

Objectives

- Event Handling
- Design Patterns

Nov 10, 2008

Sprenkle - CS209

1

Assignment 12 Questions

Nov 10, 2008

Sprenkle - CS209

2

Animation Review

- How do we “draw” in Java?
- What did we use to trigger animation?

Nov 10, 2008

Sprenkle - CS209

3

EVENT HANDLING

Nov 10, 2008

Sprenkle - CS209

4

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

Nov 10, 2008

Sprenkle - CS209

5

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
 - Change listener type and register it using a different (but similar) method call

Nov 10, 2008

Sprenkle - CS209

6

The WindowListener Interface

- Contains **seven** methods
 - One for each type of window event
 - A class that implements **WindowListener** must implement all seven 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);
}
```

Nov 10, 2008

Sprenkle - CS209

9

Implementing a WindowListener

- To create an object that can listen for window events on a frame and end the program when the frame is closed...

```
class Terminator implements WindowListener {
    public void windowClosing(WindowEvent evt) {
        System.exit(0);
    }
    // For JFrames use setDefaultCloseOperation

    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 10, 2008

Sprenkle - CS209

8

Adapter Classes

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

Nov 10, 2008

Sprenkle - CS209

9

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?

Nov 10, 2008

Sprenkle - CS209

10

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 10, 2008

Sprenkle - CS209

11

Registering a WindowListener

- Register **Terminator** to listen for window events
- Assuming that our "main" window frame is named **frame1** (e.g., if **frame1** is closed the program should exit)...

```
WindowListener listener1 = new Terminator();
frame1.addWindowListener(listener1);
```

