**Distributed System Trains booking**

**Jinmin YU**

2710205

Submitted in partial fulfillment for the degree of

B.Sc. in Computing Science

Griffith College Dublin

April , 2013

Tony Mullins

**Disclaimer**

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the Degree of Bachelor of Science in Computing at Griffith College Dublin, is entirely my own work and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

**Signed: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Acknowledgements

I would like to thank my supervisor Tony Mullins for support that he gave me throughout my project of studies and for help me to talk through my ideas for the project. And thanks to my parents support me to finish the college.

**Table of Contents**

[Acknowledgements iii](#_Toc353981645)

[List of Figures v](#_Toc353981646)

[List of Tables v](#_Toc353981647)

[Abstract vi](#_Toc353981648)

[Chapter 1. Introduction 1](#_Toc353981649)

[1.1 Distributed Systems 1](#_Toc353981650)

[1.2 Goals 1](#_Toc353981651)

[Chapter 2. Background 2](#_Toc353981652)

[2.1 Literature Review 2](#_Toc353981653)

[Chapter 3. Route search Algorithm (graph) 4](#_Toc353981654)

[Chapter 4. System Design and Specifications 6](#_Toc353981655)

[4.1 Structure of the system 6](#_Toc353981656)

[4.2 Use case diagram 7](#_Toc353981657)

[4.3 Database ER Diagram 8](#_Toc353981658)

[4.3.1 Operator server 9](#_Toc353981659)

[4.3.2 Agent 11](#_Toc353981660)

[4.4 Class diagram 12](#_Toc353981661)

[4.3.1 Operator. 12](#_Toc353981662)

[4.3.2 Agent 13](#_Toc353981663)

[Chapter 5. Implementation 14](#_Toc353981664)

[Chapter 6. Testing and Evaluation 16](#_Toc353981665)

[6.1 Operator side 16](#_Toc353981666)

[6.1.1Database 16](#_Toc353981667)

[6.1.2 Java program 17](#_Toc353981668)

[6.2 Agent side 18](#_Toc353981669)

[6.2.1 Database 18](#_Toc353981670)

[6.2.2 Java program 19](#_Toc353981671)

[Chapter 7 Project diary 23](#_Toc353981672)

[Chapter 8. Conclusions 24](#_Toc353981673)

[References 25](#_Toc353981674)

[Appendex I 26](#_Toc353981675)

[Agent.java 26](#_Toc353981676)

[AgentThred.java 34](#_Toc353981677)

[BookRrturn.java 36](#_Toc353981678)

[Node.java 36](#_Toc353981679)

[PassClass.java 36](#_Toc353981680)

[Route.java 36](#_Toc353981681)

[RouteTimePrice.java 37](#_Toc353981682)

[SearchParam.java 37](#_Toc353981683)

[Ticket.java 37](#_Toc353981684)

[Train.java 38](#_Toc353981685)

[Graph.java 39](#_Toc353981686)

[OperatorThread.ava 48](#_Toc353981687)

[operaterServer.java 50](#_Toc353981688)

# List of Figures

[Figure 1 example diagram 3](#_Toc353979724)

[Figure 2 route map 4](#_Toc353979725)

[Figure 3 structure 6](#_Toc353979726)

[Figure 4 use case 8](#_Toc353979727)

[Figure 5 operator database 9](#_Toc353979728)

[Figure 6 agent database 11](#_Toc353979729)

[Figure 7 operator class diagram 12](#_Toc353979730)

[Figure 8 agent class diagram 13](#_Toc353979731)

[Figure 9 GUI 15](#_Toc353979732)

[Figure 10 software start 20](#_Toc353979733)

[Figure 11 one server on 21](#_Toc353979734)

[Figure 12 two server on 22](#_Toc353979735)

# List of Tables

[Table 1 operator database 14](#_Toc353757629)

[Table 2 operator server 15](#_Toc353757630)

[Table 3 operator thread 15](#_Toc353757631)

[Table 4 agent database 16](#_Toc353757632)

[Table 5 agent server 17](#_Toc353757633)

[Table 6 agent thread 17](#_Toc353757634)

[Table 7 GUI 18](#_Toc353757635)

# Abstract

In 21st century, travelling has already been a very important part of human life. Almost people will go to a travel agency or give them a call then let the staff do everything. This project is to design and implement of a system to allow travel agents to booking ticket to train journeys. And the project is choosing suitable distributed algorithms for ticket booking and cancellation. The distributed system use plurality of autonomous computers, through a computer network for communication. All of computers work on one job like my project. My project have multiple transport for the product, you can booking tickets from two or more different agents to different train companies.

# Chapter 1. Introduction

## 1.1 Distributed Systems

Distributed systems by computer are connected to the computer network and distributed system software with independent collection. The software allows the computer to be able to coordinate their activities and the sharing of resources, the system's hardware, software, and data. The user should be aware of the distributed system into single, integrated computing devices, even if it can be achieved in different locations of multiple computers. Rather, it is in a network, the user knows the location of several machines; storage replication, load balancing, and functionality are not transparent.This project is investigating and researching the distributed Systems area how to use socket to implement. Then design and create a system for users to book tickets. This system will allow user to book train tickets from multiple agents to multiple train companies.

## 1.2 Goals

A distributed Java platform has been designed and built for the simplified implementation of distributed Java applications. Its programmable nature means that code as well as data is distributed over a network. The generality of the system is demonstrated through the emulation of a MIMD (multiple instruction, multiple data) architecture. One of the key features of our system is that it can dynamically alter the size of work units to achieve the optimal processing time per unit. [6]

The prototype implementation is using only sockets for communication between the parts of the system and the program is writing in Java. My design is considered that nodes and disks can fail at any time when the system running. (Possibility of communication failure). Also, user interface is developed for easy using of this system. And the system is also considering the scalability that my system could scale to support different deployments.

# Chapter 2. Background

## 2.1 Literature Review

**Socket**

The sockets API provides access to the standard network protocols used for communications between any client and server on the Net. They are low level tools that snd streams of data between applications that may or may not be on the same host. The programmer must provide their own application level protocol for handling and interpreting the data. The basic socket class provides a connection-oriented protocol that issimilar to a telephone conversation. After establishing a connection two applications can send data back and forth. The protocol ensures that no data is lost and that data arrives in the order in which it is sent. The underlying protocol class is TCP IP. [7]

**Server**

The server opens a socket and listens for incoming conversations. It does this by creating a Server Socket object and waits, blocked in a call to its accept until a connection arrives.[8]

**Socket level server communication**

Socket is created with no name. A remote process has no way to refer to a socket until an address is bound to the socket. Processes that communicate are connected through addresses. In the Internet family, a connection is composed of local and remote addresses and local and remote ports. Duplicate ordered sets, such as: protocol, local address, local port, foreign address, foreign port cannot exist. Connection establishment is usually asymmetric, with one process acting as the client and the other as the server. The server binds a socket to a well-known address associated with the service and blocks on its socket for a connect request. An unrelated process can then connect to the server. The client requests services from the server by initiating a connection to the server's socket. On the client side, the [connect](http://docs.oracle.com/docs/cd/E19253-01/816-5170/connect-3socket/index.html) call initiates a connection.

The following example diagram is a server. The server creates a socket and binds a name to the socket, then displays the port number. The program calls to mark the socket as ready to accept connection requests and to initialize a queue for the requests. The rest of the program is an infinite loop. Each pass of the loop accepts a new connection and removes it from the queue, creating a new socket. The server reads and displays the messages from the socket and closes the socket.[9]

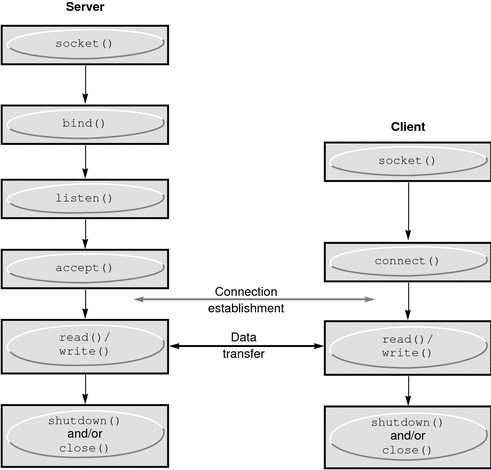
****

Figure 1 example diagram

My project was on parallel computing, following:

* Books
  + Tony Mullins (Concurrent Programming with Java)
  + C Wu(A Comprehensive Introduction to Object-Oriented Programming with Java)
* Software Documentation
  + java, Sun Microsystems Inc, 7.0
  + MYSQL mysql.com 5.5

# Chapter 3. Route search Algorithm (graph)

Graphs are good in modelling real world problems like representing cities which are connected by roads and finding the paths between cities, modelling air traffic controller system, etc. So graph is the perfect match for this project.

Graph data structure contains two algorithms used for traversing and searching a node.

**Depth First Search (DFS)**

DFS algorithm is to traverse the graph in such a way that it tries to go far from the root node. Stack is used in the implementation of the depth first search.

**Breadth First Search (BFS)**

BFS algorithm is to traverse the graph as close as possible to the root node. Queue is used in the implementation of the breadth first search.

In this project, I use the BFS to search the routes.

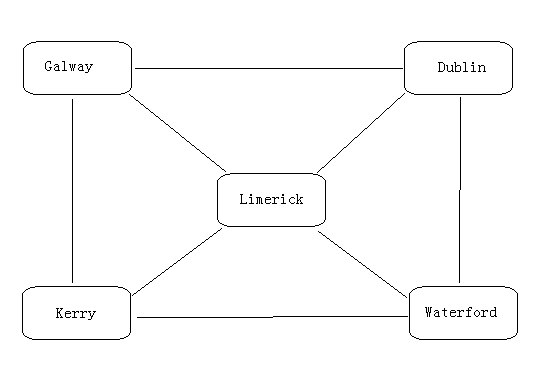
****

Figure 2 route map

Algorithmic steps

Push the root node in the Queue. (Figure 1 set Dublin city is root)

Loop until the queue is empty.

Remove the node from the Queue.

If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

In the upper procedure, the node could stand for a city, and those procedures could print out the city when the program goes there. However, those procedures could not remember the path (a list of city the program arrived). So I change the program, I change the node to an Array List of node, so when the program arrive a node, it will put the node into the Array List, so this Array List could remember the path (a list of city).

# Chapter 4. System Design and Specifications

## 4.1 Structure of the system

**The following figure shows my system structure:**

This system is a multi hop distributed system. Each operator server could connect many agent clients; also, each agent could handle many connections to different operator server. The communication is implemented by socket and threads programming in java. This system has great scalability.

