IFT-1025 - TP2

Remise:

- Ce projet est à faire **seul** ou en équipe de **deux**.
- **Aucune** équipe de 3 ne sera tolérée.
- On encourage fortement de collaborer sur le projet en groupes de deux.
- Remettez tout votre code dans le même dossier sous format compressé **.zip** avec les noms des membres de l'équipe directement dans le nom du projet sous la forme suivante: **Nom1_prenom1_nom2_prenom2.zip**
- Ajoutez un fichier **equipe.txt** qui contient la matricule et le nom de chaque membre de l'équipe dans chaque ligne dans le **zip**
- Le TP doit être remis par **un seul** membre de l'équipe.
- Vous devez utiliser Github dans ce projet. Mettez votre repo public et ajouter un fichier **github.txt** qui contient le lien dans votre fichier **zip.** L'historique de votre repo doit clairement montrer la collaboration et la contribution (commits) de chaque membre de l'équipe.
- Les membres de l'équipe vont être notés individuellement si on remarque un écart remarquable dans les contributions.
- En plus de votre code, vous devez fournir **trois fichiers jar exécutables: client_simple.jar, client_fx.jar et server.jar** dans le dossier **zip.**

Les projets avec des jars non exécutables avec la commande *java –jar* auront une déduction de 30% de la note finale.

- Vous devez aussi fournir le lien d'une démo vidéo de votre projet dans un fichier **demo.txt** dans le dossier **zip**. Voici un exemple d'une démo: tp2-demo.mp4
- La remise du TP est au **plus tard** le **14 avril**, à **23h59,** sur **Studium**. **Aucune extension n'est permise.**

Description:

Vous devez créer une application Java client-serveur qui permet aux étudiants de s'inscrire aux cours.

On vous donne le code coté **serveur** (qui manque deux fonctionnalités à compléter).

https://github.com/OussamaSghaier/IFT1025-TP2-server/

Dans ce TP, vous êtes invités à compléter le code côté serveur et à écrire le code du client

Serveur:

Le code du serveur contient trois classes java ainsi que deux dossiers: *models* et *data*. Dans le dossier *data*, il y a le fichier *cours.txt* qui contient la liste des cours et le fichier *inscription.txt* qui contient les demandes d'inscription.

Le fichier *cours.txt* contient la liste des cours. Le fichier *cours.txt* a le format suivant:

code_du_cours	**nom_du_cours**	**session**
Exemple:		
IFT1025	Programmation2	Hiver
IFT2255	Génie Logiciel	Automne

Les champs sont séparés par des tabulations (\t).

Le fichier *inscription.txt* contient la liste des inscriptions des étudiants. Il a la forme suivante :

session	**code_cours**	**matricule**	**prenom**	**nom**	**email**
Exemple:					
Hiver	IFT2015	12345678	Leblanc	Mathieu	mathieu@umontreal.ca
Automne	IFT2255	87654321	Lanuze	Charlotte	charlotte@umontreal.ca

Les champs sont séparés par des tabulations (\t).

0. Tâche 0 (10 points)

Écrivez et générez la documentation en fichiers html pour le code du serveur qu'on vous a fourni, sous un dossier **javadoc/serveur**.

Votre documentation doit être du javadoc (/**) bien formatté, utilisant les tags javadoc (par exemple une ligne @param pour chaque paramètre, et une ligne @throws pour chaque exception declarée). Vous devez documenter chaque élément public (classe, interface, méthode, champ et constante). Votre documentation doit être en français, bien écrite (phrases complètes et claires). Votre documentation doit être au bon niveau d'abstraction: ce que le code fait -- pas comment. L'idée est qu'un programmeur qui voudrait réutiliser votre code soit capable de le faire juste à partir de la javadoc générée, sans avoir à lire votre code.

Remarque: La documentation est déjà fournie pour les deux méthodes non-implémentées **handleLoadCourses()** et **handleRegistration().** Cette documentation explique le *comment* car on va vous demander d'implémenter ces deux méthodes après. Donc, ne la considérez pas comme référence!

1. Tâche 1 (20 points) [méthode 1 (10points) + méthode 2 (10 points)]

Complétez le code des deux fonctions **handleLoadCourses()** et **handleRegistration()** selon la documentation qu'on vous a fourni dans la classe **Server.java**. Ne modifier aucune autre portion du code!

- **HandleLoadCourses:** Vous devez lire le fichier *cours.txt*, filtrer les cours selon la *session* donnée en argument à la fonction. Ensuite, vous devez retourner une liste d'objets *Course* au client via le socket en utilisant *objectOutputStream*.
- **HandleRegistration:** Vous devez lire un objet *RegistrationForm* du socket en utilisant *objectInputStream* et ensuite l'enregistrer dans le fichier *inscription.txt* selon le format donné ci-dessus.

