ECOLE SUPERIEURE D'INFORMATIQUE DE GESTION

EPREUVE REGROUPÉE
Juin 2023
Enoncé

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

Classes : **ESIG 1 Classes A et B (groupes 1, 2 et 3)**

Date : **12 juin 2023**

Nom étudiant-e : .....

Prénom étudiant-e : .....

Professeur-e : .....

N° poste de travail : .....

N° clé USB : .....

Modalités

- Durée : 240 minutes.
- Travail individuel.
- Documentation personnelle (livres, papiers) : Autorisée.
- Documentation électronique (disque externe, 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 de la fraude et sanctionnés comme tels par la note minimale.

Démarrage

1. Après vous être connecté au réseau sur le poste de travail qui vous a été attribué, copiez dans **C:\ESIGUsers** le contenu du dossier dont le chemin vous sera indiqué au début de l'épreuve de façon à avoir un répertoire **C:\ESIGUsers\G_AVA_Juin2023** contenant le projet de l'épreuve.
2. Ce répertoire contient cet énoncé (un fichier PDF) et le répertoire de projet IntelliJ IDEA qui, lui, s'appelle simplement **G_AVA**.
3. Ouvrez ce répertoire de projet (**G_AVA**) dans IntelliJ IDEA.
4. Commencez par ***lire*** tous les documents fournis.

Consignes importantes !

- Avant de modifier un fichier source, inscrivez dans un commentaire d'en-tête vos ***prénom et nom***.
- Certaines méthodes vous sont fournies. Ne complétez que celles qui sont marquées, en commentaire, comme « à compléter » ou « à modifier ». Quand vous complétez une méthode, vous pouvez enlever ce commentaire.
- Compléter une méthode, c'est écrire du code dans le corps de la méthode mais c'est aussi ***ajouter des paramètres*** si nécessaire.
- Vous êtes encouragés, parfois contraints, à définir **des méthodes non demandées explicitement**, y compris dans les classes de données fournies, par exemple des accesseurs. Ceci vous rapporte des points (pour autant que la création des méthodes soit pertinente).
- Pour certaines questions une approche particulière peut être demandée. Si vous adoptez une autre approche, qui produit les bons résultats (ou des résultats partiels) et qui peut s'adapter à des données différentes, vous recevrez tous les points (ou une partie des points ; par exemple si vous recalculez des résultats qui auraient pu être stockés.)
- La présentation des affichages se conformera à celle des sorties produites données en exemple. Examinez attentivement ces **sorties produites** pour chaque question (dans l'énoncé) pour déterminer les détails de la présentation demandée.
- Faites-en sorte de supprimer tout code inutile et soignez la présentation. N'hésitez pas à commenter votre code pour en clarifier la compréhension.
- L'évaluation de votre travail portera sur plusieurs fichiers `.java`. A la fin de l'épreuve, vous copierez votre répertoire de projet sur une clé USB qui vous sera fournie.

Image de fin tirée de https://bertrandpiccard.com/exploration-fr/solar-impulse

G_AVA : Geneva _ AVionics Alliance

Contexte général de l'épreuve

Nous sommes en 2050.

Malgré les difficultés climatiques, le secteur de l'aviation a connu un regain d'intérêt grâce aux technologies des avions électriques et solaires.

Un groupe de camarades a constitué une alliance pour mettre en œuvre ces techno–logies à Genève. Fidèle à une vision humaniste du travail, ce groupe fait appel à des program–meurs humains pour développer certaines fonctionnalités autour de ses ressources humaines.

Plus précisément

Il existe deux sortes de profil parmi les employés de G_AVA.

Certains employés travaillent ***au sol***, comme des agents de service (comme l'entretien ou la sécurité) et des agents administratifs. Ce profil d'employés concernera ***exclusivement*** la partie A consacrée à la construction d'une hiérarchie d'héritage et son utilisation.

D'autres employés font partie du personnel ***navigant***, comme les pilotes et les agents de bord (`AdB` par la suite pour gommer des formules comme hôtesse de l'air et steward jugés désuètes en 2050). Cette catégorie concernera ***exclusivement*** la partie B consacrée à quelques petits calculs.

Toutefois la hiérarchie d'héritage sera ***commune*** à ces deux parties car tant les personnels au sol que les personnels navigants sont des employés de G_AVA.

