Contents

User requirements

The company that wants to create platform for selling cars online needs a program to store data about the vehicles, their owners, and salons for further use.

Each vehicle has its’ model name, price, production year, horsepower value, kmage, features, owner or salon (as keeper). Each vehicle has its’ own manufacturer. There can be many features in vehicle. For all vehicle there are also a power unit (engine, battery, or hybrid engine). Each vehicle has its own oldness rank, which can be calculated, and coefficient reflects oldness level.

Every vehicle must have only one type of keeper: private owner or car salon. Both, owner and salon can have many vehicles as their property. Besides name, surname and phone number, owner has his/her sex (as enumeration). Salon has name, phone number and rating as attributes.

Every vehicle must have a service passport that gives an information about lastly done service. Service passport includes the date when it was done, service type (full or partial) and the date when service passport will lose its’ validity.

Service passport can be given only by specific service center. Every service center has its’ name, phone number and rating as a salon. Both salon and service center must have their address that are given as a complex attribute.

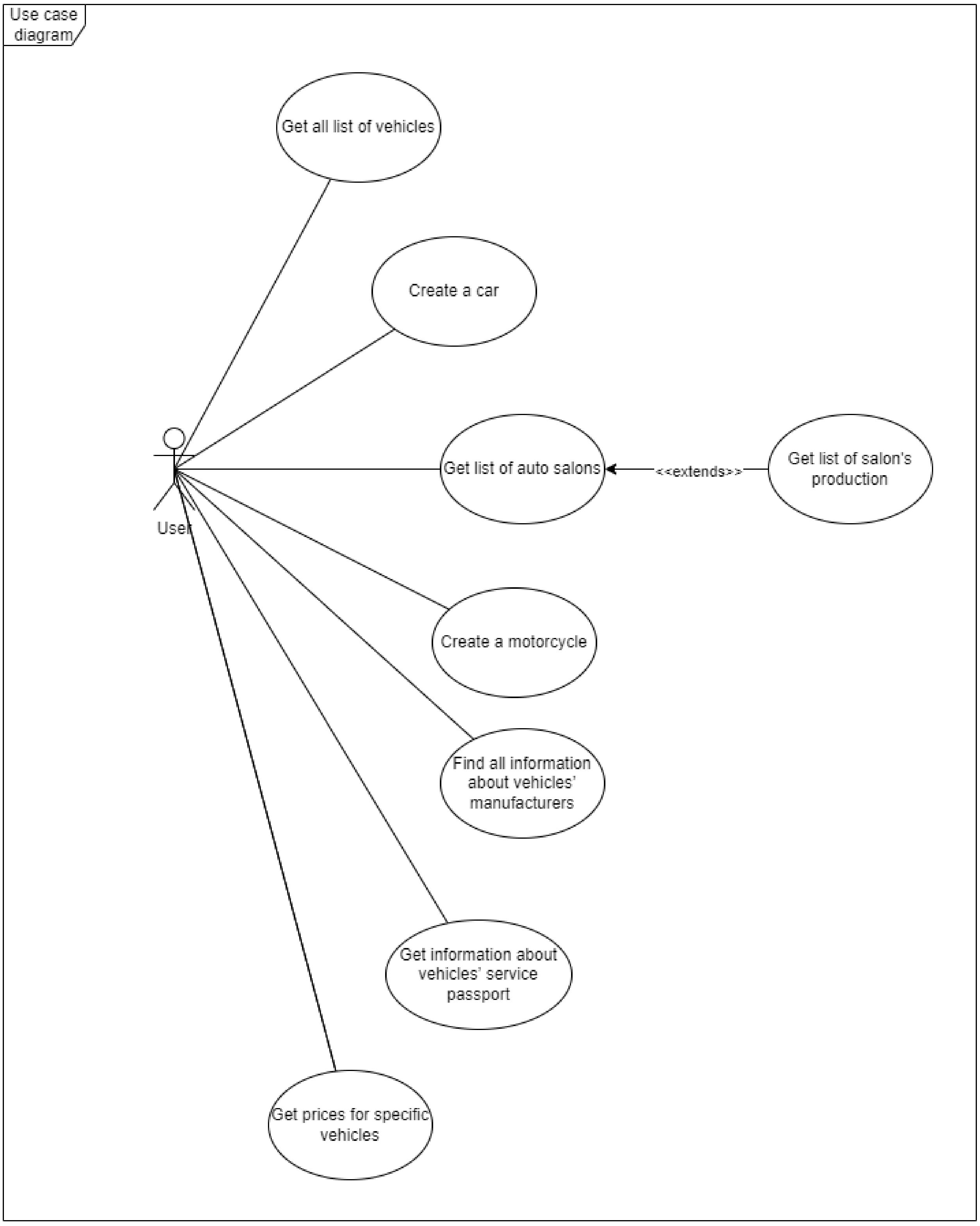
Every address must contain city, street, building number and zip-code.

Vehicle could be a car or a motorcycle. Both types have their unique fields, that makes them an separated data structures. For car it’s type of body (enumeration that gives a variety of body types), for motorcycle it’s swingarm value.

