Projet de Compléments en Programmation Orientée Objet : Gestionnaire de téléchargements et Aspirateur de site web

Projet à réaliser en binôme. Il faut, a minima, faire la « Phase 1 » du projet.

Phase I : gestionnaire de téléchargements

Un gestionnaire de téléchargements est un logiciel permettant de télécharger plusieurs fichiers simultanément depuis Internet en permettant à l'utilisateur de contrôler les tâches de téléchargement : fonctionnalités de planification, de pause/reprise, réglage du nombre de téléchargements simultanés, etc.

On peut aussi imaginer des fonctionnalités plus rares, comme l'arrêt et le démarrage programmé par rapport à l'horloge.

La plupart des navigateurs web intègrent un tel gestionnaire.

Premier objectif : écrire un gestionnaire de téléchargements.

Structure : la première phase du projet sera décomposée en 2 modules :

- une bibliothèque fournissant les fonctionnalités du gestionnaire de téléchargements
- une interface utilisateur (UI) permettant à l'utilisateur de profiter pleinement des fonctionnalités de la bibliothèque.

La bibliothèque du gestionnaire de téléchargement Celle-ci devra être programmée dans son propre module (projet Gradle indépendant et/ou module JPMS indépendant du module UI).

Cette bibliothèque implémente toutes les fonctionnalités demandées dans le projet, sauf la présentation à l'utilisateur final. Elle n'a notamment pas de méthode `main` (sauf dans les classes de test).

Son API devra permettre d'instancier au moins les objets suivants :

- gestionnaire : l'objet principal, contenant (ou référençant) tout l'état d'une instance du gestionnaire de téléchargements. Typiquement, une application donnée n'en instanciera qu'un seul.
- tâche de téléchargement unique : objet représentant le téléchargement d'un seul fichier.

Ces objets devront être munis de méthodes non bloquantes permettant de piloter toutes les fonctionnalités du projet : créer une tâche, la démarrer, la mettre en pause, la reprendre, l'arrêter, vérifier son avancement et son statut (non démarré, en cours, annulé, erreur, ...), lister les tâches courantes, ...

UI du gestionnaire de téléchargement L'UI devra être programmée dans dans son propre module (projet Gradle et/ou module JPMS), qui aura pour dépendance externe la bibliothèque (fournie dans un fichier `.jar`).

L'UI pourra être, par exemple, sous forme une des formes suivantes :

- interface graphique
- textuelle interactive (logiciel en mode texte qui reste ouvert et lit des commandes entrées au clavier)
- démon (logiciel résident, que l'on contrôle par des lignes de commande dans le shell du système d'exploitation)

Quel que soit le choix, l'UI doit rester répondante : aucune commande entrée ne doit bloquer l'UI en attendant que la commande soit traitée.

L'UI doit pouvoir être capable de piloter toutes les fonctionnalités implémentées par votre bibliothèque.

Phase II : aspirateur de site web

Un aspirateur de site web est un logiciel permettant de télécharger tous les fichiers d'un site web en suivant les liens jusqu'à une certaine profondeur. L'idée étant que le site téléchargé puisse ensuite être ouvert par un navigateur Web, même sans connexion à internet.

Dans la phase II du projet, nous voulons fournir cette fonctionnalité, en sous-traitant la partie téléchargement au gestionnaire programmé en phase I. Quelques fonctionnalités supplémentaires seront proposées, permettant notamment de contrôler ensemble toutes les tâches associées au téléchargement d'un même site (ex : annulation de l'aspiration, ou bien exécution d'un callback quand l'aspiration est terminée).

Objectif : écrire un tel aspirateur de site web.

Modules additionnels : on ajoutera au projet les modules suivants :

- une bibliothèque fournissant toutes les fonctionnalités de l'aspirateur de site sous forme d'objets instanciables avec toutes les méthodes permettant de contrôler une aspiration
- une interface utilisateur permettant à l'utilisateur de profiter pleinement des fonctionnalités de la bibliothèque.

La bibliothèque de l'aspirateur de site Celle-ci devra être programmée dans son propre module, indépendant de son UI, mais dépendant du module du gestionnaire de téléchargements (et peut-être de bibliothèques tierces).

Cette bibliothèque implémente toutes les fonctionnalités de l'aspirateur de site, sauf la présentation à l'utilisateur final.

Son API devra permettre d'instancier notamment des tâches d'aspiration de site (depuis lesquelles on créera de multiples tâches de téléchargement de fichiers individuels) une telle tâche sera paramétrée par une URL de départ, une profondeur, ainsi qu'un gestionnaire de téléchargement sur lequel les téléchargements seront programmés.

L'UI de l'aspirateur de site L'UI devra être programmée dans dans son propre module (projet Gradle et/ou module JPMS), qui aura pour dépendances externe la bibliothèque aspirateur et la bibliothèque du gestionnaire de téléchargement.

