Cours de PARAPA II
Labo 1

Temps à disposition : 6 périodes

1 Objectifs pédagogiques

- Réaliser un framework pour le monitoring de la concurrence, et plus précisément des files d'attente sur les éléments de synchronisation en Java, en utilisant ici le patterne Lecteurs-Rédacteurs.

2 Enoncé du problème

Dans le cadre du paradigme lecteurs-rédacteurs, nous sommes intéressés à la réalisation d'une application permettant de visualiser les files d'attente des différents objets de synchronisation. Ici des lecteurs et rédacteurs partagent des documents. La classe java.util.concurrent.locks.ReentrantReadWriteLock sera utilisé pour l'accés aux documents.

Les interfaces minimales de ces classes sont fournies. A vous d'ajouter ce qui pourrait être nécessaire à votre solution de monitoring. Vous êtes libre de choisir la manière d'afficher les informations demandées (en console ou sur UI), sachant qu'une solution avec UI peut obtenir un bonus si l'interface est bien travaillée.

D'autres classes sont déjà présentent dans les sources, afin de vous guider dans votre implémentation :

1 - WaitingLogger

- Sert à stocker l'état actuel des files d'attentes
- Est un singleton
- Offre les méthodes suivantes
  - addWaiting(Person p, long timer), appelé par l'objet de synchronisation pour signaler qu'il ajoute un thread dans la file d'attente d'un document
  - removeWaiting(Person p, long timer), appelé par l'objet de synchronisation pour signaler qu'il supprime un thread de la liste d'attente d'un document
  - finished(Person p, long timer), appelé par l'objet de synchronisation pour signaler qu'un thread a terminé de traiter un document
  - popNextLog(), appelé par le thread principal afin de traiter le prochain log et de demander à l'interface visuelle de s'actualiser

2 - Database

- Contient la totalité des documents à traiter par les personnes (threads)
- Est un singleton
- Offre les méthodes suivantes
  - Init(int size), permet de définir le nombre de documents à générer dans la base de données
  - getRandomDocument(), fourni un document aléatoire de la base de données, permettant de l'attribuer à un utilisateur

- getNames(), fourni une liste de tous les noms de documents stockés

3 - Document

- Représente la ressource concurrente que les threads se partagent
- Contient uniquement une chaine de caractère et un nom
- Offre les méthodes suivantes
  - getName(), retourne le nom du document
  - readContent(), retourne le contenu du document (appelé par les lecteurs)
  - setContent(String newContent), permet de mettre à jour le contenu du document (appelé par les rédacteurs)

4 - Person

- Représente à la fois les lecteurs et les rédacteurs
- Implémente l'interface runnable
- Possède un nom, un document à traité, un temps de traitement et une durée de traitement
- Un timer interne afin de garder une trace du temps passé en activité
- Offre les méthodes suivantes
  - run(), méthode principale à implémenter gérant toute l'activité du thread lancé avec ce runnable. Gestion de la lecture ou de l'écriture du document attribué
  - pauseTimer(), arrête le timer lorsque la personne est mise en attente
  - resumeTimer(), relance le timer lorsque la personne peut à nouveau traiter le document
  - timePassed(), retourne le temps passé à traité le document

5 - Timer

- Garde une trace du temps passé dans l'application
- Est un singleton
- Offre la méthode suivante
  - timePassed(), retourne le temps passé de manière générale et non par thread concurrent

6 - Log

- Représente une action concurrente à afficher
- Possède un type (WAITING, REMOVE ou FINISHED)
- Contient une référence à la personne concernée

7 - Main

- Représente le thread principal gérant le bon déroulement et le démarrage de l'application
- Contient une « FutureTask » contenant le déroulement du programme pouvait être arrêté sur demande
- Offre la méthode suivante
  - generatePopulation(Database db, int nbPersons), s'occupe de générer un nombre de personnes défini à propriétés aléatoires et d'attribuer à chacun un document aléatoire de la base de données.

Un exemple de sortie est fourni sous format vidéo.

3 Travail à rendre

Une implémentation fonctionnelle d'une solution de monitoring du paradigme lecteurs-rédacteurs (code source et jar)

Le framework peut varier du code source proposé, la solution doit cependant répondre aux besoins du problème mentionné. Il n'existe pas de solution unique à ce problème, soyez créatifs

Un retour UI afin de visualiser l'état des files d'attentes et de l'états des threads. Un retour console suffit, un bonus peut être attribué si la solution offre une interface visuelle

- Un rapport expliquant vos solution et resultats commentés.
- Travail doit se faire par chaque élève.
- A rendre le 21. avril.2023 un fichier zippé avec le nom de l'étudiant par message sur Teams au groupe aicha.rizzotti@he-arc.ch et julien julien.senn@he-arc.ch

4 Barème de correction

Conception	20%
Exécution et fonctionnement	20%
Tests	10%
Codage	40%
Documentation et commentaires	10%