19/06/2017

Projet de développement objet :

Système d'apprentissage des langues

Formation ingénieur ENSSAT en
Informatique, Multimédia et Réseaux par apprentissage
1ère année

Cheikh Saliou Ndiaye
IMR °ENSSAT LANNION

Table des matières

Introduction

Dans le cadre de notre formation d'ingénieur en IMR (informatique, multimédia et réseaux), nous avons suivi des cours en développement objet.
Apres avoir suivi la première partie des cours d'initiation au langage Java, nous avons démarré la deuxième partie du programme qui concernait l'architecture MVC (Modèle, Vue, Contrôleur avec les interfaces graphiques Swing).
En effet, après les séances de travaux pratiques nous avons été amenés à réaliser un mini projet. Ce mini projet consiste à développer un système d'apprentissage des langues (anglais, espagnol, Italien, Allemand) en utilisant les technologies vues en développement objet afin de mettre en pratique les connaissances acquises en cours.
Ce rapport est subdivisé en trois parties: dans un premier temps nous allons voir l'étude de projet puis nous entamerons la phase de développement et de test de l'application et enfin nous allons terminer par une conclusion.

I. Etude du projet

1. Cahier des charges

L'objectif de ce projet est de concevoir un système pour l'apprentissage des langues étrangères destiné à des utilisateurs francophones. Le système de base propose quatre fonctions.
Pour accéder à ces fonctions, il faudra passer par un menu principal (point d'accueil 'menu principal').
Le menu principal doit présenter la liste des fonctions comme un ensemble de choix et l'utilisateur effectue son choix. Chaque choix sélectionné doit déclencher une fonction associée parmi les cas suivants:

Cas 1 : Créer un nouveau compte utilisateur, dans ce premier cas l'utilisateur aura à fournir un identifiant personnel, son âge, genre (H/F), à choisir une langue cible parmi une liste qui lui est proposée par le système (Anglais, Espagnol, Italien et Allemand).
Une fois que l'utilisateur a créé son compte, il peut entamer une leçon à travers une session ou revenir au menu principal.

Cas 2 : Dans le deuxième cas l'utilisateur peut accéder à son compte (compte préalablement Créé), en utilisant son identifiant personnel, une fois la saisie faite le système vérifie la validité de l'identifiant fourni par l'utilisateur. Si le système n'arrive pas à trouver le compte d'utilisateur, le système affiche un message d'erreur. Dans le cas contraire le système fournit une session et un tableau de bord associé à l'utilisateur (attention le système doit prendre en compte les informations antérieures de l'utilisateur, un genre d'historique). Une fois dans la session, l'utilisateur peut entamer une leçon donnée. Il y a quatre types de leçon (Grammaire, Conjugaison, Orthographe et Vocabulaire) et chaque leçon est composée d'un nombre fini d'exercices.

Cas 3 : Le système permet à un utilisateur anonyme d'accéder à une session et de faire des leçons (sans enregistrer ses données personnelles). Là aussi le système permet à l'utilisateur de quitter la session en cours (revenir au menu principal) ou bien de créer un compte (voir cas 1 ci-dessus).

