UNIVERSITÉ NATIONALE DU VIETNAM À HANOÏ
INSTITUT FRANCOPHONE INTERNATIONAL

Option : Systèmes Intelligents et Multimédia (SIM)

Promotion : XXI

Génie logiciel avancé
Rapport du Travail pratique #1

Modélisation et programmation orientée-objet avec Java

Un petit gestionnaire de tâches

DIALLO Azise Oumar

Encadrant :

Dr HO Tuong Vinh, Professeur (VNU, IFI)

Année académique 2016-2017

Table des matières

1 Introduction générale

Dans le cadre du cours « module Génie logiciel avancé », nous avons reçu un travail pratique (TP1) de programmation à réaliser. Ce travail dont le but est de nous rappeler les concepts de la modélisation avec UML et la programmation orientée-objet avec Java, consiste à réaliser d'un programme permettant de gérer les taches dans une équipe de travail. Il s'agit principalement de permettre au gestionnaire de gérer les taches (création, modification, suppression), de gérer les membres de l'équipe (création, modification, suppression), d'assigner et de rechercher des taches. Outre ce but principal de rappel, ce TP vise également à nous familiariser avec l'IDE Eclipse et la plate-forme collaborative GitHub.

La suite de ce présent rapport consistera à présenter les exigences (fonctionnelles et non fonctionnelles) et la conception de notre application à l'aide d'UML ainsi que son implémentation avec le langage JAVA. Nous allons conclure après avec une auto-évaluation de notre travail.

2 Analyse et conception de l'application avec UML

Après avoir identifié les besoins du Gestionnaire de l'équipe de travail, nous allons procéder maintenant à la définition des composants logiciels qui réaliseront les fonctionnalités souhaitées par les utilisateurs. Ainsi, ces composants qui seront définis permettront de réaliser le système avec suffisamment de précision. Pour cela, nous utiliserons les diagrammes UML pour avoir une vue détaillée du système. Pour ce faire, nous utiliserons principalement trois diagrammes : le diagramme de cas d'utilisation, le diagramme de séquence et le diagramme de classe.

2.1 Analyse des besoins

Dans cette partie, nous nous intéressons à l'analyse des besoins c'est à dire la description et la spécification du système logiciel à développer. Il existe deux type d'exigences à savoir fonctionnelles et non fonctionnelles.

Les exigences fonctionnelles. Nous allons utiliser le diagramme de cas d'utilisation (Use Case) pour illustrer les exigences fonctionnelles de notre application. Ce diagramme est un moyen d'exprimer les besoins des utilisateurs d'un système informatique. Dans notre système, nous avons un seul. Le tableau (Table 1) résume les exigences fonctionnelles de notre application.

Ainsi, de l'analyse du tableau (Table 1),nous avons le diagramme de cas d'utilisation de notre système dans la figure (Figure 1).

Les exigences non fonctionnelles. Ce sont exigences liées notamment à la performance, sûreté, confidentialité, etc. De plus, il faut chercher des critères mesurables de l'application. Le tableau (Table 2) regroupe les exigences non fonctionnelles de notre application.

TABLE *1 – Les exigences fonctionnelles de l'application*

Exigences	Description
Créer ou ajouter un membre	Le système doit demander à l'utilisateur de renseigner le nom du membre. Ensuite, le système va enregistrer le nouveau membre en lui assignant automatiquement un Id.
Supprimer un membre	Le système doit permettre de supprimer un membre. Pour éviter tout erreur de manipulation, le système doit pouvoir demander la confirmation de la suppression.
Modifier les informations d'un membre	Le système doit permettre et demander de renseigner les nouvelles informations du membre. Le système doit pouvoir enregistrer les nouvelles informations renseignées
Créer une tache	Le système doit demander à l'utilisateur de renseigner le nom et une brève description de la tache. Ensuite, le système va enregistrer la tache en lui assignant automatiquement un Id et en mettant son statut par défaut à « nouveau ».
Assigner une tache à un membre	Le système doit demander à l'utilisateur de sélectionner la tache en question. Ensuite, demander le membre à qui la tache doit être attribuée.
Modifier les informations d'une tache	Le système doit permettre et demander de renseigner les nouvelles informations de la tache. Le système doit pouvoir enregistrer les nouvelles informations renseignées.
Ajouter une tache	Le système doit permettre d'ajouter une tache à un membre. En effet, un membre peut avoir une ou plusieurs taches. Mais, une tache est assignée uniquement à un membre.
Supprimer une tache	Le système doit permettre de supprimer une tache. Pour éviter tout erreur de manipulation, le système doit pouvoir demander la confirmation de la suppression.
Afficher les informations d'une tache	Le système doit permettre d'afficher les différentes informations renseignées de la tache. De plus il doit permettre d'afficher la liste de toutes les taches enregistrées.
Rechercher une tache	Le système doit rechercher une tache selon son statut () et/ou selon son assignation à un membre donne (par son Id). Le système doit afficher les informations de la tache recherchée.

