**Introduction**

The Hotel Management System (HMS) is designed to simplify the operations involved in managing a hotel. It provides an integrated solution for handling various aspects such as room bookings, customer management, billing, and staff management. The system leverages Java programming to deliver a robust, efficient, and user-friendly interface that ensures smooth operations and enhanced customer satisfaction. By automating routine tasks, the system reduces manual effort, minimizes errors, and allows hotel staff to focus more on delivering quality service.

**Features of the System**

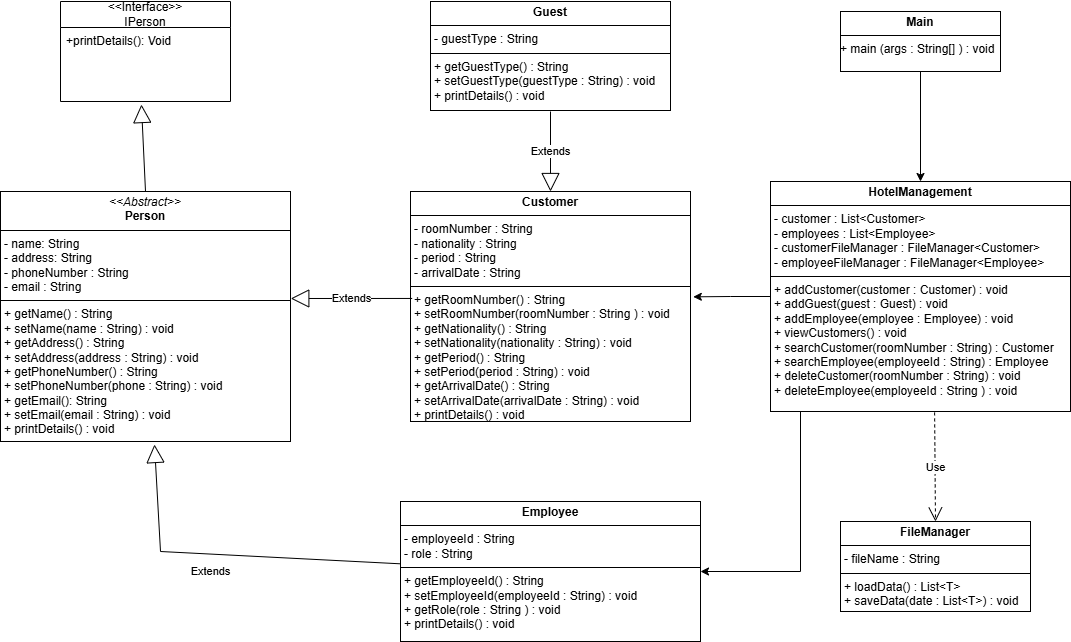
**1. Room Management:** Allows the addition, updating, and deletion of room details. Keeps track of room availability.

**2. Customer Management:** Facilitates customer check-in and check-out processes. Stores customer details for easy retrieval.

**4. Billing System:** Automatically generates bills based on customer usage and services availed. Ensures accurate and timely billing.

**5. Employee Management:** Maintains a database of employee details, including roles, and contact information.

**UML Diagram**



**Main Class**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

HotelManagement hotel = new HotelManagement();

Scanner sc = new Scanner([System.in](http://system.in/));

while (true) {

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\*\*\*\*\*\*\*\*\*\* East West Hotel \*\*\*\*\*\*\*\*\*\*\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\n1. CheckIn Customer");

System.out.println("2. Add Guest");

System.out.println("3. View Customers");

System.out.println("4. Search Customer");

System.out.println("5. CheckOut Customer");

System.out.println("6. Add Employee");

System.out.println("7. View Employees");

System.out.println("8. Delete Employee");

System.out.println("9. Exit");

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

int choice = sc.nextInt();

sc.nextLine(); // Consume newline

switch (choice) {

case 1: // Add Customer

System.out.print("Enter Room Number: ");

