Letellier Aodren – Monfort Jordan

CSR

Rapport de conception du projet de simulation d'un supermarché

M1 MIAGE - TP 2B

Encadrant : Cédric Tédeshi

2016-2017

Introduction

Ce projet a été réalisé en deux étapes distinctes, d'abord la création propre de la simulation de supermarché, puis l'intégration d'une API Rest permettant de contrôler et d'observer cette simulation.

I. Partie A : Simulation parallèle du Supermarché

Il a donc été demandé de créer un simulation de supermarché, c'est à dire, de simuler des clients faisant leurs courses dans un supermarché.

A partir d'une liste de course, ils récupéreront, dans les rayons du supermarché, les produits dont ils ont besoin et qu'ils pourront déplacer grâce a un chariot, qu'ils auront récupéré au préalable. Finalement ils pourront quitter le supermarché après avoir effectué leurs paiements.

Nous avons décidé de structurer notre programme de la forme suivante :

- Une partie correspondant aux éléments propres du supermarché (elle contient les éléments partagé des threads) et située dans le package « partieA.schema » :
  - La classe **Supermarche**, qui est la classe principale du programme, elle contient tous les éléments du schéma plus les employés du supermarché, et peut accueillir des clients. Elle sera en charge de « débuter une journée de course », c'est à dire de lancer les différents threads.
  - La classe **Chariot**, qui est l'élément partagé entre les Clients et qui correspond au stock de chariot du supermarché.
  - La classe **Rayon**, qui est un élément partagé entre les Clients et le ChefDeRayon et qui correspond à un rayon d'un seul produit.
  - La classe **Caisse**, qui est l'élément partagé entre les Clients et le Caissier, et qui contient un Tapis.
  - La classe **Tapis**, qui est l'élément partagé entre le Client déposant ses articles en caisse et le Caissier.
- Une partie correspondant aux acteurs de ce supermarché (ce sont les threads) et située dans le package « partieA.intervenants » :
  - La classe **Client**, qui est le thread représentant un client, sa liste de course est générée aléatoirement dans le constructeur de classe, et son comportement est prédéfini dans la méthode Thread.run().
  - La classe **ChefDeRayon**, qui est le thread représentant le chef de rayon, qui sera chargé de parcourir les rayons et des les remplir si nécessaire.
  - La classe **Caissier**, qui est le thread représentant le caissier, qui sera chargé de géré la caisse, c'est à dire de prendre en charge les produits déposés sur le tapis par les clients et de procéder à l'encaissement du paiement.

Le processus de synchronisation est donc réalisé via des méthodes appliquées sur les éléments partagés. Ces méthodes utilisées par exemple pour récupérer un chariot ou déposer un produit sur le tapis de caisse ont été rendues atomiques grâce au « *synchronized* ».

Les principaux problèmes de synchronisation rencontrés ont été lors de la création de la caisse, l'accès au tapis pour un client est seulement possible lorsque le booléen « *tapisDisponible* » du Tapis est vrai, sinon il est mis en attente et sera seulement réveillé lorsque le caissier aura récupéré le tag « client suivant » déposé sur le tapis par le client précédent.

De plus, le client en caisse et le caissier doivent se synchroniser pour effectuer le paiement, ainsi le Thread.notifyAll() de la méthode Caisse.parcourirTapis() permet non seulement de réveiller les clients en attente dans la caisse mais aussi de réveiller le client déjà présent en attente du caissier pour payer, ou en attente d'une place sur le tapis pour déposer le reste de ses produits.

Les threads en attente au niveau de la caisse seront réveillé seulement par le caissier en effet il n'y a que lui qui peut faire avancer le tapis et le vider.

II. Partie B : Intégration de l'api de contrôle et de visualisation Rest

L'intégration de l'api Rest dans le programme a été réalisée grâce à l'ajout de classes gérant les requêtes utilisateur, de classes gérant la mise en base de donné des clients et la transformation de la classe **Supermarche.java** en Application.

Cette partie se compose donc de :

- Classes ressources permettant de gérer les requêtes :
  - **ClientRessource :** permet de gérer les requêtes pour un seul client. Une requête GET permettant d'afficher sous forme de JSON le client désiré par l'utilisateur (qui aura soit cliqué sur le client voulu, soit écrit à la main dans l'URI l'idClient). Renvoie un message « *404 not found* » si le client n'est pas en base de données.
  - **ClientsRessource :** permet de gérer les requêtes GET et POST. GET affichera sous forme de JSON ou intégré dans le HTML la liste des clients. POST quant à elle permet de lancer x nombre de clients (défini par l'utilisateur) dans l'application Supermarche.
  - **StockRessource :** permet de gérer la requête GET qui affichera sous forme de JSON les stocks courants des différents rayons de l'application Supermarche.
  - **RootRessource :** permet de gérer la requête GET qui affichera la page HTML d'accueil du supermarché ou l'utilisateur pourra ajouter des clients.
- Classes gérant le **backend** de l'application, mettant en base de données les clients rentrant dans le supermarché dont le nombre est décidé par l'utilisateur.
- D'une classe **Main.java** pouvant lancer l'application Supermarche et l'API Rest.