IG230 – Projet informatique intégré

Informatique de gestion – Bloc 2

Haute-École de Namur-Liège-Luxembourg

Programmation orientée objet avancée
- Série 3 -

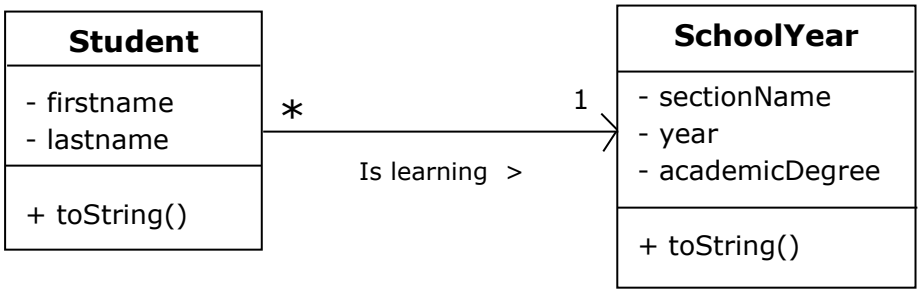
Objectifs

- Sérialiser et désérialiser des objets
- Accéder en lecture et en écriture à des fichiers d'objets
- Synchroniser des threads travaillant en parallèle

Contrainte

- Écrivez le code en **anglais**, que ce soit pour les noms des classes, des variables ou des méthodes.
- Toutes les variables d'instance et les variables de classe doivent être déclarées avec la protection `private`.
- Ne prévoyez **que les getters et setters nécessaires** pour pouvoir exécuter la méthode `main`.

Soit le diagramme de classes :



Étape 1 : Énumération AcademicDegree

Créez l'énumération `AcademicDegree` permettant de gérer les niveaux d'études ou degrés académiques.

Cette énumération ne contient que deux constantes : BACHELOR et MASTER.

Faites-en sorte que les chaînes de caractères correspondant à BACHELOR et MASTER soient respectivement « bac » et « master ».

Étape 2 : Classe SchoolYear

Créez la classe `SchoolYear` qui permet de gérer des années d'études. Une année d'études est caractérisée par le nom d'une section, une année (valeur 1, 2 ou 3) et un niveau d'études (`academicDegree`).

Complétez, s'il y a lieu, les variables d'instance en fonction du diagramme de classes.

> **Contrainte**
>
> La variable d'instance `academicDegree` ne peut contenir qu'une des valeurs de l'énumération `AcademicDegree`.

Prévoyez la méthode `toString`.

Exemple : bac 1 en Informatique

Étape 3 : Classe Student

Créez la classe `Student`. Un étudiant est caractérisé par un prénom et un nom de famille. Complétez, s'il y a lieu, les variables d'instance en fonction du diagramme de classes.

Prévoyez la méthode `toString` dans la classe `Student` qui affiche également les informations sur l'année d'étude.

Exemple : Julien Petit est étudiant(e) en master 1 en Informatique

Étape 4 : Enregistrer des objets dans un fichier

Dans la méthode `main` de la classe `Main`, générez 10 objets de type `Student` et enregistrez-les dans un **fichier** en utilisant la **sérialisation** d'objets.

Fermeture du fichier

Dans la méthode `main`, vous allez **essayer** d'écrire des enregistrements dans un fichier. Vous allez donc placer ces instructions d'écriture dans un fichier dans un bloc `try`.

Cependant, dans tous les cas, c'est-à-dire que l'écriture soit réussie ou qu'une exception soit levée, il faut que le fichier soit fermé à la fin de l'exécution du bloc `try`. Pour ce faire, utilisez le statement ***try-with-resource*** qui est une instruction `try` dans laquelle on déclare les ressources qui doivent être fermées à la fin du bloc `try`. Ces ressources doivent être des objets de classes implémentant l'interface `Closeable`.

Sérialisation

Lorsque des objets sont **transférés d'une machine donnée à une autre machine distante en passant par un réseau**, les objets doivent être sérialisés. Pour pouvoir sérialiser des objets d'une classe, il faut déclarer la classe `implements Serializable`. Aucune méthode cependant ne doit être redéfinie dans cette classe.
À noter que si on souhaite sérialiser un objet, il faut aussi **sérialiser tous les objets qui lui sont reliés** (liens avec d'autres objets implémentés via ses variables d'instance). Autrement dit, si on souhaite sérialiser une classe, il faut aussi sérialiser les classes qui correspondent à toutes ses variables d'instance de type référence.
Si des objets doivent être stockés dans un fichier d'objets, il faut également les sérialiser.

Testez le contenu du fichier ainsi créé pour vous assurer que le mécanisme de la sérialisation/désérialisation a bien été appliqué.

Suggestion

Pour tester la (dé)sérialisation, relisez le fichier et vérifiez que ce sont bien des objets qui sont récupérés en affichant à la console la description de chacun des objets lus dans le fichier (via appel au `toString`).

Avertissement

La désérialisation des données non fiables (*untrusted data*) est dangereuse et doit être évitée.

Des directives de codage sécurisé pour Java SE sont proposées à la section "Serialization and Deserialization" du lien :
https://www.oracle.com/java/technologies/javase/seccodeguide.html

Étape 5 : Threads producteur et consommateur

Prévoyez un thread **producteur** (classe `Producer`) et un thread **consommateur** (classe `Consumer`).

But

Le producteur récupère les objets sérialisés dans le fichier, les désérialise et les place un par un dans une zone commune afin que le consommateur les lise.

Les thread producteur et consommateur partagent une **zone commune** (un objet d'une classe `CommonZone`) qui contient un objet de type `Student` (une variable d'instance de type `Student`).

Le producteur va effectuer une boucle pour lire les objets de type `Student` contenus dans le fichier et écrire ces objets à tour de rôle dans la zone commune. Il préviendra le consommateur quand il aura terminé.

Le consommateur va boucler afin de récupérer un par un les objets écrits dans la zone commune et afficher à l'écran la description des objets qu'il lit (appel au `toString`).

Les deux threads doivent **se synchroniser**.

Synchronisation

Consommateur : le thread qui veut accéder en ***lecture*** à la zone commune doit être placé en attente si l'étudiant n'a pas encore été mis à jour (réécrit).

Producteur : le thread qui veut accéder en ***écriture*** à la zone commune doit être placé en attente si l'étudiant n'a pas encore été lu.

Faites appel aux méthodes `wait` et `notify` à bon escient.

Afin de visualiser la synchronisation des processus, affichez un message à l'écran chaque fois qu'un thread accède à la zone commune en identifiant le thread dans le message (précisez "producteur" ou "consommateur").

Dans la méthode `main`, placez en commentaires les instructions correspondant à l'étape 3, à savoir l'affichage du contenu du fichier.

Créez la zone commune et lancez les threads dans la méthode `main` après les instructions que vous avez placées en commentaires.