EXAM **MAIN SESSION**	esprit Se former autrement HONORIS UNITED UNIVERSITIES

Semester : 2

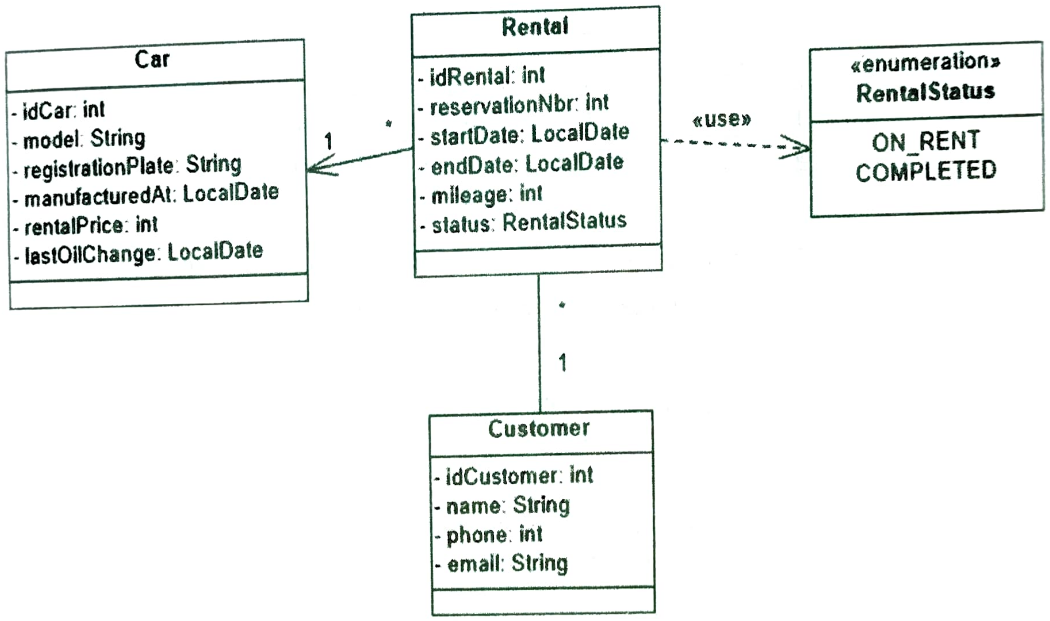
Module : Information Systems Architecture II | **Teachers :** Spring Team

Class(es) : 4DS/4INFINI | **Authorized documents :** YES | **Internet allowed :** NO

Date : 18/05/2023 | **Time :** 09:00 AM | **Duration :** 1H30 | **Number of pages :** 3

The evaluation of the exam is based on an executable source code.
Non-functional source code will not be taken into account during the validation.

We suggest to implement a simplified rental management application intended to be used in a car rental park. The application will have **the following class diagram:**



Part I (5 points) :

Implement the entities that allow generating the database schema as illustrated in the class diagram, considering that:

- The identifiers of all entities are auto-generated using the "**IDENTITY**" strategy.
- Enumerations should be stored as strings in the database (enumeration type: "String").

Part II (15 points) :

Develop the necessary services in Spring beans **@Service** and expose them as web services in beans **@RestController.**
- You can test the methods using **Swagger** or **Postman**.

1) Add 3 cars (Car) having the following details, respecting the following signature **(1pt)** :

public Car addCar(Car car)

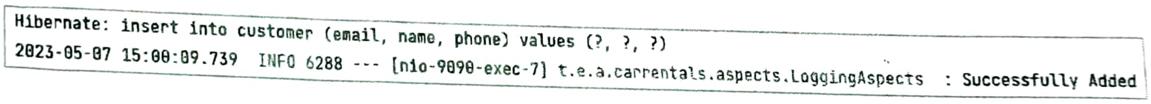
Model	Registration Plate	Manufactured At	Rental Price	Last Oil Change
Kia Rio	204TU5480	2021-03-09	100	2023-01-05
Renault Clio	182TU5026	2019-06-17	80	2021-08-06
Toyota Yaris	198TU5481	2020-03-09	90	2022-12-03

