**Name – Shriya Pathak**

**Roll no – 61**

**Class – BE(AI&DS)**

**Subject – Computer Laboratory III (Computational Intelligence)**

**Assignment No 01**

**Title: -** Design a distributed application using RPC for remote computation where client submits

an integer value to the server and server calculates factorial and returns the result to the client.

**Objective: -**

1. Implement a distributed application using Remote Procedure Call (RPC) to allow a client

to submit an integer value to the server.

2. Enable the server to calculate the factorial of the received integer and return the result to

the client program.

3. Demonstrate the functionality and efficiency of RPC in remote computation tasks.

**Outcome: -**

• Successfully implement an RPC-based client-server architecture for remote factorial

calculation.

• Demonstrate the ease of implementing distributed applications using RPC.

**Requirements: -**

• Python (3.x recommended)

• Jupyter Notebook or any Python IDE

**Hardware Requirements: -**

• A machine with sufficient RAM and processing power for model training (8GB RAM

recommended)

**Prerequisite: -**

• Basic understanding of Python programming

• Familiarity with the concepts of RPC

**Theory: -**

**What is RPC?**

Remote Procedure Call (RPC) is an intercrosses communication technique. The Full form of

RPC is Remote Procedure Call. It is used for client-server applications. RPC mechanisms are

used when a computer program causes a procedure or subroutine to execute in a different

address space, which is coded as a normal procedure call without the programmer specifically

coding the details for the remote interaction.

This procedure call also manages low-level transport protocol, such as User Datagram

Protocol, Transmission Control Protocol/Internet Protocol etc. It is used for carrying the

message data between programs.

**Types of RPC**

Three types of RPC are:

• Callback RPC

• Broadcast RPC

• Batch-mode RPC

**Callback RPC**

This type of RPC enables a P2P paradigm between participating processes. It helps a process to

be both client and server services.

Functions of Callback RPC:

• Remotely processed interactive application problems

• Offers server with clients handle

• Callback makes the client process wait

• Manage callback deadlocks

• It facilitates a peer-to-Peer paradigm among participating processes.

**Broadcast RPC**

Broadcast RPC is a client’s request, that is broadcast on the network, processed by all servers

which have the method for processing that request.

Functions of Broadcast RPC:

• Allows you to specify that the client’s request message has to be broadcasted.

• You can declare broadcast ports.

• It helps to reduce the load on the physical network

**Batch-mode RPC**

Batch-mode RPC helps to queue, separate RPC requests, in a transmission buffer, on the client-

side, and then send them on a network in one batch to the server.

Functions of Batch-mode RPC:

• It minimizes overhead involved in sending a request as it sends them over the network in

one batch to the server.

• This type of RPC protocol is only efficient for the application that needs lower call rates.

• It needs a reliable transmission protocol.

RPC Architecture

RPC architecture has mainly five components of the program:

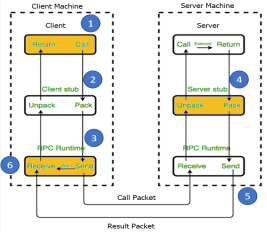
1. Client

2. Client Stub

3. RPC Runtime

4. Server Stub

5. Server

****

**How RPC Works?**

Following steps take place during the RPC process:

Step 1) The client, the client stub, and one instance of RPC run time execute on the client

machine.

Step 2) A client starts a client stub process by passing parameters in the usual way. The client

stub stores within the client’s own address space. It also asks the local RPC Runtime to send

back to the server stub.

Step 3) In this stage, RPC accessed by the user by making regular Local Procedural Cal. RPC

Runtime manages the transmission of messages between the network across client and server.

It also performs the job of retransmission, acknowledgment, routing, and encryption.

Step 4) After completing the server procedure, it returns to the server stub, which packs

(marshalls) the return values into a message. The server stub then sends a message back to the

transport layer.

Step 5) In this step, the transport layer sends back the result message to the client transport

layer, which returns back a message to the client stub.

Step 6) In this stage, the client stub demarshalls (unpack) the return parameters, in the resulting

packet, and the execution process returns to the caller.

**Characteristics of RPC**

Here are the essential characteristics of RPC:

• The called procedure is in another process, which is likely to reside in another machine.

• The processes do not share address space.

• Parameters are passed only by values.

• RPC executes within the environment of the server process.

• It doesn’t offer access to the calling procedure’s environment.

**Features of RPC**

Here are the important features of RPC:

• Simple call syntax

• Offers known semantics

• Provide a well-defined interface

• It can communicate between processes on the same or different machines

Advantages of RPC

**Here are Pros/benefits of RPC:**

• RPC method helps clients to communicate with servers by the conventional use of

procedure calls in high-level languages.

• RPC method is modeled on the local procedure call, but the called procedure is most

likely to be executed in a different process and usually a different computer.

• RPC supports process and thread-oriented models.

• RPC makes the internal message passing mechanism hidden from the user.

• The effort needs to re-write and re-develop the code is minimum.

• Remote procedure calls can be used for the purpose of distributed and the local

environment.

• It commits many of the protocol layers to improve performance.

• RPC provides abstraction. For example, the message-passing nature of network

communication remains hidden from the user.

• RPC allows the usage of the applications in a distributed environment that is not only in

the local environment.

• With RPC code, re-writing and re-developing effort is minimized.

• Process-oriented and thread-oriented models support by RPC.

**Disadvantages of RPC**

Here are the cons/drawbacks of using RPC:

• Remote Procedure Call Passes Parameters by values only and pointer values, which is

not allowed.

• Remote procedure calling (and return) time (i.e., overheads) can be significantly lower

than that for a local procedure.

• This mechanism is highly vulnerable to failure as it involves a communication system,

another machine, and another process.

• RPC concept can be implemented in different ways, which is can’t standard.

• Not offers any flexibility in RPC for hardware architecture as It is mostly interaction-

based.

• The cost of the process is increased because of a remote procedure call.

RPC Architecture To implement this in Python, we can use the xmlrpc library, which provides

support for writing RPC servers and clients. The server program will create an XML-RPC

server using SimpleXMLRPCServer, register a function to compute the factorial, and then

start the server to listen for incoming requests. The client program will create an XML-RPC

proxy object to communicate with the server, then call the remote procedure with the integer

value as an argument to request the factorial calculation.

**Conclusion: -** Thus, implemented distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

**Code: -**

1) server.py

from xmlrpc.server import SimpleXMLRPCServer

def factorial(n):

fact = 1

for i in range(1,n+1):

fact=fact\*i

return fact

server = SimpleXMLRPCServer(("localhost", 8000), logRequests=True)

server.register\_function(factorial, "factorial\_rpc")

try:

print("Starting and listening on port 8000...")

print("Press Ctrl + C to exit.")

server.serve\_forever()

except:

print("Exit.")

