CS 390 Unix Programming Environment

Topics to be covered:
- Java Packages
- Thread Example
- Java Socket Programming
Recap

- In the previous class we discussed how to create classes & instantiate objects

- Type of Classes:
  - Abstract class – Does not implement fully. Can be instantiated
  - Interface class – Only provides methods. Cannot be instantiated
Classes contd...

- There is one more type of class
  - Static class – Just like static methods, a static class is common to all the other classes in the application
  - A static class cannot be instantiated
  - Its methods can be called as follows:
    
    ```
    Classname.method();
    ```

- Now we will continue with today’s lecture
Packages

- Package is a loose affiliation of classes
- Packages are the primary mechanism of adding additional functionality into java.
- Packages are like libraries
- One can create a package using the `package` keyword
- In order to use a package use `import` keyword
Short Package example

```java
package MyPack;

class Balance {
    String name;
    double bal;

    ...

}

Save this as Balance.java in MyPack directory
```
Using this Package

```java
import MyPack.*;
class TestBalance {
    Balance test = new Balance();
    ...
}
```
Some common Java Packages

```java
import java.io.*;
import java.awt.*;
import java.lang.*;
import java.util.*;
```

Good idea to import only libraries that you need:

```java
import java.io.*;
DataInputStream dis = new DataInputStream();
```

OR

```java
java.io.DataInputStream dis = new java.io.DataInputStream();
```
The String Class

- Used to create and manipulate strings in Java
- This is better than a null terminated sequence of characters.
- Automatic sized
- Has rich set of functions:
  - Concatenation
  - Lowercase / uppercase conversion
  - Length of string
  - Substrings

- Please refer to the O’Reily Java reference

CS390- Unix Programming Environment
Some important String Methods

- `equals` – Compares two strings, returns true if equal, false otherwise
  
  **Usage:** `str1.equals(str2)`

- `length` – returns length of a string
  
  **Usage:** `str1.length()`

- `concat` – concatenates two strings and returns the result
  
  **Usage:** `str1.concat()`
The Vector Class

- The vector class is useful as a dynamic array
- The vector class can hold any java object

```java
import java.util.*;
Vector v = new Vector();
v.removeAllElements();
v.addElement(new String("1"));
v.addElement(new String("2"));
v.addElement(new String("3"));
for(int i = 0; i <v.size(); i++)
    System.out.println(v.elementAt(i));
```
Data Conversion

- Java contains robust set of classes that help in converting from one data type to another
- See the reference book for more details
- Typical functionality includes:
  - Convert to/from bases
  - Convert to/from String
  - Equality checking
Concurrent Programs

- Concurrent programs contain one or more thread of execution.
- Supported by the Java `Thread` class and `Runnable` interface.
- To make a class behave like a thread:
  - Implement a `Runnable` interface
  - An interface is used to specify methods that your class must support
  - The `Runnable` interface requires that any class implementing this interface have a method named: `run()`
- See the following example:
Thread Example

class T1 implements Runnable {
    Parent m_p;

    T1(Parent P) {
        m_p = P;
    }
    public void run () {
        int before, after;
        Long sVal;
        double sLen;
        sLen = Math.random() * (double)500.0;
        Double rLen = new Double (sLen);
        while (true) {
            ...
        }
    }
}

CS390 - Unix Programming
Environment
Thread Example

Thread main body:
while (true) {
    before = m_p.GetValue();
    try { // defining exceptions
        Thread.sleep(rLen.longValue());
    } catch(InterruptedException e) {
        System.out.println(e.toString());
    }
    after = m_p.GetValue();
    m_p.IncrValue();
    System.out.println("("+before+","+after+")");
    System.out.flush();
}
Thread Example

class Parent {
    int m_val = 0;
    public int GetValue() {
        return m_val;
    }
    public void IncrValue() {
        m_val++;
    }
    public void RunSimulation() {
        Thread pa = new Thread(new T1(this));
        Thread pb = new Thread(new T1(this));
        Thread pc = new Thread(new T1(this));
        try {
            Thread.sleep(9000);
        } catch (InterruptedException e) {
            System.out.println(e.toString());
        }
        pa.stop; pb.stop; pc.stop;
    }
}
Thread Example

```java
public static void main(String args[]) {
    Parent P = new Parent();
    P.RunSimulation();
}
```

- Must create an instance of the parent class
  - Due to the main routine being static
- Once the parent object is created we can call the RunSimulation() method
- Must try this example and understand clearly the working of this example
- As an assignment, run this example and the other sample programs and bring the output in the next class. You might be asked to explain the working of the program.

