Développement de programme dans un environnement graphique 420-203-LI Automne 2022 Cégep Limoilou Département d'informatique Professeur : Martin Simoneau	Formatif 5 Service JavaFX

Objectifs

- Utilisez des services pour garder l'UI active lors de longues tâches.
- Informer l'utilisateur de l'état d'avancement des tâches en arrière-plan.

À remettre :

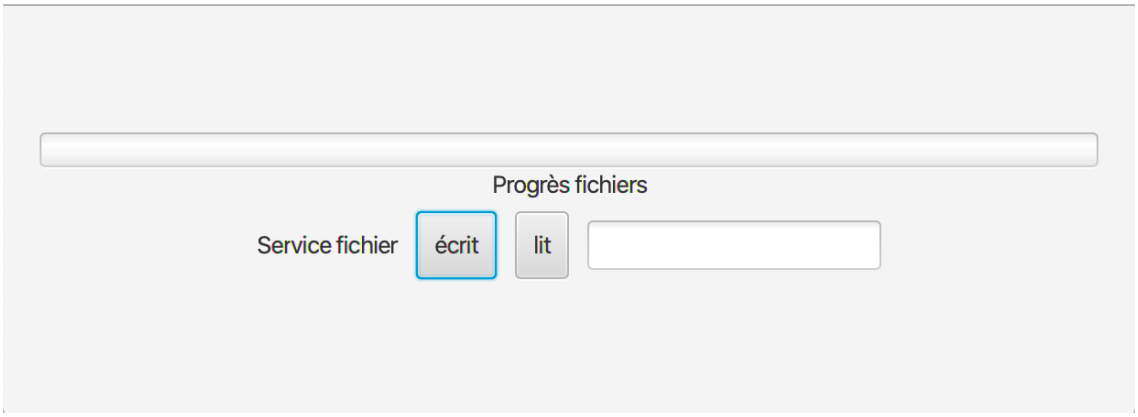
- Le travail sera remis sur Léa à la date indiquée.

Contexte :

- Remettre vos projets sur Léa
- Une remise par étudiant

À faire

On vous a remis un projet avec un fichier *primary.fxml* qui contient l'UI suivant :



Étape1 Écrire

- L'objectif est de réaliser une écriture et une lecture longue en arrière-plan.
- Dans le package *formatif5*, ouvrez la classe ***FichierEcritureServce.java*** et faites :
  - Cette classe doit être un enfant de *Service* qui retourne un *Void (oui avec une majuscule)*.
  - Ajoutez à la classe un attribut *contenuString*, qui contient le message à mettre dans un fichier. Vous pouvez l'initialiser avec un *setter*.
  - La méthode *createTask* doit
    - retourner un nouvel objet de type *FichierWriterVoidTask* .
      - Utilisez une classe interne non statique pour cette classe
      - La classe hérite de *Task<**V**oid>. (oui avec une majuscule)*
      - Elle implémente la méthode call de Task
        - La méthode retourne **toujours** null (à cause du type **V**oid).
        - La méthode doit écrire dans un fichier texte nommé ***texte.txt***, le contenu de l'attribut *contenuString*. Notez que pour plus de robustesse il serait approprié de copier le contenu de *contenuString* dans un attribut local à la tâche pour éviter un conflit d'accès.
      - N'oubliez pas de définir un ***dossier de travail*** dans votre configuration de lancement IntelliJ!
- Dans la classe *PrimaryController :*