Vous devez aussi fournir un jar exécutable pour votre application serveur **server.jar**. **Les projets avec des jars non exécutables avec la commande *java –jar* auront une déduction de 30% de la note finale.**

2. Tâche 2: 20 points [F1 (10 points) + F2 (10 points)]

Pour tester le bon fonctionnement du serveur, Vous devez créer un client de ligne de commande simple et non-graphique qui se connecte au serveur et assure deux fonctionnalités:

- **F1**: une première fonctionnalité qui permet au client de récupérer la liste des cours disponibles pour une session donnée. Le client envoie une requête **charger** au serveur. Le serveur doit récupérer la liste des cours du fichier *cours.txt* et l'envoie au client. Le client récupère les cours et les affiche.

- **F2**: une deuxième fonctionnalité qui permet au client de faire une demande d'inscription à un cours. Le client envoie une requête **inscription** au serveur. Les informations suivantes sont données nécessaires (voir le format du fichier *inscription.txt* ci-dessus) en arguments. Le choix du cours doit être valide c.à.d le code du cours doit être présent dans la liste des cours disponibles dans la session en question. Le serveur ajoute la ligne correspondante au fichier *inscription.txt* et envoie un message de réussite au client. Le client affiche ce message (ou celui de l'échec en cas d'exception).

Vous devez aussi fournir un jar exécutable pour votre application client simple **client_simple.jar**. **Les projets avec des jars non exécutables avec la commande *java –jar* auront une déduction de 30% de la note finale.**

Exemple d'exécution:

Cette image montre un scénario d'exécution de l'application.

```
*** Bienvenue au portail d'inscription de cours de l'UDEM ***
Veuillez choisir la session pour laquelle vous voulez consulter la liste des cours:
1. Automne
2. Hiver
3. Ete
> Choix: 1
Les cours offerts pendant la session d'automne sont:
1. IFT1015      Programmation1
2. IFT1025      Programmation2
> Choix:
1. Consulter les cours offerts pour une autre session
2. Inscription à un cours
> Choix: 2

Veuillez saisir votre prénom: Oussama
Veuillez saisir votre nom: Ben Sghaier
Veuillez saisir votre email: oussama@umontreal.ca
Veuillez saisir votre matricule: 12345678
Veuillez saisir le code du cours: IFT1025

Félicitations! Inscription réussie de Oussama au cours IFT-1025.
```

3. Tâche 3: 50 points [interface javaFx (15points) + contrôle saisie (5 points) + MVC (10 points) + Javadoc (10 points) + exceptions (5 points) + démo mp4 (5points)]

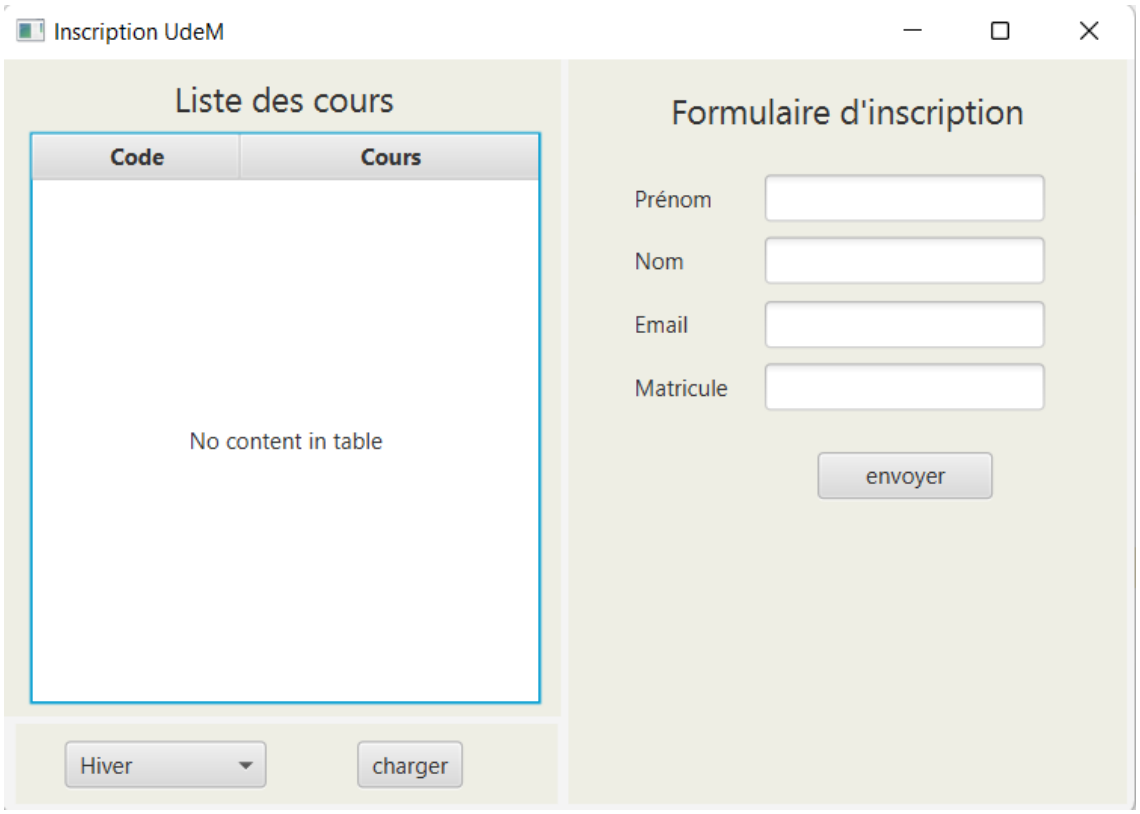
Créer un autre client graphique en tant qu'une application Java FX GUI. Vous pouvez réutiliser des parties du code client de la tâche 2.