2) client.py

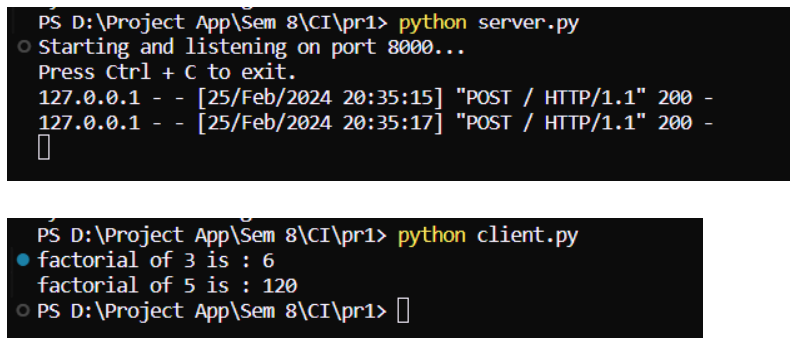
import xmlrpc.client

proxy = xmlrpc.client.ServerProxy("http://localhost:8000/")

print("factorial of 3 is : %s" % str(proxy.factorial\_rpc(3)))

print("factorial of 5 is : %s" % str(proxy.factorial\_rpc(5)))

**Output: -**

****

**Name – Shriya Pathak**

**Roll no – 61**

**Class – BE(AI&DS)**

**Subject – Computer Laboratory III (Computational Intelligence)**

**Assignment No 02**

**Title: -** Distributed String Concatenation Application using RMI.

**Objective: -**

1. Develop a client-server application using Remote Method Invocation (RMI) for remote computation.

2. Enable the client to submit two strings to the server for concatenation.

3. Ensure that the server concatenates the given strings and returns the result to the client program.

**Software Requirements: -**

• Jupyter Notebook, any Java IDE

• A machine with at least 8GB of RAM is recommended for model training.

• A multi-core CPU is suitable, and for faster training, a GPU (Graphics Processing Unit) is

highly recommended.

**Prerequisites: -**

• Basic understanding of Java programming

**Theory: -**

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed

application in java. The RMI allows an object to invoke methods on an object running in another

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

**Understanding stub and skeleton**

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

A remote object is an object whose method can be invoked from another JVM. Let's understand the stub and

**Skeleton objects:**

**Stub**

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It

resides at the client side and represents the remote object. When the caller invokes method on the stub object, it

does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),

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

3. It waits for the result

4. It reads (unmarshals) the return value or exception, and

5. It finally, returns the value to the caller.

**skeleton**

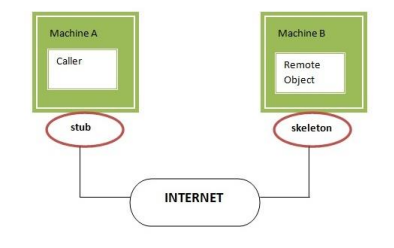
The skeleton is an object, acts as a gateway for the server-side object. All the incoming requests are routed

through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method

2. It invokes the method on the actual remote object, and

3. It writes and transmits (marshals) the result to the caller. In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications -

If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method

2. It need to provide the communication with the remote objects, and

3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

Java RMI Example

The is given the 6 steps to write the RMI program.

1. Create the remote interface

2. Provide the implementation of the remote interface

3. Compile the implementation class and create the stub and skeleton objects using the rmic tool

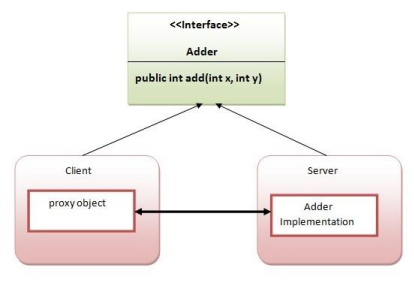
4. Start the registry service by rmiregistry tool

5. Create and start the remote application

6. Create and start the client application

RMI Example

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 interact 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.

1. import java.rmi.\*;

2. public interface Adder extends Remote{

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

4. }

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

o Either extend the UnicastRemoteObject class,

o or use the exportObject() method of the UnicastRemoteObject class

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

RemoteException.

1. import java.rmi.\*;

2. import java.rmi.server.\*;

3. public class AdderRemote extends UnicastRemoteObject implements Adder{

4. AdderRemote()throws RemoteException{

5. super();

6. }

7. public int add(int x,int y){return x+y;}

8. }

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.

1. 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.

1. 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.

**Conclusion: -** Thus, this application demonstrates the use of RMI to create a distributed application for string concatenation, where the server receives two strings from the client, concatenates them, and returns the result to the client.

**Code: -**

1. interface AddRem.java

import java.rmi.\*;

public interface AddRem extends Remote{

public int addNum(int a,int b) throws RemoteException;

}

2. class AddRemImpl.java

import java.rmi.\*;

import java.rmi.server.UnicastRemoteObject;

public class AddRemImpl extends UnicastRemoteObject implements AddRem {

public AddRemImpl() throws RemoteException{}

public int addNum(int a,int b)

{

return(a+b);

}

}

3. Client.java

import java.rmi.\*;

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Client {

public static void main(String[] args) {

String host="localhost";

Scanner sc=new Scanner(System.in);

System.out.println("Enter 1st number: ");

int a=sc.nextInt();

System.out.println("Enter 2st number: ");

int b=sc.nextInt();

try{

AddRem remoobj=(AddRem)Naming.lookup("rmi://"+host+"/AddRem");

System.out.print("Nirmal ID:");

System.out.println(remoobj.addNum(a,b));

}

catch (RemoteException re)

{

re.printStackTrace();

}

catch (NotBoundException nbe)

{

nbe.printStackTrace();

}

catch (MalformedURLException mfe)

{

mfe.printStackTrace();

}

}}

4. Server.java

import java.rmi.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) {

try{

AddRemImpl locobj=new AddRemImpl();

Naming.rebind("rmi:///AddRem",locobj);

}catch (RemoteException e)

{

e.printStackTrace();

