CS112 Lecture: Introduction to Event-Driven Programming in Java

Last revised 1/8/09

Objectives:

1. To introduce the notion of event-driven programming
2. To talk about the use of names in a Java program

Materials:

1. BlueJ project with SimplePicture, Second version of TouchyWindow, SimpleEscaperRobot (from iteration lecture), plus main program Main.java and Maze.world; ScrollingSun (from chapter 3)
2. Projectable of Problem 1.5.3 (p. 23) from book

I. Introduction

A. There are three basic "styles" of program

1. Non-interactive programs - no interaction with a user while program is running.

   Examples:

   programs used in Lab 1
   all our robot programs

2. Interactive programs - dialogue with a user, with program "in the driver's seat"

   Though once common, pure examples of this sort of program are becoming increasingly rare - in fact, I couldn't really come up with one for this lecture!

3. Responsive, or event-driven programs - program consists of a set of "handlers" for various events (e.g. mouse click on a button, menu choice). The user is "in the driver's seat".

   Examples:

   Most GUI programs are of this sort - e.g. a word processor
   Many embedded systems - e.g. software on a microwave oven responds to events like user pressing a button, user opening/closing door, timer "ticking"

   There may also be a method for initially setting things up.

B. The Bruce book focusses on the latter style of program

1. Event driven programs actually subsume the other two styles - e.g. a button click may initiate a non-interactive operation (e.g. sorting a list) or a dialogue with the user (via dialog boxes)

2. The early examples in the book are not terribly useful - rather, they are meant to help us get our feet wet. Most of the examples are graphical in nature because it is easier to really see what's going on.

II. Fundamental Graphics Concept

A. Because we will be using graphics a lot, we will spend a bit of time on fundamental concepts.

B. We will be making use of a special graphics library created by the authors of our book called objectdraw. (Many of the things the book talked about - e.g classes like Line, FramedOval, WndowController ... - are part of this library)

1. The names of things in this library are a bit different from the code we used in Lab 1, but the concepts are the same.

2. Its purpose is instructional - it is not “industrial strength”

3. However, “industrial strength” systems use the same basic ideas

C. Graphical systems like objectdraw are typically based on the concept of a window in which various figures are displayed.

1. One object represents the window.

2. The remaining objects represent the various figures that appear - e.g. lines, ovals, rectangles, arcs, boxes containing text.

3. Coordinates of objects in a window are typically expressed relative to the upper left-hand corner of window (= 0, 0). Units are typically pixels. [A whole screen may be something like 1024 x 768].

4. Coordinates go l-r in X direction, but top-bottom in Y direction. (Opposite of math conventions) An object's location is generally spelled out by giving its upper left coordinate plus width, height.

   [Note: figure 1.10 on p16 of the text is a bit confusing, since it is implies the origin is at the center, and coordinates can be negative or positive.]

D. What you see on the screen is represented by a collection of objects - one representing the window, and the remainder the various individual objects.

Example: SimplePicture

1. Demonstrate

2. Show code: Four objects: the WindowController, two FramedRects, one Text object.

3. When the program starts up, the WindowController object is created (by code that is part of the library but not directly visible.) It has a begin() method which creates the other objects.

4. What type of program is this?

   ASK

   Non-interactive.

E. Let's do one of the exercises from the book to drive home some of these ideas.

PROJECT Exercise 1.5.3 from page 23

DO AS A CLASS

III. The Structure of a Class Definition

A. We now consider an example, which illustrates many characteristics of a Java class: The second version of TouchyWindow (from book

1. Demonstrate

2. Show code

3. Here, there is one object that exists all the time. A text object is created at startup, but destroyed the first time the canvas is cleared. A new text object is created when the mouse button is pressed, and destroyed when it is released.

4. Recall our definition of an object from the first lecture. What three properties does every object have?

ASK

Identity
State
Behavior

5. Let's consider the text objects in this example program

   a) Each had a distinct identity.

      One came into being at startup, and went away the first time the canvas was cleared.

      An additional object was created - and then destroyed - each time the mouse was pressed and released.\

   b) Each had a distinct state. What was part of the state of a text object?

      ASK

      Its position on the screen and its contents.

   c) Not evident in the example were any behaviors. In fact, in the objectdraw library, Text objects do have a number of behaviors - for example, they have a move behavior, which we will demonstrate shortly.

   d) We are not given any Java class definition for the Text class - we simply use a library class as provided for us.

6. Now let's consider the TouchyWindow object in this example

   a) Here we are given a class definition that extends a library class (WindowController). In fact, some aspects of its state and behavior are inherited from this base class, while others are explicitly specified as part of the Java code we are given.

   b) There was only one object of this kind (it was what we call a singleton). However, it would be possible to run the program several times, in which case there would be several such objects.

      Demo: run two copies of the program at once. Note how each TouchyWindow responds only to clicks within itself

c) Its state included a list of the other objects displayed within it, which need to be drawn whenever it is drawn.

Demo: Run again to get text. Show what happens when window is occluded then rexposed; minimized then re-displayed.

Note that the state information about objects displayed and the redraw behavior are inherited from the base class.

d) What behaviors did it have?

