Coté cours – Application médicaments pour le laboratoire GSB

Bases de la POO Java : utiliser des objets, parcours d'ArrayList

Propriétés	Description
Intitulé long	Utilisation et exploitation d'une application Java Android à partir de données libres de droit
Formation concernée	BTS Services Informatiques aux Organisations
Matière	Bases de la programmation
Présentation	L'objectif est de comprendre l'utilisation et la manipulation d'objets à partir de tableaux dynamiques (ArrayList). Ces ArrayList d'objets sont construits en fonction de données issues d'une base de données SQLite.
Notions	D4.1 – Conception et réalisation d'une solution applicative A4.1.7 Développement, utilisation ou adaptation de composants logiciels Savoir-faire Programmer à l'aide d'un langage de programmation structurée Programmer en utilisant des classes d'objets fournies Utiliser un environnement de développement Savoirs associés Concepts de base de la programmation objet
Transversalité	U22- Mathématiques, algorithmique appliquée
Pré-requis	Base de la programmation structurée : variable, boucle, condition, fonction, tableaux Savoir instancier un objet et l'utiliser en Java
Outils	Environnement de développement intégré (IDE) Android Studio, git
Mots-clés	Initiation à la programmation orientée objet, objet, classe métier, classe technique, parcours d'ArrayList, utilisation de git
Durée	6 heures
Auteur(es)	Fabrice Missonnier, relecture Hervé Le Guern et Yann Barrot
Version	v 1.0
Date de publication	Mai 2019

L'application programmée lors de la mission précédente permet d'afficher des médicaments ainsi que leurs présentations (les différents conditionnements).

Dans les applications, les objets ne sont pas construits « en dur » dans le code source. Les développeurs utilisent différentes techniques de **persistance** soit en sauvegardant les objets dans des fichiers, soit en les stockant dans des bases de données locales ou sur des serveurs distants.

Dès le début du projet, il a été décidé de persister les données des médicaments dans la base de données relationnelle SQLite. La base a été installée en local sur le smartphone ou la tablette.

Quelques éléments techniques issus de la *daily scrum*

Avant d'attaquer votre mission, comme tous les jours, l'équipe projet se réunit 15 minutes pour discuter du travail effectué le jour précédent. Voici une partie de leur intervention.

Kévin

« J'ai programmé la semaine dernière le *mapping* objet-relationnel dans la classe `DAO`. Au lieu de créer des objets en dur dans le code comme on l'avait fait pour les tests, je les génère automatiquement à partir de la base de données. Par exemple, si tu veux récupérer un médicament dans la base, le code que je te fournis va créer un objet de type `Medicament` et tous les liens qui lui sont associés : ses présentations, ses dosages, ses avis et ses groupes génériques.

La classe DAO n'a pas beaucoup de méthodes programmées. Ses deux points d'entrée sont le médicament et la présentation. On peut rechercher, par exemple, un médicament par son code CIS. Je retourne ce médicament s'il existe. Tu peux aussi récupérer les 50 « premiers » médicaments de la base de données dans un tableau. C'est cette méthode qu'Anne-Lise a utilisée pour afficher les médicaments dans la maquette.

J'ai aussi essayé de charger les 14000 médicaments de la base de données dans un seul tableau de médicaments, mais vu qu'ils sont stockés en mémoire, l'application est tellement ralentie qu'elle plante. Son chargement dure plus de 10 minutes sur l'AVD. Donc c'est impossible de travailler comme ça. Il faut récupérer et construire au maximum une cinquantaine d'objets, pas plus.

Par contre, la classe `DAO` n'est pas facile à maintenir. Il faut faire attention en la modifiant.

Il y aurait la possibilité de programmer d'autres requêtes sur la base SQLite, mais je pars en vacances pendant 3 semaines en Polynésie. Je n'ai pas le temps de développer d'autres méthodes pour le moment, mais le code que je te fournis a été testé et il fonctionne. »

Anne-Lise

« Bon ça y est, ça a été compliqué, mais on est arrivé à fusionner nos branches avec Kevin avant qu'il parte en vacances. Le projet se trouve sur la branche *mission-fusionMView*.

