Event-driven and Multi-threaded Programming in CS-1 using the ObjectDraw Package

Mark Fienup
Computer Science Department
University of Northern Iowa
Cedar Falls, IA 50614-0507
fienup@cs.uni.edu

Abstract

Learning to program is a challenging and time-consuming task for most students. To motivate CS-1 students through this difficult period and to ease the learning of object-oriented programming in Java at the University of Northern Iowa, we are experimenting with the ObjectDraw package developed at Williams College [1, 2]. Labs developed using the ObjectDraw package allow beginning programmers to use graphical objects (ovals, rectangles, lines, text, etc.) in event-driven and multi-threaded programs with relative easy. The use of the ObjectDraw package allows us to focus on the key concepts of object-oriented programming without overwhelming students with the complexity of "raw" Java. This paper will describe the ObjectDraw package in more detail, and outline how we incorporate ObjectDraw into the lectures and labs of our CS-1 course. Additionally, we will report on the impact this approach has had on our attrition in the CS-1 course.

Introduction

Learning to program has always been a challenging and time-consuming task for most students with relatively high attrition rates typical in CS-1 courses. The paradigm switch from procedural languages (Pascal and C) to object-oriented languages (Java and C++) in CS-1 have not made learning to program any easier [3].

One way to motivate CS-1 students through this difficult period is to try to make learning to program more interesting and realistic. To ease the learning of object-oriented programming in Java at the University of Northern Iowa, we are experimenting with the ObjectDraw package developed at Williams College [1, 2]. Labs developed using the ObjectDraw package allow beginning programmers to use graphical objects (ovals, rectangles, lines, text, etc.) in event-driven and multi-threaded programs with relative easy. The hope is that students find these labs highly motivational due to their graphical nature and realness of being event-driven.

The use of the ObjectDraw package allows us to focus on the key concepts of

object-oriented programming without overwhelming students with the complexity of "raw" Java in the first halve of the CS-1 course. In the second halve of the course, students have little difficulty making the transition from the simplified ObjectDraw event-driven model to the more complex GUI components and event-driven model of the Java AWT.

This paper will describe the ObjectDraw package in more detail, and outline how we incorporate ObjectDraw into the lectures and labs of our CS-1 course. Additionally, we will report on the impact this approach has had on our attrition in the CS-1 course.

Graphical Objects of the ObjectDraw Package

The ObjectDraw package was developed at Williams College [1] by Kim Bruce, Andrea Danyluk, and Thomas Murtagh for uses in their CS-1 course. ObjectDraw allows beginning programmers to use graphical objects (ovals, rectangles, lines, text, etc.) in event-driven programs without overwhelming students with the complexity of "raw" Java.

Figure 1 is a sample ObjectDraw program that allows the user to drag a box around the screen using the mouse. Classes using ObjectDraw extend "WindowController" which itself extends the Java Applet class. The WindowController "begin" method is executed once at the beginning of the program to perform the necessary initialization. Here it creates a "box" which is an ObjectDraw FilledRect object of a specified size and location on the "canvas." The "canvas" in ObjectDraw's WindowController is essentially the applet window. See Appendix A for a complete summary of ObjectDraw graphical objects and their methods.

The remaining methods in Figure 1: "onMousePress" and "onMouseDrag" are part of ObjectDraw's simplified mouse-event handling methods. Other ObjectDraw mouse-event handling methods are onMouseRelease, onMouseClick, onMouseMove, onMouseEnter, and onMouseExit. These methods all return a Location object that encapsulates the X and Y coordinate of where the mouse-event occurred on the canvas. Frequently, the returned Location object is used to see if the mouse-event occurred within the area of ObjectDraw graphical object. For example, the onMousePress method in Figure 1 returns the Location-object "point" which is used to check if the FilledRect-object "box" "contains" the point where the mouse button was pressed ("`box.contains(point)`").

