S.Y.M.C.A.
(TWO YEARS PATTERN)
SEMESTER - III (CBCS)

DISTRIBUTED SYSTEM AND CLOUD COMPUTING LAB

SUBJECT CODE : MCAL34

© UNIVERSITY OF MUMBAI

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programe Co-ordinator : **Mr. Mandar L. Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai – 400098.

Course Co-ordinator : **Ms. Reshma Kurkute**
Assistant Professor B.Sc.IT, IDOL,
IDOL, University of Mumbai- 400098.

Course Writers : **Ms. Rani Podichetty**
Assistant Professor
K.B.College ofArts and Commerce for women, Thane (East)

: **Dr. Rakhee Yadav**
Assistant Professor
S. K Somaiya college, somaiya vidyavihar university

: **Mrs. Pinky Rane**
Assistant Professor
New horizon college of commerce

: **Mrs. Saba Ansari**
Assistant Professor
J.K College of Science and Commerce

: **Mr. Sandeep Kamble**
Assistant Professor
Cosmopolitan's Valia College

: **Mr. Kashafurrehman Ansari**
Assistant Professor

June 2022, Print I

Published by
Director
Institute of Distance and Open Learning, University of Mumbai,Vidyanagari, Mumbai - 400 098.

DTP COMPOSED AND PRINTED BY
Mumbai University Press,
Vidyanagari, Santacruz (E), Mumbai - 400098.

CONTENTS

Chapter No.	Title	Page No.
Module I		
1.	Remote Process Communication	1
Module II		
2 .	Remote Procedure Call	12
Module III		
3.	Remote Method Invocation	20
Module IV		
4.	Remote Object Communication	39
Module V		
5.	Mutual Exclusion	57
Module VI		
6.	Implementation Of Cloud Computing Services	75
Module VII		
7.	Implementation Of Identity Management Using Cloud	84
Module VIII		
8.	App Development Using Cloud Computing	97

*****

SYLLABUS

Course Code	Course Name
MCAL34	Distributed System and Cloud Computing Lab

Module	Detailed Contents	Hours
1	**Module: Remote Process Communication:** Develop a multi-client chat server application where multiple clients chat with each other concurrently. The messages sent by different clients are first communicated to the server and then the server, on behalf of the source client, communicates the messages to the appropriate destination client. **Self Learning Topics:** Other applications based on Remote process communication	**02**
2	**Module: Remote Procedure Call:** A remote procedure call is an inter process communication technique that is used for client-server-based applications. 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. **Self Learning Topics:** Other types of call semantics	**04**
3	**Module: Remote Method Invocation:** The Remote Method Invocation is an API that provides a mechanism to create distributed application in java. The client invokes methods via an interface. These methods are implemented on the server side. **Self Learning Topics:** Concept of client and server applications, remote interface, RMI registry tools	**04**

4	**Module: Remote Object Communication:** Pass remote objects from the server to the client. The client will receive the stub object (through remote interfaces) and saves it in an object variable with the same type as the remote interface. Then the client can access the actual object on the server through the variable. **Self Learning Topics:** Concept of JDBC	**0**4
5	**Module: Mutual Exclusion:** Token ring algorithm solves the mutual exclusion existing in the process communication. **Self Learning Topics:** Other algorithms of Mutual Exclusion	**0**2
6	**Module: Implementation of Cloud Computing Services:** Cloud Computing provides different services such as SaaS, PaaS, IaaS, Storage as service and many more. Storage as a Service is a business model in which a large company rents space in their storage infrastructure to a smaller company or individual. **Self Learning Topics:** Other types of Cloud Services	**0**2
7	**Module: Implementation of Identity Management using Cloud Computing concept:** The main goal of identity management is to ensure that only authenticated users are granted access to the specific applications, systems or IT environments for which they are authorized. **Self Learning Topics:** Other tools to implement the technique	**0**2
8	**Module: App Development using Cloud Computing:** Make use of various tools and techniques to develop efficient, dynamic applications. **Self Learning Topics:** Other Technique of application Development and its Complexity	**06**

*****

MODULE I

1

REMOTE PROCESS COMMUNICATION

Unit Structure

1.0 Aim
1.1 Objective
1.2 Pre-Requiste
1.3 Steps: 1
1.4 Description
1.5 Socket Programming
1.6 Socket creation
1.7 Important methods of socket class
1.8 Buffered Reader class Constructors
1.9 Buffered Reader class methods
1.10 Threads
1.11 Bibliography
1.12 Reference

EXPERIMENT 1

1.0 AIM

Write a program to develop multi-client server application where multiple clients chat with each other concurrently.

1.1 OBJECTIVE

In this experiment both client and server program runs on two different command prompt.

1.2 PRE-REQUISTE

Install JDK 8.0 or 8.1 on your system for the experiment.

1.3 STEPS: 1

To develop this experiment we are using the Thread concepts for developing a multi chat application using JDK.

1.4 DESCRIPTION

There are two ways to create thread in java.:

a) Extending the Thread Class.

b) And by implementing the Runnable interface

- Multithreading in java is a process where multiple threads are executed simultaneously.
- In a multi threaded program, the program consist of two or more process that can run threads concurrently and each process(thread) can handle a different task at the same time where it make optimal use of resources.
- The process of executing multiple threads simultaneously is known as multithreading.
- In a Single Thread Socket program where only one clients can communicate with the server.
- If you want to connect or communicate with more than one client then we have to write the code using Multithreaded Socket Programming.

Multithreaded Server Socket Program using java.

Write this code using notepad

```
import java.net.*;
import java.io.*;
public class MultithreadedSocketServer
{
  public static void main(String[] args) throws Exception
{
    try
{
      ServerSocket server=new ServerSocket(8888);
      int counter=0;
      System.out.println("Server Started ....");
      while(true)
{
        counter++;
        Socket serverClient=server.accept();  //server accept the client connection request
        System.out.println(" >> " + "Client No:" + counter + " started!");
        ServerClientThread sct = new ServerClientThread(serverClient,counter); //send  the request to a separate thread
        sct.start();
```

```
    }
   }
catch(Exception e)
{
    System.out.println(e);
   }
  }
}
```

Save the above program as MultithreadedSocketServer.java

Comiple:

javac MultithreadedSocketServer.java

Run:

Java MultithreadedSocketServer

Server Client Program:

Write this code using notepad:

This Server Client thread class handled the request independent of any other incoming requests. The following Java program is the part of Multithreaded Server Socket program.

```
class ServerClientThread extends Thread
{
  Socket serverClient;
  int clientNo;
  int squre;
  ServerClientThread(Socket inSocket,int counter)
{
    serverClient = inSocket;
    clientNo=counter;
  }
  public void run()
{
    try
{
 DataInputStream inStream = new DataInputStream(serverClient.getInputStream());
```

```
DataOutputStream outStream = new DataOutputStream(serverClient.getOutputStream());
    String clientMessage="", serverMessage="";
    while(!clientMessage.equals("bye"))
{
      clientMessage=inStream.readUTF();
      System.out.println("From Client-" +clientNo+ ": Number is
       :"+clientMessage);
      squre = Integer.parseInt(clientMessage) * Integer.parseInt(clientMessage);
      serverMessage="From Server to Client-" + clientNo + " Square of " + clientMessage + " is " +squre;
      outStream.writeUTF(serverMessage);
      outStream.flush();
    }
    inStream.close();
    outStream.close();
    serverClient.close();
  }
catch(Exception ex)
{
    System.out.println(ex);
  }
finally
{
    System.out.println("Client -" + clientNo + " exit!! ");
  }
 }
}
```

Save the above program as ServerClientThread.java

Comiple:

javac ServerClientThread .java

Run:

java ServerClientThread

Client Program

Write this code using notepad

This is the real Client program that request connection to the server. For each Client, you need to open separate console window to run the client program.

```
import java.net.*;
import java.io.*;
public class TCPClient
{
  public static void main(String[] args) throws Exception
{
  try
{
    Socket socket=new Socket("127.0.0.1",8888);
    DataInputStream inStream=new DataInputStream(socket.getInputStream());
    DataOutputStream outStream=new DataOutputStream(socket.getOutputStream());
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    String clientMessage="",serverMessage="";
    while(!clientMessage.equals("bye"))
{
      System.out.println("Enter number :");
      clientMessage=br.readLine();
      outStream.writeUTF(clientMessage);
      outStream.flush();
      serverMessage=inStream.readUTF();
      System.out.println(serverMessage);
    }
    outStream.close();
    outStream.close();
    socket.close();
  }
catch(Exception e)
{
    System.out.println(e);
```

```
  }
  }
}
```

Save the above program as TCPClient**.java**

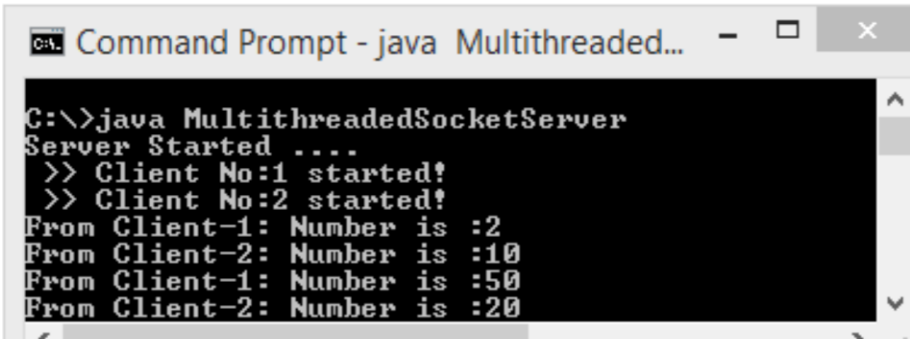
Comiple:

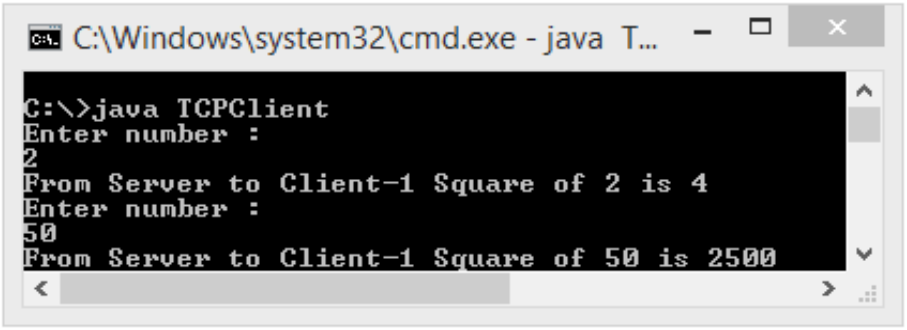
javac TCPClient **.java**

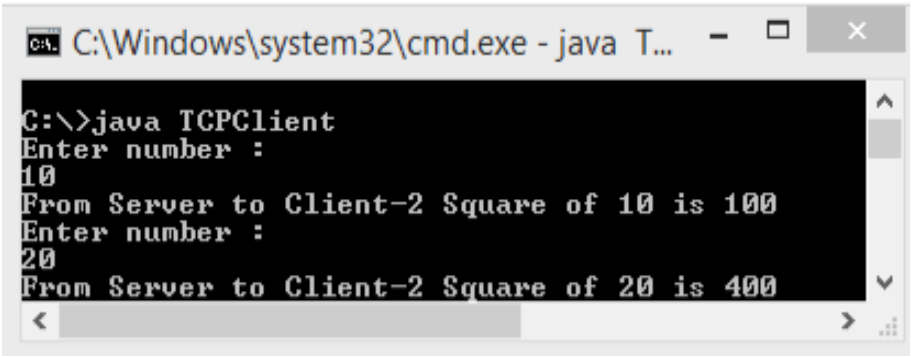
Run:

java TCPClient

OUTPUT:







1.5 SOCKET PROGRAMMING

- With the help of socket programming in java we can connect two nodes in a network that can communicate with each other.

- Where one socket listens to an IP through a particular node while the other one reaches out to the other form to establish a communication.

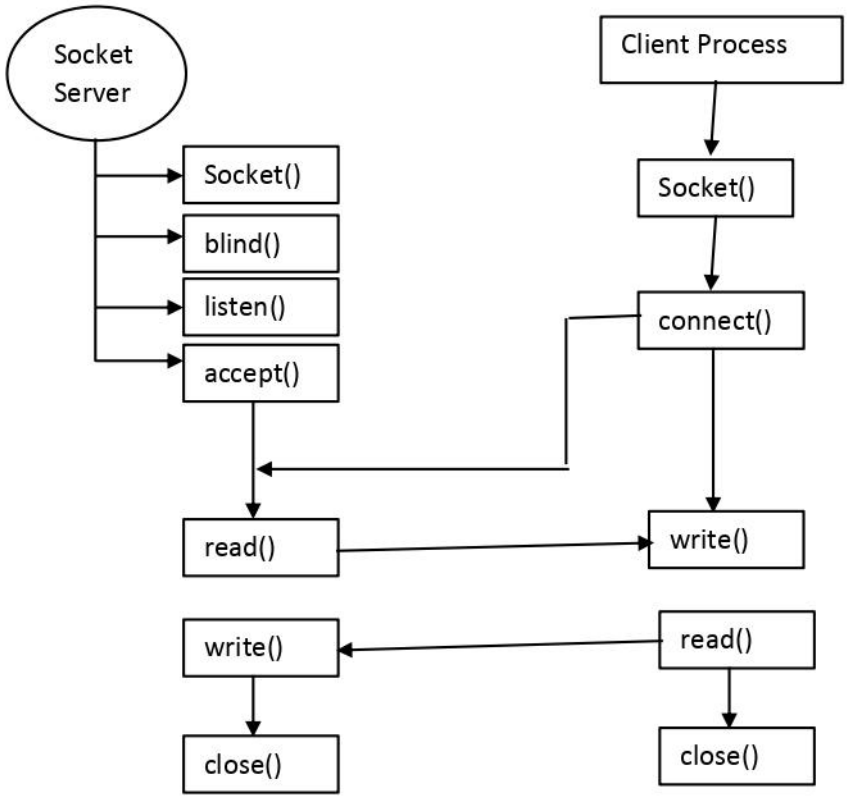


Figure (a)

1.6 SOCKET CREATION

1) Socket creation:

Syntax: int a =socket(domain,type,protocol)

Where (a) is the socket descriptor of type integer.

domain: it is used to specifies the domain for communication of type integer.

type: it is used to specifies the communication type.

protocol: it is used to specifies the protocol values to the Internet Protocol, which is 0.

And it is the same numbers which will be appear on the protocol field in the IP header of a packet.

2) setSocket: It is used to manipulate the options of the sockets which is referred by the descriptor that is a. it is an optional.

3) Bind: After the creation of a socket, the bind function binds the socket to the address and port number.

4) **Listen:** It is used to put the socket in a passive mode where the clients wait to approach to the server to make a connection.

5) **Accept:** It is used to create a new connected socket and returns a new file descriptor that is referring to that socket which is newly created.

6) **Socket connection**: It is same as socket creation.

7) **Connect**: It is used to specify the address and port number in the address. Its connects the system calls that is to be referred by the socket.

1.6 IMPORTANT METHODS OF SOCKET CLASS ARE

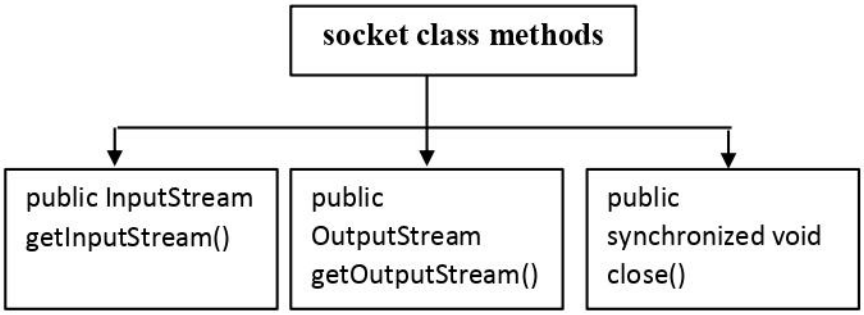
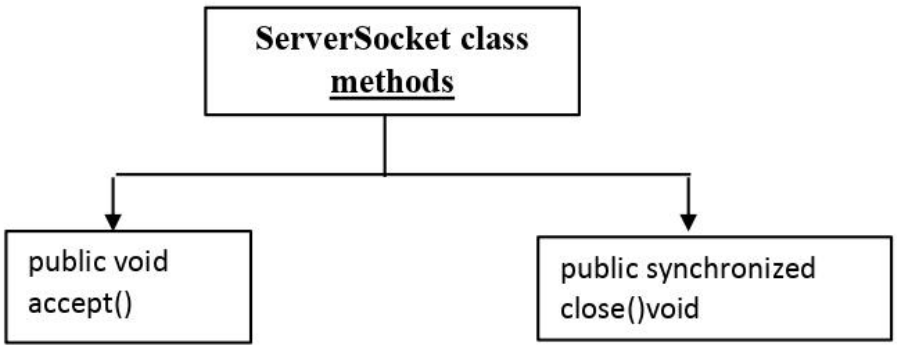


Figure (b)

Important methods of ServerSocket class are:



Figure(c)

Creating Server:

To create a server we need to create the object that is the instance of a SeverSocket class.

1. Syntax: ServerSocket ss=**new** ServerSocket(4444);
2. Socket s=ss.accept();//establishes connection and waits for the client**C reating Client:**

To create a client we need to create the instance of a socket class.

1. Syntax: Socket s=**new** Socket("localhost",4444);

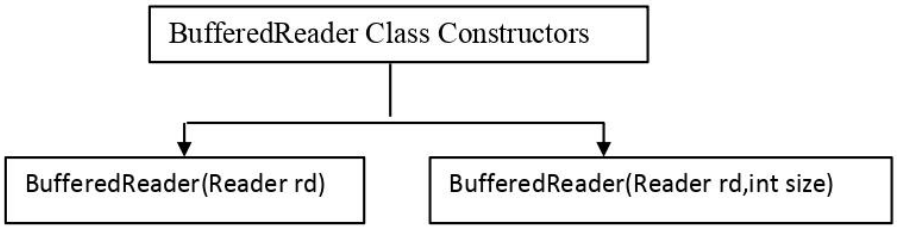
BufferedReader class:

- It is a class used to read the text from a input stream which is a character based.
- To read the text line by line we use the method readLine() method.
- It extends the Reader class.

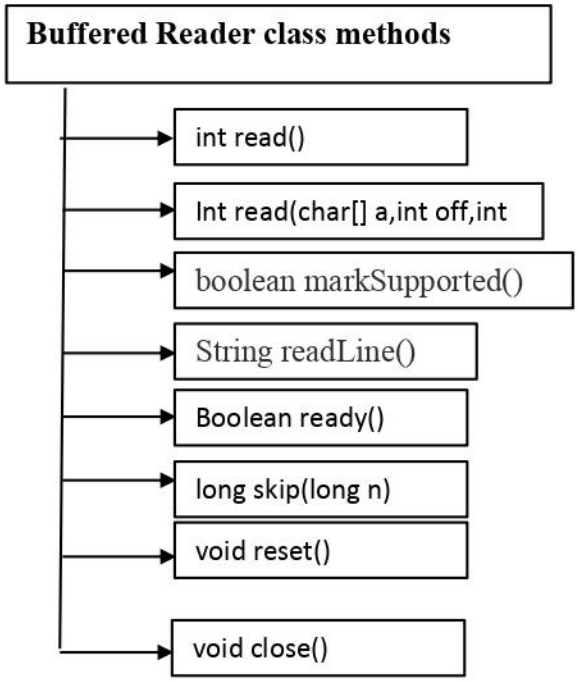
Syntax:

public class BufferedReader extends Reader

1.8 BUFFERED READER CLASS CONSTRUCTORS



1.9 BUFFERED READER CLASS METHODS:



1.10 THREADS

- A thread is a smallest unit of a process.
- It is used to achieve multitasking and multiprocessing
- It shared a common memory area.

- It is mainly used in animation and games.
- Its performs many operations together.
- It's also known as multitasking
- And thread based multitasking

Important methods of threads:

1) void start()
2) void run()
3) static void sleep()
4) static thread currentThread()
5) void join()
6) int getPriority()
7) void setPriority()
8) String getName()
9) Void setName()
10) long getID()
11) boolean isAlive()
12) static void yield()
13) void suspend()
14) void resume()
15) void stop()
16) void destroy()
17) void isinterrupted()
18) static boolean interrupted()
19) static int activeCount()
20) void checkAccess()

1.11 BIBLIOGRAPGHY

- https://www.javatpoint.com/socket-programming
- https://www.javatpoint.com/

1.12 REFERENCE

- https://www.javatpoint.com/socket-programming
- https://www.javatpoint.com/

*****

MODULE II

2

REMOTE PROCEDURE CALL

Unit Structure

2.1 Aim
2.2 Objective
2.3 Theory
2.4 Practical No-1
    2.4.1 Software Required
    2.4.2 Program
    2.4.3 Output
2.5 Practical No-2
    2.5.1 Software Required
    2.5.2Program
    2.5.3 Output
2.6 Questions

2.1 AIM

- To implement a server calculator using RPC concept.