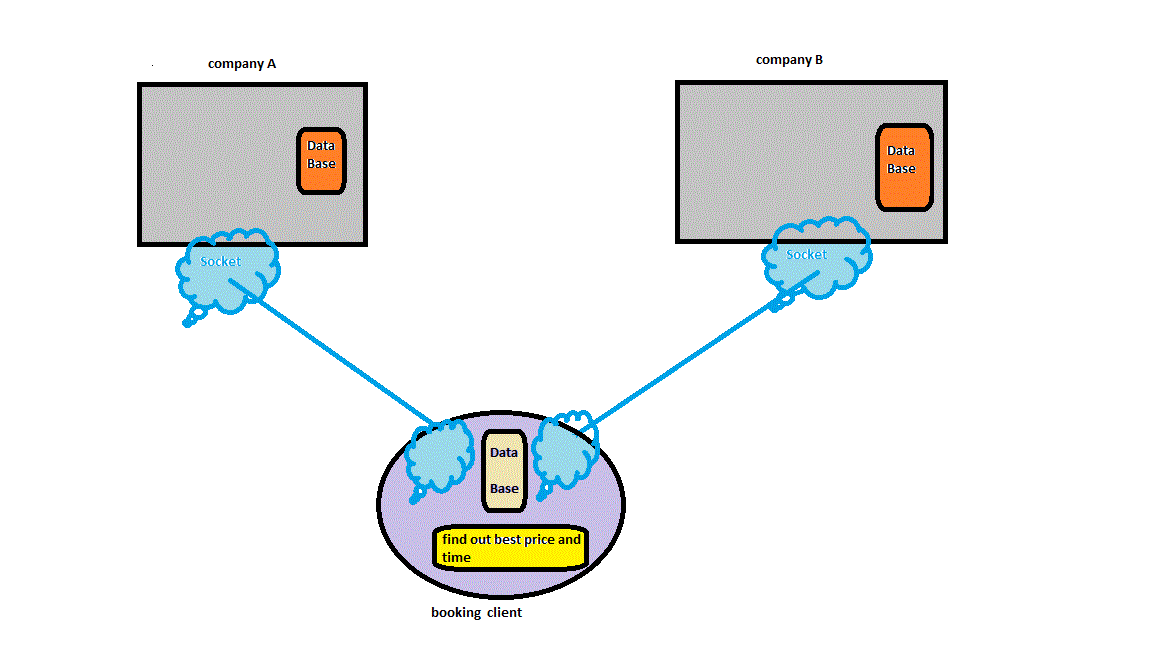
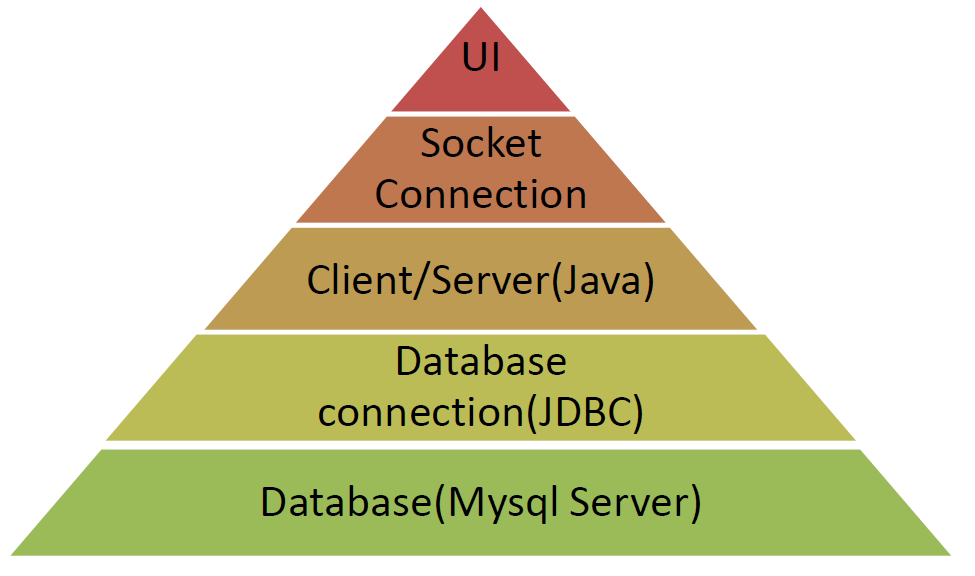


Figure 3 structure

**This system has 5 layers and diagram is shown below:**

****

**UI:** allow users input their booking, cancellation and checking ticket status request.

**Socket:** interchange the request between computers and processes.

**Client/Server:** using socket to interchange requests and process the requests associate with database.

**JDBC:** allow the process connect with database and retrieve or update the data in database.

**Database:** store the request and updates. All the information (e.g. request, confirmation and etc) will be stored in it. They are where users get relevant information.

## 4.2 Use case diagram

The Use case diagram is used to identify the primary elements and processes that form the system. The primary elements are termed as "actors" and the processes are called "use cases." The Use case diagram shows which actors interact with each use case. The main purpose of a use case diagram is to show what system functions are performed for which actors. Roles of the actors in the system can be depicted. The following figures show the use case for this system. The following diagram shows the use case for this system

User – the end user who wants to book a ticket.

Enter information for searching ticket – This use case includes select depart city, select destination, select depart date.

View ticket detail – the user view the search result and try to find one ticket to book.

Book ticket – user books to book ticket.

Cancel booking - user will delete booking. (Figure 3)

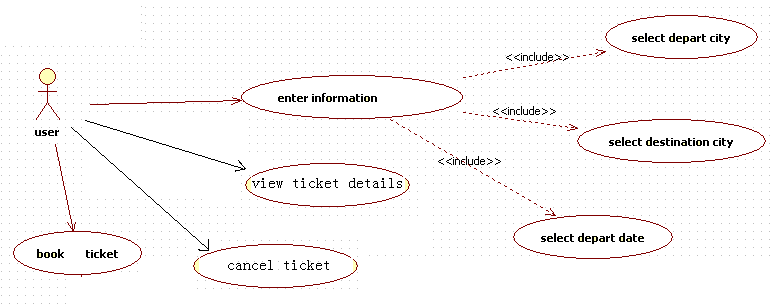


Figure 4 use case

## 4.3 Database ER Diagram

ER Diagram is used for design database, it use a graphical representation of entities and their relationships to each other to show the structure of the database. It includes almost everything need in the database: tables, almost constrains, and relationships…

This project contains two operator servers, and two agents. Actually my design could allow many server and agent. Each operator servers and agent has their own database. In this system, I use My SQL database. The following figures show the database diagram.

## Operator server

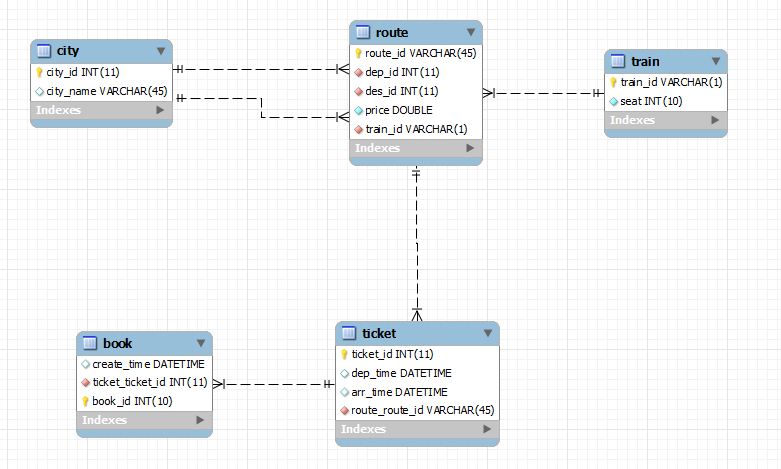


Figure 5 operator database

1. City:

Store the depart city and destination city in the operator train service. It stores the city name in the table, and gives each city a city id.

1. Route:

Store the routes which are provided by the operator. Every route contain depart city and the destination city, and every route has their own price. The train id attribute specify which kind of train is running this route.

1. Train

Train table contains the different train id, and different train has different number of seat. This attribute could affect the number of ticket on certain route.

1. Ticket

This table contain different kinds of ticket. The attributes of this table are depart time, arrival time and the route id. Those values could define every single type of ticket.

1. Book

This table just stores the information of booking ticket. When a booking is successful, one more row will insert into this table. It will record the booking time and the ticket number. However, the same tickets have certain quantity, which depends by the train seat number for this kind of ticket. When the train is full, you will not be allowed to book the ticket for that train.

## 4.3.2 Agent

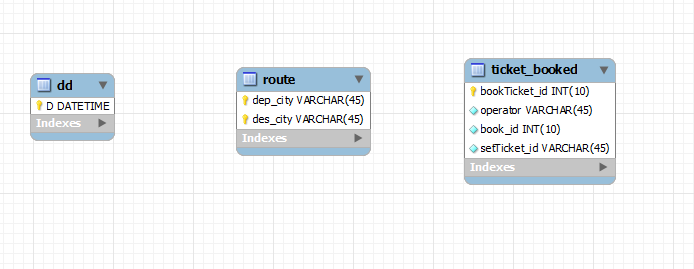


Figure 6 agent database

1. DD

The table is just stores the date for the ticket.

1. Route

This table is just stores the routes of all the operators. This means every operator’s routes are stored in one table. So the attributes are only depart city and the destination city. There is no any other attributes to specify which route is provide by which operator. It just simply mixes them together. This design is focus on the scalability for many operators. Also, this table is always keeping updated when the operator database is updated.

1. Ticket\_booked

This table just keep the record of the tickets booked from this agent. The attribute are operator, book id, which used to record the ticket from which operator, and used for cancel ticket service from operator database. The bookTicket\_id is generated by the agent; this id is used for cancelling ticket service both from local database and the operator database.

