CS12 Lecture: Introduction to Event-Driven Programming in Java

Materials:

SimplePicture
Second version of TouchyWindow
SimpleEscaperRobot
ScrollingSun (from chapter 3)

Three basic "styles" of program

• Non-interactive programs - no interaction with a user while program is running.
  Examples: programs used in Lab 1
             all our robot programs

• Interactive programs - dialogue with a user, with program "in the driver's seat"
  Examples: Show checkreg - get roster for CS112

• 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.

The Bruce book focusses on the latter style of program.

• Event handlers 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)

  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.

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

We will be making use of a special graphics library created by the authors of our book called objectdraw.

- 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.

- Its purpose is instructional - it is not "industrial strength"

- However, "industrial strength" systems use the same basic ideas

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

- One object represents the window.

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

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

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.

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
- Demonstrate
- Show code

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

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.

What type of program is this? ASK
Non-interactive.
We now consider an event-driven example.

Example: Second version of TouchyWindow (from book)

- Demonstrate
- Show code

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.

-----

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

ASK

Identity
State
Behavior

Example: the program we just looked at had many text objects.

• 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.

• Each had a distinct state. What was part of the state of a text object?
  
  ASK

  Its position on the screen and its contents.

• 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.

Example: the program we just looked at had a WindowController object.
• 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 WindowController responds only to clicks within itself.

• 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, minimized then re-displayed.

• What behaviors did it have?

ASK

begin(), onMousePress(), onMouseRelease()

Important: names such as WindowController, Text, canvas, begin, onMousePress, onMouseRelease etc were chosen by the authors of the library. Their capabilities exist because someone wrote Java code for them.

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.

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

- Add instance variable: Text mostRecentText;
- Assign the newly created text to this in onMousePress();
- Move it 20, 20 instead of canvas.clear() in onMouseRelease();

Run program.

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

ASK

Hint: note how the moved text gets darker
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:

• 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.

• Mutator behaviors change the state of the object.

Example: What mutators do all robots have?  ASK

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

What kinds of mutators did we create?  ASK

turnRight() etc.

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

• Accessor behaviors allow finding out about the state of the object _without changing it_.

Example: What accessors do all robots have?

ASK

frontIsClear(), facingNorth() ...

What kinds of accessors did we create?

ASK

leftIsClear() etc.
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.

• 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)

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

- Demonstrate

- Show code

- 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.

private - more on the meaning of this later in the course
type name - Java is what we call a strongly-typed language, which means that every variable represents one specific type of thing variable name

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

- 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).

- 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....].

- 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 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.

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

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.

A bit of terminology:

• The names of parameters appearing in a method header (e.g. mousePosition) are called _formal parameters_.
• The parameters actually sent when a method is called are called _actual parameters_.
• 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.
Rules about names.

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.

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

The following are established conventions which the Java compiler does not enforce, but which should still be observed:

• Use of "mixed case"

  Example: SimpleEscaperRobot, ScrollingSun

• Class names begin with an upper-case letter

  Examples: the above

• Variable and method (behavior) names begin with a lower-case letter

• The parameter list of a method is enclosed in (); empty () if no parameters.

  Example: pickBeeper()

When writing about methods, it is conventional to include the () with the name.
A few other things

• 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. Most Java classes do not, in fact 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

• 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.