INSTITUT FRANCOPHONE INTERNATIONAL

ĐHQGHN

VNU
ĐẠI HỌC QUỐC GIA HÀ NỘI
Vietnam National University, Hanoi

TRAVAIL PERSONNEL

MODULE : GENIE LOGICIEL AVANCE

CONCEPTION ET REALISATION D'UNE APPLICATION DE GESTION DE TACHES

Période **: du 08 Août au 19 Août 2017**

Rédigé par :

MEDOU DANIEL MAGLOIRE, Etudiant en Master 1
des Systèmes Intelligents et Multimédia, IFI.
Promotion 21

Enseignant :

Dr. HO TUONG VINH

Année académique
2016-2017

LISTE DES TABLEAUX ET FIGURES

TABLE DES MATIERES

INTRODUCTION

Dans le but de se rappeler des concepts de modélisation avec UML, de se familiariser avec un environnement de développement intégré (IDE) libre et Open-Source comme éclipse et Netbeans et de la programmation orientée objet avec java comme langage de programmation, le TP1 dans le cadre du module de Génie Logiciel Avancé, a été mis à notre disposition avec pour objectif général la conception et la réalisation d'un gestionnaire de tâches. Mettre sur pied une application de gestion est un processus qui comprend de nombreux domaines parmi lesquels : *Faire une étude du sujet, réunir toutes les ressources favorables au bon déroulement du projet, analyser le système, le concevoir, le réaliser et enfin le déployer*. Le présent rapport nous donnera de façon détaillée les axes du côté conceptuel ainsi que ceux du côté réalisation de notre projet selon le plan prédéfini par le maître d'ouvrage.

I. EXIGENCES

1. Exigences fonctionnelles

Les exigences fonctionnelles listent les opérations réalisables avec l'application, elles correspondent également à la manipulation et précisent l'environnement de l'application. Il s'agit des fonctionnalités du système. De ce fait, notre application devra nous permettre de :

- ❖ Gérer les tâches
  - Créer une tâche
  - Modifier une tâche
  - Supprimer une tâche
  - Assigner une tâche à un membre
  - Rechercher et afficher toutes les tâches assignées à un membre (par son ID)
  - Rechercher et afficher toutes les tâches en fonction de leur statut (avec le nom de l'assigné)
- ❖ Gérer les membres
  - Créer un membre
  - Modifier un membre
  - Supprimer un membre

Le diagramme de cas d'utilisation étant l'un des cinq diagrammes d'UML qui représentent la vue statique d'un système, est utilisé à ce niveau pour montrer l'interaction entre l'utilisateur ou les utilisateurs du futur système et les différentes fonctionnalités de ce dernier. Avant la réalisation du diagramme de cas d'utilisation du futur système, il est important pour nous de faire mention de la liste des futurs ou potentiels utilisateurs ainsi que la liste des cas d'utilisations de chaque acteur s'il en existe plusieurs. Dans notre cas, nous aurons à faire avec un seul **acteur (**entité externe, physique ou logique qui interagit avec le système étudié.**)** qui sera le potentiel utilisateur du système et interagira avec les différentes fonctionnalités ou cas d'utilisations citées plus haut.

a. Formalisme du diagramme

Le diagramme de cas d'utilisation comporte les éléments suivants :

- ✓ **Un acteur, r**eprésenté par un bonhomme avec son nom inscrit dessous ;
- ✓ **Le Système,** représenté par un rectangle avec son nom inscrit au-dessus ;
- ✓ **Le cas d'utilisation,** représenté par une *ellipse* contenant le nom du cas (généralement un verbe à l'infinitif). Les cas d'utilisation sont représentés à l'intérieur du système ;
- ✓ **Les relations,** elles sont multiples à savoir
  - **La relation d'association**, chemin de communication entre un acteur et un cas d'utilisation. Elle est représentée par un *trait continu.*
  - **La relation d'inclusion ou relation « include »,** elle est l'une des relations existantes entre deux cas d'utilisation. Elle montre que le cas d'utilisation de base utilise systématiquement le cas inclus. Elle est représentée par une flèche interrompue, stéréotypé d'un « ***include*** », d'un « ***extend*** » *et* « ***use*** ».

- **La relation de généralisation ou relation d'héritage**, cette relation est existante dans les deux situations (entre cas et entre acteur).

La représentation graphique d'un diagramme de cas d'utilisation simplifié se présente comme suit :