String roomNumber = sc.nextLine();

System.out.print("Enter Name: ");

String name = sc.nextLine();

System.out.print("Enter Address: ");

String address = sc.nextLine();

System.out.print("Enter Phone Number: ");

String phoneNumber = sc.nextLine();

System.out.print("Enter Email: ");

String email = sc.nextLine();

System.out.print("Enter Nationality: ");

String nationality = sc.nextLine();

System.out.print("Enter Period: ");

String period = sc.nextLine();

System.out.print("Enter Arrival Date: ");

String arrivalDate = sc.nextLine();

hotel.addCustomer(new Customer(roomNumber, name, address, phoneNumber, email, nationality, period, arrivalDate));

break;

case 2: // Add Guest

System.out.print("Enter Room Number: ");

String guestRoomNumber = sc.nextLine();

System.out.print("Enter Name: ");

String guestName = sc.nextLine();

System.out.print("Enter Address: ");

String guestAddress = sc.nextLine();

System.out.print("Enter Phone Number: ");

String guestPhoneNumber = sc.nextLine();

System.out.print("Enter Email: ");

String guestEmail = sc.nextLine();

System.out.print("Enter Nationality: ");

String guestNationality = sc.nextLine();

System.out.print("Enter Period: ");

String guestPeriod = sc.nextLine();

System.out.print("Enter Arrival Date: ");

String guestArrivalDate = sc.nextLine();

System.out.print("Enter Guest Type (VIP/Regular): ");

String guestType = sc.nextLine();

hotel.addGuest(new Guest(guestRoomNumber, guestName, guestAddress, guestPhoneNumber, guestEmail, guestNationality, guestPeriod, guestArrivalDate, guestType));

break;

case 3: // View Customers

hotel.viewCustomers();

break;

case 4: // Search Customer

System.out.print("Enter Room Number to Search: ");

Customer foundCustomer = hotel.searchCustomer(sc.nextLine());

if (foundCustomer != null) {

foundCustomer.printDetails();

} else {

System.out.println("Customer not found.");

}

break;

case 5: // Delete Customer

System.out.print("Enter Room Number to CheckOut: ");

hotel.deleteCustomer(sc.nextLine());

break;

case 6: // Add Employee

System.out.print("Enter Employee ID: ");

String empId = sc.nextLine();

System.out.print("Enter Name: ");

String empName = sc.nextLine();

System.out.print("Enter Address: ");

String empAddress = sc.nextLine();

System.out.print("Enter Phone Number: ");

String empPhone = sc.nextLine();

System.out.print("Enter Email: ");

String empEmail = sc.nextLine();

System.out.print("Enter Role: ");

String empRole = sc.nextLine();

hotel.addEmployee(new Employee(empId, empName, empAddress, empPhone, empEmail, empRole));

break;

case 7: // View Employees

hotel.viewEmployees();

break;

case 8: // Delete Employee

System.out.print("Enter Employee ID to Delete: ");

hotel.deleteEmployee(sc.nextLine());

break;

case 9: // Exit

System.out.println("Exiting...");

sc.close();

System.exit(0);

default:

System.out.println("Invalid choice. Try again.");

}

}

}

}

**Analysis**

**Customer Management:**

* **CheckIn Customer:** Registers a new customer in the hotel system, capturing details like name, address, phone number, email, nationality, stay period, and arrival date.
* **Add Guest:** Allows adding guests associated with an existing room reservation. This might be useful for scenarios where multiple people are staying in the same room.
* **View Customers:** Provides a facility to view the list of registered customers.
* **Search Customer:** Enables searching for a specific customer by room number.
* **CheckOut Customer:** Processes a customer's check-out, potentially including finalizing charges, updating room availability, and handling any other necessary steps.

