PROJECT REPORT

TITLE

TCP Multiple Client Server Chat Application in JAVA using Sockets.

Bhagyashree Gawade

CNT 5008 Fall 2016

• Abstract:

In this Programming Project, we study the working of a Multiple Client-Server Chat Application implemented in JAVA. The project focuses on developing a fast connection between multiple clients and a server using a trusted connection oriented protocol like TCP. JAVA TCP server promotes very transparent and simple way to transfer files across network using Java sockets. The server that we are using is a multiclient supported Java Application. The server on one side provides the data only to authorized and authenticated clients and will eventually refuse requests made by other clients, on the other hand Client validates the user request and will improve server's performance. The server is profiled for various clients.

• Introduction:

Open Source file transfer protocols are difficult to use and are also not very easy to implement. The implementation and further management of these protocols is a specialized task in a way. So, this application is a personal multiple client server application that has the ability of supporting different clients.

In this application, we will be using Socket Based Communication.

Socket based communication:

At the transport layer, socket provides an interface. Socket handling is treated like file handling for a reason that help communication. Communication using sockets is very much like file I/O. Socket-based implementation is independent of programming language used for implementing the application. Hence, we are using JAVA programming language. Java carries the benefit of being acquainted to multiple programmers. The design of Java is such that it resembles C++. Java is Object-oriented as well as Network-Oriented. Hence, it focuses only on the data in the application and methods that will manipulate that data and it does not focus much on the procedures. Java has an extension to C++, that is more dynamic method resolution. Network-Oriented Java's net class is a part of the package in java and makes networking very easy in java. The server runs on computer and has a socket which is connected to a specific port.

How a network connection is created??

The network connection in this Chat Application will be initiated by client program once it creates a socket for communication with the server. In order to create the socket in JAVA, the client will call the Socket object with the client address (local host) and port number (1111) and a ListenThread() instance is created which then helps to connect or disconnect the server. At this stage, the server must be in the start phase on the machine containing the specified addresses and listening for connections on the specified port numbers. The server will also use a distinct port dedicated only to listening for incoming connection requests from multiple clients on the same network. Server Socket is a port associated with server side socket. Whenever a connection request will be sent by a client to this socket, the client and server will be connected to the client. Sockets are means to establish a communication link between machines over the network. The java.net package provides 4 kinds of Sockets:

- java.net.Socket} is a TCP client API, and will typically be used to {@linkplainjava.net.Socket#connect(SocketAddress)connect} to a remote host.
- java.net.ServerSocket} is a TCP server API, and will typically {@linkplain java.net.ServerSocket#accept accept} connections from client sockets.</li>
- java.net.DatagramSocket} is a UDP endpoint API and is used to {@linkplain java.net.DatagramSocket#send send} and {@linkplain java.net.DatagramSocket#receive receive}
- {@linkplain java.net.DatagramPacket datagram packets}.
- {@link java.net.MulticastSocket} is a subclass of {@code DatagramSocket} used when dealing with multicast groups.</li>A large number of classes in java.net package do provide for much higher level of abstraction and allow for easy access to resources on the network.

<u>The following steps are followed by the client and server to build a connection</u>:

1. The Server is started on a computer system, it will initialize itself and wait for the client to send a request for communication.
2. The client which will start on another system or the same system send a service request to the serve and gets connected to the server's system over a network.
3. Once the server processes the clients request, it waits for the next client request or waits for any message that the same client will send.
4. The server will get a new socket connected to a same port number. The server requires a completely new socket so that it can continue to listen to the original socket for requests while it is serving the requests of other clients in the network.
5. If multiple clients on different computers are connected to the same server. The clients can chat without any interruption.

• **<u>Related Work/ Background</u>**

Implementation and working with a Chat Application is one of the most popular network programming project. Great highly scalable and configurable applications are available as proprietary as well as an open source software. There are few examples of open chat applications which are implemented using JAVA:

1. FreeCS
2. ChipChat
3. Chat Everywhere
4. GujChat
5. Claros Chat

6. Java MSN messenger Library

With the help of these existing chat applications available online and with the little understanding of Networking protocols, I have tried to implement a chat application not like the ones executed but I have tried implementing a new version on my own.

Programmers usually turn to RMI (Remote Method Invocation) of Sun Microsystems or CORBA (Common Object Request Broker Architecture) of OMG (Object Management Group), for distributed systems. Java programmers can write Object-Oriented programs with the help of RMI, in which objects from various different computers can interact with one another where a client can call any remote object located on the server and the server can also be a client for any other remote object. The implementation of CORBA as well as RMI is very complex as well as expensive for small scale applications. Thus, Java Socket programming is a better and easy choice for implementation. Programmers may also find difficulty in using socket programming when achieving functionality with the realm of distributed systems. In case if any client have the data required by other client, it could not serve him, this becomes the major drawback in traditional client/server architecture. But in Socket Programming Multiple clients can chat using a single Server.

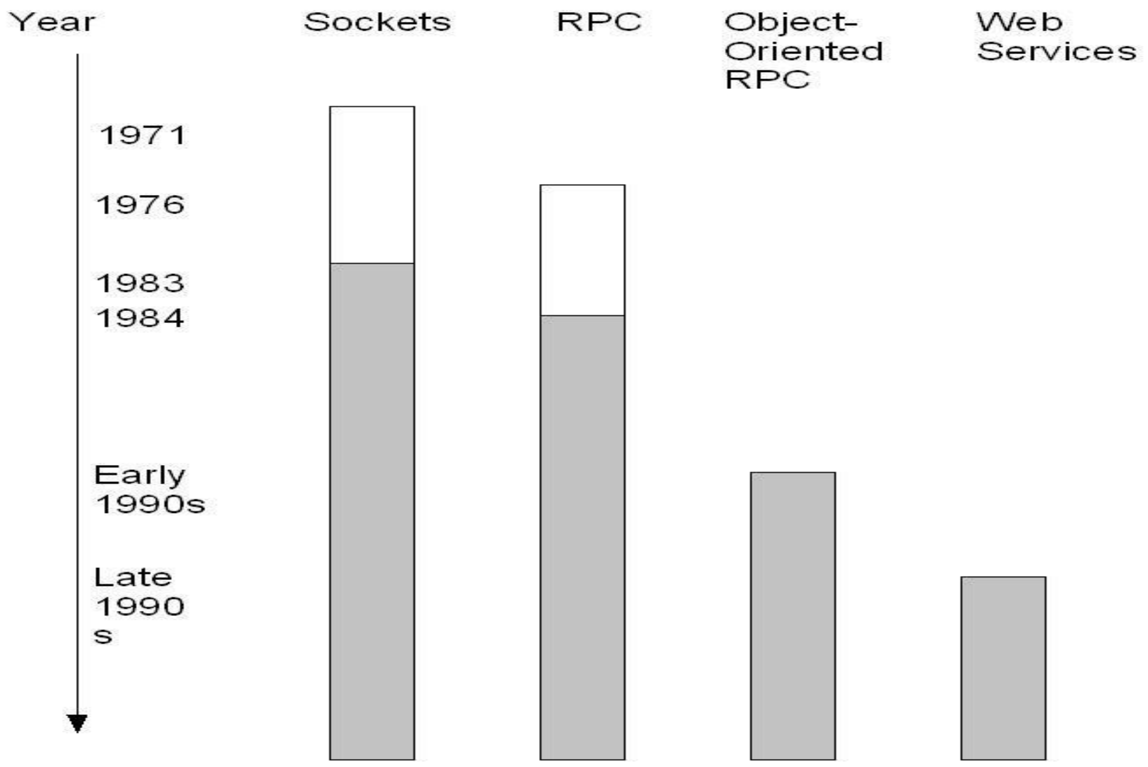


Figure 1.3: A graphical representation when each type of network programming became significant. (Reference: From_sockets_and_RMI_to_web_services)