## 4.4 Class diagram

### 4.3.1 Operator.

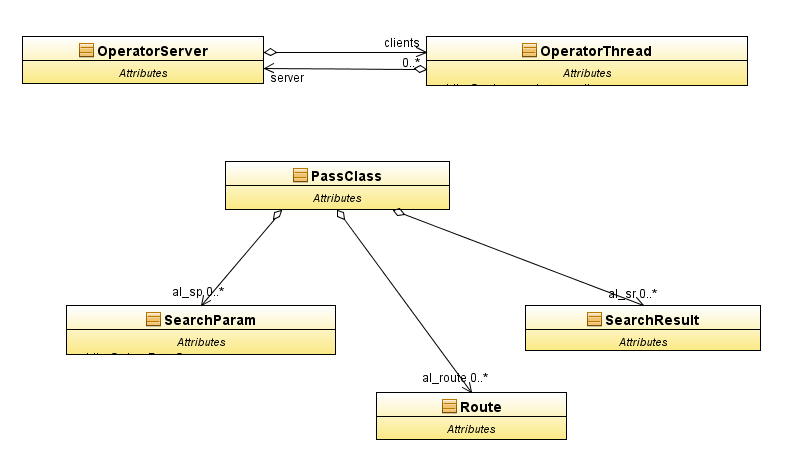


Figure 7 operator class diagram

### 4.3.2 Agent

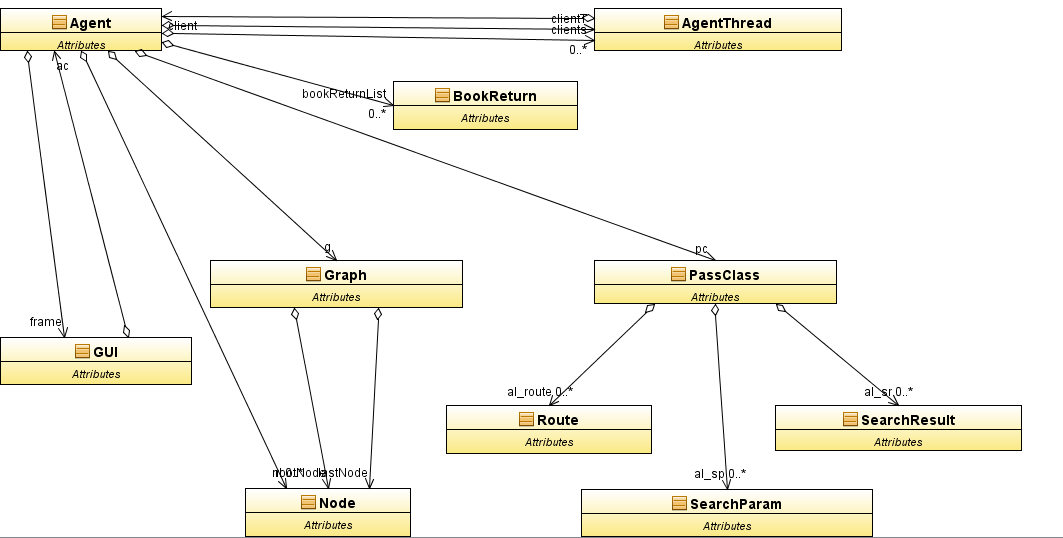


Figure 8 agent class diagram

# Chapter 5. Implementation

This chapter is talk about the code of the implementation. It will describe my problems, issues occurred and how they are fixed.

Through the previous chapters design, the new system contains many functionalities and tasks; it is difficult to implement them together. So the idea is split the implementation into different parts, develop them one to one, and put them together. For implementation the code all learning from the book” Concurrent Programming with Java Tony Mullins”

This will be reducing the developing time; also it is easy for testing.

1. Environment setting

Java environment setting

1. Setting database management system

MYSQL environment setting my sql need download and install. Also download and install the MySQL workbench 5.2, use it could be can access and control database easily.

1. After that build the database

I will sue the stored procedure. Use the A stored procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again.  So if you think about a query that you write over and over again, instead of having to write that query each time you would save it as a stored procedure and then just call the stored procedure to execute the SQL code that you saved as part of the stored procedure.[1] it will be easy find out the problem and not change code for database. In this part I reference the command from mysql website. [2] [3]

1. JDBC

Download the |JDBC drive use JDBC to connection database.[4][5]

1. Implementation the function.
2. Set the GUI

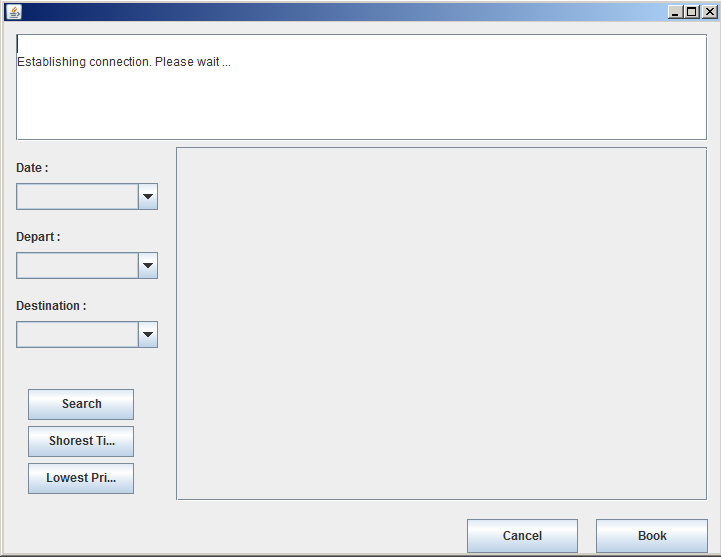


Figure 9 GUI

# Chapter 6. Testing and Evaluation

In my testing plan, I use black box testing. The testing has gone on throughout my whole project. When a new function or feature is designed, it was always been tested before the feature add in my program. And I have created many test projects and create my new function in the project, and run it; this method will make sure the function is run perfectly.

## 6.1 Operator side

### 6.1.1Database

|  |  |  |
| --- | --- | --- |
| Stored procedure / function | Test code | Expect results |
| getRouteCity | Call getRouteCity() | Get the routes provided by operator |
| getLeftTicket | SELECT operator1.getLeftTicket(ticket id) | Get the how many that type of tickets in total |
| Search ticket | Call search\_ticket(dep\_city,arr\_city,date) | Get all the details of tickets available |
| Insert\_book | Call insert\_book(ticket id) | Insert a new row in book table |
| Delete\_book | Call delete\_book(book id) | Delete a row from the book table |
|  |  |  |

Table 1 operator database

### 6.1.2 Java program

Operator Server class

|  |  |  |
| --- | --- | --- |
| Function | Action | Expect result |
| public PassClass getRouteTable() throws Exception | Call the stored procedure getRouteCity(), get all the available routes from operator | All the available returned and saved in the ArrayList, put the arrayList into object PassClass |
| public synchronized PassClass bookTicket(PassClass pc) | Receive the passclass class, then take the parameters and call the stored procedure insert\_book() | A new row inserted into database. And return a confirm information. |
| public synchronized void cancelTicket(PassClass pc) | Receive the passclass class as parameter, call the stored procedure cancel\_ticket() | The ticket in database is deleted. |
| public PassClass searchTicket(PassClass pc) | Receive the passclass as parameter, call the stored procedure search\_ticket() | Select all the available tickets from the database. |
| public void run() | A thread, waiting a client connect, then put this connection into a array. | The thread is put into array. |
| private int findClient(int ID) | Find out correct thread, and use it send and receive message. | The thread is founded. |
| public synchronized void broadcast(int ID, PassClass input) | Server send message to client | The message is sent successfully. |
| public synchronized void remove(int ID) | Remove the thread from thread array | The thread is removed successfully |
| private void addThread(Socket socket) | Add the thread into array | The thread is added into array successfully. |

Table 2 operator server

OperatorThread class

|  |  |  |
| --- | --- | --- |
| public void send(PassClass pc) | Send message to client | The message is sent successfully. |
| public void run() | A thread, waiting the client response. | Receive the message from client successfully. |
| public void close() throws IOException | Close socket, input stream and output stream | They are closed. |
| public void connect() throws Exception | Connect to database MySQL | Database connected |
|  |  |  |

Table 3 operator thread

## 6.2 Agent side

### 6.2.1 Database

|  |  |  |
| --- | --- | --- |
| Stored procedure / function | Test code | Expect results |
| insert\_route | CALL agent.insert\_route(dep city,arr city) | Insert the route into route table |
| agent\_book | CALL agent.agent\_book(operator, book id,set ticket id) | Insert the booking information into ticket\_bookde table |
| Agent\_cancel | CALL agent\_cancel(,set ticket id) | Delete book from ticket\_book table |
| Get\_agent\_booked\_ticket | CALL get\_agent\_booked\_ticket(set ticket id) | Select all the ticket from the book table which has the same id. |
|  |  |  |

Table 4 agent database

### 6.2.2 Java program

|  |  |  |
| --- | --- | --- |
| public void connect() throws Exception | Connect to database MySQL | Database connected |
| private void addThread(Socket socket) | Add the thread into array | The thread is added into array successfully. |
| private int findClient(int ID) | Find out correct thread, and use it send and receive message. | The thread is founded. |
| protected synchronized void sendTextToChat(String operator,PassClass pc) | Send the message to server | Send successfully |
| public synchronized void handle(int ID,PassClass pc) | Handle different messages received from server. | Analyse and handle the messages well. |
| public void buildMap() throws Exception | Build map for route | Map is build successfully. |
| public void searchRoute(String fi, String la) | Search route | All the possible is successfully returned. |
| public PassClass getSearchObj(String s) | Put the route result into an ArrayList, wrap them and prepare for sending. | The corrected sending item is built. |
| public synchronized ArrayList <BookReturn> agent\_cancel (String setTicket\_id) | Cancel the book from the agent database | The book is delete from database successfully |
| public synchronized void agent\_book(String operator,int bookid,String setTicket\_id) | Insert the book into agent database | A new row is inserted into agent database. |
| public void bindComBox() throws SQLException | Fill data into combobox. | Data is successfully inserted. |
| public synchronized void remove(String operator) | remove client thread from client thread array |  |

Table 5 agent server

**Java program (Agent Thread)**

|  |  |  |
| --- | --- | --- |
| public void send(String operator,PassClass pc) | Send the message to server | Message is successful sent. |
| public void run() | A thread, waiting and receive message from server | Messages are successfully received. |

Table 6 agent thread

**Java program (GUI)**

|  |  |  |
| --- | --- | --- |
| Function | ACTION | Expect result |
| private void jButton1ActionPerformed() | Search button clicked | Return the search results and display in the jtable. |
| private void jButton2ActionPerformed | Book button clicked | the book details has been insert into both server and agent database, it one of them is fail, cancel the successful one. |
| private void jButton4ActionPerformed | Cancel button clicked | Pup up a input diag. then delete the book from agent and server database. |
| public void appendText(String s) | Display the useful info on the jtextarea | The useful info display successfully. |
|  |  |  |
|  |  |  |

