



Atelier 10 - DI, Packaging, Properties, Javadoc

Table des matières

1	Objectifs.....	2
2	Théorie.....	2
3	Introduction.....	2
4	Exercices.....	2
4.1	Blacklist – Fichiers *.properties.....	3
4.2	Javadoc	3
4.3	Packaging de l'application - version IntelliJ.....	5
4.4	Packaging de l'application - version Maven (pour CI/CD)	5
4.5	Création d'un exécutable Windows (*.exe)	6
4.6	 Analyse de contenu.....	7
4.7	 Injection de dépendances plus complexe.....	8

1 Objectifs

ID	Objectifs
AJ01	Packaging d'une application
AJ02	Utiliser les fichiers *.properties
AJ03	Utiliser l'inspection
AJ04	Utiliser l'injection de dépendances

2 Théorie

La théorie sur l'injection de dépendances, les fichiers *.properties... a été vue en cours de COO. N'hésitez pas à vous référer aux slides, à vos notes, et aux exemples de code écrits en cours.

3 Introduction

L'application développée la semaine dernière a maintenant une architecture qui correspond aux exigences de :

- Faible couplage entre les packages
- Encapsulation des classes/implémentations... qui sont inutiles à l'extérieur du package
- Flexibilité à l'exécution grâce à l'injection de dépendances

Nous allons donc maintenant mettre en place un certain nombre de concepts "annexes" que vous aurez l'occasion de rencontrer dans des projets professionnels :

- Configuration de l'application grâce aux fichiers *.properties
- Rédaction et génération automatique de la javadoc
- Packaging de l'application

Bonnes découvertes !

4 Exercices

Avant de commencer les exercices, récupérez la solution de la séance précédente (version finale avec DI, pas uniquement l'exercice 4). Tous les exercices doivent être faits dans le même projet, en local sur la machine, pas sur un drive (OneDrive, Google Drive...).

4.1 Blacklist – Fichiers *.properties

Mettez en place un fichier `properties` qui va contenir une clef `blacklistedDomains` qui aura comme valeur une liste de domaines interdits (ex : `google.be`, `facebook.com...`), séparés par des « ; ».

N'hésitez pas à consulter les slides du cours de COO pour savoir comment utiliser les fichiers `*.properties`. De plus, la javadoc est également une source de données fiable et complète concernant l'usage des propriétés en JAVA :

<https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html>

Attention de bien placer le fichier `*.properties` à la racine de votre projet. Surtout pas dans votre dossier `src`. Effectivement, le dossier `src` est intégré dans le "bundle" de l'application lorsque vous la compilez. Or le principe de base d'un fichier `*.properties` c'est qu'il sera lu à l'exécution, et doit pouvoir être changé à tout moment sans recompiler l'application. Il doit donc être en dehors des sources de votre projet.

Créez un service `BlacklistService` dans un package `blacklist` qui va lire ce fichier `properties`, une et une seule fois au chargement de la classe, et qui va s'occuper de vérifier si le site visité est autorisé ou non. Injectez ce service là où on a besoin de lui (`ProxyServer`).

Ce service contiendra un attribut static `blacklistedDomains` de type `Set<String>`. Celui-ci sera initialisé au chargement de la classe (clinit). Il contiendra également une méthode d'instance (non static) : `public boolean check(Query query)`. C'est elle qui s'occupera de vérifier si le site est autorisé ou non. Pour cela, vérifiez juste si l'URL de la requête contient le domaine.

Astuce : il est possible de faire cette vérification (URL contient le domaine) en une ligne grâce aux stream 😊

4.2 Javadoc

Nous avons maintenant une belle application qui mérite d'être distribuée. Qui plus est, nous avons une structure intéressante, avec notamment un package `domaine` qui ne montre pas certaines classes à l'extérieur (`QueryImpl`). Il est donc temps de documenter l'application avant de la distribuer.

Java possède un mécanisme simple d'utilisation pour documenter le code : la Javadoc. Il s'agit d'un outil qui permet de compiler certains types de commentaires écrits dans le code, et d'en faire des pages web qu'on peut déployer facilement sur le net.

