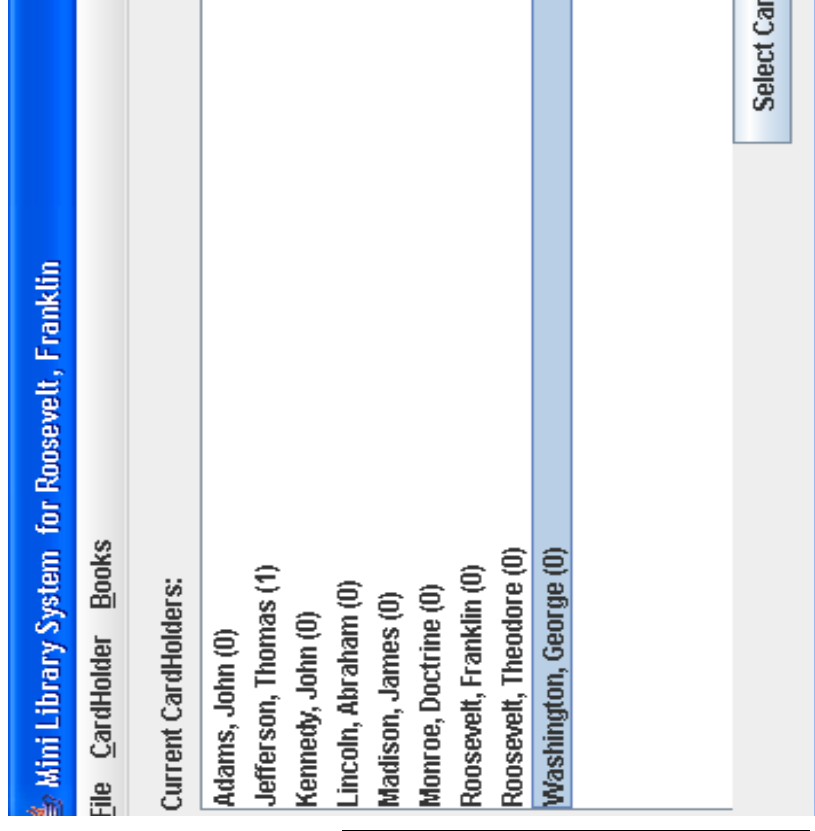
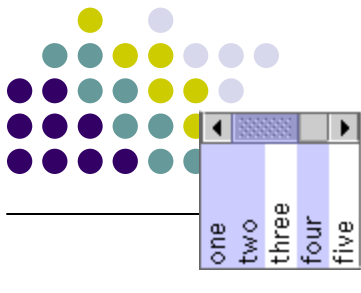
JList

- Allows users to select one or more items from a list
- Each **JList** has a

ListSelectionModel

- keeps track of selected items
- enforces selection mode (single, single interval, multiple interval)
- notifies listeners of selection changes

```
DefaultListModel chListModel = new  
    DefaultListModel(); // 1  
chList = new JList(chListModel); // 2  
chList.setSelectionMode(ListSelectionModel.SI  
    NGLE_SELECTION); // 3  
chList.setSelectedIndex(0); // 4
```



**** SelectCardHolder() in LibFrame might come in useful for other book searches.**

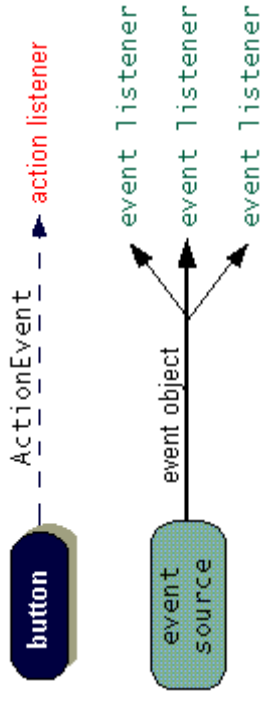
Events



● A way of informing the system that something has happened

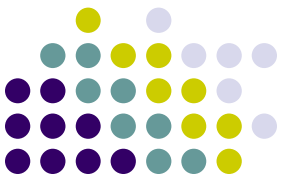
● Involves two things:

- **Events:** signals generated by
 - mouse clicks or double clicks
 - menu selections
 - text entered etc.



- **Event Listeners:** Interfaces which are designed to capture and process certain types of events;
 - register with component;
 - informed when component generates an applicable event.
 - respond by performing some appropriate action.

Implementing an Event Handler



All event handlers require three pieces of code:

1. declaration (extend/implement a listener interface)

```
public class MyClass implements ActionListener {
```

2. registration of an instance of the event handler class as a listener

```
someComponent.addActionListener(instanceOfMyClass);
```

3. providing code that implements the methods in the listener interface in the event handler class

```
public void actionPerformed(ActionEvent e) { ...//code that reacts to the action... }
```

To detect when the user clicks an onscreen button



- A program must have an object that implements the ActionListener interface
- The program must register this object as an action listener on the button (the event source), using the addActionListener method.
- When the user clicks the onscreen button, the button fires an action event.
- This results in the invocation of the action listener's actionPerformed method (the only method in the ActionListener interface)

An Example



```
public class ButtonClickExample extends JFrame // 1
    implements ActionListener // 2
{
    JButton b = new JButton("Click me!"); // 3
    public ButtonClickExample() // 4
    {
        b.addActionListener(this); // 5
        getContentPane().add(b); // 6
        pack(); // 7
        setVisible(true); // 8
    }
    public void actionPerformed(ActionEvent e) // 9
    {
        b.setBackground(Color.RED); // 10
    }
    public static void main(String[] args) // 11
    {
        new ButtonClickExample(); // 12
    }
}
```





Sources

- Swing Tutorial:
<http://java.sun.com/docs/books/tutorial/uiswing/index.htm>
- www.cs.uno.edu/~fred/nhText/ Resources/Slides/Chapter17.ppt
- Building Graphical User Interfaces with Java
www.sce.carleton.ca/courses/ sysc-2004/w04/lectures/SYSC-2004-16a-Gui.ppt
- Graphical User Interfaces with Java
www.csd.uwo.ca/courses/CS212a/notes/javagui.ppt

Questions from today

SerialVersionUID



- The serialVersionUID is a universal version identifier for a Serializable class. Deserialization uses this number to ensure that a loaded class corresponds exactly to a serialized object. If no match is found, then an [InvalidClassException](#) is thrown.
- Guidelines for serialVersionUID :
 - always include it as a field, for example: "private static final long serialVersionUID = 752647229562276147L; " include this field even in the first version of the class, as a reminder of its importance
 - do not change the value of this field in future versions, unless you are knowingly making changes to the class which will render it incompatible with old serialized objects
 - new versions of Serializable classes may or may not be able to read old serialized objects; it depends upon the nature of the change; provide a pointer to Sun's [guidelines](#) for what constitutes a compatible change, as a convenience to future maintainers