UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

CSR

Rapport du Projet Restaurant (TP5)

Groupe de TP : A1 - M1 Miage

Git Reposytory :

Réalisé par :

EL MAHJOUB Abdrahamane

SIMPARA Yaya

Encadrant :

Tedeschi Cédric

Mehdi Belkhiria

Architecture du projet

Notre projet comprend 4 packages

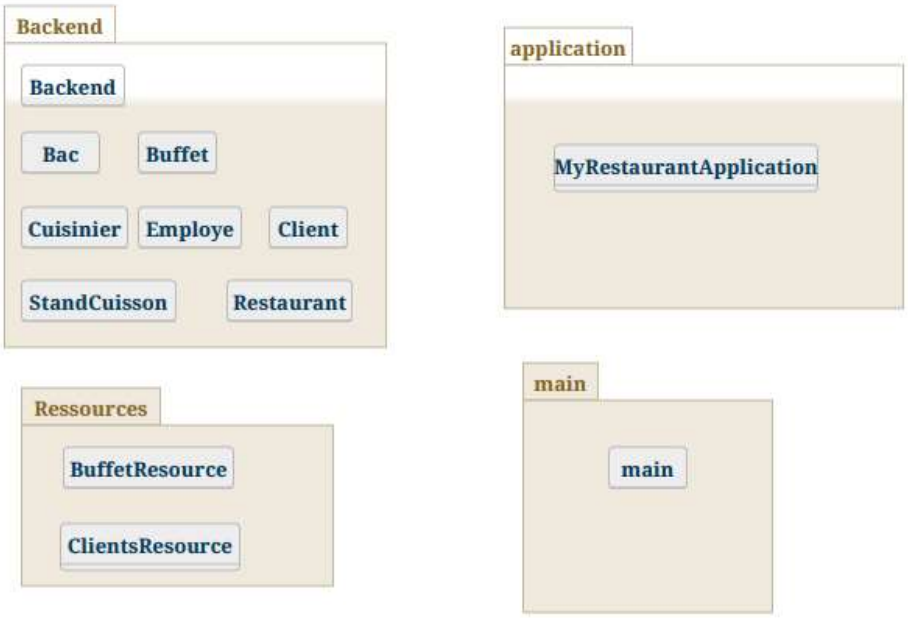

Figure 5. package csr Diagram

Le Package org.inria.restlet.TP5Resto.backend

Ce package est composé de sept classes qui sont :

- **La classe Bac :** représente un Bac et possède comme attribut une quantité qui indique la quantité du bac.
  Cette classe contient deux méthodes : la méthode **consommer (int param) qui** sera appelé par un **client** pour prendre une certaine quantité du bac et la méthode la méthode **remplir (int param)** appelée par l'employé du buffet pour remplir le Bac.

- **La classe Buffet** : représente un buffet contenant 4 bacs (bac poisson, bac viande, bac légumes et bac nouille) et des getter et setter pour chaque bac.

- **La classe Employe :** représente une entité active qui remplit chacun des Bacs s'il a une quantité inférieure à 1000 g

- **La classe StandCuisson** : Représente le stand de cuisson et possède deux méthodes : la méthode **deposserAssiette**() invoquée par le client pour faire cuire son plat, et la méthode **cuire()** invoquée par le cuisinier pour cuire le plat d'un client.

- **La classe Client** : Héritant de la classe Thread, cette classe représente le client possède comme attribut le restaurant et son id.

  Dans la fonction **run** le client essaie de rentrer dans le restaurant s'il n'y a plus de place libre il est mis en attente sinon il entre puis va au buffet pour prendre une certaine quantité dans chaque Bac, il est mis en attente si un autre client utilise le même Bac auquel il veut accéder ou si la quantité du Bac est insuffisante. Ensuite il se dirige vers le stand de cuisson pour déposer son assiette, attend la fin de la cuisson par le cuisinier, Mange et sort du restaurant.

