HOTEL ROOM BOOKING System

Contents

[INTRODUCTION 2](#_Toc267685124)

[Design overview 2](#_Toc267685125)

[Design Rationale 2](#_Toc267685126)

[Data layer (DB package) 3](#_Toc267685127)

[Functionalities 3](#_Toc267685128)

[Implementation 3](#_Toc267685129)

[Problems encountered 5](#_Toc267685130)

[User Interface (GUI specification) 5](#_Toc267685131)

[Functionalities 5](#_Toc267685132)

[Implementation 5](#_Toc267685133)

[Problems encountered 6](#_Toc267685134)

[Server (The data server and its network connection) 6](#_Toc267685135)

[RMI 6](#_Toc267685136)

[Functionalities 7](#_Toc267685137)

[Implementations: 7](#_Toc267685138)

[RMI Server 7](#_Toc267685139)

[Implementations 7](#_Toc267685140)

[RMI Client 7](#_Toc267685141)

[Implementation 7](#_Toc267685142)

[RMI Server Port 8](#_Toc267685143)

[OTHER DECISIONS / MISCELLANEOUS 8](#_Toc267685144)

[LOG 8](#_Toc267685145)

[Logger functionalities 8](#_Toc267685146)

[Implementation 8](#_Toc267685147)

[10 CONCLUSION 8](#_Toc267685148)

# INTRODUCTION

URLyBird is a broker of discount hotel rooms. The company sells accommodations for business and pleasure travellers at short notice, helping hotels to fill rooms that would otherwise be left empty. They take bookings only within 48 hours of the start of room occupancy. URLyBird currently sells rooms over the phone using a custom-written application which has come under increasing criticism.

A Hotel room booking system designed in java was considered as an alternative because in addition to solving the problem posed by the current application, it is cross platform (portable) and therefore makes it easier for the company to later migrate to the web.

## Design overview

This system was designed using the 3 tiers system architecture called MVC. The reason for this approach was to breakdown the system into 3 layers based on their functionalities. The Model or the data layer for storing and accessing information, controller for enforcing the business logic of the system and the view which is responsible for presenting the information to the end user.

The adopted methodology used is the bottom up approach. This approach was considered in the design for the simple reason of first developing the data layer then followed by the other components that depended on it such as the controller and view.

The application was designed to function as a stand alone system in a non networked environment where the data is embedded into system. It could also function in a network mode where it connects over the network to access information stored on another system. In order to fulfil this requirement, the client and server architecture was considered.

In a network mode, the system was designed so that it can be ran as a client application that connects to another instance of the application running locally or remotely in a server mode to retrieve the information.

Applying Object oriented analysis, nouns like Hotel room from the problem statement provided was identified. This noun led to the identification of a class to be implemented. The attributes of this class were identified and methods such as getter and setter were created to access these variables. A good example can be found in the HotelRoom class that models the database records. Since the characteristics of Object oriented programming facilitates re-use, the supplied interface such as DBAccess was re-used and implemented.

In addition to class creation, the legacy java classes were re-used in our design where considered suitable and some were extended to suit the design. The adopted Java platform used is JDK version 1.6 because it has not been superseded by a newer one for more than 18 months.

# Design Rationale

This section consists of the design choices made while implementing some of the functionalities of the system. These choices include the assumptions made during design, the problem encountered and the adopted solution and the reason behind it.

## Data layer (DB package)

## Functionalities

1. Data formatting: Ensures that the information stored and loaded from the database file are in the right format as provided in the requirements.
2. Access security: Controls access to the database by granting access to the legitimate user to carry out the required operation such as delete and updates.
3. CRUD: Implements create, read and update and delete operations on the database file.
4. Business logic requirements: Implements the end user operation such as search and book and hence it enforces the logic when these operations are carried out.
5. Hiding network complexities: hides the connectivity and access to the database from the end user in the network and non network environment. So that from the end user perspective the same operations are carried using one interface without knowing if it is a local client or the server operating.

## Implementation

1. To extract the information from the database file, two java.io classes were considered; RandomAccessFile and DataInputStream. A RandomAccessFile was preferred because the seek time is constant and faster than that of the DataInputStream (sequential files access). In order to use RandomAccessFile, the position to seek in file was calculated by using a file offset. This offset was calculated using the fixed record length that was provided in the requirement. By also reading and storing into fixed field length array, the information format in the database file was preserved.
2. To control access to the database in a multithreaded environment, the java concurrent utilities and normal thread synchronisation method were considered. The concurrent class such as ReadWriteLock and Condition were preferred because they offer more functionality in addition to normal thread approach.

In order to provide authentication to the user, the lock cookie was used in addition to the record number. This cookie needed to be stored temporarily in a data structure for later authentication. Two choices were considered for this data structure; List and HashMap.

The HashMap was preferred because it stores the association of a key to a value. In this scenario, this key was the record number and the value as the lock cookie.

1. The business logic is enforced during booking operation by comparing the empty room’s available date to the system date and ensuring that the difference between the two is within 48hrs from the start of the room’s available date. In order to compare the rooms during booking, the data must be cached to reduce round trip time of retrieving information. The choice of cached memory is a collection rather than a fixed length array because collection resizes itself dynamically at runtime. The choice of collection used for the cache is a HashMap rather than a List; although both are used to store collection objects, the map allows us to store and associate a record number to a HotelRoom Object. Since these record numbers are for non deleted records, they are not linear and hence the list is eliminated from our choice.

In addition, to allow the end user to be able to retrieve information based on a particular search criterion. Two different string comparison methods were considered; namely using the methods in java.lang.string or the regular expression methods in java.util.regex. The java.lang.string was preferred because the amount field contains a currency sign ($) which has a particular meaning in regular expression and makes it hard to match.

