|  |
| --- |
| Résultat de recherche d'images pour "apu logo" |

### Student number :

### TP056994

### Nassim Guettat

JAVA Assignement report

Class : CX007-2.5-3-JAV-L

Lecturer: M.Kau Guan Kiat

# Assumptions

We are assuming that our APU Automotive software will be used by the staff and customers of APU Automotive Service Centre (AASC) company. This software (developed in Java language), will allow 3 types of end user to log in: Managers, Technicians and Customers.

* Managers have the main control of different users. In fact, they can register other users, edit their information or even delete them from the file-based system. The manager is also the person who books appointments between customers and technicians and these users have no control on this functionality. Basically, the Manager makes the main junction between Technicians and Customers, a pillar so to speak. Finally, he is also the person who collect payments according to the prices that he has set and generate receipt for the customer when the appointment is done.
* Technicians have a control into their own profile, they can edit their information into the file-based system thanks to the software. The technician decided when the appointment is done, after that he will provide a feedback to the customer and the payment will be done. He can also access to the comments from customer.
* The customer has only control on his own profile. Indeed, as the technician user, he can update his data into the file-based system thanks to the software. Then, he can check his Services and payments histories while providing a comment and a rate to the technician who provided the service. He can also access to feedback according to the right service that he applied for.

# Diagrams

## Use case diagram:

## Une image contenant texte, carte Description générée automatiquement

## Use case specification:

|  |  |
| --- | --- |
| Use Case Name | Log In |
| Description | This use case allows user to login into the system to access the relevant functions according to the user’s role. The various user roles are Manager, Customer and technician. To login to the system, all users have to enter their (unique) username and their password. |
| Primary Actor | User |
| Secondary Actor | None |
| Preconditions | 1. The user has to be registered 2. The user has to put valid information (username and password) |
| Postconditions | 1. The system displays the main page according to the user type |

|  |  |
| --- | --- |
| Use Case Name | Edit profile |
| Description | This use case allows technicians and customers to edit and update their personal data. |
| Primary Actor | Technician, Customer |
| Secondary Actor | None |
| Preconditions | The user has to be logged in |
| Postconditions | The system updates the user data into the file-based system |

|  |  |
| --- | --- |
| Use Case Name | Provide requirement |
| Description | This use case allows technicians and customers to provide software experience feedback and improvement advices |
| Primary Actor | Technician, Customer |
| Secondary Actor | None |
| Preconditions | * The user has to be logged in * The user must not leave a blank message |
| Postconditions | The system updates the data into the file-based system unless the message is blank. |

|  |  |
| --- | --- |
| Use Case Name | Check history of services |
| Description | This allows the customer to access his history of services including payment and feedback from the technician who provided the service. It also allows him to leave a comment and a rate to the technician. |
| Primary Actor | Customer |
| Secondary Actor | none |
| Preconditions | The customer has to be logged in and must have at least one finished appointment |
| Postconditions | The system displays a table with all information and an access to the comment form. |

|  |  |
| --- | --- |
| Use Case Name | Check Comments and rates |
| Description | Allows the technician to access the comments and rates provided by the customer according to the service |
| Primary Actor | Technician |
| Secondary Actor | none |
| Preconditions | The technician has to be logged in and must have at least one comment from customers |
| Postconditions | The system displays a table with all comments and appointments history |

|  |  |
| --- | --- |
| Use Case Name | Check assigned appointment |
| Description | Allows the technician to access his assigned appointment and if he needs to, he can finish it. |
| Primary Actor | Technician |
| Secondary Actor | None |
| Preconditions | The technician has to be logged in and must have at least one assigned appointment |
| Postconditions | The system displays a table with all assigned appointments |

|  |  |
| --- | --- |
| Use Case Name | Register new end user |
| Description | Allows the Manager to register new end user (manager, technician or customer). |
| Primary Actor | Manager |
| Secondary Actor | None |
| Preconditions | The Manager has to be logged in and the username has to be unique (that means that it does not already exist in the file base system). |
| Postconditions | The system saves inputs into the file-based system as user’s data. |

