Design Document

My Hotel Booking System

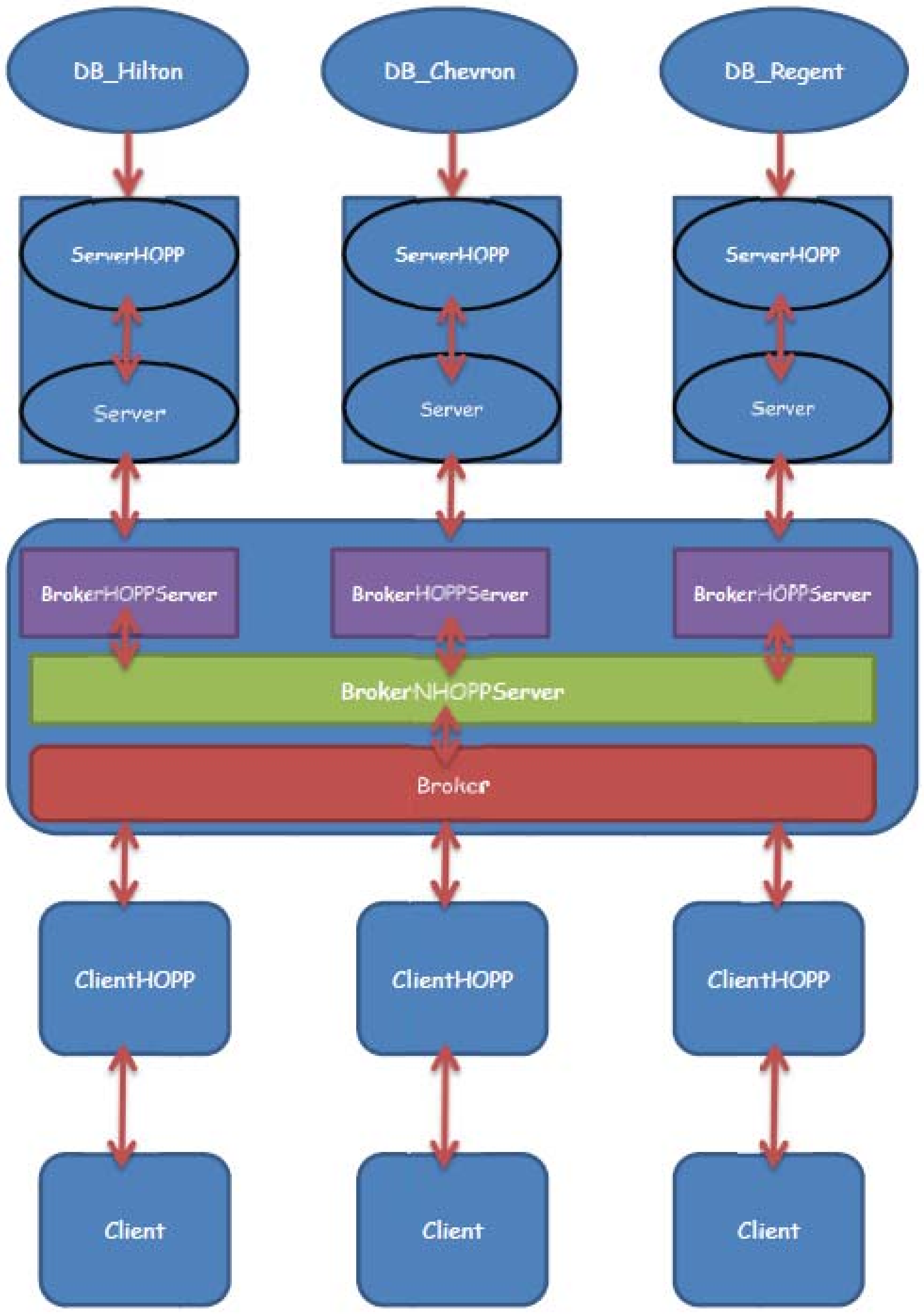
2015/4/2

Yu Jun 26346702

Design Document for My Hotel Booking System

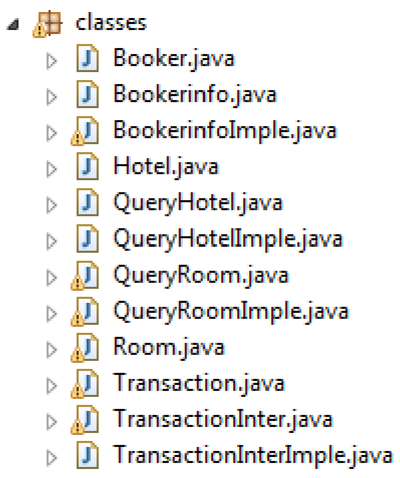
Content

1. The Model of the Hotel System



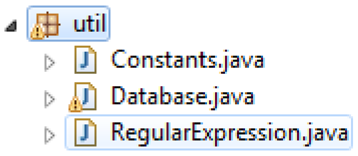
Comments: you can see from this above model, and this system has three Database and corresponding three Server: Hilton Server, Regent Server, Chevron Server. The Servers can get the records of the Database by invoking the ServerHOPP Object. Then the Server opens a socket to get connection with BrokerHOPPServer. And the BrokerHOPPServer has an array of BrokerHOPPServer with the size 3. So thanks to this Object, the Broker can get the information from these 3 Servers, and then pass it to the ServerHOPP. Between the ServerHOPP and Database, and also between the BrokerHOPPServer and the Server, these two half- objects will communicate using the private protocol that is unknown to the rest of application. This is the Half Object plus Protocol. This allows the implementation of the HOPP objects to be changed without affecting the rest of the application.

2. The main Class of my Java Program:



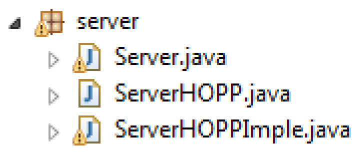
```
classes
  Booker.java
  Bookerinfo.java
  BookerinfoImple.java
  Hotel.java
  QueryHotel.java
  QueryHotelImple.java
  QueryRoom.java
  QueryRoomImple.java
  Room.java
  Transaction.java
  TransactionInter.java
  TransactionInterImple.java
```

1) Hotel, Room and Transaction are used to define the properties of the hotel, room and transaction and are also used to maintain the information from Database. What's more, QueryRoom, QueryHotel, TransactionInter, Bookerinfo are three interfaces to define the functions I may use later. These three interfaces are implemented by QueryRoomImple, QueryHotelImple, TransactionInterImple and Bookerinfoimple respectively. These three classes are used to query or update the tables in the database.



