5. Car Rental System

A Car Rental System is a software built to handle the renting of automobiles for a short period of time, generally ranging from a few hours to a few weeks. A car rental system often has numerous local branches (to allow its user to return a vehicle to a different location), and primarily located near airports or busy city areas.

Car Rental System

System Requirements

We will focus on the following set of requirements while designing our Car Rental System:

1. The system will support the renting of different automobiles like cars, trucks, SUVs, vans, and motorcycles.
2. Each vehicle should be added with a unique barcode and other details, including a parking stall number which helps to locate the vehicle.

3. The system should be able to retrieve information like which member took a particular vehicle or what vehicles have been rented out by a specific member.
4. The system should collect a late-fee for vehicles returned after the due date.
5. Members should be able to search the vehicle inventory and reserve any available vehicle.
6. The system should be able to send notifications whenever the reservation is approaching the pick-up date, as well as when the vehicle is nearing the due date or has not been returned within the due date.
7. The system will be able to read barcodes from vehicles.
8. Members should be able to cancel their reservations.
9. The system should maintain a vehicle log to track all events related to the vehicles.
10. Members can add rental insurance to their reservation.
11. Members can rent additional equipment, like navigation, child seat, ski rack, etc.
12. Members can add additional services to their reservation, such as roadside assistance, additional driver, wifi, etc.

Use Case Diagram

We have four main Actors in our system:

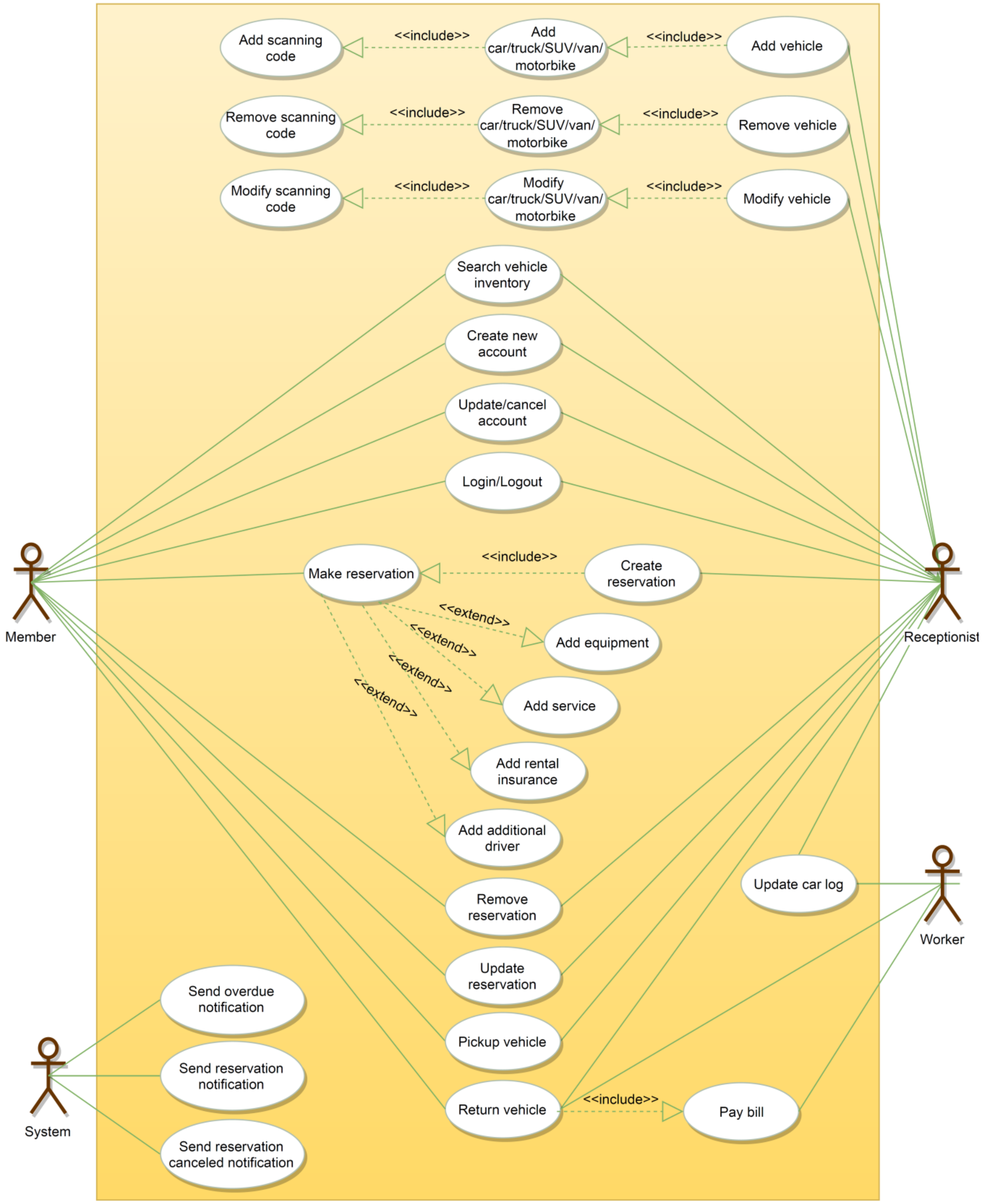
- **Receptionist**: Mainly responsible for adding and modifying vehicles and workers. Receptionists can also reserve vehicles.
- **Member**: All members can search the catalog, as well as reserve, pick-up, and return a vehicle.
- **System**: Mainly responsible for sending notifications about overdue vehicles, canceled reservation, etc.