2.2 OBJECTIVE

The Objective of this module is to make students understand the concepts of Remote Object Communication.

2.3 THEORY

A remote procedure call is an interprocess 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.

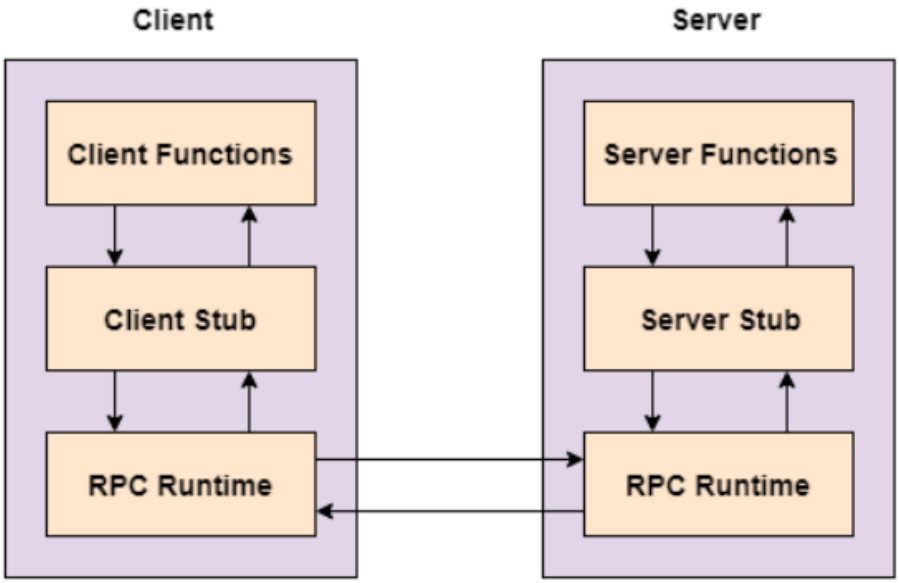


Figure 1: working of RPC

Stub: The stub is a client-side object that functions as a gateway. It is via which all outbound requests are routed. It is a client-side object that represents a distant object. The caller does the following duties when calling a method on the stub object:

1) It establishes a connection with a remote Virtual Machine (JVM), then writes and sends (marshals) the parameters to the remote Virtual Machine (JVM).

2) It sits and waits for the outcome. It reads (un-marshals) the return value or exception after receiving the result, and then returns the value to the caller.

2.4 PRACTICAL NO-1

2.4.1 Software Required:

- Java 8

2.4.2 Program:

First, we will execute server-side file.

RPCServer.java

```
import java.util.*;
import java.net.*;
class RPCServer
{
DatagramSocket ds;
DatagramPacket dp;
String str,methodName,result;
int val1,val2;
RPCServer()
{
try
{
ds=new DatagramSocket(1200);
byte b[]=new byte[4096];
while(true)
{
dp=new DatagramPacket(b,b.length);
 ds.receive(dp);
 str=new String(dp.getData(),0,dp.getLength());
 if(str.equalsIgnoreCase("q"))
{
 System.exit(1);
}
else
{
StringTokenizer st = new StringTokenizer(str," ");
int i=0;
while(st.hasMoreTokens())
{
 String token=st.nextToken();
 methodName=token;
 val1 = Integer.parseInt(st.nextToken());
 val2 = Integer.parseInt(st.nextToken());
}
}
```

```
System.out.println(str);
InetAddress ia = InetAddress.getLocalHost();
if(methodName.equalsIgnoreCase("add"))
{
result= "" + add(val1,val2);
}
else if(methodName.equalsIgnoreCase("sub"))
{
result= "" + sub(val1,val2);
}
else if(methodName.equalsIgnoreCase("mul"))
{
result= "" + mul(val1,val2);
}
else if(methodName.equalsIgnoreCase("div"))
{
result= "" + div(val1,val2);
}
byte b1[]=result.getBytes();
DatagramSocket ds1 = new DatagramSocket();
DatagramPacket dp1 = new
DatagramPacket(b1,b1.length,InetAddress.getLocalHost(), 1300);
System.out.println("result : "+result+"\n");
ds1.send(dp1);
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
public int add(int val1, int val2)
{
return val1+val2;
}
```

```
public int sub(int val3, int val4)
{
return val3-val4;
}
public int mul(int val3, int val4)
{
return val3*val4;
}
public int div(int val3, int val4)
{
return val3/val4;
}
public static void main(String[] args)
{
new RPCServer();
}
}
```

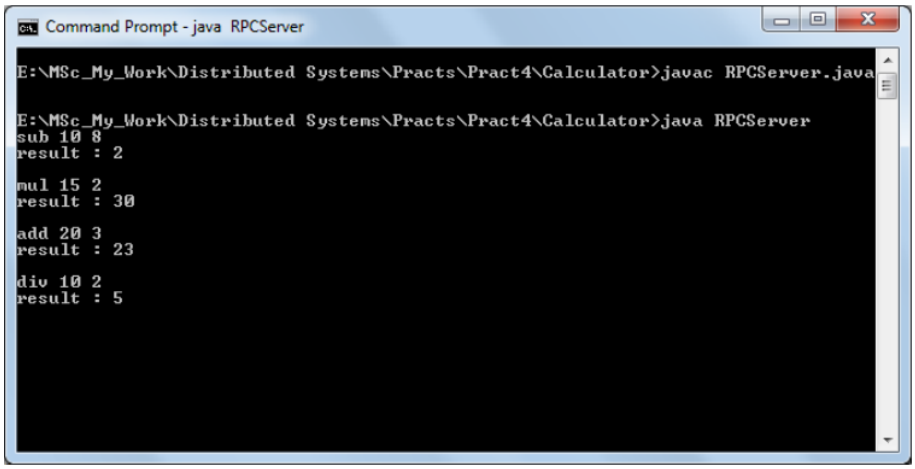
Client-side java file:

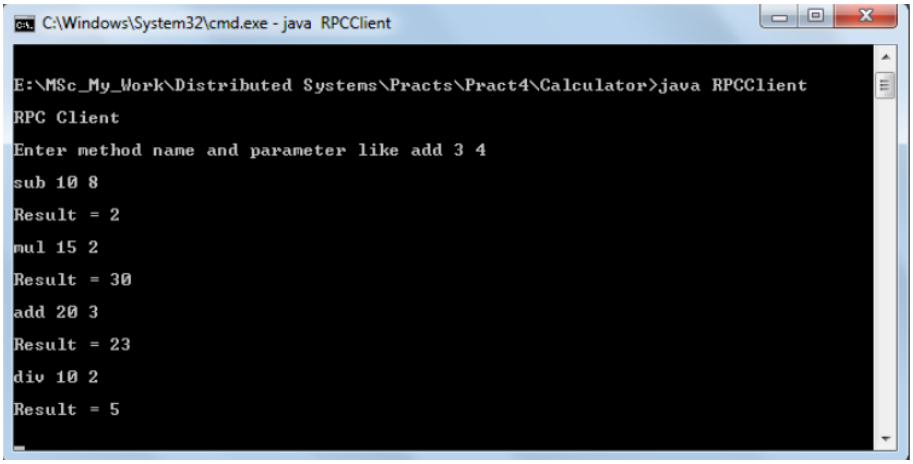
RPCClient.java

```
import java.io.*;
import java.net.*;
class RPCClient
{
RPCClient()
{
try
{
InetAddress ia = InetAddress.getLocalHost();
DatagramSocket ds = new DatagramSocket();
DatagramSocket ds1 = new DatagramSocket(1300);
System.out.println("\nRPC Client\n");
System.out.println("Enter method name and parameter like add 3
4\n");
while (true)
{
```

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
String str = br.readLine();
byte b[] = str.getBytes();
DatagramPacket dp = new
DatagramPacket(b,b.length,ia,1200);
ds.send(dp);
dp = new DatagramPacket(b,b.length);
ds1.receive(dp);
String s = new String(dp.getData(),0,dp.getLength());
System.out.println("\nResult = " + s + "\n");
}
}
catch (Exception e)
{
e.printStackTrace();
}
}
public static void main(String[] args)
{
new RPCClient();
}
}
```

2.4.3 Output:





2.5 PRACTICAL NO-2

2.5.1 Software Required:

- Java 8

2.5.2Program:

Server-side program:

```
import java.net.*;
import java.io.*;
import java.util.*;
class DateServer
{
   public static void main(String args[]) throws Exception
   {
      ServerSocket s=new ServerSocket(5217);
      while(true)
      {
         System.out.println("Waiting For Connection ...");
         Socket soc=s.accept();
         DataOutputStream out=new
DataOutputStream(soc.getOutputStream());
         out.writeBytes("Server Date: " + (new Date()).toString() + "\n");
         out.close();
         soc.close();
      }
   }
}
```

Client side:

```
import java.io.*;
import java.net.*;
class DateClient
{
   public static void main(String args[]) throws Exception
   {
      Socket soc=new Socket(InetAddress.getLocalHost(),5217);
      BufferedReader in=new BufferedReader(new
InputStreamReader(soc.getInputStream() ));
      System.out.println(in.readLine());
   }
}
```

2.5.3 Output:

First compile the client code on console and then compile the server code on different console.

Run the server code, after that client code. Now the client console shows the time and date of server machine.

Server Date: Fri Dec 16 17:05:42 IST 2016

2.6 QUESTIONS

Q.1) What is Remote Procedure call?

Q.2) What is Client Stub in remote procedure call?

Q.3) What is Server Stub in remote procedure call?

Q.4) What is the sequence of events during remote procedure call?

*****

MODULE III

3

REMOTE METHOD INVOCATION

Unit Structure

3.0 Aim

3.1 Objective

3.2 Pre-Requisite

EXPERIMENT 1

3.0 AIM

Write a client-server program which displays the server machine's date and time on the client machine.

3.1 OBJECTIVE

In this example both client and server program run on different command prompt. The time and date of server machine's is displayed on client machine.

3.2 PRE-REQUISITE

Make sure that JDK 1.8 is installed on your system for all Experiment.

STEPS 1: Create file name DateClient.java

Type following code

```
import java.io.*;
import java.net.*;
class DateClient
{
  public static void main(String args[]) throws Exception
  {
    Socket soc=new Socket(InetAddress.getLocalHost(),5217);
    BufferedReader in=new BufferedReader(new
InputStreamReader(soc.getInputStream() ));
    System.out.println(in.readLine());
  }
}
```

STEPS 2: Create file name DateServer.java

```
import java.net.*;
import java.io.*;
import java.util.*;
class DateServer
{
   public static void main(String args[]) throws Exception
   {
      ServerSocket s=new ServerSocket(5217);
      while(true)
      {
         System.out.println("Waiting For Connection ...");
         Socket soc=s.accept();
         DataOutputStream out=new
DataOutputStream(soc.getOutputStream());
         out.writeBytes("Server Date: " + (new Date()).toString() + "\n");
         out.close();
         soc.close();

      }
   }
}
```

STEP 3:

Output:

First compile the client code on console and then compile the server code on different console.

- Run the server code, after that client code. Now the client console shows the time and date of server machine.

Server Date: Sat June 03 10:05:42 IST 2022

EXPERIMENT 2

AIM

Demonstrate an sample RMI Java application.

OBJECTIVE

In this example demonstrating **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

PRE-REQUISITE

Make sure that JDK 1.8 is installed on your system For all Experiment.

DESCRIPTION

To write an RMI Java application, you would have to follow the steps given below:

- Define the remote interface
- Develop the implementation class (remote object)
- Develop the server program
- Develop the client program
- Compile the application
- Execute the application

Defining the Remote Interface:

A remote interface provides the description of all the methods of a particular remote object. The client communicates with this remote interface.

To create a remote interface:

- Create an interface that extends the predefined interface **Remote** which belongs to the package.
- Declare all the business methods that can be invoked by the client in this interface.
- Since there is a chance of network issues during remote calls, an exception named **RemoteException** may occur; throw it.

Following is an example of a remote interface. Here we have defined an interface with the name **Hello** and it has a method called **printMsg()**.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
// Creating Remote interface for our application
public interface Hello extends Remote {
  void printMsg() throws RemoteException;
}
```

Developing the Implementation Class (Remote Object):

We need to implement the remote interface created in the earlier step. (We can write an implementation class separately or we can directly make the server program implement this interface.)

To develop an implementation class:

- Implement the interface created in the previous step.
- Provide implementation to all the abstract methods of the remote interface.

Following is an implementation class. Here, we have created a class named **ImplExample** and implemented the interface **Hello** created in the previous step and provided **body** for this method which prints a message.

```
// Implementing the remote interface
public class ImplExample implements Hello {

  // Implementing the interface method
  public void printMsg() {
    System.out.println("This is an example RMI program");
  }
}
```

Developing the Server Program:

An RMI server program should implement the remote interface or extend the implementation class. Here, we should create a remote object and bind it to the **RMIregistry**.

To develop a server program:

- Create a client class from where you want invoke the remote object.
- **Create a remote object** by instantiating the implementation class as shown below.
- Export the remote object using the method **exportObject()** of the class named **UnicastRemoteObject** which belongs to the package **java.rmi.server**.

- Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- Bind the remote object created to the registry using the **bind()** method of the class named **Registry**. To this method, pass a string representing the bind name and the object exported, as parameters.

Following is an example of an RMI server program:

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server extends ImplExample {
  public Server() {}
  public static void main(String args[]) {
    try {
      // Instantiating the implementation class
      ImplExample obj = new ImplExample();

      // Exporting the object of implementation class
      // (here we are exporting the remote object to the stub)
      Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

      // Binding the remote object (stub) in the registry
      Registry registry = LocateRegistry.getRegistry();

      registry.bind("Hello", stub);
      System.err.println("Server ready");
    } catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

Developing the Client Program:

Write a client program in it, fetch the remote object and invoke the required method using this object.

To develop a client program:

- Create a client class from where your intended to invoke the remote object.
- Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- Fetch the object from the registry using the method **lookup()** of the class **Registry** which belongs to the package **java.rmi.registry**.
- To this method, you need to pass a string value representing the bind name as a parameter. This will return you the remote object.
- The lookup() returns an object of type remote, down cast it to the type Hello.
- Finally invoke the required method using the obtained remote object.

Following is an example of an RMI client program.

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
  private Client() {}
  public static void main(String[] args) {
    try {
      // Getting the registry
      Registry registry = LocateRegistry.getRegistry(null);

      // Looking up the registry for the remote object
      Hello stub = (Hello) registry.lookup("Hello");

      // Calling the remote method using the obtained object
      stub.printMsg();

      // System.out.println("Remote method invoked");
    } catch (Exception e) {
```

```
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Compiling the Application:

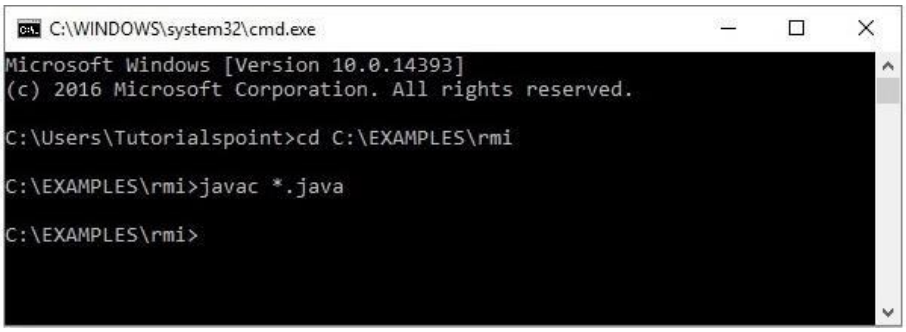
To compile the application:

- Compile the Remote interface.
- Compile the implementation class.
- Compile the server program.
- Compile the client program.

Or,

Open the folder where you have stored all the programs and compile all the Java files as shown below.

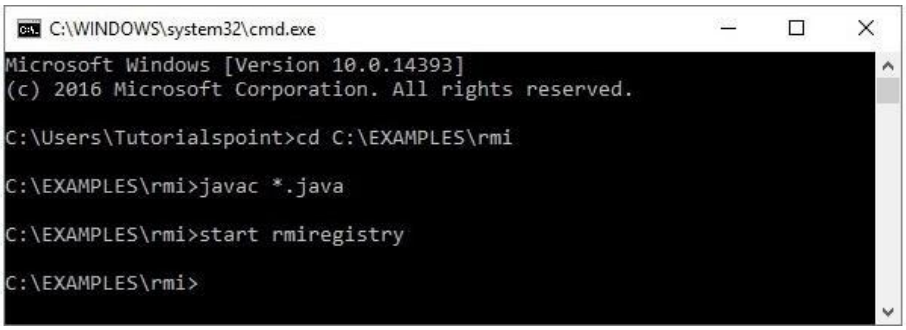
Javac *.java



Executing the Application:

Step 1: Start the **rmi** registry using the following command.

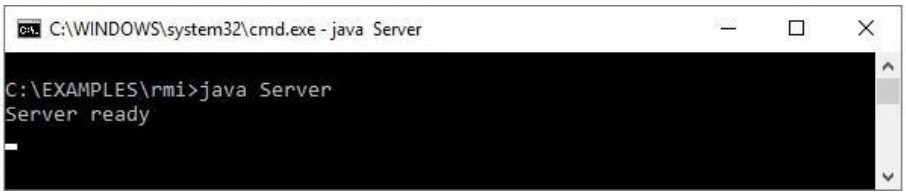
Start rmiregistry



This will start an **rmi** registry on a separate window as shown below.

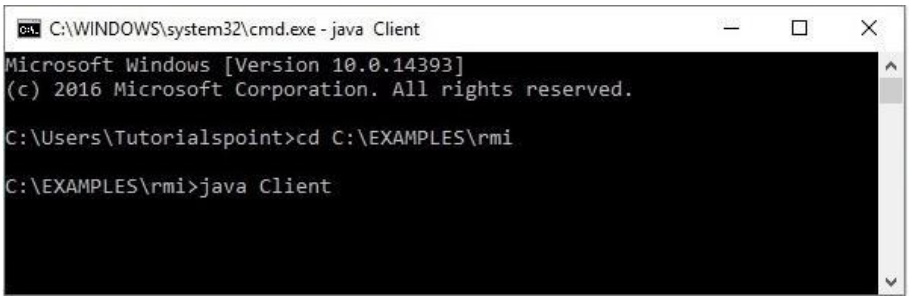
Step 2: Run the server class file as shown below.

Java Server

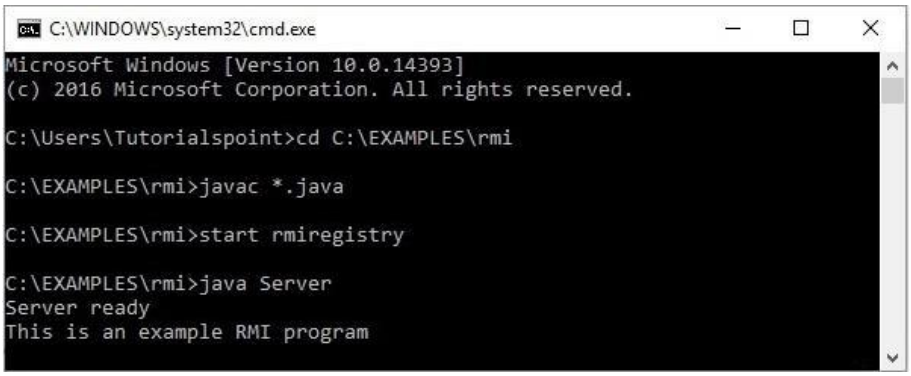


Step 3: Run the client class file as shown below.

Java Client



Verification: As soon you start the client, you would see the following output in the server.



EXPERIMENT 3

AIM

Demonstrate an how a client program can retrieve the records of a table in MySQL database residing on the server.

OBJECTIVE

In this example using **Remote Method Invocation**. How a client program can retrieve the records of a table in MySQL database residing on the server.

PRE-REQUISITE

Make sure that JDK 1.8 is installed on your system for all Experiment.

DESCRIPTION

Assume we have a table named **student_data** in the database **details** as shown below.

+----+--------+--------+------------+--------------------+

| ID | NAME | BRANCH | PERCENTAGE | EMAIL |

+----+--------+--------+------------+--------------------+

| 1 | Ram | IT | 85 | ram123@gmail.com |

| 2 | Rahim | EEE | 95 | rahim123@gmail.com |

| 3 | Robert | ECE | 90 | robert123@gmail.com |

+----+--------+--------+------------+--------------------+

Assume the name of the user is **myuser** and its password is **password**.

