Hotel Room Management System

A Simple and Interactive Hotel Booking System

Team Member

Tarek Hossam eldin	2200595	System Logic
Mohamed Samy Abdelfattah Ali	2200685	Error Handling and system Logic
Ahmed Mohamed Mohyeldin	2200955	GUI
Abdelrahman Ahmed Shawky	2200585	GUI

Project allows users to:

01. View Available Rooms:
- See details of rooms like room ID, type, price, and capacity.

02. Select Check-In and Check-Out Dates:
- Choose the dates they plan to stay at the hotel

03. Book Rooms:
- Select a room based on availability and confirm the booking.

Project Structure

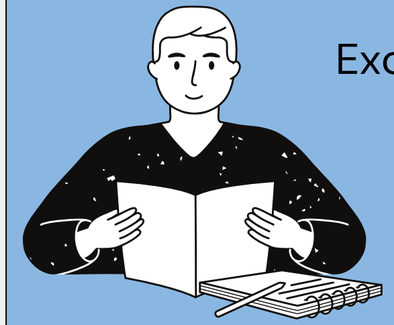
Exploring creativity

UML

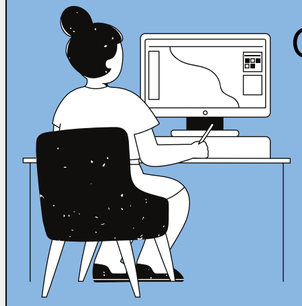
Encapsulation

Abstract

Polymorphism

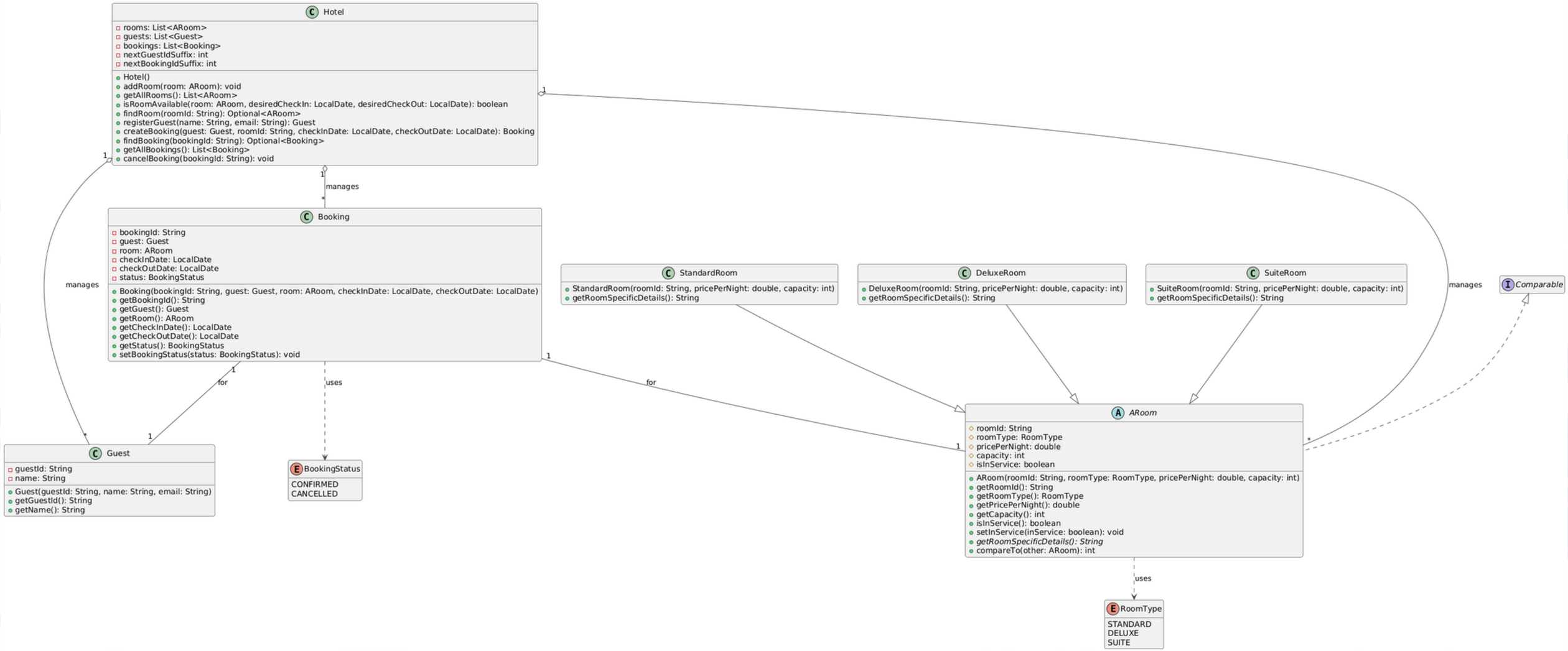


Exception handling



Graphic User Interface

UML


Hotel
rooms: List<ARoom>
guests: List<Guest>
bookings: List<Booking>
nextGuestIdSuffix: int
nextBookingIdSuffix: int
Hotel()
addRoom(room: ARoom): void
getAllRooms(): List<ARoom>
isRoomAvailable(room: ARoom, desiredCheckIn: LocalDate, desiredCheckOut: LocalDate): boolean
findRoom(roomId: String): Optional<ARoom>
registerGuest(name: String, email: String): Guest
createBooking(guest: Guest, roomId: String, checkInDate: LocalDate, checkOutDate: LocalDate): Booking
findBooking(bookingId: String): Optional<Booking>
getAllBookings(): List<Booking>
