Java Networking

Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocket classes all use TCP to communicate over the network. The DatagramPacket, DatagramSocket, and MulticastSocket classes are for use with UDP.

This hands-on lab takes you through the basics of using Java networking.

Resources

- Exercises on URL (base on **http://java.sun.com/docs/books/tutorial/networking/urls/index.html**)

Exercises

- **Exercise 1: Write client and server**
-
-

Exercise 1: Writing Client and Server

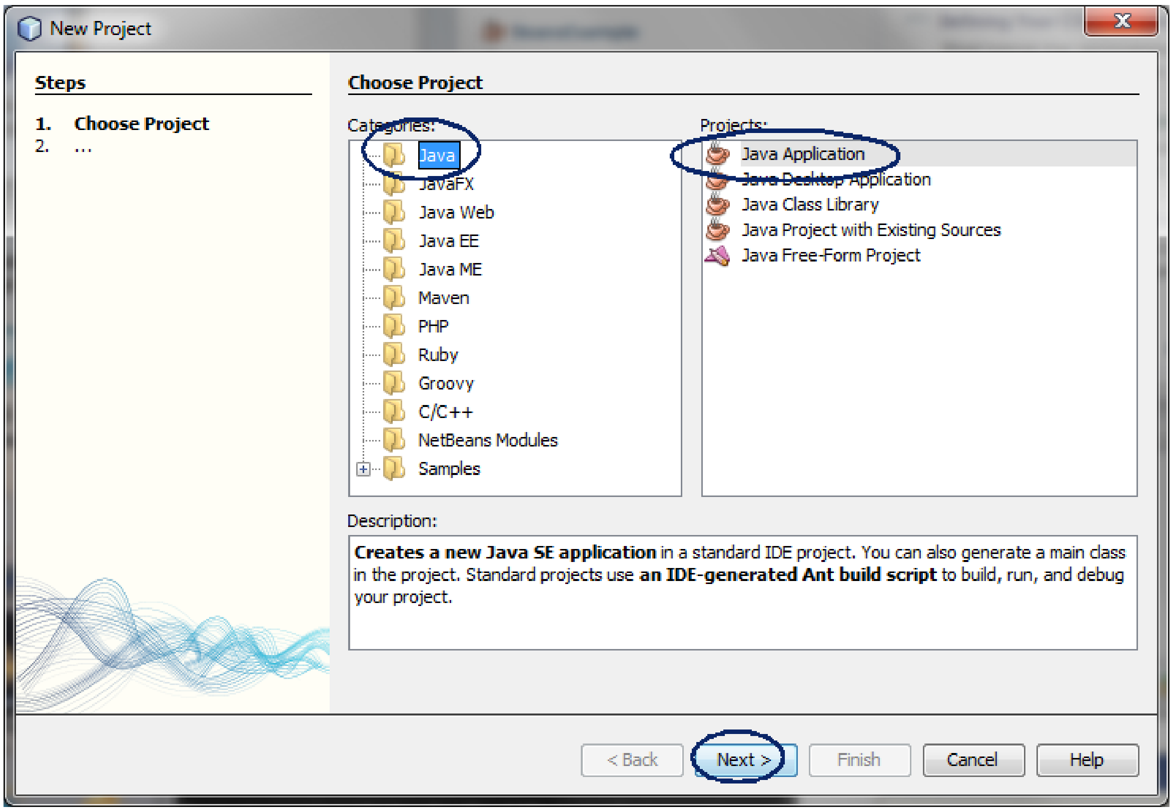
In this exercise, you are going to build and run a simple Hello server and client using Networking API.

1. **Build and run the server side code**
2. **Build and run the client side code**

(1.1) Build and run the server side code

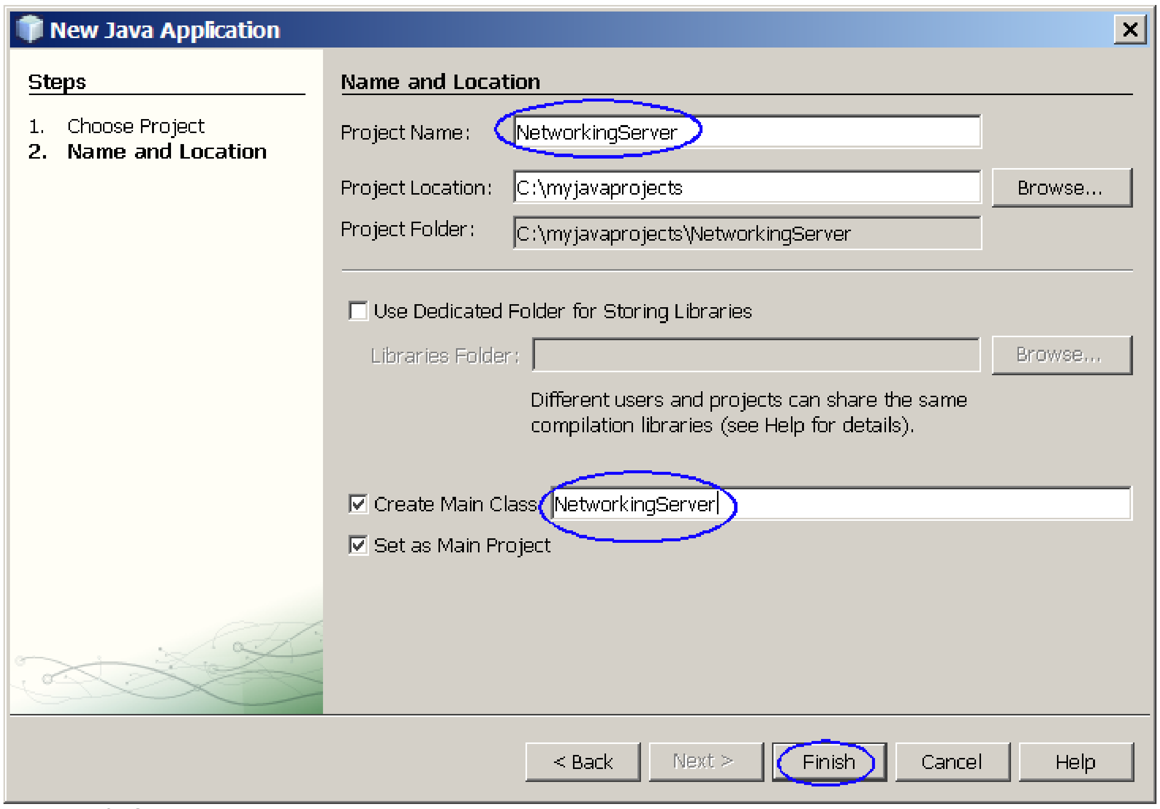
0. Start NetBeans IDE if you have not done so yet.
1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.



Creates a new Java SE application in a standard IDE project. You can also generate a main class in the project. Standard projects use **an IDE-generated Ant build script** to build, run, and debug your project.

- Under **Name and Location** pane, for the **Project Name** field, type in **NetworkingServer** as project name.
- For **Create Main Class** field, type in **NetworkingServer**. (Figure-1.10 below)
- Click **Finish**.



Project Name: NetworkingServer

Project Location: C:\myjavaprojects

Project Folder: C:\myjavaprojects\NetworkingServer

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class NetworkingServer

☑ Set as Main Project

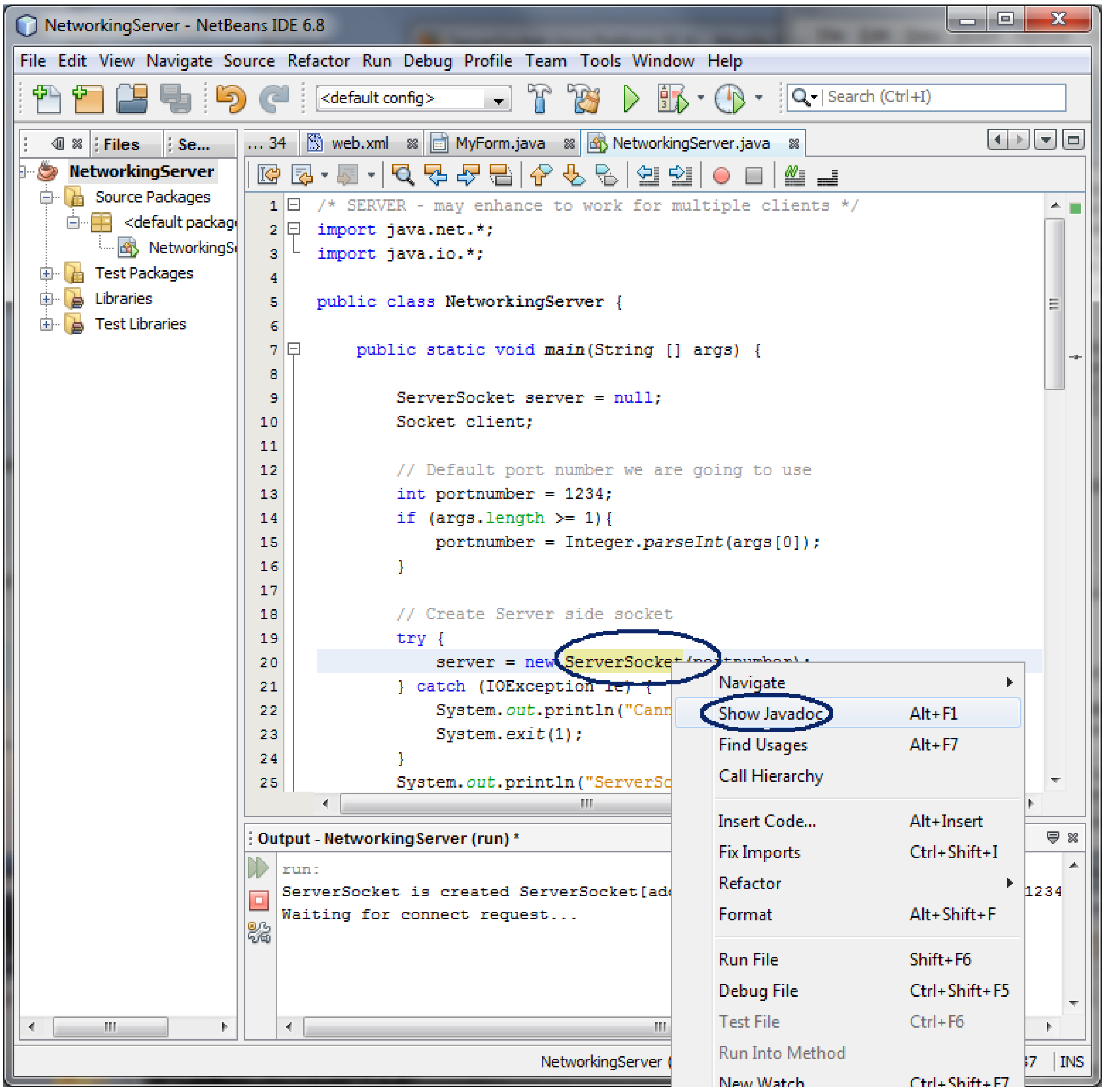
Figure-1.10: Create a new project

- Observe that **NetworkingServer** project appears and IDE generated **NetworkingServer.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **NetworkingServer.java** as shown in Code-1.11 below. Study the code by paying special attention to the bold fonted parts.

