Objectives

- Event Handling
- Animation

GUI Review

- What is the purpose of a Layout Manager?
- Describe two different layout managers
- How can we create a customized layout?
- What are the components of event handling?

Review

- Show different versions of ColoredBackground GUI

EVENT HANDLING

Window Events

- Not every event is as simple to handle as a button click
- When a user closes a window, the window simply stops being displayed
  - Program will not end
- Suppose we want our program to end when a certain frame is closed
- Closing a frame is a ***window event***
  - In contrast to an *action event*

Catching Window Events

- To catch window events, create an object of a class that implements `WindowListener` interface
  - `WindowListener` is registered with frame using its `addWindowListener` method
- Note the parallels with action events
  - Different listener type and register it using a different (but similar) method call

The WindowListener Interface

- Contains 7 methods
  - One for each type of window event
  - A class that implements WindowListener must implement **all 7** methods

```
public interface WindowListener {
      void windowOpened(WindowEvent e);
      void windowClosing(WindowEvent e);
      void windowClosed(WindowEvent e);
      void windowIconified(WindowEvent e);
      void windowDeiconified(WindowEvent e);
      void windowActivated(WindowEvent e);
      void windowDeactivated(WindowEvent e);
}
```

Example: Implementing a WindowListener

What does this class do?

```
class Terminator implements WindowListener {
      public void windowClosing(WindowEvent evt) {
            System.exit(0);
      }

      public void windowOpened(WindowEvent e) {}
      public void windowClosed(WindowEvent e) {}
      public void windowIconified(WindowEvent e) {}
      public void windowDeiconified(WindowEvent e) {}
      public void windowActivated(WindowEvent e) {}
      public void windowDeactivated(WindowEvent e) {}
}
```


Example: Implementing a WindowListener

- Listens for window events on a frame and ends the program when the frame is closed

```
class Terminator implements WindowListener {
      public void windowClosing(WindowEvent evt) {
            System.exit(0);
      }

      public void windowOpened(WindowEvent e) {}
      public void windowClosed(WindowEvent e) {}
      public void windowIconified(WindowEvent e) {}
      public void windowDeiconified(WindowEvent e) {}
      public void windowActivated(WindowEvent e) {}
      public void windowDeactivated(WindowEvent e) {}
}
```

For JFrames use setDefaultClosedOperation

Adapter Classes

- Writing code for 6 methods that don't do anything is somewhat tedious
  - Eclipse helps
- Most AWT listener interfaces have a corresponding ***adapter class***
  - Implements each of interface's methods but does nothing inside each
  - No adapter classes for AWT interfaces with only one method (such as ActionListener)

Adapter Classes

- If you want a WindowListener class that does nothing with 6 of the 7 window events but ends program when window is closed
  - Create a new class that **extends** WindowAdapter and override relevant method (s)
- When could extending a class be a problem?
  - How big of a concern is that for this specific case/type of class?

Extending an Adapter Class

- Redefine Terminator in much less code…

```
class Terminator extends WindowAdapter {
      public void windowClosing(WindowEvent evt) {
            System.exit(0);
      }
      // all other methods are the same as in
      // WindowAdapter--all do nothing.
}
```

Registering a WindowListener

- Register `Terminator` to listen for window events
- Assuming that our "main" window frame is named `frame` (i.e., if `frame` is closed the program should exit)…

```
WindowListener listener = new Terminator();
frame.addWindowListener(listener);
```


Alternative: Registering a WindowListener

```
frame.addWindowListener( new
      WindowAdapter() {
            public void windowClosing(WindowEvent evt) {
                  System.exit(0);
            }
      } );
```

What is going on in code?

Anonymous Inner Class

```
frame.addWindowListener( new
      WindowAdapter() {
            public void windowClosing(WindowEvent evt) {
                  System.exit(0);
            }
      } );
```

- Defines a new class without a name that extends `WindowAdapter` class
- Adds `windowClosing` method to anonymous class
- Inherits other 6 methods from `WindowAdapter`
- Creates an object of this new class
  - Object also does not have a name
- Passes new no-name object to `addWindowListener` method of `frame`

TYPES OF EVENTS

AWT Event Hierarchy

- 10 different types of events in AWT
  - Semantic events
  - Low-level events

AWT Event Types: Semantic Events

- **Semantic event**: event that expresses what a user did

Type	Cause
`ActionEvent`	button click, menu selection, selecting a list item, pressing ENTER in a text field
`AdjustmentEvent`	User adjusted a scroll bar
`ItemEvent`	user made a selection from a set of checkboxes or list items
`TextEvent`	the contents of a text field or text area were changed

AWT Event Types: Low-Level Events

- **Low-level event**: makes a semantic event possible

Type	Cause
`ComponentEvent`	component changed (resized, moved, shown, etc…)
`KeyEvent`	a key pressed or released
`MouseEvent`	mouse moved or dragged, or mouse button pressed
`FocusEvent`	component got or lost focus
`WindowEvent`	window activated, closed, etc.
`ContainerEvent`	component added or deleted

AWT Event Types

- Example:
  - Adjusting a scrollbar is a *semantic* event
  - Made possible by low-level events, such as dragging the mouse
- As a general rule,

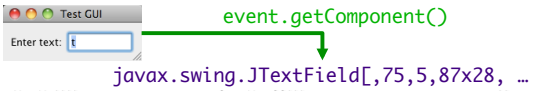
low-level events cause semantic events to happen

AWT Event Listeners

- 11 Event Listener Interfaces
  - `ActionListener, AdjustmentListener, ItemListener, TextListener, ComponentListener, ContainerListener, FocusListener, KeyListener, MouseListener, MouseMotionListener, and WindowListener`
- See API for interfaces and their methods
- Each listener interface with > 1 method has a corresponding ***adapter class***
  - Implements interface with all empty methods

