NOM : BARRY
PRENOMS : Mamadou Dian
IFI (Institut Francophone Internationale) /Année Universitaire 2017

RAPPORT DU TRAVAIL PRATIQUE(TP1)- GENIE LOGICIEL

Contexte :

Le génie logiciel est un domaine des sciences de l'ingénieur dont l'objet d'étude est la conception, la fabrication, et la maintenance des systèmes informatiques complexes. C'est pourquoi il est devenu ces dernières années un domaine de recherche très convoité et porteur d'intérêt. Le génie logiciel est apparu dans les années 1970 sous la coordination de l'OTAN.

Au début il visait à répondre à la crise du logiciel dans les années 1968. À cette époque les logiciels n'étaient pas fiables et il était difficile pour les développeurs de fournir dans les délais une application respectant les spécifications du cahier de charge. Même quand le logiciel était bien construit, la difficulté de la maintenance aussi se présentait. Et ces problèmes de systèmes non qualifiés ont causés d'énormes dégâts matériels et Humain dans le monde dont voici quelques-uns de ses conséquences à titre d'exemple :

- **Le projet TAURUS**. Ce fut un projet d'informatisation de la bourse de Londres. Il a été abandonné définitivement après quatre années de labeur et a engendré environ 100 millions de livres de perte.
- Environ 280 civils ont été abattus lors de la guerre du Golfe. La cause fut que les abatteurs n'ont pas fait la différence entre **un Airbus 655 iranien** transportant ces civils et un avion militaire.
- À cause du remplacement d'une virgule par un point, la mission VENUS a été un échec retentissant.

Le Génie Logiciel a toutefois amélioré cette situation en fournissant des techniques et des outils facilitant le développement de logiciels sécurisés avec les qualités requises. Il est alors important aujourd'hui de se référer à ces outils standardisés pour la création des applications de haut niveau.

Dans ce projet, notre travail consiste à créer un Gestionnaire de tache. Nous appréhendons spécifiquement les concepts de la modélisation avec UML et programmation orientée-objet avec Java. Egalement nous aborderons la notion de test (test unitaire) avec le Framework JUnit.

Table des matières

Spécification de l'application :

Comme mentionner plus haut notre application est une application de gestionnaire de taches pour une équipe de travail. L'application développée permettra à un administrateur de gérer non seulement les membres de son équipe mais aussi les taches que l'équipe doit exécuter. Cet outil lui offre plusieurs avantages dont spécifiquement la gestion automatique de toutes ses activités.

Elle est composée de deux principaux attributs qui sont « Tache » et «Membre ». Dont les entêtées pour chaque attribut sont :

- Une tâche:
  - ID
  - Nom
  - Une description
  - Statut : nouvelle, en-cours, terminée
- Un membre:
  - ID
  - Nom

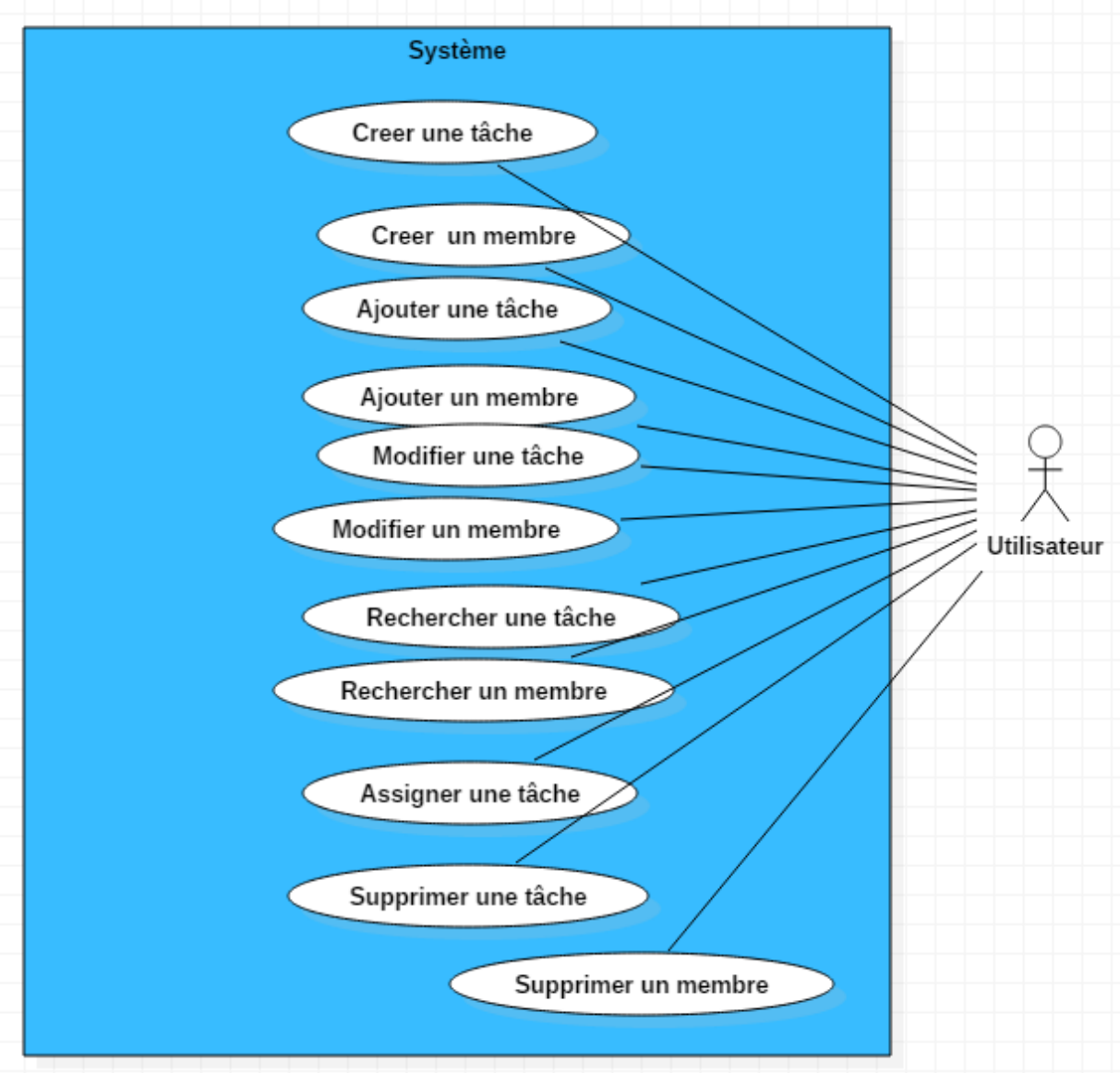
Les spécifications du programme est qu'il doit fournir à l'utilisateur les fonctions suivantes :

- Créer, modifier, supprimer, ajouter une tâche ;
- Créer, modifier, supprimer, ajouter un membre ;
- Assigner une tâche à un membre ;
- Chercher et afficher tous les tâches assignées à un membre (par son ID) ;
- Chercher et afficher tous les tâches en fonction de leur statut (avec le nom du assigné).

Les exigences Fonctionnelles et non Fonctionnelles du Programme :

C_1. Exigences Fonctionnelles : Les spécifications fonctionnelles définissent les actions fondamentales pour accepter les entrées, effectuer les traitements et générer les sorties. Inclure: - Le modèle objet détaillé: modèles structural, évolutif, fonctionnel, accompagnés d'un répertoire des classes d'objets.

Dans le cadre de notre application, nous avons représenté ces exigences à partir d'un diagramme de cas d'utilisation d'UML comme le montre la capture suivante :



C_2. Les exigences non Fonctionnelles (FF):

Une exigence non-fonctionnelle est une condition qui spécifie les critères qui peuvent être utilisés pour juger du fonctionnement d'un système, plutôt que des comportements de celui-ci. Ces exigences peuvent être:

- Fiabilité ;
- Disponibilité ;
- Maintenabilité ;
- IHM.
- Utilisabilité ;
- Sécurité ;
- Portabilité ;
- Traçabilité.

	Qualité	Description
Exigences Non Fonctionnelles (ENF)	Sécurité	Concerne l'accès non autorisé aux fonctions du logiciel
	La Fiabilité	La capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation
	La facilité d'utilisation (utilisabilité)	Ceci porte sur l'interaction entre l'homme et le logiciel
	Pertinence	Effectuer la bonne opération lorsque demandé
	La performance.	Le rapport entre la quantité de ressources utilisées (moyens matériels, temps, personnel) et la quantité de résultats délivrés
	La maintenabilité	L'effort nécessaire en vue de corriger ou de transformer le logiciel en y ajouté de nouvelles fonctions.
	La portabilité.	C'est l'aptitude d'un logiciel de fonctionner dans un environnement matériel ou logiciel différent de son environnement initial.
	L'IHM	C'est la facilitation de l'utilisation de l'interface, la convivialité de l'interface.
	La Traçabilité	

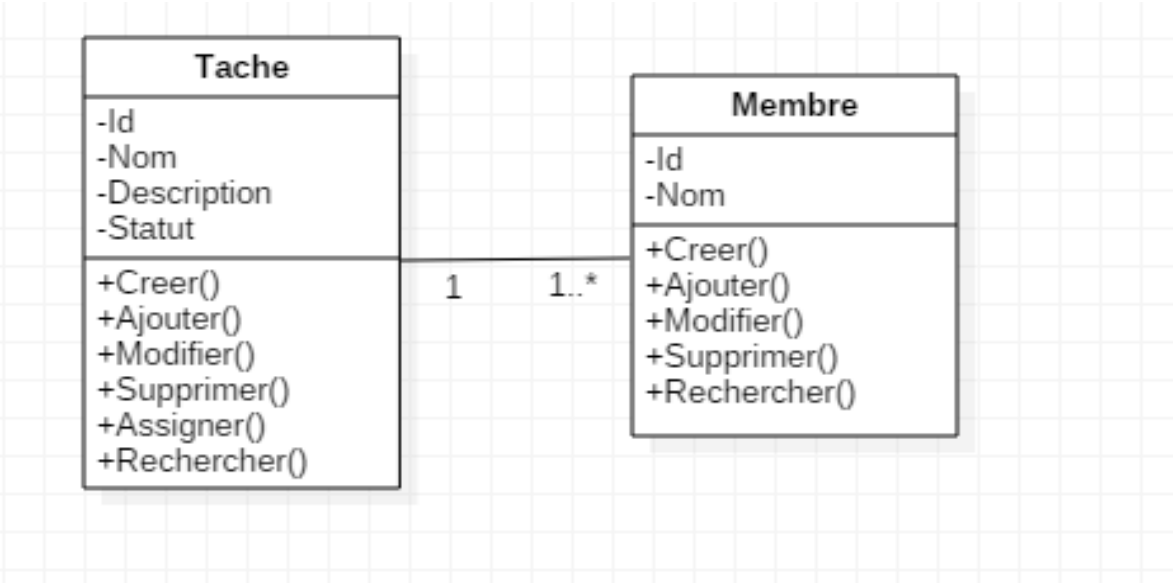
Au vue de ces critères nous concluons que dans le respect des exigences non-fonctionnelles notre application doit présenter absolument tous les attributs de qualité. Il doit être un logiciel facile à maintenir, mais également très performant, il doit être disponible en tout temps. Toutefois, une réelle disponibilité en tout temps imposera des restrictions sur la maintenance. Également, un système facilement maintenable engendra des limitations de performance.

Conception de l'application:

D_1. Diagramme de Classe :

Le diagramme de classe représente les classes constituant le système et les associations entre elles. Elle exprime de manière générale la structure statique du système, en termes de classe et de relations entre ces classes de notre application.

