

HAND OUT DATE: 10th 2022

HAND IN DATE: 10th 2022

WEIGHTAGE:

**Student Name: Cheng Yi Heng**

**Student TP Number: TP058994**

Table of Contents

[Introduction 4](#_Toc99976662)

[1.0 Sample Outputs 5](#_Toc99976663)

[1.1 Login & Account Management System 5](#_Toc99976664)

[1.2 Booking System 8](#_Toc99976665)

[1.3 Booking Management 11](#_Toc99976666)

[2.0 Sample Code 13](#_Toc99976667)

[2.1 Import Files / Header Files 13](#_Toc99976668)

[2.2 Variables 15](#_Toc99976669)

[2.2.1 Static variables 15](#_Toc99976670)

[Private, Public, and Final variables 15](#_Toc99976671)

[2.3 Control Structure 17](#_Toc99976672)

[2.3.1 If – else 17](#_Toc99976673)

[2.4 Looping Structure 18](#_Toc99976674)

[2.4.1 For Loops 18](#_Toc99976675)

[2.5 OOP Concepts 19](#_Toc99976676)

[2.5.1 Class 19](#_Toc99976677)

[2.5.2 Object 20](#_Toc99976678)

[2.5.3 Encapsulation 21](#_Toc99976679)

[2.5.4 Generalization 23](#_Toc99976680)

[2.5.5 Constructor 24](#_Toc99976681)

[2.5.6 Get-Set Method 25](#_Toc99976682)

[2.5.7 Normal Methods 26](#_Toc99976683)

[2.5.8 Exceptional Handling 27](#_Toc99976684)

[2.5.9 File Concept 27](#_Toc99976685)

[3.0 Additional Features 29](#_Toc99976686)

[3.1 Account Management System 29](#_Toc99976687)

[3.2 Room Filtration 31](#_Toc99976688)

[3.3 Integer Filter 34](#_Toc99976689)

[4.0 Assumptions 35](#_Toc99976690)

[5.0 References 36](#_Toc99976691)

# Introduction

The aim of this project is to create a functional booking system for a hotel, namely **The Grand Hotel**. To achieve that, multiple core features have been included in the program:

* Account Management System
* Hotel Room Selection System
* Hotel Room Booking System
* Information Validation System
* Receipt Generation System

These features combine together to build up the final output of the program itself. Alongside with these features are the modern user-friendly user interface. These user interface allows users to interact with the system effortlessly without the need of deep knowledge in information technology and/or how the system works under the hood.

Throughout the usage of the application, data will be saved multiple times during the exit and enter action of several windows. This is done by saving the data in a text file as mentioned in the project requirements. By doing this, users are able to pick up where the left off previously.

# 1.0 Sample Outputs

## 1.1 Login & Account Management System

Graphical user interface, application

Description automatically generated

*Figure 1.1: Login Page*

A screenshot of a computer

Description automatically generated

*Figure 1.2: Accounts Manager Page*

Graphical user interface, application

Description automatically generated

*Figure 1.3: Username & Password validation*

Login system consists of mainly a simple frame for staffs to login into the system to book rooms. The login page will bring users to different page depending on the account entered.

1. Admin  
   When “admin” account is used, the login page will direct the user to the accounts management page.
2. Staff  
   When “staff” account is used, the login page will direct the user to the booking page.

When the “**LOGIN**” button is pressed, the system will validate the username and password entered by the user. If it matches either one of the staff accounts or the admin account, login will be successful. However, when it doesn’t, a message will pop up stating “Invalid password or username”.

In the accounts management page, admins can add, edit, and delete staff accounts via the table shown in Figure 1.2.

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 1.4: saved accounts data*

After editing the accounts, admins will be able to save the data by clicking the “**SAVE**” button. Alternatively, the admin can also click the “**CANCEL**” button to cancel all changes made. Figure 1.4 shows the result of the saved data in *accounts.txt*.

## 1.2 Booking System

Graphical user interface, application

Description automatically generated

*Figure 1.5: Booking Rooms Page*

Staffs can pick and choose a starting and ending date to be booked. The UI will automatically update depending on the dates by removing rooms that are unavailable on that date range. In our case, all rooms are still available from 24th of March to 25th of March.

