BITP 3113: OBJECT ORIENTED PROGRAMMING

Mini Project Deliverable

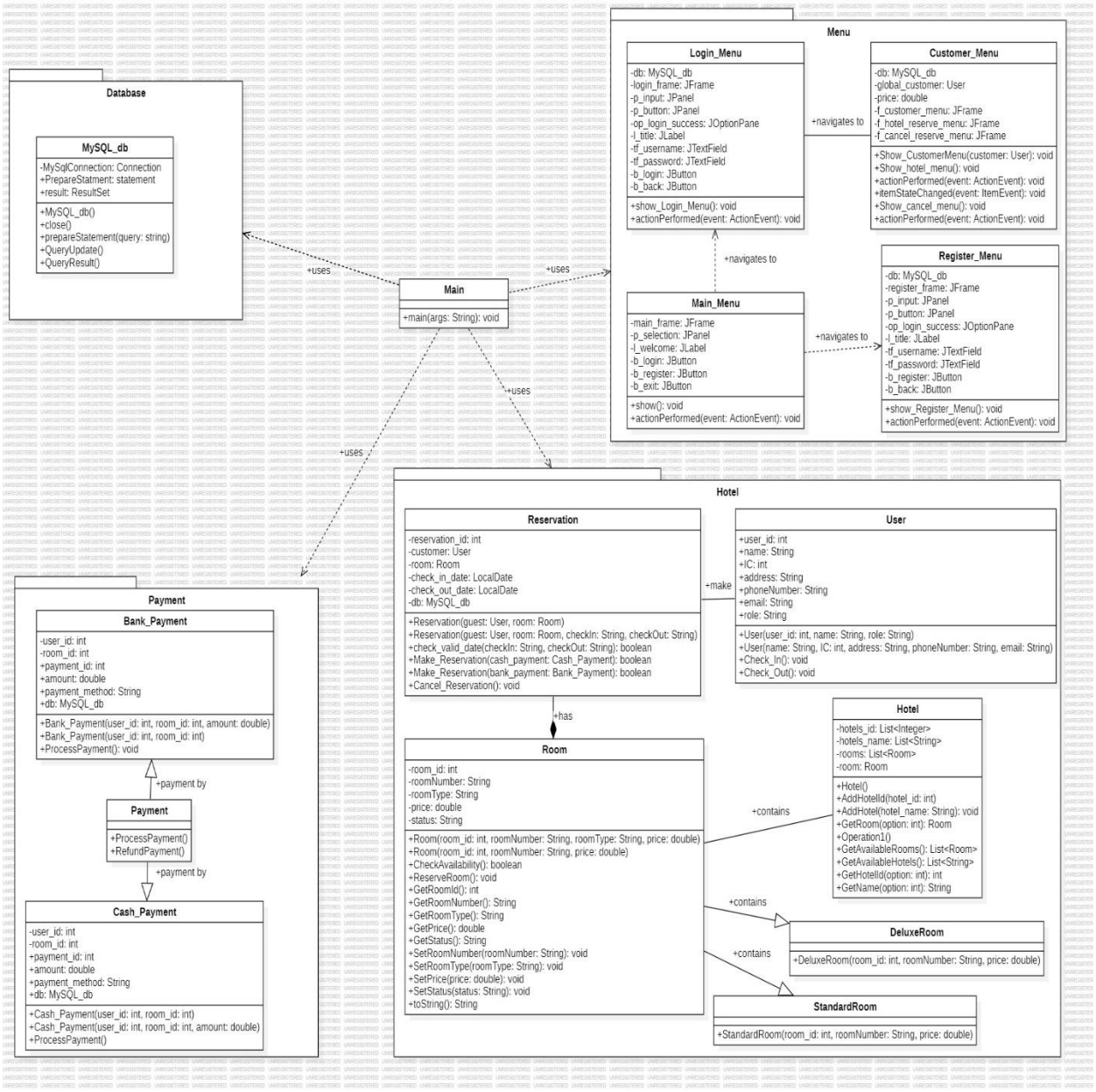
Project Title: Hotel Reservation System

Team members:

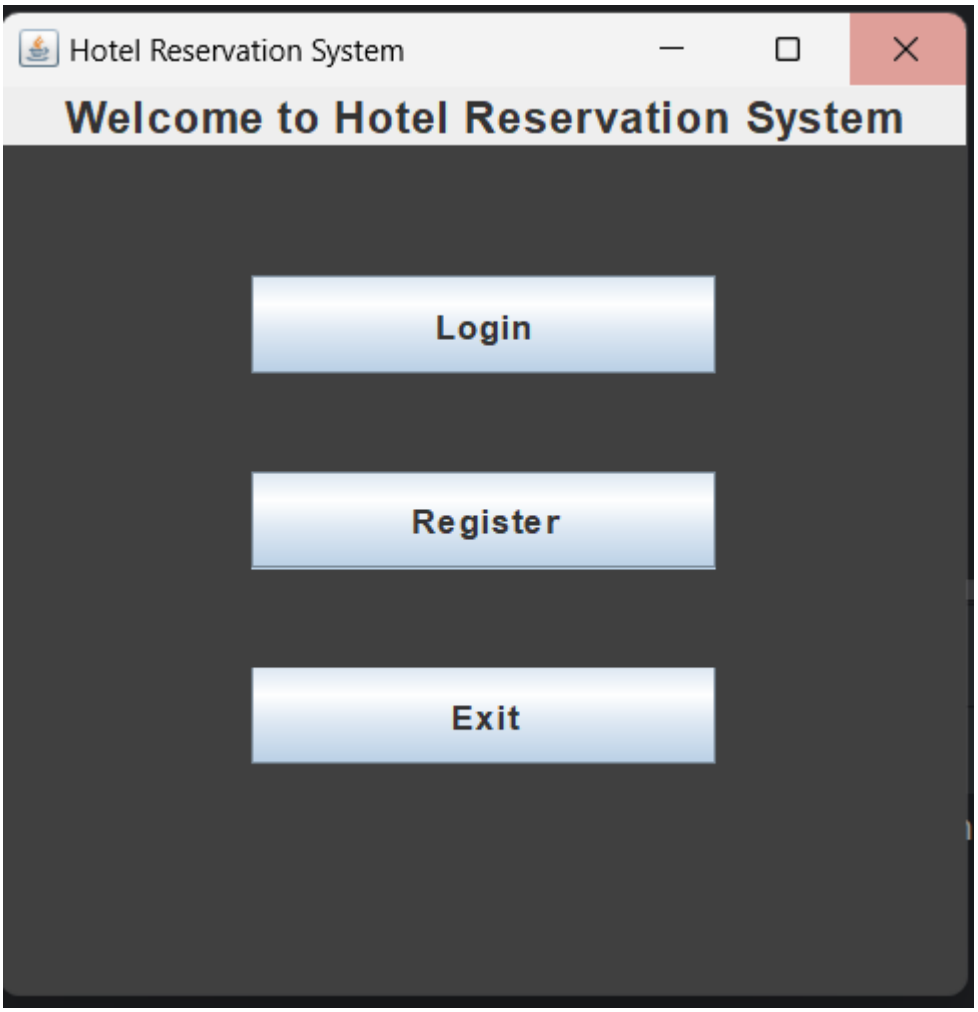
Name	Matric	Section/ Group
DING JIA JUN	B032310366	S1G1
CHEN JIN HAN	B032310333	S1G1
ELDHON CHEONG JIE QI	B032310606	S1G1
CHOONG WILLIAM	B032310371	S1G1
TAN YEE	B032310306	S1G2

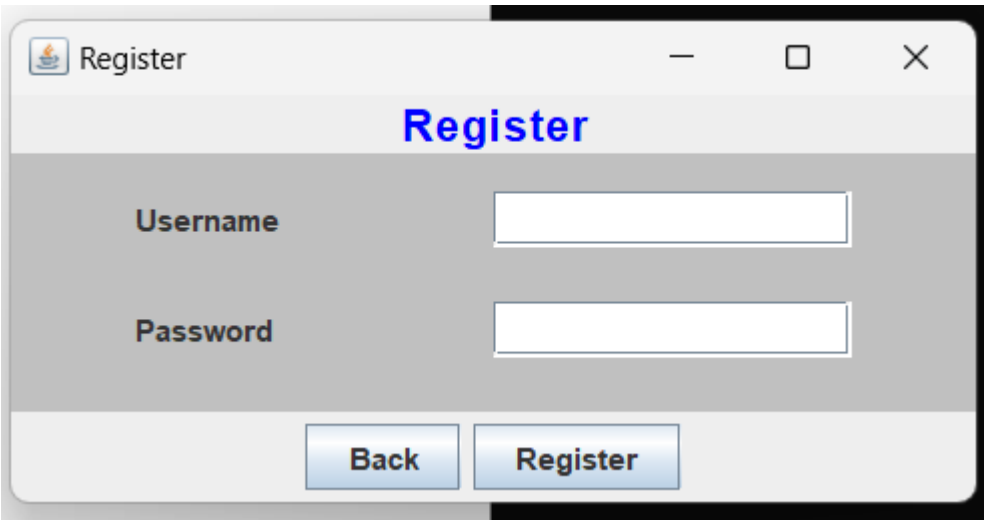
Diagram:

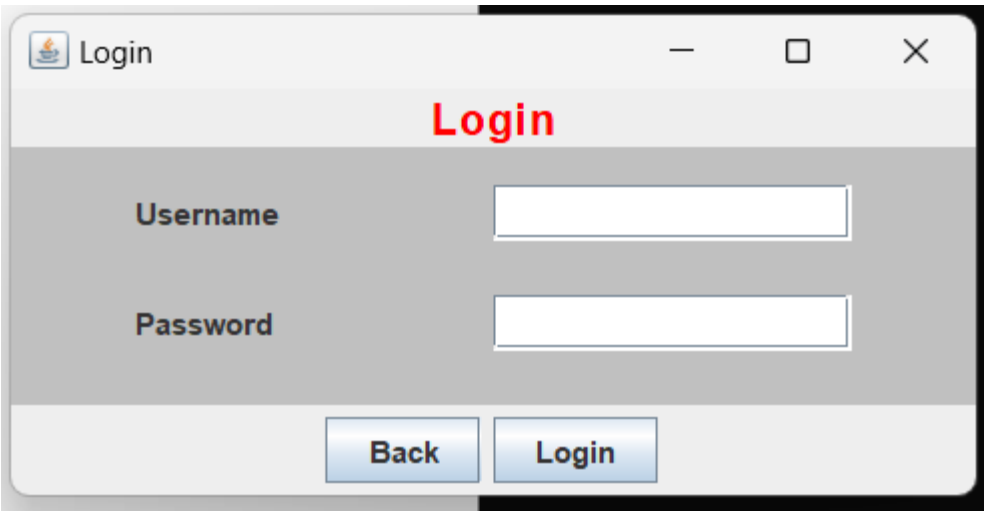
1) UML diagrams

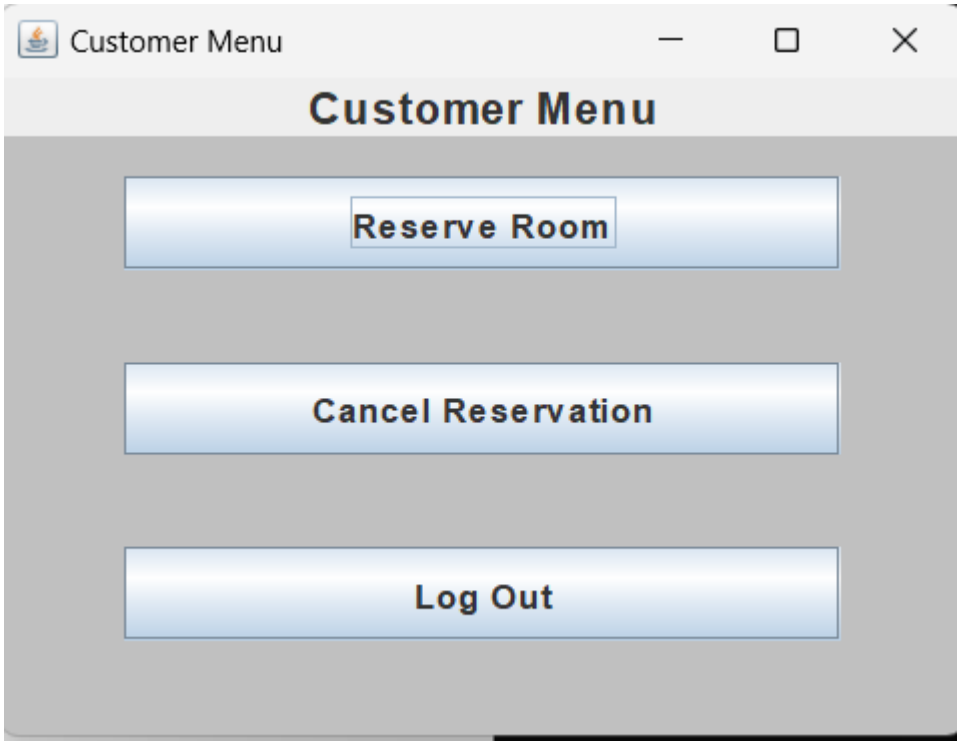


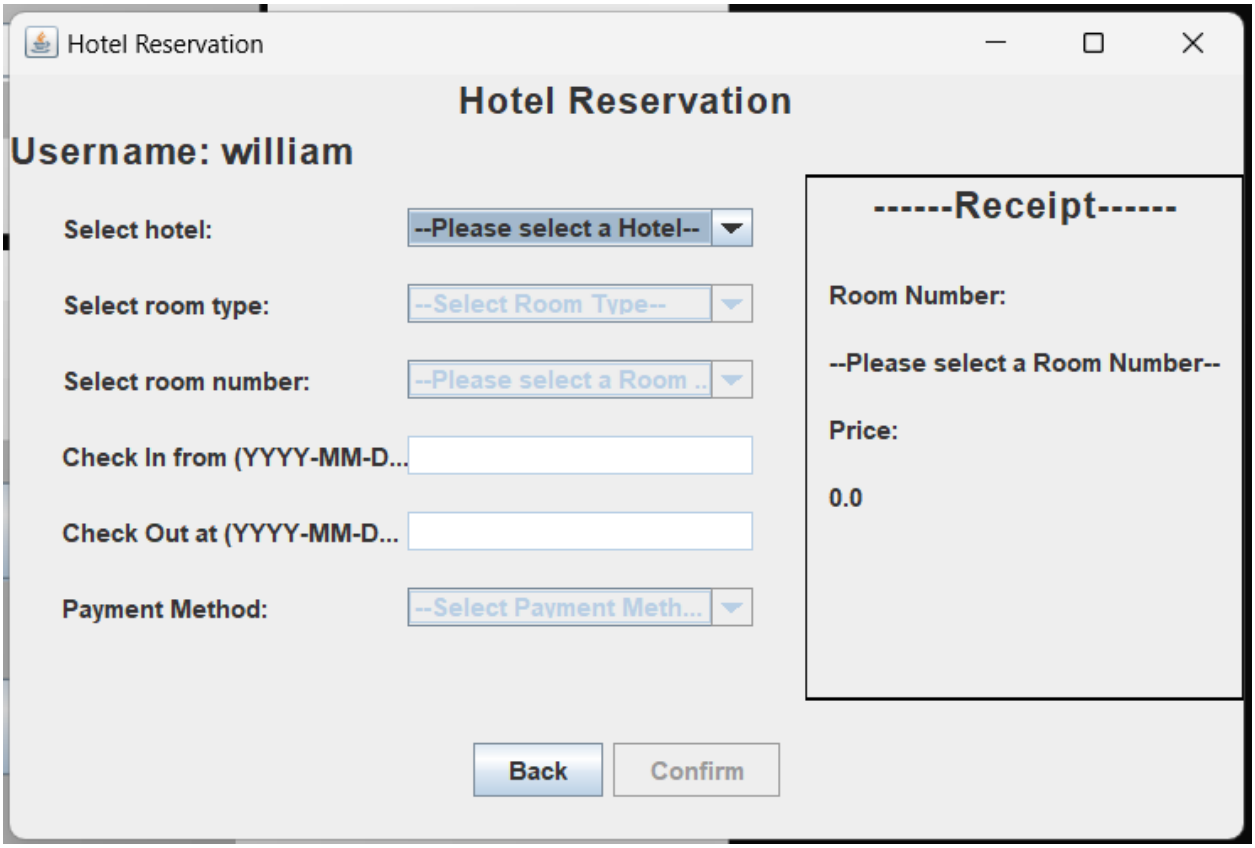
2) GUI

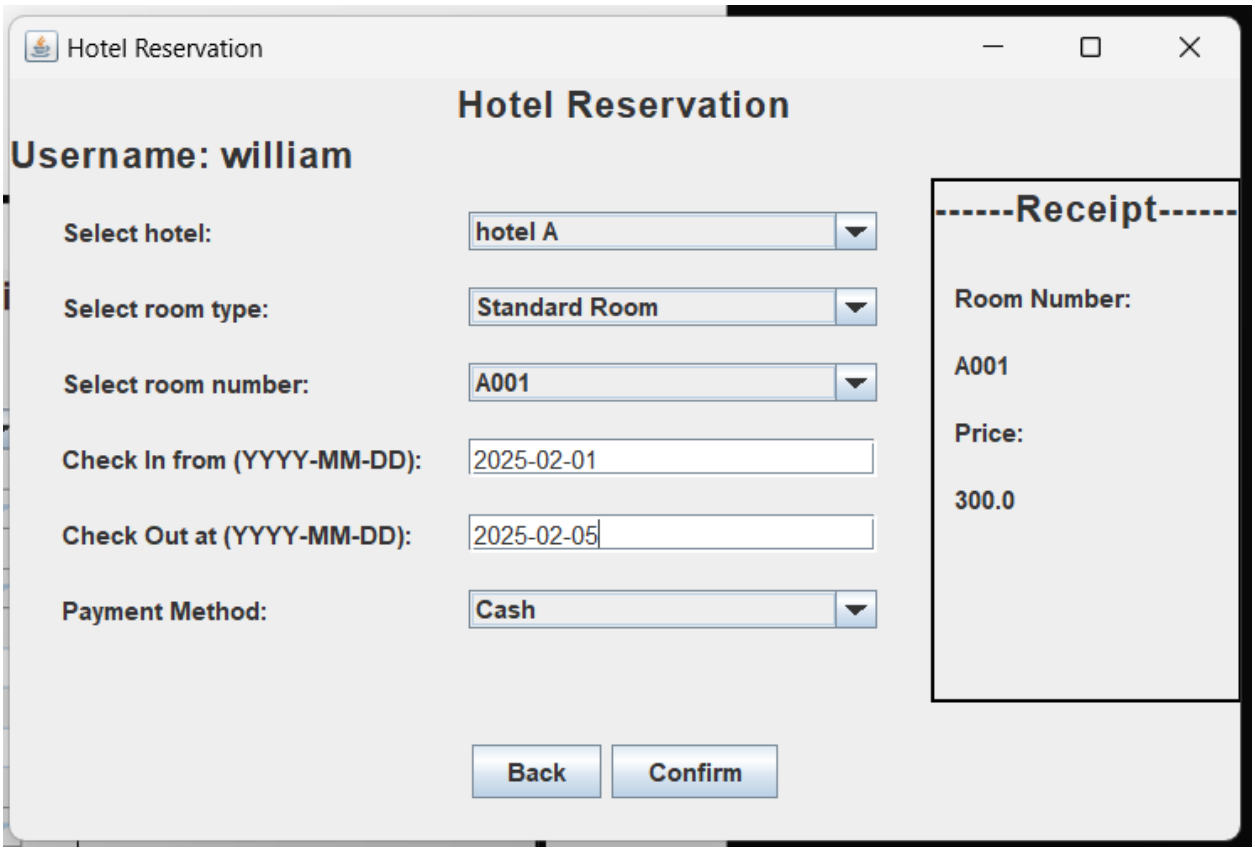

Hotel Reservation System
Welcome to Hotel Reservation System
Login
Register
Exit

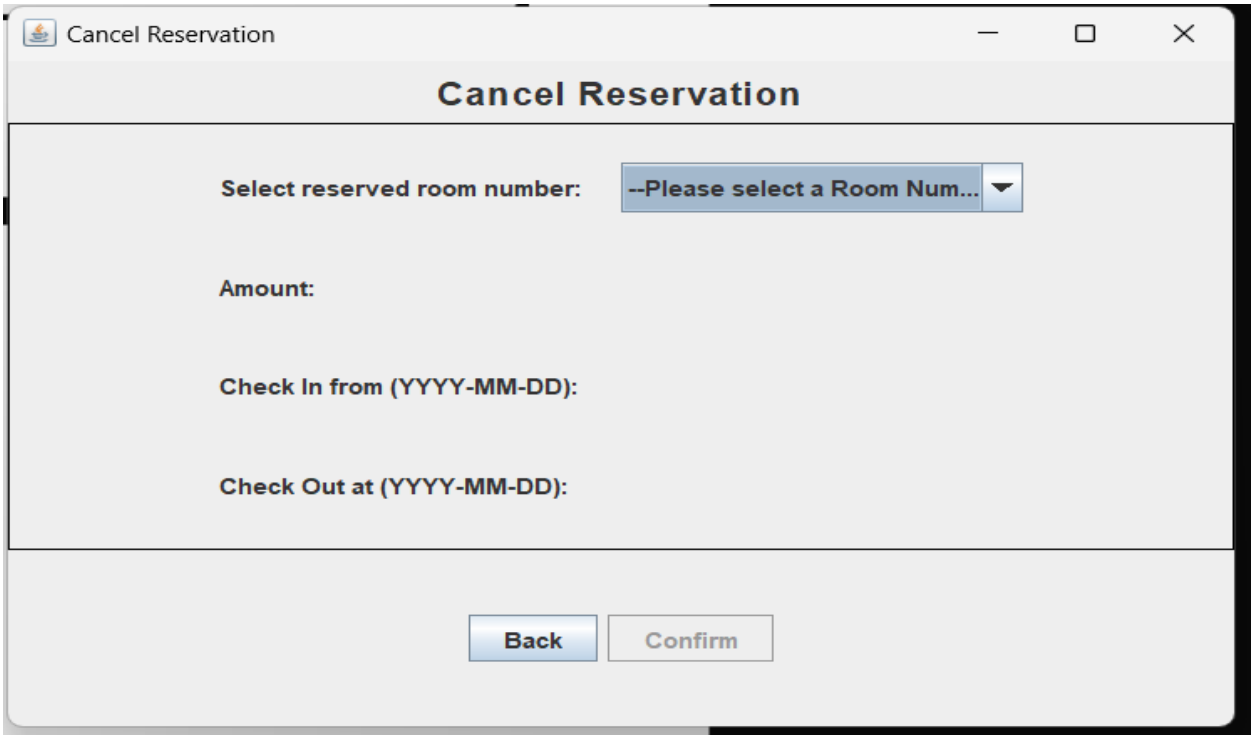

Register
Register
Username
Password
Back
Register

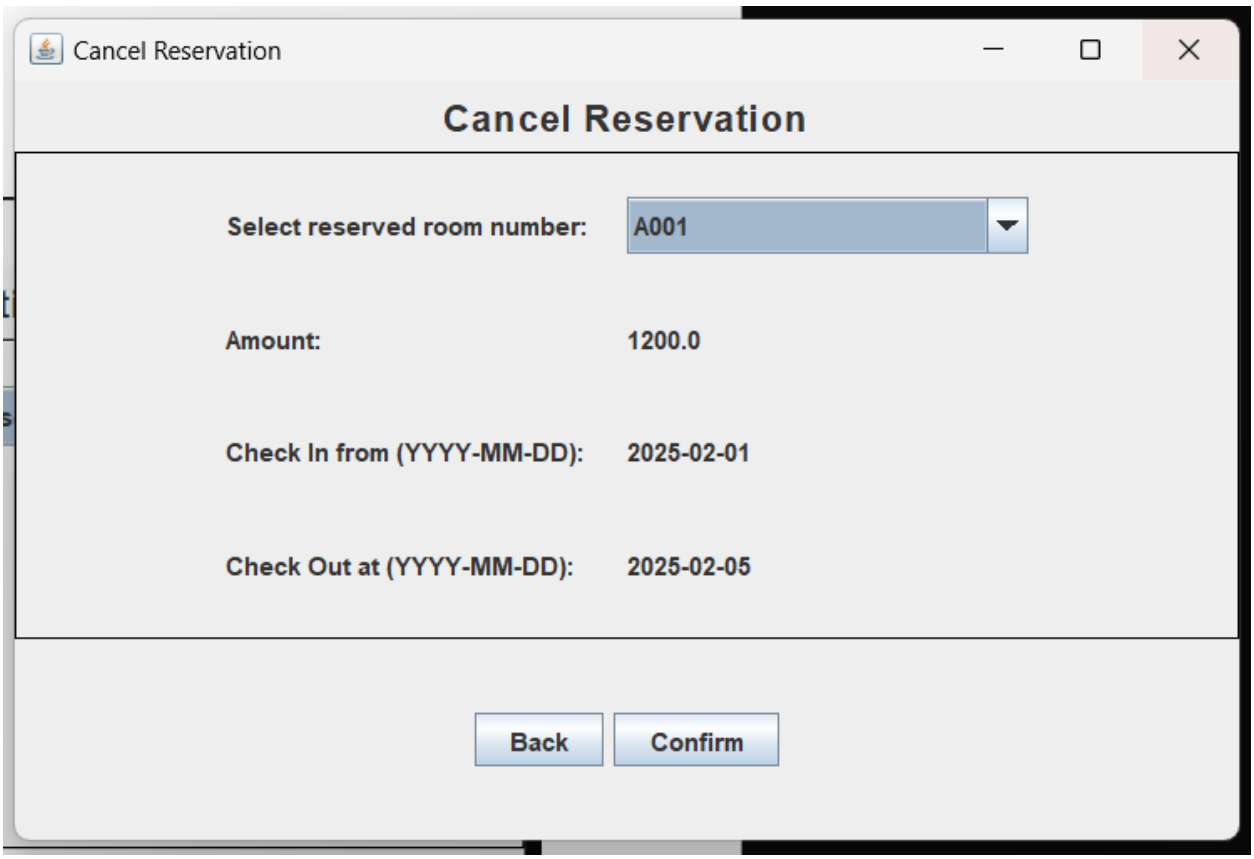

Login
Login
Username
Password
Back
Login


Customer Menu
Customer Menu
Reserve Room
Cancel Reservation
Log Out


Hotel Reservation
Hotel Reservation
Username: william
Select hotel:
--Please select a Hotel--
Select room type:
--Select Room Type--
Select room number:
--Please select a Room ..
Check In from (YYYY-MM-D...
Check Out at (YYYY-MM-D...
Payment Method:
--Select Payment Meth...
------Receipt------
Room Number:
--Please select a Room Number--
Price:
0.0
Back
Confirm


Hotel Reservation
Hotel Reservation
Username: william
Select hotel:
hotel A
Select room type:
Standard Room
Select room number:
A001
Check In from (YYYY-MM-DD):
2025-02-01
Check Out at (YYYY-MM-DD):
2025-02-05
Payment Method:
Cash
------Receipt------
Room Number:
A001
Price:
300.0
Back
Confirm


Cancel Reservation
Cancel Reservation
Select reserved room number:
--Please select a Room Num...
Amount:
Check In from (YYYY-MM-DD):
Check Out at (YYYY-MM-DD):
Back
Confirm


Cancel Reservation
Cancel Reservation
Select reserved room number:
A001
Amount:
1200.0
Check In from (YYYY-MM-DD):
2025-02-01
Check Out at (YYYY-MM-DD):
2025-02-05
Back
Confirm

Code snapshot:

1) **Inheritance / Abstract**

```java
package Hotel;

public class DeluxeRoom extends Room {  1 usage
    public DeluxeRoom(int room_id, String roomNumber, double price) {  1 usage
        super(room_id, roomNumber, roomType: "Deluxe", price);
    }
}
```

```java
package Hotel;

public class StandardRoom extends Room{  3 usages
    public StandardRoom(int room_id, String roomNumber, double price) { super(room_id, roomNumber, roomType: "Standard", price); }
}
```

- Inheritance is a concept where a class (child class) gets properties and methods from another class (parent class), making it easier to reuse code. DeluxeRoom and StandardRoom extends it, providing a concrete implementation for creating specific room types.

```java
package Hotel;

public abstract class Room {  19 usages  2 inheritors
    private int room_id;  3 usages
    private String roomNumber;  6 usages
    private String roomType;  3 usages
    private double price;  4 usages
    private String status;  3 usages
```