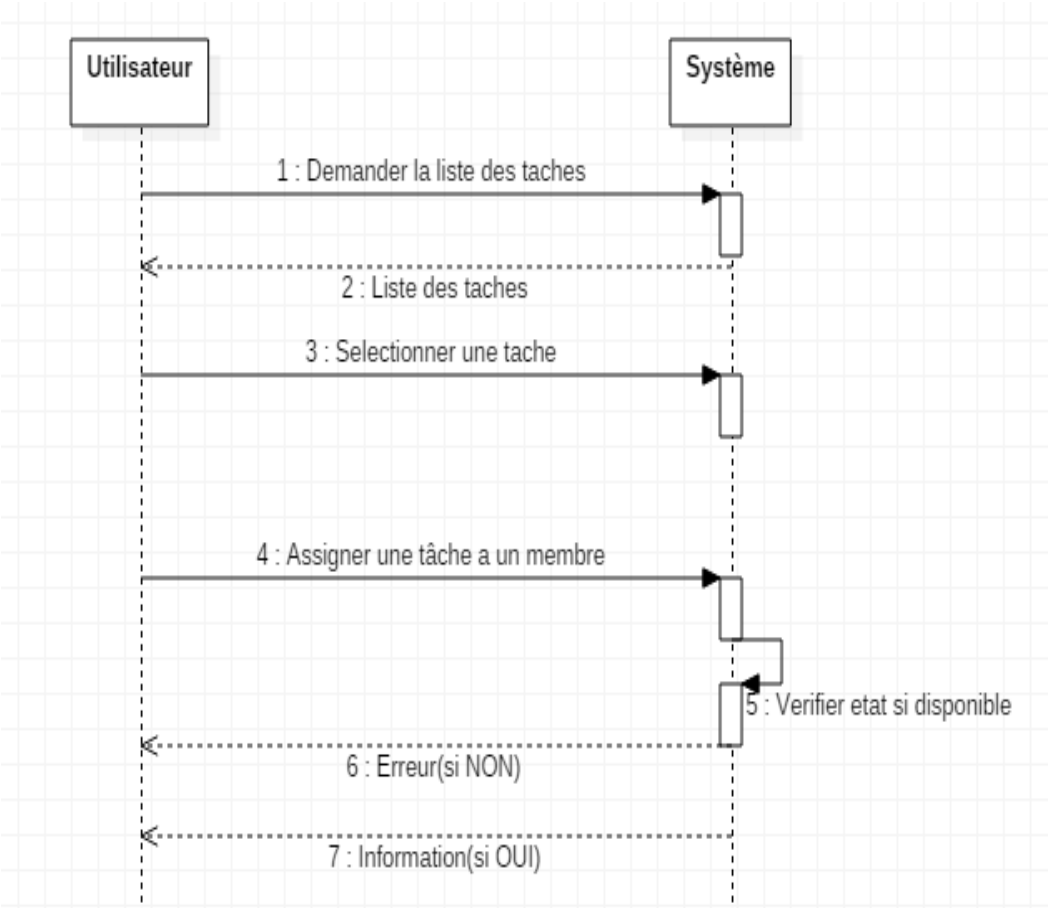
Voici une illustration de notre diagramme de classe ci-dessous :



D_2. Diagramme de Séquence :

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique. Dans notre cas nous avons choisie de présenter le diagramme pour l'option assignation d'une tache.

1- Diagramme Assignation d'une tache



Scénario assignation d'une tâche

IMPLÉMENTATION ET TEST

E_1. Environnement Matériel :

Au niveau matériel notre application est développée sur « **EliteBook 8470p** » avec un processeur Intel(R) Core (TM) i5-3320M CPU @ 2,60 GHz. Une mémoire vive de 4Go, un disque dur 700 Go, un écran 14 pouces.

E_2. Environnement Logiciel :

Au niveau logiciel le développement de l'application se fait exclusivement sur des plateformes libres. L'implémentation se fait sous une machine Linux (Ubuntu), version 16.04 LTS utilisant **NetBeans (version 8.1)** comme IDE.

E_3. Langage de programmation

L'une de nos contraintes ici c'est de programmer de façon orientée objet. Ainsi, le langage utilisé pour l'implémentation est le langage java, comme suggérer par le responsable du cours qui de bien structurée de notre application.

Méthode de test

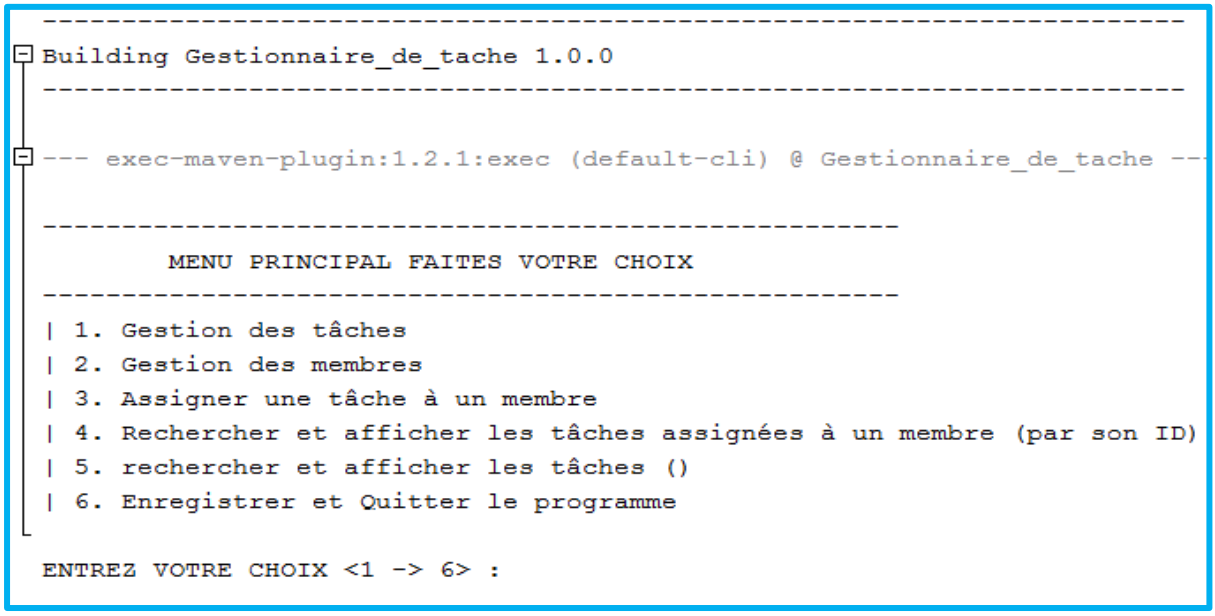
F_1. Les Tests d'acceptation: Ce sont des tests qui permettent de vérifier si la totalité des fonctionnalités du programme marchent convenablement. Ils permettent de vérifier que l'application répond aux attentes de l'utilisateur c'est-à-dire conforme aux besoins/cahier des charges. On vérifie l'application dans son ensemble, généralement avec des scénarios réalistes d'utilisation. Pour ce faire nous avons choisi de faire ces tests selon les spécifications de notre application, ainsi nous avons considéré les actions suivantes afin de vérifier la réponse satisfiable dudit programme.

Les taches sont choisies en fonction des spécifications du système et ce sont : (création d'une tache, modification d'une tache, assignation d'une tache a un membre).

Les prints écran suivants montrent le résultat des réponses du système pour les taches précitées.

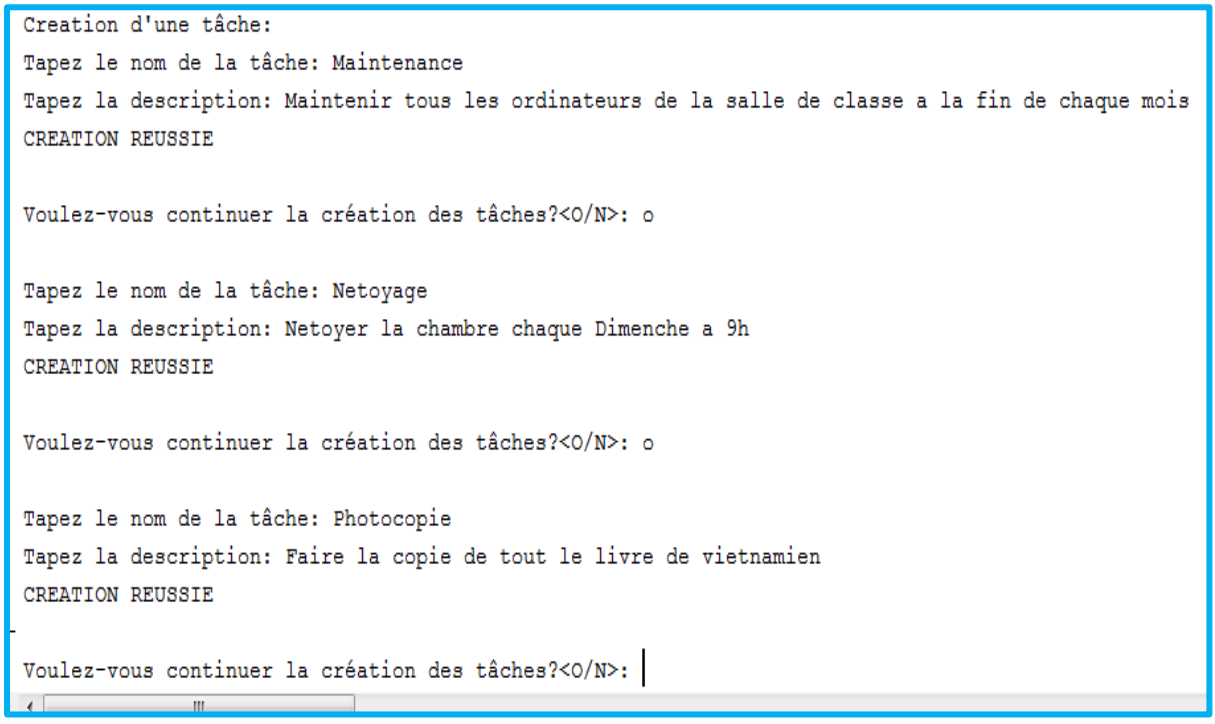
- **Lancement du Programme** :

A son lancement, notre programme présente l'interface suivant a l'utilisateur pour lui proposer de faire son choix parmi ceux spécifier dans l'application.



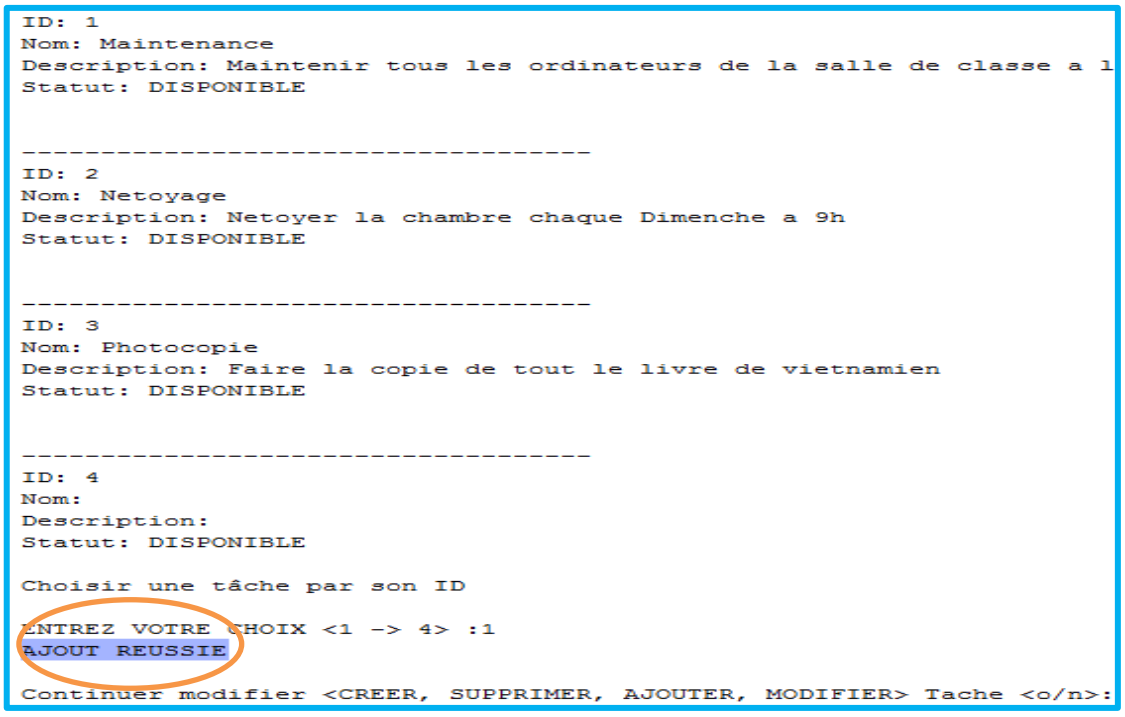
NB : Il n'y a aucun prérequis pour exécuter notre programme, lorsque vous avez déjà NetBeans quel que soit le système d'exploitation utilisé, il vous suffit d'importer le projet à partir dans NetBeans IDE et compiler la méthode « Main » du programme.