Creating a Student Class

Create a **Student** class with **setter** and **getter** methods as shown below.

```
public class Student implements java.io.Serializable {
  private int id, percent;
  private String name, branch, email;

  public int getId() {
    return id;
  }
  public String getName() {
    return name;
  }
  public String getBranch() {
    return branch;
  }
```

```
  public int getPercent() {
    return percent;
  }
  public String getEmail() {
    return email;
  }
  public void setID(int id) {
    this.id = id;
  }
  public void setName(String name) {
    this.name = name;
  }
  public void setBranch(String branch) {
    this.branch = branch;
  }
  public void setPercent(int percent) {
    this.percent = percent;
  }
  public void setEmail(String email) {
    this.email = email;
  }
}
```

Defining the Remote Interface:

Define the remote interface. Here, we are defining a remote interface named **Hello** with a method named **getStudents ()** in it. This method returns a list which contains the object of the class **Student**.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.*;

// Creating Remote interface for our application
public interface Hello extends Remote {
  public List<Student> getStudents() throws Exception;
}
```

Developing the Implementation Class:

Create a class and implement the above created **interface.**

Here we are implementing the **getStudents()** method of the **Remote interface**. When you invoke this method, it retrieves the records of a table named **student_data**. Sets these values to the Student class using its setter methods, adds it to a list object and returns that list.

```
import java.sql.*;
import java.util.*;

// Implementing the remote interface
public class ImplExample implements Hello {

  // Implementing the interface method
  public List<Student> getStudents() throws Exception {
    List<Student> list = new ArrayList<Student>();

    // JDBC driver name and database URL
    String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    String DB_URL = "jdbc:mysql://localhost:3306/details";

    // Database credentials
    String USER = "myuser";
    String PASS = "password";

    Connection conn = null;
    Statement stmt = null;

    //Register JDBC driver
    Class.forName("com.mysql.jdbc.Driver");

    //Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");
```

```
    //Execute a query
    System.out.println("Creating statement...");

    stmt = conn.createStatement();
    String sql = "SELECT * FROM student_data";
    ResultSet rs = stmt.executeQuery(sql);

    //Extract data from result set
    while(rs.next()) {
      // Retrieve by column name
      int id  = rs.getInt("id");

      String name = rs.getString("name");
      String branch = rs.getString("branch");

      int percent = rs.getInt("percentage");
      String email = rs.getString("email");

      // Setting the values
      Student student = new Student();
      student.setID(id);
      student.setName(name);
      student.setBranch(branch);
      student.setPercent(percent);
      student.setEmail(email);
      list.add(student);
    }
    rs.close();
    return list;
  }
}
```

Server Program:

An RMI server program should implement the remote interface or extend the implementation class. Here, we should create a remote object and bind it to the **RMI registry**.

Following is the server program of this application. Here, we will extend the above created class, create a remote object and register it to the RMI registry with the bind name **hello**.

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server extends ImplExample {
  public Server() {}
  public static void main(String args[]) {
    try {
      // Instantiating the implementation class
      ImplExample obj = new ImplExample();

      // Exporting the object of implementation class (
        here we are exporting the remote object to the stub)
      Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

      // Binding the remote object (stub) in the registry
      Registry registry = LocateRegistry.getRegistry();

      registry.bind("Hello", stub);
      System.err.println("Server ready");
    } catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

Client Program:

Following is the client program of this application. Here, we are fetching the remote object and invoking the method named **getStudents()**. It retrieves the records of the table from the list object and displays them.

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;

public class Client {
  private Client() {}
  public static void main(String[] args)throws Exception {
    try {
      // Getting the registry
      Registry registry = LocateRegistry.getRegistry(null);

      // Looking up the registry for the remote object
      Hello stub = (Hello) registry.lookup("Hello");

      // Calling the remote method using the obtained object
      List<Student> list = (List)stub.getStudents();
      for (Student s:list)v {

        // System.out.println("bc "+s.getBranch());
        System.out.println("ID: " + s.getId());
        System.out.println("name: " + s.getName());
        System.out.println("branch: " + s.getBranch());
        System.out.println("percent: " + s.getPercent());
        System.out.println("email: " + s.getEmail());
      }
    // System.out.println(list);
    } catch (Exception e) {
      System.err.println("Client exception: " + e.toString());
```

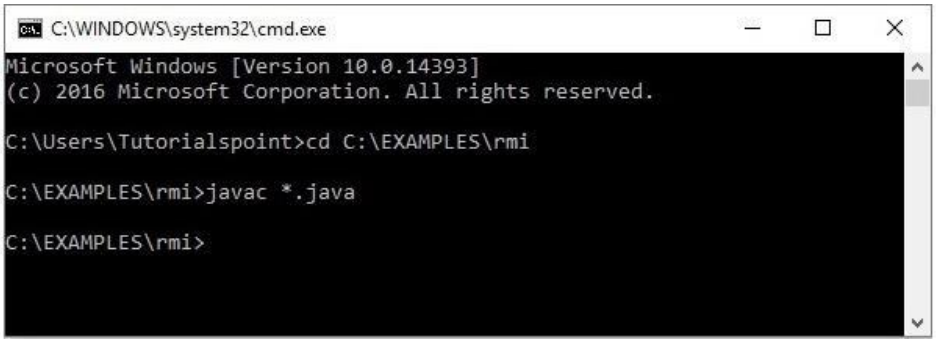
```
            e.printStackTrace();
        }
    }
}
```

Steps to Run the Example:

Following are the steps to run our RMI Example.

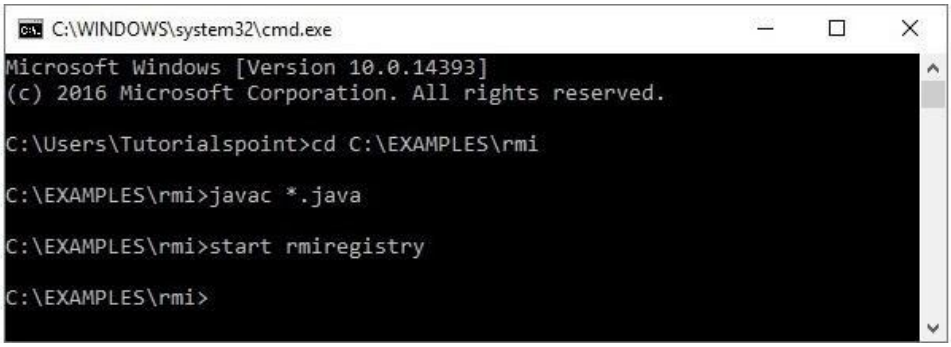
Step 1: Open the folder where you have stored all the programs and compile all the Java files as shown below.

Javac *.java

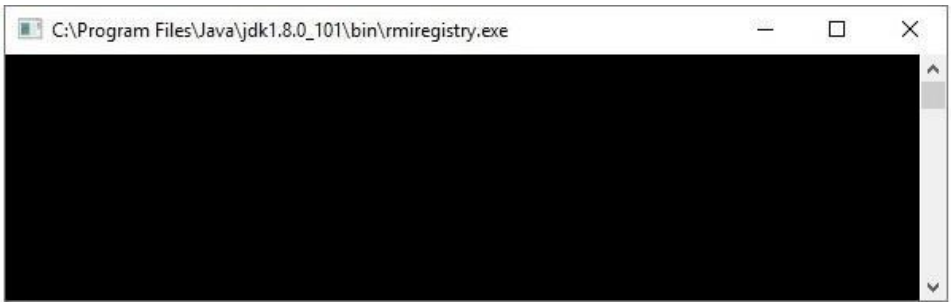


Step 2: Start the **rmi** registry using the following command.

start rmiregistry

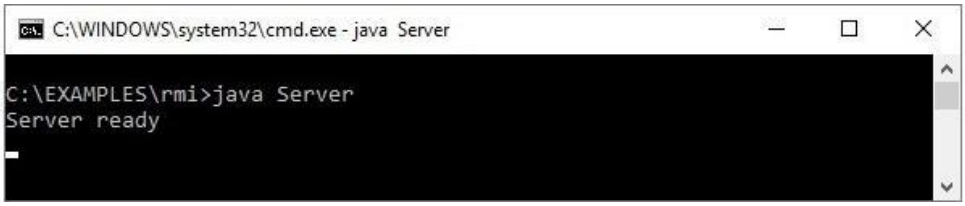


This will start an **rmi** registry on a separate window as shown below.



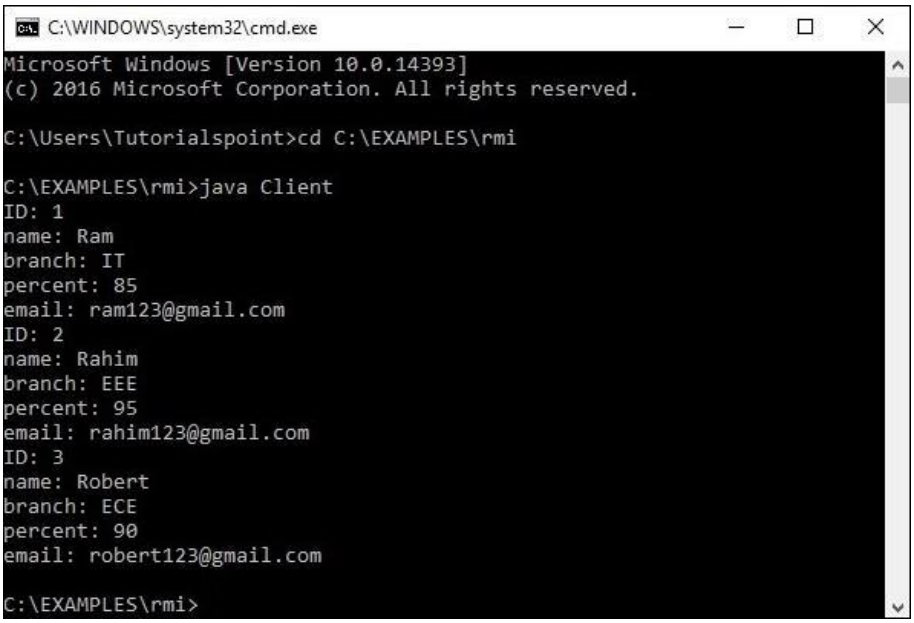
Step 3: Run the server class file as shown below.

Java Server



Step 4: Run the client class file as shown below.

java Client



EXPERIMENT 4

AIM

Demonstrate an how a client should provide an equation to the server through an interface. The server will solve the expression given by the client.

OBJECTIVE

In this example using **Remote Method Invocation,** how a client should provide an equation to the server through an interface. The server will solve the expression given by the client

PRE-REQUISITE

Make sure that JDK 1.8 is installed on your system for all Experiment.

DESCRIPTION

In this example, we have followed all the 6 steps to create and run the rmi application. The client application need only two files, remote interface and client application. In the rmi application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.

1) create the remote interface:

For creating the remote interface, extend the Remote interface and declare the RemoteException with all the methods of the remote interface. Here, we are creating a remote interface that extends the Remote interface. There is only one method named add() and it declares RemoteException.

```
import java.rmi.*;

public interface Adder extends Remote{

public int add(int x,int y)throws RemoteException;

}
```

2) Provide the implementation of the remote interface:

Now provide the implementation of the remote interface. For providing the implementation of the Remote interface, we need to

- Either extend the UnicastRemoteObject class,
- or use the exportObject() method of the UnicastRemoteObject class

In case, you extend the UnicastRemoteObject class, you must define a constructor that declares RemoteException.

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Ad
der{
AdderRemote()throws RemoteException{
super();
}
public int add(int x,int y){return x+y;}
}
```

3) create the stub and skeleton objects using the rmic tool:

Next step is to create stub and skeleton objects using the rmi compiler. The rmic tool invokes the RMI compiler and creates stub and skeleton objects.

```
rmic AdderRemote
```

4) Start the registry service by the rmiregistry tool:

Now start the registry service by using the rmiregistry tool. If you don't specify the port number, it uses a default port number. In this example, we are using the port number 5000.

rmiregistry 5000

5) Create and run the server application:

Now rmi services need to be hosted in a server process. The Naming class provides methods to get and store the remote object.

In this example, we are binding the remote object by the name sonoo.

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[]){
try{
Adder stub=new AdderRemote();
Naming.rebind("rmi://localhost:5000/sonoo",stub);
}catch(Exception e){System.out.println(e);}
}
}
```

6) Create and run the client application:

At the client we are getting the stub object by the lookup() method of the Naming class and invoking the method on this object. In this example, we are running the server and client applications, in the same machine so we are using localhost. If you want to access the remote object from another machine, change the localhost to the host name (or IP address) where the remote object is located.

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/sonoo");
System.out.println(stub.add(34,4));
}catch(Exception e){}
}
}
```

For running this rmi example:

1) compile all the java files:

javac *.java

2) **create stub and skeleton object by rmic tool**

   rmic AdderRemote

3) **start rmi registry in one command prompt**

   rmiregistry 5000

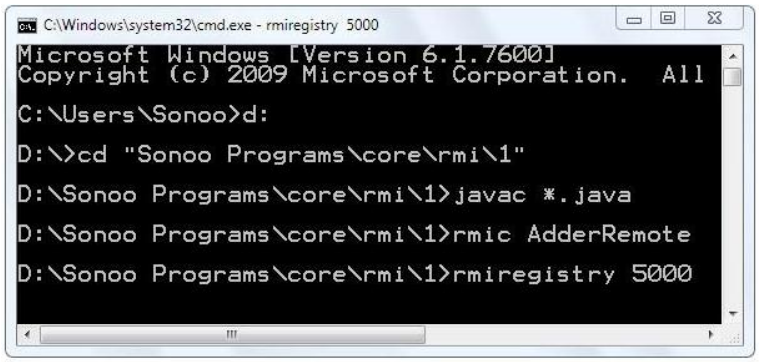
4) **start the server in another command prompt:**

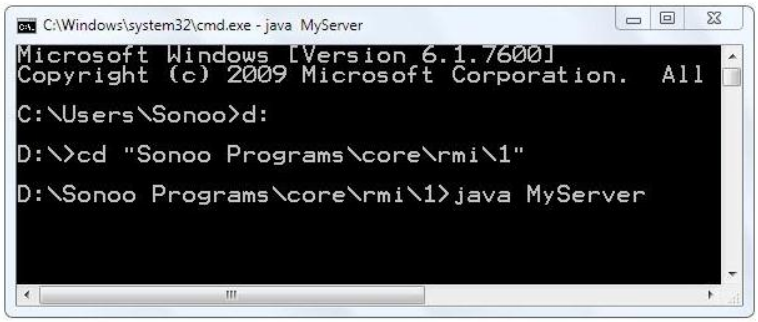
   java MyServer

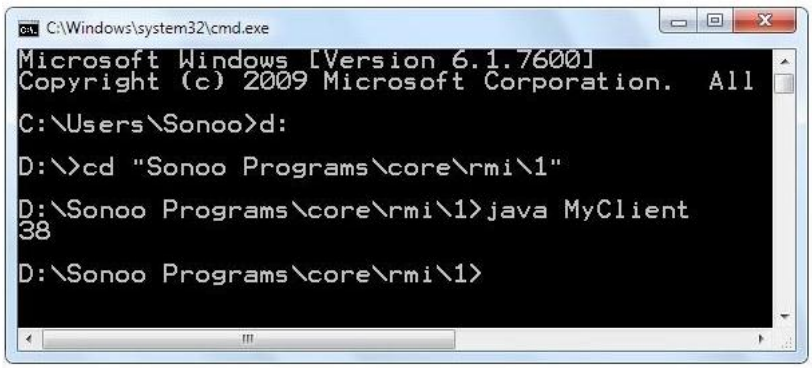
5) **start the client application in another command prompt**

   java MyClient

Output of this RMI example:







*****

MODULE IV

4

REMOTE OBJECT COMMUNICATION

Unit Structure

4.1 Aim
4.2 Objective
4.3 Theory
4.4 Practical No-1
    4.4.1 Software Required
    4.4.2 Program
    4.4.3 Procedure to execute the program
4.5 Practical No-2
    4.5.1 Software Required
    4.5.2Program
    4.5.3 Procedure to execute the program
4.6 Questions
4.7 Self Learning Topic

4.1 AIM

Ceating RMI Database Application.

4.2 OBJECTIVE

The Objective of this module is to make students understand the concepts of Remote Object Communication.

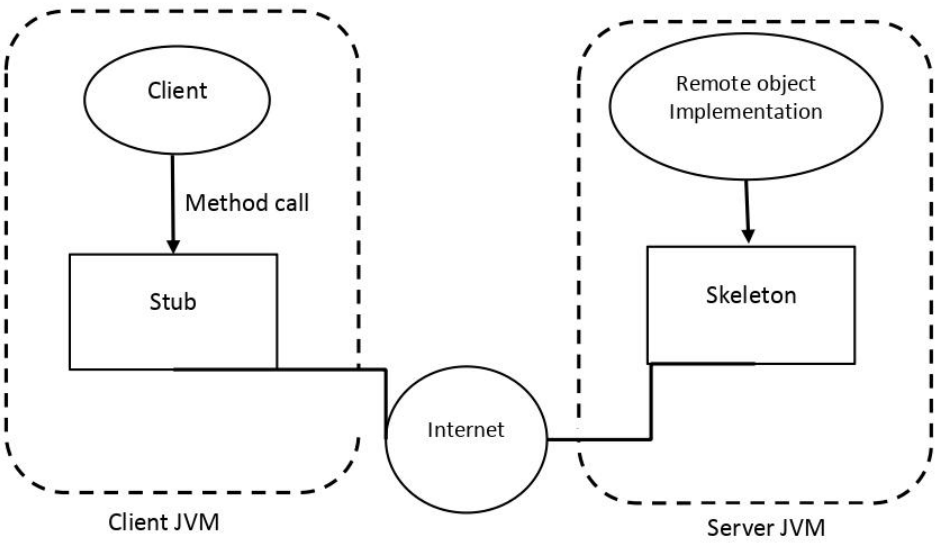
4.3 THEORY

Remote object communication is possible via RMI (Remote Method Invocation).

A remote object is an object whose method can be invoked from another JVM.

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

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



RMI uses stub and skeleton object for communication with the remote object.

Stub:

The stub is an object that acts as a gateway at the client side. All the outgoing requests are passed 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:

a) It starts a connection with remote Virtual Machine (JVM),

b) It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),

c) It waits for the result.

d) After the result is received, it reads (unmarshals) the return value or exception, and

e) It finally, returns the value to the caller.

Skeleton:

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

a) It reads the parameter for the remote method.

b) It invokes the method on the actual remote object, and

c) It writes and transmits (marshals) the result to the caller.

RMI Registry:

RMI registry is a namespace where all server objects are placed. Each time the server creates an object, it registers this object with the

RMIregistry using bind () or ebind () methods. These objects are registered using a unique name known as **bind name**.

To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name using lookup () method.

4.4 PRACTICAL - NO 1

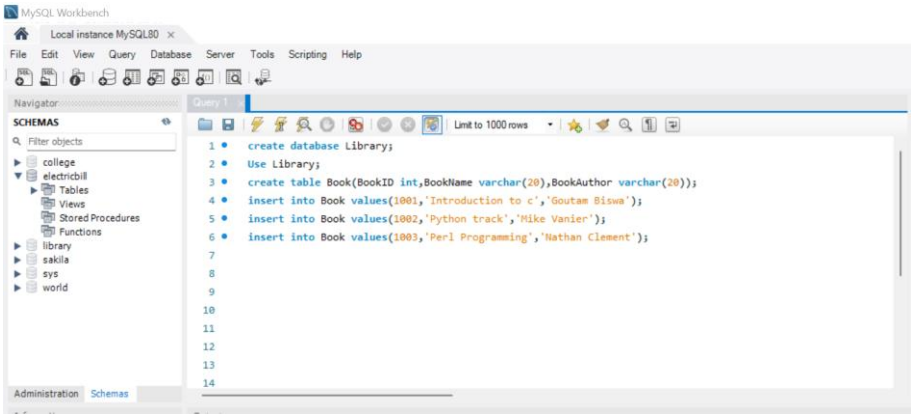
Using MySQL create Library database. Create table Book (Book_id,

Book_name, Book_author) and retrieve the Book information from Library database using Remote Object Communication concept.

4.4.1 Software Required:

- Java 8
- MySQL 8.0

4.4.2 Program:



Before starting with coding in java,create Libray database in MySql.In Library Database Create Book(Book_id,Book_name, Book_author) table.

Note: In this practical we are going to create the following java classes:

1. Library.java
2. Hello.java (Remote Interface)
3. ImplExample.java (Implementation class)
4. Client.java
5. Server.java

1) Creating a Library Class:

Create a Library class with setter and getter methods as shown below.

Library.java

```
public class Library implements java.io.Serializable {
  private int BookID;
  private String BookName,BookAuthor;

  public int getBookId() {
    return BookID;
  }
  public String getBookName() {
    return BookName;
  }
  public String getBookAuthor() {
    return BookAuthor;
  }

  public void setBookID(int BookID) {
    this.BookID =BookID;
  }
  public void setBookName(String BookName) {
    this.BookName =BookName;
  }
  public void setBookAuthor(String BookAuthor) {
    this.BookAuthor = BookAuthor;
  }

}
```

2) Defining the Remote Interface:

Define the remote interface. Here, we are defining a remote interface named Hello with a method named getBookInfo () in it. This method returns a list which contains the object of the class Library**.**

Hello.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.*;
```

```
// Creating Remote interface for our application
public interface Hello extends Remote
{
  public List<Library> getBookInfo() throws Exception;
}
```

3) Developing the Implementation Class (Remote Object):

Create a class and implement the above created interface.