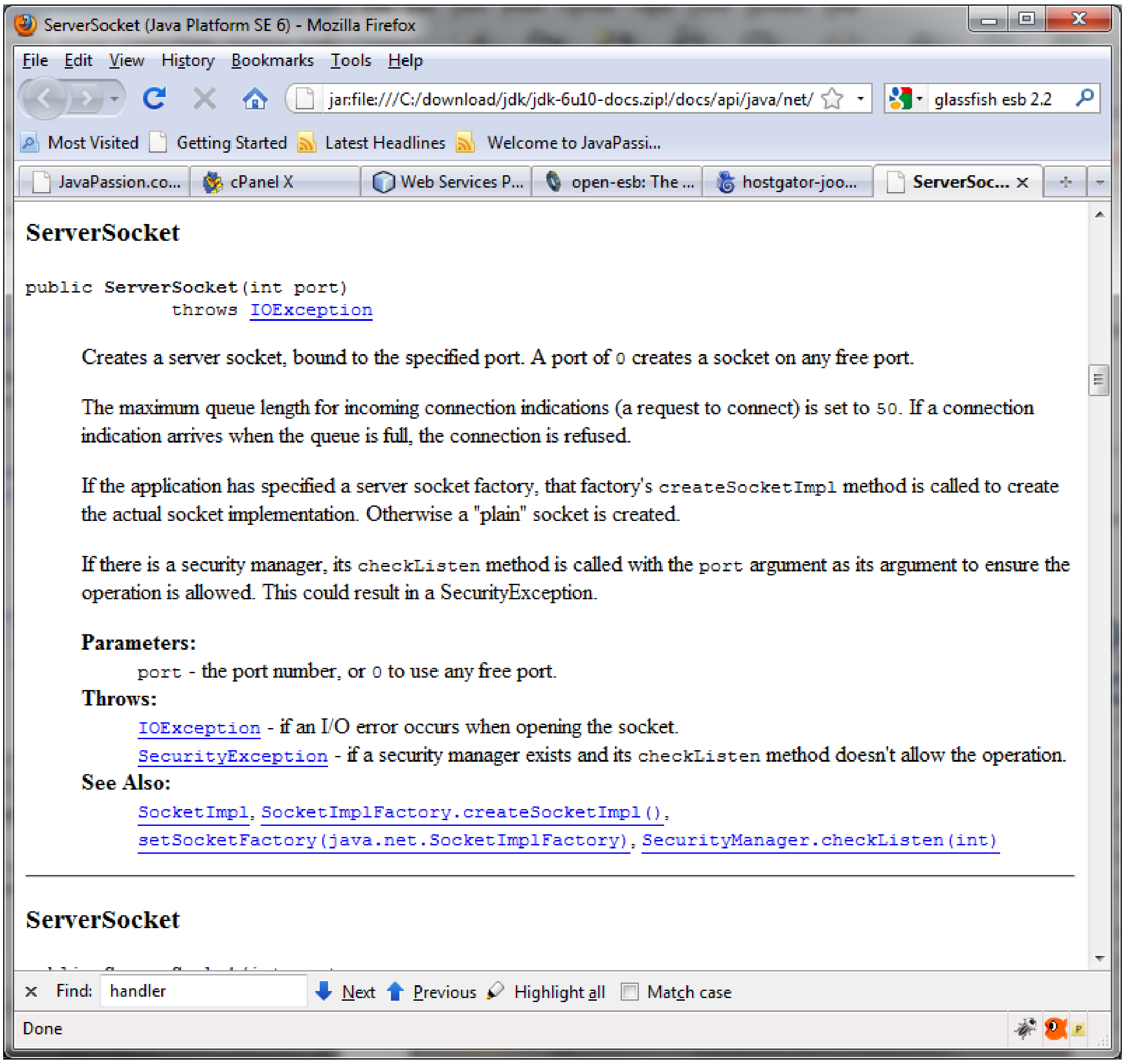
```
/* SERVER – may enhance to work for multiple clients */
import java.net.*;
import java.io.*;

public class NetworkingServer {

    public static void main(String [] args) {

        ServerSocket server = null;
        Socket client;

        // Default port number we are going to use
        int portnumber = 1234;
        if (args.length >= 1){
            portnumber = Integer.parseInt(args[0]);
        }

        // Create Server side socket
        try {
            server = new ServerSocket(portnumber);
        } catch (IOException ie) {
            System.out.println("Cannot open socket." + ie);
            System.exit(1);
        }
        System.out.println("ServerSocket is created " + server);

        // Wait for the data from the client and reply
        while(true) {

            try {

                // Listens for a connection to be made to
                // this socket and accepts it. The method blocks until
                // a connection is made
                System.out.println("Waiting for connect request...");
                client = server.accept();

                System.out.println("Connect request is accepted...");
                String clientHost = client.getInetAddress().getHostAddress();
                int clientPort = client.getPort();
                System.out.println("Client host = " + clientHost + " Client port = " + clientPort);

                // Read data from the client
                InputStream clientIn = client.getInputStream();
                BufferedReader br = new BufferedReader(new
                        InputStreamReader(clientIn));
                String msgFromClient = br.readLine();
                System.out.println("Message received from client = " + msgFromClient);

                // Send response to the client
                if (msgFromClient != null && !msgFromClient.equalsIgnoreCase("bye")) {
                    OutputStream clientOut = client.getOutputStream();
                    PrintWriter pw = new PrintWriter(clientOut, true);
                    String ansMsg = "Hello, " + msgFromClient;
                    pw.println(ansMsg);
                }

                // Close sockets
                if (msgFromClient != null && msgFromClient.equalsIgnoreCase("bye")) {
                    server.close();
                    client.close();
                    break;
                }

            } catch (IOException ie) {
            }
        }
    }
}
```
Code-1.11: NetworkingServer.java

3. Study **ServerSocket** class through context-sensitive Javadoc.



```
/* SERVER - may enhance to work for multiple clients */
import java.net.*;
import java.io.*;

public class NetworkingServer {

    public static void main(String [] args) {

        ServerSocket server = null;
        Socket client;

        // Default port number we are going to use
        int portnumber = 1234;
        if (args.length >= 1){
            portnumber = Integer.parseInt(args[0]);
        }

        // Create Server side socket
        try {
            server = new ServerSocket(portnumber);
        } catch (IOException ie) {
            System.out.println("Cann
            System.exit(1);
        }
        System.out.println("ServerSo
```

```
run:
ServerSocket is created ServerSocket[ad                1234
Waiting for connect request...
```



ServerSocket

```
public ServerSocket(int port)
             throws IOException
```

Creates a server socket, bound to the specified port. A port of `0` creates a socket on any free port.

The maximum queue length for incoming connection indications (a request to connect) is set to `50`. If a connection indication arrives when the queue is full, the connection is refused.

If the application has specified a server socket factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its `checkListen` method is called with the `port` argument as its argument to ensure the operation is allowed. This could result in a SecurityException.

Parameters:
`port` - the port number, or `0` to use any free port.

Throws:
`IOException` - if an I/O error occurs when opening the socket.
`SecurityException` - if a security manager exists and its `checkListen` method doesn't allow the operation.

See Also:
`SocketImpl`, `SocketImplFactory.createSocketImpl()`, `setSocketFactory(java.net.SocketImplFactory)`, `SecurityManager.checkListen(int)`