- **Créer une Tache** : Cette fonction permet de créer une tache dans le programme temporairement. Si la tâche n'est pas ajouter avant la recompilation du programme, cette tâche se supprime automatiquement.

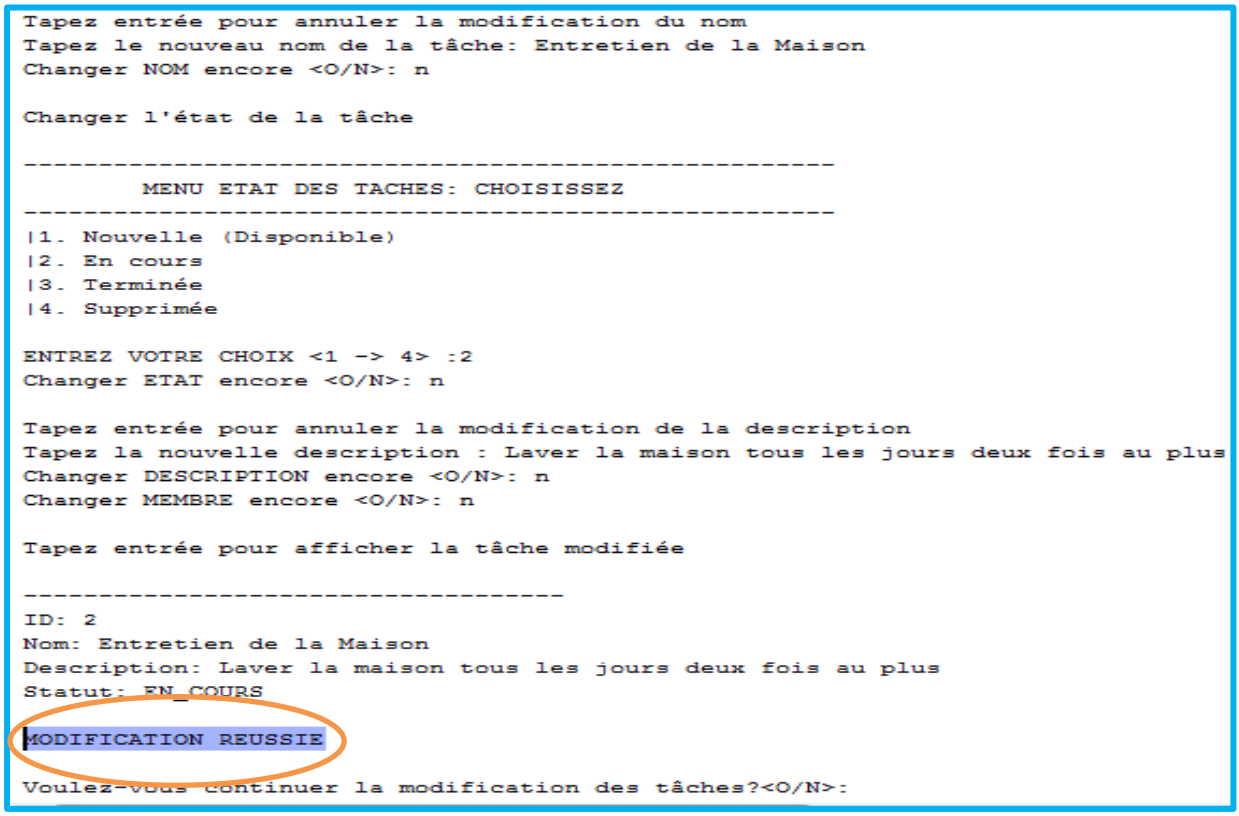


- **Ajouter une tache :**

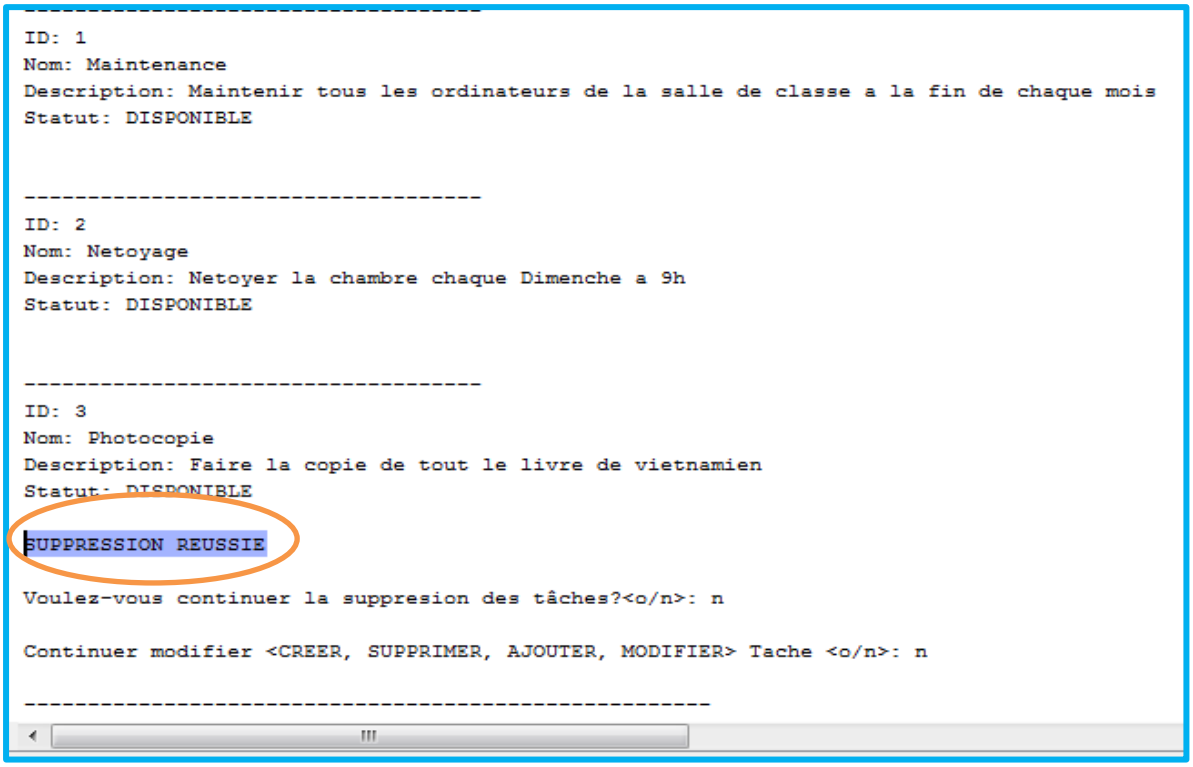
Après la création de la tâche, sous demande de l'utilisateur le programme l'affiche la liste des tache créée, cette fonction ajout permet d'enregistrer les taches de façon définitive dans le fichier que le programme génère automatiquement.



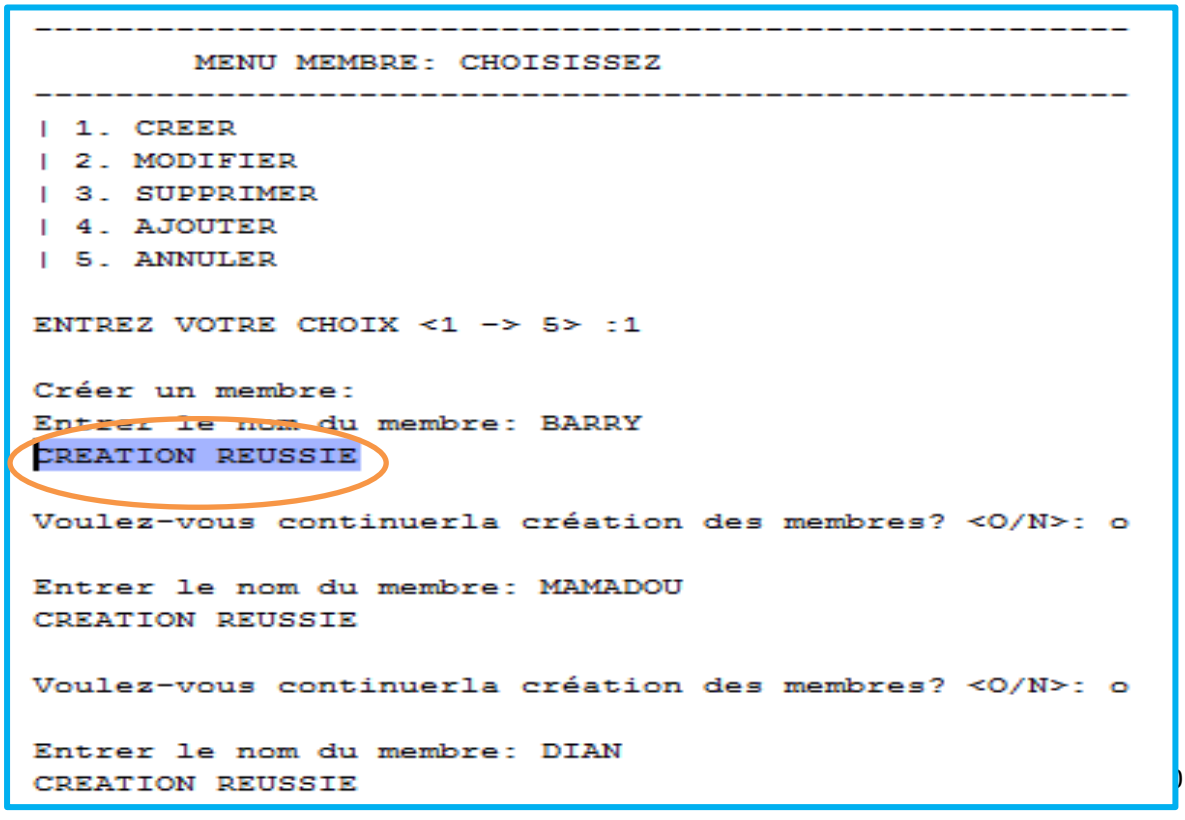
- **Modifier une tache :** Permet d'apporter des corrections sur les informations d'une tache déjà enregistrée.



- **Supprimer une tache :** Efface une tache définitivement de la liste des taches enregistrées.