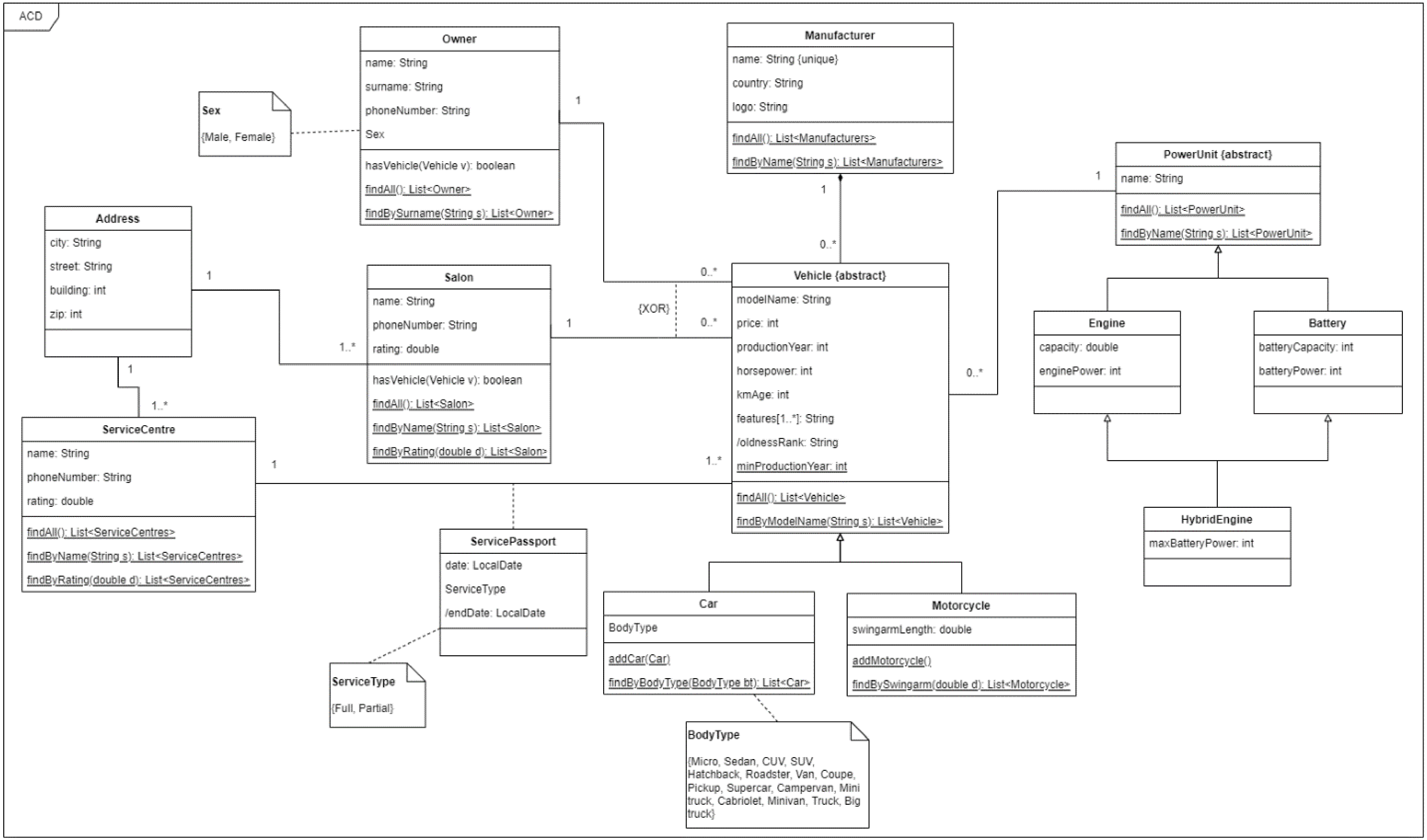
Power unit implies a variety of structures for vehicle “heart” mission. It could be engine, battery, or hybrid of two previous types. Every variant has its’ own fields, which underline their uniqueness.

1. The user must be able to get a list of salons.
2. The user must be able to get all the information about the selected salon and its’ vehicles.
3. The user must be able to get all list of vehicles.
4. User must be able to create a car.
5. User must be able to create a motorcycle.
6. User must be able to find all information about vehicles’ manufacturers.
7. User must be able to get all information about vehicles’ service passport.
8. User must be able to get prices for specific vehicles.

