/\*

Name : Rohit Narayan Telgote

PRN : 1941054

Batch : B4

\*/

// **Aim** : 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

1. Book the room for the specific guest
2. Cancel the booking of a guest

**RoomBookingServer.java**

import java.io.\*;

import java.rmi.\*;

import java.rmi.server.\*;

class RoomBookingServer extends UnicastRemoteObject implements RoomBookingInterface {

/\*\*

\* This is the Server Class. It contains the working methods which can be used

\* by the client.

\*/

protected int day;

protected int time;

protected int room;

protected String str = new String();

public String RoomListTemp[] = new String[100]; // Temporary store for list of rooms

public String temp = new String();

public Room RoomArray[] = new Room[100]; // Array of Room Objects

RoomList tempList = new RoomList();

public RoomBookingServer() throws RemoteException {

super();

}

/\*\*

\* This method is called once by the client when the application starts. It

\* reads

\* in the input from the text file and creates an Object for each room with the

\* name and capacity that was specified in the file.

\*/

public void initRooms() throws RemoteException {

String record = null;

String tempRoom = null;

String tempCap = null;

int recCount = 0;

int num;

int capacity;

try {

// This reads in the text from the file and uses that to create the

// Room Objects. The name is specified first in the text file and the

// capacity is specified last. This is manipulated in order to take in

// these parameters when creating the Rooms.

BufferedReader b = new BufferedReader(new FileReader("Rooms.txt"));

while ((record = b.readLine()) != null) {

num = (record.lastIndexOf(" ", record.length())) + 1;

tempRoom = record.substring(0, num - 1); // Reads in the Room name from file

tempCap = record.substring(num, record.length());

capacity = Integer.parseInt(tempCap); // Reads in the capacity from file

RoomArray[recCount] = new Room(tempRoom, capacity); // Fills the array with the created Objects.

recCount++;

}

b.close(); // close the input stream.

} catch (IOException e) {

System.out.println("Error!" + e.getMessage());

}

}

/\*\*

\* This method is used to return the list of rooms and there capacity to the

\* client.

\* It returns a RoomList Object which contains the arrayList of Rooms. The

\* Client

\* can then retrieve a full list of rooms.

\*/

public RoomList allRooms() throws RemoteException {

try {

BufferedReader in = new BufferedReader(new FileReader("rooms.txt")); // read in the text file.

if ((str = in.readLine()) != null) {

tempList.RoomList[0] = str;

for (int i = 1; i < 100; i++) {

if ((str = in.readLine()) != null) {

tempList.RoomList[i] = str;

}

}

}

in.close();

} catch (IOException e) {

}

return tempList;

}

/\*\*

\* This method takes in a string and then compares that string with the name of

\* each Object

\* in the array of Rooms. If it finds the room it returns the index, -1

\* otherwise.

\*/

public int compareRoom(String str) {

for (int i = 0; i < RoomArray.length; i++) {

if (RoomArray[i].name.equals(str)) {

return i;

}

}

return -1;

}

/\*\*

\* This method is used to check whether a room is available or not. Firstly it

\* checks

\* for the room in the array, if it finds it it then checks whether the

\* requested

\* time slot on the requested day is available. It returns a string to the

\* client

\* depending on the value of the timeslot.

\*/

public String checkRoom(String r, int day, int startTime) throws RemoteException {

int i = compareRoom(r);

if (RoomArray[i].slotAvailable(day, startTime) == true) // calls methos available to Room Object

{

String s = "Room is available for booking";

return s;

} else {

String s = "Sorry the room is not available for booking";

return s;

}

}

/\*\*

\* This method is used to book a Room. Again it checks whether the slot is

\* available and depending

\* on the result it reserves that slot and informs the client or it informs them

\* that

\* the slot has already been reserved.

\*/

public String bookRoom(String r, int day, int startTime) throws RemoteException {

int i = compareRoom(r);

if (RoomArray[i].slotAvailable(day, startTime) == true) {

RoomArray[i].book(day, startTime);

String s = "Room has been successfully booked.";

return s;

} else {

String s = "Sorry but the Room has already been booked.";

return s;

}

}

/\*\*

\* This method is used to calculate the timetable for each room. It returns

\* relevant

\* the 2D array to the client displaying the weekly timetable for the requested

\* room.

