FACULTY OF COMPUTING

SEMESTER 2 2024/2025

(SECJ2154) OBJECT ORIENTED PROGRAMMING

SECTION 2

MINI PROJECT: HOTEL BOOKING SYSTEM

LECTURER: DR. ZUIRANI BINITI ALI SHAH

STUDENT NAME	MATRIC NO
GUI KAH SIN	A23CS0080
MUHAMMAD AFIQ DANIAL BIN ROZAIDIE	A23CS0117
NABIL AFLAH BOO BINTI MOHD YOSUF BOO YONG CHONG	A23CS0252

Table of Contents

1.0 INTRODUCTION

The Hotel Booking System is designed to streamline the overall operation of room reservations through an online platform. The system enables users to choose from a variety of room types, check the availability of rooms, view the pricing and make reservations based on their desired dates without the need to physically visit the hotel or call the front desk.

Once the room has been selected and reservation details are confirmed, the system prompts users to enter their personal information, including their name, contact number, identification number and so on. This step ensures that the booking is properly recorded and associated with the correct individual. Following this, users are directed to the payment process, where they are given the flexibility to choose between two payment options, online or offline. For online bookings, the system requires full payment upfront to confirm the reservation, providing a seamless and immediate transaction process. On the other hand, the offline option allows users to pay only 50% of the total amount as deposit, offering more flexibility for those who prefer to complete the remaining payment at a later stage or upon arrival. Upon successful booking, the system will automatically generate a digital receipt containing all relevant reservation details for user reference.

Beyond just handling bookings, the Hotel Booking System also performs several other essential functions that assist with hotel management. It maintains comprehensive reservation records, continuously updates room availability, and supports booking cancellation. By automating these tasks, the system significantly reduces the amount of manual work required by hotel staff, minimizes the potential for human errors, and ensures a smoother experience for customers. Overall, this structured and user-friendly system not only helps improve efficiency in hotel management but also enhances customer satisfaction, making it a practical solution for both hotel staff and customers.

2.0 PROBLEM STATEMENT

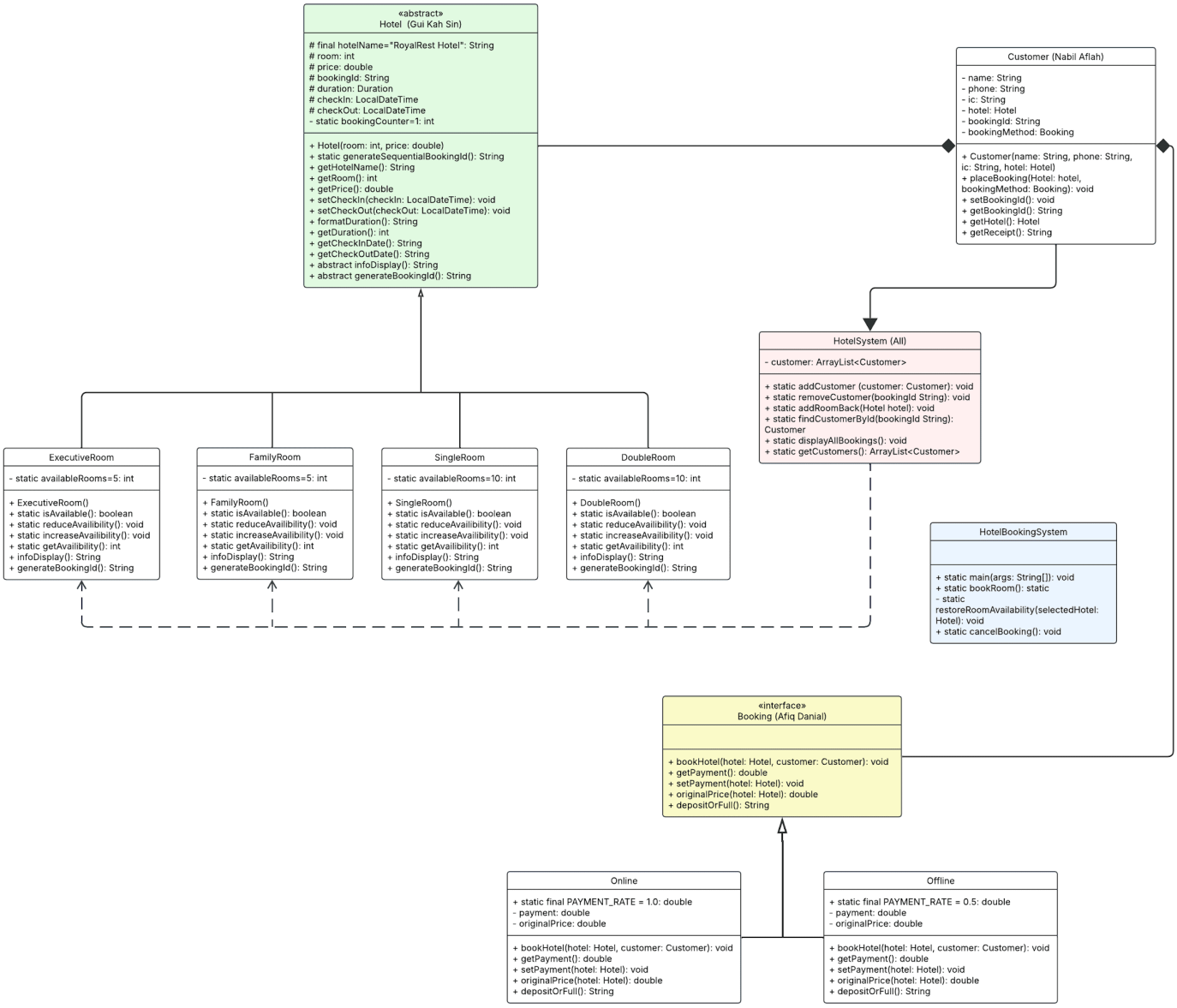
The effectiveness of hotel service delivery is severely hampered by the manual installation of the hotel booking system that was recently implemented. The majority of reservation records are manually maintained in physical registers, which allows for human mistakes in the form of duplicate bookings, inaccurate customer information, and missed reservations. Additionally, it takes a lot of time since employees must update the records and verify that there are rooms available, which keeps customers waiting. There is a risk of overbooking or underutilized rooms because there is no real-time booking system in place. Furthermore, data management is inaccurate since it necessitates manual labor to compile reports, retrieve guest records, and evaluate booking trends, all of which could result in information loss. Since they can only make bookings over the phone or in person, customers frequently have limited access to this manual booking system approach, which limits their convenience, especially after office hours. Employees must manually manage every consumer request for changes or cancellations, which leads to mistakes and delays.

The suggested automated hotel booking system is a case study of object-oriented programming, which aims for a system structure that is clear and modular. The technology will provide real-time updates on hotel availability in an attempt to reduce the number of reservations and increase data accuracy. Customers will be able to make, amend, change, or cancel reservations online at any time thanks to an intuitive interface. The database will be updated in accordance with the changes since the system manages them automatically. The system's primary features are centered on processing payments, managing reservations, and managing client records. These functions are divided into distinct classes, each of which performs a specific role to protect data and keep the system operating. In comparison to traditional processes, the system will also enable online payments and generate receipts with the customer's reservation details, making the entire booking process quicker, more efficient, and more customer-satisfying.

