Lecture 8
Client-Server Programming Threads
Winter 2020

Reading: Chapter 2, Relevant Links - Threads

Some Material in these slides from J.F Kurose and K.W. Ross
All material copyright 1996-2007
Introduction

• So far,
• Studied client-server programs with Java
  • Sockets – TCP, UDP
  • Reading and writing to a socket
  • One client at a time
• Today,
• Study client-server programs
  • Sockets – with threads, allow multiple clients in parallel
  • Also, go over the Input/Output of Sockets
Details of Socket I/O

• New Example

  • DailyAdviceServer
    • Server provides valuable advice to clients
    • You connect as a client and it sends back the “advice of the day”
    • Random selection of advice messages

• Still one client at a time

• Review ...

  Connection-oriented, what type of socket are we using?
Daily Advice Client/Server

Client

I need Advice

“You need to rethink that hairdo”

Server
Algorithm for Daily Advice Client

1. **Creates** a socket to Advice Server
2. **Waits** for Server to provide excellent advice
3. **Reads** advice from server
4. **Prints** advice to screen
5. **Quits**
import java.io.*;
import java.net.*;

//The client
public class DailyAdviceClient {

    public void go() {
        try {
            Socket s = new Socket("127.0.0.1", 4200);
            InputStreamReader streamReader =
                new InputStreamReader(s.getInputStream());
            BufferedReader reader =
                new BufferedReader(streamReader);
            String advice = reader.readLine();
            System.out.println("Today you should: " + advice);
            reader.close();
        } catch(IOException ex) {
            ex.printStackTrace();
        }
    } // close go

    What IP is this?
public static void main(String[] args) {
    DailyAdviceClient client = new DailyAdviceClient();
    client.go();
}
Socket I/O - Client

• **Input**

• `InputStreamReader` acts like a bridge between low-level byte stream, `getInputStream()` and high-level character stream like `BufferedReader`

```java
InputStreamReader streamReader = new InputStreamReader(s.getInputStream());
```

• Converts bytes to characters

• Then .....
Details of Socket I/O - Client

• Input Continued ...

• We chain high-level character stream like BufferedReader to the InputStreamReader to read buffered characters

```java
BufferedReader reader =
    new BufferedReader(streamReader);
String advice = reader.readLine();
```

• Chaining of input streams from server to client looks like ...
Details of Socket I/O - Client

Chain of input from server to client

Why do you want to use a Buffered Reader class?

Client

BufferedReader

InputStreamReader

getInputStream

Server

0110 10011

Sockets input stream in bytes

Converted to characters

Characters

Characters are buffered

Buffered Characters
Buffered Readers are Efficient

• Reason to use buffered reader
  • More efficient I/O
  • Each time you read a character, must access the disk
  • Buffered reader gets several characters at once
  • Stores them in a buffer then writes to disk

Explains Buffered Reader performance

http://tutorials.jenkov.com/java-io/bufferedReader.html

https://alvinalexander.com/java/java-bufferedReader-readline-string-examples
import java.io.*;
import java.net.);

public class DailyAdviceServer {

    String[] adviceList = {"Take smaller bites", "Go for the tight jeans. No they do NOT make you look fat.", "One word: inappropriate", "Just for today, be honest. Tell your boss what you *really* think", "You might want to rethink that haircut."};

    public static void main(String[] args) {
        DailyAdviceServer server = new DailyAdviceServer();
        server.go(); // Main Server code in this routine
    }
}
DailyAdviceServer
public void go() {
    try {
        ServerSocket serverSock = new ServerSocket(4200);

        while(true) {
            Socket sock = serverSock.accept();
            PrintWriter writer =
                new PrintWriter(sock.getOutputStream());
            String advice = getAdvice(); // select advice string, next slide

            writer.println(advice); // sends advice string to client
            writer.close(); // need THIS or flush() or never writes...

            System.out.println(advice); // Writes to screen too
        }
    } catch(IOException ex) {
        ex.printStackTrace();
    }
} // close go
private String getAdvice() {
    int random = (int) (Math.random() * adviceList.length);
    return adviceList[random];
}
Socket I/O - DailyAdviceServer

Output

• Create a PrintWriter object gets chained to low-level socket output getOutputStream

• PrintWriter acts like its own bridge between character data and the bytes it gets from the Socket’s low-level output stream

• Can then write strings to the socket connection

PrintWriter writer =
    new PrintWriter(sock.getOutputStream());
String advice = getAdvice();
writer.println (advice);  //adds a newline to string
Socket I/O - DailyAdviceServer

Chain of output from server to client

- **Server**
- **PrintWriter**
  - “message …”
  - Message is in characters
- **getOutputStream**
  - 0110 10011
  - Sockets output stream in bytes
- **Client**

**Note:** Can also use a BufferedWriter on the output stream
Demo Time

Run the Great Advice of the Day program!

Run: DailyAdviceClient.java
DailyAdviceServer.java
Threads in Java
Threads in Java

What is a thread?

