# 1 Introduction

## 1.1 Purpose

This subsection should

- a) Delineate the purpose of the SRS;
- b) Specify the intended audience for the SRS.

## 1.2 Scope

Name of software to be developed: ParkMS System

This subsection should

- b) Explain what the software product(s) will, and, if necessary, will not do;
- c) Describe the application of the software being specified, including relevant benefifits, objectives, and goals;
- d) Be consistent with similar statements in higher-level specifiﬁcations (e.g., the system requirements speciﬁcation), if they exist.

## 1.3 Product Overview

### 1.3.1 Product perspective

This subsection of the SRS should put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection should relate the requirements of that larger system to functionality of the software and should identify interfaces between that system and the software.

This subsection should also describe how the software operates inside various constraints. For example,
these constraints could include

- a) System interfaces;
- b) User interfaces;
- c) Hardware interfaces;
- d) Software interfaces;
- e) Communications interfaces;
- f) Memory;
- j) Operations;
- k) Site adaptation requirements.

#### 1.3.1.1 System interfaces

**SI1 - ParkMSSystem**

| | |
|---|---|
| **Service Name:** | ParkMSSystem |
| **Service ID:** | SI1 |
| **Description:** | |
| **Operation:** | - openPark<br>- closePark |
| **Temporary Variable** | **Variable Description** |
| CurrentPark | CurrentPark is a object of Park |
| CurrentParkRecord | CurrentParkRecord is a object of ParkRecord |
| CurrentMember | CurrentMember is a object of Member |
| CurrentPayment | the type of CurrentPayment is Real |

#### SI2 - ThirdPartyServices

| | |
|---|---|
| **Service Name:** | ThirdPartyServices |
| **Service ID:** | SI2 |
| **Description:** | |
| **Operation:** | |

#### SI3 - SetPriceService

| | |
|---|---|
| **Service Name:** | SetPriceService |
| **Service ID:** | SI3 |
| **Description:** | |
| **Operation:** | - setSmallPrice<br>- setLargePrice<br>- setMotoPrice |

#### SI4 - GetHistoryService

| | |
|---|---|
| **Service Name:** | GetHistoryService |
| **Service ID:** | SI4 |
| **Description:** | |
| **Operation:** | - getHistoryByPlateNumber<br>- getHistoryByEntryTime<br>- getHistoryByOutTime<br>- getHistoryByMember |

#### SI5 - AutomaticEntryService

| | |
|---|---|
| **Service Name:** | AutomaticEntryService |
| **Service ID:** | SI5 |
| **Description:** | |
| **Operation:** | - automaticEntry |
| **Temporary Variable** | **Variable Description** |
| RecordID | the type of RecordID is Integer |

### SI6 - ManuallyAllowOutService

| | |
|---|---|
| **Service Name:** | ManuallyAllowOutService |
| **Service ID:** | SI6 |
| **Description:** | |
| **Operation:** | - manuallyAllowOut |

### SI7 - ManuallyAllowEntryService

| | |
|---|---|
| **Service Name:** | ManuallyAllowEntryService |
| **Service ID:** | SI7 |
| **Description:** | |
| **Operation:** | - manuallyAllowEntry |

### SI8 - AutomaticOutService

| | |
|---|---|
| **Service Name:** | AutomaticOutService |
| **Service ID:** | SI8 |
| **Description:** | |
| **Operation:** | - scanPlateNumber<br>- onlinePay |

### SI9 - RegisterService

| | |
|---|---|
| **Service Name:** | RegisterService |
| **Service ID:** | SI9 |
| **Description:** | |
| **Operation:** | - registerMember<br>- registerVehicle |

### SI10 - RechargeService

| | |
|---|---|
| **Service Name:** | RechargeService |
| **Service ID:** | SI10 |
| **Description:** | |
| **Operation:** | - recharge |

#### SI11 - ManageParkCRUDService

| | |
|---|---|
| **Service Name:** | ManageParkCRUDService |
| **Service ID:** | SI11 |
| **Description:** | |
| **Operation:** | - createPark<br>- queryPark<br>- modifyPark<br>- deletePark |

#### SI12 - ManageVehicleCRUDService

| | |
|---|---|
| **Service Name:** | ManageVehicleCRUDService |
| **Service ID:** | SI12 |
| **Description:** | |
| **Operation:** | - createVehicle<br>- queryVehicle<br>- modifyVehicle<br>- deleteVehicle |

## 1.3.2 Product functions

#### Use Case Diagram

Use Case Diagram

| ID | Use Case Name | Use Case Description | Subfunction |
|---|---|---|---|
| UC1 | manuallyAllowOut | | manuallyAllowOut |
| UC2 | manuallyAllowEntry | | manuallyAllowEntry |
| UC3 | automaticEntry | | automaticEntry |
| UC4 | automaticOut | | scanPlateNumber<br>onlinePay |
| UC5 | setPrice | | setSmallPrice<br>setLargePrice<br>setMotoPrice |
| UC6 | getHistory | | getHistoryByPlateNumber<br>getHistoryByEntryTime<br>getHistoryByOutTime<br>getHistoryByMember |
| UC7 | register | | registerMember<br>registerVehicle |
| UC8 | recharge | | recharge |
| UC9 | openPark | | |
| UC10 | closePark | | |

### 1.3.3 User characteristics

| ID | Actor | Description | Super Actor |
|---|---|---|---|
| A1 | Driver | | |
| A2 | SystemManager | | |
| A3 | ParkManager | | |

### 1.3.4 Limitations

This subsection of the SRS should provide a general description of any other items that will limit the developer’s options. These include

- a) Regulatory policies;
- b) Hardware limitations (e.g., signal timing requirements);
- c) Interfaces to other applications;
- d) Parallel operation;
- e) Audit functions;
- f) Control functions;
- g) Higher-order language requirements;
- h) Signal handshake protocols (e.g., XON-XOFF, ACK-NACK);
- i) Reliability requirements;
- j) Criticality of the application;
- k) Safety and security considerations.
- l) physical/mental considerations; and
- m) limitations that are sourced from other systems, including real-time requirements from the controlled system through interfaces.

## 1.4 Definitions

This subsection should provide the defifinitions of all terms required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.

# 2 References

This subsection should

- a) Provide a complete list of all documents referenced elsewhere in the SRS;
- b) Identify each document by title, report number (if applicable), date, and publishing organization;
- c) Specify the sources from which the references can be obtained.

This information may be provided by reference to an appendix or to another document.

# 3 Requirements

## 3.1 Functions

### 3.1.1 Use Case

**UC1 - manuallyAllowOut**

System Sequence Diagram:

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | manuallyAllowOut |
| **UseCase ID:** | UC1 |
| **Brief Description:** | |
| **Involved Actor:** | ParkManager |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

### UC2 - manuallyAllowEntry

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | manuallyAllowEntry |
| **UseCase ID:** | UC2 |
| **Brief Description:** | |
| **Involved Actor:** | ParkManager |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