}

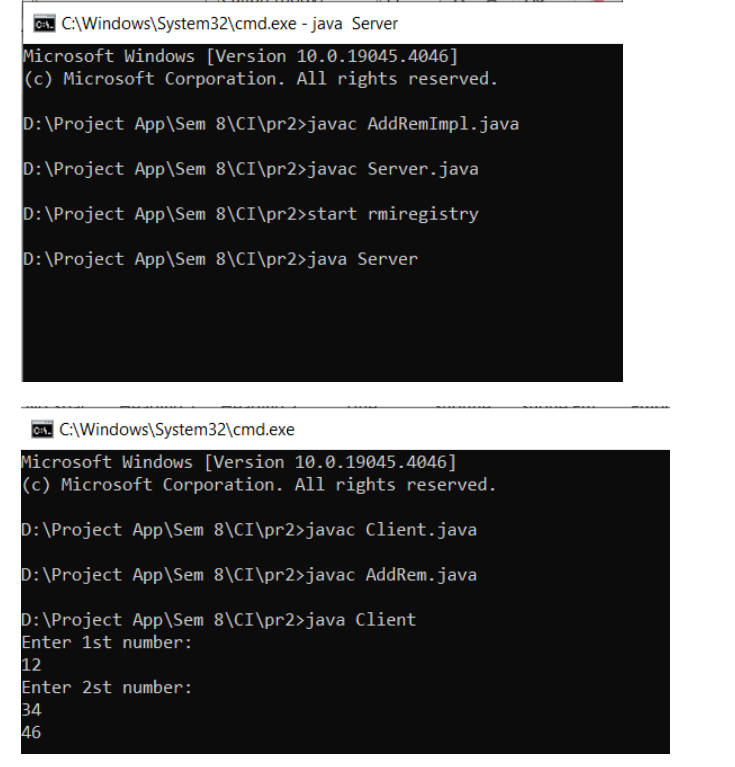
catch (MalformedURLException mfe)

{

mfe.printStackTrace();

}}}

**Output: -**



**Name – Shriya Pathak**

**Roll no – 61**

**Class – BE(AI&DS)**

**Subject – Computer Laboratory III (Computational Intelligence)**

**Assignment No 03**

**Title: -** Distributed Application using MapReduce for Text Analysis:

a) Character counting in a given text file.

b) Counting no. of occurrences of every word in a given text file.

**Objective: -**

1. Implement a MapReduce program under Hadoop for character counting in a given text file.

2. Develop a MapReduce program under Hadoop for counting the number of occurrences of every word in a given text file.

**Outcome: -**

1. Successfully count the number of characters in the given text file using MapReduce.

2. Count the occurrences of every word in the given text file using MapReduce.

**Software Requirement: -**

• Python (3.x recommended)

• Jupyter Notebook or any Python IDE

**Hardware Requirement: -**

A machine with sufficient RAM and processing power for model training (8GB RAM recommended)

**Prerequisites: -**

Counting the number of words in any language is a piece of cake like in C, C++, Python, Java, etc. MapReduce also uses Java but it is very easy if you know the syntax on how to write it.

It is the basic of MapReduce. You will first learn how to execute this code similar to “Hello World” program in other languages. So here are the steps which show how to write a MapReduce code for Word Count.

**What is MapReduce?**

MapReduce is a programming model and framework within the Hadoop ecosystem that enables efficient processing of big data by automatically distributing and parallelizing the computation. It consists of two fundamental tasks: Map and Reduce.

In the Map phase, the input data is divided into smaller chunks and processed independently in parallel across multiple nodes in a distributed computing environment. Each chunk is transformed or “mapped” into key-value pairs by applying a user-defined function. The output of the Map phase is a set of intermediate key-value pairs.

The Reduce phase follows the Map phase. It gathers the intermediate key-value pairs generated by the Map tasks, performs data shuffling to group together pairs with the same key, and then applies a user-defined reduction function to aggregate and process the data. The output of the Reduce phase is the final result of the computation.

Map Reduce example allows for efficient processing of large-scale datasets by leveraging parallelism and distributing the workload across a cluster of machines. It simplifies the development of distributed data processing applications by abstracting away the complexities of parallelization, data distribution, and fault tolerance, making it an essential tool for big data processing in the Hadoop ecosystem.

**Advantages of using MapReduce**

The advantages of using MapReduce are as follows:

• MapReduce can define mapper and reducer in several different languages using Hadoop streaming.

• MapReduce facilitates automatic parallelization and distribution, reducing the time required to run programs.

• MapReduce provides fault tolerance by re-executing, writing map output to a distributed file system, and restarting failed map or reducer tasks.

• Processing of data using MapReduce is a cost-effective solution.

• MapReduce processes large volumes of unstructured data very quickly.

• Using HDFS and HBase security, Map Reduce ensures data security by allowing only approved users to access data stored in the system.

• MapReduce programming utilizes a simple programming model to handle tasks more efficiently and quickly and is easy to learn.

• MapReduce is flexible and works with several Hadoop languages to handle and store data.

**MapReduce Architecture**

Map Reduce example process has the following phases:

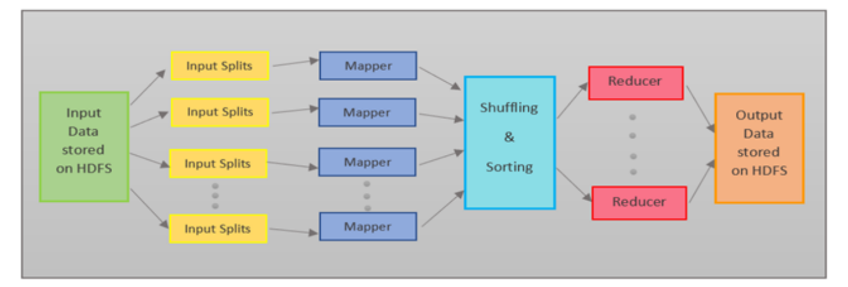
1. Input Splits

2. Mapping

3. Shuffling

4. Sorting

5. Reducing



**Input Splits**

MapReduce splits the input into smaller chunks called input splits, representing a block of work with a single mapper task.

**Mapping**

The input data is processed and divided into smaller segments in the mapper phase, where the number of mappers is equal to the number of input splits. RecordReader produces a key-value pair of the input splits using TextFormat, which Reducer later uses as input. The mapper then processes these key-value pairs using coding logic to produce an output of the same form.

**Shuffling**

In the shuffling phase, the output of the mapper phase is passed to the reducer phase by removing duplicate values and grouping the values. The output remains in the form of keys and values in the mapper phase. Since shuffling can begin even before the mapper phase is complete, it saves time.