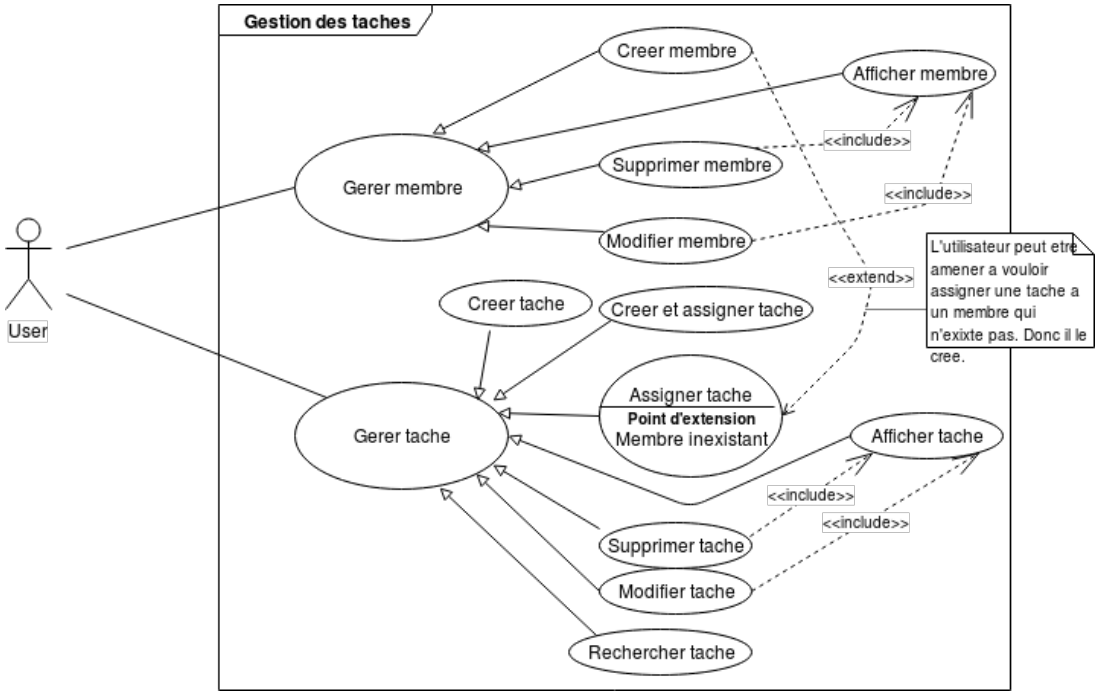

FIGURE *1 – Diagramme de cas d'utilisation*

TABLE *2 – Les exigences non fonctionnelles de l'application*

Exigences	Description	Exigences fonctionnelles requises
Disponibilité	Le système doit être disponible durant toutes les opérations	Toutes les exigences fonctionnelles
Disponibilité	Les données doivent être disponibles et accessibles à tout moment où l'utilisateur en a besoin	Toutes les exigences fonctionnelles
Intégrité	En cas de défaillance du système, les données ne doivent pas subir un changement	Toutes les exigences fonctionnelles
Performance	Le Système doit répondre rapidement aux demandes de l'utilisateur	Toutes les exigences fonctionnelles

2.2 Conception de l'application

Diagramme de classes. Après avoir illustré les besoins des utilisateurs de notre système par le diagramme de cas d'utilisation, nous passons à l'aspect conceptuel avec un des diagrammes structurels à savoir le diagramme de classes. Il est généralement considéré comme le plus important dans le développement orienté objet. Il représente l'architecture conceptuelle de notre système (Voir FIGURE 2).