Table 7 GUI

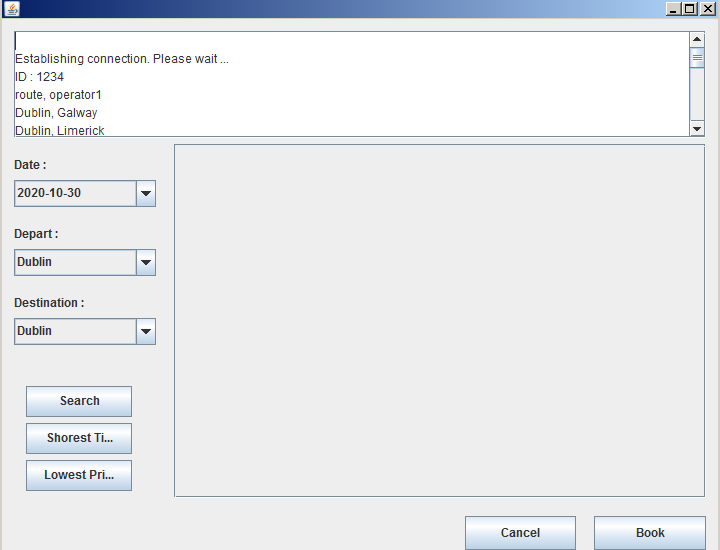


Figure 10 software start

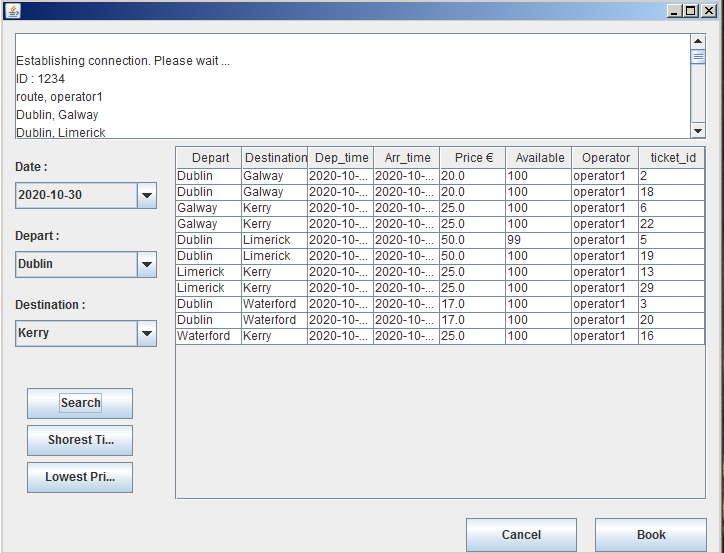


Figure 11 one server on

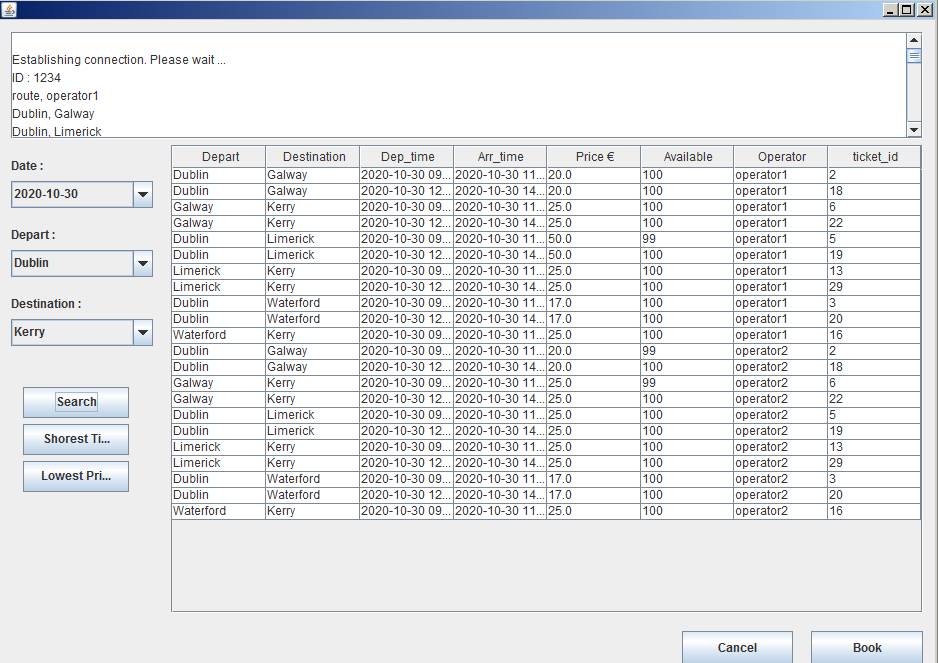


Figure 12 two server on

# Chapter 7 Project diary

|  |  |
| --- | --- |
| 2013-10-09 | Setting environment for project. eclipse, My SQL |
| 2013-10-10 | Learning java socket thread programming. |
| 2013-10-16 | find and learn a sample code for java socket and thread |
| 2013-10-18 | design database and create table |
| 2013-02-20 | testing class passing, redesign database and re create tables, insert data into tables, |
| 2013-02-20 | create stored procedures and update database |
| 2013-03-01 | start java socket and thread programming |
| 2013-03-02 | find some mistake on database design |
| 2013-03-03 | start redesign database |
| 2013-03-04 | continue design the database and writing the Stored procedure |
| 2013-03-05 | start operator server programming |
| 2013-03-09 | operator server side programming, threads programming |
| 2013-03-10 | agent side database programming, agent side programming |
| 2013-11-11 | communicate server and client by socket, synchronize database from  server to agent |
| 2013-11-12 | start think the data structure for searching the route and ticket |
| 2013-11-13 | start the UI programming |
| 2013-03-14 | Change my design structure. fix bugs in graph, database can not  connected. |
| 2013-03-15 | doing the search ticket function then return to client |
| 2013-03-16 | learning jTable and try to show data on UI |
| 2013-03-17 | Show data on jTable and learn using net bean to instead of eclipse.  Change environment, redraw the UI by net beans. |
| 2013-03-18 | Add a new server java code, create new database for new server,  start do the book button function. |
| 2013-03-19 | Continue the booking button function. if one book fail, all other  ticket cancel |
| 2013-03-20 | continue the booking button function, cancel button |
| 2013-03-21 | Add new agent to join the booking system. final testing and debug,  and tidy the code |
| 2013-03-22 | start write report |
| 2013-03-27 | write report |
| 2013-04-03 | write the report |

# Chapter 8. Conclusions

This project was to build multi-hops booking system for train tickets. The first aim of this project is to learn java sockets and threads programming, and try to understand the structure of distributed system.

During the project implementation, the areas of sockets and threads in java are widely investigated and studied. I have deeper understanding on the sockets and threads domain. And my java programming skills is getting practised and going up into a new higher level. During the implementation, I change my design over and over again, because I don’t have any experience to design a distributed system. After finishing the project, I have achieve a lot of precious experience for designing, and now it is much clearer in my mind that how to design and develop a distributed system in an industrial software development standard. Those great goals and experience which I achieved from the implementation of this project is really good fruit of my GCD studies.

# References

[1] http://www.mssqltips.com/sqlservertutorial/160/sql-server-stored-procedure/

[2]http://dev.mysql.com/doc/refman/5.5/en/c-api-prepared-call-statements.html

[3] http://dev.mysql.com/doc/refman/5.5/en/c-api.html

[4] http://www.java-samples.com/showtutorial.php?tutorialid=202

[5] http://jdbc.postgresql.org/download.html

[6] http://www.cs.may.ie/research/reports/2003/nuim-cs-tr-2003-03.pdf

[7]Concurrent Programming with Java Tony Mullins

[8] Concurrent Programming with Java Tony Mullins

[9] http://docs.oracle.com/cd/E19120-01/open.solaris/817-4415/sockets- 18552/index.html

# Appendex I

### Agent.java

import java.net.\*; import java.io.\*; import java.sql.CallableStatement;import java.sql.Connection; import java.sql.DriverManager; import java.sql.ResultSet;

import java.sql.SQLException; import java.sql.Statement; import java.text.DateFormat;

import java.text.ParseException; import java.text.SimpleDateFormat; import java.util.\*;

import java.util.logging.Level; import java.util.logging.Logger; import java.util.Date;

public class Agent

{ private Socket socket = null;

private Thread thread = null;

private BufferedReader console = null;

private AgentThread clientT = null;

private String chatName;

public GUI frame;

private Connection conn = null;

private Graph g = null;

private ArrayList <Node> nl = null;

public ArrayList <ArrayList>finalList = null;

public PassClass pc = null;

// array of Threads to servers

private AgentThread clients[] = new AgentThread[50];

public HashMap hashmap = new HashMap();

private int clientCount = 0;

public int returnCount = -1;

public ArrayList <BookReturn> bookReturnList = new ArrayList<BookReturn>();

public ArrayList<RouteTimePrice> sortList = new ArrayList<RouteTimePrice>();

public ArrayList<SearchResult> directly = new ArrayList<SearchResult>();

public ArrayList<ArrayList<SearchResult>> notDirectly = new ArrayList<ArrayList<SearchResult>>();

public ArrayList<SearchResult> step1 = new ArrayList<SearchResult>();

public ArrayList<SearchResult> step2 = new ArrayList<SearchResult>();

public ArrayList<RouteTimePrice> shortTimeList = new ArrayList<RouteTimePrice> ();;

public ArrayList<RouteTimePrice> lowPriceList = new ArrayList<RouteTimePrice> ();;

public ArrayList<String> operatorList = new ArrayList<String>();

public void connect() throws Exception

{ String userName = "root";

String password = "admin";

String url = "jdbc:mysql://localhost/test";

Class.forName ("com.mysql.jdbc.Driver").newInstance ();

conn = DriverManager.getConnection (url, userName, password);

System.out.println ("Database connection established"); }

public Agent()

{ frame = new GUI(this);

frame.setVisible(true);

frame.appendText("\n"+"Establishing connection. Please wait ...");

try{ connect();

File file = new File("D:\\prog2\\JavaApplication5\\build\\classes\\agent.txt");

BufferedReader input = null;

try { input = new BufferedReader(new FileReader(file));

String line = null;

while(( line = input.readLine()) != null)

{ String[] lineStr = line.split(" ");

socket = new Socket(lineStr[0],Integer.parseInt(lineStr[1]));

addThread(socket); }}

catch (FileNotFoundException e) { e.printStackTrace();}

catch (IOException e) { e.printStackTrace(); }

finally { input.close();} //declare a new client and put it in the array}

catch(UnknownHostException uhe){

System.out.println("Host unknown: " + uhe.getMessage());}

catch(IOException ioe){

System.out.println("Unexpected exception: " + ioe.getMessage());}

catch (Exception e)

{ System.err.println ("Cannot connect to database");}}

private void addThread(Socket socket)

{ if (clientCount < clients.length){

System.out.println("Client accepted: " + socket);

clients[clientCount] = new AgentThread(this, socket);

System.out.println("new instance");

try{ clients[clientCount].open();

clients[clientCount].start();

clientCount++;}

catch (Exception e)

{ e.printStackTrace();}

System.out.println("finish add thread");}

else

System.out.println("Client refused: maximum " + clients.length + " reached."); }

private int findClient(String operator)

