ECOLE SUPERIEURE D'INFORMATIQUE DE GESTION

ÉPREUVE REGROUPÉE

Session de mars-avril 2023

BRANCHE : **ALGORITHMES ET LANGAGE OBJET (ALO)**

CLASSE : **ESIG1 Classes A et B (Groupes 1, 2 et 3)**

DATE : 22 mars 2023

NOM : .....

PRÉNOM : .....

PROFESSEUR : EB MS CV

N° du poste de travail : ESIG-.....

N° de clé USB: .....

Modalités

- Durée : 240 minutes.
- Travail individuel.
- Documentation personnelle (livres, papiers) : Autorisée.
  Documentation électronique (disque dur, clé USB, …) : Autorisée si elle a été recopiée sur **C:\ESIGUsers\Doc *avant*** le début de l'épreuve.
- Tout partage de ressources de votre poste de travail avec le réseau ainsi que toute autre tentative de communication seront considérés comme fraude et sanctionnés comme tels par la note minimale. La présence de supports de stockage (clé USB, …) ou de communication (smartphone, smart watch, smart glasses…) à proximité de votre place de travail constitue également une fraude tout comme l'utilisation d'un assistant I.A.

Démarrage

- Connectez-vous au réseau sur le poste de travail qui vous a été attribué.
- Copiez dans **C:\ESIGUsers** le contenu du dossier réseau qui vous sera indiqué au début de l'épreuve de façon à avoir un répertoire **C:\ESIGUsers**<*vosInitiales*>**_ALO_mars23** contenant l'énoncé et les deux projets IntelliJ, PartieA et OMetro.
- Lisez la totalité de l'énoncé.

Travail à faire

- Pensez à inscrire *votre nom* dans chacun des fichiers .java que vous modifierez.
- Lisez tous les documents fournis.
- La présentation des affichages effectués par vos procédures se conformera strictement à celle des sorties produites données en exemple.
- Faites bien attention aux exemples de sorties produites car ils font partie intégrante de l'énoncé.
- L'évaluation de votre travail portera uniquement sur le contenu de la clé USB qui vous sera fournie quand vous serez prêt à rendre (ou à la fin de l'épreuve). Il est de votre responsabilité de vérifier que l'ensemble de votre travail soit sauvegardé et recopié sur cette clé.

CONSIGNES ALO IMPORTANTES

- Vous êtes encouragés à définir **des méthodes non demandées explicitement**.
- Il est important d'**éviter la répétition de code et la duplication inutile de données.**
- Respecter les conventions de nommage Java, choisissez des noms cohérents et veillez de manière générale à la **lisibilité du code.**

À la fin du travail, vous copierez sur la clé USB fournie le répertoire **C:\ESIGUsers**<*vosInitiales*>**_ALO_mars23**.

C'est à vous de vérifier que le répertoire rendu contient bien la dernière version de votre travail.

Les deux parties de cette épreuve sont indépendantes et peuvent être traitées dans n'importe quel ordre.

Partie A – A comme Abonnements (projet `PartieA` à compléter)

Dans cette partie, nous allons définir successivement trois classes pour décrire les abonnées à des magazines :

- La classe `Magazine` qui décrit un magazine
- La classe `Client` qui décrit un client (abonné)
- La classe `Abo` qui décrit un abonnement.

Pour ces trois classes, vous devrez définir un constructeur et un `toString()` qui produit des chaînes conformes aux affichages attendus (précisés ci-dessous dans chaque classe).

Tous les attributs seront `private` et vous ne définirez ***que*** les accesseurs utiles à votre solution. (Dans le cas où votre solution n'arriverait pas à utiliser des accesseurs, définissez au moins un accesseur pour un attribut à choix.). Les autres méthodes demandées seront `public`.

Les dates seront de type `LocalDate`. La liste sera basée sur `ArrayList`.

Pour tester, une classe `TestAboMagClient` est fournie mais toutes les instructions du `main()` et des méthodes sont en commentaire ! Vous devrez décommenter l'appel et la méthode appelée comme indiqué dans chacune des étapes ci-dessous.