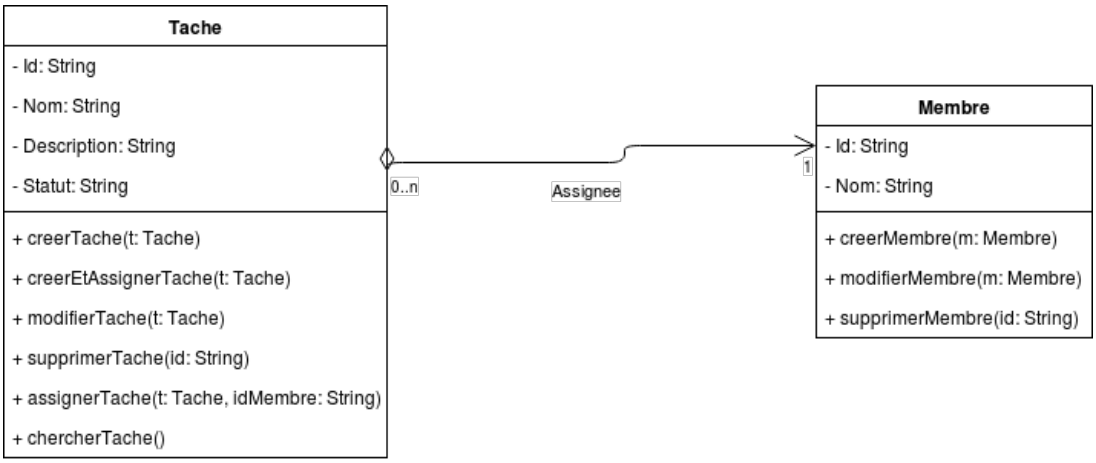

FIGURE *2 – Diagramme de classe*

Pour notre système, nous avons identifié principalement deux (02) classes à savoir :

— La classe Membre. Pour uns instance de Membre, nous pouvons avoir plusieurs taches.

— La classe Tache. Pour une instance de Tache, nous avons un seul membre.

Quelques diagrammes de séquences. Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur, ce qui fait de lui un des diagrammes d'interaction (dynamique).

Les FIGURES 3 4 5 illustrent quelques interactions avec le système.

2.3 Outils de réalisation

Le programme de Gestion de taches sera codé en Java. Pour ce faire, nous utilisons l'interface de développement intégré (IDE) Eclipse. Le choix du langage Java n'est pas fortuit. En effet, en plus d'être un choix de l'équipe projet, ce langage est le mieux pour illustrer la programmation orientée objet (POO). Pour les tests unitaires, nous utilisons l'outil Junit.

Pour la gestion des codes sources, nous allons travailler sur la plate-forme Github (github.com).

Pour le développement et les expérimentations nous allons utiliser un ordinateur portable avec les caractéristiques suivantes :

