CS 580 Client-Server Programming
Spring Semester, 2007
Assignment 2 Comments
April 3, 2007

Reusing Names

```
public synchronized void processAdd(PrintWriter out, String command) {
    try {
        command = command.substring(4, command.lenght()-1);
        statement.executeUpdate("insert into contestant values (" + command +
                                                            ", 0 ,0)");
        out.println("Success");
    }
    catch ( blah) { blah}
}
```

Can you spot the two other errors?

Some improvements

```
public synchronized String processAdd(PrintWriter out, String command) {
        try {
                String name = command.substring(4, command.lenght()-1);
                statement.executeUpdate("insert into contestant values (" + name +
                                                                        ", 0 ,0)");
                return "success";
        }
        catch ( blah) { blah}
}
```

println()

Rule of thumb:

Follow syntax of protocol strictly when generating messages

Be lenient in what syntax you accept

println() generates invalid syntax for our protocol

Who knows about Ant?

When should this happen in source code?

```
}
}
```

Which is better?

```
int delimiter = 59;
```

```
int delimiter = SEMICOLON;
```

```
int delimter = 59; // 59 = semicolon
```

Defining Fields

```
public class Foo {
    int port = 6543;

    public void bar() {
        blah blah
    }

    String BDLocation;

    public void go() {
        blah
    }

    String name;
}
```

Java allows fields to be defined in many places

Always define fields in one location in a class

Always use the same location in your classes

Only use the end or the beginning of the class

System.exit()

```
public class ClientHandler extends Thread {
    blah

    public void run() {
        blah
        try {
            more blah
        }
        catch (IOException e) {
            System.exit(-1);
        }
    }
    etc.
```

System.exit() kills the program

When does it make sense to kill the server?

Config files & Command line args

For program parameters that need changing with recompiling

Which are candidates for server config files:

- port number
- database url
- database password
- database username
- maximum number of threads or connection
- socket timeout

Logging

```
try {
	blah
	blah
}
catch (IOException e) {
	What should be done here?
}
```

Options

Print message to standard out
Print message to System.err
Log an error message

Unreachable Code

```
boolean listening = true;
while (listening) {
        Socket client = server.accept();
        new ClientHandler(client).start();
}
server.close():
```

Complex Ifs

```
if (lengthargs == 4) {
    for (int i=0; i < lengthargs; i++)
        blah
        if (args[i].equalsIngnoreCase("-p") && !portSet) {
            try {
                blah
            } catch () {
                blah
            }
        }
        else if (args[i].equalsIngnoreCase("-f") && !logSet) {
            blah
        }
    }
    if (logSet && portSet)
        blah
    }
    else {
        blah
    }
}
else{
```

DataStreams

Java's DataStreams read and write objects

So a client

- can create a vote object
- use a DataOutputStream to write the object to a socket

A Server can read the object using DataInputStream

Neither client or server has to parse content from the network

However - datastreams do not use the assignment protocol

MySQL connections

How long will MySQL keep database connections open?

Why do we care?

See:

http://dev.mysql.com/doc/refman/5.0/en/gone-away.html

Loading the Driver

How many times do we have to load the database driver?

Does it hurt to load it multiple times?

Does it matter where we load the driver?

wait-notify

Which thread is woken up when notify() is called?

If it matters which thread is called use notifyAll()

Some texts recommend using notifyAll() rather than notify()

thread safety

```
while(connection = server.accept())
    thread = Thread.new() do
        handleConnection(connection)
        connectionClose()
    end
end
```

What happens if two clients connect at the same time

```
while(connection = server.accept())
    thread = Thread.new(connection) do |client|
        handleConnection(client)
        connectionClose()
    end
end
```

This version is Thread safe