


Core Java Course
Java 1.1 Event Model:
Individual Events


Prof. Jeff Blessing
MSOE



Agenda

- ❖ Homework
- ❖ Focus events
- ❖ Window events
- ❖ Keyboard events
- ❖ Mouse events
- ❖ Menus
- ❖ Event queues
- ❖ Custom events
- ❖ Home assignment

2



Focus Events

- ❖ A component has the focus when it can receive keystrokes
- ❖ Only one component can have focus at a time
- ❖ A component loses focus when another component gains focus
- ❖ Focus can be gained by clicking inside the component
- ❖ Tab key moves focus among all the components
- ❖ Focus Events are useful for error checking

3

Focus Event Handlers

- ❖ Implement interface `FocusListener`
- ❖ The interface includes methods


```
void focusGained(FocusEvent e)
void focusLost(FocusEvent e)
```
- ❖ `FocusEvent` class has an access method


```
boolean isTemporary()
```
- ❖ `FocusEvent` class is derived from `ComponentEvent`, which class has an access method:


```
Component getComponent()
```

4

Window Events

- ❖ A window is active when it can receive keystrokes (has a highlighted title bar)
- ❖ Only one window can be active at a time
- ❖ A window is iconified when it is minimized to an icon
- ❖ Example: animation in a window makes sense only while window is not minimized

5

Window Event Handlers

- ❖ Implement interface `WindowListener`
- ❖ The interface includes methods


```
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)
```
- ❖ `WindowEvent` class has an access method


```
Window getWindow()
```

6

Keyboard Events

- ❖ Events handled:
 - A key is pressed
 - A key is released
 - A key is typed (combination of the two above)
- ❖ Distinction between Virtual Key Codes of keys pressed/released and Characters typed
- ❖ Virtual key codes identify a key on the keyboard (defined in `KeyEvent` class)
 - ◆ `VK_A...VK_Z, VK_0...VK_9, VK_COMMA,...`

7

Keyboard Events (cont.)

- ❖ Characters identify actual characters typed
 - ◆ `A...Z, a...z, 0...9, Comma, ...`
- ❖ Example: *a* typed means the same as
 - ◆ `VK_A` pressed
 - ◆ `VK_A` released
- ❖ Example: *A* typed means the same as
 - ◆ `VK_SHIFT` pressed
 - ◆ `VK_A` pressed
 - ◆ `VK_A` released
 - ◆ `VK_SHIFT` released

8

KeyEvent Handlers

- ❖ Implement interface `KeyListener`
- ❖ The interface includes methods


```
void keyPressed(KeyEvent e)
void keyReleased(KeyEvent e)
void keyTyped(KeyEvent e)
```
- ❖ `KeyEvent` class has access methods


```
boolean isActionKey()
int getKeyCode()
char getKeyChar()
static String getKeyText(int keyCode)
static String getModifiersText(int modif)
```

9

Key Event Handlers (cont.)

- ❖ `KeyEvent` class inherits from `InputEvent` class that has the following access methods (applicable to Mouse Events also):


```
int getModifiers() (returns a bitmask)
boolean isAltDown()
boolean isControlDown()
boolean isMetaDown()
boolean isShiftDown()
```

10

Mouse Events

- ❖ Not AWT button clicks or menu selections (handled by components themselves)
- ❖ Mouse events are needed for more complicated situations like mouse drawing, changing cursors
- ❖ Discreet mouse events include
 - Pressing/releasing a mouse button
 - Clicking a mouse button (combination of the two above)
 - Entering/leaving a component

11

Mouse Events (cont.)

- ❖ Motion mouse events include
 - Moving mouse
 - Dragging mouse

12

Mouse Event Handlers

- ❖ Implement one or two interfaces
MouseListener and
MouseMotionListener
- ❖ MouseListener interface includes methods


```
void mousePressed(MouseEvent e)
void mouseReleased(MouseEvent e)
void mouseEntered(MouseEvent e)
void mouseExited(MouseEvent e)
void mouseClicked(MouseEvent e)
```

13

Mouse Event Handlers (cont.)

