Announcements

• Game Project Complete Code
  – Due: Monday, December 10th
  – NO LATE SUBMISSIONS PERMITTED

• Final Exam
  – Thursday, December 20th, 6:00-8:00PM
  – Locations: TBA
  – makeup exam
    form available on the course web site
Paint Revisited
(thoroughly discussed in Chap 31)

Using Anonymous Inner Classes
Refactoring for OO Design

(See PaintA.java ... PaintE.java)
Adapters

MouseAdapter

KeyAdapter
Mouse Interaction in Paint

Point Mode

Line Start Mode

Line End Mode

Mouse Released (in the canvas)
[add new point]

Mouse Pressed
[store point, set preview shape]

Mouse Released
[add new line, set preview to null]

Mouse Dragged
[update preview]
Two interfaces for mouse listeners

interface MouseListener extends EventListener {
    public void mouseClicked(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
}

interface MouseMotionListener extends EventListener {
    public void mouseDragged(MouseEvent e);
    public void mouseMoved(MouseEvent e);
}

Lots of boilerplate

- There are seven methods in the two interfaces.
- We only want to do something interesting for three of them.
- Need "trivial" implementations of the other four to implement the interface...

```java
public void mouseMoved(MouseEvent e) {
    return;
}
public void mouseClicked(MouseEvent e) {
    return;
}
public void mouseEntered(MouseEvent e) {
    return;
}
public void mouseExited(MouseEvent e) {
    return;
}
```

- Solution: MouseAdapter class...
Adapter classes:

- Swing provides a collection of abstract event adapter classes
- These adapter classes implement listener interfaces with empty, do-nothing methods
- To implement a listener class, we extend an adapter class and override just the methods we need

```java
private class Mouse extends MouseAdapter {
    public void mousePressed(MouseEvent e) { ... }
    public void mouseReleased(MouseEvent e) { ... }
    public void mouseDragged(MouseEvent e) { ... }
}
```
Mushroom of Doom

How do we put Swing components together to make a complete game?
### Game State

<table>
<thead>
<tr>
<th>GameCourt</th>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>snitch</strong></td>
<td>pos_x: 170</td>
</tr>
<tr>
<td><strong>poison</strong></td>
<td>pos_y: 170</td>
</tr>
<tr>
<td><strong>square</strong></td>
<td>v_x: 2</td>
</tr>
<tr>
<td><strong>playing</strong></td>
<td>v_y: 3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Square</th>
<th>Poison</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pos_x</strong></td>
<td>pos_x: 130</td>
</tr>
<tr>
<td><strong>pos_y</strong></td>
<td>pos_y: 130</td>
</tr>
<tr>
<td><strong>v_x</strong></td>
<td>v_x: 0</td>
</tr>
<tr>
<td><strong>v_y</strong></td>
<td>v_y: 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
How can we share code between the game objects, but show them differently?
void tick() {
    if (playing) {
        square.move();
        snitch.move();
        snitch.bounce(snitch.hitWall()); // bounce off walls...
        snitch.bounce(snitch.hitObj(poison)); // ...and the mushroom
    }
    else if (square.intersects(poison)) {
        playing = false;
        status.setText("You lose!");
    } else if (square.intersects(snitch)) {
        playing = false;
        status.setText("You win!");
    }
    repaint();
}
Updating the Game State: keyboard

```java
setFocusable(true);
addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent e) {
        if (e.getKeyCode() == KeyEvent.VK_LEFT)
            square.v_x = -SQUARE_VELOCITY;
        else if (e.getKeyCode() == KeyEvent.VK_RIGHT)
            square.v_x = SQUARE_VELOCITY;
        else if (e.getKeyCode() == KeyEvent.VK_DOWN)
            square.v_y = SQUARE_VELOCITY;
        else if (e.getKeyCode() == KeyEvent.VK_UP)
            square.v_y = -SQUARE_VELOCITY;
    }

    public void keyReleased(KeyEvent e) {
        square.v_x = 0;
        square.v_y = 0;
    }
});
```

Make square's velocity nonzero when a key is pressed

Make square's velocity zero when a key is released
How does the user interact with the game?

1. Clicking Reset button restarts the game
2. Holding arrow key makes square move
3. Releasing key makes square stop
Example 1: Mushroom of Doom
Example: MOD Program Structure

- **GameCourt, GameObj + subclass local state**
  - object location & velocity
  - status of the game (playing, win, loss)
  - how the objects interact with each other (tick)

- **Draw methods**
  - `paintComponent` in `GameCourt`
  - draw methods in `GameObj` subclasses
  - status label

- **Game / GameCourt**
  - Reset button (updates model)
  - Keyboard control (updates square velocity)
Example: Paint Program Structure

• Main frame for application (class Paint)
  – List of shapes to draw
  – The current color
  – The current line thickness

• Drawing panel (class Canvas, inner class of Paint)

• Control panel (class JPanel)
  – Contains radio buttons for selecting shape to draw
  – Line thickness checkbox, undo and quit buttons

• Connections between Preview shape (if any...)
  – Preview Shape: View <-> Controller
  – MouseAdapter: Controller <-> Model
Example: CheckBox

- Views
  - Checkbox
  - true or false

- Model
  - Selected?
  - Pressed?

- Controllers
  - setSelected
  - mouseListener
  - keyListener

Class JToggleButton.ToggleButtonModel

boolean isSelected() Checks if the button is selected.
void setPressed(boolean b) Sets the pressed state of the button.
void setSelected(boolean b) Sets the selected state of the button.
Example: Chat Server

getChannels
getUsers
getOwner
...
Views

Internal
Representation
owners: Map<Channel,
Users>
users: Map<Channel,
Set<Users>>
...
Model

createChannel
joinChannel
invite
kick
...
Controllers

ServerModel
Example: Web Pages

Internal Representation:
DOM (Document Object Model)

Model

View

Controllers

JavaScript API

document.addEventListener()
MVC Pattern

- Model
  - Manipulates
  - Updates
- View(s)
  - Adjusts
- Controller
  - Uses
  - Sees
- User
MVC Benefits?

• Decouples important "model state" from how that state is presented and manipulated
  – Suggests where to insert interfaces in the design
  – Makes the model testable independent of the GUI

• Multiple views
  – e.g. from two different angles, or for multiple different users

• Multiple controllers
  – e.g. mouse vs. keyboard interaction
MVC Variations

• Many variations on MVC pattern

• Hierarchical / Nested
  – As in the Swing libraries, in which JComponents often have a "model" and a "controller" part

• Coupling between Model / View or View / Controller
  – e.g. in MOD the Model and the View are coupled because the model carries most of the information about the view
Design Patterns

- Design Patterns
  - Influential OO design book published in 1994 (so a bit dated)
  - Identifies many common situations and "patterns" for implementing them in OO languages

- Some we have seen explicitly:
  - e.g. Iterator pattern

- Some we've used but not explicitly described:
  - e.g. The Broadcast class from the Chat HW uses the Factory pattern

- Some are workarounds for OO's lack of some features:
  - e.g. The Visitor pattern is like OCaml's fold + pattern matching