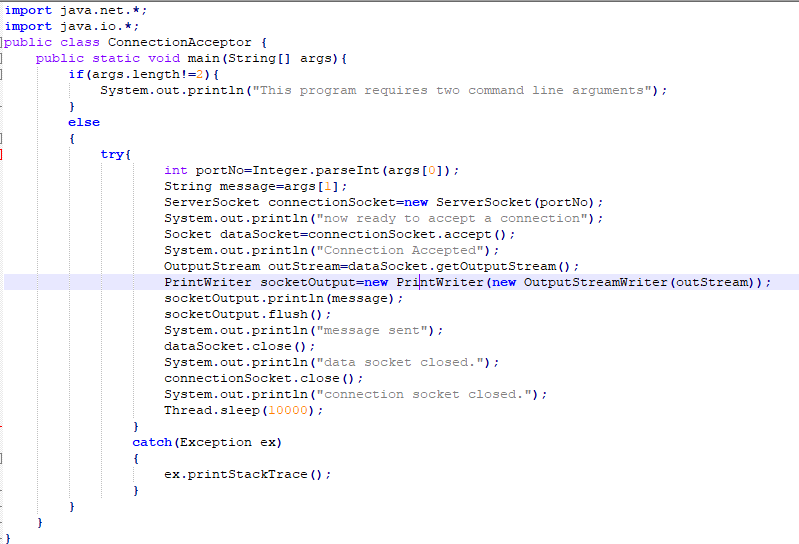
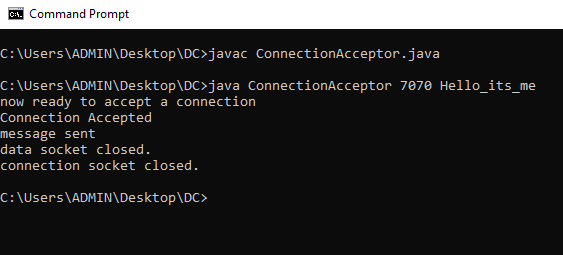
**LAB # 2**

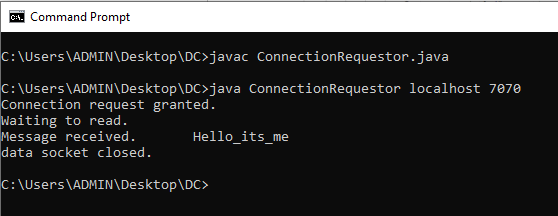
**OBJECT : To learn creation of stream sockets.**

**Task 1 : Compile and run the above code. Start the acceptor first and then the requestor with appropriate command line arguments. Describe and explain the output.**





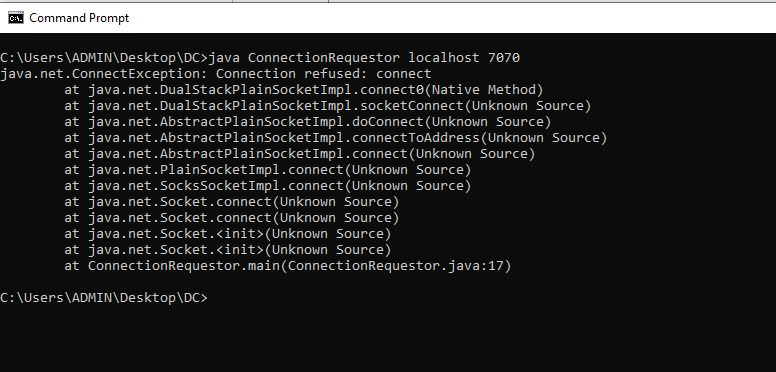




**TASK 2 : Now run the code again, but reverse the order of program’s execution. Start the requestor first and then the acceptor. Describe and explain the outcome.**

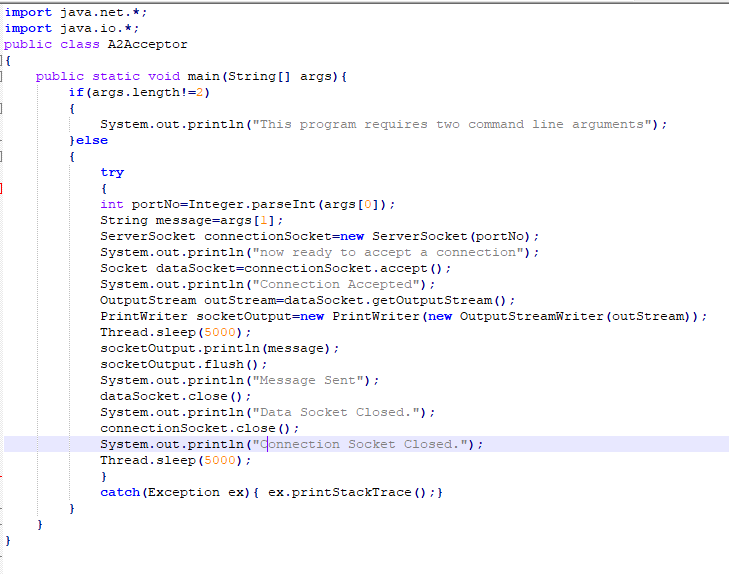
**OUTPUT :**

**In order to make connection, Connection Request needs a port no that comes from Connection Acceptor. Connection Acceptor allocates the port no to Connection Requestor. Hence Connection Acceptor is not running so Connection Request can’t make connection and “Connection refused” exception has occurred.**

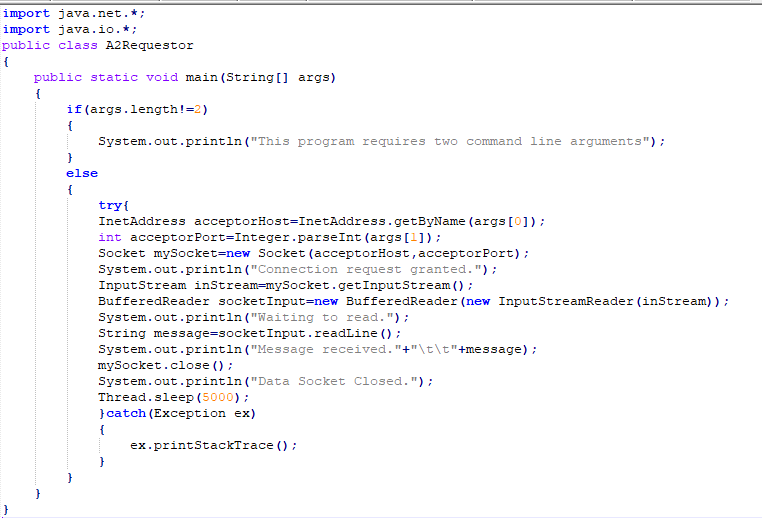


**TASK 3 : Add a time delay of 5 seconds in the ConnectionAcceptor process just before the message is written to the socket, then run the program. This will show you the blocking at the receiver. Show a trace of the output of the processes.**

