ECE 480 DESIGN TEAM 2

How to Program a Java Network Server

Application Note

Jacob D'Onofrio

3/31/2011

Java is a platform independent object oriented programming language that has ports to a variety of architectures. It provides an easy programming environment with lots of built in and third party libraries to accomplish just about any task with simplicity. With this guide I will introduce some of these libraries to help - build a basic efficient network server. The steps that we will take to build this will be first to prepare and configure our development environment. Next we will go through the necessary libraries needed to create our server, and lastly we will go through step by step how to make it work. In the end, you will have a basic layout of a network server that can be customized your needs.

Contents

Key Terms

API – Application Programming Interface: An interface between one program to another where the first program generally provides tools to assist in the programming of the second.

IDE – Integrated Development Environment: a program designed to assist a programmer to make programming and executing applications easier.

SDK – Software Development Kit: a set of tools that assist in the development of an application.

JDK – Java Development Kit: Java's set of tools to assist in developing an application.

History

Programming has come a long way over the years and as technology becomes more embedded into our everyday lifestyle, this will also become more of a common skill set. Over time, languages which we use to run our computers have evolved and become simpler to use. Assembly language, or machine code, consists of individual commands that can be sent to the processor. This is what all programs turn into in order for them to be run. With the introduction of compilers, higher level languages that are more user friendly and offer more functionality could be created leading the one of the most popular languages today, C. This language is known for its somewhat tricky syntax but brings with it speed. One of the major downsides to this though is that because it is compiled into machine language, it needs to be recompiled each time you want to run it on a different platform.

The Java programming language has been originally designed with one thing in mind, portability. This is one of the major strengths of this language and because of this has become more popular as a mainstream language. Java also brings simplicity with it. It takes care of a lot of the lower level code and hardware configuration for you. This makes coding easier on the regular user but also gives you limits. Our needs fit in these limits and make Java ideal.

Introduction

The internet is tied in with our everyday lives more than we probably know. It aims at having a global network of interconnected devices that can send and receive information with each other. This is becoming truer every day with sorts of different devices being able to plug themselves in and communicate.

The basic model of network communication can be broken down into two parts, the client and the server. We find ourselves using the client devices every day such as our laptops, or smartphones to view different kinds of data. The server side is what the clients communicate with and makes data queries against. This is what we will be looking into as we explore Java.

Getting Started

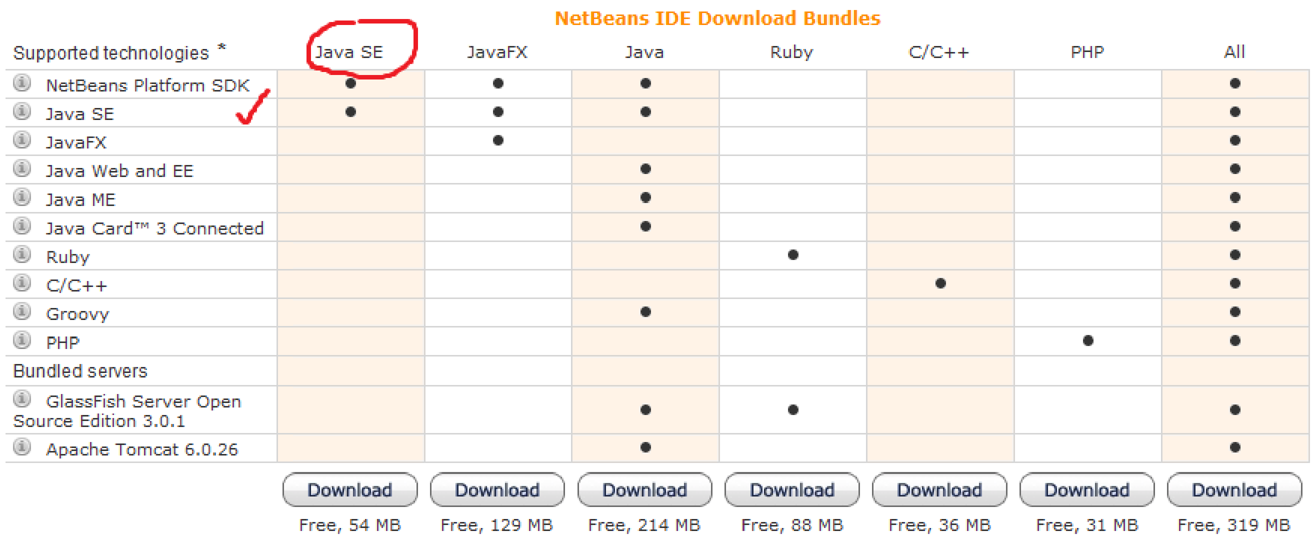
Required Tools

SDK

To develop any kind of program, you are required to download and install their respective development tools, or software development kit (SDK). Java has changed the name slightly and refers to it as the JDK, or Java development kit. It is available free on Oracles website, http://www.oracle.com/technetwork/java/javase/downloads/index.html. As of writing this, the latest version is 1.6_24.

