****

**RAMNIRANJAN JHUNJHUNWALA COLLEGE**

**GHATKOPAR (W), MUMBAI - 400 086**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**2020 - 2021**

**M.Sc.( I.T.) SEM I**

**Distributed System**

**Name: SHARMA SIMRAN**

**Roll No.: 14**



**CERTIFICATE**

This is to certify that Mr. / Miss/Mrs. **Sharma Simran** with Roll No.**14** has successfully completed the necessary course of experiments in the subject of **DISTRIBUTED SYSTEM** during the academic year **2020 – 2021** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -I.

Internal Examiner Date:

Head of Department College Seal External Examiner

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Prac no.** | **Aim** | **Date** |
| 1 | Write a program for implementing Client Server communication model | 26/11/2020 |
| 2 | Write a program to show the object communication using RMI | 05/02/2021 |
| 3 | Show the implementation of Remote Procedure Call. | 11/12/2020 |
| 4 | Show the implementation of web services. | 12/02/2021 |
| 5 | Write a program to execute any one mutual exclusion algorithm | 03/01/2021 |
| 6 | Write a program to implement any one election algorithm. | 10/01/2021 |
| 7 | Show the implementation of any one clock synchronization  algorithm | 10/01/2021 |
| 8 | Write a program to implement two phase commit protocol | 18/02/2021 |

**Practical No: 01**

**Aim:** Write a program for implementing a Client Server communication model.

**Client Server communication model:**

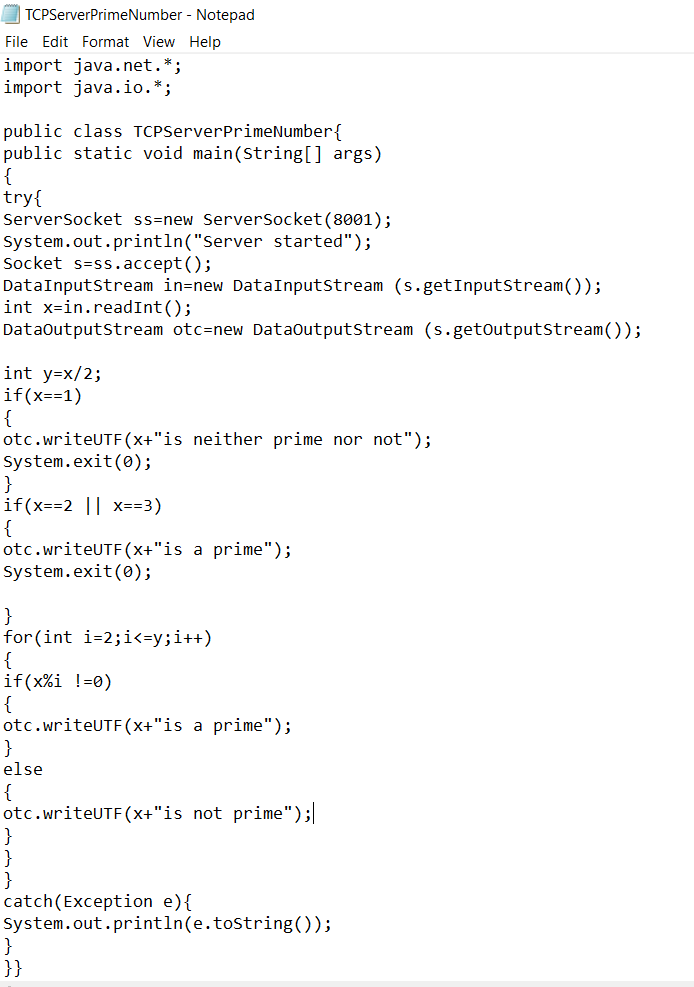
Client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client does not share any of its resources, but it requests content or service from a server. Clients therefore initiate communication sessions with servers, which await incoming requests. Examples of computer applications that use the client–server model are Email, network printing, and the World Wide Web.

 The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services. Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.

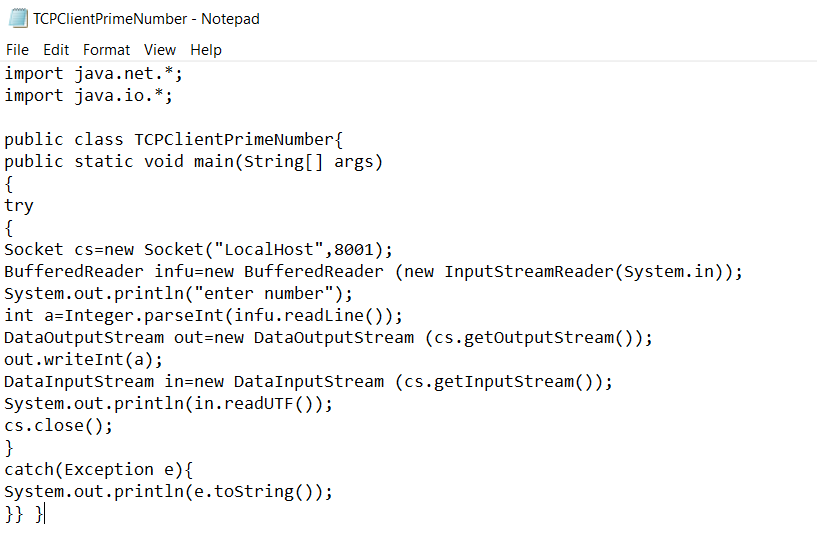
**Example 1A: A client server based program using TCP to find if the number entered is prime.**

**CODE:**

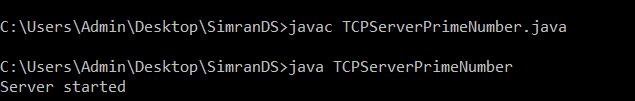
**TCPServerPrimeNumber.java**

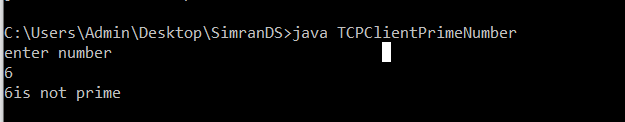


**TCPClientPrimeNumber.java**



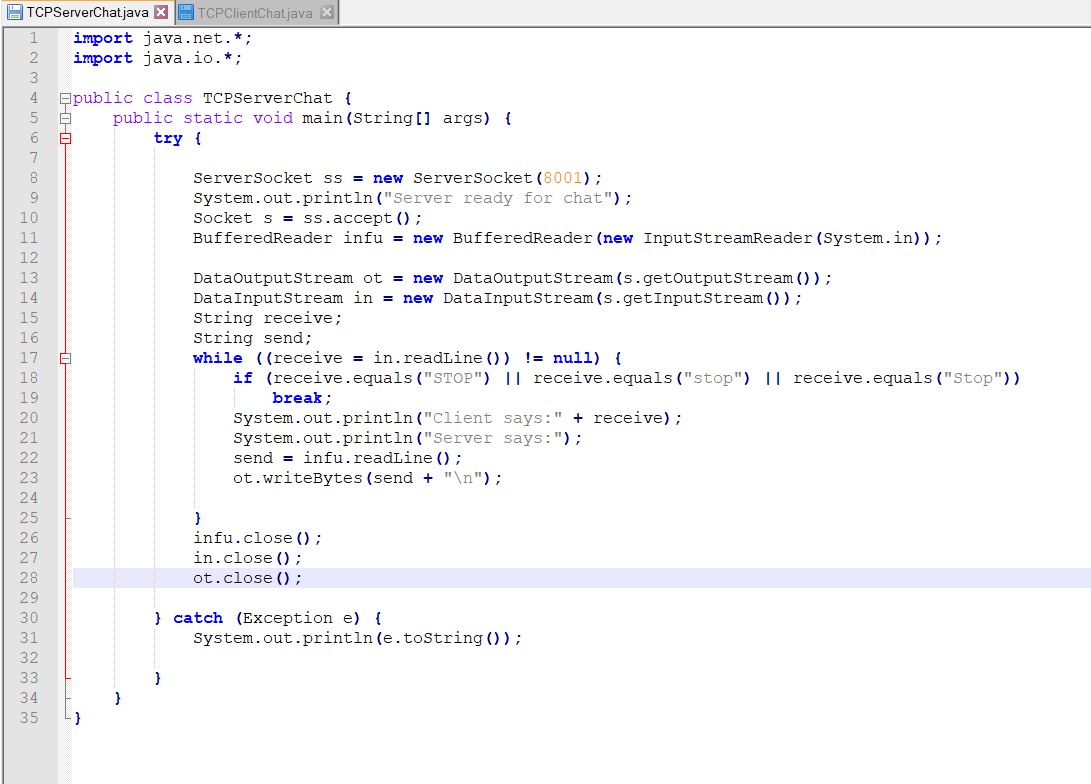
**OUTPUT:**





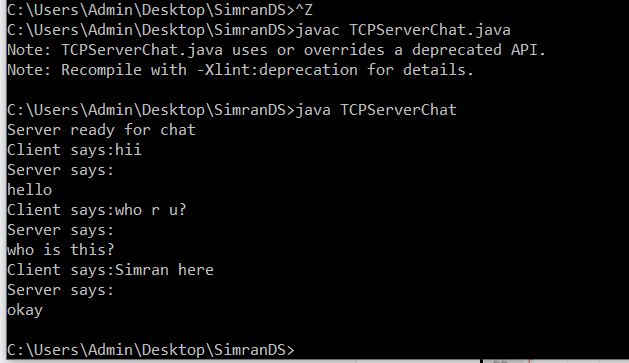
**Example 1B: A client server TCP based chatting application.**

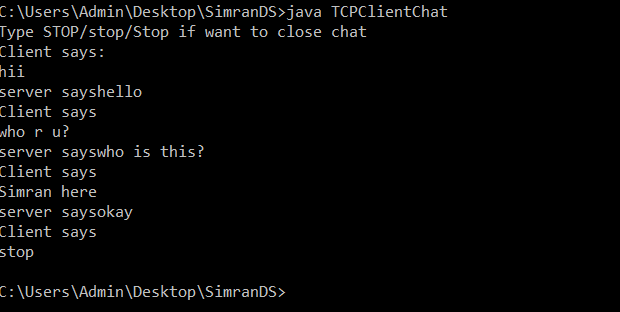
**TCPServerChat.java**

**TCPClientChat.java**



**OUTPUT:**





**Practical No: 02**

**Aim:** Write a program to show the object communication using RM**I.**

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

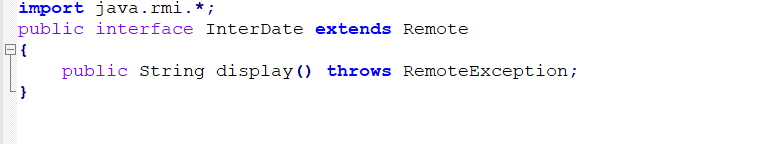
1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for skeletons.

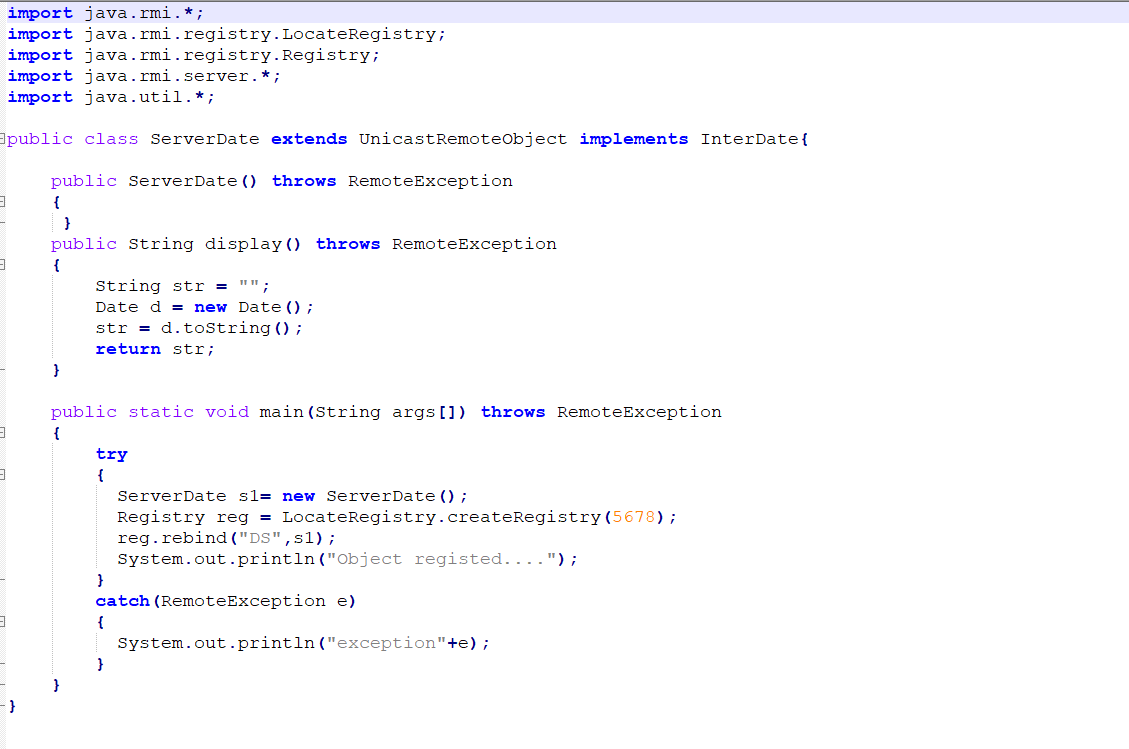
**Example 2A: A RMI based application program to display current date and time.**