Here we are implementing the getBookInfo() method of the Remote interface. When you invoke this method, it retrieves the records of a table named Book. Sets these values to the Library class using its setter methods, adds it to a list object and returns that list.

ImplExample.java

```
import java.sql. *;
import java.util.*;

// Implementing the remote interface
public class ImplExample implements Hello {

  // Implementing the interface method
  public List<Library> getBookInfo() throws Exception {
    List<Library> list = new ArrayList<Library>();

    // JDBC driver name and database URL
    String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    String DB_URL = "jdbc:mysql://localhost:3306/Library";

    // Database credentials
    String USER = "root";
    String PASS = "system";

    Connection conn = null;
    Statement stmt = null;

    //Register JDBC driver
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance ();
```

```
        //Open a connection
        System.out.println("Connecting to a selected database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database successfully...");

        //Execute a query
        System.out.println("Creating statement...");

        stmt = conn.createStatement();
        String sql = "SELECT * FROM Book";
        ResultSet rs = stmt.executeQuery(sql);

        //Extract data from result set
        while(rs.next()) {
          // Retrieve by column name
          int id  = rs.getInt("BookID");

          String name = rs.getString("BookName");
          String author = rs.getString("BookAuthor");

          // Setting the values
          Library info = new Library();
          info.setBookID(id);
          info.setBookName(name);
          info.setBookAuthor(author);
          list.add(info);
        }
        rs.close();
        return list;
      }
}
```

4) Server Program:

An RMI server program should implement the remote interface or extend the implementation class. Here, we should create a remote object and bind it to the RMI registry.

Following is the server program of this application. Here, we will extend the above created class, create a remote object and register it to the RMI registry with the bind name bookinfo.

In this program server is using port no:6666,we can use any available port.

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;


public class Server extends ImplExample {
  public Server() {}
  public static void main(String args[]) {
    try {
      // Instantiating the implementation class
      ImplExample obj = new ImplExample();


      // Exporting the object of implementation class (here we are exporting the remote object to the stub)
      Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);


      // Binding the remote object (stub) in the registry
Registry registry = LocateRegistry.createRegistry(6666);
   registry.rebind("bookinfo",stub);


      System.err.println("Server ready");
    } catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

5) Client Program:

Following is the client program of this application. Here, we are fetching the remote object and invoking the method named **getBookInfo()**. It retrieves the records of the table from the list object and displays them.

In this program, LocateRegistry.getRegistry() method is taking two parameter, first is the ip address of the server and second is the port no on which server is listening.

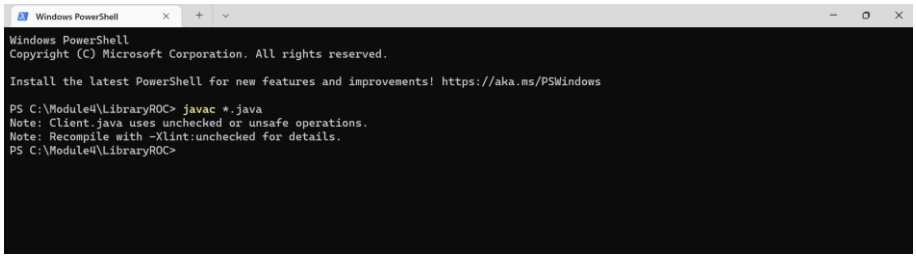
Client.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;

public class Client {
  private Client() {}
  public static void main(String[] args)throws Exception {
    try {
      // Getting the registry
      Registry registry = LocateRegistry.getRegistry("192.168.0.101",6666);

      // Looking up the registry for the remote object
      Hello stub = (Hello) registry.lookup("bookinfo");

      // Calling the remote method using the obtained object
      List<Library> list = (List)stub.getBookInfo();
      for (Library l:list)
{
        System.out.println("Book ID: " + l.getBookId());
        System.out.println("Book Name: " + l.getBookName());
        System.out.println("Book Author: " + l.getBookAuthor());

        System.out.println("----------------------------------------------");
      }

    } catch (Exception e) {
      System.err.println("Client exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

4.4.3 Procedure to execute the program:

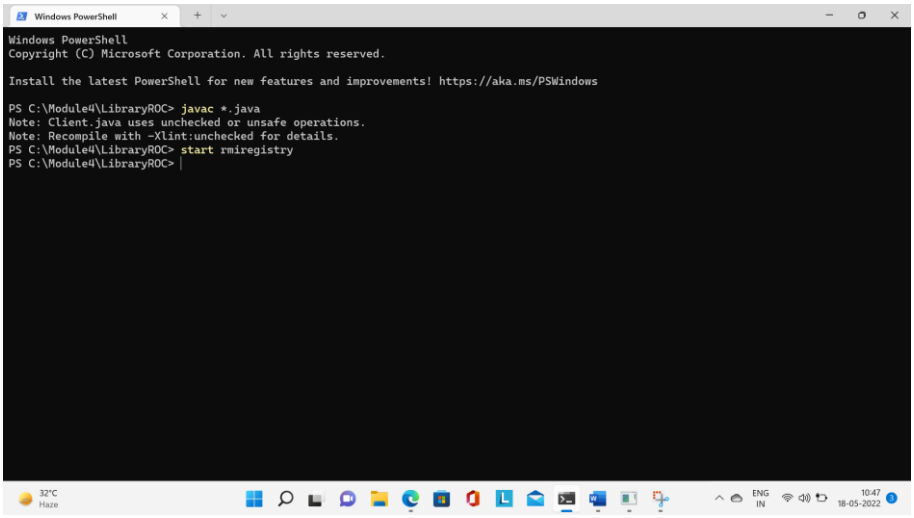
Step1: Open the command prompt and go to the folder where you have save all the java files and compile all the files by using the following command:

>javac *.java;

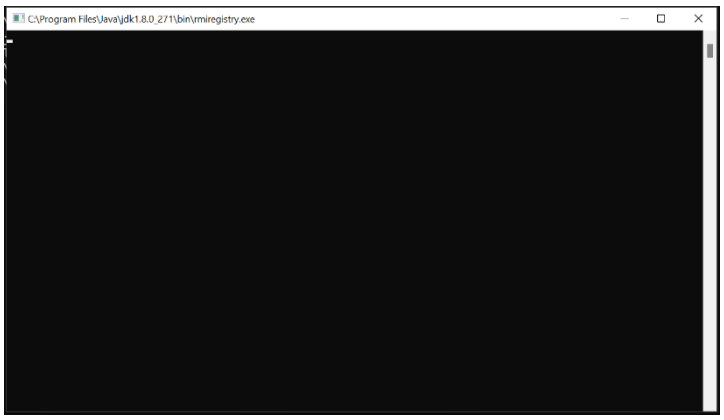


Step2: Start the rmi registry using the following command.

>start rmiregistry

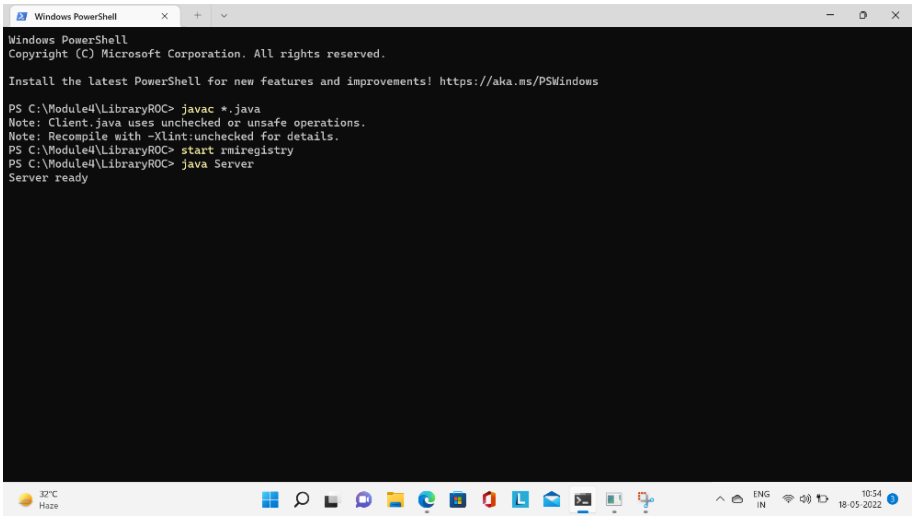


once you execute start rmiregistry command one more command prompt will be open as shown below:



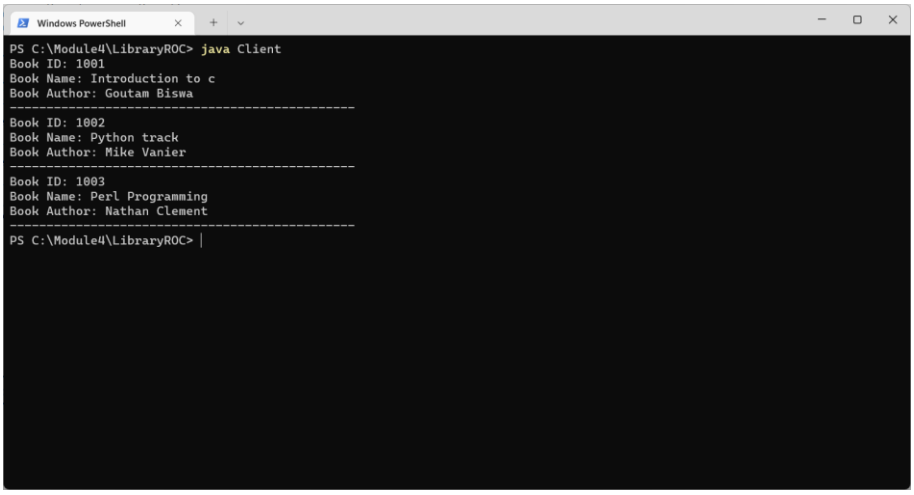
Step3: Run the server class file by the following command:

>java Server



Step 4: Run the client class file on new command prompt if running client and server on same machine as shown below.

>java Client



4.5 PRACTICAL - NO2

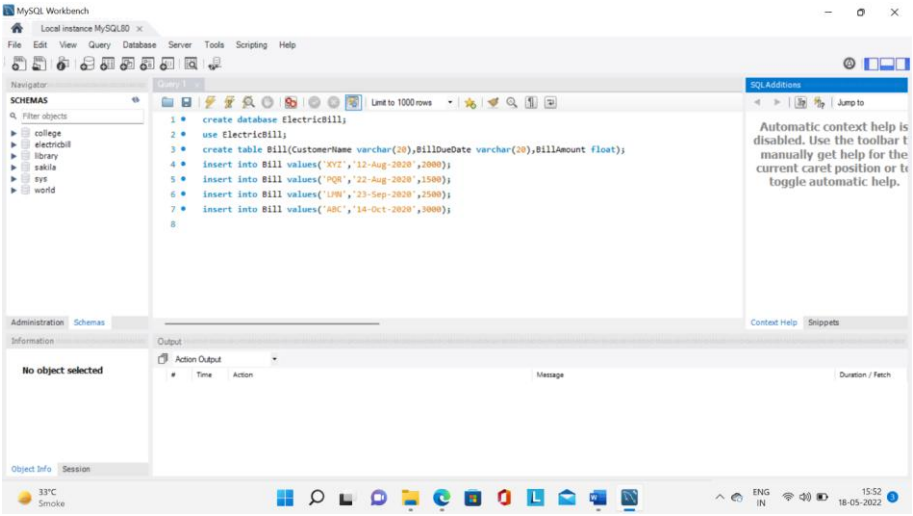
Using MySQL create ElecrticBill database. Create table Bill(consumer_name, bill_due_date, bill_amount) and retrieve the Bill information from the ElecrticBill database using Remote Object Communication concept.

4.5.1 Software Required:

- Java 8
- MySQL 8.0

4.5.2 Program:

Before starting with codingin java, create ElectricBill database in MySql.In ElectricBill Database Create Bill(ConsumerName, BillDuedate, billamount)table.



Note: In this practical we are going to create the following java classes:

1. Electric.java
2. Hello.java (Remote Interface)
3. ImplExample.java (Implementation class)
4. Client.java
5. Server.java

1) Creating a Electric Class:

Create a Electric class with setter and getter methods as shown below.

Electric.java

```
public class Electric implements java.io.Serializable {
  private float BillAmount;
  private String CustomerName,BillDueDate;

  public float getBillAmount() {
    return BillAmount;
  }
  public String getCustomerName() {
    return CustomerName;
  }
  public String getBillDueDate() {
    return BillDueDate;
  }
```

```
  public void setBillAmount(float BillAmount) {
    this.BillAmount =BillAmount;
  }
  public void setCustomerName(String CustomerName) {
    this.CustomerName =CustomerName;
  }
  public void setBillDueDate(String BillDueDate) {
    this.BillDueDate = BillDueDate;
  }


}
```

2) Defining the Remote Interface:

Define the remote interface. Here, we are defining a remote interface named Hello with a method named getBillInfo () in it. This method returns a list which contains the object of the class ElectricBill**.**

Hello.java

```
 import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.*;


// Creating Remote interface for our application
public interface Hello extends Remote
{
  public List<Electric> getBillInfo() throws Exception;
}
```

3) Developing the Implementation Class (Remote Object):

Create a class and implement the above created interface.

Here we are implementing the getBillInfo() method of the Remote interface. When you invoke this method, it retrieves the records of a table named Bill. Sets these values to the Electric class using its setter methods, adds it to a list object and returns that list.

ImplExample.java

```
import java.sql. *;
import java.util.*;
```

```
// Implementing the remote interface
public class ImplExample implements Hello {

  // Implementing the interface method
  public List<Electric> getBillInfo() throws Exception {
    List<Electric> list = new ArrayList<Electric>();

    // JDBC driver name and database URL
    String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    String DB_URL = "jdbc:mysql://localhost:3306/electricbill";

    // Database credentials
    String USER = "root";
    String PASS = "system";

    Connection conn = null;
    Statement stmt = null;

    //Register JDBC driver
    Class.forName("com.mysql.cj.jdbc.Driver").newInstance ();
    //Open a connection
    System.out.println("Connecting to a selected database...");
    conn = DriverManager.getConnection(DB_URL, USER, PASS);
    System.out.println("Connected database successfully...");

    //Execute a query
    System.out.println("Creating statement...");

    stmt = conn.createStatement();
    String sql = "SELECT * FROM Bill";
    ResultSet rs = stmt.executeQuery(sql);

    //Extract data from result set
    while(rs.next()) {
      // Retrieve by column name
```

```
        float amount  = rs.getFloat("BillAmount");

        String name = rs.getString("CustomerName");
        String Date = rs.getString("BillDueDate");
              // Setting the values
        Electric info = new Electric();
        info.setBillAmount(amount);
        info.setCustomerName(name);
        info.setBillDueDate(Date);
        list.add(info);
      }
      rs.close();
      return list;
    }
}
```

4) Server Program:

An RMI server program should implement the remote interface or extend the implementation class. Here, we should create a remote object and bind it to the RMI registry.

Following is the server program of this application. Here, we will extend the above created class, create a remote object and register it to the RMI registry with the bind name billinfo.

In this program server is using port no:6666,we can use any available port.

Server.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;


public class Server extends ImplExample {
  public Server() {}
  public static void main(String args[]) {
    try {
      // Instantiating the implementation class
      ImplExample obj = new ImplExample();
```

```
        // Exporting the object of implementation class (here we are exporting the remote object to the stub)
        Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);

        // Binding the remote object (stub) in the registry
        Registry registry = LocateRegistry.createRegistry(6666);
        registry.rebind("billinfo",stub);
          System.err.println("Server ready");
      } catch (Exception e) {
        System.err.println("Server exception: " + e.toString());
        e.printStackTrace();
      }
    }
}
```

5) Client Program:

Following is the client program of this application. Here, we are fetching the remote object and invoking the method named **getBillInfo()**. It retrieves the records of the table from the list object and displays them.

In this program, LocateRegistry.getRegistry() method is taking two parameter, first is the IP address of the server and second is the port no on which server is listening.

Client.java

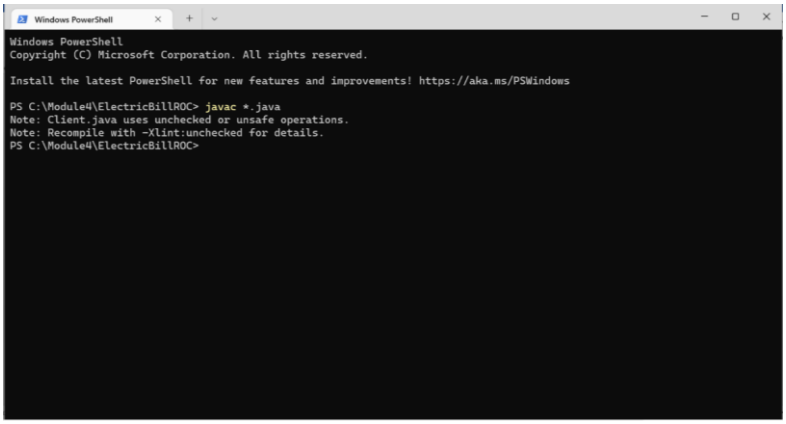
```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.*;
public class Client {
  private Client() {}
  public static void main(String[] args)throws Exception {
    try {
      // Getting the registry
      Registry registry = LocateRegistry.getRegistry("192.168.0.102",6666);

      // Looking up the registry for the remote object
      Hello stub = (Hello) registry.lookup("billinfo");
```

```
        // Calling the remote method using the obtained object
        List<Electric> list = (List)stub.getBillInfo();
        for (Electric l:list)
{



          System.out.println("Customer name: " + l.getCustomerName());
          System.out.println("Bill Due Date: " + l.getBillDueDate());
          System.out.println("Bill Amount: " + l.getBillAmount());

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

        }
      // System.out.println(list);
      } catch (Exception e) {
        System.err.println("Client exception: " + e.toString());
        e.printStackTrace();
      }
    }
}
```

4.5.3 Procedure to execute the program:

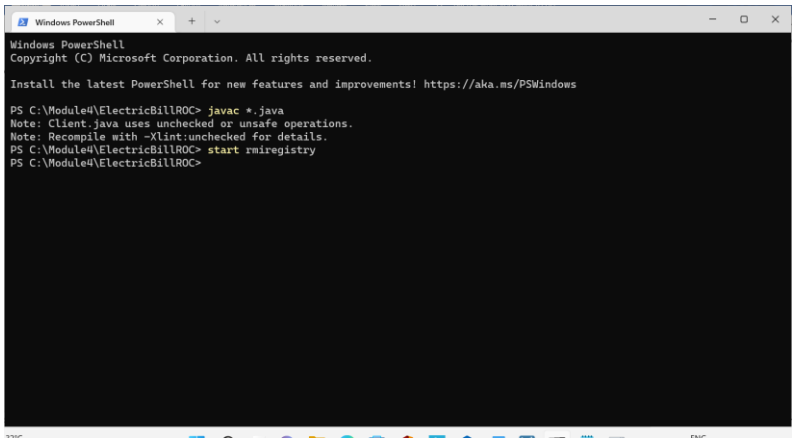
Step1: Open the command prompt and go to the folder where you have save all the java files and compile all the files by using the following command:

```
>javac *.java;
```

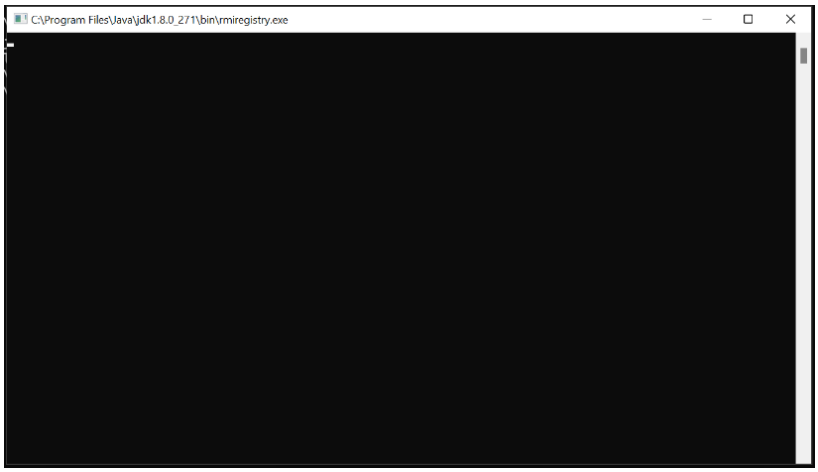


Step2: Start the rmi registry using the following command.

>start rmiregistry

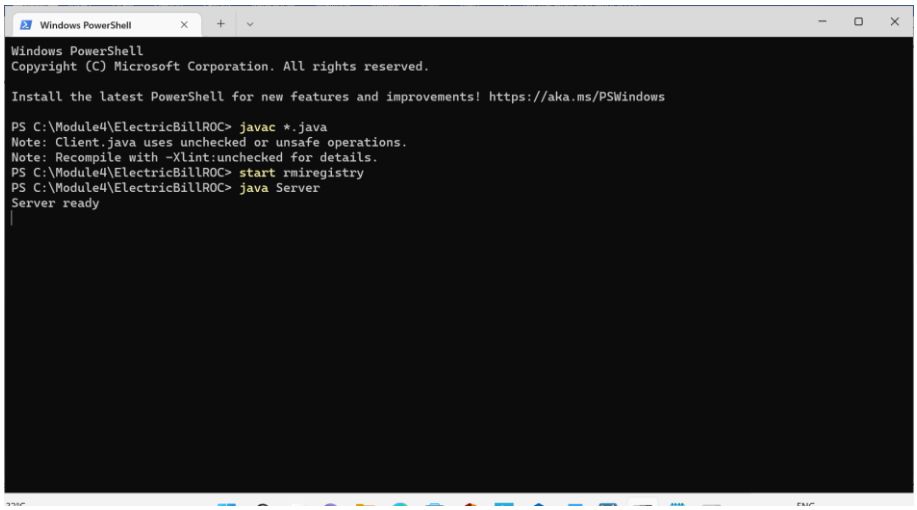


once you execute start rmiregistry command one more command prompt will be open as shown below:



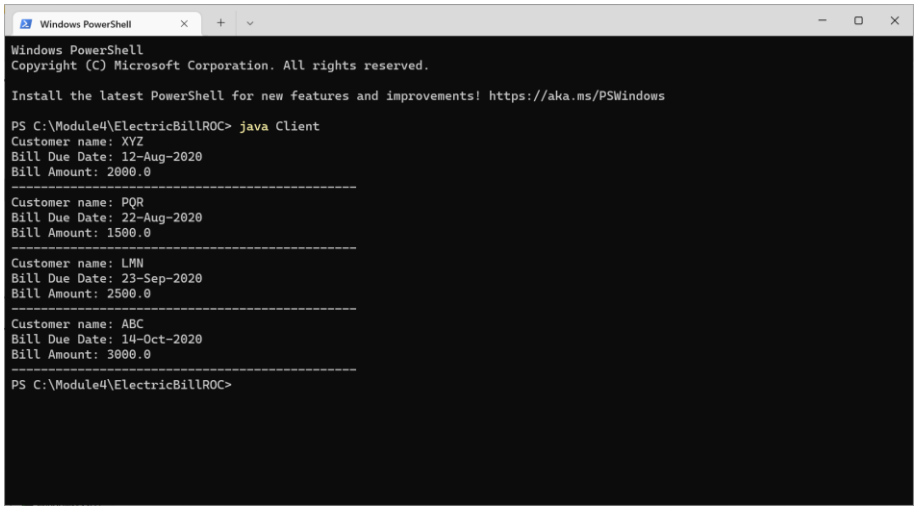
Step3: Run the server class file by the following command:

>java Server



Step 4: Run the client class file on new command prompt if running client and server on same machine by the following command.

>java Client



4.6 QUESTIONS

1) Explain the difference between RPC and RMI.
2) What is the function of java.net.unknownhostException?
3) What are the different terms that are used in RMI?
4) What are the different types of classes that are used in RMI?
5) What is the process of making a class serializable?

4.7 SELF LEARNING TOPICS: CONCEPT OF JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API that is used to connect and execute the query with any database.

*****

MODULE V

5

MUTUAL EXCLUSION

Unit Structure

5.0 Objective
5.1 Introduction
5.2 Summary
5.3 References
5.4 Unit End Exercises

5.0 OBJECTIVE

Mutual exclusion ensures that concurrent access of processes to a shared resource or data is serialized, that is, executed in a mutually exclusive manner. Mutual exclusion in a distributed system states that only one process is allowed to execute the critical section (CS) at any given time.

5.1 INTRODUCTION

Mutual exclusion is a concurrency control property which is introduced to prevent race conditions. It is the requirement that a process can not 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.

Mutual exclusion in single computer system Vs. distributed system:

In single computer system, memory and other resources are shared between different processes. The status of shared resources and the status of users is easily available in the shared memory so with the help of shared variable (For example: Semaphores) mutual exclusion problem can be easily solved.

In Distributed systems, we neither have shared memory nor a common physical clock and there for we can not solve mutual exclusion problem using shared variables. To eliminate the mutual exclusion problem in distributed system approach based on message passing is used.

A site in distributed system do not have complete information of state of the system due to lack of shared memory and a common physical clock.

During concurrent execution of processes, processes need to enter the critical section (or the section of the program shared across processes) at times for execution. It might so happen that because of the execution of multiple processes at once, the values stored in the critical section become inconsistent. In other words, the values depend on the sequence of

execution of instructions – also known as a race condition. The primary task of process synchronization is to get rid of race conditions while executing the critical section.

This is primarily achieved through mutual exclusion.

Mutual exclusion is a property of process synchronization which states that "no two processes can exist in the critical section at any given point of time". The term was first coined by Dijkstra. Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition.

Requirements of Mutual exclusion Algorithm:

No Deadlock:

Two or more site should not endlessly wait for any message that will never arrive.

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 are repeatedly executing critical section

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.

Fault Tolerance:

In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Solution to distributed mutual exclusion:

As we know shared variables or a local kernel can not be used to implement mutual exclusion in distributed systems. Message passing is a way to implement mutual exclusion. Below are the three approaches based on message passing to implement mutual exclusion in distributed systems:

Token Based Algorithm:

- A unique **token** is shared among all the sites.
- If a site possesses the unique token, it is allowed to enter its critical section
- This approach uses sequence number to order requests for the critical section.

- Each requests for critical section contains a sequence number. This sequence number is used to distinguish old and current requests.
- This approach insures Mutual exclusion as the token is unique

Example:

Suzuki-Kasami's Broadcast Algorithm

Non-token based approach:

- A site communicates with other sites in order to determine which sites should execute critical section next. This requires exchange of two or more successive round of messages among sites.
- This approach use timestamps instead of sequence number to order requests for the critical section.
- Whenever a site make request for critical section, it gets a timestamp. Timestamp is also used to resolve any conflict between critical section requests.
- All algorithm which follows non-token-based approach maintains a logical clock. Logical clocks get updated according to Lamport's scheme

Example:

Lamport's algorithm, Ricart–Agrawala algorithm

Quorum based approach:

- Instead of requesting permission to execute the critical section from all other sites, Each site requests only a subset of sites which is called a **quorum**.
- Any two subsets of sites or Quorum contains a common site.
- This common site is responsible to ensure mutual exclusion

Example:

- Maekawa's Algorithm

Programming primitives for mutual exclusion:

Peterson's algorithm does not generalize easily to a situation where there are more than two concurrent processes. For n processes, there are other solutions, such as Lamport's Bakery algorithm. However, the problem with these protocols is that, in general, they have to be designed anew for different situations and checked for correctness.

A better solution is to include support in the programming language for mutual exclusion. The first person to propose such primitive was Edsger Dijkstra, who suggested a new datatype called a semaphore.

A semaphore is an integer variable that supports an atomic test-and-set operation. More specifically, if a S is a variable of type semaphore, then two atomic operations are supported on S: P(S) and V(S). (The letters P and V come from the Dutch words passeren, to pass, and vrygeven, to release.)

The operation P(S) achieves the following in an atomic manner:

1. if (S > 0)
2. decrement S;
3. else
4. wait for S to become positive;

The operation V(S) is defined as follows:

5. if (there are threads waiting for S to become positive)
6. wake one of them up; //choice is nondeterministic
7. else
8. increment S;

Using semaphores, we can now easily program mutual exclusion to critical sections, as follows:

9. Thread 1 Thread 2
10.
11. ... ...
12. P(S); P(S);
13. // Enter critical section // Enter critical section
14. ... ...
15. // Leave critical section // Leave critical section
16. V(S); V(S);
17. ... ...

Mutual exclusion, starvation freedom and deadlock freedom are all guaranteed by the definition of a semaphore.

One problem with semaphores are that they are rather ``low-level''. The definition of a semaphore is orthogonal to the identification of the critical region. Thus, we connect critical regions across threads in a somewhat arbitrary fashion by using a shared semaphore. Correctness requires cooperation from all participating threads in resetting the semaphore. For instance, if any one thread forgets to execute V(S), all other threads get

blocked. Further, it is not required that each V(S) match a preceding P(S). This could lead to a semaphore being ``preloaded'' to an unsafe value.

Bare Machine Approaches:

Simple Shared Variables:

In most cases a single one-word variable can be read or written atomically. To share such a variable between threads you just have to guarantee the value really is in memory, and not optimized into a register. In Java and C, just mark the variable volatile.

Read/Writes on multi-word variables (like long and double in Java) are not guaranteed to be atomic. Code like x++ is almost definitely **not** atomic.

The Wrong Way:

If multiple threads execute this code

```
1:  while (something) {
2:      if (inUse) {        // THIS CODE IS ALL WRONG!
3:          inUse = true;
4:          doSomethingWithTheSharedResource();
5:          inUse = false;
6:      } else {
7:          doSomethingElse();
8:      }
9:  }
```

there's no mutual exclusion — two threads A and B can suffer this interleaving: A2 B2 A3 B3, etc. Now both are executing their critical sections at the same time.

Atomic Test and Set:

To fix the problem above the test (line 2) and the set (line 3) have to be atomic! On many hardware architectures there exists an atomic testAndSet instruction. It examines a variable and if it is false, it sets it to true and returns true. Otherwise it leaves the variable true and returns false. This is all done as a single atomic hardware instruction. With it we could write:

```
1:  while (something) {
2:      if (atomicTestAndSet(&inUse)) {
3:          doSomethingWithTheSharedResource();
```

```
4:          inUse = false;
5:      } else {
6:          doSomethingElse();
7:      }
8:  }
```

Other Atomic Operations:

Sometimes with advanced hardware, or with support from an operating system, larger words of memory can be protected with a very simple API. (Example: Win32 InterlockedIncrement and the Java classes in java.util.concurrent.atomic.)

Peterson's Algorithm:

If you don't have a hardware atomic test and set, then enforcing mutual exclusion on a bare machine that meets all the requirements above takes a surprising amount of work.

```
// Thread 0                    // Thread 1
claim0 = false;                claim1 = false;
while (true) {                 while (true) {
   claim0 = true;                 claim1 = true;
   turn = 1;                      turn = 0;
   while (claim1 && turn != 0) {}     while (claim0 && turn != 1) {}
   CRITICAL SECTION               CRITICAL SECTION
   claim0 = false;                claim1 = false;
   NON_CRITICAL_SECTION           NON_CRITICAL_SECTION
}                              }
```

Problems with this:

- Wordy and complex
- Burns CPU cycles while waiting
- Doesn't directly scale up to more than two processors

Bakery Algorithm:

The bakery algorithm solves the mutual exclusion problem for multiple threads on a bare machine.

Scheduler-Assisted Mutual Exclusion:

It's nice if the operating system can provide some way to make a thread go to sleep (i.e. remove it from the pool of ready threads that get scheduled on a processor) and wake it up when something happens.

Primitives to make this happen include barriers (like latches, countdowns, and cyclic barriers), mutexes, semaphores, futexes, and exchangers.

Barriers:

Barriers are simple objects used to allow one or more threads to wait until other threads have completed some work. They can be used for mutual exclusion if you like. Java has a few of these in its java.util.concurrent library.

Semaphores:

Semaphores are pretty much designed to implement mutual exclusion. They work like this:

Semaphore s = new Semaphore(MAX, true);

.

.

.

s.acquire();

CRITICAL_SECTION

s.release();

.

.

.

The idea is that the thread will block on the semaphore if the maximum number of threads have already successfully "acquired." When a thread does a "release" then one of the block threads will wake up (meaning the acquire call finally returns).

The internal implementation is something like this:

Semaphore(max, fifo)

Creates a new semaphore with the given maximum value and fifo flag. If fifo is true, threads block in a FIFO queue so a release always wakes the one blocked the longest; otherwise they block in a set and a release wakes an arbitrary blocked thread.

s.acquire()

atomically {if (s.value > 0) s.value--; else block on s}

s.release()

```
atomically {
   if (there are threads blocked on s)
      wake one of them
   else if (s.value == s.MAX)
      fail
   else
      s.value++
}
```

A semaphore with a maximum count of 1 is called a binary semaphore or sometimes just a mutex.

Semaphores are an "unstructured" low-level primitive; you might not use them directly in high-level applications:

- Responsibility of proper use is distributed among all the threads; one malicious or buggy thread could:
  - "Forget" to call wait before entering its critical section and start whacking shared data in violation of mutual exclusion
  - "Forget" to call signal when done, causing the other threads to deadlock.
- There is no guarantee that resources are accessed only within critical sections.

So the modern approach is for **resources to protect themselves**.

Data-Oriented Synchronization:

A simple way to put the resource itself in charge of the protection (instead of the threads) is to have a lock associated with every object, like in Java:

```
public class Account {
   private BigDecimal balance = BigDecimal.ZERO;

   public synchronized BigDecimal getBalance() {
      return balance;
   }

   public synchronized void deposit(BigDecimal amount) {
      balance = balance.add(amount);
```

```
    }
}
```

Every object in Java has an associated monitor which a thread can lock and unlock. Only one thread at a time can hold a monitor's lock. A synchronized method or synchronized statement does an implicit lock at the beginning and an unlock at the end. An attempt to lock a monitor that is already locked causes the thread to wait until the lock is free (unless the lock owner itself is trying to lock again).

By making the account responsible for managing its own synchronization we free the gazillions of threads that wish to use accounts from having to remember to lock and unlock themselves!

Note: You don't have to make the scope of the lock be an entire method:

```
    .
    .
    .
    synchronized (x) {
       ...
    }
    .
    .
    .
```

Notification:

Simply using the synchronized keyword might not be sufficient. Consider a message board where threads can post messages and other threads can take them out. The message buffer has a fixed, bounded size. Adds can only be done when the buffer isn't full, and removes when not empty.

```
public class UnsynchronizedBuffer {
    private Object[] data;
    private int head = 0;
    private int tail = 0;
    private int count = 0;
    public Buffer(int capacity) {
        data = new Object[capacity];
    }
    public boolean add(Object item) {
        if (count < data.length) {
            data[tail] = item;
```

```
            tail = (tail + 1) % data.length;
            count++;
            return true;
        }
        return false;
    }
    public Object remove() {
        if (count > 0) {
            Object item = data[head];
            head = (head + 1) % data.length;
            count--;
            return item;
        }
        return null;
    }
    public int getSize() {
        return count;
    }
}
```

Notice the race conditions! The buffer could have room for only one more item, but two threads both try to add at the same time. They each execute the size test before either completes the adding of an item. There's a similar race when multiple threads try to remove from a buffer with only one message.

Exercise: Explain the race conditions a bit more precisely.

What if we just synchronize add and remove? Yeah, that'll remove the races, but threads wanting to add (or remove) before proceeding will have to busy wait! NOOOOOOOO! Can't they just go to sleep until some other thread makes room (in the wants-to-add case) or puts something in (in the wants-to-remove case)? Yep. Like this:

```
public class SynchronizedBuffer {
    private Object[] data;
    private int head = 0;
    private int tail = 0;
    private int count = 0;
    public Buffer(int capacity) {
        data = new Object[capacity];
```

```
    }
    public synchronized void add(Object item) {
        while (count == data.length) {
            try {wait();} catch (InterruptedException e) {}
        }
        data[tail] = item;
        tail = (tail + 1) % data.length;
        count++;
        notifyAll();
    }
    public synchronized Object remove() {
        while (count == 0) {
            try {wait();} catch (InterruptedException e) {}
        }
        Object item = data[head];
        head = (head + 1) % data.length;
        count--;
        notifyAll();
        return item;
    }
    public int getSize() {
        return count;
    }
}
```

wait() moves a thread into the buffer's wait set (making them blocked), and notifyAll removes all the threads in the object's wait set (and makes them runnable again).

This makes the buffer a blocking queue, which is pretty nice; the API for it is quite clean. But, doing things this way in Java still leaves room for improvement: all threads here are waiting on the buffer object only, we're not distinguishing between threads waiting for "not full" and those waiting for "not empty".

Exercise: What if we placed two new fields within the SynchronizedBuffer class

```
private Object notFull = new Object();
    private Object notEmpty = new Object();
```

and then rewrote the body of add like this:

```
synchronized (notFull) {
   while (count == data.length) {
      try {notFull.wait();} catch (InterruptedException e) {}
   }
}
data[tail] = item;
tail = (tail + 1) % data.length;
count++;
synchronized (notEmpty) {notEmpty.notifyAll();}
```

and made similar changes to remove(). Does this work? If so, is it better than the previous version? How? If not, how does it fail? (Describe any race conditions, deadlock or starvation in detail.)

By the way, Java does have its own blocking queue class, but you wanted to learn about wait and notifyAll, right?

Ada Protected Objects:

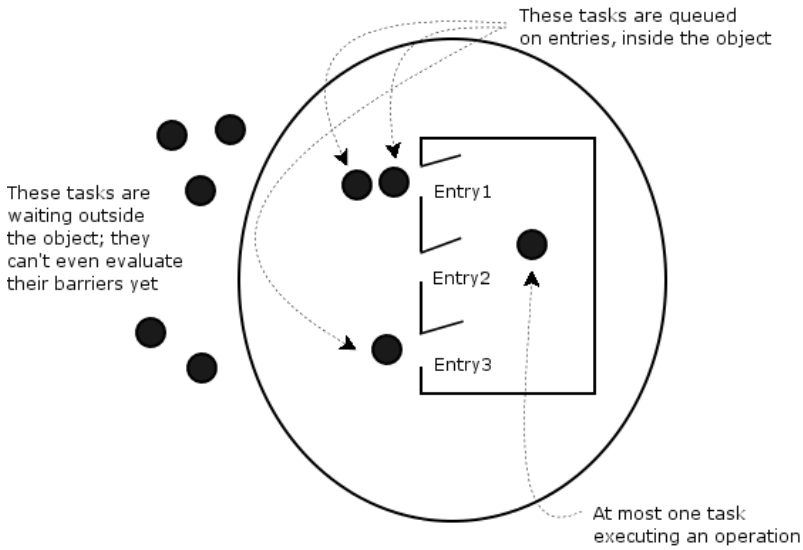
Ada's protected objects are like monitors but a bit more sophisticated — most everything you normally do in these cases is taken care of declaratively! The bounded buffer example (shown in the context in which Index is defined as a modular type and Item is the type of objects to be stored in a buffer):

```
protected type Buffer is            -- SPECIFICATION
   entry Put(I: Item);              --
   entry Get(I: out Item);          --  Public part specifies:
   function Size return Natural;    --  entries, procs, fns
private
   Data: array (Index) of Item;     --  Private part holds the shared
   Head, Tail: Index := 0;          --  data (this is in the spec so
   Count: Natural := 0;             --  the compiler knows how much space
end Buffer;                         --  to allocate for an object)


protected body Buffer is            -- BODY (operation bodies ONLY; no data)


   entry Put(I: Item) when Count < Index'Modulus is
   begin
```

```
      Data(Tail) := I;
      Count := Count + 1;
      Tail := Tail + 1;
   end Put;

   entry Get(I: out Item) when Count > 0 is
   begin
      I := Data(Head);
      Count := Count - 1;
      Head := Head + 1;
   end Get;

   function Size return Natural is
      return Count;
   end Size;

end Buffer;
```

How it works:

- Entries, procedures and functions by definition provide mutually exclusive access.
- Entries and procedures can read and write shared data; functions can only read. Thus, an implementation can optimize by allowing multiple function calls at the same time.
- Only entries have barriers. There is a queue associated with each entry "inside" the object.



- When an entry is called the barrier is evaluated. If false, the calling task is queued on the entry.
- At the end of a procedure or entry body, barriers are re-evaluated.
- Tasks already queued on entries **have priority over tasks trying to get in**. Therefore a caller can not even evaluate a barrier until the protected object is finished with the current and all queued tasks.

It's the programmer's responsibility to see that all barriers execute quickly and don't call a potentially blocking operation!

Advantages of protected objects (please read the Ada 95 Rationale section on protected types):

- Scalability
- Adaptability
- Modularity
- Efficiency
- Expressiveness
- Compatibility
- Great for implementing interrupt handlers

In computer science, mutual exclusion refers to the requirement of ensuring that no two concurrent processes are in their critical section at the same time; it is a basic requirement in concurrency control, to prevent race conditions. Here, a critical section refers to a period when the process accesses a shared resource, such as shared memory. The requirement of mutual exclusion was first identified and solved by Edsger W. Dijkstra in his seminal 1965 paper titled Solution of a problem in concurrent programming control, and is credited as the first topic in the study of concurrent algorithms.

Program Code:

```
#include <pthread.h>

#include <stdio.h>

int count = 0;

pthread_mutex_t thread_lock;

void* run_thread()

{

 pthread_mutex_lock(&thread_lock);

 pthread_t thread_id = pthread_self();
```

```
 printf("Thread %u: Current value of count = %d\n", thread_id, count);
 printf("Thread %u incrementing count ...\n");
 count++;
 sleep(1);
 printf("Value of count after incremented by thread %u = %d\n",
thread_id, count);
 pthread_mutex_unlock(&thread_lock);
 pthread_exit(NULL);
}
int main (int argc, char *argv[])
{
 pthread_t thread_array[4];
 int i = 0, ret, thread_num = 4;
 for (i = 0; i < thread_num; i++) {
 if ((ret = pthread_create(&thread_array[i], NULL, run_thread, NULL)==-
1) { printf("Thread
creation failed with return code: %d", ret);
 exit(ret);
 }
 }
 pthread_exit(NULL);
}


