
CmpE 473

Internet Programming

Pınar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

Java Networking

Java Networking

- Allow connection from one computer to another
- Some well-known protocols can be used without knowing much detail (e.g., HTTP)
- Basic classes
 - URL
 - URLConnection
- Support for other protocols through
 - Socket
 - ServerSocket

URL (1)

- Uniform Resource Locator (URL)
 - Specifies how to find a resource
 - ProtocolName://ResourceName
 - <http://www.cmpe.boun.edu.tr/index.html>
 - <file:///C:/tmp/mem-rdf.rdf>
 - Other protocols?
- General parts of a Resource Name
 - Host name: www.cmpe.boun.edu.tr
 - File name: index.html (last / means index.html)
 - Port number (optional, assumed 80 for http)
 - Reference to a html anchor (optional)

URL (2)

- Absolute URLs give the entire address
 - `http://www.cmpe.boun.edu.tr`
- Relative URLs give part of the address relative to another URL
 - Used mostly in HTML pages
 - `Publications`
 - It searches for `publications.html` in the directory of the file that contains the statement

Java URL (1)

- Used to communicate with a Web server using HTTP
- Create a URL object from the absolute URL string
 - `URL(String absoluteString)`
 - `URL cmpe = new URL("http://www.cmpe.boun.edu.tr/~pyolum/")`
- Create a URL object from the relative URL string
 - `URL(URL baseURL, String relativeAddress)`
 - `URL pubs= new URL(cmpe, "publications.html")`
 - `URL thirdSection= new URL(pubs, "#THIRD")`
- Create a URL from its parts
 - `URL p = new URL("http", "www.cmpe.boun.edu.tr", 80, "publications.html");`

Java URL (2)

- Java URLs are write-once
 - You can't modify its properties
- Compare two URLs for equality
- `getPort`, `getHost`, ...
- Throws *MalformedURLException*
- Once you have a URL, open a connection to retrieve the contents (HTML commands)
 - `java.io.InputStream URL.openStream()`

Reading from a URL

```
import java.net.*;
import java.io.*;

public class URLReader {
    public static void main(String[] args) throws Exception {
        URL home = new URL("http://www.cmpe.boun.edu.tr/~pyolum");
        BufferedReader in = new BufferedReader(
            new InputStreamReader(
                home.openStream()));

        String inputLine;

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

        in.close();
    }
}
```

Sample Output

```
<!DOCTYPE html PUBLIC "-//w3c//dtd html 4.0 transitional//en">
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-9">

<!--windows-1254 -->
<title>pİnar Yolum</title>
</head>
<body text="#000000" link="#0000ff" vlink="#000080" bgcolor="#ffffcc" alink="#ff0000">
<br>
<p><font face="Arial" size="+2"><b>Pİnar Yolum</b></font></p>
<br>
<br>á
<br>á
<br>
<ul>
<li> <p><font face="Arial" size="+1"><a href="education.html">Education</a></font>
<li> <p><font face="Arial" size="+1"><a href="publications.html">Publications</a></font>
<li> <p><font face="Arial" size="+1"><a href="teaching.html">Teaching</a></font>
<li> <p><font face="Arial" size="+1"><a href="professional.html">Professional
    Activities</a></font>
<li> <p><font face="Arial" size="+1"><a href="491-projects.html">491/492 Projects</a></font>
```

HTTP Request

- HyperText Transfer Protocol
- Client requests contain information about the client, protocol version used, acceptable file types, and so on

GET /~pyolum/publications.html HTTP/1.0

User-Agent: Javaxx

Host: www.cmpe.boun.edu.tr

Accept: text/html, image/jpeg, *; q=.1

Connection: keep-alive

HTTP Reply

- Example reply

HTTP/1.0 200 OK

Date: xxx

Server: yyy

Content-type: text/html

Last modified:xxx

Content-length: 223

- Status code 200 for success, 400 bad request, 404 not found
- Content length in bytes