ObjectDraw allows for a truly objects-first approach to teaching CS-1. The graphical objects are used from the start. The event-driven nature of ObjectDraw allows for interesting early programs with only the "if-statement" for a control structure. Students can practice constructing new classes consisting of a collection of graphical objects (e.g., a t-shirt object consisting of rectangles for the body and arms with an oval for the neck) with same methods (move, contains, etc.) of the primitive ObjectDraw graphical objects.

```
import objectdraw.*;
import java.awt.*;

public class DraggableBox extends WindowController{

    // Starting position and size for box
    private static final int START_LEFT = 200, START_TOP = 100,
                             BOX_HEIGHT = 30,  BOX_WIDTH = 30;

    private FilledRect box;     // box to be dragged

    private Location lastPoint;   // point where mouse was last seen

    // whether the box has been grabbed by the mouse
    private boolean boxGrabbed = false;

    // begin is for initialization.  Here it makes the box
    public void begin() {
        box = new FilledRect(START_LEFT, START_TOP, BOX_WIDTH,
                             BOX_HEIGHT, canvas);
    } // end begin

    // Save starting point and whether point was in box
    public void onMousePress(Location point) {
        lastPoint = point;
        boxGrabbed = box.contains(point);
    } // end onMousePress

    // if mouse is in box, then drag the box
    public void onMouseDrag(Location point) {
        if ( boxGrabbed ) {
            box.move( point.getX() - lastPoint.getX(),
                      point.getY() - lastPoint.getY() );
            lastPoint = point;
        } // end if
    } // end onMouseDrag

} // end class DraggableBox
```

Figure 1: ObjectDraw program to drag a box around the screen with the mouse.

Multi-threading using ActiveObjects of ObjectDraw Package

For the first 5 weeks students use the graphical objects (FilledRect, FilledOval, Line, Text, etc.) with the simplified mouse-events of the WindowController class. Shortly after the introduction of the "while" loop, students are introduced to "active" objects that run even when the user doesn't do anything with the mouse. The ActiveObject class in the ObjectDraw library allows many of these active objects to execute in parallel with each other within their own thread.

ActiveObject's includes a number of instance variables and methods that are used to keep track of objects which can execute in parallel with each other. To create an ActiveObject one must: (1) define a class that "extends ActiveObject," (2) define its constructor and say

"start()" at the end, (3) define a "public void run()" method. The constructor typically places the active object on the canvas before calling the "start" method which does some housekeeping resulting in the creation of a new Java thread. The last thing the "start" method does is call the "run" method. The "run" method typically consists of a loop that animates the ActiveObject by checking for collisions with other objects, moving a few pixels on the screen, and then pausing before iterating the loop again. When the "run" method terminates, the thread dies, and the object is no longer "active."

Figure 2 contains the code for the MovingBall ActiveObject class that is used in a one player Pong-like game shown in Figure 3. The player clicks on the mouse to start a new MovingBall falling. She then tries to keep the ball aloft by positioning the paddle under it before it reaches the bottom. If the player misses the ball with the paddle, the ball falls off the bottom of the screen. The ball will bounce off the sides and top of the playing area as well as the paddle.

The MovingBall constructor in Figure 2 (a) is passed references to the paddle and game boundary which is stores in instance variables. It constructs a FilledOval to represent the falling ball, then it determines a random velocity for the ball in the X and Y directions. Finally, it calls the "start" method which initialized the thread before calling the "run" method.

The MovingBall "run" method in Figure 2 (b) consists of a while-loop that iterates while the ball has not fallen off the bottom of the screen. On each iteration of the loop (1) the time since the last ball movement is calculated, (2) the ball is moved based on this time and the balls velocity in the X and Y directions, (3) the direction of the ball is reversed if the ball is in contact with the paddle or the sides of the playing area, and (4) the ball is paused momentarily.

Finally, the Pong class that uses the MovingBall ActiveObjects is given in Figure 4. It extends WindowController and uses the a simplified mouse-events onMouseClick and onMouseMove. The onMouseClick method just drops a new MovingBall object when the mouse is clicked. The onMouseMove method moves the paddle corresponding to the mouse's location within the confines of the playing area.