\*/

public int[][] roomTimeTable(String room) throws RemoteException {

int i;

System.out.println("TimeTable" + room);

for (i = 0; i < RoomArray.length; i++) {

if (RoomArray[i].name.equals(room)) {

return RoomArray[i].daySlot;

} else {

System.out.println("Searching for the room");

}

}

return RoomArray[i].daySlot;

}

// Main Method

public static void main(String[] args) {

try {

RoomBookingServer server = new RoomBookingServer();

String name = "rmi://localhost:9999/RoomBookingSystem";

// Naming.bind (name, server);

// String name = "RoomBookingSystem";

Naming.bind(name, server);

System.out.println(name + " is running");

} catch (Exception ex) {

System.err.println(ex);

}

}

}

**RoomBookingClient.java**

import java.rmi.\*;

import java.rmi.server.\*;

import java.io.\*;

class RoomBookingClient {

/\*\*

\* This is the Client Class. It takes an input from the user, calls the methods

\* available

\* to the client from the server class and gives an ouput depending on the

\* operation performed.

\*/

public static boolean validChoice = true;

static String[] daysOfWeek = { "Monday |", "Tuesday |", "Wednesday|", "Thursday |", "Friday |", "Saturday |",

"Sunday |" };

public static void main(String[] args) {

try {

// System.setSecurityManager ( new RMISecurityManager ( )); //set up the

// security manager

String name = "rmi://localhost:9999/RoomBookingSystem"; // connect on local

// host on port 9999

// String name = "rmi://127.0.0.1/RoomBookingSystem";

RoomBookingInterface rbi = (RoomBookingInterface) Naming.lookup(name);

rbi.initRooms(); // set up the room booking system

while (validChoice != false) {

// A small command line interface for the user to use the system.

System.out.println(" ");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Room Booking Service\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("");

System.out.println(" Please select a service");

System.out.println("");

System.out.println("1. List of all rooms.");

System.out.println("2. Check availability of a room.");

System.out.println("3. Book a room.");

System.out.println("4. Display weekly timetable for a room.");

System.out.println("");

// A buffered reader to allow input from the command line from the user.

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));

System.out.println("");

System.out.println("Select a number between 1 and 4, 0 to exit");

System.out.println("");

System.out.flush();

String response = input.readLine();

int i = Integer.parseInt(response);

RoomList ListOfAllRooms = new RoomList(); // RoomList Object which stores

// a list of all the rooms available.

try {

switch (i) {

case 0:

System.out.println("Goodbye"); // User has quit the application.

validChoice = false;

break;

case 1:

System.out.println("");

System.out.println("The full list of rooms is as follows");

System.out.println("");

System.out.println("Room|Capacity");

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

ListOfAllRooms = rbi.allRooms(); // Run the allRooms method which

// returns the list of all rooms.

for (int c = 0; c < 100; c++) // Print the list.

{

if (ListOfAllRooms.RoomList[c] == null) {

break;

}

System.out.println(ListOfAllRooms.RoomList[c]);

}

System.out.println("");

break;

case 2:

System.out.println("");

System.out.println("Check a room");

System.out.println("Enter the room name");

String check\_room = input.readLine();

System.out.println("Enter the day - ");

System.out.println("0=Mon , 1=Tues, 3=Wed ,4=Thurs , 5=Fri, 6=Sat, 7=Sun");

String check\_day = input.readLine();

int real\_day = Integer.parseInt(check\_day);

System.out.println("Enter the start time - ");

System.out.println(

"0=8am , 1=9am , 2=10am , 3=11am , 4=12pm , 5=1pm , 6=2pm , 7=3pm , 8=4pm , 9=5pm , 10=6pm , 11= 7pm");

String check\_time = input.readLine();

int real\_time = Integer.parseInt(check\_time);

// This checks whether a room is available given the room name, day and time.

