CS 51 Laboratory # 12
Pictionary

Objective: To gain experience with Streams.

—

Networked Pictionary This week's assignment will give you the opportunity to practice working with *Streams* in the context of a networked Pictionary program. You are encouraged to work in pairs on this program. You must do *all* of the coding together. Only turn in one copy of the program, but make sure both of your names are on the folder and at the top of each class.

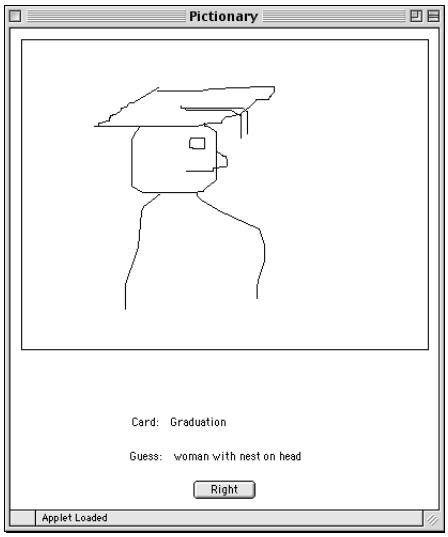
In Pictionary, one player selects a word or phrase to act out, and another player must guess what the word or phrase is. The only way the player may communicate with the guessing player, though, is by drawing: the one player draws a picture representing the word or phrase, while the other player tries to guess it.

Our networked Pictionary program allows two players to play the game. One is the *Drawer*; the other is the *Guesser*. When the program starts up, a window with two buttons appears. The buttons are labeled "Drawer" and "Guesser" to allow the player to select which they will be.

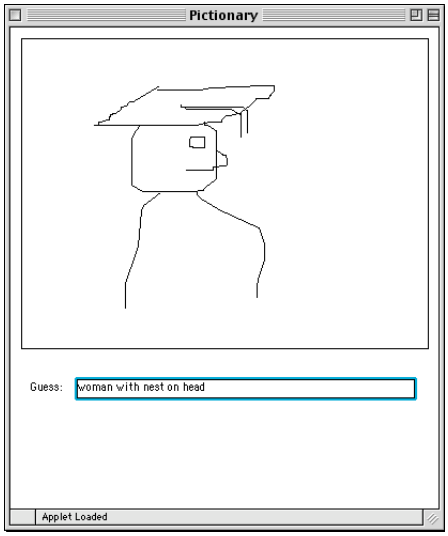
If the player selects *Drawer*, he or she is asked to select a file from which a card is picked. The card tells the player what to draw. The *Drawer*'s window contains a drawing canvas, the card display, and another label for the *Guesser*'s guess. The *Drawer* then draws their representation of the word/phrase with the mouse. Guesses appear in the *Drawer*'s window when made by the *Guesser* (with a return at the end). When the *Guesser* makes the right guess, the *Drawer* clicks on the "Right" button to notify the *Guesser* and to end the game.

If the player selects *Guesser*, the window shows a drawing canvas as well as a text field into which the player can type his or her guess. The *Guesser* cannot draw on the canvas. Instead, the *Drawer*'s drawing appears in this canvas. At any point, the *Guesser* can type a guess in the text field. When the *Guesser* hits the return/enter key, the guess is sent to the *Drawer* and appears on the *Drawer*'s screen.

Since this is a networked program, we do not provide a demo online. Below are shapshots of a game in progress. The first display shows the *Drawer*'s window. The *Drawer* is attempting to draw "graduation."



The next display shows the *Guesser*'s window. The *Guesser* thinks that the drawing looks like a woman with a nest on her head.



Organization Our implementation comprises four classes:

1. `Pictionary`, which is a WindowController.
2. `Cards`, which handles the reading of the cards from file.
3. `Guesser`, which represents the guesser in the game.
4. `Drawer`, which represents the drawer in the game.

