Lab 10 – Shared Scribbling (Bonus Lab!)
CSCI 1101B – Spring 2015

Objective: To learn the basics of networked programs.

In this (bonus!) lab, you will write a simple networked program that allows you to scribble on your (and your classmates') screens. Your program will allow you to draw arbitrary shapes in your window using the cursor like pen – but everything you draw will also appear on your classmate's screens, and vice versa!

1 Basic Scribbling

First, start by writing a program in the `SharedScribble` class that allows you to scribble on your own screen. You can scribble arbitrary lines on your screen by simply drawing a series of lines from one point to the next as you draw your mouse across the screen. This is quite similar to dragging an object as we've seen before, except instead of moving an existing object, you are just creating new `Line` objects between points as you drag.

2 Networking Basics

To allow for shared scribbling, your program is going to communicate with another computer (the instructor machine). In networking, the computer that you connect to is called the **server**. For example, every time you go to a web page like Google, you are connecting to a web server. The machine that you are connecting from (in this case, your own laptop or lab machine) is called the **client**. Every time you open up a web browser, your browser is acting as the client talking to the web server. The basic process of networked communication is that a client sends messages to a server, and the server responds with messages back to the client.

Messages can be arbitrary data – you can think of them like `Strings`. An important question is the following: how do we know what messages to send to the server, or what messages we will get back from the server? The answer is that we use a **protocol**. A protocol is simply a set of rules determining what kinds of messages are acceptable, and what those messages mean.

3 The SharedScribble Protocol

For our shared scribbling service, we will use a very simple protocol. The SharedScribble protocol consists of only two message types that you send to the server: `draw` messages and `update` messages.

1. **Draw** messages are used to inform the server that we wish to draw a new line in the window. Remember that a line simply consists of a start location and an end location, each of which consists of an X and Y coordinate. As such, the format of a `draw` message is the following:

   ```
   draw [startX] [startY] [endX] [endY]
   ```

   For example, a draw message that you send might consist of the following:

   ```
   draw 32 53 80 65
   ```

   This message says to draw a line from the location (32, 53) to the location (80, 65).

   The response from the server to this type of message is either "`ok`" if the line was successfully drawn, or "`bad request`" if the request was not properly formed (for example, missing one of the coordinates).

2. **Update** messages are used to inform the server that we wish to fetch the lines that **others** have drawn from the service. As the server draws lines that have been sent from clients, it is storing them all in an array, thus giving each line an index. The format of `update` messages is the following"

   ```
   update [index]
   ```

   This message says to fetch all lines from the server **starting from** the given index. For example:

   ```
   update 121
   ```

   This messages say to get all lines starting from the 120th line.

   The response from the server to this type of message is either "`bad request`" if the request was not properly formed, or a message describing the set of lines. The message describing the set of lines is formatted as follows:

   ```
   [startX] [startY] [endX] [endY],[startX] [startY] [endX] [endY], ...
   ```

   In other words, the response consists of sets of four numbers (each separated by a space), with each set of four numbers separated by a comma. For example, a response describing two lines might be the following:

   ```
   20 25 33 30,5 22 30 8
   ```

   This message describes the line from (20, 25) to (33, 30) and the line from (5, 22) to (30, 8).

4 Implementation

Now that we have described the SharedScribble protocol, you should extend your program so that it can use the protocol. I have provided a helper class called `NetConnection` that allows you to easily set up a network connection. To create a NetConnection object, just pass the name of the server as an argument (which will be given in lab). You can then call `sendMessage(message)` on the connection object to send a message, which will return either the response to the message, or `null` if the message could not be sent for any reason.

You can also check whether the connection has been successfully established using the `isConnected` method.

Modify the `SharedScribble` method to send a message to the server whenever you draw a new line. Remember that the message will need to follow the protocol specified above (using `draw` messages). This should make new lines appear on other machines running the server.

Once that's working, to get lines drawn from others, add a second class called `ScribbleUpdater`. This should be an `ActiveObject` class that simply periodically send an `update` message to the server and draws all the lines returned from the server. To do this, you will need to process the response message to figure out the coordinates, then construct `Line` objects appropriately. You can use the `index` component of the update messages to ensure that you are not requesting the same lines from the server repeatedly.

Useful methods in processing the response String are the `split` method of the String class and `Double.parseDouble`, which lets you turn a numeric String into an actual numeric variable (i.e, a double).

Once both components are working, you should be able to scribble on the screens of your classmates and see their scribbles on your own machine!