C# OOP Retake Exam

E-Drive Rent

Overview

You are chosen to take part in a Start-up company, which is developing an electric vehicles rent-a-car application. Your task is to create the classes needed for the application and implement the logic, standing behind some important buttons. The application must have support for **`User`**, **`Vehicle`** and **`Route`**. The project will consist of **model classes** and a **controller class**, which manages the **interaction** between the **users, vehicles** and **routes**.

Setup

- Upload **only the `EDriveRent`** project in every task **except Unit Tests.**
- **Do not modify the interfaces or their packages.**
- Use **strong cohesion** and **loose coupling.**
- **Use inheritance and the provided interfaces wherever possible.**
  - This includes **constructors**, **method parameters,** and **return types.**
- **Do not** violate your **interface implementations** by adding **more public methods** in the concrete class than the interface has defined.
- Make sure you have **no public fields** anywhere.
- **Exception messages** and **output messages** can be found in the **`"Utilities"`** folder.
- For solving this problem use **Visual Studio 2019/ Visual Studio 2022** and **netcoreapp 3.1/netcoreapp6.0**

Task 1: Structure (50 points)

For this task's evaluation logic in the methods isn't included.

You are given **4** interfaces (**`IUser`, `IVehicle`, `IRoute` and `IRepository`**) and you must implement their functionality in the **correct classes**.

There should be 3 types of entities and 3 repositories in the application: **`User`, `Vehicle`, `Route` and `Repository` (`UserRepository`, `VehicleRepository` and `RouteRepository`) for each of them**:

User

Data

- **`FirstName`** – **`string`**
  - If the **`FirstName` is null or whitespace,** throw an **`ArgumentException`** with the message **`"FirstName cannot be null or whitespace!"`**
- **`LastName`** - **`string`**
  - If the **`LastName` is null or whitespace,** throw an **`ArgumentException`** with the message **`"LastName cannot be null or whitespace!"`**
- **`DrivingLicenseNumber – string`**
  - If the **`DrivingLicenseNumber` is null or whitespace,** throw an **`ArgumentException`** with the message **`"Driving license number is required!"`**
- **`Rating – double`**
  - Set **`Rating's`** initial value to **zero**. The value of the **`Rating`** will be changed every time a **`User`** drives a **`Vehicle`**. Remember to keep the setter private.

- **IsBlocked – bool**
  - Set **IsBloked's** initial value to **false**.

Behavior

void IncreaseRating()

Every time a **User** rents a **Vehicle** and completes the trip without any accidents, his **Rating** will **increase by 0.5**:

- If the **Rating's** value **exeeds 10.0**, **set the value to 10.0**.

void DecreaseRating()

Every time a **User** rents a **Vehicle** and completes the trip with an accident, his **Rating** will **decrease by 2.0**:

If the **Rating's** value **drops below 0.0**, **set the Rating's value to 0.0 and IsBlocked's value to true**.

Override ToString() method:

Override the existing method **ToString()** and modify it, so the returned string must be in the following format:

"{FirstName} {LastName} Driving license: **{drivingLicenseNumber}** Rating: **{rating}"**

Constructor

A **User** should take the following values upon initialization:

```
string firstName, string lastName, string drivingLicenseNumber
```

Vehicle

Vehicle is a **base class** for any **type of Vehicle,** and it **should not be able to be instantiated**.

Data

- **Brand – string**
  - If the **Brand is null or whitespace,** throw an **ArgumentException** with the message "Brand cannot be null or whitespace!"
- **Model - string**
  - If the **Model is null or whitespace,** throw an **ArgumentException** with the message "Model cannot be null or whitespace!"
- **MaxMileage - double**
- **LicensePlateNumber – string**
  - If the **LicensePlateNumber is null or whitespace,** throw an **ArgumentException** with the message "License plate number is required!"
- **BatteryLevel – int**
  - Set **BatteryLevel's** initial value to **100**. This would be 100%. The value of the **BatteryLevel** will be changed every time a **User** drives a **Vehicle** or the **Vehicle** is being recharged. Remember to keep the setter private.
- **IsDamaged – bool**
  - Set **IsDamaged's** initial value to **false**.

Behavior

void Drive(double mileage)

The **Drive()** method should reduce the **BatteryLevel** by a certain **percentage**. It should be calculated **what part of the MaxMileage will be passed** (for example: if the given mileage is 90 kilometers and the Vehicle's MaxMileage

is 180 kilometers, then you should reduce BatteryLevel by 50%). Also when driving **CargoVan** you should reduce **additional 5%**, because of the load. **The percentage** should be **rounded to the closest integer number**.