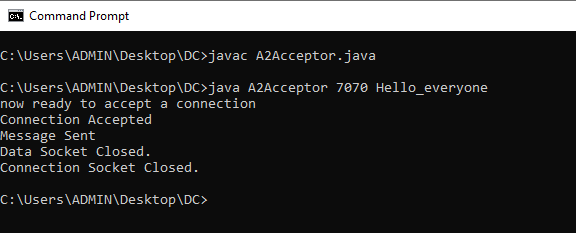
TASK2Acceptor.java

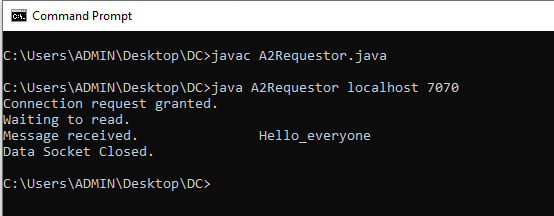


TASK2 Requestor.java



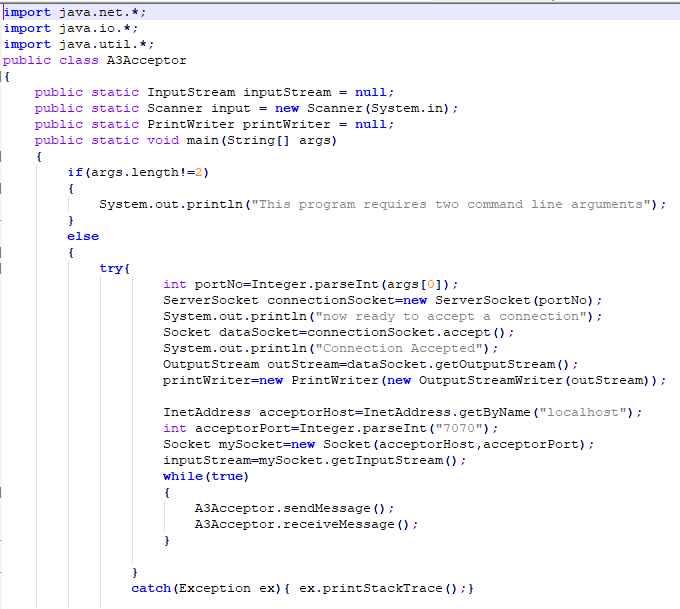
**OUTPUT :**

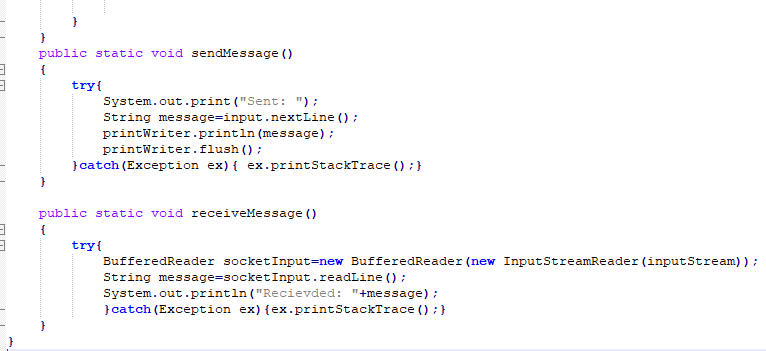




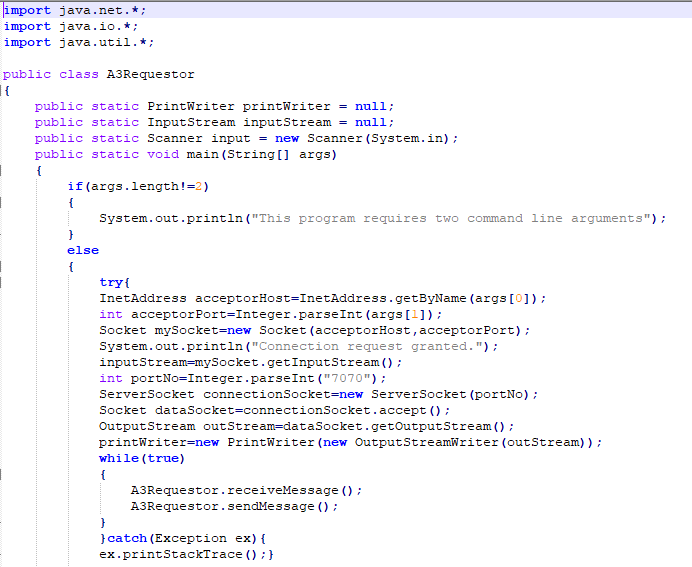
**Task#04: Modify the sample code to include two way communication between the client and the server.**