- ❖ MouseListener interface includes methods


```
void mouseMoved(MouseEvent e)
void mouseDragged(MouseEvent e)
```
- ❖ MouseEvent class has access methods


```
Point getClickPoint()
int getX()
int getY()
int getClickCount() (single, double, triple...)
boolean isPopupTrigger() (left button in Windows??)
```

14

Mouse Test Example

- ❖ Draw a square on a click
- ❖ Erase on a double-click
- ❖ Move a square when dragged
- ❖ Change mouse cursor when it is inside a square
- ❖ Cursor shapes:
 - ◆ DEFAULT_CURSOR (an arrowhead)
 - ◆ CROSSHAIR_CURSOR
 - ◆ HAND_CURSOR
 - ◆ TEXT_CURSOR
 - ◆ WAIT_CURSOR (an hourglass)

15

AWT Menus

- ❖ A Menu consists of
 - Menu Bar (class `MenuBar`)
 - Pull-down Menu (class `Menu`)
 - Menu Items (class `MenuItem`)
 - Submenus (class `MenuItem`)

16

Menu and MenuItem classes

- ❖ Class `Menu`
 - `Menu(String label)`
 - `void addItem(MenuItem item)`
 - `void addSeparator()`
 - `void insert(String label, int index)`
 - `void insert(MenuItem menu, int index)`
 - `void insertSeparator(int index)`
 - `void remove(int index)`
- ❖ Class `MenuItem`
 - `MenuItem(String label)`

17

Advanced Menus

- ❖ Check Box Menu items
 - Use `CheckBoxMenuItem` class
 - Implement `ItemListener` interface
 - Method `itemStateChanged()` is called when the menu item is checked/unchecked
- ❖ Popup menus (floating menus)
 - Use `PopupMenu` class
 - Add menu items and listeners as usually
 - Call `popupMenu.show(component, x, y)` to display the popup menu

18

Menu Test Example

- ❖ makeMenu() method is used to avoid the following tedious menu creation process

```

Menu m = new Menu("Edit");
MenuItem mi = new MenuItem("Undo");
m.add(mi);
mi.addActionListener(this);
mi = new MenuItem("Redo");
m.add(mi);
mi.addActionListener(this);
. . . . .
menuBar.add(m);

```

19

Advanced Event Handling

- ❖ Multicasting - registering a single event source with multiple listeners
- ❖ Consuming Events
 - Consuming is used to intercept an event before it is passed on to proper listeners
 - For the component and all the registered listeners it looks as if the event never occurred
 - In order to trap an event, add event listener to the event **source** component itself and after desirable action call `consume()` method for `AWTEvent` class to stop further propagation of the event

20

Event Queue

- ❖ Event Queue is a series of AWT events converted by AWT from operating system level events and placed into a queue
- ❖ AWT also
 - Fetches events from Event Queue
 - Locates the listener object(s) for an event
 - Calls the appropriate listener procedure for that event
- ❖ For performance reasons some events in the queue can be combined or replaced with a newer event. This is applicable to move and paint events

21

Event Queue API

- ❖ `AWTEvent peekEvent()`
 - returns a reference to the next event
- ❖ `AWTEvent getNextEvent()`
 - same as `peekEvent()` but also removes the event from the queue
- ❖ `void postEvent(AWTEvent evt)`
 - places a new event into the queue

22

Custom Events

- ❖ In order to use custom events, a new event class (inherited from `AWTEvent`) needs to be created
- ❖ Objects of the custom event class have to be inserted into the Event Queue by the custom event source
- ❖ Also, a `processEvent(AWTEvent evt)` method needs to be implemented in the custom event source to identify and dispatch custom events
- ❖ Custom Event Example

23

Homework Assignment

- ❖ Create a menu driven `ShapeBox` application
- ❖ Menus:
 - Shape
 - ♦ Circle, Rectangle, Square, Triangle, Exit
 - Move
 - ♦ Left, Right, Up, Down
 - Scale
 - ♦ Inflate, Deflate
- ❖ Advanced: create a popup Move/Scale menu

24