{


while(!kbhit())
if(cs!=0)
{
t2 = time(NULL);
if(t2-t1 > run)
{
printf("Process%d ",pro-1);
printf(" exits critical section.\n");
```

```
cs=0;
}
}
key = getch();
if(key!='q')
{
if(cs!=0)
printf("Error: Another process is currently executing critical
section Please wait till its execution is over.\n");
else
{
printf("Process %d ",pro);
printf(" entered critical section\n"); cs=1;
pro++;
t1 = time(NULL);
}
}
}
}
```

OUTPUT:

```
{   while(!kbhit())  if(cs!=0)
{
 t2 = time(NULL);
if(t2-t1 > run)
{
printf("Process%d ",pro-1);
 printf(" exits critical section.\n");
cs=0;
  }
  }
 key = getch();
  if(key!='q')
```

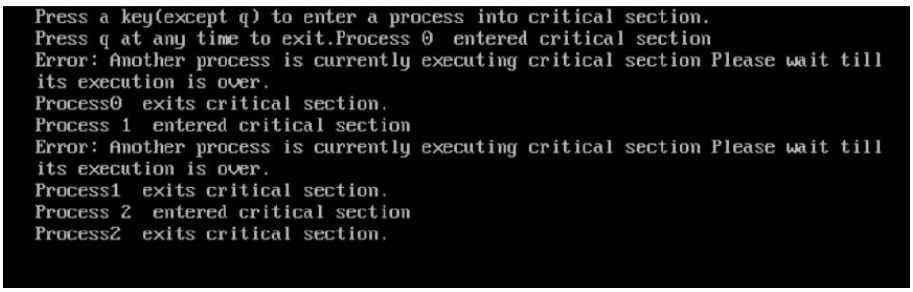
```
 {
 if(cs!=0)  printf("Error: Another process is currently executing critical section Please wait till its execution is over.\n");
 else
 {
 printf("Process %d ",pro);
 printf(" entered critical section\n");
cs=1;
 pro++;
 t1 = time(NULL);
 }
 }
 }
 }
```

OUTPUT:

```
{

while(!kbhit())
if(cs!=0)
{
t2 = time(NULL);
if(t2-t1 > run)
{
printf("Process%d ",pro-1);
printf(" exits critical section.\n");
cs=0;
}
}
key = getch();
if(key!='q')
{
if(cs!=0)
```

```
printf("Error: Another process is currently executing critical
section Please wait till its execution is over.\n");
else
{
printf("Process %d ",pro);
printf(" entered critical section\n"); cs=1;
pro++;
t1 = time(NULL);
}
}
}
}
```

OUTPUT:



5.2 SUMMARY

A mutual exclusion (mutex) is a program object that prevents simultaneous access to a shared resource. This concept is used in concurrent programming with a critical section, a piece of code in which processes or threads access a shared resource.

5.3 REFERENCES

1) The Basics of Hacking and Penetration Testing

2) Hacking: The Art of Exploitation

3) The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws

5.4 UNIT END EXERCISES

Write a code to perform Distributed and Primitive systems.

*****

MODULE VI

6

IMPLEMENTATION OF CLOUD COMPUTING SERVICES

Unit Structure

6.0 Objective
6.1 Introduction
6.2 Summary
6.3 References
6.4 Unit End Exercises

6.0 OBJECTIVE

Simply put, cloud computing is the delivery of computing services—including servers, storage, databases, networking, software, analytics, and intelligence—over the Internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale. You typically pay only for cloud services you use, helping lower your operating costs, run your infrastructure more efficiently and scale as your business needs change.

6.1 INTRODUCTION

SaaS:

Software as a Service:

Essentially, any application that runs with its contents from the cloud is referred to as Software as a Service, As long as you do not own it.

Some examples are Gmail, Netflix, OneDrive etc.

AUDIENCE: End users, everybody

IaaS:

Infrastructure as a Service means that the provider allows a portion of their computing power to its customers, It is purchased by the potency of the computing power and they are bundled in Virtual Machines. A company like Google Cloud platform, AWS, Alibaba Cloud can be referred to as IaaS providers because they sell processing powers (servers, storage, networking) to their users in terms of Virtual Machines.

AUDIENCE: IT professionals, System Admins

PaaS:

Platform as a Service is more like the middle-man between IaaS and SaaS, Instead of a customer having to deal with the nitty-gritty of servers, networks and storage, everything is readily available by the PaaS providers. Essentially a development environment is initialized to make building applications easier.

Examples would be Heroku, AWS Elastic Beanstalk, Google App Engine etc

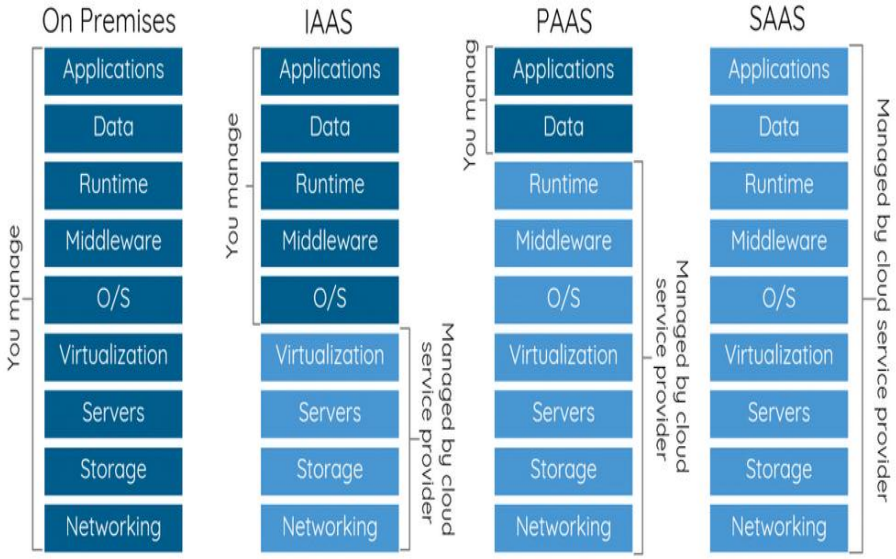
AUDIENCE: Software developers.

There are various cloud services available today, such as Amazon's EC2 and AWS, Apache Hadoop, Microsoft Azure and many others. Which category does each belong to and why?

Amazon EC2 and AWS:

Is an Infrastructure as a Service because you'll need System Administrators to manage the working process of your operating system. There is no abstraction to build a fully featured app ordinarily. Microsoft Azure would also fall under this category following the aforementioned guidelines.

AWS Basic Details:



IaaS (Infrastructure as a Service):

You get the whole infrastructure with hardware. You chose the type of OS that needs to be installed. You will have to install the necessary software.

AWS Example:

EC2 which has only the hardware and you select the base OS to be installed. If you want to install Hadoop on that you have to do it yourself, it's just the base infrastructure AWS has provided.

PaaS (Platform as a Service):

Provides you the infrastructure with OS and necessary base software. You will have to run your scripts to get the desired output.

AWS Example:

EMR which has the hardware (EC2) + Base OS + Hadoop software already installed. You will have to run hive/spark scripts to query tables and get results. You will need to invoke the instance and wait for 10 min for the setup to be ready. You have to take care of how many clusters you need based on the jobs you are running, but not worry about the cluster configuration.

SaaS (Software as a Service):

You don't have to worry about Hardware or even Software. Everything will be installed and available for you to use instantly.

AWS Example:

Athena, which is just a UI for you to query tables in S3 (with metadata stored in Glu). Just open the browser login to AWS and start running your queries, no worry about RAM/Storage/CPU/number of clusters, everything the cloud takes care of.

Infrastructure as a service (IaaS):

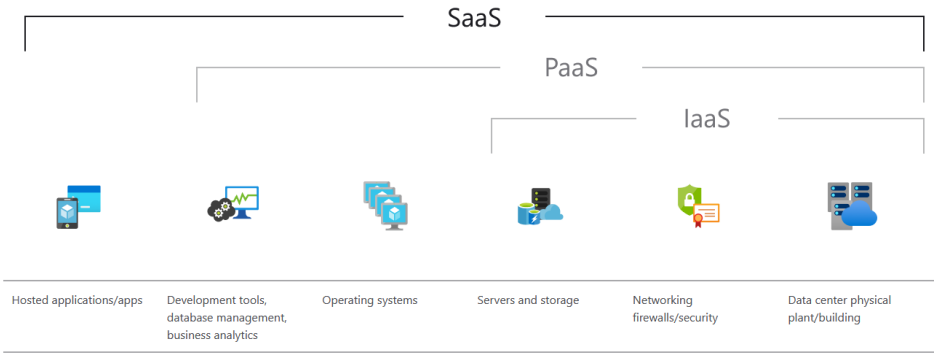
The most basic category of cloud computing services. With IaaS, you rent IT infrastructure—servers and virtual machines (VMs), storage, networks, operating systems—from a cloud provider on a pay-as-you-go basis.

Platform as a service (PaaS):

Platform as a service refers to cloud computing services that supply an on-demand environment for developing, testing, delivering and managing software applications. PaaS is designed to make it easier for developers to quickly create web or mobile apps, without worrying about setting up or managing the underlying infrastructure of servers, storage, network and databases needed for development.

Software as a service (SaaS):

Software as a service is a method for delivering software applications over the Internet, on demand and typically on a subscription basis. With SaaS, cloud providers host and manage the software application and underlying infrastructure and handle any maintenance, like software upgrades and security patching. Users connect to the application over the Internet, usually with a web browser on their phone, tablet or PC.



Infrastructure as a service (IaaS)

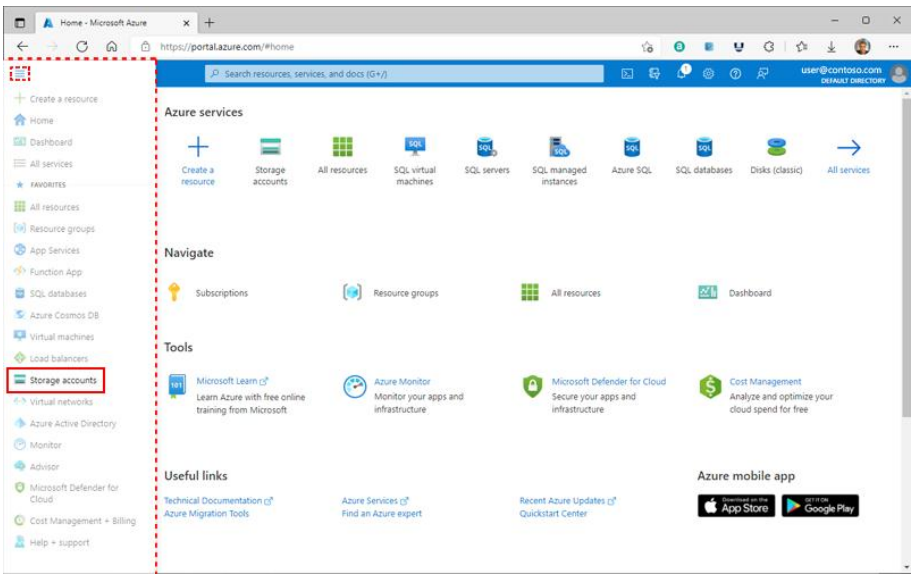
Create a storage account:

A storage account is an Azure Resource Manager resource. Resource Manager is the deployment and management service for Azure. For more information, see Azure Resource Manager overview.

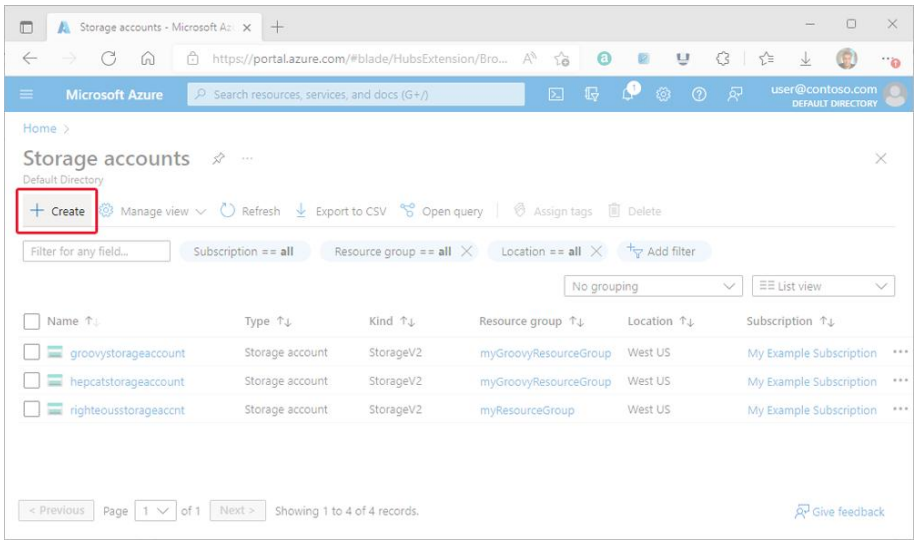
Every Resource Manager resource, including an Azure storage account, must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group, or use an existing resource group. This how-to shows how to create a new resource group.

To create an Azure storage account with the Azure portal, follow these steps:

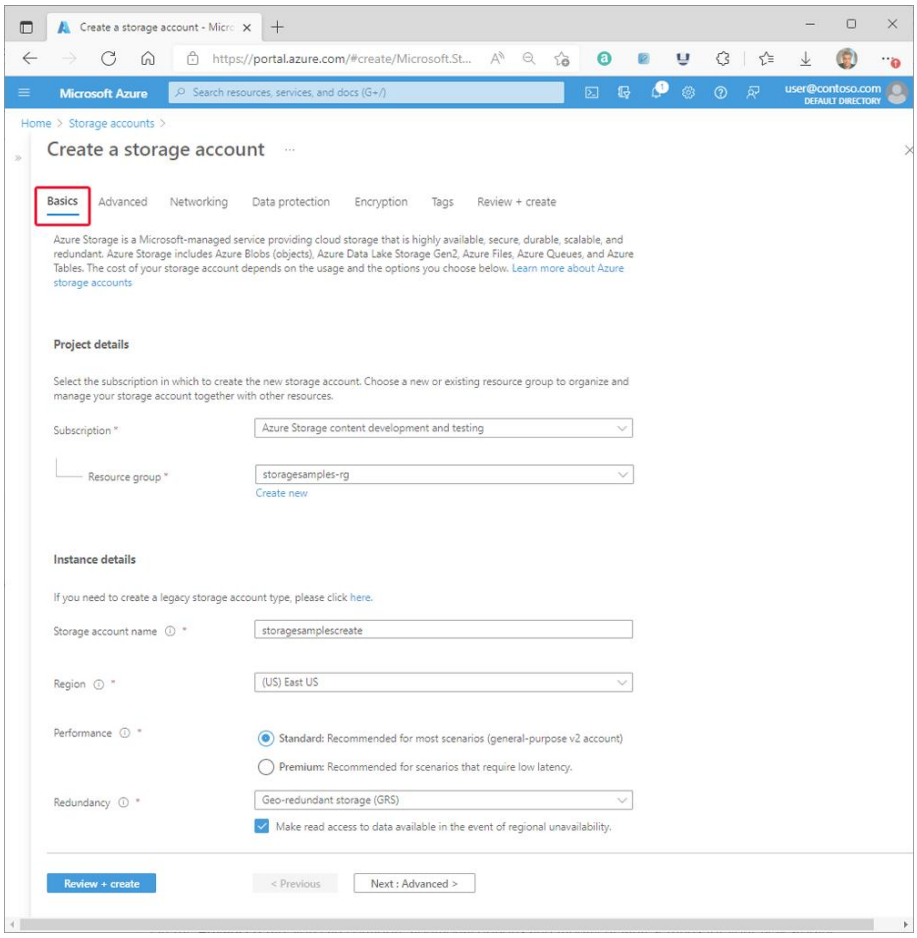
1. From the left portal menu, select Storage accounts to display a list of your storage accounts. If the portal menu isn't visible, click the menu button to toggle it on.



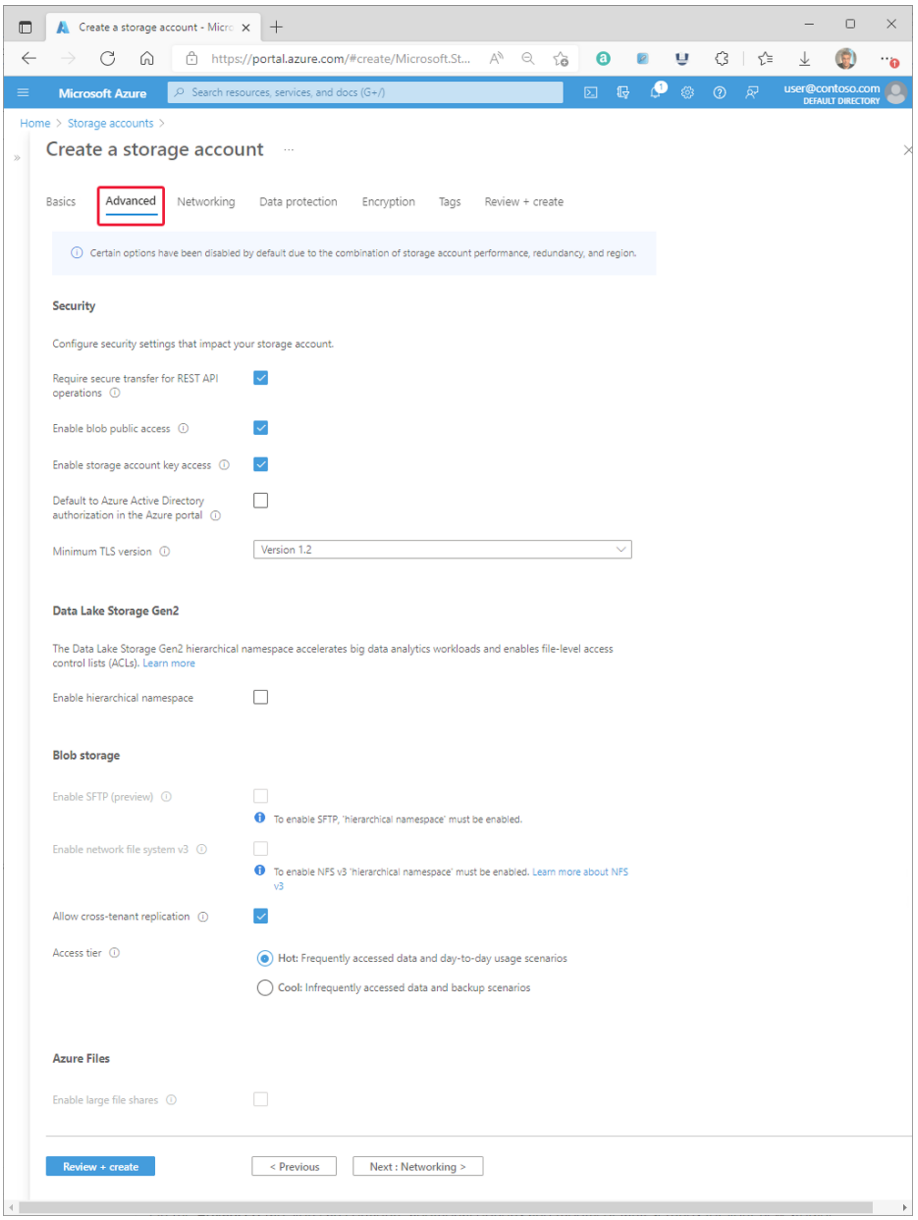
2. On the **Storage accounts** page, select **Create**.



The following image shows a standard configuration of the basic properties for a new storage account.



The following image shows a standard configuration of the advanced properties for a new storage account.



Platform as a service (PaaS):

Platform as a service (PaaS) is a complete development and deployment environment in the cloud, with resources that enable you to deliver everything from simple cloud-based apps to sophisticated, cloud-enabled enterprise applications. You purchase the resources you need from a cloud service provider on a pay-as-you-go basis and access them over a secure Internet connection.

Like IaaS, PaaS includes infrastructure—servers, storage and networking—but also middleware, development tools, business intelligence (BI) services, database management systems and more. PaaS is designed to support the complete web application lifecycle: building, testing, deploying, managing and updating.

PaaS allows you to avoid the expense and complexity of buying and managing software licenses, the underlying application infrastructure and middleware, container orchestrators such as Kubernetes or the development tools and other resources. You manage the applications and services you develop and the cloud service provider typically manages everything else.

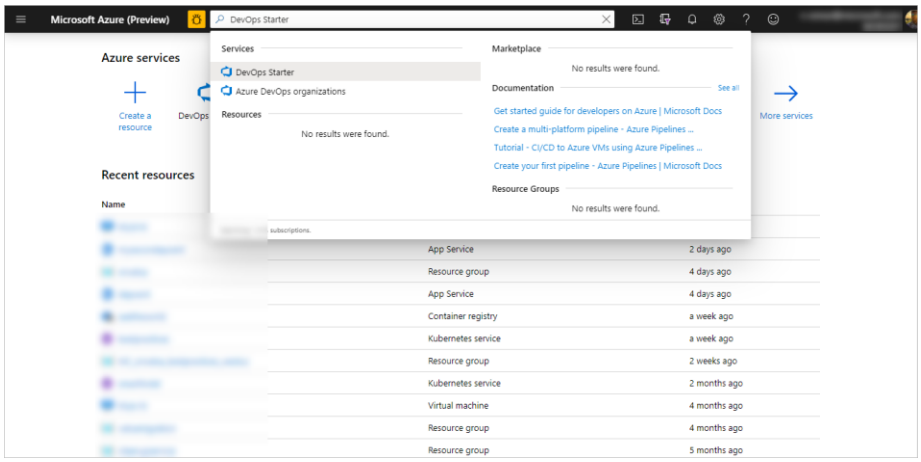
Tutorial: Deploy Node.js app to Azure Web App using DevOps Starter for GitHub Actions

Use DevOps Starter to deploy a Node.js app

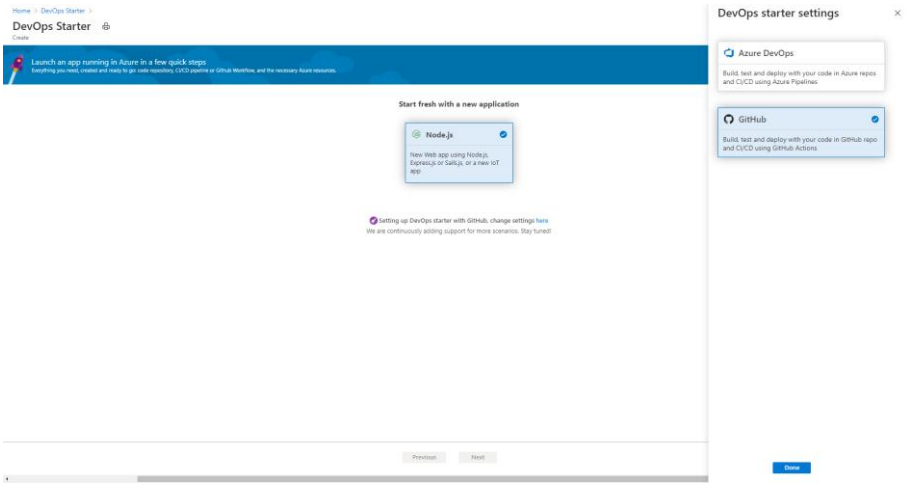
DevOps Starter creates a workflow in GitHub. You can use an existing GitHub organization. DevOps Starter also creates Azure resources such as Web App in the Azure subscription of your choice.

1. Sign in to the Azure portal.

2. In the search box, type DevOps Starter, and then select. Click on Add to create a new one.



3. Ensure that the CI/CD provider is selected as **GitHub Actions**.



4. Select **Node.js**, and then select **Next**.

5. Under **Choose an application Framework**, select **Express.js**, and then select **Next**. The application framework, which you chose in a previous step, dictates the type of Azure service deployment target that's available here.

6. Select the **Windows Web App**, and then select **Next**.

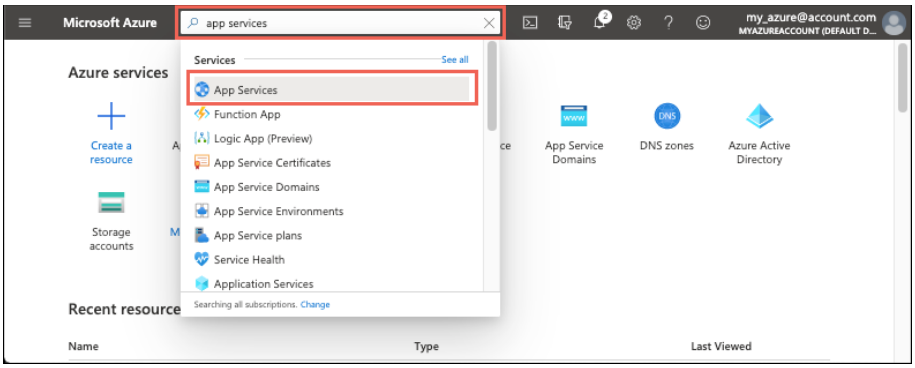
Software as a service (SaaS):

Software as a service (SaaS) allows users to connect to and use cloud-based apps over the Internet. Common examples are email, calendaring and office tools (such as Microsoft Office 365).

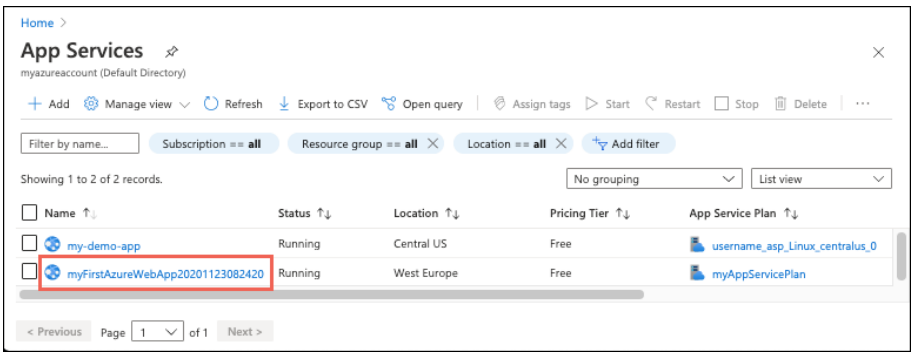
SaaS provides a complete software solution which you purchase on a pay-as-you-go basis from a cloud service provider. You rent the use of an app for your organisation and your users connect to it over the Internet, usually with a web browser. All of the underlying infrastructure, middleware, app software and app data are located in the service provider's data center. The service provider manages the hardware and software and with the appropriate service agreement, will ensure the availability and the security of the app and your data as well. SaaS allows your organisation to get quickly up and running with an app at minimal upfront cost.

Manage the Azure app:

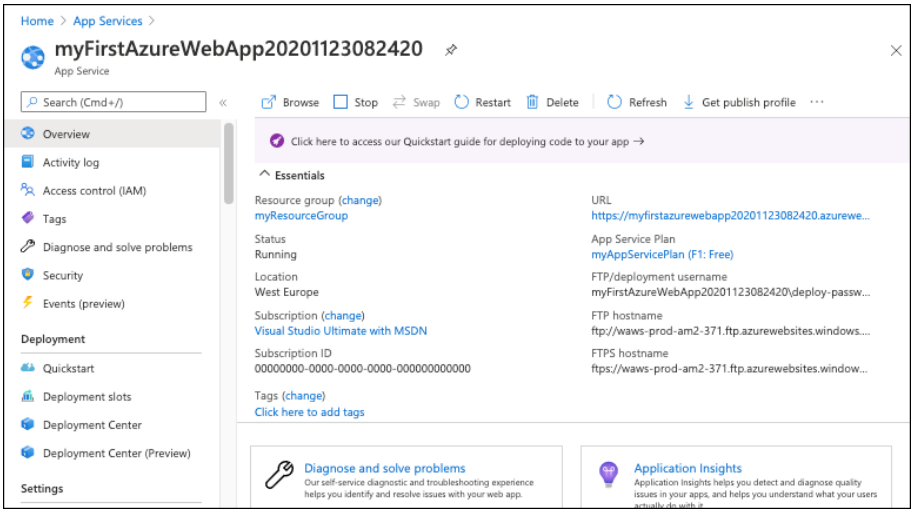
To manage your web app, go to the Azure portal, and search for and select App Services.



On the **App Services** page, select the name of your web app.



The **Overview** page for your web app, contains options for basic management like browse, stop, start, restart, and delete. The left menu provides further pages for configuring your app.



6.3 REFERENCES:

1. What is cloud computing? A beginner's guide | Microsoft Azure
2. Create a resource - Microsoft Azure

2.4 UNIT END EXERCISES

Create one VM on Azure Portal with free account.

*****

MODULE VII

7

IMPLEMENTATION OF IDENTITY MANAGEMENT USING CLOUD

Unit Structure

7.0 Objective
7.1 Introduction
7.2 Summary
7.3 References
7.4 Unit End Exercises

7.0 OBJECTIVE

The main goal of identity management is to ensure only authenticated users are granted access to the specific applications, systems or IT environments for which they are authorized.

7.1 INTRODUCTION

Identity management in cloud computing is the subsequent step of identity and access management (IAM) solutions. However, it is a lot more than merely a straightforward web app single sign-on (SSO) solution. This next generation of IAM solution is a holistic move of the identity provider right to the cloud.

Innovations in the user identity management space have been a trend in the past couple of years. Most of these developments across business and technology fronts have been around identity management in cloud computing, enabling the authentication and authorization processes right in the cloud.

The primary goal of identity management in cloud computing is dealing with personal identity information so that a user's access to data, computer resources, applications, and services is controlled accurately.

Identity management in cloud computing is the subsequent step of identity and access management (IAM) solutions. However, it is a lot more than merely a straightforward web app single sign-on (SSO) solution. This next generation of IAM solution is a holistic move of the identity provider right to the cloud.

Known as Directory-as-a-Service (DaaS), this particular service is the advanced version of the conventional and on-premises solutions, including Lightweight Directory Access Protocol (LDAP) as well as Microsoft Active Directory (AD).

Features of a Modern Cloud Identity Management Solution:

The following are a few advantages of identity management in cloud computing:

- **It offers a consistent access control interface:** Applicable for all cloud platform services; Cloud IAM solutions provide a clean and single access control interface.
- **It offers superior security levels:** If needed, we can easily define increased security levels for crucial applications.
- **It lets businesses access resources at diverse levels:** Businesses can define roles and grant permissions to explicit users for accessing resources at diverse granularity levels.

Why Do You Need Cloud IAM?

Identity management in cloud computing incorporates all categories of user-base who can operate in diverse scenarios and with specific devices.

A modern cloud Identity and Access Management (IAM) solution helps to:

- Connect professionals, employees, IT applications, and devices securely either on-premise or the cloud and through involved networks.
- It makes it easy to share the network abilities with the entire grid of users who were precisely connected with it.
- It offers zero management overhead, enhanced security levels, and easy management of diverse users with directory service in a SaaS solution.
- It is utterly known that cloud-based services are enabled, configured, and hosted by external providers. This scenario may also get the least hassle, either for users or clients. As a result, many organizations can enhance their productivity with cloud IAM.
- SaaS protocol is created and used as a hub for connecting with all virtual networks of distributors, suppliers, and partners.
- Business users can deal with all services and programs in one place with cloud services, and Identity management can be enabled with a click on a single dashboard.
- Easily connect your cloud servers, which are virtually hosted at Google Cloud, AWS, or elsewhere right next to your current LDAP or AD user store.
- Widen and extend your present LDAP or AD directory right to the cloud.

- Deal with Linux, Windows, and Mac desktops, laptops, and servers established at different locations.
- Connect different users to diverse applications that use LDAP or SAML-based authentication.
- Effortlessly handle user access controls to WiFi networks securely by using a cloud RADIUS service.
- Enable GPO-like functionalities across diverse Windows, Mac, and Linux devices.
- Facilitate both system-based as well as application-level multi-factor authentications (2FA).

These abilities help build a platform that connects users to virtually all IT resources through any provider, protocol, platform, or location.

AAA (Authentication, Authorization, Accounting):

AAA is a standard-based framework used to control who is permitted to use network resources (through authentication), what they are authorized to do (through authorization), and capture the actions performed while accessing the network (through accounting).

1. Authentication:

The process by which it can be identified that the user, which wants to access the network resources, valid or not by asking some credentials such as username and password. Common methods are to put authentication on console port, AUX port, or vty lines.

As network administrators, we can control how a user is authenticated if someone wants to access the network. Some of these methods include using the local database of that device (router) or sending authentication requests to an external server like the ACS server. To specify the method to be used for authentication, a default or customized authentication method list is used.

2. Authorization:

It provides capabilities to enforce policies on network resources after the user has gained access to the network resources through authentication. After the authentication is successful, authorization can be used to determine what resources is the user allowed to access and the operations that can be performed.

For example, if a junior network engineer (who should not access all the resources) wants to access the device then the administrator can create a view that will allow particular commands only to be executed by the user (the commands that are allowed in the method list). The administrator can use the authorization method list to specify how the user is authorized to network resources i.e through a local database or ACS server.

3. Accounting:

It provides means of monitoring and capturing the events done by the user while accessing the network resources. It even monitors how long the user has access to the network. The administrator can create an accounting method list to specify what should be accounted for and to whom the accounting records should be sent.

AAA implementation:

AAA can be implemented by using the local database of the device or by using an external ACS server.

Local database:

If we want to use the local running configuration of the router or switch to implement AAA, we should create users first for authentication and provide privilege levels to users for Authorization.

ACS server:

This is the common method used. An external ACS server is used (can be ACS device or software installed on Vmware) for AAA on which configuration on both router and ACS is required. The configuration includes creating a user, separate customized method list for authentication, Authorization, and Accounting.

The client or Network Access Server (NAS) sends authentication requests to the ACS server and the server takes the decision to allow the user to access the network resource or not according to the credentials provided by the user.

Authorization Techniques:

Role-based access control:

RBAC or Role-based access control technique is given to users as per their role or profile in the organization. It can be implemented for system-system or user-to-system.

JSON web token:

JSON web token or JWT is an open standard used to securely transmit the data between the parties in the form of the JSON object. The users are verified and authorized using the private/public key pair.

SAML:

SAML stands for **Security Assertion Markup Language.** It is an open standard that provides authorization credentials to service providers. These credentials are exchanged through digitally signed XML documents.

OpenID authorization:

It helps the clients to verify the identity of end-users on the basis of authentication.

OAuth:

OAuth is an authorization protocol, which enables the API to authenticate and access the requested resources.

This is a Java Program to Illustrate how User Authentication is Done.

Enter username and password as input strings. After that we match both strings against given username and password. If it matches then Authentication is successful or else Authentication fails.

Here is the source code of the Java Program to Illustrate how User Authentication is Done. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

Code:

```
1.  import java.util.Scanner;
2.  public class User_Authentication
3.  {
4.      public static void main(String args[])
5.      {
6.          String username, password;
7.          Scanner s = new Scanner(System.in);
8.          System.out.print("Enter username:");//username:user
9.          username = s.nextLine();
10.         System.out.print("Enter password:");//password:user
11.         password = s.nextLine();
12.         if(username.equals("user") && password.equals("user"))
13.         {
14.             System.out.println("Authentication Successful");
15.         }
16.         else
17.         {
18.             System.out.println("Authentication Failed");
```

```
19.         }
20.     }
21. }
```

Output:

$ javac User_Authentication.java

$ java User_Authentication

Enter username:user

Enter password:user

Authentication Successful

Enter username:abcd

Enter password:1234

Authentication Failed

Authentication is the process of verifying the identity of users or information. User authentication is the process of verifying the identity of the user when that user logs in to a computer system. The main objective of authentication is to allow authorized users to access the computer and to deny access to unauthorized users.

Example:

Default Assumption: User name = Sandeep Kamble, Password = Sandeep Kamble

Input: User name = Sandeep Kamble, Password = Sandeep Kamble

Output: Authentication Successful

Input: User name = Sandeep Kamble, Password = Hello world

Output: User name/ Password not matching

Approach:

1. Take username and password as string input from the user.
2. Check if the username matches the password or not.
3. If it matches then welcome the user.
4. Else display an appropriate message.

Below is the implementation of the above approach

Java Code:

```
// Java program to check the authentication of the user
// Importing the modules

import java.util.*;

// Gfg Class
public class Gfg
{
   // Main CLass
   public static void main(String args[])
   {
      // Declaring the username and password
      String user_name = "Sandeep Kamble";
      String password = "Sandeep Kamble";

      // Checking the validity of the input
      if(user_name.equals("Sandeep Kamble") && password.equals("Sandeep Kamble"))
      {
         // Printing Output
         System.out.println("Authentication Successful");
      }
      else
      {
         // Printing Output
         System.out.println("User name/ Password not matching");
      }
   }
}
```

Output:

Authentication Successful

Login Form Java:

In Java, a form for entering authentication credentials to access the restricted page is referred to as a Login form. A login form contains only two fields, i.e., username and password. Each user should have a unique username that can be an email, phone number, or any custom username.

After submitting the login form, the underlying code of the form checks whether the credentials are authentic or not to allow the user to access the restricted page. If the users provide unauthentic credentials, they will not be able to forward the login form.

Steps to create login form:

In order to create a login form in Java, we have to follow the following steps:

1. Create a class that uses the JFrame and ActionListener to design the login form and perform the action.
2. Create user interface components using swings and awt and add them to the panel.
3. Override the actionPerformed() method that will call on the button click.
4. In this method, we will verify the user entered credentials.
5. Create a new page using JFrame and navigate the user to it if the credentials are authentic.
6. Else show an error message to the user.

Let's follow the above steps and implement the login form using swing and awt in Java:

LoginFormDemo.java

```
1. //import required classes and packages
2. import javax.swing.*;
3. import java.awt.*;
4. import java.awt.event.*;
5. import java.lang.Exception;
6.
7. //create CreateLoginForm class to create login form
8. //class extends JFrame to create a window where our component add
9. //class implements ActionListener to perform an action on button clic
   k
```

```
10. class CreateLoginForm extends JFrame implements ActionListener
11. {
12.     //initialize button, panel, label, and text field
13.     JButton b1;
14.     JPanel newPanel;
15.     JLabel userLabel, passLabel;
16.     final JTextField  textField1, textField2;
17.
18.     //calling constructor
19.     CreateLoginForm()
20.     {
21.
22.         //create label for username
23.         userLabel = new JLabel();
24.         userLabel.setText("Username");      //set label value for textField
    1
25.
26.         //create text field to get username from the user
27.         textField1 = new JTextField(15);    //set length of the text
28.
29.         //create label for password
30.         passLabel = new JLabel();
31.         passLabel.setText("Password");      //set label value for textField
    2
32.
33.         //create text field to get password from the user
34.         textField2 = new JPasswordField(15);    //set length for the pass
    word
35.
36.         //create submit button
```

```
37.         b1 = new JButton("SUBMIT"); //set label to button
38.
39.         //create panel to put form elements
40.         newPanel = new JPanel(new GridLayout(3, 1));
41.         newPanel.add(userLabel);    //set username label to panel
42.         newPanel.add(textField1);   //set text field to panel
43.         newPanel.add(passLabel);    //set password label to panel
44.         newPanel.add(textField2);   //set text field to panel
45.         newPanel.add(b1);           //set button to panel
46.
47.         //set border to panel
48.         add(newPanel, BorderLayout.CENTER);
49.
50.         //perform action on button click
51.         b1.addActionListener(this);     //add action listener to button
52.         setTitle("LOGIN FORM");         //set title to the login form
53.     }
54.
55.     //define abstract method actionPerformed() which will be called on
    button click
56.     public void actionPerformed(ActionEvent ae)     //pass action listen
    er as a parameter
57.     {
58.         String userValue = textField1.getText();        //get user entered us
    ername from the textField1
59.         String passValue = textField2.getText();        //get user entered p
    asword from the textField2
60.
61.         //check whether the credentials are authentic or not
62.         if (userValue.equals("test1@gmail.com") && passValue.equals(
    "test")) {  //if authentic, navigate user to a new page
```

