CS102

Introduction to data structures, algorithms, and object-oriented programming

April 25, 2016
Swing Graphic Components

- **JPanel**: fundamental class in Swing.

  The basic *JPanel* is just a blank rectangle. There are two ways to make use of a *JPanel*:
  1. **add other components** to the panel
  2. **draw something** in the panel.

*JPanels* can be used as drawing surfaces:
1) define a class that is a subclass of *JPanel* and
2) write a `paintComponent` method in that class to draw the desired content in the panel.
awt.event Parts

- ActionEvent
- MouseEvent
- MouseMotionEvent

- ActionListener
- MouseListener
- MouseMotionListener

- FocusListener and FocusEvent
- KeyListener and KeyEvent
Mouse Events

- Mouse events require 2 different interface implementations, depending on what you want to do

- The simplest mouse events are defined in the MouseListener interface:
  
  ```java
  public void mousePressed(MouseEvent evt);
  public void mouseReleased(MouseEvent evt);
  public void mouseClicked(MouseEvent evt);
  public void mouseEntered(MouseEvent evt);
  public void mouseExited(MouseEvent evt);
  ```

- The second type of mouse listener is the MouseMotionListener interface:
  
  ```java
  public void mouseDragged(MouseEvent evt);
  public void mouseMoved(MouseEvent evt);
  ```
Mouse Events

• The ordinary way to register a mouse listener is to write X.addMouseListener(Y) where Y is the listener and X is the component that will generate the mouse events in the constructor.
  
  ```
  addMouseListener(this); // in the constructor
  ```

• A JPanel or JFrame or other container can be registered to listen for MouseEvents.
Focus Events

• If a component is to change its appearance when it has the input focus, it needs some way to know when it is selected.

• Objects are notified about changes of input focus by events of type `FocusEvent`
  – public void focusGained(FocusEvent evt);
  – public void focusLost(FocusEvent evt);

• A JPanel or JFrame can be registered to listen for FocusEvents:
  – addFocusListener(this); // in the constructor
4 Steps of Any XXXHandling

1. import java.awt.event.*
2. implement XXXListener somewhere in class
3. define the event-handling methods
4. addXXXListener to components
Rule of Drawing

Drawing rule: all drawing shall henceforth and forever be done in a paintComponent() method.

In real-life programming of drawing applications, many people (including the author of our book) violate this rule by obtaining the Graphics content (if class extends JPanel) like so:

```java
Graphics g = getGraphics(); // Graphics context for drawing directly out of any method in the class, not just the paintComponent method.
```

If you do use the above command to access a Graphics component, you need to use `g.dispose()` in the method when you are done using `g`. 
Three Commonly-Used Types of Listeners

• **ActionListener**
  Responds to ActionEvents generated by GUI components and a Timer object

• **MouseListener and MouseMotionListener**
  – **MouseListener** responds to events like clicking, pressing on component, entering and exiting window, releasing the mouse button.
  – **MouseMotionListener** responds to events like moving or dragging the mouse across the scene.

• **KeyListener:**
  Call `keyListener()` method in constructor (later)
Setting up a MouseListener

1. Put the import specification "import java.awt.event.*;" at the beginning of your source code;

2. Declare that class, inner class, or anonymous inner class implements the listener interface MouseListener, or that it extends MouseAdapter;

3. Provide definitions in that class for the methods specified by the interface (MouseListener has 5 methods to be overridden);

4. Register an object that belongs to the listener class with the component that will generate the events by calling a method such as addMouseListener() on the component.
Generating MouseEvents

• Classes that generate MouseEvents should somehow implement the MouseListener.

The MouseListener interface requires the implementation of 5 methods

```java
public class Mouser implements MouseListener {
    ...
    public void mouseClicked(MouseEvent e){}
    public void mousePressed(MouseEvent e){}
    public void mouseEntered(MouseEvent e){}
    public void mouseExited(MouseEvent e){}
    public void mouseReleased(MouseEvent e){}
    ...
}
```
MouseListeners

• There may be only one or two types of MouseEvent you want to respond to.

1. If the class implements MouseListener interface:

```java
public void mousePressed(MouseEvent evt) {
    int x = evt.getX(); // Location where user clicked mouse.
    int y = evt.getY();
    if (x >= x2 && x < x2+30 && y >= y2 && y < y2+30) {
        dragging = true;
        offsetX = x - x2;
        offsetY = y - y2;
    }
}

public void mouseReleased(MouseEvent evt) {
    dragging = false;
}

// empty methods required by interface
public void mouseClicked(MouseEvent evt) { }
public void mouseEntered(MouseEvent evt) { }
public void mouseExited(MouseEvent evt) { }
```
2. Add an inner class that implements MouseListener

```java
private static class MouseHandler implements MouseListener {
    public void mouseClicked(MouseEvent e) {
        saySomething("Mouse clicked (# of clicks: "+ e.getClickCount() + ")", e);
    }

    void saySomething(String eventDescription, MouseEvent e) {
        textarea.append(eventDescription + " detected on "
                        + e.getComponent().getClass().getName() + "." + newline);
    }
}

plus possibly empty implementation of 4 other mouse event handlers
```
Anonymous inner class MouseListeners

3. Add an **anonymous inner class** to a component inside the constructor. This inner class is either of sub-type MouseListener or sub-class of MouseAdapter

```java
this.addMouseListener(new MouseAdapter(){
    public void mouseClicked(MouseEvent evt){
        clickCount++;
    }
});
```

The **MouseListener** class provides null implementations for all methods you do not implement, so you don't need to write 4 empty methods. Adapter classes are useful only for creating subclasses.
Tracking Mouse Movements

• Implement the MouseMotionListener interface.

  Requires implementation of either mouseMoved or mouseDragged methods.

  You will use the mouseMoved method in lab today to make your paddle move left and right.

• Implement the mouseDragged method to cause some affect when the mouse is moved with its button down.
Sensing Impact between Objects drawn on scene

• If we want the ball to bounce off the paddle, we need to detect:
  1. When lower surface of ball is touching top of paddle;
  2. when ball is moving downward (toward paddle).
• When both these conditions hold, we want to negate the y direction of the paddle.
• How do we detect the condition when the ball is touching the panel?
What happens if the ball misses the padde?

• If we are playing a game, the ball would probably drop off the bottom of the scene (a bad thing).

• How do we make ball drop off the bottom of the scene?
Keyboard Events

A GUI uses the idea of input focus to determine the component associated with keyboard events. At any given time, exactly one interface element on the screen has the input focus, and that is where all keyboard events are directed.

In Java, keyboard event objects belong to a class called `KeyEvent`. An object that needs to listen for `KeyEvents` must implement the interface named `KeyListener`. Furthermore, the object must be registered with a component by calling the component's `addKeyListener()` method. The registration is done with the command "component.addKeyListener(listener);"

The `KeyListener` interface defines the following methods, which must be included in any class that implements `KeyListener`:

- public void keyPressed(KeyEvent evt);
- public void keyReleased(KeyEvent evt);
- public void keyTyped(KeyEvent evt);