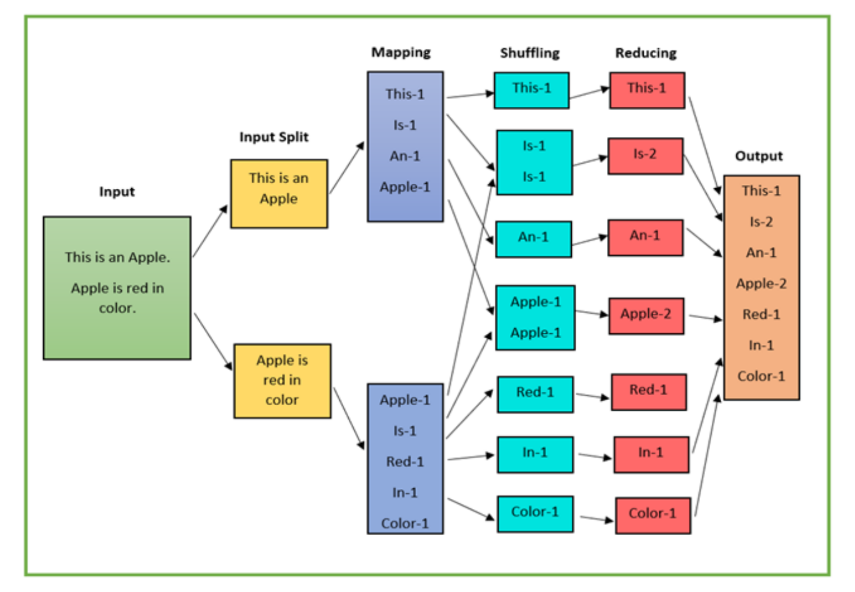
**Sorting**

Sorting is performed simultaneously with shuffling. The Sorting phase involves merging and sorting the output generated by the mapper. The intermediate key-value pairs are sorted by key before starting the reducer phase, and the values can take any order. Sorting by value is done by secondary sorting.

**Reducing**

In the reducer phase, the intermediate values from the shuffling phase are reduced to produce a single output value that summarizes the entire dataset. HDFS is then used to store the final output.

Here’s an Map Reduce example to count the frequency of each word in an input text. The text is, “This is an apple. Apple is red in color.”



• The input data is divided into multiple segments, then processed in parallel to reduce processing time.

In this case, the input data will be divided into two input splits so that work can be distributed over all the map nodes.

• The Mapper counts the number of times each word occurs from input splits in the form of key-value pairs where the key is the word, and the value is the frequency.

• For the first input split, it generates 4 key-value pairs: This, 1; is, 1; an, 1; apple, 1; and for the second, it generates 5 key-value pairs: Apple, 1; is, 1; red, 1; in, 1; color.

• It is followed by the shuffle phase, in which the values are grouped by keys in the form of key-value pairs. Here we get a total of 6 groups of key-value pairs.

• The same reducer is used for all key-value pairs with the same key.

• All the words present in the data are combined into a single output in the reducer phase. The output

shows the frequency of each word.

• Here in the example, we get the final output of key-value pairs as This, 1; is, 2; an, 1; apple, 2; red, 1; in, 1; color, 1.

• The record writer writes the output key-value pairs from the reducer into the output files, and the final output data is by default stored on HDFS.

**Limitations of MapReduce**

Map Reduce example also faces some limitations, and they are as follows:

• MapReduce is a low-level programming model which involves a lot of writing code.

• The batch-based processing nature of MapReduce makes it unsuitable for real-time processing.

• It does not support data pipelining or overlapping of Map and Reduce phases.

• Task initialization, coordination, monitoring, and scheduling take up a large chunk of MapReduce’s

execution time and reduce its performance.

• MapReduce cannot cache the intermediate data in memory, thereby diminishing Hadoop’s performance.

**Conclusion: -** Thus, designed Distributed Application using MapReduce for Text Analysis.

**Code: -**

WC\_Mapper.java

package org.code; import java.io.IOException; import java.util.StringTokenizer; import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase; import org.apache.hadoop.mapred.Mapper; import org.apache.hadoop.mapred.OutputCollector; import org.apache.hadoop.mapred.Reporter; public class WC\_Mapper extends MapReduceBase implements Mapper<LongWritable,Text,Text,IntWritable>{ private final static IntWritable one = new IntWritable(1); private Text word = new Text();

public void map(LongWritable key, Text value,OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException{

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line); while (tokenizer.hasMoreTokens()){ word.set(tokenizer.nextToken()); output.collect(word, one);}}}

WC\_Runner.java

package org.code; import java.io.IOException; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat; import org.apache.hadoop.mapred.FileOutputFormat; import org.apache.hadoop.mapred.JobClient; import org.apache.hadoop.mapred.JobConf; import org.apache.hadoop.mapred.TextInputFormat; import org.apache.hadoop.mapred.TextOutputFormat; public class WC\_Runner {

public static void main(String[] args) throws IOException{ JobConf conf = new JobConf(WC\_Runner.class); conf.setJobName("WordCount"); conf.setOutputKeyClass(Text.class); conf.setOutputValueClass(IntWritable.class); conf.setMapperClass(WC\_Mapper.class); conf.setCombinerClass(WC\_Reducer.class); conf.setReducerClass(WC\_Reducer.class); conf.setInputFormat(TextInputFormat.class); conf.setOutputFormat(TextOutputFormat.class); FileInputFormat.setInputPaths(conf,new Path(args[0]));

FileOutputFormat.setOutputPath(conf,new Path(args[1]));

JobClient.runJob(conf);}}

WC\_Reducer.java

package org.code; import java.io.IOException; import java.util.Iterator;

import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase; import org.apache.hadoop.mapred.OutputCollector; import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class WC\_Reducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable> { public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException { int sum=0;

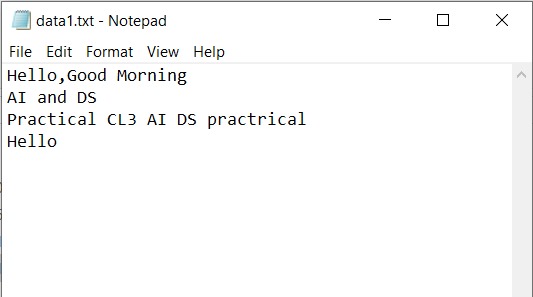
while (values.hasNext()) {

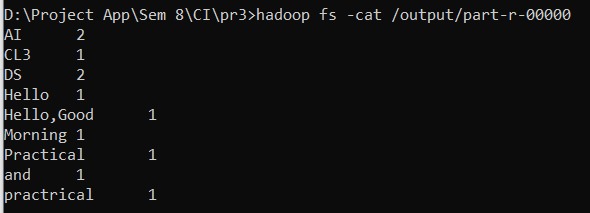
sum+=values.next().get();

}

output.collect(key,new IntWritable(sum));}}

**Output: -**





**Name – Shriya Pathak**

**Roll no – 61**

**Class – BE(AI&DS)**

**Subject – Computer Laboratory III (Distributed Computing)**

**Assignment No 07**

**Title: -** To apply the artificial immune pattern recognition to perform a task of structure damage

Classification.