- Note: **The `Vehicle` will always have enough battery to finish the trip.**

`void Recharge()`

This method restores the value of the property **`BatteryLevel`** to **100%**.

`void ChangeStatus()`

This method sets value of the property **`IsDamaged.`**

- If the value is false, set it to true
- Else set it to false.

`Override ToString() method:`

Override the existing method **`ToString()`** and modify it, so the returned string must be in the following format:

`"{Brand} {Model}` `License plate:` **`{LicensePlateNumber}`** `Battery:` **`{BatteryLevel}`**`% Status: OK/damaged`**"**

Constructor

A **`Vehicle`** should take the following values upon initialization:

```
string brand, string model, double maxMileage, string licensePlateNumber
```

Child Classes

There are two concrete types of **`Vehicle`**:

PassengerCar

`PassengerCar` **has a** constant value for **`MaxMileage = 450`**

The constructor of the **PassengerCar** should take the following parameters upon initialization:

```
string brand, string model, string licensePlateNumber
```

CargoVan

CargoVan has a constant value for **`MaxMileage = 180`**

The constructor of the **CargoVan** should take the following parameters upon initialization:

```
string brand, string model, string licensePlateNumber
```

Route

Data

- **`StartPoint`** – **`string`**
  - If the **`StartPoint`** **is null or whitespace,** throw an **`ArgumentException`** with the message `"StartPoint cannot be null or whitespace!"`
- **`EndPoint`** - **`string`**
  - If the **`EndPoint`** **is null or whitespace,** throw an **`ArgumentException`** with the message `"Endpoint cannot be null or whitespace!"`
- **`Lenght – double`**
  - If the **value is less than 1,** throw an **`ArgumentException`** with the message **`"Length cannot be less than 1 kilometer.`**".

- **RouteId – int**
- **IsLocked – bool**
    - Set **IsLocked's** initial value to **false**.

Behavior

void LockRoute()

This method **sets the value of** the property **IsLocked to true.**

Constructor

A **Route** should take the following values upon initialization:

```
string startPoint, string endPoint, double length, int routeId
```

UserRepository

The **UserRepository** is an **IRepository<IUser>. Collection** for the **users** that are created in the application.

Behavior

void AddModel(IUser user)

Adds a new **IUser** to the collection.

bool RemoveById(string identifier)

Removes the first **IUser** from the **collection,** which has the same **DrivingLicenseNumber** as the given **identifier**. **Returns true** if the removal was **successful**, **otherwise** returns **false**.

IUser FindById(string identifier)

Returns the first **IUser** from the **collection,** which has the same **DrivingLicenseNumber** as the given **identifier, or returns null.**

IReadOnlyCollection<IUser> GetAll()

Returns all added models as a readonly collection**.**

VehicleRepository

The **VehicleRepository** is an **IRepository<IVehicle>. Collection** for the **vehicles** that are created in the application.

Behavior

void AddModel(IVehicle vehicle)

Adds a new **IVehicle** to the collection.

bool RemoveById(string identifier)

Removes the first **IVehicle** from the **collection,** which has the same **LicensePlateNumber** as the given **identifier**. **Returns true** if the removal was **successful**, **otherwise** returns **false**.

IVehicle FindById(string identifier)

Returns the first **IVehicle** from the **collection,** which has the same **LicensePlateNumber** as the given **identifier, or returns null.**

IReadOnlyCollection<IVehicle> GetAll()

Returns all added models as a readonly collection.

RouteRepository

The **RouteRepository** is an **IRepository<IRoute>. Collection** for the **routes** that are created in the application.

Behavior

void AddModel(IRoute route)

Adds a new **IRoute** to the collection.

bool RemoveById(string identifier)

Removes the first **IRoute** from the **collection,** which has the same **RouteId** as the given **identifier (int.Parse())**. **Returns true** if the removal was **successful**, **otherwise** returns **false**.

IRoute FindById(string identifier)

Returns the first **Route** from the **collection,** which has the same **RouteId** as the given **identifier (int.Parse()), or returns null.**

IReadOnlyCollection<IRoute> GetAll()

Returns all added models as a readonly collection.

Task 2: Business Logic (150 points)

The Controller Class

The business logic of the program should be concentrated around several **commands**. You that you musts, which you have to implement in the correct classes.

The first interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "`this.controller = new Controller();`" line.

Data

You need to keep track of some things, this is why you need some private fields in your controller class:

- **users – UserRepository**
- **vehicles – VehicleRepository**
- **routes – RouteRepository**

NOTE: For best evaluation, keep the private collections' names as shown.