— Processeur : Intel(R) CoreT M i5-M370 @2.4 GHZ

— RAM : 8.00 Go

— OS : fedora 26

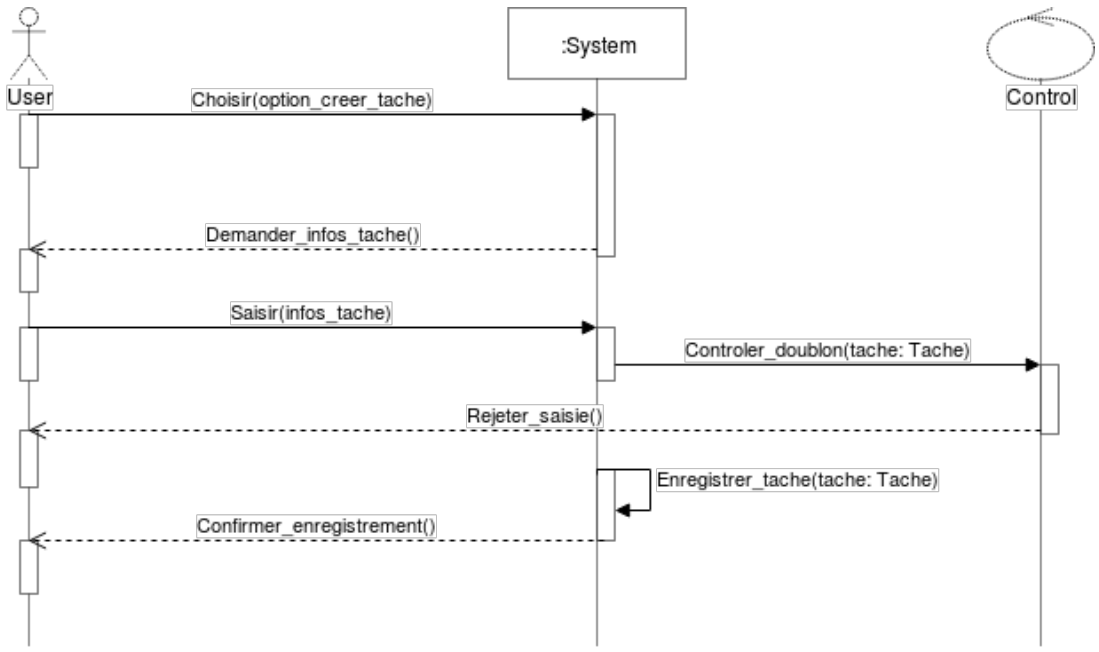

FIGURE *3 – Diagramme de séquence : Créer tache*

— JDK8

3 Implementation

Après avoir spécifié notre application en utilisant le langage UML, nous sommes passé à son implémentation avec le langage JAVA sur l'environnement Eclipse. Nous avons ainsi créer une classe principale sous le nom « TaskManager.java » qui contient la méthode principale (main) permettant l'exécution du programme.

Le choix du langage Java et de l'IDE Eclipse n'est pas fortuit. En effet, en plus d'être une exigence du projet, le langage Java est par « défaut » le langage de la programmation orientée objet. Il existe bien sur d'autres langages tels que C++. Mais, Java est plus flexible et offre de nombreux avantages notamment en terme de librairies. Quant à l'IDE Eclipse, il offre beaucoup d'avantages et est assez intuitif dans la programmation. De plus, il intègre de nombreux librairies. Comme exemple, pour les tests unitaires nous avons utilisé Junit qui est intégré dans Eclipse.

Nous avons implémenté plusieurs tests unitaires dont nous présentons quatre cas dans ce rapport. Il s'agit de la création d'un membre, d'une tache et de l'affichage des membres et des taches (voir (FIGURE 6)).

4 Tests d'acceptation

Ces tests visent à assurer formellement que le programme est conforme aux spécifications (réponse donnée à un instant t aux attentes formulées).

