VIETNAMESE – GERMAN UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER SCIENCE DEPARTMENT

PROJECT REPORT

<HOTEL MANAGEMENT>

Module 13: Object Oriented Programming with Java

1. <Nguyễn Tiến Đạt – 13948>
2. <Đỗ Hoàng Long – 15001>

Lecturer: Ngoc H. Tran, Ph.D.

Binh Duong, WS2021

Abbreviation

GUI	Graphical User Interface
OOP	Object-Oriented Programming
URL	Uniform Resource Locator
DB	Database

List of Figures

List of Tables

Table of Contents

I. INTRODUCTION

a) Abstract

- In this project, we investigate the business functions of the hotel management system and provide a desktop application supporting the hotel manager to keep track of their customers, bookings, payments, rooms in the hotel, and employees. We also use Java OOP for designing classes and applying the OOP techniques, as well as designing a GUI using the Java Swing library.
- This project provides the basic functions for the users as follows:
    - Input customer information.
    - Delete customer information.
    - Display customer information.
    - Input bookings.
    - Delete bookings.
    - Display booking details.
    - Input payments.
    - Delete payments.
    - Keep track of payments.
    - Add new rooms when the hotel enlarges.
    - Delete rooms when they are removed.
    - Display the room list and details.
    - Add new employees.
    - Delete employees.
    - Adjust salary of an official employee.
    - Make an intern to become an official employee.
    - Display list of employees.

- This project is a part of the Computer Science curriculum, evaluated and instructed by Ph.D. Ngoc H. Tran, and was carried out for us to collect firsthand experiences in Java applications, as well as the implementation of a database and functional features to link with Java applications.

Keywords: Java OOP, Integrated Database, GUI

b) Requirements and constraints

Requirements:

- The application must provide a UI for the user to interact with.
- The application must allow the user to input and delete a record in `Customer` table.
- The application must allow the user to input and delete a record in `Booking` table.
- The application must allow the user to input and delete a record in `Payment` table.
- The application must allow the user to input and delete a record in `Room` table.
- The application must allow the user to input and delete a record in `Employee` table.
- The application must follow an OOP structure with OOP's properties.

Constraints:

- The project must be done by ourselves and cannot be copied from an existing design or major structure on the Internet.
- The duration of the project is limited to 2.5 months and we must learn a new programming language (Java) to create the application.

II. CLASS ANALYSIS

a) Objects

No.	Object Name	State	Behaviors
1.	CustomerInfo	ID, name, address, phone	Getters and Setters methods
2.	Booking	ID, customerID, roomID, bookingDate, checkInDate, checkOutDate, paymentStatus	
3	Payment	paymentDate, bookingID, customerID, roomID, bookingDate, checkInDate, checkOutDate, description, amount	

4	Room	ID, description, cost	
5	Employee (abstract)	ID, name, address, phone, position	
6	Intern -> Employee	Inherits attributes and methods from parent Employee class. Another attribute is duration, which is of its own.	Inherits Getters and Setters methods from Employee. Getter and Setter methods of its own, and MakeOfficial()
7	Official -> Employee	Inherits attributes and methods from parent Employee class. Another attribute is salary, which is of its own.	Inherits Getters and Setters methods from Employee. Getter and Setter methods of its own.

b) Classes:

1. Communication with Database layer:

No.	Classes	Methods
1	`DBConnection`	`GetConnection(),CloseConnection()`
2	`DBQueryCustomer`	`GetInfoFromDB(), GetIDAL(), CheckInputAdd(), AddObjectToDB(), GetCustomerInfoToDelete(), DeleteInfoFromDB().`
3	`DBQueryBooking`	`GetInfoFromDB(), GetCustomerIDAL(), GetBookingIDAL(), AddObjectToDB(), GetBookingInfoToDelete(), DeleteInfoFromDB(), GetRoomIDAvailableAL()`
4	`DBQueryPayment`	`GetInfoFromDB(), GetNonPaymentBookingIDAL(), GetInfoNonPaymentBooking(), GetAllBookingIDAL(),`

		AddObjectToDB(), GetInfoPaymentToDelete(), DeleteInfoFromDB()
5	DBQueryRoom	GetInfoFromDB(), CheckInputAdd(), AddObjectToDB(), GetRoomIDAL(), GetRoomInfo(), DeleteInfoFromDB()
6	DBQueryEmployee	GetInternInfoFromDB(), GetOfficialInfoFromDB(), GetEmployeeIDAL(), GetOfficialIDAL(), GetInternIDAL(), CheckInputAdd(), AddInternToDB(), AddOfficialToDB(), GetEmployeeType(), GetInfoIntern(), GetInfoOfficial(), DeleteInfoFromDB(), ChangeSalary(), MakeOfficial()

2. *Business logic/Application layer:*

No.	Classes	Methods
1.	ServiceCustomer	GetAL(), GetIDAL(), CheckInputAdd(), AddRecord(), GetCustomerInfoToDelete(), DeleteRecord()
2.	ServiceBooking	GetAL(), GetCustomerIDAL(), GetNewBookingID(), GetBookingIDAL(), GetBookingInfoToDelete(), AddRecord(), DeleteRecord()
3.	ServicePayment	GetAL(), GetNonPaymentBookingIDAL(), GetInfoNonPaymentBooking(), GetAllBookingIDAL(), AddRecord(), GetInfoPaymentToDelete(), DeleteRecord()
4.	ServiceRoom	GetAL(), CheckInputAdd(), AddRecord(), GetRoomIDAL(), GetRoomInfo(), DeleteRecord()
5.	ServiceEmployee	GetInternAL(), GetOfficialAL(), CheckInputAdd(), AddRecordIntern(), AddRecordOfficial(), GetEmployeeIDAL(), GetInfoIntern(), GetInfoOfficial(), GetEmployeeType(), DeleteRecord(), GetOfficialIDAL(), GetInternIDAL(), ChangeSalary(), MakeOfficial()

3. GUI layer:

No.	Classes	Methods
1	GUICustomer	SetModelBooking(), SetModelPayment(), Draw(), AddBehaviorToAddButton(), UpdateAdd(), AddBehaviorToDeleteButton(), UpdateDelete()
2	GUIBooking	SetModelPayment(), GetModelBooking(), Draw(), AddBehaviorToAddButton(), UpdateAdd(), AddBehaviorToDeleteButton(), UpdateDelete()
3	GUIPayment	SetModelBooking(), GetModelPayment(), Draw(), AddBehaviorToAddButton(), UpdateAdd(), AddBehaviorToDeleteButton(), UpdateDelete()
4	GUIRoom	SetModelBooking(), SetModelPayment(), Draw(), AddBehaviorToAddButton(), UpdateAdd(), AddBehaviorToDeleteButton(), UpdateDelete()
5	GUIEmployee	Draw(), AddBehaviorToAddButton(), UpdateAddIntern(), UpdateAddOfficial(), AddBehaviorToDeleteButton(), UpdateDelete(), AddBehaviorToSetSalaryButton(), UpdateSalary(), AddBehaviorToMakeOfficialButton(), UpdateIntern()

4. Data Classes:

No.	Classes	Methods
1	CustomerInfo	GetID(), GetName(), GetAddress(), GetPhone(), SetID(), SetName(), SetAddress(), SetPhone()
2	Booking	GetBookingID(), GetCustomerID(), GetRoomID(), GetCheckInDate(), GetCheckOutDate(), GetPaymentStatus(), SetBookingID(),

		SetCustomerID(), SetRoomID(), SetCheckInDate(), SetCheckOutDate(), SetPaymentStatus()
3	Payment	GetPaymentDate(), GetBookingID(), GetCustomerID(), GetRoomID(), GetBookingDate(), GetCheckOutDate(), GetAmount(), GetDescription(), SetPaymentDate(), SetBookingID(), SetCustomerID(), SetRoomID(), SetBookingDate(), SetCheckOutDate(), SetAmount(), SetDescription()
4	Room	GetID(), GetCost(), GetDescription(), SetID(), SetCost(), SetDescription()
5	Employee (abstract)	GetID(), GetName(), GetAddress(), GetPhone(), GetPosition(), SetID(), SetName(), SetAddress, SetPhone(), SetPosition()
6	Intern -> Employee	GetDuration(), SetDuration(), MakeOfficial()
7	Official -> Employee	GetSalary(), SetSalary()

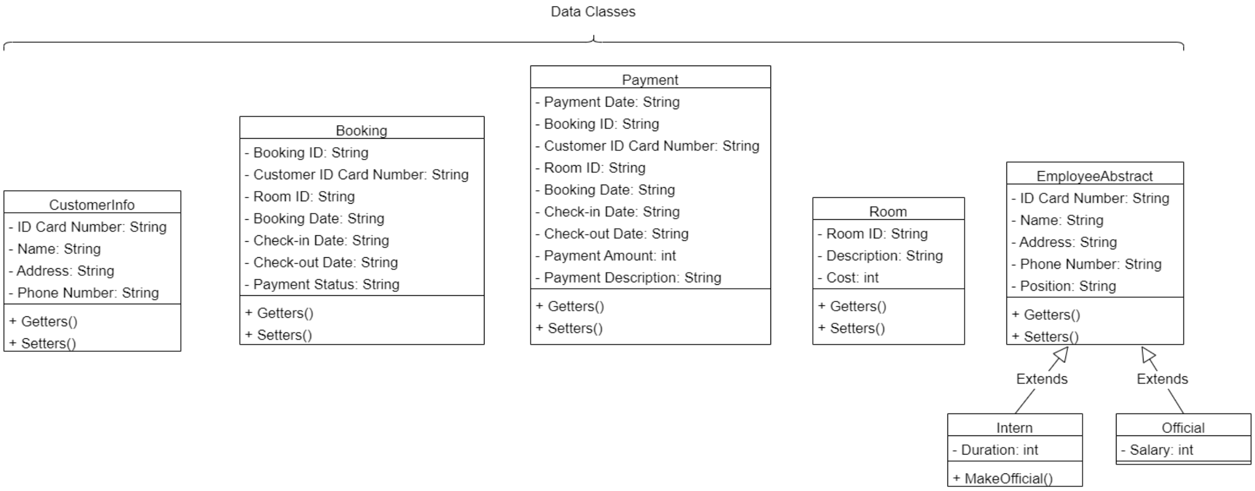


Figure 1. Data classes with inheritance and abstract class

III. CLASS DESIGN

1. Classes

- An overall view of the GUI and Service layers class diagram is shown below (a high-resolution image can be found in the compressed zip file).