**Employee Management:**

* **Add Employee:** Creates a new employee record in the system, storing essential details like employee ID, name, contact information, and role (e.g., receptionist, manager, housekeeping).
* **View Employees:** Provides a way to view a list of all employees currently registered in the system.
* **Delete Employee:** Enables removing an employee's record from the system.
* **User Menu for Interaction:** The system presents a user-friendly menu with clear options for managing customers, employees, and exiting the program. This allows users to interact with the system and perform desired actions in an intuitive way.

**IPerson.java**

package com.demo.hotelmanagment;

public interface IPerson {

void printDetails();

}

**Analysis**

Iperson is an interface and we took an abstract method named printDetails to print person details.

**Person Class**

package com.demo.hotelmanagment;

import java.io.Serializable;

public abstract class Person implements IPerson ,Serializable {

private String name;

private String address;

private String phoneNumber;

private String email;

public Person(String name, String address, String phoneNumber, String email) {

this.name = name;

this.address = address;

this.phoneNumber = phoneNumber;

this.email = email;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

// Print details method

@Override

public void printDetails() {

System.out.println("Name: " + name);

System.out.println("Address: " + address);

System.out.println("Phone: " + phoneNumber);

System.out.println("Email: " + email);

}

}

**Analysis**

* **Person (abstract):** Base class for Customer and Employee, holding common attributes like name, address, phone number, and email.
* **Customer:** Extends Person with specific attributes like name, address, phone number, email.
* **Employee:** Extends Person with attributes like employee ID and role.

**Employee Class**

package com.demo.hotelmanagment;

public class Employee extends Person {

private String employeeId;

private String role;

public Employee(String employeeId, String name, String address, String phoneNumber, String email, String role) {

super(name, address, phoneNumber, email);

this.employeeId = employeeId;

this.role = role;

}

public String getEmployeeId() {

return employeeId;

}

public void setEmployeeId(String employeeId) {

this.employeeId = employeeId;

}

public String getRole() {

return role;

}

public void setRole(String role) {

this.role = role;

}

// Print details method for Employee

public void printDetails() {

super.printDetails(); // Print common details from Person

System.out.println("Employee ID: " + employeeId);

System.out.println("Role: " + role);

}

}

**Analysis**

* **Person (abstract)**: Base class for Customer and Employee, holding common attributes like name, address, phone number, and email.
* **Employee**: Extends Person with attributes like employee ID and role.
* **HotelManagement:** Core class managing customer and employee data, including methods for adding, searching, viewing, and deleting records.

**Customer.Java**

**public class Customer extends Person:** This declares a public class named Customer that extends the Person class.

**public** **Customer(String roomNumber, String name, String address, String phoneNumber, String email, String nationality, String period, String arrivalDate):** This is the constructor for the Customer class.

This initializes the roomNumber, nationality, period, and arrivalDate fields of the Customer object.

**Getter and Setter Methods:**

* The class provides getter and setter methods for the roomNumber, nationality, period, and arrivalDate fields, allowing access and modification of these properties.

**printDetails() Method:**

The method prints the specific details of the Customer, which are the roomNumber, nationality, period, and arrivalDate.

**Guest.Java**

p**ublic class Guest extends Customer:** This line declares a public class named Guest that extends the Customerclass.

**public** **Guest(String roomNumber, String name, String address, String phoneNumber, String email, String nationality, String period, String arrivalDate, String guestType)**: This is the constructor for the Guest class.

The constructor initializes the guestType field of the Guest object.

**Getter and Setter Methods:**

* The class provides getter and setter methods for the guestType field, allowing access and modification of this property.

**public void printDetails():** This method overrides the printDetails() method inherited from the Customer class.

The method prints the specific detail of the Guest, which is the guestType.

