Inheritance & Interfaces

Beat the Bugs!
Inheritance & Interfaces
Inheritance & Interfaces
The Plan

- Motivate Inheritance
- Motivate Interfaces
- Examples
- Continue Practice on Observer/Observable
Inheritance Motivation

Inheritance in Java is achieved through extending classes

Inheritance enables:

- Code re-use
- Grouping similar code
- Flexibility to customize
Interface Motivation

• Interfaces in Java are achieved by implementing interfaces

Benefits of interfaces

• Can specify functionality without specifying implementation details
• Helps separate code for more local modifications
• Allows incremental development
Motivation

The real motivation is that inheritance and interfaces were used extensively in developing the video gaming package. Without these two, the code would be much more difficult to modify, extend, and use.
Examples from the Video Game Package

- Keyboard
- Mouse
- GameLoop
- Tracker
- AttractorTracker
- Alarm
- TimeSpriteKiller
- BlurSprite
From the Java API Documentation

KeyListener
Interface KeyListener

All Known Implementing Classes:
AWTEventMulticaster, BasicTableUI, KeyHandler, KeyAdapter

public interface KeyListener
extends EventListener

The listener interface for receiving keyboard events (keystrokes). The class that is interested in processing a keyboard event either implements this interface (and all the methods it contains) or extends the abstract KeyAdapter class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's addKeyListener method. A keyboard event is generated when a key is pressed, released, or typed (pressed and released). The relevant method in the listener object is then invoked, and the KeyEvent is passed to it.

Since: 1.1

See Also:
AWTEvent, TreeFocusListener, KeyListener, Reference: The Java Class Libraries (update file)

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyPressed</td>
<td>Invoked when a key has been pressed.</td>
</tr>
<tr>
<td>keyReleased</td>
<td>Invoked when a key has been released.</td>
</tr>
<tr>
<td>keyTyped</td>
<td>Invoked when a key has been typed.</td>
</tr>
</tbody>
</table>
package tipgame;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

/**
 * @author Jam Jenkins
 */
public class Keyboard implements KeyListener
{
    private char key;
    private boolean keyDown;

    /** Creates a new instance of Keyboard */
    public Keyboard()
    {
        clear();
    }

    public void clear()
    {
        keyDown=false;
        key=KeyEvent.CHAR_UNDEFINED;
    }

    public char consumeKey()
    {
        char temp=key;
        key=KeyEvent.CHAR_UNDEFINED;
        return temp;
    }

    public char getLastKey()
    {
        return key;
    }

    public void keyPressed(KeyEvent e)
    {
        key=e.getKeyChar();
        keyDown=true;
    }

    public void keyReleased(KeyEvent e)
    {
        key=e.getKeyChar();
        keyDown=false;
    }

    public void keyTyped(KeyEvent e)
    {
    }
}
From the Java API Documentation

MouseListener
public interface MouseListener
extends EventListener

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component. (To track mouse moves and mouse drags, use the MouseMotionListener.)

The class that is interested in processing a mouse event either implements this interface (and all the methods it contains) or extends the abstract MouseAdapter class (overriding only the methods of interest).

The listener object created from that class is then registered with a component using the component's addMouseListener method. A mouse event is generated when the mouse is pressed, released clicked (pressed and released). A mouse event is also generated when the mouse cursor enters or leaves a component. When a mouse event occurs, the relevant method in the listener object is invoked, and the MouseEvent is passed to it.