Graphical user interface

Description automatically generated

*Figure 1.6: Enter Customer Details Page*

We choose to book “Room 1” by clicking on the “**BOOK**” button on the booking rooms page. By doing that, a new page pops up prompting the staffs to enter the customer details.

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 1.7: Invalid detail entered*

A screenshot of a computer

Description automatically generated with medium confidence

*Figure 1.8: Details entered correctly and saved*

Data validation is also performed before a booking successfully occur. As shown in Figure 1.7, a message box will appear if the input format is incorrect or invalid data is entered. Data collected includes: “*Name, IC, Phone Number, and Email*”.

A screenshot of a home screen

Description automatically generated with low confidence

*Figure 1.9: Room 1 no longer available for the time period selected*

As mentioned before, the UI will update according to the availability of each room. In our case, Room 1 was booked during this period, which is why it is not present in Figure 1.9.

Text

Description automatically generated

*Figure 1.10: saved bookings data*

Figure 1.10 shows the data format that is being stored in *bookings.txt* file when a new booking is created.

## 1.3 Booking Management

Graphical user interface

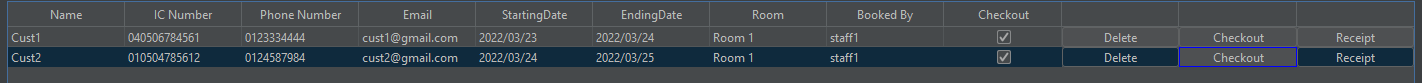
Description automatically generated

*Figure 1.11: Booking Management Page*

The booking management page contains all current booking information. Staffs are able to edit the data here. However, not all data is editable, columns that are not editable includes: “*Starting Date, Ending Date, Room, Booked By, and Checkout*” Columns that are editable includes: “*Name, IC, Phone Number, and Email*”. These choices are in place to prevent staffs from creating conflicts on booking dates when editing the booking data.

Graphical user interface, application

Description automatically generated



*Figure 1.12: Checkout confirmation & Checkout result*

When a customer decides to checkout from the hotel, the staff will be able to do so with the “**Checkout**” button. The staff can also generate receipts based on the booking details as shown in Figure 1.13 below via the “**Receipt**” button. Lastly, the staff can delete a record if a customer choose to cancel their booking via the “**Delete**” button.

Graphical user interface

Description automatically generated

*Figure 1.13: Receipt Page*

Receipt page shows the final booking details along with the final price that the customer needs to pay (including all taxes) for their stay in The Grand Hotel.

# 2.0 Sample Code

## 2.1 Import Files / Header Files

Text

Description automatically generated

*Figure 2.1: Import 1*

In Figure 2.1, *FlatDarkLaf* user interface theme is imported to provide the modern design and dark background feel of the entire application. Next, *JFrame* is imported to access and create new frames in the applications. *UIManager* is also imported as it is used to set the theme of the application to the *FlatDarkLaf* theme.



*Figure 2.2: Import 2*

In Figure 2.2, *LocalDate* is imported to allow the comparison between different dates. For example, to compare if a given date is before or after today’s date. Next, *Date* is imported to support the date picker used in the booking page.



*Figure 2.3: Import 3*

In Figure 2.3, *ArrayList* is imported to create custom lists that supports dynamic length of elements in the array. Next, *HashMap* is imported to support dictionary like data structure whereby it is used to map a string name to an integer index.

Text

Description automatically generated

*Figure 2.4 Import 4*

In Figure 2.4, *File* is imported to support the function of opening different files. Next, *FileWriter* is imported to support the function of writing data to a file. *IOException* is also imported as it will be used to throw exception when a file cannot be read successfully. Finally, *Scanner* is used to read text contents from a file in a line by line basis.

Text

Description automatically generated

*Figure 2.5: Import 5*

In Figure 2.5, multiple *awt* components are imported to support the customization of generated user interfaces. *Color* is used to define a custom color for buttons, *FlowLayout* and *GridLayout* is used to define the layout for a panel while *Font* is used to determine the type of font used for text on labels and buttons. Next, *PropertyChangeEvent* and *PropertyChangeListener* is used for callbacks on buttons when pressed. Finally, several *swing* components are imported to support the generation of custom user interfaces. *JLabel* is used to create custom labels, *JOptionPane* is used to create pop up dialogs for error messages or confirmation messages, *JPanel* is used to create custom panels that include other *swing* components.