We have completely implemented the `Pictionary` and `Guesser` classes, and most of `Cards` and `Drawer`. We have placed comments in the `Cards` and `Drawer` classes indicating where you should add code and what that code should do.

How to Run the Program Once your program has been written, you will begin by running two versions of the program by just selecting "run" twice. When both versions are open, move the top-most window so that you can see both windows. You will need to make one of these the Drawer and the other the Guesser. Since the Drawer is the server, you **must** start the Drawer first. You can do this by selecting the "Drawer" button in one of the windows. (If you mistakenly start the Guesser first, the System console will display the error message "Could not connect to server".) The Drawer will ask you to find the "cards.txt" file. It can be found inside your starter directory. Now select the "Guesser" button on the *other* window. You are now ready to play. The game ends when the Drawer clicks the "Right" button.

The Code The `Cards` class
Begin by filling in the details of the `Cards` class. We have implemented everything except the part of the constructor which reads the card information from the `cardFile` and loads each line into an element of the cards array. We have provided a `BufferedReader` for you to use. You should read each successive line from this `BufferedReader` into a new element of the array cards.

You will be done when you either run out of stuff to read (`readLine()` returns null) or you've filled the array. When you are done, the variable `numCards` *must* contain the total number of cards. Make sure this is accurate as it will be used by `shuffle`. [Yes, we know, we don't really need to shuffle since we only ever pull one card from the deck. We just wanted you to see how it could be done!]

The `Drawer` class

The `Drawer` acts as a server in a client/server relationship. Its constructor sets up a `ServerSocket`, `drawingServer`, and then waits to accept a connection from the client. Once this has happened, the `Drawer` will create an input stream, `guessStream`, and an output stream, `out`, in order to communicate with the `Guesser`. The `writeLine` method should send along the output stream the `Line` segments that should be drawn on the Guesser's canvas. (The `writeLine` method is called from the `onMouseDrag` method of `Pictionary.java`.) From the input stream it will accept guesses.

We have implemented everything but the `run()` and `writeLine` methods. The `writeLine` method's only job is to write its `Line` parameter to the network output stream so that `Guesser` can pick it up and write it to its canvas. If anything goes wrong, just call the `close()` method to close up all open streams.

Because the `Line`s being sent to the `Guesser` are `Object`s, it is appropriate to send them through the `ObjectOutputStream`, `out`, created in the constructor. Unlike the examples in class, you should not use a `BufferedOutputStream` because we want the Lines to appear as soon as they are produced, not being passed a handful at a time as a `BufferedOutputStream` would do.

Guesses are `String`s. We made `guessStream` a `BufferedReader`, as that will handle `String`s appropriately.

Getting Started Read through the `Card` and `Drawer` classes, noting the lines that ask you to add code. Those are the *only* places where you will need to make changes to the files. You may also find it useful to refer to the networked Drawing Panel example from class and also look at the `Guesser` class to see exactly what it is expecting to receive and send on its end of the streams.

The starter code is set up to allow both the `Drawer` and `Guesser` to be on the same computer. While it is beyond the requirements of the assignment, you might enjoy actually getting them to play on different computers. To do that, go into the `Guesser` class and change the `Socket` constructor call. Replace "`localhost`" with the Internet (IP) address of another machine.

To find the name of another computer in the lab, go to that machine and click on the apple icon at the top left of the screen. Click on "About this Mac", then "More Info ...". On the left of the window that appears, click on "Network". Look at the top right corner of the window in the line labeled "Ethernet" and write down the number in the last column, which should look something like "134.173.66.240". This number should replace "localhost" in your code. Be sure to place quotation marks around the number so it is treated as a string. For example, on this computer, I would replace "localhost" by "134.173.66.240".

Please be sure, however, that the code you turn in uses "`localhost`". If you pass a string that is not a valid computer name, when you start the `Guesser`, you will get the message "Unknown host". Remember that the `Drawer` must start first since it is the server.

Submitting Your Work Your program is due on Monday at 11 p.m. When your work is complete, follow the same procedure for submitting as usual. Make sure the folder name begins with your name(s) and includes the phrase "Lab12." Also make sure that the `Drawer.java` and `Card.java` files include a comment containing your name(s).

Before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments.

Table 1: Grading Guidelines

Value	Feature
	Syntax Style (4 pts total)
1 pt.	Descriptive comments
1 pt.	Good names
1 pt.	Good use of constants
1 pt.	Appropriate formatting
	Semantic style (3 pts total)
1 pt.	conditionals and loops
1 pt.	General correctness/design/efficiency issues
1 pts.	Parameters, variables, and scoping
	Correctness (3 pts total)
1 pt.	sending Lines
1 pt.	receiving guesses
1/2 pt.	reading in data to cards array
1/2 pt.	closing up socket and streams

Pictionary.java

```
import objectdraw.*;
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.*;

// This plays a Pictionary game on 2 computers.  The drawer draws on one
// computer, while the guesser makes guesses on a second computer.
public class Pictionary extends WindowController implements ActionListener {
   // Size and location of the drawing area
   private static final int DRAWING_WIDTH = 380;
   private static final int DRAWING_HEIGHT = 290;
   private static final int DRAWING_OFFSET = 10;

   // Port for sending pictionary data
   private static final int PICTIONARY_PORT = 1340;

   // UI component that displays the text of the next card.
   private JLabel cardLabel;

   // UI component where the guesser enters the guess
   private JTextField guess;

   // UI component where the drawer sees the guess
```

```
private JLabel guessLabel;

// Remember starting point of the next line segment
private Location nextLineStarts;

// The deck of cards
private Cards deck;

// Remembers if the user can draw.
private boolean isDrawer = false;

// JPanel to add UI components to under the drawing area
private JPanel gamePanel = new JPanel();

// Buttons to identify which player will draw and which player will guess
private JButton drawerButton, guesserButton;

// Button to signal that the guesser got it right
private JButton rightButton;

// Panels seen by the drawer
private JPanel cardPanel, guessViewPanel, buttonPanel;

// JPanel seen by the guesser
private JPanel guessPanel;

// The drawer object
private Drawer drawer;

// Initialize the display with the drawing area and buttons to identify
// the drawer and the guesser.
public void begin () {
   // Draw a border around the drawing area.
   new FramedRect (DRAWING_OFFSET, DRAWING_OFFSET,
               DRAWING_WIDTH, DRAWING_HEIGHT, canvas);

   // Draw the common parts of the drawer and guesser UI
   gamePanel.setLayout (new GridLayout (0, 1));

   buttonPanel = new JPanel();
   drawerButton = new JButton ("Drawer");
   buttonPanel.add (drawerButton);
   drawerButton.addActionListener (this);

   guesserButton = new JButton ("Guesser");
   buttonPanel.add (guesserButton);
   guesserButton.addActionListener (this);
```

```
      // Create the drawer and guesser specific parts of the UI, but keep
      // them hidden initially.
      createGuesserDisplay();
      createDrawerDisplay();

      // Put the UI on the screen.
      add (gamePanel, BorderLayout.SOUTH);
   }

   // Initialize the guesser's display.  The guesser has a text field
   // to make guesses in.  This is not shown initially.
   public void createGuesserDisplay () {
      guessPanel = new JPanel();
      guessPanel.add (new JLabel ("Guess:  "));
      guess = new JTextField (20);
      guessPanel.add (guess);
      gamePanel.add (guessPanel);
      guessPanel.setVisible(false);
   }

   // Show the guesser specific parts of the UI
   public void guesserDisplay() {
      guessPanel.setVisible(true);
   }

   // Iniitialize the drawer's display.  The drawer has a field showing
   // the card so the drawing user knows what to draw.  A second field shows
   // the guess made by the guesser.  There is also a button for the drawer
   // to click when the guesser guesses right.  These are all hidden initially.
   public void createDrawerDisplay () {
      // Create the display
      cardPanel = new JPanel();
      cardPanel.add (new JLabel ("Card:  "));
      cardLabel = new JLabel ("                        ");
      cardPanel.add (cardLabel);

      gamePanel.add (cardPanel);
      guessViewPanel = new JPanel();
      guessViewPanel.add (new JLabel ("Guess:  "));
      guessLabel = new JLabel ("                      ");
      guessViewPanel.add (guessLabel);
      gamePanel.add (guessViewPanel);

      rightButton = new JButton ("Right");
      buttonPanel.add (rightButton);
      rightButton.addActionListener (this);
      gamePanel.add (buttonPanel);
```

```
      cardPanel.setVisible(false);
      guessViewPanel.setVisible (false);
      rightButton.setVisible(false);
   }

   // Show the UI components for the drawer.
   public void drawerDisplay() {
      cardPanel.setVisible(true);
      guessViewPanel.setVisible(true);
      rightButton.setVisible(true);

      // Create the deck of cards and show the first card.
      deck = new Cards ();
      cardLabel.setText (deck.nextCard());
      this.validate();
   }

   // Handle clicks of the drawer, guesser, and right buttons.
   public void actionPerformed (ActionEvent evt) {
         if (evt.getSource() == drawerButton) {
            // When the drawer button is clicked, hide the drawer and
            // guesser buttons and make the drawer UI components visible.
            drawerButton.setVisible(false);
            guesserButton.setVisible(false);
            drawerDisplay();
            buttonPanel.validate();

            // Remember that this user is allowed to draw.
            isDrawer = true;

            // Create the drawer and establish communication with the guesser.
            drawer = new Drawer (PICTIONARY_PORT, guessLabel);
         }

         else if (evt.getSource() == guesserButton) {
            // When the guesser button is clicked, hide the drawer and
            // guesser buttons and make the guesser UI components visible.
            drawerButton.setVisible(false);
            guesserButton.setVisible(false);
            guesserDisplay();

            // Create the guess and establish communication with the drawer.
            new Guesser (PICTIONARY_PORT, canvas, guess);
         }

      else if (evt.getSource() == rightButton) {
         // When the write button is clicked, disable any more drawing.
         // Close the connection between the drawer and the guesser.
```

```
         isDrawer = false;
         drawer.quit();
      }
   }

   // Begin drawing when the user depresses the mouse button
   public void onMousePress(Location point){
      nextLineStarts = point;
   }

   // Draw a line segment as the user drags the mouse with the button down.
   // Have the drawer transmit the segment to the guesser.
   public void onMouseDrag(Location point){
      if (isDrawer) {
         Line newLine = new Line(nextLineStarts, point, canvas);
         nextLineStarts = point;
         drawer.writeLine (newLine);
      }
   }

}
```

Guesser.java

```
import objectdraw.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

// This class manages the network communication for the user who is making
// guesses.
public class Guesser extends ActiveObject implements ActionListener {
   // Location and size of winning message
   private static final int WINNING_MSG_OFFSET = 50;
   private static final int WINNING_MSG_SIZE = 18;

   // The stream on which the Lines are received as the drawer draws them.
   private ObjectInputStream in;

   // The stream on which guesses are sent as the guesser enters them.
   private PrintWriter out;
```

```
// The canvas where the new Lines should appear.
private DrawingCanvas canvas;

// The field where the guesser types the guesses
private JTextField guessField;

// The socket used for communicating with the server.
private Socket clientSocket;

// Create a new guesser and establish communication with the drawer.
// Parameters:
//      pictionaryPort - the port on the server to connect to
//       theCanvas - the canvas to draw Lines on
//       theGuessField - the UI component where the user types guesses
public Guesser (int pictionaryPort, DrawingCanvas theCanvas,
                JTextField theGuessField) {
   canvas = theCanvas;
   guessField = theGuessField;
   guessField.addActionListener (this);
   try {
      // Establish communication with the server.  For now,
      // these are assumed to run on the same machine.
      clientSocket = new Socket ("localhost", pictionaryPort);

      // Initialize the stream to receive Lines on
      InputStream clientInStream = clientSocket.getInputStream();
      in = new ObjectInputStream (clientInStream);

      // Initialize the stream to send guesses on.
      out = new PrintWriter (new OutputStreamWriter (clientSocket.getOutputStream()));
   } catch (UnknownHostException e) {
      System.out.println ("Unknown host.");
      return;
   } catch (IOException e) {
      System.out.println ("Could not create client.");
      close();
      return;
   }

   start();
}

// Receive Lines until the Line stream closes.
public void run ()  {
   // Next line received on the stream.
   Line nextLine;

   try {
```

```
         while (true) {
            // Get a line from the stream and put it on the canvas.
            nextLine = (Line) in.readObject();
            nextLine.addToCanvas (canvas);
            canvas.repaint();
         }
      } catch (IOException e) {
         // Assume the connection is closed because the guesser was right, not because
         // of a network problem.  Display a congratulatory message.
         Text rightText = new Text ("You're right!!", WINNING_MSG_OFFSET,
                                                      WINNING_MSG_OFFSET, canvas);
         rightText.setFontSize (WINNING_MSG_SIZE);
      } catch (Exception e) {
         System.out.println (e);
      }
      close();
   }

   // React to carriage returns in the guess field by sending the guesses
   // to the drawer.  Extra carriage return and flush needed for some reason.
   public void actionPerformed (ActionEvent evt) {
      out.println (guessField.getText());
       out.flush();
   }

   // Clean up all the open streams
   private void close () {
      if (out != null) {
         out.close();
      }

      try {
         if (in != null) {
            in.close();
         }
      } catch (IOException e) {
            // ignore it if can't close in
      }

      try {
         if (clientSocket != null) {
            clientSocket.close();
         }
      } catch (IOException e){
           // ignore it if can't close clientSocket
      }
   }
}
```

