Swing, Part 2

Christopher Simpkins
chris.simpkins@gatech.edu

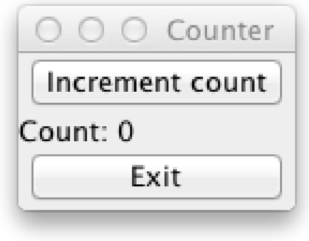
Outline

- Components and Containers
- Layout Managers
- Listening to multiple components
- Menus
- Actions

Components and Containers

- A component is an object with a visual representation in the GUI.
- A container is a component that contains other componets.
- Most GUI aplications have multiple components, and thus need containers to manage them.

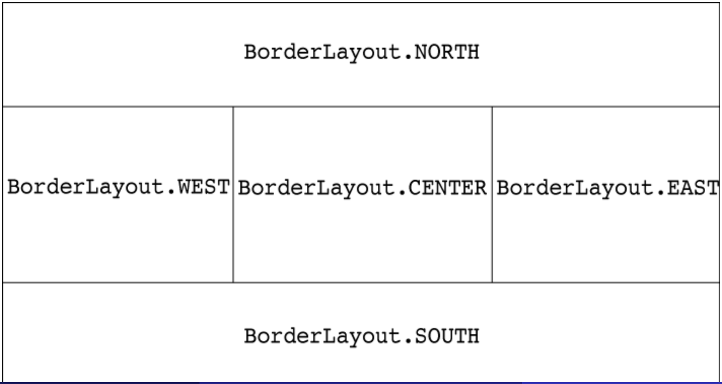
For example, this CounterFrame has four `java.awt.Component`s:



- a `javax.swing.JButton` labeled "Increment count",
- a `javax.swing.JLabel` that displays the count,
- a `javax.swing.JButton` labeled "Exit", and
- a `javax.swing.JFrame` that contains the other three components.

Layout Managers

- Layout managers control the position of components in a container.
- `CounterFrame` uses a `java.awt.BorderLayout`.
- `java.awt.BorderLayout` places components in one of 5 regions:



Using a BorderLayout

To use a BorderLayout, import `java.awt.BorderLayout`:

```
import java.awt.BorderLayout;
```

Call the `setLayout` method in `java.awt.Container` (here `CounterFrame` is a subclass of `JFrame`, which is a subclass of `java.awt.Container`)[1]

```
setLayout(new BorderLayout());
```

See CounterFrame.java for the full code referenced above.

[1] Technically, `JFrame` contains a `JRootPane` container for all non-menu components, which is returned by `JFrame`'s `getContentPane` method. For convenience, `JFrame`'s `setLayout`, `add`, and `remove` methods forward to the content pane.

Adding Components Using a Layout Manager

When you add a component to a container, you often pass layout manager-specific arguments to the `add` method to specify where the component should go. For example, using a `BorderLayout` in `CounterFrame`'s constructor:

```
add(counterButton, BorderLayout.NORTH);
add(counterLabel, BorderLayout.CENTER);
add(exitButton, BorderLayout.SOUTH);
```

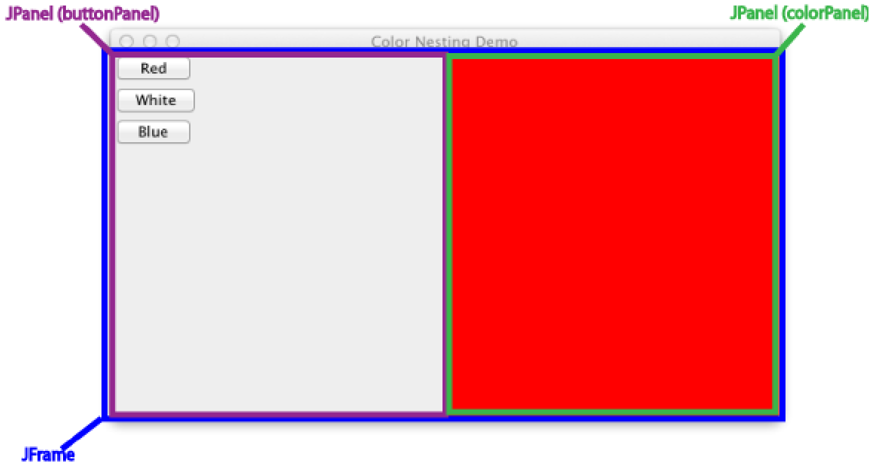
To use a `GridLayout`, simply add components in row-major order:

```
setLayout(new GridLayout(3,1)); // 3 rows, 1 column
add(counterButton);
add(counterLabel);
add(exitButton);
```

Becoming proficient at laying out GUI components requires practice. Have a look at Oracle's tutorial and experiment with the sample code we provide.