- **Worker**: Mainly responsible for taking care of a returned vehicle and updating the vehicle log.

Here are the top use cases of the Car Rental System:

- **Add/Remove/Edit vehicle**: To add, remove or modify a vehicle.
- **Search catalog**: To search for vehicles by type and availability.
- **Register new account/Cancel membership**: To add a new member or cancel an existing membership.
- **Reserve vehicle**: To reserve a vehicle.
- **Check-out vehicle**: To rent a vehicle.
- **Return a vehicle**: To return a vehicle which was checked-out to a member.
- **Add equipment**: To add an equipment to a reservation like navigation, child seat, etc.
- **Update car log**: To add or update a car log entry, such as refueling, cleaning, damage, etc.

Here is the use case diagram of our Car Rental System:

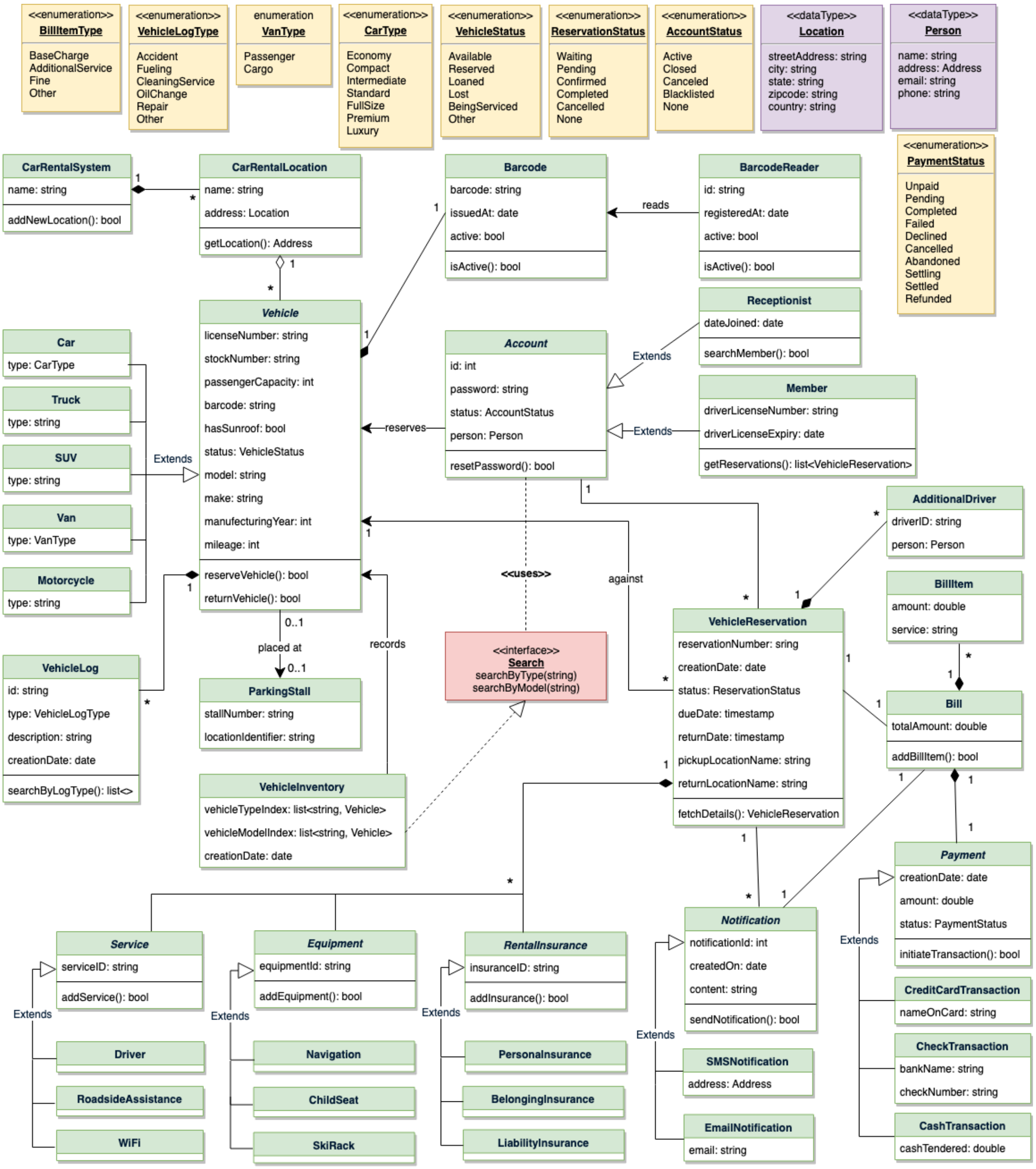


Use Case Diagram for Car Rental System

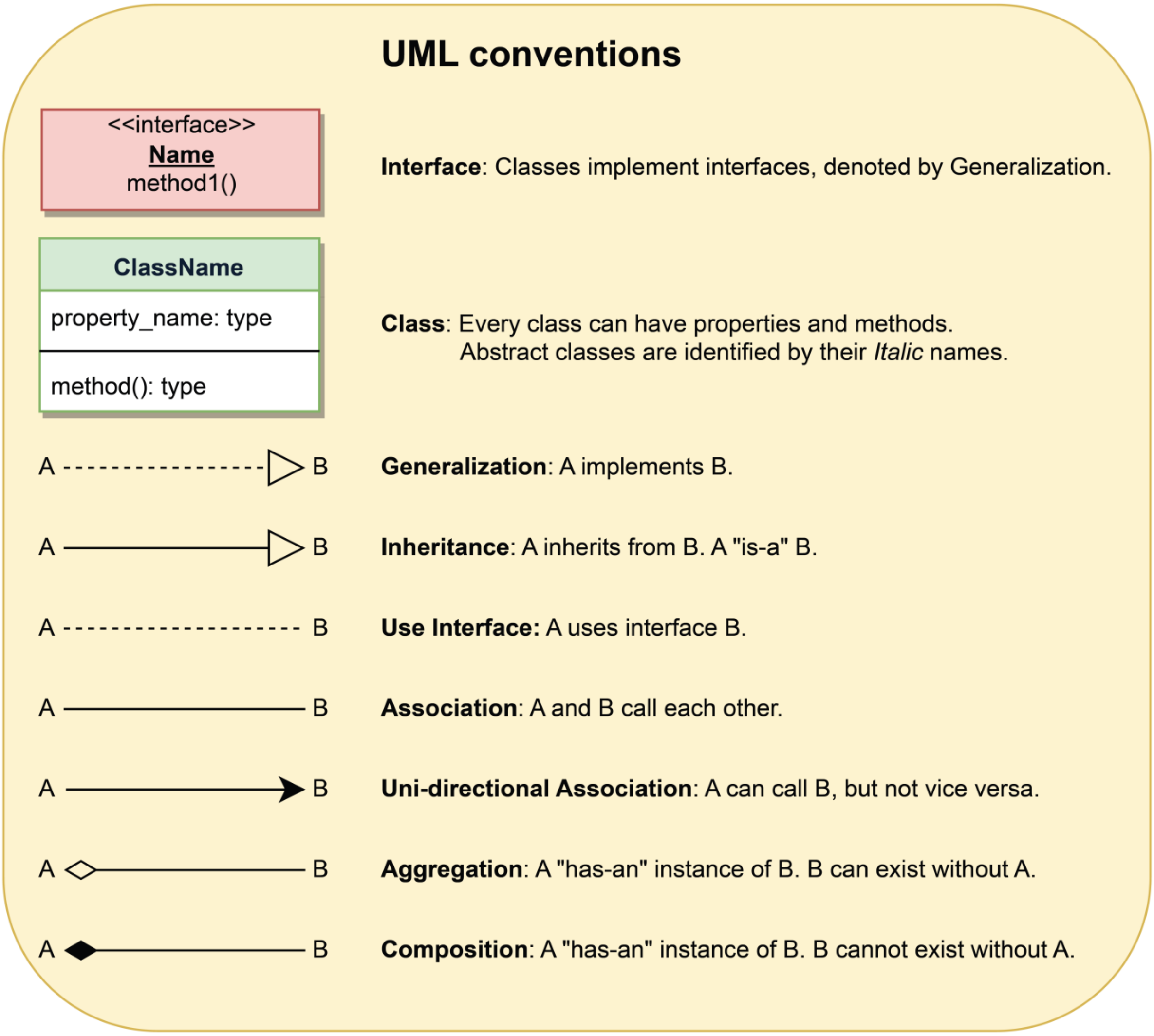
Class Diagram

Here are the main classes of our Car Rental System:

- **CarRentalSystem**: The main part of the organization for which this software has been designed.
- **CarRentalLocation**: The car rental system will have multiple locations, each location will have attributes like ‘Name’ to distinguish it from any other locations and ‘Address’ which defines the address of the rental location.
- **Vehicle**: The basic building block of the system. Every vehicle will have a barcode, license plate number, passenger capacity, model, make, mileage, etc. Vehicles can be of multiple types, like car, truck, SUV, etc.
- **Account**: Mainly, we will have two types of accounts in the system, one will be a general member and the other will be a receptionist. Another account can be of the worker taking care of the returned vehicle.
- **VehicleReservation**: This class will be responsible for managing reservations for a vehicle.
- **Notification**: Will take care of sending notifications to members.
- **VehicleLog**: To keep track of all the events related to a vehicle.
- **RentalInsurance**: Stores details about the various rental insurances that members can add to their reservation.
- **Equipment**: Stores details about the various types of equipment that members can add to their reservation.
- **Service**: Stores details about the various types of service that members can add to their reservation, such as additional drivers, roadside assistance, etc.
- **Bill**: Contains different bill-items for every charge for the reservation.



Class Diagram for Car Rental System



UML conventions

Interface: Classes implement interfaces, denoted by Generalization.

Class: Every class can have properties and methods.
Abstract classes are identified by their *Italic* names.

Generalization: A implements B.

Inheritance: A inherits from B. A "is-a" B.

Use Interface: A uses interface B.

Association: A and B call each other.

Uni-directional Association: A can call B, but not vice versa.