Cards.java

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

// Cards manages the deck of cards used to play Pictionary.  The file
// is a text file.  Each line represents a different card.
public class Cards {
   // Maximum deck size
   private static final int MAX_DECK_SIZE = 20;

   // The cards
   private String[] cards = new String[MAX_DECK_SIZE];

   // The number of cards in the deck.
   private int numCards;

   // The next card to return in the deck
   private int next = 0;

   // Construct a new Cards object.  Ask the user for a file containg cards.
   // Read in the cards to initialize the array.  Shuffle the cards.
   public Cards () {
      // Set up the input stream.
      File cardFile = getDeckFile();
      try {
         BufferedReader in = new BufferedReader (new FileReader (cardFile));

         // -- Insert code here to read in the cards. Stop when readLine() returns null.
         // -- Be sure numCards contains the number of cards in deck after reading.

         // Shuffle the cards
         shuffle();

      } catch (FileNotFoundException e) {
         // If the card file can't be found, just create a deck with one card.
         cards[0] = "cecil sagehen";
         numCards = 1;
      } catch (IOException e) {
         // If we get an error reading the file, make sure the deck has at least one
         // card and then shuffle what we did read in.
         if (numCards == 0) {
            cards[0] = "cecil sagehen";
            numCards = 1;
         } else {
            shuffle();
         }
```

```
    }
  }

    // Using a file dialog, get the name of a file to load
    // as the deck of cards.  Return null if the user cancels the
    // dialog box
  private File getDeckFile() {
    String fileName;

