```
import java.util.*;

class UniqueRoom {
    private int uniqueRoomNumber;
    private String uniqueRoomType;
    private boolean uniqueWifiAvailability;
    private boolean uniqueOccupancyStatus;
    private double uniqueRoomRate;

    public UniqueRoom(int roomNumber, String roomType, boolean wifiAvailability, double
roomRate) {
        this.uniqueRoomNumber = roomNumber;
        this.uniqueRoomType = roomType;
        this.uniqueWifiAvailability = wifiAvailability;
        this.uniqueRoomRate = roomRate;
        this.uniqueOccupancyStatus = false;
    }

    public int getUniqueRoomNumber() {
        return uniqueRoomNumber;
    }

    public String getUniqueRoomType() {
        return uniqueRoomType;
    }

    public boolean isUniqueWifiAvailable() {
        return uniqueWifiAvailability;
    }

    public boolean isUniqueOccupied() {
        return uniqueOccupancyStatus;
    }

    public double getUniqueRoomRate() {
        return uniqueRoomRate;
    }

    public void occupyRoom() {
```

```
        uniqueOccupancyStatus = true;
    }

    public void vacateRoom() {
        uniqueOccupancyStatus = false;
    }
}
```

This piece of code introduces a Java class named UniqueRoom, representing a distinctive hotel room.

Within the UniqueRoom class, private instance variables are utilized to encapsulate various attributes peculiar to a room:

- **int uniqueRoomNumber: Represents the exclusive room identifier.**
- **String uniqueRoomType: Captures the unique room classification (e.g., Elite Suite, Signature, Premier).**
- **boolean uniqueWifiAvailability: Indicates the availability of an exclusive Wi-Fi connection in the room (true or false).**
- **boolean uniqueOccupancyStatus: Signifies the occupancy status of the room (true if currently occupied, false if vacant).**

double uniqueRoomRate: Records the bespoke room rate per day.
Upon instantiation of a UniqueRoom object, a constructor (public UniqueRoom(int uniqueRoomNumber, String uniqueRoomType, boolean uniqueWifiAvailability, double uniqueRoomRate)) initializes these distinctive attributes, receiving parameters such as uniqueRoomNumber, uniqueRoomType, uniqueWifiAvailability, and uniqueRoomRate to set the initial values of the instance variables.
Getter methods (retrieveUniqueRoomNumber(), fetchUniqueRoomType(), isUniqueWifiAvailable(), isUniqueOccupied(), and obtainUniqueRoomRate()) are thoughtfully provided to facilitate the retrieval of these private instance variables from external sources. These methods foster encapsulation by enabling attribute access without direct field manipulation.
The occupyRoom() method facilitates the transition of a unique room into an occupied state by setting the uniqueOccupancyStatus flag to true.

- **Similarly, the vacateRoom() method orchestrates the transition of a unique room into a vacant state by resetting the uniqueOccupancyStatus flag to false.**

```
package secret;

import java.util.*;


class ExclusiveGuest {

    private String specialName;

    private String specialContact;

    private int specialRoomNumber;

    private int specialNumberOfDays;


    public ExclusiveGuest(String specialName, String specialContact, int
specialRoomNumber, int specialNumberOfDays) {

        this.specialName = specialName;

        this.specialContact = specialContact;

        this.specialRoomNumber = specialRoomNumber;

        this.specialNumberOfDays = specialNumberOfDays;

    }


    public String retrieveSpecialName() {
```

```
        return specialName;
    }

    public String fetchSpecialContact() {
        return specialContact;
    }

    public int obtainSpecialRoomNumber() {
        return specialRoomNumber;
    }

    public int grabSpecialNumberOfDays() {
        return specialNumberOfDays;
    }
}
```

The Guest class holds onto important details about our visitors:

- **The name stores their name.**
- **Contact keeps their phone number or email.**
- **Room number shows where they'll stay.**
- **Number of days tells how long they're booked for.**

When we make a new Guest, we use a special method called a constructor. It sets up all these details like name, contact, room number, and number of days.