DESCRIPTION DES ATTRIBUTS ET METHODES DES 3 CLASSES

La classe `Magazine`

Un magazine est défini par deux chaînes de caractères : le titre et la fréquence. Pour information (ce sont des valeurs de la fréquence, pas des attributs de la classe), nous allons considérer ces 4 fréquences simplifiées :

- hebdo : un hebdomadaire paraît toutes les semaines,
- bimensuel : un bimensuel paraît toutes les 2 semaines,
- mensuel : un mensuel paraît toutes les 4 semaines,
- bimestriel : un bimestriel paraît toutes les 8 semaines.

Pour tester, décommenter l'appel à `testCreationMagazines()` et ses instructions pour obtenir :

```
Le temps week-end est un hebdo
Femina est un hebdo
TV8 est un hebdo
Bilan est un mensuel
Moins! est un bimestriel
Galaxies est un trimestriel
```

Méthode supplémentaire à définir : Définissez une méthode, `nbSemaines()`, qui, en fonction d'une chaîne représentant une fréquence, renvoie le nombre de semaines entre deux parutions (soit 1, 2, 4 ou 8). Si la fréquence est inconnue, la fonction renverra 0 et affichera « Fréquence inconnue pour » suivi du titre du magazine.

Cette fonction vous servira pour le calcul de la date de fin d'un abonnement (plus loin).

La classe `Client`

Un client possède un identifiant (entier), un nom complet (chaîne) et une liste d'abonnements (basée sur `ArrayList`) vide au départ.

Indication sur le constructeur : Comme la liste des abonnements sera initialisée à vide, il est inutile (donc faux) d'avoir un paramètre pour cet attribut dans le constructeur. Mais il faudra initialiser l'attribut.

Pour tester, décommenter l'appel à `testCreationClientsSansAbo()` et ses instructions pour obtenir :

```
J. Ziegler (client n°122) a 0 abonnement(s) :
P. Fischer (client n°215) a 0 abonnement(s) :
J.-F. Thomas (client n°382) a 0 abonnement(s) :
```

La classe `Abo`

Un abonnement est défini par un n° d'abonnement (entier), un n° de client (entier), un magazine (instance de la classe `Magazine`), une date de début et une date de fin (les deux en `LocalDate`, notez qu'une seule instance de `DateTimeFormatter` dans le `toString()` suffit pour les deux dates).

On demande que le constructeur de cette classe ne prenne que les paramètres suivants :

- le numéro d'abonnement,
- le numéro de client,
- une instance de la classe `Magazine`,
- le jour, le mois et l'année du début de l'abonnement (ces 3 paramètres entiers permettant de construire la date de début),
- ainsi que le nombre de parutions voulues de l'abonnement.

Ce dernier paramètre, le nombre de parutions, mérite une explication. Par exemple pour des abonnements d'un an, on indiquera 52 pour un abonnement à un hebdomadaire, 26 pour un bimensuel, 13 pour un mensuel.

Vous noterez que le constructeur demandé ne prend pas la date de fin d'abonnement en paramètre. En effet nous allons la calculer (dans le constructeur) en ajoutant à la date de début, le nombre de semaines nécessaires pour recevoir le nombre de parutions voulues.
Le nombre de semaines, c'est-à-dire la durée en semaines de l'abonnement, se calcule facilement en multipliant le nombre de parutions voulues par le nombre de semaines entre deux parutions (la fréquence, une caractéristique du magazine).

Ainsi un abonnement de 13 parutions (par exemple) durera 13*1=13 semaines pour un hebdomadaire (1 par semaine), 13*2 = 26 semaines pour un bimensuel (un toutes les 2 semaines), 13*4 = 52 semaines pour un mensuel (un toutes les 4 semaines), 12*8 = 104 semaines pour un bimestriel (un toutes les 8 semaines).

