EBU6501 - Middleware
Week 4, Day 5: Revision - Client and Server Middleware

Gokop Goteng & Ethan Lau

To read from socket to client

- **Since in this application we are reading and writing text use a stream that has text handling capability**

```
inStream = new BufferedReader(
new
    InputStreamReader(socket.getInputStream()));
```

Returns an input stream for reading characters FROM this socket.

To write from client to socket

outStream = new PrintWriter(

new OutputStreamWriter(

socket.getOutputStream()));

Returns: an output stream for writing characters TO this socket.

```
    }
catch(Exception exc)
    {System.out.println("Error! - " + exc.toString());}
}
```

Accept method

- When the **ServerSocket receives the accept** method it **waits until a client starts up** and **requests a connection** on the **port it is listening to**
  - the *listening* port number is known to the client
- When connection successfully established
  - returns a socket object which is bound *to a new local port which is different from the port it was (or is still) listening to for connections*
  - the server communicates with the client over this new socket *so server can continue to listen on original port through the ServerSocket*

A *very* simple Web server, i.e. simple http server

- Handles only one HTTP request (!!!!!)
  - request is in form GET path&file_name HTTP/1.0
- Accepts and parses the HTTP request
- Gets the requested file from the server's file system
- Creates an HTTP response message consisting of header lines and the requested file.
- Sends the response directly to the client
  - i.e. browser

Examples of requests from the browser

We will make server listen on 6666

http://localhost:6666/myfile.html

where file.html is in same directory as the server class

or

http://localhost:6666/nep3/background.gif

where nep3 is a directory inside the directory where the server is running

- Before making a request must set the simple server running

Structure of code

```
public class SimpleWebServer
{String requestMessageLine;
      …..
 SimpleWebServer(){……}

 public static void main(String args[]){
          SimpleWebServer sws=new
                        SimpleWebServer();
                }
}
```

A simple client

```
class TCPClient {  public static void main(String argv[]) throws Exception  {
String sentence;   String modifiedSentence;
BufferedReader inFromUser = new BufferedReader( new InputStreamReader(System.in));
 Socket clientSocket = new Socket("localhost", 6789);
DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
BufferedReader inFromServer = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
 sentence = inFromUser.readLine();
outToServer.writeBytes(sentence + '\n');
modifiedSentence = inFromServer.readLine();  //blocks till reply comes - synchronous
System.out.println("FROM SERVER: " + modifiedSentence);
clientSocket.close();  } }
```

A simple server: listening

```
class TCPServer {    public static void main(String argv[]) throws Exception      {
String clientSentence;
String capitalizedSentence;
ServerSocket welcomeSocket = new ServerSocket(6789);
while(true)
{Socket connectionSocket = welcomeSocket.accept();
BufferedReader inFromClient =   new BufferedReader(new
InputStreamReader(connectionSocket.getInputStream()));
DataOutputStream outToClient = new
DataOutputStream(connectionSocket.getOutputStream());
clientSentence = inFromClient.readLine();
 capitalizedSentence = clientSentence.toUpperCase()+'\n';    // If long, blue lines would be in a thread
//and we would return immediately to the start of the while loop
 outToClient.writeBytes(capitalizedSentence);
}      } }
```

The server typically spawns threads to manage its TCP queue better

Class work:

Write a simple java program for a server that listens on port 59090. When a client connects, the server sends the **current datetime** to the client. The connection socket is created in a try-with-resources block so it is automatically closed at the end of the block. Only after serving the datetime and closing the connection will the server go back to waiting for the next client.

```
public class DateClient {
        public static void main(String[] args) throws IOException {
        if (args.length != 1) {
            System.err.println("Pass the server IP as the sole command line argument");
            return;
        }
        var socket = new Socket(args[0], 59090);
        var in = new Scanner(socket.getInputStream());
        System.out.println("Server response: " + in.nextLine());
    }
}
```

Discussion:

- **Does not handle multiple clients**; each client must wait until the previous client is completely served before it even gets accepted.
- As in virtually all socket programs, a **server socket** *just listens*, and a different, "plain" **socket**communicates with the client.
- The ServerSocket.accept() call is a **BLOCKING CALL**.
- Socket communication is always with bytes; therefore sockets come with input streams and output streams.
- But by wrapping the socket's output stream with a PrintWriter, Java will automatically convert (decode) to bytes. This means nothing is sent or received until the buffers fill up, *or* you explicitly **flush** the buffer.
- The second argument to the PrintWriter, in this case **true** tells Java to flush automatically after every **println**.
- Defined all sockets in a **try-with-resources** block so they will automatically close at the end of their block. After sending the datetime to the client, the try-block ends and the communication socket is closed, so in this case, closing the connection is initiated by the server.

EBU6501 - Middleware
Week 4, Day 5: Revision, quiz, exercises

Dr. Gokop Goteng & Ethan Lau

Quiz Questions

Provide True or False Answers with Reasons

- JavaScript is created from Java
- JavaScript is similar to PHP
- JavaScript is a type of programming languages
- JavaScript Syntax is derived from C programming language
- JavaScript is important in text manipulation because of its powerful expression language (regex) syntax
- Most mobile and web applications are developed using JavaScript

Provide True or False Answers with Reasons

- JavaScript is a static programming language
- JavaScript is must start with <head> and end with </head>
- ”var” is used to declare JavaScript variables
- JSON is language dependent
- Authorisation is to keep information secretive
- Network time protocol (NTP) stabilises the variations in time latency and hence it is very secure

JavaScript

Using JavaScript, create an object that displays student's First and Last name, as well as the exam mark for EBU6501 – Middleware. Begin with <script> to simplify the scripts.

First name: Zhang
Last name: Wang
Exam mark: 85

JavaScript

Using a function, write a simple JavaScript that calculate sum of two numbers *a* and *b*. i.e., a = 3, b = 4

JSON

Task 1: Write a single command to **create JSON Strings** from JavaScript Variable *var* shopping_cart

Task 2: Write a single command to **create Javascript variable** from JSON string *var* JSONString

EJB

Task 1: Describe the Modularity and Security features provided in EJB implementations.

Task 2: Briefly describe session, entity and message-driven beans

Pass year questions !!!