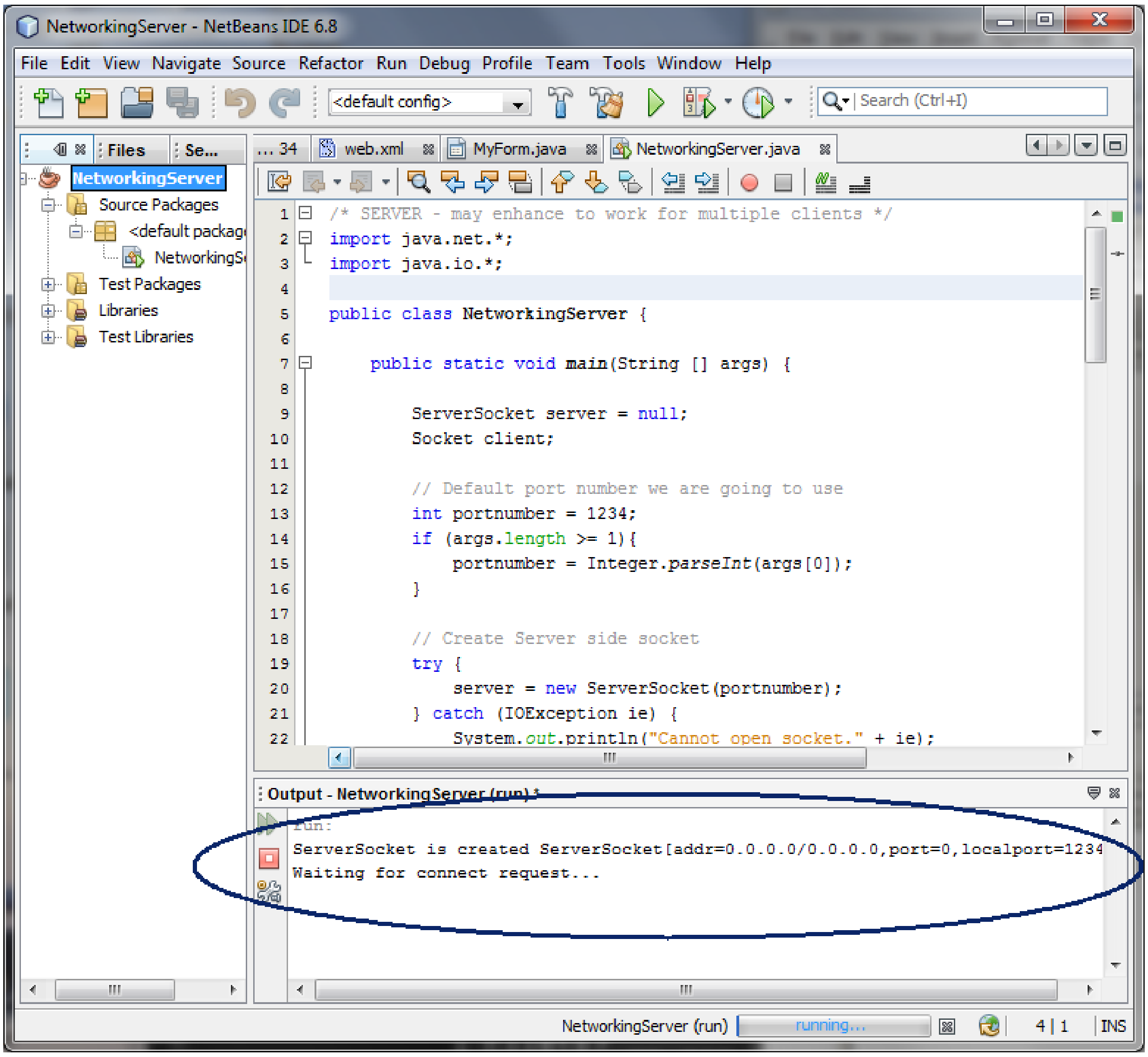
ServerSocket

4. Build and run the project

- Right click **NetworkingServer** project and select **Run**.
- Observe that the server is waiting for a connection request from a client in the **Output** window. (Figure-1.13 below)

```
ServerSocket is created ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=1234]
Waiting for connect request...
```

Figure-1.13: Result of running NetworkingSever application



```
/* SERVER - may enhance to work for multiple clients */
import java.net.*;
import java.io.*;

public class NetworkingServer {

    public static void main(String [] args) {

        ServerSocket server = null;
        Socket client;

        // Default port number we are going to use
        int portnumber = 1234;
        if (args.length >= 1){
            portnumber = Integer.parseInt(args[0]);
        }

        // Create Server side socket
        try {
            server = new ServerSocket(portnumber);
        } catch (IOException ie) {
            System.out.println("Cannot open socket." + ie);
```

```
run:
ServerSocket is created ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=1234
Waiting for connect request...
```

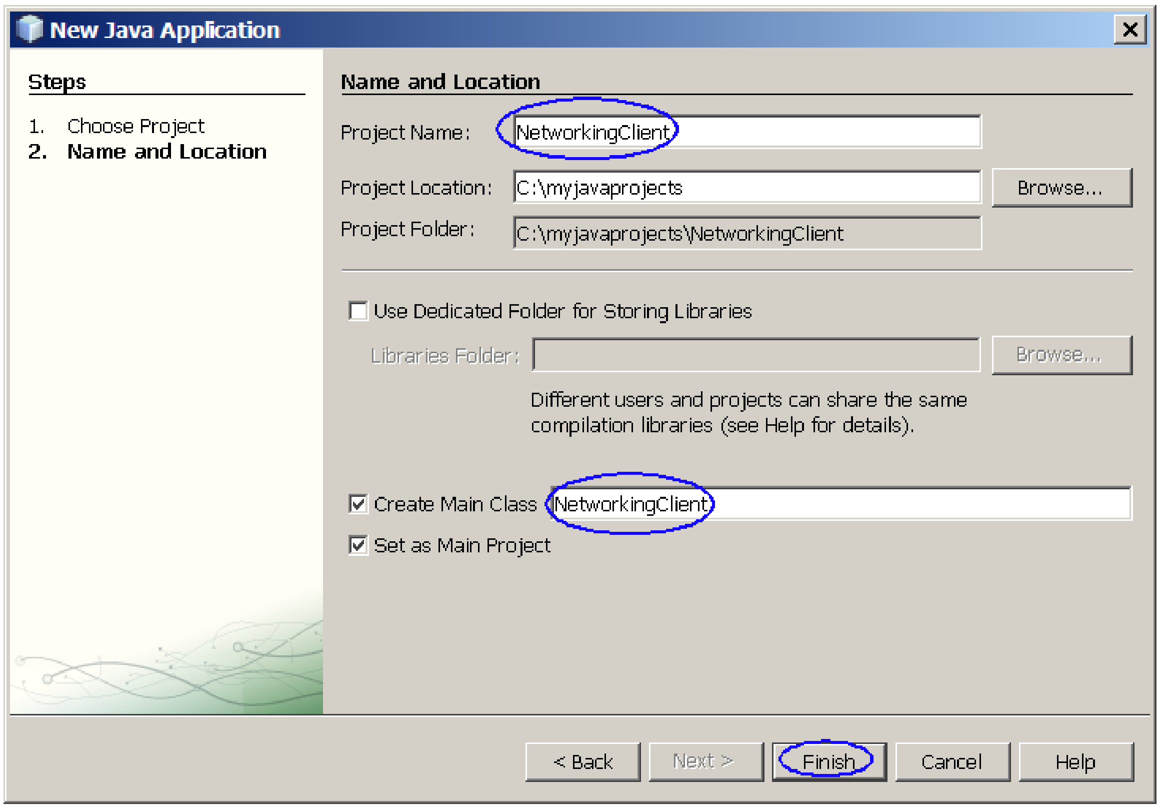
return to top of the exercise

(1.2) Build and run the client side code

1. Create a new NetBeans project

- Select **File**->**New Project (Ctrl+Shift+N)**. The **New Project** dialog box appears.
- Under **Choose Project** pane, select **Java** under **Categories** and **Java Application** under **Projects**.
- Click **Next**.

- Under **Name and Location** pane, for the **Project Name** field, type in **NetworkingClient** as project name.
- For **Create Main Class** field, type in **NetworkingClient**.
- Click **Finish**.


New Java Application

Steps

1. Choose Project
2. **Name and Location**

Name and Location

Project Name: NetworkingClient

Project Location: C:\myjavaprojects

Project Folder: C:\myjavaprojects\NetworkingClient

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Different users and projects can share the same compilation libraries (see Help for details).

