CHAPON Elise
GHERAIA Maëlla

Rapport du projet CSR

Chargé de TP :
Mr BELKHIRIA Mehdi

Ce projet est divisé en deux parties. La première est une simulation multithreadée qui consiste à simuler le fonctionnement d'un restaurant en vue d'améliorer la fluidité de son service. Et la deuxième partie utilise l'API Rest afin de lister les clients avec leur état et retourner l'état du buffet.

Dans un premier temps, nous avons écrit quatre classes contenant chacune un bac du buffet. Dans ces classes, on peut observer deux concurrences. La première, lorsque qu'on accède au bac, une seule personne peut y accéder à la fois. La deuxième, lors de l'approvisionnement, l'employé de buffet ne peut pas accéder au bac si un client est présent et vice-versa. On a donc mis un synchronized. Dans la classe réapprovisionnement, il y a également un notifyAll() afin que l'employé prévienne qu'il ne s'occupe plus du bac.

Ensuite, nous avons écrit la classe Employé du buffet. Lorsque celui-ci démarre, on réapprovisionne chaque bac du buffet en utilisant les méthodes réapprovisionner de chaque classe de buffet. Puis, nous sommes occupés du stand de cuisson. Nous avons fait le choix de ne pas lancer les processus dans la classe StandCuisson mais plutôt dans la classe Restaurant. Dans le stand de cuisson, nous y avons mis les méthodes qui permettent de récupérer le nombre de client du stand, d'augmenter ce nombre dès qu'un client attend au stand et de le diminuer quand la cuisson est terminée.

Dans la classe Cuisinier, nous avons choisi de gérer la cuisson des ingrédients et le fait que le cuisinier puisse s'occuper que d'un client à la fois. Lorsque le processus cuisinier démarre, la fonction fait appelle à la méthode cuisiner qui dure 100ms.

Les deux grosses classes de notre projet sont les classes Restaurant et Clients. Dans la classe restaurant, nous avons lancé tous les processus. En effet, lors de la création d'un restaurant et du lancement de la méthode début(), les processus Employebuffet, Cuisinier et Client démarrent. Il y avait une concurrence ici car pour que le client rentre il faut qu'il y ait de la place dans le restaurant. C'est pourquoi, nous avons mis un wait() dans cette méthode et un notify() dans la méthode qui simule la sortie d'un client. Ça nous permet de simuler la file d'attente devant le restaurant et autorise un client un rentrer lorsque qu'il y a de la place seulement. Le restaurant gère donc l'entrée du client mais pour tout le reste, c'est la classe.

Lorsque le processus client démarre, celui-ci va voir tous les bacs du buffet. On avait un problème de concurrence car les clients ne peuvent pas se servir si la quantité qu'ils souhaitent est supérieure à la quantité disponible. Pour cela nous avons mis des synchronized dans les fonctions qui accèdent au buffet et des wait() afin que le client attendent s'il n'y a pas assez à manger (c'est ici que le notify de notre fonction reapprovisionner() dans chaque classe buffet sert). Afin d'éviter les erreurs, lorsqu'on appelle ces fonctions elles se trouvent dans des try{} catch{}. Après avoir accédé au buffet, le client va successivement aller au standCuisson, manger et sortir. On observera une dernière concurrence au standCuisson où un seul client peut être servi à la fois. C'est pour cela que la fonction standCuisson à un wait() tant que le nombre de client est supérieur à 1.

Concernant la partie Rest, Afin de récupérer le JSON qui représente l'état du buffet, nous avons créé la méthode getEtatBuffet. Dans cette méthode, nous créons un JSONObject où nous mettons les tailles de tous les bacs. Afin de pouvoir récupérer l'information dans le serveur, on a rajouté la route du buffet dans le MyRestaurantApplication.

Afin de récupérer le JSON représentant l'état du client, on récupère dans un premier temps l'id du client en question, puis on met l'état dans un JSONObject.