GUI Programming

CMSC 202
Why Java GUI Development?

- After all CMSC 202 is about Object Oriented Programming, not GUls
- GUls are a good example of OOP
- GUls are another example of containers
Java and GUIs

• There are two primary built-in packages that provide GUI components in Java
  – java.awt.*
  – java.swing.*

• The Abstract Window Toolkit (AWT)
  – Java's original GUI toolkit
  – Leverages native toolkits to draw widgets

• Swing
  – Offers a more complete set of widgets
  – System or Java look and feel
  – Leverages AWT throughout APIs
Containers

• In Java, all GUI components go into a Container - which is simply a widget that can contain other widgets
• A top-level container can stand alone in a window environment
  – e.g. JFrame
• Some containers may only be added to other containers
  – e.g. JPanel
Components

• A component is simply an object that has a graphical representation that can be displayed on screen
• A component acts as a base class for all swing components, except top level containers
• Examples of components include:
  – JButton, JComboBox, JLabel, JList, JMenuBar, JPanel, JSlider, JSpinner, JTable, etc...
JFrame

• A JFrame is often the highest-level widget in your application in which all other widgets will get packed

• JFrames are usually constructed using the following constructor:

```java
public JFrame(String title);
```
Common JFrame Methods

• add(Component c)
  – adds objects to the frame
• setVisible(boolean b)
  – makes the frame visible
• setLocation(int x, int y)
  – aligns top left corner of frame with coordinates on screen
• setSize(int width, int height)
  – sets size of frame in pixels
• setDefaultCloseOperation(int operation)
  – defines what should happen when the window is closed, usually call with the constant WindowConstants.EXIT_ON_CLOSE
import javax.swing.JFrame;
import javax.swing.WindowConstants;

public class UpperCaseConverter extends JFrame {

    public UpperCaseConverter() {
        super("Upper Case Converter");
        setLocation(100, 100);
        setSize(480, 320);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
        UpperCaseConverter ucc = new UpperCaseConverter();
        ucc.setVisible(true);
    }
}

JFrame Example
JFrame Example
Layout Managers

• Every container has an underlying default LayoutManager

• The LayoutManager determines:
  – The size of the objects in the container and
  – How the objects will be laid out

• The default LayoutManager for a JFrame is a BorderLayout
BorderLayout

- The BorderLayout manager divides container into five regions
  - BorderLayout.NORTH
  - BorderLayout.SOUTH
  - BorderLayout.CENTER
  - BorderLayout.EAST
  - BorderLayout.WEST
- One component per region
- Component takes size of region
- Center region is greedy
- Components are added to center by default
A JButton provides a basic button that a user can interact with.

A JButton may consist of a combination of label and/or icon and is typically constructed using one of the following constructors:

- JButton(Icon icon)
  - Creates a button with an icon
- JButton(String text)
  - Creates a button with text
- JButton(String text, Icon icon)
  - Creates a button with initial text and an icon
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;

public class BorderLayoutExample extends JFrame {

    public BorderLayoutExample(String name) {
        super(name);
        setSize(300, 300);
        add(new JButton("North"), BorderLayout.NORTH);
        add(new JButton("South"), BorderLayout.SOUTH);
        add(new JButton("East"), BorderLayout.EAST);
        add(new JButton("West"), BorderLayout.WEST);
        add(new JButton("Center"), BorderLayout.CENTER);
    }

    public static void main(String args[]) {
        BorderLayoutExample b = new BorderLayoutExample("BorderLayoutExample");
        b.setVisible(true);
    }
}
BorderLayoutExample
JPanel

• Say we want to put several buttons in the North region of the GUI, but BorderLayout only allows one component per region...
• Add a second level container like a JPanel
• JPanes have a FlowLayout manager by default
FlowLayout

- Lays components in a fluid direction as determined by its orientation
- By default, orientation is L → R, T → B
- Possible to set the horizontal and vertical width between components
- Components take preferred size
  - For buttons, preferred size is the size of the text within them
public UpperCaseConverter() {

    // code from previous slides ...

    JPanel topPanel = new JPanel();
    JButton upperButton = new JButton("To Upper");
    JButton clearButton = new JButton("Clear");
    topPanel.add(upperButton);
    topPanel.add(clearButton);
    add(topPanel, BorderLayout.NORTH);
}

// ...
FlowLayout
The **JLabel** represents a static label

Like buttons, they can consist of text and/or images and are usually constructed using one of the following constructors...