1. In order to hide the complexities of network mode in a local client, a client interface with methods that throw IOException is provided. This is to allow the server to implement this interface using an adapter pattern to map the client request to the correct remote methods at the server side which throws RemoteException that is a type of IOException.

## Problems encountered

1. How to prevent an illegitimate user from unlocking the record. A hacker may be able to corrupt the record if they successful guess the lock cookie that was generated using a random number. In order to make it difficult to guess, the system time in millisecond was used as a lock cookie instead.
2. How to prevent a user from booking a deleted record or an already booked record by another user running another instance of application simultaneously. In order to prevent this scenario, the contents of the local cached memory are often refreshed to update the stored records before booking a room.

# User Interface (GUI specification)

A user-friendly user interface was created with classes from the Java Swing API.

## Functionalities

1. Positioning components in ways that are user friendly
2. Enabling the user to search for hotel rooms
3. Enabling the user to book for hotel rooms
4. Hiding the back-end implementation of the application from the user
5. Managing application process output

## Implementation

1. To position the text fields, buttons, table and other components on the GUI in a user friendly manner, that is making it intuitive to the user based on the mental model some layout managers were considered including BorderLayout, GridLayout, and GridBagLayout. BorderLayout was chosen over the others because it makes it easy to position components proportionately and it handles window resizes.
2. To search for a hotel room two text fields were provided for the user to specify a String search criteria for either name or location or both at the same time which is intended to match the room they are searching for. The Java Swing component, JTextField was used because it takes String as its default values.
3. In order to book for a hotel room and enter an owner id to correspond with the booked hotel room, two approaches were considered:
4. To use a separate dialog to accept the owner id entered and
5. To input the owner id into the JTable directly through an editable cell.

The first solution was chosen because it enabled the functionality to be put into a separate class other than being merged with the same one that the JTable is in.

1. In order to connect the GUI to the other side of the code (network and non-networked) the Model View Controller Pattern (MVC) was used.The view was separated from the data layer as well as the controller and the user only sees a user-interface but not real code implementation of the application.
2. To tell the user the result of an operation they carried out on the GUI such as searching and booking for a hotel room, that is being successful or unsuccessful, two approaches were considered. They were whether to deal with these results internally for instance when an exception occurs or propagate them and show the user a message. The latter choice use adopted by using a dialog box to inform the user of the result of their operation. This approach was chosen because it makes the system more interactive with the user.
3. In using a JTable to display hotel rooms, two approaches were considered. They were whether to use the default table model of the JTable or to use a custom table model. A custom table model was preferred over a default table model because the custom table model gives more flexibility in the manageability and display of objects in the JTable than the default one.

## Problems encountered

1. How to make sure that the user is only able to specify an eight (8)-digit character and nothing else when entering an owner id during the booking of a hotel room. In solving this, two approaches were considered. Whether to use a JTextField or a JFormattedTextField. The JFormattedTextField in connection with a MaskFormatter was preferred to the JTextField because the JFormattedTextField could be formatted to accept only figures of a certain length, in this sense an eight (8) digit number and nothing else.

# Server (The data server and its network connection)

Since the database file could be stored on another computer which is different from the client computer, two approaches were considered to connect to another computer to be able to access the database file.

The two approaches are Remote Method Invocation (RMI) and Sockets

I chose to use Remote Method Invocation (RMI).

The advantages of choosing RMI

1. It is an abstraction layer over sockets that hides lots of complexities than sockets from the user by creating methods that can be invoked remotely.
2. With RMI, you do not need to write a lot of protocols, that is the client only needs to call the required methods rather than sticking to lots of protocols to send and clarify information.
3. RMI is very secured in terms that you do not have to worry over things like buffer overflow, because you pass in parameters that are used as arguments on the server.

# RMI

A class was implemented which starts the RMI server, looks up the address and binds it to the remote object.

## Functionalities

* RMI Server
* RMI client

## RMI Server

## Implementations

1. RMI Server – a RemoteDatabase interface was created which has only the client methods from DBClient and extends Remote as well.
2. Created a factory interface and implemented it in a subclass
3. Created an instance of the remote object and bound it to the server

Problems encountered

There was the problem of code duplication of DBClient.

How this problem was solved:

A Factory method pattern was used to let the subclass instantiate the appropriate methods that are needed in order to carry out the required functionalities. In this sense, implementation of the DBClient methods was deferred to the subclass.

The subclass has done this by implementing the interface methods of DBClient which tend to be the remote methods.

The server uses a Factory method to create a remote instance of DBClient on which remote methods are called.

## RMI Client

## Implementation

In connecting remotely to the server by the client, the RMI stubs needed to be generated to be used by the remote client to call the remote methods.

Problem encountered

RMI stubs needed to be generated for each client before remote methods could be called.

How it was solved

RMI stubs were not manually generated because as of Java 5 and above the stubs are automatically generated for you at runtime. And Java 6 was used in this projects implementation hence there was no need to re-generate stubs which would have been a redundant job. Moreover it is good to make use of the advancement of the Java language.

## RMI Server Port

It is assumed that the default RMI port (1099) will always be used.

Hostname is localhost. No other port for now.

# OTHER DECISIONS / MISCELLANEOUS

# LOG

## Logger functionalities

A custom logger class was created to log all messages.

## Implementation

Adapter Pattern was used to refine the way java.util.logging is being used to log messages.

Singleton pattern was also used in the Logger implementation. In this sense one instance of the log is created in each package and used for all the logging activities in that package.

I have not turned off the logging and I logged all exceptions.

# 10 CONCLUSION

1. This system can be easily transported unto the internet

2. Again, it serves as a very good learning exercise before the system fully goes onto the web