Master 2 mention Informatique

IL

Rapport de AOC

Réalisé par :

David Lafia-Monwoo

Brandon Largeau

Professeurs :

Noël Plouzeau

Janvier 2021

Objectifs et contexte

L'objectif de ce TP est de réaliser un service de diffusion de données de capteur. Il s'agit de mettre en place un mécanisme sûr de simulation de réception des données qui proviennent d'une entité distante (capteur) par des entités abonnées (afficheurs), en se basant sur la mise en œuvre parallèle d'Observer. Les données diffusées seront une séquence croissante d'entier. Le compteur sera incrémenté à intervalle fixe.

Architecture du projet

Pour le projet nous avons décidé d'utiliser Gradle, ce qui nous permettait d'avoir une même base de projet, avec les mêmes versions de dépendance et même version de Java, ici la version 11. Nous avons aussi automatisé certains processus comme le formatage du code, réorganisation des imports, suppression des imports inutiles, copie du copyright sur chaque fichier du code source, etc.

Avant de charger le projet sur Eclipse :

gradle build

ou (*gradlew* ou *gradlew.bat* sur Windows)

gradlew build

Choix d'implémentation

Pour la mise en œuvre, nous avons respecté l'architecture globale du PC ActiveObject.

CLASSE PRINCIPALE

La class principale est la class App.java qui contient la boucle dans laquelle les ticks sont exécutés, et dans laquelle sont instanciés l'algorithme de diffusion désiré, le capteur, ExecutorService, les 4 afficheurs, les 4 canaux, un nombre prédéfinis pour les testes. Les canaux sont liés au capteur qu'ils observent via la méthode attach() du capteur. Chaque canal est instancié avec l'ExecutorService et l'afficheur avec lequel il est censé communiquer.

DIFFUSION DE L'INFORMATION

Pour réaliser les différents algorithmes de diffusion nous avons fait en sorte que chaque algo ai une référence vers le capteur qui appelle sa méthode execute ainsi il a accès aux différents canaux de communication. Un semaphore avec un nombre de token égale au nombre de canaux pour les algo de diffusion atomique et de diffusion séquentielle parce qu'ils sont thread safe et moins gourmand que d'utiliser une liste comme vu en TD . Dans ces deux algo une fois le execute() appelé tous les tokens disponibles sont pris(nombres de tokens disponible est égale au nombre de canaux qui observent le capteur qui égale au nombre

d'afficheur), la valeur du capteur est enregistrée dans une variable verrouillée, une référence du capteur est settée sur tous les canaux et la méthode update est appelée sur tous les canaux du capteur. La méthode update de l'afficheur est donc schédulée avec en paramètre le canal de communication, en un temps aléatoire compris entre 0 et 2s. Une fois le temps écoulé l'afficheur fais un getValue sur le canal qu'il a reçu en paramètre. Le canal schedule le getValue du capteur avec un temps aléatoire compris entre 0 et 2s également. L'afficheur fait ensuite un get() sur le future pour récupérer la valeur (bloquant le temps que ça arrive). On stock ensuite la valeur dans ce qui nous permettra de faire l'oracle. Pour le causale, comme c'est chaotique, on vérifie si la dernière valeur enregistrée est bien inférieure à la nouvelle car on ne veut que des valeurs successives croissantes. On fait un synchronize pour éviter les problèmes de concurrence et d'édition d'une même valeur au même moment par plusieurs threads à la fois. Des traitements particuliers sont effectués quand l'algo de diffusion est atomique ou séquentielle. Dans ces deux cas une méthode releaseLock() qui permet de notifier qu'un getValue() vient de se terminer et donc de release un token. Tous les token seront libérés une fois que le nombre de getValue terminé sera égale au nombre d'afficheur.

RÉSULTATS OBTENUS AVEC L'ORACLE POUR LES TESTS

Les tests (oracle) prennent environ 1min 30s au total avant de se terminer et nous obtenons les résultats suivant (par exemple) :

Les résultats sont affichés dans la console (dans Eclipse).

Algorithme de diffusion atomique :

- Afficheur 1 : 1 2 3 4 5 6
- Afficheur 2 : 1 2 3 4 5 6
- Afficheur 3 : 1 2 3 4 5 6
- Afficheur 4 : 1 2 3 4 5 6