String temp = rbi.checkRoom(check\_room, real\_day, real\_time);

System.out.println(temp);

System.out.println("");

break;

case 3:

System.out.println("Room Booking Service - Rooms can be booked from 8am to 8pm");

System.out.println("");

System.out.println(

"Time slots go from 0 for 8am up to 11 for 7pm - Enter a value in this range");

System.out.println("");

System.out.println("Enter the room name");

String book\_room = input.readLine();

System.out.println("");

System.out.println("Enter the day -");

System.out.println("0=Mon , 1=Tues, 3=Wed ,4=Thurs , 5=Fri, 6=Sat, 7=Sun");

String book\_day = input.readLine();

int real\_day2 = Integer.parseInt(book\_day);

System.out.println("");

System.out.println("Enter the start time -");

System.out.println(

"0=8am , 1=9am , 2=10am , 3=11am , 4=12pm , 5=1pm , 6=2pm , 7=3pm , 8=4pm , 9=5pm , 10=6pm , 11= 7pm");

String book\_time = input.readLine();

int realb\_time = Integer.parseInt(book\_time);

// This checks whether a room is available, if it is it then reserves the room.

String resp = rbi.bookRoom(book\_room, real\_day2, realb\_time);

System.out.println(resp);

System.out.println("");

break;

case 4:

System.out.println("Enter the Room name");

String Room1 = new String();

Room1 = input.readLine();

// This checks the timetable for a room. A 2D array containing

// the timetable is returned from the server.

System.out.println("TimeSlot | 0 1 2 3 4 5 6 7 8 9 10 11");

int rtt[][] = (int[][]) rbi.roomTimeTable(Room1).clone();

for (int f = 0; f < 7; f++) {

System.out.println("");

System.out.print(daysOfWeek[f]);

for (int j = 0; j < 12; j++) {

System.out.print(" ");

System.out.print(rtt[f][j]);

}

}

System.out.println("");

System.out.println(" ");

System.out.println("The key to start times is as follows... ");

System.out.println(

"0 = 8am , 1 = 9am , 2 = 10am , 3 = 11am , 4 = 12pm , 5 = 1pm , 6 = 2pm , 7 = 3pm , 8 = 4pm , 9 = 5pm , 10 = 6pm , 11 = 7pm");

System.out.println("");

break;

}

} catch (Exception e) {

System.err.println("Sorry but you have entered one of the fields incorrectly, Please try again ");

}

}

} catch (Exception ex) {

System.err.println(ex);

}

}

}

**Room.java**

import java.io.\*;

import java.rmi.\*;

import java.rmi.server.\*;

import java.io.Serializable;

class Room implements Serializable {

/\*

\* This is the Room class. Each Room Object has a name and a capacity. It also

\* contains a 7 \* 12 array which represents the 7 days of the week and the

\* 12 hours between 8 am and 8pm(The valid hours for booking a room).

\*/

int daySlot[][] = new int[7][12]; // represents days and hours

String name;

int capacity;

public Room(String n, int cap) // constructor that sets all slots to zero - unbooked

{

this.name = n;

this.capacity = cap;

for (int i = 0; i < 7; i++) {

for (int j = 0; j < 11; j++) {

this.daySlot[i][j] = 0;

}

}

}

/\*\*

\* This Method is used to check whether a particular timeslot on a particular

\* day

\* has already been booked. If the slot contains a 1 then it has already been

\* booked.

\* If it contains a 0 then it is available. The method returns a true or false

\* value.

\*/

public boolean slotAvailable(int day, int slot) {

if (daySlot[day][slot] == 1) {

return false;

} else {

return true;

}

}

/\*\*

\* This Method is used to book a slot. It sets the relevant slot to a 1.

\*/

public void book(int day, int slot) {

this.daySlot[day][slot] = 1;

}

}

**RoomBookingInterface.java**

import java.rmi.\*;

import java.rmi.server.\*;

/\*\*

\* This is the interface, it contains the 5 methods which the Client can use.

\*/

public interface RoomBookingInterface extends Remote {

public void initRooms() throws RemoteException;

public RoomList allRooms() throws RemoteException;

public String checkRoom(String r, int day, int startTime) throws RemoteException;

public String bookRoom(String r, int day, int startTime) throws RemoteException;

public int[][] roomTimeTable(String room) throws RemoteException;

}

**RoomList.java**

import java.io.\*;

import java.rmi.\*;

import java.rmi.server.\*;

import java.io.Serializable;

class RoomList implements Serializable {

public String RoomList[] = new String[100];

// contains an array which holds the maximun number of rooms. To allow for

// more rooms just increase the size of this array.