☑ Create Main Class NetworkingClient

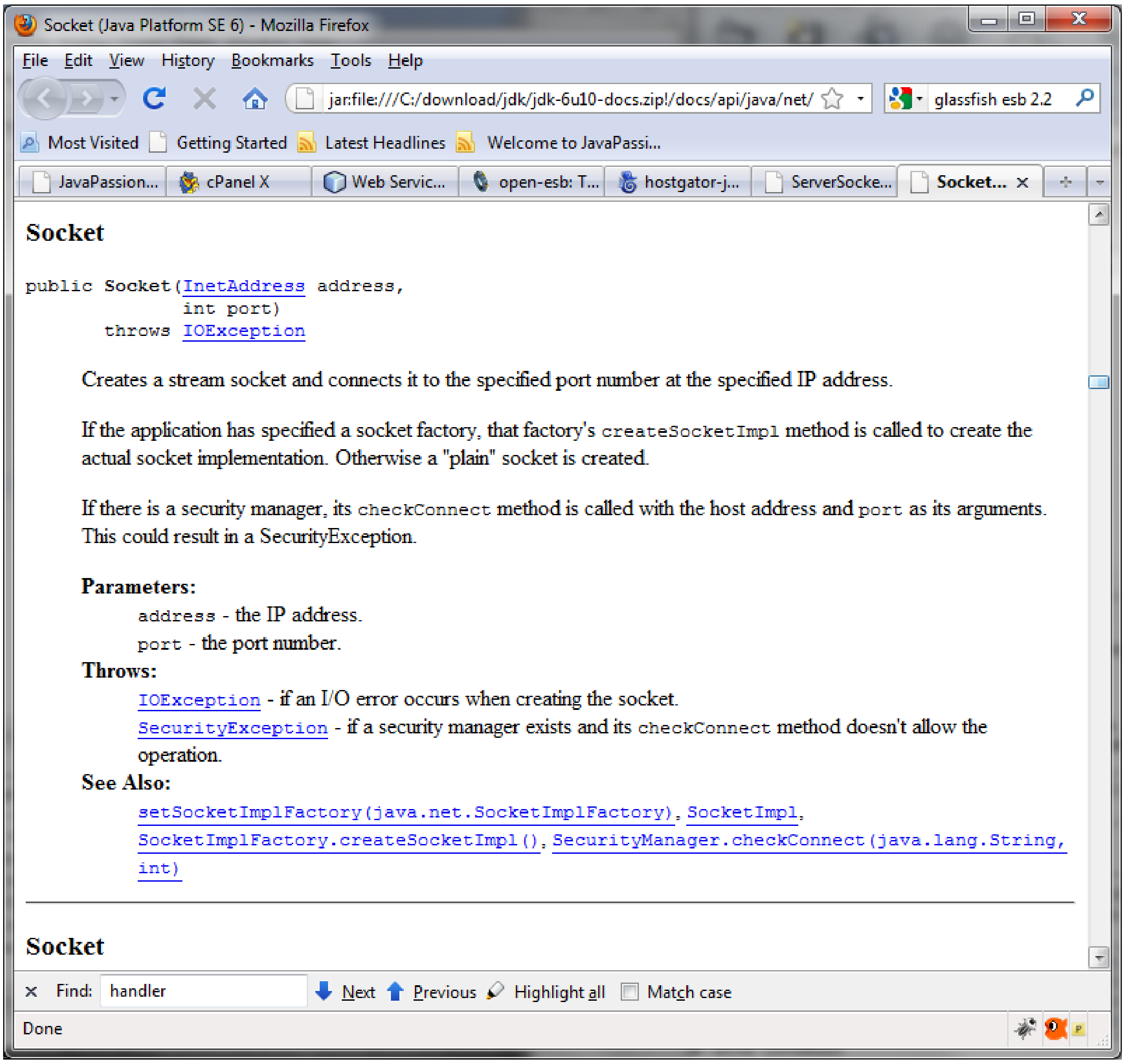
☑ Set as Main Project

- Observe that **NetworkingClient** project appears and IDE generated **NetworkingClient.java** is displayed in the source editor window of NetBeans IDE.

2. Modify the IDE generated **NetworkingClient.java** as shown in Code-1.21 below. Study the code by paying special attention to the bold fonted parts.

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

public class NetworkingClient {

    public static void main(String args[]) {

        Socket client = null;

        // Default port number we are going to use
        int portnumber = 1234;
        if (args.length >= 1){
            portnumber = Integer.parseInt(args[0]);
        }

        for (int i=0; i <10; i++) {
            try {
                String msg = "";

                // Create a client socket
                client = new Socket(InetAddress.getLocalHost(), portnumber);
                System.out.println("Client socket is created " + client);

                // Create an output stream of the client socket
                OutputStream clientOut = client.getOutputStream();
                PrintWriter pw = new PrintWriter(clientOut, true);

                // Create an input stream of the client socket
                InputStream clientIn = client.getInputStream();
                BufferedReader br = new BufferedReader(new
                        InputStreamReader(clientIn));

                // Create BufferedReader for a standard input
                BufferedReader stdIn = new BufferedReader(new
                        InputStreamReader(System.in));

                System.out.println("Enter your name. Type Bye to exit. ");

                // Read data from standard input device and write it
                // to the output stream of the client socket.
                msg = stdIn.readLine().trim();
                pw.println(msg);

                // Read data from the input stream of the client socket.
                System.out.println("Message returned from the server = " + br.readLine());
            pw.close();
            br.close();
            client.close();

            // Stop the operation
            if (msg.equalsIgnoreCase("Bye")) {
               break;
            }

         } catch (IOException ie) {
            System.out.println("I/O error " + ie);
         }
      }
   }
}
```
Code-1.21: NetworkingClient.java

3. Display and study Socket class through context-sensitive Javadoc'ing.


Socket (Java Platform SE 6) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
jar:file:///C:/download/jdk/jdk-6u10-docs.zip!/docs/api/java/net/
glassfish esb 2.2
Most Visited Getting Started Latest Headlines Welcome to JavaPassi...
JavaPassion... cPanel X Web Servic... open-esb: T... hostgator-j... ServerSocke... Socket...
Find: handler Next Previous Highlight all Match case
Done

Socket

```
public Socket(InetAddress address,
              int port)
       throws IOException
```

Creates a stream socket and connects it to the specified port number at the specified IP address.

If the application has specified a socket factory, that factory's `createSocketImpl` method is called to create the actual socket implementation. Otherwise a "plain" socket is created.

If there is a security manager, its `checkConnect` method is called with the host address and `port` as its arguments. This could result in a SecurityException.

Parameters:
`address` - the IP address.
`port` - the port number.

Throws:
`IOException` - if an I/O error occurs when creating the socket.
`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the operation.

See Also:
`setSocketImplFactory(java.net.SocketImplFactory)`, `SocketImpl`, `SocketImplFactory.createSocketImpl()`, `SecurityManager.checkConnect(java.lang.String, int)`

Socket

4. Build and run the project

- Right click **NetworkingClient** project and select **Run**.
- Observe the client is prompting you to enter your name. (Figure-1.23 below)