Commands

There are several commands, which control the business logic of the application. They are stated below.

RegisterUser Command

Parameters

- **firstName – string**
- **lastName – string**

- **drivingLicenseNumber - string**

Functionality

The method should **create and add** a new entity of **IUser** to the **UserRepository.**

- If there is already a user with the same **drivingLicenseNumber**, return the following message: **"{drivingLicenseNumber}** `is already registered in our platform.`**"**
- If the above case is **NOT** reached, create a new **User** and add it to the **UserRepository**. Return the following message: **"{firstName} {lastName}** `is registered successfully with DLN-` **{drivingLicenseNumber}"**

UploadVehicle Command

Parameters

- **vehicleTypeName – string**
- **brand – string**
- **model – int**
- **licensePlateNumber – string**

Functionality

The method should **create and add** a new entity of **IVehicle** to the **VehicleRepository.**

- If the given **vehicleTypeName** is **NOT** presented as a valid **Vehicle's** child class (**PassengerCar or CargoVan**), return the following message: **"{typeName}** `is not accessible in our platform.`**"**
- If there is already a vehicle with the same **licensePlateNumber**, return the following message: **"{licensePlateNumber}** `belongs to another vehicle.`**"**
- If none of the above cases is reached, **create a correct type** of **IVehicle** and **add** it to the **VehicleRepository**. Return the following message: **"{brand} {model}** `is uploaded successfully with LPN-` **{licensePlaneNumber}"**

AllowRoute Command

Parameters

- **startPoint – string**
- **endPoint – string**
- **length - double**

Functionality

The method should **create and add** a new entity of **IRoute** to the **RouteRepository.**

HINT: Route's constructor will be expecting as the last parameter **routeId.** So it should be created by taking the count of already added routes in the **RouteRepository + 1**.

- If there is already added **Route** with the given **startPoint**, **endPoint** and **length**, return the following message: **"{startPoint}/{endPoint} – {length}** `km is already added in our platform.`**"**
- If there is already added **Route** with the given **startPoint**, **endPoint** and **Route.Length** is less than the given **length** return the following message: **"{startPoint}/{endPoint}** `shorter route is already added in our platform.`**"**
- If the above case is not reached, create a new **Route** and add it to the **RouteRepository**.

- If there is already added **Route** with the given **startPoint**, **endPoint** and greater **Length**, **lock the longer Route.**
- Return the following message: **"{startPoint}/{endPoint} – {length} km is unlocked in our platform."**

MakeTrip Command

Parameters

- **drivingLicenseNumber – string**
- **licensePlateNumber – string**
- **routeId – string**
- **isAccidentHappened - bool**

Constraints

- There will always be a user with the corresponding **drivingLicenseNumber,** already added to the **UserRepository**.
- There will always be a vehicle with the corresponding **licensePlateNumber,** already added to the **VehicleRepository**.
- There will always be a route with the corresponding **routeId**, already added to the **RouteRepository.**
- **The Vehicle will always have enough battery to finish the trip.**

Functionality

A user with the given **drivingLicenseNumber** will take a trip on the route with the given **routeId**, with the vehicle with the given **licensePlateNumber**:

- If the **User** with the given **drivingLicenseNumber** is blocked (**User.IsBlocked == true**) in the application, abort the trip and return the following message: **"User {drivingLicenseNumber} is blocked in the platform! Trip is not allowed."**
- If the **Vehicle** with the given **licensePlateNumber** is damaged (**Vehicle.IsDamaged == true**) in the application, abort the trip and return the following message: **"Vehicle {licensePlateNumber} is damaged! Trip is not allowed."**
- If the **Route** with the given **routeId** is locked (**Route.IsLocked == true**) in the application, abort the trip and return the following message: **"Route {routeId} is locked! Trip is not allowed."**
- Drive the specific vehicle on the specific route (Use the **Vehicle.Drive(route.Length)** method). The trip should take effect to the **BatteryLevel** of the vehicle.
- If the value of the parameter **isAccidentHappened** is **true**, the **IsDamaged** status of the vehicle should be changed to **true**. The **Rating** of the **User** who has rented the **Vehicle** should be **decreased**.
- Else **increase** the **User's Rating**
- Return actual information about the vehicle, **after making the trip**, in the following format: **"{Brand} {Model} License plate: {LicensePlateNumber} Battery: {BatteryLevel}% Status: OK/damaged"**

RepairVehicles Command

Parameters

- **count – int**

Functionality

The method should select **only those vehicles** from the **VehicleRepository**, which are **damaged**. **Order** the selected vehicles **alphabetically by their Brand**, then **alphabetically by their Model.** Take the first **{count}** vehicles, if there are as many damaged vehicles, else take all of the damaged vehicles.