}

**Rooms.txt**

Room1 1

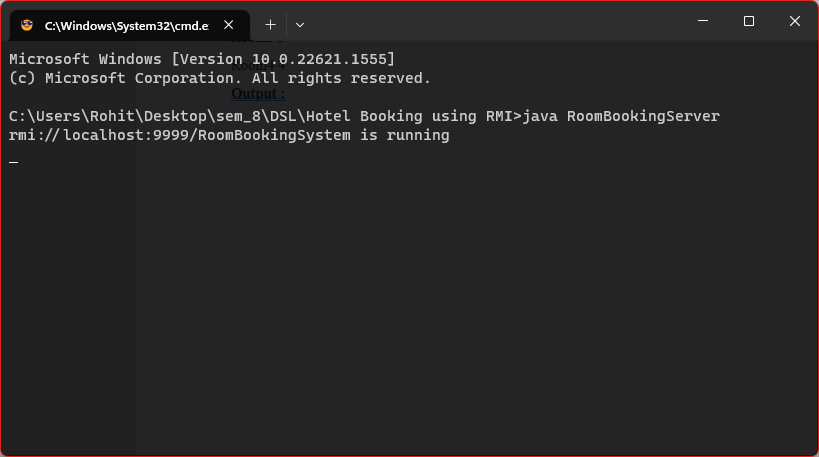
Room2 2

Room3 3

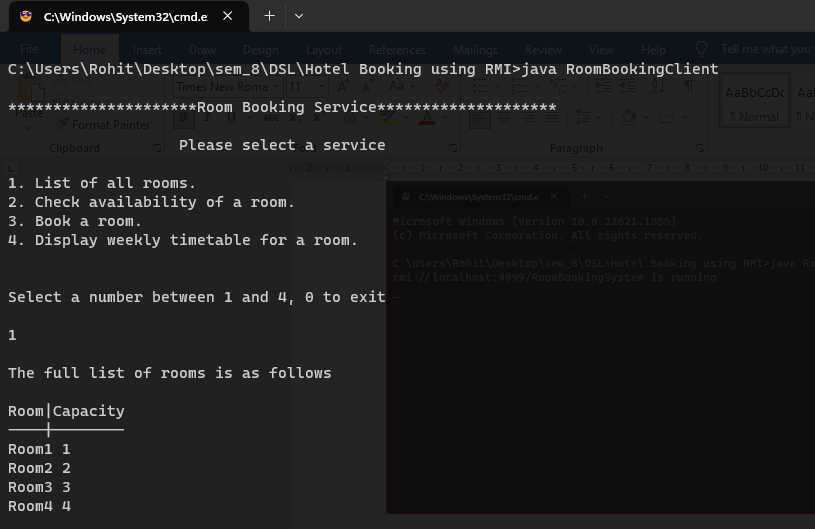
Room4 4

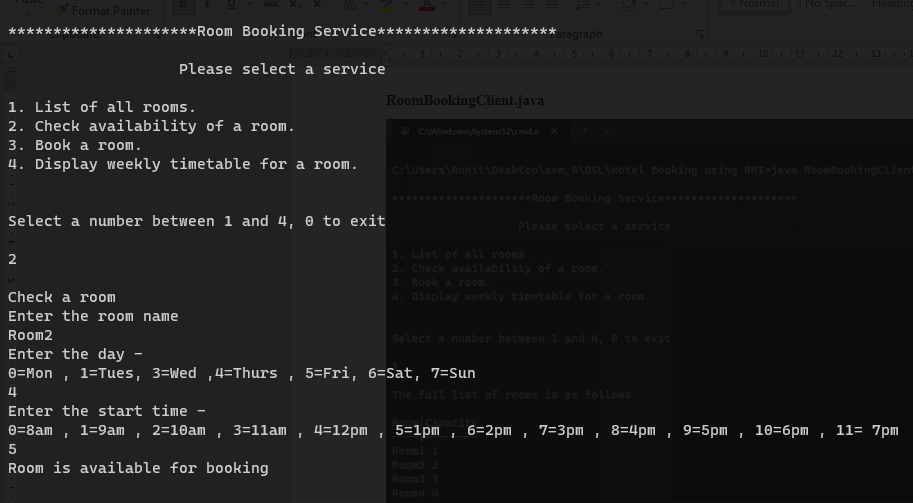
