Socket and Multi-Threaded Programming

Network Distributed Programming Mark Cummins (ITB)

Up till now we have looked at how we handle various streams for input and output. We have also briefly looked at sockets. In this lab we will revisit sockets and also take a look at how we handle multiple threads within our programs. So read carefully through each section and implement each of the sample programs, before answering the questions relating to each program. These programs should serve as a reference for any other programs you may need to write yourself in the future. You can download a zip folder containing all the sample code from www.markcummins.net/NDC/Labs/NDC_Labs.zip

1. Socket Essentials

The basic idea behind socket communications is simple. A client establishes a connection with a server. Once a connection is made, the client can write to the socket to send data to the server. Conversely, the server sends data that the client will read to the socket. It's almost that simple; the details can be complex but the idea is just that simple.

There are three main types of socket classes that java provides. **DatagramSocket** is the class that implements the UDP protocol. The other two socket classes are **Socket** and **ServerSocket**, and they both support TCP connections.

If you are connecting to a server you'll use **Socket**. If you are writing a server, you'll use **ServerSocket**. Why the difference? A client socket doesn't really care what port it uses locally, it does need to connect to a specific port on the server computer. On the other hand, a server is very concerned with its local port assignment (that's how clients find it). Servers also have to stay listening for incoming connections and clients do not.

2. Addressing

No matter what kind of socket we want to use, we need to specify the address of the server we want to connect to. You might think we could just pass a hostname or an IP address, but this isn't quite the case. Instead we need to use **InetAddress** to represent the remote computer's address. How do we do this?

Most of the time we use one of the following two methods: **getLocalHost** or **getByName** both of these methods return an **InetAddress** object.

```
InetAddress address = null;

//we can use IP address
address = InetAddress.getByName(“192.168.1.100”);

//or a host address
address = InetAddress.getByName(“localhost”);
address = InetAddress.getByName(“www.yahoo.com”);

// address of local computer
address = InetAddress.getLocalHost();
```

Once we have an InetAddress object we can use the following methods **getHostName** and **getHostAddress** to return the hostname and IP address.

```
address.getHostAddress();
address.getHostName();
```

Ok let's see them all in action.

3. GetName

If you're writing a network based application, you may need to know the hostname or IP address of the machine your application is going to run on. If you were running the application on your own machine you might hard-code the machine name. (For example String localHost = "machine-name"). But what if you wanted your program to be able to run on different machines? Having the localHost hard-coded is not a good idea. It is better to have your program automatically figure out the hostname or IP address of the machine it is running on.

The source code shown below shows you how you would find the symbolic name (hostname) of the machine you are working on.

```
// GetName.java
import java.net.*;

public class GetName
{
     public static void main(String argv[]) throws Exception
     {
          InetAddress host = null;
          host = InetAddress.getLocalHost();
          System.out.println(host.getHostName());
     }
}
```

1) **Compile and run GetName.java**
2) **What does it give as the host name of your PC?**
   Answer: ____________________________________
3) **Change GetName.java so that it prints out both the hostname and IP address.**
4) **What method did you add to GetName so it prints the local IP address?**
   Answer: ____________________________________
5) **What java Package contains the InetAddress class and its included methods?**
   Answer: ____________________________________

4. NsLookup

As well as needing to know details about the local Pc we are running on. It is sometimes essential that we connect to a remote machine. Again rather than hard-code this information it is better for our programs to automatically discover the information we need. The getByName() method can be used by our programs as a simple DNS lookup. Our next program NsLookup can be used as a simple utility that allows us to find the IP address for any accessible hostname on the internet.

```
// NsLookup.java
import java.net.*;

public class NsLookup
{
        public static void main(String argv[])
        {
                if (argv.length == 0)
                {
                  System.out.println("Usage: java NsLookup <hostname>");
                  System.exit(0);
                }

                String host = argv[0];
                InetAddress address = null;

                try
                {
                  address = InetAddress.getByName(host);
                }
                catch(UnknownHostException e)
                {
                   System.out.println("Unknown host");
                   System.exit(0);
                }

                System.out.println(address.getHostAddress());
        }
}
```

1) **Compile and run NsLookup.java**
2) **What is the IP address of the www.itb.ie website?**
   Answer: ___________________________________
3) **Change NsLookup.java so that it prints out both the hostname and IP address.**
4) **What method did you add to NsLookup so it prints the hostname?**
   Answer: ___________________________________
5) **Create a program called IPtoName that converts an IP address to hostname**

5. TCP Sockets

There are four steps to programming a client or server using sockets:

- Opening a socket
- Creating an input stream
- Creating an output stream
- Closing the socket

We've already looked at streams in detail, so we'll just focus on opening & closing sockets.

If you are programming a client, open a socket like this

```
Socket MyClient = null;

try
     { MyClient = new Socket(“host”,PortNumber);  }
catch(UnknownHostException uhe)
     { uhe.printStackTrace();  }
catch(IOException e)
     { e.printStackTrace();     }
```

If you are programming a server, open a socket like this

```
ServerSocket MyService = null;

try
     { MyService = new ServerSocket(PortNumber);  }
catch(IOException ioe)
     { IOe.printStackTrace();   }
```

If you are implementing a server, you also need to create a socket object from the ServerSocket in order to listen for accept connections from client. This is done as follows:

```
Socket serviceSocket = null;

try
     { serviceSocket = Myservice.accept();  }
catch(IOException iex)
     { iex.printStackTrace();   }
```

We then open any input and output streams we need. And when finished doing whatever we are doing we need to first close these input and output streams, before finally closing any sockets we have opened.