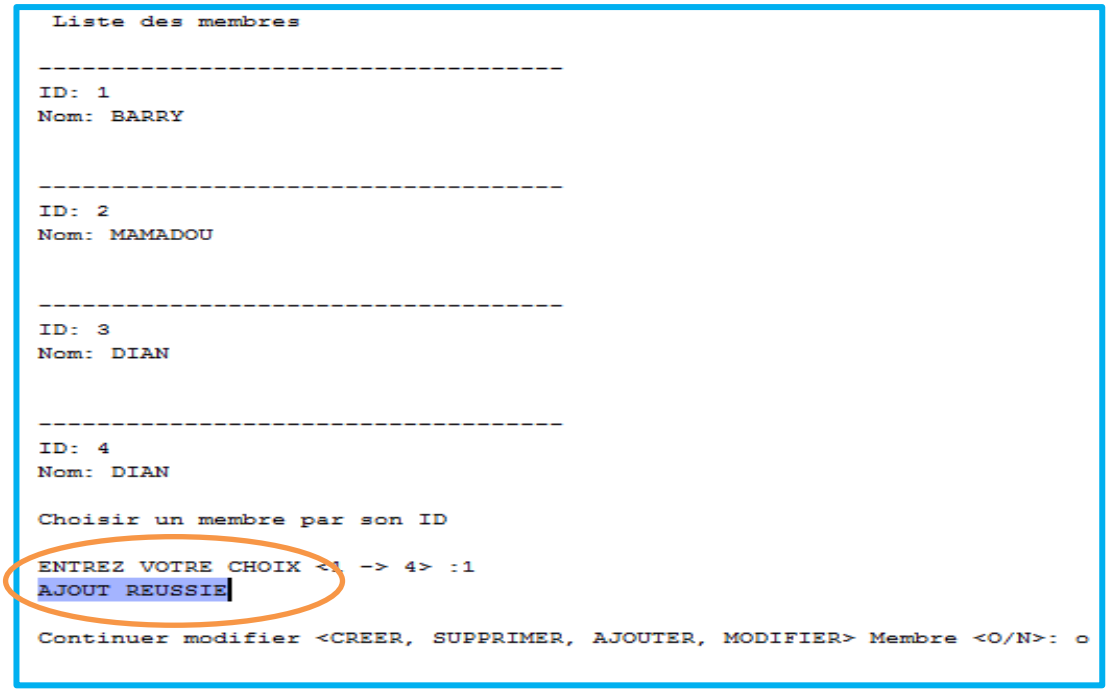


- **Création d'un Membre :** Ici nous précisons tout simplement que le fonctionnement reste le même que celui de la création d'un membre.

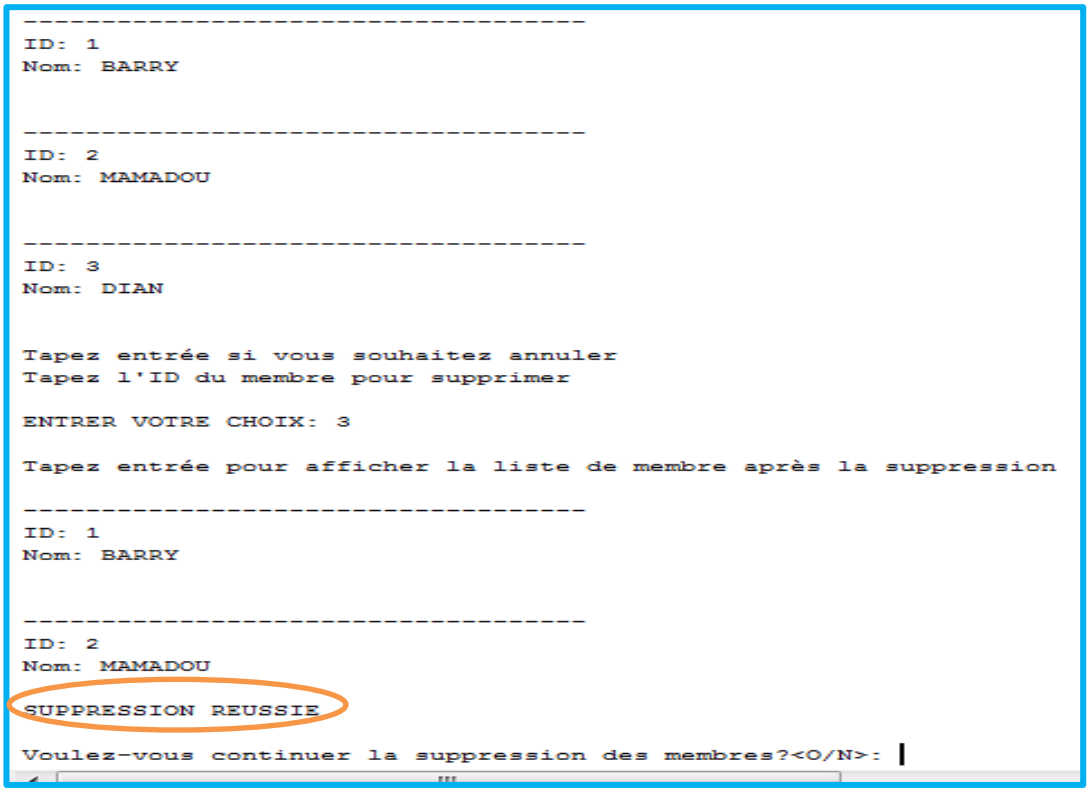


- **Ajouter un Membre :**

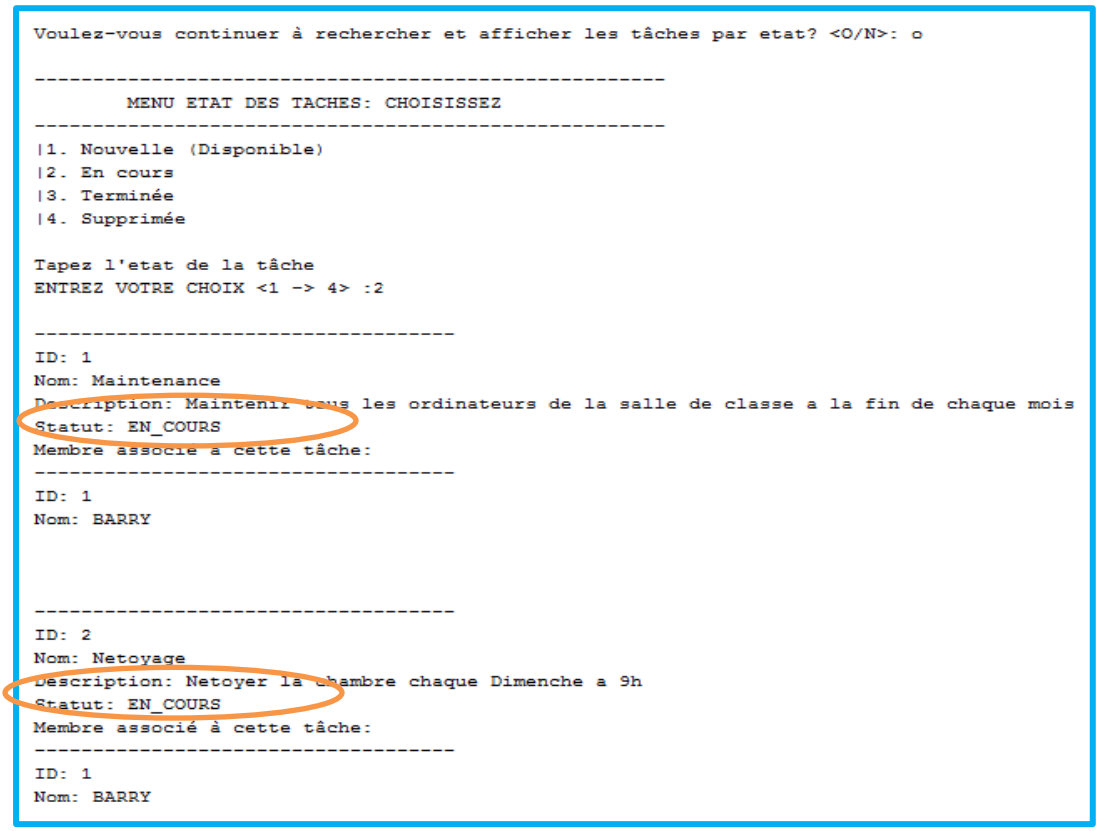
Après la création du membre, sous demande de l'utilisateur le programme l'affiche la liste des membres créées, cette fonction ajout permet d'enregistrer les membres de façon définitive dans le fichier que le programme génère automatiquement.



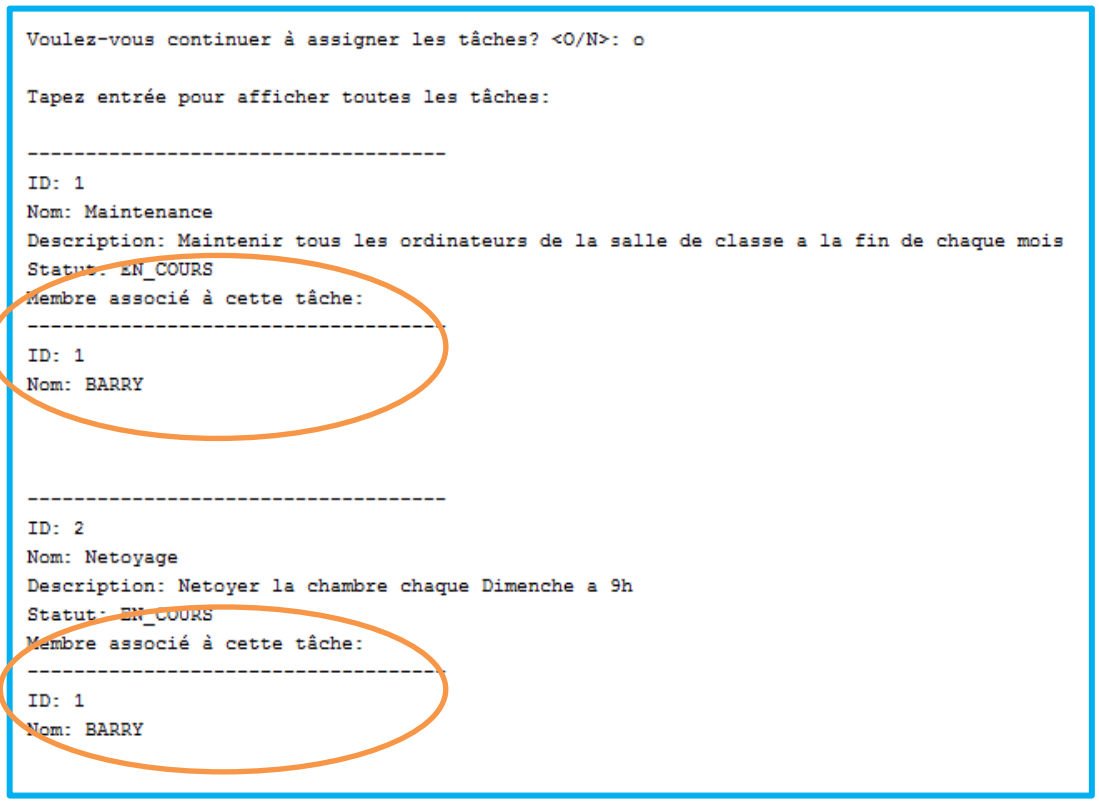
- **Suppression d'un membre :** Supprime définitivement le membre ajouté de la liste.



- **Rechercher et afficher les Tache à partir de leur Etat :** Permet de retrouver les taches en fonction de leur état (nouveau : disponible, en cours, terminer).