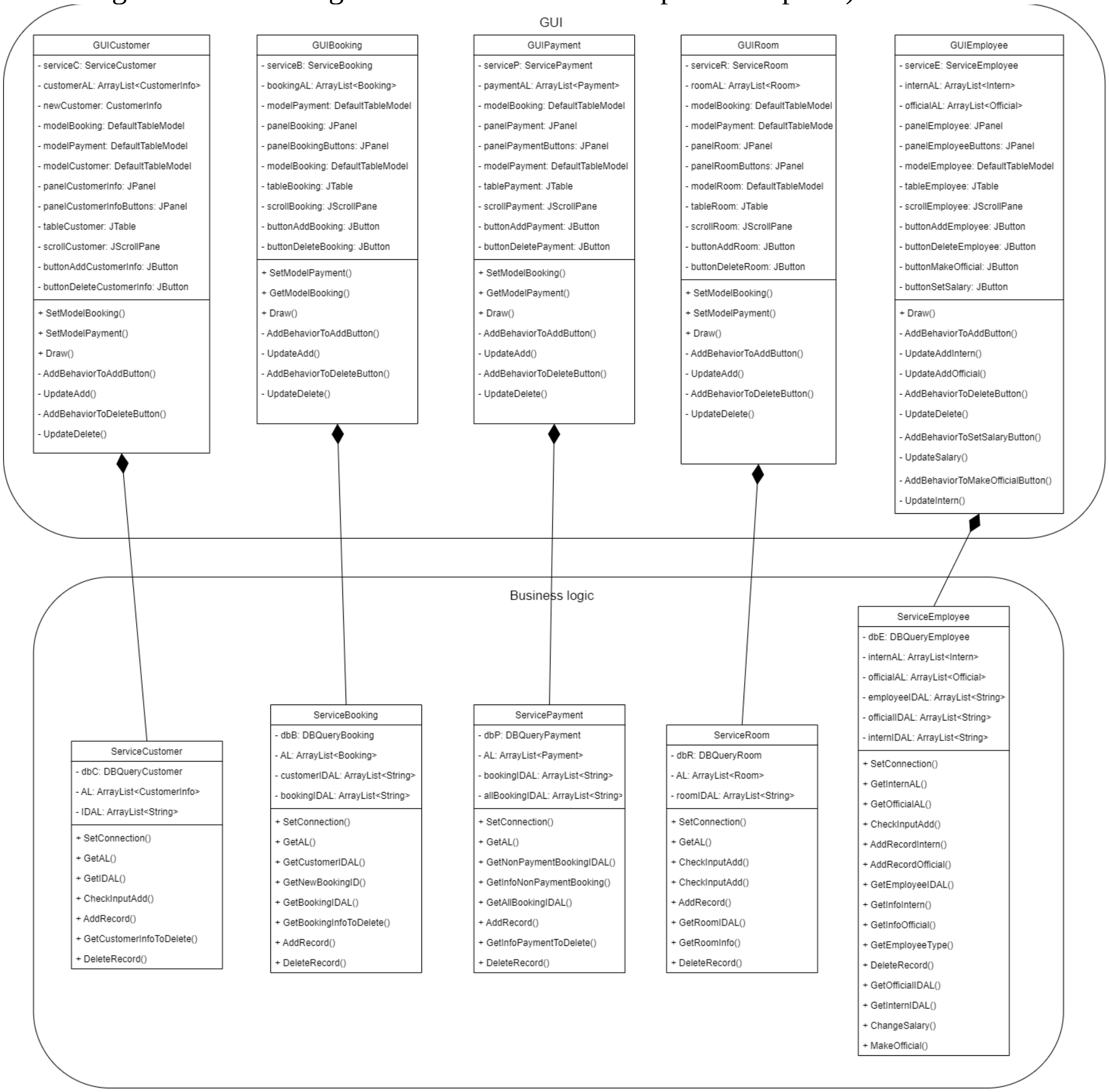

GUI
GUICustomer
- serviceC: ServiceCustomer
- customerAL: ArrayList<CustomerInfo>
- newCustomer: CustomerInfo
- modelBooking: DefaultTableModel
- modelPayment: DefaultTableModel
- modelCustomer: DefaultTableModel
- panelCustomerInfo: JPanel
- panelCustomerInfoButtons: JPanel
- tableCustomer: JTable
- scrollCustomer: JScrollPane
- buttonAddCustomerInfo: JButton
- buttonDeleteCustomerInfo: JButton
+ SetModelBooking()
+ SetModelPayment()
+ Draw()
- AddBehaviorToAddButton()
- UpdateAdd()
- AddBehaviorToDeleteButton()
- UpdateDelete()
GUIBooking
- serviceB: ServiceBooking
- bookingAL: ArrayList<Booking>
- modelPayment: DefaultTableModel
- panelBooking: JPanel
- panelBookingButtons: JPanel
- modelBooking: DefaultTableModel
- tableBooking: JTable
- scrollBooking: JScrollPane
- buttonAddBooking: JButton
- buttonDeleteBooking: JButton
+ SetModelPayment()
+ GetModelBooking()
+ Draw()
- AddBehaviorToAddButton()
- UpdateAdd()
- AddBehaviorToDeleteButton()
- UpdateDelete()
GUIPayment
- serviceP: ServicePayment
- paymentAL: ArrayList<Payment>
- modelBooking: DefaultTableModel
- panelPayment: JPanel
- panelPaymentButtons: JPanel
- modelPayment: DefaultTableModel
- tablePayment: JTable
- scrollPayment: JScrollPane
- buttonAddPayment: JButton
- buttonDeletePayment: JButton
+ SetModelBooking()
+ GetModelPayment()
+ Draw()
- AddBehaviorToAddButton()
- UpdateAdd()
- AddBehaviorToDeleteButton()
- UpdateDelete()
GUIRoom
- serviceR: ServiceRoom
- roomAL: ArrayList<Room>
- modelBooking: DefaultTableModel
- modelPayment: DefaultTableMode
- panelRoom: JPanel
- panelRoomButtons: JPanel
- modelRoom: DefaultTableModel
- tableRoom: JTable
- scrollRoom: JScrollPane
- buttonAddRoom: JButton
- buttonDeleteRoom: JButton
+ SetModelBooking()
+ SetModelPayment()
+ Draw()
- AddBehaviorToAddButton()
- UpdateAdd()
- AddBehaviorToDeleteButton()
- UpdateDelete()
GUIEmployee
- serviceE: ServiceEmployee
- internAL: ArrayList<Intern>
- officialAL: ArrayList<Official>
- panelEmployee: JPanel
- panelEmployeeButtons: JPanel
- modelEmployee: DefaultTableModel
- tableEmployee: JTable
- scrollEmployee: JScrollPane
- buttonAddEmployee: JButton
- buttonDeleteEmployee: JButton
- buttonMakeOfficial: JButton
- buttonSetSalary: JButton
+ Draw()
- AddBehaviorToAddButton()
- UpdateAddIntern()
- UpdateAddOfficial()
- AddBehaviorToDeleteButton()
- UpdateDelete()
- AddBehaviorToSetSalaryButton()
- UpdateSalary()
- AddBehaviorToMakeOfficialButton()
- UpdateIntern()
Business logic
ServiceCustomer
- dbC: DBQueryCustomer
- AL: ArrayList<CustomerInfo>
- IDAL: ArrayList<String>
+ SetConnection()
+ GetAL()
+ GetIDAL()
+ CheckInputAdd()
+ AddRecord()
+ GetCustomerInfoToDelete()
+ DeleteRecord()
ServiceBooking
- dbB: DBQueryBooking
- AL: ArrayList<Booking>
- customerIDAL: ArrayList<String>
- bookingIDAL: ArrayList<String>
+ SetConnection()
+ GetAL()
+ GetCustomerIDAL()
+ GetNewBookingID()
+ GetBookingIDAL()
+ GetBookingInfoToDelete()
+ AddRecord()
+ DeleteRecord()
ServicePayment
- dbP: DBQueryPayment
- AL: ArrayList<Payment>
- bookingIDAL: ArrayList<String>
- allBookingIDAL: ArrayList<String>
+ SetConnection()
+ GetAL()
+ GetNonPaymentBookingIDAL()
+ GetInfoNonPaymentBooking()
+ GetAllBookingIDAL()
+ AddRecord()
+ GetInfoPaymentToDelete()
+ DeleteRecord()
ServiceRoom
- dbR: DBQueryRoom
- AL: ArrayList<Room>
- roomIDAL: ArrayList<String>
+ SetConnection()
+ GetAL()
+ CheckInputAdd()
+ CheckInputAdd()
+ AddRecord()
+ GetRoomIDAL()
+ GetRoomInfo()
+ DeleteRecord()
ServiceEmployee
- dbE: DBQueryEmployee
- internAL: ArrayList<Intern>
- officialAL: ArrayList<Official>
- employeeIDAL: ArrayList<String>
- officialIDAL: ArrayList<String>
- internIDAL: ArrayList<String>
+ SetConnection()
+ GetInternAL()
+ GetOfficialAL()
+ CheckInputAdd()
+ AddRecordIntern()
+ AddRecordOfficial()
+ GetEmployeeIDAL()
+ GetInfoIntern()
+ GetInfoOfficial()
+ GetEmployeeType()
+ DeleteRecord()
+ GetOfficialIDAL()
+ GetInternIDAL()
+ ChangeSalary()
+ MakeOfficial()

Figure 2. Overall class diagram of GUI and Service layers

- An overall view of the Service and Database layers class diagram is shown below (a high-resolution image can be found in the compressed zip file).

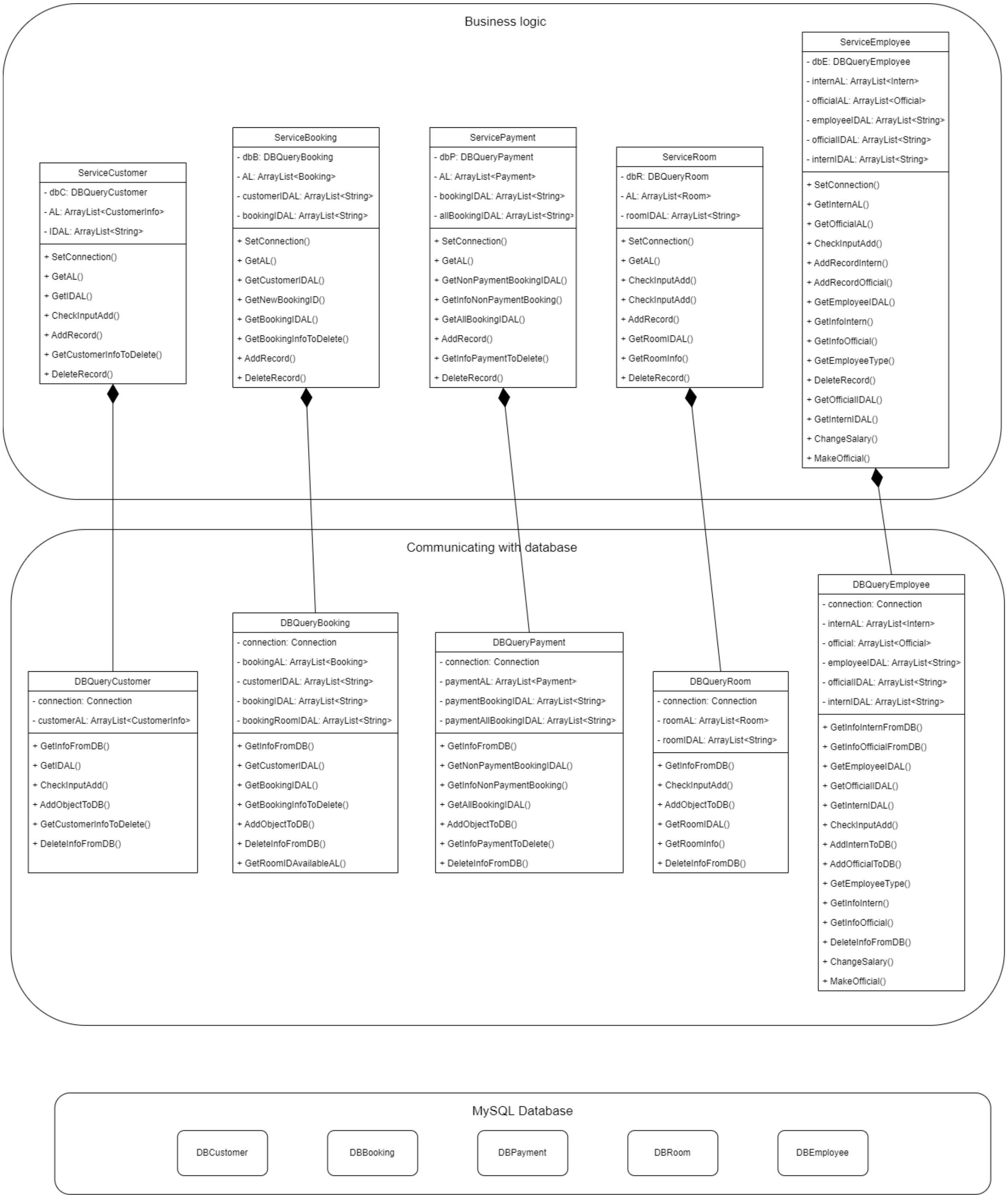


Figure 3. Class diagram of Service and Database layer

- Below is a class diagram containing inheritance and abstract class.