3.0 OBJECTIVES

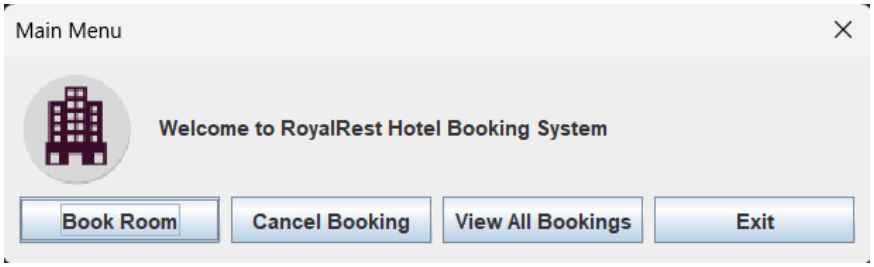
1. To develop an automated hotel reservation system that enables customers to browse room types, check availability, and make bookings online without the need for physical interaction or phone calls.
2. To minimize the human errors and double bookings by validating customer inputs, enforcing booking rules and automatically updating the system's database.
3. To enhance customer convenience and satisfaction by offering 24/7 access to the booking platform, providing instant confirmation with a digital receipt, and streamlining the check-in and check-out process.
4. To apply Object-Oriented programming (OOP) principles in the system development, ensuring modular and scalable code by organizing functionalities into clearly defined classes.
5. To reduce the workload of the hotel by automating common administrative tasks such as updating room availability, managing booking records, and generating receipts.

4.0 PROJECT DESIGN (CLASS DIAGRAM)



5.0 PROJECT OUTPUT

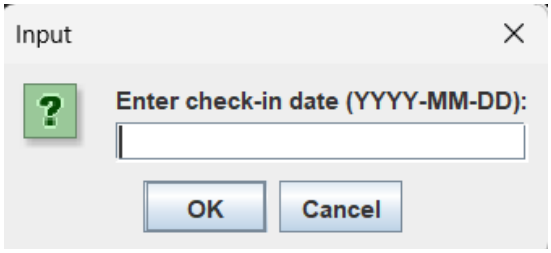
1. Main Menu of Hotel Booking System
   - Displays the main options: Book Room, Cancel Booking, View All Bookings, and Exit. Users select an option to proceed. Icon added for a decoration.



2. Book Room: Room Selection Menu
   - Show available room types with their respective price and availability. Users choose their preferred room type to book.

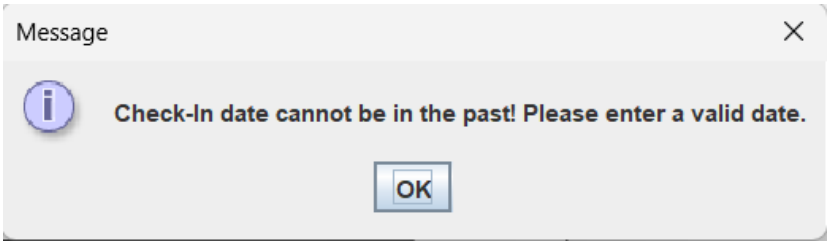
2.1 Book Room: Enter Check-in and Check-out Dates

- After users choose their preferred room type, they are prompted to enter the check-in and check-out dates.

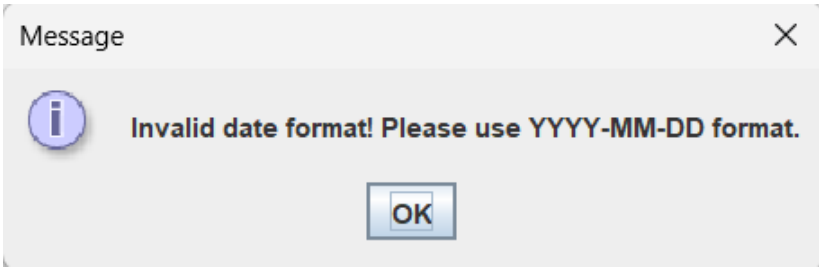


Validation Rules:

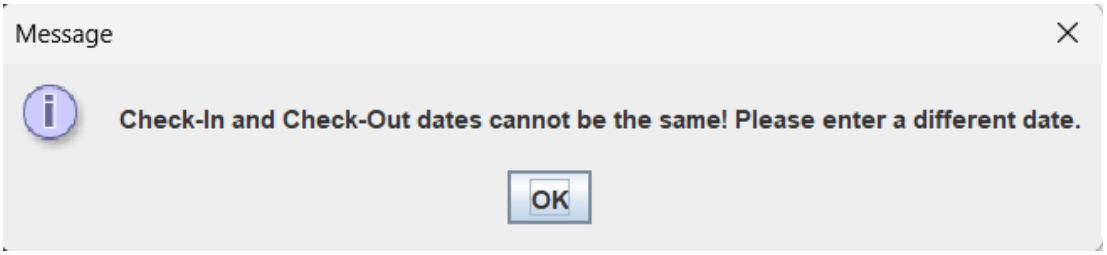
2.1a Dates cannot be in the past.



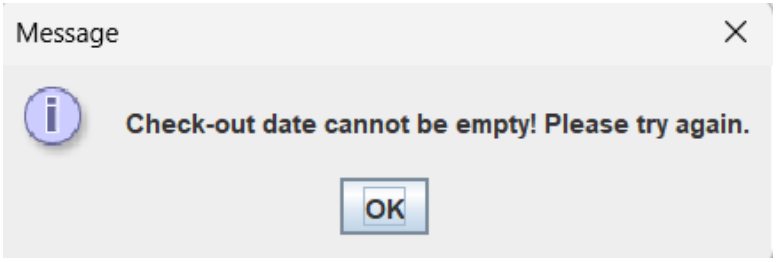
2.1b The format must be YYYY-MM-DD.



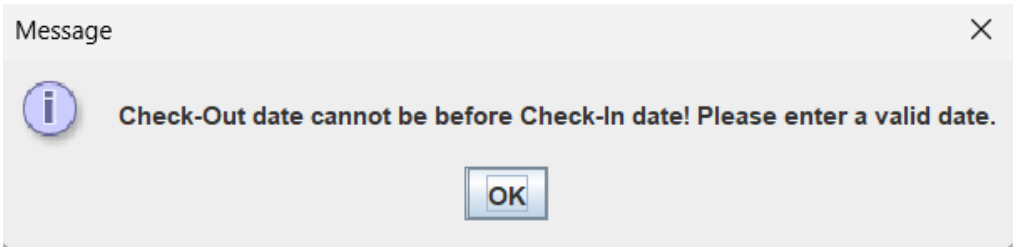
2.1c Check-out date cannot be the same as check-in



2.1d Both fields must not be empty.

