DOCUMENT DE DÉFINITION

KORALLIA FRENETTE  
WILLIAM LEMIRE

TRAVAIL PRÉSENTÉ À JEAN-CHRISTOPHE DEMERS  
DANS LE CADRE DU COURS **420-C61-IN – PROJET SYNTHÈSE**

TECHNIQUES DE L’INFORMATIQUE  
CÉGEP DU VIEUX-MONTRÉAL  
08/09/2022

Toutes les maquettes de l’interface graphique – détaillées

Conception UML

• Digramme(s) des cas d’usage – détaillé(s)

• Diagramme(s) de classes – détaillé(s)

Schéma(s) de la structure de données externe – détaillés

Éléments de conception

• Structures de données

La première structure de données que nous avons choisi avec une attention particulière, est un dictionnaire pour stocker les templates de produits, ainsi que les . Ce qui importe le plus est l’accès au template, peu importe l’ordre, donc la complexité de cette structure (ordre de (1)) va être optimale.

La deuxième structure de données est l’ArrayList, qui servira à contenir des objets Notes, Expenses, et Points. Nous avons choisi cette structure de données puisqu’elle peut dynamiquement s’agrandir, contrairement à une bonne vieille liste (array), mais surtout parce que nous devons parcourir ces listes fréquemment et qu’elles utilisent des itérateurs, qui sont optimisés pour faire cette opération.

Finalement, la troisième structure de données, et celle dont nous allons implémenter de toutes pièces, est une liste chaînée.

Graphical user interface, text, application, email

Description automatically generated

AL : Things we have to access by index 0(1) ++, if we have to remove things O(n) --,   
**it's actually a sensible choice when the number of reads is far more than the number of writes**.

LL: Access/remove by index --, adding (0(1)) **LinkedLists are more suitable when the addition rate is much higher than the read rate**.

Liste chaînée :

* Pour les produits pour stocker les matériaux les composants, insertions (si connait le point d’entrée) ordre(1). Puisque chaque nœud contient le prochain, parcourir la liste en entier est très rapide.

ArrayList :

* Non-syncronisé (plusieurs threads peuvent accéder, donc plus vite, contrairement à vecteur)
* Grossi dynamiquement
* ArrayList grossi de 50% au lieu de 100% quand dépasse son nb d’éléments, plus efficient en gestion de mémoire
* Using for Expenses, Notes, Points, pcq ces listes peuvent grossir (who knows how many will be added)
* Itérateur : optimisé pour itérer dans liste, insertion pas très rapide, mais pas grosses listes donc pas un problème

• Patrons de conception

Le premier patron de conception que nous avons décidé d’utiliser est le patron Builder

Builder :

* Permet créer objets avec toutes les possibilités de variables d’instance, permet à l’utilisateur de laisser certains champs vides
* Flexibilité : pas besoin de 1000 constructeurs
* Template

DAO :

* Permet d’encapsuler la communication entre le logiciel et les sources externes de données
* Permet aux autres classes de se concentrer sur leur logique, les DAO gèrent l’accès aux données nécessaires
* Aspect-oriented programming : chaque aspect (inventaire, connection, etc.) a son DAO pour accéder aux données qui lui est pertinent

MVC :

* Spring MVC
* Permet de le contexte du développement Web, d’encapsuler les différents aspects (vue, logique,)
* Contrôleur permet la non-communication entre la vue et la logique
* Aspect fondamental de Spring

• Expressions régulières

SKU : Pattern-building

Password/email :

• Algorithmes

Encryption pour mdp

• Mathématiques

Extrapolation

Encryption