Graphics

Using what we have learned – inheritance, interfaces, polymorphism.
GUI Definitions

- **Graphical User Interface**
  - A “window” – a smaller screen within your monitor.
    - Usually contains text, buttons, menus, etc.
    - Has a border, close-window “X” button, inside.

- **How’s the window know which button you push?**
  - Each click on a window is an “event”.
  - Window has a “listener”.
    - Listens for events.
      - Just sits there constantly checking for events.
Most of the Java GUI is in package called Swing.

- import javax.swing. *;

- The classes have names like
  - JFrame
  - JLabel
  - JButton
  - JMenu
  - etc.

- The J in front distinguishes from an older “AWT” graphics package.
  - We still use many AWT classes that don’t have J.
  - e.g., WindowAdapter
  - import java.awt.*;
Parts of the Java GUI: JFrame

- **JFrame class**
  - This is the window.
  - **JFrame myWindow = new JFrame();**
  - Can set the size in pixels.
    - **myWindow.setSize(width, height);**
  - Must make window visible.
    - **myWindows.setVisible(true);**
    - Default is invisible.
    - Handy if you want to hide windows temporarily (set to false).
Parts of the Java GUI: Content Pane

- **Content Pane (or Container)**
  - The inner part of the window (not the border).
  - This is where you put the text, buttons, menus etc.
  - `Container myContentPane = myWindow.getContentPane();`
  - Can add any GUI Component.
    - `myContentPane.add(…);`
  - Must import the Container class
    - `import java.awt.*;`
Parts of the Java GUI: JComponent

- JComponent
  - All labels, buttons, menus, scroll bars, etc.
  - JLabel, JButton, JMenu, JScrollBar, etc. inherit from this.

- e.g.,
  - JLabel myLabel = new JLabel("Yo, how’s life?");
    myContentPane.add(myLabel);
    • Now this label is displayed in the window.

  - JButton myButton = new JButton("Push me");
    myContentPane.add(myButton);
    • Now this button is displayed in the window.

- Because of class casting the “add” method takes any JComponent.
Parts of the Java GUI: WindowAdapter

- **WindowAdapter**
  - The JFrame has a close-window box.
    - How’s it know when it was clicked?
  - WindowAdapter listens for window events.

- **WindowAdapter is abstract.**
  - How does it know what you would like to happen when the close-window box is clicked?
    - Maybe you want a confirmation dialog to pop up.
    - Maybe you want to end the program… System.exit(0);
  - You have to “implement” the methods.
Parts of the Java GUI: WindowAdapter Inheritance Picture

Object

WindowAdapter

WindowListener

WindowState Listener

WindowFocus Listener
Parts of the Java GUI: Creating WindowAdapter

- Has 10 methods from the interfaces.
  - windowClosing
    - activated (automatically called by GUI) when the window is closed.
  - windowIconified
    - activated when the window is iconified.
  - windowDeiconified
    - activated when the window is de-iconified.
  - Etc.

- All the interface methods have been implemented!
  - They’re just empty and do nothing.
  - No abstract methods, even though an abstract class.
  - They did this for your convenience, so you wouldn’t have to implement every single method. WindowAdapter is called a “convenience class”.

- So override any methods that you want to do something.
import java.awt.event.*;

public class WindowEventListener extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
After defining our listener, can use it with JFrame.

```java
JFrame myWindow = new JFrame();
WindowEventListener myListener = new WindowEventListener();
myWindow.addWindowListener(myListener);
```

- public void addWindowListener(WindowListener l) {...}
- The parameter is the interface!
- Class casting – we pass it our WindowEventListener.

Now the JFrame listens for a window closing event.
Annoying Window Closing Behavior

- When click the close window “x”, JFrame hides the window as a default behavior.

- Bad – the program is still running, but you can’t see it.
  - Eats up your CPU.
  - Poor design of JFrame class.

- Be sure to call “System.exit(0);” in your windowClosing() method.
The JFrame will **usually** exit your application when it closes.

But it only calls System.exit(0) if you *haven’t* done anything in `windowClosing()`.

- If you have done something like “System.out.println(“Done”);”, then it won’t call `System.exit(0)`.
- It just HIDES the application. It is still running in the background.
- How annoying! Confusing.

To be safe, always call `System.exit(0)` from somewhere in your code.

- Somewhere that your code will actually be when it is ending.

If you don’t want to end your program in the `windowClosing()` method, then reset the default behavior.