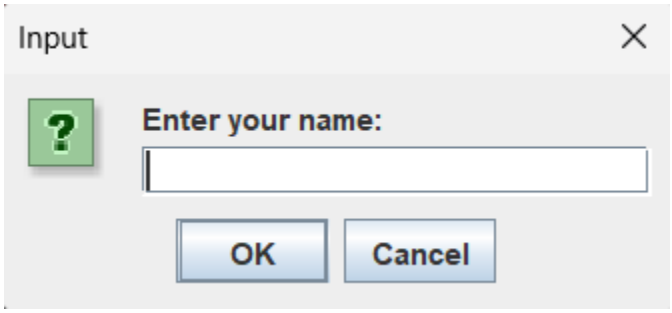


2.1e Check-out must be after check-in.



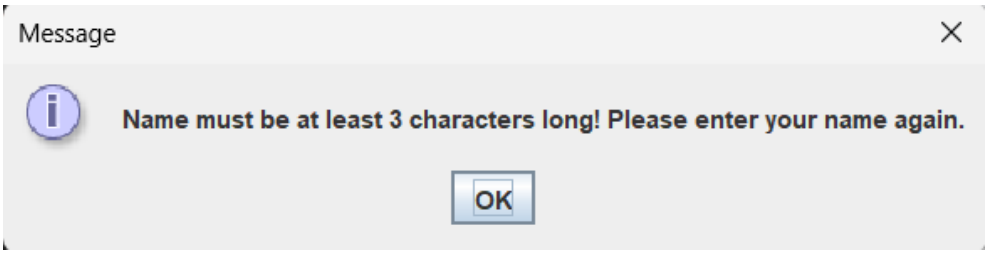
2.2 Book Room: Enter Personal Information - Name Field

- After entering the dates, the user needs to fill in their name.

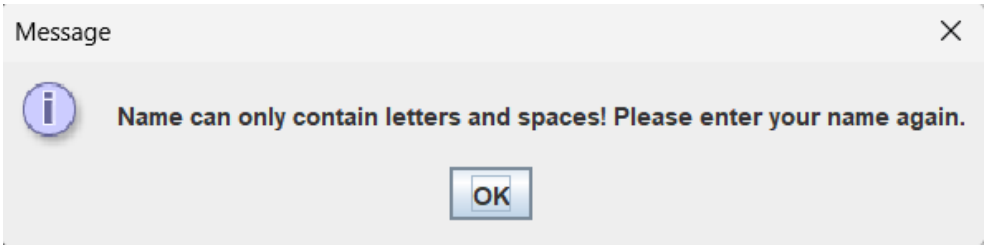


Validation Rules:

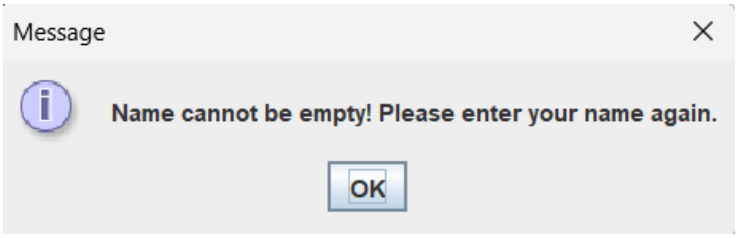
2.2a Must contain more than 2 characters.



2.2b Only letters and spaces are allowed.

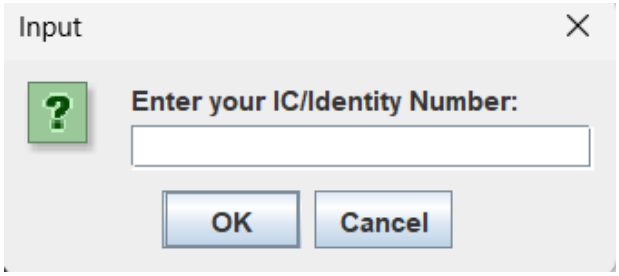


2.2c Field cannot be empty.



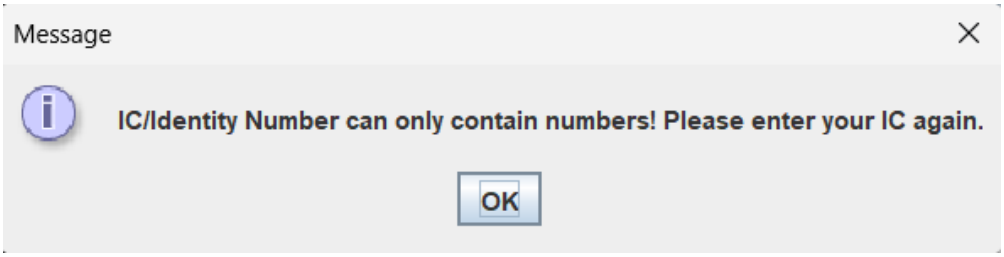
2.3 Book Room: Enter Personal Information - IC Number Field

- Users need to enter their identity number as well.

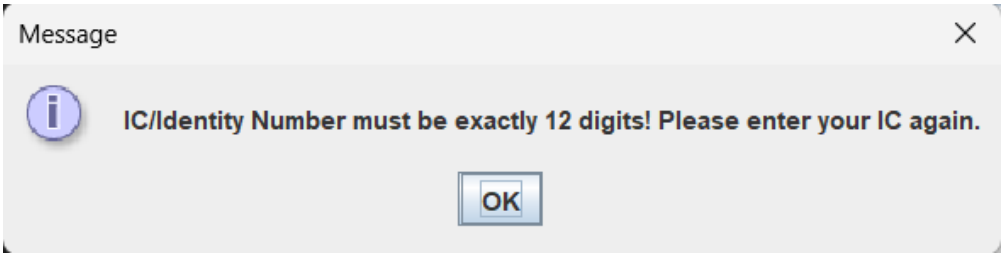


Validation Rules:

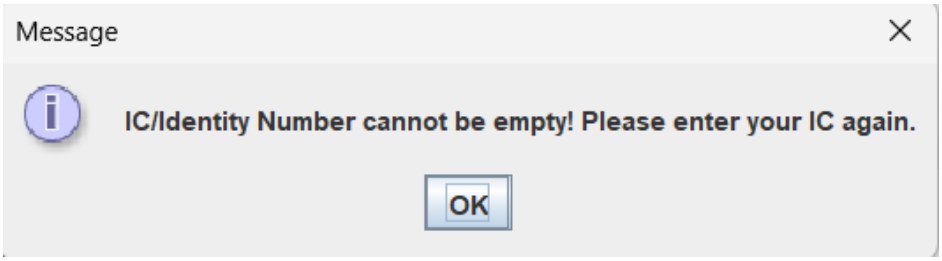
2.3a Only numbers are allowed.



2.3b Must contain exactly 12 digits.