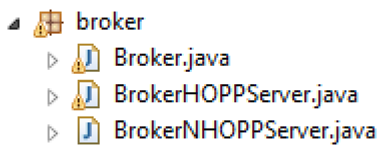
```
util
  Constants.java
  Database.java
  RegularExpression.java
```

2) In this package, I define the Constants Class to store the basic constants I would use very often. Database Class is used to initialize the records in the Database. RegularExpression Class is used to normalize the data the client inputs in.



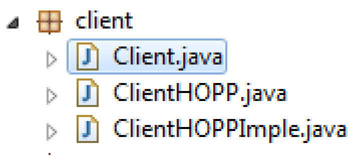
server
Server.java
ServerHOPP.java
ServerHOPPImple.java

3) In this package, ServerHOPP is the interface and implemented by the Class ServerHOPPImple which I use to get the record from the database by invoking the Classes (like QueryRoomImple)in the classes package. Server Class is used to establish the connection between Server and BrokerHOPPServer by invoking ServerSocket Class with port Number starting from 26341 to 26343.



broker
Broker.java
BrokerHOPPServer.java
BrokerNHOPPServer.java

4) In this package, I define the brokerHOPPServer class to communicate with the Server by invoking the Socket with port NO. from 26341 to 26343. Similarly, I also define the BrokerNHOPPServer. In this Class, I can create Multiple instances of BrokerHOPPServer through which we can get information from multiple Servers. The broker Class is acting like a server by getting connection to the ClientHOPP which we would talk about it later. The Broker invokes the ServerSocket to establish the channel between ClientHOPP and Broker, through this Broker, the Client can get the information it requested.



client
Client.java
ClientHOPP.java
ClientHOPPImple.java

5) The ClientHOPP is an interface which is implemented by the ClientHOPPImple class. In the ClientHOPPImple, I define the required function to exchange the flow between this class and Broker using OutputStream and InputStream Class. It also invokes the Socket Class to get connection to the Broker port. In the Client, I tend to use the text-based interface, reading from System.in and writing to System.out.