- An abstract class is a class that can't be created directly and is meant to be extended by other classes. This is an example of abstract where Room is an abstract class that cannot be instantiated directly

2) Polymorphism

```java
public boolean Make_Reservation(Cash_Payment cash_payment){ 1 usage shuishui
    try {
        // Insert into Reservation
        db.prepareStatement( query: "INSERT INTO reservation (user_id, room_id, checkIn_date, checkOut_date)" +
                " VALUES (?,?,?,?)");
        db.statement.setInt( i: 1, customer.user_id);
        db.statement.setInt( i: 2, room.GetRoomId());
        db.statement.setDate( i: 3, Date.valueOf(check_in_date));
        db.statement.setDate( i: 4, Date.valueOf(check_out_date));
        db.QueryUpdate();

        // Change the status of the room from 'Vacant' to 'Reserved'
        db.prepareStatement( query: "UPDATE room SET status = 'Reserved' WHERE room_id = ?");
        db.statement.setInt( i: 1, room.GetRoomId());
        db.QueryUpdate();

        cash_payment.ProcessPayment();

        return true;
    } catch (SQLException e){
        e.printStackTrace();
        return false;
    }
}
```

```java
public boolean Make_Reservation(Bank_Payment bank_payment){ 1 usage shuishui
    try {
        // Insert into Reservation
        db.prepareStatement( query: "INSERT INTO reservation (user_id, room_id, checkIn_date, checkOut_date)" +
                " VALUES (?,?,?,?)");
        db.statement.setInt( i: 1, customer.user_id);
        db.statement.setInt( i: 2, room.GetRoomId());
        db.statement.setDate( i: 3, Date.valueOf(check_in_date));
        db.statement.setDate( i: 4, Date.valueOf(check_out_date));
        db.QueryUpdate();

        // Change the status of the room from 'Vacant' to 'Reserved'
        db.prepareStatement( query: "UPDATE room SET status = 'Reserved' WHERE room_id = ?");
        db.statement.setInt( i: 1, room.GetRoomId());
        db.QueryUpdate();

        bank_payment.ProcessPayment();

        return true;
    } catch (SQLException e){
        e.printStackTrace();
        return false;
    }
}
```

- Methods make reservation in the same class have the same name but different parameters, allowing them to perform similar tasks on different types of inputs.

3) Interface

Cash_Payment.java Bank_Payment.java Payment.java

```java
package Payment;

public interface Payment {  2 usages  2 implementations
    void ProcessPayment();  2 usages  2 implementations
}
```

Cash_Payment.java Bank_Payment.java

```java
package Payment;
import Database.MySQL_db;

import java.sql.SQLException;

public class Cash_Payment implements Payment{  5 usages
    public int payment_id;  2 usages
    public double amount;  2 usages
    public String payment_method = "Cash";  3 usages
```

Cash_Payment.java Bank_Payment.java

```java
package Payment;
import Database.MySQL_db;

import java.sql.SQLException;

public class Cash_Payment implements Payment{  5 usages
    public int payment_id;  2 usages
    public double amount;  2 usages
    public String payment_method = "Cash";  3 usages
```

4) Java Collection: List / Set / Map

```
public class Hotel {  4 usages
    private List<Integer> hotels_id = new ArrayList<>();  3 usages
    private List<String> hotels_name = new ArrayList<>();  4 usages
    private List<Room> rooms = new ArrayList<>();  6 usages
    Room room = new StandardRoom( room_id: 0,  roomNumber: "--Please select a Room Number--",  price: 0);  2 usages
```

-The List interface in Java is part of the java.util package and provides an ordered collection. It allows duplicates and maintains insertion order.

5) Exception Handling (Create your own)

```
public boolean check_valid_date(String checkIn, String checkOut){  1 usage  shuishui
    try{
        check_in_date = LocalDate.parse(checkIn);
        check_out_date = LocalDate.parse(checkOut);

        // Check is the date logically
        if(check_out_date.isBefore(check_in_date) || check_in_date.isEqual(check_out_date)){
            return false;
        }

        return true;
    } catch (DateTimeParseException e){
        return false;
    }
}
```

- The DateTimeParseException occurs when a date-time string being parsed does not adhere to the expected format or contains invalid date-time information.

6) Connecting to database (Insert, view, search, update, delete)

i. Insert

```java
public void actionPerformed(ActionEvent event) {
    String command = event.getActionCommand();

    switch (command) {
        case "Register":
            try {
                int new_user_id;
                db.prepareStatement( query: "SELECT count(*) AS total FROM user");
                db.QueryResult();
                db.result.next();
                new_user_id = db.result.getInt( columnLabel: "total") + 1;

                db.prepareStatement( query: "INSERT INTO user (user_id, username, password, role) VALUES (?,?,?,'customer')");
                db.statement.setInt( parameterIndex: 1, new_user_id);
                if(tf_username.getText().isEmpty()){throw new Exception("Username cannot be empty");}
                else{db.statement.setString( parameterIndex: 2, tf_username.getText());}
                if(tf_password.getText().isEmpty()){throw new Exception("Password cannot be empty");}
                else{db.statement.setString( parameterIndex: 3, tf_password.getText());}
                db.QueryUpdate();

                JOptionPane.showMessageDialog(register_frame, message: "Register Successfully!", title: "Register"
                        , JOptionPane.INFORMATION_MESSAGE);

                register_frame.dispose();
                Main_Menu main_menu = new Main_Menu();
                main_menu.show();
```