To check on a guest's details, we've got handy methods like getName(), getContact(), getRoomNumber(), and getNumberOfDays(). They let us peek into the guest's info without messing with their privacy. It's like keeping their secrets safe!

```
import java.util.*;


class Hotel {
    private List<Room> rooms;
    private Map<String, Guest> reservations;

    public Hotel() {
        rooms = new ArrayList<>();
        reservations = new HashMap<>();
    }

    public void addRoom(Room room) {
        rooms.add(room);
    }

    public void viewRoomAvailability() {
        for (Room room : rooms) {
```

```
            System.out.println("Room " + room.getRoomNumber() + ": " + (room.isOccupied()
? "Occupied" : "Vacant"));

        }

    }



    public boolean makeReservation(String name, String contact, String roomType, int
numberOfDays) {

        for (Room room : rooms) {

            if (!room.isOccupied() && room.getRoomType().equalsIgnoreCase(roomType)) {

                room.occupy();

                int roomNumber = room.getRoomNumber();

                Guest guest = new Guest(name, contact, roomNumber, numberOfDays);

                reservations.put(contact, guest);

                double roomCharge = room.getRoomRate() * numberOfDays;

                System.out.println("Reservation confirmed. Room Charge: $" + roomCharge);

                return true;

            }

        }

        System.out.println("No rooms available of type " + roomType);

        return false;

    }
```

```
    public boolean cancelReservation(String contact) {
        if (reservations.containsKey(contact)) {
            Guest guest = reservations.get(contact);
            int roomNumber = guest.getRoomNumber();
            for (Room room : rooms) {
                if (room.getRoomNumber() == roomNumber) {
                    room.vacate();
                    reservations.remove(contact);
                    System.out.println("Reservation for " + guest.getName() + " canceled.");
                    return true;
                }
            }
        }
        System.out.println("No reservation found for contact: " + contact);
        return false;
    }
}
```

The Hotel class manages room reservations and availability. It contains two important parts:

Room Storage: It keeps track of rooms using a list called rooms and reservations using a map named reservations.
Functionalities:

- **Adding a Room: This feature allows adding a new room to the hotel.**
- **Viewing Room Availability: It shows whether each room is occupied or vacant.**
- **Making a Reservation: Customers can reserve a room by providing their name, contact, desired room type, and the number of days for their stay. The system finds an available room matching the requested type and marks it as occupied. It then creates a reservation for the guest and calculates the room charge based on the rate and duration of stay.**
- **Canceling a Reservation: Guests can cancel their reservation by providing their contact information. If the reservation exists, the associated room is marked as vacant, and the reservation is removed. Otherwise, a message is displayed indicating that the reservation was not found.**

These features ensure efficient management of hotel bookings and provide a seamless experience for both guests and hotel staff.

```
import java.util.Scanner;



public class UniqueHotelReservationSystem {

    public static void main(String[] args) {

        Room room1 = new Room(101, "Super Luxury", true, 150);

        Room room2 = new Room(102, "Deluxe", true, 100);

        Room room3 = new Room(103, "Deluxe", true, 100);

        Room room4 = new Room(201, "Super Suite", false, 200);

        Room room5 = new Room(202, "Super Suite", false, 200);
```

```
Hotel hotel = new Hotel();
hotel.addRoom(room1);
hotel.addRoom(room2);
hotel.addRoom(room3);
hotel.addRoom(room4);
hotel.addRoom(room5);

Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.println("\nWelcome to the Unique Hotel Reservation System:");
    System.out.println("1. View Room Availability");
    System.out.println("2. Make a Reservation");
    System.out.println("3. Cancel a Reservation");
    System.out.println("4. Exit");
    System.out.print("Enter your choice: ");

    int choice = scanner.nextInt();
    scanner.nextLine(); // Consume newline
```

```
        switch (choice) {
            case 1:
                hotel.showAvailableRooms();
                break;
            case 2:
                System.out.print("Enter your name: ");
                String name = scanner.nextLine();
                System.out.print("Enter your contact: ");
                String contact = scanner.nextLine();
                System.out.print("Enter desired room type (Super Luxury/Deluxe/Super
Suite): ");
                String roomType = scanner.nextLine();
                System.out.print("Enter number of days: ");
                int numberOfDays = scanner.nextInt();
                hotel.reserveRoom(name, contact, roomType, numberOfDays);
                break;
            case 3:
                System.out.print("Enter your contact to cancel the reservation: ");
                String cancelContact = scanner.nextLine();
                hotel.cancelReservation(cancelContact);
                break;
```

```
            case 4:

                System.out.println("Exiting the Unique Hotel Reservation System. Thank you
for choosing us!");

                scanner.close();

                System.exit(0);

            default:

                System.out.println("Invalid selection. Please choose a valid option.");

        }

    }

}
```

