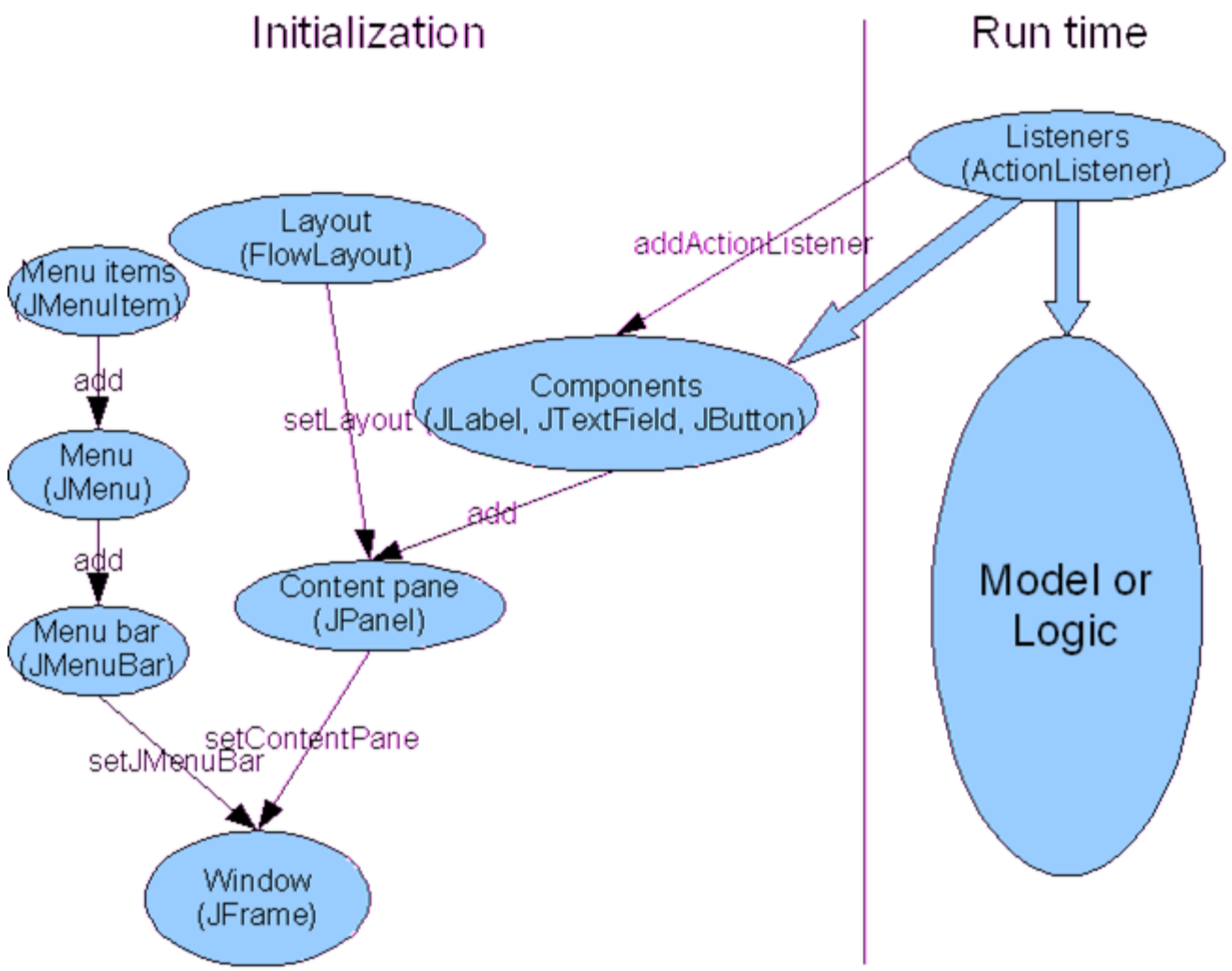
GUI Components and Events

Objectives:

- Get a more systematic introduction to basic *Swing* components, their methods, and events they generate.
- Components :
  - JLabel
  - JButton
  - JToggleButton
  - JCheckBox
  - JComboBox
  - JSlider
  - JTextField
  - JPasswordField
  - JTextArea

GUI Overall organization



GUI Organization

- 1. **Top-level Containers** are windows (JFrame, JDialog). Each window has two intermediate containers that are commonly used - a ***menu bar*** which contains the menus, and the ***content pane*** which holds the components. If there is no menu bar, the content pane expands to fill that area.
- 2. **Intermediate Containers** (eg, JPanel) contain components. Every container must have a *layout manager*.
- 3. **Layouts** (sometimes called layout managers) specify how to arrange and size components in a JPanel (or other intermediate container). Every JPanel starts with a default layout manager, but it's better to set it explicitly. Common layout managers include FlowLayout, BorderLayout, GridLayout, etc.
- 4. **Components** are user interface controls like buttons (JButton, ...), text boxes (JTextField, JTextArea, ...), labels (JLabel), etc. These are added to a container with the **add()** method.
- 5. **Listeners** are attached to components and their methods are called when the component is used (eg, a button is clicked). The listeners interact with the *model*, the basic logic of the program.
- **Other** - These are essential elements for user interfaces, but others will be useful in some programs, eg, graphics, animation, mouse, keyboard, sound, threads, look anb feel,

GUI Components

- Components are created using constructors:

```
JLabel guest = new JLabel ("Guest”);
```

- To be usable, a component must be added to the application’s “content pane” or to another component:

```
JPanel scorePanel = new JPanel();
scorePanel.add (guest);
```

GUI Events

- Components (except JLabel) can generate events.
- Events are captured and processed by “listeners” — objects equipped to handle a particular type of events.
- Different types of events are processed by different types of listeners.

Listeners

- Different types of “listeners” are described as interfaces:
  - ActionListener
  - ChangeListener
  - ItemListener
  - etc.
- The same object can serve as different listeners (as long as its class implements all the corresponding interfaces).

Listeners (cont'd)

```
public class GoHandler
        implements ActionListener
{
   ...
   public void actionPerformed (ActionEvent e)
   {
      ...
   }
}
```

Objects of this class are **GoHandler**s but also **ActionListener**s

This method is called automatically when the button is clicked

```
   ...
   JButton go = new JButton ("Go");
   go.addActionListener (new GoHandler ());
```

This method expects an **ActionListener**; a **GoHandler** object qualifies.

Listeners (cont'd)

- When implementing an event listener, programmers can use a private *inner* class that has access to all the fields of the surrounding public class.
- An inner class can have constructors and private fields, like any other class.
- A private inner class is accessible only in its outer class.

Listeners (cont'd)

```
public class MyPanel extends JPanel
{
   private JButton go;
   ...
      go = new JButton ("Go");
      go.addActionListener (new GoHandler ());
      ...
   private class GoHandler implements ActionListener
   {
       public void actionPerformed (ActionEvent e)
       {
           go.setText("Stop");
           ...
       }
   }
}
```

go is accessible in constructors and methods of the inner class

Listeners (cont'd)

- You don't have to capture all events.
- If you don't want to deal with events from a component, just don't attach a listener to it.
- If you do want to capture events but forget to add a listener, no events will be captured (a common omission).

Action event

- public ActionEvent(Object source, int id, String command, int modifiers)
- Constructs an ActionEvent object. This method throws an IllegalArgumentException if source is null.
- A null command string is legal, but not recommended.

  Parameters:
  source - The object that originated the event
  id - An integer that identifies the event. For information on allowable values, see the class description for ActionEvent
  command - A string that may specify a command (possibly one of several) associated with the event
  modifiers - The modifier keys down during event (shift, ctrl, alt).
- Passing negative parameter is not recommended. Zero value means that no modifiers were passed

Listeners (cont'd)

- actionPerformed (or another listener method) takes a corresponding type of event as a parameter.
- Event objects have useful methods. For example, getSource returns the object that produced this event.
- A MouseEvent has methods getX, getY.