- Each of the chosen vehicles will be repaired (**IsDamaged == false**) and recharged (**battery level** restored to **100%**).
- Return the following message: **"{countOfRepairedVehicles}** `vehicles are successfully repaired!`**"**

UsersReport Command

Functionality

Returns information about each user from the **UserRepository**. Arrange the users by **Rating**, **descending**, then by **LastName alphabetically, then by FirstName alphabetically**. In order to receive the correct output, use the **ToString()** method **of each user:**

"`*** E-Drive-Rent ***`

{user$_1$}

{user$_2$}

...

{user$_n$}"

Note: Do not use `"\r\n"` **for a new line.**

End Command

Ends the program.

Input / Output

You are provided with one interface, which will help with the correct execution process of your program. The interface is **Engine** and the class implementing this interface should read the input and when the program finishes, this class should print the output.

Input

Below, you can see the **format** in which **each command** will be given in the input:

- **RegisterUser {firstName} {lastName} {drivingLicenseNumber}**
- **UploadVehicle {vehicleType} {brand} {model} {licensePlateNumber}**
- **AllowRoute {startPoint} {endPoint} {length}**
- **MakeTrip {drivingLicenseNumber} {licensePlateNumber} {routeId} {isAccidentHappened}**
- **RepairVehicles {count}**
- **UsersReport**
- **End**

Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

Examples

Input
RegisterUser Tisha Reenie 7246506 RegisterUser Bernard Remy CDYHVSR68661 RegisterUser Mack Cindi 7246506 UploadVehicle PassengerCar Chevrolet Volt CWP8032 UploadVehicle PassengerCar Volkswagen e-Up! COUN199728 UploadVehicle PassengerCar Mercedes-Benz EQS 5UNM315 UploadVehicle CargoVan Ford e-Transit 726QOA UploadVehicle CargoVan BrightDrop Zevo400 SC39690 UploadVehicle EcoTruck Mercedes-Benz eActros SC39690 UploadVehicle PassengerCar Tesla CyberTruck 726QOA AllowRoute SOF PLD 144 AllowRoute BUR VAR 87 AllowRoute BUR VAR 87 AllowRoute SOF PLD 184 AllowRoute BUR VAR 86.999 MakeTrip CDYHVSR68661 5UNM315 3 false MakeTrip 7246506 CWP8032 1 true MakeTrip 7246506 COUN199728 1 false MakeTrip CDYHVSR68661 CWP8032 3 false MakeTrip CDYHVSR68661 5UNM315 2 false RepairVehicles 2 UsersReport
Output
Tisha Reenie is registered successfully with DLN-7246506 Bernard Remy is registered successfully with DLN-CDYHVSR68661 7246506 is already registered in our platform. Chevrolet Volt is uploaded successfully with LPN-CWP8032 Volkswagen e-Up! is uploaded successfully with LPN-COUN199728 Mercedes-Benz EQS is uploaded successfully with LPN-5UNM315 Ford e-Transit is uploaded successfully with LPN-726QOA BrightDrop Zevo400 is uploaded successfully with LPN-SC39690 EcoTruck is not accessible in our platform. 726QOA belongs to another vehicle. SOF/PLD - 144 km is unlocked in our platform. BUR/VAR - 87 km is unlocked in our platform.

```
BUR/VAR - 87 km is already added in our platform.
SOF/PLD shorter route is already added in our platform.
BUR/VAR - 86.999 km is unlocked in our platform.
Mercedes-Benz EQS License plate: 5UNM315 Battery: 81% Status: OK
Chevrolet Volt License plate: CWP8032 Battery: 68% Status: damaged
User 7246506 is blocked in the platform! Trip is not allowed.
Vehicle CWP8032 is damaged! Trip is not allowed.
Route 2 is locked! Trip is not allowed.
1 vehicles are successfully repaired!
*** E-Drive-Rent ***
Bernard Remy Driving license: CDYHVSR68661 Rating: 0.5
Tisha Reenie Driving license: 7246506 Rating: 0
```

Task 3: Unit Tests (100 points)

You will receive a skeleton with three classes inside – **`Garage`** and **`Vehicle`**. **`Garage`** class will have some methods, fields, and constructors. Cover the whole class with the unit test to make sure that the class is working as intended. If some of the methods in **`Garage`** changes anything from the other classes, you should cover that functionality also. In Judge, you upload **.zip (**with **`VehicleGarage.Tests`** inside**)** from the **skeleton.**