3259.1 Paradigmes de program. avancés II– Rapport technique – ISC3il-b

|  |
| --- |
| **Laboratoire 1** |

|  |
| --- |
| Étudiants participant à ce travail :  **Nicolas Aubert, ISC3il-b**  Présenté à :  **Aïcha Rizzotti**  Restitution du rapport : **21.07.2023**  Période : **2023**  École : **HE-Arc, Neuchâtel** |

**Framework pour le monitoring de la concurrence en Java**

Table des matières

[1 - Glossaire 2](#_Toc132732514)

[2 - Introduction 3](#_Toc132732515)

[2.1 - Contexte 3](#_Toc132732516)

[3 - Analyse 4](#_Toc132732517)

[4 - Conception 5](#_Toc132732518)

[5 - Réalisation 6](#_Toc132732519)

[6 - Résultats 7](#_Toc132732520)

[7 - Limitations et perspectives 8](#_Toc132732521)

[8 - Conclusion 9](#_Toc132732522)

[9 - Annexes I](#_Toc132732523)

[9.1 - Guide utilisateur I](#_Toc132732524)

[9.2 - Cahier des charges II](#_Toc132732525)

[9.3 - Table des illustrations III](#_Toc132732526)

[9.4 - Bibliographies et références IV](#_Toc132732527)

[9.4.1 - Sites Web IV](#_Toc132732528)

[9.4.2 - Livres IV](#_Toc132732529)

[9.4.3 - Autres IV](#_Toc132732530)

1. Glossaire

|  |  |
| --- | --- |
| **Database** | Une classe qui représente la base de données de documents à traiter par les threads, et offre des méthodes pour initialiser la base de données, obtenir un document aléatoire et obtenir la liste des noms de documents stockés. |
| **Log** | Une classe qui représente une action concurrente à afficher, avec un type (WAITING, REMOVE ou FINISHED) et une référence à la personne concernée. |
| **Main** | Une classe qui représente le thread principal gérant le bon déroulement et le démarrage de l'application, avec une FutureTask contenant le déroulement du programme pouvant être arrêté sur demande, et une méthode pour générer une population de personnes avec des propriétés aléatoires et attribuer à chacune un document aléatoire de la base de données. |
| **Paradigme Lecteurs-Rédacteur** | Un modèle de synchronisation dans la programmation concurrente où plusieurs threads peuvent lire simultanément une ressource partagée, mais seulement un thread à la fois peut écrire (rédiger) sur cette ressource. |
| **Person** | Une classe qui représente à la fois les lecteurs et les rédacteurs, implémente l'interface Runnable pour être exécutée comme un thread, et offre des méthodes pour gérer l'activité du thread, la pause et la reprise du timer, et le temps passé à traiter le document. |
| **ReentrantReadWriteLock** | Une classe Java qui implémente le paradigme lecteurs-rédacteurs en fournissant un verrou réentrant permettant à plusieurs threads de lire simultanément une ressource partagée, mais n'autorisant qu'un seul thread à la fois à écrire sur cette ressource. |
| **Singleton** | Un modèle de conception en programmation où une classe ne peut avoir qu'une seule instance dans tout le programme, avec un point d'accès global à cette instance. |
| **Timer** | Une classe qui garde une trace du temps passé dans l'application de manière générale et non par thread concurrent. |
| **WaitingLogger** | Une classe qui stocke l'état actuel des files d'attente des threads sur les objets de synchronisation, et offre des méthodes pour ajouter, supprimer et traiter les logs de files d'attente. |

1. Introduction
   1. Contexte

Le monitoring de la concurrence est un enjeu majeur dans le développement d'applications multi-threadées en Java. Dans ce contexte, nous avons été chargés de concevoir un framework pour le monitoring des files d'attente sur les éléments de synchronisation, en utilisant le pattern Lecteurs-Rédacteurs. Ce rapport présente notre solution pour visualiser les files d'attente des différents objets de synchronisation, où des lecteurs et des rédacteurs partagent des documents.

Nous utilisons la classe java.util.concurrent.locks.ReentrantReadWriteLock pour assurer l'accès concurrent aux documents. Notre solution s'appuie sur plusieurs classes déjà présentes dans les sources, telles que WaitingLogger, Database, Document, Person, Timer et Main, pour implémenter le monitoring de la concurrence de manière efficace.

Dans ce rapport, nous détaillerons les objectifs pédagogiques de notre solution, ainsi que les problématiques à résoudre et les interfaces minimales des classes impliquées. Nous expliquerons également les choix de conception que nous avons faits pour l'interface utilisateur, et nous mettrons en évidence les fonctionnalités clés de notre solution de monitoring de la concurrence.

1. Structure du framework
   1. Diagramme de classes
   2. Classe Document
   3. Classe Person
   4. Classe Timer
   5. Classe Log
   6. Classe WaitingLogger
   7. Déroulement du programme
2. Résultats
3. Limitations et perspectives
4. Conclusion

En fonction de la longueur des premières pages du rapport (avant la table des matières), il se peut que le numéro de la dernière page courante (normalement celle-ci) ne corresponde pas au nombre de pages total.

Il faut alors modifier ce dernier champ (numéro de page total, à droite du « / »).

* Placer la souris sur ce champ, qui sera alors grisé
* Appuyer les touches SHIFT + F9 pour afficher le code du champ
* Modifier le nombre à soustraire au nombre de pages total (à droite du signe « - »)
* Appuyer sur les touches SHIFT + F9 pour rebasculer en mode normal

1. Annexes
   1. Guide utilisateur
   2. Cahier des charges
   3. Table des illustrations
   4. Bibliographies et références
      1. Sites Web
      2. Livres
      3. Autres