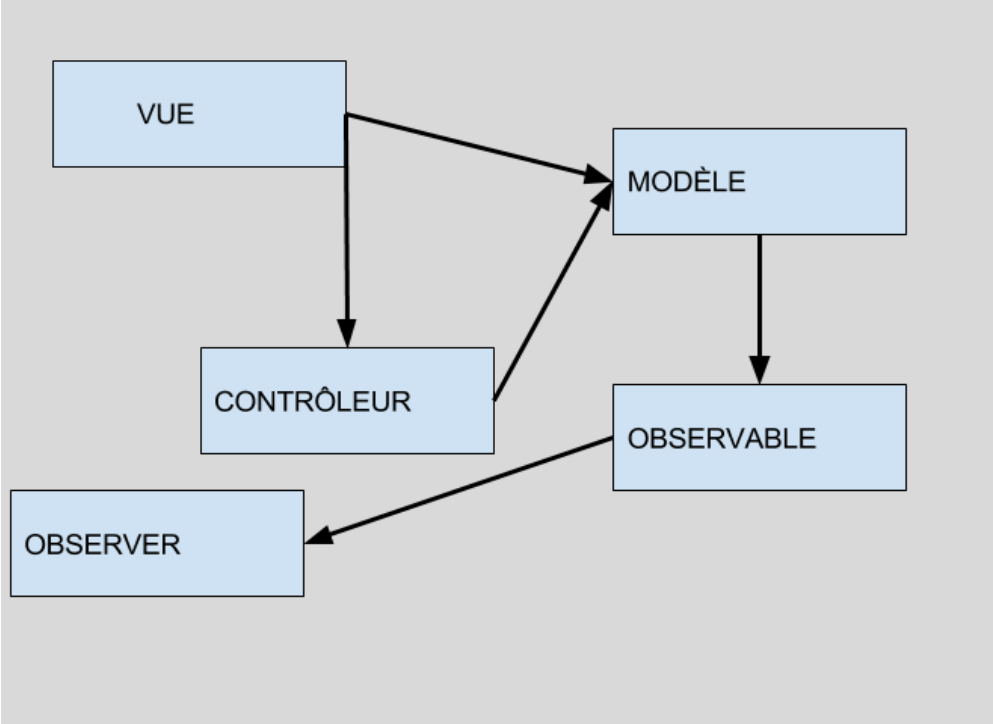
Cas 4 : La dernière fonction est celle qui permet à l'utilisateur de quitter définitivement le système.

Après l'étude du cahier des charges qui nous a été fourni, j'ai identifié les différentes classes à mettre en œuvre ainsi que leurs relations.
Une des contraintes de ce projet consiste à le développer en implémentant sur le modèle MVC (Modèle Vue Contrôleur) vue en cours. Nous allons aussi réaliser une interface graphique (Swing) pour cette application.
Effet, la première tâche que j'ai effectuée est la production de l'architecture de développement et les différentes classes de l'application qui fait intervenir les quatre fonctions que j'ai à développer.

II. Développement de l'application

Dans cette partie, nous allons expliquer les différentes méthodes utilisées pour réussir le projet étape par étape.

Ci-dessous le type d'architecture que j'ai utilisé :



Nous avons un modèle, une vue, un contrôleur et les interfaces **Observers**, **Observable**.

Détail et explication de l'architecture :

Pour développer l'application je l'ai divisée en plusieurs sous-parties (création de compte, connexion à son compte, liste des utilisateurs, etc. Chaque de ces partie dispose d'un contrôleur et d'une vue.

Le modèle :

C'est dans le modèle que nous avons les données de l'application. Le package modèle comporte 4 classes.

Data : cette classe nous permet :

- D'ajouter l'utilisateur dans liste des utilisateurs nous avons utilisé « Add » qui permet d'ajouter l'élément spécifié (utilisateur) à la fin de la liste.
- De faire un exercice en utilisant le type de leçon, le numéro de l'exercice, la langue utilisé. L'exercice fait avec le type de leçon, le numéro de l'exercice, la langue sera ajouté dans la session.
- De gérer l'utilisateur connecté, si l'utilisateur est connecté (userConnected est diffèrent de **null).** userConnected sera utilisé pour déconnecter un utilisateur et savoir quel utilisateur est connecté.

Exercice : cette classe prend en paramètre le type de leçon, le numéro de l'exercice et la langue utilisé pour faire l'exercice. Ces trois attributs correspondent à des entiers. J'ai implémenté un switchesCase type de leçon `switch (typeLecon)` du coup en récupérant le numéro du type de leçon correspondant je fais la correspondance.

```
Exemple : case 0:
          txt += "de grammaire";
```

Session :

Après la création d'un compte ou connexion à son compte, l'utilisateur peut entamer une session. Dès que l'utilisateur se connecte, on sauvegarde l'heure de début de la session.
this.dateStart = System.*currentTimeMillis*(); //sauvegarde l'heure du début de la session en milliseconde
Une fois dans la session, l'utilisateur peut faire un exercice en choisissant le numéro de l'exercice et le type de leçon.
Lorsque l'entier type de leçon est zéro, ceci correspondant à la leçon de grammaire (voir classe exercice au-dessus). Ce même procédé sera utilisé pour le numéro de l'exercice et la langue utilisé pour faire l'exercice.