ASK

begin(),
onMousePress(),
onMouseRelease()

(1) These behaviors <u>are</u> spelled out for us in the Java code, rather than being inherited.

(2) An event-driven program - like this one - is characterized by having some behaviors that correspond to events - e.g. two of the three behaviors in this case. In many cases (but not all), these behaviors are spelled out in the code we write, rather than being inherited.

B. An important point: names such as WindowController, Text, canvas, begin, onMousePress, onMouseRelease etc were chosen by the authors of the library. (They could have chosen other names had they wished)

1. Their capabilities exist because someone wrote Java code for them.

2. In Java, the kinds of information included in an object's state, and the behaviors it exhibits, are specified in its class definition. If we have access to the source code for a class, we can change them.

3. Demo: Modify touchy window so that a text is moved when the mouse is released, rather than being destroyed.

a) Add instance variable: Text mostRecentText;

b) Assign the newly created text to this variable in onMousePress();

c) Move it by move(20, 20) instead of canvas.clear() in onMouseRelease();

4. Run program.

   Why does the text appear to disappear the second and subsequent times?

   ASK

   Hint: note how the moved text gets darker

C. A Java class definition can be thought of as a "template" or mold that specifies how objects are to be created.

   Example: In the Karel portion of the course, all robots were created from a Robot class (which we extended). The definition within the simulator spelled out components of a robot's state - e.g. its position, number of beepers, etc. It also spelled out certain behaviors (e.g. move()). When we extended classes, we added behaviors.

   Behaviors of objects are actually of four different kinds:

   1. A constructor behavior specifies how an object is created.

      Example: The BlueJ template we used for creating Robot classes created a constructor for us.

      Show: Constructor in SimpleEscaperRobot (from iteration lecture)

      We will learn more fully about constructors later in the course.

   2. Mutator behaviors change the state of the object.

      a) Example: What mutators do all robots have?

         ASK

         move(),
         turnLeft(),
         pickBeeper(),
         putBeeper(),
         turnOff()

      b) What kinds of mutators did we create?

         ASK

         turnRight() etc.

c) The signature for a mutator typically contains the word "void"

3. Accessor behaviors allow finding out about the state of the object without changing it.

   a) Example: What accessors do all robots have?

      ASK

      frontIsClear()
      facingNorth() ...

   b) What kinds of accessors did we create?

      ASK

      leftIsClear() etc.

   c) The signature for an accessor contains some return type other than "void" - e.g. in the examples we did boolean, but many others are possible as well.

4. A Destructor behavior would specify what is to be done when the object is destroyed.

   This is rare in Java, but more important in some other OO languages. (You won't really see this until CS212)

D. Let's look at a class specification that has all of these: ScrollingSun (from chapter 3

   Demonstrate

   Show code

   1. The state of a ScrollingSun object includes two other objects: a Text object containing the instructions (which is either visible or hidden) and a FilledOval object representing the sun. The state also includes a canvas that part of the WindowController which FilledSun extends. (This declaration is not visible in this class, but the code does make use of the canvas)

      Note format of a declaration for a component of the state - which is called an instance variable.

a) private - more on the meaning of this later in the course

b) type name - Java is what we call a strongly-typed language, which means that every variable represents one specific type of thing

c) variable name

2. A ScrollingSun object has a begin() behavior that fulfills the role of a constructor, in terms of setting up the initial state.

3. A ScrollingSun object has two other behaviors that specify behaviors associated with "mouse drag" and "mouse exit" events. Are these mutators or accessors? Why?

   ASK

   Mutators - they change the state of the object (actually, the state of the objects comprising it).

4. All of the behaviors of The ScrollingSun object make use of the getWidth() behavior (and in two cases the getHeight() behavior) of the associated canvas. Are these behaviors of the canvas object accessors or mutators? Why?

   ASK

   They appear to be accessors - they get information about the canvas object, but (at least as implied by the name) they don't change its state. [Accessors often have names like get....].

5. One other thing to note: one of the behaviors makes use of a parameter called mousePosition to control where the sun is moved to based on where the user drags it.

   a) A parameter is not part of the state of an object, but rather information sent to one of its behaviors to affect a particular use of that behavior. It is strictly local to that method, and a specific value applies only to that one call of the method.

   b) Note format of the declaration: Type followed by variable name, appearing in the method header

   c) Does the ScrollingSun object ever _send_ parameters to the behaviors of other objects? Where?

ASK

The constructors of the FilledOval and Text objects in the begin() method involve parameters specifying the initial position of both objects, the size of the oval and the contents of the text (from which the size is inferred).

The use of the moveTo behavior of instructions in begin() and of sun in the other two behaviors includes parameters that specify where the object is to be moved to.

d) A bit of terminology:

(1) The names of parameters appearing in a method header (e.g. mousePosition) are called formal parameters.

(2) The parameters actually sent when a method is called are called actual parameters.

(3) In the class definition, the formal parameter serves as a "place holder" to stand for the actual parameter that is specified when the method is used.