This code introduces the UniqueHotelReservationSystem class, which serves as the central hub for managing hotel reservations.

Within its main function, it initializes Room and Hotel objects to establish the system's starting point.

Five distinct Room instances (room1, room2, room3, room4, room5) are crafted, each representing a unique room within the hotel, complete with its number, type, Wi-Fi availability, and pricing.

A Hotel object named 'hotel' is instantiated to manage these rooms, and they are integrated into the hotel using the addRoom function.

A Scanner object named 'scanner' is employed to gather user inputs via the console.

A perpetual loop is established to present the primary menu and handle user interactions throughout the hotel reservation system.

Within this loop, a menu with four selections is exhibited:

1. Display Room Availability

2. Reserve a Room

3. Cancel a Reservation

4. Exit

User input, interpreted as an integer, is gathered using scanner.nextInt(), with the following newline character consumed using scanner.nextLine().

A switch statement is employed to execute actions in response to the user's selection:

Option 1 (Display Room Availability): Invokes the hotel's viewRoomAvailability method to exhibit the availability status of each room.

Option 2 (Reserve a Room): Prompts the user to provide their name, contact details, desired room type, and reservation duration. Subsequently, it calls the hotel's reserveRoom method to secure a reservation.

Option 3 (Cancel a Reservation): Requests the user's contact information and triggers the hotel's cancelReservation method to annul an existing reservation.

Option 4 (Exit): Concludes the program execution, closing the Scanner and concluding the application.

In the event of an invalid selection, an error message is presented, and the loop persists.

The system remains operational within the loop until the user elects to exit (Option 4).