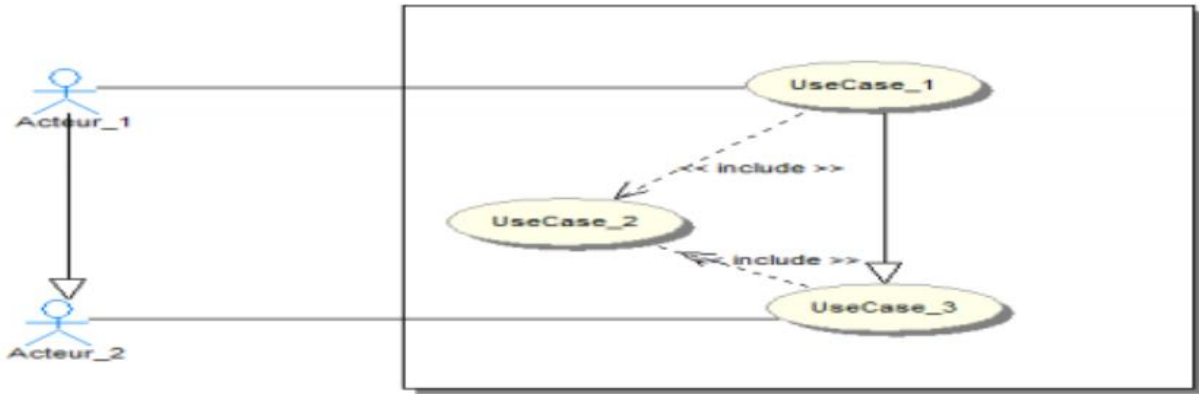

Figure 1: Formalisme du diagramme de cas d'utilisation

b. Représentation du diagramme de cas d'utilisation du système

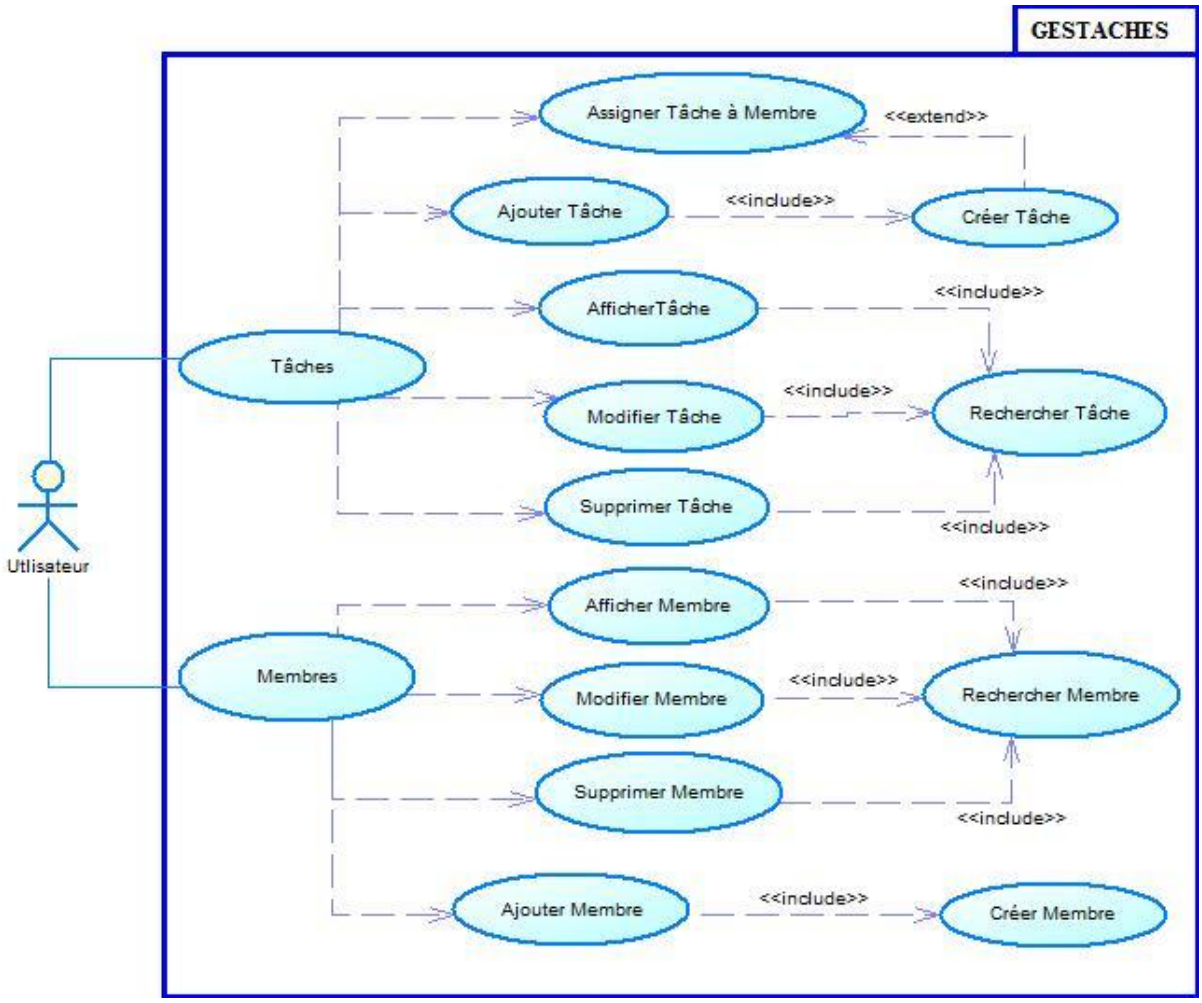

Figure 2: Diagramme de cas d'utilisation du système

2. Exigences non fonctionnelles

Ils accompagnent les exigences fonctionnelles et sont généralement les spécificités avec lesquelles les besoins fonctionnels doivent être réalisés. Ce sont les besoins en matière de performance, de type de matériel ou de type de conception. Ces besoins peuvent concerner les contraintes d'implémentation (langage de programmation, type de SGBD, de système d'exploitation…)

- Contraintes logicielle et d'analyse

Le projet en lui doit être analysé et modélisé avec le langage UML en utilisant le processus 2TUP, Java (JEE ou JSE) comme langage de programmation et éclipse ou NetBeans comme environnement de développement intégré et Open-Source. Pas de contrainte pour la base de données mais, nous avons choisi de travailler avec « **sqlite-jdbc-3.20.0** ».

- Contrainte de temps

Le produit et le rapport devront être livrés dans un délai de dix-sept (17) jours.

Dans le cadre de ce travail, l'application devra être extensible, c'est-à-dire qu'il pourra y avoir une possibilité d'ajouter ou de modifier de nouvelles fonctionnalités.

II. CONCEPTION

La conception nous permet de présenter l'aspect technique de l'application qui sera réaliser, de ressortir son architecture et aussi l'ensemble des composants nécessaire à son bon fonctionnement. Dans la cadre de notre projet, nous allons illustrer cette partie par deux diagrammes UML l'un de la vue statique donc le diagramme de classe et l'autre de la vue dynamique le diagramme de séquences [1].

1. Diagramme de classe

Le diagramme de classe est considéré comme le plus important de la Modélisation Orienté Objet [2]. Il présente la structure interne du système modélisé. Le diagramme de classe permet de fournir une représentation abstraite des objets du système qui vont intervenir ensemble pour réaliser les cas d'utilisations. Les principaux éléments du diagramme de classe sont : les classes et les relations (Généralisation, dépendance, associations etc.).

a. Formalisme du diagramme de classe

Le diagramme de classe le plus simple présente deux classes et une association comme le montre la figure :

Figure 3: Formalisme simplifié du diagramme de classe.

b. Diagramme de classe du système à réaliser

Il est tout de même possible qu'un diagramme de classe présente une ou plusieurs classe-association. Mais dans cas contrait de notre projet, le nôtre ne présente que deux classe raison pour laquelle le choix du formalisme à deux classe dans notre rapport. Dans le cadre de visibilité des attributs et des opérations, le document [2], dans sa page 37 nous renseigne sur les différents signes à utiliser devant les attributs ou les opérations afin de montrer ou signifier le type de visibilité autorisé pour les autres classes. Nous avons les droits associés à chaque niveau de confidentialité qui sont :

- ✓ **Public (+)** – Attribut ou opération visible par tous ;
- ✓ **Protégé (#)** – Attribut ou opération visible seulement à l'intérieur de la classe et pour toutes les sous-classes de la classe ;
- ✓ **Privé (-)** – Attribut ou opération seulement visible à l'intérieur de la classe ;
- ✓ **Paquetage (~)** – Attribut ou opération ou classe seulement visible à l'intérieur du paquetage où se trouve la classe.

Sur ce, nous allons, conformément à l'utilisation de ces annotations des droits de visibilité représenter le diagramme de classe de notre futur système. Diagramme ci-dessous.

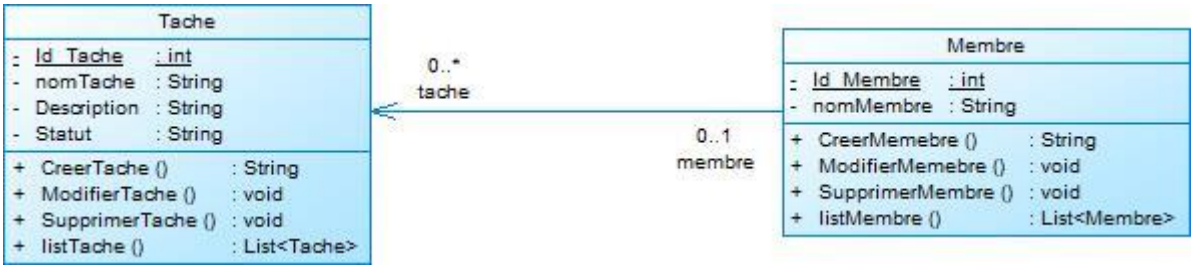

Figure 4: Diagramme de classe du système

Nous pouvons le lire comme suit :

- ✓ Un membre peux ou ne pas effectuer plusieurs tâches
- ✓ Pour une tâche donnée, nous voulons un membre correspondant.

2. Diagramme de séquence

Dans [2], **un diagramme de séquence** permet de documenter les interactions à mettre en œuvre entre les objets pour réaliser un résultat tel qu'un cas d'utilisation. Nous pouvons également dire que ce dernier permet de décrire les différents scénarios d'utilisation du système. Pour réaliser un diagramme de séquence, nous nous basons sur le diagramme des cas d'utilisation.

a. Formalisme du diagramme de séquences

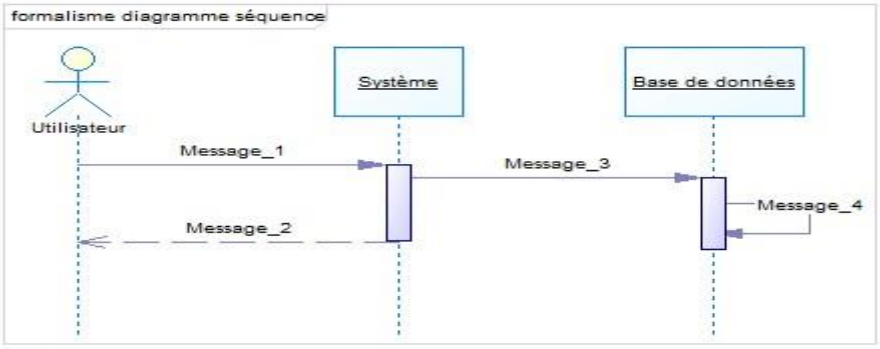

Figure 5: Formalisme du diagramme de séquence

b. Description textuelle des scénarii du cas d'utilisation à représenter

Dans [2] de la page 84 à 85, le document nous présente le processus de description textuelle d'un cas d'utilisation. Nous allons décrire le cas d'utilisation « **modifier un membre** »

Nom **: Modifier un membre**
Résumé **: Ce cas permet à l'utilisateur de modifier un membre existant dans le système.**
Acteur **: Utilisateur**
Pré-condition **: Aucune**
Déclencheur **: l'utilisateur saisi le chiffre correspondant à la modification du membre.**
Scénario nominal **:** **1. Le système affiche la liste des membres** **2. L'utilisateur renseigne les champs et modifie ce dont il en a besoin et valide** **3. Le système vérifie le type des données**
Scénario alternatif **:** **1. Données mal saisie à la confirmation** **2. Retourner à l'étape 2** **3. L'utilisateur corrige l'erreur** **4. Le cas d'utilisation reprend à l'action 3 du scénario nominal**
Post-condition **: Mise à jour du membre a été bien faite et enregistrée avec succès**

c. Diagramme de séquence « **Modifier le membre** »

