Assignment No:- B9

- **Title**

  To implement Echo server using Java.

- **Problem Definition**

  Implement a Multi-threading application for echo server using socket programming in JAVA.

- **Learning Objective**

  To study a Multi-threading application for echo server using socket programming in JAVA.

- **Learning Outcome**

  Successfully implemented Echo server using Java.

- **Software and Hardware Requirements**

  - Software
    - Ubuntu 14.04 OS.
    - Java Compiler .
    - Java .Net package.
  - Hardware
    - Dual core CPU.
    - 64 bit processor .

- **Theory** :

1. Introduction

A. Thread life cycle

A thread can be one of five states. According to sun, their is only four states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state. But for understanding the threads, we are explaining it in the five states. The life cycle thread in java is controlled by JVM. The java thread states are as follows:

1. New

   The thread is in new state if you create an instance of Thread class But before the invocation of start() method.

2. Runnable

   The thread is in runnable state after invocation of start() method, But the thread scheduler has not selected it to be the running thread.

3. Running

   The thread is in running state if the thread scheduler has selected it.

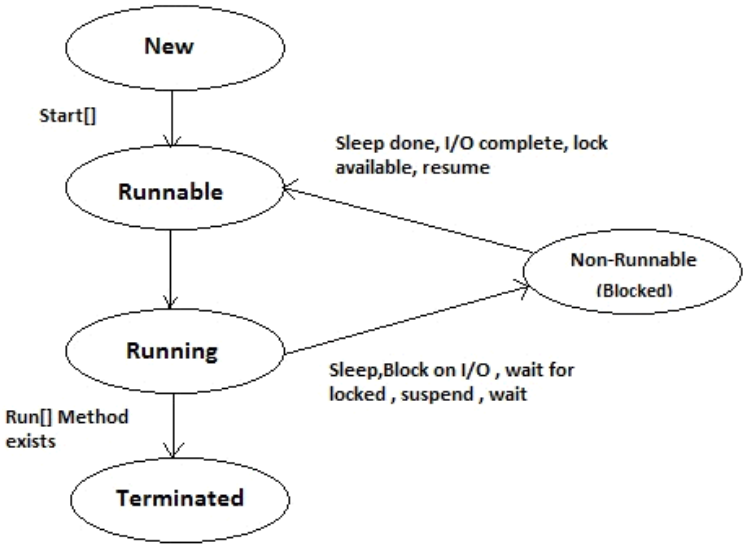


Figure 1: Thread Life Cycle

4. Non-runnable (Blocked)

   This is the state when the thread is still alive, but is currently not eligible to run.

5. Terminated

   A thread is in terminated or dead state when its run() method exits.

B. Multithreading

Multithreading in java is a process of executing multiple threads simultaneously. Thread is basically light-weight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share common mem- ory area. They don't allocate separate memory area so saves memory, and context- switching between the threads takes less time than process. Java multithreading is mostly used in games, animation etc.

2. Related Concepts– Socket Programming

Socket programming is useful for building client-server applications.

The Server

1. Creates a socket with some port number (≥ 1024): ServerSocketechoServer = new ServerSocket(6789);
2. Waits for client connection : Socket clientSocket = echoServer.accept();
3. Gets input/output streams : BufferedReader is = new BufferedReader(new InputStreamReader(clientSocket.getInput PrintStreamos = new PrintStream(clientSocket.getOutputStream());
4. Exchanges information with the client : String line = is.readLine(); os.println(”Echo: ” + line);
5. Clean up : is.close(); os.close(); clientSocket.close(); echoServer.close();

The client

1. Creates a socket with the same port number : Socket clientSocket = new Socket(”localhost”, 6789);
2. Gets input/output streams: BufferedReader is = new BufferedReader(new InputStreamReader(clientSocket.getInput PrintStreamos = new PrintStream(clientSocket.getOutputStream());
3. Exchanges information with the server: String line = is.readLine(); os.println(”Echo: ” + line);
4. Clean up : is.close(); os.close(); clientSocket.close(); echoClient.close();

Echo server

What is it and what is it for ?