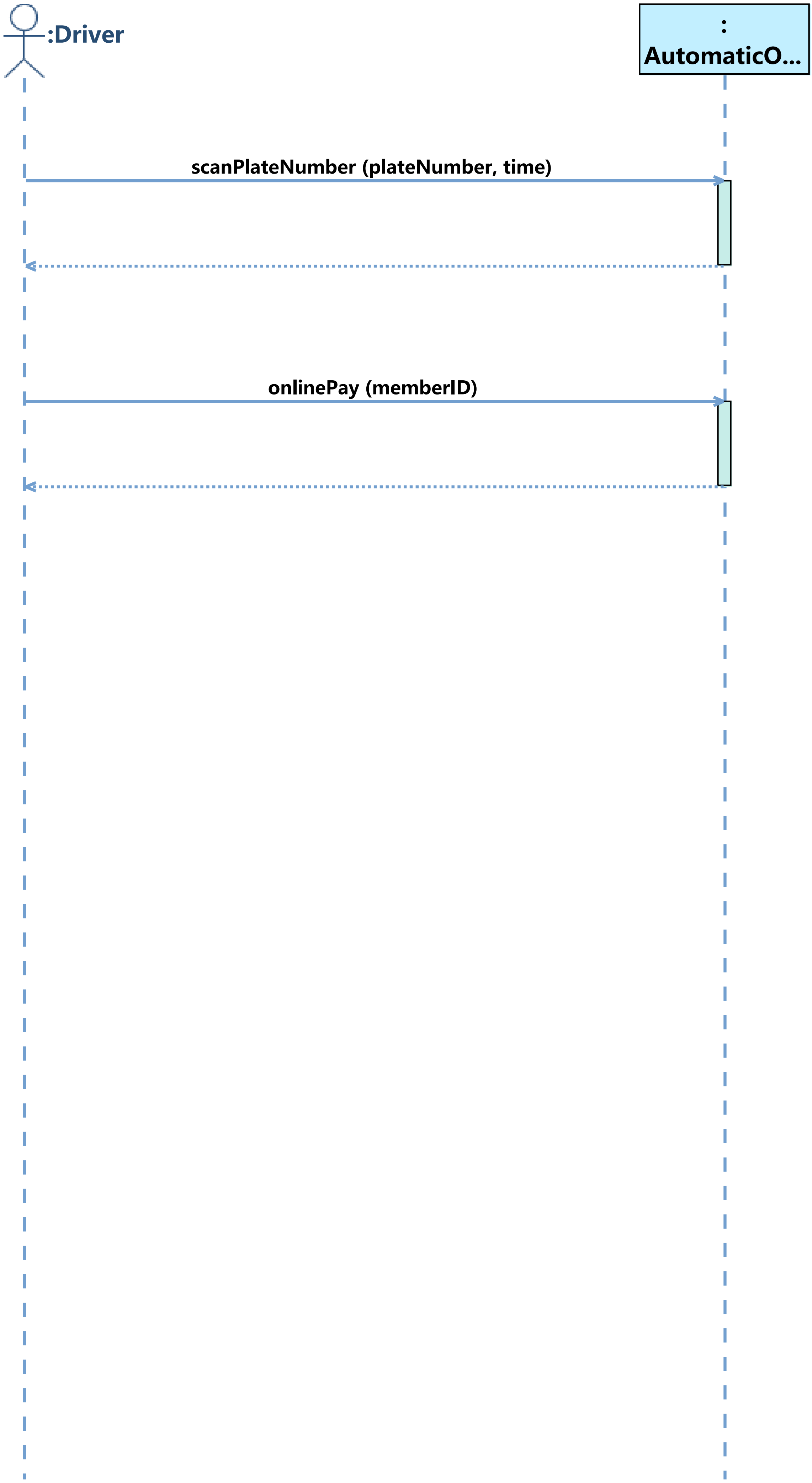
### UC3 - automaticEntry

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | automaticEntry |
| **UseCase ID:** | UC3 |
| **Brief Description:** | |
| **Involved Actor:** | Driver |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

### UC4 - automaticOut

System Sequence Diagram:


:Driver
: AutomaticO...
scanPlateNumber (plateNumber, time)
onlinePay (memberID)

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | automaticOut |
| **UseCase ID:** | UC4 |
| **Brief Description:** | |
| **Involved Actor:** | Driver |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | 1. Driver clicks to execute the operation scanPlateNumber, with entering plateNumber, time<br>2. Driver clicks to execute the operation onlinePay, with entering memberID |
| **Alternative Path:** | |

**UC5 - setPrice**

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | setPrice |
| **UseCase ID:** | UC5 |
| **Brief Description:** | |
| **Involved Actor:** | SystemManager |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

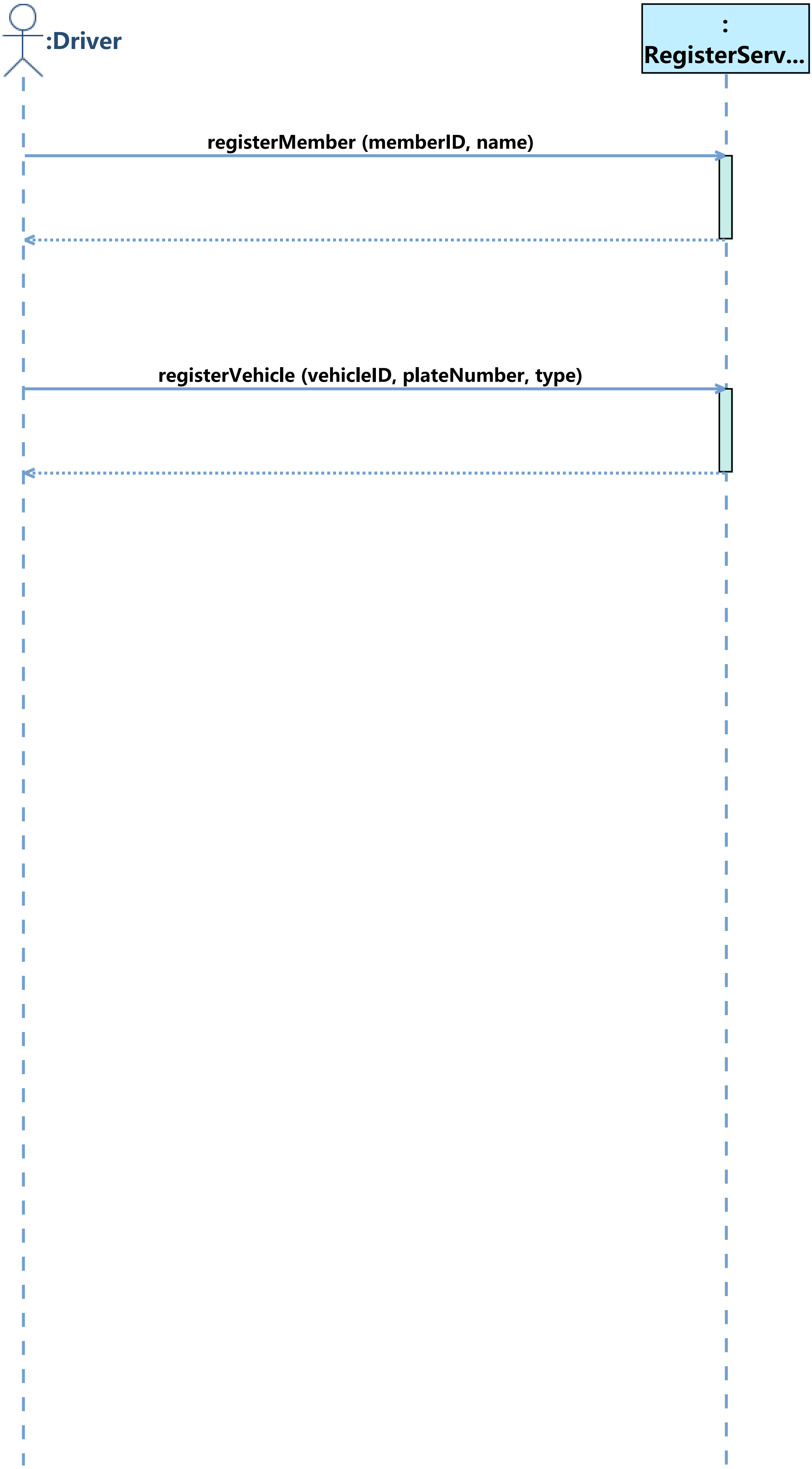
**UC6 - getHistory**

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | getHistory |
| **UseCase ID:** | UC6 |
| **Brief Description:** | |
| **Involved Actor:** | SystemManager |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