- **La classe cuisinier** : représentant le cuisinier, cette classe fait appel à la méthode cuire du stand de cuisson, s'il n'y a aucun client le cuisinier est bloqué jusqu'à ce que un client le réveille pour faire cuire son plat.

- **La classe Restaurant :** Simule le comportement d'un restaurant : elle possède le Buffet, le stand de cuisson, le cuisinier, l'employé du buffet, le nombre place disponible comme attributs. La classe restaurant initialise chacun de ces attributs.

  Elle possède plusieurs méthodes :

  `lancer() :` Permet de lancer les threads clients, cuisinier et employé

  `entrerRestaurant() :` Invoquer par un client pour rentrer dans le resto, le client reste bloqué s'il n'y a plus de place disponible jusqu'à ce qu'une place se libère.

  `sortirRestaurant()` : Invoquer par un client lorsqu'il termine, libère une place.

  `Client getClient(int id)` : retourne le client ayant l'Id passé comme parametre

- La classe **Backend** : Pemet de lancer la simulation de notre restaurant, cette classe est appélé dans la Main

Le Package org.inria.restlet.TP5Resto.application

Dans ce package se trouve la classe MyRestaurantApplication dans laquelle est définie les URLS

Le Package org.inria.restlet.TP5Resto.main :

Contient le main permettant de lancer le server.

Le Package org.inria.restlet.TP5Resto.ressource :

Ce package contient les classes clientsRessources et BuffetRessource dans lesquelles contiennent les traitements à effectuer à la suite des requêtes

Les problèmes de synchronisations

Synchronisation entre clients

- A l'entrée et à la sortie du restaurant :
  Les clients rentrent un par un dans le restaurant quand le nombre de place disponible dans le restaurant est atteint les autres clients sont bloqués jusqu'à ce qu'un client libère une place, la libération d'une place fait entrer un client

```
public synchronized void entrerRestaurant() throws InterruptedException {
	while(nbPlace<=0) {
		wait();
	}
	nbPlace--;
}
public synchronized void sortirRestaurant() throws InterruptedException {
	nbPlace++;
	notifyAll();
}
```

l'entrée et la sortie des clients dans le restaurant se fait un par un, pour protéger la variable « nbplace » d'où le Synchronized devant les méthodes.

- Au niveau du Bac :
  Deux clients n'accèdent pas en même temps au même Bac d'où le synchronized devant la méthode consommer se trouvant dans le bac.
  Aussi lorsque le client veut prendre une quantité supérieur à la quantité du bac il est bloqué jusqu'à ce que le Bac soit approvisionné par l'employé .

```
public synchronized void consommer(int kiloconso) throws
InterruptedException {
```

```
            while(kiloconso > quantite ) {
                  wait();
            }
            int temps = 200 + ( (int)Math.random()  * (300-200) );
            Thread.sleep(temps);
            quantite= quantite-kiloconso;
            notifyAll();
      }
```

Synchronisation entre clients entre Employé du Buffet

- Au niveau du Bac :
  Le client et l'employé ne doivent pas accéder en même temps au même Bac d'où le synchronized devant la méthode remplirBac(int qte) invoquer par le l'employé et devant la méthode `consommer(int qte) invoquer par le client`

Synchronisation entre clients entre le cuisinier pour la cuisson

Nous avons utilisé deux sémaphores pour synchroniser le client et le cuisinier : le cuisinier reste bloquer jusqu'à l'arrivée d'un client qui le débloque, ce client se bloque également (après avoir débloqué le cuisinier) attendant à ce que le cuisiner lui délivre son plat. Plusieurs clients peuvent être bloqués attendant leur plat le cuisinier débloque l'un d'entre eux .

```
public synchronized void deposerAssiete() throws InterruptedException
      {
            nbAssiete++;
            semaCuisinier.release();//debloquer cuisinier
            semaClient.acquire();

            //  attendre son assiete
      }

      public void cuire() throws InterruptedException
      {
            semaCuisinier.acquire();//attend client
            Thread.sleep(1000); // temps de preparation
            nbAssiete--;

            semaClient.release();// debloque un client

            }
```