{ int ID = -1;

String o = (hashmap.get(operator)).toString();

ID = Integer.parseInt(o);

for (int i = 0; i < clientCount; i++)

if (clients[i].getID() == ID)

return i; return -1; }

// send message to server called when click enter.

protected synchronized void sendTextToChat(String operator,PassClass pc)

{ AgentThread c = clients[findClient(operator)];

c.send(operator,pc);

System.out.println("client broadcast"); }

public Date stringToDate (String sd)

{ DateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.S");

Date myDate = null;

try { myDate = (Date)formatter.parse(sd);

return myDate; }

catch (ParseException ex) {

Logger.getLogger(Agent.class.getName()).log(Level.SEVERE, null, ex); }

return null; }

// set jtable for shortest Time

public void showShortestTimeRoutes()

{ frame.model= (javax.swing.table.DefaultTableModel)frame.jTable1.getModel();

frame.model.setColumnCount(0);

frame.model.setRowCount(0);

frame.model.addColumn("Depart");

frame.model.addColumn("Destination");

frame.model.addColumn("Dep\_time");

frame.model.addColumn("Arr\_time");

frame.model.addColumn("Price €");

frame.model.addColumn("Available");

frame.model.addColumn("Operator");

frame.model.addColumn("ticket\_id");

for (int i = 0;i<shortTimeList.size();i++)

{ RouteTimePrice rtp = shortTimeList.get(i); SearchResult sr1 = rtp.sr1;

SearchResult sr2 = rtp.sr2;

frame.appendText("\n"+ sr1.toString());

frame.model.addRow(new Object[] {sr1.dep,sr1.arr,sr1.dep\_time,sr1.arr\_time,sr1.price,sr1.avai,sr1.operator,sr1.ticket\_id});

if (sr2 != null)

{ frame.appendText("\n"+ sr2.toString());

frame.model.addRow(new Object[] {sr2.dep,sr2.arr,sr2.dep\_time,sr2.arr\_time,sr2.price,sr2.avai,sr2.operator,sr2.ticket\_id}); } }}

// set jtable for shortest Time

public void showLowestPriceRoutes()

{ frame.model= (javax.swing.table.DefaultTableModel)frame.jTable1.getModel();

frame.model.setColumnCount(0);

frame.model.setRowCount(0);

frame.model.addColumn("Depart");

frame.model.addColumn("Destination");

frame.model.addColumn("Dep\_time");

frame.model.addColumn("Arr\_time");

frame.model.addColumn("Price €");

frame.model.addColumn("Available");

frame.model.addColumn("Operator");

frame.model.addColumn("ticket\_id");

for (int i = 0;i<lowPriceList.size();i++)

{ RouteTimePrice rtp = lowPriceList.get(i);

SearchResult sr1 = rtp.sr1;

SearchResult sr2 = rtp.sr2;

frame.appendText("\n"+ sr1.toString());

frame.model.addRow(new Object[] {sr1.dep,sr1.arr,sr1.dep\_time,sr1.arr\_time,sr1.price,sr1.avai,sr1.operator,sr1.ticket\_id});

if (sr2 != null) {frame.appendText("\n"+ sr2.toString());

frame.model.addRow(new Object[] {sr2.dep,sr2.arr,sr2.dep\_time,sr2.arr\_time,sr2.price,sr2.avai,sr2.operator,sr2.ticket\_id}); } } }

// client receive the response from server, deal with the response. (show data on UI)

public synchronized void handle(int ID,PassClass pc)

{ frame.appendText("\nID : "+ ID + pc.toString());

if (pc.content.equals("route"))

{ try { CallableStatement insert\_route = conn.prepareCall("{call agent.insert\_route(?,?)}");

for (int i = 0; i< pc.al\_route.size(); i++)

{ Route c = pc.al\_route.get(i);

try { insert\_route.setString(1, c.dep);

insert\_route.setString(2, c.arr);

insert\_route.execute();

} catch (Exception e) { }

frame.appendText(c.toString());}

} catch (Exception e) {

e.printStackTrace();

} // bind data to combox

try { bindComBox();

} catch (SQLException e) { e.printStackTrace(); }

hashmap.put(pc.operator,ID); // remember how many operator connect. }

if (pc.content.equals("SearchResult"))

{ String oper=pc.operator;

if (operatorList.contains(oper)) { operatorList.clear();

operatorList.add(oper); shortTimeList.clear(); lowPriceList.clear();

}else

{ operatorList.add(oper); }

directly = new ArrayList<SearchResult>();

step1 = new ArrayList<SearchResult>();

step2 = new ArrayList<SearchResult>();

sortList = new ArrayList<RouteTimePrice>();

frame.model= (javax.swing.table.DefaultTableModel)frame.jTable1.getModel();

frame.model.addColumn("Depart");

frame.model.addColumn("Destination");

frame.model.addColumn("Dep\_time");

frame.model.addColumn("Arr\_time");

frame.model.addColumn("Price €");

frame.model.addColumn("Available");

frame.model.addColumn("Operator");

frame.model.addColumn("ticket\_id");

if (frame.model.getColumnCount()>8)

{frame.model.setColumnCount(8); }

for (int i = 0;i<pc.al\_sr.size();i++)

{ SearchResult sr = pc.al\_sr.get(i);

frame.appendText("\n"+ sr.toString());

frame.model.addRow(new Object[] {sr.dep,sr.arr,sr.dep\_time,sr.arr\_time,sr.price,sr.avai,sr.operator,sr.ticket\_id});

}

// try to put all 2 step routes into 2 list // start -> mid // mid -> end

// put all onr step route into sortList. RouteTimePrice list

for (int z = 0;z<finalList.size();z++)

{ for (int i = 0;i<pc.al\_sr.size();i++)

{ SearchResult sr = pc.al\_sr.get(i);

ArrayList al = finalList.get(z);

if (al.size() == 2)

{ String dep = (String)al.get(0);

String arr = (String)al.get(1);

if (sr.dep.equals( dep) && sr.arr.equals(arr))

{ directly.add(sr);

Date sr1Arr = stringToDate (sr.arr\_time);

Date sr1Dep = stringToDate (sr.dep\_time);

long time = sr1Arr.getTime() - sr1Dep.getTime();

RouteTimePrice rtp = new RouteTimePrice(time,sr.price,sr,null); sortList.add(rtp); } }

if (al.size() == 3)

{ String dep = (String)al.get(0);

String mid = (String)al.get(1);

String arr = (String)al.get(2);

if (sr.dep.equals( dep) && sr.arr.equals(mid))

{step1.add(sr); }

else if (sr.dep.equals(mid) && sr.arr.equals(arr))

{

step2.add(sr); }} } }

for (int l = 0;l < step1.size();l++)

{ SearchResult sr1 = step1.get(l);

Date sr1Arr = stringToDate (sr1.arr\_time);

Date sr1Dep = stringToDate (sr1.dep\_time);

for (int m = 0;m<step2.size();m++)

{

SearchResult sr2 = step2.get(m);

Date sr2Dep = stringToDate (sr2.dep\_time);

Date sr2Arr = stringToDate (sr2.arr\_time);

if (sr1.arr.equals(sr2.dep) && sr1Arr.compareTo(sr2Dep)<=0)

{

long time = sr2Arr.getTime() - sr1Dep.getTime();

double totalPrice = sr1.price + sr2.price;

RouteTimePrice rtp = new RouteTimePrice(time,totalPrice,sr1,sr2);

sortList.add(rtp);

}

}

}

if (sortList.size() != 0)

{

// final short time list, low price list

for(int n = 0; n< sortList.size();n++)

{

RouteTimePrice rtp = sortList.get(n);

if (shortTimeList.size() == 0)

{

shortTimeList.add(rtp);

}

else

{

if (rtp.time < shortTimeList.get(0).time)

{

shortTimeList.clear();

shortTimeList.add(rtp);

}

if (rtp.time == shortTimeList.get(0).time)

{

shortTimeList.add(rtp);

}

}

if (lowPriceList.size() == 0)

{ lowPriceList.add(rtp);

} else

{

if (rtp.price < lowPriceList.get(0).price)

{

lowPriceList.clear();

lowPriceList.add(rtp);

}

if (rtp.price == lowPriceList.get(0).price)

{

lowPriceList.add(rtp);

} } } } }

if (pc.content.equals("bookResult"))

{

returnCount--;

if (pc.bookId > 0)

{

bookReturnList.add(new BookReturn(pc.operator,pc.bookId));

System.out.println("put into hashmap book id : " + pc.bookId );

}

else if (pc.bookId == 0)

{

frame.appendText("\n ticket has been booked");

}

else

{

frame.appendText("\n unexpect error : connect or database.");

}

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

} }

public void stop()

{

try

{ if (console != null) console.close();

if (socket != null) socket.close();

}

catch(IOException ioe)

{

System.out.println("Error closing ...");

}

clientT.close();

thread = null;

}

public void buildMap() throws Exception

{

Statement s1 = conn.createStatement();

s1.executeQuery("SELECT distinct dep\_city FROM agent.route r");

ResultSet rs1 = s1.getResultSet();

nl = new ArrayList<Node>();

g=new Graph();

while (rs1.next())

{

String a = rs1.getString(1);

Node n = new Node(a);

nl.add(n);

g.addNode(n);

}

rs1.close ();

Statement s = conn.createStatement ();

s.executeQuery ("SELECT \* FROM agent.route r");

ResultSet rs = s.getResultSet ();

int count = 0;

while (rs.next ())

{

String dep\_id = rs.getString(1);

String des\_id = rs.getString(2);

int a = -1;

int b = -1;

for (int i = 0; i<nl.size();i++)

{

Node c = nl.get(i);

if (c.label.equals(dep\_id))

{

a = i;

}

if (c.label.equals(des\_id))

{

b = i;

}

}

g.connectNode(nl.get(a),nl.get(b));

} rs.close ();

}

// called by button click

public void searchRoute(String fi, String la)

{

finalList = null;

int a = -1;

int b = -1;

for (int i = 0; i<nl.size();i++)

{

Node c = nl.get(i);

if (c.label.equals(fi))

{

a = i;

}

if (c.label.equals(la))

{

b = i;

}

}

g.setRootNode(nl.get(a));

g.setLastNode(nl.get(b));

g.bfs();

finalList = g.finalList;

}

public PassClass getSearchObj(String s)

{

String DateStr = s;

ArrayList <SearchParam> set = new ArrayList <SearchParam>();

for (int i = 0;i<finalList.size();i++)

{

ArrayList al = finalList.get(i);

if (al.size() == 2)

{

String dep = (String)al.get(0);

String arr = (String)al.get(1);

SearchParam sp = new SearchParam(s,dep,arr);

set.add(sp);

}

if (al.size() == 3)

