2. Add the missing functionality to remote operation. In particular: the option code needs to handle more colors and needs to transmit the fill flag. DrawText needs to send the appropriate events. While implementing this, think about remote object invocation in Java and what it lacks. Note that you have to implement remote method invocation yourself; a distributed object model would let you invoke remote objects transparently, without all the fuss of encoding data into a stream.

3. The drawRender class puts a block of pixels on the screen; currently this is just a colored circle. Make the drawRender class do something more interesting.

4. Extend the program to draw splines. Make a class DrawSpline that subclasses DrawObject and add a button to activate it. The simplest approach is to collect four mouse clicks and then draw the Bezier spline that uses these points as control points. Hint: define an array of four ints with "int xpts[] = new int[4];" for the x and y control points. You can use graphics.drawLine to draw the line segments.

Advanced: make the current spline editable. When you get a mouseDown, see if it is close to any of the control points. If so, let the user drag the control point around and redraw the spline dynamically. Note: this gets tricky if you want to avoid messing up the screen. The xor trick won't work well since you're likely to draw some pixels twice. One solution: save the current drawing image before you start displaying the spline. Then, whenever you update the spline, draw the old background and then the spline. Code to do this:

```
class DrawSpline extends DrawObject {
Image saveImg = null;
public DrawSpline(...) {
   // We get an image buffer of the proper size
   saveImg = parent.createImage(parent.size().width, parent.size().height);
   // Tricky bit: we tell Draw to update our image rather than the screen image
   parent.update(saveImg.getGraphics());
}
void drawTheSpline() {
   graphics.drawImage(saveImg,0,0,null); // redraw what was there before
   // now render the spline segments with graphics.drawLine(...);
}
}
```

Draw: this is the top-level class and extends the Java Applet class. This class provides the top-level window and the user interface. The init() method sets things up, and then the class just waits for user-interface events.

DrawingCanvas: this class extends the Java Canvas class. It provides the area in which you draw. It provides a double-buffered drawing area (through backingImg and backingGraphics). Mouse and key events are passed to an object that does the actual drawing.

DrawObject: this is the superclass that does most of the work. A DrawObject takes mouse events and draws the appropriate stuff on the DrawingCanvas. In addition, this class provides the remote drawing support. This class is subclassed to provide the different operations: DrawLine, DrawCircle, etc. Each of these classes must implement the mouseDown, mouseUp, mouseDrag, and keyPress methods and do the appropriate thing.

DrawConnection: this class sits around and waits for incoming remote drawing invocations. It extends the Java Thread class, so it runs as a separate thread; otherwise the program would hang while we wait for incoming data.

The user interface

The user interface is defined in Draw. The screen is divided into three parts: a Panel of buttons (buttonpanel) to the left, the DrawingCanvas to the right, and a Panel of option controls on the bottom. The buttonpanel contains buttons for the different drawing modes. The optionpanel contains a Choice to select the color, a Checkbox to select fill mode, a TextField to specify the network connection, and various Labels.

Network connections

To connect multiple drawing applets together, things get a little tricky because of the Java security model. An applet can't listen on random sockets, but a stand-alone application can. So I've made a stand-alone application to do the listening and tie the servers together.

Now, what you do is have all the draw programs connect to "chatServer"; each one sends drawing operations to the chatServer and the others all receive these operations. To link up, type the hostname in the "connection" window in the draw window and hit return.

Note: an applet running with appletviewer can connect to any host, but an applet running under Netscape can only connect to the same host that is providing the applet. I.e., if you're loading this page locally and it says "//localhost/...", you have to specify "localhost" as the hostname. If you're getting the pages off foo.bar.edu, you have to run chatServer on foo.bar.edu and specify foo.bar.edu for the hostname.

The Assignment

1. The drawing program currently only supports three colors. Add support for additional colors. You will have to do more addItem calls to add the colors to the Choice component. You will also need to update the event handling code to do the right thing when these are selected. Finally, you need to add a setColor call to DrawCircle and DrawText to make color selection work here.

Java Assignment

In this assignment, you modify a simple networked drawing program

Running the demo program

Standalone

1. Save the HTML file as draw.html and the java file as Draw.java.
2. Compile Draw.java with "javac Draw.java".
3. Run the applet with "appletviewer draw.html".
4. Click on buttons to select drawing modes and draw with the mouse.
5. You should also be able to run the applet by loading draw.html into Netscape.

Networked

The drawing program also has a mode where you can connect multiple drawing applets together. Anything drawn in one applet will appear in all the others. To do this, we use a helper program. This helper program accepts connections on port 4321. Any input on a connection is echoed to all the other connections. Since this is like a simple "chat" program, I call it "chatServer", although we'll be using it to broadcast graphics events rather than text.

1. Save the chatServer program in chatServer.java.
2. Compile it with "javac chatServer.java".
3. Run it with "java chatServer". Note that it runs as a standalone program, not an applet.
4. To see what chatServer does, do "telnet localhost 4321" from a couple windows (or "telnet *name-of-host* 4321" from a different machine). Note that anything you type into one session is echoed to the rest. Exit telnet with control-] and quit.
5. With chatServer still running, start up a couple drawing applets ("appletviewer draw.html")
6. Connect each applet to chatServer by hitting "return" in the connection window. If the applet is running on a different machine from chatServer, you need to explicitly enter the hostname.
7. Now draw in one applet, and the results should be echoed to the other applets.
8. Kill the chatServer.

The program internals

The program is a drawing applet that lets you draw lines, ovals, text, and colored circles on the screen. In addition, it allows you to connect multiple drawing programs together over the network to provide a collaborative drawing environment.

The program is structured as multiple object-oriented classes: