CS 580 Client-Server Programming
Spring Semester, 2007
Comments on Assignment 2.1
Feb 20, 2007

Hard Coded Server

```
public void sendMessage(String message) {
        String host = "bismarck.sdsu.edu";
        int port = 8009;
```

Formating

```
private String send(String text) {
        try {
// creates a stream socket and connects it to the server
                Socket connection = new Socket(strServer, iPort);
//getOutputStream will return an output stream for this socket
                OutputStream rawOut = connection.getOutputStream();
//OutputStream is super class to BufferedOutputStream. PrintStream creates a new print stream.
//BufferedOutputStream creates a new buffered output stream to write data to the specified output stream
                PrintStream out = new PrintStream(new BufferedOutputStream(rawOut));
//getInputStream returns an inputstream for this socket
                InputStream rawIn = connection.getInputStream();
//BufferedReader creates a buffering character-input stream tht uses a default input buffer
//InputStreamReader creates an InputStreamReader that uses the default-size input buffer
                BufferedReader in = new BufferedReader(new InputStreamReader(rawIn));
                out.print(text);
//flush() will flush the stream
                out.flush();
```

Is this better

```
private String send(String text) {
        try {
                // creates a stream socket and connects it to the server
                Socket connection = new Socket(strServer, iPort);
                //getOutputStream will return an output stream for this socket
                OutputStream rawOut = connection.getOutputStream();
                //OutputStream is super class to BufferedOutputStream. PrintStream creates a new
                //       print stream.
                //BufferedOutputStream creates a new buffered output stream to write data to the
                //       specified output stream
                PrintStream out = new PrintStream(new BufferedOutputStream(rawOut));
                //getInputStream returns an inputstream for this socket
                InputStream rawIn = connection.getInputStream();
                //BufferedReader creates a buffering character-input stream tht uses a default input
                // buffer
                //InputStreamReader creates an InputStreamReader that uses the default-size input
                // buffer
                BufferedReader in = new BufferedReader(new InputStreamReader(rawIn));
                out.print(text);
                //flush() will flush the stream
                out.flush();
```

What do we lose without the Comments?

```
private String send(String text) {
        try {
                Socket connection = new Socket(strServer, iPort);
                OutputStream rawOut = connection.getOutputStream();
                PrintStream out = new PrintStream(new BufferedOutputStream(rawOut));
                InputStream rawIn = connection.getInputStream();
                BufferedReader in = new BufferedReader(new InputStreamReader(rawIn));
                out.print(text);
                out.flush();
```

Comments

```
public class Client {
        /**
         * Client constructor
         * @param server Address fo server with which to connect.
         * @param port Port number to use for connection
         */
        public Client(String server, int port) {
```

What do the comments add to the code?

Does this need comments?

```
public class Client {
	public Client(String serverAddress, int portNumber) {
```

Improved names reduce the need for the given comments

How about this comment?

```
public class Client {
	/*
	 * @param serverAddress as DNS name ie "rohan.sdsu.edu"
	 */
	public Client(String serverAddress, int serverPort) {
```

Constants

```
class Client {
        int space = 32;
        int semicolon = 59;
        String spaceString = this.toString(space);
        String semicolonString = this.toString(semicolon);

class Client {
        static final String SEMICOLON = ";";
        static final String SPACE = " ";
        static final String SPACE = new Character(32).toString();
```

Mixing UI with Domain Code

Just Say No

```
public class Client {
	public String add() {
		System.out.print("Type name to add to server");
		String name = Console.readLine();
		Socket connection = new Socket(strServer, iPort);
		OutputStream rawOut = connection.getOutputStream();
		PrintStream out = new PrintStream(new BufferedOutputStream(rawOut));
		InputStream rawIn = connection.getInputStream();
		BufferedReader in = new BufferedReader(new InputStreamReader(rawIn));
		out.print("add " + name + ';');
		out.flush();
		etc.
```

Mixing UI with Domain Code

Put UI code in separate layer

Keep Domain code ignorant of UI code

UI & Domain Code

- Perform different tasks
- Change at different rates

No need to tie doman code to particular UI

IO In Constructor

Constructor as a Function
Class as a holder of a return value

```
public class Client {
	public Client() {
		System.out.print("Type server command:");
		String command = Console.readLine();
		sendCommand( command);
	}
```

etc.

The Text UI Menu System

```
pubic class ClientTUI {
        private Client voter;
        private static ENDLINE = System.getProperty("line.separator");
        string mainMenu() {
                return "Select one of the following options:" + ENDLINE +
                "1) add" + ENDLINE +
                "2) list" + ENDLINE +
                "3) vote" + ENDLINE +
                "4) result" + ENDLINE +
                "5) quit";
        }

void add() {
        name = Console.readLine ("Enter a name to add");
        String result = voter.add( name);
        Console.println(" The result: " + result);
}

public void run() {
        while (true) {
                int method = Console.readInt( mainMenu());
                switch (method) {
                        case: 1:    add(); break;
                        case: 2:    list(); break;
                        etc.
```

Why Is Text UI done from Scratch Each Time?

Build some structure that can be reused

Why bother with Text UI?

Use Unit tests

```
import junit.framework.TestCase;
public class VotingClientTest extends TestCase {

    private VotingClient client;
    private final String testName = "Test" + new Date().getTime();

protected void setUp() throws Exception {
        client = new VotingClient();
}

public void testAddNew() {
        String response = client.add(testName);
        assertEquals("Add failed", "success;", response);
        }
}
```

readLine()

Avoid readline (what is a line?)

Does not work with protocol on server

Whtswthllthabrnsthyrhrdtrd

Flag Names

```
private String serverResponse(BufferedReader inBuffer) {
        StringBuffer response = new StringBuffer();
        boolean termTest = true;
        int currentChar = 0;
        while (termTest == true) {
                currentChar = inBuffer.read();
                if (currentChar == -1 )
                        termTest = false;
                else if((char)currentChar == ';')
                        termTest = false;
                else
                        response.append((char) currentChar);
        }
        return response.toString();
}
```

Flag Names

What role does the flag play

```
private String serverResponse(Reader fromServer) {
        StringBuffer response = new StringBuffer();
        boolean hasMoreChars = true;
        int next = 0;
        while (hasMoreChars) {
                next = fromServer.read();
                if (-1 == next)
                        hasMoreChars = false;
                else if(';' ==(char) next)
                        hasMoreChars = false;
                else
                        response.append((char) currentChar);
        }
        return response.toString();
}
```

Do we need the flag?

```
private String serverResponse(Reader fromServer) {
        StringBuffer response = new StringBuffer();
        int next = 0;
        while (true) {
                next = fromServer.read();
                if (isEOF(next))
                        return response.toString();
                else if (isEndOfMessage(next))
                        return response.toString();
                else
                        response.append((char) next);
        }
}
```

EOF as special case of EOM

```
private String serverResponse(Reader fromServer) {
        StringBuffer response = new StringBuffer();
        while (true) {
                int next = fromServer.read();
                if (isEndOfMessage(next))
                        return response.toString();
                else
                        response.append((char) next);
        }
}
```

DataInputStream & DataOutputStream

Reads & writes primitive Java data types in binary

Only use if you are reading & writing Java data types

How do you know what language the server (client) is using?