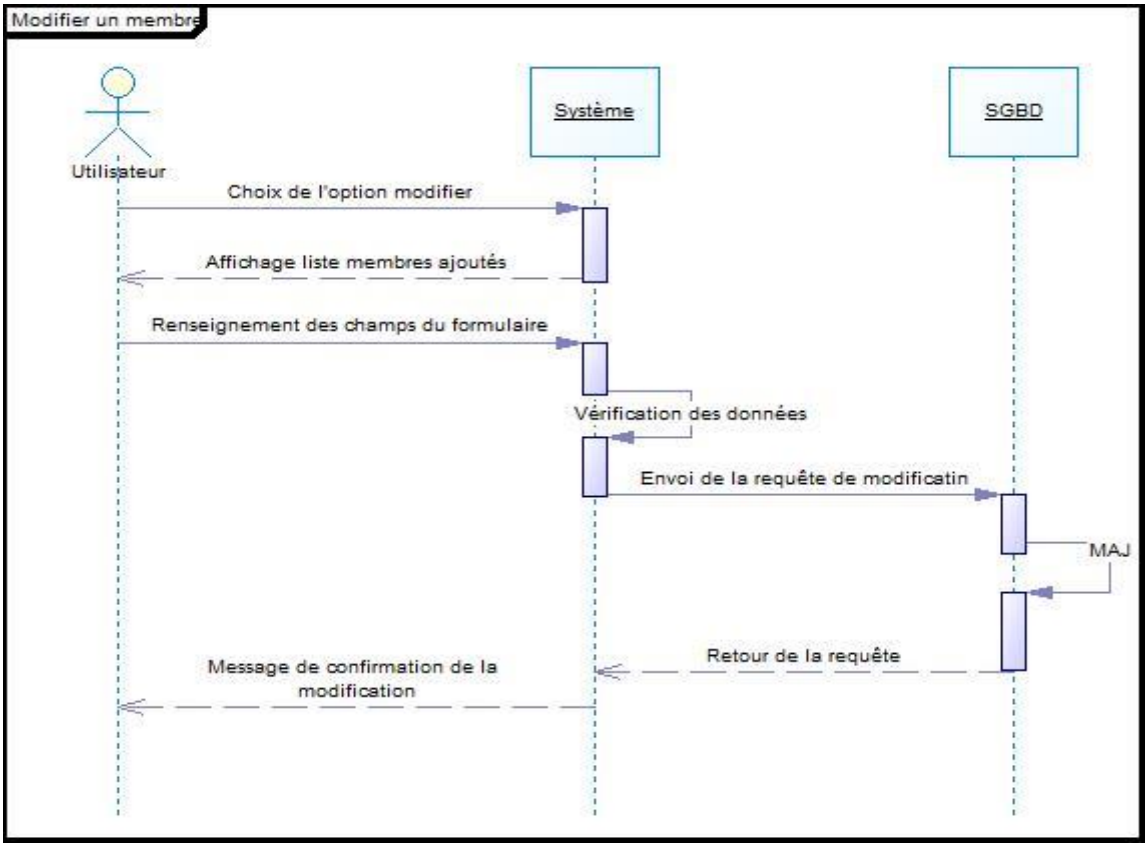

Figure 6: Diagramme de séquence du système

III. IMPLEMENTATION

Cette partie nous permet de traduire dans un langage quelconque de programmation les différentes fonctionnalités définies lors de conception de notre application. De ce faite, notre logiciel est développé en JAVA SE comme langage de programmation sous Linus (16.04) comme système d'exploitation et via l'IDE Eclipse.

Deux classes ont été créées dans le cadre de l'implémentation de notre logiciel à savoir le classe « **Tache** » et « **Membre** » et chacune définissant les attributs et méthodes. Ces classes, au cours de notre implémentation, ce sont vues ajouter d'autres comme « **ApplicationTP1** », «**DataBaseConnexion** » toutes ayant des méthodes. Chaque méthode est soumisse à un test unitaire **JUnit** pour vérifier que chaque sous-ensemble de notre système implémenté est conforme aux spécifications.

1. Diagramme de classes générale de « GESTACHE » : Vue côté développeur

Ce dernier représente la structure interne de notre logiciel donc, toutes les classes et est présenté ci-dessous.

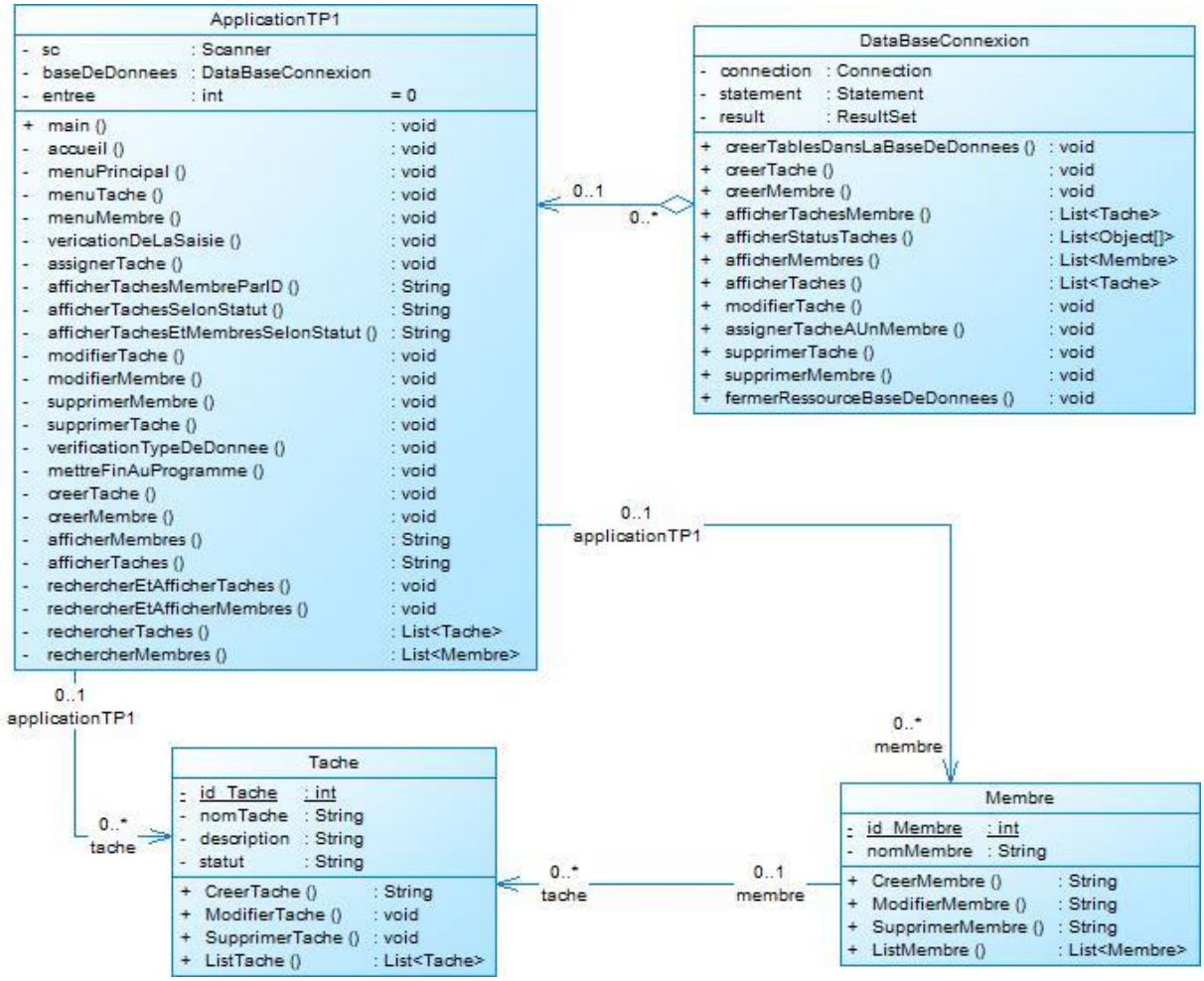

Figure 7: Digramme des classes utilisées pour tout le système

2. Test unitaire : cas de JUnit

JUnit est un Framework ou un outil nous donnant la possibilité d'écrire et d'exécuter des tests unitaires sur des programmes implémentés en Java. Cependant, nous avons comme objectif pour notre TP1 d'écrire et d'exécuter des tests avec JUnit pour un ensemble de classes Java qui implémentent notre Gestionnaire de tâche (GESTACHE). Ceci à partir des spécifications données. Dans le document [3], les tests en **JUnit** sont regroupés au sein d'une classe Java qui doit hériter de la classe **TestCase**. Le nom des méthodes de test doit commencer par **test**. Le corps d'une méthode de test doit comporter trois parties :

- ✓ **le préambule**, qui permet l'initialisation des objets sur lesquels le test va être exécuté ;
- ✓ **le corps de test,** dans lequel la méthode à tester est appelée sur les objets créés ;
- ✓ **le post ambule,** qui permet de délivrer le verdict du test (succès ou échec) en vérifiant un ensemble de propriétés (assertions) sur l'état des objets après le test.