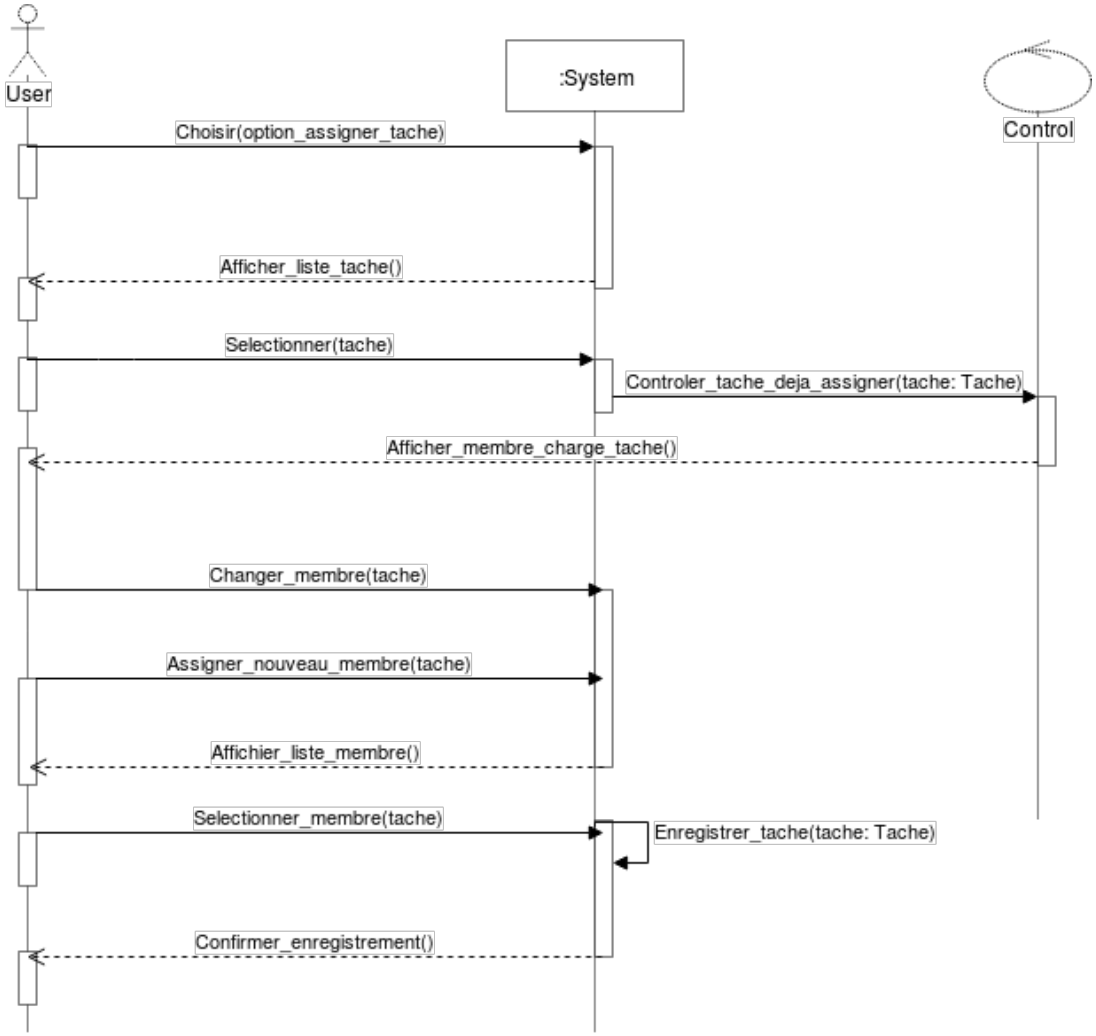

FIGURE 4 – *Diagramme de séquence : Assigner tache*

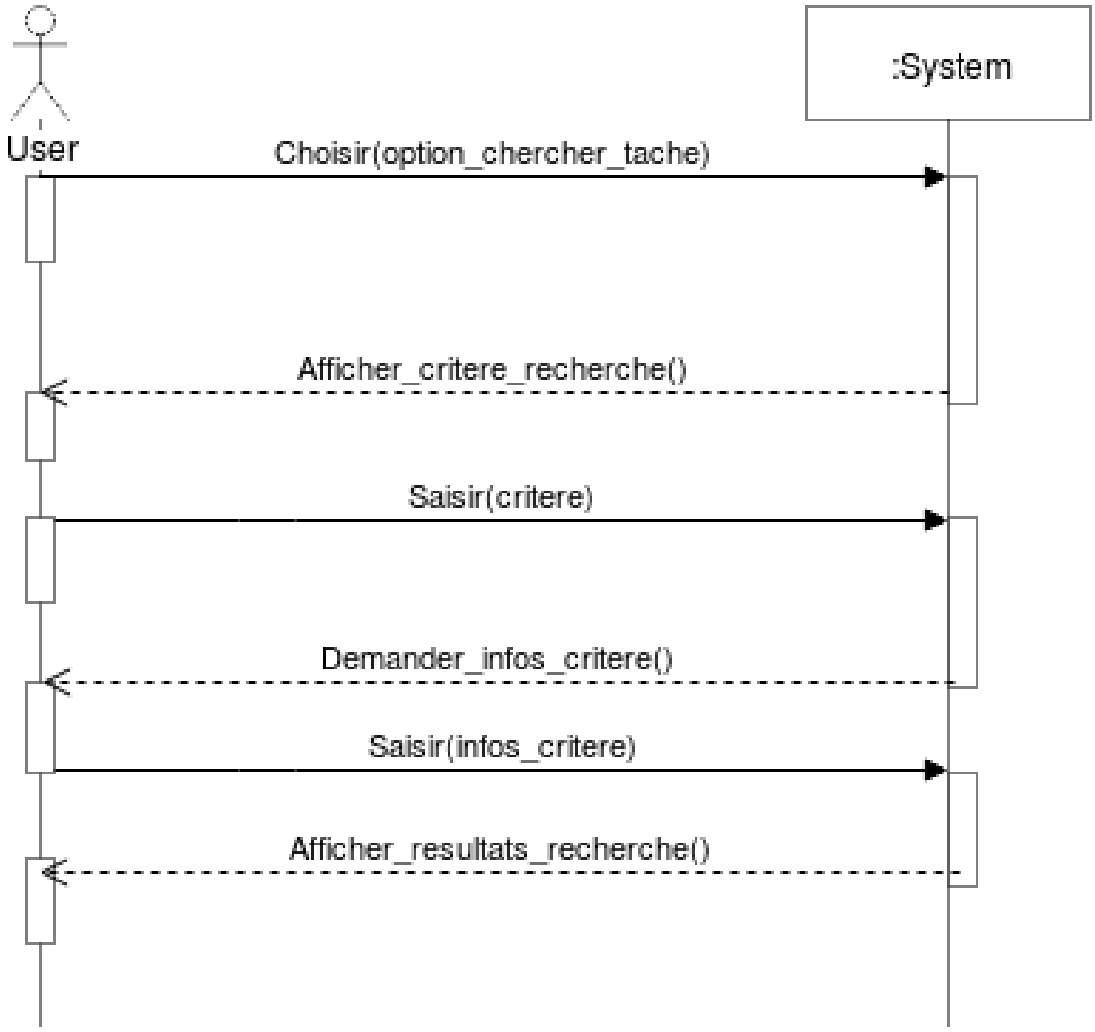

FIGURE 5 – *Diagramme de séquence : Chercher tache*