Nov 10, 2008

Sprenkle - CS209

12

Anonymous Inner Class

```
frame1.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 frame1

Nov 10, 2008

Sprenkle - CS209

13

AWT Event Hierarchy

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

Nov 10, 2008

Sprenkle - CS209

14

AWT Event Types: Semantic Events

- **Semantic event**: event that expresses what a user did, such as clicking a button.
- **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 10, 2008

Sprenkle - CS209

15

AWT Event Types: Low-Level Events

- **Low-level event**: makes a semantic event possible
- **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 10, 2008

Sprenkle - CS209

16

AWT Event Types

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

Nov 10, 2008

Sprenkle - CS209

17

AWT Event Listeners

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

Nov 10, 2008

Sprenkle - CS209

18

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: if a key event was generated because of an input into a text field, then `getComponent` returns a reference to that text field

Nov 10, 2008

Sprenkle - CS209

19

Containers and ContainerEvents

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

Nov 10, 2008

Sprenkle - CS209

20

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 10, 2008

Sprenkle - CS209

21

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 – it runs when key is pressed and then released and signifies a keystroke

Nov 10, 2008

Sprenkle - CS209

22

KeyEvents

- With what type of object does a **KeyListener** register with?
- What is an event source for a **KeyEvent**?
- 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 textfield that textfield will generate appropriate **KeyEvents**

Nov 10, 2008

Sprenkle - CS209

23

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

Nov 10, 2008

Sprenkle - CS209

24

MouseEvents

- MouseEvents are also generated when mouse pointer enters and leaves components (mouseEntered() and mouseExited()).
- All of those methods are 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 10, 2008

Sprenkle - CS209

25

DESIGN PATTERNS

Nov 10, 2008

Sprenkle - CS209

26

Design Pattern

- General reusable solution to a commonly occurring problem in software design
- Not a finished design that can be transformed directly into code
- Description or *template* for how to solve a problem that can be used in many different situations

Nov 10, 2008

Sprenkle - CS209

27

Defined Design Patterns

- Software best practices
- Catalogued and discussed in *Design Patterns: Elements of Reusable Object-Oriented Software*
 - Written by the "Gang of Four"
 - Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
 - Erich Gamma also co-wrote JUnit framework

Nov 10, 2008

Sprenkle - CS209

28

Applying Design Patterns

- Recognize problem as one that can be solved by a design pattern
- Apply pattern to your problem
- Danger: overapplying design patterns
 - Fall back: Identify and resolve code smells

Nov 10, 2008

Sprenkle - CS209

29

Design Pattern: Factory Methods

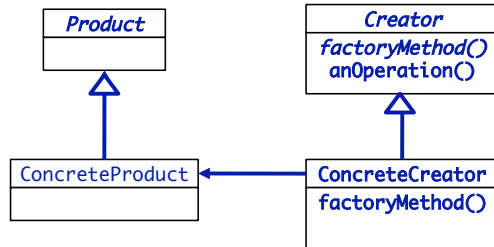
- Deals with problem of creating objects without specifying exact (concrete) class of created object
- Defines a method for creating objects
 - Subclasses can override method to specify the derived type of product that will be created
- Often used to refer to any method whose main purpose is creating objects

Nov 10, 2008

Sprenkle - CS209

30

Factory Method Pattern



UML Diagram

Nov 10, 2008

Sprenkle - CS209

31

Mapping to Screen Savers

- How does the screen saver application use factory methods?
- What would be the alternative solution?
- What problems are the factories addressing?

Nov 10, 2008

Sprenkle - CS209

32

Mapping to Screen Savers

- How does the screen saver application use factory methods?
- What would be the alternative solution?
- What problems are the factories addressing?
 - Delegate creation of concrete Movers
 - Likely to change
 - Encapsulate change in factory
 - Using abstraction instead of specifying concrete classes
 - Reduces dependencies to concrete classes

Nov 10, 2008

Sprenkle - CS209

33

Dependency Inversion Principle

Depend upon abstractions.
Do not depend upon concrete classes.

- High-level components should not depend on low-level components
 - Both should depend on abstractions
- Abstractions should not depend upon details. Details should depend upon abstractions
- "Inversion" from the way you think
- Other techniques besides Factory Method for adhering to principle

Nov 10, 2008

Sprenkle - CS209

34

Dependency Inversion Principle

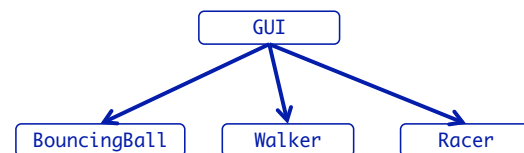
- How would we build/design the screen saver application?
 - Know we need to view/display a screen saver
 - Buttons, slider, objects that move
 - Top-down
 - Know we need to create a bunch of types of screen savers
 - Abstraction
 - Bottom-up

Nov 10, 2008

Sprenkle - CS209

35

Traditional Screen Saver Dependencies



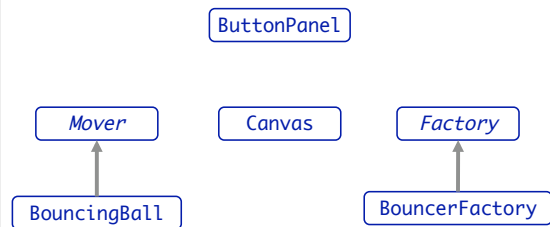
High-level component is dependent on concrete classes.
If implementations change, GUI may have to change

Nov 10, 2008

Sprenkle - CS209

36

Screen Saver Dependencies

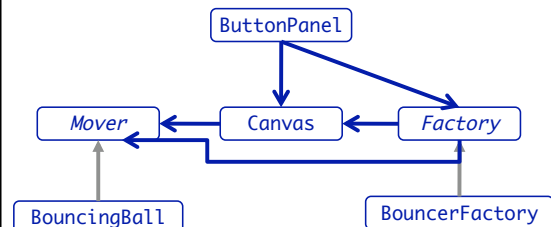


Nov 10, 2008

Sprenkle - CS209

37

Screen Saver Dependencies



Nov 10, 2008

Sprenkle - CS209

38

Guidelines to Follow DIP

- No variable should hold a reference to a concrete class
 - Using *new* → holding reference to concrete class
 - Use factory instead
- No class should derive from a concrete class
 - Depending on a concrete class
 - Derive from an interface or abstract class instead
- No method should override an implemented method of any of its base classes
 - Base class wasn't an abstraction
 - Those methods are meant to be shared by subclasses

What's the problem with following all of these guidelines?

Nov 10, 2008

Midterm Prep

- Document posted online
- Software Development
 - Models
 - Testing
 - Design Principles
 - Code smells
 - Refactoring
- GUI programming
 - Event handling, inner classes, animation
- Jar files
- Unix commands

Nov 10, 2008

Sprenkle - CS209

40