Since:
1.1
See Also:
MouseListener, MouseEvent, Tutorial: Writing a Mouse Listener, Reference: The Java Class Libraries (JDK 1.1)

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mouseClicked</td>
<td>Invoked when the mouse button has been clicked (pressed and released) on a component.</td>
</tr>
<tr>
<td>mouseEntered</td>
<td>Invoked when the mouse enters a component.</td>
</tr>
<tr>
<td>mouseExited</td>
<td>Invoked when the mouse exits a component.</td>
</tr>
<tr>
<td>mousePressed</td>
<td>Invoked when a mouse button has been pressed on a component.</td>
</tr>
<tr>
<td>mouseReleased</td>
<td>Invoked when a mouse button has been released on a component.</td>
</tr>
</tbody>
</table>

Method Detail

mouseClicked
From the Java API Documentation

MouseMoveMotionListener
All Known Implementing Classes:
AWTEventMultimatcher, BasicButtonListener, BasicInternalFrameUI, GlassPaneDispatche
BasicClassNotFoundException, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler, BasicMenuUI, MouseInputHandler,
package tipgame;

import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseMotionListener;

/**
 * This class uses polling as opposed to event based methods for determining
 * mouse positions and actions.
 *
 * @author Jam Jenkins
 */

public class Mouse implements MouseListener, MouseMotionListener {

    private Point mousePosition;
    private Point mouseClicked;
    private boolean mouseDown;

    /** Creates a new instance of Mouse */
    public Mouse()
    {
        mouseDown=false;
    }

    public void clear()
    {
        mousePosition=null;
        mouseClicked=null;
        mouseDown=false;
    }
}
Mouse.java

/**@return the last position of the mouse*/
public Point getMousePosition()
{
    return mousePosition;
}

/**determines the last position of a click and once called, clears that click
@return the last clicked position, null if not clicked since last call*/
public Point getClickPosition()
{
    Point toReturn=mouseClick;
    mouseClick=null;
    return toReturn;
}

/**@return true if the mouse is currently down, false otherwise*/
public boolean mousePressed()
{
    return mouseDown;
}
public void mouseClicked(java.awt.event.MouseEvent mouseEvent) {
    mouseClick=mouseEvent.getPoint();
}

public void mouseDragged(java.awt.event.MouseEvent mouseEvent) {
    mousePosition=mouseEvent.getPoint();
}

public void mouseEntered(java.awt.event.MouseEvent mouseEvent) {
}

public void mouseExited(java.awt.event.MouseEvent mouseEvent) {
}

public void mouseMoved(java.awt.event.MouseEvent mouseEvent) {
    mousePosition=mouseEvent.getPoint();
}

public void mousePressed(java.awt.event.MouseEvent mouseEvent) {
    mouseDown=true;
}

public void mouseReleased(java.awt.event.MouseEvent mouseEvent) {
    mouseDown=false;
}
GameLoop

• FrameAdvancer animates the AnimationCanvas.
• GameLoop extends FrameAdvancer by adding a Keyboard and a Mouse as Listeners to the AnimationCanvas.
• GameLoop can respond to user interaction while Frame Advancer does not.
public class GameLoop extends FrameAdvance
{
    private JPanel canvasPanel;
    protected Mouse mouse;
    protected Keyboard keyboard;

    public GameLoop()
    {
        super();
        mouse = new Mouse();
        keyboard = new Keyboard();
        canvas.addMouseListener(mouse);
        canvas.addMouseMotionListener(mouse);
        canvas.addKeyListener(keyboard);
        canvasPanel = new JPanel(new BorderLayout());
        canvasPanel.add(canvas);
    }

    /** used to put the canvas into a GUI */
    public AnimationCanvas getCanvas()
    {
        return canvas;
    }

    public void start()
    {
        keyboard.clear();
        mouse.clear();
        super.start();
    }
}
Tracker

• Used with Sprites
• Describes general motion
  – location
  – size
  – orientation
package tipgame;
import java.awt.geom.Point2D;
/**
 * @author Jam
 */
public interface Tracker
{
    Point2D.Double getLocation();
    double getScaleFactor();
    double getRotationAddition();
    void advanceTime();
}
TrackerAdapter

• You don't want to write all the methods needed to implement an interface.
• The default implementations of the methods in interfaces exist.