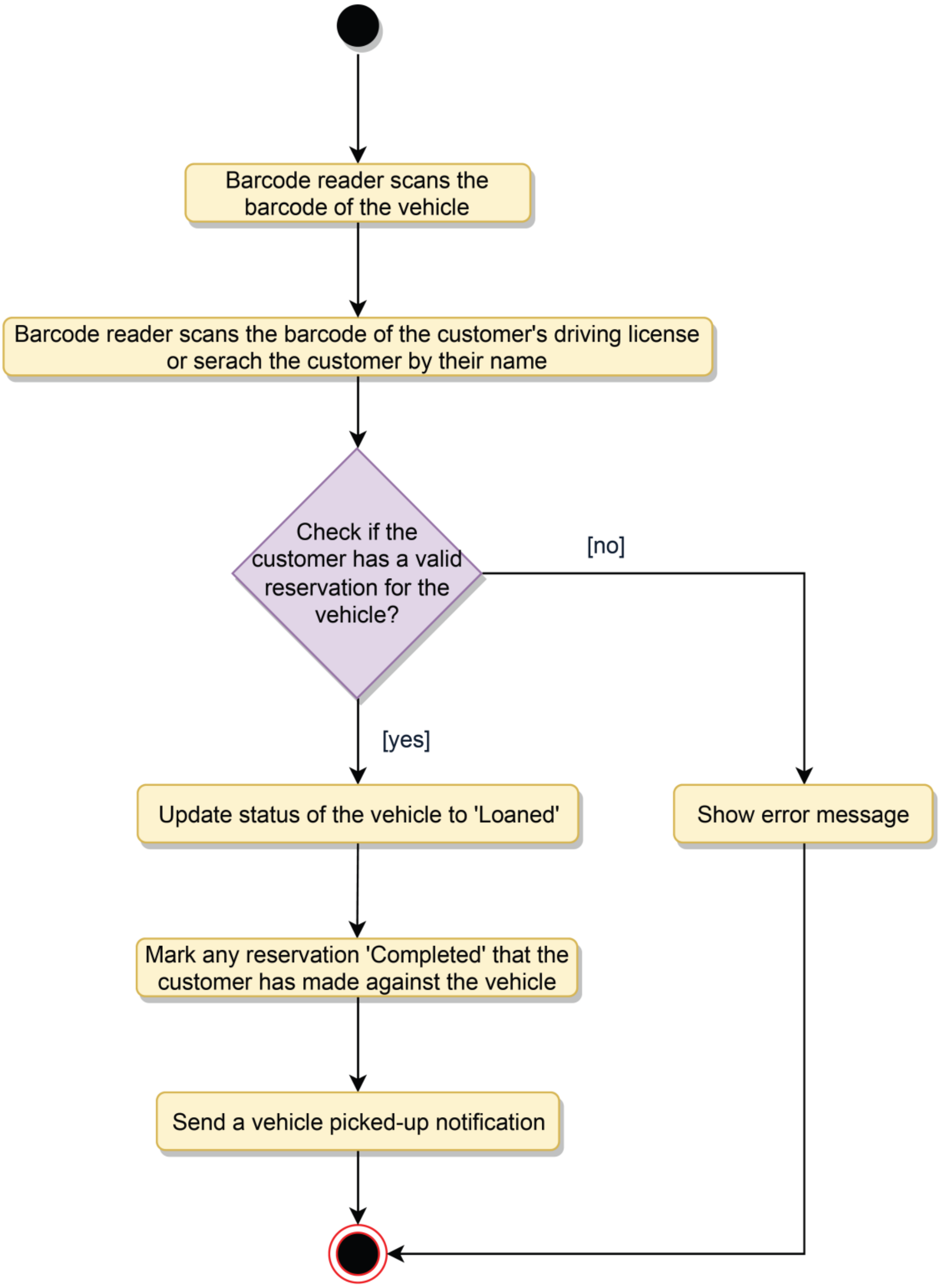
Aggregation: A "has-an" instance of B. B can exist without A.

Composition: A "has-an" instance of B. B cannot exist without A.