cancelBooking(bookingId: String): void
manages
Booking
bookingId: String
guest: Guest
room: ARoom
checkInDate: LocalDate
checkOutDate: LocalDate
status: BookingStatus
Booking(bookingId: String, guest: Guest, room: ARoom, checkInDate: LocalDate, checkOutDate: LocalDate)
getBookingId(): String
getGuest(): Guest
getRoom(): ARoom
getCheckInDate(): LocalDate
getCheckOutDate(): LocalDate
getStatus(): BookingStatus
setBookingStatus(status: BookingStatus): void
StandardRoom
StandardRoom(roomId: String, pricePerNight: double, capacity: int)
getRoomSpecificDetails(): String
DeluxeRoom
DeluxeRoom(roomId: String, pricePerNight: double, capacity: int)
getRoomSpecificDetails(): String
SuiteRoom
SuiteRoom(roomId: String, pricePerNight: double, capacity: int)
getRoomSpecificDetails(): String
Comparable
manages
for
uses
Guest
guestId: String
name: String
Guest(guestId: String, name: String, email: String)
getGuestId(): String
getName(): String
BookingStatus
CONFIRMED
CANCELLED
ARoom
roomId: String
roomType: RoomType
pricePerNight: double
capacity: int
isInService: boolean
ARoom(roomId: String, roomType: RoomType, pricePerNight: double, capacity: int)
getRoomId(): String
getRoomType(): RoomType
getPricePerNight(): double
getCapacity(): int
isInService(): boolean
setInService(inService: boolean): void
getRoomSpecificDetails(): String
compareTo(other: ARoom): int
RoomType
STANDARD
DELUXE
SUITE