IV. Rules about names.

A. All of our examples (both today and in conjunction with Karel) have used names for various things. It turns out that the names we choose must obey certain rules. One of these is imposed by the Java language (and could be different for other languages); the others are established conventions.

1. Java requires that names be composed of alphabetic characters, digits, and the underscore (_), with the first character being either alphabetic or _. (Actually, $ is also permitted, but should not be used for ordinary purposes. Java uses it in the some of the names it creates automatically)

2. The use of "mixed case" is an established conventions which the Java compiler does not enforce, but which should still be observed.

B. So far, we have seen a number of different things that can have names in a Java program. :Let's review what they are.

ASK

1. Names of classes. Tipoff - they begin with an uppercase letter

   Examples in ScrollingSun?

   ASK

   ScrollingSun,
   WindowController,
   FilledOval,
   Text

2. Names of methods - Tipoff - they begin with a lower case letter and are always followed by ().

   Examples in ScrollingSun?

   ASK

   begin(),
   onMouseDrag(),
   onMouseExit()

   Note: it is conventional to include parentheses with the name when writing it - so one usually writes onMouseDrag() rather than just onMouseDrag

3. Names of packages - Tipoff: they only occur in import or package statements, often are multi-part with dots, often end in * (which means everything in that package), though sometimes we just import an individual class from a package

   Examples in ScrollingSun?

   ASK

   objectdraw.*,
   java.awt.*

   In this example, could just import java.awt.Color

4. Names of instance variables - begin with a lowercase letter, and are declared outside any method within the class.

   An instance variable represents a property that each individual instance of the class has its own unique value of.

Examples in ScrollingSun?

ASK

sun,
instructions

5. Names of formal parameters - begin with a lowercase letter, and are declared inside the parameter list of a method

   A formal parameter stands for information that will be supplied when the method is used - e.g. the location of the mouse when one of the events occurred

   Examples in ScrollingSun?

   ASK

   mousePosition (in header of onMouseDrag()),
   point (in header of onMouseExit()

C. We have seen two kinds of variables that can appear in Java programs: instance variables and formal paremeters. There are actually more, as we shall see later.

   1. In Java, a variable is always explicitly declared to be of some type. What kind of variable something is depends on where/how it is declared.

   2. But regardless of what type of variable something is, the main place where it is used is in the body of some method. A variable is always a name for something that the program works with. Two possible uses include:

      a) As the name of an object a message is being sent to [whose method is being invoked]

         (1) Example: sun in sum.moveTo(... or instructins in instructions.hide() in onMouseDrag(), onMouseExit().

             Are there any other examples in ScrollingSun?

             ASK

             instructions in instructions.moveTo(... in begin()

(2) Sometimes, a method invokes another method of the same object. In this case, no explicit destination need be specified:

Example: escape() method of SimpleEscaperRobot uses the advanceOneBlock() method of the same object.

(3) As an alternative, in this latter case, one can use a special reserved name that Java recognizes: *this*. This name always refers to the object whose method is being executed. Thus, instead of writing advanceOneBlock() in escape(), we could have written this.advanceOneBlock() - the meaning would have been identical. (Though explicitly using *this* here is not necessary, there will be cases where using it explicitly is necessary).

b) As a value being used in a computation or as an actual parameter to a method (no examples in ScrollingSun, but we will look at an example elsewhere shortly.)

3. A variable can be made to refer to something by an assignment statement - example sun =, instruction = in begin().

a) In these two cases, the variables in question didn't refer to anything at all before the assignment, but it is also possible to change a variable that used to refer to one thing and make it refer to something else.

Example: onMousePress() methd in modified TouchyWindow (show)

b) In the cases we have looked at thus far, variables have been used to refer to newly-created objects. Variables can also be used to refer to computed values

Example: The for loop in SimpleEscaperRobot (escape method) - the variable count successively refers to the values 0, 1, 2 ... 20. On each iteration of the loop, count ++ makes it refer to the next value in sequence.

V. A few other things

A. Most of the classes we have talked about today have represented visible objects (e.g. FilledOval). The same has been true of our Robot classes. That is a peculiarity of the approach I have taken and that taken by our book, using graphics as a teaching tool. In fact, most Java classes do not represent visible objects.

The book chapter talked about two classes that do not represent visible objects. What were they?

ASK

Color, Location

Show how to access javadoc for each. Note that Color is in java.awt; Location in objectdraw

B. The book pointed out the distinction between a syntax error in a program and a logic error in a program. What is the distinction?

ASK

1. Example of a syntax error: In the escape() method of SimpleEscaperRobot, if we mistyped for as foor, we would have a syntax error.

   DEMO: Change, then compile. (Note how message from the compiler is not terribly helpful at identifying what was actually wrong in this case. The compiler makes a "guess" at what you intended, but sometimes guesses very incorrectly!)

2. Example of a logic error: In the same method, if we used < 19 instead of < 20.

   DEMO: Fix syntax error, add logic error, then compile - note that the compiler cannot catch this error.

   DEMO: Run of main program

   "The computer always does what you tell it to do, not what you want it to do"

C. The book also talked about layering objects on a canvas - i.e. when two or more visible objects overlap, what do you see. We will not add to the book's discussion.