# JavaScript OOP Exam – Vehicle Park

A manager of a vehicle park in a big organization is set to build an integrated system for the **employees**, so they know what **vehicle** they can use. A **vehicle** holds **brand**, **age**, **terrain coverage** and **number of wheels**. **Bikes** are vehicles with **frame size** and **number of shifts** and always have **2 wheels**. **Automobiles** are vehicles with **consumption** and **type of fuel**. **Trucks** are automobiles with **number of doors**, default **terrain coverage** **“all”** and always have **4 wheels**. **Limos** are automobiles with **terrain coverage “road”** and a **set of employees** that can use it.

### Engine

You are given an engine which consists of two main parts: **ParkManager** which holds the core logic of the application and **Models** which holds the classes. The **ParkManager** contains a module for **processing commands** from the console. It currently supports the following commands:

* **insert** – adds a travel/destination to the engine
* **delete** – deletes a travel/destination from the engine. **Note**: Deleted destinations are removed from all travels they are in.
* **append-employee** – appends an employee to a specified Limo, so they can use it
* **detach-employee** – detaches an employee from the specified Limo
* **list** – prints all travels in their order they were added to the engine. Travel **destinations** are also printed in the order they were added to the travel. (Refer to the output to see printing format.)

**Note:** The engine can insert and delete multiple instances of the same vehicle. However an employee can be appended only once to a Limo instance and second append will produce nothing.

You are **not allowed to change the Engine structure** so you may only write in the **ParkManager** and **Models** modules.

### Implement the Missing Classes

**Implement all missing classes** to model the vehicle park using the **best practices** of JavaScript object-oriented programming (OOP). **Avoid duplicated code** through inheritance. Encapsulate correctly all fields with **getters** and **setters** with **data validation**. Create the following function constructors (i.e. "classes"):

* **Employee** – already fully implemented, holds **name**, **position** and **grade**
* **Vehicle (abstract)** – holds **brand**, **age**, **terrain coverage** and **number of wheels**
* **Bike** – holds a **frame size** and **number of shifts** and always has **2 wheels**
* **Automobile (abstract)** – holds **consumption** and **type of fuel**
* **Truck** – holds **number of doors**, has default **terrain coverage** **“all”** and always has **4 wheels**
* **Limo** – always has **terrain coverage “road”** and holds a **set of employees**, has **appendEmployee()** and **detachEmployee()** methods– throws an error if it contains no such employee.

### Implement Additional Commands

Implement the following commands:

* **list-employees** – filters all employees above or equal to a certain **grade** (unless specified "all"). Returns the employees **sorted** by **name** (always unique).

### Validate the Data

The "class" fields should follow the given validity rules:

* Vehicle **brand** should be a non-empty string
* Vehicle **age** should always be a non-negative **number**
* Vehicle **terrain coverage** can be either “**road**” or “**all**”
* Vehicle **number of wheels** should always be a non-negative **number**
* Bike **frame size** should always be a non-negative **number**
* Bike **number of shifts** is **optional** andshould be a non-empty stringwhen it exists
* Automobile **consumption** should be a non-negative **number**
* Automobile **type of fuel** should be a non-empty string
* Truck **number of doors** should be a non-negative **number**

All thrown errors are handled by the engine and the message "**Invalid command.**" is displayed.

#### Sample Test #1

|  |
| --- |
| **Input** |
| insert(type=employee;name=Angel;position=Trainer;grade=10)  insert(type=bike;brand=RAM;age=2;terrain-coverage=all;frame-size=19;number-of-shifts=21)  insert(type=truck;brand=Opel;age=15;terrain-coverage=road;consumption=8;type-of-fuel=gas;number-of-doors=3)  insert(type=truck;brand=Ford;age=8;consumption=10;type-of-fuel=diesel;number-of-doors=5)  insert(type=limo;brand=GM;age=5;number-of-wheels=8;consumption=15;type-of-fuel=diesel)  list  delete(type=truck;brand=Opel)  delete(type=bike;brand=RAM)  append-employee(name=Angel;brand=GM)  list  delete(type=truck;brand=Ford)  insert(type=limo;brand=GM;age=5;number-of-wheels=8;consumption=15;type-of-fuel=diesel)  insert(type=employee;name=Vlado;position=Programmer;grade=20)  append-employee(name=Vlado;brand=GM)  list  list-employees(grade=all) |

|  |
| --- |
| **Output** |
| Employee created.  Bike created.  Truck created.  Truck created.  Limo created.  List Output:  -> Bike: brand=RAM,age=2.0,terrainCoverage=all,numberOfWheels=2,frameSize=19,numberOfShifts=21  -> Truck: brand=Opel,age=15.0,terrainCoverage=road,numberOfWheels=4,consumption=[8l/100km gas],numberOfDoors=3  -> Truck: brand=Ford,age=8.0,terrainCoverage=all,numberOfWheels=4,consumption=[10l/100km diesel],numberOfDoors=5  -> Limo: brand=GM,age=5.0,terrainCoverage=road,numberOfWheels=8,consumption=[15l/100km diesel]  --> Employees, allowed to drive: ---  Truck deleted.  Bike deleted.  Added employee to possible Limos.  List Output:  -> Truck: brand=Ford,age=8.0,terrainCoverage=all,numberOfWheels=4,consumption=[10l/100km diesel],numberOfDoors=5  -> Limo: brand=GM,age=5.0,terrainCoverage=road,numberOfWheels=8,consumption=[15l/100km diesel]  --> Employees, allowed to drive:  ---> Angel,position=Trainer  Truck deleted.  Limo created.  Employee created.  Added employee to possible Limos.  List Output:  -> Limo: brand=GM,age=5.0,terrainCoverage=road,numberOfWheels=8,consumption=[15l/100km diesel]  --> Employees, allowed to drive:  ---> Angel,position=Trainer  ---> Vlado,position=Programmer  -> Limo: brand=GM,age=5.0,terrainCoverage=road,numberOfWheels=8,consumption=[15l/100km diesel]  --> Employees, allowed to drive:  ---> Vlado,position=Programmer  List Output:  ---> Angel,position=Trainer  ---> Vlado,position=Programmer |