Architecture of Multiple Client Server Chat Application:

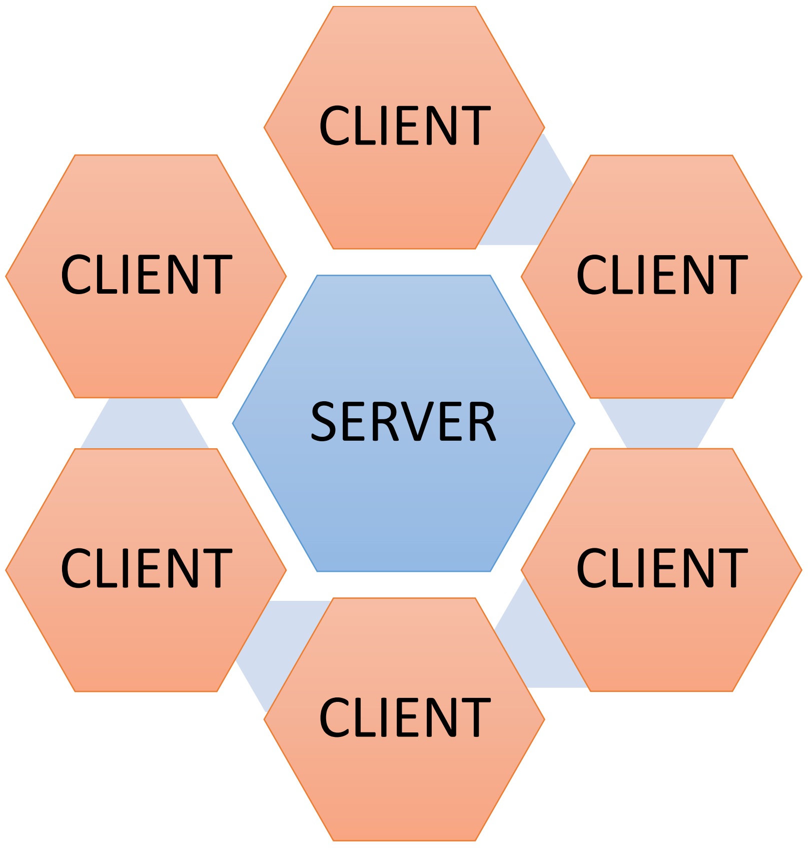


Figure 1.2 Architecture of Multiclient Server Chat Application

E-R diagram of the Chat Application:

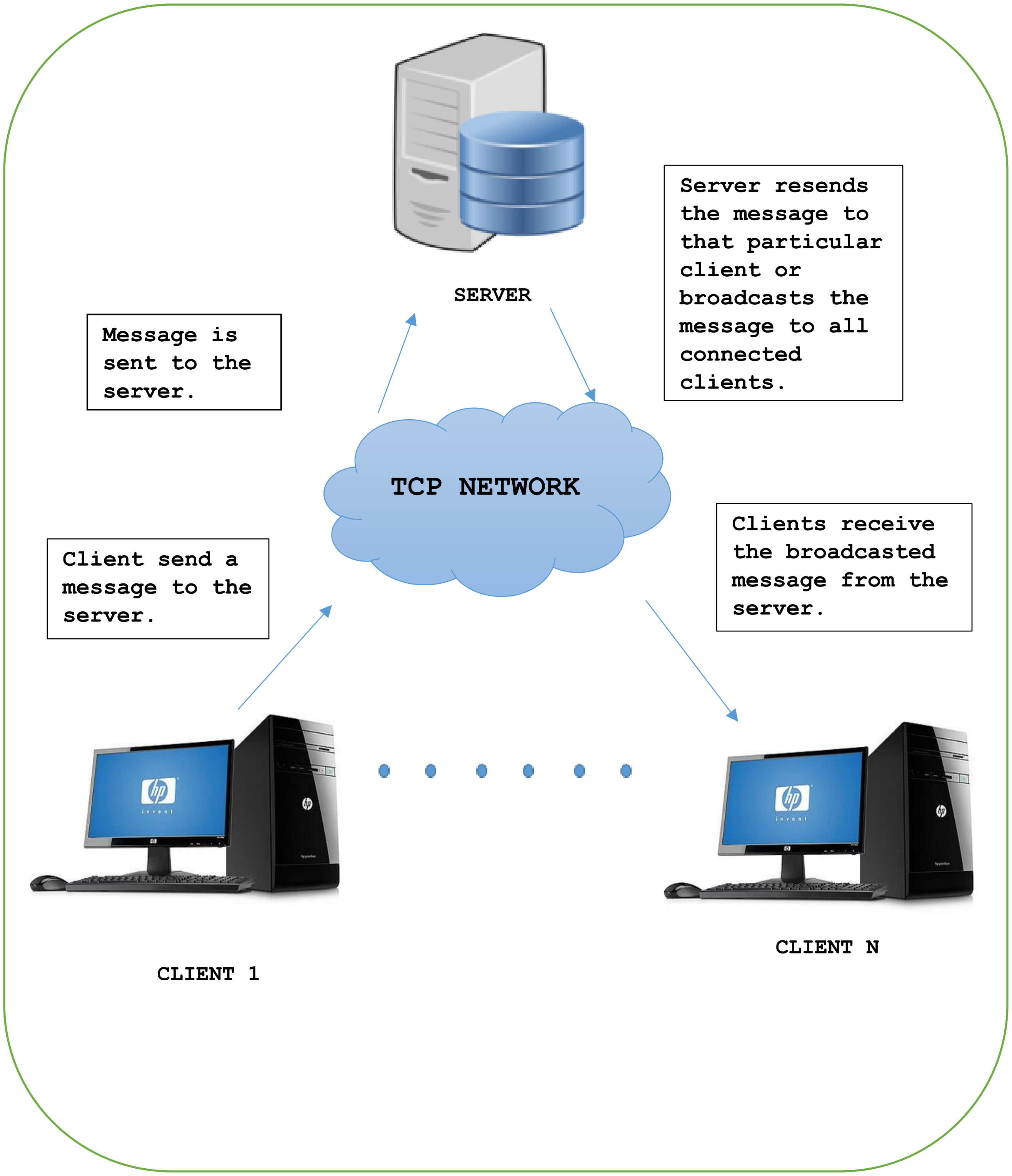
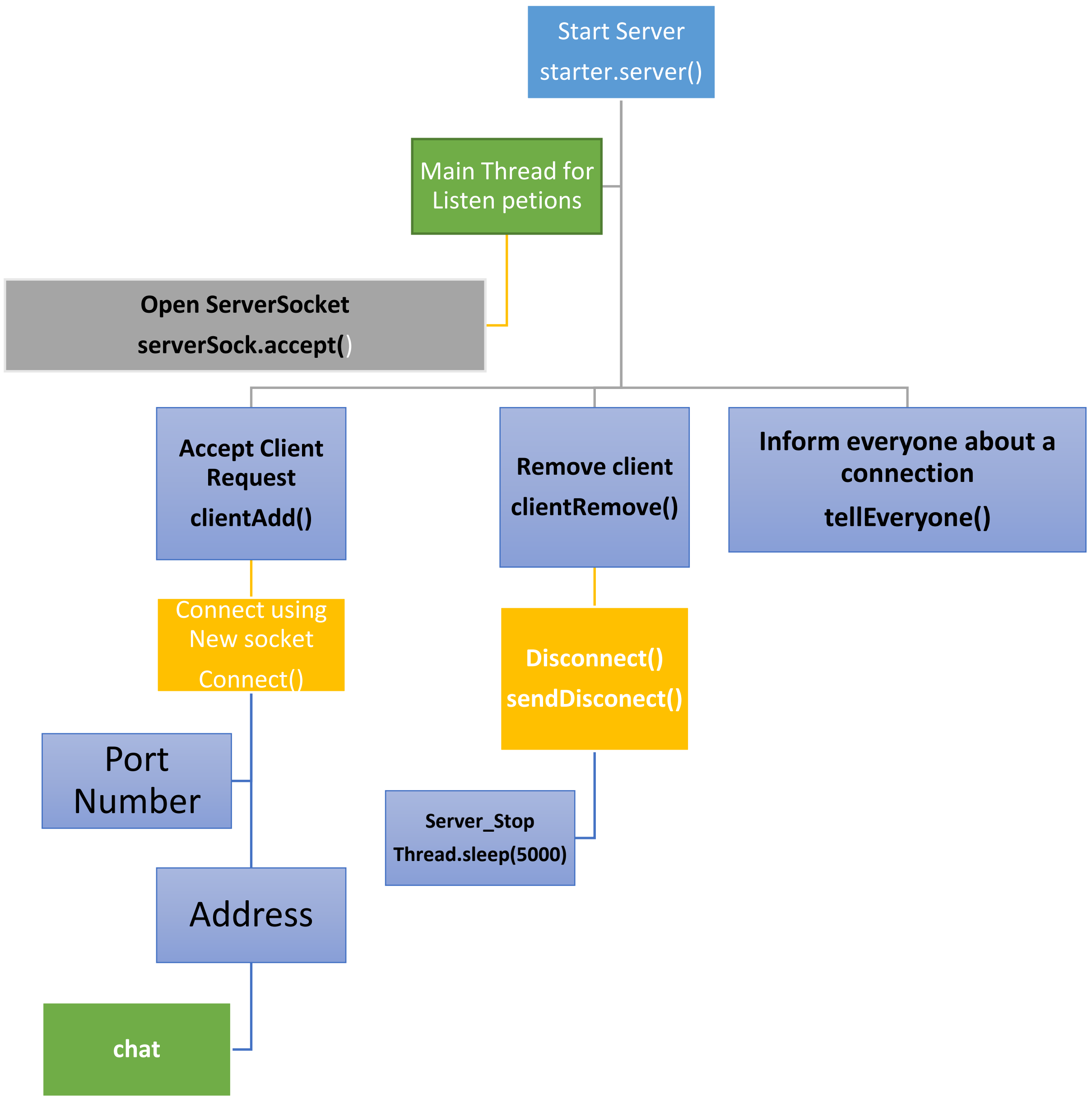