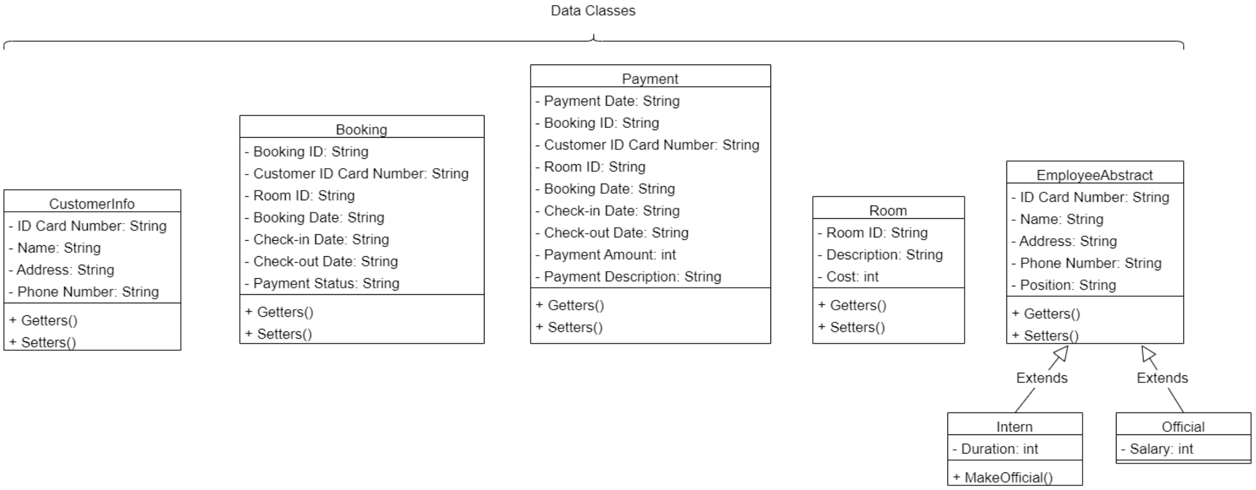


Figure 4. Data classes diagram

Table 1. Details of GUI classes

No.	Classes and description	Instance Variable	Methods and functionalities
1	`GUICustomer`: This class is used for displaying tables of `GUICustomer` class, buttons in the panel, and adding behavior to them. The public methods are for the `main` method of the `Client` class can access. The private methods are for this class to use only.	`private: ServiceCustomer serviceC; ArrayList<CustomerInfo> customerAL; CustomerInfo newCustomer; DefaultTableModel modelBooking, modelPayment, modelCustomer; JPanel panelCustomerInfo, panelCustomerInfoButtons; JTable tableCustomer; JScrollPane scrollCustomer; JButton buttonAddCustomerInfo, buttonDeleteCustomerInfo`	`public void SetModelBooking(DefaultTableModel modelBookingToSet)`: this function is used to set the `modelBooking` of this class. `public void SetModelPayment(DefaultTableModel modelPaymentToSet)`: this function is used to set the `modelPayment` of this class. `public JPanel Draw()`: this function is used to create a `JPanel` object containing a table to display information, and buttons to add and delete records. Return a `JPanel` if the panel is created successfully and `null` if not. `private void AddBehaviorToAddButton()`: this function is used to add behavior to the “Add Customer” button.

			`private void UpdateAdd()`: this function is used to update the table when a new record is added successfully. `private void AddBehaviorToDeleteButton()`: this function is used to add behavior to the “Delete Customer” button. `public void UpdateDelete()`: this function is used to update the table when a record is deleted successfully.

2	`GUIBooking`: this class is used for displaying tables of `GUIBooking` class, buttons in the panel, and adding behavior to them. The public methods are for the `main` method of the `Client` class can access. The private methods are for	`private: ServiceBooking serviceB; ArrayList<Booking> bookingAL; DefaultTableModel modelPayment, modelBooking; JPanel panelBooking, panelBookingButtons; JTable tableBooking; JScrollPane scrollBooking; JButton buttonAddBooking, buttonDeleteBooking`	`public void SetModelPayment(DefaultTableModel modelPaymentToSet)`: this function is used to set the `modelPayment` of this class.
			`public DefaultTableModel GetModelBooking()`: this function is used to get the `modelBooking` of this class.
			`public JPanel Draw();`
			`private void AddBehaviorToAddButton();`
			`private void UpdateAdd();`

	this class to use only.		`private void AddBehaviorToDeleteButton(); private void UpdateDelete()`: the functionality of these methods are the same as that of `GUICustomer`.
3	`GUIPayment`: this class is used for displaying tables of `GUIPayment` class, buttons in the panel, and adding behavior to them. The public methods are for the `main` method of the `Client` class can access. The private methods are for this class to use only.	`private: ServicePayment serviceP; ArrayList<Payment> paymentAL; DefaultTableModel modelBooking, modelPayment; JPanel panelPayment, panelPaymentButtons; JTable tablePayment; JScrollPane scrollPayment; JButton buttonAddPayment, buttonDeletePayment`	`public void SetModelBooking(DefaultTableModel modelBookingToSet)`: this function is used to set the `modelBooking` of this class. `public DefaultTableModel GetModelPayment()`: this function is used to get the `modelPayment` of this class. `public JPanel Draw(); private void AddBehaviorToAddButton(); private void UpdateAdd(); private void AddBehaviorToDeleteButton(); private void UpdateDelete()`: the functionality of these methods are the same as that of `GUICustomer`.

4	`GUIRoom`: This class is used for displaying tables of `GUIRoom` class, buttons in the panel, and adding behavior to them. The public methods are for the `main` method of the `Client` class can access. The private methods are for this class to use only.	`private: ServiceRoom serviceR; ArrayList<Room> roomAL; DefaultTableModel modelBooking, modelPayment, modelRoom; JPanel panelRoom, panelRoomButtons; JTable tableRoom; JScrollPane scrollRoom; JButton buttonAddRoom, buttonDeleteRoom`	`public void SetModelBooking(DefaultTableModel modelBookingToSet); public void SetModelPayment(DefaultTableModel modelPaymentToSet)`: the functionality of these methods are the same as that of `GUIPayment` and `GUIBooking`. `public JPanel Draw(); private void AddBehaviorToAddButton(); private void UpdateAdd(); private void AddBehaviorToDeleteButton(); private void UpdateDelete()`: the functionality of these methods are the same as that of `GUICustomer`.
5	`GUIEmployee`: This class is used for displaying tables of `GUIEmployee` class, buttons in the panel, and adding behavior to them. The public methods are for the	`private: ServiceEmployee serviceE; ArrayList<Intern> internAL; ArrayList<Official> officialAL; JPanel panelEmployee, panelEmployeeButtons; DefaultTableModel`	`public JPanel Draw(); private void AddBehaviorToAddButton(); private void AddBehaviorToDeleteButton(); private void UpdateDelete(String employeeIDToDelete)`: the functionality of these methods

	`main` method of the `Client` class can access. The private methods are for this class to use only.	`modelEmployee; JTable tableEmployee; JScrollPane scrollEmployee; JButton buttonAddEmployee, buttDeleteEmployee, buttonMakeOfficial, buttonSetSalary`	are the same as that of `GUICustomer`. `private UpdateAddIntern(Intern newEmployee); private UpdateAddOfficial(Official newEmployee)`: these functions are used to update the Employee table when a new employee is added successfully. `private void AddBehaviorToSetSalaryButton(); private void AddBehaviorToMakeOfficialButton()`: this function is used to add behavior to “Set Salary” and “Make Official” buttons. `private void UpdateSalary(Official officialToChangeSalary, int amount)`: this function is used to update the salary of an employee in the `Employee` table. `private void UpdateIntern(Intern internToPromote, int salary)`: this function is used to update the `Employee` table

			when an intern is promoted to an official employee.

Table 2. Details of Services classes

No.	Class	Instance Variable	Methods and functionalities
1	`ServiceCustomer`: this class is used to communicate with the `DBQueryCustomer` class at the lower layer.	`private DBQueryCustomer dbC; ArrayList<CustomerInfo> AL; ArrayList<String> IDAL`	`public void SetConnection(Connection connection)`: this function is used to set the connection of the communication channel between the application and the database. `public ArrayList<CustomerInfo> GetAL()`: this function is used to get the array list containing all the `CustomerInfo` objects obtained from the database. `public ArrayList<String> GetIDAL()`: this function is used to get an array list

			containing all the IDs of the customers from the database. `public void CheckInputAdd(Customer newCustomer)`: will take an object of `CustomerInfo` class and check if attributes of that object have already existed or not, then return a boolean value, `true` if the record already existed, `false` if otherwise. `public void AddRecord(CustomerInfo newCustomer)`: after checking the `newCustomer` object, this function will add this object to the database. `GetCustomerInfoToDelete(CustomerInfo customerToDelete)`: will take a `CustomerInfo` object having only an `ID` attribute, then return a `CustomerInfo` object with every other information

			obtained from the database of that object. `DeleteRecord(CustomerInfo customerToDelete)`: will take a `CustomerInfo` object and its information to send to the layer that communicates with the DB to delete.
2	`ServiceBooking`: This class is used to communicate with the `DBQueryBooking` class at the lower layer.	`private DBQueryBooking dbB; ArrayList<Booking> AL; ArrayList<String> customerIDAL; ArrayList<String> bookingIDAL`	`public void SetConnection(Connection connection)`: this function is the same as that of `ServiceCustomer` class. `public ArrayList<Booking> GetAL()`: this function is similar to that of ServiceCustomer class, the difference is the objects contained in the array list.