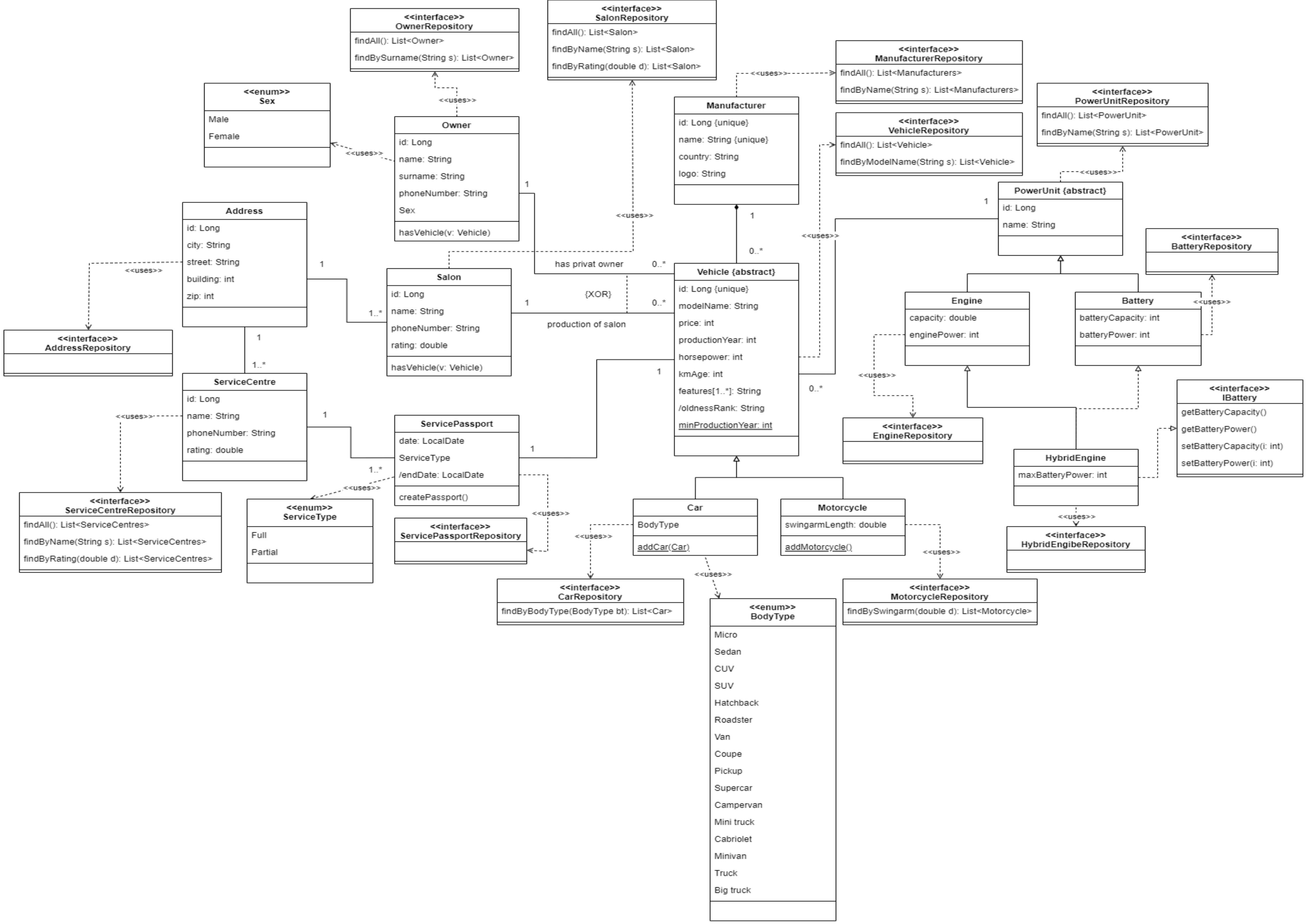
The use case diagram


Use case diagram
Get all list of vehicles
Create a car
Get list of auto salons
<<extends>>
Get list of salon's production
User
Create a motorcycle
Find all information about vehicles' manufacturers
Get information about vehicles' service passport
Get prices for specific vehicles

The class diagram – analytical



The class diagram – design



<<interface>> OwnerRepository
findAll(): List<Owner>
findBySurname(String s): List<Owner>
<<interface>> SalonRepository
findAll(): List<Salon>
findByName(String s): List<Salon>
findByRating(double d): List<Salon>
<<interface>> ManufacturerRepository
findAll(): List<Manufacturers>
findByName(String s): List<Manufacturers>
<<enum>> Sex
Male
Female
Owner
id: Long
name: String
surname: String
phoneNumber: String
Sex
hasVehicle(v: Vehicle)
Manufacturer
id: Long {unique}
name: String {unique}
country: String
logo: String
<<interface>> PowerUnitRepository
findAll(): List<PowerUnit>
findByName(String s): List<PowerUnit>
<<interface>> VehicleRepository
findAll(): List<Vehicle>
findByModelName(String s): List<Vehicle>
PowerUnit {abstract}
id: Long
name: String
Address
id: Long
city: String
street: String
building: int
zip: int
Salon
id: Long
name: String
phoneNumber: String
rating: double
hasVehicle(v: Vehicle)
has privat owner
{XOR}
production of salon
Vehicle {abstract}
id: Long {unique}
modelName: String
price: int
productionYear: int
horsepower: int
kmAge: int
features[1..*]: String
/oldnessRank: String
minProductionYear: int
<<interface>> BatteryRepository
Engine
capacity: double
enginePower: int
Battery
batteryCapacity: int
batteryPower: int
<<interface>> AddressRepository
ServiceCentre
id: Long
name: String
phoneNumber: String
rating: double
ServicePassport
date: LocalDate
ServiceType
/endDate: LocalDate
createPassport()
<<interface>> EngineRepository
<<interface>> IBattery
getBatteryCapacity()
getBatteryPower()
setBatteryCapacity(i: int)
setBatteryPower(i: int)
HybridEngine
maxBatteryPower: int
<<interface>> ServiceCentreRepository
findAll(): List<ServiceCentres>
findByName(String s): List<ServiceCentres>
findByRating(double d): List<ServiceCentres>
<<enum>> ServiceType
Full
Partial
<<interface>> ServicePassportRepository
Car
BodyType
addCar(Car)
Motorcycle
swingarmLength: double
addMotorcycle()
<<interface>> HybridEngibeRepository
<<interface>> CarRepository
findByBodyType(BodyType bt): List<Car>
<<interface>> MotorcycleRepository
findBySwingarm(double d): List<Motorcycle>
<<enum>> BodyType
Micro
Sedan
CUV
SUV
Hatchback
Roadster
Van
Coupe
Pickup
Supercar
Campervan
Mini truck
Cabriolet
Minivan
Truck
Big truck
<<uses>>
1
0..*
1..*