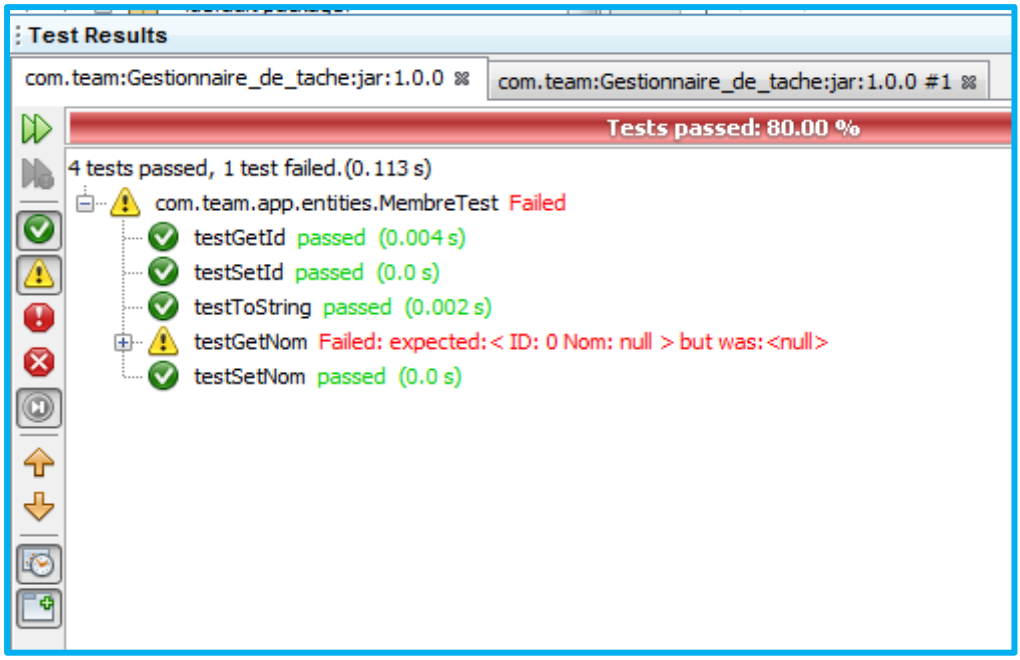
- **Affichage des tâches assignées à un Membre :** Cette commande montre toutes les tâches assignées à un membre donné.

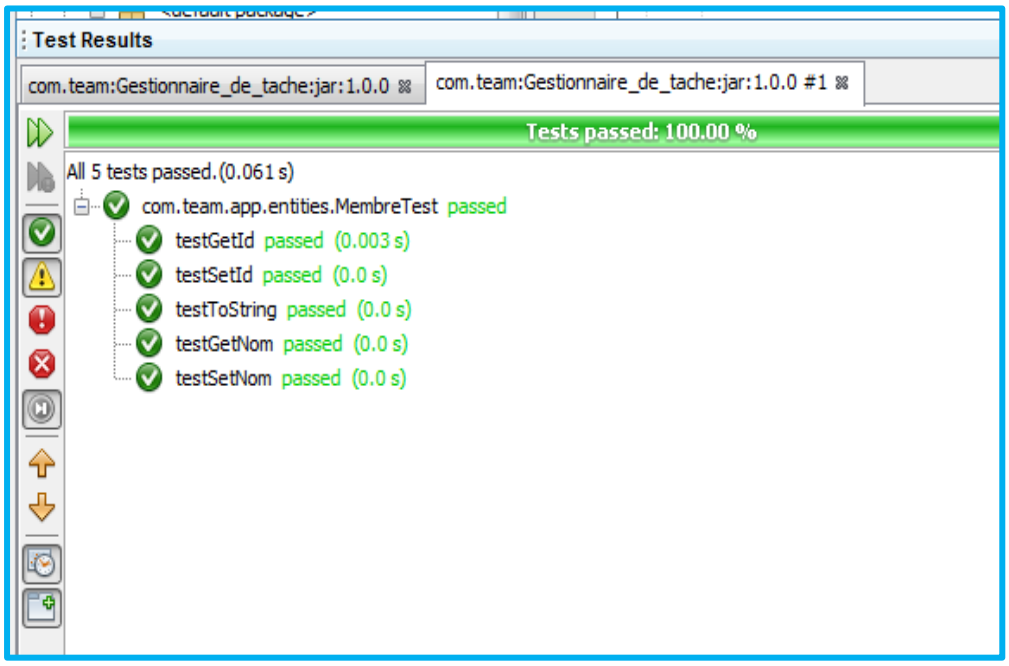


E_2. Test Unitaire :

La méthode de test imposée dans ce travail est les tests unitaires à partir de (**JUnit**) qui est un Framework open source pour le développement et l'exécution de tests unitaire manuellement ou automatiquement pour Java pour s'assurer du bon fonctionnement des méthodes utilisés.

Pour ce faire nous nous sommes contentées de tester une seule classe du programme « **la classe Membre** ». L'objectif de ce test était de découvrir les erreurs (syntaxiques, logiques …) contenus dans cette classe. Les résultats obtenus affichés sur les captures plus-bas nous montrent l'efficacité du test utilisé et a permis en un premier temps de détecter l'erreur et en second temps de corriger les erreurs détectées.





ANNEXE

F- CODE SOURCE

Classe_Main :

```
package com.team.app.entities;
import java.util.Scanner;
public class Menu {
public static final int TACHE_STATUT_NEW = 1;
public static final int STATUS_TACHE_EN_COURS = 2;
 public static final int TACHE_STATUT_TERMINER = 3;
 public static final int TACHE_STATUT_QUITTER = 4;

 public static final int MODIFIER_CREATION = 1;
 public static final int MODIFIER_EDITION = 2;
 public static final int MODIFIER_SUPPRESSION = 3;
 public static final int MODIFIER_AJOUT = 4;
 public static final int MODIFIER_QUITTER = 5;
 public static final int MODIFIER_TACHE = 1;
 public static final int MODIFIER_MEMBRE = 2;
 public static final int ASSIGN_TACHE_AU_MEMBRE = 3;
 public static final int RECHERCH_VOIR_TACHE_PAR_ID_DU_MEMBRE = 4;
 public static final int RECHERCH_VOIR_TACHE_PAR_STATUS = 5;
 public static final int QUITTER_LE_PROGRAMME = 6;
public static final int ANNULER_ACTION = -1;

private static Scanner sc = new Scanner(System.in);
public static void showUserMenu() {
      String saperate = "\n----------------------------------------------------";
      StringBuilder description = new StringBuilder();
      description.append(saperate);
      description.append("\n\tMENU PRINCIPAL FAITES VOTRE CHOIX");
      description.append(saperate);
```

```
        description.append("\n| 1. Gestion des tâches");
        description.append("\n| 2. Gestion des membres");
        description.append("\n| 3. Assigner une tâche à un membre");
      description.append("\n| 4. Rechercher et afficher les tâches assignées à un membre (par son ID)");
        description.append("\n| 5. rechercher et afficher les tâches ");
        description.append("\n| 6. Enregistrer et Quitter le programme ");
        System.out.println(description);
    }
public static void showTaskStatusMenu() {
        String saperate = "\n-------------------------------------------------------";
        StringBuilder description = new StringBuilder();
        description.append(saperate);
        description.append("\n\tMENU ETAT DES TACHES: CHOISISSEZ");
        description.append(saperate);
        description.append("\n|1. Nouvelle (Disponible)");
        description.append("\n|2. En cours");
        description.append("\n|3. Terminée");
        description.append("\n|4. Supprimée");
        System.out.println(description);
    }
public static void showMenuModifyTask() {
        String saperate = "\n-------------------------------------------------------";
        StringBuilder description = new StringBuilder();
        description.append(saperate);
        description.append("\n\tMENU TACHE: CHOISISSEZ");
        description.append(saperate);
        description.append("\n| 1. CREER");
        description.append("\n| 2. MODIFIER");
        description.append("\n| 3. SUPPRIMER");
        description.append("\n| 4. AJOUTER");
        description.append("\n| 5. ANNULER");
        System.out.println(description);
  }
```

```
 public static void showMenuModifyMember() {
        String saperate = "\n--------------------------------------------------------";
        StringBuilder description = new StringBuilder();
        description.append(saperate);
        description.append("\n\tMENU MEMBRE: CHOISISSEZ");
        description.append(saperate);
        description.append("\n| 1. CREER");
        description.append("\n| 2. MODIFIER");
        description.append("\n| 3. SUPPRIMER");
        description.append("\n| 4. AJOUTER");
        description.append("\n| 5. ANNULER");
        System.out.println(description);
    }
public static int chooseUserMenu() {
        return chooseMenu(1, 6);
 }
public static int chooseTaskStatusMenu() {
        return chooseMenu(1, 4);
    }
   public static int chooseMenuModify() {
        return chooseMenu(1, 5);
    }
   public static int chooseMenu(int start, int end) {
        int choice = end;
        boolean isChooseAgain;
        do {
            System.out.print("\nENTREZ VOTRE CHOIX <" + start + " -> " + end + "> :");
            isChooseAgain = false;
            try {
                choice = Integer.parseInt(sc.nextLine());
                if (choice < start || choice > end) {
                    isChooseAgain = true;
                }
```

```
        } catch (Exception e) {
            isChooseAgain = true;
        }
        if (isChooseAgain) {
            System.out.println("\nMAUVAIS CHOIX, S'il vous plait faites votre choix encore!");
        }
    } while (isChooseAgain);
    return choice;
}
public static int chooseCancellableMenu(int start, int end) {
    int choice = end;
    boolean isChooseAgain;
    do {
        System.out.print("\nENTRER VOTRE CHOIX: ");
        isChooseAgain = false;
        try {
            String query = sc.nextLine();
            if (query.equals("")) {
                choice = ANNULER_ACTION;
                break;
            }
            choice = Integer.parseInt(query);
            if (choice < start || choice > end) {
                isChooseAgain = true;
            }
        } catch (Exception e) {
            isChooseAgain = true;
        }
        if (isChooseAgain) {
            System.out.println("\nMAUVAIS CHOIX, S'il vous plait faites votre choix encore!");
        }
    } while (isChooseAgain);
    return choice;
```

```
        }
}
```

Classe_Membre :

```
package com.team.app.entities;

import java.io.Serializable;

public class Membre implements Serializable {

    private int id;

    private String nom;

    public Membre() {

    }

    public Membre(int id, String nom) {

        this.id = id;

        this.nom = nom;

    }

    public int getId() {

        return id;

    }

    public void setId(int id) {

        this.id = id;

    }

    public String getNom() {

        return nom;

    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    @Override

    public String toString() {

        String resultat = "\nID: " + id + "\nNom: " + nom + "\n";

        return resultat;
 }
}
```

- **Classe_Tache :**

```
package com.team.app.entities;

import com.team.app.enumeration.Etats;

import java.io.Serializable;

public class Tache implements Serializable {

   private int id;

   private String nom;

   private String description;

   private Etats statut;

   private Membre membre;

   public Tache() {

   }
   public Tache(int id, String nom, String description) {

      this(id, nom, description, Etats.DISPONIBLE);

   }
   public Tache(int id, String name, String description, Etats status) {

      this.id = id;

      this.nom = name;

      this.description = description;

      this.statut = status;

   }
   public int getId() {

      return id;    }

   public void setId(int id) {

      this.id = id;    }

   public String getName() {

      return nom;

   }
   public void setName(String name) {

      this.nom = name;

   }
```

```
public String getDescription() {
    return description;
}
public void setDescription(String description) {
    this.description = description;
}
public Etats getStatus() {
    return statut;
}
public void setStatus(Etats status) {
    this.statut = status;
}
public Membre getMember() {
    return membre;
}
public void setMember(Membre member) {
    this.membre = member;
}
@Override
public String toString() {
    String saperate = "\n-----------------------------------";
    if (membre != null) {
        return saperate + "\nID: " + id
                + "\nNom: " + nom
                + "\nDescription: " + description
                + "\nStatut: " + statut
                + "\nMembre associé à cette tâche: " + membre.toString() + "\n";       }
    return saperate + "\nID: " + id
            + "\nNom: " + nom
            + "\nDescription: " + description
                    + "\nStatut: " + statut + "\n";
            }
        }
```