**Problem Statement: -** Develop an artificial immune pattern recognition system for the task of

structural damage classification.

**Prerequisite: -** Basics of Python

**Software Requirements: -** Jupyter Notebook or any Python IDE

**Required Python libraries: -** (scikit-learn, numpy, matplotlib)

**Hardware Requirements: -** PIV, 2GB RAM, 500 GB HDD

**Learning Objectives: -**

1. Design a robust algorithm capable of identifying patterns indicative of structural damage within complex datasets.

2. Implement a scalable solution that can accurately classify various types and extents of structural damage with high efficiency.

**Outcomes: -**

After completion of this assignment students are able to understand how to The developed system will enable automated and accurate identification of structural damage, aiding in timely maintenance and ensuring safety.

**Theory: -**

Introduction to AIS:

• Briefly discuss the biological immune system and its key functions.

• Introduce the concept of AIS and its principles for pattern recognition.

• Explain the analogy between antigens (foreign invaders) and damage patterns, and antibodies

(immune cells) and damage detectors.

2. Setting Up the AIS Model:

• Divide the students into groups and assign each group a specific damage type

(e.g., crack, corrosion).

• Provide each group with a training dataset containing features of healthy and damaged structures

related to their assigned damage type.

• Guide students through the process of encoding the data into a format suitable for the AIS algorithm

(e.g., binary strings representing feature values).

• Introduce the chosen AIS algorithm and its key parameters (e.g., population size, mutation rate).

• Assist students in implementing the algorithm using the provided library or framework.

3. Training and Testing the Model:

• Run the AIS algorithm on the training data for each group.

• Explain the selection, mutation, and cloning processes within the algorithm.

• Observe how the "antibodies" (damage detectors) evolve over generations to better recognize the assigned damage pattern.

• Once training is complete, test the model with unseen data containing both healthy and damaged structures.

• Evaluate the performance of the AIS model based on its accuracy in detecting the assigned damage type.

4. Analysis and Discussion:

• Each group presents their results and discusses the performance of their AIS model.

• Compare the effectiveness of different damage detection algorithms or parameter settings across groups.

• Discuss the advantages and limitations of using AIS for damage detection.

• Explore potential applications of AIS in real-world engineering scenarios.

5. Diagram:

A simplified diagram illustrating the key steps of the lab experiment:

| Structure Data | (Healthy & Damaged)

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

|

v

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

| Feature Extraction |

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

|

v

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

| Data Encoding | (Binary Strings)

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

|

v

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

| AIS Algorithm | (Clonal Selection)

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

|

v

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

| Trained Detectors | (Damage Patterns)

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

|

v

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

| Test Data | (Unseen Structures)

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

|

v

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

| Damage Detection | (Accuracy Evaluation)

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

|

v

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

| Results & Discussion| (Performance Analysis)

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

Note: This is a general outline and can be adapted based on the specific AIS algorithm, chosen

damage types, and available resources.

This code represents a process flow for a damage detection system using an AIS algorithm.

1. Structure Data: Represents the input data for the system, which includes information about

healthy and damaged structures.

2. Feature Extraction: Extracts relevant features from the input data to be used in the detection

process.

3. Data Encoding: Converts the extracted features into binary strings for processing by the AIS

algorithm.

4. AIS Algorithm: Utilizes a Clonal Selection algorithm to detect damage patterns in the

encoded data.

5. Trained Detectors: Represents the detectors that have been trained to recognize specific

damage patterns.

6. Test Data: Contains unseen structures that will be used to test the performance of the

detection system.

7. Damage Detection: Evaluates the accuracy of the detection system in identifying damage in

the test data.

8. Results & Discussion: Analyzes the performance of the system and discusses the results

obtained from the damage detection process.

**Conclusion: -** This way, Implemented the artificial immune pattern recognition to perform a task of structure damage Classification.

**Code: -**

import numpy as np

# Define AIRS algorithm

class AIRS:

def \_\_init\_\_(self, num\_detectors=10, hypermutation\_rate=0.1):

self.num\_detectors = num\_detectors

self.hypermutation\_rate = hypermutation\_rate

def train(self, X, y):

# Initialize detectors using k-means clustering or other techniques

# For simplicity, initializing randomly in this example

self.detectors = X[np.random.choice(len(X), self.num\_detectors, replace=False)]

def predict(self, X):

predictions = []

for sample in X:

distances = np.linalg.norm(self.detectors - sample, axis=1)

prediction = int(np.argmin(distances))

predictions.append(prediction)

return predictions

# Generate dummy data

def generate\_dummy\_data(samples=100, features=10):

data = np.random.rand(samples, features)

labels = np.random.randint(0, 2, size=samples)

return data, labels

# Split data into training and testing sets

def split\_data(data, labels, split\_ratio=0.8):

split\_index = int(split\_ratio \* len(data))

train\_data, test\_data = data[:split\_index], data[split\_index:]

train\_labels, test\_labels = labels[:split\_index], labels[split\_index:]

return train\_data, test\_data, train\_labels, test\_labels

# Evaluate accuracy

def evaluate\_accuracy(predictions, true\_labels):

accuracy = np.mean(predictions == true\_labels)

return accuracy

# Main function

def main():

# Generate dummy data

data, labels = generate\_dummy\_data()

# Split data into training and testing sets

train\_data, test\_data, train\_labels, test\_labels = split\_data(data, labels)

# Initialize and train AIRS

airs = AIRS(num\_detectors=10, hypermutation\_rate=0.1)

airs.train(train\_data, train\_labels)

# Test AIRS on the test set

predictions = airs.predict(test\_data)

# Evaluate accuracy

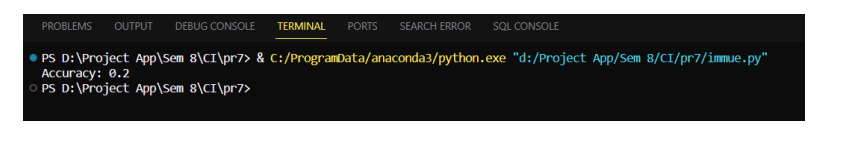
accuracy = evaluate\_accuracy(predictions, test\_labels)

(f"Accuracy: {accuracy}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output: -**

****

**Name – Shriya Pathak**

**Roll no – 61**

**Class – BE(AI&DS)**

**Subject – Computer Laboratory III (Distributed Computing)**

**Assignment No 08**

**Title: -** Implement DEAP (Distributed Evolutionary Algorithms) using Python.

**Problem Statement: -** Develop a distributed evolutionary algorithm using DEAP (Distributed Evolutionary Algorithms in Python) to optimize a complex problem that requires intensive computational resources.