|  |  |
| --- | --- |
| Use Case Name | Book appointment |
| Description | Allows the manager to assign an appointment according to the customer’s needs and the technician availability.  The manager has to:   * input the customer ID * choose between normal and major service. * choose a date * choose a technician among the showing list (on the list, there is only the available ones according to the chosen date). |
| Primary Actor | Manager |
| Secondary Actor | None |
| Preconditions | * The Manager has to be logged in. * The customer’s ID must exist into the file-based system. * It must have available technicians on the chosen date. |
| Postconditions | The system assigns an appointment between the chosen customer and the chosen technician according to the chosen date. It writes those informations into the file-based system as a schedule for the technician (it represents the appointment). |

|  |  |
| --- | --- |
| Use Case Name | Set and edit prices |
| Description | Allows the manager to set or edit the prices of major and normal service. The price has to be a positive float number. |
| Primary Actor | Manager |
| Secondary Actor | None |
| Preconditions | * The Manager has to be logged in. * He has to input the prices as he wants to (positive float number). |
| Postconditions | The system updates the information in the file base system according to the inputs. |

|  |  |
| --- | --- |
| Use Case Name | Manage user’s information |
| Description | Allows the manager to:   * Search specific user * Edit user’s information * Delete users |
| Primary Actor | Manager |
| Secondary Actor | None |
| Preconditions | * The Manager has to be logged in. * If he is searching users, he has to input an existing username. |
| Postconditions | The system updates the information in the file base system according to manager’s choices. |

|  |  |
| --- | --- |
| Use Case Name | Cancel booking |
| Description | Allows the manager to cancel an appointment according to the customer needs. He first has to put the customer’s ID who needs to cancel an appointment and then cancel the appointments according to the customer’s needs. |
| Primary Actor | Manager |
| Secondary Actor | None |
| Preconditions | * The Manager has to be logged in. * The chosen customer has to exist into the filed base system. |
| Postconditions | The system removes the appointment from the file base system according to manager’s choices. |

|  |  |
| --- | --- |
| Use Case Name | Check company balance |
| Description | Allows the manager to check how much the company balance worth. |
| Primary Actor | Manager |
| Secondary Actor | None |
| Preconditions | * The Manager has to be logged in. |
| Postconditions | The system displays the company balance. |

## Class diagram:

Une image contenant fichier

Description générée automatiquement

# Applied oriented object concept:

## Inheritance

In our Java code. we apply a principle called Inheritance. In Java, a class can be derived from one (and only one) other class. That class is called a “parent class”. We call the derived class a “child class”. The concept of inheritance is simple, if we have a child class, it will have the same attribute and the same method as his parent class (and his own set of caracteristics).

Example:

If we have a class called “City”, we can create a class that “extends” city and we will call it “capital”. So, if city has an attribute called “numberOfResident”, the class “Capital” will automatically have as an attribute “numberOfResident” even if we don’t declare it. Nevertheless, this variable “numberOfResident” have to be declared as “protected” and not “private” because the “Capital” class will not be able to access the variable. “protected” allows the parent class to disable a direct access to the variable to its users excepting its children.

How do we apply it in our code?

As a matter of fact, we have different types of users who will use our program. So, we made one parent class and three children class called “Manager”, “Technician” and “Customer”. Their parent class is simply called “User”.

Here are our classes (only the declaration and the constructor):

* User:

Une image contenant capture d’écran

Description générée automatiquement

* Manager:

Une image contenant capture d’écran

Description générée automatiquement

* Technician:

Une image contenant capture d’écran

Description générée automatiquement

* Customer

Une image contenant capture d’écran

Description générée automatiquement

* You can notice that we used the protected keyword to declare the “username” and “userPassword” variables as we mentioned above.
* You can also see that we use the extends keyword after the class declaration. This keyword allows the inheritance of our subclasses.
* Finally, as you can see, we used a specific function called super. This function allows us to call the parent constructor to initiate the “User” class variables which is username and userPassword.