```
Client socket is created Socket[addr=Passion2/192.168.2.4,port=1234,localport=1775]
Enter your name. Type Bye to exit.
```
Figure-1.23: Waiting for the user to enter name

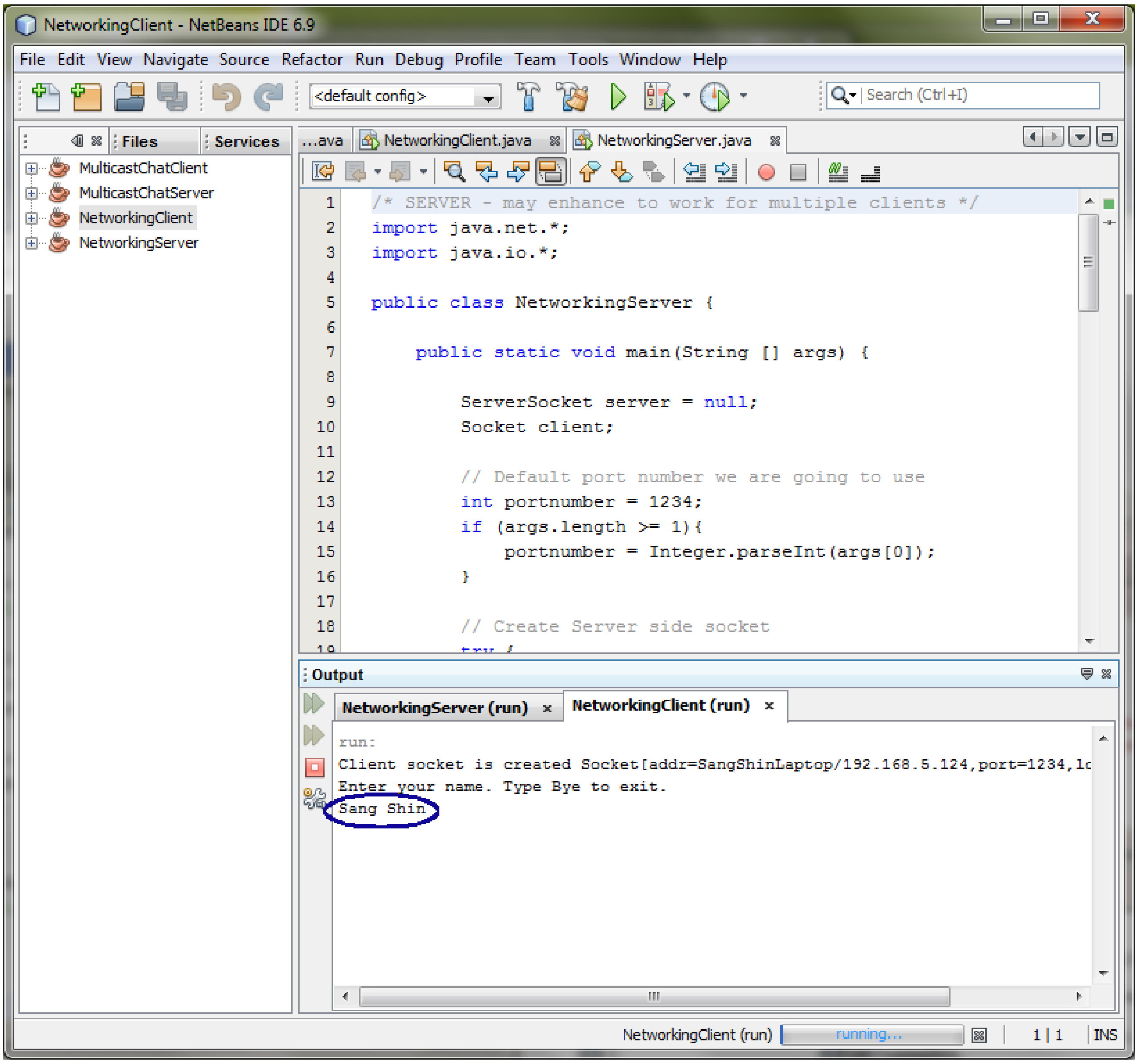
Trouble-shooting: If you see the following exception, it is highly likely that either you have not started the server or if you started the server, the firewall on your system blocks the incoming connection request.

```
I/O error java.net.ConnectException: Connection refused: connect
I/O error java.net.ConnectException: Connection refused: connect
```