**Output :**

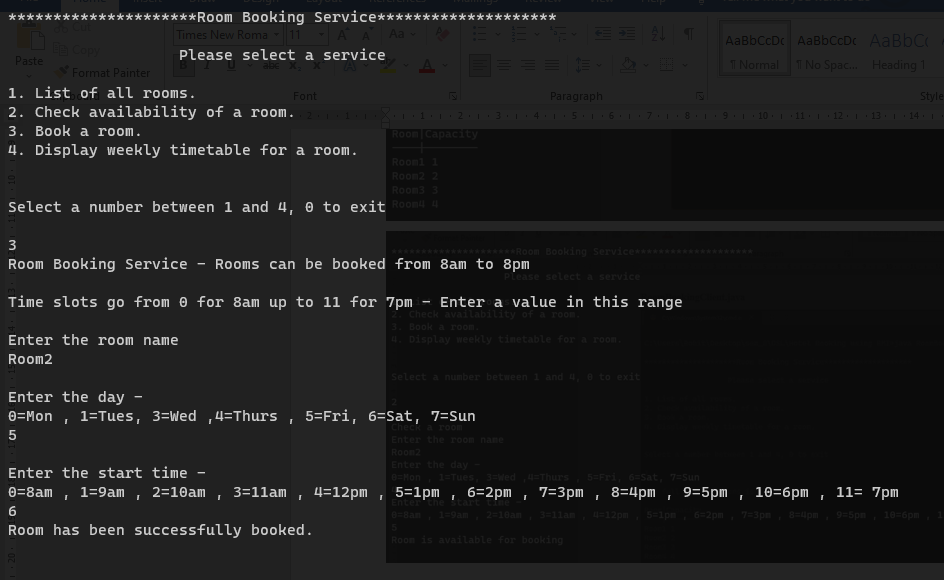
**RoomBookingServer.java**

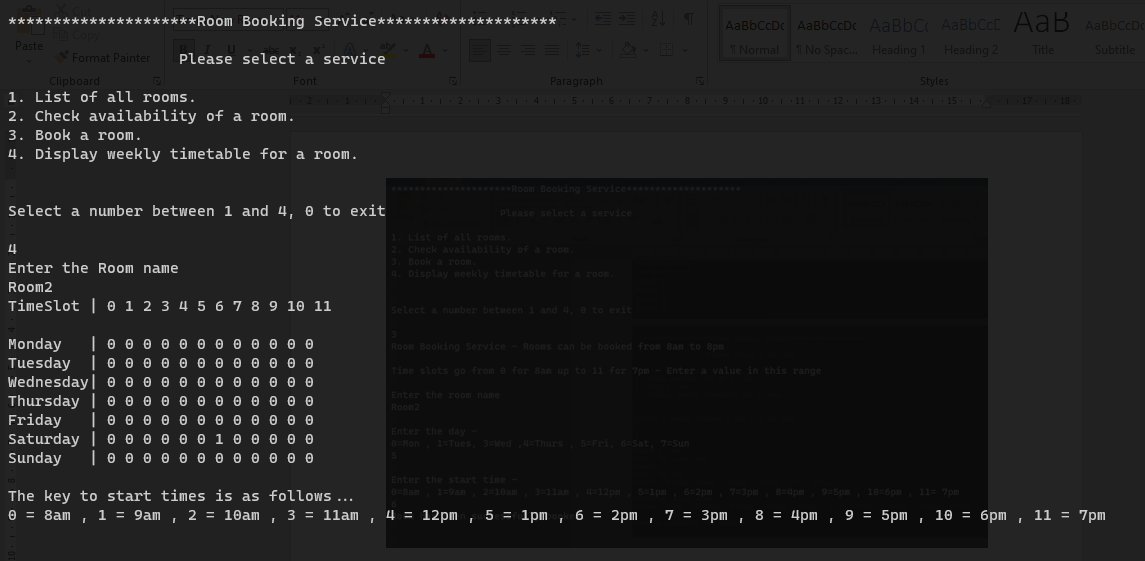
****

**RoomBookingClient.java**

****

****

****

****