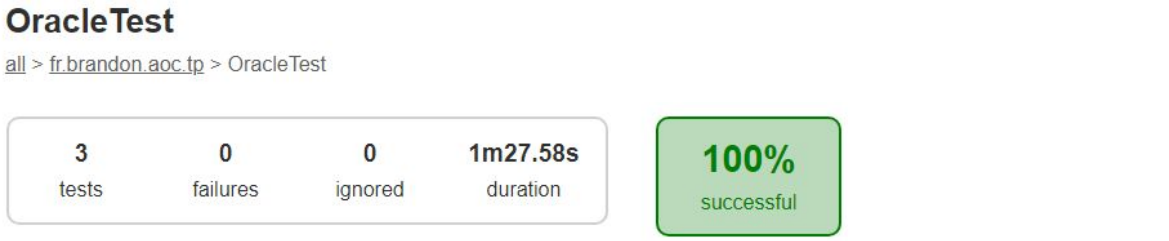
Algorithme de diffusion séquentielle :

- Afficheur 1 : 1 5 9 15 17 21
- Afficheur 2 : 1 5 9 15 17 21
- Afficheur 3 : 1 5 9 15 17 21
- Afficheur 4 : 1 5 9 15 17 21

Algorithme de diffusion causal :

- Afficheur 1 : 1 3 8 9 12 15
- Afficheur 2 : 1 2 4 7 9 14
- Afficheur 3 : 2 4 7 9 13 16
- Afficheur 4 : 1 3 5 9 12 14

Les résultats des tests sont visibles dans le dossier **test**.
Les 3 tests passent correctement :



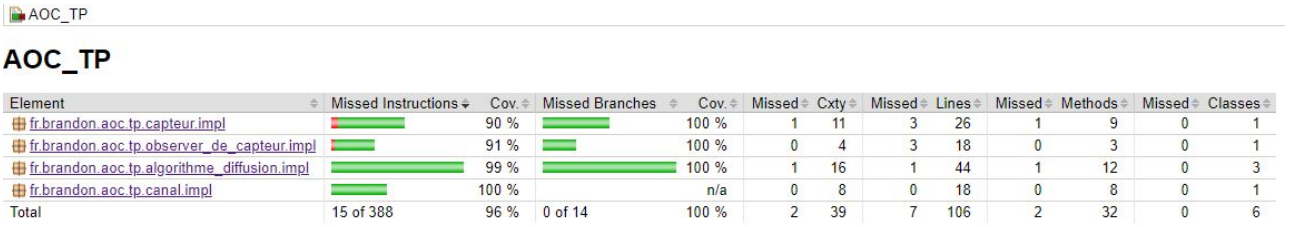
Tests Standard output

Test	Method name	Duration	Result
Test oracle algorithme atomique	Oracle_Algorithme_Atomique()	27.589s	passed
Test oracle algorithme causal	Oracle_Algorithme_Causal()	29.673s	passed
Test oracle algorithme sequentielle	Oracle_Algorithme_Sequentielle()	30.321s	passed

Voici un exemple d'output après avoir lancé l'oracle :

```
START - Algorithme sequentielle
Afficheur 1 -------------
6 13 20 26 33

Afficheur 2 -------------
6 13 20 26 33

Afficheur 3 -------------
6 13 20 26 33

Afficheur 4 -------------
6 13 20 26 33

END - Algorithme sequentielle
START - Algorithme atomique
Afficheur 1 -------------
1 2 3 4 5 6

Afficheur 2 -------------
1 2 3 4 5 6

Afficheur 3 -------------
1 2 3 4 5 6

Afficheur 4 -------------
1 2 3 4 5 6

END - Algorithme atomique
START - Algorithme causal
Afficheur 1 -------------
2 3 4 5 6 8 10 11 12 14

Afficheur 2 -------------
2 4 6 7 8 10 11 12 13 15

Afficheur 3 -------------
3 4 6 8 10 11 12 13 14 15

Afficheur 4 -------------
2 3 5 6 7 9 10 11 13 15

END - Algorithme causal
```

Il est aussi possible de voir le coverage de jacoco via l'oracle dans le dossier **jacoco**.



AOC_TP

AOC_TP

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
fr.brandon.aoc.tp.capteur.impl		90 %		100 %	1	11	3	26	1	9	0	1
fr.brandon.aoc.tp.observer_de_capteur.impl		91 %		100 %	0	4	3	18	0	3	0	1
fr.brandon.aoc.tp.algorithme_diffusion.impl		99 %		100 %	1	16	1	44	1	12	0	3
fr.brandon.aoc.tp.canal.impl		100 %		n/a	0	8	0	18	0	8	0	1
Total	15 of 388	96 %	0 of 14	100 %	2	39	7	106	2	32	0	6

La Javadoc est disponible dans le dossier **javadoc** sur le root.

Diagrammes

Les diagrammes sont un peu trop conséquent pour être affichés ici, ils sont disponible dans le dossier **diagrammes** au format drawio, PDF et PNG :

- M3_Diagram_Class_GetValue
- M3_Diagram_Class_Update
- M3_Diagram_Sequence_1
- M3_Diagram_Sequence_2
- M3_Diagram_Sequence_3
- M3_Diagram_Sequence_4