L'application client est composée principalement de deux parties:

Une première partie qui permet à l'étudiant de récupérer la liste des cours disponible pour une session bien déterminée. L'étudiant sélectionne la session envisagée (exemple: Hiver, Été, ou Automne). Ensuite, en cliquant sur un bouton *charger*, la liste des cours est récupérée et affichée dans l'interface.

Dans la deuxième partie, l'étudiant remplit et envoie son formulaire d'inscription en cliquant sur un bouton *envoyer*. L'étudiant doit entrer le code du cours auquel il veut s'inscrire. En plus, il doit entrer quelques informations personnelles nécessaires à son identification (*prénom, nom, matricule, email*). L'étudiant est capable de s'inscrire à un seul cours à la fois.

L'interface du client doit ressembler à la figure ci-dessous. Elle doit comprendre un panel avec un bouton *charger* à gauche, et un formulaire avec un bouton *envoyer* à droite.



Pour le code coté client, vous devez suivre le pattern MVC. Pour la partie Modèle du MVC, réutilisez les classes dans le dossier *models* (*https://github.com/OussamaSghaier/IFT1025-TP2-server/tree/main/src/main/java/server/models*).

Vous pouvez vous référer à la démo fournie pour plus de détails. tp2-demo.mp4

Vous devez aussi documenter votre code en créant la javadoc pour votre code coté client (écrire les docstrings sous forme de commentaires et ensuite générer la documentation sous format web sous le dossier **javadoc/client**).

Vous devez aussi fournir un jar exécutable pour votre application client graphique **client_fx.jar**. **Les projets avec des jars non exécutables avec la commande *java –jar* auront une déduction de 30% de la note finale.**

Vous devez également créer une courte démonstration de l'exécution de votre application sous format vidéo. Nous ne vous demandons pas un exposé de la structure de votre code mais une démonstration de l'utilisation de votre application. Mettez le lien de votre vidéo dans le fichier **démo.txt.**

Spécifications:

1. **Chargement des cours:** Quand on clique sur le bouton *charger*, le client doit envoyer une requête au serveur. Le serveur doit récupérer la liste des cours du fichier *cours.txt* pour une session donnée et l'envoyer au client. La liste des cours doit être affichée sur le panel à gauche. Le client affiche ces cours donnés selon la session sélectionnée par l'utilisateur dans la liste déroulante a cote du bouton *charger* (voir figure). Cette liste comprend trois valeurs: Hiver, Automne et Été.

2. **Inscription:** Quand on clique sur le bouton *envoyer*, le client doit envoyer les données du formulaire (*prénom*, *nom*, *matricule*, *email*, *session*, et *code du cours*) au serveur. La session a trois valeurs: Hiver, Automne et Été. Le serveur doit enregistrer l'inscription en ajoutant une ligne dans le fichier *inscription.txt* contenant les données du formulaire (séparées par des tabulations) selon le format du fichier *inscription.txt* indiqué ci-dessus. En plus, le client doit afficher le succès ou l'échec de l'inscription. Un exemple d'échec d'application pourrait être l'absence du fichier *inscription.txt*. Un échec ne doit pas entrainer l'arrêt de l'application. Ceci devrait être géré en utilisant les *exceptions*. Vous devez aussi faire un contrôle de saisie sur la *matricule étudiant* et le *code cours*. La matricule de l'étudiant doit être un entier composé de 6 chiffres.

Critères d'évaluation:

- **Les projets avec des jars non exécutables avec la commande *java –jar* auront une déduction de 30% de la note finale.**
- L'interface graphique sera évaluée si et seulement si le jar **client_fx** est exécutable.
- Votre client GUI (Tâche 3) doit être bien documenté. Vous devez écrire des javadocs pour chaque élément public (classe, interface, méthode, champ et constante) de votre code et vous devez générer des fichiers HTML.

Bonus : 10 points

On veut pouvoir lancer plusieurs clients en parallèle pour simuler le fait que plusieurs étudiants utilisent l'application en même temps. Ainsi, le serveur devrait pouvoir servir les requêtes de plusieurs clients à la fois. Vous pouvez utiliser le *multithreading* pour assurer çeci. A la réception d'une requête (suite à un appui sur le bouton *charger* ou *envoyer*), le serveur doit créer un thread séparé pour traiter la requête. Ceci permettra de traiter la requête d'un client sans bloquer le serveur de servir d'autres clients en parallèle.

Si vous avez implémenter cette fonctionnalité, ajoutez un fichier *bonus.txt* dans votre **zip** et montrez cet aspect dans votre démo. Dans le fichier *bonus.txt*, expliquez clairement les changements que vous avez fait coté serveur