Les méthodes utilisées pour effectuer le test unitaire de « GESTACHE » sont les suivantes :

- ✓ rechercherMembres ()
- ✓ rechercherTaches ()

Les classes de ces méthodes sont respectivement « JUnitRechercherTache.java » et « JUnitRechercherMembre.java » qui sont exécutées avec succès et une classe les regroupant pour les passer en test ensemble appelée « AllTests.java ».

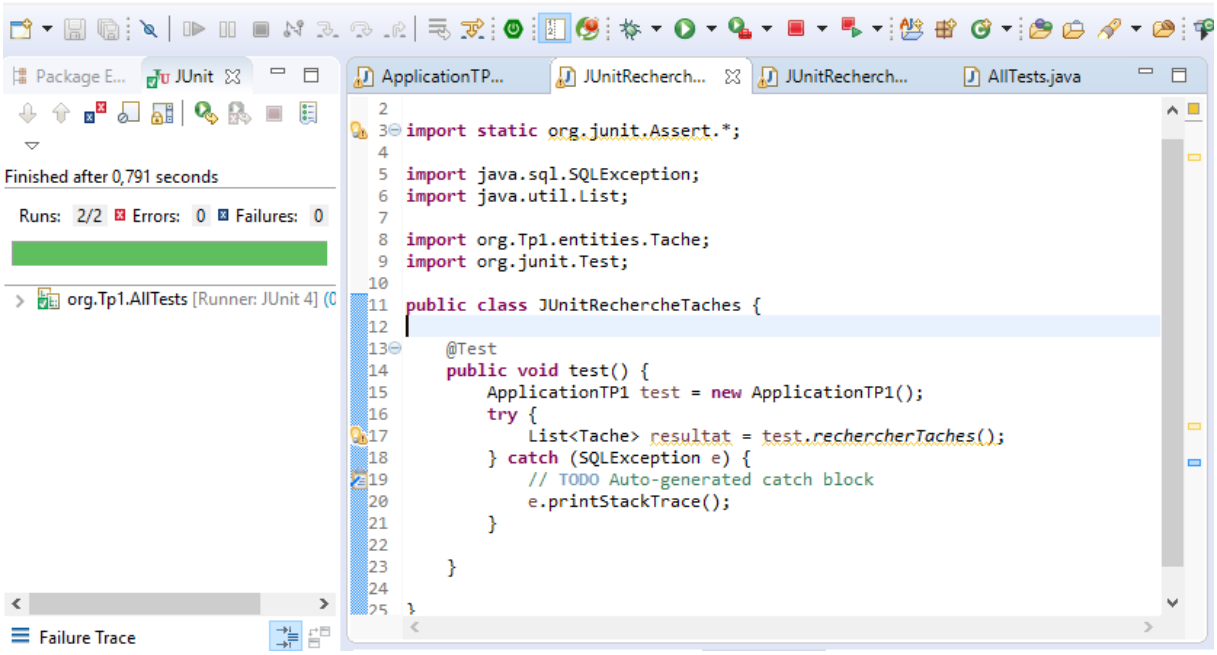

Figure 8: Ecran test JUnit de "rechercherTaches"

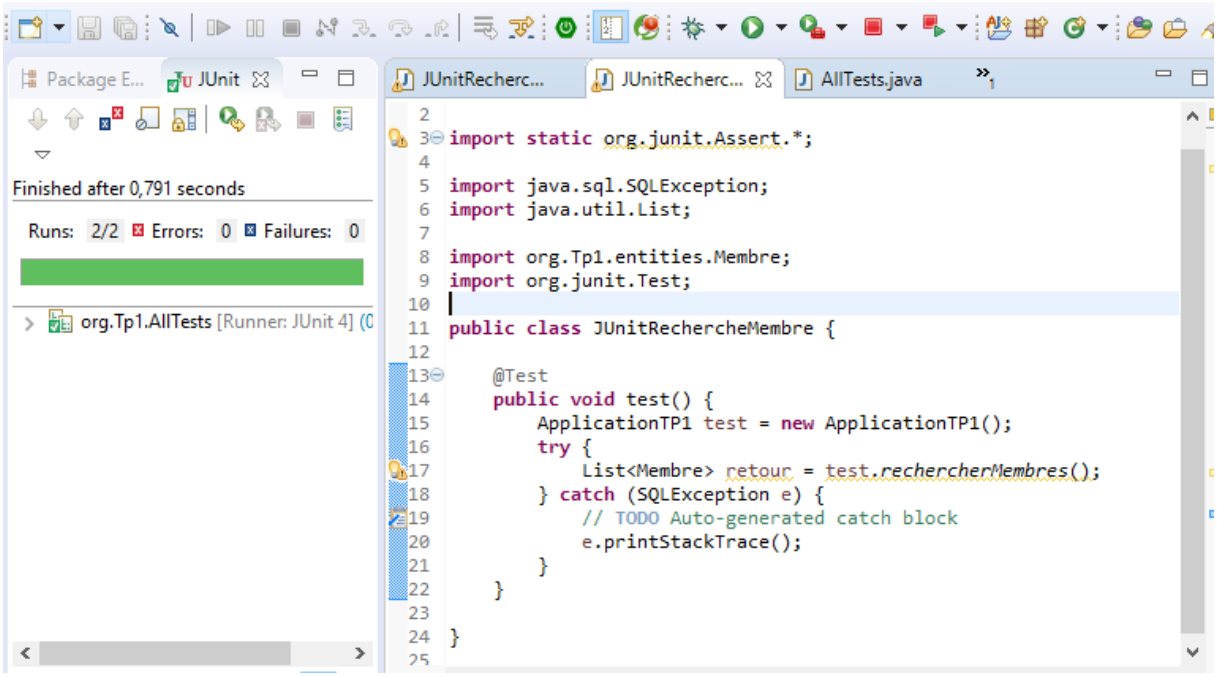

Figure 9: Ecran test JUnit de "rechercherMembres"

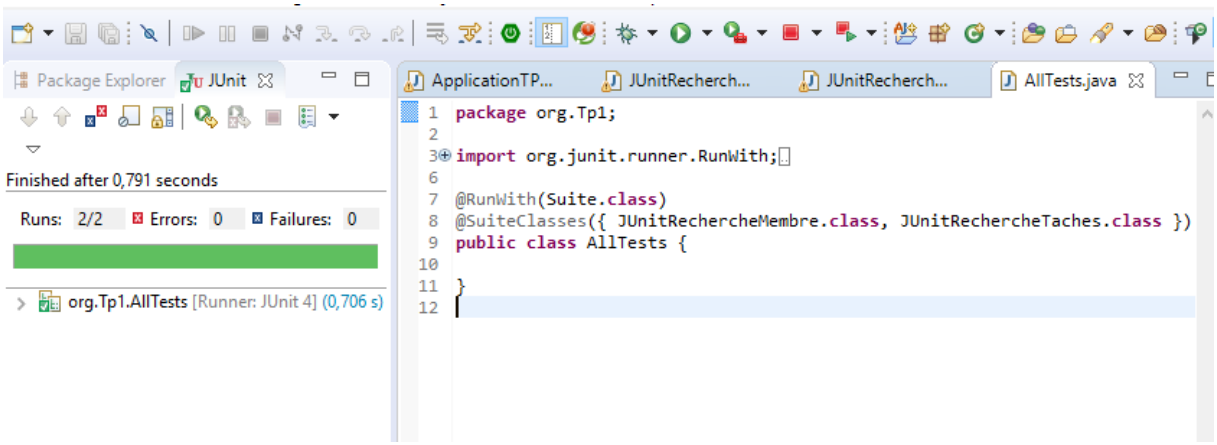

Figure 10: Ecran test JUnit de "AllTests"

