Laboratoire 8 – Files

ITI 1521. Introduction à l'informatique II

Semaine 22-26 Mars 2021

Dû en ligne après une semaine de votre Lab

/10

- ## Objectifs
  • **Utiliser** des files afin de résoudre des problèmes

Introduction

Pour ce laboratoire, nous développons une modélisation des files d'attente pour un supermarché ayant des files rapides ainsi que des files normales (régulières). Les files d'attente seront modélisées à l'aide de files (objets réalisant l'interface **Queue**).
Il y a donc une interface, nommée **Queue**, ainsi qu'une implémentation simple, nommée, **ArrayQueue** qui vous **sont fournies**.
Pour l'écriture des programmes, il faut faire certaines suppositions.
• Il s'agit d'une modélisation où le temps a été discrétisé. Ainsi, une modélisation est constituée d'un nombre fixe et prédéterminé de périodes. Au début, la valeur de l'horloge est mise à 0, par la suite, pour chaque itération la valeur de temps est augmentée d'une quantité fixe (ITER) ;
• Pour chaque itération, un ou plusieurs des événements suivants peuvent survenir :
– Un nouveau client joint la file. La probabilité de l'arrivée d'un nouveau client est **PROBABILITY_NEW_CLIENT**. Le nombre de produits achetés suit une distribution uniforme dont l'intervalle est 1 . . .**MAX_TEMS**;

– Un caissier traite exactement un item par itération (sauf si sa file est vide).

Le contenu de ce laboratoire est adapté du livre Lambert et Osborne (2004) *Java : A Framework for Program Design and Data Structures*. Brooks/Coles, pages 266–274.

Voici la description des classes de la simulation.

1. Classe `Client` :
La classe **`Client`** doit modéliser un client. Un client mémorise le moment de son arrivée, le nombre initial d'articles, ainsi que le nombre d'articles qui n'ont pas encore été traités par le caissier. Le nombre maximal d'articles est le même pour tous les clients (**MAX_ ITEMS**).

• La **variable de classe** est donc:
private static final int MAX_ITEMS = 25;

• Les **variables d'instances** sont :

```
private int arriveTime;
private int Items;
private int initialItems;
```

• Les **méthodes d'instance** suivantes :
- **int getArriveTime()** retourne l'heure d'arrivée ;
- **int getItems()** retourne le nombre d'articles à traiter ;
- **int getItemsDone ()** retourne le nombre total d'articles traités ;
-**serve()** décrémente de 1 le nombre d'articles de ce client.

L'unique constructeur de cette classe n'a qu'un paramètre. Ce dernier spécifie l'heure à laquelle le client s'est joint à la file. Le nombre initial d'articles est déterminé à l'aide des instructions suivantes :

```
Items =  (int) ( ( MAX_ITEMS - 1 ) * Math.random() ) + 1;
initialItems = Items;
```

pour s'assurer qu'il y ait au moins un item.

2. Classe `Cashier`

Un caissier traite une file de clients. Il sert un client à la fois. Puisque le but de cette modélisation est d'estimer la valeur de certains paramètres, tel que le temps moyen d'attente, le caissier doit aussi mémoriser certaines données telles que le nombre de clients servis, le temps total passé dans la file d'attente, ainsi que le nombre total d'articles traités.

• La variable de classe est:

```
private static final String str = System.getProperty("line.separator" );
```

(Où line.separator : Sequence used by operating system to separate lines in text files)

• Les **variables d'instance** sont :

```
private Queue<Client> queue;
private Client currentClient;
private int ClientTime;
private int ClientsServed;
private int ItemsDone;
```

Cette classe n'a qu'un constructeur sans paramètres. Il sert à initialiser les variables d'instance de ce caissier.

Le comportement du caissier est comme suit :
• Si le cassier n'a pas de client courant, le prochain client en file devient le client courant, sauf si la file est vide, dans ce cas, le caissier sera inactif pour cette itération. Lorsqu'un client est retiré de la file, il faut comptabiliser le temps total passé à attendre ;
• Le caissier traite un article (appel à la méthode **serve()** du client courant) ;
• Si tous les articles du client courant ont été traités, le caissier comptabilise le nombre total d'articles traités. Il laisse son client partir (ce caissier n'a plus de client courant).

• Les **méthodes d'instance** sont donc:
- ***add(Client client)*** ajoute un client à l'arrière de la file.
- ***int getQueueSize()*** retourne le nombre de clients qui attendent présentement.
- ***servedClients(int currentTime)*** est appelée exactement une fois par itération. Le paramètre **currentTime** servira à déterminer le temps total qui s'est écoulé depuis que le client a joint la file.