• Separate thread of execution
• Called from an existing program
• A java class, Thread represents a thread in Java
• Has a separate stack !!!
• Allows you to run separate processes from main thread
• Can do concurrency
Threads

- What does it mean to have more than one stack?
  - Gives appearance of having multiple things happen at once
  - Execution is actually moving back and forth between stacks
  - It might look like this ...
Threads

- Main gets started, invokes Dog method
- Thread is started, cat
- Execution switches between main and user thread
Java Thread Class

- The Thread class has three primary methods that are used to control a thread:
  - public void start()
  - public void run()
  - public final void stop()

- The start() method prepares a thread to be run
- The run() method actually performs the work of the thread
- The stop() method halts the thread

The thread dies when the run() method terminates or when the thread's stop() method is invoked
Java Thread Class

- Running Threads
- You never call run() explicitly !!!
- It is called automatically by runtime as necessary once you've called start()
- There are also methods to suspend and resume threads, to put threads to sleep, wake them up, and to yield control to other threads
  - These will be covered later in another lecture
Creating Java Threads

Two ways to create threads in Java

1. Extend the Thread Class

   Declare new class as subclass of Thread, then override run() method with code you want executed by Thread

   Example: Compute Prime numbers via thread
   class PrimeThread extends Thread {
       public void run() {
           // compute primes...
       }
   }

   To start this thread, instantiate it, then call start()

   Run method is executed when thread is started
   PrimeThread p = new PrimeThread();
   p.start();
Creating Java Threads

Two ways to create threads in Java

2. Create a Thread via Runnable Interface

Create a thread by Runnable interface.
Any object that implements Runnable interface can be run in a thread.

Example: Compute Prime numbers via thread

```java
class Primes implements Runnable {
    public void run() {
        // compute primes...
    }
}
```

To start this thread, instantiate the thread, pass it a runnable job, then call start()

Run method executed when thread started

```java
Primes p = new Primes();
Thread myprime = new Thread (p);
myprime.start();
```
Threads in Java

• In choosing the two ways to implement threads,
• Advice is to implement Runnable not Extend Thread class, why?

• Answer,
• If thread class is to be subclass of some other class, it can’t extend from the Thread class
• Java does not allow a class to inherit from more than one class ... no multiple inheritance
• Advice is to use **Runnable interface** to implement threads
Recall .... Java Interface

• Interface looks like a class but it is not a class
• Interface can have methods and variables just like a class but methods declared in interface are by default **abstract**
• Only method signature, no body
• Use Interface to implement desired functionality but does not need to inherit from a class
• Can instantiate multiple interfaces within a class

• Nice explanation of Interface
Threads in Java

• **What are the Advantages of Thread?**
  • Multithreading has several advantages over Multiprocessing
    • Threads are **lightweight** compared to processes
    • Threads share same address space and can share both data and code
    • Context switching between threads is less expensive
    • Cost of thread intercommunication relatively low
    • Threads allow different tasks to be performed concurrently
Example Threads via Runnable

• Thread uses the run() method of the runnable interface. Example below

```java
class MyRunnable implements Runnable {
    public void run () {
        go ();
    }

    public void go () {
        doMore ();
    }

    public void doMore () {
        System.out.println("Top o’ the stack");
    }
}
```
Example Threads via Runnable

• Example Continued

```java
class ThreadTester {
    public static void main (String[] args) {
        Runnable threadJob = new MyRunnable ();
        Thread myThread = new Thread (threadJob);
        myThread.start();
        System.out.println ("back in Main");
    }
}
```

Run: MyRunnable.java
Thread States

Thread t = new Thread (r);

Thread created but not started

Thread ready to run, runnable state
Waiting to be selected for execution

Thread selected to run and is the currently running thread

Running!

Once thread is runnable, can go back and forth between running, runnable and blocked
Threads

• The JVM thread scheduler is responsible for deciding who gets to run next
• You, have little control over the decision

• Threads can be Blocked !!!
  • Thread might be blocked for many reasons
  • Examples
    • Executing code to read from socket input stream, but no data to read
    • Executing code told thread to sleep
    • Tried to call a method but object was “locked”
Scheduling Threads

• Don’t base your program’s correctness on the scheduler working in a particular way
  • Can’t control it or even predict the scheduler behavior
  • Implementations are different with different JVM’s
  • We will compile and run example code several times to see this
  • Results will differ even on the same machine
Multiple Clients

• Original Problem
  • Need to handle multiple requests for service from a given server

• Without threads
  • One client at a time, process each until done

• With threads
  • Many clients, process one, move on and process another etc.
Threads and multiple clients

- Same problem, **DailyAdvice Server**
- Now,
  - Add threads so we can service multiple clients at once
  - Everyone will have the benefit of having great advice to start their day
import java.io.*;
import java.net.*;

public class DailyAdviceServer {

    String[] adviceList = {"Take smaller bites", "Go for the tight jeans. No they do NOT make you look fat.", "One word: inappropriate", "Just for today, be honest. Tell your boss what you *really* think", "You might want to rethink that haircut."};

    public static void main(String[] args) {
        DailyAdviceServer server = new DailyAdviceServer();
        server.go(); // Main Server code in this routine
    }
}
public void go() {
    try {
        ServerSocket serverSock = new ServerSocket(4200);

        while(true) {
            AdviceRequest request = new AdviceRequest (serverSock.accept());
            Thread thread = new Thread (request);
            thread.start();
        }
    } catch(Exception ex) {
        ex.printStackTrace();
    }
} // close go
final class AdviceRequest implements Runnable {
    Socket socket;

    public AdviceRequest (Socket socket) throws Exception {
        this.socket = socket;
    }

    // Implement the run method of the Runnable interface
    public void run() {
        try {
            processRequest();
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    private void processRequest () throws Exception {
        // Attach a PrintWriter to socket's output stream
        PrintWriter writer =
            new PrintWriter(this.socket.getOutputStream());
        String advice = getAdvice();
        writer.println(advice);
        writer.close(); // must have THIS or flush() or it never writes...
        System.out.println(advice);
    }
}
Summary

• Brief coverage of sockets with threads
• Should be enough for you to get started
  • This example will be available to download
  • Code is on the main class page
• We will be implementing ..... 
  • A multi-threaded Web Server!!!!
• Also, practice client-server in the lab
• Can read references in RelevantLinks for more information