J'ai passé du temps à comprendre le Material View. En fait, j'ai récupéré une partie du code source sur Internet et je l'ai adapté à notre projet. Je vous passe les détails techniques, mais il y a maintenant un module *materialview* dans l'application que je n'ai presque pas touché. C'est à partir de ce module que je génère les différents onglets de l'application. Tout est programmé dans le package *vue* du projet et dans la classe `MainActivity`.

En fait, je récupère des `ArrayList` d'objets métiers qui me sont fournis par le code de Kévin pour afficher les différentes cartes de *material view*. Ce n'est qu'une maquette mais je pense qu'on pourra arriver à travailler avec cette librairie pour notre application.

Il faudrait maintenant qu'on se focalise sur les besoins attendus par le client pour ce sprint : quelles recherches souhaite-t-il faire avec l'application, de quels onglets a-t-il besoin, etc. ? Pour l'instant, puisque Kévin part en vacances, on a créé une classe technique pour que tu puisses t'entrainer sur les recherches dans les tableaux. Tu pourras, ainsi, mieux voir comment les objets métiers fonctionnent. »

Etape 1 : Comprendre ce qui a été dit dans la daily scrum

Le premier objectif est de comprendre ce qu'Anne-Lise et Kévin vous ont expliqué dans la daily scrum : pourquoi a-t-on besoin de tableaux d'objets pour l'affichage ?

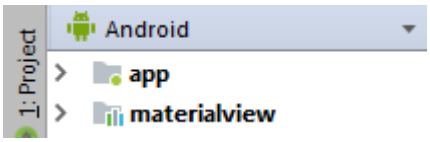
Anne-Lise a travaillé sur la branche *mission-fusionMView*.

1- Récupérer ce code source. Si le projet a déjà été cloné, il faut basculer sur cette branche (Menu *VCS/Git/Branches/Checkout As new local branch*).

Avant d'exécuter le projet, il est préférable de nettoyer le code déjà compilé dans les missions précédentes.

2- Sous Android Studio, exécuter la commande *Clean project* dans le menu *Build.*

Le projet devrait avoir deux modules distincts : *app,* notre application et *materialview*, le code récupéré par Anne Lise (issu du projet https://github.com/florent37/MaterialViewPager) :



Si jamais le deuxième module n'apparait pas (ou si le code ne compile pas), faire *File/Invalidate Caches/Invalidate and restart* sous Android Studio.

3- Supprimer l'application sur la machine virtuelle Android.

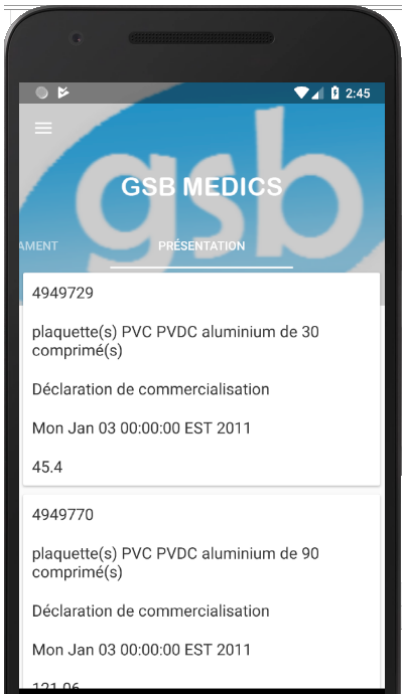
La vue (l'interface graphique) est programmée avec une technologie particulière, le *material design*
https://material.io/design/

Google a présenté cette couche d'interface pour la première fois en 2014. Elle a officiellement remplacé la couche *Holo*. **L'intérêt de cette technologie pour le projet est de pouvoir fournir des classes qui, à partir de tableaux d'objets métiers, génèrent automatiquement des cartes**. Par exemple, deux objets métiers instances de la classe `Medicament` seront affichés de la manière suivante :

Une **carte** représentant un objet de type `Medicament`

Une autre **carte** représente un autre objet de type `Medicament`

Lorsqu'on clique sur le bouton « afficher les présentations », l'onglet PRESENTATION met à jour les présentations du médicament sur lequel l'utilisateur a cliqué :



Une **carte** représentant un objet de type `Presentation` (objet lié au médicament « ANASTROZOLE »)

Une autre **carte** représentant un autre objet de type `Presentation` (un autre objet lié au médicament « ANASTROZOLE »)

Le code de l'application fonctionne de la manière suivante :

- La couche *material design* récupère les objets à afficher **sous la forme de tableaux d'objets**, instances des classes métiers `Medicament` et `Presentation`. La classe principale `MainActivity` utilise la classe technique `DAO` pour récupérer ces tableaux. Cette classe accède à la base de données SQLite et crée les tableaux. Elle fait partie de la couche « modèle » de l'application.