Selon la façon de l'implémenter, cette UI peut aussi dépendre de l'UI du gestionnaire de téléchargements, afin de réutiliser certains de ses composants graphiques.

Cette nouvelle UI doit permettre, comme la précédente, de piloter le gestionnaire de téléchargement, mais aussi de programmer des aspirations de sites sur ce gestionnaire.

Conseils de réalisation technique

En attendant le cours sur la concurrence en Java. Si le *multi-threading* vous fait peur, vous pouvez, en un premier temps, programmer un gestionnaire de téléchargements factice qui simule le fonctionnement du gestionnaire sans pour autant tourner en tâche de fond (pour cela,

les méthodes devront retourner un résultat crédible, alors qu'en réalité aucun téléchargement n'est en cours ; par exemple, l'avancement reporté d'un téléchargement pourrait être une fonction du temps écoulé depuis son démarrage). Cela vous permettra de programmer sereinement l'UI en attendant que vous ayez fini le cours sur la concurrence.

Autre piste : il faut savoir que lorsque qu'on crée une interface graphique, celle-ci s'exécute, de base, dans un *thread* différent du *thread main*.

Il serait ainsi possible d'exécuter dans le thread *main*, après avoir lancé l'IG, une boucle qui va chercher des tâches dans une file d'attente synchronisée (par exemple `LinkedBlockingQueue`), qui serait alimentée par les demandes de téléchargement produites dans les gestionnaires d'évènement de l'interface graphique.

Cela permettrait déjà d'avoir un logiciel qui fonctionne sans encore faire de concurrence avancée (avec la limitation qu'il n'y aura qu'un seul téléchargement à la fois ; les autres resteront en attente jusqu'à ce que leur tour arrive).

Dès que vous saurez comment faire, lancerez plutôt vos téléchargements dans des tâches soumises à un *thread pool*.

Gradle. Attention à la version de Java utilisée. Si vous avez plusieurs versions de Java sur votre système, configurez la variable `JAVA_HOME` pour qu'elle pointe vers une version de Java compatible avec la version de Gradle que vous utiliserez.

En outre, la dernière version de Gradle (à l'heure où j'écris ces lignes) est la 5.6.3. Celle-ci est compatible avec Java 11 (comme on a pu le constater dans le TP1), mais pas avec Java 13 ! (compatibilité prévue pour Gradle 6.0).

Pour fixer les choses : travaillez avec Java 11 et Gradle 5.6.3 !

Dans ce projet, il y a 4 projets Gradle à créer. Vous pourrez les initialiser avec le wizard `./gradlew init`. Dans ce cas, pour les bibilothèques, vous répondrez « `3: library` » à la première question. Pour les UI, vous répondrez « `2: application` ».

UI via JavaFX Si vous faites une interface graphique, JavaFX est conseillé. Malheureusement, contrairement à Swing, JavaFX n'est plus inclus dans le JDK depuis Java 10. JavaFX est donc désormais considéré comme une dépendance externe.

Si vous n'utilisez pas Gradle, vous pouvez télécharger le fichier jar directement sur le site de JavaFX. Dans ce cas, il faudra spécifier son chemin dans l'argument `module-path` de `javac` et `java` (cf. `https://openjfx.io/openjfx-docs/#install-javafx`).

Si vous utilisez Gradle, suivez plutôt le tutoriel `https://openjfx.io/openjfx-docs/#gradle`.

Pour résumer, une fois que vous avez créé le projet Gradle pour l'UI (en suivant les étapes du TP1) vous ajoutez la déclaration du plugin JavaFX dans la section `plugins` de votre `build.gradle` :

```
plugins {
    ...
    id 'org.openjfx.javafxplugin' version '0.0.8'
}
```

Puis vous ajoutez une section JavaFX pour configurer ce plugin :

```
javafx {
    version = "11.0.2"
    modules = [ 'javafx.controls' ]
}
```

Téléchargements simples : on peut utiliser la nouvelle API de client HTTP fournie dans Java 11 (package `java.net.http`) (remarque : JSoup sait télécharger des fichiers aussi, cf. ci-dessous).

Plein d'exemples ici : `https://openjdk.java.net/groups/net/httpclient/recipes.html` (en attendant d'avoir fait le cours sur la concurrence, regardez les exemples dits « synchrones » ; dans la version finale du projet, ce sera plutôt les exemples « asynchrones » qui devront vous inspirer).

Recherche des liens dans un fichier HTML. On peut le faire à l'aide d'expressions régulières, mais il est plus fiable de faire une vraie analyse syntaxique pour extraire le contenu des attributs `src` des différentes balises HTML. Pour cela, on peut utiliser la bibliothèque JSoup `https://jsoup.org`. Ajoutez la dépendance suivante dans la section `dependencies` de votre `build.gradle` :

```
compile 'org.jsoup:jsoup:1.12.1'
```

Sinon, si vous n'utilisez pas Gradle téléchargez juste le fichier `.jar` dans un répertoire et ajoutez-le au *class path* de votre projet.