- `JLabel()`
  - Creates a JLabel instance with no image and with an empty string for the title.
- `JLabel(Icon image)`
  - Creates a JLabel instance with the specified image.
- `JLabel(String text)`
  - Creates a JLabel instance with the specified text.
- `JLabel(String text, Icon icon, int horizontalAlignment)`
  - Creates a JLabel instance with the specified text, image, and horizontal alignment
**JTextField**

- A [JTextField](#) provides an entry for a single line of text
- A JTextField may be constructed with a set width or with default text and is usually constructed using one of the following constructors:
  - JTextField()
    - Constructs a new TextField
  - JTextField(int columns)
    - Constructs a new empty TextField with the specified number of columns
  - JTextField(String text)
    - Constructs a new TextField initialized with the specified text
  - JTextField(String text, int columns)
    - Constructs a new TextField initialized with the specified text and columns
Overriding a Panel’s Layout Manager

• You can also over-ride the layout manager for most containers
• For example, we can change a JPanel’s layout from a flow layout to a border layout if that’s more appropriate for what we’re laying out
Overriding a Panel’s Layout Manager

// ...

public UpperCaseConverter() {

    // code from previous slides ...

    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new BorderLayout());
    JLabel enterTextLabel = new JLabel("Enter text: ");
    JTextField textField = new JTextField(20);
    bottomPanel.add(enterTextLabel, BorderLayout.WEST);
    bottomPanel.add(textField, BorderLayout.CENTER);
    add(bottomPanel, BorderLayout.SOUTH);
}

// ...
Overriding a Panel’s Layout Manager
JTextArea

• A JTextArea is similar to a JTextField, except that it is capable of displaying multiple lines of text
• A JTextArea can be constructed with a given size and/or default text and is typically constructed using one of the following constructors:
  – JTextArea()
    • Constructs a new TextArea.
  – JTextArea(int rows, int columns)
    • Constructs a new empty TextArea with the specified number of rows and columns.
  – JTextArea(String text)
    • Constructs a new TextArea with the specified text displayed.
  – JTextArea(String text, int rows, int columns)
    • Constructs a new TextArea with the specified text and number of rows and columns.
public UpperCaseConverter() {

    // code from previous slides ...

    JTextArea textArea = new JTextArea();
    textArea.setEditable(false);
    add(textArea, BorderLayout.CENTER);
}

// ...
JTextArea
Responding to Actions

• Currently our button doesn't do anything when pressed, to respond to this action we need to add an `ActionListener`

• An ActionListener can be added to a button using the following method:
  – `public void addActionListener(ActionListener l);`

• The ActionListener interface is quite simple, in that it only requires one to implement a single method:
  – `void actionPerformed(ActionEvent e)`
Letting our class implement ActionListener

- One approach to implementing an ActionListener is to have “this” class implement it

  button.addActionListener(this);
Implementing ActionListener as an Anonymous Class

• Another approach is to actually define an inline anonymous class to handle the actions
• The class is considered inline as it is declared in the context of another class
• It is also considered anonymous, as the new class is not given a name
public UpperCaseConverter() {

    // code from previous slides ...
    upperButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("button pressed");
        }
    });

    clearButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            System.out.println("button pressed");
        }
    });

    // ...
}
Manipulating Widgets

• Rather than simply printing “button pressed” let’s modify our ActionListener to read the value of the JTextField and write to the JTextArea

• As such we're going to need to references to those widgets — we have 2 options
  – Store the widgets as members of the class
  – Mark the widget references as final
public UpperCaseConverter() {

    // code from previous slides...
    // with upperButton and clearButton declared final

    upperButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            textArea.setText(textField.getText().toUpperCase());
        }
    });

    clearButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            textArea.setText("\n");
            textField.setText("\n");
            textField.requestFocus();
        }
    });

    // ...
}
More Interesting Action Listeners
Look and Feel

• By default, Java uses its own Look and Feel
• If you’d like to use the native look and feel for your OS, simply perform the following before displaying any windows:

```java
try {
    UIManager.setLookAndFeel(
        UIManager.getSystemLookAndFeelClassName()
    );
} catch (Exception e) {
    // handle or ignore
}
```
Scratching the Surface

• What we've looked at here is really just the tip of the iceberg, there a lot to swing

• Some selected references...
  – A Visual Guide to Layout Managers
  – Swing Features
  – A Visual Guide to Swing Components
  – Trail: Creating a GUI With JFC/Swing