- La classe principale instancie ensuite les fenêtres de l'interface graphique. `MainActivity` va faire le lien entre les objets métiers qui seront créés et la vue qui affichera ces objets métiers.

- Les méthodes récupérant les tableaux de médicaments de la couche `DAO` pour les afficher dans la vue sont programmées dans la classe `FragmentStatePagerAdapterMedicament` (dans le package `vue/fragment`).

Pour pouvoir continuer à programmer l'application, il est donc indispensable de comprendre le fonctionnement des tableaux et des classes métiers. C'est l'objet de cette mission.

4- Ouvrir la classe `FragmentStatePagerAdapterMedicament` (dans le package `vue/fragment`). Parcourir la classe et aller sur la méthode :

```
@Override
public Fragment getItem(int position) {
```

Les tableaux de médicaments sont récupérés à partir de la classe technique `DAO`. Cette classe est instanciée dans un objet nommé `objetdao`. Pour utiliser une méthode définie dans la classe DAO, on procède, comme dans la mission précédente, avec la notation pointée :

```
ArrayList<Medicament> lesMedicaments = objetdao.getMedicamentsParNom("Paracetamol");
```

La méthode `getMedicamentsParNom` retourne un tableau dynamique (`ArrayList`) où sont stockés les objets de type `Medicament` qui ont Paracetamol dans leur nom ("Paracetamol" est un paramètre).

Elle retourne un `ArrayList` d'objets. Comme un tableau standard, un `ArrayList` permet de stocker n'importe quel type d'objet.

L'opérateur *diamant* < > correspond à la **notation paramétrée. Elle permet de spécifier le type d'objet stocké (ici des objets de type** `Medicament`**)** : `ArrayList<Medicament>`

5- Rechercher les médicaments ayant « ibuprofene » dans leur nom. Exécuter et vérifier que l'interface graphique les affiche bien.

La collection ArrayList est d'une grande souplesse par rapport à un tableau standard, particulièrement lorsqu'on ne connaît pas à l'avance le nombre d'éléments qu'elle va contenir. Sa capacité varie automatiquement en fonction des besoins.

L'ArrayList stocke les hash codes des objets du type donné. Voici un exemple d'algorithme de parcours d'un `ArrayList` :

```
// parcours de tous les éléments à l'aide d'un itérateur. mtmp est affecté,
// à chaque itération, à la valeur du hashcode d'un médicament stocké dans l'ArrayList

for (Medicament mtmp : lesMedicaments){
    Log.d( tag: "Test médicament ", mtmp.toString());
}
```

La boucle se lit de la manière suivante : « pour chaque objet de type Medicament stocké dans l'ArrayList, j'affecte cet objet dans une variable mtmp (et j'affiche mtmp) ». Le résultat sera l'affichage, dans la fenêtre log, de tous les médicaments contenus dans l'ArrayList :

```
04-25 03:44:25.263 5256-5256/com.gsb.javamedicaments I/Test médicament:
Medicament{codeCIS='62944693', denomination='SPASFON LYOC 80 mg, lyophilisat oral',
formePharma='lyophilisat' … }

04-25 03:44:25.264 5256-5256/com.gsb.javamedicaments I/Test médicament:
Medicament{codeCIS='65923640', denomination='SPASFON, suppositoire',
formePharma='suppositoire' …}

04-25 03:44:25.265 5256-5256/com.gsb.javamedicaments I/Test médicament:
Medicament{codeCIS='68081368', denomination='SPASFON, comprimé enrobé',
formePharma='comprimé enrobé',
…
```

6- Afin de vérifier que l'interface graphique d'Anne-Lise fonctionne bien, on souhaiterait afficher les médicaments qui ont « Paracetamol » dans leur nom dans la fenêtre de Log. Pour ce faire, on utilisera comme dans la mission précédente, la méthode `toString()` des objets. Attention, par défaut, la méthode `getItem()` est appelée deux fois, l'affichage dans le la console *log* sera donc double.