## Abstraction

We also apply a concept called “Abstraction”:

An abstract class is almost identical to a usual class. The main difference being that you can’t instantiate it. Its goal is to hide unnecessary complex detail from the user, allowing only then the user to implement an abstract class without understanding what is happening inside.

Example

Let’s assume that we are developing a program that handle different types of animals. In that program, we will have wolves, dogs, cat…

But the problem is that it would not be optimized if we just declare one class for each animal like that even if they have the same characteristics. But, what do they all have in common?

Indeed, we can declare an “Animal” class that will handle everything that animals have in commons. We will declare this class as an abstract class because we can’t instantiate an Animal object without knowing what type of animal it is.

This is **abstraction**.

How do we apply it in our code?

Let’s take a look at our User Class. We use an “abstract” keyword in our User class. Indeed, our User class is an abstract class because we can’t instantiate a user without knowing its type. Moreover, all users have in common a username and a password so that is why every user is a child of the User class:

Une image contenant capture d’écran

Description générée automatiquement

## Polymorphism

Polymorphism means “many forms”, it occurs when classes are related by inheritance. A common use of polymorphism is when we use a parent class reference to refer to a child class object. Another common use is when we re-write a parent method into the child class.

Example

If we have class animal and a class cat which extend the animal class. Thanks to polymorphism, we can write as follows:

*Animal cat = new Cat();*

Thanks to polymorphism, the object “cat” is an instance of the “Cat” class.

How do we apply it in our code?

In our program we have a function that return an ArrayList<?>. This ArrayList will either be a Manager, Technician or Customer Arraylist.

However, to log in the user, the program just needs to access to the username and the userpassword variables. So, polymorphism allows us to do as follows:

Une image contenant outil

Description générée automatiquement

We also use static polymorphism, meaning that we overload a method into the same class. Here is an example:

Une image contenant capture d’écran

Description générée automatiquement

In fact, both of these methods return the same thing but what happen inside depends on the parameters.

## Encapsulation

Encapsulation in OOP consists of wrapping data into a single unit (a class for example) which allows units to read or manipulate it. This is usually linked with the hiding of these data so it can be protected from users who are not allowed to use or access it.

Example

If we think about a car, the car builder creates the wheels, the engine… But the driver who owns the car has no access to these elements. This is encapsulation, you can drive the car easily, but you have no access to its attributes because it is protected so you only have access to the commands.

How do we apply it in our code?

Indeed, as we explained in the inheritance part, our “User” class attribute is declared as protected. So only our User class and its children can access to the attribute. Usually, we declare all classes attribute as private; so, only the class can directly access to it. Moreover, most of our methods are publics : an external user can have access to the method and manipulate the object.

# Additional features

## Cancel booking:

When we developed the booking part, we were wondering what to do if the manager misunderstands the customer needs or if the customer wants to cancel his appointment.

So, we developed a cancel part which allow the manager to cancel an appointment according to the customer’s needs. Indeed, he just has to input the customer’s ID and a table with all the chosen customer’s appointments will appear and he can cancel the one he needs to, it will remove the appointment from the technician’s schedule and he will be available for the time he had to provide the service.

## Check the company balance:

We were thinking that it would be useful if the Manager can check the current company balance before making payment or whatever. So, we just implemented a simple dialog window that display the company balance when the manager pushes the right button on his interface.

## Requirements/advices system:

To improve our software, there is nothing better than users’ feedback. So we implemented a requirement system:

* Customers and Technicians can provide feedbacks on what they liked or not in the software and improvement advices
* Managers can check the requirements so they can ask developers what to do.

# Reference list:

[https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java](https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/exercises/14)

(French web site)

<https://stackoverflow.com/>

(chat forum where we can ask for advices)

<https://www.lucidchart.com/>

(diagram designing tool)

<https://www.youtube.com/>

(video tutorials)

Conclusion:

In the end, this project helped us gained even more experience in Object-Oriented Programming. As it turns out, the assignment opened my mind regarding the Java language because I didn’t have any knowledge about it before having these classes.

Special thanks to M.KAU GUAN KIAT who helped us throughout the project.