Comme cet outil se base sur les commentaires du code, il est primordial de commenter correctement le code, notamment à l'aide des balises `/** ... */`. Ce sont ces balises qui seront utilisées pour générer la documentation. Dans ces commentaires, vous pouvez ajouter des tags comme, par exemple : `@author` pour spécifier l'auteur de la classe/méthode... ou encore `@throws` pour définir l'exception qui sera lancée...

Ecrivez la javadoc pour les interfaces du package `domaine`. Ecrivez la javadoc pour chaque interface, et chaque méthode des interfaces, ainsi que l'énuméré.

N'hésitez pas à consulter <https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html#tag> pour plus d'informations concernant l'écriture de la javadoc. C'est beaucoup trop complet par rapport à ce dont nous avons besoin ici, mais n'hésitez pas à faire des recherches directement dedans pour trouver les informations concernant ce dont vous avez besoin.

Une fois la javadoc écrite, nous allons générer la documentation web associée :

- Dans le menu “tools” d’intelliJ, cliquez sur “generate JavaDoc”.
- Observez les différentes options qui s’offrent à vous, mais inutile de les changer.
- Complétez “l’output directory” en choisissant le dossier “doc” de votre projet IntelliJ. Comme ce dossier n’existe pas, vous devez le créer avec le bouton “new directory” dans la fenêtre pour choisir le dossier.
- Cliquez sur “generate”.
- IntelliJ est en train de générer la javadoc. Observez les messages rouges de la console. Lisez-les. Vous remarquerez que IntelliJ vous avertit qu’il manque de la documentation dans votre application ! Ce n’est pas grave, nous n’allons pas faire une documentation exhaustive. Cependant, faites en sorte qu’il n’y ait aucun avertissement en ce qui concerne les 2 interfaces du package domaine.
- Une fois que IntelliJ a terminé de générer la javadoc, vous pouvez ouvrir le dossier “doc” qui devrait maintenant contenir plusieurs fichiers.
- Faites un clic droit sur “index.html” et ouvrez-le avec votre navigateur favori.
- Naviguez dans la documentation générée, et observez particulièrement votre documentation des interfaces. Vous remarquerez :
 - Il n’y a pas l’implémentation package-friendly `QueryImpl`. C’est tout à fait ce qu’on souhaitait en mettant en place une factory et l’injection de dépendances. Cacher les implémentations pour ne pas en être dépendant.
 - Par contre, on voit l’implémentation de la factory. Ce n’est pas souhaitable mais il s’agit d’une contrainte technique pour faire l’injection de dépendances dans le main, sans introspection. Si on avait voulu faire les choses à fond, on aurait dû la rendre package-friendly, et l’instancier par introspection dans le main.
 - Dans l’implémentation, observez la javadoc de la méthode `getQuery` qui est copiée depuis l’interface. Inutile donc de documenter les méthodes des implémentations, sauf s’il y a des informations techniques importantes à y ajouter.

En écrivant la javadoc des interfaces et de leurs méthodes, vous avez rencontré les tags suivants : `@return`, `@param`. Il y a quelques tags qui sont intéressants et que nous n’avons pas encore eu l’occasion de rencontrer.

- Ajoutez une méthode `setMethod(String method) throws IllegalArgumentException;` dans l’interface `Query`.
- Ecrivez son implémentation dans la classe `QueryImpl`. Elle doit récupérer le nom de `method` (GET, POST...) sous format `String`, et setter la valeur de l’attribut `method`. L’implémentation doit être écrite en une seule ligne. Consultez les méthodes de l’énuméré `QueryMethod` si nécessaire.
- Ecrivez la javadoc de cette méthode, en n’oubliant pas les tags `@param` et `@throws`.
- Ajoutez le tag `@deprecated` car cette méthode historique (oui oui, considérons que c’est une vieille méthode qui était là depuis longtemps) n’est plus utile aujourd’hui...
- Générez la javadoc
- Observez les nouveaux tags ajoutés dans la javadoc, mais également dans IntelliJ, lorsque vous faites un CTRL-ESPACE pour voir les méthodes que vous pouvez appeler sur l’objet `Query` (observez le deprecated).