SERVER
Server resends the message to that particular client or broadcasts the message to all connected clients.
Message is sent to the server.
TCP NETWORK
Client send a message to the server.
Clients receive the broadcasted message from the server.
CLIENT 1
CLIENT N

Figure 1.1 – Multiclient Server System Design

Algorithm of JAVA chat application using Sockets:



CODE EXPLANATION:

Client side:

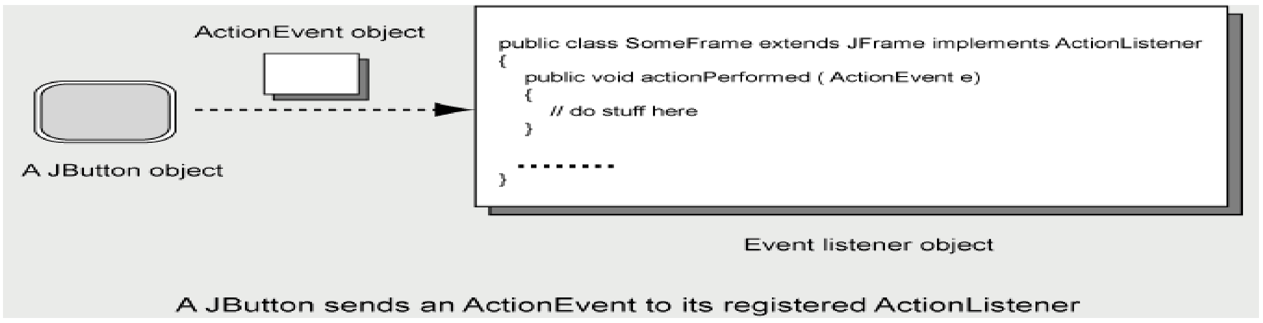
```
import java.net.*;

import java.io.*;

import java.util.*;


public class Frame_clientside extends javax.swing.JFrame {………..}
```

As shown in above code snippet, the class Frame_client extends Frame and we implement Runnable interface to start a Thread which recieves messages from the server. The run() methid will receives the messages sent by the server, the EventQueue object is actually implemented as NUM_PRIORITIES queues and all Events on a particular internal Queue have identical priority. Events are pulled off the EventQueue starting The class Frame_clientside extends frame. We implement Runnable interface which will help us to start a Thread with the Queue of highest priority. We progress in decreasing order across all Queues and the main() method implements the network connection.

```
private void connect_clientActionPerformed(java.awt.event.ActionEvent evt)

{

…….

}
```

As shown in above code snippet, the parameter used for ActionPerformed( ) is a reference to an object ActionEvent, which is subclass of AWTEvent. Whenever a button is clicked, ActionPerformed( ) is called with a new ActionEvent object.

The diagram shows the flow:


ActionEvent object
A JButton object
Event listener object
A JButton sends an ActionEvent to its registered ActionListener

```
public class SomeFrame extends JFrame implements ActionListener
{
   public void actionPerformed ( ActionEvent e)
   {
      // do stuff here
   }
  ........
}
```

Figure 1.3 (Reference: https://chortle.ccsu.edu/java5/Notes/chap59/ch59_2.html **)**

This following code is used for opening the client**:**

```
  try
     {
         S = new Socket(address, port_number);
     }
 catch(Exception e) {
        ta_chat.append("Cannot connect to the server at this moment! Please try again later!! \n");
     }
```

Here,
Address is the client address which is open to connection
Port_number is the port number on which the server which is ready to connect is listening.

```
try
          {
             S = new Socket(address, port_number);
             InputStreamReader streamreader = new InputStreamReader(S.getInputStream());
             reader = new BufferedReader(streamreader);
             writer = new PrintWriter(S.getOutputStream());
             writer.println(anon + "the client is conected to the server: connect");
             writer.flush();
             isConnected = true;
          } catch(Exception e)
          {
```

The above code explains the method Flush(). Flush() method flushes the output stream and makes the output bytes that are buffered to be written out. Calling flush is an indication that, if any of the bytes that were previously written are being buffered by the implementation then those bytes will be written out immediately to their respective destinations.

Server side:

```
public class ClientHandler implements Runnable
      {
             ……
       }
```

A ClientHandler handles communication between a server and client.

The following code is used for starting a server:

```
    public class ServerStart implements Runnable
  {
        try
           {
             ServerSocket serverS = new ServerSocket(1111);
           }
     catch(Exception e)
           {
            System.out.println(e)
           }
}
```

The ServerSocket is used to listen to or accept connections from client.