Nesting Containers

Containers can be nested for more complex layouts:



Creating Nested Containers

Create components from inside out: first UI widgets, then their inner containers, then outer container(s):

```
// Set up button panel
JButton redButton = new JButton("Red");
redButton.addActionListener(this);
// ...
JPanel buttonPanel = new JPanel();
buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.Y_AXIS));
buttonPanel.add(redButton);
// ...
// Set up color panel
colorPanel = new JPanel();
colorPanel.setSize(200, 200);
// Set up main application frame
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLayout(new GridLayout(1, 2)); // 1 row, 2 columns
add(buttonPanel);
add(colorPanel);
```

See ColorBox.java for full code.

Listening to Multiple Components

Notice that ColorBox.java implements `ActionListener`:

```
public class ColorBox extends JFrame implements ActionListener { ...
```

We register the `ColorBox` instance as a listener to its own components using `this`:

```
redButton.addActionListener(this);
```

And implement the `actionPerformed` method:

```
public void actionPerformed(ActionEvent e) {
    String button = e.getActionCommand();
    if (button == "Red") {
        colorPanel.setBackground(Color.RED);
    } else if ...
```

- Each component has an `actionCommand` that's passsed into its constructor or set with `setActionCommand(String s)`.
- You can use the `actionCommand` to identify which component fired an event.

Adding Menus

- Create JMenuItems, add listeners to them.
- Create JMenus, add JMenuItems to them.
- Create a JMenuBar, add JMenus to it.
- Set the JMenuBar as the frame's menu

The entire process for a simple 3-item color menu is:

```
JMenuItem redMenuItem = new JMenuItem("Red");
redMenuItem.addActionListener(this);
/ ...
JMenu colorMenu = new JMenu("Color");
colorMenu.add(redMenuItem);
// ...
JMenuBar menuBar = new JMenuBar();
menuBar.add(colorMenu);
setJMenuBar(menuBar);
```

- Notice that you add action listeners to JMenuItems the same way you add them to JButtons.
- javax.swing.JMenuItem and javax.swing.JButton are both subclasses of javax.swing.AbstractButton.

ColorBox isn't DRY

Did you notice that we had to make sure that each button and its corresponding menu items had the same text and actionCommand?

```
  JButton redButton = new JButton("Red");
  redButton.addActionListener(this);
```

```
  JMenuItem redMenuItem = new JMenuItem("Red");
  redMenuItem.addActionListener(this);
```

Also, if we wanted to disable the "make the color box red" command, we'd have to hold references to both the button and menu item and remember to disable them both:

```
redButton.setEnabled(false);
redMenuItem.setEnabled(false);
```

There's a better way ...

The `Action` Interface

An `Action` is an object that listens to events, defines label text, and so on. using `Action`s let's you define the behavior of an action in one place and hook that action up to several components. Consider:

```
redAction = new AbstractAction("Red") {
  public void actionPerformed(ActionEvent e) {
    colorPanel.setBackground(Color.RED);
  }
};
```

`AbstractAction` is a class that makes it easier to define actions. Once this action is defined we can use it to create the button:

```
JButton redButton = new JButton(redAction);
```

... and the menu item:

```
JMenuItem redMenuItem = new JMenuItem(redAction);
```

One place to define the behavior the `redAction`, and one place to enable/disable all components that activate the `redAction`
See ColorBox2.java for an example of using actions.

Closing Thoughts

- GUI programming requires two things:
  - Knowledge of GUIs (widgets, how they work, how they're used)
  - Knowledge of a particular GUI framework (like Swing)
- The Swing classes you've seen make extensive use of OOP (like `JMenuItem` and `JButton` being subclasses of `AbstractButton`.
- GUI programs are straightforward, but get complex quickly.
- In the next few lectures, we'll begin to learn how to deal with the complexity of GUI programs with the Model-View-Controller pattern.