La modélisation et l'implémentation du ProjetInfo

Comme vous avez vu dans le diagramme de UML, nous avons créé une classe **'User'** comme le parent de la sous-classe **'Administrator'**, **'Teacher'** **'Student'**, et **'Group',** pour bien simplifier de coder les attributs communs. Nous choisissons le Hashmap pour sauvegarder et modifier les données puisque Hashmap a des avantages en flexibilité et facilité de l'opération avec plusieurs méthodes propres comme **`get()`**, **`put()`**, **`remove()`**,etc.

1. UserDB.java

Pour permettre de charger la base de données au démarrage de l'application, nous avons défini un administrateur 'Su' au début et fait l'affectation des attributs dans la méthode de construction de **`UserDB()`**.

Pour la gestion de donnée, Nous choisissons le Hashmap pour sauvegarder et modifier les données puisque le Hashmap a des méthodes pratiques telles que **`get()`**, **`put()`**, **`remove()`**, etc. Donc, nous établissons quatres Hashmaps (***`mapStudent`***, ***`mapTeacher`***, ***`mapAdmin`***, ***`mapGroup`***) dans **`UserDB`**. Le Key de Hashmap des utilisateurs est leur login et le Key de Hashmap de groupe est son groupId.

Notre solution est de gérer les fichiers et arbre XML uniquement dans **`loadDB()`** et **`saveDB()`**. Dans la méthode **`loadDB()`**, nous ouvrons le fichier ***`UserDB.xml`*** et pour chaque élément parcouru via la méthode (**`Iterator<Element>.hasnext()`**) , nous créons l'objet correspondant, puis l'ajoutons dans le Hashmap correspondant. Dans la méthode **`saveDB()`**, nous parcourons nos Hashmaps via la méthode(**`Entry<Type,Objet> entry`**) et nous créons une arbre XML pour les sauvegarder dans le fichier ***`UserDB.xml`***.

Globalement, afin de modifier (ajouter, associer, supprimer) les éléments dans la base de données de ***`UserDB.xml`***, il nous faut exécuter la méthode **`loadDB()`** pour enregistre les données existantes de ***`UserDB.xml`*** dans les Hashmaps établis au préalable. Et puis nous pouvons directement rappeler les méthodes du Hashmap telles que **`get()`**,**`put()`**, **`containsKey()`**, **`remove()`** pour réaliser les opérations telles que **`addStudent()`**, **`addTeacher()`**, **`removeUser()`**. A la fin, la méthode **`saveDB()`** sera exécutée pour sauvegarder les changement de Hashmap dans l'arbre de UML.

Exemple 1:

```
public void addGroup(String adminLogin, int groupID) {
        if (mapAdmin.containsKey(adminLogin)) {
            Group group = new Group();  //créer un nouveau objet ;
            mapAdmin.get(adminLogin).createGroup(group, groupID); //modifier les
           valeurs correspondant dans group
            mapGroup.put(groupID, group); //mettre le key en groupID et l'objet
groupe dans le Hashmap mapGroup ;
        }
        saveDB(); // enregistrer les données modifiées
```

2. Admin.java

Admin.java joue principalement le rôle de aider à réaliser les méthodes **`addAdmin()`**, **`addStudent()`**, **`addTeacher()`**et **`addGroup()`** de **UserDB.java .** Il contient les méthodes **`createAdmin()`**, **`createStudent()`**, **`et createTeacher()`** pour faire l'affectation des attributs du nouveau objet.

Exemple 2:

```
public void createAdmin(Admin admin, String newAdminlogin, int adminID, String
firstname, String surname, String pwd) {
        admin.Login = newAdminlogin; //Le nouveau login du Admin ;
        admin.adminId = adminID; //Le nouveau ID du Admin ;
        admin.Firstname = firstname; //Le nom du nouveau Admin ;
        admin.Lastname = surname; // Le prénom du nouveau Admin ;
        admin.Password = pwd; //Le mot de passe de nouveau Admin ;
        }
```

3. UserController.java

Dans UserController.java , nous rappelons les méthodes du **UserDB.java** pour réaliser les méthode demandées.