			`public ArrayList<String> GetCustomerIDAL()`: will return an array list of `String`, containing the `Customer ID` numbers obtained from the database. `public String GetNewBookingID()`: this function is used to generate a new Booking ID and return that value. `public ArrayList<String> GetBookingIDAL()`: will return an array list of `String`, containing the `Booking ID`s obtained from the database. `public Booking GetBookingInfoToDelete(Booking bookingToDelete)`: will take a `Booking` object having only an `ID` attribute, then return a `Booking` object with every other information

			obtained from the database of that object. `public void AddRecord(Booking newBooking); public void DeleteRecord(Booking bookingToDelete)`: the functionality of these methods are similar to that of `ServiceCustomer` class, the difference is the object passed to them.
3	`ServicePayment`: this class is used to communicate with the `DBQueryPayment` class at the lower layer.	`private DBQueryPayment dbP; ArrayList<Payment> AL; ArrayList<String> bookingIDAL; ArrayList<String> allBookingIDAL`	`public void SetConnection(Connection connection)`: this function is the same as that of `ServiceCustomer` class. `public ArrayList<Payment> GetAL()`: this function is similar to that of ServiceCustomer class, the difference is the objects contained in the array list. `public ArrayList<String> GetNonPaymentBookingID`

			`AL()`: will return an array list of type `String`, containing the Booking IDs of the bookings that have not been paid. `public Payment GetInfoNonPaymentBooking(Payment paymentID)`: will take a `Payment` object having only an `ID` attribute, then return a `Payment` object with that `ID` and every other attribute. `GetAllBookingIDAL()`: will return an array list of `String`, containing all `Booking IDs`. `public void AddRecord(Payment newPayment); public void DeleteRecord(Booking paymentToDelete)`: the functionality of these methods are similar to that of `ServiceCustomer` class, the difference is the object passed to them.

			public Payment GetInfoPaymentToDelete (Payment paymentToDelete): this function is similar to that of ServiceCustomer class, the difference is the return and parameter, which is of Payment type.
4	ServiceRoom: This class is used to communicate with the DBQueryRoom class at the lower layer.	private DBQueryRoom dbR; ArrayList<Room> AL; ArrayList<String> roomIDAL	public void SetConnection(Connection connection): this function is the same as that of ServiceCustomer class. public ArrayList<Room> GetAL(): this function is similar to that of ServiceCustomer class, the difference is the objects contained in the array list. public boolean CheckInputAdd(Room roomToAdd): this function is similar to that of ServiceCustomer class, the difference is parameter passed to it.

			`GetRoomIDAL()`: will return an array list of `String`, containing the `Room IDs`. `public Room GetRoomInfo(Room room)`: this function is similar to that of `ServiceCustomer` class, the difference is the return and parameter, which is of `Room` type. `public void AddRecord(Room newRoom); public void DeleteRecord(Room roomToDelete)`: the functionality of these methods are similar to that of `ServiceCustomer` class, the difference is the object passed to them
5	`ServiceEmployee`: this class is used to communicate with the `DBQueryEmployee` class at the lower layer.	`private DBQueryEmployee dbE; ArrayList<Intern> internAL; ArrayList<Official> officialAL; ArrayList<String> employeeIDAL; ArrayList<String> officialIDAL;`	`public void SetConnection(Connection connection)`: this function is the same as that of `ServiceCustomer` class. `public ArrayList<Intern>`

		`ArrayList<String> internIDAL`	`GetInterAL(), public ArrayList<Official> GetOfficialAL()`: this function is similar to method `GetAL()` of `ServiceCustomer` class, the difference is the objects contained in the array list. `public boolean CheckInputAdd(Employee newEmployee)`: this function is similar to that of `ServiceCustomer` class, the difference is parameter passed to it. `public void AddRecordIntern(Intern newIntern)`; `public void AddRecordOfficial(Official newOfficial); public void DeleteRecord(String employeeIDToDelete)`: the functionality of these methods are similar to methods `AddRecord()` and `DeleteRecord()` of ServiceCustomer class, the difference is the object passed to them.

			`public ArrayList<String> GetEmployeeIDAL()`: will return an array list of `String`, containing the employees' IDs and their position. `public Intern GetInfoIntern(Intern intern); public Official GetInfoOfficial(Official official)`: these methods are similar to `GetRoomInfo()`, the difference is the parameters passed to them. `public String GetEmployeeType(String employeeID)`: will take an employee's ID String and see if this employee is an intern or an official employee. Then return a `String` "`intern`" or "`official`" depending on the type. `public ArrayList<String> GetOfficialIDAL();`

			`public ArrayList<String> GetInternIDAL()`: will return an array list of `String`, containing the official employees’ IDs and interns’ IDs (shown in the GUI). `public void ChangeSalary(Official officialIDToChangeSalary, int amount)`: will take an `Official` object and an integer number, then change that Official’s salary to the integer number. `public void MakeOfficial(Intern internToPromote, int salary)`: will take an `Intern` object and an integer number, then convert that intern to an `Official` employee with the specified integer number as its salary.

Table 3. Details of DB classes

No.	Classes and description	Instance Variables	Methods and functionalities

1	`DBConnection`: it is used to establish connection to the database.	`private:`
		`Connection connection;`
		`final String url;`
		`final String user;`
		`final String password`
			`public Connection GetConnection()`: will return a `Connection` object.
			`public void CloseConnection()`: will close the connection to the database.

2	`DBQueryCustomer`: it is used to communicate with the database and send requests to get or delete information related to the `Customer Info` table.	`private Connection connection;`
		`ArrayList<CustomerInfo> customerAL`
			`public ArrayList<CustomerInfo> GetInfoFromDB()`: will return an array list of `CustomerInfo` objects obtained from the database.
			`public ArrayList<String> GetIDAL()`: will return an array list of `String`, containing the `Customer ID` numbers obtained from the database.
			`public boolean CheckInputAdd()`: will take an object of `CustomerInfo` class and check if attributes of that object have already existed or not, then return a boolean value, `true` if the record already existed, `false` if

			otherwise. public void AddObjectToDB(CustomerInfo newCustomer): will take a new CustomerInfo object and add it to the database. public CustomerInfo GetCustomerInfoToDelete(CustomerInfo customerToDelete): will take a CustomerInfo object having only an ID attribute, then return a CustomerInfo object with every other information obtained from the database of that object. public void DeleteInfoFromDB(CustomerInfo customerToDelete): will take a CustomerInfo object and its ID to send delete requests to the database.

3	DBQueryBooking: it is used to communicate with the database and send requests to get or delete information related to the Booking table.	private Connection connection;
		ArrayList<Booking> bookingAL;
		ArrayList<String> customerIDAL;
		ArrayList<String> bookingIDAL;
		ArrayList<String> bookingRoomIDAL
			public ArrayList<Booking> GetInfoFromDB(): will return an array list of Booking objects obtained from the database.

			`public ArrayList<String> GetCustomerIDAL()`: will return an array list of `String`, containing the `Customer ID` numbers obtained from the database. `public ArrayList<String> GetBookingIDAL()`: will return an array list of `String`, containing the `Booking IDs` obtained from the database. `public void AddObjectToDB(Booking newBooking)`: will take a new `Booking` object and add it to the database. `public Booking GetBookingInfoToDelete(Booking bookingToDelete)`: will take a `Booking` object having only an `ID` attribute, then return a `Booking` object with every other information obtained from the database of that object.

			public void DeleteInfoFromDB(Booking bookingToDelete): will take a Booking object and its ID to send delete requests to the database. public ArrayList<String> GetRoomIDAvailableAL(String datetimeCheckInStr, String datetimeCheckOutStr): will take two strings containing information of check-in and check-out date and time, then return an array list of String containing the Room IDs available for the given check-in and check-out date and time.

4	DBQueryPayment: it is used to communicate with the database and send requests to get or delete information related to the Payment table.	private Connection connection;
		ArrayList<Payment> paymentAL;
		ArrayList<String> paymentBookingIDAL;
		ArrayList<String> paymentAllBookingIDAL
			public ArrayList<Payment> GetInfoFromDB(): will return an array list of Payment objects obtained from the database.
			public ArrayList<String> GetNonPaymentBookingIDAL(): will return an array list of type String, containing the Booking IDs of the bookings that have not been paid.

			`public Payment GetInfoNonPaymentBooking(public Payment)`: will take a `Payment` object having only an `ID` attribute, then return a `Payment` object with that `ID` and every other attribute. `public ArrayList<String> GetAllBookingIDAL()`: will return an array list of `String`, containing all `Booking ID`s `public void AddObjectToDB()`: will take a new `Payment` object and add it to the database. `public Payment GetInfoPaymentToDelete(Payment paymentToDelete)`: will take a `Payment` object having only an `ID` attribute, then return a `Payment` object with every other information obtained from the database of that object. `public void DeleteInfoFromDB(Payme`

			`nt paymentToDelete)`: will take a `Payment` object and its `ID` to send delete requests to the database.

5	`DBQueryRoom`: it is used to communicate with the database and send requests to get or delete information related to the `Room` table.	`private Connection connection;`
		`ArrayList<Room> roomAL;`
		`ArrayList<String> roomIDAL`
			`public ArrayList<Room> GetInfoFromDB()`: will return an array list of `Payment` objects obtained from the database.
			`public boolean CheckInputAdd(Room roomToAdd)`: will take an object of `Room` class and check if attributes of that object have already existed or not, then return a boolean value, `true` if the record already existed, `false` if otherwise.
			`public void AddObjectToDB(Room roomToAdd)`: will take a new `Room` object and add it to the database.
			`public ArrayList<String> GetRoomIDAL()`: will return an array list of `String`, containing the `Room IDs`.
			`public Room`

			GetRoomInfo(Room room): will take a Room object having only an ID attribute, then return a Room object with every other information obtained from the database of that object. public void DeleteInfoFromDB(Room roomToDelete): will take a Room object and its ID to send delete requests to the database.

6	DBQueryEmployee: it is used to communicate with the database and send requests to get or delete information related to the Employee table.	private Connection connection;
		ArrayList<Intern> internAL;
		ArrayList<Official> officialAL;
		ArrayList<String> employeeIDAL;
		ArrayList<String> officialIDAL;
		ArrayList<String> internIDAL
			public ArrayList<Intern> GetInfoInternFromDB(): will return an array list of Intern objects obtained from the database.
			public ArrayList<Official> GetInfoOfficialFromDB(): will return an array list of Official objects obtained from the database.
			public ArrayList<String> GetEmployeeIDAL(): will return an array list of String, containing the employees' IDs and their position.

			`public ArrayList<String> GetOfficialIDAL()`: will return an array list of `String`, containing the official employees’ IDs (IDs that are shown in the GUI). `public ArrayList<String> GetInternIDAL()`: will return an array list of `String`, containing the interns’ IDs (IDs that are shown in the GUI). `public boolean CheckInputAdd(Employee newEmployee)`: will take an object of `Employee` class and check if attributes of that object have already existed or not, then return a boolean value, `true` if the record already existed, `false` if otherwise. `public void AddInternToDB(Intern newIntern)`: will take a new `Intern` object and add it to the database. `public void AddOfficialToDB(Offici`

			al newOfficial): will take a new Official object and add it to the database. public String GetEmployeeType(String employeeID): will take an employee's ID String and see if this employee is an intern or an official employee. Then return a String "intern" or "official" depending on the type. public Intern GetInfoIntern(Intern intern): will take an Intern object having only an ID attribute, then return an Intern object with every other information obtained from the database of that object. public Official GetInfoOfficial(Official official): will take an Official object having only an ID attribute, then return an Official object with every other information obtained from the database of that object.

			`public void DeleteInfoFromDB(String employeeIDToDelete)`: will take a String of employee’s `ID` to send delete requests to the database. `public void ChangeSalary(Official officialIDToChangeSalary, int amount)`: will take an `Official` object and an integer number, then change that `Official`’s salary to the integer number. `public void MakeOfficial(Intern internToPromote, int salary)`: will take an `Intern` object and an integer number, then convert that intern to an `Official` employee with the specified integer number as its salary.