Development Environment

Java programs can be developed in a variety of ways ranging from manual text entry to using integrated development environments (IDE). In this tutorial we will be using the Oracle Netbeans IDE. The program can be obtained here www.netbeans.org. Directions for installation are also available there. As of writing this, the latest stable version available is 6.9.1. You will notice that there are quite a few versions available of Netbeans. Like many IDE's out there, it can be used for more than one programing language. For our needs, you will be fine with the Java SE edition. See picture below to settle and confusion.



NetBeans IDE Download Bundles

Supported technologies *	[✓ Java SE]	JavaFX	Java	Ruby	C/C++	PHP	All
NetBeans Platform SDK	●	●	●				●
Java SE ✓	●	●	●				●
JavaFX		●					●
Java Web and EE			●				●
Java ME			●				●
Java Card™ 3 Connected			●				●
Ruby				●			●
C/C++					●		●
Groovy			●				●
PHP						●	●
Bundled servers							
GlassFish Server Open Source Edition 3.0.1			●	●			●
Apache Tomcat 6.0.26			●				●
	Download	Download	Download	Download	Download	Download	Download
	Free, 54 MB	Free, 129 MB	Free, 214 MB	Free, 88 MB	Free, 36 MB	Free, 31 MB	Free, 319 MB

Configuration

To get started with our network server, we are going to need to create a new project. This can be done by going to start, then “New Project”. From there a menu will be brought up with a variety of choices depending upon which version of Netbeans your installed. We want to make a standard Java Application which is located under the Java directory. Give your program a proper name and set it as the main project. The IDE should take care of any other configuration for you.

Design

You will find that making this server doesn’t actually require a lot of code. Java does a lot of work for you that would normally have to take care of in traditional languages such as C++. The downside to this is that you have less control over the lower level code and hardware. This is the price that you have to play for simplicity though.

Libraries

Precompiled packages that are available to reference are called libraries. The JDK comes with a lot of very useful ones that we will be using to build this project. Documentation can be found online that has detailed information about each and every class. We will be using the network and input output packages.

```
import java.io.*;

import java.net.*;
```

Socket Programming

Network communication is done via sockets. Each program on your computer that requires the use of a network first has to set up its connection before it can begin sending packets and making queries. The same goes for servers. The network library contains all of the classes that we will need to complete this step. The server uses two types of socket classes, ServerSocket, and Socket.

```
try{
    port = 1024;
    ServerSocket Listener = new ServerSocket(port);
    Socket server;
    //Next Section
}
catch (IOException e){
    System.out.println(e);
}
```

The ServerSocket class is the main object that will be receiving connections. In my example code above, I have named mine “Listener” which describes exactly what it is doing. When you create a socket, you have to define which TCP/UDP port that you wish for it to listen on. For more information on ports or TCP/UDP, please see OSI model under the transport layer. http://en.wikipedia.org/wiki/Transport_layer.

The ServerSocket does not perform any communication besides acknowledging that a connection has been established. To confirm the connection with the client on the other end we need to execute the accept() function. This returns a Socket object which is then handed off to perform communication.

```
while(1){
    server = Listener.accept();
    in = new DataInputStream(server.getInputStream());
    out = new PrintStream(server.getOutputStream());
    String message = “”;
    while((message = in.readLine()) != null){
        //Next Section
    }
    server.close();
}
```

At this point the client knows that a connection has been successfully established and is now able to send and receive messages. On the server side, we will see these messages in an InputStream class that he Socket has. In order for us to use this, the Socket class has a built in function which returns it called getInputStream(). We create our own InputStream as something to reference and then set it equal to the Socket’s. Sending messages works in the same fashion. We create an OutputStream class then set it equal to the Socket class’ OutputStream. The last step at this point is to have our server run in a loop listening to the InputStream.

Communication Protocol

At this current point, the client can send messages but nothing will happen. Our server will receive the message and discard it immediately. Some kind of protocol needs to be established so that the two sides of the connection know how to communicate properly. A protocol is just a set of rules that need to be obeyed in order for the client to receive the information it needs and the server to understand what the client wants. We will start with an extremely basic protocol.

```
while((message = in.readLine()) != null){

    out.println(“I Received this message: “ + message);

}
```

This is not really much of a protocol since all that it really does is repeat what the client said. This is good for testing. Feel free to skip to the next section for instruction on how to do this.

