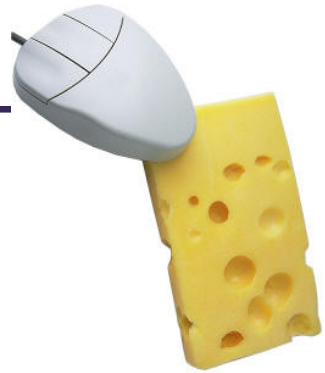
CSE 331

More Events

(Mouse, Keyboard, Window,
Focus, Change, Document ...)

slides created by Marty Stepp
based on materials by M. Ernst, S. Reges, D. Notkin, R. Mercer, Wikipedia
http://www.cs.washington.edu/331/

Mouse events



- Usage of mouse events:
  - listen to clicks / movement of mouse in a component
  - respond to mouse activity with appropriate actions
  - create interactive programs that are driven by mouse activity

- Usage of keyboard events:
  - listen and respond to keyboard activity within a GUI component
  - control onscreen drawn characters and simulate text input

- Key and mouse events are called **low-level events** because they are close to the hardware interactions the user performs.

MouseListener interface

```
public interface MouseListener {
    public void mouseClicked(MouseEvent event);
    public void mouseEntered(MouseEvent event);
    public void mouseExited(MouseEvent event);
    public void mousePressed(MouseEvent event);
    public void mouseReleased(MouseEvent event);
}
```

- Many AWT/Swing components have this method:
  - `public void addMouseListener(MouseListener ml)`

- Mouse listeners are often added to custom components and drawing canvases to respond to clicks and other mouse actions.

Implementing listener

```
public class MyMouseListener implements MouseListener {
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event)  {}

    public void mousePressed(MouseEvent event) {
        System.out.println("You pressed the button!");
    }

    public void mouseReleased(MouseEvent event) {}
}

// elsewhere,
myComponent.addMouseListener(new MyMouseListener());
```

- Problem: Tedious to implement entire interface when only partial behavior is wanted / needed.

Pattern: Adapter

an object that fits another object into a given interface

Adapter pattern

- *Problem:* We have an object that contains the functionality we need, but not in the way we want to use it.
  - Cumbersome / unpleasant to use. Prone to bugs.
- *Example:*
  - We want to write one or two mouse input methods.
  - Java makes us implement an interface full of methods we don't want.
- *Solution:*
  - Provide an adapter class that connects into the setup we must talk to (GUI components) but exposes to us the interface we prefer (only have to write one or two methods).

Event adapters

- **event adapter**: A class with empty implementations of all of a given listener interface's methods.
  - examples: `MouseAdapter`, `KeyAdapter`, `FocusAdapter`
  - Extend `MouseAdapter`; override methods you want to implement.
    - Don't have to type in empty methods for the ones you don't want!
  - an example of the **Adapter** design pattern
  - Why is there no `ActionAdapter` for `ActionListener`?

An abstract event adapter

```
// This class exists in package java.awt.event.

// An empty implementation of all MouseListener methods.
public abstract class MouseAdapter implements MouseListener {
    public void mousePressed(MouseEvent event) {}
    public void mouseReleased(MouseEvent event) {}
    public void mouseClicked(MouseEvent event) {}
    public void mouseEntered(MouseEvent event) {}
    public void mouseExited(MouseEvent event) {}
}
```

- Now classes can extend `MouseAdapter` rather than implementing `MouseListener`.
  - client gets the complete mouse listener interface it wants
  - implementer gets to write just the few mouse methods they want
  - Why did Sun include `MouseListener`? Why not just the adapter?

Abstract classes

- **abstract class**: A hybrid between an interface and a class.
  - Defines a superclass type that can contain method declarations (like an interface) and/or method bodies (like a class).
  - Like interfaces, abstract classes that cannot be instantiated (cannot use `new` to create any objects of their type).

- What goes in an abstract class?
  - Implementation of common state and behavior that will be inherited by subclasses (parent class role)
  - Declare generic behavior for subclasses to implement (interface role)

Abstract class syntax

```
// declaring an abstract class
public abstract class name {
    ...

    // declaring an abstract method
    // (any subclass must implement it)
    public abstract type name(parameters);

}
```

- A class can be `abstract` even if it has no abstract methods
- You can create variables (but not objects) of the abstract type

Writing an adapter

```
public class MyMouseAdapter extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
        System.out.println("You pressed the button!");
    }
}

// elsewhere,
myComponent.addMouseListener(new MyMouseAdapter());
```

Abstract class vs. interface

- Why do both interfaces and abstract classes exist in Java?
  - An abstract class can do everything an interface can do and more.
  - So why would someone ever use an interface?

