The Objectdraw Event-Handling Library

Contact Information: Kim B. Bruce (`kim@cs.williams.edu`), Andrea P. Danyluk (`andrea@cs.williams.edu`), and Thomas P. Murtagh (`tom@cs.williams.edu`).

Problem Statement: As detailed in sections A3 and A4 of the Task Force's Taxonomy of Problems in Teaching Java, event-driven programming can provide an attractive introduction to object-oriented programming. Yet it is difficult to teach to novices because of the heavy syntactic overhead in using event-driven programming in standard Java. One reason for the attraction of event-driven programming is that the programs students use are event-driven rather than being based on line-oriented input and output. An even more important reason is that event-driven programming leads to programs that naturally involve a number of short methods that are called in response to events rather than a large monolithic program. Thus students don't have to be taught later to break code into small pieces, and they get experience writing methods that might be called in different orders. The main problem is how to support the event-handling style without the overhead of having students declare classes that implement event-handling interfaces, associate listeners with components, and write methods that have to unpack events to determine useful information.

Solution Overview: The objectdraw library supports a `WindowController` class, a specially designed extension of `JApplet` that facilitates student use of event-driven programming by supporting methods that are executed when events are generated by mouse actions on a drawing canvas. The `WindowController` class implements both `MouseListener` and `MouseMotionListener` interfaces and includes code that declares the applet as the listener for mouse actions on the canvas.

In order to simplify the event-handling methods, we introduce variations of the standard Java methods for handling mouse events that take locations on the canvas as parameters rather than events. This saves students from having to extract from the event object the location of the mouse at the time the event is generated.

Figure 1 illustrates a very simple program that allows the user to drag around a filled oval on a canvas. It illustrates the use of the event-driven features as well as some of the graphics classes in the objectdraw library. The `begin` method is executed when the program starts, drawing a red ball on the screen along with a message telling the user to drag the ball. When the mouse button is pressed, the `onMousePress` method is executed. The value of the parameter `point` is the location of the mouse when its button is pressed. Similarly, the `onMouseDrag` method is executed when the mouse is moved with the button down. Finally the `onMouseRelease` method will be invoked when the mouse button is released. Notice that (1) there is no need to declare that the class implements any listeners, (2) there is no need to construct mouse listeners and mouse motion listeners and associate them with the canvas, and (3) the mouse event-handling methods take a location rather than an event as a parameter, making it easier for students to access the location where the mouse was when the event was triggered.

An important feature of the library is that while we have reduced the syntactic overhead of handling mouse events, the style of programming is the same as for standard Java. In our course we introduce these mouse event-handling methods in the first week of classes. About half way through the course we introduce Swing GUI components like `JButton`, `JSlider`, and `JTextField`, and introduce students to the standard Java event-handling constructs. This is now easy for

```
public class DragBall extends WindowController {
  private FilledOval ball; // ball
  private Location lastPos;  // last mouse posn

  // make the ball
  public void begin() {
    ball = new FilledOval(95,50,30,30, canvas);
    ball.setColor(Color.red);
    new Text("Drag the ball!", 75,20, canvas);
  }

  // Save starting point and if point in box
  public void onMousePress(Location point) {
    lastPos = point;
  }

  // if mouse is in ball, then drag the ball
  public void onMouseDrag(Location point) {
    if ( ball.contains(lastPos) ) {
      ball.move( point.getX()-lastPos.getX(),
                 point.getY()-lastPos.getY());
      lastPos = point;
    }
  }

  // if dragging when release, go back to start
  public void onMouseRelease(Location point) {
    if (ball.contains(lastPos))
      ball.moveTo(95,50);
  }
}
```

Figure 1: Class illustrating the objectdraw graphics library

students as they have been writing programs using this style since the beginning of the term. It is only the extra syntax needed to set up listeners that is hidden from them. Thus it is easy for students to transition from using our library to using standard Java event-handling code.[1]

Experience with the solution: We have been using the objectdraw library for 5 years. Faculty at more than a dozen colleges, universities, and high schools have used the library in conjunction with our on-line text and lecture notes. We have recently ported the library from AWT to Swing. Those using the library have reported that it is reliable and highly motivating for students.

We have found the event-driven support in the library especially useful in conjunction with the support for truly object-oriented graphics that is also contained in the library and is described in a separate document. The combination of truly object-oriented graphics and event-driven programming leads to the possibility of writing very interesting and motivating programs from the first week of classes.

API Documentation: Documentation for the classes in the library may be found at:

`http://cortland.cs.williams.edu/~cs134/eof/library/javadocs/`

Supplemental Materials: The web page

`http://cortland.cs.williams.edu/~cs134/eof/`

includes links to an early AWT version of the objectdraw library, textbook examples, papers on our approach, instructions on using the library with different IDE's, and a collection of resources for instructors. The resources for instructors include lecture notes, other sample programs, and laboratory assignments. This page also has a link to another page that contains the Swing version of the objectdraw library as well as a more recent version of the text that supports the Swing version of the library.

[1]Because proper event-handling style require the use of separate threads to respond to events, the objectdraw library also includes a class `ActiveObject` that reduces the syntactic complexity of using threads by supporting an exceptionless `pause()` method.