**UC7 - register**

System Sequence Diagram:


:Driver
: RegisterServ...
registerMember (memberID, name)
registerVehicle (vehicleID, plateNumber, type)

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | register |
| **UseCase ID:** | UC7 |
| **Brief Description:** | |
| **Involved Actor:** | Driver |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | 1. Driver clicks to execute the operation registerMember, with entering memberID, name<br>2. Driver clicks to execute the operation registerVehicle, with entering vehicleID, plateNumber, type |
| **Alternative Path:** | |

**UC8 - recharge**

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | recharge |
| **UseCase ID:** | UC8 |
| **Brief Description:** | |
| **Involved Actor:** | Driver |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

**UC9 - openPark**

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | openPark |
| **UseCase ID:** | UC9 |
| **Brief Description:** | |
| **Involved Actor:** | ParkManager |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

**UC10 - closePark**

Use Case Description:

| | |
|---|---|
| **UseCase Name:** | closePark |
| **UseCase ID:** | UC10 |
| **Brief Description:** | |
| **Involved Actor:** | ParkManager |
| **Preconditions:** | |
| **Postconditions:** | |
| **Basic Path:** | |
| **Alternative Path:** | |

## 3.1.2 System Operation

**OP1 - createPark**

| | |
|---|---|
| **Operation Name:** | createPark |
| **Operation ID:** | OP1 |
| **Description:** | |
| **Service:** | ManageParkCRUDService |
| **Input:** | 1. name: *id*, type: Integer<br>2. name: *name*, type: String<br>3. name: *location*, type: String<br>4. name: *smallprice*, type: Real<br>5. name: *largeprice*, type: Real<br>6. name: *motoprice*, type: Real |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *par* in the instance set of class Park. *par* represents an object of class Park, and *par* meets:<br>The attribute *Id* of the object *par* is equal to *id* |
| **Preconditions:** | The object *park* doesn't exist |
| **Postconditions:** | 1. *par* represented the object of class Park<br>2. The object *par* was created<br>3. The attribute *Id* of the object *par* became *id*<br>4. The attribute *Name* of the object *par* became *name*<br>5. The attribute *Location* of the object *par* became *location*<br>6. The attribute *SmallPrice* of the object *par* became *smallprice*<br>7. The attribute *LargePrice* of the object *par* became *largeprice*<br>8. The attribute *MotoPrice* of the object *par* became *motoprice*<br>9. The object *par* was put into the instance set of class Park<br>10. The return value was **true** |

Contract of **createPark**:

```
Contract  ManageParkCRUDService::createPark(id : Integer, name : String,
location : String, smallprice : Real, largeprice : Real, motoprice : Real) :
Boolean {
        /* definition: find specific Park instance by id */
        definition:
            park:Park = Park.allInstance()->any(par:Park | par.Id = id)
        /* precondition: the instance park was not found in the system */
        precondition:
            park.oclIsUndefined() = true
        /* postcondition:
         * A Park instance par was created.
         * all properties of par became the same values as inputs.
         */
        postcondition:
            let par:Park in
            par.oclIsNew() and
            par.Id = id and
            par.Name = name and
            par.Location = location and
            par.SmallPrice = smallprice and
            par.LargePrice = largeprice and
            par.MotoPrice = motoprice and
            Park.allInstance()->includes(par) and
            result = true
}
```

### OP2 - queryPark

| | |
|---|---|
| **Operation Name:** | queryPark |
| **Operation ID:** | OP2 |
| **Description:** | |
| **Service:** | ManageParkCRUDService |
| **Input:** | name: *id*, type: Integer |
| **Output Type:** | Park |
| **Definition:** | *park* is the object *par* in the instance set of class Park. *par* represents an object of class Park, and *par* meets:<br>The attribute *Id* of the object *par* is equal to *id* |
| **Preconditions:** | The object *park* exists |
| **Postconditions:** | The return value was *park* |

Contract of **queryPark**:

```
Contract  ManageParkCRUDService::queryPark(id : Integer) : Park {
        /* definition: find specific Park instance by id */
        definition:
            park:Park = Park.allInstance()->any(par:Park | par.Id = id)
        /* precondition: the instance park was found in the system */
        precondition:
            park.oclIsUndefined() = false
        /* postcondition: return found the instance park */
        postcondition:
            result = park
}
```

### OP3 - modifyPark

| | |
|---|---|
| **Operation Name:** | modifyPark |
| **Operation ID:** | OP3 |
| **Description:** | |
| **Service:** | ManageParkCRUDService |
| **Input:** | 1. name: *id*, type: Integer<br>2. name: *name*, type: String<br>3. name: *location*, type: String<br>4. name: *smallprice*, type: Real<br>5. name: *largeprice*, type: Real<br>6. name: *motoprice*, type: Real |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *par* in the instance set of class Park. *par* represents an object of class Park, and *par* meets:<br>The attribute *Id* of the object *par* is equal to *id* |
| **Preconditions:** | The object *park* exists |
| **Postconditions:** | 1. The attribute *Id* of the object *park* became *id*<br>2. The attribute *Name* of the object *park* became *name*<br>3. The attribute *Location* of the object *park* became *location*<br>4. The attribute *SmallPrice* of the object *park* became *smallprice*<br>5. The attribute *LargePrice* of the object *park* became *largeprice*<br>6. The attribute *MotoPrice* of the object *park* became *motoprice*<br>7. The return value was **true** |

Contract of **modifyPark**:

```
Contract  ManageParkCRUDService::modifyPark(id : Integer, name : String,
location : String, smallprice : Real, largeprice : Real, motoprice : Real) :
Boolean {
        /* definition: find specific Park instance by id */
        definition:
            park:Park = Park.allInstance()->any(par:Park | par.Id = id)
        /* precondition: the instance park was found in the system */
        precondition:
            park.oclIsUndefined() = false
        /* postcondition: all properties of par became the same values as
inputs. */
        postcondition:
            park.Id = id and
            park.Name = name and
            park.Location = location and
            park.SmallPrice = smallprice and
            park.LargePrice = largeprice and
            park.MotoPrice = motoprice and
            result = true
}
```

### OP4 - deletePark

| | |
|---|---|
| **Operation Name:** | deletePark |
| **Operation ID:** | OP4 |
| **Description:** | |
| **Service:** | ManageParkCRUDService |
| **Input:** | name: *id*, type: Integer |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *par* in the instance set of class Park. *par* represents an object of class Park, and *par* meets:<br>The attribute *Id* of the object *par* is equal to *id* |
| **Preconditions:** | 1. The object *park* exists<br>2. The object *park* is in the instance set of class Park |
| **Postconditions:** | 1. The object *park* was deleted from the instance set of class Park<br>2. The return value was **true** |

Contract of **deletePark**:

```
Contract  ManageParkCRUDService::deletePark(id : Integer) : Boolean {
        /* definition: find specific Park instance by id */
        definition:
            park:Park = Park.allInstance()->any(par:Park | par.Id = id)
        /* precondition: the instance park was found in the system */
        precondition:
            park.oclIsUndefined() = false and
            Park.allInstance()->includes(park)
        /* postcondition: the instance park was deleted from the system */
        postcondition:
            Park.allInstance()->excludes(park) and
            result = true
}
```

