CLEMENTINE : SPÉCIFICATIONS TECHNIQUES

Sommaire

L'application Clémentine : Vue globale

Présentation générale

L'application *Clémentine* est une application d'aide au programme sanitaire éponyme dont le but est de promouvoir l'intérêt et la pratique d'une activité physique à des fins de santé pour les séniors. Elle est développée en java selon le paradigme MVC (Modèle-Vue-Contrôleur) et est connectée à une base de données stockée sous PostgreSQL. Madame Sara Husson, éducatrice sportive et animatrice du programme est actuellement la seule utilisatrice de l'application.

La base de données

La base de données nommée *M2L* est stockée sur une instance PostgreSQL du serveur 192.168.222.86 sur le port 5432.

Une partie de la structure de *M2L* est illustrée à travers le schéma ci-dessous :

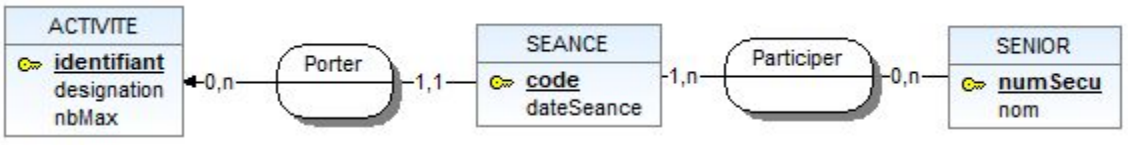


Schéma entité-association de M2L

Les fonctionnalités

Les fonctionnalités de l'application sont illustrées à travers le schéma ci-dessous :

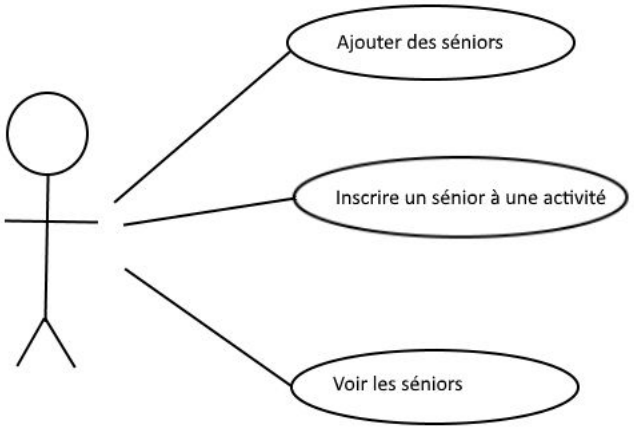
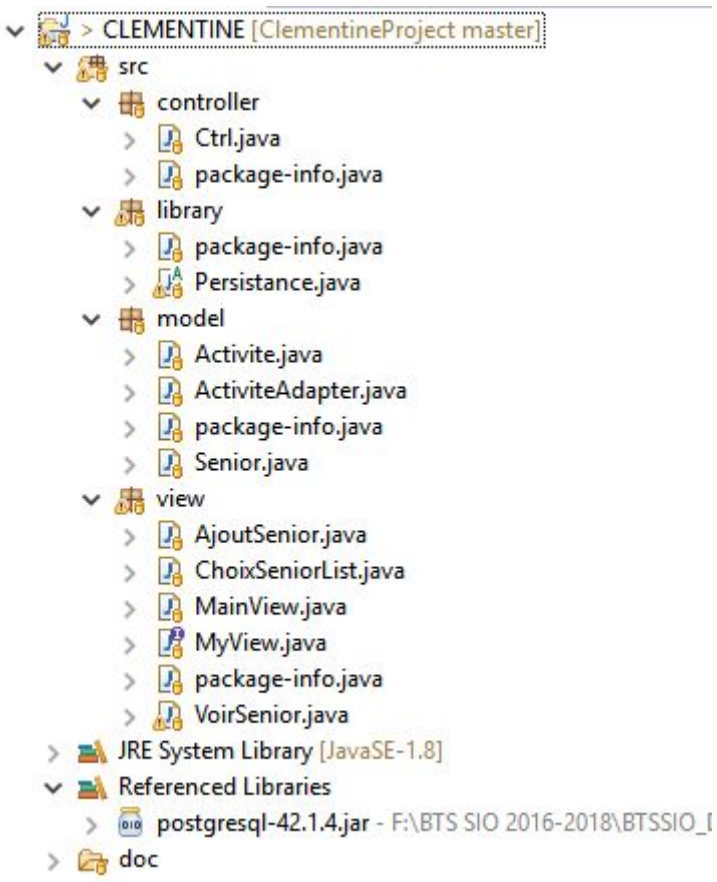


Diagramme des cas d'utilisation de Clémentine

L'application Clémentine : Vue technique

L'application java

L'archive jar M2L_CLEMENTINE contient les codes sources de l'application organisés sous l'arborescence suivante :



- Le dossier *doc* contient toute la javadoc relative à l'applicatif ;
- Le dossier *Referenced Librairies* contient toutes les archives java nécessaires au fonctionnement de l'application (ici seul le connecteur *postgresql-42.1.4* permettant de se connecter à une base de données PostgreSQL est présent) ;
- Le dossier *JRE System Library* contient le JRE nécessaire à l'exécution du programme ;
- Le dossier *src* contient les codes sources java du projet répartis en quatre packages : *controller*, *library*, *model* et *view* ;
- Dans chacun de ces packages se trouve un fichier *package-info.java* définissant la javadoc du package ;