UML for Car Rental System

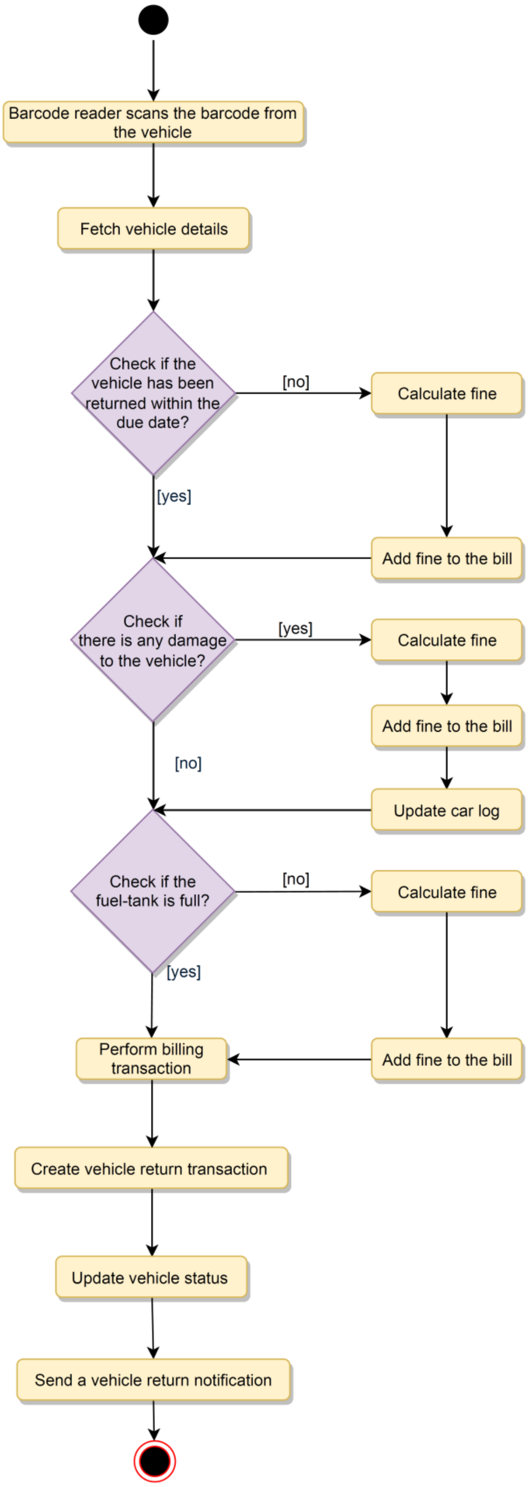
Activity Diagrams

Pick up a vehicle: Any member can perform this activity. Here are the steps to pick up a vehicle:


Barcode reader scans the barcode of the vehicle
Barcode reader scans the barcode of the customer's driving license or serach the customer by their name
Check if the customer has a valid reservation for the vehicle?
[no]
[yes]
Update status of the vehicle to 'Loaned'
Show error message
Mark any reservation 'Completed' that the customer has made against the vehicle
Send a vehicle picked-up notification

Activity Diagram for Car Rental System Pick Up

Return a vehicle: Any worker can perform this activity. While returning a vehicle, the system must collect a late fee from the member if the return date is after the due date. Here are the steps for returning a vehicle:


Barcode reader scans the barcode from the vehicle
Fetch vehicle details
Check if the vehicle has been returned within the due date?
[no]
Calculate fine
[yes]
Add fine to the bill
Check if there is any damage to the vehicle?
[yes]
Calculate fine
Add fine to the bill
[no]
Update car log
Check if the fuel-tank is full?
[no]
Calculate fine
[yes]
Perform billing transaction
Add fine to the bill
Create vehicle return transaction
Update vehicle status
Send a vehicle return notification

Activity Diagram for Car Rental System Return

Code

Here is the high-level definition for the classes described above.