### OP5 - createVehicle

| | |
|---|---|
| **Operation Name:** | createVehicle |
| **Operation ID:** | OP5 |
| **Description:** | |
| **Service:** | ManageVehicleCRUDService |
| **Input:** | 1. name: *id*, type: Integer<br>2. name: *platenumber*, type: String<br>3. name: *type*, type: [SMALL\|LARGE\|MOTOCYCLE\|SPECIAL] |
| **Output Type:** | Boolean |
| **Definition:** | *vehicle* is the object *veh* in the instance set of class Vehicle. *veh* represents an object of class Vehicle, and *veh* meets:<br>The attribute *Id* of the object *veh* is equal to *id* |
| **Preconditions:** | The object *vehicle* doesn't exist |
| **Postconditions:** | 1. *veh* represented the object of class Vehicle<br>2. The object *veh* was created<br>3. The attribute *Id* of the object *veh* became *id*<br>4. The attribute *PlateNumber* of the object *veh* became *platenumber*<br>5. The attribute *Type* of the object *veh* became *type*<br>6. The object *veh* was put into the instance set of class Vehicle<br>7. The return value was **true** |

Contract of **createVehicle**:

```
Contract  ManageVehicleCRUDService::createVehicle(id : Integer, platenumber :
String, type : VehicleType[SMALL|LARGE|MOTOCYCLE|SPECIAL]) : Boolean {
        /* definition: find specific Vehicle instance by id */
        definition:
            vehicle:Vehicle = Vehicle.allInstance()->any(veh:Vehicle | veh.Id =
id)
        /* precondition: the instance vehicle was not found in the system */
        precondition:
            vehicle.oclIsUndefined() = true
        /* postcondition:
         * A Vehicle instance veh was created.
         * all properties of veh became the same values as inputs.
         */
        postcondition:
            let veh:Vehicle in
            veh.oclIsNew() and
            veh.Id = id and
            veh.PlateNumber = platenumber and
            veh.Type = type and
            Vehicle.allInstance()->includes(veh) and
            result = true
}
```

### OP6 - queryVehicle

| | |
|---|---|
| **Operation Name:** | queryVehicle |
| **Operation ID:** | OP6 |
| **Description:** | |
| **Service:** | ManageVehicleCRUDService |
| **Input:** | name: *id*, type: Integer |
| **Output Type:** | Vehicle |
| **Definition:** | *vehicle* is the object *veh* in the instance set of class Vehicle. *veh* represents an object of class Vehicle, and *veh* meets:<br>The attribute *Id* of the object *veh* is equal to *id* |
| **Preconditions:** | The object *vehicle* exists |
| **Postconditions:** | The return value was *vehicle* |

Contract of **queryVehicle**:

```
Contract  ManageVehicleCRUDService::queryVehicle(id : Integer) : Vehicle {
        /* definition: find specific Vehicle instance by id */
        definition:
            vehicle:Vehicle = Vehicle.allInstance()->any(veh:Vehicle | veh.Id =
id)
        /* precondition: the instance vehicle was found in the system */
        precondition:
            vehicle.oclIsUndefined() = false
        /* postcondition: return found the instance vehicle */
        postcondition:
            result = vehicle
}
```

### OP7 - modifyVehicle

| | |
|---|---|
| **Operation Name:** | modifyVehicle |
| **Operation ID:** | OP7 |
| **Description:** | |
| **Service:** | ManageVehicleCRUDService |
| **Input:** | 1. name: *id*, type: Integer<br>2. name: *platenumber*, type: String<br>3. name: *type*, type: [SMALL\|LARGE\|MOTOCYCLE\|SPECIAL] |
| **Output Type:** | Boolean |
| **Definition:** | *vehicle* is the object *veh* in the instance set of class Vehicle. *veh* represents an object of class Vehicle, and *veh* meets:<br>The attribute *Id* of the object *veh* is equal to *id* |
| **Preconditions:** | The object *vehicle* exists |
| **Postconditions:** | 1. The attribute *Id* of the object *vehicle* became *id*<br>2. The attribute *PlateNumber* of the object *vehicle* became *platenumber*<br>3. The attribute *Type* of the object *vehicle* became *type*<br>4. The return value was **true** |

Contract of **modifyVehicle**:

```
Contract  ManageVehicleCRUDService::modifyVehicle(id : Integer, platenumber :
String, type : VehicleType[SMALL|LARGE|MOTOCYCLE|SPECIAL]) : Boolean {
        /* definition: find specific Vehicle instance by id */
        definition:
            vehicle:Vehicle = Vehicle.allInstance()->any(veh:Vehicle | veh.Id =
id)
```

```
        /* precondition: the instance vehicle was found in the system */
        precondition:
            vehicle.oclIsUndefined() = false
        /* postcondition: all properties of veh became the same values as
inputs. */
        postcondition:
            vehicle.Id = id and
            vehicle.PlateNumber = platenumber and
            vehicle.Type = type and
            result = true
}
```

### OP8 - deleteVehicle

| | |
|---|---|
| **Operation Name:** | deleteVehicle |
| **Operation ID:** | OP8 |
| **Description:** | |
| **Service:** | ManageVehicleCRUDService |
| **Input:** | name: *id*, type: Integer |
| **Output Type:** | Boolean |
| **Definition:** | *vehicle* is the object *veh* in the instance set of class Vehicle. *veh* represents an object of class Vehicle, and *veh* meets:<br>The attribute *Id* of the object *veh* is equal to *id* |
| **Preconditions:** | 1. The object *vehicle* exists<br>2. The object *vehicle* is in the instance set of class Vehicle |
| **Postconditions:** | 1. The object *vehicle* was deleted from the instance set of class Vehicle<br>2. The return value was **true** |

Contract of **deleteVehicle**:

```
Contract  ManageVehicleCRUDService::deleteVehicle(id : Integer) : Boolean {
        /* definition: find specific Vehicle instance by id */
        definition:
            vehicle:Vehicle = Vehicle.allInstance()->any(veh:Vehicle | veh.Id =
id)
        /* precondition: the instance vehicle was found in the system */
        precondition:
            vehicle.oclIsUndefined() = false and
            Vehicle.allInstance()->includes(vehicle)
        /* postcondition: the instance vehicle was deleted from the system */
        postcondition:
            Vehicle.allInstance()->excludes(vehicle) and
            result = true
}
```

#### OP9 - automaticEntry

| | |
|---|---|
| **Operation Name:** | automaticEntry |
| **Operation ID:** | OP9 |
| **Description:** | |
| **Service:** | AutomaticEntryService |
| **Input:** | 1. name: *plateNumber*, type: String<br>2. name: *time*, type: LocalDate |
| **Output Type:** | Boolean |
| **Definition:** | *isParking* is the object *r* in the instance set of class ParkRecord. *r* represents an object of class ParkRecord, and *r* meets:<br>The attribute *PlateNumber* of the object *r* is equal to *plateNumber*<br>The attribute *IsParking* of the object *r* is equal to **true** |
| **Preconditions:** | 1. The object *CurrentPark* exists<br>2. The attribute *IsOpened* of the object *CurrentPark* is equal to **true**<br>3. The object *isParking* doesn't exist |
| **Postconditions:** | 1. *parkRecord* represented the object of class ParkRecord<br>2. The object *parkRecord* was created<br>3. The attribute *PlateNumber* of the object *parkRecord* became *plateNumber*<br>4. The attribute *EntryTime* of the object *parkRecord* became *time*<br>5. The attribute *IsParking* of the object *parkRecord* became **true**<br>6. The attribute *Id* of the object *parkRecord* became the previous value of temporary variable RecordID plus **1**<br>7. The value of temporary variable RecordID became the previous value of temporary variable RecordID plus **1**<br>8. The object *parkRecord* was put into the instance set of class ParkRecord<br>9. The object *CurrentPark* was linked to the object *parkRecord* by *OwningRecords*<br>10. The return value was **true** |