**CODE:**

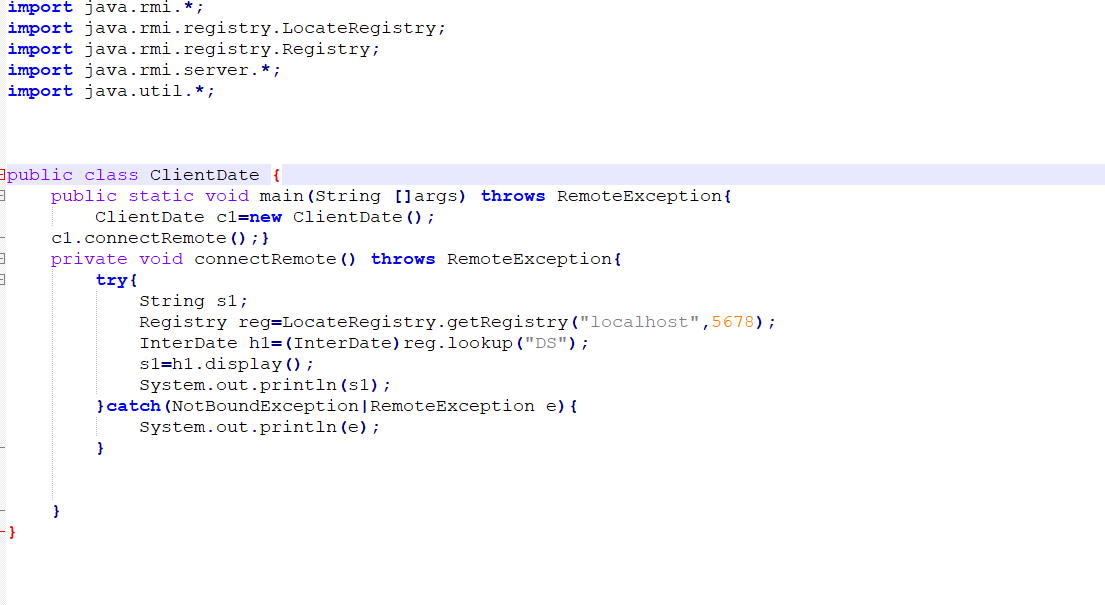
**InterDate.java**



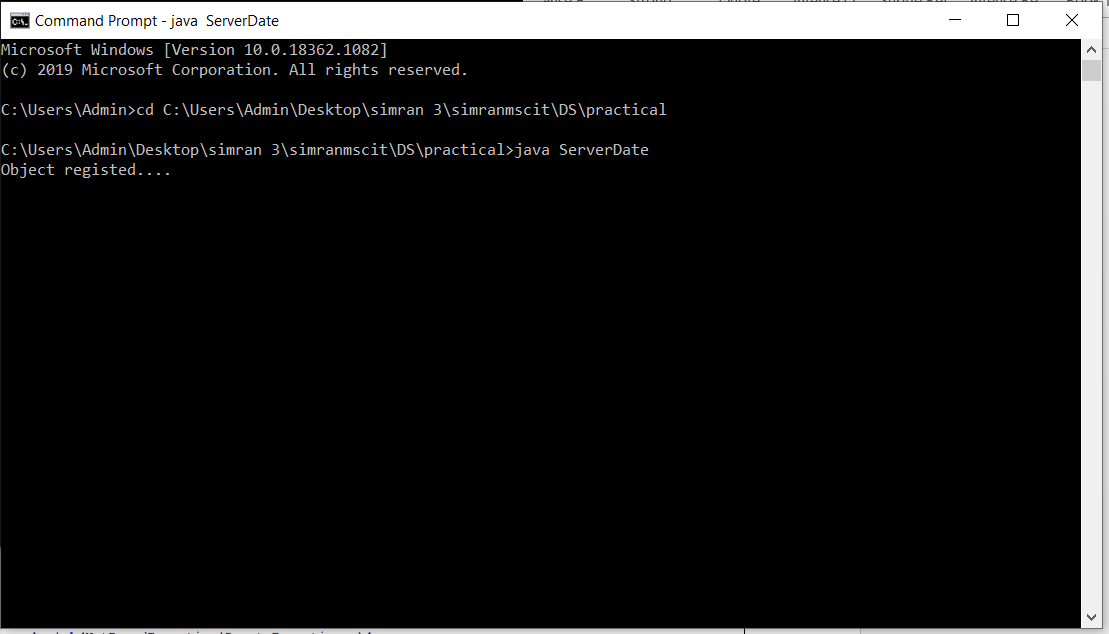
**ServerDate.java**

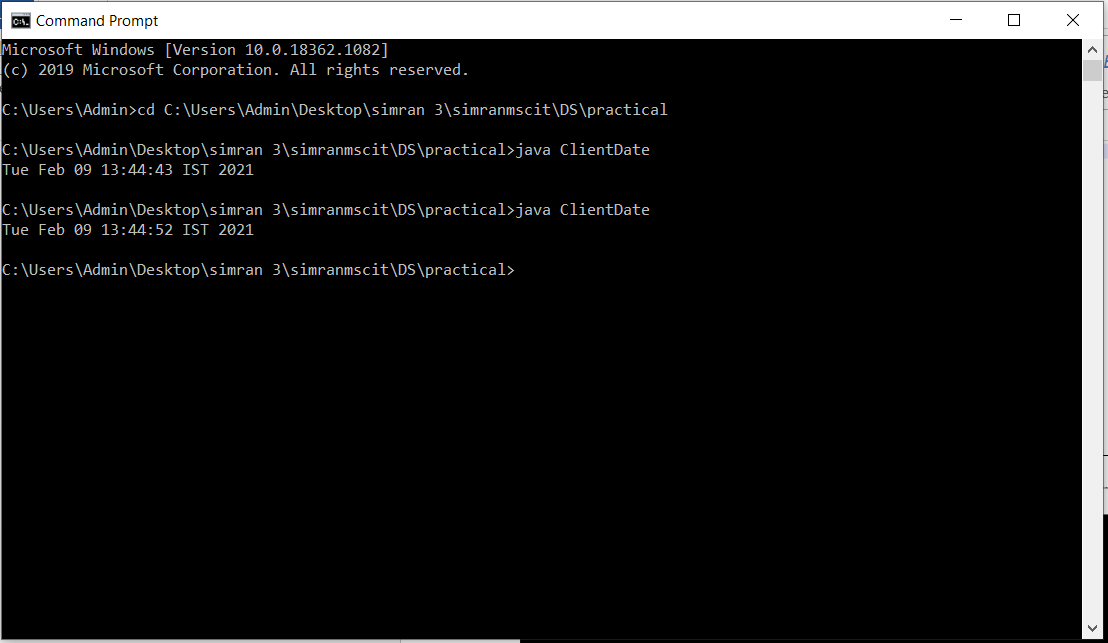


**ClientDate.java**



**OUTPUT:**

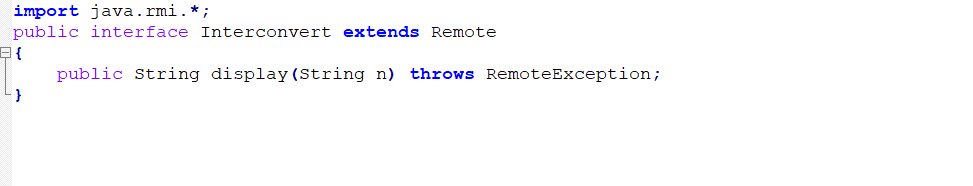




**Example 2B: A RMI based application program that converts digits to words, e.g. 123 will be converted to one two three**

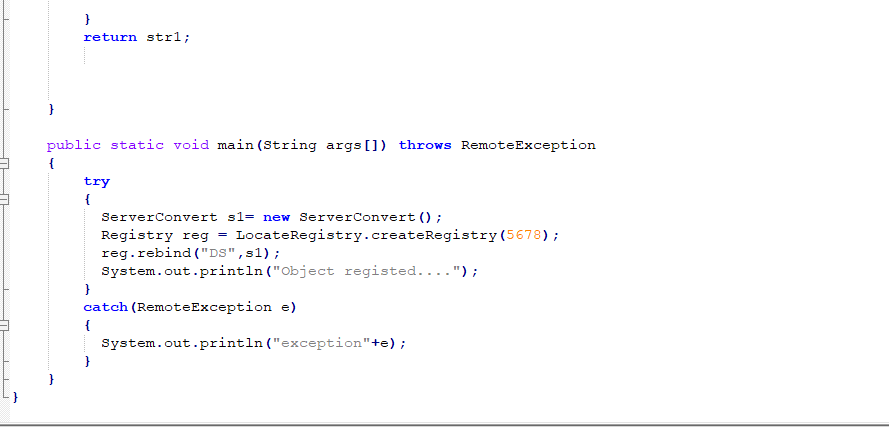
CODE:

**Interconvert.java**

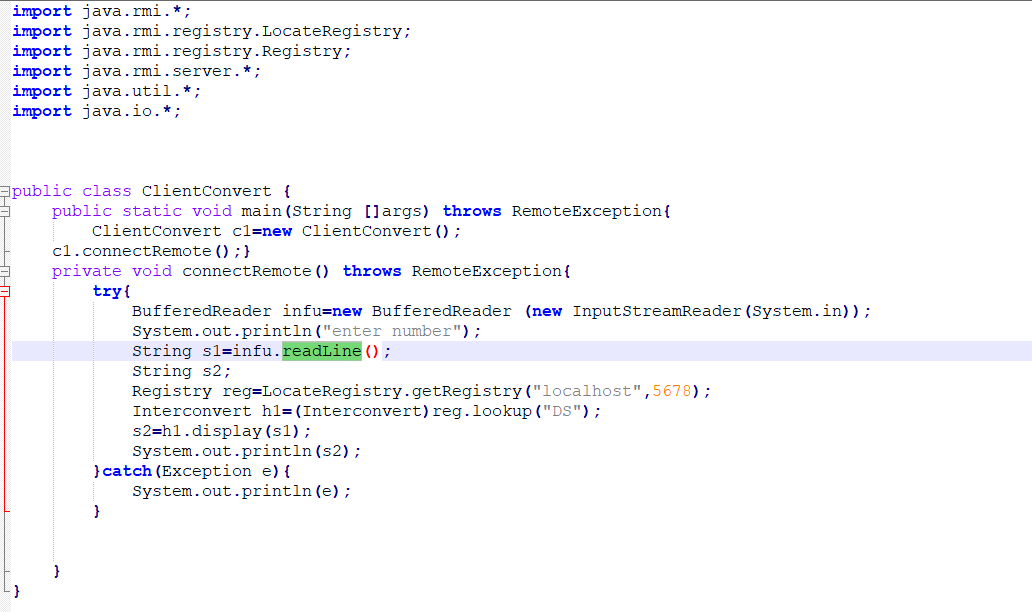


**ServerConvert.java**

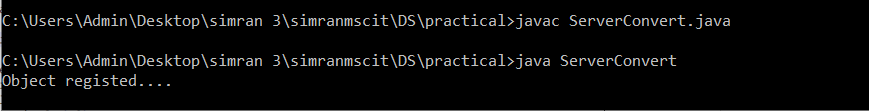


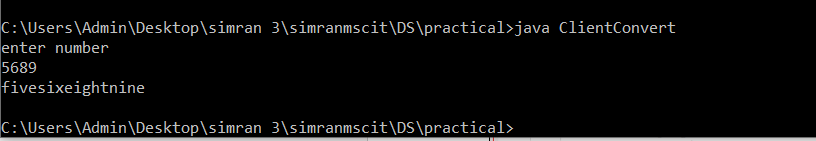


**ClientConvert.java**



**OUTPUT:**





**Practical No: 03**

**Aim: Show the implementation of Remote Procedure Call.**

**Remote Procedure Call (RPC):**

A remote procedure call is an inter process communication technique that is used for client server-based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call are given as follows:

• The client stub is called by the client

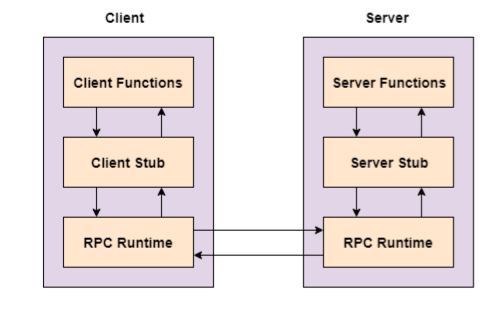
 • The client stub makes a system call to send the message to the server and puts the parameters in the message.

• The message is sent from the client to the server by the client’s operating system

 • The message is passed to the server stub by the server operating system.

• The parameters are removed from the message by the server stub. • Then, the server procedure is called by the server stub.

A diagram that demonstrates this is as follows:

****

The following steps take place during a RPC:

1. A client invokes a client stub procedure, passing parameters in the usual way. The client stub resides within the client’s own address space.

2. The client stub marshalls(pack) the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.

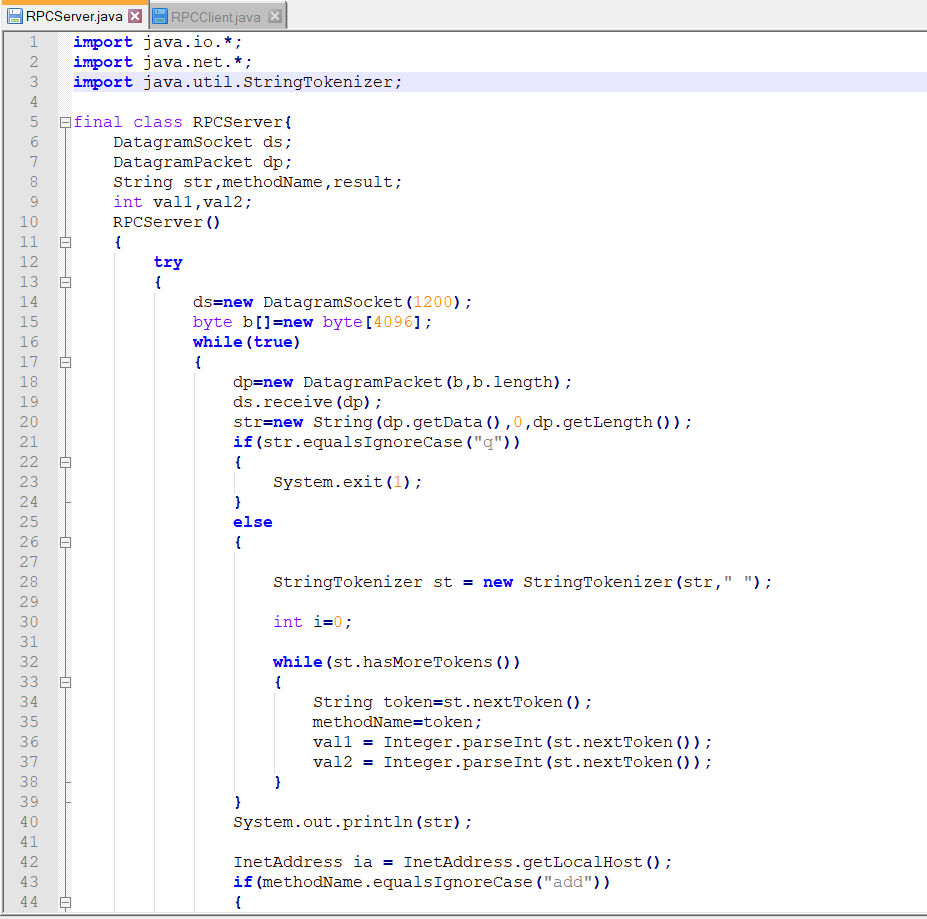
3. The client stub passes the message to the transport layer, which sends it to the remote server machine.

