Java: An Eventful Approach

Chapter 6

* Adapted from Lecture Notes by K.B. Bruce, A.P. Danyluk, and T.P. Murtagh

A Class of Our Own

- We've used many classes:
  - Location
  - FilledRect
  - Color
  - Text
  - And More!
- But what if we want to make our own class?

What if...

We can free ourselves from the limitations of ovals and rectangles and move on to...

A Draggable Face

```java
public class RevFaceDrag extends WindowController {
    private FunnyFace happy; // FunnyFace to be dragged
    private Location lastPoint;
    private boolean happyGrabbed = false;  // Whether happy has been grabbed by the mouse

    public void begin() {  // Make the FunnyFace
        happy = new FunnyFace( FACE_LEFT, FACE_TOP, canvas );
    }

    public void onMousePress( Location point ){
        lastPoint = point;
        happyGrabbed = happy.contains( point );
    }

    public void onMouseDrag( Location point ) {
        if (happyGrabbed ) {
            happy.move( point.getX() – lastPoint.getX(),
                        point.getY() – lastPoint.getY() );
            lastPoint = point;
        }
    }
}
```

Making a Funny Face

- Physical Characteristics:
  - Head
  - Mouth
  - Two Eyes
- Behaviors
  - Can check contains
  - Movable (by dragging)

Components of a Class

- Instance Variables
- Methods
- Constructors (these are new!)
Instance Variables

• The data stored in a class
• Maintain the state of an object
• Instance variables for FunnyFace:

```java
private FramedOval head;
private FramedOval mouth;
private FramedOval leftEye, rightEye;
```

Methods

• Methods: Sections of code executed at certain times:
  – onMousePress
  – onMouseDrag
  – begin
• Special Types of Methods:
  – Mutator Methods
  – Accessor Methods

Actual Parameters

• Information given to a method when the method is invoked

• In the invocation of: happy.move (10, 20);
  – happy is the object
  – move is the name of the method
  – the values 10 and 20 are passed to happy.move

• Order of parameters matters!

A Mutator Method for FunnyFace

//Move the entire face by (dx,dy)
public void move( double dx, double dy ) {
    head.move( dx, dy );
    leftEye.move( dx, dy );
    rightEye.move( dx, dy );
    mouth.move( dx, dy );
}

Formal Parameters

• dx and dy are the formal parameters

• Formal Parameters:
  – used in the method declaration
  – determine the order of actual parameters
  – determine the type of the actual parameters

Accessor Methods

• Return information about an object

// Determine whether pt is inside FunnyFace
public boolean contains( Location pt ) {
    return head.contains( pt );
}
Constructors

- A constructor creates an object and initializes relevant instance variables
- Invoked by the name of the class

```java
public FunnyFace( double left, double top, DrawingCanvas canvas ){
    head = new FramedOval( left, top, FACE_WIDTH, FACE_HEIGHT, 
                          canvas );
    mouth = new FramedOval( left+(FACE_WIDTH-MOUTH_WIDTH)/2, 
                       top+2*FACE_HEIGHT/3, MOUTH_WIDTH,
                       MOUTH_HEIGHT, canvas );
    leftEye = new FramedOval( left+EYE_OFFSET-EYE_RADIUS/2, 
                      top+2*FACE_HEIGHT/3, MOUTH_WIDTH,
                      MOUTH_HEIGHT, canvas );
    rightEye = new FramedOval( left+FACE_WIDTH-EYE_OFFSET-
                      EYE_RADIUS/2, 
                      top+EYE_OFFSET, 
                      EYE_RADIUS, EYE_RADIUS, canvas );
}
```

Class Convention

```java
public class Name {
    constant definitions
    variable declarations
    constructor
    methods
}
```

Putting it all together

Complete source code

```java
public void moveTo( double x, double y ) {
    this.move( x – head.getX(), y – head.getY() );
}
```

Defining Methods Indirectly

- What if we wanted to move the FunnyFace to a specified location?
- We can reuse methods already defined

```java
public void moveTo( double x, double y ) {
    this.move( x – head.getX(), y – head.getY() );
}
```

- We can use “this” to refer to object itself
**What if**

• We want to keep track of the amount of time a *FunnyFace* was dragged

• We need an invisible class that keeps track of time…

**Timer Class**

```java
public class Timer {
    private double startTime; // Time when Timer started or reset

    public Timer() { // Create timer, initializing startTime with current time
        startTime = System.currentTimeMillis();
    }

    public double elapsedMilliseconds() { // Number of milliseconds since last reset
        return System.currentTimeMillis() - startTime;
    }

    public double elapsedSeconds() { // Return number of seconds since last reset
        return this.elapsedMilliseconds() / 1000;
    }

    public void reset() { // Reset startTime
        startTime = System.currentTimeMillis();
    }
}
```

**Using a Timer**

```java
public class TimedFace {
    private Timer stopwatch;

    public void begin() { // Make the *FunnyFace*
        happy = new FunnyFace(FACE_LEFT, FACE_TOP, canvas);
        stopwatch = new Timer();
    }

    public void onMousePress(Location point) {
        lastPoint = point;
        happyGrabbed = head.contains(point);
        stopwatch.reset();
    }

    public void onMouseDrag(Location point) {
        if (happyGrabbed) {
            happy.move(point.getX() - lastPoint.getX(), point.getY() - lastPoint.getY());
            lastPoint = point;
        }
    }

    public void onMouseRelease(Location point) {
        if (happyGrabbed) {
            System.out.println("The Face was dragged for " + stopWatch.elapsedSeconds() + " seconds.");
        }
    }
}
```

**Local Variables**

• Recall variables and their uses:
  – Instance Variables: maintain the state of an object
  – Formal Parameters: Determine the information required by a method

• Local Variables:
  – Declared, initialized and used within a method
  – Locals do not retain their values between method invocations

**Using Local Variables**

```java
public double elapsedMilliseconds() { // Return number of milliseconds since last reset
    double diffTime;
    diffTime = System.currentTimeMillis() - startTime;
    return diffTime;
}
```

• `diffTime` is a local variable
Using Local Variables

• To simplify difficult calculations

• To eliminate redundant computations

Overloading

• Evaluating 5 + 7 gives 12
• Evaluating "Good" + " Day" gives "Good Day"

+ is an overloaded operator: it can add numbers and concatenate strings

• The type of operation is determined by the types of the operands

Overloading II

• In Java, users cannot define new overloaded operations

• We can overload methods!

Overloading III

• Consider:

  public void moveTo( double x, double y ) {
    this.move( x – head.getX(), y – head.getY() );
  }

  And

  public void moveTo( Location pt ) {
    this.move( pt.getX() – head.getX(), pt.getY() – head.getY() );
  }

Overloading IV

• What determines when each moveTo is invoked?

• Answer: The actual parameters

• moveTo can be invoked either with two doubles or with a Location

Review

• Components of a class
  – Instance Variables
  – Constructors
  – Methods

• Formal vs Actual Parameters
• Instance Variables and Local Variables
• Overloading