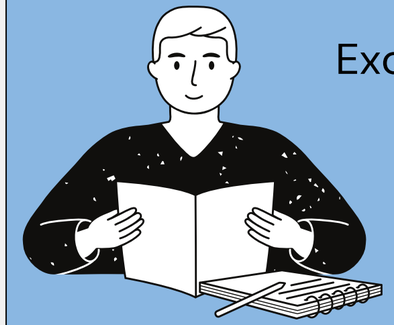
Project Structure

Exploring creativity

UML

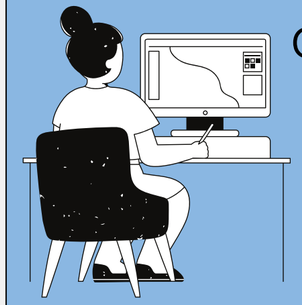
Polymorphism

Encapsulation



Exception handling

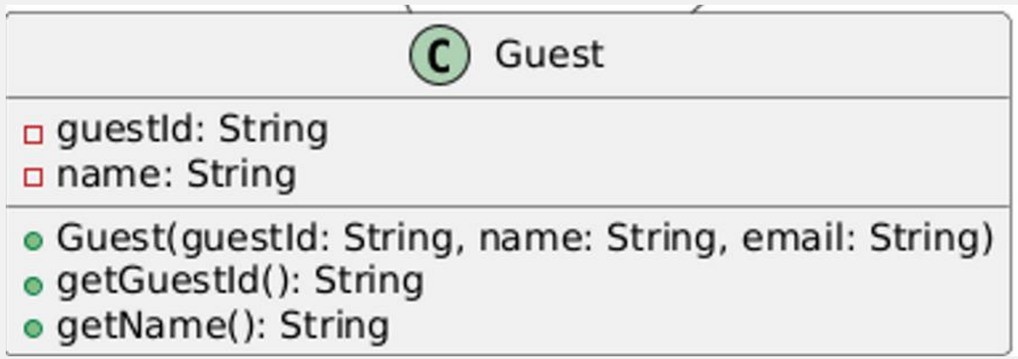
Abstract



Graphic User Interface

1. Encapsulation

Encapsulation means that an object's internal data (variables) is hidden from the outside world, and the only way to interact with that data is through specific methods (Getter and Setters).




```java
class Guest {
    private String guestId;
    private String name;
    public Guest(String guestId, String name, String email) {
        this.guestId = guestId;
        this.name = name;
    }
    public String getGuestId() { return guestId; }
    public String getName() { return name; }
```

Project Structure

Exploring creativity

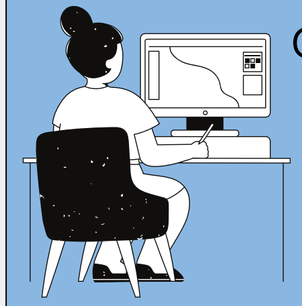
UML

Encapsulation

Abstract

Polymorphism

Exception handling



Graphic User Interface

2 .Abstract

Abstract classes in the Hotel Room Management System serve as a blueprint for other classes, defining common attributes and methods while leaving specific implementations to subclasses.


```java
abstract class ARoom implements Comparable<ARoom> {
    protected String roomId;
    protected RoomType roomType;
    protected double pricePerNight;
    protected int capacity;
    protected boolean isInService;

    public ARoom(String roomId, RoomType roomType, double pricePerNight, int capacity) {
        this.roomId = roomId;
        this.roomType = roomType;
        this.pricePerNight = pricePerNight;
        this.capacity = capacity;
        this.isInService = true;
    }

    public String getRoomId() { return roomId; }
    public RoomType getRoomType() { return roomType; }
    public double getPricePerNight() { return pricePerNight; }
    public int getCapacity() { return capacity; }
    public boolean isInService() { return isInService; }
    public void setInService(boolean inService) { this.isInService = inService; }
```

```java
// -------------- CONCRETE ROOM SUBCLASSES --------------
class StandardRoom extends ARoom {
    public StandardRoom(String roomId, double pricePerNight, int capacity) {
        super(roomId, RoomType.STANDARD, pricePerNight, capacity);
    }
    @Override public String getRoomSpecificDetails() { return "Standard Room features: Basic and comfortable accommodation."; }
}
```

Project Structure

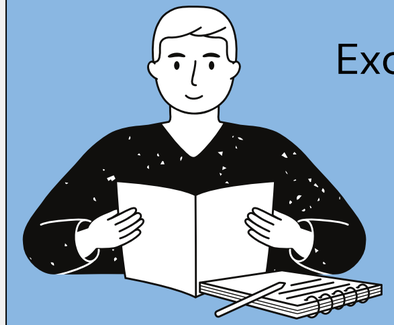
Exploring creativity

UML

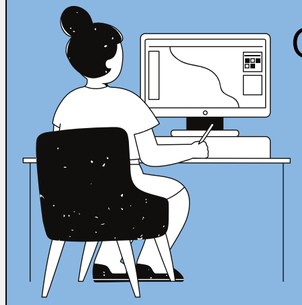
Encapsulation

Abstract

Polymorphism



Exception handling



Graphic User Interface

3.polymorphism

polymorphism in the Hotel Room Management System allows different room types (like StandardRoom, DeluxeRoom, SuiteRoom) to be treated as instances of the ARoom class, enabling the same method (getRoomSpecificDetails()) to behave differently based on the object type.

```
public abstract String getRoomSpecificDetails(); // Example of polymorphism
```

```
class StandardRoom extends ARoom {
    public StandardRoom(String roomId, double pricePerNight, int capacity) {
        super(roomId, RoomType.STANDARD, pricePerNight, capacity);
    }
    @Override public String getRoomSpecificDetails() { return "Standard Room features: Basic and comfortable accommodation."; }
}

class DeluxeRoom extends ARoom {
    public DeluxeRoom(String roomId, double pricePerNight, int capacity) { super(roomId, RoomType.DELUXE, pricePerNight, capacity); }
    @Override public String getRoomSpecificDetails() { return "Deluxe Room features: Enhanced amenities and more spacious."; }
}
```