The scenario

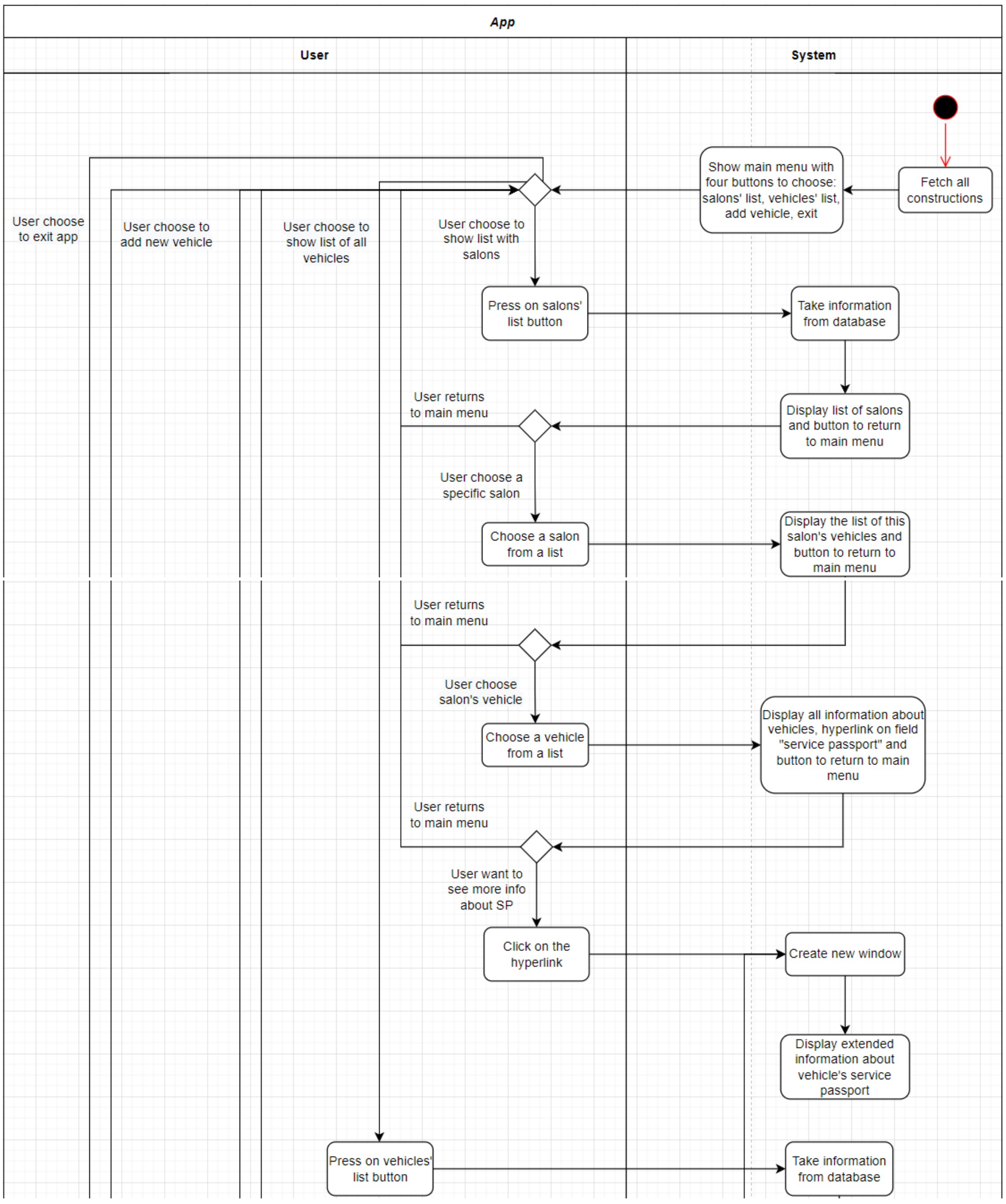
List salons use case scenario:

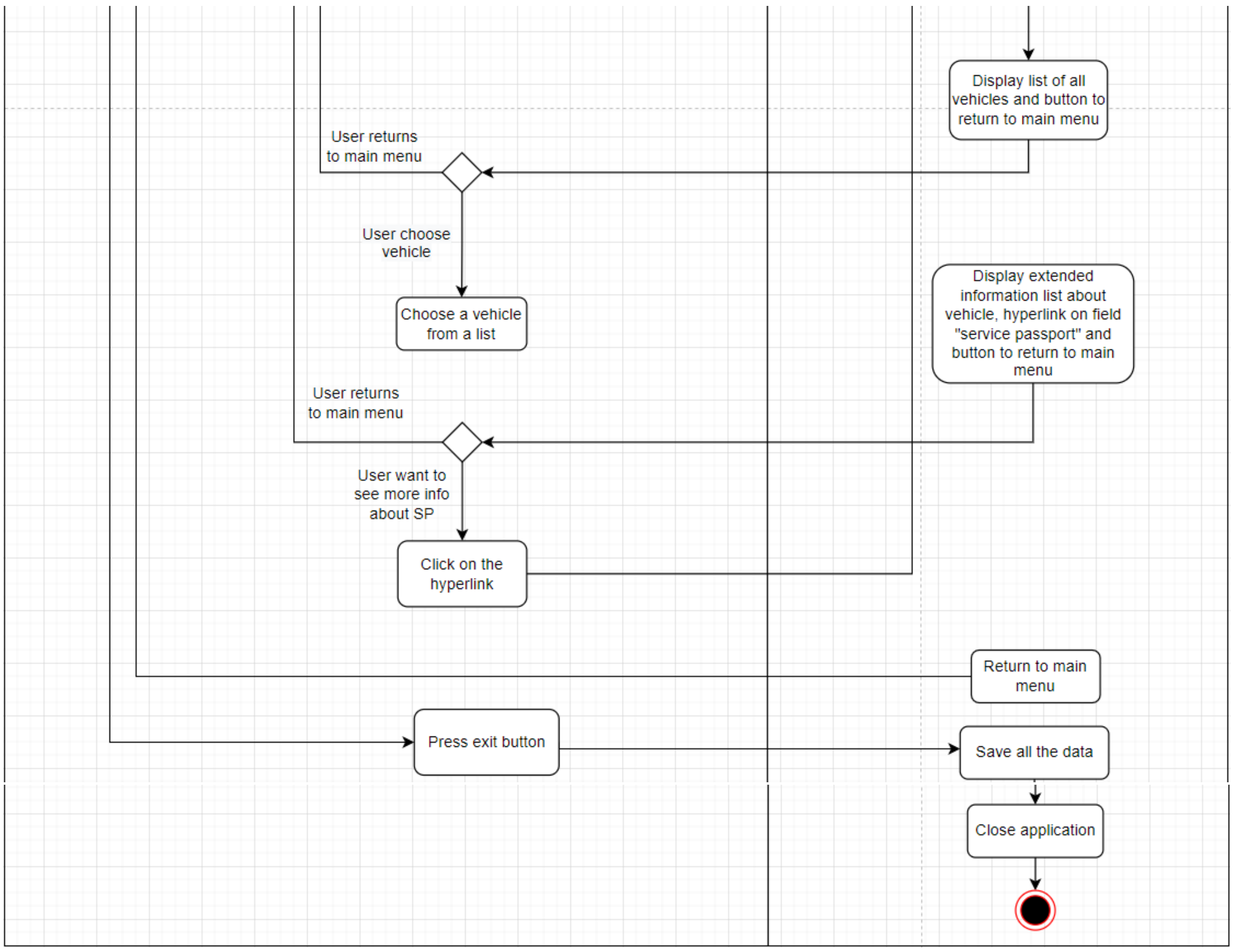
1. User push salons’ list button.
2. System fetch data from database.
3. System shows all the salons.
4. User choose one of salons.
5. System fetch data from database.
6. System shows the list of vehicles of chosen salon.
7. User clicks on specific vehicle.
8. System fetch data from database.
9. System shows all information connected with this car or motorcycle.
10. User clicks on hyperlink at “service passport”.
11. System fetch data from database.
12. System opens in new window extended information about service passport of this vehicle.
13. User returns to main menu.
14. User ends work with application by clicking exit button.