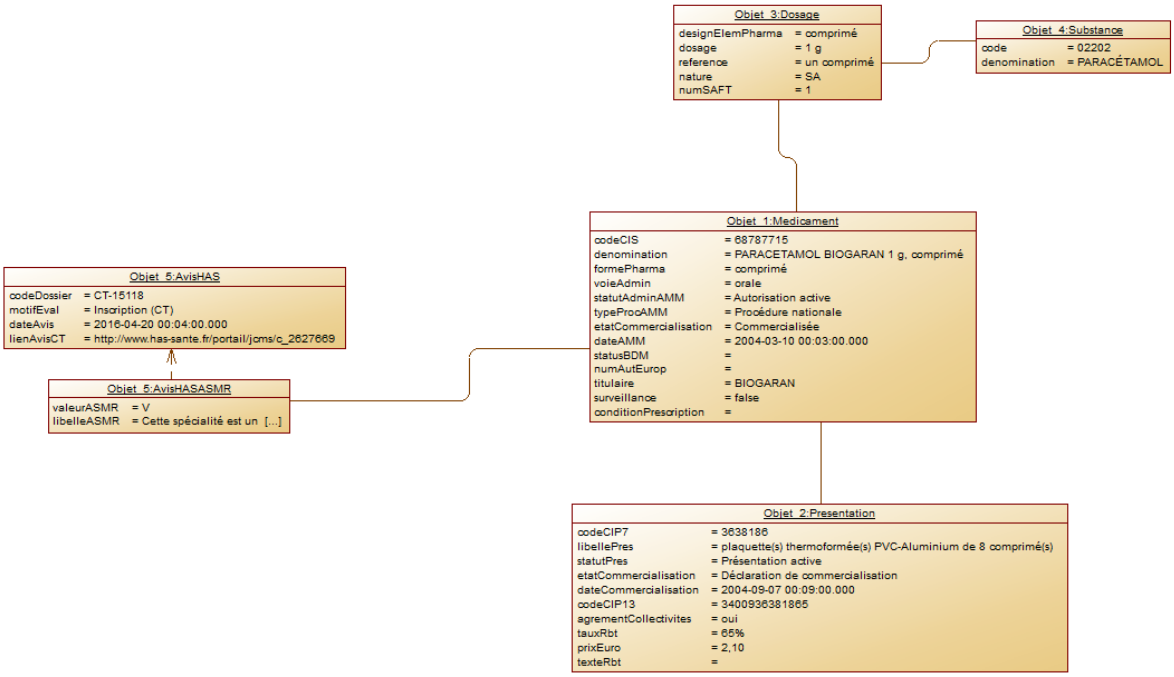
7- Effectuer ce même test en cherchant « Spasfon ».

8- Afficher dans le Log uniquement les codes CIS des médicaments en utilisant le bon accesseur.

9- Afficher dans le Log le code CIS ainsi que la dénomination des médicaments en utilisant les bons accesseurs.

Kévin a dit qu'il avait créé tous les liens entre le médicament et les autres classes qui lui sont associées.

Un diagramme d'objets UML permet d'illustrer les instances des classes :



Dans cet exemple, le médicament n'a qu'une présentation. Il pourrait en avoir plusieurs.

Techniquement, lors de l'exécution du code en Java, les 6 objets présents dans ce diagramme d'objet ont tous des *hash codes* (une « adresse ») différents en mémoire. **A la différence du modèle relationnel dans la base de données, les liens ne sont plus créés entre des clés primaires et des clés étrangères mais entre les adresses des objets** (voir mission précédente)**.**

Ainsi, lorsqu'on charge un objet de type `Medicament` à partir de la base de données, son code construit en profondeur un objet où seront stockés :

- ses présentations (ses conditionnements),
- ses avis de la Haute Autorité de Santé,
- ses dosages (quantité des différentes substances actives pour le médicament),
- ses groupes génériques.

Lors de l'instanciation des ArrayList de médicaments, seuls les hash codes de ces objets sont stockés dans le tableau dynamique.