Le package *library*

Le package *library* contient une classe abstraite qui propose des méthodes statiques :

- La classe *Persistance* qui offre des services liés à la persistance des objets c'est-à-dire à leur enregistrement dans la base de données ;

Le package *model*

Le package *model* contient les classes métier de l'application. On y trouve les classes Senior, Activite et ActiviteAdapter

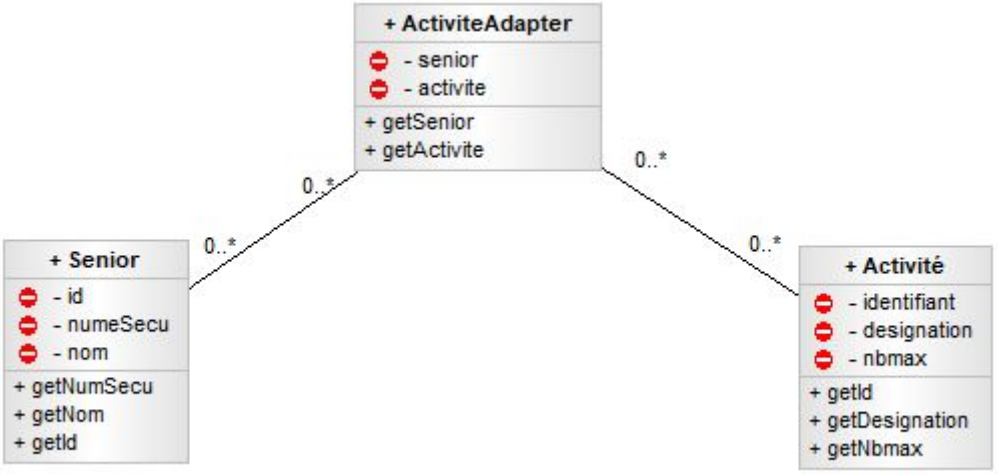


Diagramme des classes de Clémentine

Le package *view*

Le package *view* contient les différentes vues de l'application. Celles-ci héritent toutes de l'interface *MyView* qui déclare une méthode abstraite *assignListener*. Cette méthode est donc obligatoirement renseignée dans chacune des vues. Elle permet de définir l'observateur des événements (clics, frappes claviers...) et donc le traitement de ceux-ci.
Au démarrage de l'application, la *MainView* crée une instance de la classe *Ctrl*. À chaque création de nouvelle fenêtre, cette instance y sera renseignée comme étant l'observateur.

La navigation au sein de l'application se réalise selon le schéma suivant :

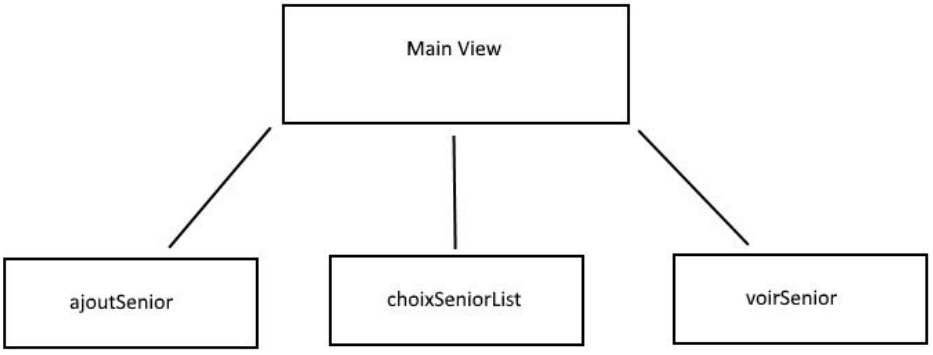


Schéma d'enchaînement des fenêtres de Clémentine

Le package *controller*

Le package *controller* contient la classe contrôleur de l'application : *Ctrl*. Ce contrôleur implémente l'interface *ActionListener*. De plus, il implémente de design pattern *Singleton*. Il est l'observateur unique de l'application et en fonction des événements opérés sur les vues, il réalisera les traitements correspondants. Il dispose de plusieurs méthodes publiques :

- Le point d'accès vers l'instance unique du singleton (*getCtrl()*) ;
- La méthode héritée de *ActionListener* (*actionPerformed()*) qui permet les traitements (interrogation du modèle et/ou navigation dans les vues) ;

La navigation dans Clémentine

Rappels sur l'interface *ActionListener*

1. Aux composants graphiques de type *JButton* d'une vue peut être assigné un observateur à travers la méthode *addActionListener.*
2. Il est également possible de définir une « commande » pour le composant à travers la méthode *setActionCommand.* Cette commande correspond à une signature que le composant envoie à l'observateur.
3. L'observateur assigné implémente obligatoirement l'interface *ActionListener*. Celle-ci force la redéfinition de la méthode *actionPerformed*. Cette méthode est automatiquement exécutée lors d'un clic sur le bouton observé.
4. Dans la méthode a*ctionPerformed*, il est possible de récupérer la commande du bouton (et d'en déduire l'action de l'utilisateur) grâce à la méthode *getActionCommand*.