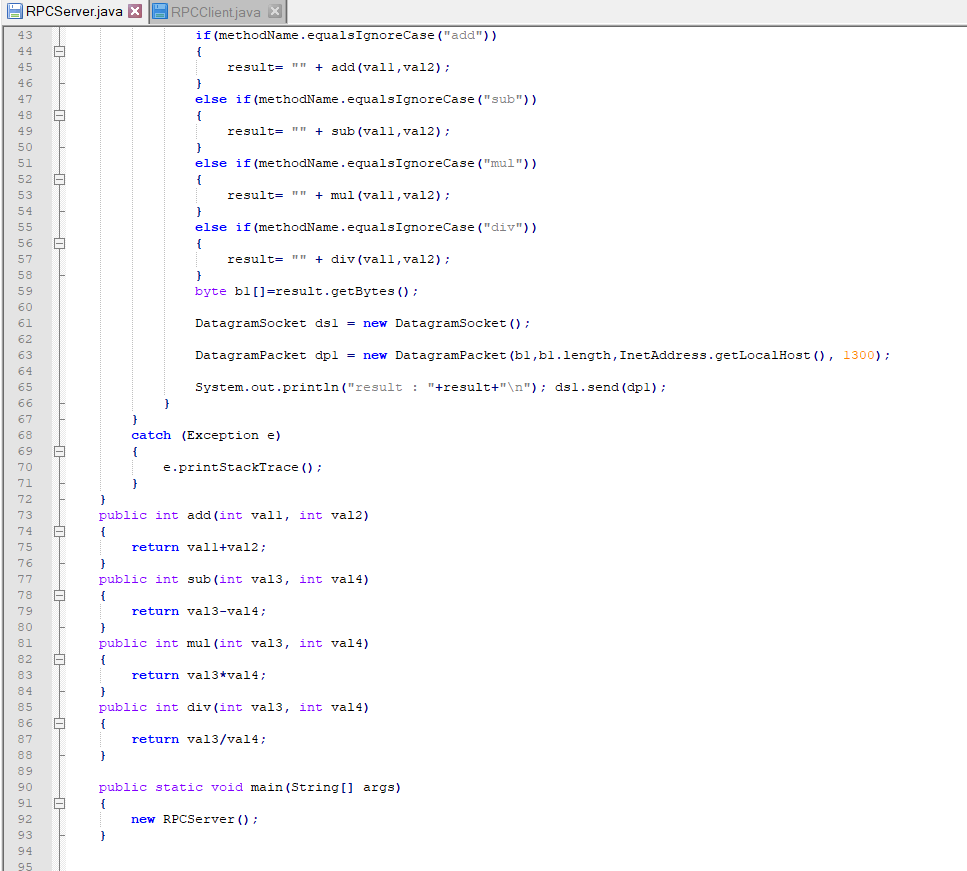
4. On the server, the transport layer passes the message to a server stub, which demarshalls(unpack) the parameters and calls the desired server routine using the regular procedure call mechanism.

**Example 3A: A program to implement simple calculator operations like addition, subtraction, multiplication and division.**

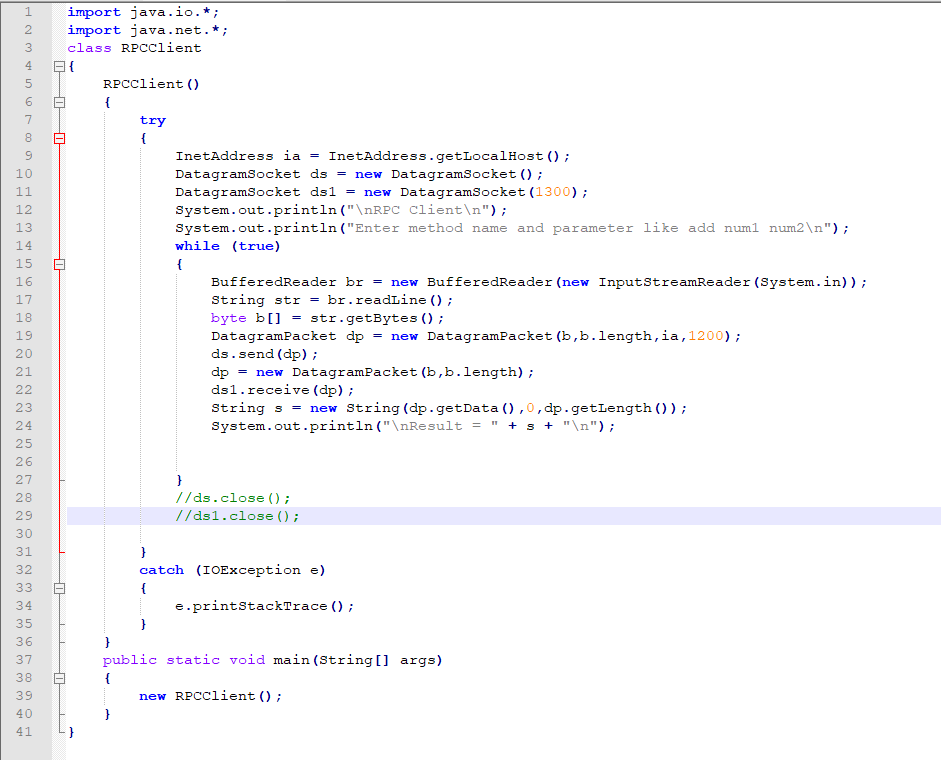
**CODE:**

**RPCServer.java**

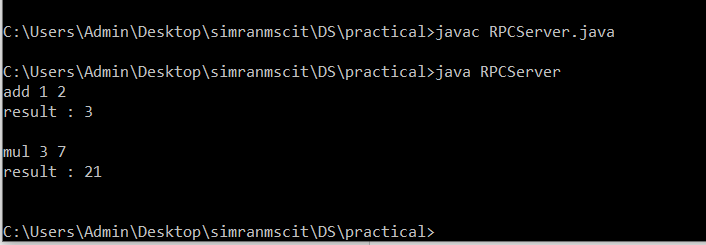


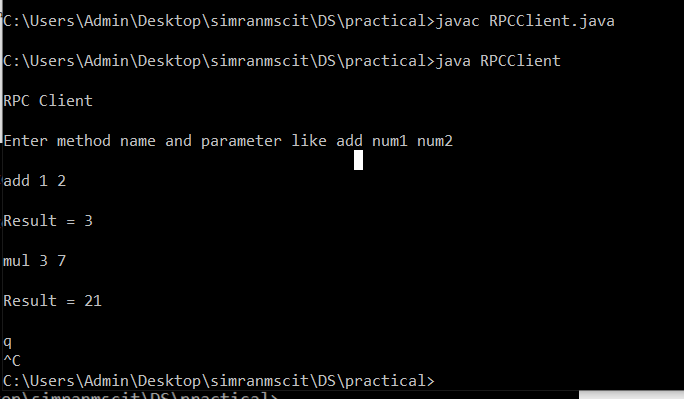


**RPCClient.java**



**OUTPUT:**





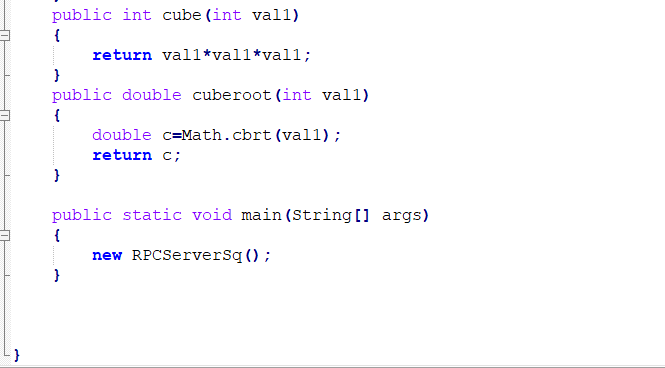
**Example 3B: A program that finds the square, square root, cube and cube root of the entered number.**

**CODE:**

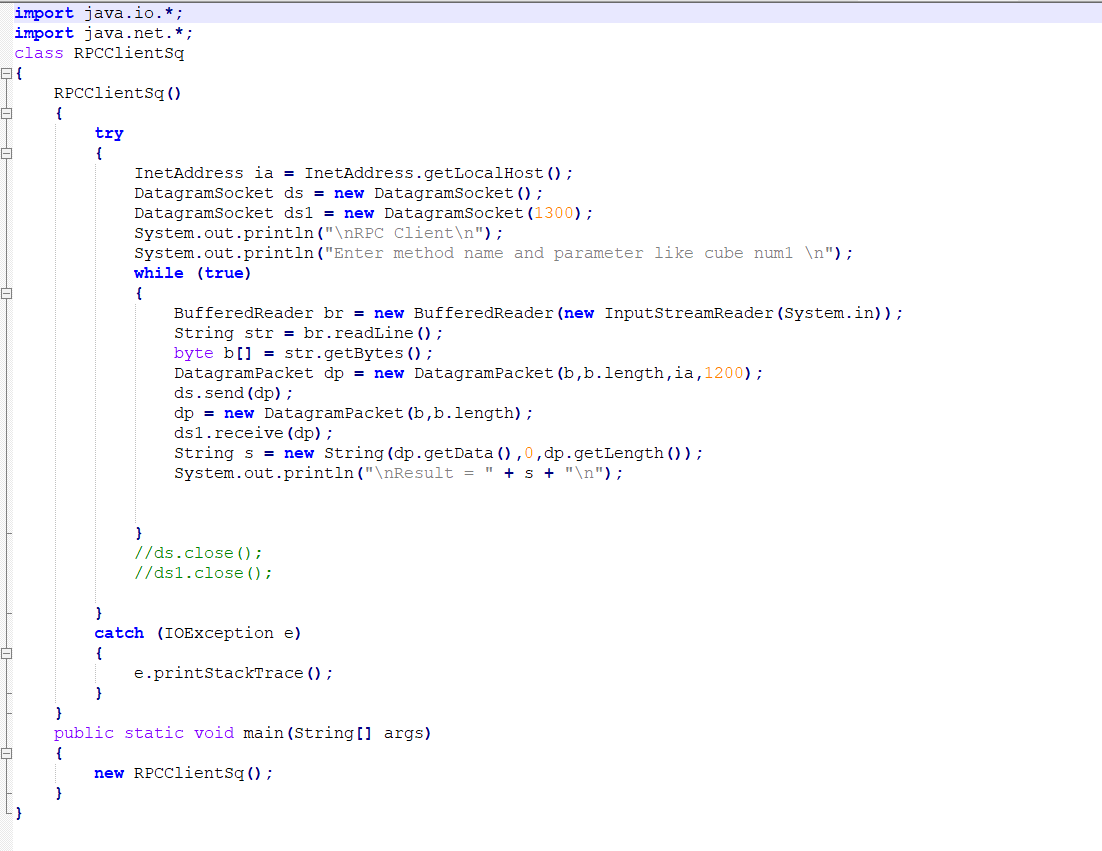
**RPCServerSq.java**



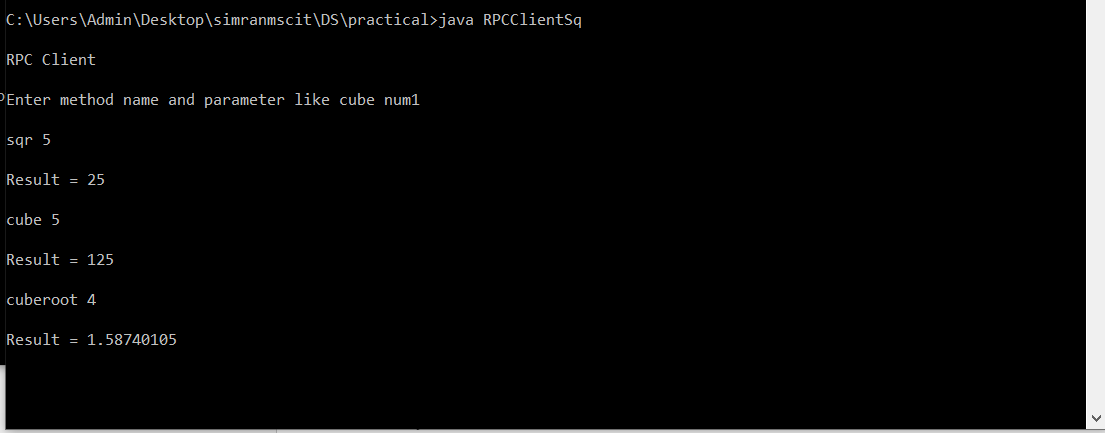




**RPCClientSq.java**



**OUTPUT:**





**Practical No: 04**

**Aim: Show the implementation of web services.**

**Web services:**

A Web Service is a software program that uses XML to exchange information with other software via common internet protocols. In a simple sense, Web Services are a way of interacting with objects over the Internet.