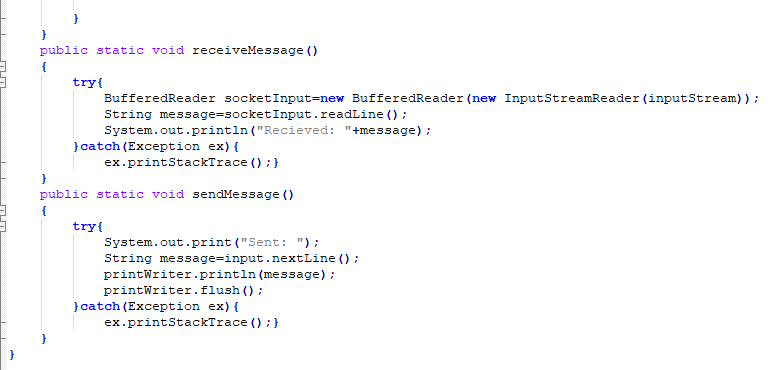
**A3Acceptor.java**



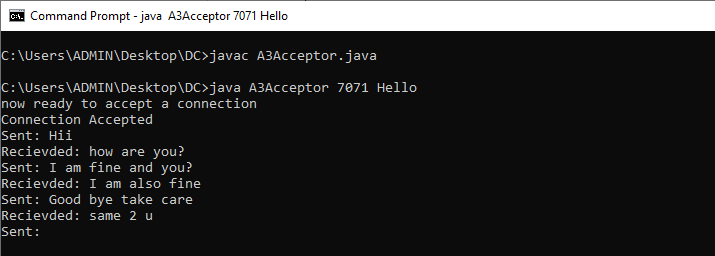


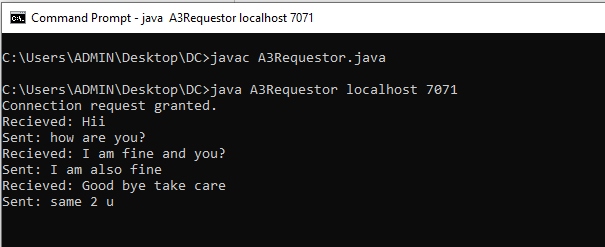
**A3Requestor.java**





**OUTPUT :**





**Task#05: Modify the sample code to send complete files between the client to the server.**

**ClientFile.java**

import java.io.BufferedOutputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.Socket;

public class SimpleFileClient {

public final static int SOCKET\_PORT = 13267;

public final static String SERVER = "127.0.0.1";

public final static String

FILE\_TO\_RECEIVED = "EdxA.docx";

public final static int FILE\_SIZE = 6022386;

public static void main (String [] args ) throws IOException {

int bytesRead;

int current = 0;

FileOutputStream fos = null;

BufferedOutputStream bos = null;

Socket sock = null;

try {

sock = new Socket(SERVER, SOCKET\_PORT);

System.out.println("Connecting...");

byte [] mybytearray = new byte [FILE\_SIZE];

InputStream is = sock.getInputStream();

fos = new FileOutputStream(FILE\_TO\_RECEIVED);

bos = new BufferedOutputStream(fos);

bytesRead = is.read(mybytearray,0,mybytearray.length);

current = bytesRead;

do {

bytesRead =

is.read(mybytearray, current, (mybytearray.length-current));

if(bytesRead >= 0) current += bytesRead;

} while(bytesRead > -1);

bos.write(mybytearray, 0 , current);

bos.flush();

System.out.println("File " + FILE\_TO\_RECEIVED

+ " downloaded (" + current + " bytes read)");

}

finally {

if (fos != null) fos.close();

if (bos != null) bos.close();

if (sock != null) sock.close();

}

}

}

SimpleFileServer.java

import java.io.BufferedInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.OutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class SimpleFileServer {

public final static int SOCKET\_PORT = 13267;

public final static String FILE\_TO\_SEND = "Edx.docx";

public static void main (String [] args ) throws IOException {

FileInputStream fis = null;

BufferedInputStream bis = null;

OutputStream os = null;

ServerSocket servsock = null;

Socket sock = null;

try {

servsock = new ServerSocket(SOCKET\_PORT);

while (true) {

System.out.println("Waiting...");

try {

sock = servsock.accept();

System.out.println("Accepted connection : " + sock);

File myFile = new File (FILE\_TO\_SEND);

byte [] mybytearray = new byte [(int)myFile.length()];

fis = new FileInputStream(myFile);

bis = new BufferedInputStream(fis);

bis.read(mybytearray,0,mybytearray.length);

os = sock.getOutputStream();

System.out.println("Sending " + FILE\_TO\_SEND + "(" + mybytearray.length + " bytes)");

os.write(mybytearray,0,mybytearray.length);

os.flush();

System.out.println("Done.");

}

finally {

if (bis != null) bis.close();

if (os != null) os.close();

if (sock!=null) sock.close();

}

}

}

finally {

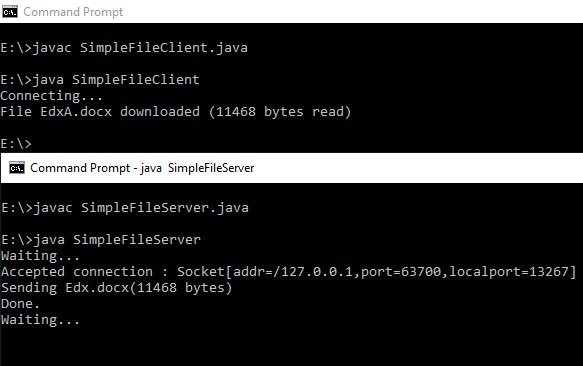
if (servsock != null) servsock.close();