An ”echo server” is a server that does nothing more than sending back whatever is sent to it. Hence the name : echo What can you use it for ? Whatever you feel like. Practical applications could be network and connectivity testing and troubleshoot- ing.

Assume you’ve build a rather complex network with VLANs and subnets, and really strick firewalls between those subnets, and you’re beginning to wonder if a client on one segment of the network will still

be able to connect to your web server, database server, ... on some other segment. A ping or a traceroute will establish if the server (IP address) can be reached but does not tell you if an application will be able to connect to the desired port on the server and whether a reply from the server will be able to reach the client again.

This ”echo server” can be set up to listen on any desired (tcp) port to simulate whatever application you want to run (eg web server = port 80, Microsoft SQL Server = port 1433, etc). From the client machine, you can then telnet to this port. When a telnet connection has been established, everything you type will be echoed back to your screen, indicating that the telnet client and the echo server can talk to each other : you've established connectivity at the application level.

In a similar way, you can use this echo server to troubleshoot networks, test a firewall (eg ”if I have a server listening on port 123, wil my firewall allow connections to it ?) and so on.

Echo server

1. The client reads a line from its standard input and writes that line to the server.
2. The server reads a line from its network input and echoes the line back to the client over the network.
3. The client reads the echoed line from the network and prints it on its standard output.

Multithreaded Server Advantages

The advantages of a multithreaded server compared to a single threaded server are summed up below:

* Less time is spent outside the accept() call.
* Long running client requests do not block the whole server.
* In a single threaded server long running requests may make the server unresponsive for a long period. This is not true for a multithreaded server, unless the long-running request takes up all CPU time and/or network bandwidth.

Multi-threading Environment in Java

Java is a multi threaded programming language which means we can develop multi threaded program using Java. A multi threaded program contains two or more parts that can run concurrently and each part can handle different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition multitasking is when multiple processes share common processing resources such as a CPU. Multi threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Runnable Interface

Java Thread By Implementing Runnable Interface

A Thread can be created by extending Thread class also. But Java allows only one class to extend, it wont allow multiple inheritance. So it is always better to create a thread by implementing Runnable interface. Java allows you to impliment multiple interfaces at a time.

By implementing Runnable interface, you need to provide implementation for run() method.

To run this implementation class, create a Thread object, pass Runnable implementation class object to its constructor. Call start() method on thread class to start executing run() method.

Implementing Runnable interface does not create a Thread object, it only defines an entry point for threads in your object. It allows you to pass the object to the Thread(Runnable implementation) constructor.

public interface Runnable

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has not yet been stopped.

In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying or enhancing the fundamental behavior of the class.

void run()

When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

The general contract of the method run is that it may take any action whatsoever.

A Simple Example

```
class MultithreadingDemo implements Runnable{
  public void run(){
    System.out.println("My thread is in running state.");
  }
  public static void main(String args[]){
     MultithreadingDemo obj=new MultithreadingDemo();
     Thread tobj =new Thread(obj);
     tobj.start();
 }
}
```

Output:

My thread is in running state.

Thread Class

public class Thread
extends Object
implements Runnable

A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if and only if the creating thread is a daemon.

When a Java Virtual Machine starts up, there is usually a single non-daemon thread (which typically calls the method named main of some designated class). The Java Virtual Machine continues to execute threads until either of the following occurs:

The exit method of class Runtime has been called and the security manager has permitted the exit operation to take place.
All threads that are not daemon threads have died, either by returning from the call to the run method or by throwing an exception that propagates beyond the run method.

There are two ways to create a new thread of execution. One is to declare a class to be a subclass of Thread. This subclass should override the run method of class Thread. An instance of the subclass can then be allocated and started. For example, a thread that computes primes larger than a stated value could be written as follows:

Constructors of Thread class

Thread ( )
Thread (String str)
Thread (Runnable r)
Thread (Runnable r, String str)

You can create new thread, either by extending Thread class or by implementing Runnable interface. Thread class also defines many methods for managing threads. Some of them are,

Method	Description
setName()	to give thread a name
getName()	return thread's name
getPriority()	return thread's priority
isAlive()	checks if thread is still running or not
join()	Wait for a thread to end
run()	Entry point for a thread
sleep()	suspend thread for a specified time
start()	start a thread by calling run() method

Some Important points to Remember

When we extend Thread class, we cannot override setName() and getName() functions, because they are declared final in Thread class.