2) Using the following signature, add the 3 Customers below **(1pt) :**

public Customer addCustomer (Customer customer)

Name	Phone	Email
Asma Ben Ahmed	53698157	Asma.benAhmed@gmail.com
Rayen Ahmadi	92158456	rayen.ahmadi@gmail.com
Mahmoud Abbes	25468796	mahmoud.abbes@gmail.com

3) Create an **Aspect** that displays the message «Successfully Added» after the successful execution of the add functions (starting with 'add') for each method called in the service layer. **(1.5pts)**



```
Hibernate: insert into customer (email, name, phone) values (?, ?, ?)
2023-05-07 15:00:09.739  INFO 6288 --- [nio-9090-exec-7] t.e.a.carrentals.aspects.LoggingAspects   : Successfully Added
```

4) Add 3 Locations (Rental) with the following details and assign a customer by his email and a car by its model to each of them, respecting the following signature **(2.5pts)** :

Note: The rental status is set to 'ON_RENT' by default and should be automatically assigned in the code.

public Rental addRental (Rental rental, String costumerEmail, String carModel)

reservationNbr	startDate	endDate	Customer	Car
1121	2021-10-23	2021-11-09	Asma Ben Ahmed	Kia Rio
1122	2022-01-13	2022-02-13	Mahmoud Abbes	Renault Clio
1123	2022-02-16	2022-02-17	Mahmoud Abbes	Toyota Yaris

5) Update the status of the rental for a car passed as a parameter using its license plate number and mileage, following this signature **(2pts)** :

Note: It is necessary to check that the rental status is ON_RENT before changing it to COMPLETED.

public Rental updateMileageAndRentalStatus(String registrationPlate, int mileage);

Registration Plate	**Mileage**
204TU5480	7668
182TU5026	7612
198TU5481	7513

6) Respecting the following signature, display the list of cars that have been rented (**rental status is COMPLETED**) by a customer provided as a parameter **(2pts)** :

public List<Car> getCarsByCustomerEmail (String customerEmail)

7) Respecting the following signature, display the list of cars that will be available between two dates provided as parameters and that were manufactured after the date provided as a parameter **(2pts)** :

public List<Car> getFreeCarsByAge(LocalDate startDate,
LocalDate endDate,
LocalDate manifacturedAfter)

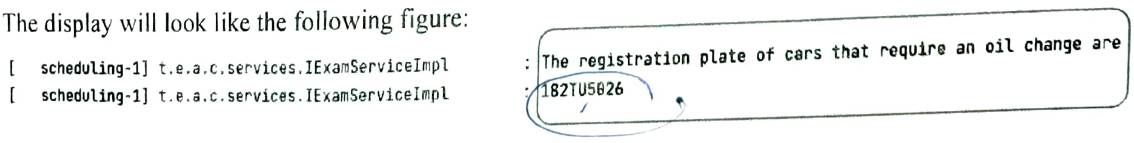
Display example:	
startDate : 2022-02-01 / endDate 2022-02-18 / manifacturedAfter 2019-01-01	**Kia Rio**
startDate : 2022-02-16 / endDate 2022-02-18 / manifacturedAfter 2019-01-01	**Kia Rio + Renault Clio**
startDate : 2022-02-18 / endDate 2022-02-18 / manifacturedAfter 2019-01-01	**Kia Rio + Renault Clio+ Toyota Yaris**

8) Create an automatically scheduled service that displays, every minute, using logs on command line, the license plates of cars that require an oil change. This is determined by checking if the mileage driven since the last oil change exceeds 7600km, respecting the following signature **(3pts)** :

Note: The mileage driven is the sum of the mileages of each rental after the date of the last oil change.

public void getCarsThatNeedOilChange ()

The display will look like the following figure:



```
[   scheduling-1] t.e.a.c.services.IExamServiceImpl        : The registration plate of cars that require an oil change are
[   scheduling-1] t.e.a.c.services.IExamServiceImpl        : 182TU5026
```

Good Luck 😊