Table 4. Details of Data Classes

No.	Class	Instance Variable	Methods and functionalities
1	`CustomerInfo`: this is a data class containing customer information.	`private String ID, name, address, phone`	`public String Get*()`: getter methods to retrieve attributes of CustomerInfo class. `public void Set*(param)`: setter methods to set value for attributes of CustomerInfo class.
2	`Booking`: this is a data class containing booking information.	`private String bookingID, customerID, roomID, bookingDate, checkInDate, checkOutDate, paymentStatus`	`public String Get*()`: getter methods to retrieve attributes of `Booking` class. `public void Set*(param)`: setter methods to set value for attributes of `Booking` class.

3	`Payment`: this is a data class containing payment information.	`private String paymentDate, bookingID, customerID, roomID, bookingDate, checkInDate, checkOutDate, description`
		`private int amount`
			`public String Get*(param)`: getter methods to retrieve `String` attributes of `Payment` class.

			`public int Get*()`: getter method to retrieve `int` attributes of `Payment` class. `public void Set*(param)`: setter methods to set value for attributes of `Payment` class.

4	`Room`: this is a data class containing room information.	`private String ID, description`
		`private int cost`
			`public String Get*()`: getter methods to retrieve attributes of `Room` class.
			`public int Get*()`: getter method to retrieve `int` attributes of `Room` class.
			`public void Set*(param)`: setter methods to set value for attributes of `Room` class.

5	`Intern`: this is a data class containing interns' information.	`private int duration` This class is a subclass of Employee class, so it inherits other attributes and methods of the superclass.	Inherits `Get*()` and `Set*()` methods from the superclass. `public void MakeOfficial()`: this function is used to convert an intern to an official employee.

6	`Official`: this is a data class containing official employee information.	`private int salary` This class is a subclass of Employee class, so it inherits other attributes and methods of the superclass.	Inherits `Get*()` and `Set*()` methods from the superclass.

Table 5. Abstract classes

No.	Abstract Class	Concrete Methods and Description
1	`Employee`: This class is an abstract class that is used by subclasses `Intern` and `Official` (See Figure 2.)	`public String GetID(); public String GetName(); public String GetAddress(); public String GetPhone(); public String GetPosition()`: these are getter methods. `public void SetID(String IDToSet); public String SetName(String nameToSet); public void SetAddress(String addressToSet); public void SetPhone(String phoneToSet); public void SetPosition(String positionToSet)`: these are setter methods.

2. Review OOP techniques:

- **Encapsulation:**
  - The data classes have private attributes and other classes cannot directly read or modify them. The only way for those attributes to be read and/or modified is via public methods, which are called getters and setters.
  - Example code:

```
public class Booking
```

```
{
    private String ID;
    public String GetBookingID()
    {
        return ID;
    }
    public void SetBookingID(String bookingIDToSet)
    {
        this.ID = bookingIDToSet;
    }
}
```

- **Inheritance:**
    - Example code:

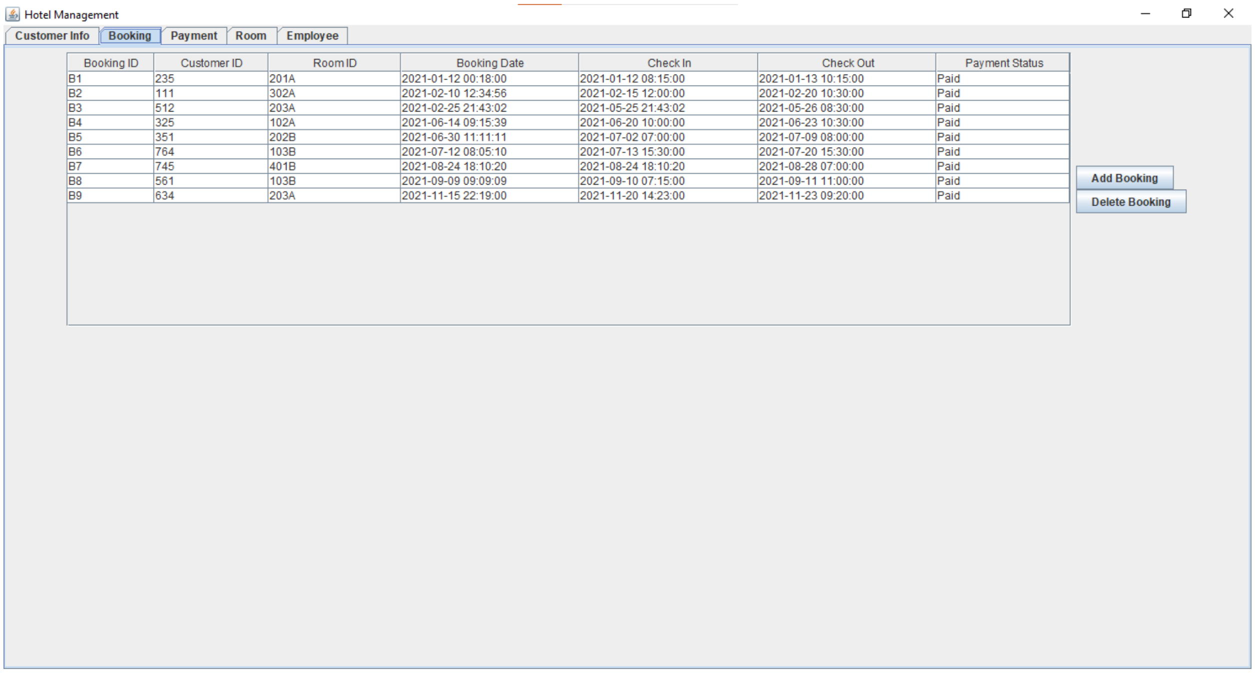
```
public class Intern extends Employee

{

    private int duration;

}
```

IV. Package Design

- The package design follows the three-tier architecture as shown in Figure 2 and 3:
    - A package containing GUI classes.
    - A package containing Application classes.
    - A package containing classes that communicate with the database.
- A package to contain data classes.
- A package containing classes that use a library for date and time chooser.

V. Interface Design

- The application will have five tabs containing the tables from the database.



Booking ID	Customer ID	Room ID	Booking Date	Check In	Check Out	Payment Status
B1	235	201A	2021-01-12 00:18:00	2021-01-12 08:15:00	2021-01-13 10:15:00	Paid
B2	111	302A	2021-02-10 12:34:56	2021-02-15 12:00:00	2021-02-20 10:30:00	Paid
B3	512	203A	2021-02-25 21:43:02	2021-05-25 21:43:02	2021-05-26 08:30:00	Paid
B4	325	102A	2021-06-14 09:15:39	2021-06-20 10:00:00	2021-06-23 10:30:00	Paid
B5	351	202B	2021-06-30 11:11:11	2021-07-02 07:00:00	2021-07-09 08:00:00	Paid
B6	764	103B	2021-07-12 08:05:10	2021-07-13 15:30:00	2021-07-20 15:30:00	Paid
B7	745	401B	2021-08-24 18:10:20	2021-08-24 18:10:20	2021-08-28 07:00:00	Paid
B8	561	103B	2021-09-09 09:09:09	2021-09-10 07:15:00	2021-09-11 11:00:00	Paid
B9	634	203A	2021-11-15 22:19:00	2021-11-20 14:23:00	2021-11-23 09:20:00	Paid

Figure 5. User Interface of the application

- The design is similar for every tab, the only difference is the content of the tab, I.E. name of the tab, table contents, and name of buttons.
- The Employee tab has two more buttons: “Make Official” and “Set Salary”.

VI. Access Control

- This table will demonstrate the type for each instance variable in this project.

Table 6. Access Control for variables

No.	Classes	Instance Variable	Type

1	`GUICustomer`	`ServiceCustomer serviceC;`
		`ArrayList<CustomerInfo> customerAL;`
		`CustomerInfo newCustomer;`
			Private (other classes should not be able

		DefaultTableModel modelBooking, modelPayment, modelCustomer;
		JPanel panelCustomerInfo, panelCustomerInfoButtons;
		JTable tableCustomer;
		JScrollPane scrollCustomer;
		JButton buttonAddCustomerInfo, buttonDeleteCustomerInfo
			to directly access or modify these variables.

2	GUIBooking	ServiceBooking serviceB;
		ArrayList<Booking> bookingAL;
		DefaultTableModel modelPayment, modelBooking; JPanel panelBooking, panelBookingButtons;
		JTable tableBooking;
		JScrollPane scrollBooking;
		JButton buttonAddBooking, buttonDeleteBooking
			Private (other classes should not be able to directly access or modify these variables)

3	GUIPayment	ServicePayment serviceP;
		ArrayList<Payment> paymentAL;
		DefaultTableModel modelBooking, modelPayment;
		JPanel panelPayment, panelPaymentButtons;
		JTable tablePayment;
		JScrollPane scrollPayment;
		JButton buttonAddPayment, buttonDeletePayment
			Private (other classes should not be able to directly access or modify these variables)

4	GUIRoom	ServiceRoom serviceR;
		ArrayList<Room> roomAL;
		DefaultTableModel modelBooking,
			Private (other classes should not be able to directly access or

		modelPayment, modelRoom;
		JPanel panelRoom, panelRoomButtons;
		JTable tableRoom;
		JScrollPane scrollRoom;
		JButton buttonAddRoom, buttonDeleteRoom
			modify these variables)

5	GUIEmployee	ServiceEmployee serviceE;
		ArrayList<Intern> internAL;
		ArrayList<Official> officialAL;
		JPanel panelEmployee, panelEmployeeButtons;
		DefaultTableModel modelEmployee;
		JTable tableEmployee;
		JScrollPane scrollEmployee;
		JButton buttonAddEmployee, buttDeleteEmployee, buttonMakeOfficial, buttonSetSalary
			Private (other classes should not be able to directly access or modify these variables)

6	ServiceCustomer	DBQueryCustomer dbC; ArrayList<CustomerInfo> AL; ArrayList<String> IDAL	Private (other classes should not be able to directly access or modify these variables)

7	ServiceBooking	DBQueryBooking dbB;
		ArrayList<Booking> AL;
		ArrayList<String> customerIDAL;
		ArrayList<String> bookingIDAL
			Private (other classes should not be able to directly access or modify these variables)

8	ServicePayment	DBQueryPayment dbP;
		ArrayList<Payment> AL;
			Private (other classes

		ArrayList<String> bookingIDAL; ArrayList<String> allBookingIDAL	should not be able to directly access or modify these variables)

9	ServiceRoom	DBQueryRoom dbR;
		ArrayList<Room> AL;
		ArrayList<String> roomIDAL
			Private (other classes should not be able to directly access or modify these variables)

10	ServiceEmployee	DBQueryEmployee dbE;
		ArrayList<Intern> internAL;
		ArrayList<Official> officialAL;
		ArrayList<String> employeeIDAL;
		ArrayList<String> officialIDAL;
		ArrayList<String> internIDAL
			Private (other classes should not be able to directly access or modify these variables)

11	DBConnection	Connection connection;
		final String url;
		final String user;
		final String password
			Private (other classes should not be able to directly access or modify these variables; final variables are used to store URL and credentials to access the database)

12	DBQueryCustomer	Connection connection; ArrayList<CustomerInfo> customerAL	Private (other classes should not be able

			to directly access or modify these variables)