While using sleep(), always handle the exception it throws.

static void sleep(long milliseconds) throws InterruptedException

3. **Algorithm:**

– Step 1:- Start connection with port Number greater than 1024 and less than 65535.

– Step 2:- Waiting for Client connection.

– Step 3:- Request from client to Server with same port number.

– Step 4:- Create new thread for every new client connection.

– Step 5:- Accept the input form client and perform some basic operation on it.

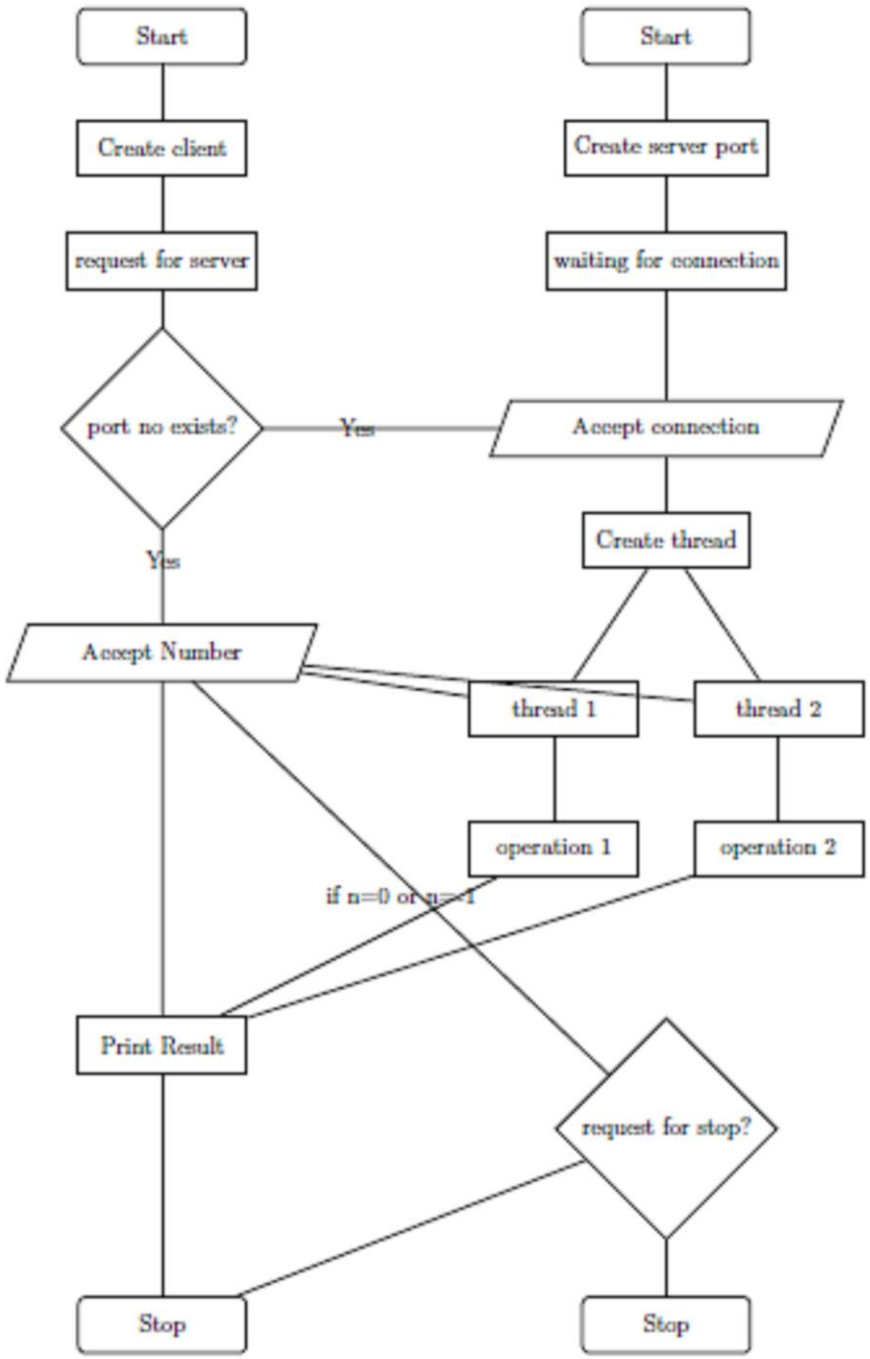
– Step 6:- Return result to client.