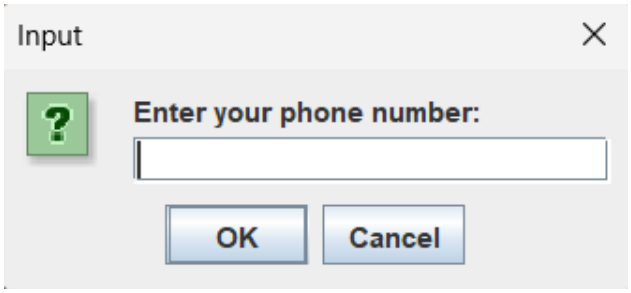


2.3c Field cannot be empty.



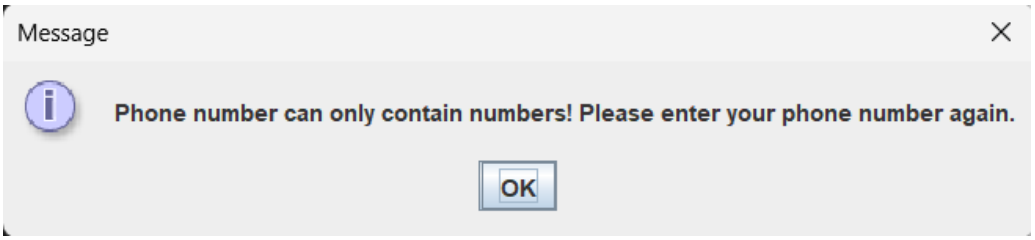
2.4 Book Room: Enter Personal Information - Phone Number Field

- Users then enters the phone number.

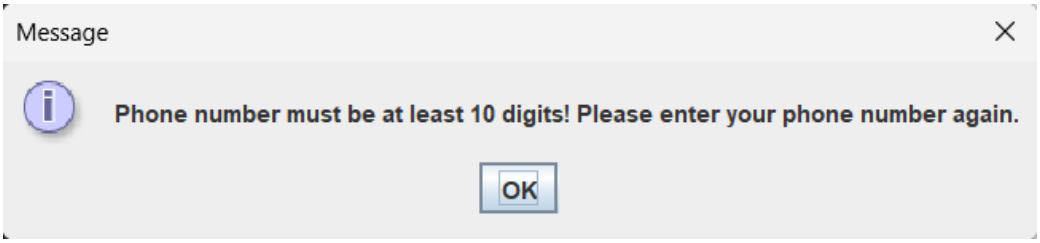


Validation Rules:

2.4a Only numbers are allowed.

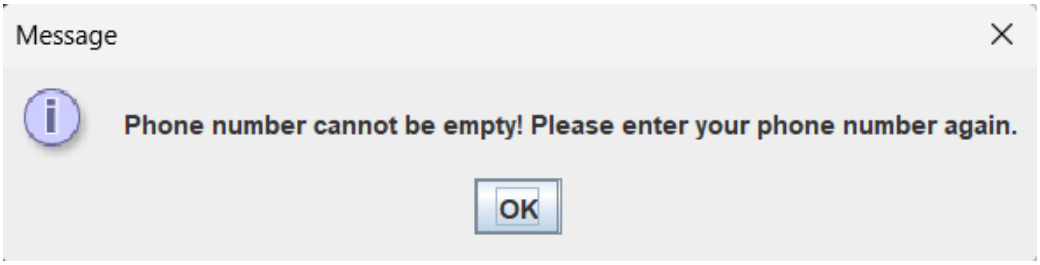


2.4b Must contain at least 10 digits.



2.4c Must less than 11 digits.

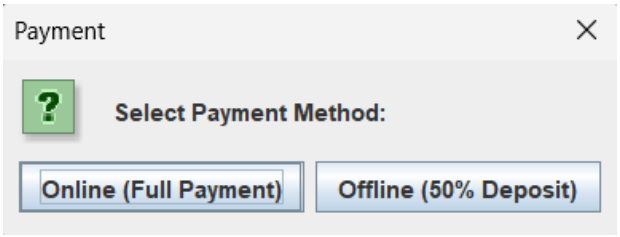
2.4d Field cannot be empty



2.5 Book Room: Select Payment Method

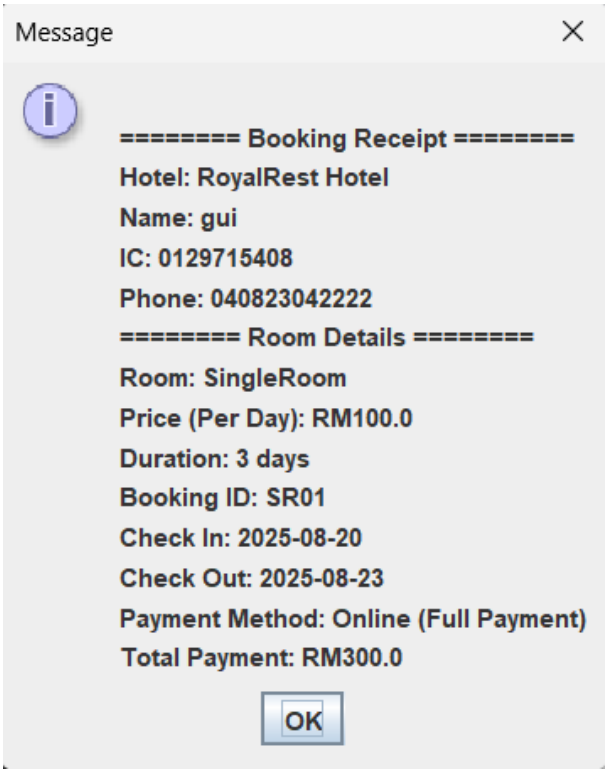
There are two payment options available:

- **Online Payment** – Requires full payment to confirm the booking.
- **Offline Payment** – Only 50% of the total is paid as a deposit



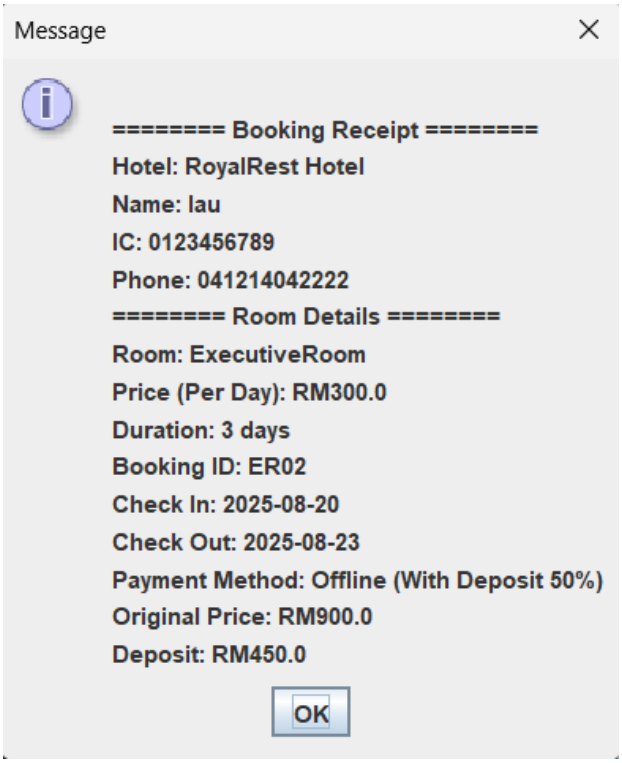
- After the users select the payment method, the system displays a summary of the reservation.

