The UBC Java graphics tutorial—a simple interactive applet

In addition to animation, applets can offer you interaction. Very complicated things are possible, but even simple ones will enable you to make mathematical demonstrations more interesting. Here is a very simple applet extending the still picture we drew earlier.

```
package psapplets;

import java.applet.*;
import java.awt.*;
import psplot.*;
import real.*;

class psInteractiveCanvas extends PlotCanvas {

    controlNode cn;
    RealPoint corner = new RealPoint(0.5, 0.5);

    public psInteractiveCanvas(Applet app, int w, int h) {
        super(app, w, h);
        double ht = 3;
        // preserve aspect ratio
        double wd = w*(ht/h);
        setCorners(-wd, -ht, wd, ht);
        cn = new controlNode(this, corner, Color.red);
    }

    public void draw() {
        PlotPath p;
        p = new PlotPath(this);
        p.moveto(-corner.x, -corner.y);
        p.lineto( corner.x, -corner.y);
        p.lineto( corner.x,  corner.y);
        p.lineto(-corner.x,  corner.y);
        p.close();
        p.fill(Color.red);
        p.stroke(Color.black);
        cn.draw();
    }

    boolean active = false;

    public boolean mouseDown(Event evt, int x, int y) {
        // if (x, y) inside a node set mouseState = MouseActive
        if (cn.contains(x, y)) {
            active = true;
        }
        return true;
    }

    public boolean mouseDrag(Event evt, int x, int y) {
        if (active) {
            RealPoint q = toRealPoint(x, y);
```

```
            corner.x = q.x; corner.y = q.y;
            cn.locate(q);
            repaint();
        }
        return true;
    }

    public boolean mouseUp(Event evt, int x, int y) {
        // set nodes inactive
        active = false;
        return true;
    }
}

public class psInteractiveApplet extends Applet {
    psInteractiveCanvas p;

    public void init() {
        p = new psInteractiveCanvas(this, bounds().width, bounds().height);
        add(p);
        show();
    }
}
```

The last part has not changed except for a name or two. The new things are the variable `cn` and the procedures referring to mouse actions.

A **control node** is one of the objects in the `psplot` package. Graphically, it appears as a rather small square on the canvas. It is used to respond to certain mouse actions. The line

```
cn = new controlNode(this, corner, Color.red);
```

initializes `cn`. Its image will be located at the upper right corner of the rectangle we are drawing, and its inside color will be red. The word 'this' refers to the canvas we are using it in, so it knows where to draw itself. Note that in the `draw()` procedure the control node is drawn last so it isn't hidden.

The three entirely new procedures here are `mouseDown`, `mouseDrag`, `mouseUp`. These are called automatically when one of the three types of mouse event occur. The variables x, y are the coordinates of the event, in the original Java integer coordinate system. The `mouseDown` procedure just recognizes that the mouse button has been pressed inside the control node. The `mouseDrag` event is where the only interesting things occur—basically, the corner of the figure we are drawing tracks the pressed mouse button as it moves around. This stops when the button is released. The procedure `paint()` that is called is defined in the basic class `PlotPath` to call `draw()` and display the picture drawn on the screen.

There is one technical thing—events like these return a boolean variable. If this is `false`, components in which the canvas sits may also process the mouse action with their own mouse event procedures. If it returns `true`, the processing stops with the canvas.

At this point, in order to understand the `psplot` and `real` packages better, you can browse through the API `.html` files in their directories. This sort of documentation is easy to make with Java. I have not done a very thorough job with thisAPI, but it will get better as time proceeds.