– Step 7:- Stop client and goto step 9.

– Step 8:- Stop server.

– Step 9:- Stop

4. Flowchart


4. Flowchart
Start
Create client
request for server
port no exists?
Yes
Yes
Accept Number
Start
Create server port
waiting for connection
Accept connection
Create thread
thread 1
thread 2
operation 1
operation 2
if n=0 or n=1
Print Result
request for stop?
Stop
Stop

5. Mathematical model

In Java Client-Server program

Let U = {D , ND , S , F , f , In , Out}

U : Universal set

D : Deterministic data values i.e. is initial values

ND : Non deterministic data values i.e. values given by user.

S : Cases for success

F : Cases for failure

f : Function for average calculation

In : Input data

Out : Output data

In : Number

D : Port number ranges from 1024 to 65535

Out : Square of given number f : n * n

where n = Given number

S : port no > 1024, connection established successfully

F : port no < 1024, port already in use

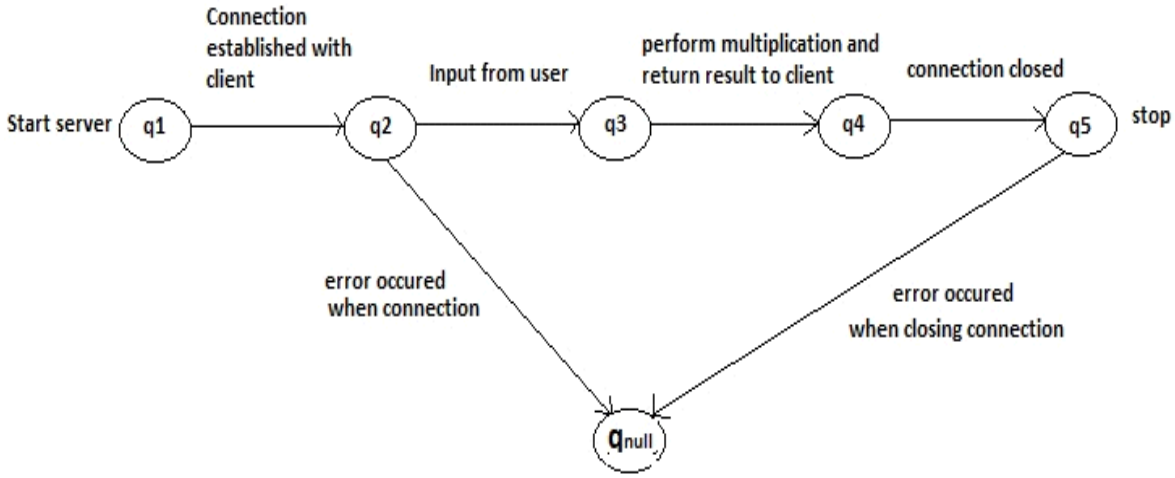


Figure 2: State diagram

6. SRS Diagram:

Software Specification	Description
1.Project Scope	To write a program to create multi-threaded client server program
2. Functional requirements	Use Client Server architecture using Java
3. Non functional requirements	Use Client Server architecture using Python
4. Design	Port number sould not excced the range 1024 to 65535
5. Implementation	Using concurrent java and concurrent threads
6.Hardware requirements	Dual core CPU, i.e. 4 threads.
7. External interfacing	No external interfacing required

Figure 3: SRS diagram

7. UML Diagrams:

– Use case Diagram:

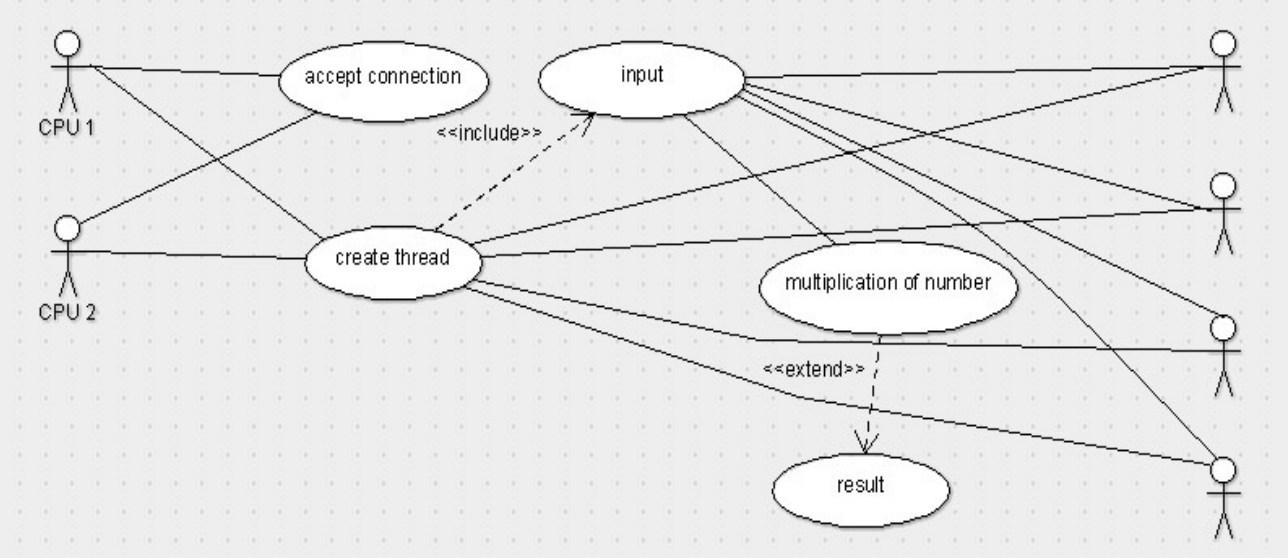
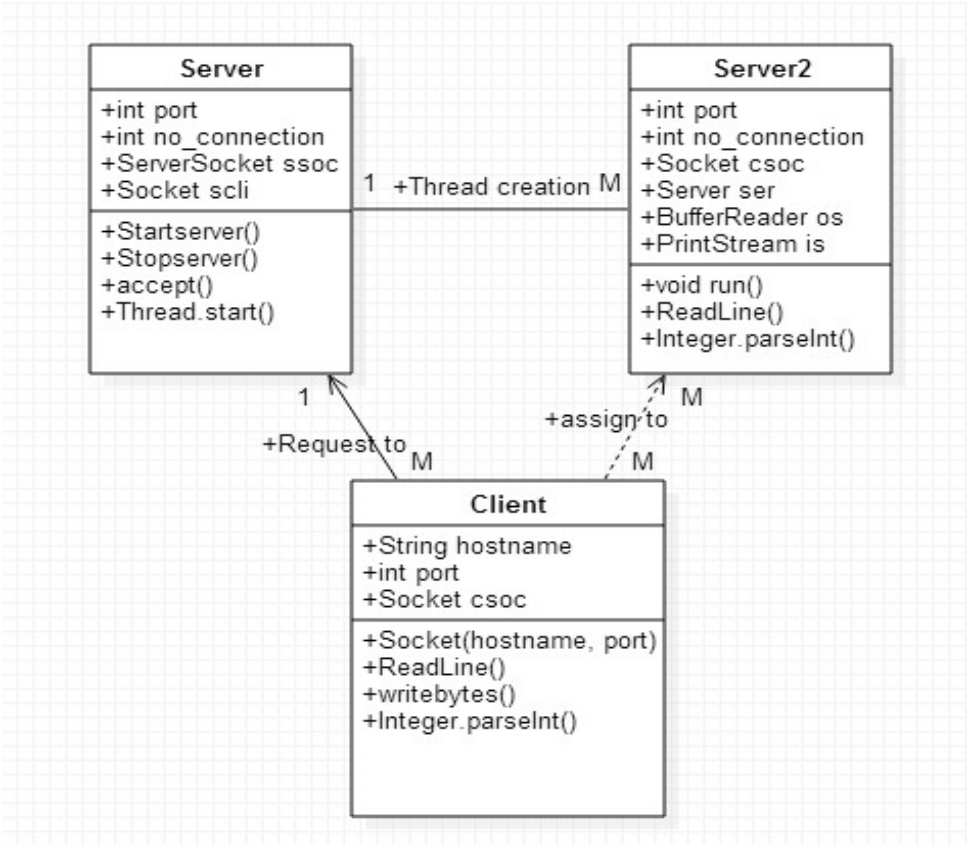