A web service is

• Language Independent.

• Protocol Independent.

• Platform Independent.

• It assumes a stateless service architecture.

• Scalable (e.g. multiplying two numbers together to an entire customer-relationship management system).

• Programmable (encapsulates a task).

• Based on XML (open, text-based standard).

• Self-describing (metadata for access and use).

• Discoverable (search and locate in registries)- ability of applications and developers to search for and locate desired Web services through registries. This is based on UDDI.

Key Web Service Technologies

1. XML- Describes only data. So, any application that understands XML-regardless of the application's programming language or platform has the ability to format XML in a variety of ways (well-formed or valid). 2. SOAP- Provides a communication mechanism between services and applications. 3. WSDL- Offers a uniform method of describing web services to other programs. 4. UDDI- Enables the creation of searchable Web services registries.

Web Services Limitations

1. SOAP, WSDL, UDDI- require further development. 2. Interoperability. 3. Royalty fees. 4. Too slow for use in high-performance situations. 5. Increase traffic on networks. 6. The lack of security standards for Web services. 7. The standard procedure for describing the quality (i.e. levels of performance, reliability, security etc.) of particular Web services – management of Web services. 8. The standards that drive Web services are still in draft form (always will be in refinement).

23

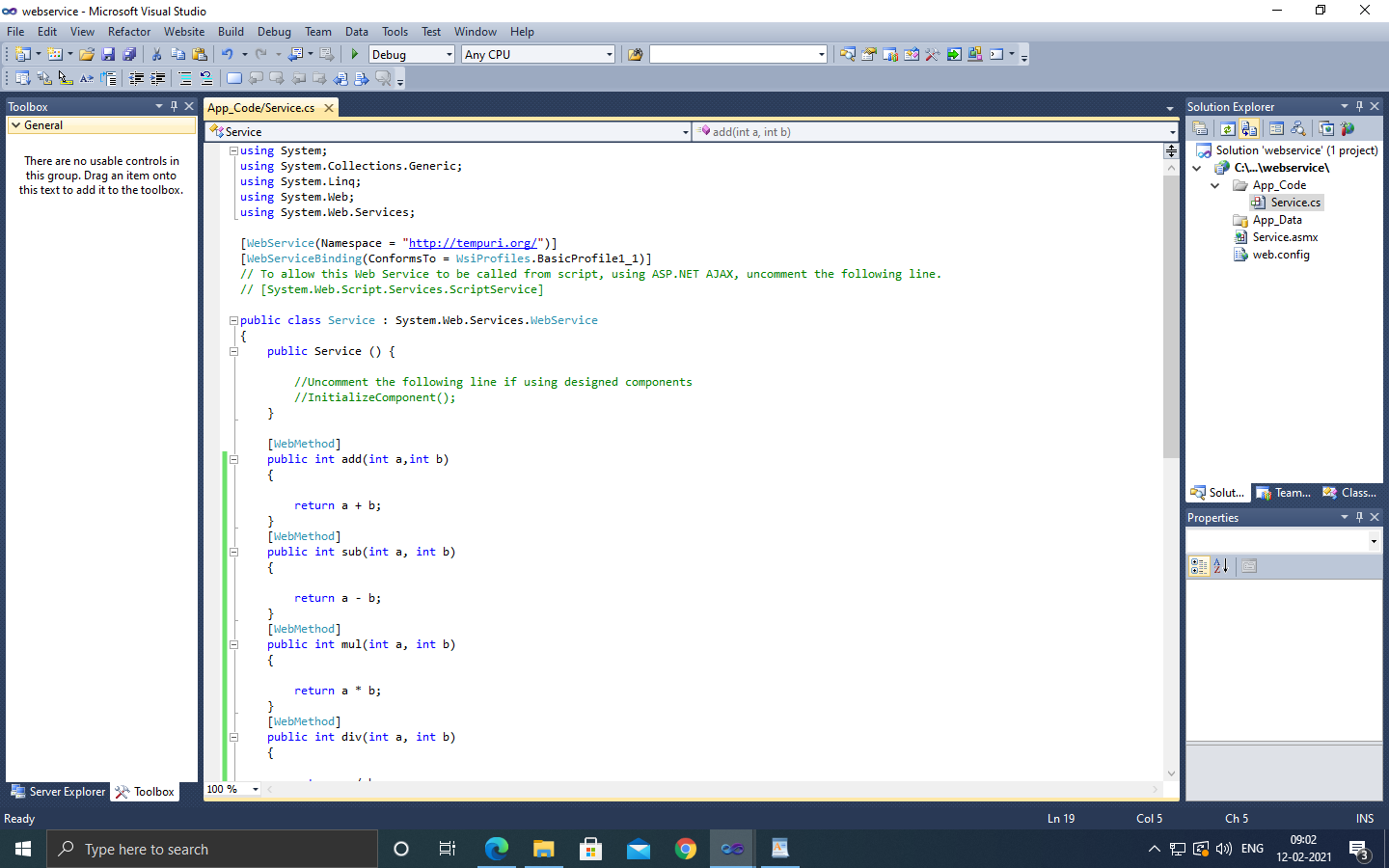
9. Some vendors want to retain their intellectual property rights to certain Web services standards.

A web service can perform almost any kind of task

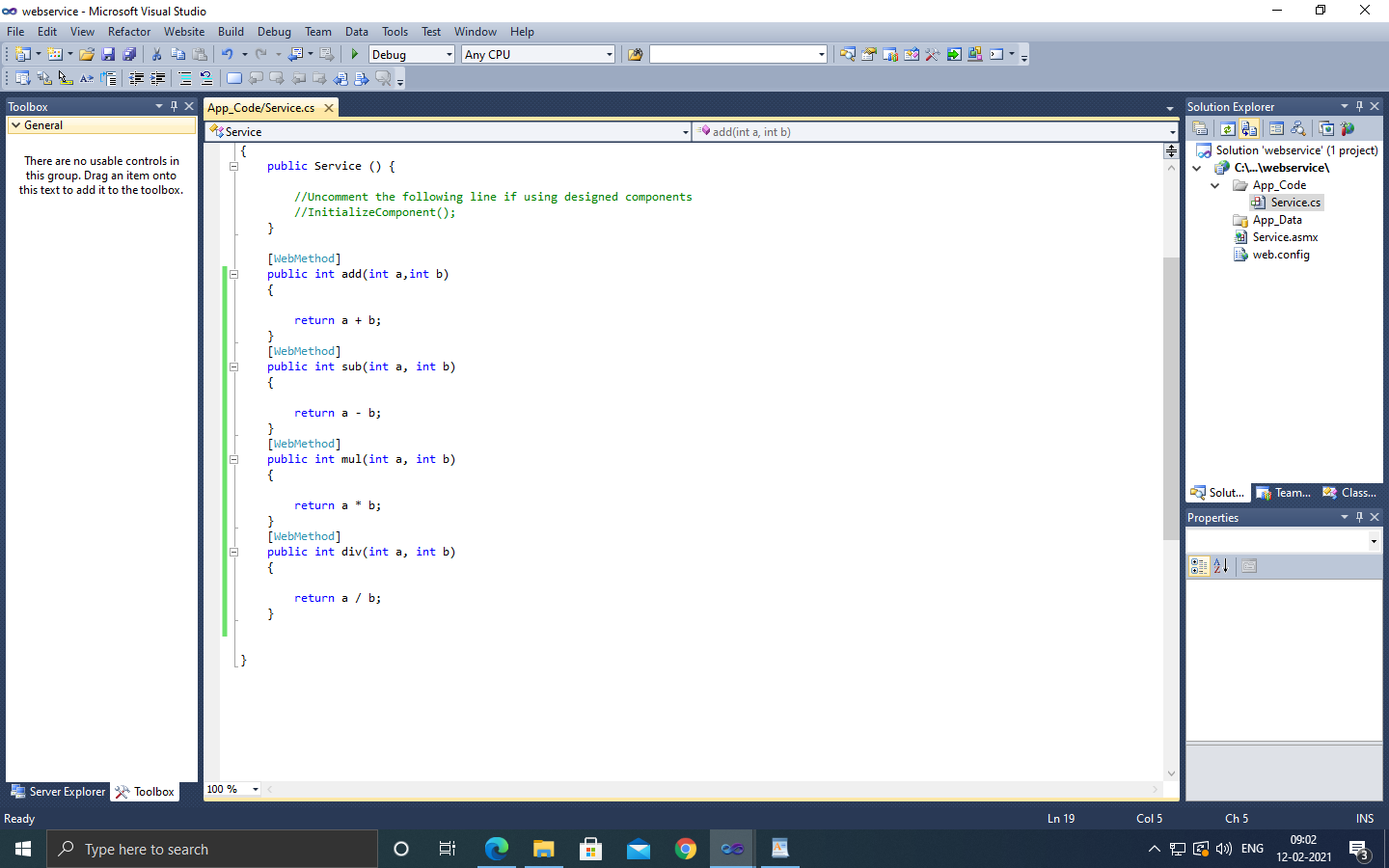
1. Web Portal- A web portal might obtain top news headlines from an associated press web service. 2. Weather Reporting- It can use as Weather reporting web service to display weather information in your personal website. 3. Stock Quote- It can display latest update of Share market with Stock Quote on your web site. 4. News Headline- You can display latest news update by using News Headline Web Service in your website. 5. You can make your own web service and let others use it. For example, you can make Free SMS Sending Service with footer with your companies’ advertisement, so whosoever uses this service indirectly advertises your company. You can apply your ideas in N no. of ways to take advantage of it.

**Step 1:** Create a ASP.NET Web Service. Click on File->New->Website->ASP.NET Web Service

and name the Web Service



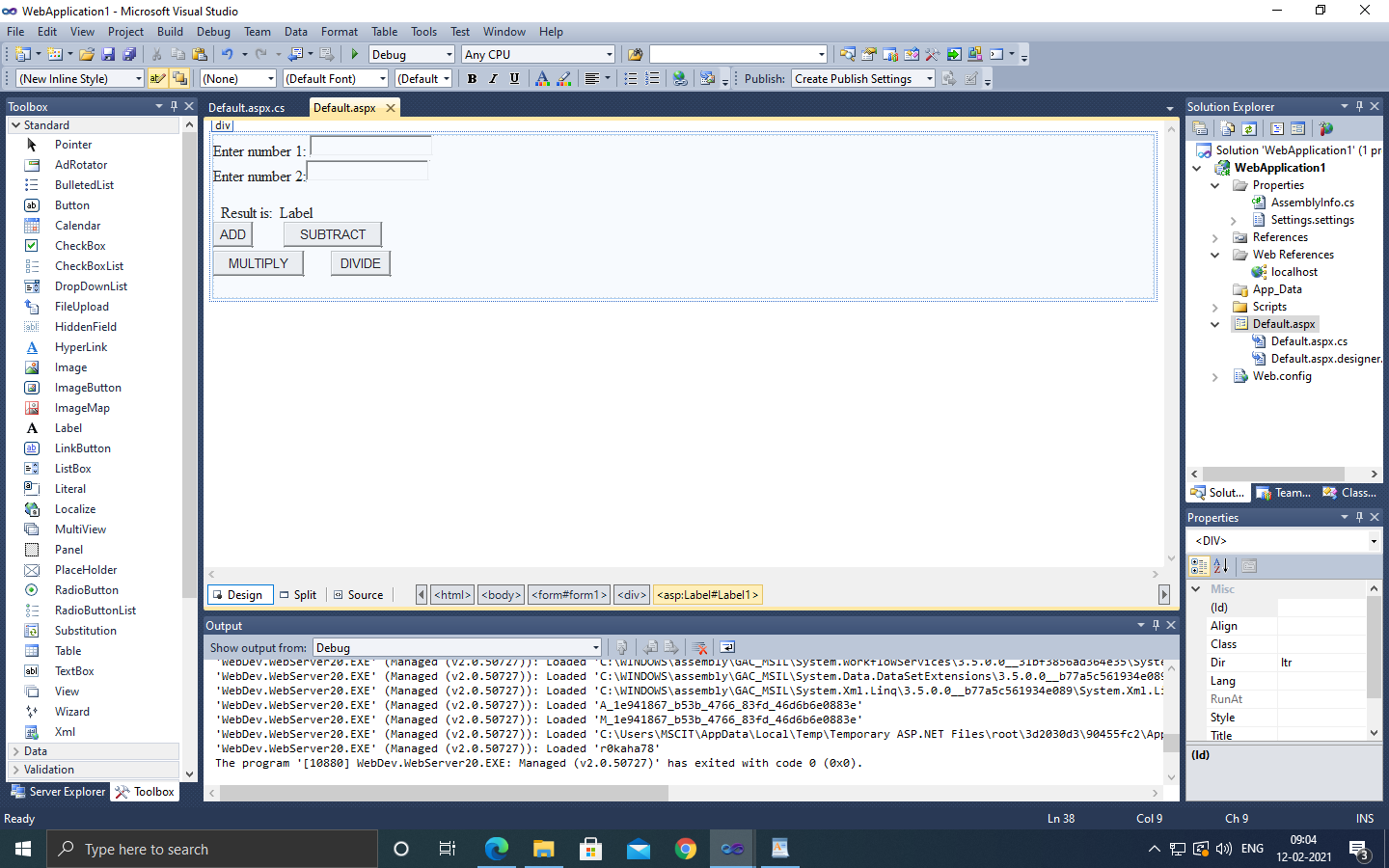
**Step 2:** Type the following code:



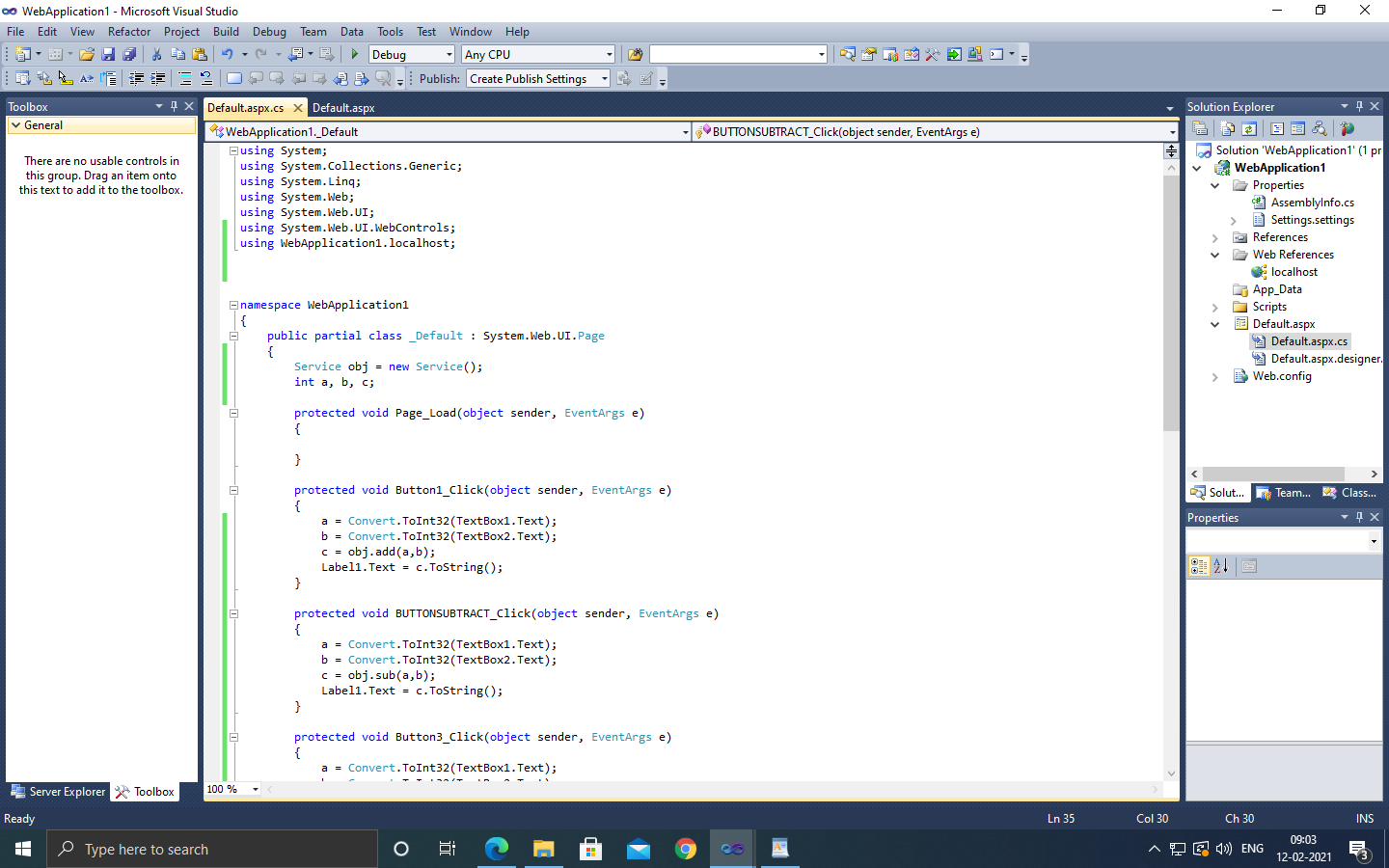
**Step 3:** Debug the Web Service and Copy the path

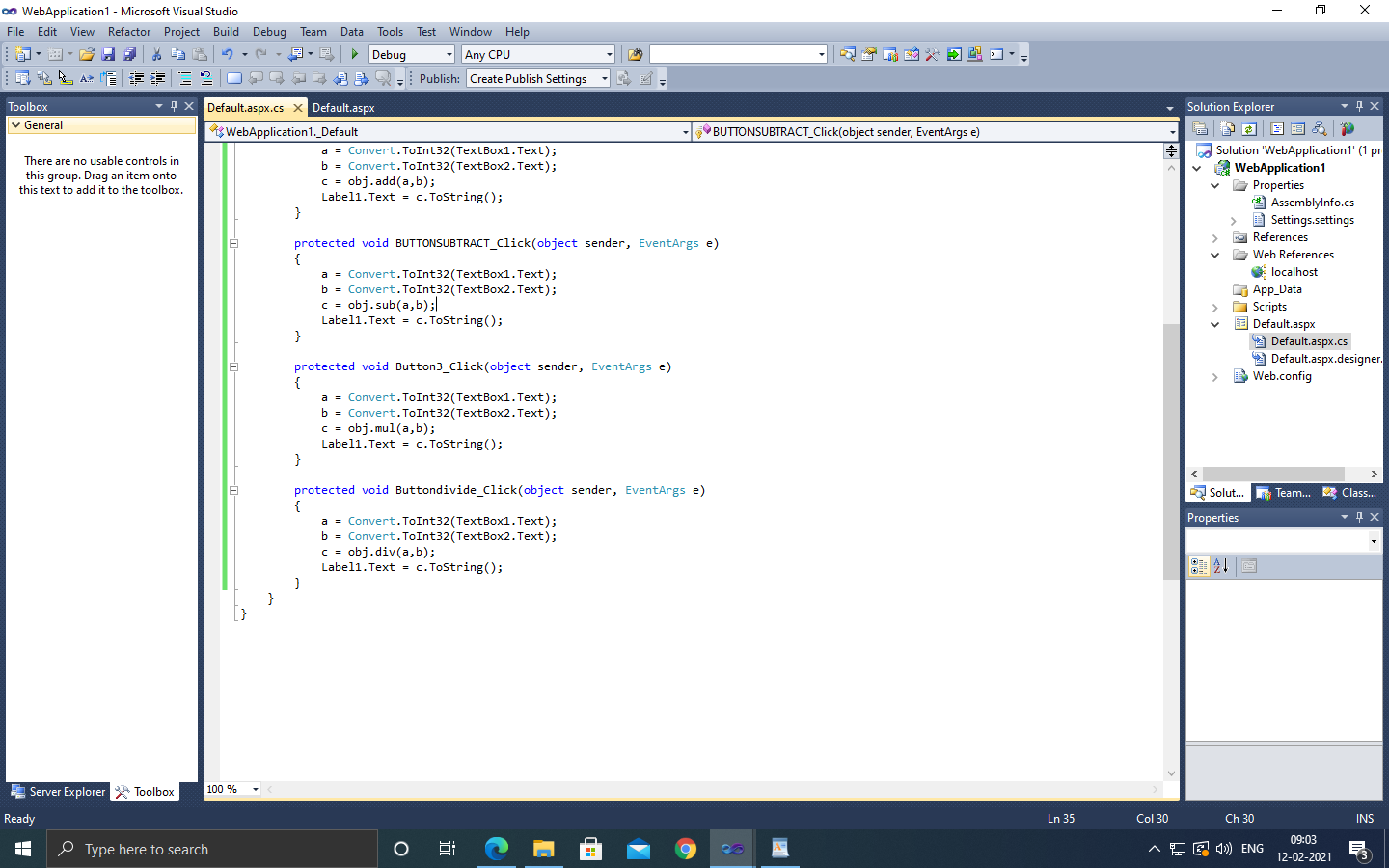
**Step 4:** Create a ASP.NET We Application. Click on File->New->Project->Web Application and name the Web Application.

**Step 5:** Design the simple calculator.

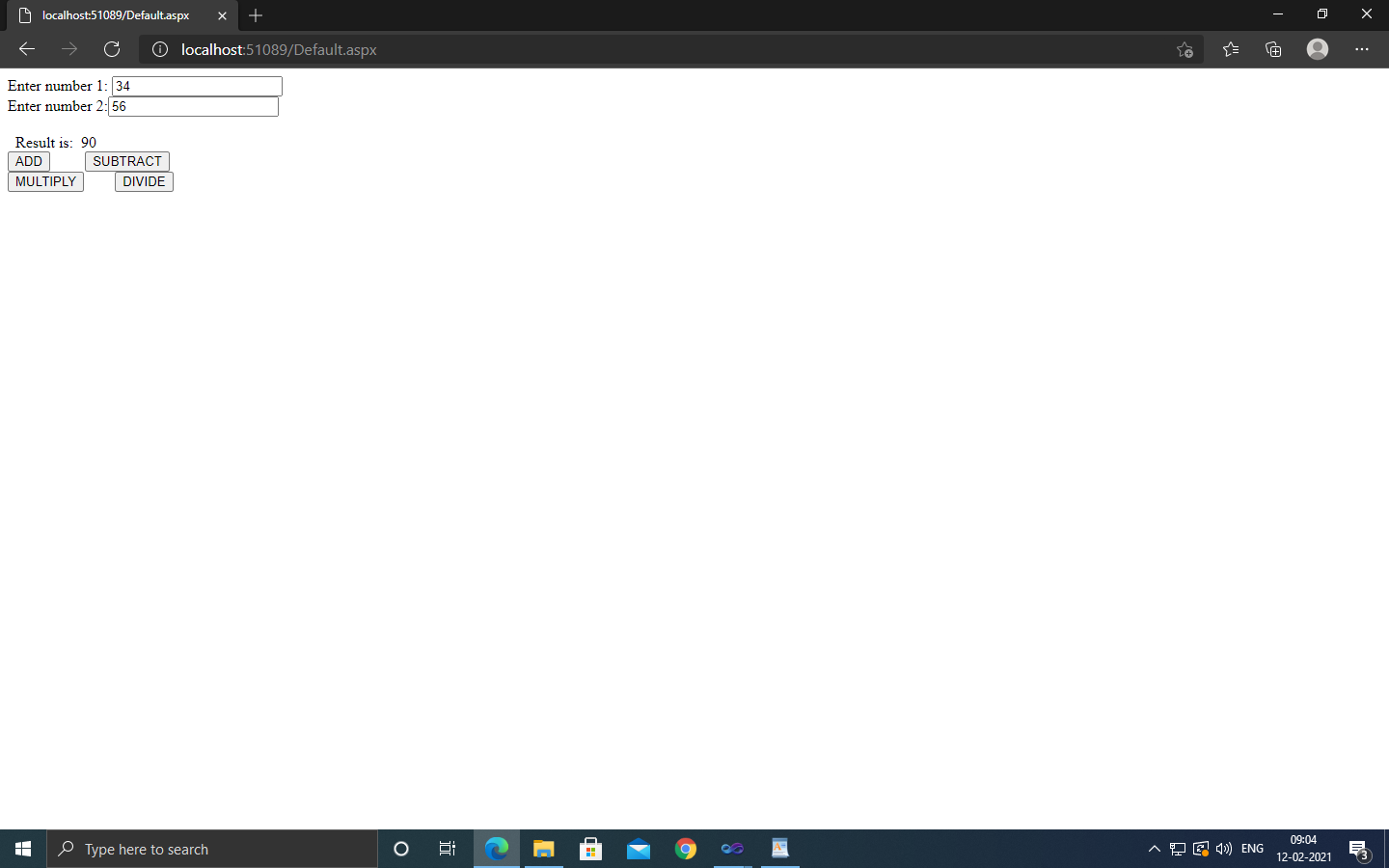


**Step 6:** Type the following code.





**OUTPUT:**



**Practical No: 05**

**Aim: Write a program to execute any one mutual exclusion algorithm.**

**Mutual exclusion:**

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process cannot enter its critical section while another concurrent process is currently present or executing in its critical section i.e. only one process is allowed to execute the critical section at any given instance of time.

Distributed mutual exclusion algorithms must deal with unpredictable message delays and incomplete knowledge of the system state.

Three basic approaches for distributed mutual exclusion:

1. Token based approach  2. Non-token-based approach  3. Quorum based approach

Token-based approach: • A unique token is shared among the sites.  • A site is allowed to enter its CS if it possesses the token. • Mutual exclusion is ensured because the token is unique. • Example: Suzuki-Kasami’s Broadcast Algorithm

Non-token-based approach: • Two or more successive rounds of messages are exchanged among the sites to determine which site will enter the CS next. • Example: Lamport's algorithm, Ricart–Agrawala algorithm

Quorum based approach: • Each site requests permission to execute the CS from a subset of sites (called a quorum). • Any two quorums contain a common site.  • This common site is responsible to make sure that only one request executes the CS at any time. • Example: Maekawa’s Algorithm

Requirements of Mutual Exclusion Algorithms:

1. No Deadlock: Two or more site should not endlessly wait for any message that will never arrive.

32

2. No Starvation: Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site is repeatedly executing critical section 3. Fairness: Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system. 4. Fault Tolerance: In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