13	DBQueryBooking	Connection connection;
		ArrayList<Booking> bookingAL;
		ArrayList<String> customerIDAL;
		ArrayList<String> bookingIDAL;
		ArrayList<String> bookingRoomIDAL
			Private
			(other classes should not be able to directly access or modify these variables)

14	DBQueryPayment	Connection connection;
		ArrayList<Payment> paymentAL;
		ArrayList<String> paymentBookingIDAL;
		ArrayList<String> paymentAllBookingIDAL
			Private
			(other classes should not be able to directly access or modify these variables)

15	DBQueryRoom	Connection connection;
		ArrayList<Room> roomAL;
		ArrayList<String> roomIDAL
			Private
			(other classes should not be able to directly access or modify these variables)

16	DBQueryEmployee	Connection connection;
		ArrayList<Intern> internAL;
		ArrayList<Official> officialAL;
		ArrayList<String> employeeIDAL;
		ArrayList<String> officialIDAL;
		ArrayList<String> internIDAL
			Private
			(other classes should not be able to directly access or modify these variables)

17	CustomerInfo	String ID, name, address,	Private

		phone	(for the customer's privacy)
18	Booking	String bookingID, customerID, roomID, bookingDate, checkInDate, checkOutDate, paymentStatus	Private (other classes should not be able to directly access or modify these variables)
19	Payment	String paymentDate, bookingID, customerID, roomID, bookingDate, checkInDate, checkOutDate, description int amount	Private (other classes should not be able to directly access or modify these variables)
20	Room	String ID, description int cost	Private (other classes should not be able to directly access or modify these variables, actually it should be public but it has been shown by public function GUI)
21	Intern	int duration	Private (other classes should not be able to directly access or modify these variables)
22	Official	int salary	Private (other classes

			should not be able to directly access or modify these variables)

23	`Employee`	`void SetID(String IDToSet);`
		`String SetName(String nameToSet);`
		`void SetAddress(String addressToSet);`
		`void SetPhone(String phoneToSet);`
		`void SetPosition(String positionToSet)`
		
		`String GetID();`
		`String GetName();`
		`String GetAddress();`
		`String GetPhone();`
		`String GetPosition()`
			Public
			(these methods are public because it is in registration form)

VII. OOP Techniques

1. Encapsulation:

- The GUI classes have private attributes which are hidden and cannot be read or modified by other classes.
  - Example code:

```
public class GUICustomer
{
        private JPanel panelCustomerInfo = new JPanel();
        private JPanel panelCustomerInfoButtons = new JPanel();
}
```

  - In the code shown above, the attributes’ access modifier of GUICustomer class is `private`, which prevents other classes from accessing them.

- The data classes also have private attributes and can only be accessed and/or modified with the use of the given public methods.
    - Example code:

```
public class CustomerInfo
{
        private String name;
        public String GetName()
        {
                return name;
        }
        public void SetName(String nameToSet)
        {
                this.name = nameToSet;
        }
}
```

    - In the code shown above, attribute `name` can only be read when using the public method `GetName()`, and modified using method `SetName()`.

2. Inheritance:

- This OOP property is achieved through the use of an abstract class called Employee. The subclasses of `Employee` are `Intern` and `Official`.

```
        public class Intern extends Employee

        {

                private int duration;

        }
```

- The class `Intern` inherits attributes and methods from the parent class `Employee`, and it has its own attribute called “`duration`”.

3. Polymorphism:

- This OOP property is achieved with the use of constructor overloading.
- Example code:

```
public Room()
```

```
{
}
public Room(String cstrID, String cstrDescription, int cstrCost)
{
	this.ID = cstrID;
	this.description = cstrDescription;
	this.cost = cstrCost;
}
```

- With these constructors, an object can be created either with or without parameters.

VIII. Experiment

1. Environment and Tools

a. Environment:

- One laptop, one keyboard, one mouse.
- CPU: Intel Core i7 - 5600U @ 2.60 GHz (4 CPUs).
- RAM: < 8GB.

b. Tools:

- Eclipse IDE - Version: 2019-06 (4.12.0)
- MySQL Workbench - Version: 8.0.23
- Java Swing Library
- JDBC Library
- LGoodDatePicker Library

2. Project functions

- The application can display data retrieved from MySQL database, data includes information about: Customer, Booking, Payment details, Room, and Employee
- For the **Customer** table, users can add a new customer. A customer has an ID number, name, address, and phone number, which are the fields to fill.
    - When adding a customer, none of the fields can be null.
    - ID number and phone number must be unique among the customers.
    - Users can also delete an existing customer, this will also delete every booking and payment made by this customer.
- For the **Booking** table, users can add a new booking. A booking can only be created with existing customer ID, check-in and check-out time, and room ID.
    - Check-in time must be at least equal or later than the time of booking creation.
    - Check-out time must be one day later than check-in time.
    - None of the fields can be null.
    - Users can also delete an existing booking, this will also delete the payment that is related to this booking (if exists).
- For the **Payment** table, users can add a new payment. A payment can only be created with existing bookings that have not been paid.
    - Payment amount must be a positive integer.
    - Payment description is optional.
    - None of the fields can be null.
    - Users can also delete an existing payment, this will also delete the booking that is related to this payment.
- For the **Room** table, users can add a new room. A room has an ID, a description, and cost of a night spent in that room.
    - Last character of ID of a room is either “A” or “B”, which indicates that room either has one bed, or two beds, respectively. Room IDs must be unique among the rooms.
    - A description of maximum 100 characters to describe the room is mandatory.
    - Cost of a room must be a positive integer.

  - None of the fields can be null.
  - Users can also delete an existing room, which will also delete every booking and payment related to this room.
- For the **Employee** table, users can add a new employee. An employee has ID number, name, address, phone number, position. And depending on either that employee is an intern or an official employee, he/she can either have `duration` or `salary` attributes, respectively.
  - ID and phone number among the employees must be unique.
  - When adding a new employee, users must either choose that employee to be an intern or an official employee. Then fill in the intern duration (in months) or salary.
  - Users can also delete an employee.

3. Database (4 tables)

- The application uses MySQL as the database.
- The database schema is shown below.

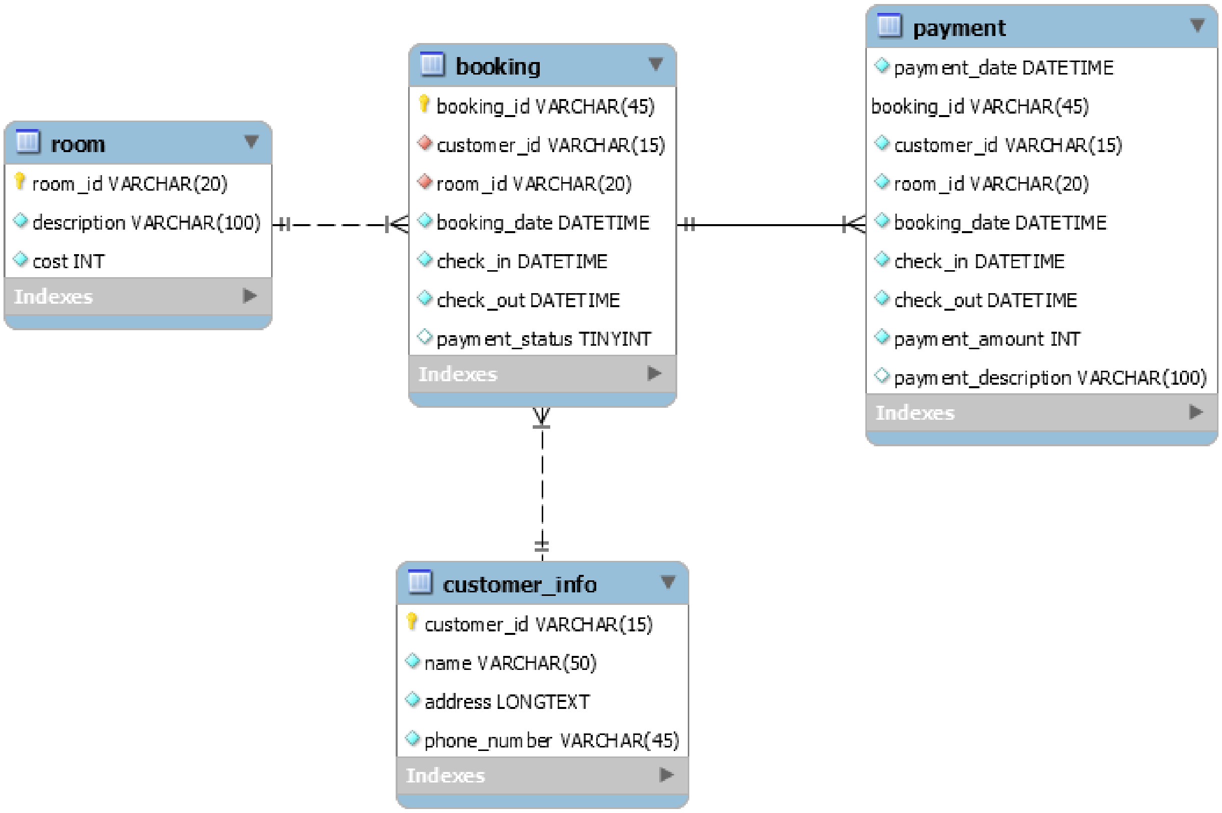


Figure 6. Schema for storing information related to customer and booking information

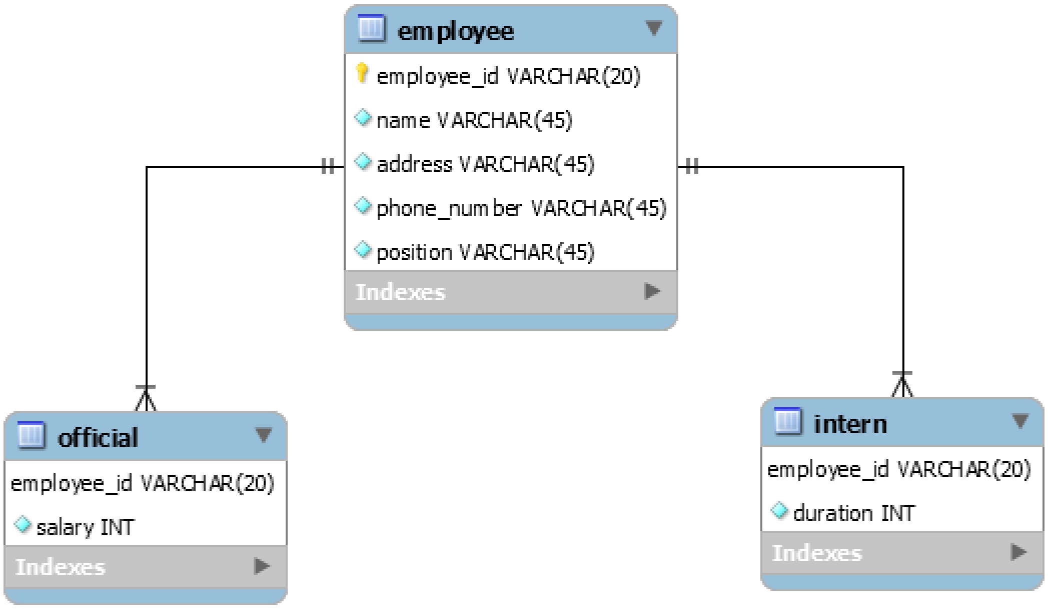
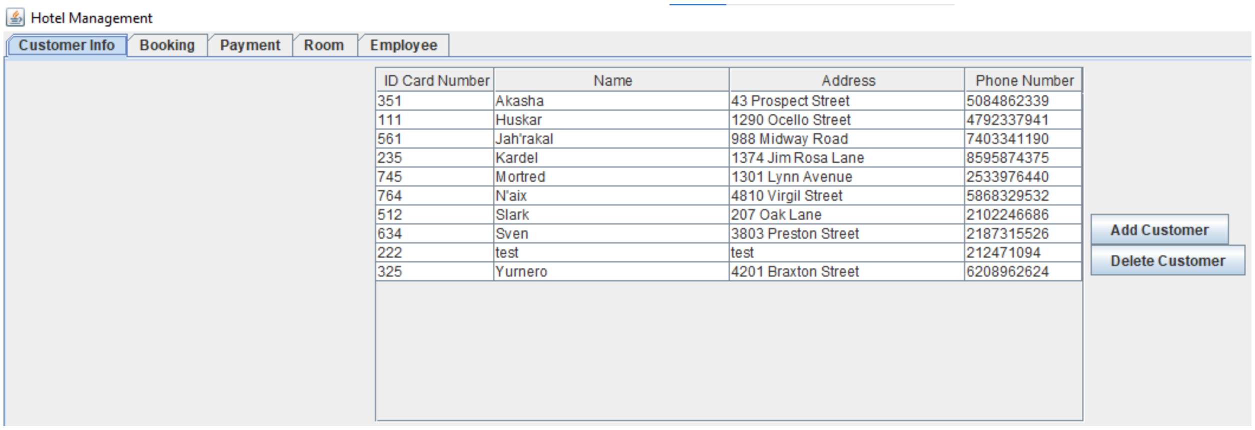