IV. TEST D'ACCEPTATION

```
================================================
!!! MODULE DE GENIE LOGICIEL AVANCE!!!


 TP1.  MEDOU DANIEL MAGLOIRE

================================================




                MENU PRINCIPAL DE GESTACHE
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1 : GESTION DES TACHES
2 : GESTION DES MEMBRES
3 : FIN

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 11: Ecran principal

1. Gestion des tâches

✓ Taper 1 pour accéder au menuTache

```
Choix : 1

                TACHES
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1 : CREER TACHE
2 : MODIFIER TACHE
3 : SUPPRIMER TACHE
4 : ASSIGNER TACHE
5 : AFFICHER TACHES
6 : AFFICHER TACHES MEMBRE PAR SON ID
7 : AFFICHER TACHES PAR STATUT
8 : SORTIR

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 12: Menu de gestion de Taches

```
Choix : 1


~~~CREATION DE TACHE~~~

NOM DE LA TACHE :
Modelisation
DESCRIPTION DE LA TACHE :
nom TacheModelisation
Concevoir des modèles pour le projet en cours
STATUT DE LA TACHE : (nouveau, en-progres ou termine)
nouveau
Voulez-vous assigner cette nouvelle tache a un membre?  1 : OUI 2 : NON
2
Voulez-vous ajouter la tache nouvellement cree
1 : OUI 2 : NON
```

Figure 13:Céer une tache si possible assigner à un membre et ajouter cette tache créée

- ✓ Taper 6 et suivre la procédure de saisie des ID pour afficher toutes les tâches assignées à un membre par son ID

```
Liste des taches du membre en recherchant par son ID


ID : 5
NOM :
DESCRIPTION : Faire la conception du systeme
STATUT : nouveau
IDMEMBRE ASSIGNE A LA TACHE : 2


ID : 6
NOM : Cultiver
DESCRIPTION : Mettre les plants au sol
STATUT : en-progres
IDMEMBRE ASSIGNE A LA TACHE : 2


ID : 7
NOM : test
DESCRIPTION : test
STATUT : termine
IDMEMBRE ASSIGNE A LA TACHE : 2


ID : 9
NOM : Modelisation
DESCRIPTION : Concevoir des modèles pour le projet en cours
STATUT : nouveau
IDMEMBRE ASSIGNE A LA TACHE : 2
```

Figure 14: Liste des tâches assignées au membre ID = 2

- ✓ Etant dans le menu Tâche, taper 7 pour et saisir le statut pour afficher toutes les tâches en fonction de ce statut et le membre assigné.

```
Choix : 7
Entrez : nouveau, en-progres ou termine pour afficher les taches selon leur statut

nouveau
9
        Modelisation
        Concevoir des modèles pour le projet en cours
        nouveau
        2
        MEDOU

9
        Modelisation
        Concevoir des modèles pour le projet en cours
        nouveau
        2
        MEDOU

9
        Modelisation
        Concevoir des modèles pour le projet en cours
        nouveau
        2
        MEDOU

9
        Modelisation
        Concevoir des modèles pour le projet en cours
        nouveau
        2
        MEDOU
```

Figure 15: Liste des tâches en fonction du statut et le nom du membre que ces tâches sont assignées

2. Gestion des membres

- ✓ Taper 2 pour afficher le menu de gestion des « Membre » et l'écran suivant est affiché

```
Choix : 2


                MEMBRE
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

1 : CREER MEMBRE
2 : MODIFIER MEMBRE
3 : SUPPRIMER MEMBRE
4 : AFFICHER MEMBRE
5 : SORTIR

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Choix :
```

Figure 16: Ecran menu "GESTION MEMBRES"

- ✓ Taper 1 pour « CREER MEMBRE » et si vous souhaitez ajouter le membre créé dans la base alors saisir 1 sinon saisir 2. Dans le 1 nous avons l'écran avec message de confirmation d'ajout.

```
Choix : 1


~~~CREATION DE MEMBRE~~~

NOM DU MEMBRE :
ADJEME ANGOULA
Voulez-vous ajouter le membre nouvellement cree
1 : OUI 2 : NON
1

Membre cree et enregistre avec succes
```

Figure 17: Création et ajout du membre

- ✓ Pour modifier un membre, étant dans le menu GESTION MEMBRE, taper 2 et sur la touche ENTER du clavier et la liste de tous les membres existant dans la base de données s'affiche, saisir l'identifiant du membre à modifier puis taper sur la touche ENTER pour engager le processus de modification et un message de confirmation est aussitôt affiché.

```
Entrez l'identifiant du membre a modifier
5


Entrez le nom du membre
ANDING JULIENNE

Mise a jour du membre a ete bien faite et enregistree avec succes
```

Figure 18: Modification d'un membre

CONCLUSION

Arrivée au terme de notre travail dans le cadre du cours de génie logiciel avancé, donc l'objectif général était de concevoir et de réaliser une application de gestion de tâches et, avec des objectifs spécifiques tels que améliorer notre compréhension en programmation orientée objet avec Java comme langage de programmation et maitriser la conception avec le langage de modélisation UML, etc. Effet, nous avons réalisé notre projet informatique tout en respectant les exigences fonctionnelles du client. En ce qui concerne le travail dans son ensemble, nous notons que des efforts nécessitent encore d'être fournie pour la maîtrise et même les bonne pratique du concept « Orienté Objet » et du langage Java pour une meilleur programmation. En dernière analyse, le TP1 a été pour nous un moyen propice pour la mise en pratique des concepts basiques vue en cours dans le module de génie logiciel avancé.

REFERENCES

[1]: Ho Tuong Vinh, « ***Présentation d 'UML*** » IFI – 2012. Pages 26, 32-42, 5-60.

[2]: Joseph Gabay**, « *UML 2 ANALYSE ET CONCEPTION* »** Mise en œuvre guidée avec études de cas**.** Dunod, Paris, 2008. Pages 32-58, 81, 105

[3] : Prof. Burkhart Wolff, D. Longuet, Lina Ye « ***Génie Logiciel Avancé*** : ***TP - Test unitaire avec JUnit***». Année 2011-2012. Page 1-3.

ANNEXE (Code)