The following code is used for connecting a client socket to a server:

```
    while (true)
              {
                         Socket clientS = serverS.accept();
                         PrintWriter writer = new PrintWriter(clientS.getOutputStream());
                         clientOutputStreams.add(writer);

                         Thread listener = new Thread(new ClientHandler(clientS, writer));
                         listener.start();
                         ta_chat.append("Got a connection. \n");
              }
```

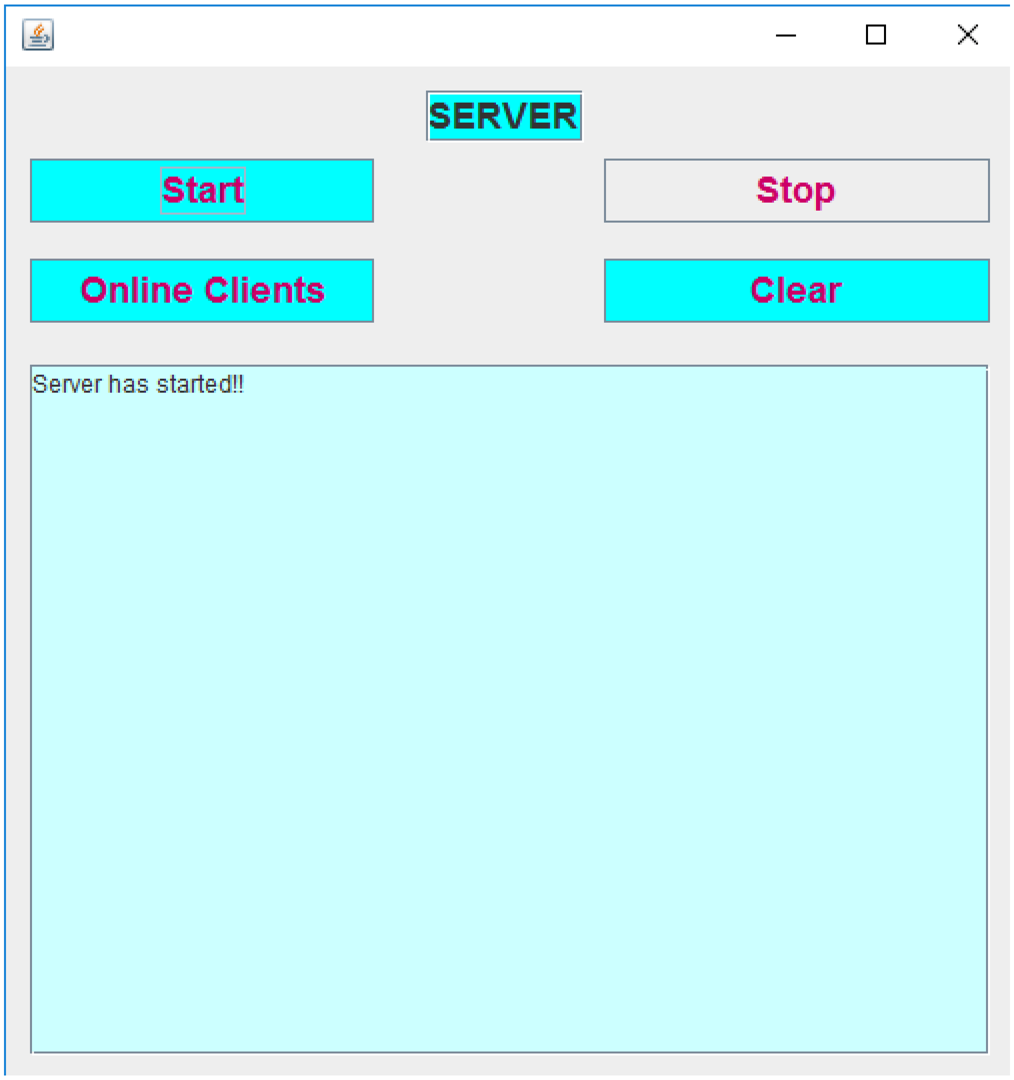
The server can now send or receive data from specific clients.

Sockets are like the descriptors that send/receive operations that are implemented like the read/write operations on the input/output stream.

The following code is used for stopping the server:

```
private void server_stopActionPerformed(java.awt.event.ActionEvent evt) {
       try
       {
          Thread.sleep(5000);    //5000 milliseconds is five second.
       }
       catch(InterruptedException e) { Thread.currentThread().interrupt();}

       tellEveryone("Server: All users will be disconnected because the server is stopping. \n");
       ta_chat.append("Server stopping . . .  \n");
}
```

1. Staring the Server.



Server has started!!

Figure 1.4 Screenshot of Server Screen

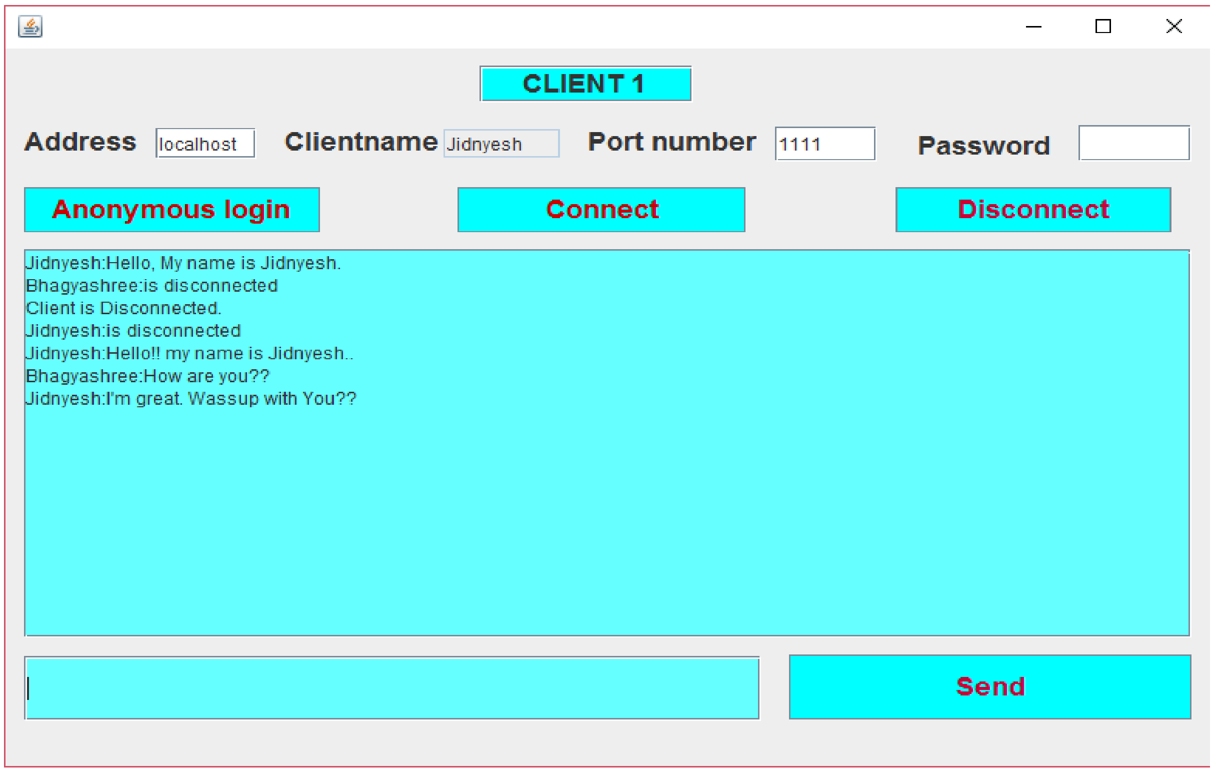
Once the server is started, multiple clients can connect the server using the IP address and port number.

