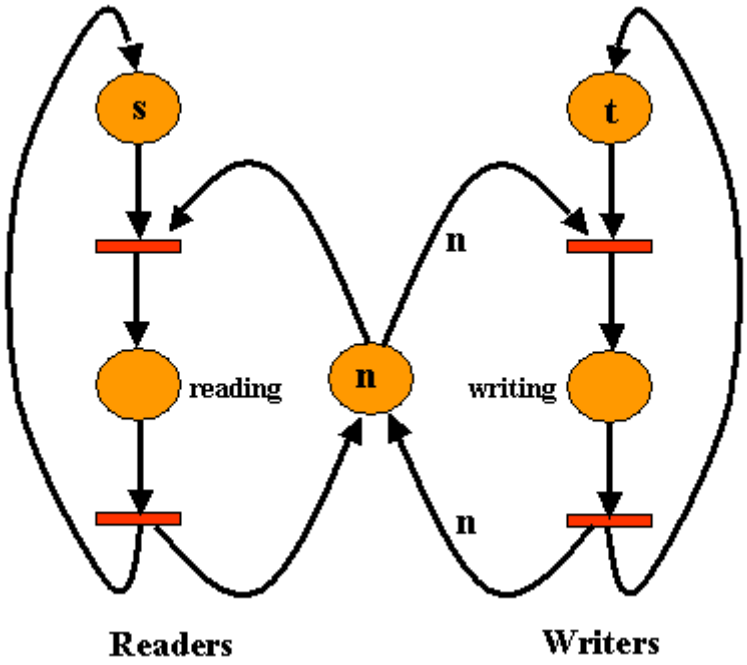
Laboratoire 1- Monitoring

Paradigmes de programmation avancés



Auteur : Timothy Peck
Classe : ISC3il-a
Cours : Paradigmes de programmation avancés II
Professeurs : Mme. Rizzotti Aicha et Mr. Senn Julien

Table des matières

1. Introduction

Dans le contexte du cours de paradigmes de programmation avancés, il nous est demandé de réaliser un programme utilisant le monitoring en Java avec des lecteurs rédacteurs.

Il était demandé de s'occuper des entrées utilisateurs, pour le nombre de lecteurs, de documents, et pour continuer ou quitter le programme. Le nombre de lecteur étant égal au nombre de threads.

Avec cela il fallait mettre en place un système de monitoring des threads pour que leurs états et actions actuels soient connus puis de terminer le programme quand cela est atteint.

Il était également demandé de gérer l'écriture aux documents, afin de garantir qu'un lecteur ne le lise pas pendant qu'un rédacteur le modifie.

Une vidéo de démonstration était fournie, montrant le fonctionnement du programme, ainsi que son affichage.

Un squelette de code contenant les classes et fonctions nécessaire était fournie, ainsi qu'un document explicatif des classes.

2. Implémentation

1. Entrées utilisateurs

Avant de pouvoir créer et lancer les threads, il fallait demander la quantité de threads voulu a l'utilisateur. Ceci a été fait grâce aux « `Scanner` » de Java. Le Scanner permet de lire une ligne et de le convertir directement dans le bon type.

```
Scanner sc = new Scanner(System.in);
```

Ceci a été fait dans une boucle pour vérifier que le nombre de lecteurs et de documents soit dans les limites définit dans le cahier des charges.

```
do {
  System.out.print("Nombre de Personnes :");
  nbPersons = sc.nextInt();
} while (nbPersons > 9 || nbPersons < 1);
do {
  System.out.print("Nombre de Documents :");
  nbDocuments = sc.nextInt();
} while (nbDocuments > 9 || nbDocuments < 1);
```

Cette même méthode, sans la boucle de vérification, a été utilisée pour gérer le continuation et l'arrêt du programme.

```
System.out.println("Please press N/n to continue or E/e to exit the program");
String input = sc.nextLine();

if (input.toLowerCase().equals("e")) {
  consoleTask.cancel(true);
} else if (input.toLowerCase().equals("n")) {
  waitingLogger.popNextLog();
  if (waitingLogger.isDone()) {
    consoleTask.cancel(true);
  }
} else {
  System.out.println("Invalid input");
}
```

2. Lecture et écriture dans le document

Pour éviter que plusieurs personnes accèdent en écriture sur le document en même temps, il faut implémenter un système de verrous. Dans ce cas, le verrou utilisé était un « `ReentrantReadWriteLock` » avec le paramètre « fair » mis a vrai. C'est le verrou idéal a ce cas de figure.

```
this.lock = new ReentrantReadWriteLock(true);
```

Lors de la lecture, le verrou de lecture *ReadLock* est verrouillé, quand le lecteur a fini sa lecture, le verrou est déverrouillé pour le prochain lecteur, ceci évite simplement qu'un rédacteur modifie le document pendant sa lecture. Ceci est également le cas avec le *WriteLock* lors de l'écriture, mais celui-ci verrouille la lecture et l'écriture.

```
lock.readLock().lock();
String value = new String(this.content);
lock.readLock().unlock();
```

Ces locks sont exclusifs a chaque document, donc on le limite pas l'ecriture sur un document quelconque pour écrire dans un autre.

3. Logs

Les logs gardent une trace du fonctionnement du programme, quand l'utilisateur entre « n » on récupère les logs de lecture/écriture et on les affiche. Les logs montrent les 4 états possible :

- R : en train de lire
- W : en train d'écrire
- F : fini
- T : lit et écrit dans la même dixième de seconde

Ces logs affichent également les personnes et leurs rôles.

```
## Threads : ##

 ~ Thread 1(WRITER) reading Document 1 at time 4400 for 4000ms
 ~ Thread 2(WRITER) reading Document 1 at time 1600 for 2200ms
 ~ Thread 3(READER) reading Document 3 at time 1300 for 1600ms
 ~ Thread 4(WRITER) reading Document 1 at time 1000 for 1500ms
 ~ Thread 5(READER) reading Document 2 at time 4500 for 3900ms
```

Cet affichage montre également qui a fait quelle action à quel moment.

```
                  0       1       2       3       4       5       6       7       8       9
Thread 1(WRITER) :                                    T---------------------------------------F
Thread 2(WRITER) :               T--------------------F
Thread 3(READER) :             T--------------F
Thread 4(WRITER) :           T--------------F
Thread 5(READER) :                                     T---------------------------------------F
```

Pour pouvoir comparer les documents et personnes, il a fallu hériter de la classe *Comparable<T>* sur les classes document et person puis implémenter la fonction *compareTo*.

3. Conclusion

Ayant déjà les squelettes de classes, il était assez simple de prendre en main ce projet. La vidéo donnait également une aide pour savoir quoi viser comme affichage et fonctionnement du programme. La possibilité de faire une interface graphique était présente et proposée, mais vu les limitations de temps cela n'as pas été implémenté.

Cependant, tout ce qui était demandé a été fait avec l'aide des verrous. Malgré le fait que le but de ce projet soit le monitoring, l'aspect qui a pris le plus de temps et a fallu le plus de travail était l'affichage.

4. References

projectsgeek. Readers-Writers-Problem-using-Semaphores. *projects geek website.* [Online] [Cited: 21 04 2023.] https://images.projectsgeek.com/2011/04/Readers-Writers-Problem-using-Semaphores.gif.