Even though the rooms are different types (StandardRoom) they are all treated as ARoom objects a heading

```
public void addRoom(ARoom room)
```

```
private void populateInitialData() {
    hotelManager.addRoom(new StandardRoom("S101", 75.00, 2));
    hotelManager.addRoom(new DeluxeRoom("D201", 120.00, 2));
    hotelManager.addRoom(new SuiteRoom("U301", 250.00, 4));
```

Sorting

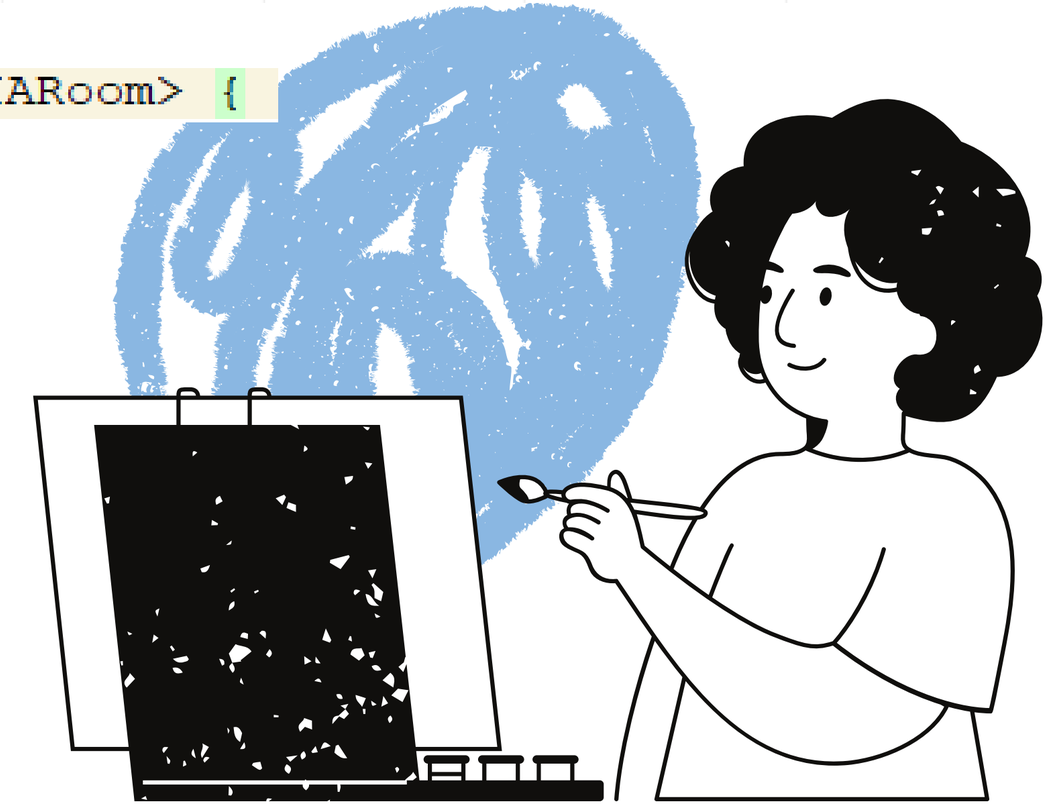
rearranging the rooms by their ID

D201	DELUXE	120.00	2	Select Dates
D202	DELUXE	125.00	3	Select Dates
S101	STANDARD	75.00	2	Select Dates
S102	STANDARD	80.00	1	Select Dates
S103	STANDARD	75.00	2	Select Dates
U301	SUITE	250.00	4	Select Dates

- ```
  abstract class ARoom implements Comparable<ARoom> {
  ```

```
FXCollections.sort(roomObservableList);
```

```
@Override
public int compareTo(ARoom other) {
    // Natural sort order is by Room ID
    if (other == null) return 1; // Basic null check
    return this.roomId.compareTo(other.roomId);
}
```