2.5a Online payment:

- Only the paid amount is shown.
- 

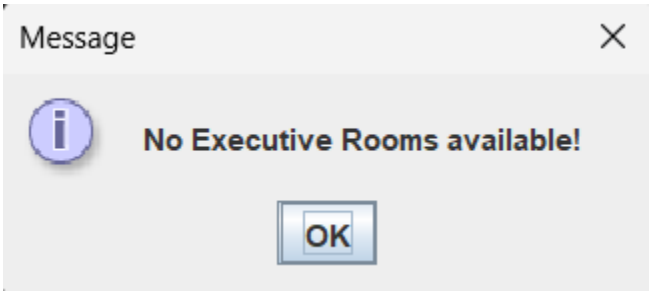
2.5b Offline Payment

- The original room price is shown along with the deposit amount.

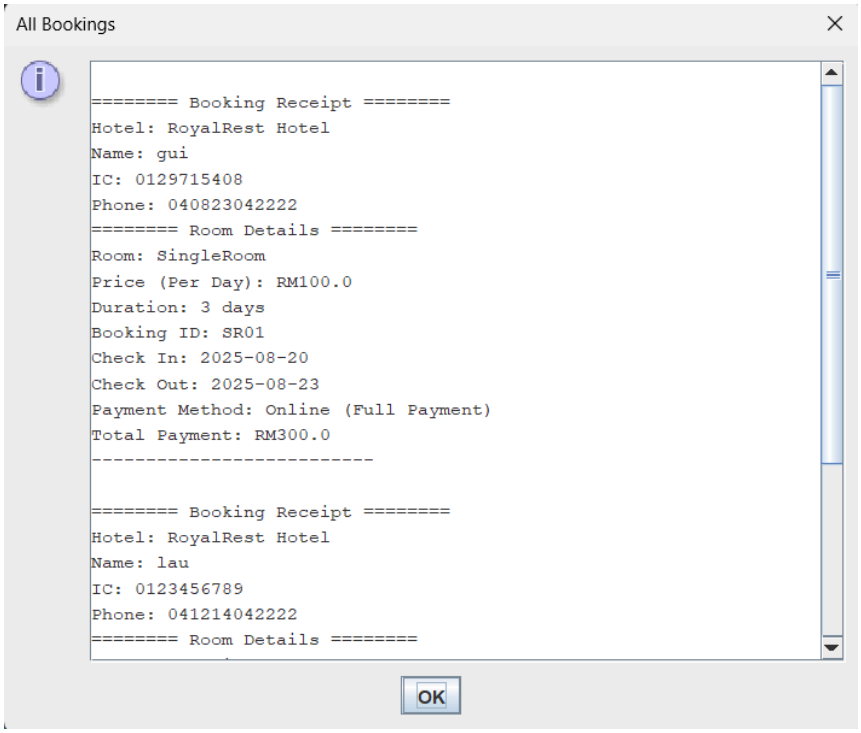


2.6 Book Room: Room Type Not Available

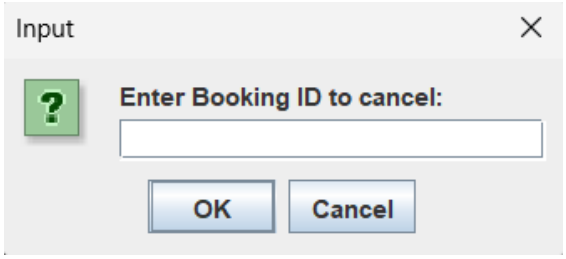
- Displays an error message when the selected room type is fully booked and unavailable for reservation.



3. View All Bookings
   - Displays a list of all current bookings, each with complete reservation details and a unique booking ID.



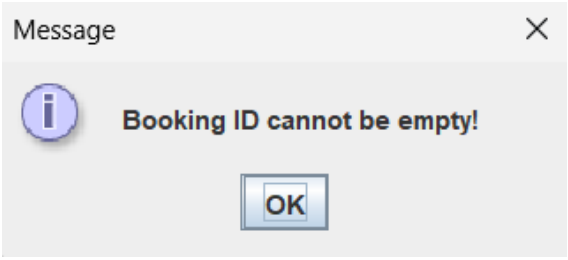
4. Cancel Booking: Input Booking ID
   - Prompts users to enter their Booking ID to proceed with cancelling their reservation.



4.1 Cancel Booking: Empty Field

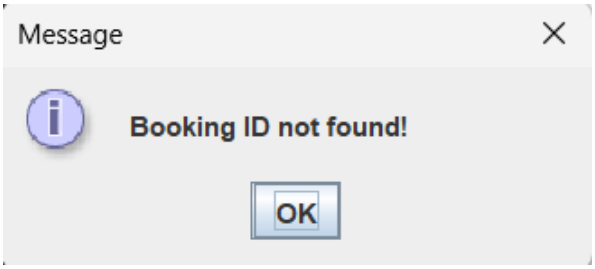
Validation Rules:

- Field cannot be empty.



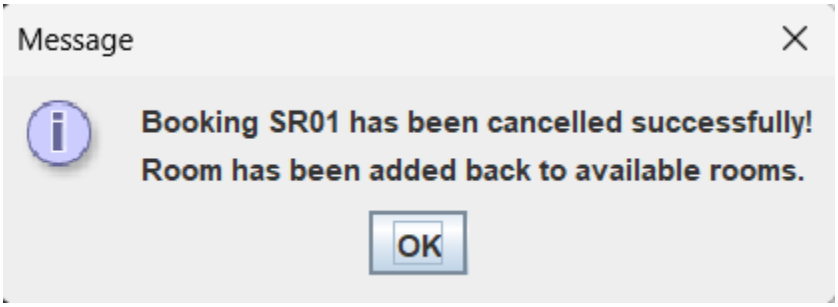
4.2 Cancel Booking: Booking ID Not Found

- Booking ID must exist in the system; otherwise, an error message will be shown.



4.3 Cancel Booking: Successful Cancellation

- Displays a confirmation message when the booking is successfully cancelled.



5. Exit
   - The system terminates immediately upon selecting the “Exit” option from the main menu.

6.0 CONCLUSION

In conclusion, the proposed Hotel Booking System offers an effective solution to the limitations of the current manual reservation process by automating key operations such as room bookings, customer data management, payment processing, and receipt generation. The system was created with the help of Object-Oriented Programming (OOP) principles, which guarantee design clarity, scalability, and modularity. Customers can make, change, or cancel reservations online with real-time room availability and flexible payment options thanks to its round-the-clock access to hotel services. The administrative burden for hotel employees is greatly reduced by this automation, which also lessens human error and avoids duplicate reservations. All things considered, the system improves customer satisfaction and operational efficiency by providing a smooth, precise, and easy-to-use booking experience.

