COMP202-08B
Computer Communications

Lecture 4 – Concurrent Java Programming

Matthew Luckie
mluckie@cs.waikato.ac.nz

THE UNIVERSITY OF WAIKATO

So far

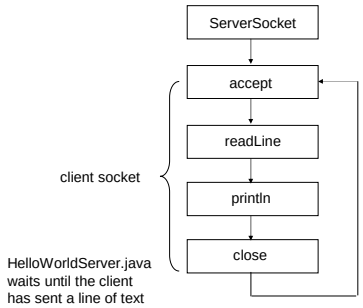
- The **InetAddress** class deals with resolving names to addresses
- The **Socket** class represents an individual connection established over the network
- The **ServerSocket** class is a special type of socket used to allow new connections to be made
- The **PrintWriter** class allows us to write individual lines of text over a socket
- The **BufferedReader** class allows us to read lines of text from a socket

HelloWorldServer.java

Note: THIS CODE IS MISSING NECESSARY EXCEPTION HANDLING

```
ServerSocket ss = new ServerSocket(1234);
while(true) {
 Socket client = ss.accept();
 PrintWriter writer = new
    PrintWriter(client.getOutputStream(), true);
 BufferedReader reader = new BufferedReader(
    new InputStreamReader(client.getInputStream()));
 String line = reader.readLine();
 writer.println("You said: " + line);
 client.close();
}
```

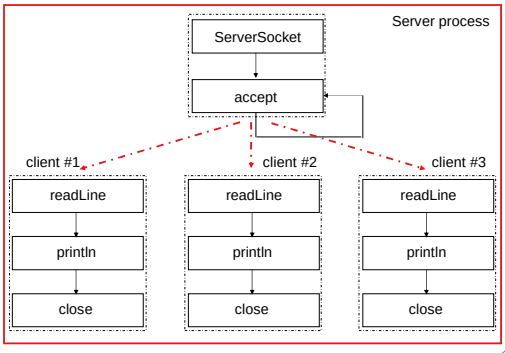
HelloWorldServer.java



HelloWorldServer.java

- Major problem: two clients cannot use the server concurrently.
- The problem is that the program *blocks* until the first client to connect sends a line of text through, causing readLine to return

What we want



Solutions

- There are many solutions to this problem
- The two we are going to discuss in this course are
  - Threads
  - Asynchronous I/O – handle events.
- We'll look at Threads first.

Threads

- Just as our computer can run multiple programs at the same time, each of which share the processor, we can have two strands of our program run at the same time
- Each strand is a Thread
- Jargon
  - Operating system runs processes
  - Programs have threads
  - Threads can be thought of light-weight processes

SleepyThreads.java

```
class Sleepy extends Thread {
   public void run() {
     try {
       System.out.println("Sleepy: hello");
       sleep(1000);
       System.out.println("Sleepy: 1.0 seconds");
       sleep(1000);
       System.out.println("Sleepy: 2.0 seconds");
       sleep(1000);
       System.out.println("Sleepy: 3.0 seconds");
       sleep(1000);
       System.out.println("Sleepy: 4.0 seconds");
       System.out.println("Sleepy: bye");
     } catch(Exception e) {
       System.err.println("Sleepy Exception: " + e);
     }
   }
```


SleepyThreads.java

```
class Snoozy extends Thread {
   public void run() {
     try {
       System.out.println("Snoozy: hello");
       sleep(1500);
       System.out.println("Snoozy: 1.5 seconds");
       sleep(1000);
       System.out.println("Snoozy: 2.5 seconds");
       sleep(1000);
       System.out.println("Snoozy: 3.5 seconds");
       sleep(1000);
       System.out.println("Snoozy: 4.5 seconds");
       System.out.println("Snoozy: bye");
     } catch(Exception e) {
       System.err.println("Snoozy Exception: " + e);
     }
   }
```


SleepyThreads.java

```
class SleepyThreads
{
    public static void main(String args[])
    {
      Sleepy sleepy = new Sleepy();
      Snoozy snoozy = new Snoozy();
      sleepy.start();
      snoozy.start();
      System.out.println("now waiting...");
    }
}
```


In class demo

HelloWorldServer2.java

- Goal is to write HelloWorldServer so multiple clients can connect and interact
- HelloWorldServer2:
  - main method will accept new clients as before
  - instead of then reading text from the client, it will then spawn a thread to read the line of text from the client and write it back