1. Classe ApplicationTP1

```
/**
 * Author: MEDOU Daniel Magloire
 * Unité d'enseignement: GENIE LOGICIEL AVANCE
 * Ecole: IFI
 * Niveau: Master 1
 * Option: Systèmes Intelligents et Multimédia (SIM)
 * Description: TP1, Conception et Réalisation d'une Application de Gestion de Tâches: "GESTACHE"
 */

package org.Tp1;

import java.sql.SQLException;
import java.util.List;
import java.util.Scanner;

import org.Tp1.BD.DataBaseConnexion;
import org.Tp1.entities.Membre;
import org.Tp1.entities.Tache;

public class ApplicationTP1 {

	private static Scanner sc = new  Scanner(System.in);
	private static DataBaseConnexion baseDeDonnees = new DataBaseConnexion();
	private  static int entree=0;



	//Methode main
	public static void main(String[] args) throws SQLException {
```

```
accueil();
while(true){
        menuPrincipal();
        entree=0;
        vericationDeLaSaisie(1,3);
        switch(entree){
                case 1:
                        entree=0;
                        menuTache();
                        vericationDeLaSaisie(1,8);
                                switch(entree){
                                        case 1 :
                                                //creerTache
                                                creerTache();
                                                break;
                                        case 2 :
                                                //modifierTache
                                                modifierTache();
                                                break;
                                        case 3 :
                                                //supprimerTache
                                                supprimerTache();
                                                break;
                                        case 4 :
                                                //assignerTache
                                                assignerTache();
                                                break;
                                        case 5 :
                                                //afficherTaches
                                                rechercherEtAfficherTaches();
                                                break;
```

```
                                        case 6 :
                                                //
                                                afficherTachesMembreParID();
                                                break;
                                        case 7 :
                                                //
                                                afficherTachesSelonStatut();
                                                break;
                                        case 8 :
                                                System.out.println("\n!!! Action
annullee. Retour au menu principal !!!");
                                                break;
                                }
                        break;
                case 2 :
                        entree=0;
                        menuMembre();
                        vericationDeLaSaisie(1,5);
                                switch(entree){
                                        case 1 :
                                                //creerMembre
                                                creerMembre();
                                                break;
                                        case 2 :
                                                modifierMembre();
                                                break;
                                        case 3 :
                                                supprimerMembre();
                                                break;
                                        case 4 :
                                                rechercherEtAfficherMembres();
```

```
                                                        break;
                                                case 5 :
                                                        System.out.println("\n!!! Action
annullee. Retour au menu principal !!!");
                                                        break;
                                                }
                                        break;
                                case 3 :
                                        mettreFinAuProgramme();
                                        break;
                        }
                }
        }

        private static void accueil(){
                System.out.println("\n\n=============================================");
                System.out.println("!!! MODULE DE GENIE LOGICIEL AVANCE!!!\n\n");
                System.out.println(" TP1.  MEDOU DANIEL MAGLOIRE\n");
                System.out.println("=============================================\n\n");
        }

        private static void menuPrincipal(){
                System.out.print("\n\n\t\tMENU PRINCIPAL DE GESTACHE\n");
                System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
                System.out.print("1 : GESTION DES TACHES\n");
                System.out.print("2 : GESTION DES MEMBRES\n");
                System.out.print("3 : FIN\n");
                System.out.print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\nChoix
: ");
        }

        private static void menuTache(){
```

```
                System.out.print("\n\n\t\tTACHES\n");
                System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
                System.out.print("1 : CREER TACHE\n");
                System.out.print("2 : MODIFIER TACHE\n");
                System.out.print("3 : SUPPRIMER TACHE\n");
                System.out.print("4 : ASSIGNER TACHE\n");
                System.out.print("5 : AFFICHER TACHES\n");
                System.out.print("6 : AFFICHER TACHES MEMBRE PAR SON ID\n");
                System.out.print("7 : AFFICHER TACHES PAR STATUT\n");
                System.out.print("8 : SORTIR\n");
                System.out.print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\nChoix
: ");
        }

        private static void menuMembre(){
                System.out.print("\n\n\t\tMEMBRE\n");
                System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
                System.out.print("1 : CREER MEMBRE\n");
                System.out.print("2 : MODIFIER MEMBRE\n");
                System.out.print("3 : SUPPRIMER MEMBRE\n");
                System.out.print("4 : AFFICHER MEMBRE\n");
                System.out.print("5 : SORTIR\n");
                System.out.print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\nChoix
: ");
        }

        private static void vericationDeLaSaisie(int borneInferieure, int borneSuperieure){
                while(entree<borneInferieure || entree>borneSuperieure){
                        verificationTypeDeDonnee();
                        entree=sc.nextInt();
                        if(entree<borneInferieure || entree>borneSuperieure){
```

```
                                System.out.println("Entrez un numero entre "+borneInferieure+" et
"+borneSuperieure+"\n");
                        }
                }
        }

        private static void assignerTache() throws SQLException{
                rechercherEtAfficherTaches();
                System.out.print("\nID Tache : ");
                verificationTypeDeDonnee();
                int identifiantTache = sc.nextInt();

                rechercherEtAfficherMembres();
                System.out.print("\nID Membre : ");
                verificationTypeDeDonnee();
                int identifiantMembre = sc.nextInt();

                baseDeDonnees.assignerTacheAUnMembre(identifiantTache, identifiantMembre);

        }

        private static void afficherTachesMembreParID() throws SQLException{
                rechercherEtAfficherMembres();
                System.out.print("\nID Membre : ");
                verificationTypeDeDonnee();
                int identifiantMembre = sc.nextInt();

                List<Tache> taches=baseDeDonnees.afficherTachesMembre(identifiantMembre);
                System.out.println("\nListe des taches du membre en recherchant par son ID\n");
                afficherTaches(taches);
        }
```

```
	private static void afficherTachesSelonStatut() throws SQLException{

		System.out.println("Entrez : nouveau, en-progres ou termine pour afficher les taches selon leur statut\n");

		sc.nextLine();

		String statut=sc.nextLine();

		while(!statut.equalsIgnoreCase("nouveau") && !statut.equalsIgnoreCase("en-progres") && !statut.equalsIgnoreCase("termine")){

			System.out.print("STATUT DE LA TACHE : (nouveau, en-progres ou termine)\n");

			statut=sc.nextLine();

			if(!statut.equalsIgnoreCase("nouveau") && !statut.equalsIgnoreCase("en-progres") && !statut.equalsIgnoreCase("termine")){

				System.out.println("Mauvaise saisie. Entrez : nouveau ou en-progres ou termine");

			}

		}


		int statusACchercher=0;

		if(statut.equalsIgnoreCase("nouveau")){

			statusACchercher=1;

		}else if(statut.equalsIgnoreCase("en-progres")){

			statusACchercher=2;

		}else{

			statusACchercher=3;

		}

		List<Object[]> tab = baseDeDonnees.afficherStatusTaches(statusACchercher);

		afficherTachesEtMembresSelonStatut(tab);

	}


	private static void afficherTachesEtMembresSelonStatut(List<Object[]> tab){

		Tache tache;

		Membre membre;
```

```
        for(Object obj[]: tab){
                tache=(Tache)obj[0];
                membre=(Membre)obj[1];
                System.out.println(tache.getId_Tache());
                System.out.println("\t"+tache.getNomTache());
                System.out.println("\t"+tache.getDesription());
                System.out.println("\t"+tache.getStatut());
                System.out.println("\t"+membre.getId_Membre());
                System.out.println("\t"+membre.getNomMembre());
                System.out.println();
        }
}

private static void modifierTache() throws SQLException{

        List<Tache> taches=rechercherTaches();
        afficherTaches(taches);
        System.out.println("\n\n");

        System.out.println("\n\nEntrez l'identifiant de la tache a modifier");
        verificationTypeDeDonnee();
        int identifiant=sc.nextInt();
        sc.nextLine();
        System.out.println("\n\nEntrez le nom de la tache");
        String nom=sc.nextLine();
        System.out.println("\n\nEntrez la description de la tache");
        String description=sc.nextLine();
        System.out.println("\n\nEntrez le statut de la tache");
        String statut=sc.nextLine();
        List<Membre> membres=rechercherMembres();
        afficherMembres(membres);
```

```
                System.out.println("\n\nEntrez l'identifiant du membre associe");
                verificationTypeDeDonnee();
                int identifiantMembre=sc.nextInt();

                baseDeDonnees.modifierTache(new
Tache(identifiant,nom,description,statut,identifiantMembre));
        }

        private static void modifierMembre() throws SQLException{
                List<Membre> membres=rechercherMembres();
                afficherMembres(membres);
                System.out.println("\n\nEntrez l'identifiant du membre a modifier");
                verificationTypeDeDonnee();
                int identifiant=sc.nextInt();
                sc.nextLine();
                System.out.println("\n\nEntrez le nom du membre");
                String nom=sc.nextLine();

                baseDeDonnees.modifierMembre(new Membre(identifiant, nom));
        }

        private static void supprimerMembre() throws SQLException{
                List<Membre> membres=rechercherMembres();
                afficherMembres(membres);
                System.out.println("\n\nEntrez l'identifiant du membre a supprimer");
                verificationTypeDeDonnee();
                int identifiant=sc.nextInt();
                baseDeDonnees.supprimerMembre(identifiant);
        }

        private static void supprimerTache() throws SQLException{
```

```
        List<Tache> taches=rechercherTaches();
        afficherTaches(taches);
        System.out.println("\n\nEntrez l'identifiant de la tache a supprimer");
        verificationTypeDeDonnee();
        int identifiant=sc.nextInt();
        baseDeDonnees.supprimerTache(identifiant);
}

//Fonction de verification de type
private static void verificationTypeDeDonnee(){
        if(!sc.hasNextInt()){
                System.out.print("Saisir votre numero d'operation !!!\nChoix :");
                sc.nextLine();
                verificationTypeDeDonnee();
        }
}

private static void mettreFinAuProgramme(){
        System.out.println("\nFermeture du programme !\n");
        System.exit(0);
}

private static void creerTache() throws SQLException{
        int idMembre=0;
        System.out.print("\n\n~~~CREATION DE TACHE~~~\n\n");
        System.out.print("NOM DE LA TACHE : \n");
        sc.nextLine();
        String nom=sc.nextLine();
        System.out.print("DESCRIPTION DE LA TACHE : \n");
        System.out.println("nom Tache" + nom);
        String description=sc.nextLine();
```

```
                String statut="";
                while(!statut.equalsIgnoreCase("nouveau") && !statut.equalsIgnoreCase("en-progres") && !statut.equalsIgnoreCase("termine")){
                        System.out.print("STATUT DE LA TACHE : (nouveau, en-progres ou termine)\n");
                        statut=sc.nextLine();
                        if(!statut.equalsIgnoreCase("nouveau") && !statut.equalsIgnoreCase("en-progres") && !statut.equalsIgnoreCase("termine")){
                                System.out.println("Mauvaise saisie. Entrez : nouveau ou en-progres ou termine");
                        }
                }
                                System.out.print("Voulez-vous assigner cette nouvelle tache a un membre?\t1 : OUI\t2 : NON\n");
                vericationDeLaSaisie(1,2);
                entree=sc.nextInt();
                if(entree==1){
                        List<Membre> membres = baseDeDonnees.afficherMembres();
                        afficherMembres(membres);
                        System.out.print("Entrez l'identifiant du tache :\n");
                        verificationTypeDeDonnee();
                        idMembre=sc.nextInt();
                }
                Tache tache = new Tache(0,nom,description,statut,idMembre);
                System.out.println("Voulez-vous ajouter la tache nouvellement cree\n1 : OUI\t2 : NON");
                entree=sc.nextInt();
                vericationDeLaSaisie(1,2);
                if(entree==1){
                        baseDeDonnees.creerTache(tache);
                }
                entree=0;
        }
```

```
	private static void creerMembre() throws SQLException{
		int idMembre=0;
		System.out.print("\n\n~~~CREATION DE MEMBRE~~~\n\n");
		sc.nextLine();
		System.out.print("NOM DU MEMBRE : \n");
		String nom=sc.nextLine();
		Membre membre = new Membre(0,nom);
		System.out.println("Voulez-vous ajouter le membre nouvellement cree\n1 : OUI\t2 :
NON");
		entree=sc.nextInt();
		vericationDeLaSaisie(1,2);
		if(entree==1){
			baseDeDonnees.creerMembre(membre);
		}
		entree=0;
	}

	private static void afficherMembres(List<Membre> liste){
		System.out.print("\nListe de tous les membres existants dans la base de
donnees\n");
		for(Membre membre : liste){
			System.out.print("\n\nID : " + membre.getId_Membre() + " \n");
			System.out.println("NOM : " + membre.getNomMembre() + " \n");
		}
	}

	private static void afficherTaches(List<Tache> liste){
		//System.out.print("\nListe de toutes les taches\n");
		for(Tache tache : liste){
			System.out.print("\n\nID : " + tache.getId_Tache() + " \n");
			System.out.print("NOM : " + tache.getNomTache() + " \n");
```

```
                        System.out.print("DESCRIPTION : " + tache.getDesription() + " \n");
                        System.out.print("STATUT : " + tache.getStatut() + " \n");
                        System.out.print("IDMEMBRE ASSIGNE A LA TACHE : " +
tache.getId_Membre() + " \n");
                }
        }

        private static List<Membre> rechercherMembres() throws SQLException{
                List<Membre> liste=baseDeDonnees.afficherMembres();
                return  liste;
        }

        private static List<Tache> rechercherTaches() throws SQLException{
                List<Tache> liste=baseDeDonnees.afficherTaches();
                return liste;
        }

        private static void rechercherEtAfficherTaches() throws SQLException{
                List<Tache> liste=rechercherTaches();
                afficherTaches(liste);
        }

        private static void rechercherEtAfficherMembres() throws SQLException{
                List<Membre> liste=rechercherMembres();
                afficherMembres(liste);
        }
}
```