    FileDialog dialog = new FileDialog( new Frame(),
                                "Select the file of cards.",
                                 FileDialog.LOAD );
    dialog.setVisible(true);

    if ( dialog.getDirectory() != null ) {
       fileName = dialog.getDirectory() + dialog.getFile();
        return new File (fileName);
     } else {
        return null;
     }
  }

  // Shuffle the cards
  private void shuffle() {
    RandomIntGenerator random = new RandomIntGenerator (0, numCards - 1);

    // The indexes of the cards to swap
    int first, second;

    // Number of swaps to do in the shuffle
    int numSwaps = 100;

    // Temporary string for swapping
    String temp;

    // Swap pairs of random cards repeatedly to shuffle
    for (int i = 0; i < numSwaps; i++) {
       first = random.nextValue();
       second = random.nextValue();
       temp = cards[first];
       cards[first] = cards[second];
       cards[second] = temp;
    }
  }

  // Returns the next card in the deck.
  public String nextCard () {
    String returnCard = cards[next];
```

```
      next = (next + 1) % numCards;
      return returnCard;
   }
}
```

Drawer.java

```
import objectdraw.*;
import javax.swing.*;
import java.net.*;
import java.io.*;

// This class manages the communication for the user doing the drawing.
public class Drawer extends ActiveObject {
   // UI component that displays the guesser's guesses.
   private Label guessLabel;

   // Server connection
   private ServerSocket drawingServer;

   // Socket that the drawer uses to communicate with the guesser
   private Socket drawingSocket;

   // Output stream used to write Lines to the guesser.
   private ObjectOutputStream out;

   // Stream that guesses arrive on
   private BufferedReader guessStream;

   private static final int TIMEOUT = 60000;  // 1 minute

   // Create a new drawer.  The drawer acts as a server.
   // Parameters:
   //     pictionaryPort - the port that Pictionary uses for communication
   //     theGuessLabel - the UI component where guesses should be displayed
   //        when they arrive over the socket's input stream
   public Drawer (int pictionaryPort, Label theGuessLabel) {
      guessLabel = theGuessLabel;
      try {
         // Register as a server
         drawingServer = new ServerSocket (pictionaryPort);

         // Set a timeout for how long to wait for the client to connect.
         drawingServer.setSoTimeout (TIMEOUT);

         // Wait for the client to connect.
         drawingSocket = drawingServer.accept();
```

```
      // Initialize the output stream used to send Lines to the guesser.
      out = new ObjectOutputStream (
               drawingSocket.getOutputStream());

      // Initialize the input stream where the guesses arrive.
      guessStream = new BufferedReader (
                  new InputStreamReader (
                     drawingSocket.getInputStream()));
   } catch (InterruptedIOException e) {
      System.out.println ("Connection timed out.  Closing server.");
      close();
      return;
   } catch (IOException e) {
      System.out.println ("Could not create server.");
      System.out.println (e);
      close();
      return;
   }
   start();
}

// Accept user guesses on the input stream and display them in the
// guess label.
public void run () {
   // -- Repeatedly display the guesses to the drawer as they arrive.
   // -- quit when guessStream goes away (i.e., readLine() throws exception)
   // Your code here!

   // Clean up by closing the stream, socket, and shutting down the server
   // connection.
   close();
}

// Write a line onto the output stream so that the guesser can see it.
// Be sure to close() if anything goes wrong.
// Parameter
//     nextScribble - the line to pass to the guesser
public void writeLine (Line nextScribble) {
   // your code here
}

public void quit () {
   close();
}

// Close everything that is still open.
private void close () {
```

```
      try {
         if (out != null) {
             out.close();
          }
      } catch (IOException e) { }
      try {
         if (guessStream != null)
         {
            guessStream.close();
         }
      } catch (IOException e) { }
      try {
         if (drawingSocket != null)
         {
            drawingSocket.close();
         }
      } catch (IOException e) { }
      try {
         if (drawingServer != null)
         {
            drawingServer.close();
         }
      } catch (IOException e) { }
   }
}
```