JLabel

Constructors:

JLabel (String text);
JLabel (ImageIcon icon);
JLabel (String text, ImageIcon icon, SwingConstants.LEFT);
// or CENTER, RIGHT, LEADING, TRAILING.

Methods:

void setText (String text);
void setIcon (ImageIcon icon);

Events: None

JButton

Constructors:

JButton (String text);
JButton (ImageIcon picture);
JButton (String text, ImageIcon picture);

Methods:

void addActionListener (ActionListener object)
void setText (String text);
void setActionCommand (String cmd);
void setIcon (ImageIcon icon);
void requestFocus();

Events:

```
class ... implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    JButton b = (JButton)e.getSource();
    String s = e.getActionCommand();
  }
}
```

JCheckBox

Constructors:

```
JCheckBox (String text, boolean checked);
JCheckBox (ImageIcon icon, boolean checked);
JCheckBox (String text, ImageIcon icon,
                                boolean checked);
```

Methods:

```
void addActionListener (ActionListener object)
boolean isSelected ()
void setSelected (boolean checked)
void setText (String text);
void setIcon (ImageIcon icon);
```

Events:

```
class ... implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {
    JCheckBox b = (JCheckBox)e.getSource();
    if (b == checkBox1 && b.isSelected())
      ...
  }
}
```

Layouts

- A layout manager is a strategy for placing components on the content pane or another component (usually a panel).
- In Java, the content pane and any GUI component is a Container.
- A layout is chosen by calling the container's setLayout method.

Layouts (cont'd)

- Layouts are used to achieve some degree of platform independence and scalability.
- awt/*Swing* support several layout managers. Here we consider four:
  - FlowLayout
  - GridLayout
  - BorderLayout
  - BoxLayout
- Each of these classes implements the java.awt.LayoutManager interface.

FlowLayout

- Places components in a line as long as they fit, then starts the next line.
- Uses “best judgement” in spacing components. Centers by default.
- Lets each component assume its natural (preferred) size.
- Often used for placing buttons on panels.

FlowLayout (cont'd)

```
Container c = getContentPane();
c.setLayout(new FlowLayout());
c.add (new JButton ("Back to Start"));
c.add (new JButton ("Previous Slide"));
c.add (new JButton ("Next Slide"));
c.add (new JButton ("Last Slide"));
c.add (new JButton ("Exit"));
```

GridLayout

- Splits the panel into a grid with given numbers of rows and columns.
- Places components into the grid cells.
- Forces the size of each component to occupy the whole cell.
- Allows additional spacing between cells.

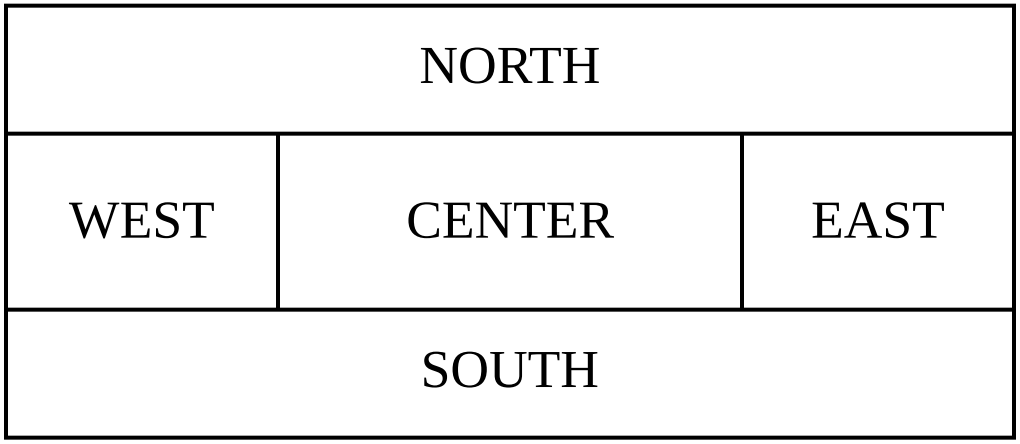
GridLayout (cont'd)

```
Container c = getContentPane();
c.setLayout (new GridLayout(3, 2,  10, 20 ));
c.add (new JButton ("Back to Start"));
c.add (new JButton ("Previous Slide"));
c.add (new JButton ("Next Slide"));
c.add (new JButton ("Last Slide"));
c.add (new JButton ("Exit"));
```