List all vehicles use case scenario:

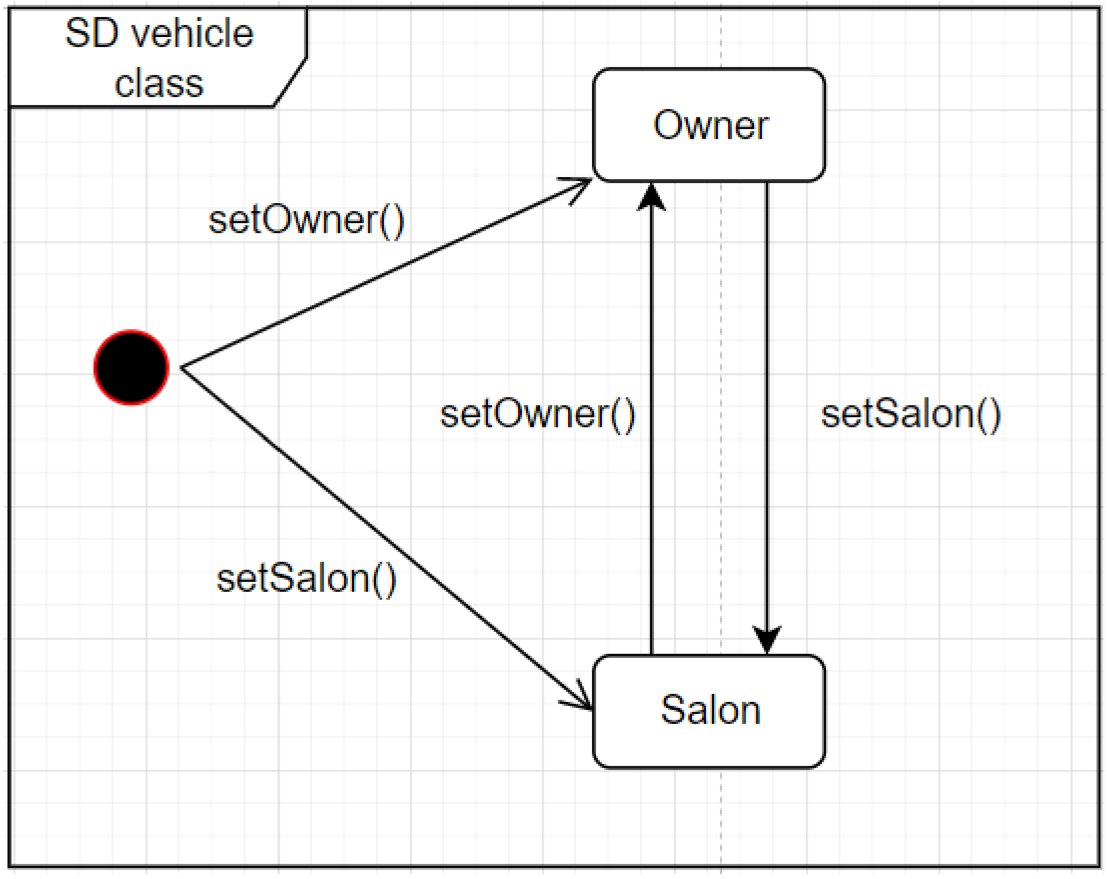
1. User push all vehicles’ list button.
2. System fetch data from database.
3. System shows all the vehicles.
4. User choose one of vehicle.
5. System fetch data from database.
6. System shows extended information about vehicle.
7. User clicks on hyperlink at “service passport”.
8. System fetch data from database.
9. System opens in new window extended information about service passport of this vehicle.
10. User returns to main menu.
11. User ends work with application by clicking exit button.