Figure 7. Schema for storing employee information

4. GUI (4 figures)



ID Card Number	Name	Address	Phone Number
351	Akasha	43 Prospect Street	5084862339
111	Huskar	1290 Ocello Street	4792337941
561	Jah'rakal	988 Midway Road	7403341190
235	Kardel	1374 Jim Rosa Lane	8595874375
745	Mortred	1301 Lynn Avenue	2533976440
764	N'aix	4810 Virgil Street	5868329532
512	Slark	207 Oak Lane	2102246686
634	Sven	3803 Preston Street	2187315526
222	test	test	212471094
325	Yurnero	4201 Braxton Street	6208962624

Figure 8. Interface of Customer Info tab

- The design is similar for every tab, the only difference is the content of the tab, i.e. name of the tab, table contents, and name of buttons.
- The Employee tab has two more buttons: “Make Official” and “Set Salary”.

- When a user clicks on the "Add Customer" button, a dialog for inputting purposes appears (figure 7).

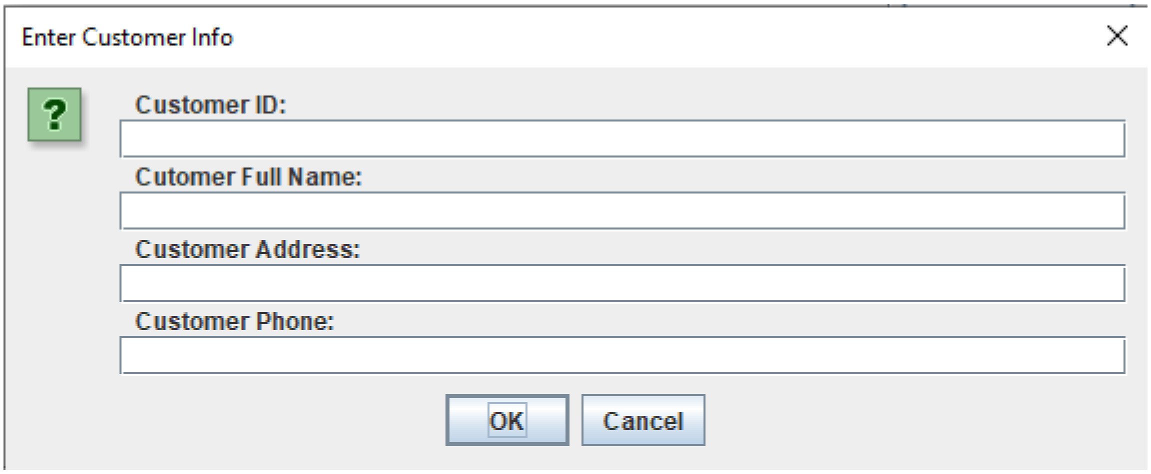
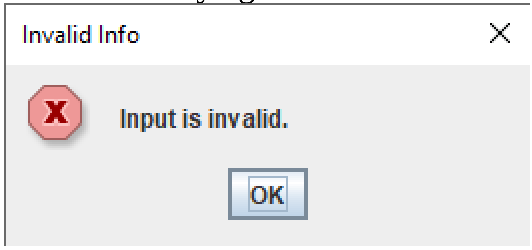


Figure 9. Dialog for inputting new customer information

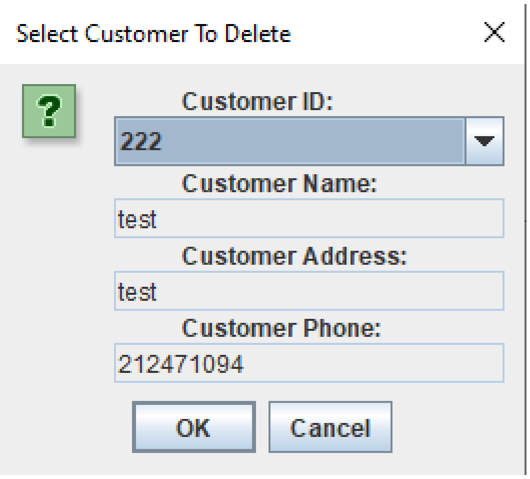
- After filling every field, if the user presses the "OK" button, the fields will be combined into a CustomerInfo object and sent to the layer that can directly communicate with the database.
- Then, a new row containing information of the new customer will appear on the table Customer.
- This can be done with the use of the two methods in the `GUICustomer` class: `AddRecord()` and `UpdateAdd()`.
- After the new record has been added successfully, a dialog will appear, letting the user know that the process has been completed.
- If the fields were filled in incorrectly (ID or phone number contain non-numeric characters, ID or phone number are not unique) or any field left blank, then the user presses the "OK" button in the input dialog, an error dialog (figure 8) appears, letting the user know that he/she must try again with different input values.



Input is invalid.

Figure 10. Error dialog when input is left blank or contains invalid characters

- The users can also delete a customer if they press the “Delete Customer” button. A dialog for the users to choose which customer to choose to delete will appear (figure 9).



Select Customer To Delete

Customer ID: 222

Customer Name: test

Customer Address: test

Customer Phone: 212471094

Figure 11. Dialog for deleting a customer

- When the user presses the “OK” button, an object of type CustomerInfo is created with only the ID, and then sent to the layer that directly communicates with the database to send a delete request and remove that record from the database.
- The row containing the deleted record will also be removed from the Customer table.
- This can be done with the use of `DeleteRecord()` and `UpdateDelete()` methods in the `GUICustomer` class.
- After that, a dialog letting the user know that the process has been completed will appear.
- In any case, if the user presses the “Cancel” or “X” (Close) button, the inputting dialog will be closed without performing any other actions.

IX. Conclusion

- Advantages of the project:
  - It helps us to learn Java with hands-on experience.
  - It teaches us to work as a small team.
  - It helps us know what steps we should take when developing an application from scratch.
  - Although it is a small project, we can still put it in our CVs as a proper project and experience.

  - It helps us to realize what our capabilities are because there are other functionalities we would want to develop but we are limited in terms of time and experience.
- Disadvantages of the project:
  - Other than being physically and mentally drained sometimes, it was a great experience for us to know a bit about how a project should be done industrially.
- Personally, I believe our project can be done with higher quality if we had more time. In terms of functionality, the application does not allow the users to edit existing records, if they want to do that, they would have to delete that record and create a new one with the edited fields of their preference. In terms of OOP when developing the project, because we did not plan it properly, hence, the OOP techniques used in the project were not many. So, we believe a score of 3.0 to 4.0 fits our work, after all, this is a Java OOP course and we did not use many OOP techniques.
- If we had more time and planned the design properly, we believe that we would be able to add a feature to let the users edit an existing record and also produce better code to optimize OOP properties that Java gives us.

DUTY ROSTER

ID	Task	In Charge	Start	End	State	Note
1	MySQL Database	Nguyen Tien Dat	24-Nov-21	11-Dec-21	Done	
2	Design Class Diagram	Nguyen Tien Dat, Do Hoang Long	15-Dec-21	19-Jan -22	Done	
3	Design Interface Classes	Do Hoang Long	15-Dec-21	19-Jan -22	Done	
4	Design Service Classes	Nguyen Tien Dat, Do Hoang Long	16-Dec-21	19-Jan -22	Done	
5	Design Database Classes	Nguyen Tien Dat	15-Dec -21	19-Jan -22	Done	

6	Report Section I	Do Hoang Long	19-Dec-21	22-Dec-21	Done	Request 1
7	Report Section II	Nguyen Tien Dat	20-Dec-21	22-Dec-21	Done	Request 1
8	Report Section III	Nguyen Tien Dat	1-Jan - 22	3-Jan - 22	Done	Request 2
9	Report Section IV	Do Hoang Long	4-Jan - 22	7-Jan - 22	Done	Request 2
10	Report Section V	Nguyen Tien Dat	10-Jan -22	18-Jan -22	Done	Request 3
11	Report Section VI	Do Hoang Long	10-Jan -22	19-Jan -22	Done	Request 3
12	Report Section VII	Nguyen Tien Dat	10-Jan -22	18-Jan -22	Done	Request 3
13	Report Section VIII	Nguyen Tien Dat	10-Jan -22	18-Jan -22	Done	Request 3

14	Report Section IX	Do Hoang Long	10-Jan-22	19-Jan-22	Done	Request 3

REFERENCES

1. Java Swing Tutorial: https://www.javatpoint.com/java-swing, https://docs.oracle.com/javase/7/docs/api/javax/swing/

2. Establish connection with Database Tutorial: https://www.knowprogram.com/jdbc/connect-mysql-database-eclipse/

3. LGoodDatePicker Library and Tutorial: https://github.com/LGoodDatePicker/LGoodDatePicker

APPENDIX A: GRAPHICAL USER INTERFACE (GUI)