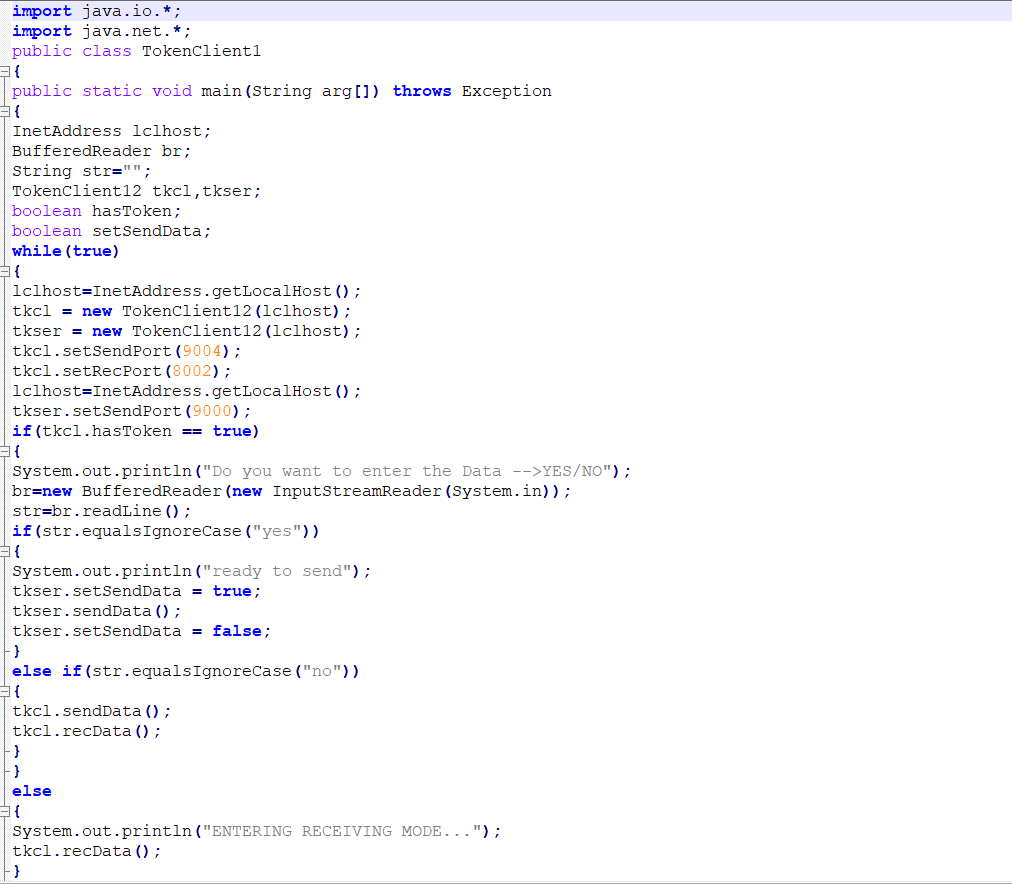
**Example 5A: Write a program to execute any one mutual exclusion algorithm**

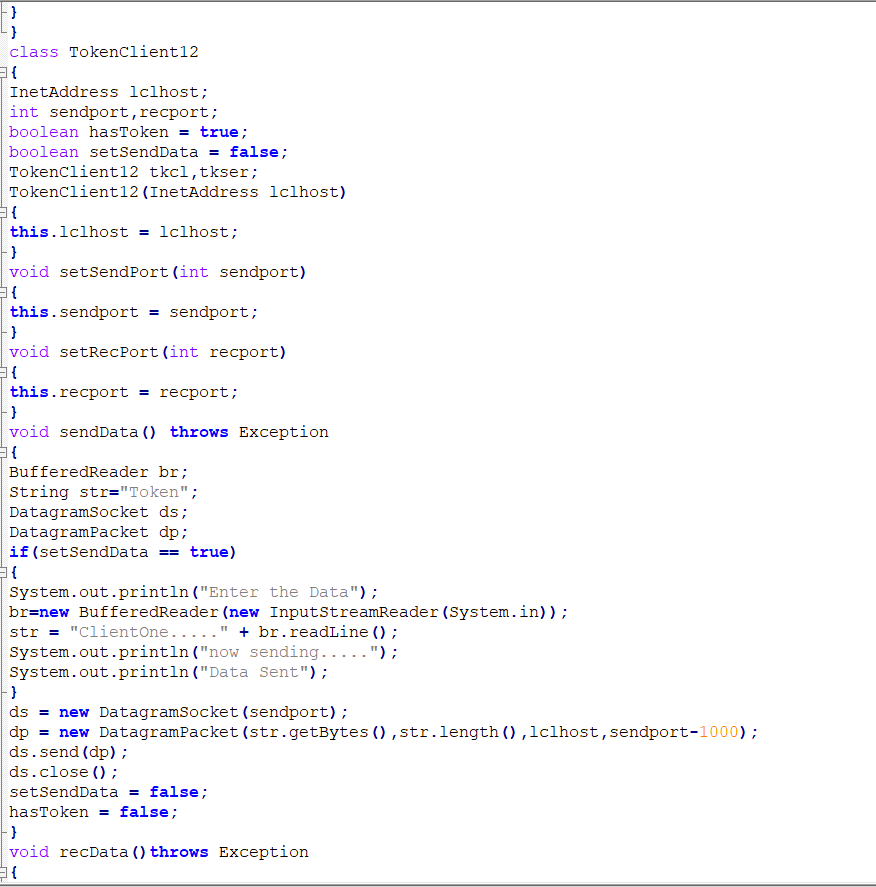
**CODE:**

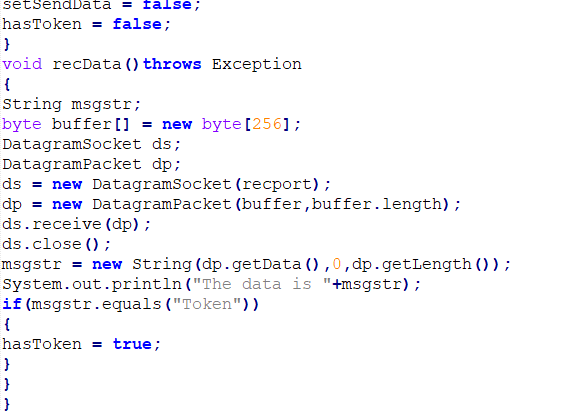
**TokenServer.java**



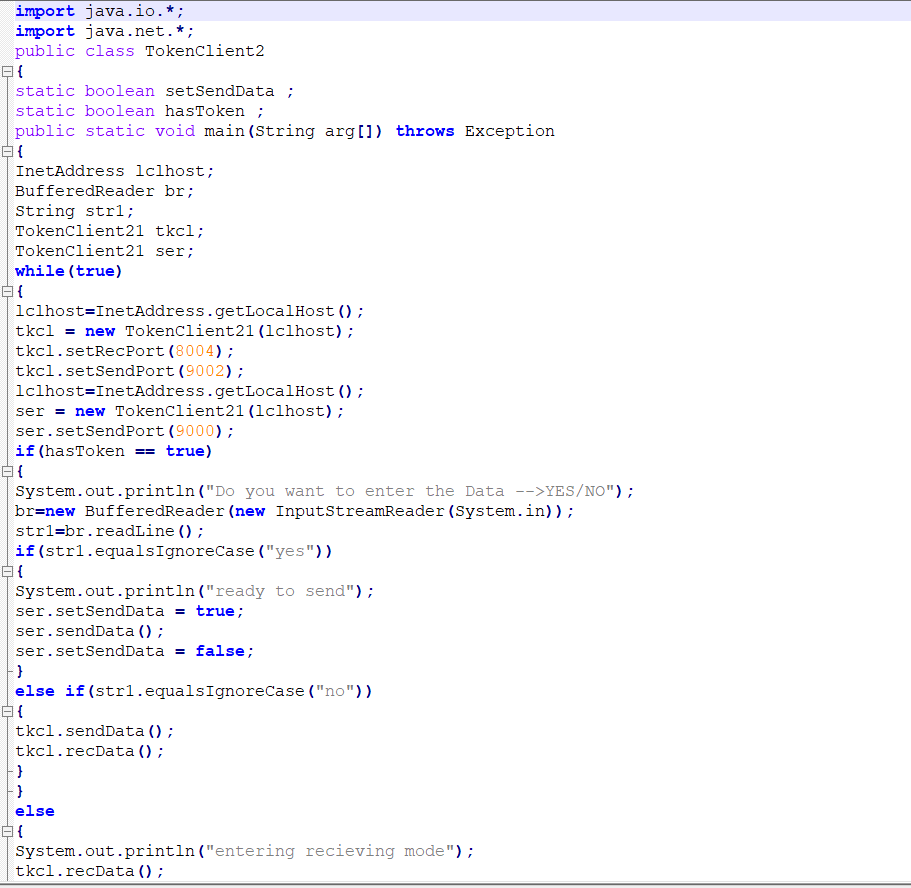
**TokenClient1.java**



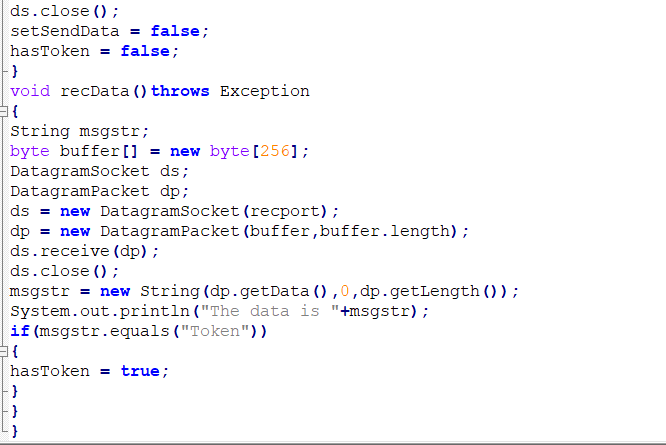




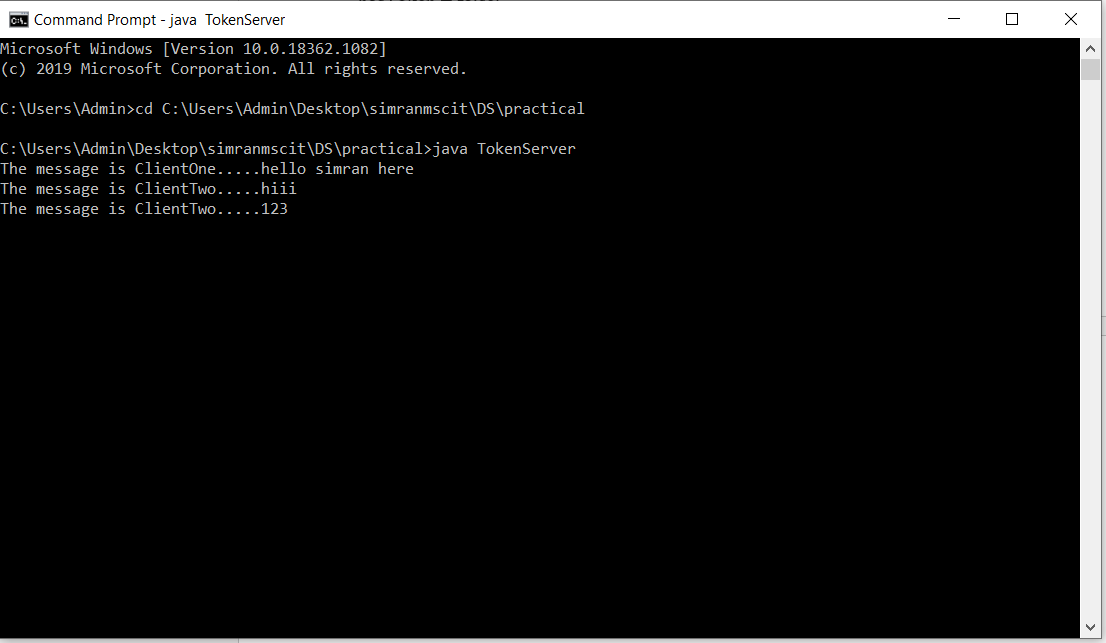
**TokenClient2.java**

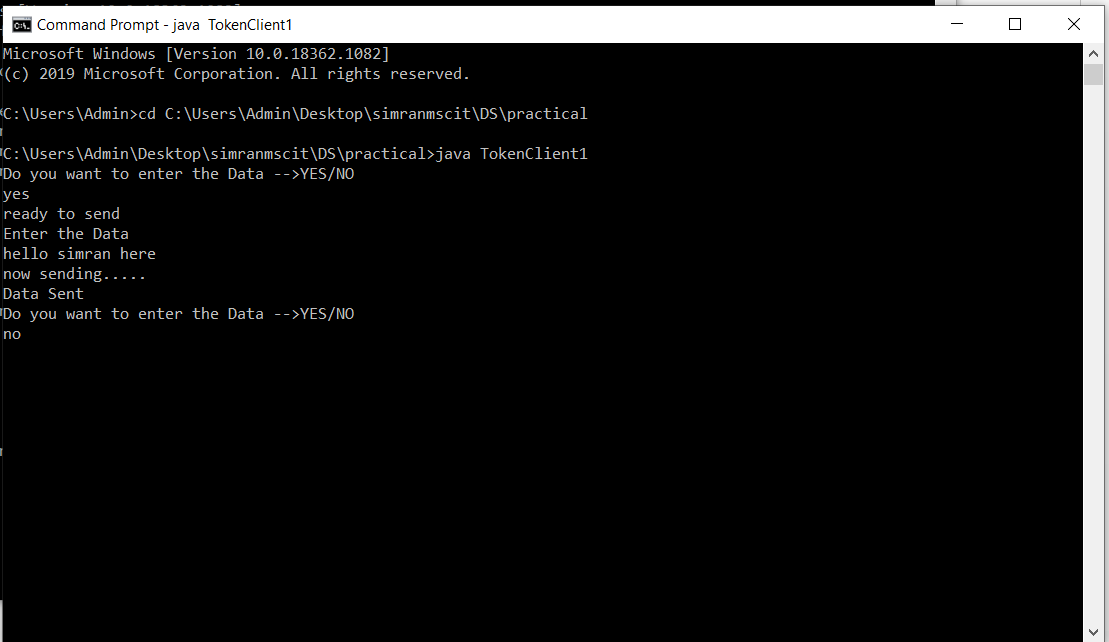


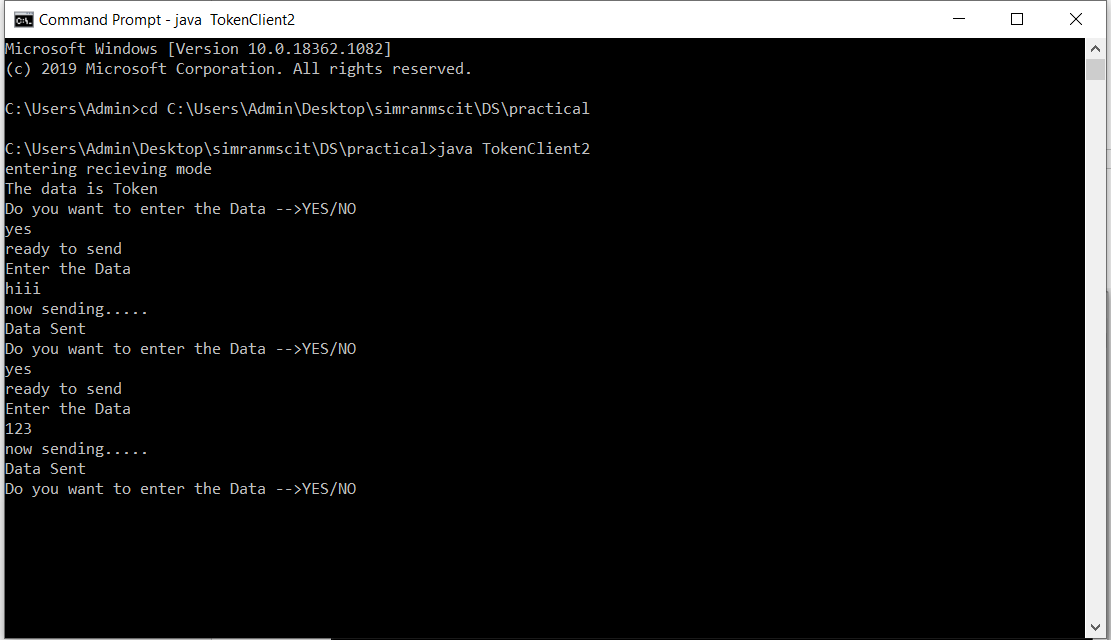




**OUTPUT:**







**Practical No: 06**

**Aim: Write a program to implement any one election algorithm.**

**Election algorithms**

Many distributed algorithms require the election of a special coordinator process; one that has a special role, or initiates something, or monitors something. Often it doesn't matter who the special process is, but one and only one must be elected, and it can't be known in advance who it will be. On a low-level you can think about the monitor in a token ring as an example.

