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

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) {
        key=e.getKeyChar();
    }
}

From the Java API Documentation

MouseListener
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 Jan Jenkins */
public class Mouse implements MouseListener, MouseMotionListener {
    private Point mousePosition;
    private Point mouseClick;
    private boolean mouseDown;
    /** Creates a new instance of Mouse */
    public Mouse() {
        mouseDown = false;
    }
    public void clear() {
        mousePosition = null;
        mouseClick = null;
        mouseDown = false;
    }
}
Mouse.java

```java
public class Mouse {
    // return the last position of the mouse
    public Point getMousePosition() {
        return mousePosition;
    }

    // determines the last position of a click and once called, clears the click
    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(MouseEvent mouseEvent) {
        mouseClick = mouseEvent.getPoint();
    }

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

    public void mouseEntered(MouseEvent mouseEvent) {
    }

    public void mouseExited(MouseEvent mouseEvent) {
    }

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

    public void mousePressed(MouseEvent mouseEvent) {
        mouseDown = true;
    }

    public void mouseReleased(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.
Tracker

- Used with Sprites
- Describes general motion
  - location
  - size
  - orientation

Tracker.java

```java
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

TrackerAdapter.java

```java
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();
}

global static 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

TimedSpriteKiller.java

```java
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

- 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 paint(Graphics2D brush) {
        if (!isEnabled())
            return;
        brush.setColor(getColor());
        PathBlur.paintExponentialBlur(brush,
            (Shape[])previous.toArray(new Shape[0]), 2);
    }
}

public void update() {
    if (!isEnabled())
        return;
    super.update();
    drawNumber++;;
    boolean bounding=getUseBoundingBox();
    setUseBoundingBox(false);
    previous.addLast(getShape());
    if (previous.size()>numFrames*numPerFrame) {
        previous.removeFirst();
    }
}

Practice (from GUls 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.