

CMSC 132: Object-Oriented Programming II

Project – Terp Idol



Department of Computer Science
University of Maryland, College Park

Overview

■ Goal

- Implement online voting for a Terp Idol contest

■ Requires knowledge of

- Networking
- Streams
- Multithreading

Terp Idol Contest Rules

- The contest consists of a number of rounds
- Each round begins with a number of contestants
- Voters vote for individual contestants
 - As many times as they wish
- At end of round, contestant w/ fewest votes eliminated
- In case of ties, contestant whose name is alphabetically first is eliminated
- Votes for remaining contestants are reset to 0 at the beginning of a new round
- Last remaining contestant is the winner

Terp Idol Contest Example

■ Round 1

- Votes – Bart, Lisa, Homer, Lisa, Bart, Lisa
- Tally – Bart 2, Homer 1, Lisa 3
- Homer eliminated



■ Round 2

- Votes – Lisa, Lisa, Bart, Lisa
- Tally – Bart 1, Lisa 3
- Bart eliminated



■ Round 3

- Lisa wins (only contestant)



Terp Idol Online Voting

■ Requirements

- Must use a client-server model
- Use threads to handle multiple simultaneous clients
- Use synchronization to avoid data races

■ Classes

- Server, Client
- ContestServer, IdolClient, IdolCommandProcessor

■ General approach

1. Set up server
2. Set up clients
3. Handle client-server requests

1) Set Up Server

■ **Server class**

- **Supports multiple Clients concurrently**

- **Put each Client connection in separate thread**

- **Handles ping & disconnect**

■ **ContestServer class extends Server**

- **Starts server in separate (daemon) thread**

■ **IdolCommandProcessor**

- **Implements processing of commands**

2) Set Up Clients

■ **Client class**

- **Creates connection to Server**
- **Sets up communication streams**
- **Handles ping & disconnect**

■ **IdolClient**

- **Support Judges and Voters**
- **Different types of requests shown in the table**

Requests & Responses

Sent by	Client Sends	Server Returns
Client	PING	SERVER RUNNING <date>
	DISCONNECT	<none>
	<unknown code>	INVALID REQUEST <code>
Voter	CONTESTANTS	CONTESTANTS <names>
	VOTE <name>	VOTE <name>
	LOOKUP <name>	LOOKUP <line in data file for name>
	IMAGES <url>	IMAGES <URLs of images>
Judge	INVITE <name>	INVITE <name>
	TALLY	TALLY <names & # votes>
	NEW ROUND	NEW ROUND <eliminated contestant>

3) Handle Client-Server Requests

■ **IdolCommandProcessor**

- **Implements processing of commands**
- **All commands in the form of Strings**

■ **Note**

- **Recommend DataInputStream / DataOutputStream**
 - **To handle newline characters in data**

Other Relevant Topics

Client Programming

■ Basic steps

1. Determine server location – IP address & port
2. Open network connection to server
3. Write data to server (request)
4. Read data from server (response)
5. Close network connection
6. Stop client

Simple Server Programming

■ Basic steps

1. Determine server location - port (& IP address)
2. Create `ServerSocket` to listen for connections
3. Loop

```
while (true) {  
    Accept network connection from client  
    Read data from client (request)  
    Write data to client (response)  
    Close network connection to client  
}
```

Socket Class

- **Creates socket for client**
- **Constructor connects to**
 - **Machine name or IP address**
 - **Port number**
- **Transfer data via **streams****
 - **Standard Java I/O streams**
 - **Bytes → InputStream, OutputStream**
 - **Characters → FileReader, PrintWriter**

ServerSocket Class

- **Create socket on server**
- **Constructor specifies local port**
 - **Server listens to port**
- **Usage**
 - **Begin waiting after invoking accept()**
 - **Listen for connection (from client socket)**
 - **Returns **Socket** for connection**

Server Example

```
public class Server {  
    public static void main(String args[ ]) throws Exception {  
        ServerSocket ss = new ServerSocket(4242);  
        while (true) {  
            Socket s = ss.accept();  
            BufferedReader r = new BufferedReader(  
                new InputStreamReader(s.getInputStream()));  
            PrintWriter out = new PrintWriter(  
                new OutputStreamWriter(s.getOutputStream()));  
            String name = r.readLine();  
            out.println("Hello " + name);  
            out.flush();  
            s.close();  
        }  
    }  
}
```

Client Example

```
public class Client {  
    public static void main(String args[ ]) throws Exception {  
        String host = "localhost";  
        InetAddress server = InetAddress.getByName(host);  
        Socket s = new Socket(server, 4242);  
        BufferedReader r = new BufferedReader(  
            new InputStreamReader(s.getInputStream()));  
        PrintWriter out = new PrintWriter(  
            new OutputStreamWriter(s.getOutputStream()));  
        out.println("MyName");  
        out.flush();  
        String response = r.readLine();  
        System.out.println(response);  
        s.close();  
    }  
}
```

URL Class

- Provides high-level access to network data
- Abstracts the notion of a connection
- Constructor opens network connection
 - To resource named by URL

Creating Threads in Java

■ Runnable Approach

1. Define class implementing Runnable interface

```
public interface Runnable {  
    public void run( );  
}
```

2. Put work to be performed in run() method

3. Create instance of the “worker” class

4. Create thread to run it

■ Create Thread object

- Pass worker object to Thread constructor

■ Or hand the worker instance to an executor

- Alternative methods for running threads

Creating Threads in Java

■ Example

```
public class MyT implements Runnable {  
    public void run( ) {  
        ... // work for thread  
    }  
}
```

```
Thread t = new Thread(new MyT( )); // create thread  
t.start();                          // begin running thread  
...                                  // thread executing in parallel
```

Lock

■ Definition

- Entity can be held by only one thread at a time



■ Properties

- A type of synchronization
- Used to enforce **mutual exclusion**
- Thread can acquire / release locks
- Thread will wait to acquire lock (stop execution)
 - If lock held by another thread

Lock Example

```
public class DataRace extends Thread {
    static int common = 0;
    static Object o;           // all threads use o's lock
    public void run() {
        synchronized(o) {    // single thread at once
            int local = common; // data race eliminated
            local = local + 1;
            common = local;
        }
    }
    public static void main(String[] args) {
        o = new Object();
        ...
    }
}
```

Stream Input/Output

■ Stream

- A connection carrying a sequence of data (ordered sequence of bytes)

■ Streams can be associated with

- Files, memory, other Strings

■ Many Java classes for handling streams

- Data consisting of characters (e.g., text files)
- Data consisting of raw bytes (e.g., binary files)
- Can buffer information

■ Combining different classes

- Can define stream with desired characteristics

Using Streams

■ Opening a stream

- Connects program to external data
- Location of stream specified at opening
- Only need to refer to stream

■ Usage

1. `import java.io.* ;`
2. Open stream connection
3. Use stream → read and / or write
 - Catch exceptions if needed
4. Close stream

■ Examples

- See `fileExamples` package

Scanner Class

■ Scanner

- Read primitive types & strings from input stream
 - Including `System.in` (standard input)
- Provides methods to treat input as `String`, `Integer`...
- Supports `String nextLine()`, `int nextInt()`...
- Throws `InputMismatchException` if wrong format

Scanner Class Examples

■ Example 1

// old approach to scanning input

```
BufferedReader br = new BufferedReader( new  
    InputStreamReader(System.in));
```

```
String name = br.readLine( );
```

// new approach using scanner

```
Scanner in = new Scanner(System.in);
```

```
String name = in.nextLine( ); int x = in.nextInt( );
```

■ Example 2

- **See ScannerExample.java**

- **Note use of printf**