Note : les interfaces *MouseListener*, *KeyListener*, etc fonctionnent sur le même modèle.

Les messages de l'actionCommand : fonctionnement

Afin de mettre en place la navigation au sein de l'application, les messages envoyés entre classe observée et classe observatrice sont structurés selon la façon suivante :

nomClasseObservée_nomBoutonCliqué

Le message ainsi réceptionné par l'observateur (c'est-à-dire le contrôleur) permettra de déterminer sans ambiguïté l'origine de l'action et ainsi de définir son traitement spécifique.

Les messages de l'actionCommand : liste

Vue	Nom composant	Message
MainView	btnSenior	MainView_senior
MainView	btnInscription	MainView_inscription
MainView	btnVueSenior	MainView_vuesenior
AjoutSenior	btnValider	AjoutSenior_valider
AjoutSenior	bntAnnuler	AjoutSenior_annuler

Note : les boutons *Fermer* ne sont pas observés par le contrôleur. Leur mise en œuvre est effectuée à travers des classes anonymes.

Les traitements

Création d'un nouveau sénior

Le traitement décrit ci-dessous correspond aux instructions exécutées dans la méthode *ActionPerformed* de la classe *Ctrl* lors de l'envoi du message *AjoutSenior_valider*.

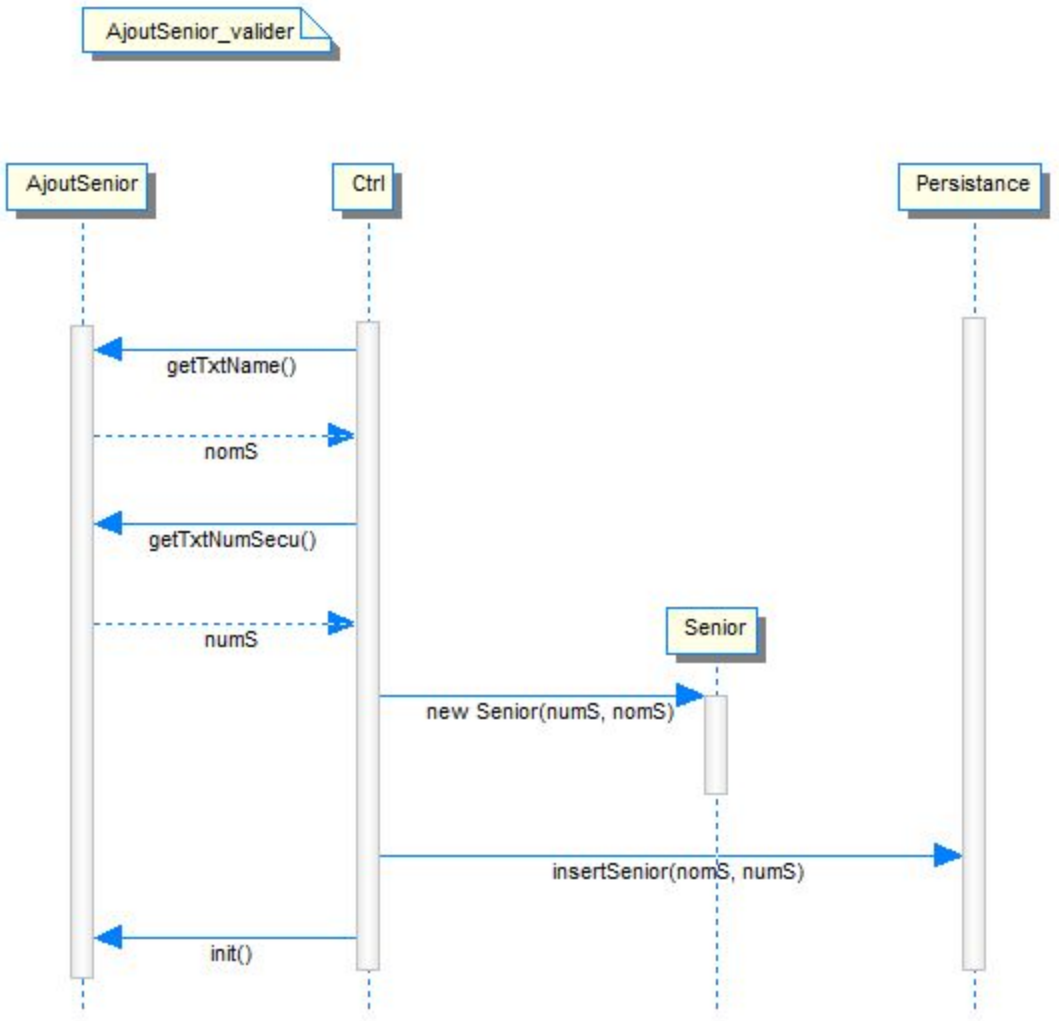


Diagramme de séquence du cas d'utilisation « Ajouter un nouveau sénior »