So on the client side we do this:

```
try
{
     MyClient.close();
}
catch(IOException io)
     { io.printStackTrace();    }
```

And on the Server side we do this:

```
try
{
     serviceSocket.close();
}
catch(IOException io)
     { io.printStackTrace();    }
```

6. A Simple TCP Client

When to want to connect to a server, you use the **Socket** class. The simplest way to create a **Socket** is to provide a hostname (or InetAddress object) and a port number. Our next example is a simple program that connects to a web server. The program doesn't transfer any data, but it does check to see if some server, presumably a web server, is listening on port 80 and connects to it.

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

public class WebPing
{
     public static void main(String[] args)
     {
          try
          {
            InetAddress addr;
            Socket sock=new Socket(args[0],80);
            addr=sock.getInetAddress();
            System.out.println("Connected to " + addr);
            sock.close();
          }
          catch (java.io.IOException e)
          {
            System.out.println("Can't connect to " + args[0]);
            System.out.println(e);
          }
    }
}
```

1) **Compile and run WebPing.java, and try connecting to www.rte.ie**
2) **What command did you type at the command line to test step 2?**
   Answer: ________________________________
3) **We used a new method getInetAddress() in this program. What does this method return? Answer:_________________________________**

7. A TCP Server

If you are writing a standard server (Telnet, Email, FTP etc.) you'll want to use the well-known port numbers associated with that server type. If you aren't writing a standard server, you can select a port that isn't in use on your system (typically above 1023)

Try issuing the following command:

```
telnet localhost 8123
```

It is very likely that the program will report that it can't connect to that port. If the port is in use, just select another number.

Our next example Techo.java will create a simple server that doesn't do anything, but if we run this program, the telnet program will be able to connect to port 8123.

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

public class Techo {
  public static void main(String[] args) {
    try {
      ServerSocket server=new ServerSocket(8123);
      while (true) {
        System.out.println("Listening");
        Socket sock = server.accept();
        InetAddress addr=sock.getInetAddress();
        System.out.println("Connection made to "
           + addr.getHostName() + " ("
           + addr.getHostAddress() + ")");
        pause(5000);
        sock.close();
        }
      }
    catch (IOException e) {
      System.out.println("Exception detected: " + e);
      }
    }
  private static void pause(int ms) {
     try { Thread.sleep(ms);      }
     catch (InterruptedException e) {}
     }
  }
```

1) **Compile and run Techo.java, and try the telnet command again.**
2) **What happens if you try to run a second copy of the server at the same time?**
   Answer:____
3) **Why does this happen?**
   Answer:____

8. LocalScan.java

If the port number we want to use is already in use, the constructor will throw an IOException. You can use this to discover the ports that are already in use on your machine. In this next example we create a program that will scan all the ports on your machine and list all the ports that are in use. This is commonly done for security reasons too.

```
import java.net.*;

public class LocalScan {
  public static void main(String [] args) {
   for (int i=1;i<1023;i++) {
     testPort(i);
     }
   System.out.println("Completed");
   }
  private static void testPort(int i) {
    try {
      ServerSocket sock=new ServerSocket(i);
      }
    catch (java.io.IOException e) {
      System.out.println("Port " + i + " in use.");
      }
    }
 }
```

1) **Compile and run LocalScan.java.**
2) **Find any port in use, and change the previous Tecno.java program to use that port instead of the 8123 port.**
3) **What happens when we try to run this new techno.java program?**
   Answer:____
4) **Why does this happen?**
   Answer:____

9. Introduction to Threads

Have a quick look back at the Techo.java program, in particular the line

```
        Socket sock = server.accept();
```

What's actually happening here? This call actually blocks and waits for an incoming connection. This is usually unacceptable for any server. Basically it can't do anything except wait for a connection and then handle each single connection then wait for another. So how should be handle listening for incoming connections?

The best way to handle clients is to create a new thread for each client. Because java has built-in thread handling this is relatively straight forward.

The easiest way to handle this is to extend the Thread object to form a new object that creates a thread. That's exactly what our next example does. Its a n example of a multithreaded server.

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

public class AMTServer extends Thread {
    Socket csocket;
    AMTServer(Socket csocket) { this.csocket = csocket; }
    public static void main(String args[]) throws Exception {
      ServerSocket ssock=new ServerSocket(1234);
      while (true) {
        new AMTServer(ssock.accept()).start();
      }

    }

   public void run() {
// client processing code here
    }
  }
```

The thread object begins when you can start().

Some of the common thread methods include:

Sleep: you can cause the thread to pause for a specified number of milliseconds. This is very efficient and allows other threads to execute while the thread is sleeping

Yield: Most systems give threads a certain amount of time to execute. If the thread does not need its time slice, it can give it up with yield.

1) **Create a simple multi threaded server and connect from several clients**