2. **Running the Clients**:

The following Screenshot shows two clients running on local host. If two different clients are running on different computers, then the client must enter the IP address of the computer on which the Server is running. Change the IP address in the snippet shown below to run multiple clients on different computers and chatting with each other.

```
public class Frame_clientside extends javax.swing.JFrame {


    String Clientname, address = "192.168.0.3";
    ArrayList<String> clients = new ArrayList();
    int port_number = 1111;
    Boolean isConnected = false;
```

In the code from client_chat1, change "192.168.0.3" to the address of the computer on which the server is running. This way we can run multiple clients on different computers too.

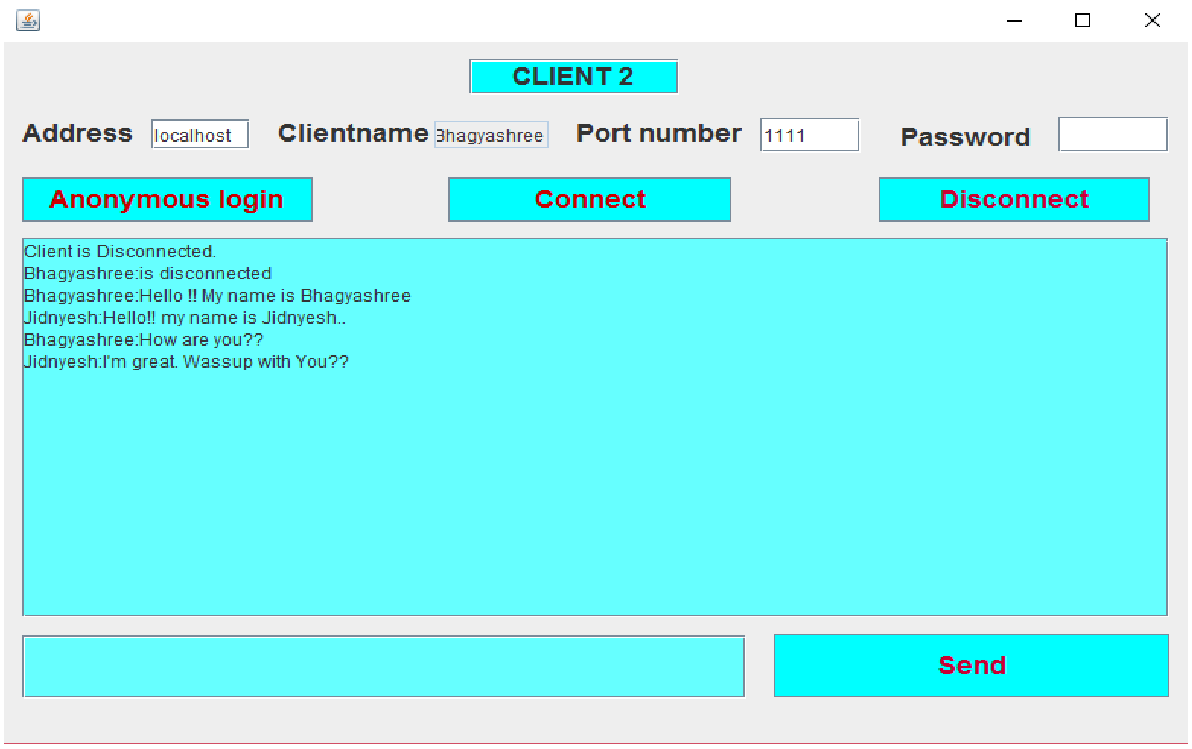


Address: localhost
Clientname: Jidnyesh
Port number: 1111

```
Jidnyesh:Hello, My name is Jidnyesh.
Bhagyashree:is disconnected
Client is Disconnected.
Jidnyesh:is disconnected
Jidnyesh:Hello!! my name is Jidnyesh..
Bhagyashree:How are you??
Jidnyesh:I'm great. Wassup with You??
```

Figure 1.5 Screenshot of Client 1.

Client 1 connected to the server through the IP address.

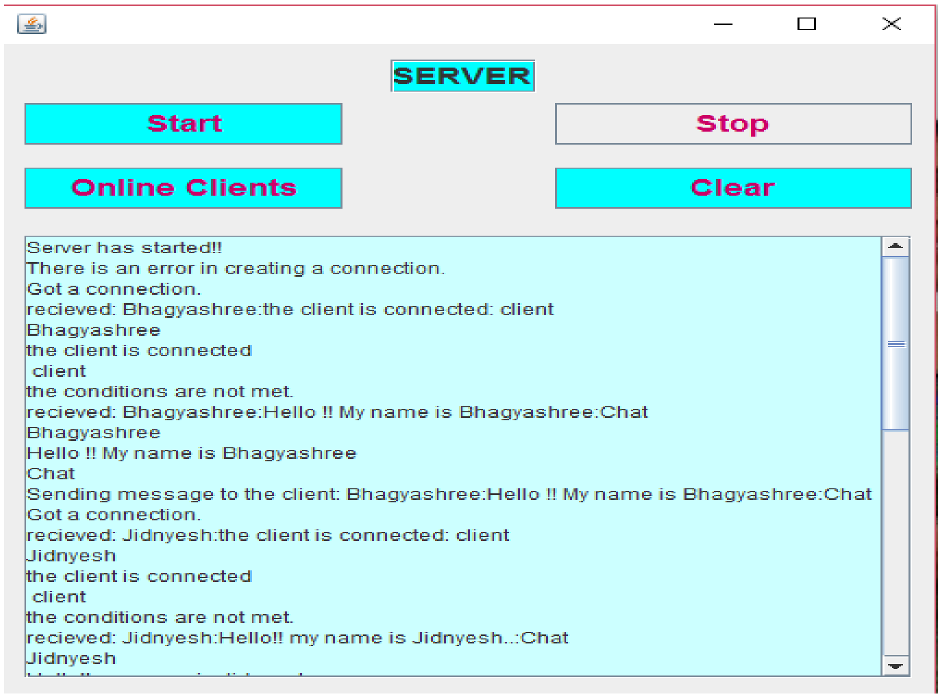


```
Client is Disconnected.
Bhagyashree:is disconnected
Bhagyashree:Hello !! My name is Bhagyashree
Jidnyesh:Hello!! my name is Jidnyesh..
Bhagyashree:How are you??
Jidnyesh:I'm great. Wassup with You??
```

Figure 1.6 Screenshot of Client 2

Client 2 connected to the server.

The following screenshot shows the server interaction with two clients:



```
Server has started!!
There is an error in creating a connection.
Got a connection.
recieved: Bhagyashree:the client is connected: client
Bhagyashree
the client is connected
 client
the conditions are not met.
recieved: Bhagyashree:Hello !! My name is Bhagyashree:Chat
Bhagyashree
Hello !! My name is Bhagyashree
Chat
Sending message to the client: Bhagyashree:Hello !! My name is Bhagyashree:Chat
Got a connection.
recieved: Jidnyesh:the client is connected: client
Jidnyesh
the client is connected
 client
the conditions are not met.
recieved: Jidnyesh:Hello!! my name is Jidnyesh..:Chat
Jidnyesh
```

Figure 1.7 Screenshot of Server interacting with 2 clients.