The assumptions of these algorithms are that every process can be uniquely identified (by IP address, for example), and that each process can find out the id of the other processes. What the processes don't know is which processes are up and which are down at any given point in time.

**Bully algorithm**

Garcia-Molina, 1982. The process with the highest identity always becomes the coordinator.

When a process P sees that the coordinator is no longer responding to requests it initiates an election by sending ELECTION messages to all processes whose id is higher than its own. If no one responds to the messages then P is the new coordinator. If one of the higher-ups responds, it takes over and P doesn't have to worry anymore.

When a process receives an ELECTION message it sends a response back saying OK. It then holds its own election (unless it is already holding one). Eventually there is only one process that has not given up and that is the new coordinator. It is also the one with the highest number currently running. When the election is done the new coordinator sends a COORDINATOR message to everyone informing them of the change.

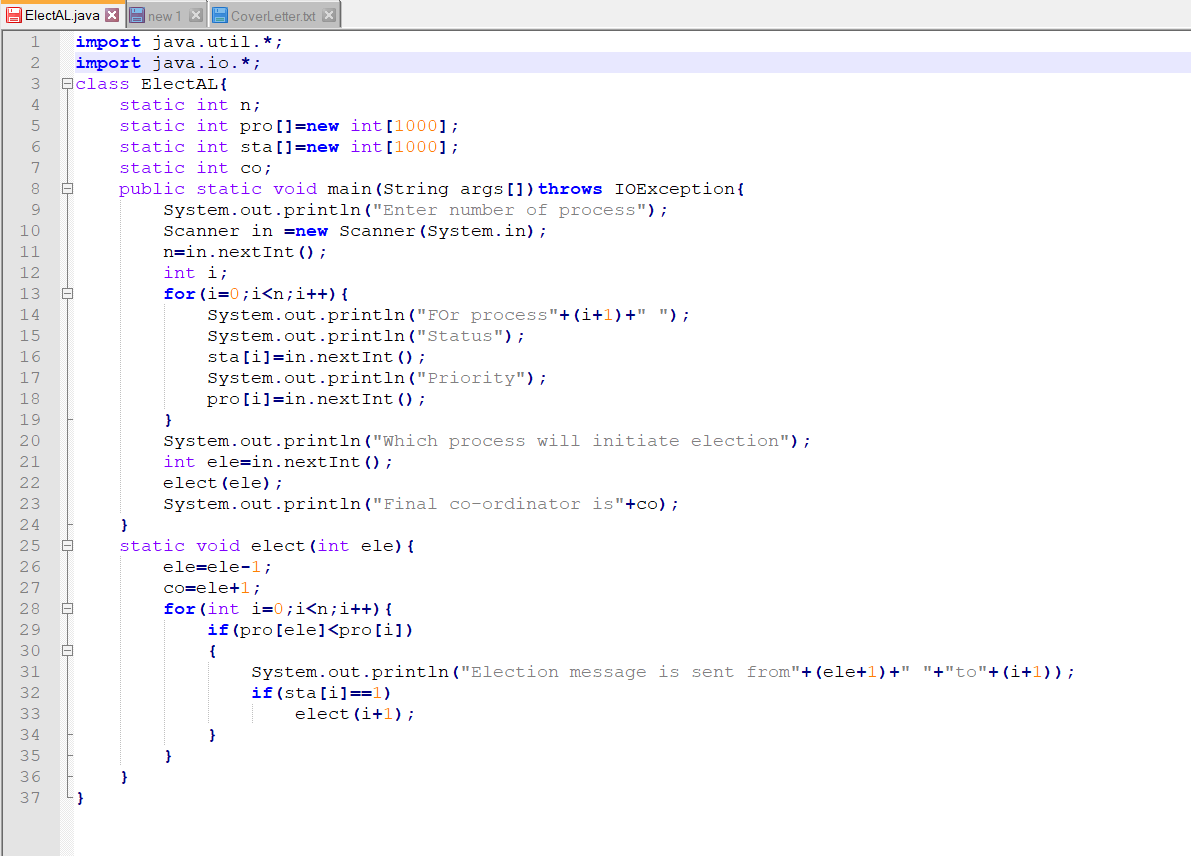
If a process which was down comes back up, it immediately holds an election. If this process had previously been the coordinator it will take this role back from whoever is doing it currently (hence the name of the algorithm).

**Ring algorithm**

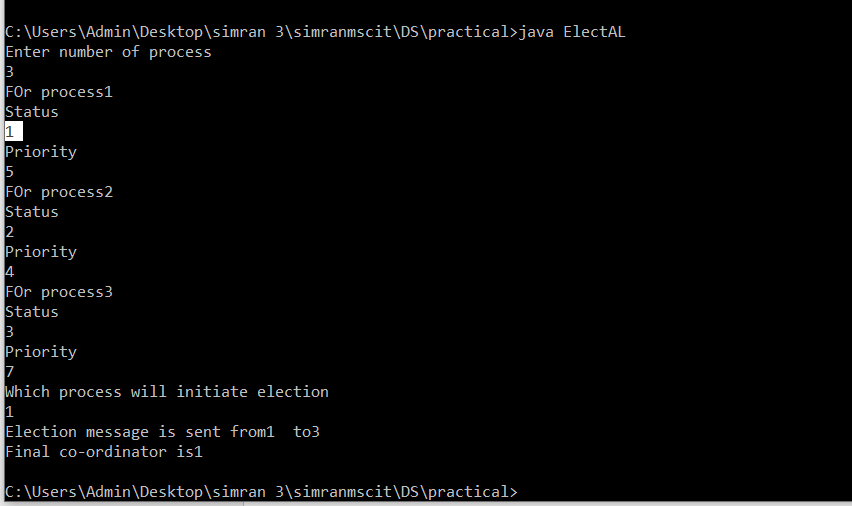
Assumes that processes are logically ordered in some fashion, and that each process knows the order and who is coordinator. No token is involved. When a process notices that the coordinator is not responding it sends an ELECTION message with its own id to its downstream neighbor. If that neighbor doesn't respond it sends it to its neighbor's neighbor, etc. Each station that receives the ELECTION message adds its own id to the list. When the message circulates back to the originator it selects the highest id in the list and sends a COORDINATOR message announcing the new coordinator. This message circulates once and is removed by the originator.

If two elections are held simultaneously (say because two different processes notice simultaneously that the coordinator is dead) then each comes up with the same list and elects the same coordinator. Some time is wasted, but nothing is really hurt by this.

**Code:**



**OUTPUT:**



**Practical No: 07**

**Aim: Show the implementation of any one clock synchronization algorithm.**

**Clock synchronization algorithm:**

Clock synchronization is a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node. During the synchronization, many factors effect on a network. As discussed above, these factors need to be considered before correcting actual clock value. Based on the approach, clock synchronization algorithms are divided as Centralized Clock Synchronization and Distributed Clock Synchronization Algorithm.

Distributed algorithm:

• There is particular time server.

• The processor periodically reach an agreement on the clock value by averaging the time of neighbour clock and its local clock.

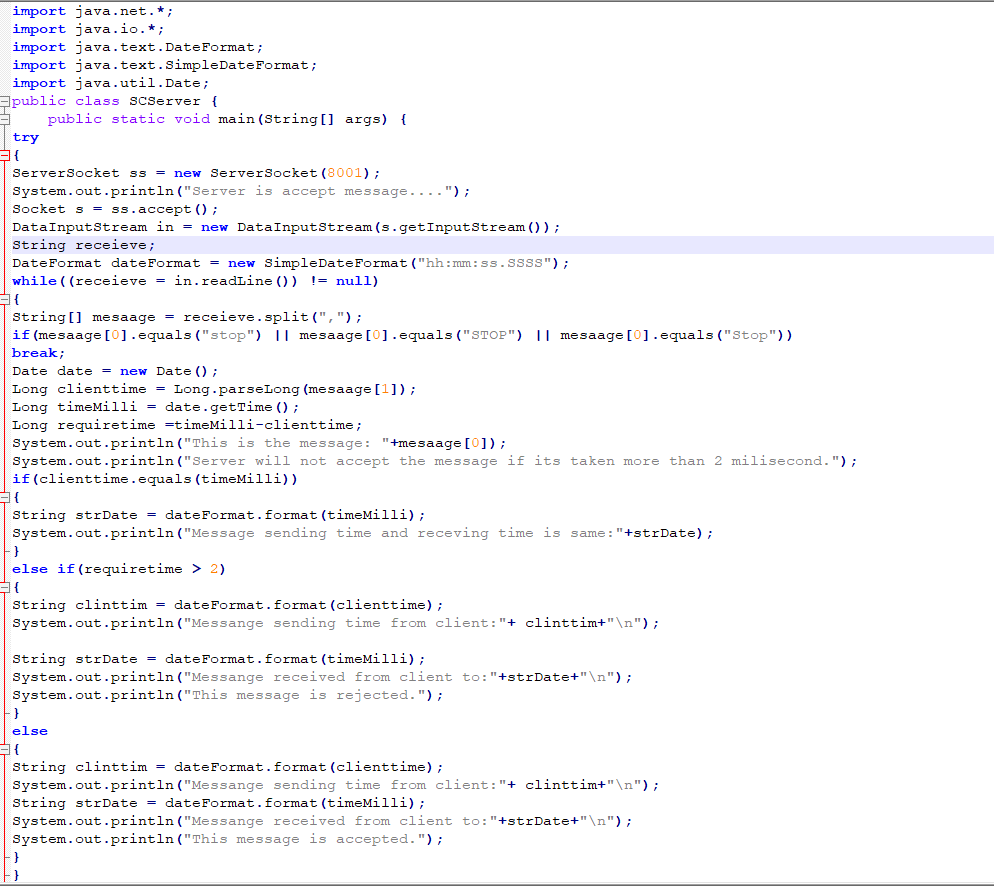
• This can be used if no UTC receiver exist (no external synchronization is needed). Only internal synchronization is performed.

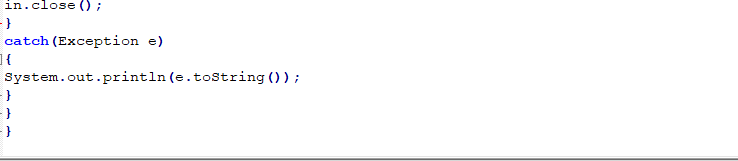
• Processes can run different machine and no global clock to judge which event happened first.