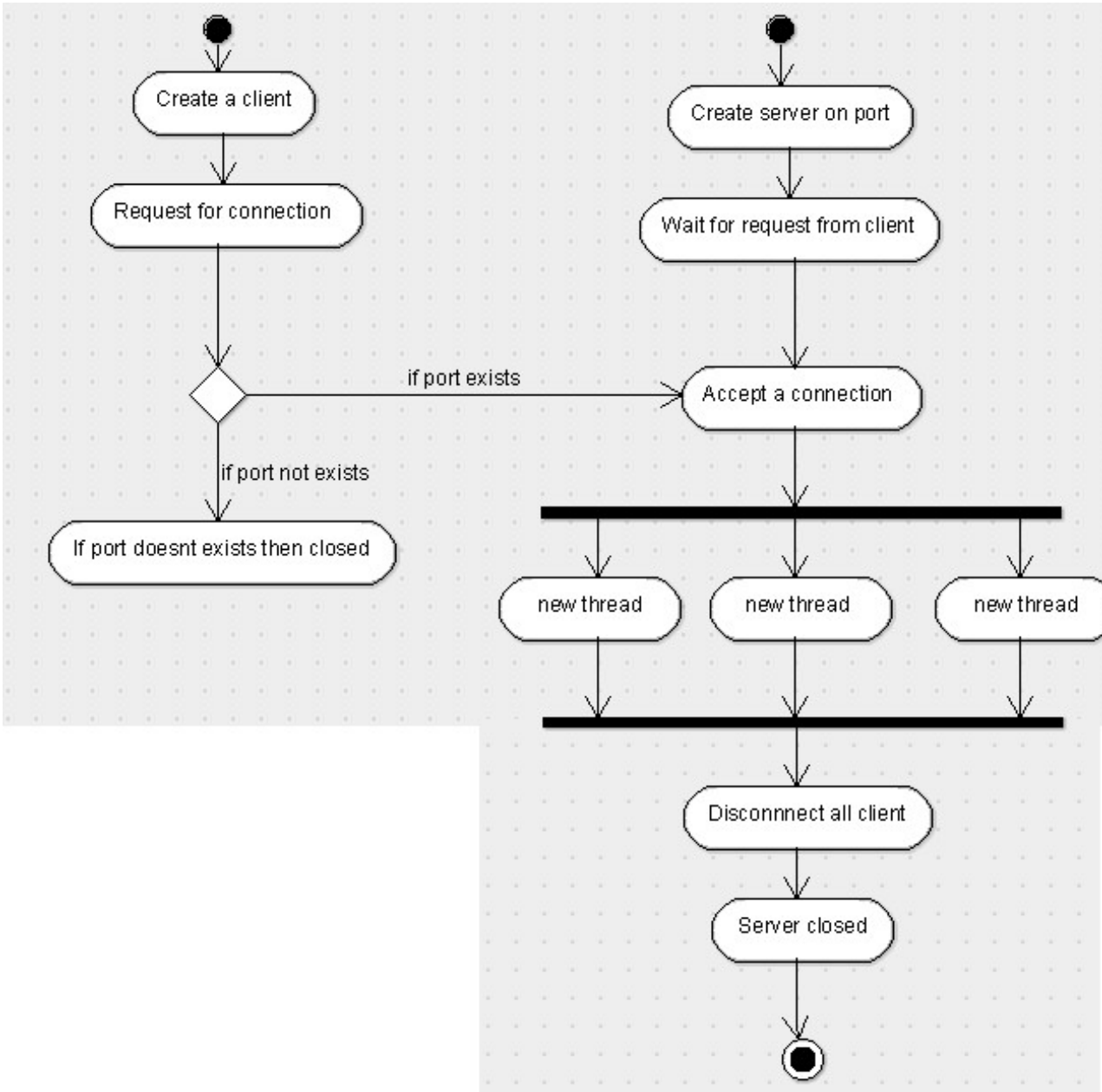