2. Classe Membre

```
/**
 * Author: MEDOU Daniel Magloire
 * Unité d'enseignement: GENIE LOGICIEL AVANCE
```

```
 *  Ecole: IFI
 *  Niveau: Master 1
 *  Option: Systèmes Intelligents et Multimédia (SIM)
 *  Description: TP1, Conception et Réalisation d'une Application de Gestion de Tâches: "GESTACHE"
 */

package org.Tp1.entities;

import java.util.ArrayList;
import java.util.List;

public class Membre {
	//Déclaration des variables
	private int id_Membre;
	private String nomMembre;

	//private List<Tache> taches = new ArrayList<>();


	//Constructeur sans parametres
	public Membre() {
		super();
		// TODO Auto-generated constructor stub
	}

	//Constructeur avec parametres
	public Membre(int id_Membre, String nomMembre) {
		super();
		this.id_Membre = id_Membre;
		this.nomMembre = nomMembre;
	}
```

```
	//GETTERS AND SETTERS

	public int getId_Membre() {
		return id_Membre;
	}

	public void setId_Membre(int id_Membre) {
		this.id_Membre = id_Membre;
	}

	public String getNomMembre() {
		return nomMembre;
	}

	public void setNomMembre(String nomMembre) {
		this.nomMembre = nomMembre;
	}

	/*public List<Tache> getTaches() {
		return taches;
	}

	public void setTaches(List<Tache> taches) {
		this.taches = taches;
	}*/


}
```

3. Classe Tache

```
/**
```

```java
 *  Author: MEDOU Daniel Magloire
 *  Unité d'enseignement: GENIE LOGICIEL AVANCE
 *  Ecole: IFI
 *  Niveau: Master 1
 *  Option: Systèmes Intelligents et Multimédia (SIM)
 *  Description: TP1, Conception et Réalisation d'une Application de Gestion de Tâches: "GESTACHE"
 */

package org.Tp1.entities;



public class Tache {
	//Déclaration des variables
	private int id_Tache;
	private String nomTache;
	private String desription;
	private String statut;
	private int Id_Membre; //Cette variable nous permettra de connaitre le membre donc la tache a été assignée

	//private Membre membre = new Membre();


	//Constructeur sans parametres
	public Tache() {
		super();
		// TODO Auto-generated constructor stub
	}

	//Constructeur avec parametres


	//Constructeur avec seul parametre

	public Tache(int id_Tache) {
		super();
		this.id_Tache = id_Tache;
	}

	public Tache(int id_Tache, String nomTache, String desription, String statut, int id_Membre) {
		super();
		this.id_Tache = id_Tache;
		this.nomTache = nomTache;
		this.desription = desription;
		this.statut = statut;
		Id_Membre = id_Membre;
	}

	//GETTERS ET SETTERS

	public int getId_Tache() {
		return id_Tache;
	}

	public void setId_Tache(int id_Tache) {
		this.id_Tache = id_Tache;
```

```
	}

	public String getNomTache() {
		return nomTache;
	}

	public void setNomTache(String nomTache) {
		this.nomTache = nomTache;
	}

	public String getDesription() {
		return desription;
	}

	public void setDesription(String desription) {
		this.desription = desription;
	}

	public String getStatut() {
		return statut;
	}

	public void setStatut(String statut) {
		this.statut = statut;
	}

	public int getId_Membre() {
		return Id_Membre;
	}

	public void setId_Membre(int id_Membre) {
		Id_Membre = id_Membre;
	}

/*	public Membre getMembre() {
		return membre;
	}

	public void setMembre(Membre membre) {
		this.membre = membre;
	}*/

}
```