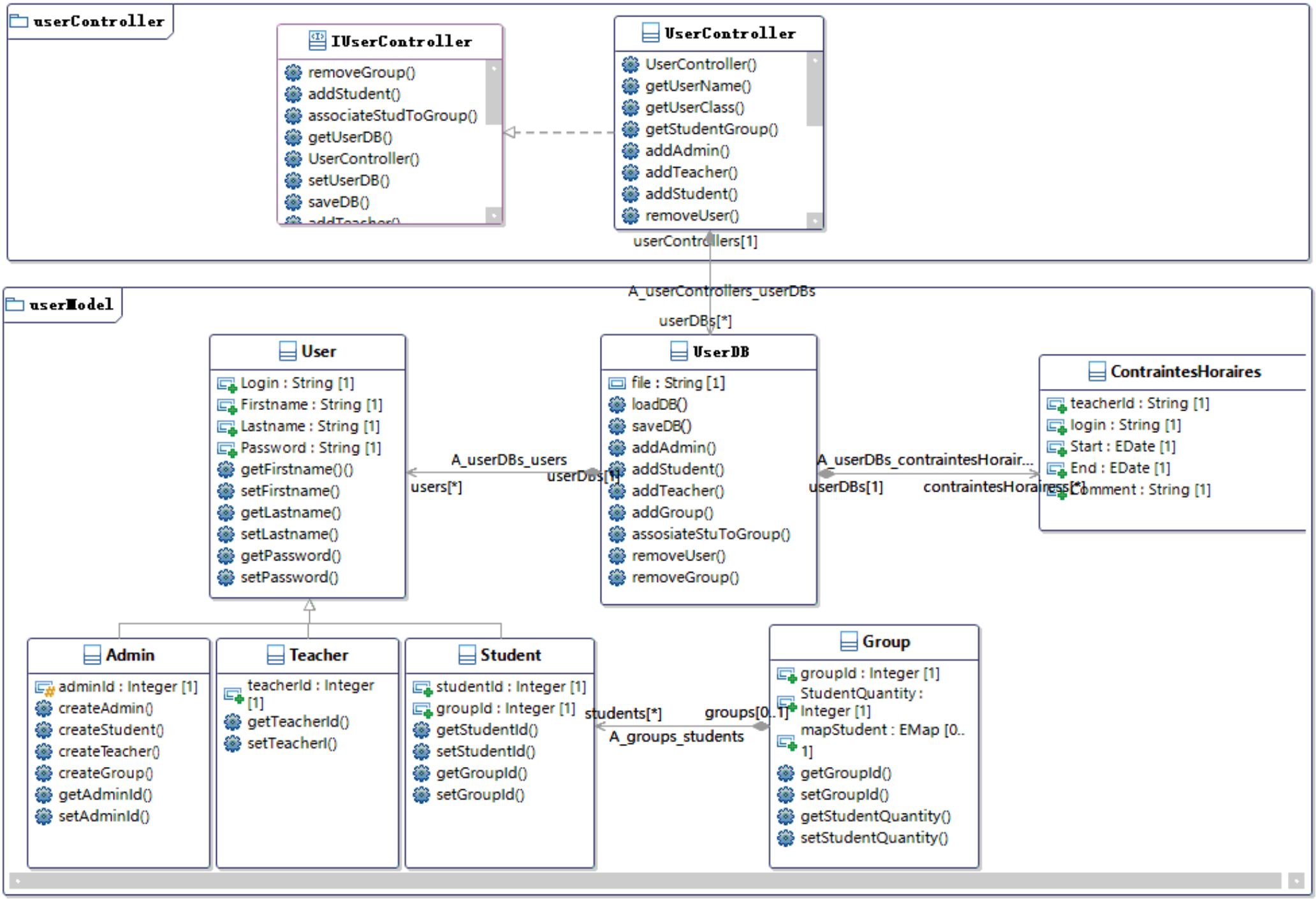
Exemple 3:

```
public boolean addTeacher(String adminLogin, String newteacherLogin, int
teacherID, String firstname, String surname, String pwd) {
        userDB.addTeacher(adminLogin, newteacherLogin, teacherID, firstname,
surname, pwd);// Rappeler la méthode dans UserDB pour ajouter le teacher;
        return userDB.mapTeacher.containsKey(newteacherLogin);
}
```

Exemple 4:

```
    public String[] groupsIdToString() {
        String[] groupId; // Définir un tableau pour enregistrer le groupID ;
        int i = 0; // Définir un compteur
        int m = userDB.mapGroup.size(); // Obtenir le largeur de ce Hashmap
        groupId = new String[m]; // Initialiser le tableau de dimension m
        for (Entry<Integer, Group> entry : userDB.mapGroup.entrySet()) {
            groupId[i] = String.valueOf(entry.getValue().groupId);
            i++; // Mettre les valeurs de chaque groupID dans le tableau
        }
        return groupId; //Retour le tableau groupID
        }
```

Annexe-UML


userController
IUserController
removeGroup()
addStudent()
associateStudToGroup()
getUserDB()
UserController()
setUserDB()
saveDB()
UserController
UserController()
getUserName()
getUserClass()
getStudentGroup()
addAdmin()
addTeacher()
addStudent()
removeUser()
userControllers[1]
A_userControllers_userDBs
userDBs[*]
userModel
User
Login : String [1]
Firstname : String [1]
Lastname : String [1]
Password : String [1]
getFirstname()()
setFirstname()
getLastname()
setLastname()
getPassword()
setPassword()
A_userDBs_users
users[*]
userDBs[1]
UserDB
file : String [1]
loadDB()
saveDB()
addAdmin()
addStudent()
addTeacher()
addGroup()
assosiateStuToGroup()
removeUser()
removeGroup()
A_userDBs_contraintesHorair...
userDBs[1]
contraintesHoraires[*]
ContraintesHoraires
teacherId : String [1]
login : String [1]
Start : EDate [1]
End : EDate [1]
Comment : String [1]
Admin
adminId : Integer [1]
createAdmin()
createStudent()
createTeacher()
createGroup()
getAdminId()
setAdminId()
Teacher
teacherId : Integer [1]
getTeacherId()
setTeacherI()
Student
studentId : Integer [1]
groupId : Integer [1]
getStudentId()
setStudentId()
getGroupId()
setGroupId()
students[*]
A_groups_students
groups[0..1]
Group
groupId : Integer [1]
StudentQuantity : Integer [1]
mapStudent : EMap [0..1]
getGroupId()
setGroupId()
getStudentQuantity()
setStudentQuantity()