CS390- Unix Programming Environment
Assignment No. 4 Extra Credit

- Run the above Thread Example
- Explain them and prepare a report having following things:
  - Prepare a header with description of the program
  - Create sample test cases and run the program using them
  - Print the test cases and the run for each program

CS390- Unix Programming Environment
Client- Server Applications

**Diagram:**
- Client Application
- TCP/IP Interface
- Network
- TCP/IP Interface
- Server Application

- TCP/IP provides peer to peer communication between two applications
- Client Application initiates the conversation
- Server Application accepts the conversation
- Client Application generally requests the services and the Server Application provides services for these requests

CS390- Unix Programming Environment
Application Responsibilities

- **Client Application**
  - Locate and request service from a server

- **Server Application**
  - Authentication: verifying the identity of the client
  - Authorization: determining if the client is permitted to access these services
  - Data security: Data exchanged is secure and is not revealed or compromised
Sockets

- A socket is a 16-bit number that is used as an index to the socket descriptor table
  - Each entry in this table references a socket data structure
- The socket data structure consists of all the information that is required to set up and exchange data over an IP connection
- A socket data structure is as follows:
Socket Data Structure

<table>
<thead>
<tr>
<th>Socket</th>
<th>Socket details</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
</tr>
<tr>
<td>9654</td>
<td></td>
</tr>
<tr>
<td>10291</td>
<td></td>
</tr>
</tbody>
</table>

Socket Data Structure

<table>
<thead>
<tr>
<th>Family</th>
<th>PF_INET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service</td>
<td>SOCK_STREAM</td>
</tr>
<tr>
<td>LocalIP</td>
<td>172.21.100.8</td>
</tr>
<tr>
<td>RemoteIP</td>
<td>175.26.238.8</td>
</tr>
<tr>
<td>LocalPort</td>
<td>2172</td>
</tr>
<tr>
<td>RemotePort</td>
<td>5143</td>
</tr>
</tbody>
</table>
Port Numbers

- A port number is a 16-bit integer that represents the logical end point of a conversation
- The mechanism that enables a computer to support more than one IP conversation at a time
- Port numbers will mostly likely be different for the same conversation on the client and server computers
An Example

Program 1, 172.21.8.0, Port 52
Program 2, 172.21.8.0, Port 721
---
Program 3, 172.21.8.0, Port 2832

Program 1, 175.21.100.0, Port 52
Program 2, 175.21.100.0, Port 721
---
Program 3, 175.21.100.0, Port 2832

CS390- Unix Programming Environment
Client/ Server Conversation

Client Program
Create a socket – `socket()`
Connect to the server – `connect()`
Send data to the server – `write()`
Receive data from server – `read()`
Terminate the conversation – `close()`

Server Program
`socket()` – create a socket
`bind()` – bind to a port number
`listen()` – get ready for requests
`accept()` – wait for clients
`read()` – Receive data from client
`write()` – Send data to client
`close()` – Terminate conversation

CS390 - Unix Programming Environment
Supporting concurrent clients

**Multiple Socket Server**

s1 = socket() // create a socket
Bind(s1, port_number) // Bind the socket to port
Listen(s1) // place socket to listen

s2 = accept(s1) // wait for client conversation

Create_process(client_handler, s2) // create a new process to service the client requests

CS390- Unix Programming Environment
Client Socket

client_handler (client_socket)
read(client_socket)
write(client_socket)
close(client_socket)

- The server creates a new process or thread to handle every new client request
  - The client connection is serviced by the new socket that was returned by the accept() call
  - The original socket is freed up to handle another client request
Example: A Distributed File viewer service (DFVS)

- The distributed file service consists of the following components
  - Client Program
    - Connect to the remote server program
    - Request a file by name
    - Receive an image of the actual file
    - Display the file to the user

CS390- Unix Programming Environment
Example: (DFVS)

- **Server Program**
  - Wait for the client request
  - Read the file from the local hard disk
  - Send an image of the file to the remote client

- The view of the application is as follows:
Example contd...

