

Objectives

- Event Handling
- Animation

Nov 9, 2011

Sprenkle - CSCI209

1

Discussion of Roulette Assignment

- How easy/difficult to refactor for extensibility?
- Was it easier to add to your refactored code?
 - What would your refactored classes have looked like if I hadn't told you that you were going to add the three other bets?
- How easy/difficult was it to test your classes?
 - Will you ever run into something similar again?
 - What lessons did you learn?

Nov 9, 2011

Sprenkle - CSCI209

2

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?

Nov 9, 2011

Sprenkle - CSCI209

3

Compiler's Names of Classes

- Contents of Eclipse project's bin directory from last class:

```

sprenkle@spartacus examples$ ls
ColorAction.class          ColoredBackgroundRefactored$ColorAction.class
ColoredBackground$ColorAction.class
ColoredBackgroundRefactored.class
ColoredBackground.class    ColoredBackgroundSelfListener.class
ColoredBackground2.class   FusedGridLayout.class
[ColoredBackgroundRefactored$1.class  ThreeButtonsFrame.class

```

Some unusual names. Why?

Nov 9, 2011

Sprenkle - CSCI209

4

Other types of events

EVENT HANDLING

Nov 9, 2011

Sprenkle - CSCI209

5

Window Events

- When a user closes a window, the window simply stops being displayed
 - Program does 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*

Nov 9, 2011

Sprenkle - CSCI209

6

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

Nov 9, 2011

Sprenkle - CSCI209

7

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);
}
```

8

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) {}
}
```

Nov 9, 2011

Sprenkle - CSCI209

9

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 setDefaultCloseOperation

Nov 9, 2011

Sprenkle - CSCI209

10

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)

Nov 9, 2011

Sprenkle - CSCI209

11

Adapter Classes

- If you want a WindowListener class that does nothing with most window events
 - 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?

Nov 9, 2011

Sprenkle - CSCI209

12

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.
}
```

Nov 9, 2011

Sprenkle - CSCI209

13

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);
```

Nov 9, 2011

Sprenkle - CSCI209

14

Alternative: Registering a WindowListener

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

What is going on in this code?

Nov 9, 2011

Sprenkle - CSCI209

15

Anonymous Inner Class

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

- Defines a new anonymous class 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

Nov 9, 2011

Sprenkle - CSCI209

16

TYPES OF EVENTS

Nov 9, 2011

Sprenkle - CSCI209

17

AWT Event Hierarchy

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

Rule of thumb: low-level events cause semantic events to happen

Nov 9, 2011

Sprenkle - CSCI209

18

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

Nov 9, 2011

Sprenkle - CSCI209

19

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

Nov 9, 2011

Sprenkle - CSCI209

20

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

Nov 9, 2011

Sprenkle - CSCI209

21

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

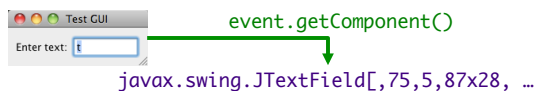
Nov 9, 2011

Sprenkle - CSCI209

22

Components and ComponentEvents

- A **component** is a user interface element
 - Examples: button, textfield, 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



Nov 9, 2011

Sprenkle - CSCI209

23

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

Nov 9, 2011

Sprenkle - CSCI209

24

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

Nov 9, 2011

Sprenkle - CSCI209

25

KeyEvent

- 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

Nov 9, 2011

Sprenkle - CSCI209

26

KeyEvent

- Any Component can be an *event source* for a **KeyEvent**
 - A component generates a **KeyEvent** whenever a key is typed in that component
- Example:
 1. User types into a text field
 2. That text field generates appropriate **KeyEvents**

Nov 9, 2011

Sprenkle - CSCI209

27

MouseEvent

- **MouseEvent**s 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

Nov 9, 2011

Sprenkle - CSCI209

28

MouseEvent

- **MouseEvent**s 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

Nov 9, 2011

Sprenkle - CSCI209

29

Example: Window Events

`WindowEventDemo.java`

- Combines **WindowListener**, **WindowFocusListener**, **WindowStateListener**

Nov 9, 2011

Sprenkle - CSCI209

30

GRAPHICS PROGRAMMING

Nov 9, 2011

Sprenkle - CSCI209

31

Review: 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)`

Nov 9, 2011

Sprenkle - CSCI209

32

Drawing Lines, Rectangles, Ovals

- Draw ovals, rounded rectangles within bounding rectangle

Starting Position of oval



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

Nov 9, 2011

Sprenkle - CSCI209

33

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

Nov 9, 2011

Using Graphics object

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

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    this.setBackground(Color.WHITE);

    // set new drawing color using integers
    g.setColor(new Color(255, 0, 0));
    g.fillRect(15, 25, 100, 20);
    g.drawString("Current RGB: " + g.getColor(), 130, 40);
    ...
}
```

From `ColorPanel.java`

Nov 9, 2011

Sprenkle - CSCI209

35

Understanding Code

Import project:
</home/courses/cs209/handouts/screensavers.tar>

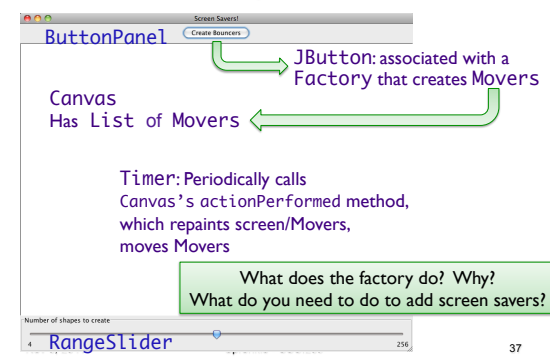
- Simple Bouncers
 - How draws
 - How animates
- Screen Savers
 - What represents an object in the screen saver?
 - How generates screen saver objects?
 - How handles animation?
 - How handles events?

Nov 9, 2011

Sprenkle - CSCI209

36

Screensavers GUI/Architecture



37

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

Nov 9, 2011

Sprenkle - CSCI209

38

TODO

- Fri: Exam
- Monday, EC: Naomi Oreskes talk
 - 5:30 p.m. talk
- Next Fri: ScreenSavers

Nov 9, 2011

Sprenkle - CSCI209

39