By combining the simplified mouse-events of ObjectDraw with the multi-threading of ActiveObjects, CS-1 students are able to write some relatively sophisticated programs. For example, the sixth programming assignment in the CS-1 class during the Fall 2002 was to implement a "Frogger" game with four lanes of traffic to cross. Each lane of traffic was implemented by an ActiveObject whose job was to construct a stream ActiveObject vehicles. Figure 5 shows a snapshot of the Frogger game.

The Frogger game also illustrates another nice feature of ObjectDraw. ObjectDraw allows you to construct a VisibleImage object using an imported picture. VisibleImage objects have all the methods (move, moveTo, contains, etc.) as the standard ObjectDraw objects (FilledRect, FilledOval, etc.).

```
import objectdraw.*;
import java.awt.*;

public class MovingBall extends ActiveObject {
    private static final int BALLSIZE = 30;
    private static final int SCREENGAP = 130;

    // Constants to determine ball speeds
    private static final int MINSPEED = 2;
    private static final int SPEEDRANGE = 10;
    private static final int PAUSETIME = 40;

    private FilledOval ball;

    // components of speed vector
    private double xSpeed, ySpeed, initYspeed;

    // boundaries of playing area
    private double bLeft, bRight, bTop, offScreen;

    // the paddle
    private FilledRect paddle;

    private RandomIntGenerator speedGen;
    private DrawingCanvas canvas;

    public MovingBall(DrawingCanvas canvas, FilledRect paddle,
                      FramedRect boundary) {
        this.canvas = canvas;
        this.paddle = paddle;

        // give names to boundary values
        bLeft = boundary.getX();
        bTop = boundary.getY();
        bRight = bLeft + boundary.getWidth() - BALLSIZE;

        offScreen = bTop + boundary.getHeight() + SCREENGAP;

        ball = new FilledOval(boundary.getLocation() ,
                              BALLSIZE, BALLSIZE, canvas);

        // pick random pixel/pause speeds
        speedGen = new RandomIntGenerator(1, SPEEDRANGE);
        xSpeed = speedGen.nextValue()-SPEEDRANGE/2;
        if (xSpeed > 0)
            xSpeed = xSpeed + MINSPEED;
        else
            xSpeed = xSpeed - MINSPEED;

        ySpeed = speedGen.nextValue() + MINSPEED;

        // compute pixel/millisecond speeds
        xSpeed = xSpeed / PAUSETIME;
        ySpeed = ySpeed / PAUSETIME;
        initYspeed = ySpeed;

        start();
    } // end MovingBall constructor
```

Figure 2 (a): MovingBall ActiveObject class

```
    public void run() {
        // used to record times and compute time between moves
        double lastTime, elapsedTime;

        lastTime = getTime();

        while (ball.getY() < offScreen) {

            elapsedTime = getTime() - lastTime;

            ball.move(xSpeed*elapsedTime, ySpeed*elapsedTime);

            if ( ball.getX() < bLeft  || ball.getX() > bRight )
                xSpeed = -xSpeed;
            if ( ball.getY() < bTop )
                ySpeed = -ySpeed;
            else if ( ball.overlaps(paddle) )
                ySpeed = -initYspeed;

            lastTime = getTime();
            pause(PAUSETIME);
        } // end while
        ball.hide();
    } // end run

} // end MovingBall class
```