4.3 Packaging de l'application - version IntelliJ

Vous vous êtes toujours demandé comment vous pouvez distribuer la super application que vous avez développée, histoire que tous vos amis puissent en profiter ? Voici une première technique pour y arriver 😊

Cette solution est simple à mettre en œuvre et permet de créer un fichier *.jar unique, que vous pouvez facilement distribuer. Ce fichier contient votre application et toutes ses dépendances. Il peut être exécuté sur n'importe quelle machine qui possède une JVM compatible avec la version de la JVM pour laquelle vous avez développé l'application.

Pour faire votre premier *.jar, suivez le tutoriel suivant :

<https://www.jetbrains.com/guide/java/tutorials/hello-world/packaging-the-application/>

Une fois terminé, vous pouvez exécuter votre application :

- Cliquez droit sur le dossier `out/artifacts/seance10_jar`
- Cliquez sur "Open In" et ensuite sur "Terminal".
- Dans le terminal qui s'ouvre, exécutez la commande `java -jar seance10.jar`
Attention, le nom de votre *.jar est potentiellement différent.
- Vous remarquerez que l'application ne se lance pas, car il ne trouve pas le fichier `blacklist.properties`. Logique, il doit être à côté du *.jar.
- Copiez donc le fichier *.properties dans le dossier `out/artifacts/seance10_jar`
- Relancez, et vérifiez que l'application fonctionne correctement.

Vous venez de créer votre premier exécutable JAVA !

Si vous avez cette erreur : `main/Main has been compiled by a more recent version of the Java Runtime ...` Il essaye de vous dire que vous avez compilé votre application pour tourner sur une version plus récente que la version de java utilisée pour lancer votre application. Mais comment est-ce possible ?!?

En réalité, lorsque vous mettez le numéro de version 21 dans votre `pom.xml`, vous dites que l'application doit être compilée pour être exécutée sur la version de java 21. Mais en console, lorsque vous exécutez votre application, vous n'avez peut-être pas la même version que IntelliJ. Vous pouvez confirmer cette hypothèse en lançant `java -version` dans votre terminal. Si vous voyez une version 17 de java par exemple, l'application ne pourra pas être lancée car votre version de java ne comprend pas la version 21. C'est un peu comme si vous tentiez d'installer une application compatible avec windows 10/11 sur un PC windows 7.

Pour corriger le souci, le plus simple est de changer le numéro de version de java dans votre `pom.xml` pour matcher avec la version installée sur votre machine (cfr. `java -version` dans votre terminal)

4.4 Packaging de l'application - version Maven (pour CI/CD)

L'exécutable que nous avons créé dans l'exercice précédent est parfaitement fonctionnel. Mais pour le générer, nous avons dû ouvrir IntelliJ, cliquer sur quelques boutons et ensuite, nous avons eu un *.jar. C'est une opération qui n'est pas possible lorsqu'on est dans le contexte d'un pipeline CI/CD, comme vous avez pu le voir et configurer en devops.

Nous avons besoin d'un processus automatisé, qui peut être lancé à l'aide d'une simple ligne de commande, sans avoir une interface graphique et encore moins un IDE installé...

Nous allons donc utiliser pour cela Maven. Et oui, Maven ce n'est pas juste un gestionnaire de dépendances, mais principalement un "build tool". N'hésitez pas à consulter : https://en.wikipedia.org/wiki/Apache_Maven pour plus d'informations.

- Ajoutez ce code dans votre pom.xml. Il définit que, pour faire un build de l'application, maven doit créer un jar avec les dépendances contenues dedans ;

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>fully.qualified.MainClass</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Remplacez `fully.qualified.MainClass` par le nom complet de votre classe main (packages compris) ; astuce : pour obtenir le fully qualified name, vous pouvez faire un clic droit sur le nom de la classe dans l'explorateur de Projet, Copy Path/Reference..., Copy Reference. Dans tous les cas, le nom du package de la classe est visible dans la classe elle-même, c'est la première ligne du fichier.
- Demandez à votre IDE de builder l'application. Pour cela, appuyez 2 fois sur la touche CTRL, et entrez `mvn clean compile assembly:single`.
- On vient de demander à Maven de supprimer le contenu du dossier target, de compiler l'application et de faire un jar ;
- Dans le dossier target, vous avez maintenant un *.jar. Exécutez-le dans un terminal comme vous avez pu le faire pour l'exercice précédent.