```
I/O error java.net.ConnectException: Connection refused: connect
I/O error java.net.ConnectException: Connection refused: connect
```
Figure-1.24: Error condition

Solution: Make sure the server is run first. Also make sure the firewall on your system is turned off.

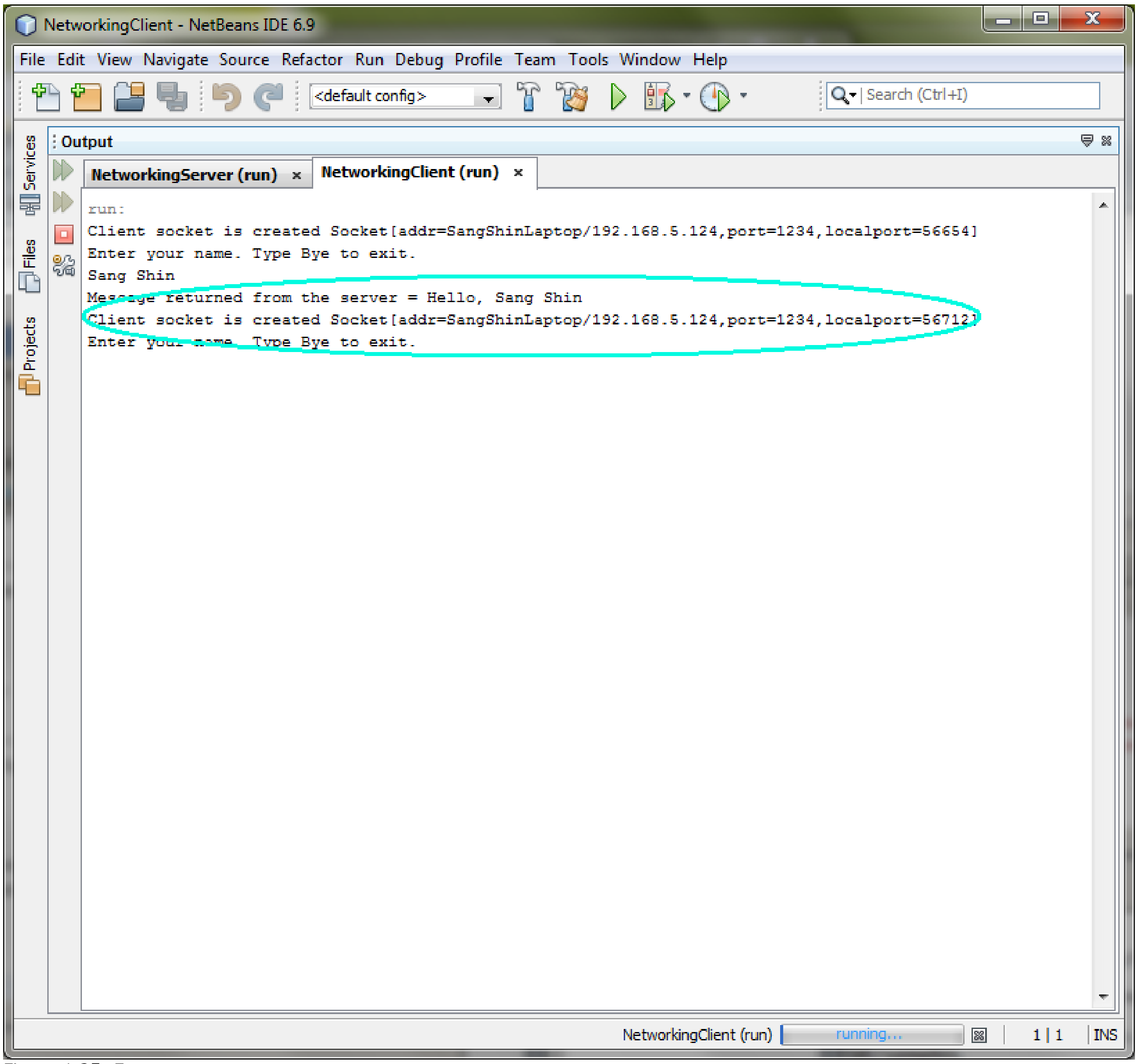
- Enter your name under the line of "**Enter your name. Type Bye to exit.**" in the Output window, like **Sang Shin** in this example, and press **Enter** key.



```java
/* SERVER - may enhance to work for multiple clients */
import java.net.*;
import java.io.*;

public class NetworkingServer {

    public static void main(String [] args) {

        ServerSocket server = null;
        Socket client;

        // Default port number we are going to use
        int portnumber = 1234;
        if (args.length >= 1){
            portnumber = Integer.parseInt(args[0]);
        }

        // Create Server side socket
```

```
run:
Client socket is created Socket[addr=SangShinLaptop/192.168.5.124,port=1234,lo
Enter your name. Type Bye to exit.
Sang Shin
```

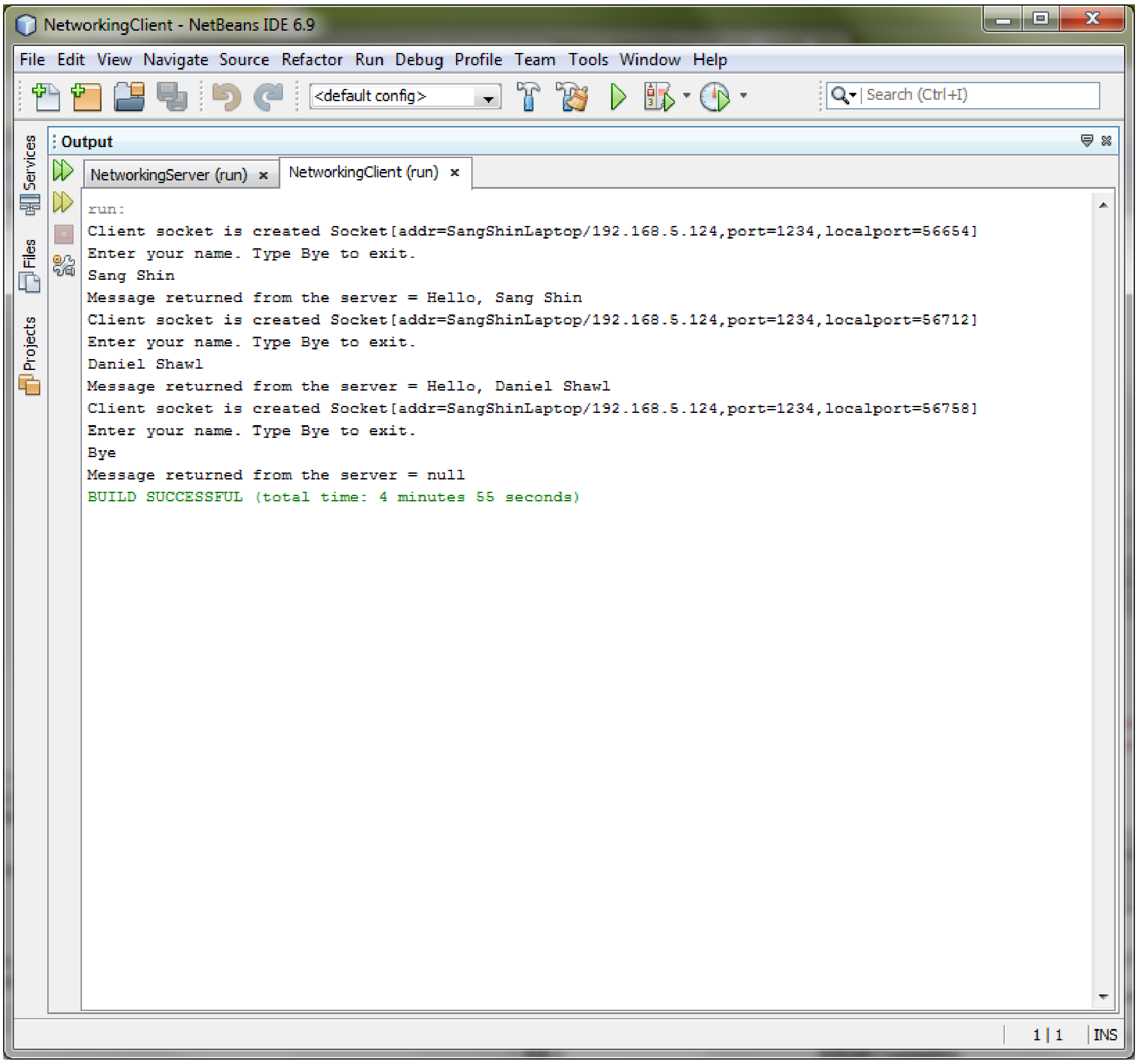
- Observe that the server responds back with "Hello, Sang Shin".