7.0 APPENDICES

Full Java Program
https://drive.google.com/file/d/1bEWotsvNSAF-ljdYDruGoH3ijfu6NdqM/view?usp=sharing

Booking.java

```java
public interface Booking { // class interface & polymorphism
    Trae: Explain | Doc | Test | Explore IDE | ×
    public void bookHotel(Hotel hotel, Customer customer);
    Trae: Explain | Doc | Test | Explore IDE | ×
    public double getPayment();
    Trae: Explain | Doc | Test | Explore IDE | ×
    public void setPayment(Hotel hotel);
    Trae: Explain | Doc | Test | Explore IDE | ×
    public double originalPrice(Hotel hotel);
    Trae: Explain | Doc | Test | Explore IDE | ×
    public String depositOrFull();
}
```

Online.java

```java
public class Online implements Booking {
    public static final double PAYMENT_RATE = 1.0;
    private double payment;
    private double originalPrice;

    Trae: Explain | Doc | Test | Explore IDE | ×
    public void bookHotel(Hotel hotel, Customer customer) {
        setPayment(hotel);
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public void setPayment(Hotel hotel) {
        this.payment = hotel.getPrice() * PAYMENT_RATE * hotel.getDuration();
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public double getPayment() {
        return payment;
    }
    Trae: Explain | Doc | Test | Explore IDE | ×
    public double originalPrice(Hotel hotel){
        this.originalPrice = hotel.getPrice() * hotel.getDuration();
        return originalPrice;
    }
    Trae: Explain | Doc | Test | Explore IDE | ×
    public String depositOrFull(){
        return " (Full Payment)";
    }
}
```

Offline.java

```java
public class Offline implements Booking {
    public static final double PAYMENT_RATE = 0.5;
    private double payment;
    private double originalPrice;

    Trae: Explain | Doc | Test | Explore IDE |
    public void bookHotel(Hotel hotel, Customer customer) {
        setPayment(hotel);
    }

    Trae: Explain | Doc | Test | Explore IDE |
    public void setPayment(Hotel hotel) {
        this.payment = hotel.getPrice() * PAYMENT_RATE * hotel.getDuration();
    }

    Trae: Explain | Doc | Test | Explore IDE |
    public double getPayment() { //accessor method
        return payment;
    }
    Trae: Explain | Doc | Test | Explore IDE |
    public double originalPrice(Hotel hotel){
        this.originalPrice = hotel.getPrice() * hotel.getDuration();
        return originalPrice;
    }

    Trae: Explain | Doc | Test | Explore IDE |
    public String depositOrFull(){
        return " (With Deposit 50%)";
    }
}
```

Hotel.java

```java
import java.time.Duration;
import java.time.LocalDateTime;

public abstract class Hotel { // abstract class & inheritance
    protected final String hotelName = "RoyalRest Hotel";
    protected int room;
    protected double price;
    protected String bookingId;
    protected Duration duration;
    protected LocalDateTime checkIn;
    protected LocalDateTime checkOut;

    private static int bookingCounter = 1;


    public Hotel(int room, double price) {
        this.room = room;
        this.price = price;
    }

    // Add method to generate sequential booking ID
    Trae: Explain | Doc | Test | Explore IDE | ×
    public static String generateSequentialBookingId() {
        String id = String.format(format:"%02d", bookingCounter);
        bookingCounter++;
        return id;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public String getHotelName() {
        return hotelName;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public int getRoom() {
        return room;
    }
```

```
Trae: Explain | Doc | Test | Explore IDE | ×
public double getPrice() {
    return price;
}

Trae: Explain | Doc | Test | Explore IDE | ×
public void setCheckIn(LocalDateTime checkIn) {
    this.checkIn = checkIn;
}

Trae: Explain | Doc | Test | Explore IDE | ×
public void setCheckOut(LocalDateTime checkOut) {
    this.checkOut = checkOut;
    if (checkIn != null && checkOut != null) {
        this.duration = Duration.between(checkIn, checkOut);
    }
}

Trae: Explain | Doc | Test | Explore IDE | ×
public String formatDuration() {
    if (duration == null) {
        return "Not Set";
    }
    return duration.toDays() + " day(s)";
}

Trae: Explain | Doc | Test | Explore IDE | ×
public int getDuration() { // convert duration into int
    long days = duration.toDays();
    int daysAsInt = (int) days;
    return daysAsInt;
}
```

```
Trae: Explain | Doc | Test | Explore IDE | ×
public String getCheckOutDate() {
    return (checkOut != null) ? checkOut.toLocalDate().toString() : "Not Set";
}

Trae: Explain | Doc | Test | Explore IDE | ×
public abstract String infoDisplay();
Trae: Explain | Doc | Test | Explore IDE | ×
public abstract String generateBookingId();
}
```

SingleRoom.java

```java
public class SingleRoom extends Hotel {
    private static int availableRooms = 10;

    public SingleRoom() {
        super(availableRooms, price:100);
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static boolean isAvailable() {
        return availableRooms > 0;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static void reduceAvailability() {
        if (availableRooms > 0) {
            availableRooms--;
        }
    }

    // Add this method to each room class
    Trae: Explain | Doc | Test | Explore IDE | ×
    public static void increaseAvailability() {
        availableRooms++;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static int getAvailability() {
        return availableRooms;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    @Override
    public String infoDisplay() {
        return "Single Room | RM" + price + "/day | Available: " + availableRooms;
    }
```

```java
    Trae: Explain | Doc | Test | Explore IDE | ×
    @Override
    public String generateBookingId() {
        return "SR" + Hotel.generateSequentialBookingId();
    }
}
```

DoubleRoom.java

```java
public class DoubleRoom extends Hotel {
    private static int availableRooms = 10;

    public Dou int availableRooms
        super(availableRooms, price:150);
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static boolean isAvailable() {
        return availableRooms > 0;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static void reduceAvailability() {
        if (availableRooms > 0) {
            availableRooms--;
        }
    }

    // Add this method to each room class
    Trae: Explain | Doc | Test | Explore IDE | ×
    public static void increaseAvailability() {
        availableRooms++;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static int getAvailability() {
        return availableRooms;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    @Override
    public String infoDisplay() {
        return "Double Room | RM" + price + "/day | Available: " + availableRooms;
    }
```

```java
    @Override
    public String generateBookingId() {
        return "DR" + Hotel.generateSequentialBookingId();
    }
}
```

FamilyRoom.java

```java
public class FamilyRoom extends Hotel {
    private static int availableRooms = 5;

    public FamilyRoom() {
        super(availableRooms, price:200);
    }

    Trae: Explain | Doc | Test | Explore IDE |
    public static boolean isAvailable() {
        return availableRooms > 0;
    }

    Trae: Explain | Doc | Test | Explore IDE |
    public static void reduceAvailability() {
        if (availableRooms > 0) {
            availableRooms--;
        }
    }

    // Add this method to each room class
    Trae: Explain | Doc | Test | Explore IDE |
    public static void increaseAvailability() {
        availableRooms++;
    }

    Trae: Explain | Doc | Test | Explore IDE |
    public static int getAvailability() {
        return availableRooms;
    }

    Trae: Explain | Doc | Test | Explore IDE |
    @Override
    public String infoDisplay() {
        return "Family Room | RM" + price + "/day | Available: " + availableRooms;
    }
```

```java
    Trae: Explain | Doc | Test | Explore IDE |
    @Override
    public String generateBookingId() {
        return "FR" + Hotel.generateSequentialBookingId();
    }
}
```