Enfin, pour la durée de la session :**this**.duree = **this**.dateEnd - **this**.dateStart; .

Utilisateur :

Cette classe prend en paramètre l'identifiant, l'âge, le genre et la langue de l'utilisateur. C'est dans cette classe qu'on met à jour le profil de l'utilisateur avec la fonction update(**int** age, String genre, **int** langue).
Nb : l'identifiant ne peut pas être mis à jour.

La Vue

Dans le package, nous pouvons distingués trois sous-parties :

- La classe principale Fenêtre : cette classe permet de basculer d'un menu à un autre. Cette classe hérite de la classe **AbstractFenetre** qui implémente un **observer** et qui hérite un JFrame.

- Les différentes classes de vue pour les menus (exemple menu création, menu connexion, etc)
- La classe bouton : cette classe permet d'implémenter des boutons personnalisés de l'application.

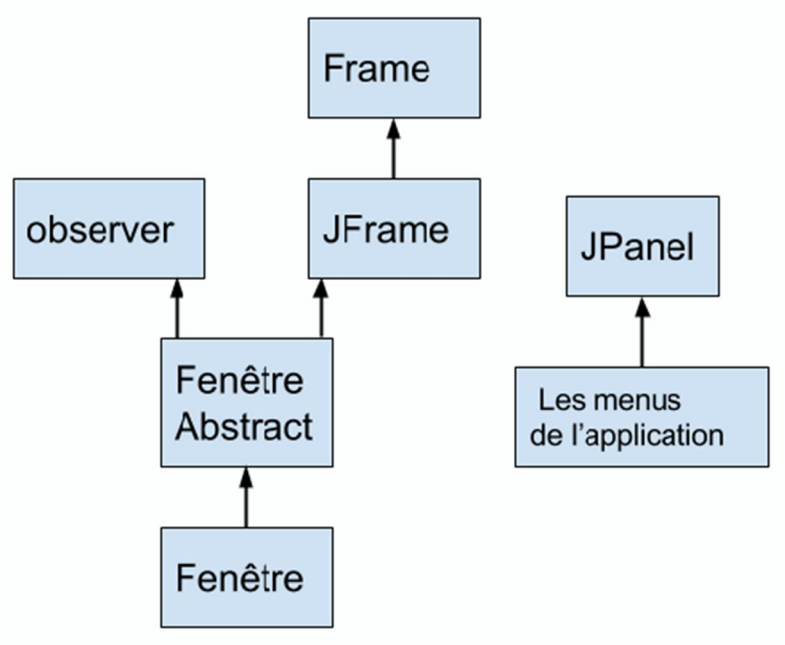


Schéma du package Vu

La classe Principale Fenêtre permet de changer l'affichage en fonction de choix de l'utilisateur. Pour cela, j'ai implémenté une méthode update qui gérer l'affichage des différentes menus ou parties de l'application.

Exemple :nb.getValue() == TypeWindow.***MENU_PRINCIPAL***.getValue() correspond menu principal de l'application. Dans ce cas on fait appel au contrôleur et à la vue correspondants.

Le même procédé est utilisé pour afficher les autres menus de l'application ou pop up.

La classe **Bouton** hérite de JBouton et implémente **MouseListener**. **MouseListener** s'intéresse au traitement d'un événement de souris (appuyez sur, relâchez, cliquez, entrez) implémente toutes les méthodes qu'elle contient exemple **mouseEntered** cette méthode permet de gérer le survol de nos boutons).

La méthode `mouseEntered` nous permet de changer le fond de notre image pour le jaune lors du survol, avec le fichier fondBoutonHover.png.

La méthode `mousePressed` nous permet de changer le fond de notre image pour le vert lorsque nous cliquons sur un bouton, avec le fichier fondBoutonClic.png.

Le Contrôleur

Dans ce package, nous avons deux parties :

- La classe **abstractMenuControlleur** : c'est l'abstraction de chaque menu qui implémente qui implémente l'observable.

- Les autres classes (menus contrôleurs des différents cas de fonctionnement) : ces classes héritent de la classe **abstractMenuControlleur** et développe la fonctionnalité attendue.