- This code snippet performs an INSERT operation to add a new user into a database table named "user," handling potential empty username/password inputs with exceptions.

ii. Read

```java
public void actionPerformed(ActionEvent event){

    switch (command){
        case "Login":
            try {
                db.prepareStatement( query: "SELECT count(*) AS total FROM user WHERE username = ? AND password = ?");
                db.statement.setString( parameterIndex: 1, tf_username.getText());
                db.statement.setString( parameterIndex: 2, tf_password.getText());
                db.QueryResult();
                db.result.next();

                if(db.result.getInt( columnLabel: "total") == 1){
                    db.prepareStatement( query: "SELECT * FROM user WHERE username = ? AND password = ?");
                    db.statement.setString( parameterIndex: 1, tf_username.getText());
                    db.statement.setString( parameterIndex: 2, tf_password.getText());
                    db.QueryResult();
                    db.result.next();
                    User user = new User(db.result.getInt( columnLabel: "user_id"), db.result.getString( columnLabel: "username")
                            ,db.result.getString( columnLabel: "role"));
                    JOptionPane.showMessageDialog(login_frame,  message: "Login Successfully!",  title: "Login"
                            , JOptionPane.INFORMATION_MESSAGE);

                    if(db.result.getString( columnLabel: "role").equals("customer")){
                        login_frame.dispose();
                        Customer_Menu customer_menu = new Customer_Menu();
                        customer_menu.Show_CustomerMenu(user);
                    } else if (db.result.getString( columnLabel: "role").equals("admin")){
```

- This code performs VIEW operation, one counting matching users and the other retrieving the user's details, to verify login credentials and fetch user information for subsequent actions.

iii. Update

```java
public boolean Make_Reservation(Bank_Payment bank_payment){  1 usage
    try {
        // Insert into Reservation
        db.prepareStatement( query: "INSERT INTO reservation (user_id, room_id, checkIn_date, checkOut_date)" +
                " VALUES (?,?,?,?)");
        db.statement.setInt( parameterIndex: 1, customer.user_id);
        db.statement.setInt( parameterIndex: 2, room.GetRoomId());
        db.statement.setDate( parameterIndex: 3, Date.valueOf(check_in_date));
        db.statement.setDate( parameterIndex: 4, Date.valueOf(check_out_date));
        db.QueryUpdate();

        // Change the status of the room from 'Vacant' to 'Reserved'
        db.prepareStatement( query: "UPDATE room SET status = 'Reserved' WHERE room_id = ?");
        db.statement.setInt( parameterIndex: 1, room.GetRoomId());
        db.QueryUpdate();

        bank_payment.ProcessPayment();

        return true;
    } catch (SQLException e){
        e.printStackTrace();
        return false;
    }
}
```

- This code performs an UPDATE operation on the "room" table, changing the "status" of a specific room to "Reserved" after a reservation is made.

iv. Delete

```java
public void Cancel_Reservation(){  1 usage
    try {
        // Get and delete the last payment id for that reservation
        db.prepareStatement( query: "SELECT payment_id FROM payment WHERE user_id = ? AND room_id = ? " +
                "ORDER BY payment_id DESC LIMIT 1;");
        db.statement.setInt( parameterIndex: 1, customer.user_id);
        db.statement.setInt( parameterIndex: 2, room.GetRoomId());
        db.QueryResult();
        db.result.next();
        int payment_id = db.result.getInt( columnLabel: "payment_id");

        db.prepareStatement( query: "DELETE FROM payment WHERE payment_id = ?");
        db.statement.setInt( parameterIndex: 1, payment_id);
        db.QueryUpdate();

        // Change the status of the room from 'Reserved' to 'Vacant'
        db.prepareStatement( query: "UPDATE room SET status = 'Vacant' WHERE room_id = ?");
        db.statement.setInt( parameterIndex: 1, room.GetRoomId());
        db.QueryUpdate();

        // Delete the reservation
        db.prepareStatement( query: "DELETE FROM reservation WHERE user_id = ? AND room_id = ?");
        db.statement.setInt( parameterIndex: 1, customer.user_id);
        db.statement.setInt( parameterIndex: 2, room.GetRoomId());
        db.QueryUpdate();

    } catch (SQLException e){
        System.out.println("Something error in Canceling Reservation!");
    }
}
```

- This code performs DELETE operations to cancel a reservation by first deleting the associated payment record, then updating the room status to "Vacant", and finally deleting the reservation record itself from the database.