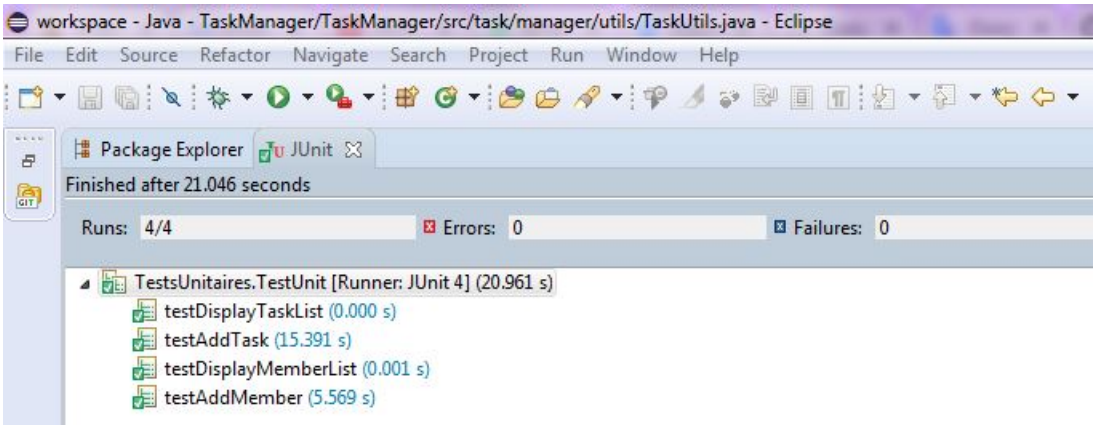

FIGURE 6 – *Tests unitaires : création d'un membre, d'une tache, de l'affichage des membres et des taches*