Contract of **automaticEntry**:

```
Contract AutomaticEntryService::automaticEntry(plateNumber : String, time :
Date) : Boolean {
        definition:
            isParking:ParkRecord = ParkRecord.allInstance()-
>any(r:ParkRecord|r.PlateNumber=plateNumber and r.IsParking=true)
        precondition:
            CurrentPark.oclIsUndefined() = false and
            CurrentPark.IsOpened = true and
            isParking.oclIsUndefined() = true
        postcondition:
            let parkRecord:ParkRecord in
            parkRecord.oclIsNew() and
            parkRecord.PlateNumber = plateNumber and
            parkRecord.EntryTime = time and
            parkRecord.IsParking = true and
            parkRecord.Id = self.RecordID@pre+1 and
            self.RecordID = self.RecordID@pre+1 and
            ParkRecord.allInstance()->includes(parkRecord) and
            CurrentPark.OwningRecords->includes(parkRecord) and
            result = true
}
```

### OP10 - scanPlateNumber

| | |
|---|---|
| **Operation Name:** | scanPlateNumber |
| **Operation ID:** | OP10 |
| **Description:** | |
| **Service:** | AutomaticOutService |
| **Input:** | 1. name: *plateNumber*, type: String<br>2. name: *time*, type: LocalDate |
| **Output Type:** | Real |
| **Definition:** | 1. *parkRecord* is the object *r* in the instance set of class ParkRecord. *r* represents an object of class ParkRecord, and *r* meets:<br>The attribute *PlateNumber* of the object *r* is equal to *plateNumber*<br>The attribute *IsParking* of the object *r* is equal to **true**<br>2. *vehicle* is the object *v* in the instance set of class Vehicle. *v* represents an object of class Vehicle, and *v* meets:<br>The attribute *PlateNumber* of the object *v* is equal to *plateNumber* |
| **Preconditions:** | 1. The object *parkRecord* exists<br>2. The attribute *IsParking* of the object *parkRecord* is equal to **true** |

| | |
|---|---|
| **Postconditions:** | 1. The attribute *IsParking* of the object *parkRecord* became **false**<br>2. The attribute *Type* of the object *parkRecord* became the attribute *Type* of the object *vehicle*<br>3. If the attribute *Type* of the object *vehicle* was equal to **SPECIAL**, take the following as postcondition(s):<br>ERROR12<br>Otherwise, take the following as postcondition(s):<br>If the attribute *Type* of the object *vehicle* was equal to **SMALL**, take the following as postcondition(s):<br>ERROR12<br>Otherwise, take the following as postcondition(s):<br>If the attribute *Type* of the object *vehicle* was equal to **LARGE**, take the following as postcondition(s):<br>ERROR12<br>Otherwise, take the following as postcondition(s):<br>ERROR12<br>4. The return value was *CurrentPayment* |

Contract of **scanPlateNumber**:

```
Contract AutomaticOutService::scanPlateNumber(plateNumber : String, time : Date)
: Real {
        definition:
            parkRecord:ParkRecord = ParkRecord.allInstance()-
>any(r:ParkRecord|r.PlateNumber=plateNumber and r.IsParking=true),
            vehicle:Vehicle = Vehicle.allInstance()-
>any(v:Vehicle|v.PlateNumber=plateNumber)
        precondition:
            parkRecord.oclIsUndefined() = false and
            parkRecord.IsParking = true
        postcondition:
            parkRecord.IsParking = false and
            parkRecord.Type = vehicle.Type and
            if
                vehicle.Type = VehicleType::SPECIAL
            then
                CurrentPayment = 0
            else
                if
                    vehicle.Type = VehicleType::SMALL
                then
                    // CurrentPayment = parkRecord.SmallPrice*(time.DayOfYear-
parkRecordtime.DayOfYear)
                    CurrentPayment = 1
                else
                    if
                        vehicle.Type = VehicleType::LARGE
                    then
                        CurrentPayment = 2
                    else
                        CurrentPayment = 3
                    endif
                endif
            endif and
            result = CurrentPayment
}
```

### OP11 - onlinePay

| | |
|---|---|
| **Operation Name:** | onlinePay |
| **Operation ID:** | OP11 |
| **Description:** | |
| **Service:** | AutomaticOutService |
| **Input:** | name: *memberID*, type: Integer |
| **Output Type:** | Boolean |
| **Definition:** | *member* is the object *m* in the instance set of class Member. *m* represents an object of class Member, and *m* meets:<br>The attribute *Id* of the object *m* is equal to *memberID* |
| **Preconditions:** | 1. The object *member* exists<br>2. The attribute *Balance* of the object *member* is greater than or equal to *CurrentPayment* |
| **Postconditions:** | 1. The attribute *Balance* of the object *member* became its previous value minus *CurrentPayment*<br>2. The return value was **true** |

Contract of **onlinePay**:

```
Contract AutomaticOutService::onlinePay(memberID:Integer) : Boolean {
        definition:
            member:Member = Member.allInstance()->any(m:Member|m.Id=memberID)
        precondition:
            member.oclIsUndefined() = false and
            member.Balance >= CurrentPayment
        postcondition:
            member.Balance = member.Balance@pre-CurrentPayment and
            result = true
}
```

### OP12 - setSmallPrice

| | |
|---|---|
| **Operation Name:** | setSmallPrice |
| **Operation ID:** | OP12 |
| **Description:** | |
| **Service:** | SetPriceService |
| **Input:** | 1. name: *parkID*, type: Integer<br>2. name: *price*, type: Real |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *p* in the instance set of class Park. *p* represents an object of class Park, and *p* meets:<br>The attribute *Id* of the object *p* is equal to *parkID* |
| **Preconditions:** | The object *park* exists |
| **Postconditions:** | 1. The attribute *SmallPrice* of the object *park* became *price*<br>2. The return value was **true** |

Contract of **setSmallPrice**:

```
Contract SetPriceService::setSmallPrice(parkID : Integer, price : Real) :
Boolean {
        definition:
            park:Park = Park.allInstance()->any(p:Park|p.Id=parkID)
        precondition:
            park.oclIsUndefined() = false
        postcondition:
            park.SmallPrice = price and
            result = true
}
```

**OP13 - setLargePrice**

| | |
|---|---|
| **Operation Name:** | setLargePrice |
| **Operation ID:** | OP13 |
| **Description:** | |
| **Service:** | SetPriceService |
| **Input:** | 1. name: *parkID*, type: Integer<br>2. name: *price*, type: Real |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *p* in the instance set of class Park. *p* represents an object of class Park, and *p* meets:<br>The attribute *Id* of the object *p* is equal to *parkID* |
| **Preconditions:** | The object *park* exists |
| **Postconditions:** | 1. The attribute *LargePrice* of the object *park* became *price*<br>2. The return value was **true** |

Contract of **setLargePrice**:

```
Contract SetPriceService::setLargePrice(parkID : Integer, price : Real) :
Boolean {
        definition:
            park:Park = Park.allInstance()->any(p:Park|p.Id=parkID)
        precondition:
            park.oclIsUndefined() = false
        postcondition:
            park.LargePrice = price and
            result = true
}
```

**OP14 - setMotoPrice**

| | |
|---|---|
| **Operation Name:** | setMotoPrice |
| **Operation ID:** | OP14 |
| **Description:** | |
| **Service:** | SetPriceService |
| **Input:** | 1. name: *parkID*, type: Integer<br>2. name: *price*, type: Real |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *p* in the instance set of class Park. *p* represents an object of class Park, and *p* meets:<br>The attribute *Id* of the object *p* is equal to *parkID* |
| **Preconditions:** | The object *park* exists |
| **Postconditions:** | 1. The attribute *MotoPrice* of the object *park* became *price*<br>2. The return value was **true** |

Contract of **setMotoPrice**:

```
Contract SetPriceService::setMotoPrice(parkID : Integer, price : Real) : Boolean
{
        definition:
            park:Park = Park.allInstance()->any(p:Park|p.Id=parkID)
        precondition:
            park.oclIsUndefined() = false
        postcondition:
            park.MotoPrice = price and
            result = true
}
```