- Answer: Java has only single inheritance.
  - can extend only one superclass
  - can implement many interfaces
  - Having interfaces allows a class to be part of a hierarchy (polymorphism) without using up its inheritance relationship.

MouseEvent properties

- **MouseEvent**
  - public static int BUTTON1_MASK, BUTTON2_MASK, BUTTON3_MASK, CTRL_MASK, ALT_MASK, SHIFT_MASK
  - public int getClickCount()
  - public Point getPoint()
  - public int **getX**(), **getY**()
  - public Object **getSource**()
  - public int getModifiers() // use *_MASK with this

- SwingUtilities
  - static void isLeftMouseButton(MouseEvent event)
  - static void isRightMouseButton(MouseEvent event)

Using MouseEvent

```
public class MyMouseAdapter extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
        Object source = event.getSource();
        if (source == button && event.getX() < 10) {
            JOptionPane.showMessageDialog(null,
                "You clicked the left edge!");
        }
    }
}
```

Mouse motion

```
public interface MouseMotionListener {
    public void mouseDragged(MouseEvent event);
    public void mouseMoved(MouseEvent event);
}
```

- For some reason Java separates mouse input into many interfaces.
  - The second focuses on the movement of the mouse cursor.

- Many AWT/Swing components have this method:
  - `public void addMouseMotionListener(MouseMotionListener ml)`

- The abstract `MouseMotionAdapter` class provides empty implementations of both methods if you just want to override one.

Mouse wheel scrolling

```
public interface MouseWheelListener {
    public void mouseWheelMoved(MouseWheelEvent event);
}
```

- The third mouse event interface focuses on the rolling of the mouse's scroll wheel button.

- Many AWT/Swing components have this method:
  - `public void addMouseWheelListener(MouseWheelListener ml)`

Mouse input listener

```
// import javax.swing.event.*;
public interface MouseInputListener
    extends MouseListener, MouseMotionListener,
            MouseWheelListener {}
```

- The `MouseInputListener` interface combines all kinds of mouse input into a single interface that you can implement.

- The `MouseInputAdapter` class includes empty implementations for all methods from all mouse input interfaces, allowing the same listener object to listen to mouse clicks, movement, and/or wheel events.

Mouse input example

```
public class MyMouseInputAdapter extends MouseInputAdapter {
    public void mousePressed(MouseEvent event) {
        System.out.println("Mouse was pressed");
    }

    public void mouseDragged(MouseEvent event) {
        Point p = event.getPoint();
        System.out.println("Mouse is at " + p);
    }
}

...

// using the listener
MyMouseInputAdapter adapter = new MyMouseInputAdapter();
myPanel.addMouseListener(adapter);
myPanel.addMouseMotionListener(adapter);
```

Keyboard Events

KeyListener interface

```
public interface KeyListener {
    public void keyPressed(KeyEvent event);
    public void keyReleased(KeyEvent event);
    public void keyTyped(KeyEvent event);
}
```

- Many AWT/Swing components have this method:
  - `public void addKeyListener(KeyListener kl)`

- The abstract class `KeyAdapter` implements all `KeyListener` methods with empty bodies.

KeyEvent objects

- `// what key code was pressed?`
  `public static int VK_A, VK_B, ..., VK_Z,`
  `VK_0, ... VK_9,`
  `VK_F1, ... VK_F10,`
  `VK_UP, VK_LEFT, ...,`
  `VK_TAB, VK_SPACE, VK_ENTER, ...`
  (one for almost every key)
- `// Were any modifier keys held down?`
  `public static int CTRL_MASK, ALT_MASK, SHIFT_MASK`
- `public char` **`getKeyChar`**`()`
- `public int` **`getKeyCode`**`()` `// use VK_* with this`
- `public Object` **`getSource`**`()`
- `public int` **`getModifiers`**`()` `// use *_MASK with this`

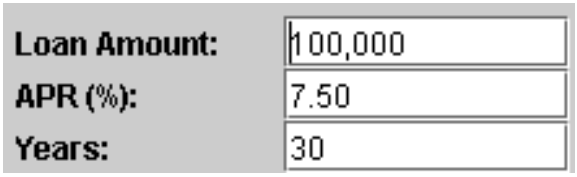
Key event example

```
public class PacManKeyListener extends KeyAdapter {
    public void keyPressed(KeyEvent event) {
        char keyChar = event.getKeyChar();
        int keyCode  = event.getKeyCode();

        if (keyCode == KeyEvent.VK_RIGHT) {
            pacman.setX(pacman.getX() + 1);
        } else if (keyChar == 'Q') {
            quit();
        }
    }
}

// elsewhere,
myJFrame.addKeyListener(new PacKeyListener());
```