The activity diagram

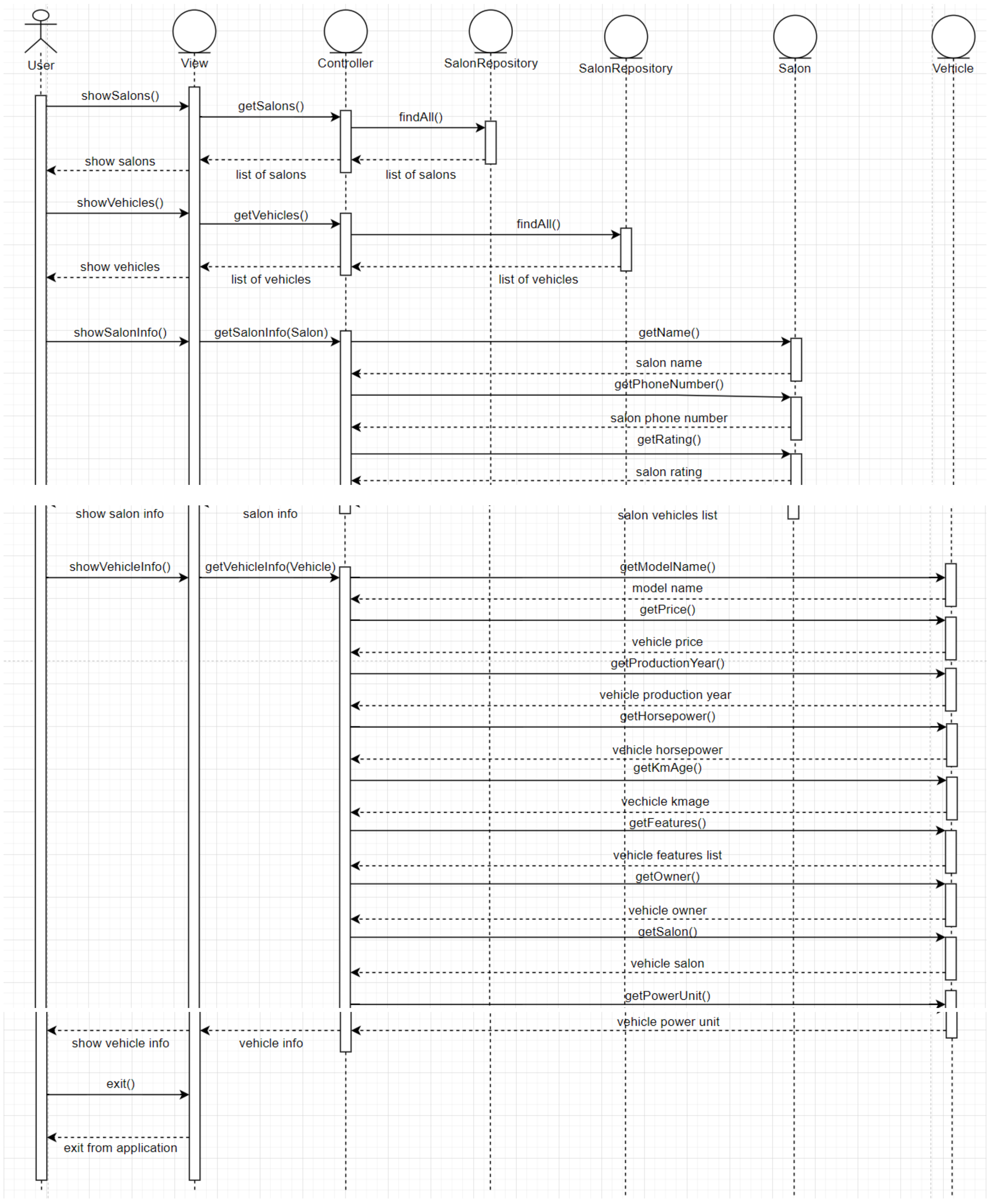

App
User
System
Fetch all constructions
Show main menu with four buttons to choose: salons' list, vehicles' list, add vehicle, exit
User choose to exit app
User choose to add new vehicle
User choose to show list of all vehicles
User choose to show list with salons
Press on salons' list button
Take information from database
Display list of salons and button to return to main menu
User returns to main menu
User choose a specific salon
Choose a salon from a list
Display the list of this salon's vehicles and button to return to main menu
User returns to main menu
User choose salon's vehicle
Choose a vehicle from a list
Display all information about vehicles, hyperlink on field "service passport" and button to return to main menu
User returns to main menu
User want to see more info about SP
Click on the hyperlink
Create new window
Display extended information about vehicle's service passport
Press on vehicles' list button
Take information from database


Display list of all vehicles and button to return to main menu
User returns to main menu
User choose vehicle
Choose a vehicle from a list
Display extended information list about vehicle, hyperlink on field "service passport" and button to return to main menu
User returns to main menu
User want to see more info about SP
Click on the hyperlink
Return to main menu
Press exit button
Save all the data
Close application