Une fois obtenu le nombre de semaines que durera l'abonnement, il suffit d'appliquer la fonction `plusWeeks` de la classe `LocalDate` à la date de début pour obtenir la date de fin, soit quelque chose comme ceci :

```
dateDeFin = dateDeDébut.plusWeeks(nombreDeSemaines);
```

Pour tester, décommenter l'appel à `testCreationAboDirect()` et ses instructions pour obtenir :

```
Abonnement n°26 (client n°215) pour le magazine TV8
a commencé le 10/12/2022 et se termine le 07/12/2024
Abonnement n°89 (client n°215) pour le magazine Bilan
a commencé le 07/06/2022 et se termine le 09/05/2023
Abonnement n°12 (client n°122) pour le magazine Le temps week-end
a commencé le 01/02/2022 et se termine le 31/01/2023
Abonnement n°15 (client n°122) pour le magazine Moins!
a commencé le 19/02/2021 et se termine le 21/01/2022
Fréquence inconnue pour Galaxies
Abonnement n°12 (client n°382) pour le magazine Galaxies
a commencé le 24/02/2023 et se termine le 24/02/2023
```

Date de début = date de fin car la fréquence est inconnue

Il faut évidemment que la classe `Magazine` soit opérationnelle.
Notez le message d'erreur quand la fréquence ne fait pas partie des quatre répertoriées.

COMPLEMENTS A LA CLASSE `Client`

Maintenant que les abonnements sont définis, on peut ajouter deux autres méthodes à `Client`.

1. Une première méthode d'affichage sans paramètre, `afficherDetailsAbo()`, qui affichera les informations complètes sur les abonnements d'un client, sur 2 lignes comme ceci :

```
Abonnement n°12 (client n°122) pour le magazine Le temps week-end
a commencé le 01/02/2022 et se termine le 31/01/2023
Abonnement n°15 (client n°122) pour le magazine Moins!
a commencé le 19/02/2021 et se termine le 21/01/2022
```

2. Une autre méthode, `ajouterAbo()`, qui va servir à faire évoluer la liste des abonnements d'un client. Il faudra passer en paramètre à cette méthode toutes les informations nécessaires pour compléter l'appel du constructeur d'`Abo` (numéro de l'abonnement, le magazine, jour, mois et année de début de l'abonnement et nombre de parutions voulues). Après avoir créé le nouvel abonnement, il faudra l'ajouter à la liste.
3. Enfin, il faudra adapter le `toString()` de `Client` pour afficher les noms des magazines auxquels le client est abonné.

Pour tester, décommenter l'appel à `testAjoutAbo()` et ses instructions pour obtenir :

```
J. Ziegler (client n°122) a 2 abonnement(s) :
- Le temps week-end
- Moins!

P. Fischer (client n°215) a 2 abonnement(s) :
- TV8
- Bilan

J.-F. Thomas (client n°382) a 1 abonnement(s) :
- Galaxies
```

TEST FINAL

Il est important de ***ne pas modifier*** les méthodes de test fournies mais d'écrire de votre côté le code qui permet de les exécuter sans problème.

La méthode `testTotal()` regroupe tout ce qui est à faire et affiche aussi le détail des abonnements.

```
Le temps week-end est un hebdo
Femina est un hebdo
TV8 est un hebdo
Bilan est un mensuel
Moins! est un bimestriel
Galaxies est un trimestriel
----------------------------------------
J. Ziegler (client n°122) a 0 abonnement(s) :
P. Fischer (client n°215) a 0 abonnement(s) :
J.-F. Thomas (client n°382) a 0 abonnement(s) :
----------------------------------------
Fréquence inconnue pour Galaxies
----------------------------------------
J. Ziegler (client n°122) a 2 abonnement(s) :
- Le temps week-end
- Moins!

Abonnement n°12 (client n°122) pour le magazine Le temps week-end
a commencé le 01/02/2022 et se termine le 31/01/2023
Abonnement n°15 (client n°122) pour le magazine Moins!
a commencé le 19/02/2021 et se termine le 21/01/2022
----------------------------------------
P. Fischer (client n°215) a 2 abonnement(s) :
- TV8
- Bilan

Abonnement n°26 (client n°215) pour le magazine TV8
a commencé le 10/12/2022 et se termine le 07/12/2024
Abonnement n°89 (client n°215) pour le magazine Bilan
a commencé le 07/06/2022 et se termine le 09/05/2023
----------------------------------------
J.-F. Thomas (client n°382) a 1 abonnement(s) :
- Galaxies

Abonnement n°12 (client n°382) pour le magazine Galaxies
a commencé le 24/02/2023 et se termine le 24/02/2023
```

Remarque finale : tous les magazines cités existent vraiment, ont la fréquence indiquée et font partie de la presse romande, à l'exception de `Galaxies`, trimestriel français.