The Solution:
• Extend an Adapter
public abstract class TrackerAdapter implements Tracker {

    public Point2D.Double getLocation() {
        return new Point2D.Double();
    }

    public double getScaleFactor() {
        return 1;
    }

    public double getRotationAddition() {
        return 0;
    }

    public void advanceTime() {
        advanceTime(GameLoop.timeInterval);
    }

    public abstract void advanceTime(double time);
}
public class AttractorTracker extends TrackerAdapter {
    double speed;
    Sprite moving;
    Sprite toward;
    Point2D.Double nextLocation;

    public AttractorTracker(Sprite from, Sprite to,
                           double rate)
    {
        moving=from;
        toward=to;
        speed=rate;
        nextLocation=new Point2D.Double();
    }

    public Sprite getTarget()
    {
        return toward;
    }
}
public Point2D.Double getLocation() {
    return nextLocation;
}

public void advanceTime(double time) {
    Point2D.Double from=moving.getLocation();
    Point2D.Double to=toward.getLocation();
    nextLocation.x=to.x-from.x;
    nextLocation.y=to.y-from.y;
    double factor=speed*time/from.distance(to);
    nextLocation.x*=factor;
    nextLocation.y*=factor;
    nextLocation.x+=from.x;
    nextLocation.y+=from.y;
}

Alarm

Used to signal timing events

Used with FrameAdvancer
public interface Alarm
{
    public void alarm();
}
TimedKillSprite

• Used to get rid of a Sprite after a given period of time.
• Can be used for splash screens
• Has other uses as well
public class TimedSpriteKiller implements Alarm
{
    Sprite sprite;

    public TimedSpriteKiller(Sprite s)
    {
        sprite = s;
    }

    public void setKillTime(double delay)
    {
        GameLoop.scheduleRelative(this, delay);
    }

    public void alarm()
    {
        sprite.kill();
    }
}
BlurSprite
BlurSprite

• Extends Sprite
• Draws history of Sprite motion
• Can augment any Sprite using a Shape (i.e. any Sprite except ImageSprite)
public class BlurSprite extends Sprite
{
    private LinkedList previous=new LinkedList();
    private static int DEFAULT_PER_FRAME=3;
    private static int DEFAULT_FRAMES=2;
    private int numFrames=DEFAULT_FRAMES;
    private int numPerFrame=DEFAULT_PER_FRAME;
    private int drawNumber=0;

    public void setEnabled(boolean enabled)
    {
        super.setEnabled(enabled);
        if(enabled==true)
        {
            previous.clear();
        }
    }

    public void setHistory(int frames, int perFrame)
    {
        numFrames=frames;
        numPerFrame=perFrame;
    }
}
public void update()
{
    if(!isEnabled())
        return;
    super.update();
drawNumber++;
    int frequency=Math.max(1, FrameAdvancer.getUpdatesPerFrame()/numPerFrame);
    if(drawNumber%frequency!=0)
    {
        return;
    }
    boolean bounding=getUseBoundingBox();
    setUseBoundingBox(false);
    previous.addLast(getShape());
    setUseBoundingBox(bounding);
    if(previous.size()>numFrames*numPerFrame)
    {
        previous.removeFirst();
    }
}
public void paint(Graphics2D brush)
{
    if (!isEnabled())
        return;
    brush.setColor(getColor());
    PathBlur.paintExponentialBlur(brush,
        (Shape[]) previous.toArray(new Shape[0]), 2);
}
Practice (from GUIs lecture)

Write a program to count the number of clicks on a yes, no and maybe button. To do this, write three classes:

• ClickCount – keeps three integer instance variables to count the number of clicks (hint: extend Observable)
• ClickCountPanel – observes ClickCount for changes and updates its components when an update occurs (hint: implement Observer)
• ClickGUI – contains three buttons and the count panel. Clicking on the buttons registers the click via ClickCount.

Look at sample programs from previous lecture (the GUIs lecture and GUIs for Video Games) to get you started.