Figure 2 (b) Continuation of MovingBall ActiveObject

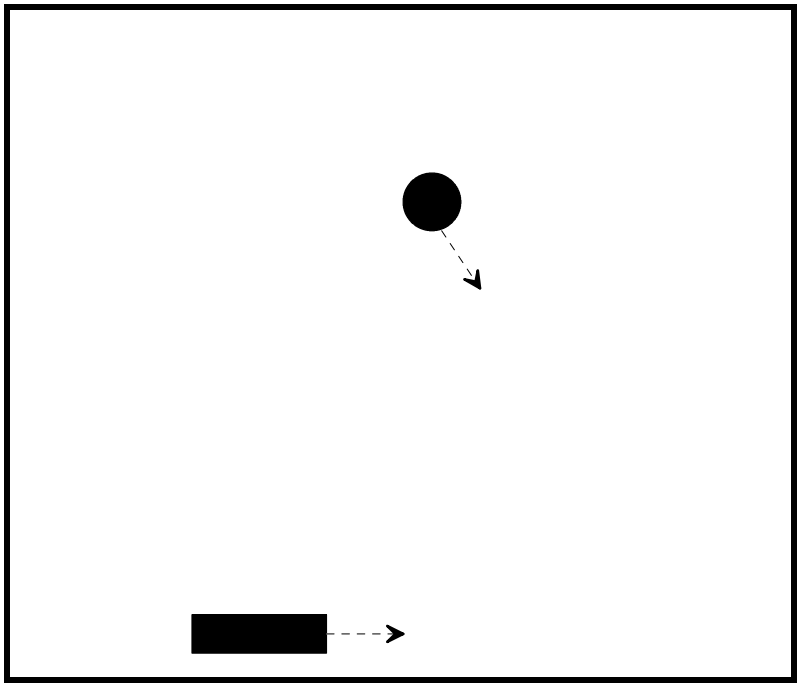

Figure 3. Pong-like game using a MovingBall ActiveObject

```
import objectdraw.*;
import java.awt.*;

public class Pong extends WindowController {

   // position and dimensions of the court
   private static final int COURT_LEFT = 50, COURT_TOP = 50,
                            COURT_HEIGHT = 300, COURT_WIDTH = 250;
   // dimensions of the paddle
   private static final int PADDLE_WIDTH = 50, PADDLE_HEIGHT = 20,
           PADDLE_TOP =COURT_TOP + COURT_HEIGHT - PADDLE_HEIGHT -1;

   private FilledRect paddle;
   private FramedRect boundary;  // boundary of the playing area.

   public void begin() {
      // make the playing area
      boundary = new FramedRect(COURT_LEFT, COURT_TOP,
                                COURT_WIDTH, COURT_HEIGHT, canvas);

      // make the paddle
      paddle = new FilledRect(COURT_LEFT +
                              (COURT_WIDTH-PADDLE_WIDTH)/2,
                              PADDLE_TOP,
                              PADDLE_WIDTH, PADDLE_HEIGHT, canvas);
   } // end begin

   // make a new ball when the player clicks
   public void onMouseClick(Location point) {
       new MovingBall(canvas, paddle, boundary);
   } // end onMouseClick

   // move paddle according to the position of the mouse
   public void onMouseMove(Location point) {
      if ( point.getX() < COURT_LEFT ) {
         // place paddle at left edge of the court
         paddle.moveTo( COURT_LEFT, PADDLE_TOP );
      } else if ( point.getX() > COURT_LEFT + COURT_WIDTH -
                                                  PADDLE_WIDTH) {
          // place paddle at right edge of the court
          paddle.moveTo( COURT_LEFT + COURT_WIDTH - PADDLE_WIDTH,
                         PADDLE_TOP );
      } else {
          // keep the edge of the paddle lined up with the mouse
          paddle.moveTo( point.getX(), PADDLE_TOP );
       } // end if
   } // end onMouseMove

} // end Pong class
```