    - Ajoutez un attribut de type fichier *FichierEcritureService*. N'oubliez pas de l'instancier.
    - Allez dans la méthode ecritFichier() et :
        - Transférez le texte contenu dans *fichierTextField* dans le service d'écriture.
        - Si l'état du service (getState) n'est pas *READY*, on ne doit pas lancer le service.
        - Lancez le service avec la méthode *start*() uniquement si le service est prêt..
- Lancez l'application, écrivez quelque chose dans le *fichierTextField* et appuyez sur le bouton *écrit*. Ça devrait fonctionner, mais ce n'est pas vraiment intéressant, la tâche s'exécute trop rapidement pour que le service soit intéressant. On va devoir le ralentir artificiellement.
    - Dans la méthode *call* de *FichierWriterVoidTask*, utilisez la méthode *slowRandom* de l'interface *SlowHelper* pour ralentir le traitement. Utilisez une plage entre 1 sec et 10 sec.
- Si vous relancez l'application, maintenant la réponse se fera attendre entre 1 et 10 secondes. Mais comment savoir que la tâche est terminée ? Vous pouvez surveiller le fichier *texte.txt* qui doit être produit, mais ce n'est pas très convivial!
- L'UI dans cet état est très risqué, si vous relancez une écriture avant la fin de la première tâche, il risque d'y avoir des problèmes ! Pourquoi?
- On va commencer par désactiver les boutons aussi longtemps que la tâche en arrière-plan n'est pas terminée.
    - Avant d'appeler *start,* sur le service, désactivez les 2 boutons (*lit* et *écrit*) avec la méthode *setDisable*().

Quand doit-on les réactiver? Lorsque la tâche aura terminé, elle enverra un message de changement d'état au service. On peut programmer un *callback* qui réactivera les boutons lorsque ce message sera reçu.

        - Utilisez *setOnSucceeded*() sur le service pour réactiver l'UI.
        - Il faut aussi réactiver l'UI sur *setOnFailed* et *setOnCanceled*.
- Si on relance l'application, le comportement est mieux, mais loin d'être conviviale. On ne sait jamais si l'application continue de travailler ou si elle est plantée. On va devoir améliorer le suivi de tâche.
    - Dans la tâche *FichierWriterVoidTask ajoutez une méthode private simulateProgress()*
        - La méthode doit appeler la méthode *updateProgress*() un nombre prédéterminé de fois (*MAX_STEPS*).
            - Le premier paramètre est double représentant l'avancement
            - Le second paramètre est un double représentant l'état d'avancement maximum.
        - La méthode peut également envoyer un message avec la méthode *updateMessage*().
            - Envoyez le texte *Étape 1*, le numéro de l'étape doit changer à chaque appel.
        - Entre chaque appel vous devez gaspiller entre 0,1 et 1 seconde.
    - Dans la méthode *call*(), remplacez l'appel à *slowRandom* par un appelle à *simulateProgress*.
- Pour l'instant, la tâche de fond envoie des mises à jour sur l'état d'avancement, mais l'UI n'en tient pas encore compte. On va programmer l'UI pour afficher ces informations:
    - Dans la méthode *ecritFichier*(), on va programmer des callback sur les propriétés de mise à jour
        - Utilisez *addListener* sur *progressProperty* du service pour ajouter un callback qui va afficher la valeur du progrès dans la *progressBar*. Le troisième paramètre reçu par le callback est la nouvelle valeur. On peut également utiliser les binding pour le faire.
        - Utilisez *addListener* sur *messageProperty* du service pour ajouter un callback qui va afficher la valeur du progrès dans la *messageFichierLabel*. On peut également utiliser les binding pour le faire.
        - Pour la finition on peut également afficher le message *Terminé!* Dans le *messageFichierLabel,* lors de la réception de l'événement *Succeeded*.
- On a bien géré l'écriture, maintenant refaites la même chose avec la lecture, mais

- au lieu d'utiliser *addListener* pour afficher le message et la progression, utilisez les binding *javaFX*.
- *FichierLectureService* doit retourner une *String* elle doit donc être typée en conséquence.
- Il y aura aussi une différence pour le retour de l'information, si vous utilisez la méthode *getValue* du Service directement dans *litFichier*, vous allez bloquer complètement l'interface! Ce qu'on essaie pourtant de ne pas faire. Vous devrez donc récupérer la chaîne avec la méthode *getValue* dans le callback mis par *setOnSucceeded*.

FIN