Enums, data types and constants: Here are the required enums, data types, and constants:

```
package carrental;

import java.util.*;

// Enums for various statuses and types
enum BillItemType { BASE_CHARGE, ADDITIONAL_SERVICE, FINE, OTHER; }

enum VehicleLogType { ACCIDENT, FUELING, CLEANING_SERVICE, OIL_CHANGE, REPAIR, OTHER; }

enum VanType { PASSENGER, CARGO; }

enum CarType { ECONOMY, COMPACT, INTERMEDIATE, STANDARD, FULL_SIZE, PREMIUM, LUXURY;
}

enum VehicleStatus { AVAILABLE, RESERVED, LOANED, LOST, BEING_SERVICED, OTHER; }

enum ReservationStatus { ACTIVE, PENDING, CONFIRMED, COMPLETED, CANCELLED, NONE; }

enum AccountStatus { ACTIVE, CLOSED, CANCELED, BLACKLISTED, BLOCKED; }

enum PaymentStatus { UNPAID, PENDING, COMPLETED, FILLED, DECLINED, CANCELLED,
ABANDONED, SETTLING, SETTLED, REFUNDED; }

// Address class
class Address {
  private String streetAddress, city, state, zipCode, country;
  public Address(String street, String city, String state, String zipCode, String country) {
    this.streetAddress = street;
    this.city = city;
    this.state = state;
    this.zipCode = zipCode;
    this.country = country;
  }
}

// Person class
class Person {
  private String name, email, phone;
  private Address address;
  public Person(String name, Address address, String email, String phone) {
    this.name = name;
    this.address = address;
    this.email = email;
    this.phone = phone;
  }
}

// Abstract Account class
abstract class Account {
  private String id, password;
  private AccountStatus status;
  private Person person;
  public Account(String id, String password, Person person, AccountStatus status) {
    this.id = id;
    this.password = password;
    this.person = person;
    this.status = status;
  }
  public void resetPassword() {}
}

// Member class
class Member extends Account {
  private int totalVehiclesReserved = 0;
  public Member(String id, String password, Person person, AccountStatus status) {
    super(id, password, person, status);
  }
  public void getReservations() {}
}



// Receptionist class
class Receptionist extends Account {
  private Date dateJoined;
  public Receptionist(String id, String password, Person person, AccountStatus status, Date
dateJoined) {
    super(id, password, person, status);
    this.dateJoined = dateJoined;
  }
  public void searchMember(String name) {}
}
```

```
// Additional Driver class
class AdditionalDriver {
    private String driverId;
    private Person person;
    public AdditionalDriver(String id, Person person) {
        this.driverId = id;
        this.person = person;
    }
}

// Car Rental Location class
class CarRentalLocation {
    private String name;
    private Address location;
    public CarRentalLocation(String name, Address location) {
        this.name = name;
        this.location = location;
    }
    public Address getLocation() { return location; }
}

// Car Rental System class
class CarRentalSystem {
    private String name;
    private List<CarRentalLocation> locations = new ArrayList<>();