Figure 4. Pong class that uses the MovingBall ActiveObject

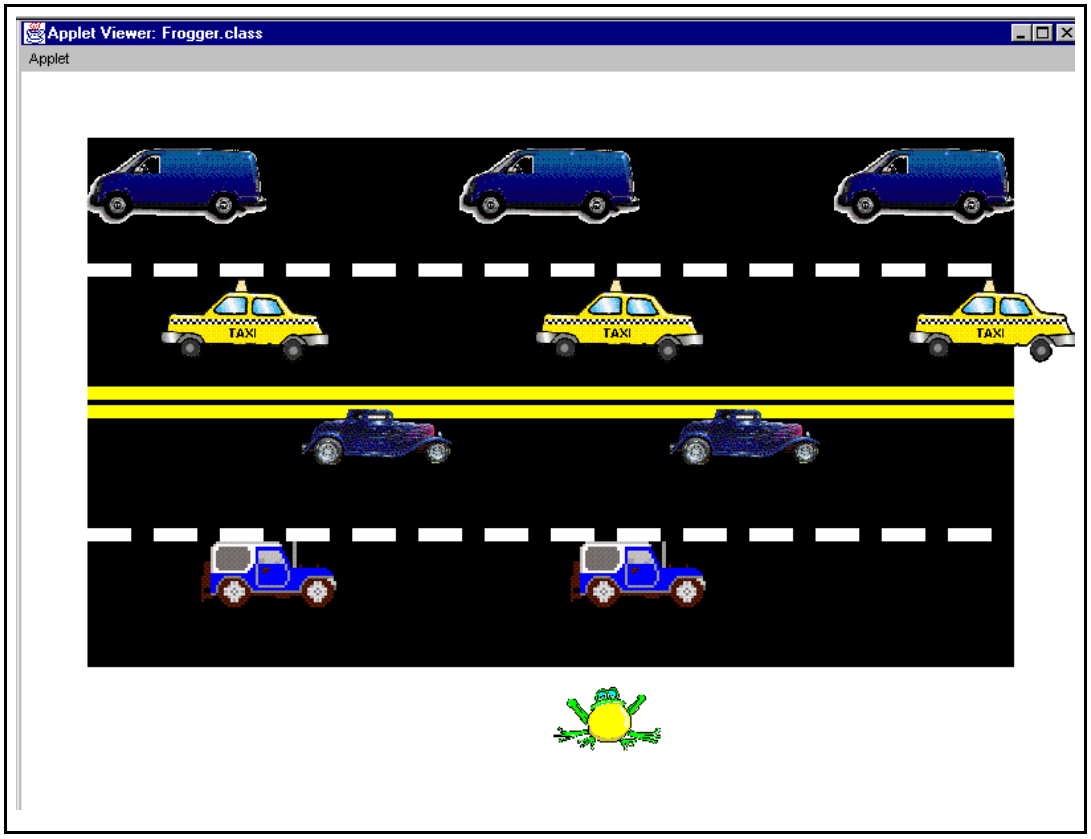

Figure 5. Snapshot of the Frogger lab

Usage of ObjectDraw in CS-1 at UNI

During the Fall of 2002, three sections of Computer Science I (810:061) at the University of Northern Iowa used the ObjectDraw package. The course required no previous programming experience by the students. Table 1 summarizes how ObjectDraw was used in the nine open-laboratory programming assignments to aid the students in learning to program.

During the first 4 weeks (and 3 programming assignments) of the course, instruction focused on using the graphical objects (FilledRect, FilledOval, Line, Text, etc.) with the simplified mouse-events of ObjectDraw's WindowController class. ObjectDraw allows for a truly objects-first approach to teaching CS-1 with graphical objects being used from the start. Early topics include creating objects, naming objects, and calling mutator methods on objects. More "traditional" topics of instance variables vs. local variables, formal parameters, numeric types, "if" statements, and usage of random number generators were discussed and used during this period.

After students were familar with using objects and their corresponding methods, they were shown how to implement user-defined classes during weeks 5 and 6 of the course.