```
63.
64.             //create instance of the NewPage
65.             NewPage page = new NewPage();
66.
67.             //make page visible to the user
68.             page.setVisible(true);
69.
70.             //create a welcome label and set it to the new page
71.             JLabel wel_label = new JLabel("Welcome: "+userValue);
72.             page.getContentPane().add(wel_label);
73.         }
74.         else{
75.             //show error message
76.             System.out.println("Please enter valid username and password
    ");
77.         }
78.     }
79. }
80. //create the main class
81. class LoginFormDemo
82. {
83.     //main() method start
84.     public static void main(String arg[])
85.     {
86.         try
87.         {
88.             //create instance of the CreateLoginForm
89.             CreateLoginForm form = new CreateLoginForm();
90.             form.setSize(300,100);  //set size of the frame
91.             form.setVisible(true);  //make form visible to the user
```

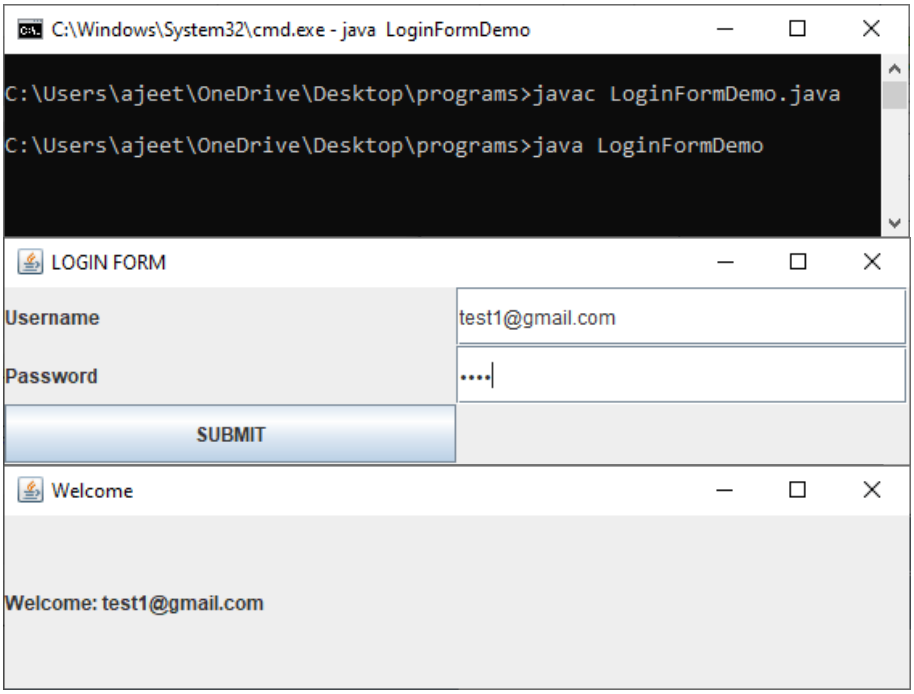
```
92.         }
93.         catch(Exception e)
94.         {
95.             //handle exception
96.             JOptionPane.showMessageDialog(null, e.getMessage());
97.         }
98.     }
99. }
```

NewPage.java:

```
1.  //import required classes and packages
2.  import javax.swing.*;
3.  import java.awt.*;
4.
5.  //create NewPage class to create a new page on which user will naviga
    te
6.  class NewPage extends JFrame
7.  {
8.      //constructor
9.      NewPage()
10.     {
11.         setDefaultCloseOperation(javax.swing.
12.         WindowConstants.DISPOSE_ON_CLOSE);
13.         setTitle("Welcome");
14.         setSize(400, 200);
15.     }
16. }
```

Output:

Now, when we run the **LoginFormDemo.java** class, a panel will be open having the label, text fields, and button. We enter the credentials and hit on the submit button. If the credentials are authentic, the login form will navigate us to the welcome page as described below:



7.2 SUMMARY

Identity management in cloud computing is highly critical to your organization. It can persuade the productivity of your employees and the security of your organization. It can also have immense control over what technology solutions you select.

However, IAM solutions have to be supple across identity management and access control in cloud computing to match the current complexities of the computing environment. If you are locked into some conventional platforms or service providers because of your active directory ad service, explore a vendor-neutral cloud identity management solution.

7.3 REFERENCES

1) The Basics of Hacking and Penetration Testing

2) Hacking: The Art of Exploitation

3) The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws

7.4 UNIT END EXERCISES

Create a Login form on your own using java.

*****

MODULE VIII

8

APP DEVELOPMENT USING CLOUD COMPUTING

Unit Structure

8.0 Objective
8.1 Introduction
8.2 Summary
8.3 References
8.4 Unit End Exercises

8.0 OBJECTIVE

Application development is the process of creating a computer program or a set of programs to perform the different tasks that a business requires. From calculating monthly expenses to scheduling sales reports, applications help businesses automate processes and increase efficiency.

Cloud application development is a hot topic of 2020. The cloud approach gives companies lots of valuable benefits: development cost reduction, (no need for hardware, servers, or even some software), higher accessibility of the final product, a new level of standardization, and scaling opportunities

8.1 INTRODUCTION

The world has seen an important increase in the demand for Cloud-based applications. This has in turn increased the demand for Cloud application development. As a result, the past few years have had a consolidation of the Cloud computing market.

Cloud apps and services are used, directly or indirectly, by almost everyone. Businesses have also increased their use of Cloud-based applications and services, even if they sometimes don't know it. If you use SaaS tools, you are surely using a Cloud app. However, Cloud apps are more than just that.

For many, Cloud apps are still a mystery — one that we plan to explain throughout this article. As an app development company, we know it is important for any business to properly make use of Cloud services. If you want to understand what Cloud computing and Cloud application development are, how your company can benefit from them, or even if you just want examples of Cloud apps, this article is for you.

What Is the 'Cloud'?

The 'Cloud' refers to HiTech computing services that travel over the internet to some servers in a different location. This IT infrastructure is usually managed by a third party who charges a fee in exchange for the computing power and other cloud-based services. In general, **Cloud services allow companies to hire the computing power they need in a flexible way** and without having to directly own or manage the IT infrastructure themselves.

This technology and its associated services have been gaining popularity because of the benefits they bring. Thanks to fast internet connections and efficient computers, it is now possible to make information move fast enough to have Cloud-based apps that feel almost as if the computing action occurred natively in the device. Thanks to 5G connections, Cloud computing is almost resembling Edge computing, helping develop better and more powerful IoT systems.

Thanks to reduced latency, it is possible to transfer information fast enough from one place to another in such a way that no delay is felt by the user. **Latency** is the delay that occurs between the action of a user and an app's response. A reduced latency allows for more real-time and fast-response apps, opening up all sorts of possibilities for businesses through better software.

The Cloud market has become more important for a variety of industries throughout 2020 due to the increased use of tools like Zoom and Google Meet, which are used by individuals and remote-ready companies alike, but also to Software as a Service (SaaS) apps like Netflix and Spotify which are used by people all over the world.

By avoiding the need to own, manage, and configure their own IT infrastructure through outsourcing, companies can focus on their core purpose. This has been a game-changer for many software-based companies and their IT-dependent business models.

What Is a 'Cloud-Based Application'?

Cloud-based applications, also known as Cloud apps, seem to be taking over. In theory, a Cloud app is one that uses Cloud-based services. So, whether an app is mobile or web, they probably use some sort of Cloud service. What really differentiates a Cloud app from a native one is the extent to which they use Cloud services.

Increased dependence on the Cloud's processing power is the result of companies building innovative and creative solutions to all sorts of problems that use technology to do things that were previously impossible. Thanks to the ability to process large amounts of data (Big Data) through third party owned IT infrastructure, companies can perform massive calculations and deliver top services.

In particular, Cloud services have opened up the possibility for many web-based Cloud applications, also known as web apps. A web app is one where most of the computation occurs in the Cloud, not on the device itself, and usually built with the use of Cloud application development services. A new form of web app, known as a Progressive Web App (PWA), is also seeing an increase in popularity.

Benefits of a Cloud App:

Cloud application development offers various benefits for businesses that wish to use technology to solve a problem. Some of the benefits are:

- **Improved app performance:** as more computations are performed on the server side of an app, users will experience a faster and more reliable service.
- **Increased uptime:** thanks to the reliability of Cloud services, a Cloud-based application will remain up easier than through your own IT infrastructure.
- **Scalability:** businesses can hire on-demand the processing power they need, being this very convenient for moments of high computer processing demand.
- **Update software easily:** through Cloud technologies, it is possible to update an app easily through a massive deployment.
- **Security:** Cloud services help reduce the risk of physical IT infrastructure failure.

Cloud Application Development: Developing Applications for the Cloud:

Cloud application development is the process through which a Cloud-based app is built. It involves different stages of software development, each of which prepares your app to go live and hit the market. The best Cloud app development teams use DevOps practices and tools like Kubernetes. However, an experienced app development company should ideally be technology agnostic, which means being able to build your Cloud app using any technology you prefer. Most apps built using the Cloud are highly dependent on the Cloud to operate.

Application development on Cloud infrastructure allows web and PWA development services to **reduce development costs**, opens up the possibility to work with remote teams, and reduces project times if used correctly with software development methodologies like Agile. However, not all companies are experienced enough to perform many complex aspects of the app development process using the Cloud. Businesses looking to develop digital products like web-based Cloud applications need to make sure that they work with a trusted Cloud-experienced app development company.

Although some businesses have their own Cloud development teams, most will hire an app development company with experience in Cloud services. A great way to verify an app development company's experience with the Cloud is through certifications like AWS. Koombea, for example, is a certified AWS partner.

Cloud Application Example:

Many of the apps we use on a day-to-day basis use the Cloud in one way or another. Cloud application development has resulted in amazing tools and services like:

- **Miro:** a virtual board where you can work with other users in a number of fun and creative ways.
- **Figma:** a powerful Cloud-based design app that is gaining many fans thanks to its collaborative nature.
- **Dropbox or Google Drive:** easily store your files on the Cloud and make them available for others, wherever they are.

Collaboration is one element that stands out from most Cloud-based apps. Although there are other important ones, the possibility to collaborate with users from all over the world, even in real-time, is one major advantage of Cloud apps.

How to Develop a Cloud Application:

Thanks to app development services, it is now possible for all sorts of businesses to develop a Cloud-based application. At Koombea, we've been building Cloud applications for companies throughout different industries, always helping our clients understand their business model and how it can make use efficiently of the Cloud to maximize their goals.

What is a cloud-based app?

A cloud-based application is a software application that is deployed in a cloud environment.

Every application has a user interface (the part the user sees and interacts with) and a back end (the part that processes data and makes the app's functions work).

In common mobile applications, data and business logic are processed by a smartphone and a computer processor. In cloud applications, these tasks are performed by a remote server. Cloud application development is beneficial because most of the data storage is located on a remote server.

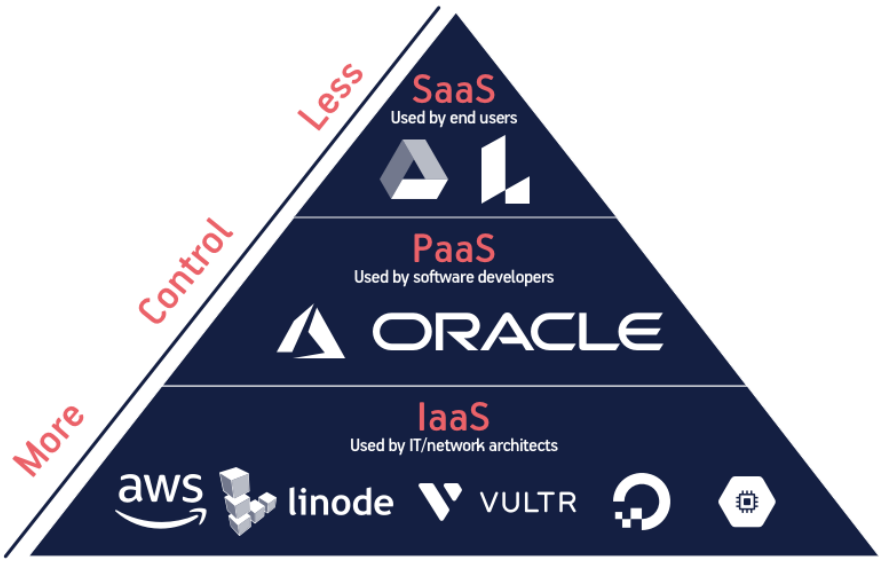
Characteristics of cloud-based apps:

- Application data is stored in cloud infrastructure, so there are minimal device requirements to run cloud applications.

- You can store data locally so an application can work offline. As soon as the device is back online, the app will automatically sync with the cloud.
- Customers can use a cloud-based app from any internet-connected device such as a smartphone, tablet, or laptop. All information is stored in the cloud, so users can pick up where they left off on any device.

There are three types of cloud-based apps: SaaS, IaaS, and PaaS. Let's figure out what each of them stands for.

Types of cloud-based apps:



Let's see the three types of cloud applications:

Software as a Service (SaaS):

A cloud-based SaaS solution can be used through mobile apps and web browsers. The SaaS model allows customers to use an application without installing and configuring it. With the internet, you can use SaaS solutions around the world from any device.

Companies tend to use SaaS office software like G Suite and messengers like Slack. However, many people also use services like Dropbox for personal use.

Platform as a Service (PaaS):

PaaS solutions offer everything necessary for application development. PaaS relies on a cloud service provider for development tools, infrastructure, and operating systems. PaaS vendors provide software and hardware tools to simplify the development process.

PaaS solutions can include:

- Development tools

- Middleware
- Operating systems
- Database management tools
- Infrastructure

Windows Azure, Heroku, and OpenShift are a few examples of services that use the PaaS cloud computing model.

Infrastructure as a Service (IaaS)

With a IaaS solution, a service provider manages your business's infrastructure — servers, network, and storage — through a public or private cloud.

Business owners can access IaaS infrastructure with an API or admin panel.

With the IaaS model, you can manage operating systems and applications while vendors such as AWS, Microsoft Azure, and DigitalOcean provide you with hardware, networks, hard drives, storage, and servers.

Cloud app development: Key differences:

What about the specifics of developing cloud applications?

- Developing a cloud application requires deep interaction between programmers, data architects, designers, and quality assurance managers. Developers need to be familiar with various cloud platforms such as Amazon Web Services (AWS), Microsoft Azure, Force.com, and Apache CloudStack. In addition, they should consider connectivity with APIs (application programming interfaces) and CDNs (content delivery networks).
- Your development team must consider that your final solution should be scalable. One of the important reasons why companies choose to store their data in the cloud is that cloud storage is easy to expand, even within a short period of time.
- Cloud applications can be technically unlimited in size, but cloud hosting isn't free. Concentrating user requests and optimizing data size should be top development priorities.
- To convince people to use your application, you need to ensure their data is stored securely, which isn't always easy since you're dealing with cloud technologies that don't have a single data store. This is why an app's codebase should be kept separate from other data.

Cloud-based application development step by step:

Developing a cloud application is different from developing a web or mobile application. A mobile app development team builds a cloud

solution by relying on your chosen service provider. Amazon Web Services (AWS) is probably the best and most reliable solution on the market right now. It offers a number of great tools and features for developing cloud applications.

You should be willing to invest time and money in creating any digital product. Cloud solutions are no exception. Before you start development, you need to understand the problems your app users face and find a way to solve them using your product.

Step #1. Research your app's target market:

When developing a cloud-based mobile app, the first thing you need to consider is the target audience. Understanding your users' needs makes development easier and leads to a better final product. Find out as much as you can about your potential users. You can start by researching the following:

- **Demographics**. Find out the average age of your users, where they live, what devices they use, etc.
- **Behavioral trends**. Find out what decreases a user's desire to download an app, your users' security expectations, and so on.

To develop an amazing mobile application, we suggest creating a user persona, or a detailed portrait of your ideal user.

Step #2. Hire a development team:

The second step is to find a development team you want to work with. The first phase of development will include business analysis, developing a technical specification, estimating the development cost, and planning the project.

Before diving into the actual development of your mobile app, you and your team should create an app development workflow, choose the main features for the app, and design an app concept. Then your development team should create project milestones and start working on the MVP.

Step #3. Consider the architecture and features:

For your application to be successful, you need to carefully consider the architecture and service model. These decisions affect your application's performance, so it's best to consult with specialists who can advise you.

Architecture:

It's a good idea to create an advanced data architecture. Classic solutions are always reliable. However, for cloud applications, a microservices architecture is commonly used.

Service model:

The service model you select — SaaS, PaaS, IaaS — must match the type of cloud solution you're developing. For example, when developing an application like Slack, you need to take a SaaS approach.

Step #4. Define the tech stack:

When choosing tools for developing cloud applications, you should consult with experts. They'll analyze your requirements, features, and designs to select the right set of technologies for your product. Also, be mindful of your application's scalability to keep your solution up to date.

Here's a possible tech stack for a cloud-based application:

Application and data:

- Akamai
- Cloudant
- ClearDB

Utilities:

- Google Analytics
- Twilio
- Optimizely
- Heap
- Recurly
- Zuora
- Cyfe
- TransmogrifAI

DevOps:

- Jenkins
- Bitbucket
- New Relic
- Datadog
- Puppet Labs
- Cloud9 IDE
- Sauce Labs

- StillAlive

Business tools:

- Jira
- G Suite
- InVision
- Salesforce Sales Cloud
- Balsamiq
- DocuSign
- UXPin

Mobile and Web App Development:

Step #5. Choose a monetization model:

The next step is to choose the right monetization model for your mobile application. Now that you know your users' needs, you can predict what your users will pay for. Here are three monetization models to choose from:

- **Paid**. This monetization model is quite straightforward: users pay once to access your app.
- **Freemium**. With this model, users can download your app for free. They can then pay to upgrade their accounts or use premium features.
- **In-app purchases**. With in-app purchases, users can pay for different items, features, or content inside the app.
- **Advertising**. You can choose one of the following ad options:
    - **Cost per click:** Charge advertisers every time a user interacts with their ads in your app.
    - **Cost per mille:** Charge advertisers for every 1,000 ad impressions in your app.
    - **Cost per action:** Charge advertisers only when users complete a target action, such as installing an app or signing up for a newsletter.

Step #6. Create an MVP:

Creating a cloud-based app is a big and complex project. We recommend launching a minimum viable product (MVP) first and testing its technical and business performance. By using an MVP approach, you'll be able to find out what users like and don't like in your app. Then you'll be able to consider their feedback and improve your app.

Step #7. Test your product carefully:

Cloud-based app development should include a testing stage. Before launching your product, your development team has to test it to find any bugs.

At this point, you'll verify that your application is working correctly and provides a satisfying user experience. To do this, it's best to cooperate with a full-cycle development company.

Full-cycle development companies offer development, design, testing, and management services. With one team working on your project from start to finish, communication is vastly simplified. This results in higher product quality.

Step #8. Launch the app and keep it up to date:

You can release your app on the App Store (iOS) and Google Play (Android). Google Play uses automated testing to speed up the app store approval process. However, if your application is rejected by Google, it can be difficult to find out why.

The App Store delegates app validation to real people. If validators don't approve your app, they'll ask you to make specific changes.

If you want to distribute your app exclusively within your organization via the App Store, you'll need to pay $299 a year to join the Apple Developer Enterprise Program. Google Play doesn't charge for its analogous service. Here's a list of information you need to prepare before submitting your application.

For Google Play:

- Screenshots
- App name
- Description keywords
- Support URL
- Marketing URL
- Privacy policy URL
- App icon
- Categories
- Rating
- Copyright
- Demo account
- Version information

- Pricing information

For the App Store:

- Title (app name)
- Short description
- Full description
- Screenshots
- High-resolution icon
- Featured graphic
- Promo video (optional)
- Type and category
- Content rating
- Languages and translations (if any)
- Contact details
- Privacy policy
- Compatible devices
- Pricing and distribution

Some of the materials listed will cost you nothing to produce, while others will be quite expensive. Creating a copyright and privacy policy usually takes time and expensive legal services. How much does it cost to list an app on the App Store and Google Play if a development company helps you? Releasing an application can take different amounts of time depending on the amount of work the company has to do.

Also, remember that before your app is published on either app store, it must go through an approval process. This procedure can take some time and require additional development costs. If your app doesn't meet platform rules or requirements, it won't be accepted.

If your app isn't accepted, you may need to make a few changes in order to get it approved. Some mobile app development companies provide their services until your app gets approved, but others don't.

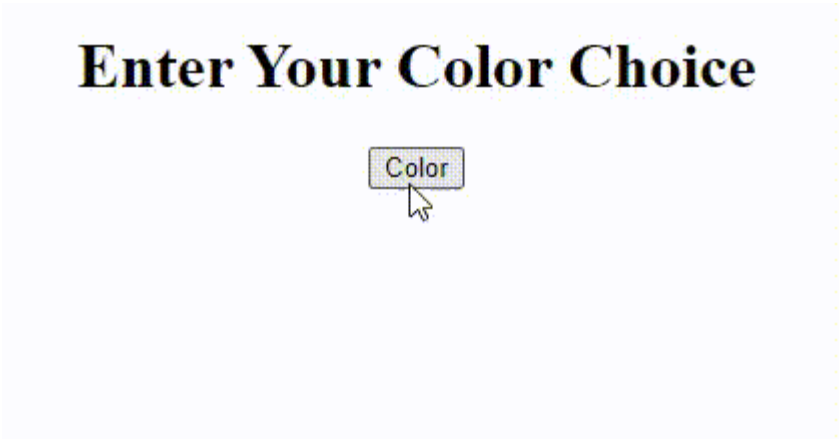
Example 1: Dynamically changing the background color of a webpage on each click

HTML

```
<!DOCTYPE HTML>
<html>
```

```
<head>
   <script src=
"https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
   </script>
</head>

<body style="text-align:center;" id="body">
   <h1>Enter Your Color Choice</h1>

   <button type="button" onclick="changecolor()">
      Color
   </button>

   <script>
      function changecolor()
      {
         // Generating random color each time
         var color = "#"+(Math.random()*16777215|0).toString(16);

         $("body").css("background-color",color);
      }
   </script>
</body>

</html>
```

Output:



8.2 SUMMARY

Cloud computing is the delivery of computing resources — including storage, processing power, databases, networking, analytics, artificial intelligence, and software applications — over the internet (the cloud).

8.3 REFERENCES

1) The Basics of Hacking and Penetration Testing

2) Hacking: The Art of Exploitation

3) The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws

8.4 Unit End Exercises

Write a code to take username as input and dynamically changing the text content of web page.

*****