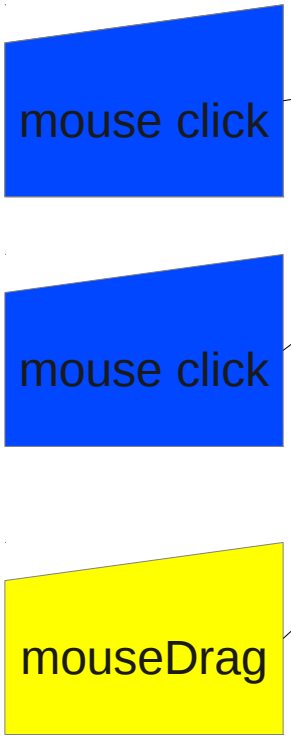
Lecture 9
CSE11 Fall 2013
Active Objects

“Active Objects”

- What is an active object?
  - Objectdraw library has a specialized version of a more general structure
  - Think of these as objects that can continuously execute code
  - They execute independently of one another
- The more general ideal is multiple “threads” of control

What we've done so far

- Single “Thread” of execution with various events
- WindowController Class handles events

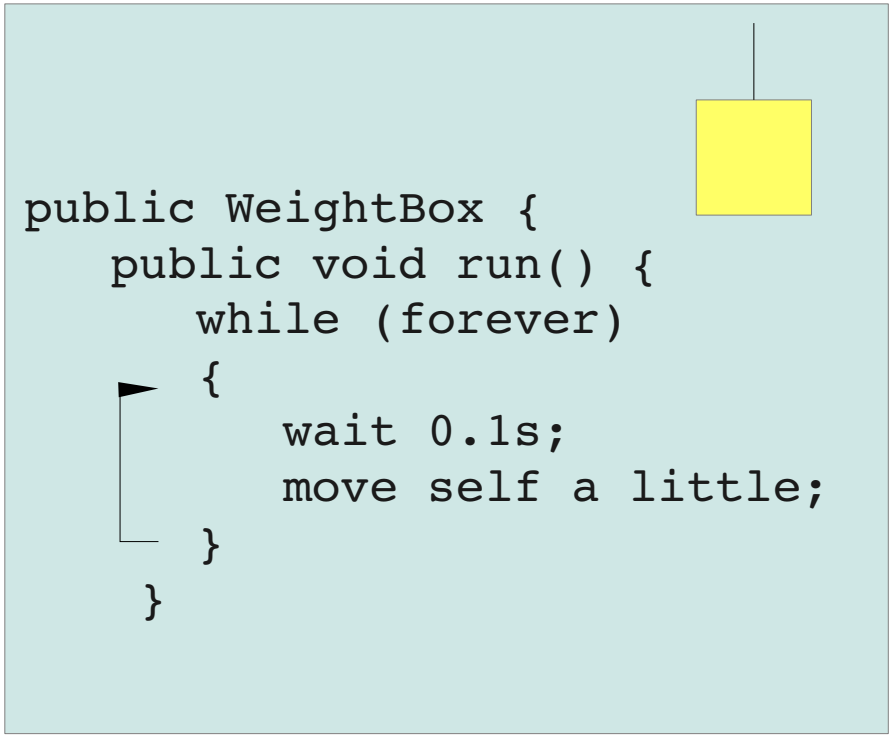


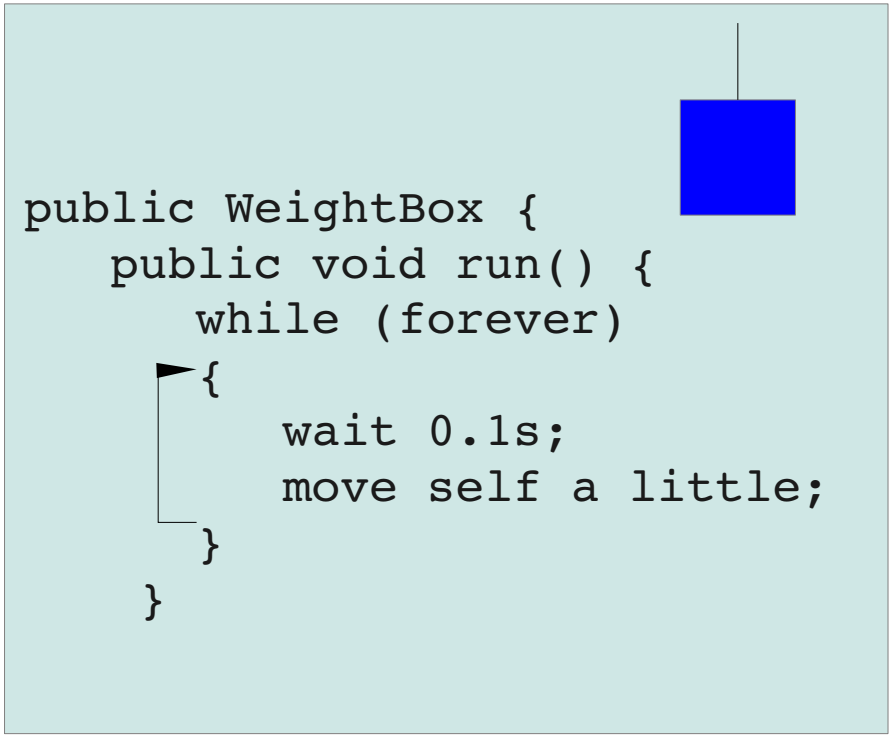

```
public MyClass extends WindowController {

    public void onMouseClick
    {
    }

    public void onMouseDrag
    {
    }
    main(String [] ...)
    {
        java statements;
    }
}
```

Active Objects

- Think about “flip book” animation.
- So far, we've used mouse clicks to cause objects to move
  - e.g. Balanced Pulley Programming Project
- What if the the WeightBox Objects moved “on their own”?
  - That is, under their program control

Define a “Run” method





- Yellow and Blue WeightBox Objects are executing independently
- When in the same program, we call these independent threads of execution

ColorBallController Example

- http://eventfuljava.cs.williams.edu/samplePrograms/ch9
- javac FallingBall.java ColorBallController.java
- java ColorBallController.java

FallingBall Sample Code

```
public class FallingBall extends ActiveObject {
    // the image of the ball
    private FilledOval ballGraphic;
    // the canvas
    private DrawingCanvas canvas;

    public FallingBall(Location initialLocation, DrawingCanvas aCanvas)
        canvas = aCanvas;
        ballGraphic = new FilledOval(initialLocation, SIZE, SIZE, canvas
        start();
    }

    public void run() {
        while (ballGraphic.getY() < canvas.getHeight() ) {
            ballGraphic.move(0, Y_SPEED);
            pause(DELAY_TIME);
        }
        ballGraphic.removeFromCanvas();
    }
 }
```