Table 1: Programming Assignments

#	Description	Educational Objectives
1	Use the ObjectDraw graphical objects to construct a sign containing the text "Clicking". Only the "start" method of the WindowController class was used.	Familiarization with the lab, edit-compile-run in IDE, creation of ObjectDraw objects.
2	Modified the sign from assignment 1 to make it responsive to mouse-events.	Usage of ObjectDraw mouse events.
3	Implemented a laundry trainer to teach someone how to sort laundry items into whites, colors, and darks. Colored rectangles needed to be dragged to the correct laundry basket.	Experience with "if" statements, boolean variables, and using random number generators
4	Implement a Die class to be used with a simplified Yahtzee game. The Die class has methods similiar to the ObjectDraw graphical primitive (move, moveTo, contains, etc.)	Experience in implementing a user-defined class.
5	Implement a simple game called Boxball to train someone how to use the mouse. The object of the game is to try to click immediately above a box which drops a ball into the box. The ball was implemented as an ActiveObject.	Practice designing multiple classes, "if" and "while" statements in an ActiveObject
6	Implement a "Frogger" game with four lanes of traffic to cross (see Figure 5). Each lane of traffic was implemented by an ActiveObject whose job was to construct a stream ActiveObject vehicles. VisibleImage objects are used for the vehicles and frog. VisibleImage objects are constructed from imported images and have all the methods (move, moveTo, contains, etc.) as the standard ObjectDraw objects (FilledRect, FilledOval, etc.).	More practice designing with multiple classes. Usage of VisibleImage objects and importing images.
7	Standard AWT components (Scrollbar, Label, Choice, and Button) are used to adjust the speed, color, and visiblity of an ActiveObject bouncing ball.	Practice with Java AWT GUI components, layout managers, and Java interfaces
8	Use recursion and ObjectDraw graphics to draw Sierpinski's Gasket and other recursive diagrams.	Practice using recursion.
9	Draw a histogram of a Java Strings.	Practice using Java Strings, arrays, and characters

The fourth programming assignment required students to implement a "die" class which was used in a simple Yahtzee game. The concepts of class-definition syntax, parameter passing, and the difference between instance variable vs. local variables was stressed in lecture.

The syntax and usage of "while" loops was the next topic. ObjectDraw was useful in graphically demonstrating the power and necessity of looping by using examples to drawing 1,000 blades of grass on a picture, etc. Once students were comfortable with the syntax and semantics of iteration, "while" loops were used in ObjectDraw's ActiveObjects "run" method (see Figure 2(a) for an example). In-class coverage of iteration and ActiveObjects required weeks 7 and 8, but due dates of programming assignments 4 and 5 extended through week 9. Assignments 4 and 5 were challenging for the students. Two weeks for each assignment would have been better. Several students dropped about this time probably due to the difficulty of these assignments.

The traditional topics of Java interfaces, AWT-layout managers, AWT-GUI components were covered during weeks 9 and 10. Programming assignment 7 and many of the in-class examples used AWT components to modify characteristics (e.g., Choice for color, Scrollbars for ball-speed in the X and Y direction) of ObjectDraw objects. The The ObjectDraw WindowController uses the BorderLayout manager with the "canvas" being in position BorderLayout.CENTER.

The introduction to recursion during week 11 utilized ObjectDraw graphics to help students learn this confusing topic. Recursive graphics examples including a Broccil plant and NestedSquares (recursively draw a series of smaller squares within a larger one). The NestedSquares example was a good introduction to recursive data structures since its instance variables included a FramedRect square and a reference to the NestedSqure nested inside of it.

During the remaining weeks of the course, traditional topics of Java Strings, Arrays, searching, and sorting were covered. Little ObjectDraw usage was done during this period of the course.

Conclusions

The ObjectDraw package is a viable option for teaching Java using an "objects-first" approach in CS-1. Not surprisingly, students completing the course seemed to have a slightly better handle on using and creating classes of objects. This is probably due the longer expossure at using them. The tradeoff is the delayed coverage of traditional topics of strings and arrays.

ObjectDraw did not prove to be the silver bullet for improving the attrition rate in the CS-1 course. We had hoped that the graphical and event-driven nature of ObjectDraw would encourage a higher percentage of CS-1 students to continue on to CS-2. However,

we did not see any improvement for this semester. Further tracking of attrition rates is needed for later semesters.

References

1. Bruce K., Danyluk A., and Murtagh, T. (2001). "A library to support a graphics based object-first approach to CS 1." *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*. pp. 6-10.