**HotelManagement.Java**

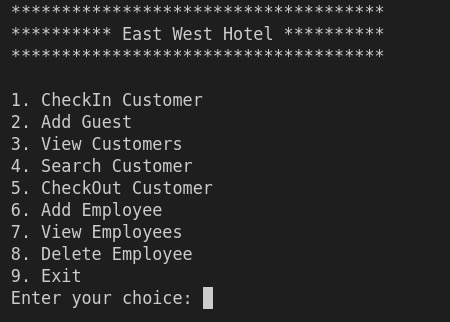
* **Data Management:**
* Maintains lists of Customer and Employee objects.
* Uses a FileManager class to load and save customer and employee data to/from files.
* **Customer Operations:**
* **AddCustomer():** Adds a new Customer object to the list.
* **addGuest()**: Adds a Guest object (which likely extends Customer) to the list.
* **viewCustomers():** Displays information about all registered customers.
* **searchCustomer()**: Searches for a specific customer by room number.
* **deleteCustomer()**: Removes a customer from the list.
* **Employee Operations:**
* addEmployee**():** Adds a new Employee object to the list.
* **viewEmployees()**: Displays information about all registered employees.
* **searchEmployee()**: Searches for a specific employee by employee ID.
* **deleteEmployee()**: Removes an employee from the list.

**FileManager.Java**

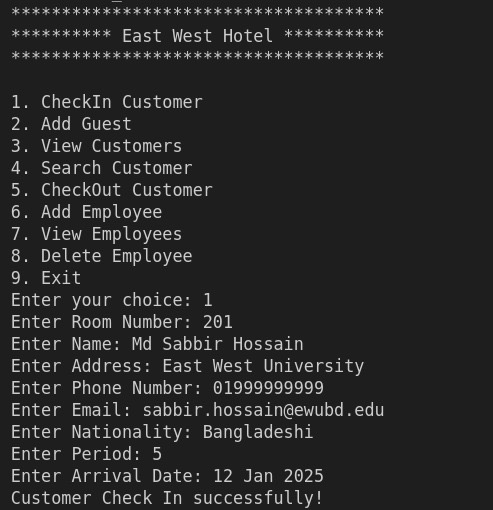
**Analysis**

* **Generic Type:** The class uses a generic type T to allow storing various data types in the files.
* **File Name:** It maintains a fileName attribute to specify the file used for data storage.
* **Loading Data:** The loadData() method attempts to read data from the specified file using an ObjectInputStream. If successful, it returns the loaded data as a List<T>. If there are any exceptions during loading, it returns an empty ArrayList.
* **Saving Data:** The saveData() method takes a list of data objects (List<T>) and writes them to the specified file using an ObjectOutputStream. Any exceptions encountered during saving are caught and a message is printed to the console.

**Output**



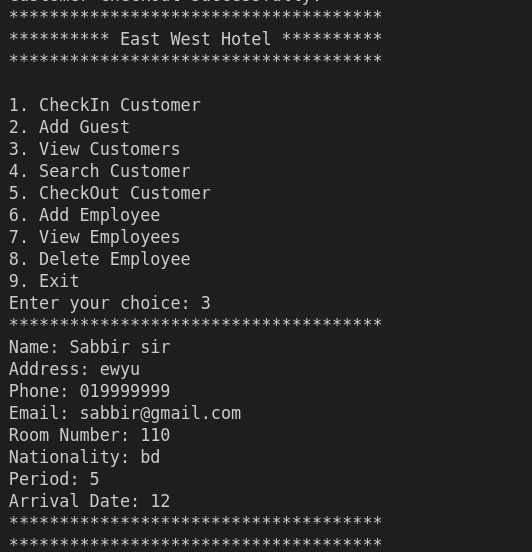
This is the output of this code and when you run the program one will be shown these above options and another option which is “enter your choice”.