Simulation Statistics:

I have compared the performance statistics of Java chat application implemented using RMI (Remote method Invocation) which is a version of RPC (Remote procedural call).

The Object-Oriented RPC is used explicitly. On the other hand, Socket programming is used implicitly.

The RPC has a triangular relationship between the server, client and a third party. The publishing and deploying called as service registers with the client and the third party informs the third party to find the respective location of the service.

Architecture of RMI:

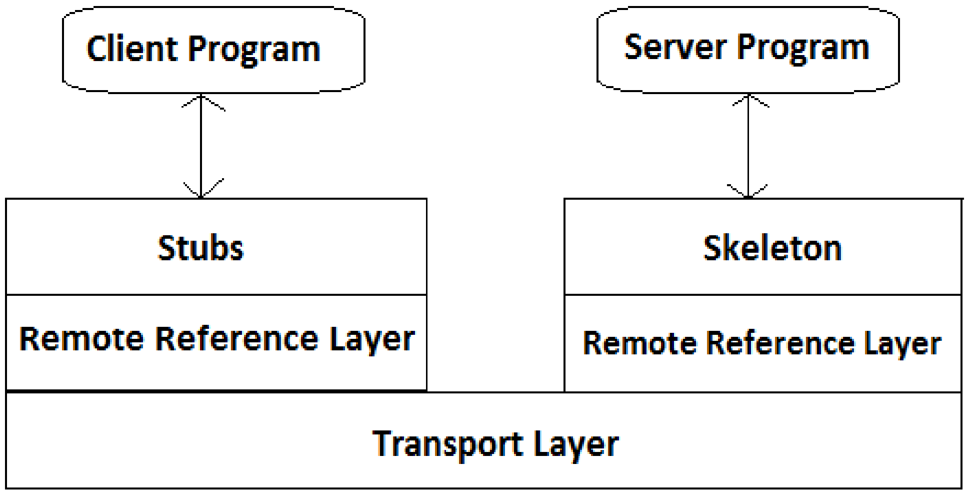


Figure 1.6 Architecture of RMI (Reference: Research paper, Implementation-of-Socket-Programming-and-RMI-Using-Simulating-Environment)

As shown in the figure, in RMI, the server calls the registry for binding with a remote object. Then, the client searches the remote object by its name in the server's registry for invoking a method on it. The diagram also shows that the RMI system uses the existing web server to load the class definition.

Comparison of Socket Programming and RMI:

	Characteristics	Socket Programming	RMI
1.	Platform	Platform Independent	Platform Dependent
2.	Compatibility	Any programming language	JAVA
3.	Abstraction	Low level	High level
4.	Speed	Faster	Comparatively Slower than Socket Programming
5.	Execution overhead	Less	High
6.	Handling Ports	Very easy	Comparatively Difficult
7.	Security	Highly Secure	Moderately Secure
8.	Utilization	For passing messages, values and transferring data between Client and Server.	For invocation for methods, used for communication between JAVA programs.
9.	Stream Optimization	Yes	No

Graphical Statistics of JAVA chat app using RMI:

The project is run using NetBeans and profiled to calculate the Heap usage and Garbage collection as well Thread usage of the client and server effectively.

Heap Used:

Java objects are placed in an area which is called as heap. Heap is created as soon as the JVM starts and it keeps increasing or decreasing depending on the running of the application. When the Heap is totally used up, the garbage is collected.

Used Memory = Working set + Garbage.

Garbage Collection:

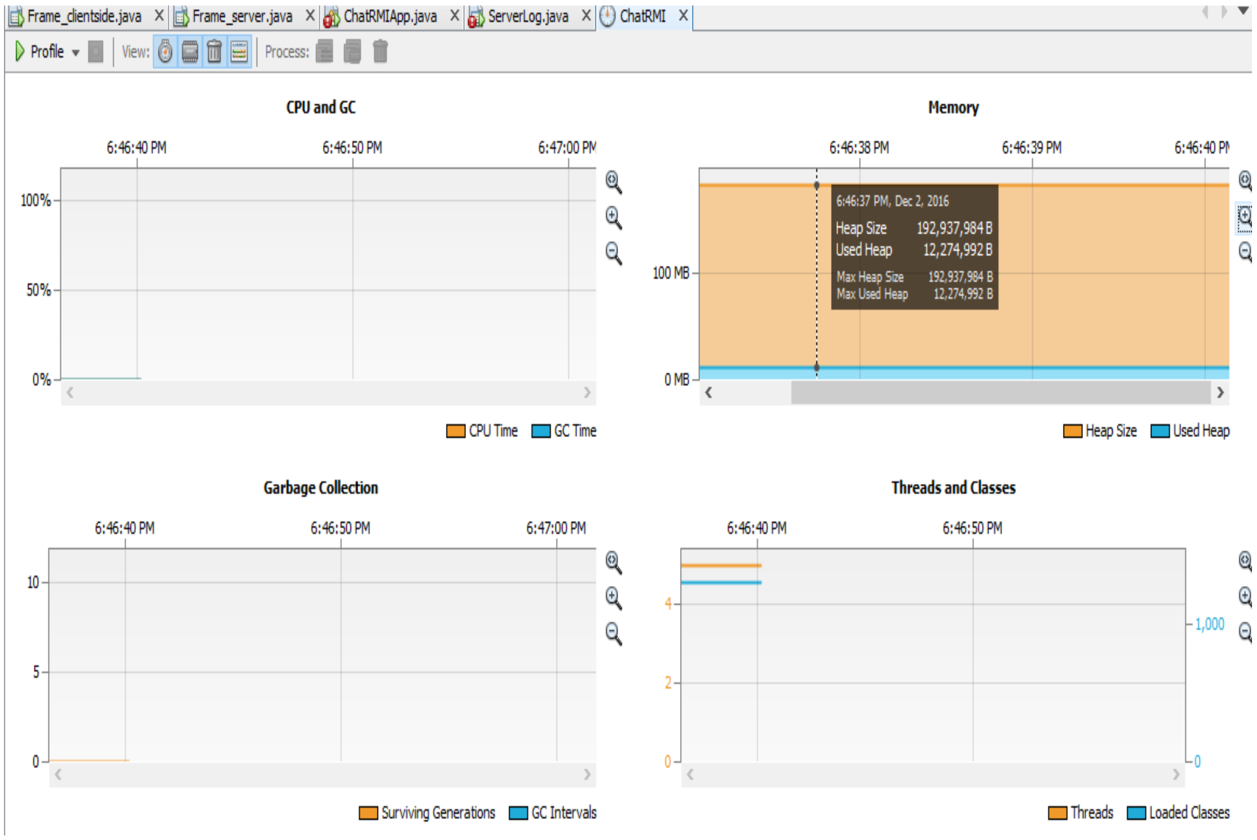
During the process of garbage collection, the objects that are not being used by the application currently are removed or cleared to free the space.

Threads and classes:

The Java Application contains threads that run JAVA code. The entire JVM process contains some internal threads and all JAVA threads, for example a code optimizer thread, one or any more finalizer threads and one or more garbage collection threads.

The following graph shows these parameters when a JAVA chat application is run using RMI.

1. **RMI uses Heap of around 12,274,992.**



6:46:37 PM, Dec 2, 2016
Heap Size 192,937,984 B
Used Heap 12,274,992 B
Max Heap Size 192,937,984 B
Max Used Heap 12,274,992 B

Figure 1.8 Graph of RMI heap usage

The Heap graph shows the total heap size, max heap size and how much heap is currently used.