Les méthodes d'instances additionnelles suivantes donnent accès aux statistiques de ce caissier :
- ***int getClientTime*()** donne accès au temps total passé en file
- ***int getItemsDone()*** donne accès au nombre total d'articles traités
- ***int getClientsServed()*** donne accès au nombre total de clients servis
- ***String toString()*** retourne une chaîne de caractères qui présente les statistiques accumulées par ce caissier (à compléter dans les parties indiquées ci-dessous).

```
public String toString() {
      StringBuffer out = new StringBuffer();

      out.append( "The total number of clients served is " );
      out.append( ClientsServed );
      out.append( str );

      out.append( "The average number of items per client was " );
          ...À COMPLETER ICI

     out.append( "The average time (in seconds) was " );
          ...À COMPLETER ICI

       return out.toString();
  }
```

3. Classe `Modeling` fournie et qui contient le programme principal `main`:
La classe **Modeling** organise la modélisation. Un objet **Modeling** à deux caissiers, l'un est responsable de la file rapide (*fast*), l'autre s'occupe d'une normale. Cet objet mémorise aussi la durée de la modélisation. Le constructeur crée les deux caissiers nécessaires.
La méthode **test()** réalise la boucle principale de la modélisation. Au début, l'horloge est mise à 0, pour chaque itération, l'horloge avance d'une quantité fixe (ITER).
À chaque itération, la méthode **test()** doit :
• Déterminer si un nouveau client est arrivé, et si oui, le placer dans la bonne file, selon le nombre d'articles achetés ;
• S'assurer que tous les caissiers traitent un client ;
• Incrémenter la valeur de temps.
À la fin de la modélisation, la méthode *display* affiche les statistiques.

Question 1 : (8 POINTS : 4 POINTS pour chaque classe (`Client et Cashier`)

Écrire vos programmes, classes `Client` et `Cashier`. Définir les méthodes de chaque classe. Exécutez la simulation plusieurs fois, et discutez vos observations.

/*Exemple de sortie*/
The duration of the modeling was 28800 seconds
FAST LINE :
The total number of clients served is 343
The average number of items per client was 6
The average time (in seconds) was 12

NORMAL LINE :
The total number of clients served is 313
The average number of items per client was 18
The average time (in seconds) was 2556

Question 2 : (2 POINTS : 0.5 POINTS pour chaque méthode de la classe `Set`)

On constate que le temps moyen d'attente pour la file rapide est très court (16 secondes !). Et, le temps d'attente à la file normale est très long (près de 45 minutes !). Le gérant du supermarché devra ajouter de nouvelles files, mais combien ? Afin de l'aider, vous devez modifier ces classes afin d'y ajouter des caisses normales. Les classes **`Client` et `Cashier`** restent les mêmes qu'avant. La nouvelle classe **`Modeling2`** qui contient le programme principal **`main` est fournie.**

Vous devez ajouter une classe **`Set`** qui représentera un ensemble de caisses.

• La variable de la classe `Set` est:
private static final String str = System.getProperty("line.separator");

• La **variable d'instance** de `Set` est:
private Cashier[] set;

Cette classe n'a qu'un constructeur avec un seul paramètre (nbr). Il sert à initialiser les variables d'instance, et lève une exception de type *IllegalArgumentException*(Integer.toString(nbr)) si l'argument *nbr* est plus petit que 1.

• Les **méthodes d'instance** de `Set` sont:
- ***add(Client client)*** ajoute un client à l'arrière de la file la plus courte.
- ***servedClients(int currentTime)*** est appelée exactement une fois par itération. Le paramètre **currentTime** servira à déterminer le temps total qui s'est écoulé depuis que le client a joint la file d'une des caisses.
- **String toString()** retourne une chaîne de caractères qui présente les statistiques accumulées par chaque caissier.

Écrire votre classe `Set` et définir ses méthodes.

Tester votre programme en utilisant le programme fourni **`Modeling2`**.

/*Exemple de sortie*/
The duration of modeling was 28800 seconds
FAST LINES :
LINE 0 :
The total number of clients served is 354
The average number of items per client was 6
The average time (in seconds) was 15

NORMAL LINES :
LINE 0 :
The total number of clients served is 247
The average number of items per client was 18
The average time (in seconds) was 44

LINE 1 :
The total number of clients served is 109
The average number of items per client was 19
The average time (in seconds) was 34

Créer et soumettre un fichier zip comme d'habitude avec les programmes suivants

- ✓ ***`Client`****.java*
- ✓ ***`Cashier`****.java*
- ✓ ***`Set`****.java*