The state diagram



The interaction diagram

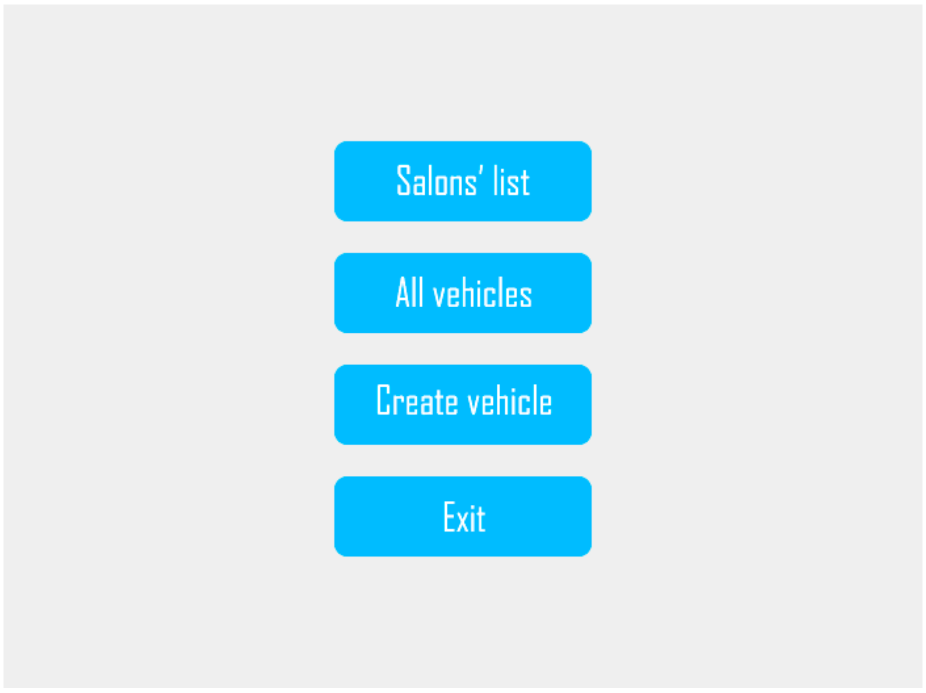


The GUI design

Here I will describe an approximate interface design

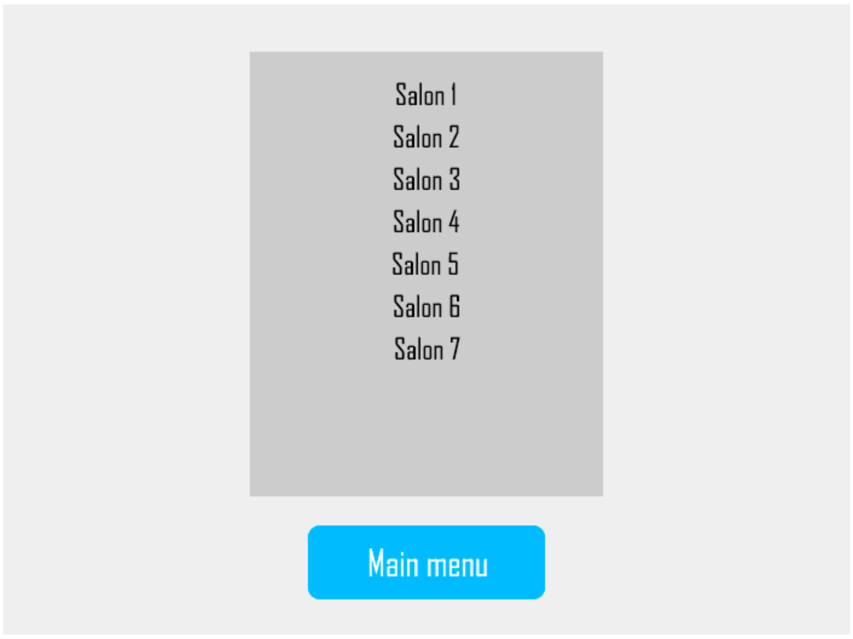
1. Start form

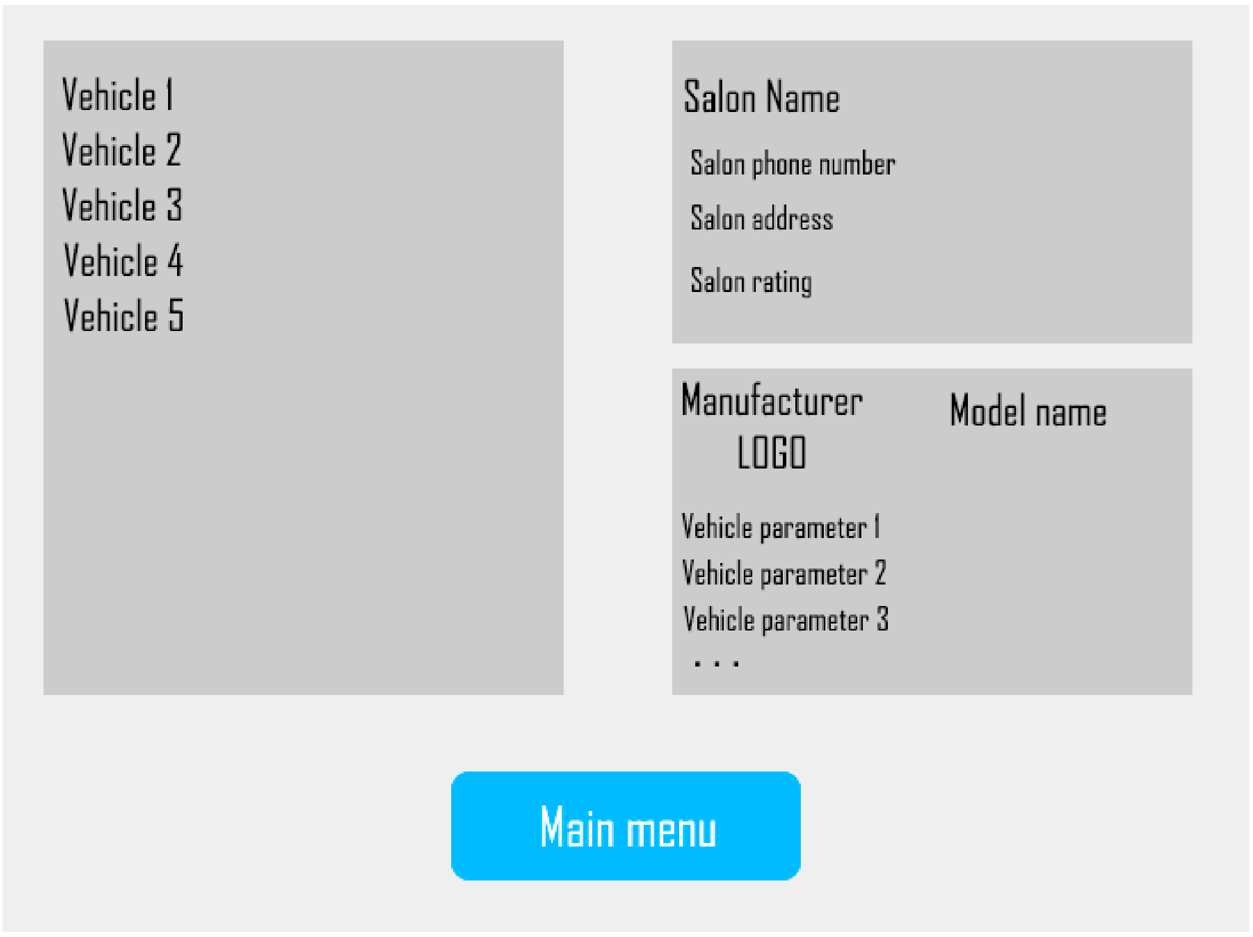
On the home page we can get four buttons.