}

}

}

**OUTPUT :**



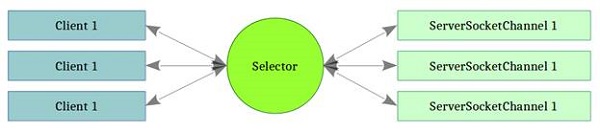
**Task#06: Explore the non-blocking java socket API in the nio package and implement a sample program.**

Starting [JDK 1.4](https://crunchify.com/where-is-java-installed-on-my-mac-osx-system/), NIO was created to allow all [Java programmers](https://crunchify.com/category/java-tutorials/) to implement very high-speed input/output without having to deal with custom native code. NIO uses java.nio.buffer library compare to [simple I/O](https://crunchify.com/java-file-copy-example-simple-way-to-copy-file-in-java/) which drains and fills back buffer internally any operating system.

In this tutorial we will go over java.nio.channels and java.nio.channels.Selectorlibraries.

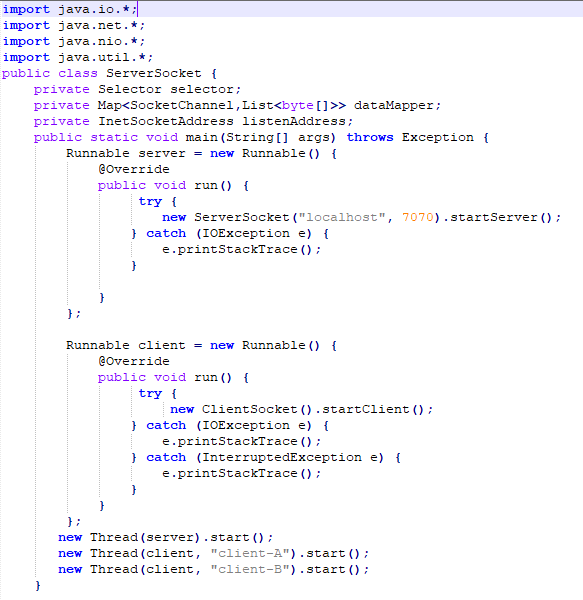
* channels represent connections to entities that are capable of performing I/O operations, such as files and sockets; defines selectors, for multiplexed, non-blocking I/O operations.
* selector may be created by invoking the open method of this class, which will use the system’s default selector provider to create a new selector.

How it works:

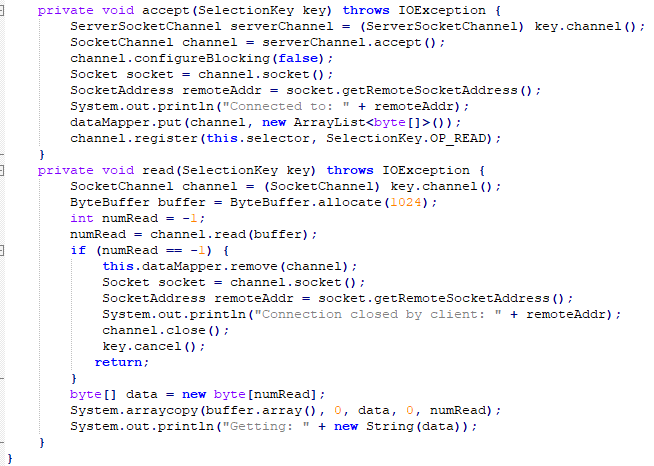


**Code:**

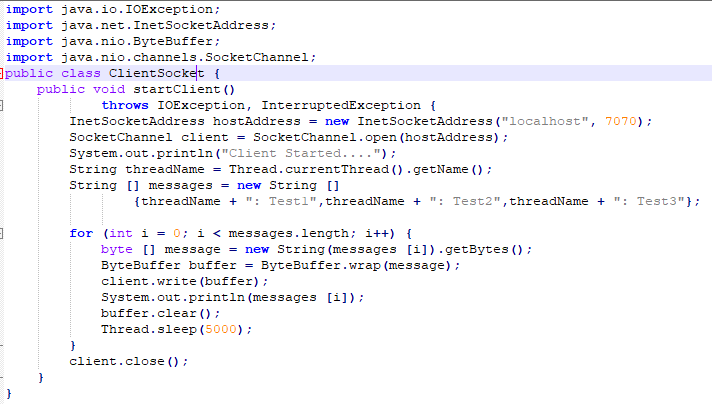
**ServerSocket.java**







**ClientSocket.java**



**OUTPUT :**