**Prerequisite: -** Basics of Python

**Software Requirements: -** Jupyter

**Hardware Requirements: -** PIV, 2GB RAM, 500 GB HDD

**Learning Objectives: -**

1. Implement a scalable and efficient distributed evolutionary algorithm using DEAP to solve

the optimization problem.

2. Improve the optimization process by leveraging parallel processing capabilities and

distributing the workload across multiple nodes.

**Outcomes: -**

After completion of this assignment students are able to understand how to developed distributed evolutionary algorithm will significantly reduce the optimization time and enable the solution of larger and more complex optimization problems.

**Theory:**

DEAP (Distributed Evolutionary Algorithms in Python) is a framework for building and analyzing distributed evolutionary algorithms. It provides tools for implementing genetic algorithms and other evolutionary computation techniques in a flexible and easy-to-use manner. DEAP allows users to parallelize their evolutionary algorithms across multiple processors or computers, making it suitable for tackling complex optimization problems that require significant computational resources.

DEAP provides a wide range of tools and functionalities to facilitate the implementation and experimentation of evolutionary algorithms. Some key features of DEAP include:

1. Genetic Operators: DEAP provides a set of standard genetic operators such as crossover, mutation, and selection, which can be easily customized and combined to suit the problem being solved.

2. Fitness Evaluation: DEAP allows users to define custom fitness functions to evaluate the

quality of candidate solutions.

3. Population Management: DEAP provides tools for managing populations of candidate

solutions, including initialization methods and selection strategies.

4. Statistics and Logging: DEAP includes utilities for tracking and logging the evolution of

candidate solutions over time, including statistics such as average fitness and best fitness.

5. Parallelization: DEAP supports parallelization of evolutionary algorithms, allowing users to

take advantage of multi-core processors and distributed computing environments to speed up

optimization tasks.

6. Integration: DEAP can be easily integrated with other Python libraries and frameworks,

making it a versatile tool for a wide range of optimization problems.

Overall, DEAP provides a convenient and efficient way to implement evolutionary algorithms in

Python, making it a popular choice for researchers and practitioners working in the field of

evolutionary computation.

Use and Applications:

• DEAP is used to solve optimization problems where traditional methods may be impractical or inefficient.

• It is commonly used in various fields such as engineering, biology, finance, and data science for optimization tasks.

• Applications include parameter optimization, feature selection, function optimization, and more.

**Conclusion: -** This way DEAP (Distributed Evolutionary Algorithms) using Python is done.

**Code: -**

import random

from deap import base, creator, tools, algorithms

# Define the evaluation function (minimize a simple mathematical function)

def eval\_func(individual):

# Example evaluation function (minimize a quadratic function)

return sum(x \*\* 2 for x in individual),

# DEAP setup

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# Define attributes and individuals

toolbox.register("attr\_float", random.uniform, -5.0, 5.0) # Example: Float values between -5

and 5

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr\_float, n=3) #

Example: 3-dimensional individual

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# Evaluation function and genetic operators

toolbox.register("evaluate", eval\_func)

toolbox.register("mate", tools.cxBlend, alpha=0.5)

toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)

toolbox.register("select", tools.selTournament, tournsize=3)

# Create population

population = toolbox.population(n=50)

# Genetic Algorithm parameters

generations = 20

# Run the algorithm

for gen in range(generations):

offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)

fits = toolbox.map(toolbox.evaluate, offspring)

for fit, ind in zip(fits, offspring):

ind.fitness.values = fit

population = toolbox.select(offspring, k=len(population))

# Get the best individual after generations

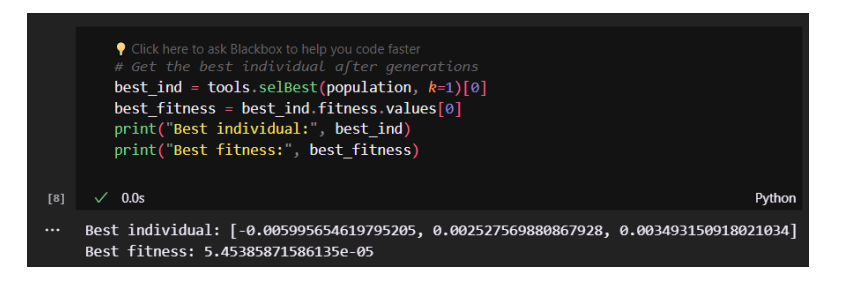
best\_ind = tools.selBest(population, k=1)[0]

best\_fitness = best\_ind.fitness.values[0]

print("Best individual:", best\_ind)

print("Best fitness:", best\_fitness)

**Output: -**

****

**Name – Shriya Pathak**

**Roll no – 61**

**Class – BE(AI&DS)**

**Subject – Computer Laboratory III (Distributed Computing)**

**Assignment No 09**

**Title: -** Title: Design and develop a distributed Hotel booking application using Java RMI. A distributed hotel booking system consists of the hotel server and the client machines. The server manages hotel rooms booking information. A customer can invoke the following operations at his machine i) Book the room for the specific guest ii) Cancel the booking of a guest.

**Problem Statement: -** Design and develop a distributed Hotel booking application using Java RMI, consisting of a server managing hotel room booking information and clients able to book or cancelrooms for guests.

**Prerequisite: -** Basics of Java

**Software Requirements: -** BlueJ IDE

**Hardware Requirements: -** PIV, 2GB RAM, 500 GB HDD

**Learning Objectives: -**

1. Learn to Perform a robust and scalable distributed system architecture using Java RMI to manage hotel room bookings efficiently.

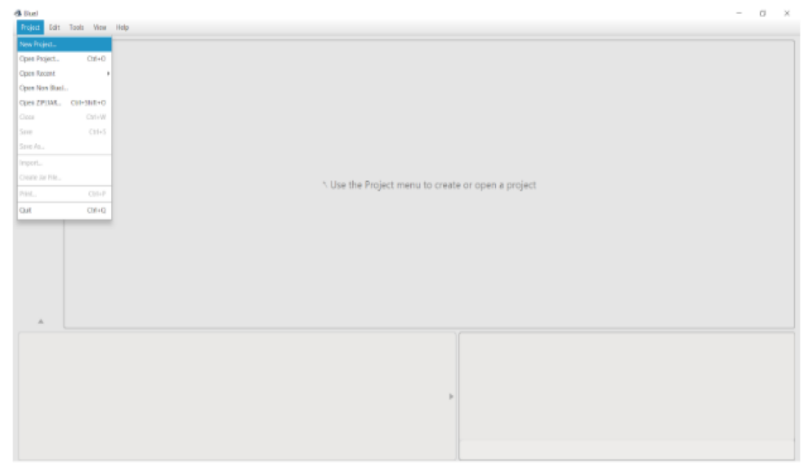
2. Provide clients with the ability to book rooms for specific guests and cancel bookings as needed, ensuring a user-friendly experience.