- JFrame myWindow = new JFrame();
  myWindow.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);

- Now you have *complete* control from within the listener.
  - e.g., you can demand they enter a password before closing!
import javax.swing.*;
import java.awt.*;
public class FirstGraphics
{
    public static void main(String[] args)
    {
        JFrame myWindow = new JFrame();
        WindowEventListener myListener = new WindowEventListener();
        JButton myButton = new JButton("Push me");

        myWindow.addWindowListener(myListener);
        myWindow.setSize(300, 400);
        myWindow.setTitle("Test Window");

        Container myContentPane = myWindow.getContentPane();
        myContentPane.add(myButton);

        myWindow.setVisible(true);
    }
}
The Result Looks Like…

This entire box is the button!
What’s the Button Do?

- Nothing until we add a Listener!
  - Clicking the button is an event.
  - As with JFrame, there is already a listener interface.
    - called ActionListener
  - Once we create the listener

```java
JButton myButton = new JButton();
ButtonListener myButtonListener = new ButtonListener();
myButton.addActionListener(myButtonListener);
myContentPane.add(myButton);
```

We have to create this listener.
ActionListener

- Just implement the interface.
  - No convenience class like WindowAdapter because there is only one method, so not such a pain.

```java
text
import java.awt.event.*;
import javax.swing.*;
public class ButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        JOptionPane.showMessageDialog(null, “You dare to PUSH me?!”);
    }
}
```

This method was in “ActionListener”. We had to implement it.
public static void main(String[] args) {
    JFrame myWindow = new JFrame();
    WindowEventListener myListener =
        new WindowEventListener();
    ButtonListener myButtonListener =
        new ButtonListener();
    JButton myButton = new JButton("Push me");
    myButton.addActionListener(myButtonListener);
    myWindow.addWindowListener(myListener);
    myWindow.setSize(300, 400);
    myWindow.setTitle("Test Window");
    Container myContentPane =
        myWindow.getContentPane();
    myContentPane.add(myButton);
    myWindow.setVisible(true);
}
Layouts Managers: Adding More Components

To add more components to window, they must be arranged.

- Use one of the layout managers
  - GridLayout (specify a grid size)
  - FlowLayout (in a line next to each other)
  - BorderLayout (north, south, east, west center)
  - GridBagLayout (most flexible – grid with merged cells)
  - Etc.

- Their only job is arranging the components in the window.
BorderLayout

- **Five areas**
  - designated north, south, east, west, and center

<table>
<thead>
<tr>
<th>If use all five:</th>
</tr>
</thead>
</table>
| ![Diagram](image1)

<table>
<thead>
<tr>
<th>If use only N, S, Center:</th>
</tr>
</thead>
</table>
| ![Diagram](image2)

<table>
<thead>
<tr>
<th>If use only E, W, Center:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Using BorderLayout

```java
JFrame myWindow = new JFrame();
WindowEventListener myListener = new WindowEventListener();
JLabel myLabel = new JLabel("Yo");
JButton myButton = new JButton("Push me");

myWindow.addWindowListener(myListener);
myWindow.setSize(300, 400);

Container myContentPane = myWindow.getContentPane();

BorderLayout layout = new BorderLayout();
myContentPane.setLayout(layout);
myContentPane.add(myLabel, BorderLayout.WEST);
myContentPane.add(myButton, BorderLayout.EAST);
```

setLayout() takes a LayoutManager. All the layout classes implement this interface!
BorderLayout With Two Elements

- Ok, not too snazzy.
- Button too big.

Options:
1. setMaximumSize of the button.
2. set an icon in JButton constructor.
3. Or put layouts inside of layouts.
   - Use JPanels.
   - Study book!
   - Adding more components to the EAST forces the button to have a normal size – it can’t take up the whole EAST side because has to share that space.

This part of the window is the button!
Imagine your JFrame’s ContentPane is divided into a set of panels or “window panes”.

- Could put separate buttons (components) in each panel.
- Each panel could have its own layout manager.
JPanels and JFrames with Layouts

JPanel #1 with BorderLayout

JPanel #2 with BorderLayout

JFrame with JPanels place in EAST and WEST positions of BorderLayout
JFrame myWindow = new JFrame();
WindowEventListener myListener = new WindowEventListener();
myWindow.addWindowListener(myListener);
myWindow.setSize(300, 400);
Container myContentPane = myWindow.getContentPane();

JPanel topPanel = new JPanel();
topPanel.setLayout(new FlowLayout());
topPanel.add(new JLabel("Yo");