Client Program

171.21.100.7

Client

TCP/IP

Server Program Port 1031

171.21.100.45

Server

Hard Disk

socket()

connect(171.21.100.45,1031)

write(c:\test.txt)

read(data-area)

close()

display(data-area)

socket()

bind(1031)

listen()

accept()

read(fileName)

readFromDisk(fileName,dataArea)

write(dataArea)

CS390- Unix Programming Environment
Example: contd...

- Read the file from the server given:
  - Server IP : 171.21.100.45
  - Server Port: 1031
  - Client IP : 171.21.100.7
  - File Name : c:\test.txt

CS390- Unix Programming Environment
Sockets in Java

- The java.net library provides several classes that enable socket-based communication in java

- The Java socket classes are:
  - java.net.Socket
  - java.net.ServerSocket
  - java.net DatagramSocket
  - java.net.SocketImpl

- Check out documentation for more details
Parameters in Java Class constructors

- String Host
  - Host name: e.g. "queen.mcs.drexel.edu"
  - IP Address: 171.21.100.7
  - Use `nslookup` to identify IP addresses

- Int port
  - Integer port number

- InetAddress address: class that represents Internet Protocol address. Has several methods. *Check reference book.*
Implementing a Server

- Create a socket, use the `ServerSocket` class
  ```
  ServerSocket s = new ServerSocket(4444);
  ```
  Socket listens to port 4444

- Accept connection from client
  ```
  Socket Client_Socket = s.accept();
  ```
  Creates a socket that communicates with the client

- Open the input and output stream to/from client
  ```
  DataInputStream in = new DataInputStream(Client_Socket.getInputStream());
  PrintStream ps = new PrintStream(client_Socket.getOutputStream());
  ```
Implementing a Server

- Close all the sockets and Streams
  - `s.close();`
  - `Client_Socket.close();`
  - `in.close();`
  - `ps.close();`

- Upon closing the sockets and streams terminates the session between the server process and client process
Implementing a Client

- Create a socket
  - Specify the hostname or IP and the port where the socket will be created
  - `Socket c = new Socket("queen.mcs.drexel.edu", 4444);`
  - If you are using a single machine then use `localhost` as your hostname

- Open the input and output stream
  - `DataInputStream in = new DataInputStream(c.getInputStream());`
  - `PrintStream out = new PrintStream(c.getOutputStream());`
Implementing the client

- Close the socket and streams
  
  \[c\text{.close}();\]
  \[\text{in}\text{.close}();\]
  \[\text{out}\text{.close}();\]
Sample Server

class EchoServer {
    public static void main(String args[]) {
        try {
            ServerSocket s = new ServerSocket(4444);
            Socket client_socket = s.accept();
            DataInputStream in = new DataInputStream(client_socket.getInputStream());
            PrintStream out = new PrintStream(client_socket.getOutputStream());
            ...
        }
    }
}

CS390- Unix Programming
Environment
Sample Server contd...

...  

```java
out.println("Hello! enter Bye to exit. \r");
boolean done = false;
while (!done) {
    String str = in.readLine();
    if(str == null){
        done = true;
    }
    else {
        out.println("Echo :" + str + " \r");
        if(str.trim().equals("BYE"))
            done = true;
    }
}
```

CS390- Unix Programming Environment
Sample Server contd...

```java
    client_socket.close();

    }
    catch (Exception e) {
        System.out.println(e.toString()); }  }

} // end of class EchoServer

What is the name of this file?
Sample Client

class EchoClient {
    public static void main(String args[]) {
        try {
            Socket c = new Socket("queen.mcs.drexel.edu", 4444); // Note that when you are working on queen try to keep the port number > 4200
            DataInputStream in = new DataInputStream(c.getInputStream());
            boolean more = true;
            while (more) {
                String str = in.readLine();
                if (str == null)
                    more = false;
                else
                    System.out.println(str);
            }
        }…
Sample Client contd...

```
}
catch(IOException ioe) {
    System.out.println("Error "+ioe);
}
} //end of class EchoClient
```

Now that you have a sample application, try to work on this and get more information from the reference book for implementing your term project.

That finishes the tutorial on Java and Socket Programming in Java.

Next lecture, we will go through some of the components of the operating system like file manager, memory manager etc.

CS390- Unix Programming
Environment