2. Salons' list

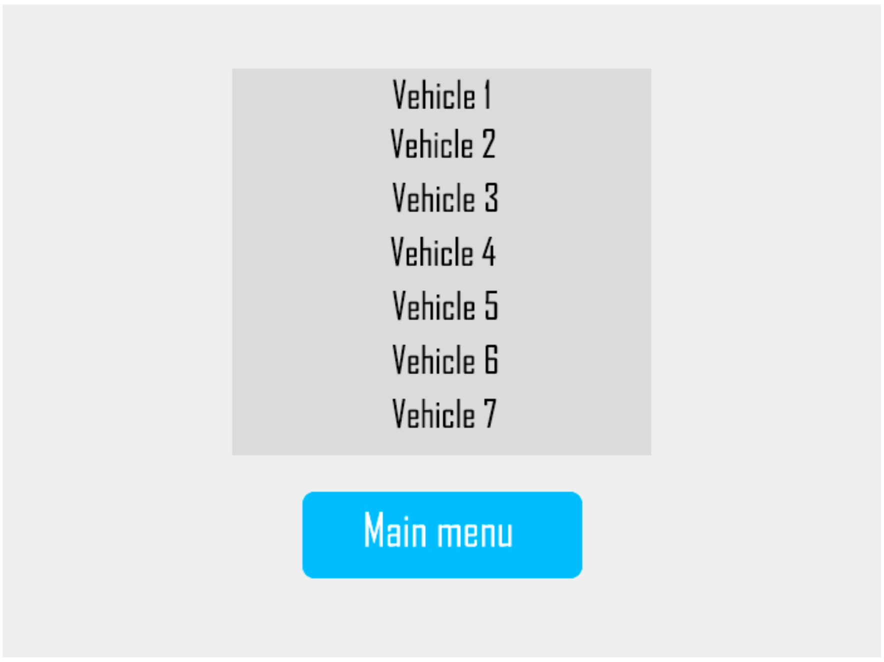
In this form we can choose the salon that interests us. By choosing a salon we can see all the vehicles in this salon and all information regarding that salon. After selecting a vehicle there will be all the information about the vehicle's parameters. We always have main menu button to return on main page.

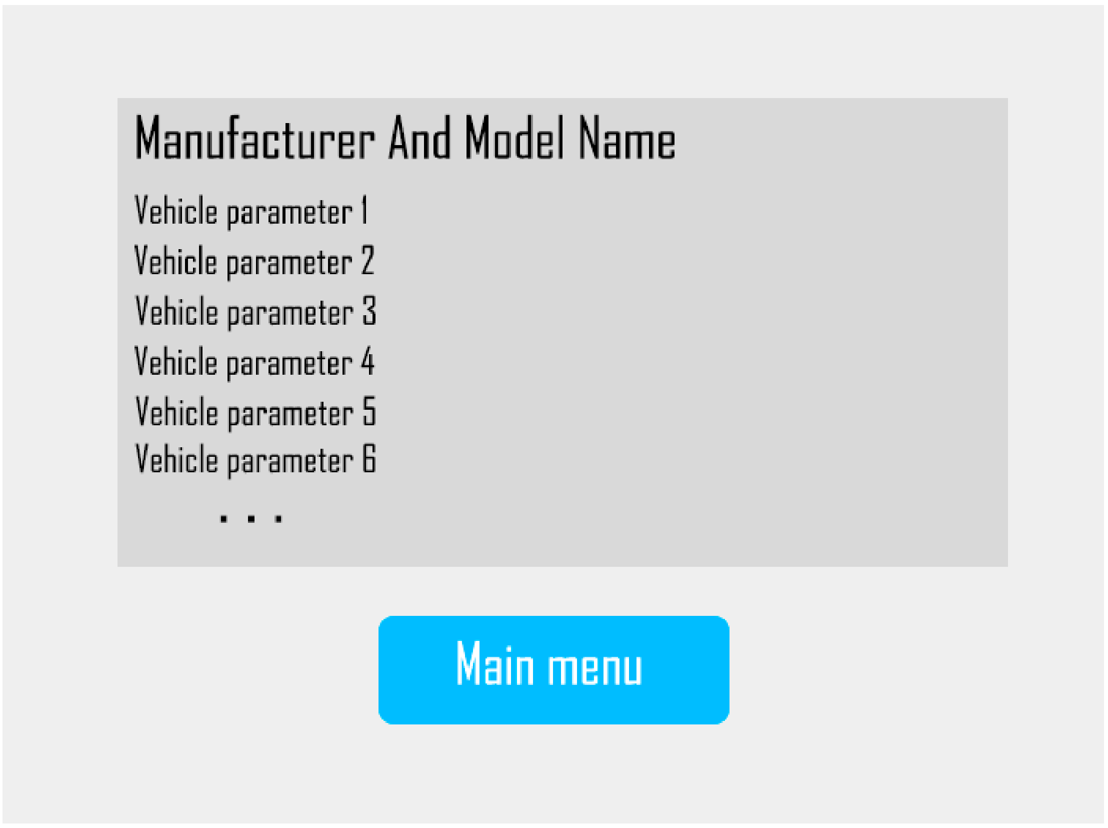




3. Vehicles' list

In this form we can choose all vehicle that interests us. By choosing a vehicle we can see all the information regarding that vehicle. We always have main menu button to return on main page.





The discussion of design decisions and the effect of dynamic analysis

The application will be created using Java, Lombok, Hibernate.

Vehicle, Service Centre, Manufacturer, Salon classes have a repository that use a class extent in methods. These methods can be used to find certain information.

Vehicle class:

Types of multi-value attributes were defined. For both multi-value attributes in this class will be used Set of strings (Set<String>), because order does not interest me, and I need only a collection to store objects.

Association between Vehicle and Service Centre implemented by creating a middle-class Service Passport with one-to-one association with Vehicle and one to many with Service Centre.

I also have to implement XOR constraint for Vehicle class, that will give opportunity to choose the type of an owner of certain vehicle. The type could be private owner (implemented as class Owner) or salon (implemented as class Salon).

Salon/Owner classes:

In Salon/Owner classes besides standard attributes was added a boolean method hasVehicle(Vehicle v). This method will be used to detect, if certain private owner or salon has the car that was chosen like a parameter.

Manufacturer class:

In this class I use composition with class Vehicle, in order to make it logically right. The vehicle (no matter car or motorcycle) cannot exist without being labeled as production of any manufacturer.

Classes connected with power units:

PowerUnit is abstract and based class for Engine and Battery classes. These two classes are extended by another class HybridEngine. In Java there is no possibility to make such inheritance directly. It is a multi-inheritance, and I implemented this inheritance by adding interface IBattery and implementation class Battery. In IBattery interface was added getters and setters which will be used for HybridEngine class. Battery class extends the super class and implements IBattery interface. Engine class extends only a super class. Class HybridEngine extends Engine class and implements IBattery interface. Thus, the HybridEngine class has the properties of both the city and the castle.