Problem Statement 1:

A car rental agency wants to create a system to manage their car rentals. The system should allow customers to rent a car by selecting the type of car they want, the rental duration, and the pick-up and drop-off locations. The agency offers different types of cars, such as economy, midsize, and luxury, with different rental rates and availability.

The car rental system should be able to do the following:

1. Display the available cars and their details (type, rental rate, and availability).
2. Allow customers to rent a car by selecting the type of car they want, the rental duration, and the pick-up and drop-off locations.
3. Calculate the total rental cost based on the rental rate and duration.
4. Update the availability of cars after a rental is made.
5. Display the rented cars and their details (type, rental rate, rental duration, pick-up, and drop-off locations).
6. Allow customers to return a rented car and update its availability.

Object-oriented approach:

To implement the above car rental system, we can use the following classes:

1. **Car**: This class will contain the details of each car, such as the type, rental rate, and availability. It will also have methods to display the car details and update its availability.

2. **Customer**: This class will contain the details of each customer, such as the rental duration, pick-up, and drop-off locations. It will also have methods to calculate the total rental cost and display the rented car details.

3. **CarRentalSystem**: This class will act as the main controller of the system. It will contain the list of available cars and rented cars and methods to display the available cars, rent a car, return a car, and update the car availability.

Using the above classes, we can create objects and use them to interact with the car rental system. For example, we can create a Car object for each car in the system, a Customer object for each customer, and a CarRentalSystem object to manage the overall system.

Please consider following edge cases while creating the solution
1. What happens if a customer tries to rent a car that is not available?
2. What happens if a customer tries to return a car that was not rented from the rental agency?
3. What happens if a customer wants to extend the rental duration of a car that they have already rented?

4. What happens if a customer wants to rent a car for a duration that is longer than the maximum rental period allowed by the rental agency?
5. What happens if a customer wants to rent a car from a location that is not a valid pick-up location according to the rental agency?
6. What happens if a customer wants to return a car to a location that is not a valid drop-off location according to the rental agency?
7. What happens if a customer selects a car type that is not offered by the rental agency?
8. What happens if a customer tries to rent a car for a duration that is less than the minimum rental period allowed by the rental agency?
9. What happens if the rental agency needs to perform maintenance on a car and it becomes unavailable for a period of time?
10. What happens if a customer tries to rent a car but does not have a valid driver's license?

Problem Statement 2:
A company wants to create a system to manage their employee data. The system should allow the company to add new employees, remove employees, update employee information, and display all the employee information.
The employee data should contain the following information:

1. **Name**: The name of the employee.
2. **Age**: The age of the employee.
3. **ID**: A unique identifier for the employee.
4. **Salary**: The salary of the employee.
5. **Department**: The department the employee works in.

The system should also have the following functionalities:

1. **Add employee**: This functionality should allow the company to add a new employee to the system by providing the employee information.
2. **Remove employee**: This functionality should allow the company to remove an employee from the system by providing the employee ID.
3. **Update employee information**: This functionality should allow the company to update the employee information by providing the employee ID and the new information.
4. **Display all employee information**: This functionality should display all the employee information in the system.

Object-oriented approach:

To implement the above employee data management system, we can use the following classes:

1. **Employee**: This class will contain employee information such as name, age, ID, salary, and department. It will also have methods to set and get the employee information.
2. **EmployeeDatabase**: This class will act as the main controller of the system. It will contain a list of all employees in the system and methods to add, remove, update, and display employee information.

Using the above classes, we can create objects and use them to interact with the employee data management system. For example, we can create an Employee object for each employee in the system, and an EmployeeDatabase object to manage the overall system.

Please consider following edge cases while creating the solution

1. **Employee ID uniqueness**: The system should ensure that the employee ID is unique for each employee. The system should not allow adding an employee with the same ID as an existing employee.
2. **Invalid input data**: The system should handle invalid or incorrect input data, such as non-numeric age, negative salary, or invalid department name.
3. **Adding an employee to a non-existent department**: The system should handle the scenario where an employee is added to a department that does not exist in the system.
4. **Removing a non-existent employee**: The system should handle the scenario where the user tries to remove an employee that does not exist in the system.
5. **Updating non-existent employee information**: The system should handle the scenario where the user tries to update the information of an employee that does not exist in the system.
6. **Displaying empty employee list**: The system should handle the scenario where there are no employees in the system, and the user tries to display all employee information.

Problem Statement 3:

A hotel wants to create a system to manage their room bookings. The system should allow the hotel staff to add new bookings, remove bookings, update booking information, and display all the bookings.

The booking data should contain the following information:

1. **Room number**: The number of room booked.
2. **Guest name**: The name of the guest who booked the room.
3. **Check-in date**: The date the guest checked into the room.
4. **Check-out date**: The date the guest checked out of the room.
5. **Total cost**: The total cost of the booking.

The system should also have the following functionalities:

1. **Add booking**: This functionality should allow the hotel staff to add a new booking to the system by providing the booking information.
2. **Remove booking**: This functionality should allow the hotel staff to remove a booking from the system by providing the room number.
3. **Update booking information**: This functionality should allow the hotel staff to update the booking information by providing the room number and the new information.
4. **Display all bookings**: This functionality should display all the bookings in the system.

Object-oriented approach:

To implement the above room booking management system, we can use the following classes:

1. **Booking**: This class will contain booking information such as room number, guest name, check-in date, check-out date, and total cost. It will also have methods to set up and get the booking information.

2. **BookingSystem**: This class will act as the main controller of the system. It will contain a list of all bookings in the system and methods to add, remove, update, and display booking information.

Using the above classes, we can create objects and use them to interact with the room booking management system. For example, we can create a Booking object for each booking in the system, and a BookingSystem object to manage the overall system.

Please consider following edge cases while creating the solution

1. Room number edge cases:
    - Negative room numbers should not be allowed as room numbers are always positive.
    - Large room numbers may exceed the maximum value for an integer, so the system should handle this appropriately.
    - The same room number cannot be booked by different guests simultaneously, so the system should handle cases where a room is already booked and not allow double-booking.
2. Guest name edge cases:
    - Guest names should be limited to a certain character limit, and the system should handle names that are too long or contain special characters.
    - Guest names that are the same as other guest names may cause confusion, so the system should handle cases where the same guest name is used for different bookings.
3. Check-in and check-out dates edge cases:
    - The check-in date should be before the check-out date, so the system should handle cases where these dates are entered incorrectly.

- The system should also handle cases where the check-in and check-out dates are the same, which would mean the guest stayed for only one day.
- The system should handle cases where the check-out date is before the check-in date, which would indicate an error.

4. Total cost edge cases:
    - The total cost of the booking should be calculated accurately based on the number of days the guest stayed and any additional charges.
    - The system should handle cases where the cost is negative or exceeds a certain amount, which may indicate an error.
5. Add booking edge cases:
    - The system should handle cases where a booking is added for a room that is already booked.
    - The system should handle cases where the guest's name or booking information is incomplete.
6. Remove booking edge cases:
    - The system should handle cases where the room number entered for removal does not exist in the system.
    - The system should handle cases where the booking to be removed does not exist in the system.
7. Update booking information edge cases:
    - The system should handle cases where the room number entered for updating does not exist in the system.
    - The system should handle cases where the booking to be updated does not exist in the system.
    - The system should handle cases where the updated information is incomplete or incorrect.
8. Display all bookings edge cases:
    - The system should handle cases where there are no bookings in the system and display an appropriate message.
    - The system should handle cases where the display format is incorrect or unreadable.