

CmpE 473 Internet Programming

Pınar Yolum
pyolum@cmpe.boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Chapter 2 Java Networking (cont'd)

Examples from java.sun.com

Reading from a URL

```
import java.net.*; import java.io.*;
public class URLConnectionReader {
    public static void main(String[] args) throws Exception {
        URL yahoo = new URL("http://www.yahoo.com/");
        URLConnection yc = yahoo.openConnection();
        BufferedReader in = new BufferedReader(
            new InputStreamReader( yc.getInputStream()));

        String inputLine;
        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
        in.close();
    }
}
```

Spring 2005—Pinar Yolum

3

Writing to a URL

```
import java.io.*; import java.net.*;
public class Reverse {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: java Reverse " + "string_to_reverse");
            System.exit(1);
        }
        String stringToReverse = URLEncoder.encode(args[0], "UTF-8");
        URL url = new URL("http://java.sun.com/cgi-bin/backwards");
        URLConnection connection = url.openConnection();
        connection.setDoOutput(true);

        PrintWriter out = new PrintWriter( connection.getOutputStream());
        out.println("string=" + stringToReverse);
        out.close();

        BufferedReader in = new BufferedReader( new InputStreamReader(
            connection.getInputStream()));
        String inputLine;

        while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine); in.close();
    }
}
```

Spring 2005—Pinar Yolum

4

Domain Name Service (DNS)

- What does DNS do?
 - Convert host and domain names into IP addresses and vice versa
- Why do we need DNS?
 - IP addresses are hard to remember
 - Allows abstraction
 - You can change the IP address of your machine but people need not worry about such details.
 - Knowing the domain name is good enough for access
- Challenges for DNS:
 - A hostname can have any number of IP addresses
 - An IP address can have any number of hostnames
 - Have to maintain correct tables

Spring 2005— Pinar Yolum

5

Domain Name Service (DNS)

- **java.net.InetAddress** to use DNS
- Extends **java.lang.Object**
- All Implemented Interfaces:
 - **Serializable**
- Direct Known Subclasses:
 - **Inet4Address**, **Inet6Address**

Spring 2005— Pinar Yolum

6

Domain Name Service (DNS)

- **java.net.InetAddress** to use DNS
- An `InetAddress` object can only be created by methods in the `java.net` package
- `String getHostName()`
 - Returns the name of the machine identified by the `InetAddress`.
- `byte[] getAddress()`
 - Returns an array of four bytes with the IP address.
- `InetAddress getByName(String)`
 - Lookup the address of a machine.
- `InetAddress getLocalHost()`
 - #4 of the missing data!
- `InetAddress[] getAllByName(String)`
 - Lookup all addresses for a machine.

Spring 2005— Pinar Yolum

7

IP lookup

```
import java.net.InetAddress;
import java.io.*;
public class SimpleDNS {
    public static void main(String[] args) {
        try {
            InetAddress address = InetAddress.getByName(args[0]);
            byte ip[] = address.getAddress();

            for (int octet=0; octet < ip.length; octet++) {
                System.out.print(((int)ip[octet]) & 0xff);
            }
            System.out.println();
        }
        catch (IOException DNSerror) {
            System.err.println(DNSerror.getMessage());
            System.exit(1);
        }
    }
}
```

Spring 2005— Pinar Yolum

8

Java Sockets

- To establish a connection between two machines:
 - The protocol (TCP or UDP)
 - The address of the local machine and the port number
 - The address of the remote machine and the port number
 - Both local and remote machine should speak the protocol

Secure Shell Example

- The protocol ssh
- The remote computer
 - Name: yunus.cmpe.boun.edu.tr
 - Port: 22 (default)
- The local computer
 - Name: few.vu.nl (The OS will provide this information to the client)
 - Port: Usually between 1024-5000 and generated by the OS

Simple Java Client

```
import java.net.*; import java.io.*;
public class GetLine {
    public static void main(String[] args) {
        Socket client = null;
        try { client = new Socket("yunus.cmpe.boun.edu.tr", 22);
        } catch (UnknownHostException hostError) {
            System.err.println(hostError.getMessage()); System.exit(1);
        } catch (IOException genericError) {
            System.err.println(genericError.getMessage()); System.exit(1);
        }
        try {
            BufferedReader in = new BufferedReader(new
            InputStreamReader(client.getInputStream()));
            String inputLine = in.readLine();
            if (inputLine != null) {
                System.out.println("Received: " + inputLine);
            }
        } catch (IOException IOError) {
            System.err.println(IOError.getMessage()); System.exit(1);
        }
    }
}
```