Si vous avez un souci avec le plugin maven-assembly-plugin qui n'est pas trouvé, ajoutez la dépendance suivante :

```
<dependency>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>3.4.2</version>
</dependency>
```

4.5 Création d'un exécutable Windows (*.exe)

Avoir un *.jar à exécuter en ligne de commande, ce n'est pas forcément le plus pratique... L'idéal serait d'avoir un fichier *.exe sur lequel on peut simplement double-cliquer pour lancer notre application. C'est ce que nous allons faire maintenant !

Astuce : cette partie de la fiche est réalisable tant sur Windows/Mac/Linux, car l'utilitaire existe pour toutes les plateformes. Cependant, il n'a été testé que sur Windows. De même, le résultat est un

*fichier *.exe qui n'est pas facilement utilisable sur un autre OS que Windows. Si vous utilisez un autre OS, n'hésitez pas à faire cette partie et à tester ensuite le résultat sur une autre machine.*

Il existe beaucoup de solutions pour faire des exécutables. Mais nous allons en choisir une très simple, qui peut être réalisée tant "à la cliquette" qu'en ligne de commandes. C'est pratique car cela permet de le faire facilement à la main, mais également de l'automatiser dans notre pipeline CI/CD.

L'outil choisi est Launch4J : <https://launch4j.sourceforge.net/>

- Nous vous invitons à parcourir rapidement le site web pour regarder un peu ce que l'outil est capable de faire
- Ensuite, téléchargez la version *.zip de l'application sur leur site web. Il s'agit d'une version portable, ce qui est indispensable si on veut pouvoir l'exécuter sur les machines de l'école.
- Une fois le dossier téléchargé, décompressez-le, et exécutez le fichier launch4j.exe. Une fenêtre s'ouvre. Il s'agit de l'interface de création de l'exécutable.
- Encodage le champ "Output file" avec la destination de votre choix. N'oubliez pas de mettre l'extension *.exe à la fin du fichier que vous souhaitez créer.
- Choisissez le fichier *.jar à utiliser. Il a été généré précédemment et est disponible dans le dossier out ou target.
- Dans l'onglet "Header", sélectionnez "console". Effectivement, nous allons générer une application en mode console, et pas avec une interface graphique.
- Dans l'onglet "JRE", entrez le numéro de version minimum de Java. Utilisez pour cela celle installée sur votre PC (executez `java -version` dans le terminal).
- Créez votre exécutable en cliquant sur la roue « Build wrapper » et sauvegarder la configuration dans un fichier *.xml. Sauvegardez-la à la racine de votre projet.
- Vous pouvez à présent lancer l'application en cliquant sur play, ou en double-cliquant sur l'application générée.

Astuce : Si vous rencontrez des soucis lorsque vous lancez votre application (l'application se ferme directement par exemple), et qu'il ne trouve pas la version de java, essayez de changer la version minimale en utilisant la version que vous trouvez dans votre terminal (`java -version` dans le terminal), ou celui du pom.xml... Essayez également de mettre %PATH% comme valeur de "JRE Paths".

Vous allez avoir un souci avec le fichier *.properties. Résolez-le, vous devriez être capable de le résoudre 😊

Une fois que vous avez une version fonctionnelle, n'hésitez pas à jouer un peu avec Launch4J, pour tester ses possibilités : ajouter une icône, une image de chargement, configurer les messages d'erreur...

4.6 Analyse de contenu

Dans le fichier properties, ajoutez une clef unauthorizedContent qui aura comme valeur une liste d'expressions régulières, séparées par des « ; ».

Ajoutez dans le service qui s'occupe de la blacklist, une méthode qui s'occupe de vérifier si le contenu du site contient des éléments définis dans les expressions régulières.

Si vous n'avez jamais utilisé les expressions régulières en Java, n'hésitez pas à consulter ces articles qui vous donneront les bases nécessaires à leur utilisation :

- <https://waytolearnx.com/2020/03/les-expressions-regulieres-en-java.html>
- https://www.w3schools.com/java/java_regex.asp

4.7 Injection de dépendances plus complexe

Mettez en place l'injection de dépendances plus complexe vue au cours de COO : avec annotation, map, introspection...