CS112 Lecture: Active Objects

Last revised 2/18/08

Objectives:

1. To introduce the notion of active objects

Materials:

1. Blue J project with demos from book: FallingBallController/FallingBall (actually modified to support the next example); ColorBallController/FallingBall; DropCollector/FallingDroplet

I. Introduction

A. Thus far in the course, the objects we have studied have been basically *passive* - i.e. they have performed operations upon specific request in the form of a call to one of their methods.

B. The "driving force" has been one of two things:

1. In our robot programs, we have had a main program that creates the necessary robot object(s) and then tells it/them to "do its thing".

   Example: For option 2 of project 1, there was a main class (called Project 1) which created two robot objects, then told each, in turn, to run its race.

2. In most of the examples in the Bruce book, operations have initiated by a human user performing some action such as clicking or dragging the mouse. Objects have had event handler methods like `onMouseClick()`.

C. In Lab 6 (and Project 2) we first met the notion of an "active object" - which is an object that carries out some task more or less autonomously once it is created. That is also the topic of the current chapter in the Bruce book.

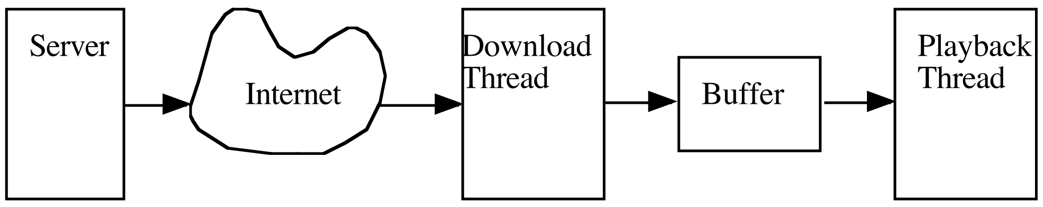
1. The examples in the chapter are all related to animations.

2. The term that is more commonly used for active objects is "thread". The standard java library provides a class called `Thread`, which the objectdraw library class `ActiveObject` extends.

3. The same idea occurs in lots of other contexts - e.g. a complex system may be structured as a set of threads (active objects) that interact with one another, rather than a single monolithic whole.

Example: We are used to the fact that, when we download a song or a movie using a web-browser, the song/movie begins playing before we have completed the download. As long as the download can "keep up with" the playing, we will hear/see the item without any breaks.

This can be managed by structuring the system as follows:



The "Download Thread" executes code like the following:

```
while (! all frames downloaded)
{
     Download a frame (will involve waiting for the server)
     Store it into the buffer
}
```

The "Playback Thread" executes code like the following:

```
while (! (all frames downloaded && buffer empty))
{
     Remove a frame from the buffer (may involve waiting
      for the Download thread if the buffer is empty)
     Decompress and play it
}
```

4. The topic of "active objects" or "threads" is a big one, and includes lots of issues related to synchronization between threads, etc. We will only touch the surface of the concept here. It is dealt with further in CS211 as well as later in the CS curriculum.

II. Animations

A. The chapter in Bruce began with an example of a "falling ball" animation. Let's look at what it does.

DEMONSTRATE:

Run `FallingBallController` as an applet - create multiple balls by multiple clicks

B. Now, let's look at the code.

   PROJECT `FallingBall`

   Note the following characteristics

   1. The class extends `ActiveObject`. (If we look at the documentation for the objectdraw library, we see that this class, in turn, extends the standard Java library class `Thread`.)

      SHOW

   2. Every active object has a method named `run()`, which specifies what this object does.

      a) Often - but not always - this method will contain a loop that performs some sequence of operations repeatedly - here moving the object on the screen, and then pausing for a short period of time.

         (Without the pause, the animation would happen so fast that we wouldn't see it.)

         DEMO: comment out line with pause, run applet. It appears that nothing is happening when I click the mouse, but that is only the case because it happens too fast.

         Now repeat demo with `DELAY_TIME` set to 1.

         Repeat demo again with `DELAY_TIME` set to 1000

      b) The loop terminates when the ball goes off the canvas. At this point, the `run()` method terminates, which terminates the thread. (As an efficiency optimization, the ball is removed from the canvas before this to prevent cluttering up the canvas with invisible objects, which could eventually slow down repainting since each time the canvas is redrawn, each object that is on it must be asked to draw its "in bounds" portion, if any.

   3. The constructor for an active object calls a method named `start()`. Note that it does <u>not</u> call `run()` directly. Rather, the call to causes the Java system to create a thread that executes the `run()` method autonomously.

C. As Bruce points out, it is also possible to animate images. (As it turns out, the code for this one is not available on their web site)

III. Interaction between Active Objects

A. It is possible to interact with active objects, or for interactive objects to interact with one another.

B. For example, here is a variant of the example in the book, which sets the falling ball color to red when the mouse is moved off the canvas (rather than gray, due to visibility). This is a case where user actions affect the active object.

   DEMO ColorBallController

   Show code - note how the mouse exit/entry event handlers send a message to the most recently-created falling ball, which happens to be an active object.

   Note how only the most recently-created ball is changed. (Changing all the balls would require the controller "remembering" all of the balls it has created - which would involve a data structure known as an array which we haven't learned about yet.

C. Another example in the book involves falling rain drops that increase the "water level" at the bottom of the canvas when they hit ground.

   1. DEMO DropCollector

   2. Show code - note how the `run()` method of each droplet modifies the "water level" rectangle appropriately.

   3. Actually, this program contains a subtle problem that is unlikely to manifest itself in practice.

      Consider what would happen if two drops landed at <u>exactly</u> the same time. (This could happen if one was started further down the canvas at the same time the other one got to that level.) What would happen in this case?

      ASK

   4. Solving problems like this beyond where we need to be at this point - however, the issue is non-trivial and a source of subtle errors.

      Example: Synchronization problem between computers that scrubbed an early flight of the space shuttle.

D. The "Pong" game we implemented in lab could have been done this way as well, with the ball being an active object that interacted with the room, the paddles, and the scores. (Actually, the paddles could be active objects, too, though there is some complexity involved here because the same keyboard controls both paddles.)

IV. Threads outside of objectdraw

A. We now look at one demonstration of the basic idea of threads outside of objectdraw.

B. You may recall that, in your karel labs/project, you invoked a method called `World.setupThread()` - passing as a parameter to it a runnable object - which was a main program that created a robot and then gave instructions to it. We will now look at an example in which our individual robots are separately runnable objects.

DEMONSTRATE RobotRace

SHOW CODE for RacerRobot and RobotRace

V. We will save further discussion of Threads for CS211