rooms are sorted by their roomId using FXCollections.sort(roomObservableList. This works because ARoom implements Comparable and defines sorting based on roomId alphabetically.
```

Project Structure

Exploring creativity

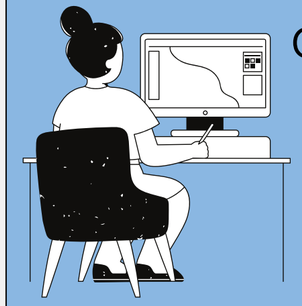
UML

Encapsulation

Abstract

Polymorphism

Exception handling



Graphic User Interface

4.Exception handling

Date Validation

```
// Validate selected dates
if (desiredCheckIn == null || desiredCheckOut == null) {
    showAlert(Alert.AlertType.ERROR, "Date Missing", "Please select both check-in and check-out dates for booking.");
    return;
}
if (!desiredCheckOut.isAfter(desiredCheckIn)) {
    showAlert(Alert.AlertType.ERROR, "Date Error", "Check-out date must be after check-in date.");
    return;
}
```

Booking Exception handling

```
        try {
            hotelManager.createBooking(currentGuest, selectedRoom.getRoomId(), desiredCheckIn, desiredCheckOut);
            refreshRoomList(); // Refresh table to update availability status
            bookingStage.close();
        } catch (IllegalArgumentException | IllegalStateException ex) {
            showAlert(Alert.AlertType.ERROR, "Booking Failed", ex.getMessage());
        }
```

```
if (guest == null || roomId == null || checkInDate == null || checkOutDate == null) {
    throw new IllegalArgumentException("All parameters for booking must be provided.");
}
if (!checkOutDate.isAfter(checkInDate)) {
    throw new IllegalArgumentException("Check-out date must be after check-in date.");
}
```

Project Structure

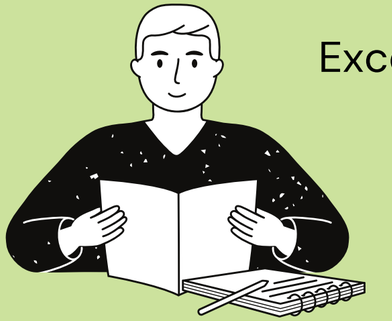
Exploring creativity

UML

Polymorphism

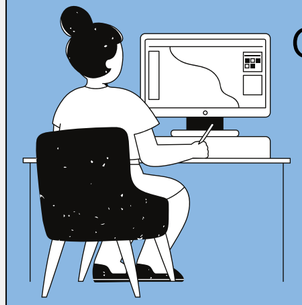
Encapsulation

Exception handling



Abstract

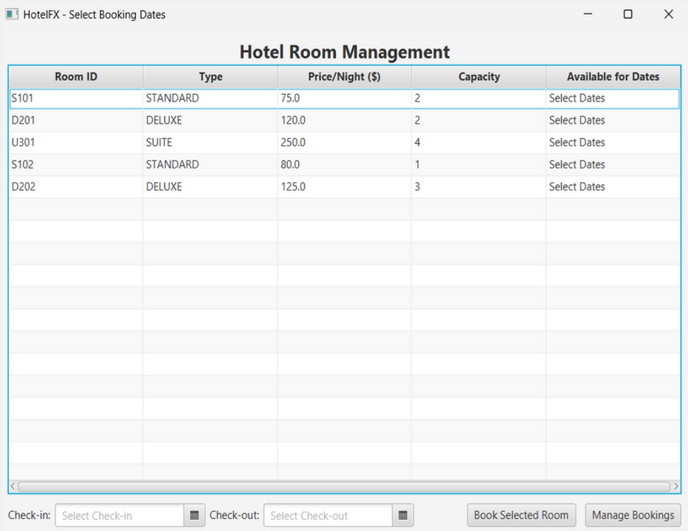
Graphic User Interface



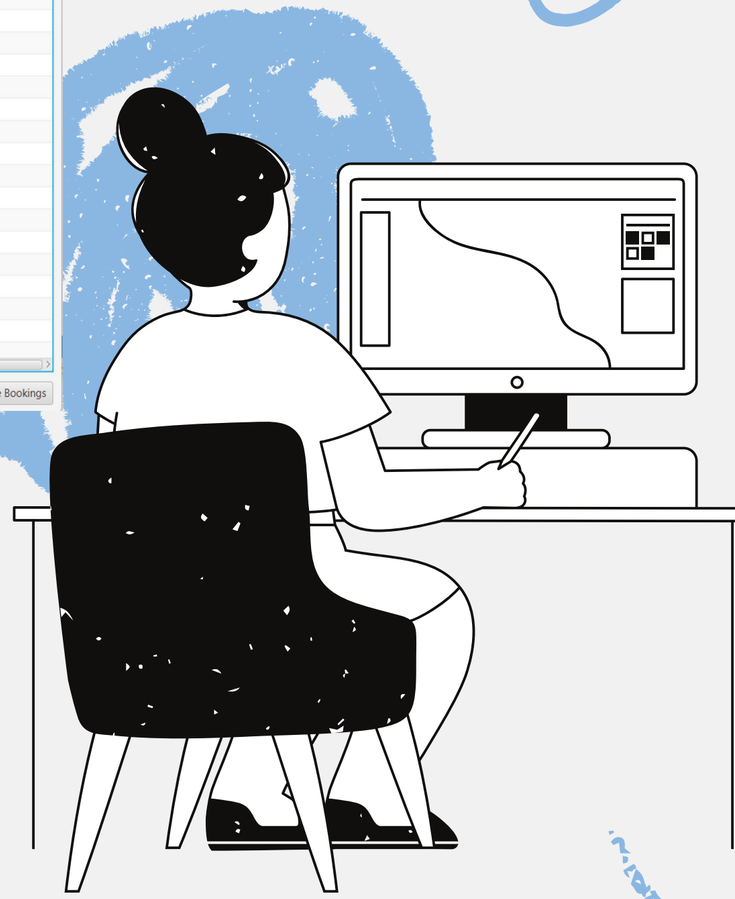
5.GUI