### OP15 - getHistoryByMember

| | |
|---|---|
| **Operation Name:** | getHistoryByMember |
| **Operation ID:** | OP15 |
| **Description:** | |
| **Service:** | GetHistoryService |
| **Input:** | name: *memberID*, type: Integer |
| **Output Type:** | Set of ParkRecord |
| **Definition:** | *member* is the object *m* in the instance set of class Member. *m* represents an object of class Member, and *m* meets:<br>The attribute *Id* of the object *m* is equal to *memberID* |
| **Preconditions:** | The object *member* exists |
| **Postconditions:** | The return value was the set of class ParkRecord, including all *r* in the instance set of class ParkRecord. *r* represented an object of class ParkRecord, and *r* meet:<br>At least one *v* existed in all objects which *member* was linked to by *OwningVehicles*. *v* represented an object of class Vehicle, and *v* meet:<br>The attribute *PlateNumber* of the object *v* was equal to the attribute *PlateNumber* of the object *r* |

Contract of **getHistoryByMember**:

```
Contract GetHistoryService::getHistoryByMember(memberID : Integer) :
Set(ParkRecord) {
        definition:
            member:Member = Member.allInstance()->any(m:Member|m.Id=memberID)
        precondition:
            member.oclIsUndefined() = false
        postcondition:
            result = ParkRecord.allInstance()-
>select(r:ParkRecord|member.OwningVehicles-
>exists(v:Vehicle|v.PlateNumber=r.PlateNumber))
}
```

### OP16 - getHistoryByOutTime

| | |
|---|---|
| **Operation Name:** | getHistoryByOutTime |
| **Operation ID:** | OP16 |
| **Description:** | |
| **Service:** | GetHistoryService |
| **Input:** | 1. name: *from*, type: LocalDate<br>2. name: *to*, type: LocalDate |
| **Output Type:** | Set of ParkRecord |
| **Preconditions:** | None |
| **Postconditions:** | The return value was the set of class ParkRecord, including all *r* in the instance set of class ParkRecord. *r* represented an object of class ParkRecord, and *r* meet:<br>The attribute *OutTime* of the object *r* was after *from*<br>The attribute *OutTime* of the object *r* was before *to* |

Contract of **getHistoryByOutTime**:

```
Contract GetHistoryService::getHistoryByOutTime(from : Date, to : Date) :
Set(ParkRecord) {
        precondition:
            true
        postcondition:
            result = ParkRecord.allInstance()-
>select(r:ParkRecord|r.OutTime.isAfter(from) and r.OutTime.isBefore(to))
}
```

**OP17 - getHistoryByEntryTime**

| | |
|---|---|
| **Operation Name:** | getHistoryByEntryTime |
| **Operation ID:** | OP17 |
| **Description:** | |
| **Service:** | GetHistoryService |
| **Input:** | 1. name: *from*, type: LocalDate<br>2. name: *to*, type: LocalDate |
| **Output Type:** | Set of ParkRecord |
| **Preconditions:** | None |
| **Postconditions:** | The return value was the set of class ParkRecord, including all *r* in the instance set of class ParkRecord. *r* represented an object of class ParkRecord, and *r* meet:<br>The attribute *EntryTime* of the object *r* was after *from*<br>The attribute *EntryTime* of the object *r* was before *to* |

Contract of **getHistoryByEntryTime**:

```
Contract GetHistoryService::getHistoryByEntryTime(from : Date, to : Date) :
Set(ParkRecord) {
        precondition:
            true
        postcondition:
            result = ParkRecord.allInstance()-
>select(r:ParkRecord|r.EntryTime.isAfter(from) and r.EntryTime.isBefore(to))
}
```

### OP18 - getHistoryByPlateNumber

|  |  |
|---|---|
| **Operation Name:** | getHistoryByPlateNumber |
| **Operation ID:** | OP18 |
| **Description:** |  |
| **Service:** | GetHistoryService |
| **Input:** | name: *plateNumber*, type: String |
| **Output Type:** | Set of ParkRecord |
| **Preconditions:** | None |
| **Postconditions:** | The return value was the set of class ParkRecord, including all *r* in the instance set of class ParkRecord. *r* represented an object of class ParkRecord, and *r* meet:<br>The attribute *PlateNumber* of the object *r* was equal to *plateNumber* |

Contract of **getHistoryByPlateNumber**:

```
Contract GetHistoryService::getHistoryByPlateNumber(plateNumber : String) :
Set(ParkRecord) {
        precondition:
            true
        postcondition:
            result = ParkRecord.allInstance()-
>select(r:ParkRecord|r.PlateNumber=plateNumber)
}
```

### OP19 - registerMember

| | |
|---|---|
| **Operation Name:** | registerMember |
| **Operation ID:** | OP19 |
| **Description:** | |
| **Service:** | RegisterService |
| **Input:** | 1. name: *memberID*, type: Integer<br>2. name: *name*, type: String |
| **Output Type:** | Boolean |
| **Definition:** | *member* is the object *m* in the instance set of class Member. *m* represents an object of class Member, and *m* meets:<br>The attribute *Id* of the object *m* is equal to *memberID* |
| **Preconditions:** | The object *member* doesn't exist |
| **Postconditions:** | 1. *newMember* represented the object of class Member<br>2. The object *newMember* was created<br>3. The attribute *Id* of the object *newMember* became *memberID*<br>4. The attribute *Name* of the object *newMember* became *name*<br>5. The object *newMember* was put into the instance set of class Member<br>6. ERROR12<br>7. The return value was **true** |

Contract of **registerMember**:

```
Contract RegisterService::registerMember(memberID : Integer, name : String) :
Boolean {
        definition:
            member:Member = Member.allInstance()->any(m:Member|m.Id=memberID)
        precondition:
            member.oclIsUndefined() = true
        postcondition:
            let newMember:Member in
            newMember.oclIsNew() and
            newMember.Id = memberID and
            newMember.Name = name and
            Member.allInstance()->includes(newMember) and
            CurrentMember = newMember and
            result = true
}
```

**OP20 - registerVehicle**

| | |
|---|---|
| **Operation Name:** | registerVehicle |
| **Operation ID:** | OP20 |
| **Description:** | |
| **Service:** | RegisterService |
| **Input:** | 1. name: *vehicleID*, type: Integer<br>2. name: *plateNumber*, type: String<br>3. name: *type*, type: [SMALL\|LARGE\|MOTO\|SPECIAL] |
| **Output Type:** | Boolean |
| **Definition:** | *vehicle* is the object *v* in the instance set of class Vehicle. *v* represents an object of class Vehicle, and *v* meets:<br>The attribute *Id* of the object *v* is equal to *vehicleID* |
| **Preconditions:** | 1. The object *CurrentMember* exists<br>2. The object *vehicle* doesn't exist |
| **Postconditions:** | 1. *newVehicle* represented the object of class Vehicle<br>2. The object *newVehicle* was created<br>3. The attribute *Id* of the object *newVehicle* became *vehicleID*<br>4. The attribute *PlateNumber* of the object *newVehicle* became *plateNumber*<br>5. The attribute *Type* of the object *newVehicle* became *type*<br>6. The object *newVehicle* was put into the instance set of class Vehicle<br>7. The object *CurrentMember* was linked to the object *newVehicle* by *OwningVehicles*<br>8. The object *newVehicle* was linked to the object *CurrentMember* by *OwnedMember*<br>9. The return value was **true** |