Figure 4: Use Case diagram

– Class Diagram



– Activity Diagram



Sequence Diagram

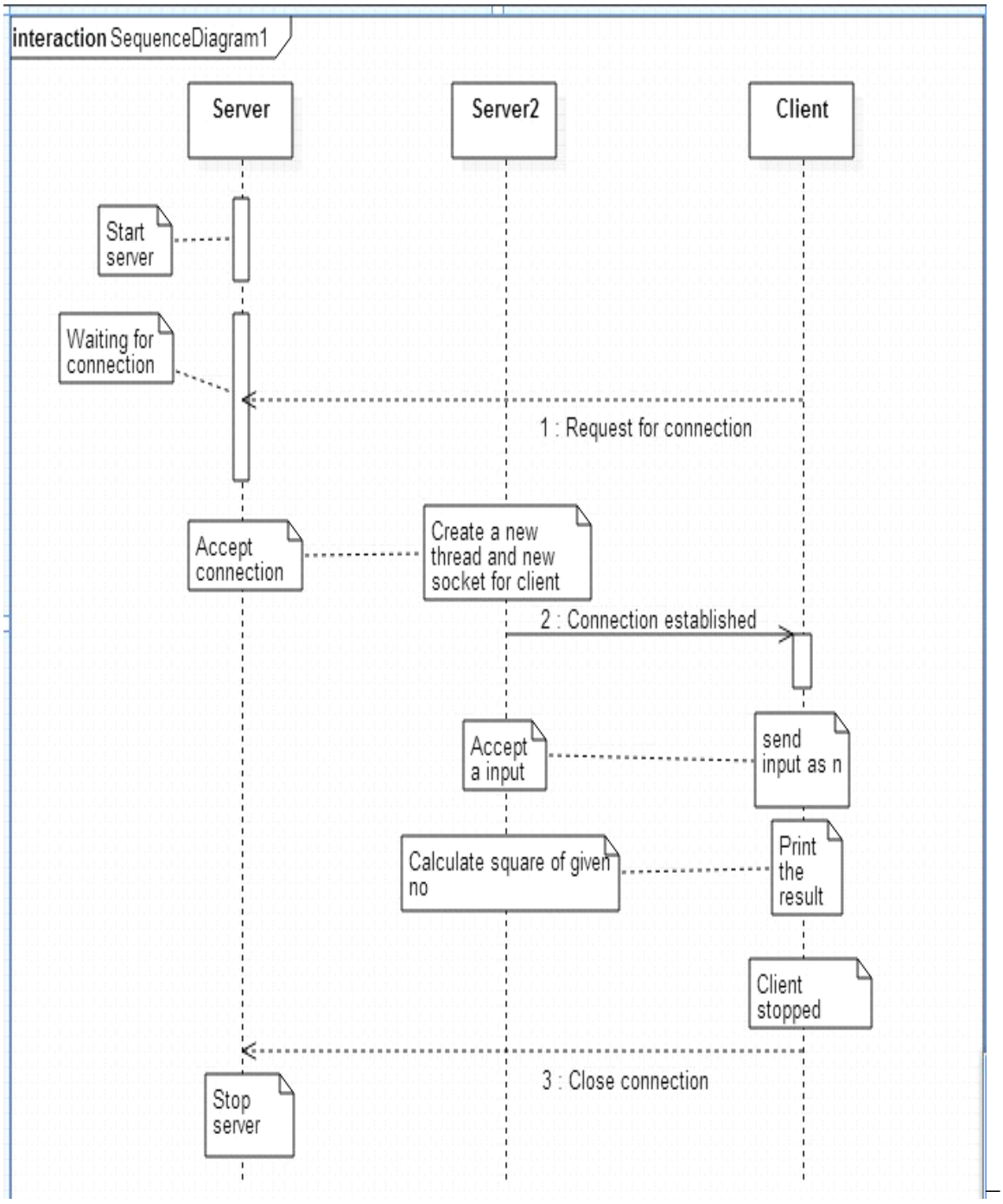


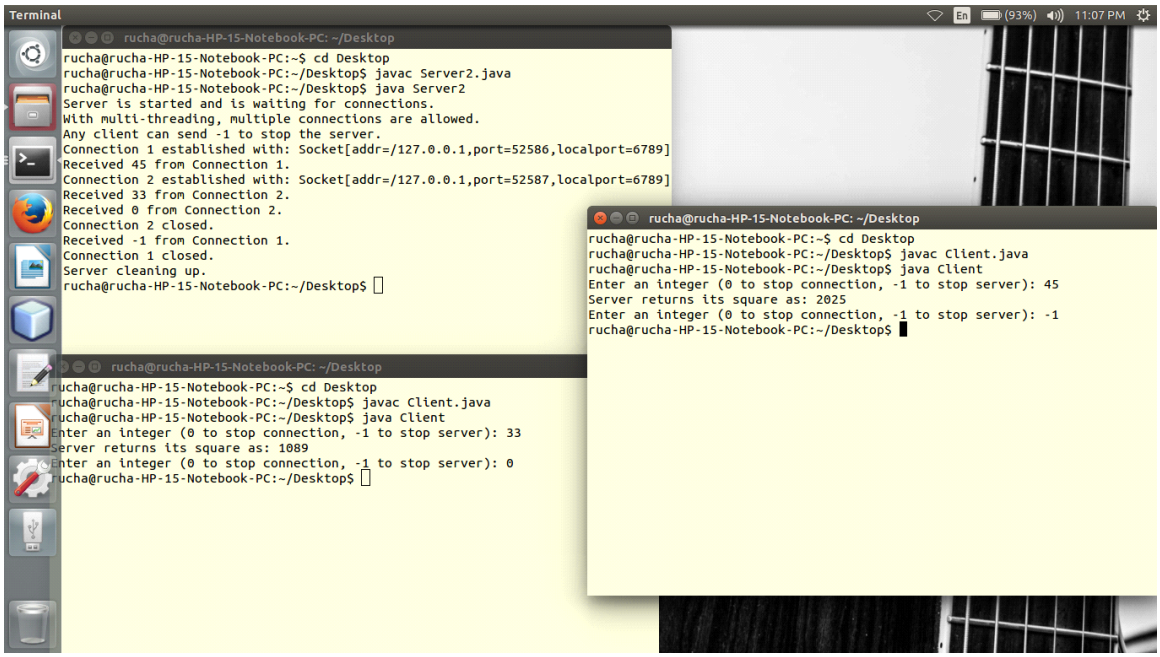
Figure 7: Sequence diagram

- Input

  Input to this program is an integer.

- Output

  Output of this program is square of given integer number.



- Conclusion

  Hence we have studied a Multi-threading application for echo server using socket programming in JAVA.

Course Objective.	Achieved outcome (Tick √)
CO I: Ability to perform multi-core, Concurrent and Distributed Programming.	
CO II: Ability to perform Embedded Operating Systems Programming using Beaglebone.	
CO III: Ability to write Software Engineering Document.	
CO IV: Ability to perform concurrent programming using GPU.	

FAQs

- What is Multithreading.
- Explain Thread Life-cycle.
- Explain the concept of socket.
- Explain multithreading in Java.
- What is runnable() interface.
- What is thread.start() and run().
- What is Echo server.
- Why we use classes PrintStream and BufferedReader.