JPanel bottomPanel = new JPanel();
bottomPanel.setLayout(new FlowLayout());
JButton cancelButton = new JButton("Cancel");
cancelButton.addActionListener(new ButtonListener());
bottomPanel.add(cancelButton);

myContentPane.setLayout(new BorderLayout());
myContentPane.add(topPanel, BorderLayout.NORTH);
myContentPane.add(bottomPanel, BorderLayout.SOUTH);

Notice that created layout "on the fly". FlowLayout arranges things from left to right in the order added to the layout.
Multiple Buttons

- How does ActionListener know what button was pushed?
  - Each button should do different things!

```java
JPanel bottomPanel = new JPanel();
bottomPanel.setLayout(new FlowLayout());

JButton continueButton = new JButton("Continue");
continueButton.setActionCommand("rock on!");
continueButton.addActionListener(new ButtonListener());
bottomPanel.add(continueButton);

JButton cancelButton = new JButton("Cancel");
cancelButton.setActionCommand("quit");
cancelButton.addActionListener(new ButtonListener());
bottomPanel.add(cancelButton);
```

The actionListener (i.e., ButtonListener) gets sent this command “rock on!” or “quit”. Distinguishes the two buttons so listener isn’t confused!
Default Action Command

- Does not have to be set.

- If not set, the default action command is the string on the button.
  - E.g., “Continue” and “Cancel” in previous example.
  - But what if have a continue button in two different windows/panels?
    - May want different behavior.
  - Better to set a unique string with setActionCommand.
import java.awt.event.*;
import javax.swing.*;
public class ButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        if( e.getActionCommand().equals("quit") )
        {
            System.exit(0);
        }
        else if( e.getActionCommand().equals("rock on") )
        {
            JOptionPane.showMessageDialog(null, "Yo to you to.");
        }
    }
}
Making JPanels/JFrames Into Listeners

Now the JFrame is its own listener!

```java
public class ListenerFrame extends JFrame implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("quit") ) {
            System.exit(0);
        } else if (e.getActionCommand().equals("super size") ) {
            Dimension d = this.getSize();
            this.setSize(d.height + 25, d.width + 25);
            this.setVisible(true);
        } else {
            JOptionPane.showMessageDialog(null, "I don’t know what you want me to do!");
        }
    }
}
```

//continued next page
public void displayFrame()
{
    WindowEventListener myListener = new WindowEventListener();
    this.addWindowListener(myListener);
    this.setSize(300, 400);
    Container myContentPane = this.getContentPane();
    JPanel topPanel = new JPanel();
    topPanel.setLayout(new FlowLayout());
    topPanel.add(new JLabel("Yo"));
    JPanel bottomPanel = new JPanel();
    bottomPanel.setLayout(new FlowLayout());
    JButton continueButton = new JButton("Make Bigger");
    continueButton.setActionCommand("super size");
    continueButton.addActionListener(this);
    bottomPanel.add(continueButton);
    JButton cancelButton = new JButton("Cancel");
    cancelButton.setActionCommand("quit");
    cancelButton.addActionListener(this);
    bottomPanel.add(cancelButton);
    myContentPane.setLayout(new BorderLayout());
    myContentPane.add(topPanel, BorderLayout.NORTH);
    myContentPane.add(bottomPanel, BorderLayout.SOUTH);
    this.setVisible(true);
}

See how we use the JFrame as the listener!

Why didn’t I incorporate the WindowEventListener into MyFrame as well? (Answer: not an interface!)
public class UseFrame
{
    public static void main(String[] args)
    {
        // calls all the graphics -- nice and clean code
        ListenerFrame frame = new ListenerFrame();
        frame.displayFrame();
    }
}
Text I/O

- Can create an area to type text:
  - `JTextArea` writing = new JTextArea(numLines, numColumns);
    - String stuffTheyTyped = writing.getText();
      - e.g., Will want to get the text after they submit it with a button.
      - e.g., in ActionListener

  - new `JTextField`(numColumns)
    - `getText()`

- Password text:
  - new `JPasswordField`(numCharacters);
  - `getPassword()` //returns char[]!

- Read Book and API!
Swing Class Hierarchy (Partial!)
Listener Hierarchy (Partial!)

- EventListener
  - WindowListener
  - WindowFocusListener
  - WindowStateListener
    - WindowAdapter
  - MouseListener
    - MouseAdapter
    - MouseInputListener
      - MouseInputAdapter
    - MouseMotionListener
      - MouseMotionAdapter
  - ActionListener

Convenience classes (abstract)
Read Book and API!

- So much more to graphics.
  - Read your book.
  - Read the API.
  - Read Java graphics book.

- And don’t forget about inheritance, interfaces, polymorphism!
  - That’s the key to understanding graphics.