ClasseGererTache :

```
package com.team.app.manager;

import com.team.app.entities.Membre;

import com.team.app.entities.Menu;

import com.team.app.entities.Tache;

import com.team.app.enumeration.Resultats;

import com.team.app.enumeration.Etats;

import com.team.app.enumeration.AtributTache;

import com.team.app.fileio.LectureEcriture;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import javafx.concurrent.Task;

public class GererTaches implements GererOperations {

  private static int id = 1;
    private Scanner sc = new Scanner(System.in);
   private final List<Tache> mListTask = LectureEcriture.getListTaskFromFile();
   private final List<Tache> mListCreatedTask = new ArrayList<>();
   private final static GererTaches TaskManager = new GererTaches();

   public static GererTaches getInstance() {
      return TaskManager;
   }
   private GererTaches() {  }
   public boolean creer() {

      Tache task = makeNewTask();

      mListCreatedTask.add(task);

      return true;
   }
   private Tache makeNewTask() {
      System.out.print("\nTapez le nom de la tâche: ");
      String nom = sc.nextLine();
      System.out.print("Tapez la description: ");
      String des = sc.nextLine();
      return new Tache(id++, nom, des);

   }
   public boolean modifier(int id) {
      boolean result = true;
```

```
    AtributTache[] taskAttributes = new AtributTache[]{AtributTache.NOM, AtributTache.ETAT,
AtributTache.DESCRIPTION, AtributTache.MEMBRE};

    int numMis = taskAttributes.length;
    Tache task = findTaskById(id);
    if (task != null) {
      for (int i = 0; i < numMis; i++) {
        boolean isInEdit;
        do {
          Resultats captureResult = editAttribute(task, taskAttributes[i], null);
          if (captureResult == Resultats.ANNULEE) {
            break;
          }
          isInEdit = askChangeAttributeAgain(taskAttributes[i]);
        } while (isInEdit);
      }

    } else {
      result = false;
    }
    return result;
  }
  private boolean askChangeAttributeAgain(AtributTache attr) {
    System.out.print("Changer " + attr + " encore <O/N>: ");
    String querry = sc.nextLine();
    boolean isInEdit = querry.equalsIgnoreCase("O");
    return isInEdit;
  }

  private Resultats editAttribute(Tache task, AtributTache attr, Membre member) {

    Resultats result = Resultats.ECHEC;
    switch (attr) {
      case NOM:
        System.out.println("\nTapez entrée pour annuler la modification du nom");
        System.out.print("Tapez le nouveau nom de la tâche: ");
        String newName = sc.nextLine();
        result = editName(task, newName);
        break;

      case DESCRIPTION:
        System.out.println("\nTapez entrée pour annuler la modification de la description");
        System.out.print("Tapez la nouvelle description : ");
        String newDes = sc.nextLine();
        result = editDescription(task, newDes);
        break;

      case ETAT:
        System.out.println("\nChanger l'état de la tâche");
        Menu.showTaskStatusMenu();
        int choice = Menu.chooseTaskStatusMenu();
```

```
            result = setNewStatus(task, choice);
            break;

        case MEMBRE:
            result = setNewMember(task, member);
            break;

        default:
            System.out.println("Modification de la tâche annulée");

            break;
    }
    return result;
}
private Resultats editName(Tache task, String newName) {       //cannot be failed
    Resultats result = Resultats.ANNULEE;
    if (!newName.equals("")) {
        task.setName(newName);
        result = Resultats.REUSSIE;
    }
    return result;
}

private Resultats editDescription(Tache task, String newDes) {      //cannot be failed
    Resultats result = Resultats.ANNULEE;
    if (!newDes.equals("")) {
        task.setDescription(newDes);
        result = Resultats.REUSSIE;
    }
    return result;
}
private Resultats setNewStatus(Tache task, int choice) {
    Resultats result = Resultats.REUSSIE;
    switch (choice) {
        case Menu.TACHE_STATUT_NEW:
            task.setStatus(Etats.DISPONIBLE);
            break;

        case Menu.STATUS_TACHE_EN_COURS:
            task.setStatus(Etats.EN_COURS);
            break;

        case Menu.TACHE_STATUT_TERMINER:
            task.setStatus(Etats.TERMINE);
            break;

       case Menu.TACHE_STATUT_QUITTER:
            result = Resultats.ANNULEE;
            break;

        default:
            result = Resultats.ECHEC;
```

```
            break;
        }

        return result;
    }
    private Resultats setNewMember(Tache task, Membre member) {
        if (member == null) {
            return Resultats.ECHEC;
        }
        Resultats result = Resultats.ECHEC;
        if (!isAssignedToTask(member)) {
            task.setMember(member);
            result = Resultats.REUSSIE;
        }
        return result;
    }
    private boolean isAssignedToTask(Membre member) {
        if (member == null) {
            return false;
        }
        Membre assignedMember;
        for (Tache task : mListTask) {
            assignedMember = task.getMember();
            if (assignedMember != null && assignedMember.equals(member)) {
                return true;
            }
        }
       return false;
    }

public boolean supprimer(int id) {
        boolean result;
        Tache task = findTaskById(id);
        if (task != null) {
          mListTask.remove(task);

          result = true;

        } else {
          result = false;       }

        return result;
    }
    private Tache findTaskById(int id) {
        for (Tache task : mListTask) {
            if (task.getId() == id) {
                return task;
            }
        }
        return null;
    }
```

```
public boolean Ajouter() {
   boolean isEmptyTask = displayListCreateTask();       //id always exist to make task not null

   if (!isEmptyTask) {
      int id = chooseCreatedTaskById();
      Tache task = findCreateTaskById(id);
      mListCreatedTask.remove(task);
      mListTask.add(task);
   }
   return true;
}
private boolean displayListCreateTask() {
   boolean isEmptyTask = mListCreatedTask.isEmpty();
   if (isEmptyTask) {
      System.out.println("\nAucune tâche n'est créee");
   } else {
      System.out.println("\nListe des tâches crées");
      for (Tache task : mListCreatedTask) {
         System.out.println(task.toString());
      }
   }
   return isEmptyTask;    }

private int chooseCreatedTaskById() {
   System.out.println("Choisir une tâche par son ID");
   Tache firstTask = mListCreatedTask.get(0);
   Tache lastTask = mListCreatedTask.get(mListCreatedTask.size() - 1);
   int id = Menu.chooseMenu(firstTask.getId(), lastTask.getId());
   return id;
}

private Tache findCreateTaskById(int id) {
   for (Tache task : mListCreatedTask) {
      if (task.getId() == id) {
         return task;
      }
   }
   return null;
}

public void displayAllTask() {
   for (Tache task : mListTask) {
      System.out.println(task.toString());
   }
}
public List<Tache> findAllTaskByMemberId(int id) {
   List<Tache> list = new ArrayList<>();
   Membre member;
   for (Tache task : mListTask) {
      member = task.getMember();
      if (member != null && member.getId() == id) {
```

```
            list.add(task);          }

        }
        return list;
    }

    public List<Tache> findAllTaskByStatus(Etats status) {
      List<Tache> list = new ArrayList<>();
        if (status == null) {
            return list;
        }
        for (Tache task : mListTask) {
            if (task.getStatus() == status) {
                list.add(task);
            }
        }
        return list;
    }
    public Tache getTaskById(int id) {
        for (Tache task : mListTask) {
            if (task.getId() == id) {
                return task;
            }
        }
        return null;
    }

public int getTotalTask() {
        return mListTask.size();
}

   public List<Tache> getmListTask() {

        return mListTask;

    }



}
```

ClassGererMembre:

```
package com.team.app.manager;
import com.team.app.entities.Membre;
import com.team.app.entities.Menu;
import com.team.app.enumeration.AtributMembre;
import com.team.app.enumeration.Resultats;
import com.team.app.fileio.LectureEcriture;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import static com.team.app.enumeration.AtributMembre.NOM;
import java.lang.reflect.Member;

public class GererMembre implements GererOperations {
   private static int id = 1;
   private Scanner sc = new Scanner(System.in);
   private final List<Membre> mListMember = LectureEcriture.getListMemberFromFile();
   private final List<Membre> mListCreatedMember = new ArrayList<>();
   private final static GererMembre memberManager = new GererMembre();

   public static GererMembre getInstance() {
      return memberManager;
   }
   private GererMembre() { }

   @Override

   public boolean creer() {
      Membre member = makeNewMember();
      mListCreatedMember.add(member);
      return true;
   }

   private Membre makeNewMember() {
      System.out.print("\nEntrer le nom du membre: ");
      String name = sc.nextLine();
      return new Membre(id++, name);
   }

   @Override
   public boolean modifier(int id) {
      boolean result = true;
      Membre member = findMemberById(id);
      if (member != null) {
        boolean isInEdit;

         do {

            Resultats captureResult = editAttribute(member, NOM);
          if (captureResult == Resultats.ANNULEE) {
              break;

            }
            isInEdit = askContinueChangeName();
         } while (isInEdit);
      } else {
         result = false;
      }
```

```
    return result;

}

private boolean askContinueChangeName() {
    System.out.print("\nVoulez encore modifier <y/n>: ");
    String querry = sc.nextLine();
    boolean isInEdit = querry.equalsIgnoreCase("y");
    return isInEdit;
}

private Resultats editAttribute(Membre member, AtributMembre attr) {
    Resultats resultManager = Resultats.ECHEC;
    if (attr == NOM) {
        String newName = getNewName();
        if (newName.equals("")) {

            resultManager = Resultats.ANNULEE;

        } else {
            member.setNom(newName);
        }
    }
    return resultManager;
}

private String getNewName() {
    System.out.println("\nTapez entrer pour annuler la modification.");
    System.out.print("Entrer le nouveau nom : ");
    String newName = sc.nextLine();
    return newName;
}
public boolean supprimer(int id) {
    boolean result;
    Membre member = findMemberById(id);
    if (member != null) {
        mListMember.remove(member);
        result = true;
    } else {
        result = false;
    }
    return result;
}
private Membre findMemberById(int id) {
        for (Membre member : mListMember) {
            if (member.getId() == id) {
          return member;
        }
    }
    return null;
}

public boolean Ajouter() {

 boolean isEmptyMember = displayListCreateMember(); //id always exist to make member not null

    if (!isEmptyMember) {
        int id = chooseCreatedMemberById();
        Membre member = findCreateMemberById(id);
```

```
            mListCreatedMember.remove(member);
            mListMember.add(member);

        }
        return true;    }

    private boolean displayListCreateMember() {
        boolean isEmptyMember = mListCreatedMember.isEmpty();
        if (isEmptyMember) {

            System.out.println("\nMembre non crée");

        } else {
            System.out.println("\n Liste des membres");
            for (Membre member : mListCreatedMember) {
                System.out.println(member.toString());
            }
        }
        return isEmptyMember;
    }

    private int chooseCreatedMemberById() {
        System.out.println("Choisir un membre par son ID");
        Membre firstMember = mListCreatedMember.get(0);
        Membre lastMember = mListCreatedMember.get(mListCreatedMember.size() - 1);
        int id = Menu.chooseMenu(firstMember.getId(), lastMember.getId());
        return id;
    }
    private Membre findCreateMemberById(int id) {
        for (Membre member : mListCreatedMember) {
            if (member.getId() == id) {
                return member;          }
        }
        return null;    }

    public void addMember(Membre member) {
        mListMember.add(member);

    }
    public void displayAllMember() {
        for (Membre member : mListMember) {
            System.out.println(member.toString());
        }
    }

    public Membre getMemberById(int id) {
        for (Membre member : mListMember) {
            if (member.getId() == id) {
                return member;          }
        }
        return null;    }

    public int getTotalMember() {
        return mListMember.size();

    }
    public List<Membre> getmListMember() {
         return mListMember;    }
}
```