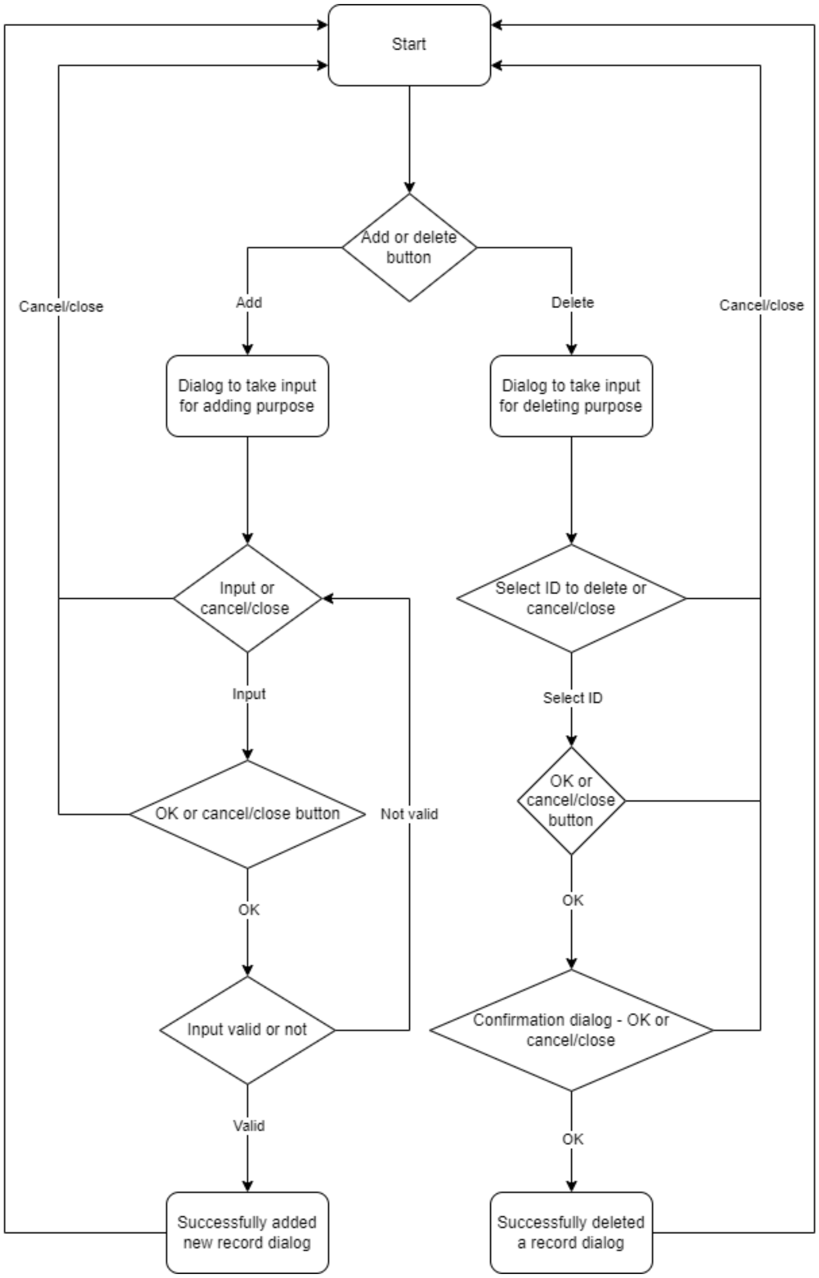

Start
Add or delete button
Add
Delete
Cancel/close
Cancel/close
Dialog to take input for adding purpose
Dialog to take input for deleting purpose
Input or cancel/close
Select ID to delete or cancel/close
Input
Select ID
OK or cancel/close button
Not valid
OK or cancel/close button
OK
OK
Input valid or not
Confirmation dialog - OK or cancel/close
Valid
OK
Successfully added new record dialog
Successfully deleted a record dialog

Figure 12. UI flow diagram

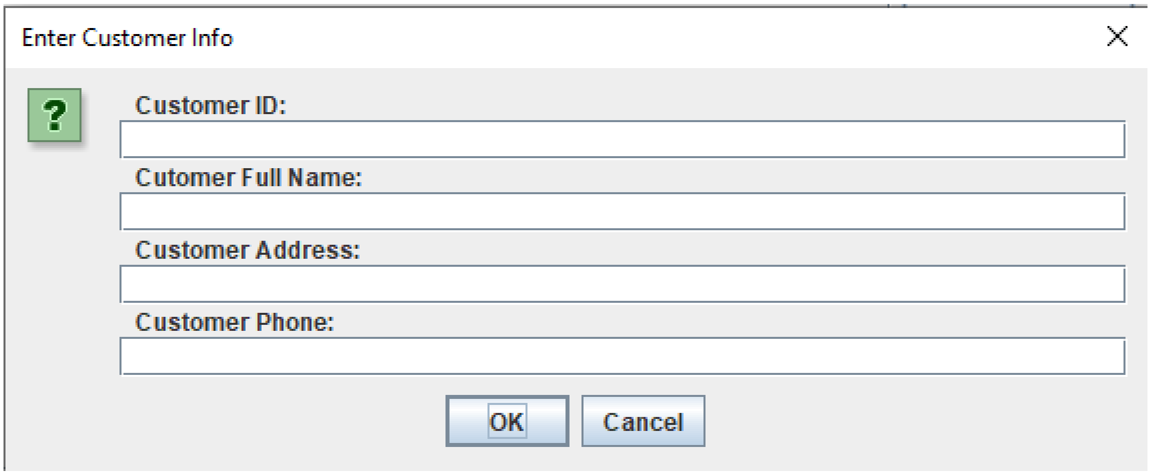
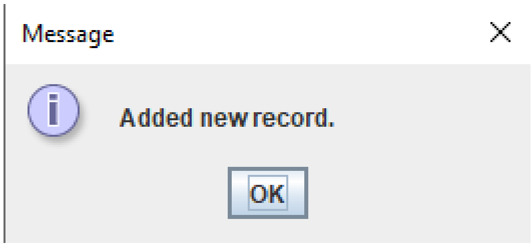
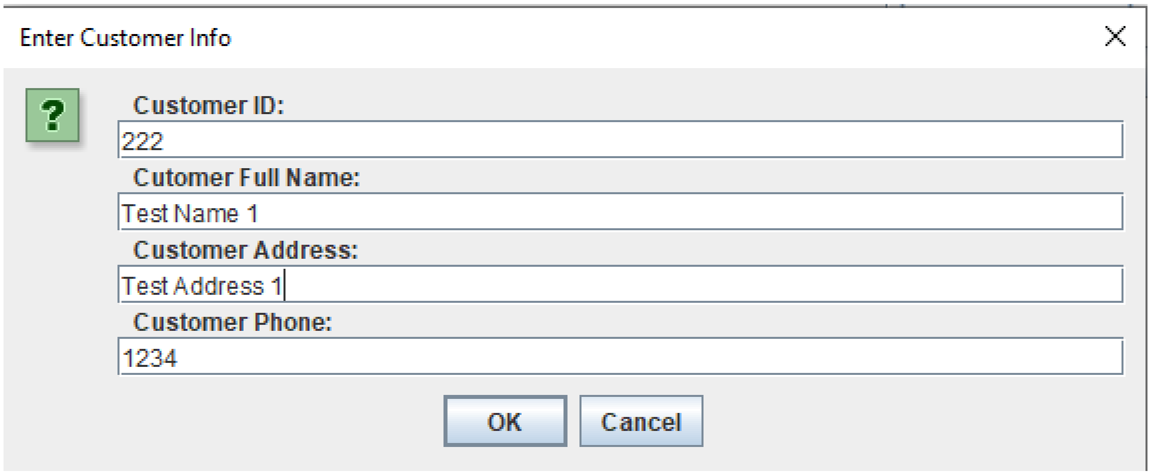


Figure 13. Dialog for inputting new customer information



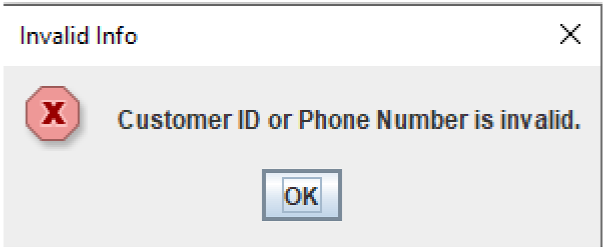
Added new record.

Figure 14. Dialog when adding a record successfully



Customer ID: 222
Cutomer Full Name: Test Name 1
Customer Address: Test Address 1
Customer Phone: 1234

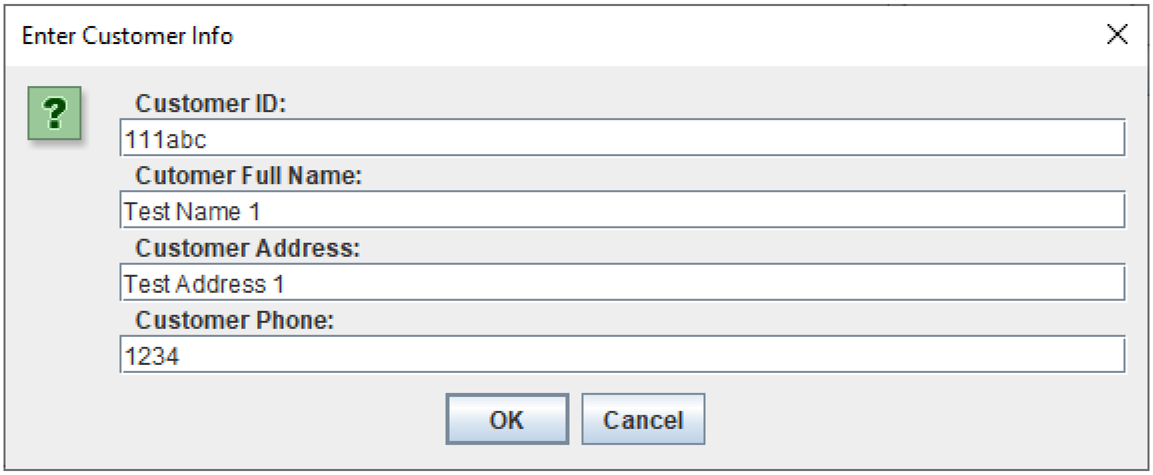
Figure 15. Adding a duplicate customer ID



Invalid Info

Customer ID or Phone Number is invalid.

Figure 16. Error dialog when adding a duplicate record



Enter Customer Info

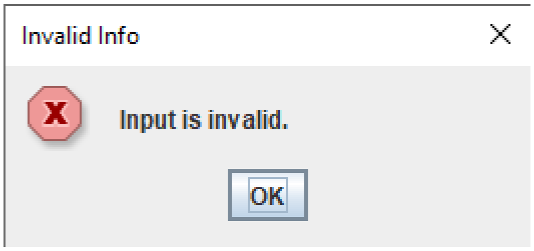
Customer ID:
111abc

Cutomer Full Name:
Test Name 1

Customer Address:
Test Address 1

Customer Phone:
1234

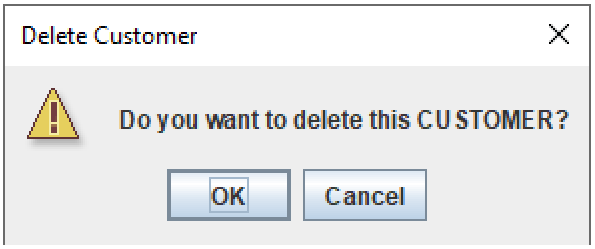
Figure 17. Customer ID is invalid because it contains non-numeric characters



Invalid Info

Input is invalid.

Figure 18. Error dialog when input is invalid



Delete Customer

Do you want to delete this CUSTOMER?

Figure 19. Confirmation dialog when deleting a record

- Input dialog for the tables is similar to each other, the only difference is the input information.
- Successfully added a new record dialog for every table is the same.
- Error dialog for every table is the same.
- The confirmation dialog when deleting a record is the same for every table, the only difference is the message of the dialog.