{

String dep = (String)al.get(0);

String mid = (String)al.get(1);

String arr = (String)al.get(2);

set.add(new SearchParam(s,dep,mid));

set.add(new SearchParam(s,mid,arr));

}

}

pc = new PassClass ("search","agent"); // pack up for send

pc.al\_sp = set;

for (int i = 0;i<set.size();i++)

{

SearchParam sp = set.get(i);

frame.appendText(sp.toString());

System.out.print(sp.toString());

}

return pc;

}

public synchronized ArrayList <BookReturn> agent\_cancel (String setTicket\_id)

{

ArrayList <BookReturn> al = new ArrayList <BookReturn>();

try {

CallableStatement get\_agent\_booked\_ticket = conn.prepareCall("{CALL agent.get\_agent\_booked\_ticket(?)}");

get\_agent\_booked\_ticket.setString(1, setTicket\_id);

boolean hadResult1 = get\_agent\_booked\_ticket.execute();

ResultSet rs1 = get\_agent\_booked\_ticket.getResultSet();

while (rs1.next())

{

String operator = rs1.getString(1);

int book\_id = rs1.getInt(2);

BookReturn br = new BookReturn(operator,book\_id);

al.add(br);

System.out.println("operator: " + operator +" "+ "book\_id : " + book\_id);

}

rs1.close();

CallableStatement cancel = conn.prepareCall("{CALL agent.agent\_cancel(?)}");

cancel.setString(1, setTicket\_id);

boolean hadResult = cancel.execute(); return al; }

catch (SQLException ex)

{ Logger.getLogger(Agent.class.getName()).log(Level.SEVERE, null, ex); }

return null; }

public synchronized void agent\_book(String operator,int bookid,String setTicket\_id)

{

try { CallableStatement book = conn.prepareCall("{CALL agent.agent\_book(?,?,?)}");

book.setString(1,operator); book.setInt(2, bookid); book.setString(3, setTicket\_id);

boolean hadResult = book.execute();}

catch (Exception e) { e.printStackTrace(); } }

public void bindComBox() throws SQLException

{ frame.jComboBox1.removeAllItems();

frame.jComboBox2.removeAllItems();

frame.jComboBox3.removeAllItems();

Statement s = conn.createStatement ();

s.executeQuery ("SELECT date(D) FROM agent.dd d;");

ResultSet rs = s.getResultSet ();

int count = 0;

while (rs.next ())

{ String dateStr = rs.getString(1); frame.jComboBox1.addItem(dateStr);

Statement s1 = conn.createStatement ();

s1.executeQuery ("SELECT distinct dep\_city FROM agent.route r");

ResultSet rs1 = s1.getResultSet ();

while (rs1.next ())

{ String cityStr = rs1.getString(1);

frame.jComboBox2.addItem(cityStr);

frame.jComboBox3.addItem(cityStr); } }

// remove client from client thread array,

public synchronized void remove(String operator)

{ int pos = findClient(operator);

if (pos >= 0){

AgentThread toTerminate = clients[pos];

System.out.println("Removing client thread " + operator);

if (pos < clientCount-1)

for (int i = pos+1; i < clientCount; i++)

clients[i-1] = clients[i];

clientCount--; toTerminate.close(); toTerminate = null; } }

public static void main(String args[])

{ Agent client = null; client = new Agent(); } }

### AgentThred.java

import java.net.\*;

import java.io.\*;

import java.util.ArrayList;

public class AgentThread extends Thread

{ private Socket socket = null;

private Agent client = null;

private ObjectInputStream in;

private ObjectOutputStream out;

private Thread thread;

private int ID= -1;

public boolean bookReturn ;

public AgentThread(Agent \_client, Socket \_socket)

{

client = \_client;

socket = \_socket;

ID = socket.getPort();

}

public void open()

{

try

{

in = new ObjectInputStream(socket.getInputStream());

out = new ObjectOutputStream(socket.getOutputStream());

}

catch(IOException ioe)

{

System.out.println("Error getting input stream: 11111" + ioe);

client.stop();

}

}

public void close()

{ try

{

if (in != null) in.close();

}

catch(IOException ioe)

{

System.out.println("Error closing input stream: " + ioe);

}

}

public int getID(){

return ID; }

public void send(String operator,PassClass pc) //(PassClass pc)

{

try{ out.writeObject(pc);

out.flush(); }

catch(IOException ioe)

{ System.out.println(ID + " ERROR sending: " + ioe.getMessage());

client.remove(operator);

thread=null; } } // thread listening to the server

public void run()

{

System.out.println("Server Thread " + ID + " running.");

thread = new Thread(this);

while (true && client!= null){

try {

PassClass pc = (PassClass)(in.readObject());

client.handle(ID,pc); }

catch(IOException ioe)

{ System.out.println("111111111111111");

client = null;

System.out.println("Listening error: " + ioe.getMessage());

ioe.printStackTrace();

}

catch (Exception e)

{

System.out.println("222222222222222222222");

client = null;

System.out.println("Listening error: " + e.getMessage());

e.printStackTrace();

} } }}

### BookRrturn.java

public class BookReturn {

public String operator;

public int book\_id;

public BookReturn (String s, int i)

{ operator = s;

book\_id = i; }}

### Node.java

public class Node

{ public String label;

public boolean visited=false;

public Node(String l)

{ this.label=l; }}

### PassClass.java

import java.io.Serializable;

import java.util.ArrayList;

public class PassClass implements Serializable{

public String content; // indicate the content you pass

public String operator; // which operator

public ArrayList<Route> al\_route = new ArrayList<Route>();

public ArrayList<SearchParam> al\_sp = new ArrayList<SearchParam>();

public ArrayList<SearchResult> al\_sr = new ArrayList<SearchResult>();

public int bookTicketNo = 0; public int bookId = -1; PassClass (String con, String op)

{ content = con; operator = op; }

public String toString()

{ String s = "\n"+content + ", " + operator; return s; }}

### Route.java

import java.io.Serializable;

public class Route implements Serializable{

public String dep;

public String arr;

Route (String s1,String s2)

{dep = s1; arr = s2; }

public String toString()

{String s = "\n"+ dep + ", " + arr ; return s; } }

### RouteTimePrice.java

public class RouteTimePrice {

public long time;

public double price;

public SearchResult sr1 = null;

public SearchResult sr2 = null;

public RouteTimePrice (long t,double p,SearchResult s1,SearchResult s2)

{ time = t; price = p; sr1 = s1; sr2 = s2; }}

### SearchParam.java

import java.io.Serializable;

public class SearchParam implements Serializable {

public String DateStr;

public String dep;

public String arr;

SearchParam (String s, String s1,String s2)

{DateStr = s; dep = s1; arr = s2; }

public String toString()

{String s = "\n"+DateStr + ", "+ dep + ", " + arr ; return s; }}

***SearchResult.***j***ava***

import java.io.Serializable;

public class SearchResult implements Serializable{

public String dep;

public String arr;

public String dep\_time;

public String arr\_time;

public double price;

public int avai;

public String operator;

public int ticket\_id;

SearchResult (String s1,String s2,String s3,String s4,double s5,int s6,String s7,int t)

{ dep = s1; arr = s2; dep\_time = s3; arr\_time = s4; price = s5; avai= s6;

operator = s7; ticket\_id = t; }

public String toString()

{String s = "\n"+ dep + ", " + arr + ", " +dep\_time +", " + arr\_time +", " + price +", " + avai + ", " + operator ; return s; } }

### Ticket.java

import java.io.Serializable;

public class Ticket implements Serializable{

public int ticket\_id;

public String dep\_time ;

public String arr\_time;

public String route\_route\_id;

public Ticket (int i, String s,String s1,String s2)

{ ticket\_id = i; dep\_time = s; arr\_time = s1; route\_route\_id = s2; }

public String toString()

{String s = "\n"+ticket\_id + ", " + dep\_time + ", " + arr\_time + ", " +route\_route\_id; return s; }}

### Train.java

import java.io.Serializable;

public class Train implements Serializable{

public String train\_id;

public int seat;

public Train (String s,int i)

{ train\_id = s; seat = i; }

public String toString()

{ String s = "\nTrain id: " + train\_id + ", seat: " + seat; return s; }}

### Graph.java

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;

public class Graph

{ArrayList<Node> route = new ArrayList<Node>();

public Node rootNode;

public Node lastNode;

public ArrayList nodes=new ArrayList();

public int[][] adjMatrix;//Edges will be represented as adjacency Matrix

int size; public ArrayList <ArrayList>finalList = null;

public ArrayList <ArrayList>interList = new ArrayList <ArrayList>();

public void setRootNode(Node n)

{ this.rootNode=n; }

public void setLastNode(Node n)

{this.lastNode = n; }

public Node getRootNode()

{return this.rootNode; }

public void addNode(Node n)

{ nodes.add(n); } //This method will be called to make connect two nodes

public void connectNode(Node start,Node end)

{ if(adjMatrix==null)

{size=nodes.size(); adjMatrix=new int[size][size]; }

int startIndex=nodes.indexOf(start);

int endIndex=nodes.indexOf(end);

adjMatrix[startIndex][endIndex]=1;

adjMatrix[endIndex][startIndex]=1; }

private Node getUnvisitedChildNode(Node n)

{ int index=nodes.indexOf(n); int j=0;

while(j<size)

{if(adjMatrix[index][j]==1 && ((Node)nodes.get(j)).visited==false)

{return (Node)nodes.get(j);} j++; }return null; }

//BFS traversal of a tree is performed by the bfs() function

public void bfs()

{ //BFS uses Queue data structure

Queue <ArrayList<Node>> q=new LinkedList<ArrayList<Node>>();

rootNode.visited=true; route.add(this.rootNode); q.add(route);

while(!q.isEmpty())

{ lastNode.visited = false;

ArrayList<Node> r = (ArrayList<Node>)q.remove();

interList.add(r); Node n = (Node)r.get(r.size()-1);Node child=null;

while((child=getUnvisitedChildNode(n))!=null )

{ArrayList<Node> al = new ArrayList<Node>();

for (int i=0;i<r.size();i++)

{al.add((Node)r.get(i) ); } child.visited=true; al.add(child); q.add(al);}}

//Clear visited property of nodes

clearNodes();

finalList = new ArrayList <ArrayList>();

for (int i = 0;i<interList.size();i++)

{boolean fn = false; boolean ln = false;

ArrayList<Node> aa = (ArrayList<Node>) interList.get(i);

if (aa.size() > 3) continue;

ArrayList intList = new ArrayList();

for (int m = 0; m < aa.size();m++)

{Node n = aa.get(m); intList.add(n.label);

if (n.label.equals(this.rootNode.label))

{fn = true; }

if (m == aa.size()-1)

