

Classic Data Processing

- Input specified as part of the program design
 - · Example: process bank account deposits Repeated set of transactions Each transaction consists of a deposit slip (transaction header) followed by 1 or more checks to be deposited to the account
- · Program expects input in required order
 - Program structure mirrors input organization while (more input) {
 - //read and process transaction header //read and process individual checks
 - }

3

Event-Driven Programming

- Idea: program initializes itself then accepts events in whatever random order they occur
- Kinds of events
 - · Mouse move/drag/click, Keyboard, Touch screen, Joystick, game controller
 - Window resized or components changed
 - · Activity over network or file stream
 - Timer interrupt
 - (can still think of this as processing an "input stream", but point of view is basically different)
- First demonstrated in the 1960s(!); major developments at Xerox PARC in the 1970s (Alto workstation, Smalltalk)
- Available outside research community with Apple Macintosh (1984)

4

Java Events

· An event is represented by an event object

- AWT/Swing events are subclasses of AWTEvent. Some examples: ActionEvent – button pressed KeyEvent – keyboard input
- MouseEvent mouse move/drag/click/button press or release
- All user interface components generate events when appropriate

5

- Event objects contain information about the event
 - · User interface object that triggered the event
 - Other information appropriate for the event. Examples: ActionEvent – contents of button text generating event (if from a button) MouseEvent – mouse coordinates of the event
- All in java.util.event need to import this to handle events

Event Listeners

Basic idea: any object that is interested in an event registers itself with
the component that can generate the event

- · The object must implement the appropriate Interface
- ActionListener, KeyListener, MouseListener (buttons), MouseMotionListener (move/drag), others ...
- When the event occurs, the appropriate method of the object is called
 actionPerformed, keyPressed, keyReleased, keyTyped, mouseClicked,
 - MouseDragged, etc. etc. Reminder – because these are part of an Interface, you can't change their signatures.

6

8

An event object describing the event is a parameter to the receiving method

Example: Mouse Clicks

public class Mouser extends JPanel implements MouseListener {
 /** Constructor - register this object to listen for mouse events */
 Mouser() {
 super();
 }
}

addMouseListener(this);

}

/** Process mouse click */

```
public void mouseClicked(MouseEvent e) {
```

```
System.out.println("mouse click at x = " + e.getX( ) + " y = " e.getY( ));
```

```
}
```

· Also must implement the other events in MouseListener (if not Mouser is abstract)

Example: Draw Button
Idea: add a button to the graphical view of the polygon application to redraw a polygon
First, in the MainClass, add a JButton

Button/View Layout

• In the buildGUI method

// place a button at the bottom of the frame JButton draw = new JButton("Draw polygon"); JPanel southPanel = new JPanel(); southPanel.add(draw); southPanel.setBackground(Color.WHITE); frame.add(southPanel, BorderLayout.SOUTH);

9

11

Handling Button Clicks

• Who should handle the draw button clicks?

- Not the PolygonModel object shouldn't know about views
- But need to catch the event and then call methods in the PolygonModel to carry out the pause/resume
- One solution: create a listener object
- New class: PolygonController
- Code in MainClass
 PolygonController controller= new PolygonController(model);
 draw.addActionListener(controller);

10

Listener Object public class PolygonController implements ActionListener {

// instance variables
PolygonModel model; // the model we are controlling
/** Constructor for objects of class PolygonController */
public PolygonController(PolygonModel model) {
 this.model = model;
 }
/** Process button clicks by creating a new polygon*/
public void actionPerformed(ActionEvent e) {
 ??2

}

<section-header><list-item><list-item><list-item><list-item><list-item><list-item>

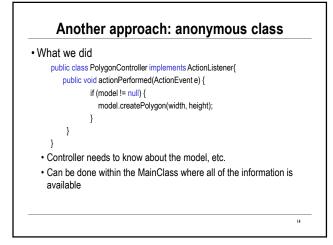
Threads and The AWT Event Thread

- Java supports "threads": apparently concurrently executing streams of instructions.
- · User programs have at least one thread running
 - · Not hard to create additional threads
 - · Can be tricky to coordinate multiple threads
- The Java system has several threads running all the time

13

15

- One important system thread: the AWT event dispatcher
- All AWT/Swing event handlers execute in this thread
- Consequence: your event handlers may be running simultaneously with your application code



Towards a Solution: Inner Classes

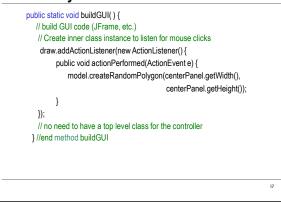
- Java 1.1 and later allows classes to be nested
- · Inner classes define a new scope nested in the containing class
- · Inner classes can access instances variables and methods of the containing class
- Inner classes can be public, protected, or private
- Example: Point2D
 - has two inner classes, named Float and Double
 - Are public, so can be used outside of class Point2D, as Point2D.Float and Point2D.Double
- Inner classes in event handling
 - A class like class PolygonController implements ActionListener{...} can be a private inner class: is only needed once, and only inside the containing class

Solution: Anonymous Inner Classes

- For the action listener, all we need to do is create one instance of the inner class and add it as a mouse listener
 - Doesn't really need a name(!)
 - Solution: create one instance of an **anonymous** inner class
- Warning!!! Ghastly syntax ahead. Here's how to create a new object of an anonymous inner class
- new <classname> (<constructor parameters>) { <method overrides>
 }

16

Anonymous class for the controller



Summary

- Event-driven programming
- Event objects
- Event listeners anything that implements the relevant interface
 Must register with object generating events as a listener
- Listener objects handle events by passing them along to other objects
- Event adapter classes implementations of event interfaces with empty methods
- Extend and override only what you want
- Commonly used to create instances of anonymous inner classes that listen for events

18