10- Afficher dans le Log l'ensemble des présentations de tous les médicaments qui contiennent « Spasfon » dans leur nom. Il existe, dans la classe `Medicament`, un accesseur qui retourne un `ArrayList` de `Presentation`. L'algorithme sera le suivant

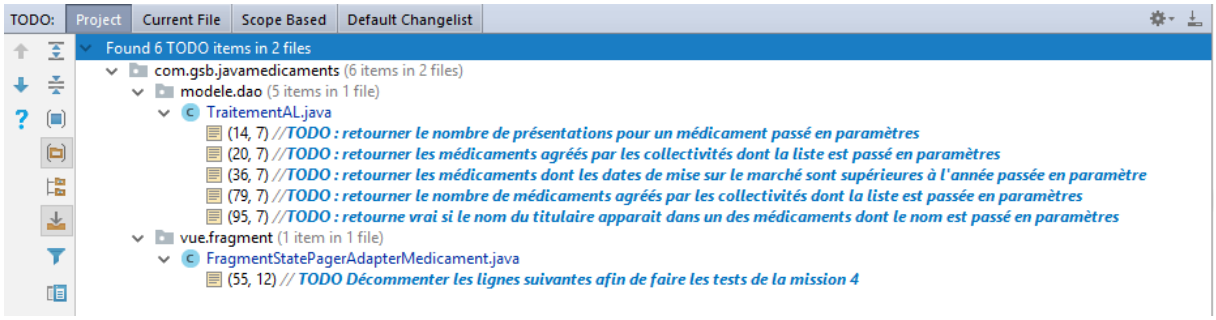
```
Pour chaque Medicament m dans l'ArrayList
      ArrayList<Presentation> lesPresentations = m.getLesPresentations() ;
      Pour chaque Presentation p dans lesPresentations
            Afficher p
      Finpour
Finpour
```

Etape 2 : Travailler sur les ArrayList

Anne-Lise a indiqué dans la daily scrum qu'elle avait créé une classe vous permettant de travailler sur les tableaux. Cette classe se nomme `TraitementAL` (pour *Traitement ArrayList*) dans le package `dao`.

Des exemples de code ont déjà été programmés dans la classe `TraitementAL` par Kevin. Les tests de ces méthodes se feront en commentant/décommentant leurs appels dans la classe `FragmentStatePagerAdapterMedicament`.

Pour mieux retrouver le travail à faire, Kévin a inséré des *tags TODO* que l'on retrouve dans la *TO DO List* dans Android Studio (onglet à côté de la fenêtre de Log) :



Les exemples suivants fournissent quelques instructions et algorithmes standards permettant de travailler sur les `ArrayList`.

```
//-----------------------------------------------
//Définition d'un ArrayList
//-----------------------------------------------
ArrayList <Medicament> lesMedicaments = new ArrayList() ;

//-----------------------------------------------
//Ajout d'un élément dans l'ArrayList
//l'adresse de l'objet m est stockée dans la collection
//-----------------------------------------------
Medicament m4 = new Medicament("61322336", "RIVASTIGMINE BIOGARAN 4,6 mg/24 h,
dispositif transdermique", "dispositif", "transdermique", ...);
lesMedicaments.add(m) ;

//-----------------------------------------------
//récupération de la taille de l'ArrayList
//-----------------------------------------------
int taille = lesMedicaments.size() ;

//-----------------------------------------------
//algorithmes standards de parcours d'ArrayList
//-----------------------------------------------

//parcours de tous les éléments à l'aide d'un itérateur
//mtmp prend, à chaque itération, la valeur de l'adresse d'un médicament
//stocké dans l'ArrayList
for (Medicament mtmp : lesMedicaments){
        //traiter mtmp
}

//recherche d'un élément dans le tableau (ici, l'exemple est la recherche d'un
//médicament par son code CIS

boolean trouve = false ;
while (i < lesMedicaments.size() && ! trouve){
        Medicament m = lesMedicaments.get(i) ;
        if (m.getCodeCIS().compareTo("60216885") == 0){
```

```
                    trouve = true ;
            }
            i++ ;
}
if(trouve){
            //traitement si trouvé
} else {
            //traitement si pas trouvé
}
```

11- Expliquer en détail le fonctionnement de la méthode `public double prixMoyenParNom(ArrayList<Medicament> lesMedicaments)` de la classe `TraitementAL`.

12- Comment peut-on tester si les valeurs retournées par cette méthode sont correctes ?

13- Compléter et tester les méthodes de la *TODO List*. Lorsque la méthode retourne un ArrayList de médicaments, le faire afficher dans la maquette d'Anne-Lise. Les appels effectifs des méthodes sont déjà programmés dans la classe `FragmentStatePagerAdapterMedicament`.