Extra space between the cells (in pixels)

BorderLayout

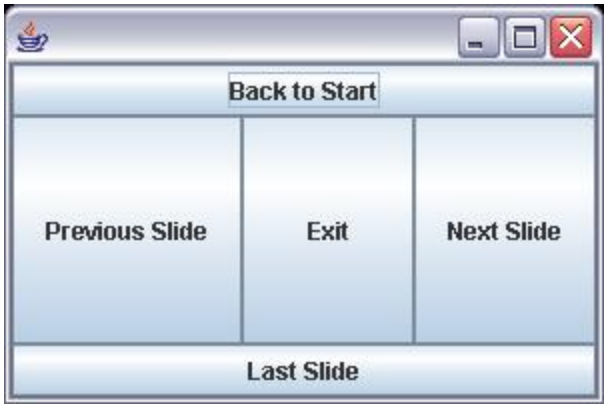
- Divides the area into five regions and adds a component to the specified region.



- Forces the size of each component to occupy the whole region.

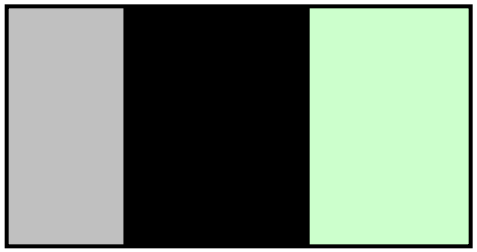
BorderLayout (cont'd)

```
Container c = getContentPane();
c.setLayout(new BorderLayout());  // optional: default
c.add (new JButton ("Next Slide"), BorderLayout.EAST);
c.add (new JButton ("Previous Slide"), BorderLayout.WEST);
c.add (new JButton ("Back to Start"), BorderLayout.NORTH);
c.add (new JButton ("Last Slide"), BorderLayout.SOUTH);
c.add (new JButton ("Exit"), BorderLayout.CENTER);
```



BoxLayout

- In a horizontal box, components are placed horizontally, left to right.



- In a vertical box, components are placed vertically, top to bottom.

"Horizontal" or "vertical" has nothing to do with the shape of the box itself.

BoxLayout (cont'd)

- BoxLayout is the default layout for a Box container.
- The idiom for working with boxes is slightly different:

```
Box box1 = Box.createHorizontalBox();
box1.add (...);

 // add a spacer, 60 pixels:
 box1.add(Box.createHorizontalStrut (60));

 Box box2 = Box.createVerticalBox();
 ...
```

BoxLayout (cont'd)

```
Container c = getContentPane();
c.setLayout(new FlowLayout());
Box box = Box.createVerticalBox();
box.add (new JButton ("Next Slide"));
box.add (new JButton ("Previous Slide"));
box.add (Box.createVerticalStrut (20) );
box.add (new JButton ("Exit"));
c.add (box);
```

Adds extra vertical space between components

Default Layouts

- Each component has a default layout manager, which remains in effect until the component's setLayout method is called.
- The defaults are:

Content pane	↔ BorderLayout
JPanel	↔ FlowLayout
Box	↔ BoxLayout

Menus

- You can add a JMenuBar object to JFrame or JApplet.
- You can add JMenu objects to a JMenuBar.
- You can add other JMenus, JMenuItems, JCheckBoxMenuItems, JRadioButtonMenuItems, etc. to a JMenu.

Review:

- Can a container contain another container?
- Name several *Swing* GUI components.
- Is an action listener a class, an interface, an object, or a method?
- How do FlowLayout and GridLayout deal with the sizes of components?

Review (cont'd):

- What is the default layout manager for the content pane?
- What type of objects can you add to a JMenu?