{if (n.label.equals(this.lastNode.label))

{ln = true;}}}

if (fn && ln)

{finalList.add(intList);}}

System.out.print("\n xxxxxxxxxxxxxxxx " + this.size);

System.out.print("\n" );

for (int i = 0;i<finalList.size();i++)

{ArrayList aa = finalList.get(i);

for (int m = 0; m < aa.size();m++)

{System.out.print(aa.get(m) + " ");

}System.out.print("\n");}}

//DFS traversal of a tree is performed by the dfs() function

public void dfs()

{//DFS uses Stack data structure

Stack s=new Stack();

s.push(this.rootNode);

rootNode.visited=true;

printNode(rootNode);

while(!s.isEmpty())

{ Node n=(Node)s.peek();

Node child=getUnvisitedChildNode(n);

if(child!=null)

{ child.visited=true;

printNode(child);

s.push(child);}

else

{s.pop(); } }

//Clear visited property of nodes

clearNodes(); }

//Utility methods for clearing visited property of node

private void clearNodes()

{ int i=0;

while(i<size)

{Node n=(Node)nodes.get(i); n.visited=false; i++; } }

//Utility methods for printing the node's label

private void printNode(Node n)

{ System.out.print(n.label+" "); }}

***GUI.***j***ava***

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JOptionPane;

import javax.swing.table.DefaultTableModel;

public class GUI extends javax.swing.JFrame {

private Agent ac = null;

public DefaultTableModel model = model = new javax.swing.table.DefaultTableModel();

/\*\* Creates new form GUI \*/

public GUI(Agent a) {

ac = a;

initComponents(); }

private void initComponents() {

jPanel1 = new javax.swing.JPanel();

jPanel2 = new javax.swing.JPanel();

jScrollPane1 = new javax.swing.JScrollPane();

jTextArea1 = new javax.swing.JTextArea();

jPanel3 = new javax.swing.JPanel();

jLabel1 = new javax.swing.JLabel();

jComboBox1 = new javax.swing.JComboBox();

jLabel2 = new javax.swing.JLabel();

jComboBox2 = new javax.swing.JComboBox();

jComboBox3 = new javax.swing.JComboBox();

jLabel3 = new javax.swing.JLabel();

jButton1 = new javax.swing.JButton();

jButton3 = new javax.swing.JButton();

jButton5 = new javax.swing.JButton();

jPanel4 = new javax.swing.JPanel();

jScrollPane2 = new javax.swing.JScrollPane();

jTable1 = new javax.swing.JTable();

jPanel5 = new javax.swing.JPanel();

jButton2 = new javax.swing.JButton();

jButton4 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

jTextArea1.setColumns(20);

jTextArea1.setRows(5);

jScrollPane1.setViewportView(jTextArea1);

org.jdesktop.layout.GroupLayout jPanel2Layout = new org.jdesktop.layout.GroupLayout(jPanel2);

jPanel2.setLayout(jPanel2Layout);

jPanel2Layout.setHorizontalGroup(

jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel2Layout.createSequentialGroup()

.addContainerGap()

.add(jScrollPane1, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, 690, Short.MAX\_VALUE)

.addContainerGap())

);

jPanel2Layout.setVerticalGroup(

jPanel2Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel2Layout.createSequentialGroup()

.addContainerGap()

.add(jScrollPane1, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, 107, Short.MAX\_VALUE))

);

jLabel1.setText("Date :");

jLabel2.setText("Depart :");

jComboBox2.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT\_CURSOR));

jLabel3.setText("Destination : ");

jButton1.setText("Search");

jButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton1ActionPerformed(evt); } });

jButton3.setText("Shorest Time");

jButton3.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton3ActionPerformed(evt); } });

jButton5.setText("Lowest Price");

jButton5.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton5ActionPerformed(evt); } });

org.jdesktop.layout.GroupLayout jPanel3Layout = new org.jdesktop.layout.GroupLayout(jPanel3);

jPanel3.setLayout(jPanel3Layout);

jPanel3Layout.setHorizontalGroup(

jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel3Layout.createSequentialGroup()

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel3Layout.createSequentialGroup()

.addContainerGap()

.add(jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)

.add(org.jdesktop.layout.GroupLayout.LEADING, jComboBox3, 0, 142, Short.MAX\_VALUE)

.add(org.jdesktop.layout.GroupLayout.LEADING, jLabel3)

.add(org.jdesktop.layout.GroupLayout.LEADING, jLabel1)

.add(org.jdesktop.layout.GroupLayout.LEADING, jLabel2)

.add(org.jdesktop.layout.GroupLayout.LEADING, jComboBox2, 0, 142, Short.MAX\_VALUE)

.add(jComboBox1, 0, 142, Short.MAX\_VALUE)))

.add(jPanel3Layout.createSequentialGroup()

.add(24, 24, 24)

.add(jButton1, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 106, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.add(0, 0, Short.MAX\_VALUE))

.add(jPanel3Layout.createSequentialGroup()

.addContainerGap()

.add(jButton3, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

.add(jPanel3Layout.createSequentialGroup()

.addContainerGap()

.add(jButton5, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))

.addContainerGap()) );

jPanel3Layout.setVerticalGroup(

jPanel3Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel3Layout.createSequentialGroup()

.addContainerGap()

.add(jLabel1)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jComboBox1, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.add(18, 18, 18)

.add(jLabel2)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jComboBox2, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.add(18, 18, 18)

.add(jLabel3)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jComboBox3, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.add(41, 41, 41)

.add(jButton1, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 31, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jButton3, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 31, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jButton5, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 31, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.addContainerGap(59, Short.MAX\_VALUE))

);

jScrollPane2.setViewportView(jTable1);

org.jdesktop.layout.GroupLayout jPanel4Layout = new org.jdesktop.layout.GroupLayout(jPanel4);

jPanel4.setLayout(jPanel4Layout);

jPanel4Layout.setHorizontalGroup(

jPanel4Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel4Layout.createSequentialGroup()

.add(jScrollPane2, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, 520, Short.MAX\_VALUE)

.addContainerGap())

);

jPanel4Layout.setVerticalGroup(

jPanel4Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jScrollPane2, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 0, Short.MAX\_VALUE)

);

jButton2.setText("Book");

jButton2.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton2ActionPerformed(evt);

}

});

jButton4.setText("Cancel");

jButton4.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jButton4ActionPerformed(evt);

}

});

org.jdesktop.layout.GroupLayout jPanel5Layout = new org.jdesktop.layout.GroupLayout(jPanel5);

jPanel5.setLayout(jPanel5Layout);

jPanel5Layout.setHorizontalGroup(

jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(org.jdesktop.layout.GroupLayout.TRAILING, jPanel5Layout.createSequentialGroup()

.addContainerGap(279, Short.MAX\_VALUE)

.add(jButton4, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 111, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.add(18, 18, 18)

.add(jButton2, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, 112, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.addContainerGap())

);

jPanel5Layout.setVerticalGroup(

jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel5Layout.createSequentialGroup()

.addContainerGap()

.add(jPanel5Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jButton4, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, 34, Short.MAX\_VALUE)

.add(jButton2, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, 34, Short.MAX\_VALUE)))

);

org.jdesktop.layout.GroupLayout jPanel1Layout = new org.jdesktop.layout.GroupLayout(jPanel1);

jPanel1.setLayout(jPanel1Layout);

jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel2, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.add(jPanel1Layout.createSequentialGroup()

.add(jPanel3, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.add(18, 18, 18)

.add(jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.TRAILING)

.add(jPanel5, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.add(org.jdesktop.layout.GroupLayout.LEADING, jPanel4, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))

);

jPanel1Layout.setVerticalGroup(

jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel1Layout.createSequentialGroup()

.add(jPanel2, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel1Layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(org.jdesktop.layout.GroupLayout.TRAILING, jPanel1Layout.createSequentialGroup()

.add(jPanel4, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED)

.add(jPanel5, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED\_SIZE))

.add(jPanel3, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)))

);

org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel1, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

);

layout.setVerticalGroup(

layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)

.add(jPanel1, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, org.jdesktop.layout.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

);

pack();

}// </editor-fold>

// search button

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

String dateStr = (String) jComboBox1.getSelectedItem();

String depStr = (String) jComboBox2.getSelectedItem();

String arrStr = (String) jComboBox3.getSelectedItem();

jTextArea1.append("\n" + dateStr + depStr + arrStr );

if (depStr.equals(arrStr))

{

jTextArea1.append("\nyou can not select same city as dep and arr" );

}

else

{

//route build map

try{

ac.buildMap();

}

catch(Exception ee)

{

ee.printStackTrace();

}

ac.searchRoute(depStr ,arrStr); // get the route

model.setColumnCount(0);

model.setRowCount(0);

Set set = ac.hashmap.entrySet();

Iterator iterator = set.iterator();

while (iterator.hasNext() )

{

Map.Entry mapentry = (Map.Entry) iterator.next();

String s = (String)mapentry.getKey();

ac.sendTextToChat(s,ac.getSearchObj(dateStr)); // build object and send.

}

}

}

// book button

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

int[] rowIndexs = jTable1.getSelectedRows();

if (rowIndexs.length == 0)

{

jTextArea1.append("\nyou have to select the tickets");

}

else

{

ac.returnCount = rowIndexs.length;

ac.bookReturnList = new ArrayList<BookReturn>();

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

{

int rowNum = rowIndexs[i];

String operator = (model.getValueAt(rowNum, model.getColumnCount()-2)).toString();

String s = model.getValueAt(rowNum, model.getColumnCount()-1).toString();

int ticket\_id = Integer.parseInt(s);

PassClass pc = new PassClass("book","agent");

pc.bookTicketNo = ticket\_id;

ac.sendTextToChat(operator,pc);

}

for (int i = 0;i<5;i++)

{

if (ac.returnCount==0)

{

break;

}

try {

Thread.sleep(1000);

} catch (InterruptedException ex) {

Logger.getLogger(GUI.class.getName()).log(Level.SEVERE, null, ex);

}

}

System.out.println("return finish :" + ac.returnCount);

if (ac.bookReturnList.size() != rowIndexs.length)

{

ArrayList <BookReturn> al = ac.bookReturnList;

for (int i = 0; i < al.size();i++)

{

BookReturn br = al.get(i);

String operator = br.operator;

int id = br.book\_id;

PassClass pc = new PassClass("cancel",operator);

pc.bookId = id;

ac.sendTextToChat(operator,pc);

}

System.out.println("==========failure");

}

else

{

String setTicket\_id = "";

for (int i = 0 ;i< ac.bookReturnList.size();i++)

{

BookReturn br = ac.bookReturnList.get(i);

int bookid = br.book\_id;

if (i == ac.bookReturnList.size()-1)