**Outcomes: -**

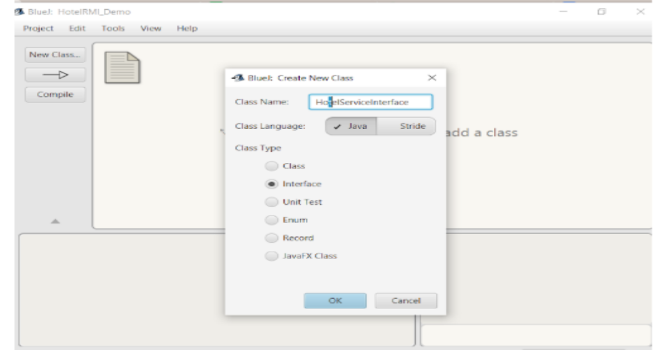
After completion of this assignment students are able to understand how to develop application will enable clients to easily book and manage hotel room bookings, while the server efficiently manages the booking information, leading to a seamless hotel booking experience.

**Theory:**

Step 1: BlueJ IDE



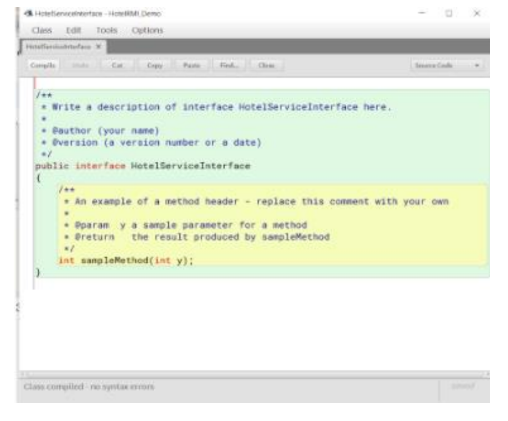
Step 2: Create Interface



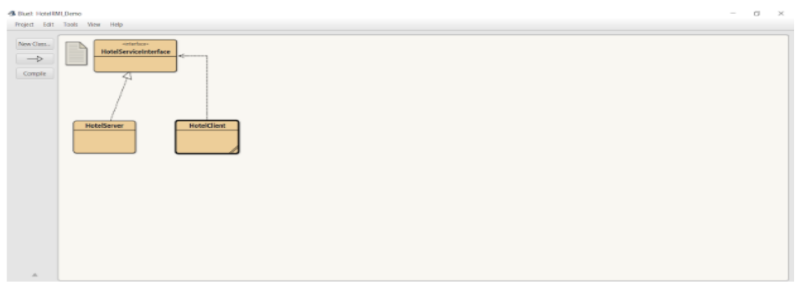
Step 3: Interface has been Created. Right click on Block of Interface.



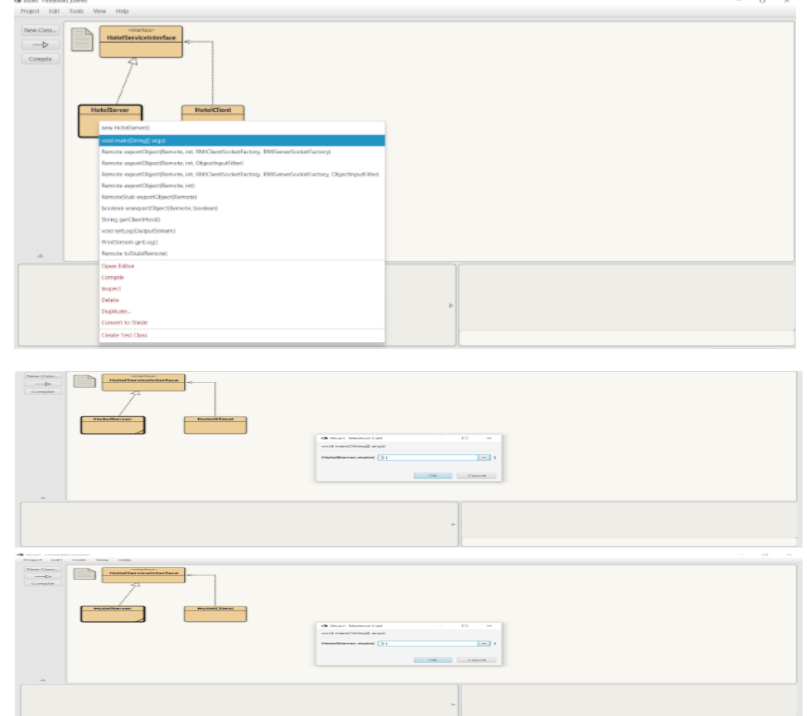
Step 4: Click on Open Editor. Write the code of Interface here and Compile it.



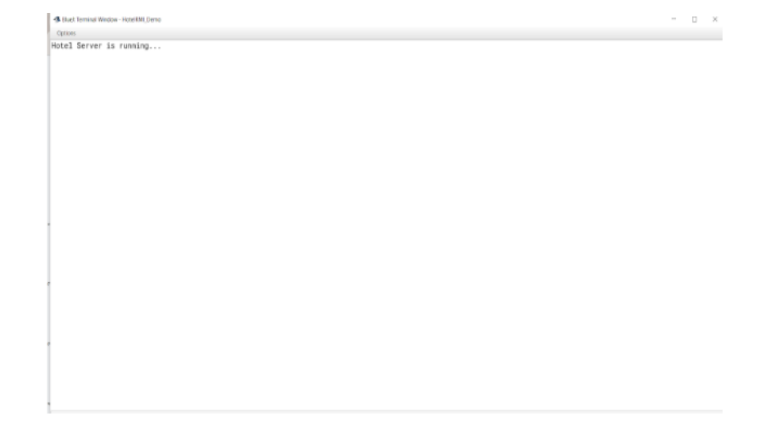
Step 5: In the similar way create two more classes HotelServer, and HotelClient. Open the editor and write the code and compile the both classes. On successfully completed the above procedure the diagram will be.