2. **Threads and Loaded Classes.**

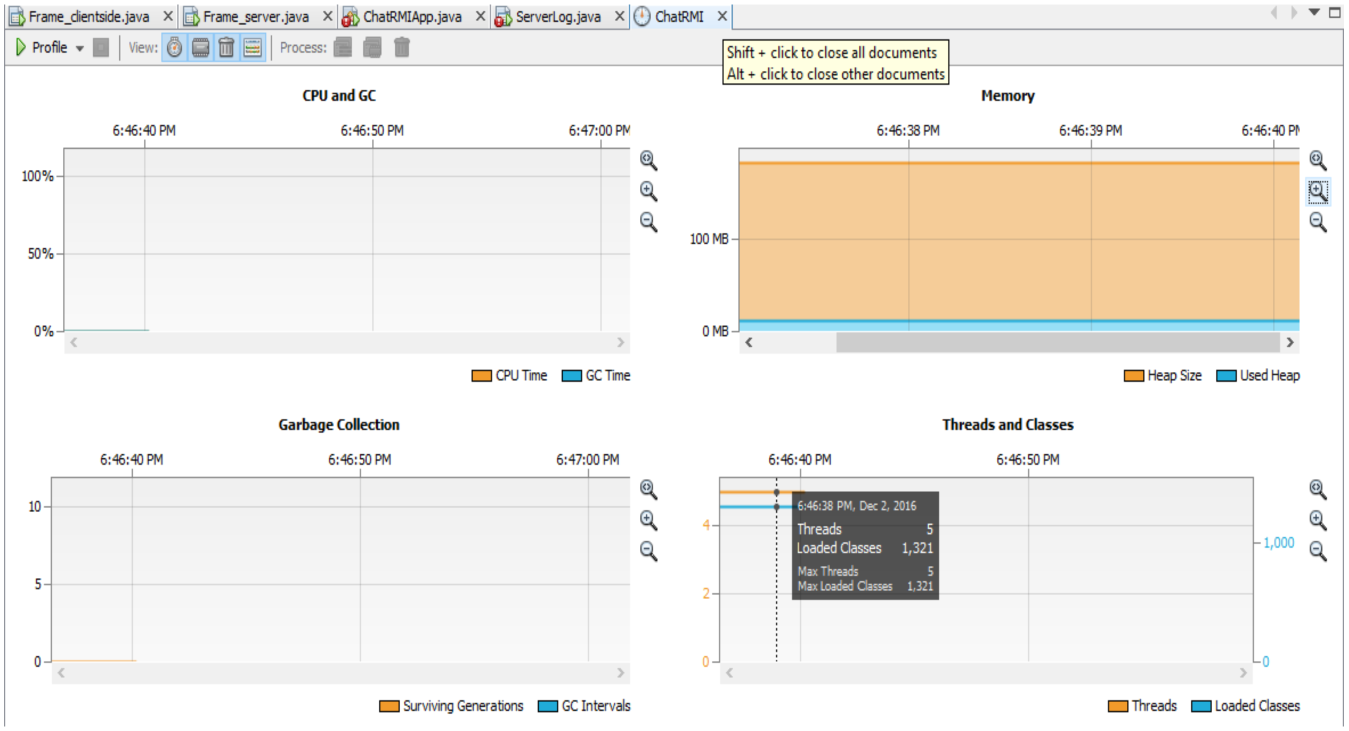


Figure 1.9 Graph of Threads and Loaded Classes.

The Threads and Loaded classes graph show the number of live and daemon threads in the application using RMI.

Now, I have compered these statistics with the java chat application using Socket Programming. It is faster and consumes less memory as well as uses more threads.

1. **Heap Usage.**

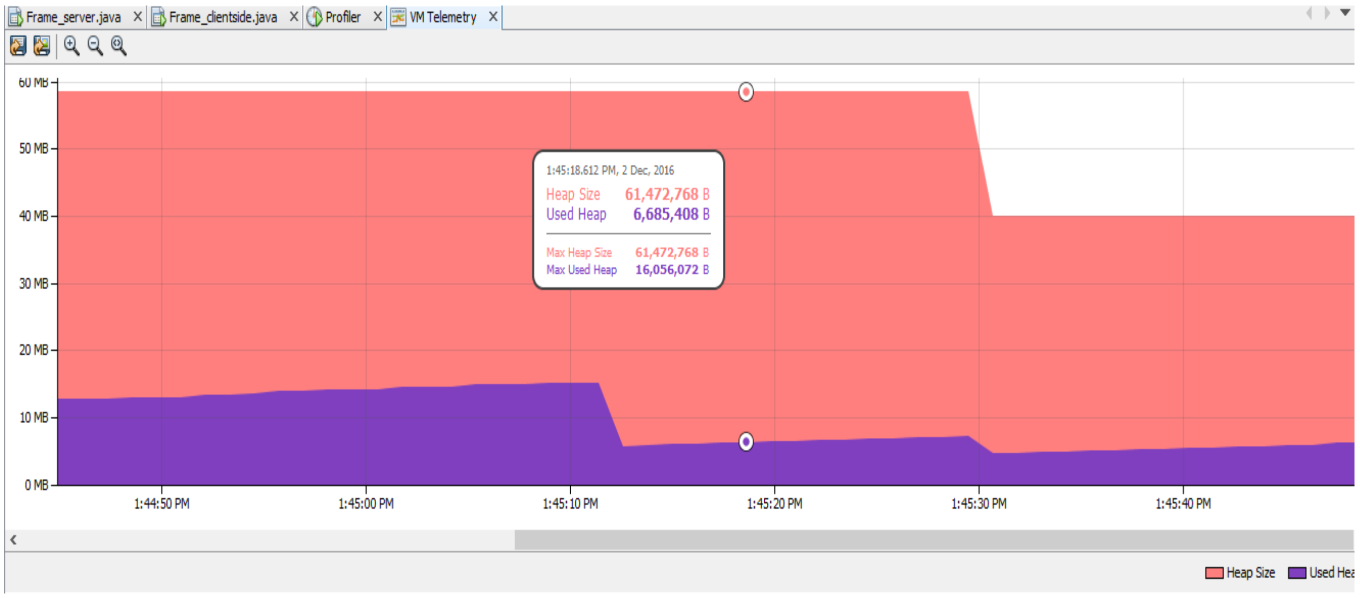


Figure 1.10 Graph displaying Heap used by Socket Programming.

The **Max Used Heap** is 16,056,072 B and Used Heap is 6,685,408 B. This value is almost half of the Heap used by running Java chat application using RMI.

The Heap memory usage and Garbage collection should be as less as possible.