```java
// --- Top Pane (Title only) ---
VBox topPane = new VBox(10); // Spacing if you add more elements later
topPane.setAlignment(Pos.CENTER); // Center the title

Label titleLabel = new Label("Hotel Room Management");
titleLabel.setStyle("-fx-font-size: 20px; -fx-font-weight: bold;");
topPane.getChildren().add(titleLabel); // Only add title
mainLayout.setTop(topPane);
```



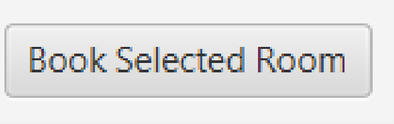

```java
private void populateInitialData() {
    hotelManager.addRoom(new StandardRoom("S101", 75.00, 2));
    hotelManager.addRoom(new DeluxeRoom("D201", 120.00, 2));
    hotelManager.addRoom(new SuiteRoom("U301", 250.00, 4));
    // Add rooms out of order to see sorting work
    hotelManager.addRoom(new StandardRoom("S103", 75.00, 2));
    hotelManager.addRoom(new StandardRoom("S102", 80.00, 1));
    hotelManager.addRoom(new DeluxeRoom("D202", 125.00, 3));
}
```



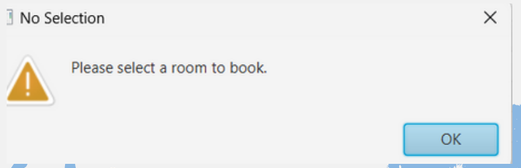
```java
populateInitialData();
refreshRoomList(); // This will populate and sort the list by Room ID
```

```java
private void refreshRoomList() {
    roomObservableList.setAll(hotelManager.getAllRooms()); // Get all rooms
    // Always sort by the natural order defined in ARoom.compareTo() (which is Room ID)
    FXCollections.sort(roomObservableList);
    roomTableView.refresh(); // Ensure visual update for all cells
}
```

```java
Button bookSelectedButton = new Button("Book Selected Room");
bookSelectedButton.setOnAction(e -> handleBookSelectedRoom());
```

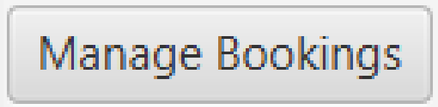