ClassGererUtilisateur:

```
package com.team.app.manager;


import com.team.app.entities.Membre;
import com.team.app.entities.Menu;
import com.team.app.entities.Tache;
import com.team.app.enumeration.Operation;
import com.team.app.enumeration.Resultats;
import com.team.app.fileio.LectureEcriture;

import java.util.List;
import java.util.Scanner;

public class GererUtilisateur {

   private Scanner sc = new Scanner(System.in);
        private static final List<Membre> listMember = GererMembre.getInstance().getmListMember();
        private static final List<Tache> listTask = GererTaches.getInstance().getmListTask();

   public void runSoftware() {
      int choice;

      do {
          Menu.showUserMenu();
         choice = Menu.chooseUserMenu();
         processQuery(choice);
      } while (true);
   }

   private boolean askContinueQuery() {
      System.out.print("\nVoulez-vous continuer en tant qu'utilisateur<O/N>: ");
      String query = sc.nextLine();
      boolean isInChoosen = query.equalsIgnoreCase("O");
      return isInChoosen;
   }

   private void processQuery(int choice) {
      boolean isInModify;
      String query;
      switch (choice) {

         case Menu.MODIFIER_MEMBRE:

            do {
               GererUtilisateurUtil.modifyMember();
               System.out.print("\nContinuer modifier <CREER, SUPPRIMER, AJOUTER, MODIFIER>
Membre <O/N>: ");

               query = sc.nextLine();
               isInModify = query.equalsIgnoreCase("O");
            } while (isInModify);
            break;

         case Menu.MODIFIER_TACHE:

            do {
```

```
                GererUtilisateurUtil.modifyTask();

                System.out.print("\nContinuer modifier <CREER, SUPPRIMER, AJOUTER, MODIFIER>
Tache <o/n>: ");

                query = sc.nextLine();
                isInModify = query.equalsIgnoreCase("O");
            } while (isInModify);
            break;

        case Menu.ASSIGN_TACHE_AU_MEMBRE:

            do {

                GererUtilisateurUtil.assignTaskToMember();
                System.out.print("\nVoulez-vous continuer à assigner les tâches? <O/N>: ");
                query = sc.nextLine();
                isInModify = query.equalsIgnoreCase("O");

            } while (isInModify);
            break;

        case Menu.RECHERCH_VOIR_TACHE_PAR_ID_DU_MEMBRE:
            do {
                GererUtilisateurUtil.displayAllTaskByMemberId();
                System.out.print("\nVoulez-vous continuer à rechercher et afficher les membres par ID?
<O/N>: ");

                query = sc.nextLine();
                isInModify = query.equalsIgnoreCase("O");
            } while (isInModify);
            break;

        case Menu.RECHERCH_VOIR_TACHE_PAR_STATUS:
            do {
                GererUtilisateurUtil.displayAllTaskByStatus();
                System.out.print("\nVoulez-vous continuer à rechercher et afficher les tâches par etat?
<O/N>: ");
                query = sc.nextLine();
                isInModify = query.equalsIgnoreCase("O");
            } while (isInModify);
            break;

        case Menu.QUITTER_LE_PROGRAMME:
            LectureEcriture.writeListMemberToFile(listMember);
            LectureEcriture.writeListTaskToFile(listTask);

            System.exit(0);

            break;

        default:

            System.out.println(Operation.ACTION_USER + " " + Resultats.ECHEC);

            break;

    }
  }
}
```

- **ClassGererUtilisateurUtil :**

```
package com.team.app.manager;


import com.team.app.entities.Membre;
import com.team.app.entities.Menu;
import com.team.app.entities.Tache;
import com.team.app.enumeration.Operation;
import com.team.app.enumeration.Resultats;
import com.team.app.enumeration.Etats;

import java.util.List;
import java.util.Scanner;

public class GererUtilisateurUtil {

    private static Scanner sc = new Scanner(System.in);
    private static final GererMembre memberManager = GererMembre.getInstance();
    private static final GererTaches taskManager = GererTaches.getInstance();

   public static void modifyTask() {
       ModificationTaches taskModification = new ModificationTaches(taskManager);
       taskModification.modifyTask();
    }

    public static void modifyMember() {
       ModificatioMembre memberModification = new ModificatioMembre(memberManager);
       memberModification.modifyMember();
    }

    public static void assignTaskToMember() {
       if (taskManager.getTotalTask() == 0) {
          System.out.println("Have no task to choose");
          return;
       }

       showAllTask();
       int taskId = choiceTaskId();
       if (taskId != Menu.ANNULER_ACTION) {  //task get here cannot be null because of taskId
always valid

          Tache task = taskManager.getTaskById(taskId);
          if (task.getStatus() == Etats.DISPONIBLE) {
             if (memberManager.getTotalMember() == 0) {
                System.out.println("Have no member to choose");
                return;
             }
             showAllMember();
             int memberId = choiceMemberId();
             if (memberId != Menu.ANNULER_ACTION) { //member get here cannot be null because of
taskId always valid

                Membre member = memberManager.getMemberById(memberId);
                    task.setMember(member);
                   task.setStatus(Etats.EN_COURS);
                   System.out.println(Operation.ASSIGNATION + " " + Resultats.REUSSIE);
```

```
            showTaskBeAssigned(task);

          } else {
            System.out.println(Operation.ASSIGNATION + " " + Resultats.ANNULEE);

          }
        }
    }
}

private static void showAllTask() {
    System.out.print("\nTapez entrée pour afficher toutes les tâches:");
    sc.nextLine();
    taskManager.displayAllTask();
}

private static void showAllMember() {
    System.out.print("\nTapez entrée pour afficher tous les membres:");
    sc.nextLine();
    memberManager.displayAllMember();
}

private static void showTaskBeAssigned(Tache task) {
    System.out.print("\nTapez entrée pour afficher toutes les tâches assignées:");
    sc.nextLine();
    System.out.println(task.toString());
}

private static int choiceTaskId() {
    System.out.println("Tapez entrée si vous souhaitez annuler l'opération");
    System.out.println("Tapez l'ID de la tâche que vous souhaitez assigner à un membre");
    int taskId = Menu.chooseMenu(1, taskManager.getTotalTask());
    return taskId;
}

private static int choiceMemberId() {
   System.out.println("Tapez entrée si vous souhaitez annuler l'opération");
    System.out.println("Tapez l'ID du membre à qui vous souhaitez assigner cette tâche");
    int memberId = Menu.chooseMenu(1, memberManager.getTotalMember());
    return memberId;
}

public static void displayAllTaskByMemberId() {
    showAllMember();
    System.out.println("Tapez l'ID du membre dont vous souhaitez voir les tâches");
    int memberId = Menu.chooseMenu(1, memberManager.getTotalMember());
    viewAllTaskByMemberId(memberId);
}

private static void viewAllTaskByMemberId(int id) {
    List<Tache> listTask = taskManager.findAllTaskByMemberId(id);
    for (Tache task : listTask) {
        System.out.println(task.toString());
    }
}
public static void displayAllTaskByStatus() {
    Menu.showTaskStatusMenu();
    System.out.print("\nTapez l'etat de la tâche");
    int choice = Menu.chooseTaskStatusMenu();
```

```
        Etats status = getStatusByChoice(choice);
        List<Tache> listTask = taskManager.findAllTaskByStatus(status);
        for (Tache task : listTask) {
            System.out.println(task.toString());
        }
    }

    private static Etats getStatusByChoice(int choice) {
        switch (choice) {
            case Menu.TACHE_STATUT_NEW :
                return Etats.DISPONIBLE;

            case Menu.STATUS_TACHE_EN_COURS:
                return Etats.EN_COURS;

            case Menu.TACHE_STATUT_TERMINER:
                return Etats.TERMINE;

            case Menu.TACHE_STATUT_QUITTER:
                System.out.println(Operation.AFFICHER + " " + Resultats.ANNULEE);

                return null;

            default:

                return null;

        }

    }

}
```

ClassModificationMembre:

```
package com.team.app.manager;


import com.team.app.entities.Menu;
import com.team.app.enumeration.Operation;
import com.team.app.enumeration.Resultats;
import com.team.app.enumeration.AtributMembre;
import java.util.Scanner;

public class ModificatioMembre {

    private static Scanner sc = new Scanner(System.in);
    private GererMembre memberManager;

    public ModificatioMembre(GererMembre memberManager) {
            this.memberManager = memberManager;
  }

    public void modifyMember() {
        int choice;
        Menu.showMenuModifyMember();
        choice = Menu.chooseMenuModify();
```

```
    if (choice == Menu.MODIFIER_QUITTER) {
      System.out.println("\nAnnuler modifier <CREER, SUPPRIMER, AJOUTER, MODIFIER>
Membre");

      return;
    }
    serveModifyMember(choice);
  }

  private void serveModifyMember(int choice) {
    switch (choice) {
      case Menu.MODIFIER_CREATION:
        System.out.print("\nCréer un membre:");
        createNewMember();
        break;

      case Menu.MODIFIER_SUPPRESSION:
        deleteMemberById();
        break;

      case Menu.MODIFIER_EDITION:
        editMemberById();
        break;

      case Menu.MODIFIER_AJOUT:
        addMemberById();
        break;

      default:

        System.out.println("Echec de la modification du membre");
        break;
    }
  }

  private void createNewMember() {
    boolean isInDelete;
    do {
      boolean result = memberManager.creer();
      if (result) {
        notifyResult(Operation.CREATION, Resultats.REUSSIE);
      } else {

        notifyResult(Operation.CREATION, Resultats.ECHEC);
      }
      isInDelete = askContinueCreate();
    } while (isInDelete);
  }

  private boolean askContinueCreate() {
    System.out.print("\nVoulez-vous continuerla création des membres? <O/N>: ");
    String query = sc.nextLine();
    boolean isInDelete = query.equalsIgnoreCase("O");
    return isInDelete;

  }

  private void deleteMemberById() {
    boolean isInDelete;
    do {
```

```
        memberManager.displayAllMember();
        int id = getDeleteMemberId();
        if (id != Menu.ANNULER_ACTION) {
            boolean isSuccess = memberManager.supprimer(id);
            if (isSuccess) {
                displayMemberAfterDeleted();
                notifyResult(Operation.SUPPRESSION, Resultats.REUSSIE);

            } else {

                notifyResult(Operation.SUPPRESSION, Resultats.ECHEC);

            }
            isInDelete = askContinueDelete();
        } else {
            notifyResult(Operation.SUPPRESSION, Resultats.ANNULEE);
            isInDelete = false;
        }

    } while (isInDelete);
}

private int getDeleteMemberId() {
    System.out.println("\nTapez entrée si vous souhaitez annuler");
    System.out.println("Tapez l'ID du membre pour supprimer");
    int id = Menu.chooseCancellableMenu(1, memberManager.getTotalMember());
    return id;
}

private void displayMemberAfterDeleted() {
    System.out.print("\nTapez entrée pour afficher la liste de membre après la suppression");
    sc.nextLine();
    memberManager.displayAllMember();
}

private boolean askContinueDelete() {
    System.out.print("\nVoulez-vous continuer la suppression des membres?<O/N>: ");
    String query = sc.nextLine();
    boolean isInDelete = query.equalsIgnoreCase("O");
    return isInDelete;
}

private void editMemberById() {
    boolean isInEdit;
    do {
        memberManager.displayAllMember();
        int id = getMemberIdToEdit();
        if (id != Menu.ANNULER_ACTION) {
            boolean isSuccess = memberManager.modifier(id);
            if (isSuccess) {
                displayMemberAfterEdit(id);
                notifyResult(Operation.MODIFICATION, Resultats.REUSSIE);

            } else {
                notifyResult(Operation.MODIFICATION, Resultats.ECHEC);
            }
            isInEdit = askContinueEdit();
        } else {
```

```
            notifyResult(Operation.MODIFICATION, Resultats.ANNULEE);
                isInEdit = false;
            }
        } while (isInEdit);
    }

    private void addMemberById() {
        boolean isSuccess = memberManager.Ajouter();
        if (isSuccess) {
            notifyResult(Operation.AJOUT, Resultats.REUSSIE);

        } else {

            notifyResult(Operation.AJOUT, Resultats.ECHEC);
        }
    }

    private int getMemberIdToEdit() {
        System.out.println("\nTapez entrée pour annuler");
        System.out.println("Tapez l'ID du membre à modifier");
        int id = Menu.chooseCancellableMenu(1, memberManager.getTotalMember());
        return id;
    }

    private void displayMemberAfterEdit(int id) {
        System.out.print("\nTapez entrée pour afficher le membre après modification");
        sc.nextLine();   //result true means we have a member with the id to avoid
memberManager.getMemberById(id) be null

        System.out.println(memberManager.getMemberById(id).toString());

    }

    private boolean askContinueEdit() {
        System.out.print("\nVoulez-vous continuer la modification des membres?<O>: ");
        String query = sc.nextLine();
        boolean isInDelete = query.equalsIgnoreCase("O");
        return isInDelete;
    }
    public void notifyResult(Operation action, Resultats result) {

        System.out.println(action + " " + result);
    }

    public void notifyResult(Operation action, AtributMembre attr, Resultats result) {

        System.out.println(action + " " + attr + " " + result);

    }

}
```