Components and ComponentEvents

- A ***component*** is a user interface element
  - Ex: button, textfield, or scrollbar
- All low-level events inherit from `ComponentEvent`
  - `getComponent()` returns component that originated event
    - Similar to `getSource()` but returns object as a `Component` and not an `Object`
- Example: A user inputs text into a text field, generating a key event. Calling `getComponent()` on the event returns a reference to that text field



Containers and ContainerEvents

- A ***container*** is a screen area or component
  - Can contain components, such as a panel
- A **`ContainerEvent`** is generated whenever a component is added or removed from the container
  - Supports programming dynamically-changing user interfaces

FocusEvents

- A **`FocusEvent`** is generated when a component gains or loses focus
- **`FocusListener`** must implement two methods:
  - `focusGained()`: called whenever listener's event source gains focus
  - `focusLost()`: called whenever listener's event source loses focus

KeyEvents

- A **KeyEvent** is generated when a key is pressed or released
- A **KeyListener** must implement 3 methods:
  - `keyPressed()` will run whenever a key is pressed
  - `keyReleased()` will run whenever that key is released
  - `keyTyped()` combines the two above
    - Runs when key is pressed and then released and signifies a keystroke

KeyEvents

- Any `Component` can be an *event source* for a `KeyEvent`
  - A component generates a `KeyEvent` whenever a key is typed in that component
- For example, if user types into a text field that text field will generate appropriate `KeyEvents`

MouseEvents

- `MouseEvents` are generated like `KeyEvents`
  - `mousePressed()`
  - `mouseReleased()`
  - `mouseClicked()`
  - You can ignore first 2 if you only care about clicking
- Call `getClickCount()` on a `MouseEvent` object to distinguish between a single and a double click
- Distinguish between mouse buttons by calling `getModifiers()` on a `MouseEvent` object
  - E.g., middle button

MouseEvents

- `MouseEvents` are also generated when mouse pointer enters and leaves components (`mouseEntered()` and `mouseExited()`)
  - Part of `MouseListener` interface
- Actual movement of mouse is handled with `MouseMotionListener` interface
  - Most applications only care about where you click and not how and where you move mouse pointer around

Example: Window Events

- Combines `WindowListener`, `WindowFocusListener`, `WindowStateListener`

`WindowEventDemo.java`

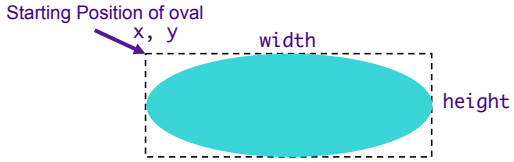
GRAPHICS PROGRAMMING

Graphics Object

- Abstract class
  - Implementation different for each platform
- A collection of settings for drawing images and text, such as colors and fonts
- Where used:
  - `paintComponent(Graphics g)`

Drawing Lines, Rectangles, Ovals

- Draw ovals, rounded rectangles within bounding rectangle



- Filled or outlined (e.g., `fillRect` vs `drawRect`)
- Can also draw arcs, polygons, polylines

Colors

- Colors made up of three components
  - Red, Green, Blue component
  - RGB values
    - Components: either 0 to 255 or 0.0 to 1.0
- `Color` class defines 13 color constants
  - `black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white,` and `yellow`
  - Also defined in all caps
  - See API

`http://en.wikipedia.org/wiki/List_of_colors`

Using Graphics object

1. Set the color/font
2. Draw the shape/string

Different Implementation Approach

```
public static void main(String args[]) {

  JFrame frame = new JFrame("Using colors");
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  ColorJPanel colorJPanel = new ColorJPanel();
  frame.add(colorJPanel);

  frame.setSize(500, 180);
  frame.setVisible(true);

}
```

Extends JPanel

ColorJPanel.java

Understanding Code

- Goblin Game
  - How draws
  - How does event handling
  - How animates

Import project:
`/home/courses/cs209/handouts/screensavers.tar`

- Bouncers
  - How draws
  - How animates

Grading Testing Project

- If a test case passes, what does that mean?
- If a test case fails, what does that mean?

Grading Testing Project

- If a test case passes, what does that mean?
  - Code is correct (positive)
  - Code has a bug that test case doesn't reveal (false negative)
- If a test case fails, what does that mean?
  - Code has a bug (negative)
    - Or doesn't follow your specification
  - Code is correct but test case is incorrect (false positive)
- Are there any other possibilities?

Grading Testing Project

		Test Case Result	
		Pass	**Fail**
Application	**Correct**	True Positive	False Negative
	Faulty	False Positive	True Negative

- Which is worse: false positives or false negatives?

My Process

- Determine your assumptions about constructor
- Plug in the appropriate Car implementation for your tests

```
Car car = new CorrectCarFullTank();
```

- Run all your tests
  - Look at pass/fail results, coverage results
- Look at your tests → specification coverage

Common Issues

- Incorrect specification of expected results
  - `getGear()` returns an int, not a String
  - Math errors
- Missing tests of functionality
- Missing assumptions
  - Car in PARK when refueling
- Changing Car API
  - Package-private constructor
- JUnit misunderstandings
  - Set up; How many exceptions

Grade breakdown

Grade	Points
A	270-300
B	240-269
C	210-239

Midterm Prep

Document posted online

- Java
  - Collections Framework
  - Comparison with Python
  - Jar files
- Software Development
  - Models
  - Testing
  - Design Principles
  - Code smells
  - Refactoring
- GUI programming
  - Event handling, inner classes