Basic Recipe of an Active Object

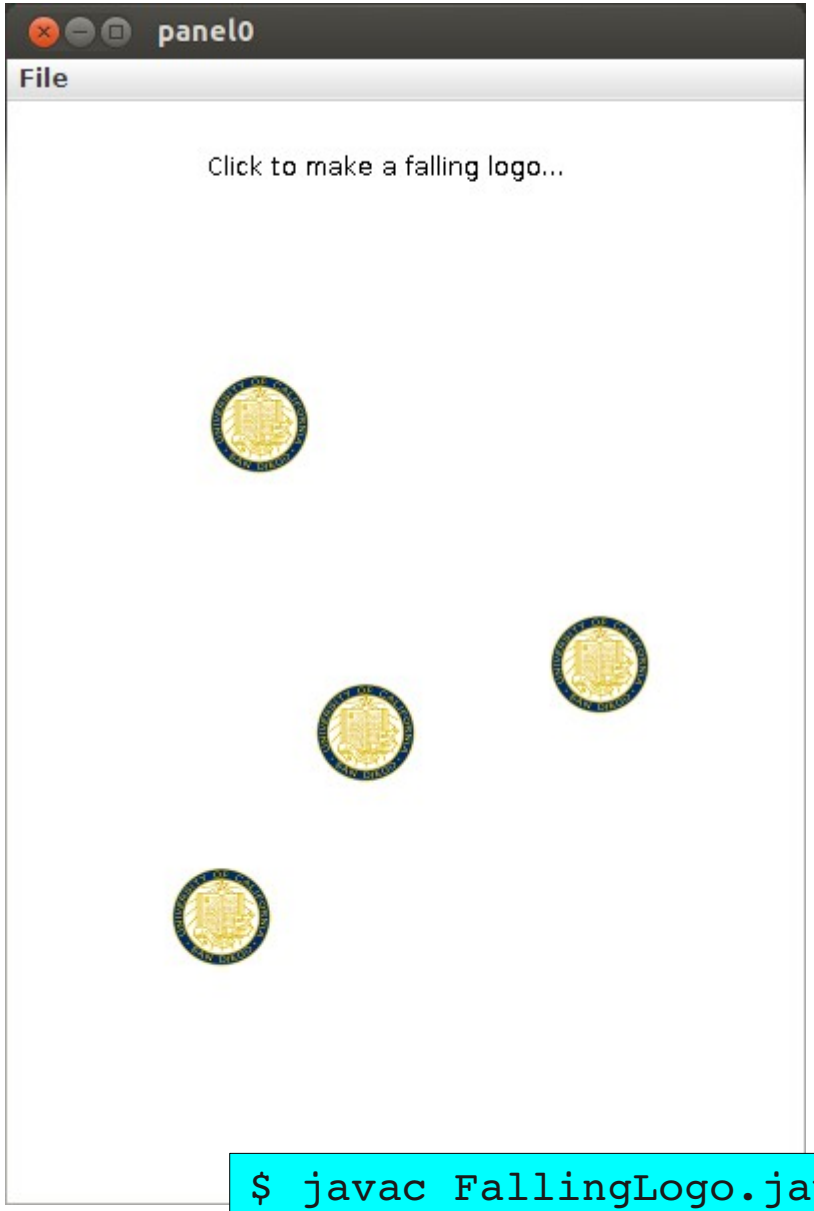
- define a class that `extends ActiveObject`
- include a `start();` as the last statement of the constructor
- define a `run()` method
- make sure to `pause()` during the run method (so that people can see what happens)

Making the Graphics more Realistic

- `Image` – Java's notion of a pixel image
  - use `getImage()` to load/open a pre-defined image and place into a local or instance variable
- `VisibleImage` – like other graphical objects with similar methods to place on a canvas. Requires an Image to be defined.
- In objectdraw, `Controller` and `WindowController` define `getImage()`
  - Call this method in classes that extend these controller classes

See: FallingLogo.java, LogoController.java, 50px-UCSD_Seal.svg.png

Sample Falling UCSD Logos



FallingLogo.java “snippet”

```
// the image of the logo
   private VisibleImage logoGraphic;
   // the canvas
   private DrawingCanvas canvas;

   public FallingLogo(Image logo, Locat
       canvas = aCanvas;
       logoGraphic = new VisibleImage(l
       start();
   }
```

```
$ javac FallingLogo.java LogoController.java
$ java LogoController
```

Thinking about how to get Active Objects to Interact with Other Objects

- Suppose we wanted to count and display the number of Logos that had fallen to or past bottom of the screen
- Two possible approaches
  - Controller knows how many logos have been created (each click creates a new one)
    - Could periodically check how many logos are still visible and subtract #created - #visible
  - Each logo object could tell the controller “I've reached the bottom of the screen”

Evaluating these two approaches

- Controller Knows
  - Pros: FallingLogo only needs to know to do two simple things:
    - How to Fall, Define Accessor method of isHidden()
  - Cons: Controller must do more work
    - Explicitly track all FallingLogo Instances
    - Query them periodically (we haven't yet learned enough java to do this efficiently)
- FallingLogo Reports Back
  - Pros: Simplified Controller, Change reflected as soon as logo hits the bottom of the canvas
  - Cons: FallingLogo needs to know how to do more than “just fall”