Spring 2005— Pinar Yolum

11

Java Sockets (1)

- Class `java.net.Socket` extends `Object`
- Direct Known Subclasses:
 - `SSLSocket`
- `Socket` is used to implement client sockets
- Work done by an instance of the `SocketImpl` class.
- Change the socket implementation to get a different socket (for example for firewalls)
- `Socket` interface stays the same

Spring 2005— Pinar Yolum

12

Java Sockets (2)

- **Constructors:**

- **Socket**(String host, int port)
Creates a stream socket and connects it to the specified port number on the named host.
- **Socket**(InetAddress address, int port)
Creates a stream socket and connects it to the specified port number at the specified IP address.
- **Socket**(String host, int port, InetAddress localAddr, int localPort)
Creates a socket and connects it to the specified remote host on the specified remote port.
- **Socket**()
Creates an unconnected socket, with the system-default type of SocketImpl.

Java Sockets (3)

- **General Socket Exceptions:**

- **UnknownHostException:** The specified host cannot be looked up, either because there is no such host or the host is temporarily not available
- **SocketException:** The connection cannot be setup (ex. There is a TCP problem; "Connection refused")
- **BindException:** Extends SocketException. Cannot bind the a socket to a local address.
- **SocketTimeoutException:** Signals that a timeout has occurred on a socket read or accept.
- **IOException:** The java.net.*Exception classes are all derived from this.

Java Sockets (4)

- **Methods:**
 - InputStream **getInputStream()**
Returns an input stream for this socket.
 - void **close()**
Closes this socket.
 - void **connect(SocketAddress endpoint)**
Connects this socket to the server.
 - void **connect(SocketAddress endpoint, int timeout)**
Connects this socket to the server with a specified timeout value.
 - OutputStream **getOutputStream()**
Returns an output stream for this socket.
 - int **getPort()**
Returns the remote port to which this socket is connected.

Spring 2005— Pinar Yolum

15

Java Server Sockets (1)

- Class `java.net.ServerSocket` extends `Object`
- Direct Known Subclasses:
 - SSLServerSocket
- `ServerSocket` is used to implement server sockets
- Again, work done by an instance of the `SocketImpl` class.
- Change the socket implementation to get a different socket (for example for firewalls)
- `ServerSocket` interface stays the same

Spring 2005— Pinar Yolum

16

Java Server Sockets (2)

- **Constructors:**
 - **ServerSocket**()
Creates an unbound server socket.
 - **ServerSocket**(int port)
Creates a server socket, bound to the specified port.
- **Methods:**
 - Socket **accept()**
 - void **bind** (SocketAddress endpoint)
 - void **close()**

Java Server Sockets (3)

```
try {
    serverSocket = new ServerSocket(4444);
} catch (IOException e) {
    System.out.println("Could not listen on port: 4444");
    System.exit(-1);
}
Socket clientSocket = null;
try {
    clientSocket = serverSocket.accept();
} catch (IOException e) {
    System.out.println("Accept failed: 4444");
    System.exit(-1);
}
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader( new InputStreamReader(
    clientSocket.getInputStream()));
out.println("hi");
String inputLine = in.readLine();
```

Threads

- Similar to a sequential program
- But, doesn't run on its own but within a program
- Interesting when a program has multiple threads
- All threads share the same program resources (variables and data structures)
- Multithreaded servers can serve many clients at the same time
- Example: HTTP server
- Web browser?