```java
private void handleBookSelectedRoom() {
    ARoom selectedRoom = roomTableView.getSelectionModel().getSelectedItem();
    if (selectedRoom == null) {
        showAlert(Alert.AlertType.WARNING, "No Selection", "Please select a room to book.");
        return;
    }
```

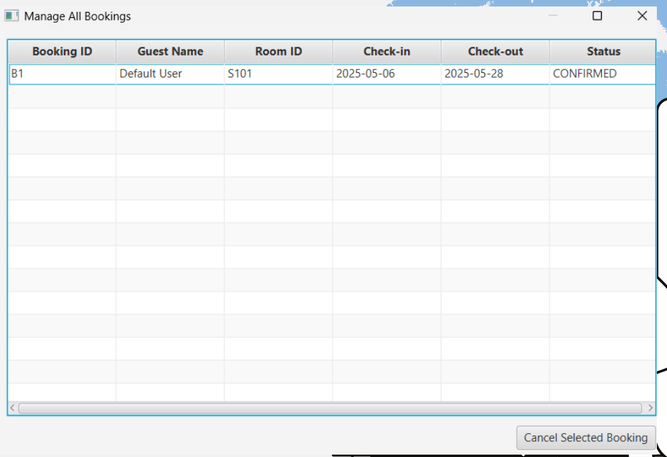


5.More GUI

```java
Button manageBookingsButton = new Button("Manage Bookings");
manageBookingsButton.setOnAction(e -> handleManageBookings(primaryStage));
```



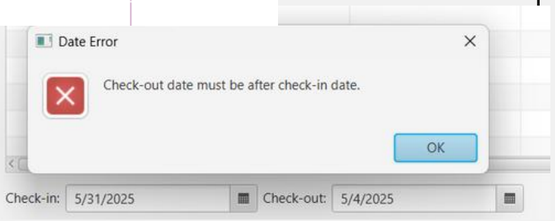

```java
private void handleManageBookings(Stage ownerStage) {
    // Create and Setup window
    Stage bookingsStage = new Stage();
    bookingsStage.initModality(Modality.WINDOW_MODAL); //can't interact with the main window until this one is closed.
    bookingsStage.initOwner(ownerStage); // tell that is the main window
    bookingsStage.setTitle("Manage All Bookings");  // set title
    // Bookings Window
    BorderPane bookingsLayout = new BorderPane();
    bookingsLayout.setPadding(new Insets(10));
    //Creating the Bookings Table
    TableView<Booking> bookingsTableView = new TableView<>();
    // Bookings don't have a natural sort order in this example, displayed as added
    //Populate the Table with Bookings
    ObservableList<Booking> bookingsObservableList = FXCollections.observableArrayList(hotelManager.getAllBookings());
    bookingsTableView.setItems(bookingsObservableList);
    //Create and Add Columns to the Table
```



Show Alert Implementation

```java
private void showAlert(Alert.AlertType alertType, String title, String message) {
    Alert alert = new Alert(alertType);
    alert.setTitle(title);
    alert.setHeaderText(null); // No header text
    alert.setContentText(message);
    alert.showAndWait();
}
```

```java
if (!desiredCheckOut.isAfter(desiredCheckIn)) {
    showAlert(Alert.AlertType.ERROR, "Date Error", "Check-out date must be after check-in date.");
    return;
}
```




```java
Button cancelSelectedBookingButton = new Button("Cancel Selected Booking");
cancelSelectedBookingButton.setOnAction(e -> {
    Booking selectedBooking = bookingsTableView.getSelectionModel().getSelectedItem();
    if (selectedBooking == null) {
        showAlert(Alert.AlertType.WARNING, "No Selection", "Please select a booking to cancel.");
        return;
```

Project Structure

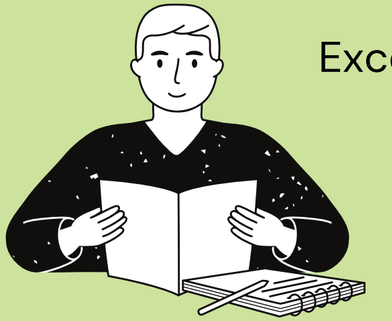
Done

UML

Polymorphism

Encapsulation

Exception handling



Abstract

Graphic User Interface

Live Demo

Thank you very much!

Q&A