Final:

```java
import java.util.*;



class Accommodation {

   private int roomNumber;

   private String roomType;

   private boolean wifiIncluded;

   private boolean isBooked;

   private double roomRate;



   public Accommodation(int number, String type, boolean wifi, double rate) {

       roomNumber = number;

       roomType = type;

       wifiIncluded = wifi;

       roomRate = rate;

       isBooked = false;

   }



   public int getRoomNumber() {

       return roomNumber;

   }



   public String getRoomType() {

       return roomType;
```

```
    }

    public boolean hasWifi() {
        return wifiIncluded;
    }

    public boolean isBooked() {
        return isBooked;
    }

    public double getRoomRate() {
        return roomRate;
    }

    public void book() {
        isBooked = true;
    }

    public void unbook() {
        isBooked = false;
    }
}
```

```
class Visitor {

   private String name;

   private String contact;

   private int roomNumber;

   private int daysStayed;



   public Visitor(String name, String contact, int roomNumber, int daysStayed)
{

       this.name = name;

       this.contact = contact;

       this.roomNumber = roomNumber;

       this.daysStayed = daysStayed;

   }



   public String getName() {

       return name;

   }



   public String getContact() {

       return contact;

   }



   public int getRoomNumber() {

       return roomNumber;
```

```
    }

    public int getDaysStayed() {
        return daysStayed;
    }
}

class HotelManager {
    private List<Accommodation> accommodations;
    private Map<String, Visitor> reservations;

    public HotelManager() {
        accommodations = new ArrayList<>();
        reservations = new HashMap<>();
    }

    public void addAccommodation(Accommodation accommodation) {
        accommodations.add(accommodation);
    }

    public void viewAvailability() {
        for (Accommodation room : accommodations) {
            System.out.println("Room " + room.getRoomNumber() + ": " +
(room.isBooked() ? "Booked" : "Available"));
```

```
        }

    }



    public boolean makeReservation(String name, String contact, String roomType,
int daysStayed) {

        for (Accommodation room : accommodations) {

            if (!room.isBooked() &&
room.getRoomType().equalsIgnoreCase(roomType)) {

                room.book();

                int roomNumber = room.getRoomNumber();

                Visitor visitor = new Visitor(name, contact, roomNumber,
daysStayed);

                reservations.put(contact, visitor);

                double roomCharge = room.getRoomRate() * daysStayed;

                System.out.println("Reservation successful. Total charge: $" +
roomCharge);

                return true;

            }

        }

        System.out.println("No available rooms of type " + roomType);

        return false;

    }



    public boolean cancelReservation(String contact) {

        if (reservations.containsKey(contact)) {
```

```
            Visitor visitor = reservations.get(contact);

            int roomNumber = visitor.getRoomNumber();

            for (Accommodation room : accommodations) {

                if (room.getRoomNumber() == roomNumber) {

                    room.unbook();

                    reservations.remove(contact);

                    System.out.println("Reservation for " + visitor.getName() +
" canceled.");

                    return true;

                }

            }

        }

        System.out.println("Reservation not found for contact: " + contact);

        return false;

    }

}



public class HotelSystem {

    public static void main(String[] args) {

        Accommodation room1 = new Accommodation(101, "Deluxe Suite", true, 200);

        Accommodation room2 = new Accommodation(102, "Standard Room", true,
150);

        Accommodation room3 = new Accommodation(103, "Standard Room", true,
150);
```

```
        Accommodation room4 = new Accommodation(201, "Luxury Suite", false,
300);

        Accommodation room5 = new Accommodation(202, "Deluxe Suite", false,
250);



        HotelManager manager = new HotelManager();

        manager.addAccommodation(room1);

        manager.addAccommodation(room2);

        manager.addAccommodation(room3);

        manager.addAccommodation(room4);

        manager.addAccommodation(room5);



        Scanner scanner = new Scanner(System.in);



        while (true) {

            System.out.println("\nHotel Management System Menu:");

            System.out.println("1. View Room Availability");

            System.out.println("2. Make a Reservation");

            System.out.println("3. Cancel a Reservation");

            System.out.println("4. Exit");

            System.out.print("Enter your choice: ");



            int choice = scanner.nextInt();

            scanner.nextLine(); // Consume newline
```

```
            switch (choice) {
                case 1:
                    manager.viewAvailability();
                    break;
                case 2:
                    System.out.print("Enter your name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter your contact: ");
                    String contact = scanner.nextLine();
                    System.out.print("Enter room type: ");
                    String roomType = scanner.nextLine();
                    System.out.print("Enter number of days: ");
                    int daysStayed = scanner.nextInt();
                    manager.makeReservation(name, contact, roomType,
daysStayed);
                    break;
                case 3:
                    System.out.print("Enter your contact to cancel the
reservation: ");
                    String cancelContact = scanner.nextLine();
                    manager.cancelReservation(cancelContact);
                    break;
                case 4:
```

```
                    System.out.println("Exiting Hotel Management System. Thank
you!");

                    scanner.close();

                    System.exit(0);

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

        }

    }

}
```

Output:

```
/Users/tawhid/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java

Hotel Management System Menu:
1. View Room Availability
2. Make a Reservation
3. Cancel a Reservation
4. Exit
Enter your choice: 1
Room 101: Available
Room 102: Available
Room 103: Available
Room 201: Available
Room 202: Available

Hotel Management System Menu:
1. View Room Availability
2. Make a Reservation
3. Cancel a Reservation
4. Exit
Enter your choice: 2
Enter your name: Tawhid Khan
Enter your contact: 8073576123
Enter room type: deluxe suite
Enter number of days: 4
Reservation successful. Total charge: $800.0
```