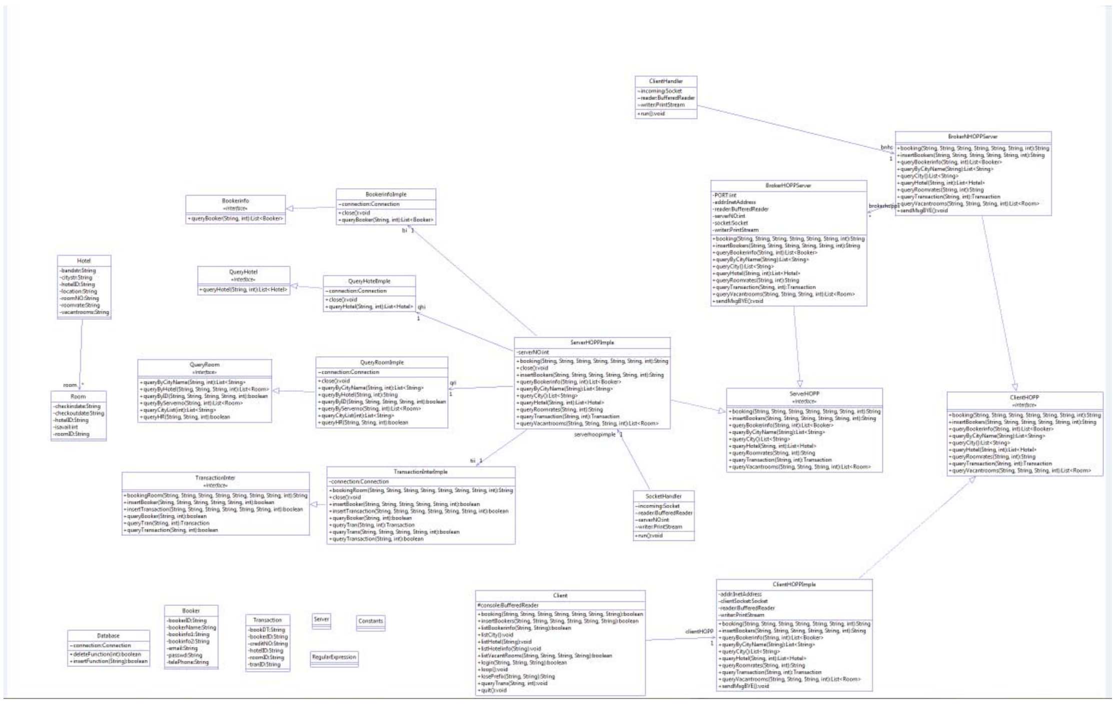
3. The message Format for transmission

Between the ClientHOPP and Broker, also between BrokerHOPPServer and Server, these two Class as processes when running communicate with each other using the message format below.

	Message Format	Meaning
QueryCity	city\r\n	Request the city list
	SOS\r\n	Indicate the start of stream
	NO\r\n	Indicate the city list is null
	YES\r\n	Indicate the city list is not null
	EOS\r\n	Indicate the end stream of querycity
QueryHotel	hotelname\r\n	Request the hotel list
	NO\r\n	Indicate the hotel list is null
	YES\r\n	Indicate the hotel list is not null
	EOS\r\n	Indicate the end stream of queryhotel
queryroomrate	Roomrate\r\n	Request the roomrate
	EOS\r\n	Indicate the end of parameters.
	NO\r\n	Indicate the room rate is null
	YES\r\n	Indicate the room rate is not null
queryBookerinfo	bookers\r\n	Request the bookers list
	EOS\r\n	Indicate the end of parameters.
	NO\r\n	Indicate the booker list is null
	YES\r\n	Indicate the booker list is not null
	EOS\r\n	Indicate the end of booker list.
querytransaction	Trans\r\n	Request the transaction list
	EOS\r\n	Indicate the end of parameters.
	NO\r\n	Indicate the transaction is null
	YES\r\n	Indicate the transaction is not null
Queryvacantrooms	Vacancies\r\n	Request the vacant rooms list
	EOS\r\n	Indicate the end of parameters.
	NO\r\n	Indicate the vacant rooms is null
	YES\r\n	Indicate the vacant rooms is not null
	EOS\r\n	Indicate the end of room vacant list.

booking	Booking\r\n	Request the book the rooms.
	EOS\r\n	Indicate the end of parameters.
	YES\r\n	Indicate you can start booking
	Bookinfo\r\n	To see whether the booking is succeed.
Queryhotel	Hotels\r\n	Request the Query hotels.
	EOS\r\n	Indicate the end of parameters.
	NO\r\n	Indicate the vacant hotels is null
	YES\r\n	Indicate the vacant hotels is not null
	EOS\r\n	Indicate the end of hotel list.

4. The UML diagram



Description: if you can see this UML diagram clearly, you can open the picture at this directory. This UML diagram show the relationship between the parent class and the extended classes. The Booker, Hotel, Transaction and the Room are the Classes needed to describe these particular objects and also store the information from the Database, and then Bookerinfo, QueryHotel, QueryRoom and TransactionInter are the interfaces we would invoke later, in the ServerHOPPImple Class, this class would invoke the different functions to get the information from the data , and then pass the information to the Server, the Server would directly send this information to the BrokerHOPPServer by using the Socket to get connection. similarly, the

BrokerHOPPServer would extend the interface ServerHOPP and override the functions of the interface, and by invoking the function to get the information from the Server. The ClientHOPPImple extends the ClientHOPP, BrokerNHOPPServer extends the same ClientHOPP, they would similarly communicate with each other.