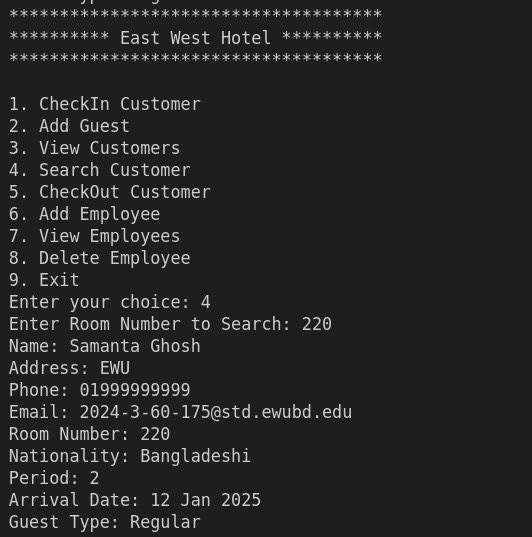
If a person clicks or choose option 1 which is CheckIn Customer then he/she will be shown few options and have to enter the details which are shown in the screen.



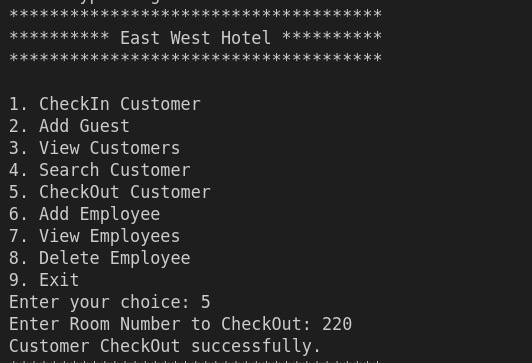
Option 2 indicates add guest where one will have to give all the details about guest and also guest type.



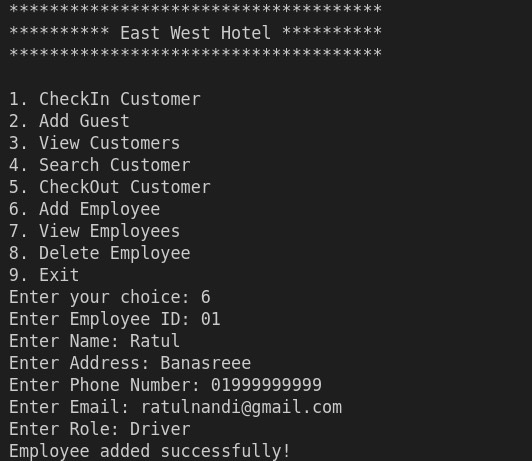
Option 3 indicates view customers where one will be able to view all the information's about customer.



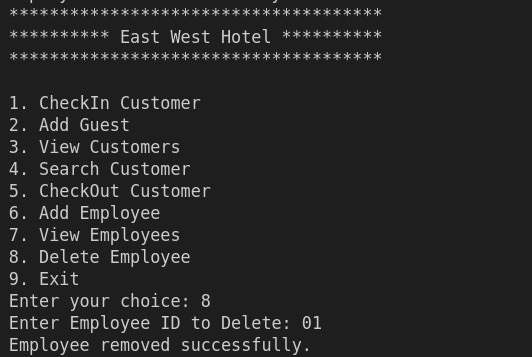
Option 4, is for search customer where one will be or the employee will be able to search for a customer details.



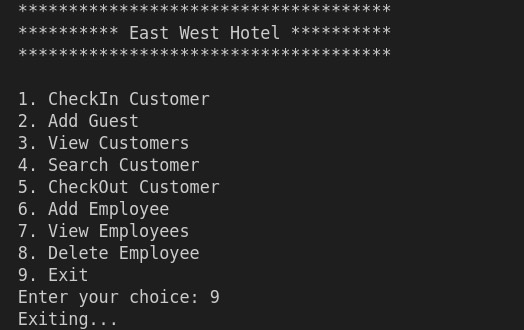
Option 5, is for Checkout Customer. If a customer wants to check out his/her room number will have to be inserted.



Option 6, is for adding a employee. For adding a employee one has to give all the information about the employee.



Option 8, is or deleting an employee.



Option 9, Program ends here.