ClassModifiactionTache:

```
package com.team.app.manager;



import com.team.app.entities.Menu;
import com.team.app.enumeration.Operation;
import com.team.app.enumeration.Resultats;
import com.team.app.enumeration.AtributTache;

import java.util.Scanner;

public class ModificationTaches {

   private static Scanner sc = new Scanner(System.in);
   private GererTaches gererTache;

   public ModificationTaches(GererTaches taskManager) {
      this.gererTache = taskManager;
   }

   public void modifyTask() {
      int choice;
      Menu.showMenuModifyTask();
      choice = Menu.chooseMenuModify();
      if (choice == Menu.MODIFIER_QUITTER) {
         System.out.println("\nAnnuler modifier <CREER, SUPPRIMER, AJOUTER, MODIFIER>
Tache");

         return;
      }
      serveModifyTask(choice);

   }

   private void serveModifyTask(int choice) {
      switch (choice) {

         case Menu.MODIFIER_CREATION:
            System.out.print("\nCreation d'une tâche:");
            createNewTask();
            break;

         case Menu.MODIFIER_SUPPRESSION:
            deleteTaskById();
            break;

         case Menu.MODIFIER_EDITION:
            editTaskById();
            break;

         case Menu.MODIFIER_AJOUT:
            addTaskById();
            break;

         default:
```

```
            System.out.println("echec de la modification de la tache");
            break;
        }
    }

    private void createNewTask() {
        boolean isInDelete;
        do {
            boolean result = gererTache.creer();
            if (result) {
                notifyResult(Operation.CREATION, Resultats.REUSSIE);
            } else {
                notifyResult(Operation.CREATION, Resultats.ECHEC);
            }
            isInDelete = askContinueCreate();
        } while (isInDelete);
    }

    private boolean askContinueCreate() {
        System.out.print("\nVoulez-vous continuer la création des tâches?<O/N>: ");
        String query = sc.nextLine();
        boolean isInDelete = query.equalsIgnoreCase("O");
        return isInDelete;
    }

    private void deleteTaskById() {
        boolean isInDelete;

        do {
            gererTache.displayAllTask();

            int id = getDeleteTaskId();
            if (id != Menu.ANNULER_ACTION) {
                boolean isSuccess = gererTache.supprimer(id);
                if (isSuccess) {
                    displayTaskAfterDeleted();
                    notifyResult(Operation.SUPPRESSION, Resultats.REUSSIE);

                } else {
                    notifyResult(Operation.SUPPRESSION, Resultats.ECHEC);
                }
                isInDelete = askContinueDelete();
            } else {
                notifyResult(Operation.SUPPRESSION, Resultats.ANNULEE);
                isInDelete = false;
            }
        } while (isInDelete);
    }

    private int getDeleteTaskId() {

        System.out.println("\nTapez entrée si vous souhaitez annuler");
        System.out.println("Tapez l'ID de la tache à supprimer");
        int id = Menu.chooseCancellableMenu(1, gererTache.getTotalTask());
        return id;
    }
    private void displayTaskAfterDeleted() {
        System.out.print("\nTapez entrée pour afficher la liste des taches apres supression");
        sc.nextLine();
```

```
        gererTache.displayAllTask();
    }

    private boolean askContinueDelete() {
      System.out.print("\nVoulez-vous continuer la suppresion des tâches?<o/n>: ");
        String query = sc.nextLine();
        boolean isInDelete = query.equalsIgnoreCase("O");
        return isInDelete;
    }

    private void editTaskById() {
        boolean isInEdit;
        do {
            gererTache.displayAllTask();
            int id = getTaskIdToEdit();
            if (id != Menu.ANNULER_ACTION) {
                boolean isSuccess = gererTache.modifier(id);
                if (isSuccess) {
                    displayTaskAfterEdit(id);
                    notifyResult(Operation.MODIFICATION, Resultats.REUSSIE);

                } else {

                    notifyResult(Operation.MODIFICATION, Resultats.ECHEC);
                }
                isInEdit = askContinueEdit();
            } else {
                notifyResult(Operation.MODIFICATION, Resultats.ANNULEE);
                isInEdit = false;
            }
        } while (isInEdit);
    }

    private void addTaskById() {
        boolean isSuccess = gererTache.Ajouter();
        if (isSuccess) {
            notifyResult(Operation.AJOUT, Resultats.REUSSIE);

        } else {
            notifyResult(Operation.AJOUT, Resultats.ECHEC);
        }

    }

    private int getTaskIdToEdit() {
        System.out.println("\nTapez entrée pour annuler la modification de taches");
        System.out.println("Tapez l'ID de la tâche à modifier");
        int id = Menu.chooseCancellableMenu(1, gererTache.getTotalTask());
        return id;
    }

    private void displayTaskAfterEdit(int id) {
        System.out.print("\nTapez entrée pour afficher la tâche modifiée");
        sc.nextLine();   //result true means we have a task with the id to avoid
taskManager.getTaskById(id) be null
        System.out.println(gererTache.getTaskById(id).toString());
    }

    private boolean askContinueEdit() {
```

```
        System.out.print("\nVoulez-vous continuer la modification des tâches?<O/N>: ");
        String query = sc.nextLine();
        boolean isInDelete = query.equalsIgnoreCase("O");
        return isInDelete;
    }
    public void notifyResult(Operation action, Resultats result) {
        System.out.println(action + " " + result);
    }
    public void notifyResult(Operation action, AtributTache attr, Resultats result) {
        System.out.println(action + " " + attr + " " + result);
    }
}
```

ClassLectureEcriture:

```
package com.team.app.fileio;



import com.team.app.entities.Membre;
import com.team.app.entities.Tache;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

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

public class LectureEcriture {

    public static final String FICHIER_MEMBRES = "membres.txt";
    public static final String FICHIER_TACHES = "taches.txt";

    public static List<Membre> getListMemberFromFile() {
        List<Membre> list = new ArrayList<>();
        try {

            FileInputStream fin = new FileInputStream(FICHIER_MEMBRES);
            ObjectInputStream objIn = new ObjectInputStream(fin);
            list = (ArrayList<Membre>) objIn.readObject();
        } catch (Exception ex) {

            System.out.println("echec de la lecture du fichier membre " + ex.getMessage());
        }
        return list;
    }

    public static List<Tache> getListTaskFromFile() {
        List<Tache> list = new ArrayList<>();
        try {
            FileInputStream fin = new FileInputStream(FICHIER_TACHES);
            ObjectInputStream objIn = new ObjectInputStream(fin);
            list = (ArrayList<Tache>) objIn.readObject();
        } catch (Exception ex) {
            System.out.println("echec de la lecture du fichier tache " + ex.getMessage());
        }
```

```
        return list;
    }

    public static void writeListMemberToFile(List<Membre> list) {
        try {
            FileOutputStream fout = new FileOutputStream(FICHIER_MEMBRES);
            ObjectOutputStream objOut = new ObjectOutputStream(fout);
            objOut.writeObject(list);
        } catch (Exception ex) {
            System.out.println("echec d'ecriture dans le fichier membre" + ex.getMessage());
        }
    }
    public static void writeListTaskToFile(List<Tache> list) {
        try {
            FileOutputStream fout = new FileOutputStream(FICHIER_TACHES);
            ObjectOutputStream objOut = new ObjectOutputStream(fout);
            objOut.writeObject(list);
        } catch (Exception ex) {
            System.out.println("echec d'ecriture dans le fichier tache " + ex.getMessage());

        }
    }
}
```

ClassGestionOperations :

```
package com.team.app.manager;

public interface GererOperations {
    boolean creer();
    boolean modifier(int id);
    boolean supprimer(int id);
    boolean Ajouter();

}
```

ClassAttributMembre :

```
package com.team.app.enumeration;

public enum AtributMembre {

    ID, NOM

}
```

ClassAttributTache :

```
package com.team.app.enumeration;
public enum AtributTache {

    ID, NOM, DESCRIPTION, ETAT, MEMBRE
}
```

ClassEtat :

```
package com.team.app.enumeration;

public enum Etats {

   DISPONIBLE, EN_COURS, TERMINE

}
```

ClassOperation :

```
package com.team.app.enumeration;

public enum Operation {

   CREATION, MODIFICATION, SUPPRESSION, AJOUT, ASSIGNATION, AFFICHER,
ACTION_USER

}
```

ClassResultat :

```
package com.team.app.enumeration;

public enum Resultats {

   REUSSIE, ECHEC, ANNULEE

}
```

ClassTestMembre :

```
package com.team.app.entities;


import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;

import org.junit.Test;
import static org.junit.Assert.*;

public class MembreTest {

     public MembreTest() { }

    @AfterClass
   public static void setUpClass() {

   }
```

```
   @BeforeClass
public static void tearDownClass() {

}

@After
public void setUp() {

}

@Before
public void tearDown() {

}



@Test
public void testGetId() {
   System.out.println("getId");
   Membre instance = new Membre();
   int expResult = 0;
   int result = instance.getId();
   assertEquals(expResult, result);

  }

@Test

public void testSetId() {
   System.out.println("setId");
   int id = 0;
   Membre instance = new Membre();
   instance.setId(id);
     }



@Test
public void testGetNom() {
   System.out.println("getNom");
   Membre instance = new Membre(0,"Barry");
   String expResult = "Barry";
   String result = instance.getNom();
   assertEquals(expResult, result);
   }

@Test
public void testSetNom() {
   System.out.println("setNom");
   String nom = "";
   Membre instance = new Membre();
   instance.setNom(nom);
     }
```

```
    @Test
    public void testToString() {
        System.out.println("toString");
        Membre instance = new Membre();
        String expResult = "";
        String result = instance.toString();
        assertEquals("\nID: " + instance.getId() + "\nNom: " + instance.getNom() + "\n", result);

        }

}
```

G- CONCLUSION

La réalisation de ce TP nous a servi de rappel sur les concepts de la modélisation avec UML et programmation orientée- objet avec Java. Nous confirmons toute fois que ce travail n'est pas à sa plus haute perfection car une application n'est jamais parfaite. Mais elle pourrait bien être améliorée en lui ajoutant plus d'autres fonctionnalités permettant de l'adapter à une utilisation à grande envergure. Cependant, l'apport de ce travail a été d'une importance très considérable, en effet, il permet : de suivre une méthodologie de travail bien étudié, d'approfondir des connaissances sur les méthodes de développement des applications.

I- Références :

[1]: https://openclassrooms.com/courses/creez-des-applications-de-qualite-avec-le-design-pattern-mvc/le-genie-logiciel-gl.

[2] : [**L'application de normes de génie logiciel dans les très petites entreprises : Historique et premiers résultats]** disponible sur : https://www.researchgate.net/publication/248393128_L%27application_de_normes_de_genie_logiciel_dans_les_tres_petites_entreprises_Historique_et_premiers_resultats.

[3]: F. Coallier : International standardization in software and systems engineering, Crosstalk, février 2003, pp.21.

[4] : C. Y. Laporte et A. April : Applying software engineering standards in small settings: Recent historical perspectives and initial achievements ; International Research Workshop for Process Improvement in Small Settings, Soft-ware Engineering Institute, Pittsburgh, 19-20 octobre, 2005.

[5] : L'application de normes de génie logiciel dans les très petites entreprises : Historique et premiers résultats. disponible sur**:** https://www.researchgate.net/publication/248393128_L%27application_de_normes_de_genie_logiciel_dans_les_tres_petites_entreprises_Historique_et_premiers_resultats [accessed Aug 19, 2017].