ExecutiveRoom.java

```java
public class ExecutiveRoom extends Hotel {
    private static int availableRooms = 5;

    public ExecutiveRoom() {
        super(availableRooms, price:300);
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static boolean isAvailable() {
        return availableRooms > 0;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static void reduceAvailability() {
        if (availableRooms > 0) {
            availableRooms--;
        }
    }

    // Add this method to each room class
    Trae: Explain | Doc | Test | Explore IDE | ×
    public static void increaseAvailability() {
        availableRooms++;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    public static int getAvailability() {
        return availableRooms;
    }

    Trae: Explain | Doc | Test | Explore IDE | ×
    @Override
    public String infoDisplay() {
        return "Executive Room | RM" + price + "/day | Available: " + availableRooms;
    }
```

```java
    Trae: Explain | Doc | Test | Explore IDE | ×
    @Override
    public String generateBookingId() {
        return "ER" + Hotel.generateSequentialBookingId();
    }
}
```

HotelSystem.java

```java
import java.util.*;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class HotelSystem {
    private static ArrayList<Customer> customers = new ArrayList<>(); // association & arraylist -> Customer links to a Hotel

    public static void addCustomer(Customer customer) {
        customers.add(customer);
    }

    public static void removeCustomer(String bookingId) {
        if (customers.isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"No bookings to cancel.");
            return;
        }

        for (int i = 0; i < customers.size(); i++) {
            Customer customer = customers.get(i);
            if (customer.getBookingId().equalsIgnoreCase(bookingId.trim())) {  // Check for case-insensitive match and trim whitespace
                // Add the room back to available count
                Hotel hotel = customer.getHotel();
                addRoomBack(hotel);

                // Remove the customer from the list
                customers.remove(i);
                JOptionPane.showMessageDialog(parentComponent:null, "Booking " + bookingId + " has been cancelled successfully!\nRoom has been added back to available rooms.");
                return;
            }
        }
        // If no matching booking ID is found, show this message
        JOptionPane.showMessageDialog(parentComponent:null, message:"Booking ID not found!");
    }
```

```java
    public static void addRoomBack(Hotel hotel) {
        if (hotel instanceof SingleRoom) {
            SingleRoom.increaseAvailability();
        } else if (hotel instanceof DoubleRoom) {
            DoubleRoom.increaseAvailability();
        } else if (hotel instanceof FamilyRoom) {
            FamilyRoom.increaseAvailability();
        } else if (hotel instanceof ExecutiveRoom) {
            ExecutiveRoom.increaseAvailability();
        }
    }

    public static Customer findCustomerById(String bookingId) {
        for (Customer c : customers) {
            if (c.getBookingId().equals(bookingId)) {
                return c;
            }
        }
        return null;
    }
```

```
    public static void displayAllBookings() {
        if (customers.isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"No bookings available.");
            return;
        }

        StringBuilder sb = new StringBuilder(); // Create a StringBuilder to hold all customer receipts
        for (Customer c : customers) {
            sb.append(c.getReceipt()).append(str:"\n--------------------------\n"); // Add separator for clarity
        }
        // Create a scrollable text area
        JTextArea textArea = new JTextArea(sb.toString()); //JTextArea:a component that displays multiline text.
        textArea.setEditable(b:false); // Prevent user from editing the text
        textArea.setFont(new java.awt.Font(name:"Monospaced", java.awt.Font.PLAIN, size:12)); // Set a monospaced font for better readability and looks liked printed format

        // Wrap the text area in a scroll pane to allow scrolling if the content is long
        JScrollPane scrollPane = new JScrollPane(textArea);
        scrollPane.setPreferredSize(new java.awt.Dimension(width:500, height:400));

        JOptionPane.showMessageDialog(parentComponent:null, scrollPane, title:"All Bookings", JOptionPane.INFORMATION_MESSAGE);
    }

        public static ArrayList<Customer> getCustomers() {
            return customers;
    }
}
```

