Partie I – Questions de Cours

Question 1

Gradle est un outil de build (build tool), il permet de structurer notre projet, de définir comment un projet doit être compilé, packagé, exécuté, et testé.

Question 2

Les tests automatiques ou le code auto-testant (self-testing code) consiste à écrire du code pour vérifier que le code d'une application fonctionne comme prévu.

Ainsi, vous pouvez, en une seule commande ou d'un seul clic, exécuter un ensemble de tests, qui vous dit si votre application contient ou non des bugs majeurs. Cela permet de réduire radicalement le nombre de bugs présent dans le code livré en production.

Ainsi, on peut développer et ajouter de nouvelles fonctionnalités, ou refactorer, en utilisant les tests pour vérifier que l'on n'est pas en train d'introduire des régressions majeures dans le code. Il devient donc beaucoup plus facile d'itérer sur notre code pour l'enrichir et en améliorer la qualité.

Les tests automatiques permettent donc, de manière générale, d'augmenter la qualité et la fiabilité des logiciels pour un investissement en temps raisonnable.

Idées importantes :

- Gain de temps / rapidité
- Non régression
- Fiabilité de l'application

Question 3

Cela consiste à utiliser les contrats de service pour manipuler de façon transparente des classes différentes qui implémentent ces contrats de service. Autrement dit, cela permet à du code de manipuler facilement des concepts en ignorant totalement les détails de l'implémentation. Le code utilisateur dépend du contrat de service, plutôt que du code qui l'implémente.

Question 4

- Nommage de la méthode, des arguments, des variables
  - En anglais
  - Convention de casse des noms de symboles
  - Noms qui ne révèlent pas l'intention
- Liste devrait être `final`
- Nombres magiques
- `List` au lieu de `LinkedList` à gauche de l'assignement et la valeur de retour
- Extraire en méthode le contenu du `if`

Question 5

Une interface fonctionnelle est une interface qui contient une seule méthode. Elle type des références de méthode à la signature équivalente à la méthode qu'elle contient.

Question 6

Permet d'aplatir une structure de donnée qui en contient d'autre, en une seule, linéaire.

Mappe chaque élément à une collection d'autre chose, puis aplatis le résultat.

Question 7

Plusieurs threads intéragissent avec un bout de code, qui partage des données, pour éviter les problèmes, il faut utiliser `syncrhonized`

Une section critique est une partie de code que l'on veut toujours exécuter atomiquement pour un thread, c'est-à-dire qu'on veut éviter que cette partie de code soit exécutée par plusieurs threads en même temps, pour éviter d'avoir des incohérences dans nos données.

Afin de protéger les sections critiques, Java fournit le mot-clé synchronized pour permet à un thread de "prendre la main" sur un objet le temps de la section critique. On peut donc protéger la section critique dans notre code pour garantir une exécution cohérente.

Partie II – Exercice

On observe que les situations spécifiques des étudiants peuvent se combiner et ont un impact sur le résultat de la classe `StudentAbsenceService`.

Afin de gérer cette combinabilité de façon modulaire et d'éviter à `StudentAbsenceService` de connaître tous les détails de la situation de chaque étudiant, on va utiliser le pattern décorateur.

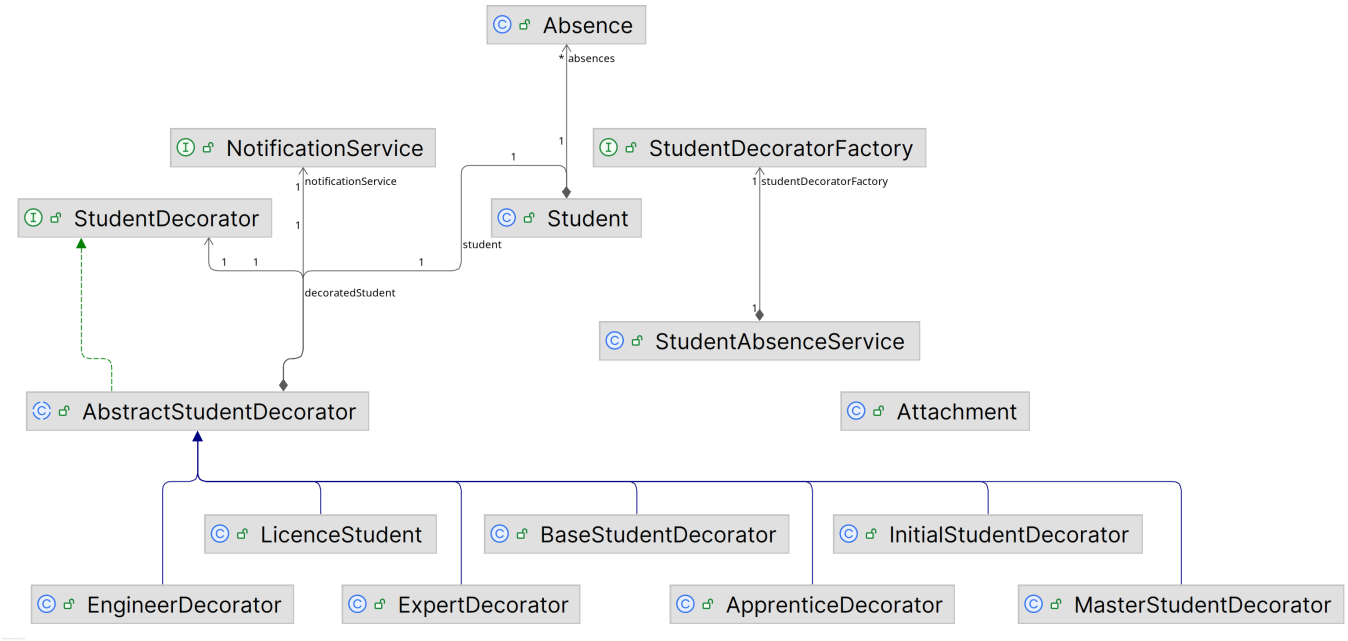
On décore la classe `Student` et donc créer une interface `StudentDecorator` avec les méthodes `hasValidJustification` et `checkSanctions`.

Les classes de décorateur à créer sont :

- `ExpertStudent` et `EngineerStudent`
- `InitialStudent` et `ApprenticeStudent`
- `LicenseStudent` et `MasterStudent`

Chaque classe de décorateur possède une référence de la classe qu'elle décore que chaque implémentation des méthodes `hasValidJustification` et `checkSanctions` appelle la méthode `processAbsence` et `checkSanctions` de la classe qu'elle décore.

On peut aussi utiliser le pattern `Factory` pour déléguer à une classe la création des décorateurs d'étudiant, ainsi `StudentAbsenceService` ignore tous les détails de situation spécifiques.


Absence
absences
NotificationService
StudentDecoratorFactory
notificationService
studentDecoratorFactory
StudentDecorator
Student
student
decoratedStudent
StudentAbsenceService
AbstractStudentDecorator
Attachment
LicenceStudent
BaseStudentDecorator
InitialStudentDecorator
EngineerDecorator
ExpertDecorator
ApprenticeDecorator
MasterStudentDecorator