```
run:
Client socket is created Socket[addr=SangShinLaptop/192.168.5.124,port=1234,localport=56654]
Enter your name. Type Bye to exit.
Sang Shin
Message returned from the server = Hello, Sang Shin
Client socket is created Socket[addr=SangShinLaptop/192.168.5.124,port=1234,localport=56712]
Enter your name. Type Bye to exit.
```

Figure-1.25: Enter your name

- Enter a few more names and observe that the server kept sending back responses.
- Type **Bye**.

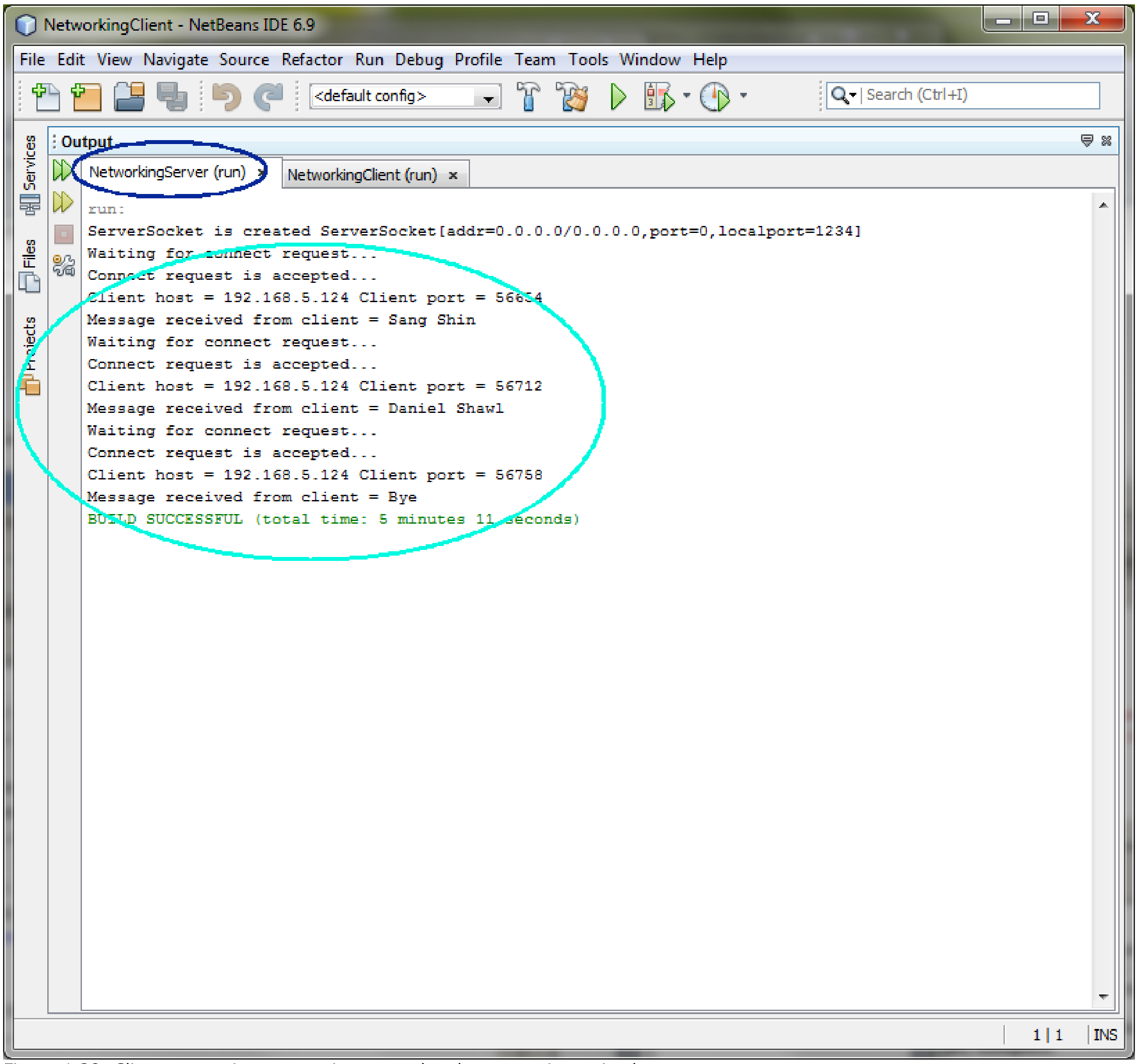

NetworkingClient - NetBeans IDE 6.9
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Search (Ctrl+I)
Output
NetworkingServer (run)
NetworkingClient (run)
Services
Files
Projects
1 | 1
INS

```
run:
Client socket is created Socket[addr=SangShinLaptop/192.168.5.124,port=1234,localport=56654]
Enter your name. Type Bye to exit.
Sang Shin
Message returned from the server = Hello, Sang Shin
Client socket is created Socket[addr=SangShinLaptop/192.168.5.124,port=1234,localport=56712]
Enter your name. Type Bye to exit.
Daniel Shawl
Message returned from the server = Hello, Daniel Shawl
Client socket is created Socket[addr=SangShinLaptop/192.168.5.124,port=1234,localport=56758]
Enter your name. Type Bye to exit.
Bye
Message returned from the server = null
BUILD SUCCESSFUL (total time: 4 minutes 55 seconds)
```

5. Observe the server side.

- Click NetworkingServer (run) tab to see the Output window of the server side.
- Observe that the server received a message, Sang Shin in this example. (Figure-1.27 and Figure-1.28 below.)

```
ServerSocket is created ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=1234]
Waiting for connect request...
Connect request is accepted...
Client host = 192.168.2.4 Client port = 1775
Message received from client = Sang Shin
Waiting for connect request...
Connect request is accepted...
Client host = 192.168.2.4 Client port = 1777
```
Figure-1.27: Client connection request is accepted and message is received


NetworkingClient - NetBeans IDE 6.9
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Search (Ctrl+I)
Output
NetworkingServer (run)
NetworkingClient (run)
Services
Files
Projects
1 | 1
INS

```
run:
ServerSocket is created ServerSocket[addr=0.0.0.0/0.0.0.0,port=0,localport=1234]
Waiting for connect request...
Connect request is accepted...
Client host = 192.168.5.124 Client port = 56654
Message received from client = Sang Shin
Waiting for connect request...
Connect request is accepted...
Client host = 192.168.5.124 Client port = 56712
Message received from client = Daniel Shawl
Waiting for connect request...
Connect request is accepted...
Client host = 192.168.5.124 Client port = 56758
Message received from client = Bye
BUILD SUCCESSFUL (total time: 5 minutes 11 seconds)
```

Figure-1.28: Client connection request is accepted and message is received

return to top of the exercise

Summary

In this exercise, you learned how to build and run a simple Hello server and client using Networking API.

1. YOUR TASK is to create your own NetworkingServer and NetworkingClient applications as following. We will name them as MyComputingServer and MyComputingClient.
The client send two numbers and the server sends back addition of the two numbers.