## 2.2 Variables

### 2.2.1 Static variables

Text

Description automatically generated

*Figure 2.6: Booking System*

The ***BookingSystem*** class holds all data that are consistent throughout the entire program, namely all *JFrame* forms and *FileManagers* that are responsible for the program.

Because these variables are consistent throughout the entire application, the **static** keyword is used.

### Private, Public, and Final variables

Text

Description automatically generated

*Figure 2.7: Segment of Booking System*

Variables that are only used within the class are marked as **private** whereas variables that are accessed on other classes are marked as **public**. During the development, **private** variables are identified with an underscore in front of the variable while **public** variables use normal camel casing naming convention. Lastly, **final** keyword is used on variables that will not change during runtime. In this example, the spawn location of the forms is fixed.

## 2.3 Control Structure

### 2.3.1 If – else

Text

Description automatically generated

*Figure 2.8: If – else structure*

In Figure 2.8, **if else** statements are used to validate if a given booking detail is valid or not. The first **if** statement make sures that all booking details are field in by checking if all of them are blank. The second **else if** statement checks if the given IC number length fulfill the expected length, which is 12. The third **else if** statement checks if the given phone number length is greater than 7. Lastly, the fourth **else if** statement checks if the given email is valid by making sure that the “@” symbol and the “.com” phrase is within the string.

## 2.4 Looping Structure

### 2.4.1 For Loops

Text

Description automatically generated

*Figure 2.9: Load Data function from Account Manager*

**For loop** is used here to iterate through all the lines stored in *accounts.txt* file. Inside the for loop, an **if else** statement is used to determine if the content length is 2 or 1. Depending on the length of the content, new *Accounts* are added differently to prevent index out of range error.

## 2.5 OOP Concepts

### 2.5.1 Class

#### Abstract Class

Text

Description automatically generated

*Figure 2.10: Abstract File Manager*

***AbstractFileManager*** is an **abstract class** that consists of all the basic level functionality of reading and writing of a file. This class also consists of an **abstract function** called *LoadData ()*. There is no context in this function because it is expected to be overridden in all classes that inherits from it.

#### Class

Text

Description automatically generated

*Figure 2.11: Account*

Figure 2.11 shows the ***Account*** class that stores all important information related to a staff account. For example, staff username and password.

### 2.5.2 Object

Text

Description automatically generated

*Figure 2.12: Objects*

In Figure 2.12, several *Objects* are created using the **new** keyword. The **new** keyword creates an instance of the class by calling the constructor of the class. For example, in the second line, a new ***AccountManager*** instance is created and assigned to a variable called *accountManager*.

### 2.5.3 Encapsulation

Text

Description automatically generated

*Figure 2.13: Room Manager*

#### Private

As shown in Figure 2.13, *\_rooms* and ­*\_nameToIdx* are marked as **private**. This is to prevent these variables from being accessed and modified from outside (e.g., another class) as these variables will only be manipulated within this class.

#### Public

Although all variables are marked as **private**, functions like *LoadData ()*, *NameToIdx ()*, and *getRooms ()* are marked as **public** as it is allowed to be called outside of this class. This way, data can be accessed and/or manipulated by calling these functions.

#### Read-only

If you look closely, *getRooms ()* function returns the *\_rooms* variable directly from the class. This makes the *\_rooms* variable a read only data as code outside this class is not able to write to *\_rooms* which is protected by the **private** keyword.

Similarly, *NameToIdx ()* function also acts as a read only method. However, the difference here is that instead of returning the entire *\_nameToIdx* variable, it only returns one of the value from the *HashMap*.

### 2.5.4 Generalization

Text

Description automatically generated

*Figure 2.14: Booking Manager*

As shown in Figure 17, ***BookingManager*** inherits from the **abstract** class, ***AbstractFileManager*** (shown in Figure 2.10)via the **extends** keyword. The *super ()* function is also used to call the constructor from the base class. This class is able to handle file data in a specific way that works only for *Booking* data, while still able to access generic functions from its base class – ***AbstractFileManager*** like *ReadFile ()*. This class also overrides a function called *LoadData ()*, which is an **abstract function** from the base class.