Partie B – Tous OMetro (projet `OMetro` à compléter)

L'Haut-Sannois, une région très pentue, vient de se doter d'une ligne de métro, la ligne MM.

Il s'agit dans cette épreuve de récupérer quelques données sur les arrêts de cette ligne et d'effectuer quelques calculs. Notez que les données seront toutes valides et on ne demande pas de traiter des erreurs (sauf quand c'est Java qui l'exige évidemment).

LES DONNEES

La classe `Arret`

Les données correspondent aux attributs de la classe `Arret` qui est fournie dans le fichier `Arret.java`.

Diagramme de la classe `Arret`

Le fait que la région du Haut-Sannois soit très en pente a une conséquence importante : le temps de trajet entre les deux mêmes arrêts consécutifs n'est pas le même selon qu'on monte du sud vers le nord ou qu'on descende du nord vers le sud. C'est pourquoi il y a deux temps de trajet (en minutes).

A part cela, deux autres attributs donnent le nom de l'arrêt et le nom de la zone où il se trouve.

Tout ce dont vous avez besoin est déjà défini dans cette classe : constructeur, accesseurs et `toString()`.

Le fichier `ligneMM.csv`

L'ensemble des arrêts de la ligne MM figure dans le fichier `ligneMM.csv`, dans l'ordre de la direction sud vers nord. Les attributs sont dans l'ordre du diagramme ci-dessus.

Comme son extension le laisse penser, c'est un fichier dont les données sont séparées par des point-virgule. Pour le lire, vous adopterez donc le principe du double `Scanner` : un `Scanner` pour lire le fichier ligne par ligne et un autre pour découper successivement chaque ligne lue. C'est ce deuxième `Scanner` qui doit utiliser le point-virgule comme délimiteur (avec la méthode `useDelimiter()`).

Une `ArrayList<Arret>` *pour votre programme*

Les données lues à partir du fichier `ligneMM.csv` serviront à compléter l'`ArrayList` appelée `ligneMM` déjà déclarée dans le projet fourni.

C'est à partir de cette liste que vous effectuerez tous les calculs demandés.

CEPENDANT si vous avez des difficultés avec la lecture du fichier ou la création de l'`ArrayList`, utilisez la constante `LISTE_SECOURS_ARRETS` pour affecter `ligneMM`. Ainsi vous ne serez pas bloqué pour faire la suite.

LES QUESTIONS

Question B1 – *calcul des durées totales*

On vous demande de compléter la procédure `afficherDureesTotales()` qui calcule la somme des temps de trajet sur la ligne (entre les deux terminus). Comme il y a cette différence selon la direction, il faut calculer deux sommes, une pour chacune des deux directions, puis les afficher.


```
Durée totale vers sud = 22 minutes
Durée totale vers nord = 41 minutes
```

Question B2 – *calcul des deux durées maximales*

On vous demande de compléter la procédure `afficherDureesMaximales()` qui calcule le plus grand temps de trajet entre deux arrêts consécutifs de la ligne. Comme il y a toujours cette différence selon la direction, il faut calculer deux valeurs, une pour chacune des deux durées. On souhaite connaître aussi les deux arrêts concernés : la position, comptée à partir de 0, et le nom.


```
Durée max vers sud = 3 minutes à l'arrêt en position 13 (Ellinges)
Durée max vers nord = 5 minutes à l'arrêt en position 8 (Panda)
```

Question B3 – *liste des arrêts d'un trajet*

On vous demande de compléter la procédure `testQuestionB3()` dont l'objectif est de tester une fonctionnalité que vous devez écrire dans (au moins) une autre méthode.

On veut connaître les arrêts qui se trouvent entre un point de départ et un point d'arrivée sur la ligne MM, points donnés par le nom des deux arrêts concernés. Il s'agira au final de créer une liste des arrêts intermédiaires entre les deux points (bornes comprises).