2. Threads used:

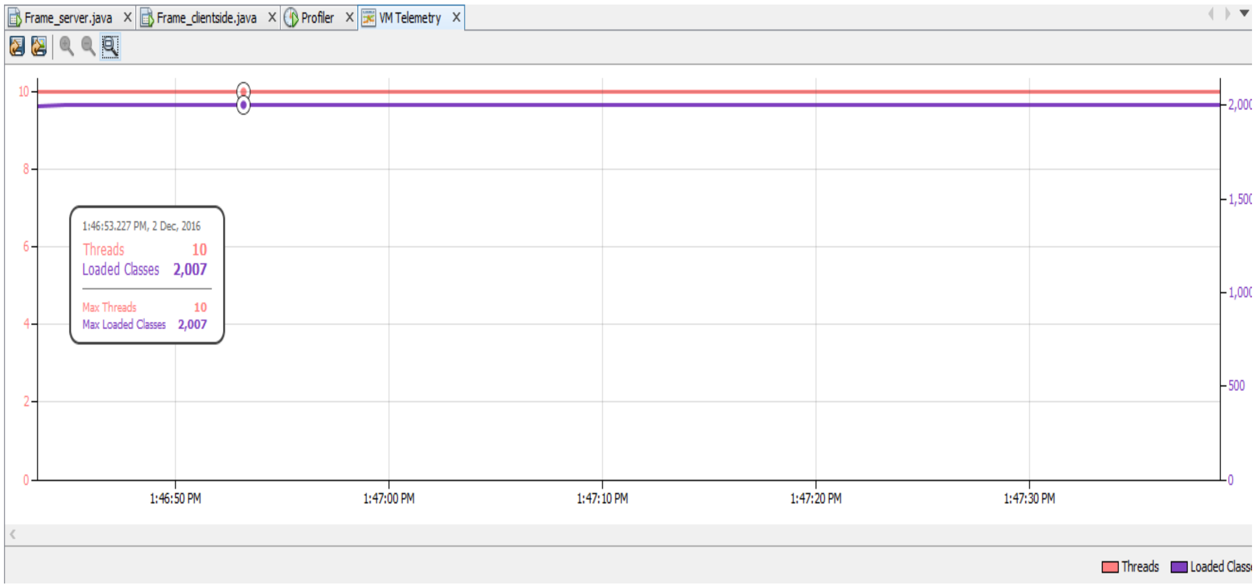


Figure 1.11 Graph displaying Threads and Loaded Classes.

3. Garbage Collection and CPU time:

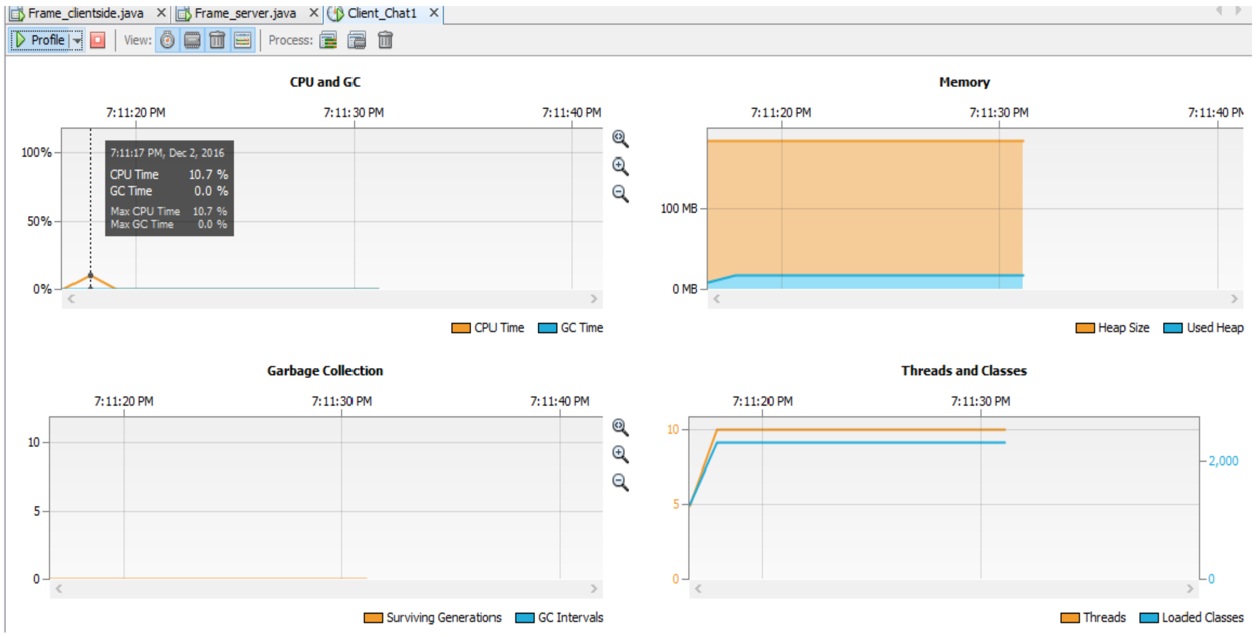
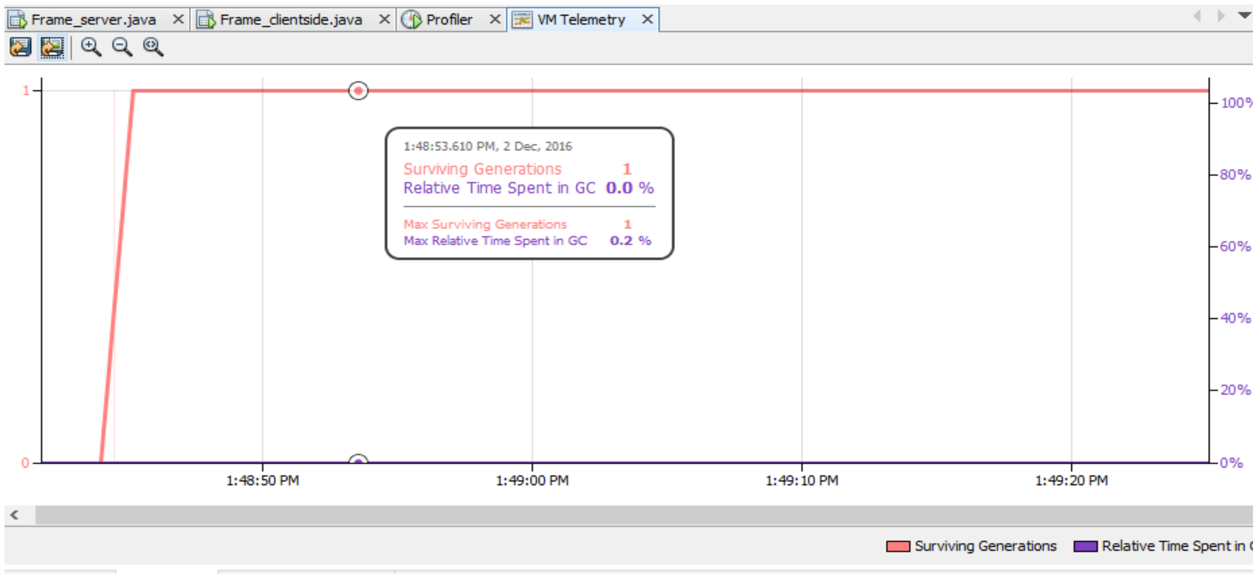


Figure 1.12 Graph displaying Garbage Collection and CPU time.

4. Surviving Generation:



1:48:53.610 PM, 2 Dec, 2016
Surviving Generations 1
Relative Time Spent in GC 0.0 %
Max Surviving Generations 1
Max Relative Time Spent in GC 0.2 %

Figure 1.12 Graph displaying Surviving Generation

The number of Surviving generations increase whenever there is a memory leak. Surviving Generations value will always be the number of different Surviving Generations that are alive on the heap. Memory leak is a memory loss caused due to consumption of memory by a JAVA program where the program does not release memory when it is not needed.

In this project, I have tried to investigate in to Remote Method Invocation and A Socket Programming. The advantages of Socket Programming over Remote Method Invocation are stated. Though Socket Programming has many advantages over RMI, RMI cannot be ignore completely. It is also very important but for very few distinct activities.