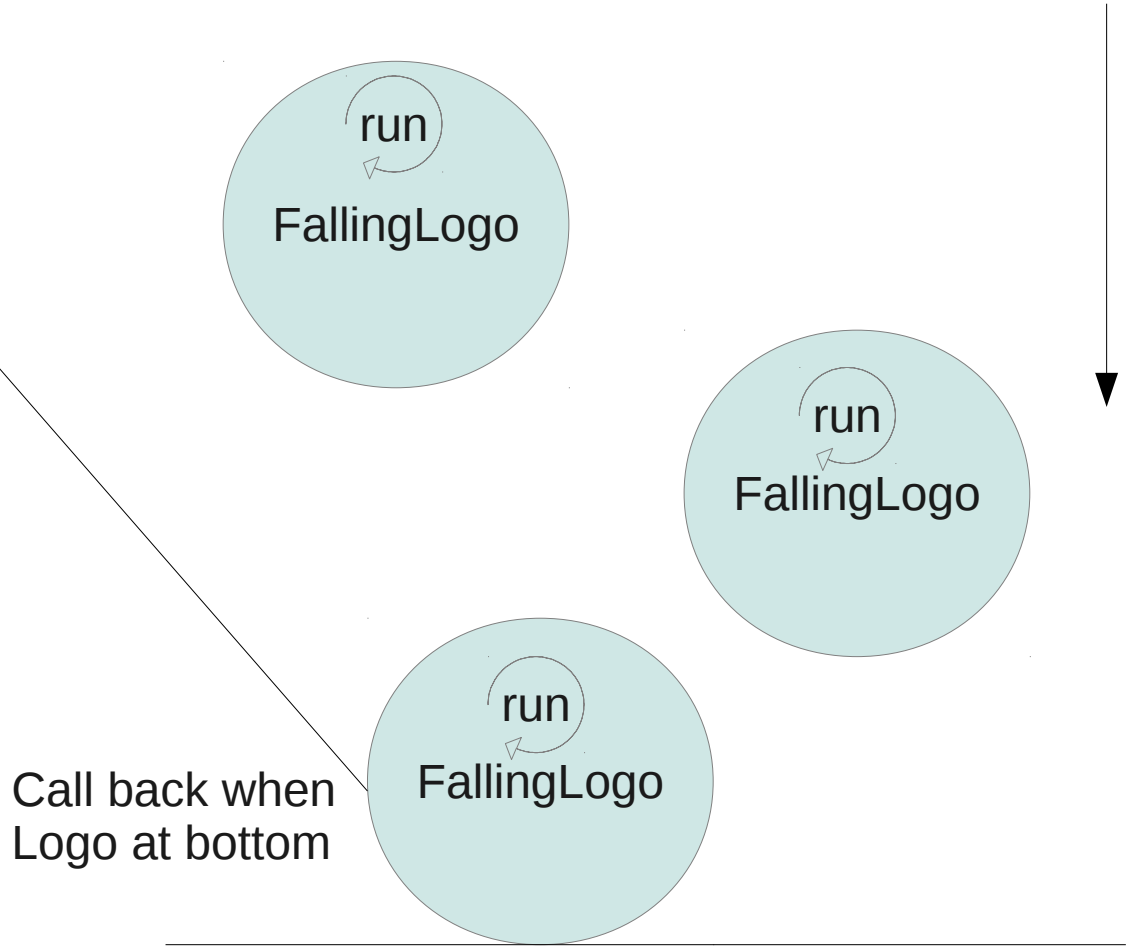
FallingLogo Reports Back

- This is often termed a “callback”
- What's needed
  - Controller needs to define a method that FallingLogo instances will call.
    - Let's term this atBottom()
  - FallingLogo needs to know
    - Its Controller
      - Pass a reference to the controller to the FallingLogo constructor
    - Logically, call `atBottom()` method when it hides itself

See: FallingLogoCallback.java, LogoControllerCallback.java

When the Callback Happens

```
LogoControllerCallback

public void atBottom(){
    … increment count;
    … display new count;
}
```




```
$ javac FallingLogoCallback.java LogoControllerCallback.java
$ java LogoControllerCallback
```

Making Animations Smooth

- So far the animations are
  - move a fixed # of pixels
  - pause -at least- n milliseconds
- Issue: when calling pause, it may take a while for your code to start executing again
  - Pause is at least, and may longer, and may be significantly longer.

One way to solve

- What we are really trying to do define *speed*
  - *speed = dist/time*
- In our case, speed = pixels/millisecond.
- Change logic of program
  - Don't pause and then move
  - Read the clock, figure out how long it has been since you last read the clock (call that `dt`)
  - Then distance to move is
    - `speed * dt`
- You will practice this in a program after the midterm

```
$ javac FallingLogoCallbackTimed.java LogoControllerCallbackTimed.java
$ java LogoControllerCallbackTimed
```