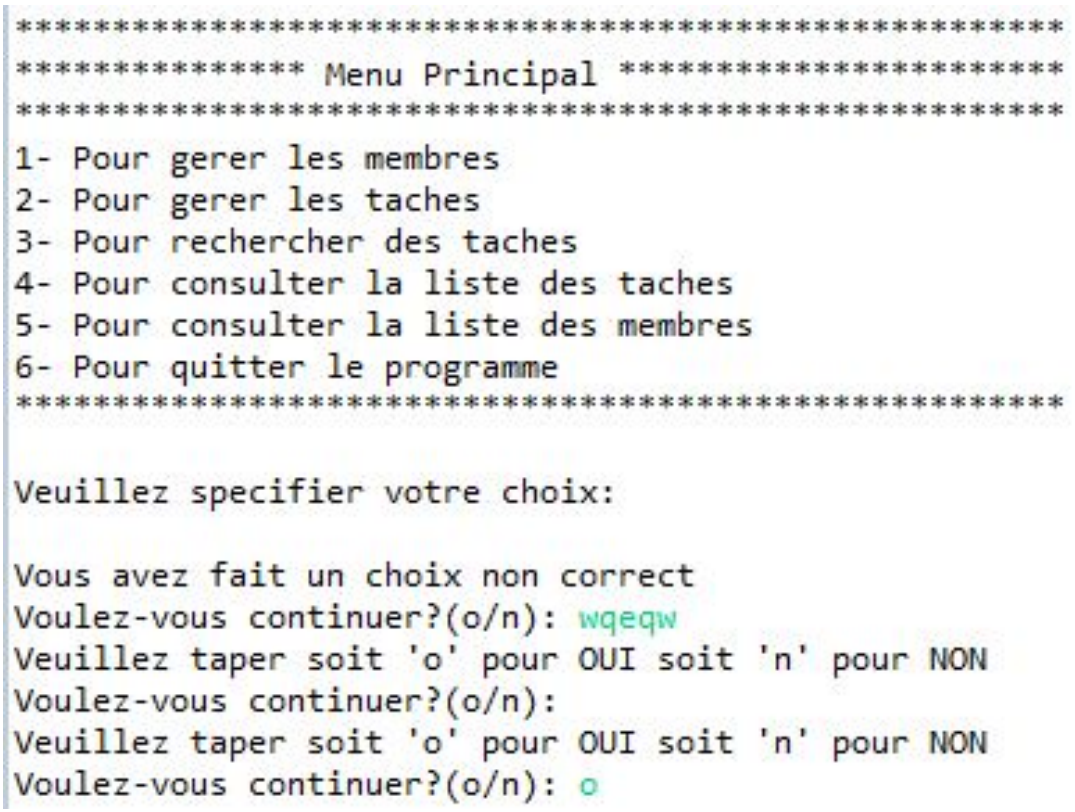

FIGURE 7 – *Test d'acception choix des opérations du MENU PRINCIPAL*

Dans cette partie de notre rapport, nous allons présenter quelques tests d'acceptation dans l'exécution de notre programme.

— Créer un membre : l'utilisateur saisit le nom du nouveau membre. Pour valider la création, il tape sur la touche « Enter » pour enregistrer les informations saisies. Nous avons aussi fait un contrôle dans la saisie des valeurs à entrer. En effet, notre application étant sans interface graphique, nous avons un MENU avec des choix bien définis à effectuer pour chaque opération souhaitée. Par ailleurs, nous refusons toutes « les saisies vides » et tout choix erroné. Les FIGURES 7 et 8 illustrent ces cas de tests.

— Supprimer un membre/tache : pour effectuer une suppression (ou modification), nous affichons la liste des membres/taches pour que l'utilisateur puisse choisir le membre/tache souhaité. Pour ce, le système lui demande de saisir l'Id du membre/tache. Une fois cela effectué, le système après vérification de l'existence du membre/tache, demande une confirmation de suppression puis lui envoie un message de la suppression effectuée. Lorsque le membre/tache n'existe pas, le programme demande de choisir le membre/tache dans la liste affichée. La FIGURES 9 montre ce cas de figure.