HotelBookingSystem.java

```
import javax.swing.*;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class HotelBookingSystem {
    Run | Debug
    public static void main(String[] args) {
        ImageIcon hotelIcon = new ImageIcon(filename:"hotel.png");
        while (true) {
            String[] options = { "Book Room", "Cancel Booking", "View All Bookings", "Exit" };
            int choice = JOptionPane.showOptionDialog(parentComponent:null, message:"Welcome to RoyalRest Hotel Booking System", title:"Main Menu",
                    JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE, hotelIcon, options, options[0]);

            if (choice == 0) {
                bookRoom();
            } else if (choice == 1) {
                cancelBooking();
            } else if (choice == 2) {
                HotelSystem.displayAllBookings();
            } else {
                break;
            }
        }
    }

    public static void bookRoom() {
        Hotel[] allRooms = { new SingleRoom(), new DoubleRoom(), new FamilyRoom(), new ExecutiveRoom() };
        StringBuilder sb = new StringBuilder(str:"Room Types:\n");
        int i = 1;
        for (Hotel room : allRooms) {
            sb.append(i++).append(str:": ").append(room.infoDisplay()).append(str:"\n");
        }

        String[] roomTypes = { "Single Room", "Double Room", "Family Room", "Executive Room", "Return" };
        int roomChoice = JOptionPane.showOptionDialog(parentComponent:null, sb.toString(), title:"Room Selection",
                JOptionPane.DEFAULT_OPTION, JOptionPane.QUESTION_MESSAGE, icon:null, roomTypes, roomTypes[0]);

        Hotel selectedHotel = null;
```

```
switch (roomChoice) {
    case 0:
        if (!SingleRoom.isAvailable()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"No Single Rooms available!");
            return;
        }
        selectedHotel = new SingleRoom();
        SingleRoom.reduceAvailability();
        break;
    case 1:
        if (!DoubleRoom.isAvailable()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"No Double Rooms available!");
            return;
        }
        selectedHotel = new DoubleRoom();
        DoubleRoom.reduceAvailability();
        break;
    case 2:
        if (!FamilyRoom.isAvailable()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"No Family Rooms available!");
            return;
        }
        selectedHotel = new FamilyRoom();
        FamilyRoom.reduceAvailability();
        break;
    case 3:
        if (!ExecutiveRoom.isAvailable()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"No Executive Rooms available!");
            return;
        }
        selectedHotel = new ExecutiveRoom();
        ExecutiveRoom.reduceAvailability();
        break;
    default:
        return;
}
```

```
try {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern:"yyyy-MM-dd");
    LocalDate checkInDate = null;
    LocalDate checkOutDate = null;

    // Check-in date validation with loop
    while (checkInDate == null) {
        String checkInStr = JOptionPane.showInputDialog(message:"Enter check-in date (YYYY-MM-DD):");
        if (checkInStr == null) { // User clicked Cancel
            restoreRoomAvailability(selectedHotel);
            return;
        }
        if (checkInStr.trim().isEmpty()) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Check-in date cannot be empty! Please try again.");
            continue;
        }

        try {
            checkInDate = LocalDate.parse(checkInStr, formatter);
            if (checkInDate.isBefore(LocalDate.now())) {
                JOptionPane.showMessageDialog(parentComponent:null, message:"Check-In date cannot be in the past! Please enter a valid date.");
                checkInDate = null; // Reset to retry
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Invalid date format! Please use YYYY-MM-DD format.");
        }
    }
```

```
// Check-out date validation with loop
while (checkOutDate == null) {
    String checkOutStr = JOptionPane.showInputDialog(message:"Enter check-out date (YYYY-MM-DD):");
    if (checkOutStr == null) { // User clicked Cancel
        restoreRoomAvailability(selectedHotel);
        return;
    }
    if (checkOutStr.trim().isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Check-out date cannot be empty! Please try again.");
        continue;
    }

    try {
        checkOutDate = LocalDate.parse(checkOutStr, formatter);

        if (checkOutDate.isBefore(LocalDate.now())) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Check-Out date cannot be in the past! Please enter a valid date.");
            checkOutDate = null;
            continue;
        }

        if (checkInDate.equals(checkOutDate)) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Check-In and Check-Out dates cannot be the same! Please enter a different date.");
            checkOutDate = null;
            continue;
        }

        if (checkOutDate.isBefore(checkInDate)) {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Check-Out date cannot be before Check-In date! Please enter a valid date.");
            checkOutDate = null;
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"Invalid date format! Please use YYYY-MM-DD format.");
    }
}
```

```
// Set check-in & check-out
selectedHotel.setCheckIn(checkInDate.atStartOfDay());
selectedHotel.setCheckOut(checkOutDate.atStartOfDay());

// Name validation with loop
String name = null;
while (name == null) {
    try {
        String nameInput = JOptionPane.showInputDialog(message:"Enter your name:");
        if (nameInput == null) { // User clicked Cancel
            restoreRoomAvailability(selectedHotel);
            return;
        }

        // Check if empty
        if (nameInput.trim().isEmpty()) {
            throw new IllegalArgumentException(s:"Name cannot be empty!");
        }

        // Check if name contains only letters and spaces
        if (!nameInput.trim().matches(regex:"[a-zA-Z\\s]+")) {
            throw new IllegalArgumentException(s:"Name can only contain letters and spaces!");
        }

        // Check if name is too short
        if (nameInput.trim().length() < 3) {
            throw new IllegalArgumentException(s:"Name must be at least 3 characters long!");
        }

        name = nameInput.trim();

    } catch (IllegalArgumentException e) {
        JOptionPane.showMessageDialog(parentComponent:null, e.getMessage() + " Please enter your name again.");
    }
}
```

```java
// IC validation with exception handling
String ic = null;
while (ic == null) {
    try {
        String icInput = JOptionPane.showInputDialog(message:"Enter your IC/Identity Number:");
        if (icInput == null) { // User clicked Cancel
            restoreRoomAvailability(selectedHotel);
            return;
        }

        // Check if empty
        if (icInput.trim().isEmpty()) {
            throw new IllegalArgumentException(s:"IC/Identity Number cannot be empty!");
        }

        // Check if IC contains only numbers
        if (!icInput.trim().matches(regex:"[0-9]+")) {
            throw new IllegalArgumentException(s:"IC/Identity Number can only contain numbers!");
        }

        // Check if IC has exactly 12 characters
        if (icInput.trim().length() != 12) {
            throw new IllegalArgumentException(s:"IC/Identity Number must be exactly 12 digits!");
        }

        ic = icInput.trim();

    } catch (IllegalArgumentException e) {
        JOptionPane.showMessageDialog(parentComponent:null, e.getMessage() + " Please enter your IC again.");
    }
}
```

```java
// Phone validation with loop
String phone = null;
while (phone == null) {
    try{
    String phoneInput = JOptionPane.showInputDialog(message:"Enter your phone number:");
    if (phoneInput == null) { // User clicked Cancel
        restoreRoomAvailability(selectedHotel);
        return;
    }
    // Check if empty
        if (phoneInput.trim().isEmpty()) {
            throw new IllegalArgumentException(s:"Phone number cannot be empty!");
        }

        // Check if phone number contains only numbers
        if (!phoneInput.trim().matches(regex:"[0-9]+")) {
            throw new IllegalArgumentException(s:"Phone number can only contain numbers!");
        }

        // Add length validation for Malaysian phone numbers
        if (phoneInput.trim().length() < 10) {
            throw new IllegalArgumentException(s:"Phone number must be at least 10 digits!");
        }

        if (phoneInput.trim().length() > 11) {
            throw new IllegalArgumentException(s:"Phone number cannot be more than 11 digits!");
        }

        phone = phoneInput.trim();

    } catch (IllegalArgumentException e) {
        JOptionPane.showMessageDialog(parentComponent:null, e.getMessage() + " Please enter your phone number again.");
    }
}
```

```
        // Payment method selection
        String[] payments = { "Online (Full Payment)", "Offline (50% Deposit)" };
        int payChoice = JOptionPane.showOptionDialog(parentComponent:null, message:"Select Payment Method:", title:"Payment",
                JOptionPane.DEFAULT_OPTION, JOptionPane.QUESTION_MESSAGE, icon:null, payments, payments[0]);

        if (payChoice == -1) { // User clicked X or Cancel
            JOptionPane.showMessageDialog(parentComponent:null, message:"Payment method must be selected!");
            restoreRoomAvailability(selectedHotel);
            return;
        }

        Customer customer = new Customer(name, ic, phone, selectedHotel);
        Booking method = (payChoice == 0) ? new Online() : new Offline();
        customer.placeBooking(selectedHotel, method);
        HotelSystem.addCustomer(customer);
        JOptionPane.showMessageDialog(parentComponent:null, customer.getReceipt());

    } catch (Exception e) {
        JOptionPane.showMessageDialog(parentComponent:null, message:"An unexpected error occurred! Please try again.");
        restoreRoomAvailability(selectedHotel);
    }
}

private static void restoreRoomAvailability(Hotel selectedHotel) {
    if (selectedHotel instanceof SingleRoom) {
        SingleRoom.increaseAvailability();
    } else if (selectedHotel instanceof DoubleRoom) {
        DoubleRoom.increaseAvailability();
    } else if (selectedHotel instanceof FamilyRoom) {
        FamilyRoom.increaseAvailability();
    } else if (selectedHotel instanceof ExecutiveRoom) {
        ExecutiveRoom.increaseAvailability();
    }
}
```

```
    public static void cancelBooking() {
        String bookingId = JOptionPane.showInputDialog(message:"Enter Booking ID to cancel:");
        if (bookingId != null && !bookingId.trim().isEmpty()) {
            HotelSystem.removeCustomer(bookingId.trim());
        } else {
            JOptionPane.showMessageDialog(parentComponent:null, message:"Booking ID cannot be empty!");
        }
    }
}
```