Contract of **registerVehicle**:

```
Contract RegisterService::registerVehicle(vehicleID : Integer, plateNumber :
String, type:VehicleType[SMALL|LARGE|MOTO|SPECIAL]) : Boolean {
        definition:
            vehicle:Vehicle = Vehicle.allInstance()-
>any(v:Vehicle|v.Id=vehicleID)
        precondition:
            CurrentMember.oclIsUndefined() = false and
            vehicle.oclIsUndefined() = true
        postcondition:
            let newVehicle:Vehicle in
            newVehicle.oclIsNew() and
            newVehicle.Id = vehicleID and
            newVehicle.PlateNumber = plateNumber and
            newVehicle.Type = type and
            Vehicle.allInstance()->includes(newVehicle) and
            CurrentMember.OwningVehicles->includes(newVehicle) and
            newVehicle.OwnedMember = CurrentMember and
            result = true
}
```

### OP21 - recharge

| | |
|---|---|
| **Operation Name:** | recharge |
| **Operation ID:** | OP21 |
| **Description:** | |
| **Service:** | RechargeService |
| **Input:** | 1. name: *memberID*, type: Integer<br>2. name: *amount*, type: Real |
| **Output Type:** | Boolean |
| **Definition:** | *member* is the object *m* in the instance set of class Member. *m* represents an object of class Member, and *m* meets:<br>The attribute *Id* of the object *m* is equal to *memberID* |
| **Preconditions:** | The object *member* exists |
| **Postconditions:** | 1. The attribute *Balance* of the object *member* became its previous value plus *amount*<br>2. The return value was **true** |

Contract of **recharge**:

```
Contract RechargeService::recharge(memberID : Integer, amount : Real) : Boolean
{
        definition:
            member:Member = Member.allInstance()->any(m:Member|m.Id=memberID)
        precondition:
            member.oclIsUndefined() = false
        postcondition:
            member.Balance = member.Balance@pre+amount and
            result = true
}
```

### OP22 - manuallyAllowOut

| | |
|---|---|
| **Operation Name:** | manuallyAllowOut |
| **Operation ID:** | OP22 |
| **Description:** | |
| **Service:** | ManuallyAllowOutService |
| **Input:** | 1. name: *plateNumber*, type: String<br>2. name: *time*, type: LocalDate |
| **Output Type:** | Boolean |
| **Definition:** | *parkRecord* is the object *r* in the instance set of class ParkRecord. *r* represents an object of class ParkRecord, and *r* meets:<br>The attribute *PlateNumber* of the object *r* is equal to *plateNumber*<br>The attribute *IsParking* of the object *r* is equal to **true** |
| **Preconditions:** | 1. The object *parkRecord* exists<br>2. The attribute *IsParking* of the object *parkRecord* is equal to **true** |
| **Postconditions:** | 1. The attribute *IsParking* of the object *parkRecord* became **false**<br>2. The return value was **true** |

Contract of **manuallyAllowOut**:

```
Contract ManuallyAllowOutService::manuallyAllowOut(plateNumber : String, time :
Date) : Boolean {
        definition:
            parkRecord:ParkRecord = ParkRecord.allInstance()-
>any(r:ParkRecord|r.PlateNumber=plateNumber and r.IsParking=true)
        precondition:
            parkRecord.oclIsUndefined() = false and
            parkRecord.IsParking = true
        postcondition:
            parkRecord.IsParking = false and
            result = true
}
```

### OP23 - manuallyAllowEntry

| | |
|---|---|
| **Operation Name:** | manuallyAllowEntry |
| **Operation ID:** | OP23 |
| **Description:** | |
| **Service:** | ManuallyAllowEntryService |
| **Input:** | 1. name: *plateNumber*, type: String<br>2. name: *type*, type: [SMALL\|LARGE\|MOTO\|SPECIAL]<br>3. name: *time*, type: LocalDate |
| **Output Type:** | Boolean |
| **Definition:** | *vehicle* is the object *v* in the instance set of class Vehicle. *v* represents an object of class Vehicle, and *v* meets:<br>The attribute *PlateNumber* of the object *v* is equal to *plateNumber* |
| **Preconditions:** | 1. The object *CurrentPark* exists<br>2. The attribute *IsOpened* of the object *CurrentPark* is equal to **true** |
| **Postconditions:** | 1. *parkRecord* represented the object of class ParkRecord<br>2. The object *parkRecord* was created<br>3. The attribute *Id* of the object *parkRecord* became the size of *CurrentPark* plus **1**<br>4. The attribute *PlateNumber* of the object *parkRecord* became *plateNumber*<br>5. The attribute *Type* of the object *parkRecord* became *type*<br>6. The attribute *EntryTime* of the object *parkRecord* became *time*<br>7. The attribute *IsParking* of the object *parkRecord* became **true**<br>8. The object *parkRecord* was put into the instance set of class ParkRecord<br>9. The object *CurrentPark* was linked to the object *parkRecord* by *OwningRecords*<br>10. The return value was **true** |

Contract of **manuallyAllowEntry**:

```
Contract ManuallyAllowEntryService::manuallyAllowEntry(plateNumber : String,
type : VehicleType[SMALL|LARGE|MOTO|SPECIAL], time : Date) : Boolean {
        definition:
            vehicle:Vehicle = Vehicle.allInstance()-
>any(v:Vehicle|v.PlateNumber=plateNumber)
```

```
        precondition:
            CurrentPark.oclIsUndefined() = false and
            CurrentPark.IsOpened = true
        postcondition:
            let parkRecord:ParkRecord in
            parkRecord.oclIsNew() and
            parkRecord.Id = CurrentPark.OwningRecords@pre.size()+1 and
            parkRecord.PlateNumber = plateNumber and
            parkRecord.Type = type and
            parkRecord.EntryTime = time and
            parkRecord.IsParking = true and
            ParkRecord.allInstance()->includes(parkRecord) and
            CurrentPark.OwningRecords->includes(parkRecord) and
            result = true
}
```

### OP24 - openPark

| | |
|---|---|
| **Operation Name:** | openPark |
| **Operation ID:** | OP24 |
| **Description:** | |
| **Service:** | ParkMSSystem |
| **Input:** | name: *parkID*, type: Integer |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *p* in the instance set of class Park. *p* represents an object of class Park, and *p* meets:<br>The attribute *Id* of the object *p* is equal to *parkID* |
| **Preconditions:** | 1. The object *park* exists<br>2. The attribute *IsOpened* of the object *park* is equal to **false** |
| **Postconditions:** | 1. The attribute *IsOpened* of the object *park* became **true**<br>2. ERROR12<br>3. The return value was **true** |

Contract of **openPark**:

```
Contract ParkMSSystem::openPark(parkID : Integer) : Boolean {
        definition:
            park:Park = Park.allInstance()->any(p:Park|p.Id=parkID)
        precondition:
            park.oclIsUndefined() = false and
            park.IsOpened = false
        postcondition:
            park.IsOpened = true and
            CurrentPark = park and
            result = true
}
```

### OP25 - closePark

| | |
|---|---|
| **Operation Name:** | closePark |
| **Operation ID:** | OP25 |
| **Description:** | |
| **Service:** | ParkMSSystem |
| **Input:** | name: *parkID*, type: Integer |
| **Output Type:** | Boolean |
| **Definition:** | *park* is the object *p* in the instance set of class Park. *p* represents an object of class Park, and *p* meets:<br>The attribute *Id* of the object *p* is equal to *parkID* |
| **Preconditions:** | 1. The object *park* exists<br>2. The attribute *IsOpened* of the object *park* is equal to **true** |
| **Postconditions:** | 1. The attribute *IsOpened* of the object *park* became **false**<br>2. The return value was **true** |

Contract of **closePark**:

```
Contract ParkMSSystem::closePark(parkID : Integer) : Boolean {
        definition:
            park:Park = Park.allInstance()->any(p:Park|p.Id=parkID)
        precondition:
            park.oclIsUndefined() = false and
            park.IsOpened = true
        postcondition:
            park.IsOpened = false and
            result = true
}
```