Voici, pour illustrer, une sortie attendue :


```
Trajet de Nectars à Flonflons
Liste des 4 arrêts entre Nectars et Flonflons
Nectars
Peutinon
Station
Flonflons
```

Pour réaliser cette sortie, il est fortement recommandé de procéder par étapes :

1. Définissez tout d'abord une fonction qui renvoie la position, dans la liste de tous les arrêts, de l'arrêt dont le nom est fourni en paramètre.
   Pour que votre code soit complet et accepté par le compilateur Java, prévoyez le cas d'un arrêt inconnu (la constante `CODE_ARRET_INCONNU` à utiliser est fournie).
   Ainsi l'arrêt Nectars est en position 2 dans l'`ArrayList` de tous les arrêts et l'arrêt Flonflons en position 5.
   (Attention : si vous ouvrez le fichier `.csv` dans IntelliJ pour vérifier, notez bien que les lignes sont comptées à partir de 1 dans l'éditeur de texte).
2. A partir des deux positions, de départ et d'arrivée, il est facile d'afficher, pour vérifier, les arrêts entre ces deux positions à l'aide d'une boucle dont les bornes sont ces positions.
3. De plus on vous demande de stocker ces arrêts dans une autre liste, qu'il faudra créer, puis remplir (un peu comme vous avez fait pour lire les données). En cas de difficultés, faites déjà les affichages.

4. Enfin il faut afficher les éléments de cette liste selon la présentation demandée dans la méthode de test. Notez que la première ligne des sorties attendues est déjà affichée par la méthode `testQuestionB3()`.

Un dernier point auquel faire attention : le raisonnement exposé ici est valable pour une direction sud vers nord. Dans le cas contraire, il faut parcourir la liste des arrêts à l'envers (en décrémentant, au besoin cf. le rappel technique dans `rappel.pdf` à part) pour avoir les arrêts composant le trajet dans le bon ordre comme le montre cette autre sortie (nord vers sud) :

Sortie attendue

```
Trajet de Poivraz à Station
Liste des 7 arrêts entre Poivraz et Station
Poivraz
Shuv
Panda
Richard
Friponne
Flonflons
Station
```

Pour information, Poivraz est en position 10 et Station en position 4 dans l'`ArrayList` des tous les arrêts.

Question B4 – *détail et prix d'un trajet*

On vous demande de compléter la procédure `testQuestionB4()` dont l'objectif est de tester une fonctionnalité que vous devez écrire dans (au moins) une autre méthode.

Cette fonctionnalité s'appuie sur la liste d'arrêts entre deux points construite avec le code testé par la méthode précédente `testQuestionB3()`. Vous réutiliserez donc les méthodes définies dans cette question pour calculer la position et remplir la liste des arrêts entre ces positions et l'afficher.

CEPENDANT si vous n'avez pas réussi à finir la question B3, vous avez deux listes de secours (définies en constantes) à votre disposition pour réaliser quand même cette question B4.

A partir de la liste des arrêts qui se trouvent entre un point de départ et un point d'arrivée, on veut connaître le prix du billet pour ce trajet.

Pour cela il faut connaître le nombre de zones traversées. La zone où se situe l'arrêt de départ compte pour 1 zone tout comme la zone de l'arrêt d'arrivée.

L'algorithme recommandé est simple : il suffit de noter dans une variable la zone de l'arrêt de départ avant de parcourir la liste, et ensuite d'examiner tous les arrêts intermédiaires un par un jusqu'à l'arrêt d'arrivée y compris. Dès qu'on rencontre un nom de zone différent, cela fait une zone de plus au compteur et il faut noter le nom de la nouvelle zone traversée dans la variable puisqu'on a changé de zone.

Au final on connaîtra grâce au compteur le nombre de zones traversées et il suffit de se servir de ce nombre pour retrouver le tarif (par indiçage direct) grâce à l'`ArrayList` fournie en constante *TARIF_PAR_NB_ZONES*.

Sortie attendue

```
Prix pour le trajet de Nectars à Flonflons
Liste des 4 arrêts entre Nectars et Flonflons
Nectars
Peutinon
Station
Flonflons

Tarif appliqué pour 3 zones traversées = 5.0
```

```
Prix pour le trajet de Poivraz à Station
Liste des 7 arrêts entre Poivraz et Station
Poivraz
Shuv
Panda
Richard
Friponne
Flonflons
Station

Tarif appliqué pour 4 zones traversées = 5.5
```

Remarque finale : il n'est pas impossible que la ligne MM ressemble à la ligne du M2 de Lausanne…