{

setTicket\_id = setTicket\_id + bookid;

}

else {

setTicket\_id = setTicket\_id + bookid+ "\_";

}

}

System.out.println("setTicket\_id : " + setTicket\_id);

for (int i = 0 ;i< ac.bookReturnList.size();i++)

{

BookReturn br = ac.bookReturnList.get(i);

String operator = br.operator;

int bookid = br.book\_id;

ac.agent\_book(operator,bookid,setTicket\_id);

System.out.println(operator + bookid + setTicket\_id);

}

}

}

}

// cancel button

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

String setTicket = JOptionPane.showInputDialog("Insert your ticket num");

if (!setTicket.equals(null))

{

ArrayList <BookReturn> al = ac.agent\_cancel(setTicket);

for (int i = 0; i < al.size();i++)

{

BookReturn br = al.get(i);

String operator = br.operator;

int id = br.book\_id;

PassClass pc = new PassClass("cancel",operator);

pc.bookId = id;

ac.sendTextToChat(operator,pc);

}

}

}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {

ac.showShortestTimeRoutes();

}

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {

ac.showLowestPriceRoutes();

}

public static void main(String args[]) {

}

// Variables declaration - do not modify

private javax.swing.JButton jButton1;

private javax.swing.JButton jButton2;

private javax.swing.JButton jButton3;

private javax.swing.JButton jButton4;

private javax.swing.JButton jButton5;

public javax.swing.JComboBox jComboBox1;

public javax.swing.JComboBox jComboBox2;

public javax.swing.JComboBox jComboBox3;

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JLabel jLabel3;

private javax.swing.JPanel jPanel1;

private javax.swing.JPanel jPanel2;

private javax.swing.JPanel jPanel3;

private javax.swing.JPanel jPanel4;

private javax.swing.JPanel jPanel5;

private javax.swing.JScrollPane jScrollPane1;

private javax.swing.JScrollPane jScrollPane2;

public javax.swing.JTable jTable1;

private javax.swing.JTextArea jTextArea1;

// End of variables declaration

public void appendText(String s)

{

jTextArea1.append(s); }}

### OperatorThread.ava

import java.net.\*;

import java.io.\*;

public class OperatorThread extends Thread

{ private operaterServer server = null;

public Socket socket = null;

private int ID = -1;

private ObjectInputStream streamIn = null;

private ObjectOutputStream streamOut = null;

private Thread thread;

public OperatorThread(operaterServer \_server, Socket \_socket)

{

super();

server = \_server;

socket = \_socket;

ID = socket.getPort();

}

// send data to client.

public void send(PassClass pc) //(PassClass pc)

{

try{

streamOut.writeObject(pc);

streamOut.flush();

}

catch(IOException ioe)

{

System.out.println(ID + " ERROR sending: " + ioe.getMessage());

server.remove(ID);

thread=null;

}

}

public int getID(){

return ID;

}

// server listening the certain client, receive query and response

public void run()

{

System.out.println("Server Thread " + ID + " running.");

thread = new Thread(this);

try {

PassClass pc = server.getRouteTable();

server.broadcast(getID(), pc);

} catch (Exception e) {

e.printStackTrace();

}

try {

streamIn = new ObjectInputStream (socket.getInputStream());

} catch (IOException e1) {

e1.printStackTrace();

}

while (true){

try{

//server get message from client, then response

PassClass pc = (PassClass)streamIn.readObject();

System.out.print("----- server receive");

if (pc.content.equals("search"))

{

PassClass searchResult = server.searchTicket(pc);

server.broadcast(ID, searchResult);

}

else if(pc.content.equals("book"))

{

System.out.print(pc.toString());

System.out.print("ticket: "+ pc.bookTicketNo);

PassClass bookResult = server.bookTicket(pc);

server.broadcast(ID, bookResult);

}

else if (pc.content.equals("cancel"))

{

System.out.print(pc.toString());

server.cancelTicket(pc);

}

else {

System.out.print("else");

}

}

catch(IOException ioe){

server.remove(ID);

thread = null;

}

catch(Exception e)

{

e.printStackTrace();

}

}

}

public void open() throws IOException

{

System.out.println("open start");

streamOut = new ObjectOutputStream(socket.getOutputStream());

System.out.println("open finish");

}

public void close() throws IOException

{

if (socket != null)

{socket.close();}

if (streamIn != null)

{streamIn.close();}

if (streamOut != null)

{streamOut.close();}

}

}

### operaterServer.java

import java.net.\*;

import java.sql.\*;

import java.util.ArrayList;

import java.util.Timer;

import java.util.TimerTask;

import java.util.Vector;

import java.io.\*;

public class operaterServer implements Runnable

{

// Array of clients threads

private OperatorThread clients[] = new OperatorThread[50];

private ServerSocket server = null;

private Thread thread = null;

private int clientCount = 0;

private Socket s;

private Connection conn = null;

public void connect() throws Exception

{

String userName = "root";

String password = "admin";

String url = "jdbc:mysql://localhost/test";

Class.forName ("com.mysql.jdbc.Driver").newInstance ();

conn = DriverManager.getConnection (url, userName, password);

System.out.println ("Database connection established");

}

public PassClass getRouteTable() throws Exception

{

PassClass pc = null;

Statement s = conn.createStatement ();

s.execute ("CALL operator2.getRouteCity()");

ResultSet rs = s.getResultSet ();

int count = 0;

ArrayList <Route> set = new ArrayList <Route>();

while (rs.next ())

{

String dep = rs.getString (1);

String arr = rs.getString (2);

System.out.println (

dep + ", " + arr

);

set.add(new Route(dep,arr));

++count;

}

rs.close ();

s.close ();

System.out.println (count + " rows were retrieved");

pc = new PassClass ("route","operator2"); // pack up for send

pc.al\_route = set;

return pc;

}

public synchronized PassClass bookTicket(PassClass pc)

{

PassClass bookResult = new PassClass("bookResult","operator2");

Boolean hadResult = false;

int book\_id = 0;

try {

CallableStatement book = conn.prepareCall("{CALL operator2.insert\_book(?)}");

book.setInt(1, pc.bookTicketNo);

hadResult = book.execute();

if (hadResult)

{

ResultSet rs = book.getResultSet();

while (rs.next ())

{

book\_id = rs.getInt(1);

bookResult.bookId = book\_id;

}

}

else

{

bookResult.bookId = -1;

}

} catch (Exception e) {

bookResult.bookId = -1;

e.printStackTrace();

}

System.out.println("bookResult" + bookResult.bookId);

return bookResult;

}

// book ticket

public synchronized void cancelTicket(PassClass pc)

{

try {

int id = pc.bookId;

CallableStatement cancel = conn.prepareCall("{CALL operator2.delete\_book(?)}");

cancel.setInt(1, pc.bookId);

boolean hadResult = cancel.execute();

}

catch (Exception e)

{

e.printStackTrace();

}

}

//search tickets

public PassClass searchTicket(PassClass pc)

{

PassClass searchResult = null;

System.out.println("server search tickets..............");

ArrayList <SearchResult> set = new ArrayList <SearchResult>();

for (int i = 0;i<pc.al\_sp.size();i++)

{

Boolean hadResult = false;

SearchParam sp = pc.al\_sp.get(i);

try {

CallableStatement getTicket = conn.prepareCall("{call operator2.search\_ticket(?,?,?)}");

getTicket.setString(1, sp.dep);

getTicket.setString(2, sp.arr);

getTicket.setString(3, sp.DateStr);

hadResult = getTicket.execute();

if (hadResult)

{

ResultSet rs = getTicket.getResultSet();

while (rs.next ())

{

String dep = rs.getString (1);

String arr = rs.getString (2);

String dep\_time = rs.getString (3);

String arr\_time =rs.getString (4);

double price = rs.getDouble (5);

int avai =rs.getInt (6);

String operator =rs.getString (7);

int ticket = rs.getInt(8);

SearchResult sr = new SearchResult(dep,arr,dep\_time,arr\_time,price,avai,operator,ticket);

set.add(sr);

System.out.print(sr.toString());

}

}

} catch (Exception e) {

e.printStackTrace();

}

}

searchResult = new PassClass ("SearchResult","operator2"); // pack up for send

searchResult.al\_sr = set;

return searchResult;

}

public operaterServer (int port)

{

try {

connect();

System.out.println("Binding to port " + port + ", please wait ...");

server = new ServerSocket(port);

System.out.println("Server started: " + server.getInetAddress());

start();

}

catch(IOException ioe)

{

System.out.println("Can not bind to port " + port + ": " + ioe.getMessage());

}

catch (Exception e)

{

System.err.println ("Cannot connect to database");

e.printStackTrace();

} }

public void run()

{

while (thread != null)

{

try{

System.out.println("Waiting for a client ...");

s = server.accept();

addThread(s); //declare a new client and put it in the array

}

catch(IOException ioe){

System.out.println("Server accept error: " + ioe);

stop();

}

catch (Exception e)

{

System.out.println("server can not get data from db: " + e);

stop();

}

}

}

public void start()

{

if (thread == null) {

thread = new Thread(this);

thread.start();

}

}

public void stop(){

thread = null;

}

private int findClient(int ID)

{

for (int i = 0; i < clientCount; i++)

if (clients[i].getID() == ID)

return i;

return -1;

}

// receive query from client and reply (ticket search, book) and reply. call from server thread

public synchronized void broadcast(int ID, PassClass input)

{

clients[findClient(ID)].send(input);

System.out.println("broadcast");

}

// remove client from client thread array,

public synchronized void remove(int ID)

{

int pos = findClient(ID);

if (pos >= 0){

OperatorThread toTerminate = clients[pos];

System.out.println("Removing client thread " + ID);

if (pos < clientCount-1)

for (int i = pos+1; i < clientCount; i++)

{ clients[i-1] = clients[i];}

clientCount--;

try{

toTerminate.close();

}

catch(IOException ioe)

{

System.out.println("Error closing thread: " + ioe);

}

toTerminate = null;

notifyAll();

}

}

// add new client thread to array.

private void addThread(Socket socket)

{

System.out.println("in add thread");

if (clientCount < clients.length){

System.out.println("Client accepted: " + socket);

clients[clientCount] = new OperatorThread(this, socket);

System.out.println("new instance");

try{

clients[clientCount].open();

clients[clientCount].start();

clientCount++;

}

catch(IOException ioe){

System.out.println("Error opening thread: " + ioe);

}

catch (Exception e)

{

e.printStackTrace();

}

System.out.println("finish add thread");

}

else

{System.out.println("Client refused: maximum " + clients.length + " reached.");}

}

public static void main(String args[]) {

operaterServer server = null;

//server = new operaterServer(1234);

server = new operaterServer(Integer.parseInt(args[0]));

}

}