**Aim: Write the implementation of any one clock synchronization algorithm**

**Code:**

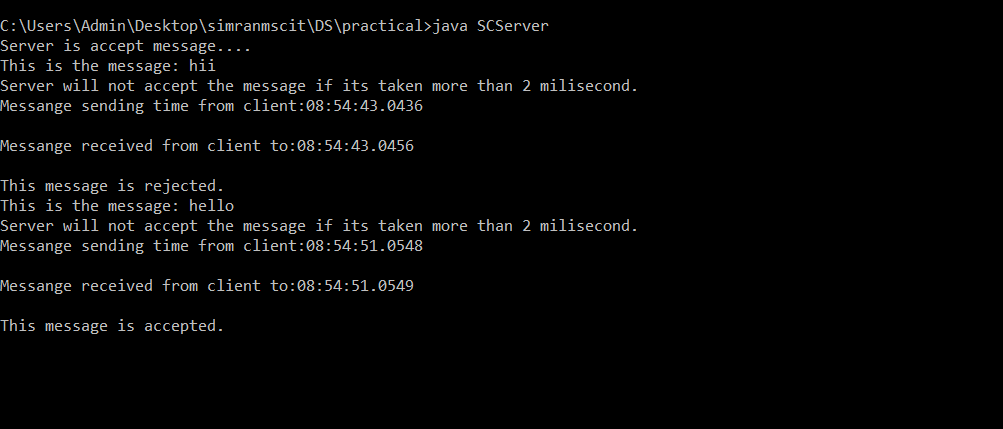
**SCServer.java**

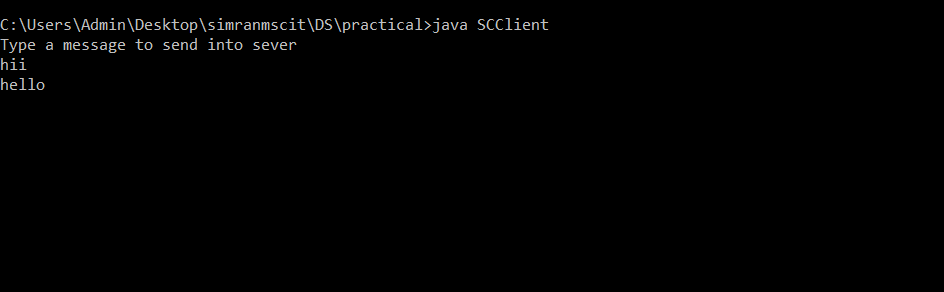




**SCClient.java**

**OUTPUT:**





**Practical 8;**

**Aim: Write a program to implement two phase commit protocol.**

This protocol is designed with the core intent to resolve the above problems, Consider we have multiple distributed databases which are operated from different servers(sites) let’s say **S1, S2, S3, ….Sn.**Where every **Si** made to maintains a separate log record of all corresponding activities and the transition **T**has also been divided into the subtransactions **T1, T2, T3, …., Tn** and each **Ti**are assigned to **Si.**This all maintains by a separate transaction manager at each **Si.**We assigned anyone site as a **Coordinator.**

**Some points to be considered regarding this protocol:**

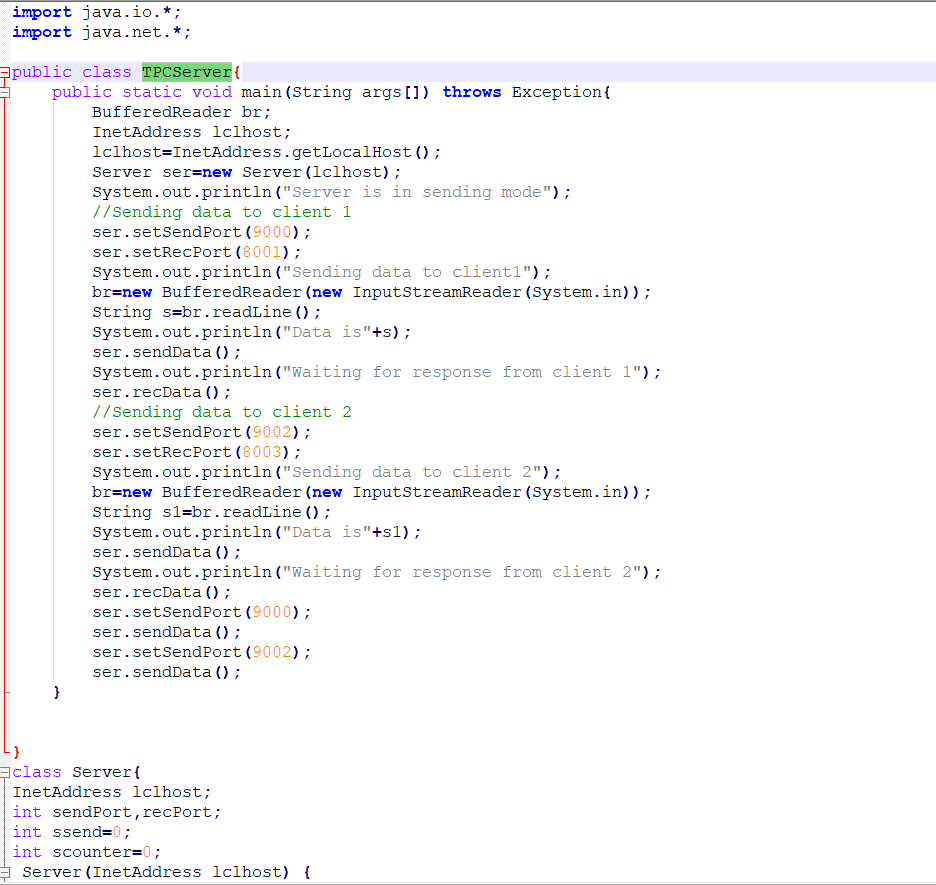
**a)**In a two-phase commit, we assume that each site logs actions at that site, but there is no global log.

**b)**The **coordinator(Ci),**plays a vital role in doing confirmation whether the distributed transaction would abort or commit.

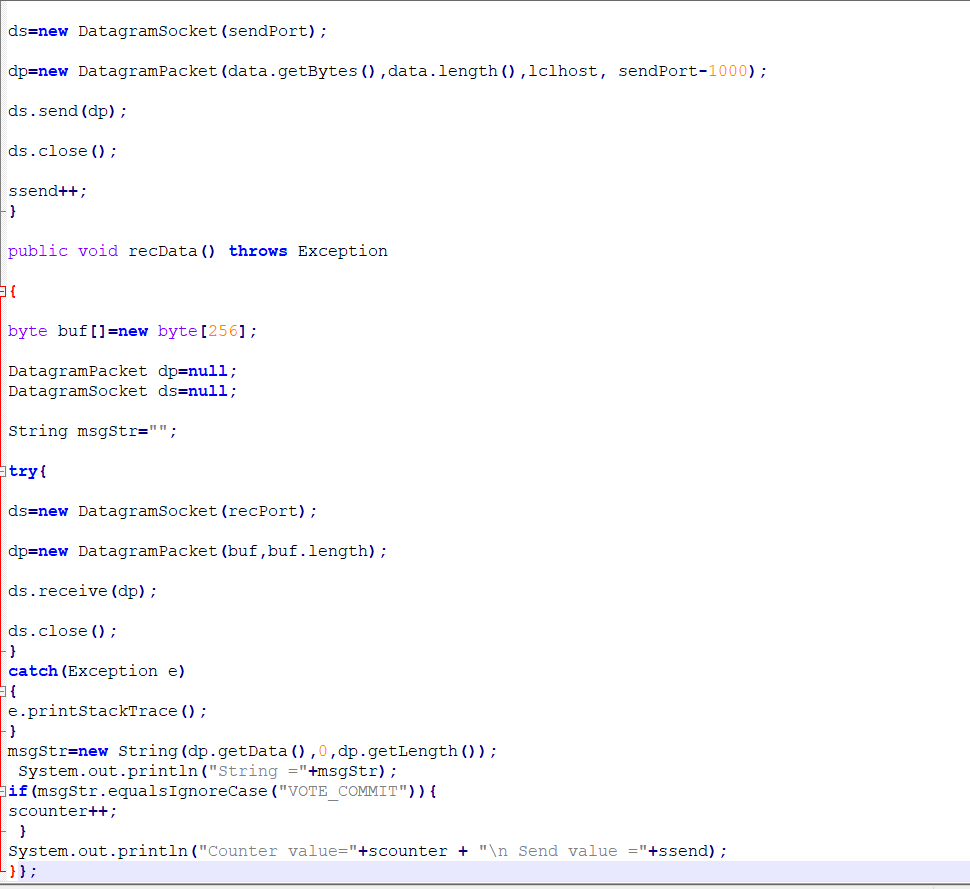
**c)** In this **protocol** messages are made to send between the **coordinator(Ci)** and the other **sites.** As each message is sent, its logs are noted at each sending site, to aid in recovery should it be necessary.

**CODE:**

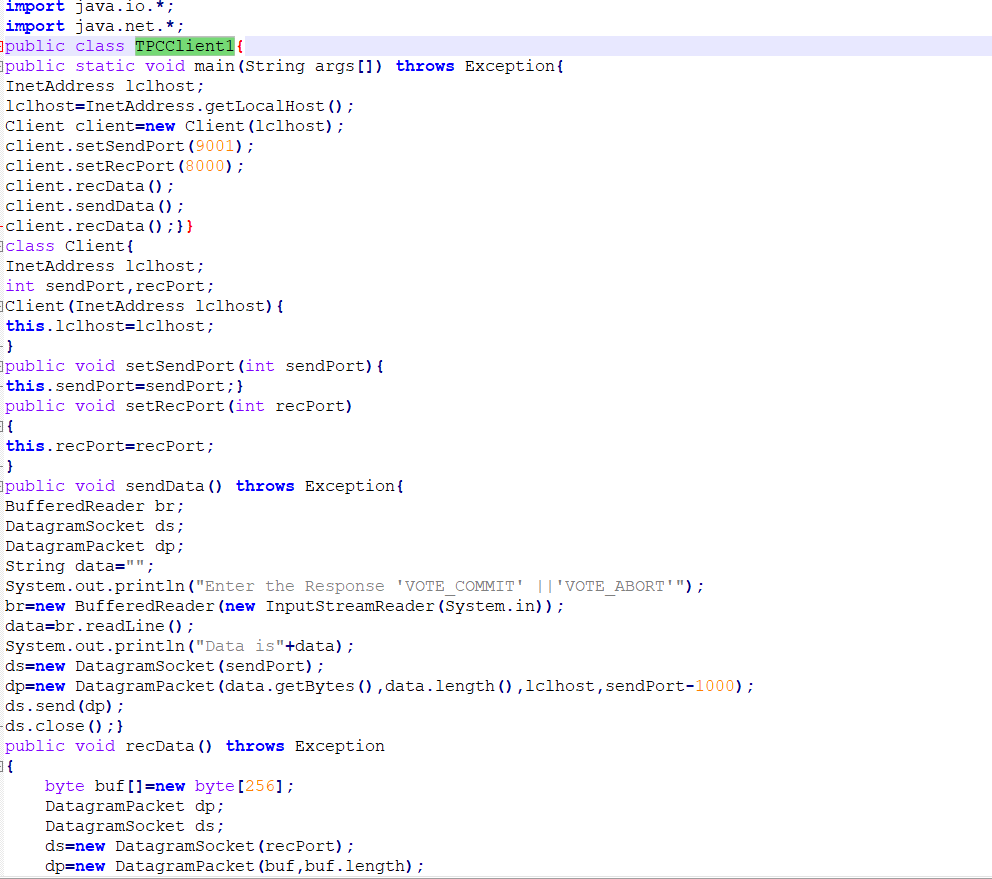
**TPCServer.java**

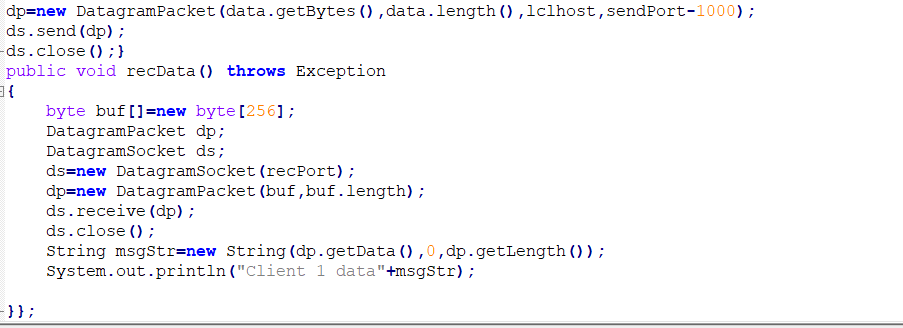




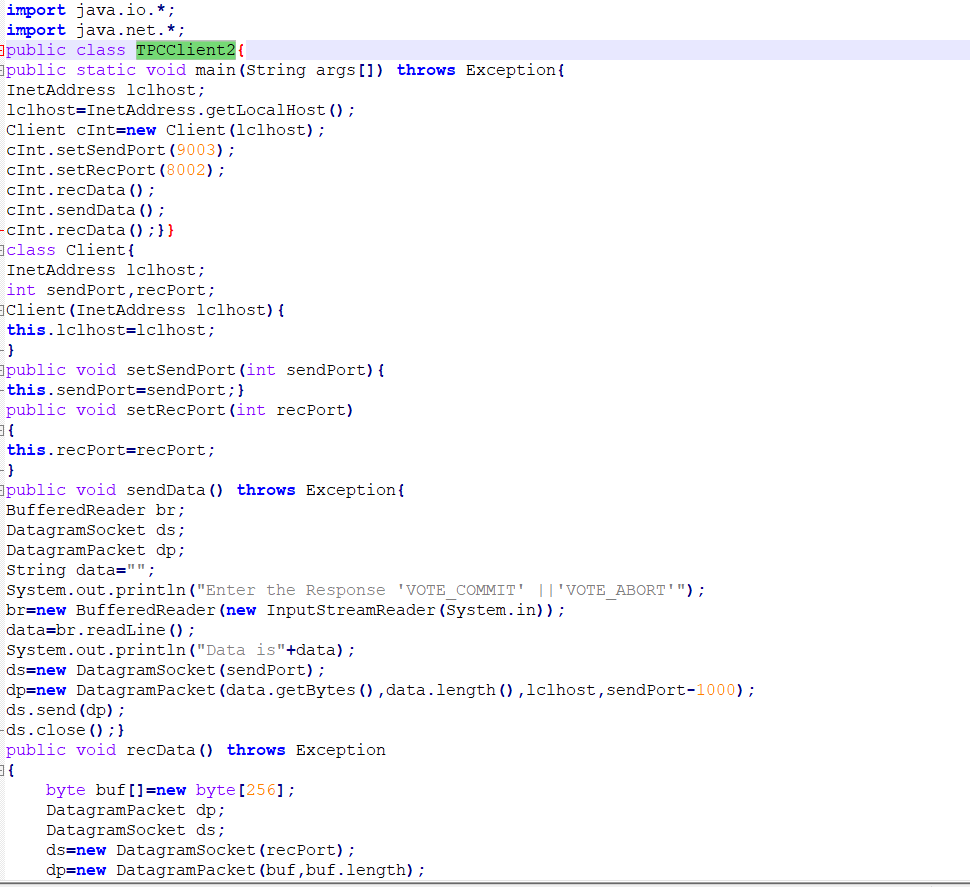


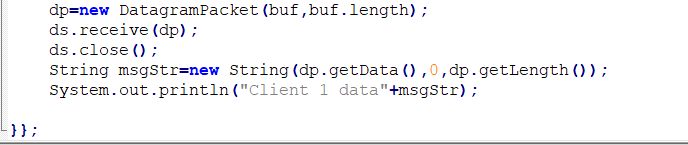
**TPCClient1.java**





**TPCClient2.java**





**OUTPUT:**