****

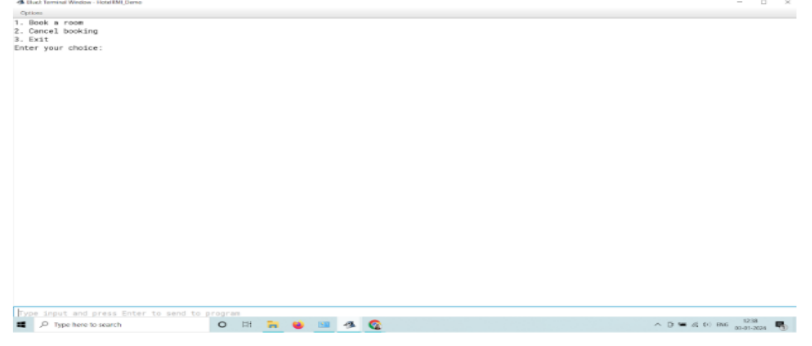
Step 6:Write Click on the block of HotelServer Class and select the main method call.

****

Step 7:Observe the HotelServer is running on Terminal Window.



Step 8: In the similar way Right click on the block of HotelClient Class and select the main method call. Observe the HotelClient is running on Terminal Window.



Step 9: Give the appropriate choice in the Terminal window and Run the code successfully.

**Conclusion:** - Thus, a distributed Hotel booking application using Java RMI. A distributed hotel booking system consists of the hotel server and the client machines. The server manages hotel rooms.

**Code: -**

**HotelServer.java**

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

public class HotelServer {

public static void main(String[] args) { try {

// Create and export the remote object HotelService hotelService = new HotelServiceImpl();

java.rmi.registry.LocateRegistry.createRegistry(1099);

java.rmi.registry.Registry registry = java.rmi.registry.LocateRegistry.getRegistry(); registry.rebind("HotelService", hotelService);

System.out.println("Hotel Server ready");

} catch (Exception e) {

System.err.println("Hotel Server exception: " + e.toString()); e.printStackTrace();

}

}

}

## HotelServiceImpl.java

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject; import java.util.HashMap;

import java.util.Map;

// Implementing the remote interface

public class HotelServiceImpl extends UnicastRemoteObject implements HotelService { private Map<String, Integer> bookings;

public HotelServiceImpl() throws RemoteException { bookings = new HashMap<>();

}

@Override

public synchronized boolean bookRoom(String guestName, int roomNumber) throws RemoteException {

if (bookings.containsValue(roomNumber)) {

System.out.println("Room " + roomNumber + " is already booked."); return false; // Room already booked

} else {

bookings.put(guestName, roomNumber);

System.out.println("Booking done for guest: " + guestName + ", Room: " + roomNumber);

return true; // Booking successful

}

}

@Override

public synchronized boolean cancelBooking(String guestName) throws RemoteException { if (bookings.containsKey(guestName)) {

bookings.remove(guestName);

System.out.println("Booking canceled for guest: " + guestName); return true; // Cancellation successful

} else {

System.out.println("No booking found for the guest: " + guestName); return false; // No booking found for the guest

}

}

}

# HotelService.java

import java.rmi.Remote;

import java.rmi.RemoteException;

// Define interface for remote methods

public interface HotelService extends Remote {

boolean bookRoom(String guestName, int roomNumber) throws RemoteException; boolean cancelBooking(String guestName) throws RemoteException;

}

# HotelClient.java

import java.rmi.RemoteException; import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; import java.util.Scanner;

public class HotelClient {

public static void main(String[] args) { try {

// Get the remote object reference

Registry registry = LocateRegistry.getRegistry("localhost", 1099); HotelService hotelService = (HotelService) registry.lookup("HotelService");

// Create a Scanner object to read user input Scanner scanner = new Scanner(System.in); int option;

do {

// Display menu options

System.out.println("Choose an option:"); System.out.println("1. Book a room"); System.out.println("2. Cancel booking"); System.out.println("3. Exit"); System.out.print("Enter option: "); option = scanner.nextInt();

switch (option) { case 1:

// Booking

boolean booked = false; do {

System.out.print("Enter guest name: "); scanner.nextLine(); // Consume newline character String guestName = scanner.nextLine();

System.out.print("Enter room number: "); int roomNumber = scanner.nextInt();

successfully!");

// Perform booking operations try {

booked = hotelService.bookRoom(guestName, roomNumber); if (booked) {

System.out.println("Room " + roomNumber + " booked

} else {

System.out.println("Room " + roomNumber + " is already booked.

Please enter a different room number.");

}

} catch (RemoteException e) {

System.err.println("Error while booking room: " + e.getMessage()); e.printStackTrace();

}

} while (!booked); break;

case 2:

// Cancellation

System.out.print("Enter guest name to cancel booking: "); scanner.nextLine(); // Consume newline character

String cancelGuestName = scanner.nextLine(); boolean cancellationStatus =

hotelService.cancelBooking(cancelGuestName); if (cancellationStatus) {

System.out.println("Booking canceled successfully!");

} else {

System.out.println("No booking found for the guest!");

}

break; case 3:

// Exit System.out.println("Exiting..."); break;

default:

System.out.println("Invalid option! Please try again."); break;

}

} while (option != 3);

// Close the scanner scanner.close();

} catch (Exception e) {

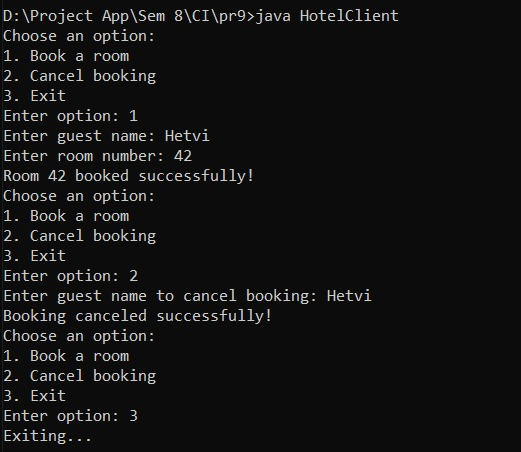
System.err.println("Hotel Client exception: " + e.toString()); e.printStackTrace();

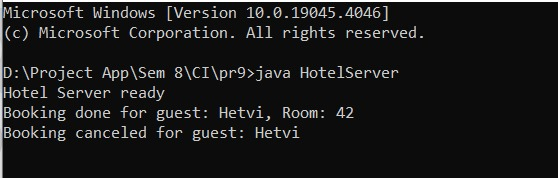
}

}

}

**Output: -**

****

****