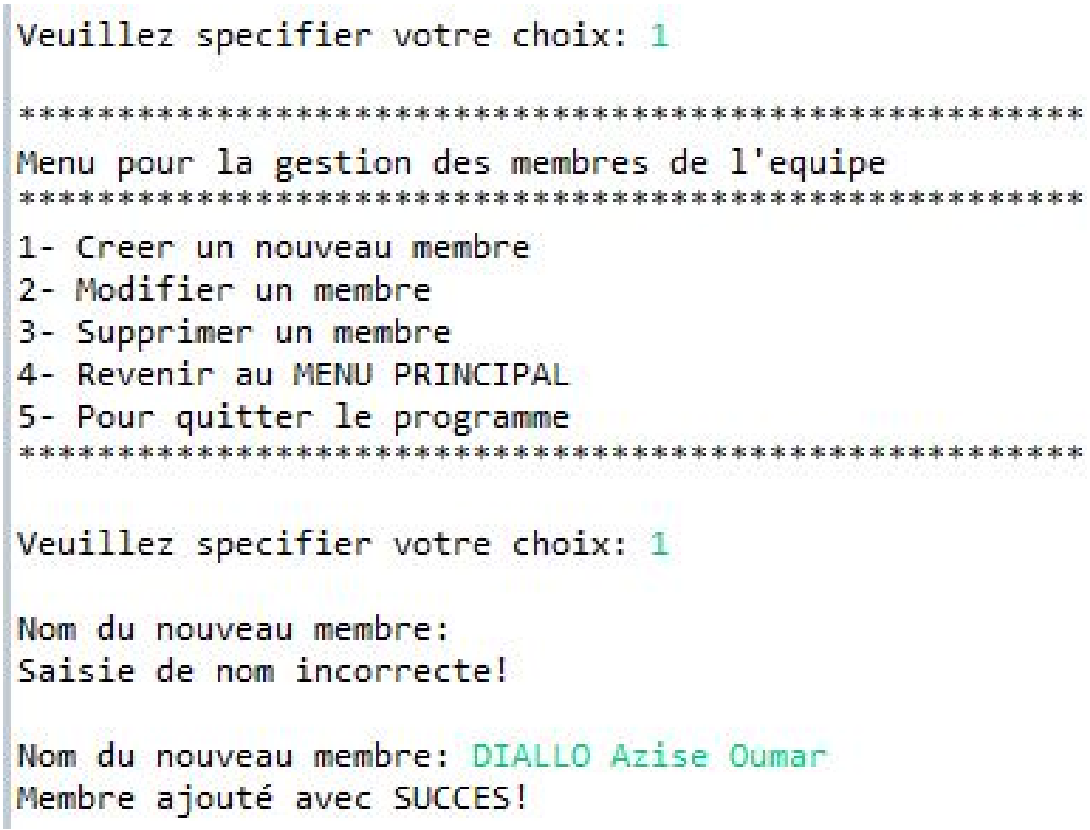

FIGURE *8 – Test d'acception Créer d'un membre*

```
******************************************************
Menu pour la gestion des membres de l'equipe
******************************************************
1- Creer un nouveau membre
2- Modifier un membre
3- Supprimer un membre
4- Revenir au MENU PRINCIPAL
5- Pour quitter le programme
******************************************************

Veuillez specifier votre choix: 3
La liste des membres est :
------------------------------------------------------------
ID    |   NOM
------------------------------------------------------------
1     |   DIALLO Azise Oumar
------------------------------------------------------------
2     |   KAFANDO Rodrigue
************************************************************

Saisir l'identifiant du membre à supprimer : 5

Le membre que vous avez specife n'existe pas dans la liste des membres
Merci de selectionner un membre existant

Saisir l'identifiant du membre à supprimer : 2
Voulez-vous vraiment supprimer ce membre (o/n) ??? : o

Suppression effectuee!
```

FIGURE *9* – *Test d'acception Suppression d'un membre*

5 Conclusion générale

Après avoir analysé le sujet, nous avons pu spécifier l'application à concevoir en utilisant le langage de modélisation UML. Pour ce faire, nous avons conçu un digramme de cas d'utilisation pour représenter les exigences fonctionnelles de l'application. Puis nous avons conçu un diagramme de classe et quelques diagrammes de séquence pour la conception de l'application.

Pour implémenter notre programme, nous avons utilisé le langage Java et travaillé sur l'environnement Eclipse. Par ailleurs, nous avons utilisé l'outil « Junit » pour effectuer les tests unitaires. De plus, pour la gestion des codes sources, nous avons travaillé sur la plate-forme Github (github.com).

Nous estimons ne pas avoir assez modulariser notre programme pour mieux faire ressortir la programmation orientée objet. En effet, nous pensions que nous pouvions faire plus de méthodes et de classes pour la gestion des opérations. Par ailleurs, notre programme actuel n'enregistre pas automatiquement les données dans un fichier ou une base de données. C'est le plus grand manquement de notre travail. Cela s'explique principalement par notre manque de maîtrise en programmation. En outre, le temps limite de ce TP nous a pas permis d'approfondir nos connaissances dans ce domaine.

En somme, nous pouvons dire que ce premier travail pratique a été très instructif pour nous. En effet, il nous a permis de nous rappeler les concepts de modélisation avec UML et d'acquérir des connaissances dans la programmation orientée objet. Par ailleurs, nous avons appris à utiliser de nouveaux outils tels que Junit et la plate-forme Github.

Difficultés rencontrées. L'une des difficultés majeures que nous avons rencontrées pour ce premier TP était la modularisation des différentes méthodes. En effet, nous avons souhaité créer le maximum de méthodes possibles pour gérer l'ensemble des opérations.

Perspectives. Notre futur travail consistera à améliorer d'avantage notre application en le modularisant d'avantage. De plus, l'intégration d'une interface graphique permettra de rendre plus conviviale l'interaction avec l'application. Enfin, nous pensions associer un gestionnaire de base de données pour une gestion plus fluide et efficace des données du gestionnaire (membres, taches).

6 Annexe

Dans le souci de ne pas rallonger notre rapport, nous ne mettrons pas les codes sources. Cependant, les codes se trouvent sur la plate-forme Github à l'adresse : https ://github.com/diallitoz/TP1_GLA-GestionnaireDeTache.