### 2.5.5 Constructor

Text

Description automatically generated

*Figure 2.15: Room*

As shown in Figure 2.15, a constructor for the ***Room*** class is defined at *public Room (…)*. In this constructor, all non-static and non-read-only variables are initialized with an initial value. As shown in the constructor, although *ImageIcon* was not given, one was created in the constructor and assigned to the *icon* variable.

### 2.5.6 Get-Set Method

Text

Description automatically generated

*Figure 2.16: get-set method*

In Figure 2.16, a get-set method is created for the *\_bookings* variable. This is done to make sure that whenever the *\_bookings* variable is modified, the entire variable is replaced as opposed to modified elements in the *ArrayList*.

### 2.5.7 Normal Methods

Text

Description automatically generated

*Figure 2.17: Add Booking & Save Bookings*

Figure 2.17 shows 2 examples of normal methods *AddBooking (...)* and *SaveBookings ()*. The first method *AddBooking (...)* is used when a new booking is created. The function simply appends the new booking into the *\_bookings* *ArrayList*. On the other hand, *SaveBookings ()* is used to compile up all existing bookings from the *\_bookings ArrayList* into a string data and write them into a text file.

### 2.5.8 Exceptional Handling

Text

Description automatically generated

*Figure 2.18: test function from Int Filter*

This function is used to test if all characters in a *String* is an integer. To do that, we iterate through each character and use a **try catch** statement to try and parse the *String* into an *Integer* and catch any *NumberFormatException*.

### 2.5.9 File Concept

In the ***AbstractFileManager*** abstract class, there is a *ReadFile ()* and *WriteFile ()* function that handles all file related concepts and functionalities (refer to Figure 2.13).

Text

Description automatically generated

*Figure 2.19: Read File*

In this function, all data from a text file is read via an instance of the *Scanner* class (a *Scanner* object). File contents are read in a line by line basis until there are no lines left. The resultant data is stored in an *ArrayList* variable called *\_contents*.

Text

Description automatically generated

*Figure 2.20: Write File*

In this function, string data is written to a given file path (namely the ­*\_filepath* variable). This is achieved by using an instance of the *FileWriter* class (a *FileWriter* object). After writing the data, the *FileWrter* object is closed to make sure the data is saved and return the accessibility of the text file to other processes (e.g., *notepad.exe*).

# 3.0 Additional Features

## 3.1 Account Management System

A screenshot of a computer

Description automatically generated

*Figure 3.1: Account management*

Account management system is an additional feature that I built but is not stated in the Assignment requirements. In terms of Java coding, the account manager allows admins to edit, add, modify, and delete staffs through the *Account Manager* panel.

Text

Description automatically generated

*Figure 3.2: Create new account*

This piece of code allows admins to create new user by adding a new **Account** and adding it into the *\_accounts* list.

Text

Description automatically generated

*Figure 3.3: Delete an account or remove all existing accounts*

The first function shown in Figure 3.3 allows admins to delete an existing staff account from. The second deletes all staff accounts at once. This is only used to clear the array before loading/reading accounts from a text file database.

Text

Description automatically generated

*Figure 3.4: Save accounts to text file database*

Figure 3.4 shows a function that allows admins to save the accounts created/modified into a text file based database. The format of doing that is by storing their username and password in a line by line basis which each element separated by a comma.

Text

Description automatically generated

*Figure 3.5: Load accounts from database*

This function is called to initialize all existing accounts from the text file based database. Similar to how we store it, we read them line by line and extract each line by separating the text with the separator – comma.

## 3.2 Room Filtration

Text

Description automatically generated

*Figure 3.6: Filtering rooms*

Instead of displaying all the rooms all at once, I created a room filtering function which filters out all rooms that has been booked by the selected date. This means that staff only get to see available rooms and not those that are not available. Instead of going through try and error, the staff immediately knows what rooms can be booked and what rooms cannot be booked.