4. Classe DataBaseConnexion

```
/**
 * Author: MEDOU Daniel Magloire
 * Unité d'enseignement: GENIE LOGICIEL AVANCE
 * Ecole: IFI
 * Niveau: Master 1
 * Option: Systèmes Intelligents et Multimédia (SIM)
```

```
 *  Description: TP1, Conception et Réalisation d'une Application de Gestion de Tâches: "GESTACHE"
 */

package org.Tp1.BD;

import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import org.Tp1.entities.Membre;
import org.Tp1.entities.Tache;

public class DataBaseConnexion {

	//Variables representant les ressources manipulees
		Connection connection = null;
		Statement statement = null;
		ResultSet result=null;

		//Constructeur de la classe permettant d'initialiser, d'instancier et de creer une connexion a la base de donnees
		public DataBaseConnexion() {
			boolean fichierExiste=false;
			File file = new File("GESTACHE.DB");

			//Test de l'existence de la base
```

```
                    if(file.exists()){
                      fichierExiste=true;
                    }

                    try {
                            Class.forName("org.sqlite.JDBC");
                            connection = DriverManager.getConnection("jdbc:sqlite:" +
"GESTACHE.DB");
                            connection.setAutoCommit(true);

                            statement = connection.createStatement();

                            if(!fichierExiste){
                                    creerTablesDansLaBaseDeDonnees();
                            }
                            //System.out.println("\nBase de donnees cree avec success\n");

                    } catch (Exception e) {
                            System.err.println(e.getClass().getName() + ": " + e.getMessage());
                            System.exit(0);
                    }
            }


            //Methode de creation des tables dans la base de donnees
            public void creerTablesDansLaBaseDeDonnees(){

                    String sql;

                    try {
                            sql = "CREATE TABLE TACHE " +
```

```
                                        "(ID INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL ," +
                                        " NOM TEXT NOT NULL," +
                                        "DESCRIPTION TEXT NOT NULL,"+
                                        "STATUS TEXT NOT NULL," +
                                        "ID_MEMBRE INTEGER NULL)";

                        statement.executeUpdate(sql);

                        sql="CREATE TABLE MEMBRE " +
                                        "(ID INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL ," +
                                        " NOM TEXT NOT NULL)";

                        statement.executeUpdate(sql);

                } catch ( Exception e ) {
                        System.err.println( e.getClass().getName() + ": " + e.getMessage() );
                        System.exit(0);
                }
                System.out.println("BRAVO LES TABLES TACHE et MEMBRE CREES AVEC
SUCCES\n");
        }

        //Methode permettant de creer une tache
        public void creerTache(Tache tache) {
                int id_Membre=tache.getId_Membre();
                String status="";

                if( id_Membre==0 || !(id_Membre > 0)){
                        status="nouveau";
                }else{
                        status="en-progres";
```

```
                }


                try {
                        statement.executeUpdate("INSERT INTO TACHE (NOM, DESCRIPTION,
STATUS, ID_MEMBRE) " +
                                        "VALUES ('" + tache.getNomTache()+"','"+
tache.getDesription() +"','" + status + "'," + id_Membre + ");");
                        System.out.println("\nTache creee et enregistree avec succes\n");
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        //Methode permettant de creer un membre
        public void creerMembre(Membre membre) {
                try {
                        statement.executeUpdate("INSERT INTO MEMBRE (NOM) " +
                                        "VALUES ('" + membre.getNomMembre()+"');");
                        System.out.println("\nMembre cree et enregistre avec succes\n");
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }

        //Methode permettant d'afficher les taches assignees a un membre
        public List<Tache> afficherTachesMembre(int identifiant) throws SQLException {
                result = statement.executeQuery("SELECT * FROM TACHE WHERE
ID_MEMBRE = " + identifiant + ";");
                List<Tache> listeTachesmembre = new ArrayList<Tache>();

                while (result.next()) {
                        int id = result.getInt("ID");
```

```
                String name = result.getString("NOM");
                String description = result.getString("DESCRIPTION");
                String status = result.getString("STATUS");
                int idMembre = result.getInt("ID_MEMBRE");
                listeTachesmembre.add(new Tache(id, name, description, status,
idMembre));
            }
            return listeTachesmembre;
        }

        //Methode permettant d'afficher toutes les taches selon leurs status et les infos du
membre associe
        public List<Object[]> afficherStatusTaches(int status_a_chercher) throws
SQLException {

            List<Object[]> listeTachesStatus = new ArrayList<Object[]>();
            //List temporaire=new ArrayList();
            Object[] temporaire =new Object[2];
            String stringStatus;

            if (status_a_chercher == 1) {
                stringStatus = "nouveau";
            } else if (status_a_chercher == 2) {
                stringStatus = "en-progres";
            } else {
                stringStatus = "termine";
            }
            result = statement.executeQuery("SELECT T.ID AS tacheID, T.NOM AS
tacheNOM, T.DESCRIPTION AS tacheDESCRIPTION,"
                    + "T.STATUS AS tacheSTATUS, M.ID AS membreID, M.NOM AS
membreNOM  FROM TACHE AS T "
                    + "LEFT JOIN MEMBRE AS M  ON T.ID_MEMBRE=M.ID WHERE
T.STATUS = '"+stringStatus+"';");
```

```
		while (result.next()) {
			//Creation objet tache
			int id = result.getInt("tacheID");
			String name = result.getString("tacheNOM");
			String description = result.getString("tacheDESCRIPTION");
			String status = result.getString("tacheSTATUS");

			//Creation objet membre
			int idMembre = result.getInt("membreID");
			String nomMembre=result.getString("membreNOM");

			temporaire[0]=new Tache(id, name, description, status, idMembre);
			temporaire[1]=new Membre(idMembre,nomMembre);
			listeTachesStatus.add(temporaire);
		}
		return listeTachesStatus;
	}

	// Methode permettant de recuperer et d'afficher tous les membres
	public List<Membre> afficherMembres() throws SQLException {
		result = statement.executeQuery("SELECT * FROM MEMBRE;");
		List<Membre> listeMembres = new ArrayList<Membre>();
		while (result.next()) {
			int id = result.getInt("ID");
			String name = result.getString("NOM");
			listeMembres.add(new Membre(id, name));
		}
		return listeMembres;
	}
```

```
		// Methode permettant de recuperer et d'afficher toutes les taches
		public List<Tache> afficherTaches() throws SQLException {
			result = statement.executeQuery("SELECT * FROM TACHE;");
			List<Tache> listeTaches = new ArrayList<Tache>();

			while (result.next()) {
				int id = result.getInt("ID");
				String name = result.getString("NOM");
				String description = result.getString("DESCRIPTION");
				String status = result.getString("STATUS");
				int idMembre = result.getInt("ID_MEMBRE");
				listeTaches.add(new Tache(id, name, description, status,
idMembre));
			}
			return listeTaches;
		}

		//Modifications
		public void modifierTache(Tache tache) {
			try {
				statement.executeUpdate("UPDATE TACHE SET
NOM='"+tache.getNomTache()+"',DESCRIPTION='"+tache.getDesription()+

	"',STATUS='"+tache.getStatut()+"',ID_MEMBRE='"+tache.getId_Membre()+
						"' WHERE ID='"+tache.getId_Tache()+"';");

				System.out.println("\nMise a jour de tache a ete bien faite et
enregistree avec succes\n");
			} catch (SQLException e) {
				e.printStackTrace();
			}
		}
```

```
	public void modifierMembre(Membre membre) {

		try {
			statement.executeUpdate("UPDATE MEMBRE SET NOM='"+membre.getNomMembre()+
					"' WHERE ID='"+membre.getId_Membre()+"';");

			System.out.println("\nMise a jour du membre a ete bien faite et enregistree avec succes\n");
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	//Methode d'assignation de tache
	public void assignerTacheAUnMembre(int idTache, int idMembre){
		try {
			statement.executeUpdate("UPDATE TACHE SET ID_MEMBRE='"+idMembre
					+"'WHERE ID='"+idTache+"';");

			System.out.println("\nTache assignee et enregistree avec succes\n");
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	//Suppressions
	public void supprimerTache(int idTache) {
		String sql = "DELETE FROM TACHE WHERE ID="+ idTache + ";";
		try {
```

```
                    statement.executeUpdate(sql);
                    System.out.println("\nTache supprimee avec succes\n");
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }

            public void supprimerMembre(int idMembre) {
                String sql = "DELETE FROM MEMBRE WHERE ID="+ idMembre + ";";
                try {
                    statement.executeUpdate(sql);
                    sql="UPDATE TACHE SET ID_MEMBRE = NULL WHERE
ID_MEMBRE=0;";
                    statement.executeUpdate(sql);
                    System.out.println("\nMembre supprime avec succes\n");
                } catch (SQLException e) {
                    e.printStackTrace();
                }

            }

            // Methode permettant de fermer et liberer les ressources
            public void fermerRessourceBaseDeDonnees() {
                try {
                    statement.close();
                    connection.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                    System.out.println("Erreur de fermeture des ressources de la base de
donnees\n\n");
                }
```

```
                }
        }
```