# 3.2 Database requirements

## 3.2.1 Entity Analysis

### Conceptual Class Diagram

Conceptual Class Diagram

**E1 - ParkRecord**

| | | |
|---|---|---|
| **Entity Name:** | ParkRecord | |
| **Entity ID:** | E1 | |
| **Entity Description:** | | |
| **Attribute Name** | **Attribute Type** | **Attribute Description** |
| Id | Integer | The Id of ParkRecord |
| PlateNumber | String | The PlateNumber of ParkRecord |
| Type | [SMALL\|LARGE\|MOTOCYCLE\|SPECIAL] | The Type of ParkRecord |
| EntryTime | LocalDate | The EntryTime of ParkRecord |
| OutTime | LocalDate | The OutTime of ParkRecord |
| IsParking | Boolean | The IsParking of ParkRecord |
| TotalPayment | Real | The TotalPayment of ParkRecord |
| Description | String | The Description of ParkRecord |
| **Relationship Name** | **Related Entity** | **Relationship Type** |
| OwnedPark | Park | Association: One-to-One |

**E2 - Park**

|  |  |  |
|---|---|---|
| **Entity Name:** | Park | |
| **Entity ID:** | E2 | |
| **Entity Description:** | | |
| **Attribute Name** | **Attribute Type** | **Attribute Description** |
| Id | Integer | The Id of Park |
| Name | String | The Name of Park |
| Location | String | The Location of Park |
| SmallPrice | Real | The SmallPrice of Park |
| LargePrice | Real | The LargePrice of Park |
| MotoPrice | Real | The MotoPrice of Park |
| IsOpened | Boolean | The IsOpened of Park |
| **Relationship Name** | **Related Entity** | **Relationship Type** |
| OwningMembers | Member | Association: One-to-Many |
| OwningRecords | ParkRecord | Composition: One-to-Many |

### E3 - Vehicle

|  |  |  |
|---|---|---|
| **Entity Name:** | Vehicle | |
| **Entity ID:** | E3 | |
| **Entity Description:** | | |
| **Attribute Name** | **Attribute Type** | **Attribute Description** |
| Id | Integer | The Id of Vehicle |
| PlateNumber | String | The PlateNumber of Vehicle |
| Type | [SMALL\|LARGE\|MOTOCYCLE\|SPECIAL] | The Type of Vehicle |
| **Relationship Name** | **Related Entity** | **Relationship Type** |
| OwnedMember | Member | Association: One-to-One |

### E4 - Member

|  |  |  |
|---|---|---|
| **Entity Name:** | Member | |
| **Entity ID:** | E4 | |
| **Entity Description:** | | |
| **Attribute Name** | **Attribute Type** | **Attribute Description** |
| Id | Integer | The Id of Member |
| Name | String | The Name of Member |
| Balance | Real | The Balance of Member |
| **Relationship Name** | **Related Entity** | **Relationship Type** |
| Parks | Park | Aggregation: One-to-Many |
| OwningVehicles | Vehicle | Composition: One-to-Many |

#### 3.2.2 Other database requirements

This should specify the logical requirements for any information that is to be placed into a database. This may include the following:

- a) Types of information used by various functions;
- b) Frequency of use;
- c) Accessing capabilities;
- d) Integrity constraints;
- e) Data retention requirements.

## 3.3 Performance requirements

### 3.3.1 Static numerical requirements

This subsection should specify both the static and the dynamic numerical requirements placed on the software or on human interaction with the software as a whole. Static numerical requirements may include the following:

- a) The number of terminals to be supported;
- b) The number of simultaneous users to be supported;
- c) Amount and type of information to be handled.

### 3.3.2 Dynamic numerical requirements

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms.

For example,

- *95% of the transactions shall be processed in less than 1 s.*

rather than,

- *An operator shall not have to wait for the transaction to complete.*

NOTE:Numerical limits applied to one specifific function are normally specified as part of the processing subparagraph description of that function.

## 3.4 Usability requirements

Define usability and quality in use requirements and objectives for the software system that can include measurable effectiveness, efficiency, satisfaction criteria and avoidance of harm that could arise from use in specific contexts of use.

## 3.5 Interface requirements

### 3.5.1 User interfaces

This should specify the following:

- a) The logical characteristics of each interface between the software product and its users. This includes those configuration characteristics (e.g., required screen formats, page or window layouts, content of any reports or menus, or availability of programmable function keys) necessary to accomplish the software requirements.
- b) All the aspects of optimizing the interface with the person who must use the system. This may simply comprise a list of do's and don'ts on how the system will appear to the user. One example may be a requirement for the option of long or short error messages. Like all others, these requirements should be verifiable, e.g., "a clerk typist grade 4 can do function X in Z min after 1 h of training" rather than "a typist can do function X." (This may also be specified in the Software System Attributes under a section titled Ease of Use.)

### 3.5.2 Hardware interfaces

This should specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics (number of ports, instruction sets, etc.). It also covers such matters as what devices are to be supported, how they are to be supported, and protocols. For example, terminal support may specify full-screen support as opposed to line-by-line support.

### 3.5.3 Software interfaces

This should specify the use of other required software products (e.g., a data management system, an operating system, or a mathematical package), and interfaces with other application systems (e.g., the linkage between an accounts receivable system and a general ledger system). For each required software product, the following should be provided:

- a) Name;
- b) Mnemonic;
- c) Specification number;
- d) Version number;
- e) Source.

For each interface, the following should be provided:

- a) Discussion of the purpose of the interfacing software as related to this software product.
- b) Definition of the interface in terms of message content and format. It is not necessary to detail any well-documented interface, but a reference to the document defining the interface is required.

### 3.5.4 Communications interfaces

This should specify the various interfaces to communications such as local network protocols, etc.

## 3.6 Design constraints

Specify constraints on the system design imposed by external standards, regulatory requirements or project limitations.

### 3.6.1 Standards compliance

This subsection should specify the requirements derived from existing standards or regulations. They may include the following:

- a) Report format;
- b) Data naming;
- c) Accounting procedures;
- d) Audit tracing.

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

## 3.7 Software system attributes

### 3.7.1 Reliability

This should specify the factors required to establish the required reliability of the software system at time of delivery.

### 3.7.2 Availability

This should specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart.

### 3.7.3 Security

This should specify the factors that protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to

- a) Utilize certain cryptographical techniques;
- b) Keep specific log or history data sets;
- c) Assign certain functions to different modules;
- d) Restrict communications between some areas of the program;
- e) Check data integrity for critical variables.

### 3.7.4 Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

### 3.7.5 Portability

This should specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include the following:

- a) Percentage of components with host-dependent code;
- b) Percentage of code that is host dependent;
- c) Use of a proven portable language;
- d) Use of a particular compiler or language subset;
- e) Use of a particular operating system.

## 3.8 Supporting information

Additional supporting information to be considered includes:

- a) sample input/output formats, descriptions of cost analysis studies or results of user surveys;
- b) supporting or background information that can help the readers of the SRS;
- c) a description of the problems to be solved by the software; and
- d) special packaging instructions for the code and the media to meet security, export, initial loading or other requirements.

The SRS should explicitly state whether or not these information items are to be considered part of the requirements.

# 4 Verification

Provide the verification approaches and methods planned to qualify the software. The information items for verification are recommended to be given in a parallel manner with the information items in Section 3.

# 5 Appendices

## 5.1 Assumptions and dependencies

This subsection of the SRS should list each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the SRS would then have to change accordingly.

## 5.2 Apportioning of requirements

Apportion the software requirements to software elements. For requirements that will require implementation over multiple software elements, or when allocation to a software element is initially undefined, this should be so stated. A cross-reference table by function and software element should be used to summarize the apportionments.

Identify requirements that may be delayed until future versions of the system (e.g., blocks and/or increments).

## 5.3 Acronyms and abbreviations

This subsection should provide the acronyms and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendixes in the SRS or by reference to other documents.