Let us define a more elaborate protocol as an example. Change your code to the following to allow processing of the message.

```
while((message = in.readLine()) != null){

    String args [] = message.split(“ “);

    boolean flag = Process(args); //Should we Wait for another message?

    if(!flag)

        break;

}
```

At this point am taking the message and splitting it into an array of Strings. That way the client can commands with more than one word. These words must be separated by spaces or it is not obeying protocol and will not work. Next I am taking that array and passing it to another function called Process(). This function will parse the message and make a decision based on is contained. This is where we will be implementing the rest of our protocol.

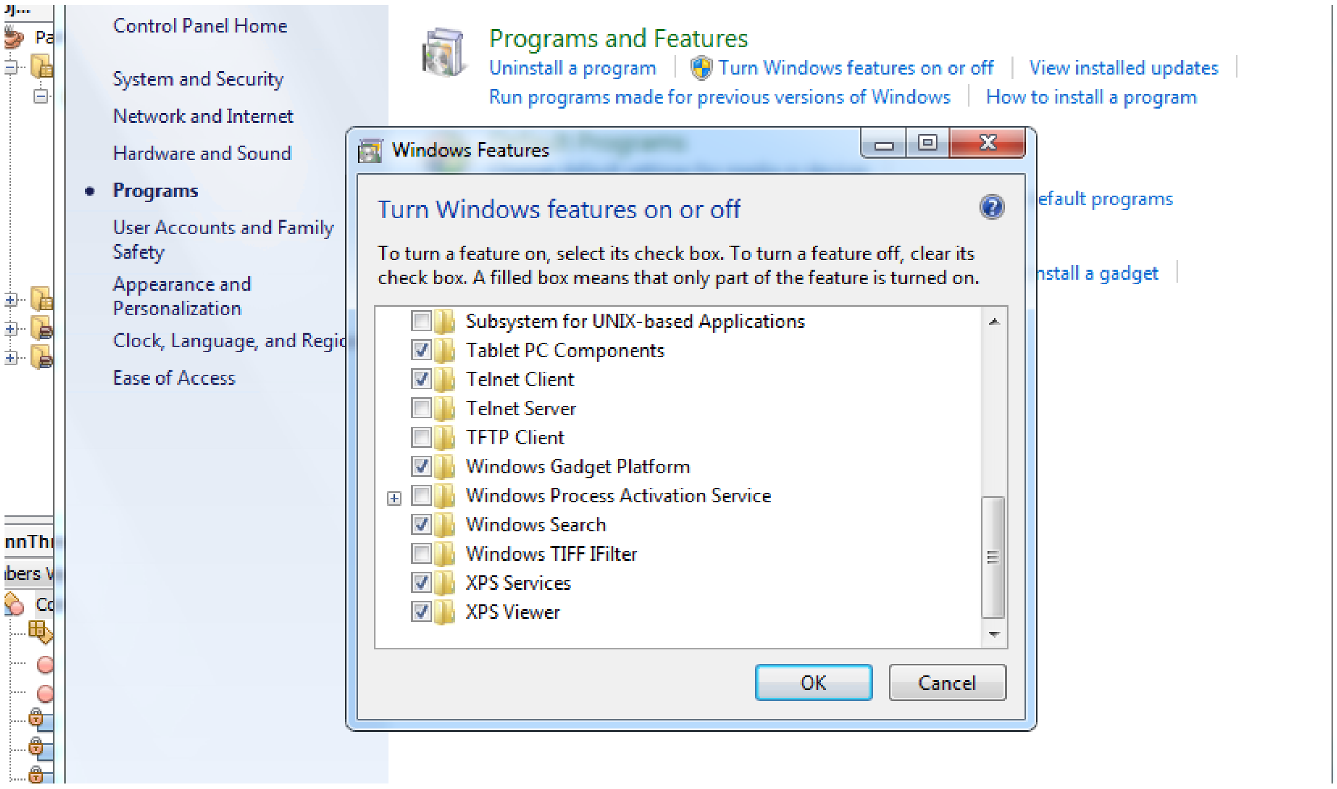
```
public boolean Process(String [] args){

    boolean flag = true;

    if(args[0].equals(“name”)){

        if(args.size() > 1){

            out.println(“Hello “ + args[1]);

        {
```

```
        else{
            out.println("Oops You forgot your name!!");
        }
    }
    else if(args[0].equals("test")){
        out.println("Here is a special message for you!");
        out.println("Java is fun");
    }
    else if(args[0].equals("quit")){
        out.println("GoodBye!");
        flag = false;
    }
    else{
        out.println("Unrecognized Command!!");
        out.println("You are not obeying protocol!!");
    }
    return flag;
}
```

As you can tell from my code, I have defined a few commands to this basic protocol. "test", "name <name>", and "quit" are all valid commands and the server will respond accordingly. If you do not send a known command, the server will tell you and then wait for another one. The "quit" command does exactly as it is supposed to and returns with a false value which closes the connection.