Other Kinds of Events

Focus events

```
public interface FocusListener {
    public void focusGained(FocusEvent event);
    public void focusLost(FocusEvent event);
}
```



- **focus**: The current target of keyboard input.
  - A *focus event* occurs when the keyboard cursor enters or exits a component, signifying the start/end of typing on that control.
- Many AWT/Swing components have this method:
  - `public void addFocusListener(FocusListener kl)`
- The abstract class `FocusAdapter` implements all `FocusListener` methods with empty bodies.

Window events

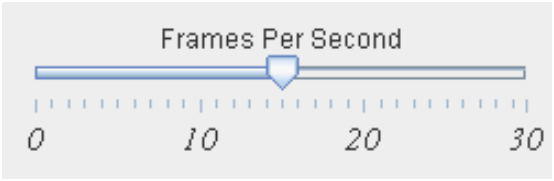
```
public interface WindowListener {
    public void windowActivated(WindowEvent event);
    public void windowClosed(WindowEvent event);
    public void windowClosing(WindowEvent event);
    public void windowDeactivated(WindowEvent event);
    public void windowDeiconified(WindowEvent event);
    public void windowIconified(WindowEvent event);
    public void windowOpened(WindowEvent event);
}
```

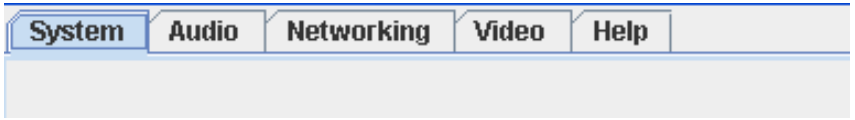
- A `JFrame` object has this method:
  - `public void addWindowListener(WindowListener wl)`
- The abstract class `WindowAdapter` implements all `WindowListener` methods with empty bodies.

Change events

```
public interface ChangeListener {
    public void stateChanged(ChangeEvent event);
}
```

- These events occur when some kind of state of a given component changes. Not used by all components, but essential for some.
- `JSpinner`, `JSlider`, `JTabbedPane`, `JColorChooser`, `JViewPort`, and other components have this method:
  - `public void addChangeListener(ChangeListener cl)`





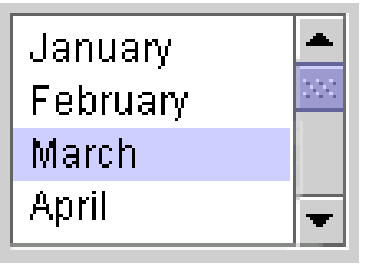
Component events

```
public interface ComponentListener {
    public void componentHidden(ComponentEvent event);
    public void componentMoved(ComponentEvent event);
    public void componentResized(ComponentEvent event);
    public void componentShown(ComponentEvent event);
}
```

- These events occur when a layout manager reshapes a component or when it is set to be visible or invisible.
- Many AWT/Swing components have this method:
  - `public void addComponentListener(ComponentListener cl)`
- The abstract class `ComponentAdapter` implements all `ComponentListener` methods with empty bodies.

JList/JTree select events

```
public interface ListSelectionListener {
    public void valueChanged(ListSelectionEvent event);
}

public interface TreeSelectionListener {
    public void valueChanged(TreeSelectionEvent event);
}
```



- These events occur when a user changes the element(s) selected within a `JList` or `JTree`.
- The `JList` component has this method:
  - `public void addListSelectionListener( ListSelectionListener lsl)`
- The `JTree` component has this method:
  - `public void addTreeSelectionListener( TreeSelectionListener tsl)`

Document events

```
public interface DocumentListener {
    public void changedUpdate(DocumentEvent event);
    public void insertUpdate(DocumentEvent event);
    public void removeUpdate(DocumentEvent event);
}
```

- These events occur when the contents of a text component (e.g. JTextField or JTextArea) change.
- Such components have this method:
  - `public Document getDocument()`
- And a `Document` object has this method:
  - `public void addDocumentListener(DocumentListener cl)`
- And yes, there is a `DocumentAdapter`.