2. Bruce K., Danyluk A., and Murtagh, T. (2001). "Event-driven programming can be simple enough for CS 1." *ITiCSE 2001 Proceedings*. pp. 6-10.

3. Fienup, M. (Sept. 1996) "Rethinking the CS-2 Course with an Object-Oriented Focus." *SIGCSE Bulletin*. pp. 23-25.

Appendix A. Summary of Graphic Objects and Methods

Constructors for Auxilliary Classes

`new Color(redness, greenness, blueness);`	Mix a new color. Parameter values are numbers between 0 and 255.
`new Location(x,y);`	Build a coordinate pair object for the point (x,y).

Accessor Methods for Auxilliary Classes

`someColor.getRed()`
`someColor.getGreen()`
`someColor.getBlue()`
	Access any of the color values associated with a Color.

`someLocation.getX()` `someLocation.getY()`	Access either of the elements of a coordinate pair.
`someLocation.distanceTo(anotherLocation)`	Determine the distance between two points.

Constructors for Graphic Objects

The parameters to a rectangle or oval constructor describe the rectangle bounding the object to be drawn. You can either:

- Specify the coordinates of the rectangle's upper left corner together with the width and height, or
- Specify the coordinates of two opposite corners.

You can fill these shapes or just frame their perimeters.

```
new FramedRect( x, y, width, height, canvas)
new FilledRect( x, y, width, height, canvas)
new FramedOval( x, y, width, height, canvas)
new FilledOval( x, y, width, height, canvas)

new FramedRect( corner1Location, corner2Location, canvas)
new FilledRect( corner1Location, corner2Location, canvas)
new FramedOval( corner1Location, corner2Location, canvas)
new FilledOval( corner1Location, corner2Location, canvas)

new FramedRect( cornerLocation, width, height, canvas)
new FilledRect( cornerLocation, width, height, canvas)
new FramedOval( cornerLocation, width, height, canvas)
new FilledOval( cornerLocation, width, height, canvas)
```

A line is described by giving its end points.

```
new Line( startX, startY, endX, endY, canvas);
new Line( startLocation, endLocation, canvas);
```

A text object is specified by the coordinates of its upper, leftmost point.

```
new Text( "some message", x, y, canvas);
new Text( "some message", baseLocation, canvas);
```

Methods Available for All Graphic Objects

`someObject.move( xOffset, yOffset);`	Move an object relative to its current position.
`someObject.moveTo( x, y);` `someObject.moveTo( someLocation);`	Move an object to point specified by coordinates.
`someObject.contains( someLocation);`	Determine if an object's bounding box contains a point.
`someObject.hide();` `someObject.show();`	Make an object invisible or visible on the display.
`someObject.removeFromCanvas();`	Delete object from its canvas.

`someObject.sendForward();`
`someObject.sendToFront();`
`someObject.sendBackward();`
`someObject.sendToBack();`
	Alter the stacking order that controls how overlapping objects appear.

`someObject.getColor();` `someObject.setColor( someColor );`	Access or change an object's color.

Methods Available for All 2-D Graphic Objects (including Text, but not Line)

`someObject.getX();`
`someObject.getY();`
`someObject.getLocation();`
	Access coordinates of the upper left corner of an object's bounding rectangle.

`someObject.getWidth();` `someObject.getHeight();`	Access the dimensions of an object's bounding rectangle.

Methods Available for Resizable 2-D Graphic Objects (not Text)

`someObject.setWidth( newWidth);` `someObject.setheight( newHt);`	Change the dimensions of an object's bounding rectangle.

Mutator Methods for Lines

`someLine.setStart( someLocation );`
`someLine.setEnd( someLocation );`
`someLine.setEndPoints( startLocation, endLocation);`
	Change either or both of a line's end points.

Mutator Methods for Text Objects

`someText.setText( "new message" );`	Change the characters displayed.
`someText.setFontSize( pointSize );`	Change the font size used.

`someText.setBold();`
`someText.setItalic();`
`someText.setPlain();`
	Change the style in which text is displayed.

`someText.setFont( someFont )`	Change the font used.