Java Threads

- Two ways to implement a thread
 - Extend `java.util.Thread` class
 - Create a class that implements `Runnable` interface (useful if the class has to extend another class)
- Creating a thread means calling the `run` method
 - Contains the actions of the thread

Extending Threads

```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                sleep((long)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }
}
public class TwoThreadsDemo {
    public static void main (String[] args) {
        new SimpleThread("Jamaica").start();
        new SimpleThread("Fiji").start();
    }
}
```

Spring 2005— Pinar Yolum

21

Java Timer

- Perform an action when a certain time hits or periodically
- Constructors:
 - Timer()
 - Timer(boolean isDeamon)
- Methods
 - cancel()
 - schedule(TimerTask task, Date time)
 - schedule(TimerTask task, long delay)
- TimerTask: Abstract class (must be extended to be initiated); implement Runnable

Spring 2005— Pinar Yolum

22

Task Scheduling

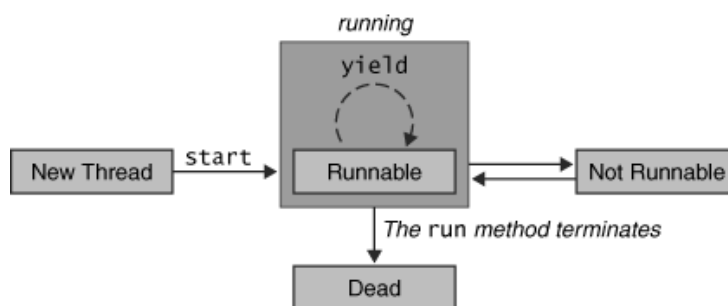
```
import java.util.Timer;
import java.util.TimerTask;
public class Reminder {
    Timer timer;
    public Reminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds*1000);
    } //OR give exact time
    class RemindTask extends TimerTask {
        public void run() {
            System.out.println("Time's up!");
            timer.cancel(); //Terminate the timer thread
        }
    }
    public static void main(String args[]) {
        new Reminder(5);
        System.out.println("Task scheduled.");
    }
}
```

Use a timer to
schedule a task

Spring 2005— Pinar Yolum

23

Thread Lifecycle



- Not runnable (sleep method invoked; wait called; blocking on I/O)
- Stop (code ends; set thread to null)

Spring 2005— Pinar Yolum

24

Multithreading

- Using same resources
 - Don't change values at the same time
 - Disturb other threads
- JVM manages thread usage
- Cooperative threading
 - Thread decides to give up the processor
 - Application developer guarantees fairness
- Preemptive threading
 - OS decides when a thread has worked enough
 - OS gives access to another thread

Scheduling Threads

- The order in which threads will run
- Fixed-priority scheduling
 - Inherit or set a priority for each thread (int)
 - When several threads are ready, the highest priority is executed
 - Choose randomly between two same priority threads
 - If the high priority thread stops/yields/becomes not Runnable, the next highest priority thread runs
- Preemptive
 - Prefer a higher priority thread
- Starvation
 - May prefer a low-priority to avoid starvation

Synchronizing Threads

- Multiple threads access the same data structure
- Example
 - Producer generates a number
 - Consumer picks up the number
- What can go wrong?
 - Producer produces too quickly
 - Consumer reads too quickly
- Race condition
 - Final result depends on the speed of the threads
- Two forms of synchronization
 - Locking: Don't access the data at the same time
 - Coordination: Signal when data is ready

Spring 2005— Pinar Yolum

27

Locking Objects

- Critical section
 - Should be accessed by one thread at a time
 - A producer writing the number; a consumer reading it
- `synchronized` keyword
 - For methods
 - Locks the object so that no synchronized methods can be accessed

Spring 2005— Pinar Yolum

28

Locking Objects

```
public class CubbyHole {
    private int contents;
    private boolean available = false;
    public synchronized void put(int value) {
        //CubbyHole locked by the Producer
        ...
        //CubbyHole unlocked by the Producer }
    public synchronized int get() {
        // CubbyHole locked by the Consumer
        ...
        // CubbyHole unlocked by the Consumer }
}
```

Spring 2005— Pinar Yolum

29

Coordination

```
public synchronized int get() {
    while (available == false) {
        try { //wait for Producer to put value
            wait();
        } catch (InterruptedException e) { }
    }
    available = false; //notify Producer that value has been retrieved
    notifyAll();
    return contents;
}
public synchronized void put(int value) {
    while (available == true) {
        try { //wait for Consumer to get value
            wait();
        } catch (InterruptedException e) { }
    }
    contents = value;
    available = true; //notify Consumer that value has been set
    notifyAll();
}
```

Spring 2005— Pinar Yolum

30