HelloWorldServer2.java: a skeleton

```
class HelloWorldServerThread extends Thread
 {
    public void run()
    {
      /* thread's main procedure – entry point */
    }
    public HelloWorldServerThread(/* parameters */)
    {
      /* initialise thread's parameters */
    }
 }
```


HelloWorldServer2.java: a skeleton

- Constructor takes parameters to create a thread object. You can think of the parameters like command line arguments to main()
- Thread does not begin execution until the start() method is called

HelloWorldServer2.java

```
class HelloWorldServer2 {
 public static void main(String args[])
 {
   try {
    ServerSocket ss = new ServerSocket(1234);
    while(true) {
      Socket client = ss.accept();
      HelloWorldServerThread thread =
        new HelloWorldServerThread(client);
      thread.start();
    }
  }
  catch(Exception e) {
    System.err.println("Exception: " + e);
  }
 }
}
```


HelloWorldServer2.java

```
class HelloWorldServerThread extends Thread {
  public void run() {
    try {
      PrintWriter writer = new
        PrintWriter(client.getOutputStream(), true);
      BufferedReader reader =
        new BufferedReader(
        new InputStreamReader(client.getInputStream()));
      String line = reader.readLine();
      writer.println("You said: " + line);
      client.close();
    } catch(Exception e) {
      System.err.println("Exception: " + e);
    }
  }
}
```


HelloWorldServer2.java

```
class HelloWorldServerThread extends Thread
{
    Socket client;
    public HelloWorldServerThread(Socket s)
    {
      client = s;
    }
}
```

In class demo

Summary so far

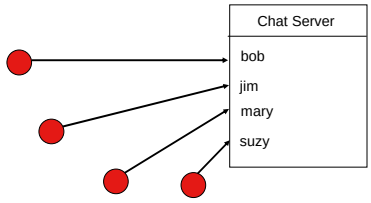
- HelloWorldServer2's main routine accepts new clients and creates a thread to handle that client
- HelloWorldServerThread's constructor takes the client variable to handle
- HelloWorldServer2's main routine calls the start method
- HelloWorldServerThread's run routine is then started.

Threads

- One problem with threads is they don't scale
  - a thread is a mini-process
  - if you had 5000 threads your operating system will have a lot to do
- A second problem is any variables shared between more than one thread have to be used carefully

Real communications systems

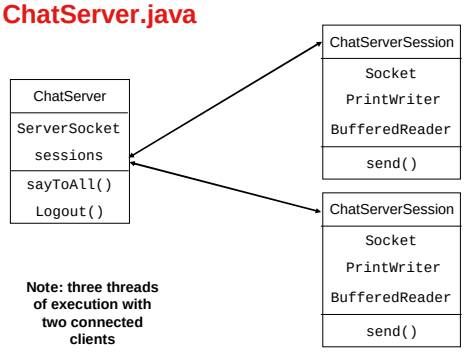
- Real communications systems will deal with multiple connections simultaneously



ChatServer.java

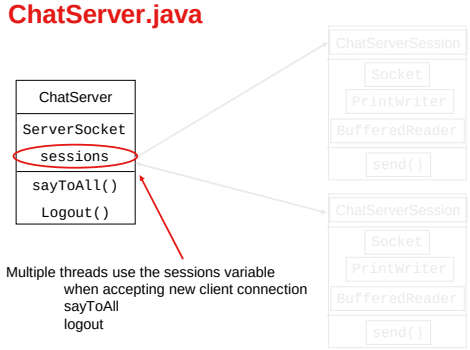
- A chat server takes lines of text from each client and relays that line to all other clients
- Required variable: List of connected clients
- How do we organise our program now?

ChatServer.java



Note: three threads of execution with two connected clients

ChatServer.java



Multiple threads use the sessions variable
- when accepting new client connection
- sayToAll
- logout

Java Threads

- Need to be careful when accessing a variable shared amongst threads to access the variable one at a time
  - i.e, lock the variable when using it to ensure its integrity
- Easy way to do this is to use the **synchronized** keyword

Java Threads

```
public void sayToAll(ChatServerSession from, String
 message)
{
 int i, len;
 synchronized(sessions) {
     len = sessions.size();
     for(i=0; i<len; i++) {
       ChatServerSession to = sessions.get(i);
       if(from != to) {
          to.send(message);
       }
     }
 }
}
```


Summary

- Methods that block are a problem for communications systems as they prevent other clients getting service
- Threads are one way to avoid blocking on input
- Homework task:
  - Add the ability to ChatServer.java for clients to specify their name and prefix it to their messages
  - Make sure the name is unique!