URLConnection

- Set up a connection to read and write to a URL
- Has extra methods for customization

```
try {
    URL home = new
    URL("http://www.cmpe.boun.edu.tr/~pyolum");
    URLConnection homeConnection =
    home.openConnection();
    System.out.println(homeConnection.getHeaderField(0));
}
catch (MalformedURLException e) { // new URL() failed
    System.out.println("No such URL");
}
catch (IOException e) { // openConnection() failed
    System.out.println("Connection could not be opened");
}
```

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 same 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);
        }
    }
}
```

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

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.

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

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

- Allow multiple instances of the same sequential 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?
- AJ: Section 1.4

Java Threads

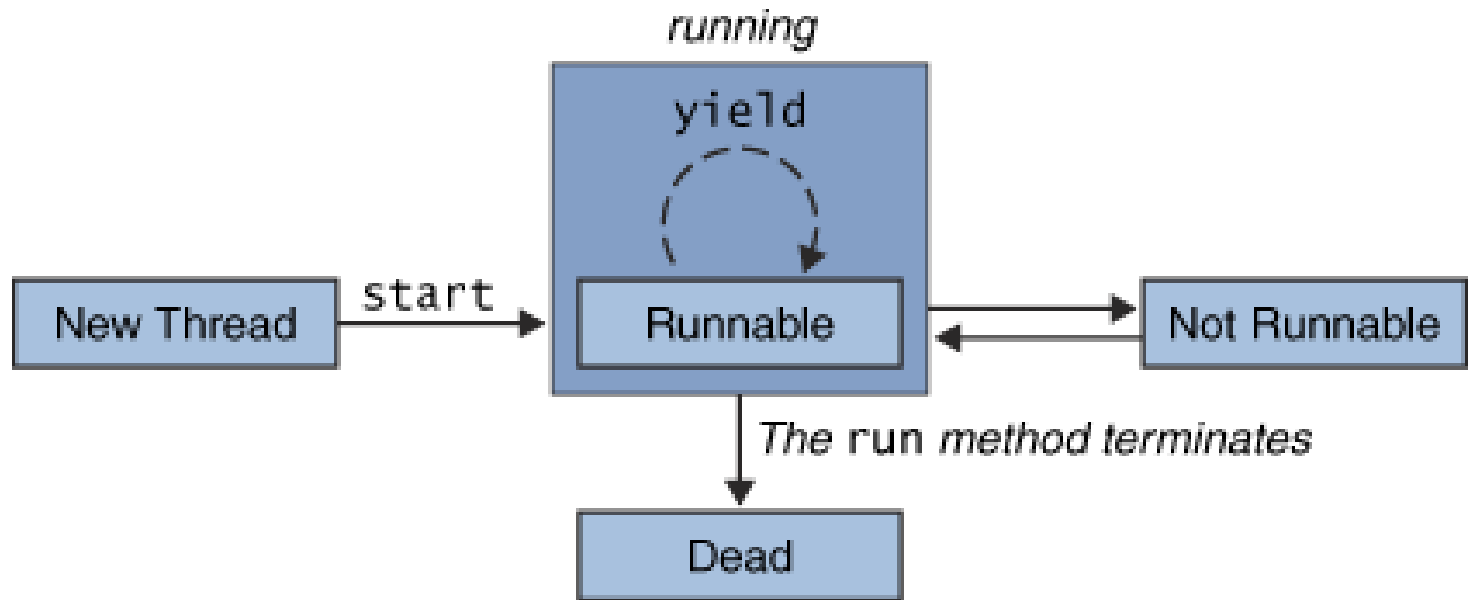
- Two ways to implement a thread
 - Extend `java.util.Thread` class
 - Create a class that implements `Runnable` interface (useful if the class has to extend another class)
- Creating a thread means calling the `run` method
 - Contains the actions of the thread

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();
    }
}
```

Thread Lifecycle



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

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

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

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 }  
    }  
}
```

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();
}
```