Text

Description automatically generated

*Figure 3.7: Rendering UIs*

At its core, room filtering is done by not rendering rooms that are unavailable. This is done by storing all unavailable room’s indices in an array and skip the rendering of that room when we encounter a room that has its index contained in the unavailable room’s indices array. After doing that, we call the *panel.repaint ()* and *panel.revalidate ()* function to refresh the UIs.

Text

Description automatically generated

*Figure 3.8: Checking for date conflict*

Figure 3.8 shows how the algorithm checks for conflict given 2 start and end date. To simplify it, I will demonstrate it with an example. Let’s say we want to compare if start/end A conflicts with start/end B. To do that, we need to check for 3 things:

1. Is start A in between start B and end B?
2. Is end A in between start B and end B?
3. Is start A before start B and end B after end B?

If either one of those statement above is true, we consider it a conflict, else no conflict occurs.

## 3.3 Integer Filter

Text

Description automatically generated

*Figure 3.9: Int Filter*

Lastly an integer only filter for text field is also introduced into the application. This functionality is surprisingly useful when it comes to text only inputs. Shown in Figure 3.9, the *IntFilter* uses a *Document* to filter out all the text inside the text field. At its core is the *test ()* method. In this method, all characters are checked to make sure that they are a number character instead of a string character.

# 4.0 Assumptions

There are quite a few assumptions to be made for this Hotel Room Booking System:

1. I assume that staffs remember their username and password and is able to login into the system successfully via their accounts.
2. I assume that all admins know that the username and password for the admin account is *admin* and *admin* so that they can edit, add, and modify staff accounts.
3. I assume that all bookings will not be in conflict as there is a filter function going on in the system.
4. I assume that all staffs will be booking for the customer instead of customer booking the rooms themselves.
5. I assume that the application will only be accessible to the admins and staff while not accessible to the customers.
6. I assume that all rooms are priced at RM 10 per night and there is an additional 10% of service tax for all customers.
7. I assume that customers are able to checkout from their hotel room sooner than their booked end date. This will also free the room for other potential customers looking to book a room at that date range.

# 5.0 References

1. Microsoft. (n.d.). Float.IComparable.CompareTo(Object) Method (Java.Lang). Microsoft Docs. Retrieved March 28, 2022, from <https://docs.microsoft.com/en-us/dotnet/api/java.lang.float.java-lang-icomparable-compareto?view=xamarin-android-sdk-12>
2. Singh, C. (2017, October 17). Java 8 - Calculate days between two dates. Beginnersbook.Com. Retrieved March 28, 2022, from <https://beginnersbook.com/2017/10/java-8-calculate-days-between-two-dates/>
3. Java Swing revalidate() vs repaint(). (2009, July 8). Stack Overflow. Retrieved March 28, 2022, from <https://stackoverflow.com/questions/1097366/java-swing-revalidate-vs-repaint>
4. Java Convert Date to String - javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved March 28, 2022, from <https://www.javatpoint.com/java-date-to-string>
5. Java Convert String to Date - javatpoint. (n.d.). Www.Javatpoint.Com. Retrieved March 28, 2022, from <https://www.javatpoint.com/java-string-to-date>
6. R. (2019, July 5). How can we make JTextField accept only numbers in Java? Tutorialspoint. Retrieved March 26, 2022, from <https://www.tutorialspoint.com/how-can-we-make-jtextfield-accept-only-numbers-in-java>
7. Oracle. (n.d.). How to Make Dialogs (The JavaTM Tutorials > Creating a GUI With Swing > Using Swing Components). Retrieved March 26, 2022, from <https://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html>
8. Subramanyam, S. (2017, December 29). Can a constructor be overridden in java? Tutorialspoint. Retrieved March 19, 2022, from <https://www.tutorialspoint.com/Can-a-constructor-be-overridden-in-java#:%7E:text=Constructor%20looks%20like%20method%20but,generates%20a%20compile%20time%20error>
9. GeeksforGeeks. (2021, December 20). Different ways of Reading a text file in Java. Retrieved March 18, 2022, from <https://www.geeksforgeeks.org/different-ways-reading-text-file-java/>
10. Camick, R. (2015, October 21). Table Button Column. Java Tips Weblog. Retrieved March 19, 2022, from https://tips4java.wordpress.com/2009/07/12/table-button-column/