Par exemple pour la classe **MenuCreationControler,** elle hérite de la classe **abstractMenuControlleur**. **MenuCreationControler** dispose d'une fonction qui ajoute l'utilisateur à liste des utilisateurs crée. Avant l'ajout on vérifie si l'identifiant et l'âge sont bien renseignés et que l'identifiant est bel et bien disponible.

Ce principe sera utilisé par les autres menus contrôleur de l'application.

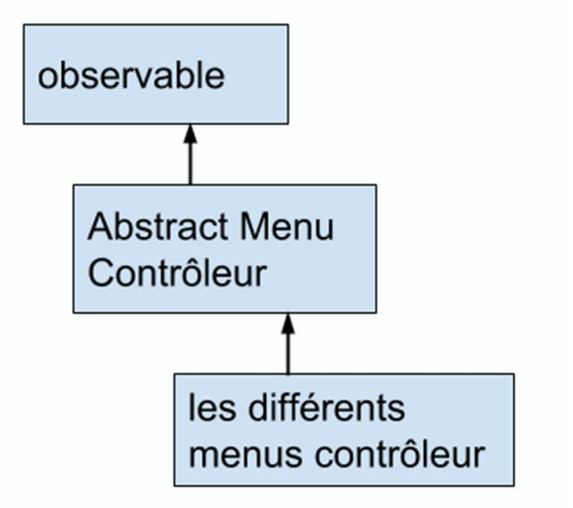


Schéma du package Contrôleur

Observable (observé) notifie aux objets dépendants : **Observers** (observateurs) tout changement de son état (mise à jour).

La classe Observable (le modèle) permet de gérer une liste d'Observer (les vues) avec les méthodes suivantes:

- Void addObserver (Observer o); : rajoute un Observer (implémentation du pattern observer).
- Vvoid removeObserver() supprime un Observer (fonction qui sert a supprimer un observer).
- Void notifyObservers (Object arg); : si un changement a eu lieu, tous les Observers sont notifiés du update ().Void notifyObservers est une fonction qui sert à notifier les observers.

La vue implémente **Observers.**

L'interface Observer permet d'observer un objet Observable au moyen de :void update. L'objet Observable signale à l'objet receveur (Observer) qu'un changement est survenu. La méthode update est indispensable pour que l'Observer se mette à jour.

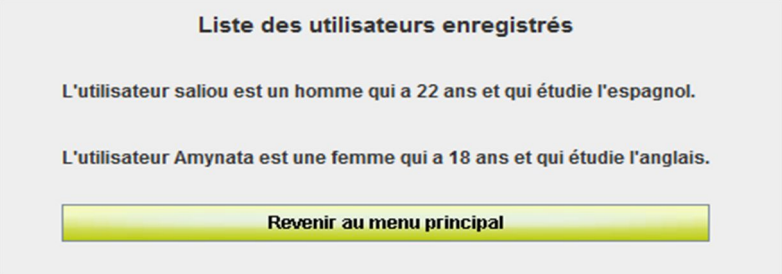
III. Teste de l'application

Le code source fournis en annexe propose une application fonctionnelle répondant au cahier des charges imposé. En effet, une fois les développements terminés, nous avons réalisé une série de tests afin de vérifier que le système développé répondait aux exigences du cahier des charges.
Les tests ont été réalisés selon les scénarios d'usage décris précédemment (4 cas de fonctionnement).

- **Test création de compte**

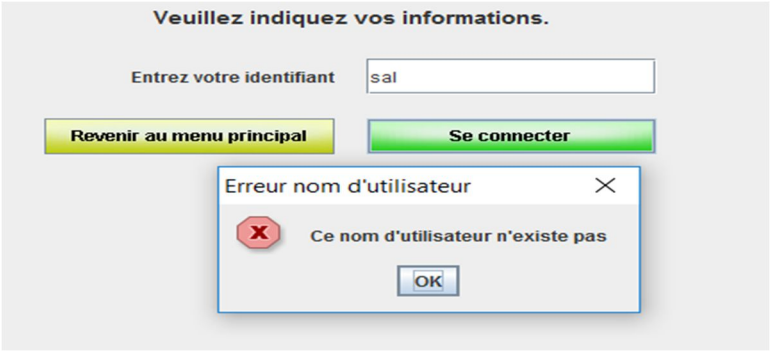
Apres avoir lancé l'application, nous allons créer de nouveaux comptes utilisateurs.
Créons deux comptes utilisateurs et affichons-les:

- Saliou – 22 ans – homme - espagnol
- Amynata – 18 ans – femme – anglais



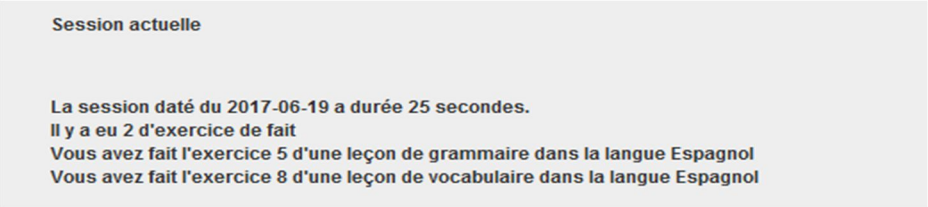
- **Test erreur de connexion**

Connectons nous sur le compte de Saliou, premièrement nous allons délibérément faire trois erreurs en renseignant l'identifiant pour tester qu'au bout du troisième essai, le système affiche un message d'erreur et revient au menu principal.



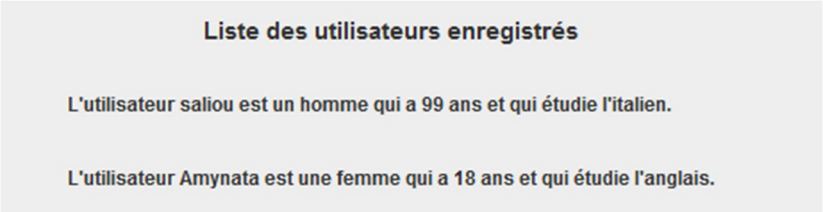
- **Test connexion à un compte et démarrons une session**

Maintenant, connectons-nous sur le compte de Saliou, démarrons une session :
- avec une leçon de grammaire, exercice 5
- avec une leçon de vocabulaire, exercice 8



- Test mis à jour d'un profil utilisateur

Mettons à jour le profil de Saliou, changeons son âge à 99 ans et sa langue en italien.



Comme on peut le constater, sur le schéma au-dessus, l'âge et la langue de l'utilisateur Saliou a été mis jour. L'utilisatrice Amynata, reste comme elle a été créée au départ.

Pour terminer, on quitte définitivement le programme.
Le programme se ferm.e
En quittant le programme définitivement, aucune nouvelle saisie n'est possible.
Toutes les informations sont perdues, car elles ont été enregistrées dans des listes et non dans un fichier txt ou une base données.

Liste des fonctionnalités testée

libellé	Résultat
Affichage page d'accueil : affichage des choix possibles	Fonctionnel
Créer un compte si nouvel utilisateur	Fonctionnel
accéder à son compte si existant	Fonctionnel
Accéder à une session après connexion	Fonctionnel
Historique de connexion	Fonctionnel
Faire un exercice	Fonctionnel
Mettre à jour profil	Fonctionnel
connexion anonyme	Fonctionnel
quitter l'application	Fonctionnel
Mettre à jour profil	Fonctionnel
Liste des utilisateurs	Fonctionnel

Conclusion

Ce projet nous a permis de concevoir un système d'apprentissage de langue en MVC à partir d'un cahier des charges. La première partie du projet était consacrée à l'étude de projet qui nous a permis d'identifier les différents cas d'utilisation et de produire un diagramme de classe.

Ensuite, nous avons débuté la phase du développement qui nous a permis de mettre en pratique au niveau logiciel l'étude effectuée dans la partie 1(étude de projet).
Une fois le développement terminé, nous avons effectué des tests permettant de valider les différents scénarios d'utilisation.
Cette phase de test, m'a permis de bien vérifier le **bon fonctionnement** de l'application.
Au niveau personnel, je suis me investit à fonds sur ce projet.

Nota : Un document utilisateur, un script de compilation et code source fonctionnel et commenté est joint au zip déposé sous nom.