    public CarRentalSystem(String name) { this.name = name; }
    public void addNewLocation(CarRentalLocation location) { locations.add(location); }
}

// Abstract Vehicle class
abstract class Vehicle {
    private String licenseNumber, stockNumber, barcode, model, make;
    private int passengerCapacity, manufacturingYear, mileage;
    private boolean hasSunroof;
    private VehicleStatus status;
    private List<VehicleLog> log = new ArrayList<>();
    public Vehicle(String licenseNum, String stockNum, int capacity, String barcode, boolean
hasSunroof,
                  VehicleStatus status, String model, String make, int manufacturingYear, int mileage) {
        this.licenseNumber = licenseNum;
        this.stockNumber = stockNum;
        this.passengerCapacity = capacity;
        this.barcode = barcode;
        this.hasSunroof = hasSunroof;
    this.status = status;
    this.model = model;
    this.make = make;
    this.manufacturingYear = manufacturingYear;
    this.mileage = mileage;
  }
  public void reserveVehicle() {}
  public void returnVehicle() {}
}

// Car class
class Car extends Vehicle {
  private CarType type;
  public Car(String licenseNum, String stockNum, int capacity, String barcode, boolean hasSunroof,
        VehicleStatus status, String model, String make, int manufacturingYear, int mileage, CarType
type) {
    super(licenseNum, stockNum, capacity, barcode, hasSunroof, status, model, make,
manufacturingYear, mileage);
    this.type = type;
  }
}

// Vehicle Log class
class VehicleLog {
  private String id, description;
  private VehicleLogType type;
  private Date creationDate;
  public VehicleLog(String id, VehicleLogType type, String description, Date creationDate) {
    this.id = id;
    this.type = type;
    this.description = description;
    this.creationDate = creationDate;
  }
  public void update() {}
  public void searchByLogType(VehicleLogType type) {}
}

// Vehicle Reservation class
class VehicleReservation {
  private String reservationNumber, pickupLocationName, returnLocationName;
  private Date creationDate, dueDate, returnDate;
  private ReservationStatus status;
  private int customerId;
  private Vehicle vehicle;
  private List<AdditionalDriver> additionalDrivers = new ArrayList<>();
```

```
    public VehicleReservation(String reservationNumber) {
        this.reservationNumber = reservationNumber;
        this.creationDate = new Date();
        this.status = ReservationStatus.ACTIVE;
        this.dueDate = new Date();
        this.returnDate = new Date();
    }
    public List<AdditionalDriver> getAdditionalDrivers() { return additionalDrivers; }
}

// Search interface
interface Search {
    void searchByType(VehicleLogType type);
    void searchByModel(String model);
}

// Vehicle Inventory class
class VehicleInventory implements Search {
    private Map<VehicleLogType, List<Vehicle>> vehicleTypes = new HashMap<>();
    private Map<String, List<Vehicle>> vehicleModels = new HashMap<>();
    public void searchByType(VehicleLogType query) {}
    public void searchByModel(String query) {}
}
```