Testing

Testing this server is as easy as establishing a connection to it. Rather than programing a client to do this, which would be very similar to the server, we are going to use Telnet. This program comes standard on Linux systems but if you are using Windows, you may have to install it yourself. This can be done by adding the Windows feature.



Turn Windows features on or off

To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- [ ] Subsystem for UNIX-based Applications
- [✓] Tablet PC Components
- [✓] Telnet Client
- [ ] Telnet Server
- [ ] TFTP Client
- [✓] Windows Gadget Platform
- [ ] Windows Process Activation Service
- [✓] Windows Search
- [ ] Windows TIFF IFilter
- [✓] XPS Services
- [✓] XPS Viewer

When this is installed, open up a command prompt and run the command:

```
telnet localhost PORT
```

where port is the number you specified above.

Extra Features

Multithreading

You may have noticed that this server can only service one client at a time. This can be very restrictive especially if you are planning on using this as a production level server. Java makes threading very easy and by assigning each connection to the server as its own thread, we can handle many clients at the same time. Here is an example of how it can be done.

```
public static void main(String[] args) {


        try{

            ServerSocket Listener = new ServerSocket(1024);

            Socket server;

            int i = 0;
            while(i < maxConn){
                ConnThread connection;

                server = Listener.accept();

                connection = new ConnThread(server);
                Thread t = new Thread(connection);
                t.start();
                i = Thread.activeCount();
            }
        }
        catch (IOException e){
            System.out.println(e);
        }
    }
public class ConnThread implements Runnable{
    private Socket server;
    private String line,input;
    private DataInputStream in;
    private PrintStream out;



    ConnThread(Socket Server){
        this.server = Server;
    }

    public void run() {
        input="";
        try{
            in = new DataInputStream(server.getInputStream());
            out = new PrintStream(server.getOutputStream());

            while((line = in.readLine()) != null && !line.equals("quit")) {
                input=input + line;
                out.println("I got:" + line);
                String args [] = line.split(" ");
                Process(args);
            }

            System.out.println(server.getInetAddress() + " Disconnected ...");
            out.println("Bye");

            server.close();
        }
        catch (IOException e){
            System.out.println(e);
        }
    }

  public void Process(String [] l){
        if(l[0].equals("test")){

        }
    }
}
```

In order to thread, you need to create a separate class that extends the Runnable interface. By doing this, the class will inherit the Run() function which gets called to start the thread. The thread then takes over and handles all communication while the main parent thread continues to listen to others. In my implementation, I have defined a maximum number of threads that my server can handle.

Security

When it comes to communicating over the Internet, keeping your transactions and data to yourself cannot be guaranteed. Anyone who is plugged in can sniff packets and read what commands and data are part of your communication stream. The most secure way to deal with this is to implement encryption. When it comes to internet encryption, SSL/TLS is usually the best way to go and Java makes it simple to implement. Here is an example of the basic echo server that we programmed earlier.

```
import javax.net.ssl.SSLServerSocket;

import javax.net.ssl.SSLServerSocketFactory;

import javax.net.ssl.SSLSocket;

import java.io.*;

public

class EchoServer {

    public static void main(String[] arstring) {

        try {

            SSLServerSocketFactory sslserversocketfactory =

                    (SSLServerSocketFactory) SSLServerSocketFactory.getDefault();

            SSLServerSocket sslserversocket =

                    (SSLServerSocket)
sslserversocketfactory.createServerSocket(1025);

            SSLSocket sslsocket = (SSLSocket) sslserversocket.accept();


            InputStream inputstream = sslsocket.getInputStream();

            InputStreamReader inputstreamreader = new
InputStreamReader(inputstream);

            BufferedReader bufferedreader = new BufferedReader(inputstreamreader);


            String string = null;

            while ((string = bufferedreader.readLine()) != null) {

                System.out.println(string);

                System.out.flush();

            }

        } catch (Exception exception) {

            exception.printStackTrace();
        }
    }
}
```

From what you can tell here, its basically the exact same thing except we are using the SSL version of the Socket classes. We are also using a factory class to quickly take the default settings to skip all advanced settings. Of course this can be changed, and you should look online for more documentation to learn to do this.

Conclusion

Java makes socket programming simple and viable for someone who is still new to coding and has lots more to offer. We went through and built a simple network server that can be customized to fit anyones needs. It can sometimes be hard to understand online documentation, so I hope that this guide was able to fit your needs.

Bibliography

Oracle. (n.d.). *Java™ Platform, Standard Edition 6 API Specification*. Retrieved from http://download.oracle.com/javase/6/docs/api/overview-summary.html