Ci-dessous le diagramme de classe avec les attributs des classes, les explications venant dans chacune des parties concernées. En particulier les 7 classes sont fournies mais certaines (`AuSol`, `DeService`, `Admin`) sont vides car ce sera à vous de les compléter.

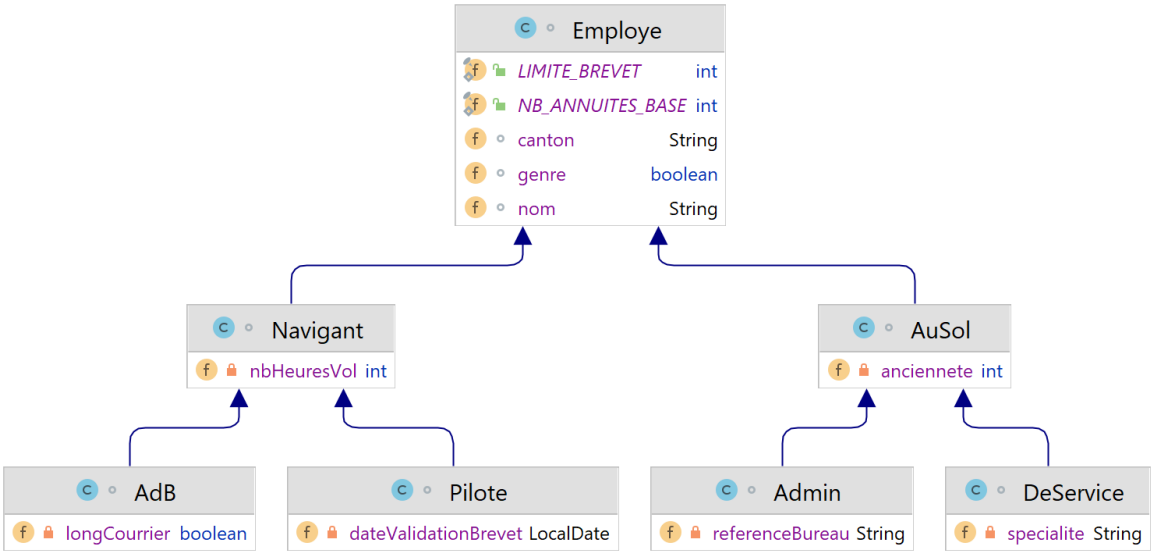

Figure 1 – diagramme de toutes les classes (ordre des attributs non significatif)

Important

Tous les attributs de toutes les classes étant `private`, c'est à *vous* d'introduire les accesseurs nécessaires.

La classe `Employe`

En plus de deux constantes expliquées plus loin, du constructeur et du `toString()`, les trois attributs, tous `private`, sont fournis et correspondent :

- au nom (`nom` de type `String`),
- au genre (`genre` de type `boolean`, `true` pour femme, `false` pour homme),
- au canton ou département de résidence (`canton` de type `String`).

Partie A – les employés au sol

L'objectif de cette partie est de compléter les classes actuellement vides (`AuSol`, `DeService` et `Admin`) avec les attributs et méthodes demandés, afin de lire les données associées de façon à obtenir une liste d'instances pour produire un récapitulatif.

Exercice 1A – Compléter les classes

Il faudra d'abord ajouter, partout où c'est nécessaire, les instructions permettant de réaliser l'héritage entre les classes `Employe`, `AuSol`, `DeService` et `Admin` (cf. Figure 1 ci-dessus).

Ensuite vous ajouterez les attributs (tous `private`).

Concernant les attributs, chaque employé `AuSol` dispose d'un attribut entier qui donne le nombre d'années déjà travaillées (cela nous servira pour le récapitulatif).

Chaque employé `DeService` dispose d'une spécialité comme *mécanicien*, *sécurité* ou *bagagiste* ; une chaîne de caractères donc. Dans ce travail nous nous limiterons à ces 3 valeurs pour l'attribut mais ce ne sont pas les seules possibles !

Chaque employé `Admin`-istratif dispose d'une référence de bureau. C'est une chaîne de caractères et nous verrons plus bas comment elle est construite.

Il faudra alors définir les constructeurs de ces 3 classes. N'oubliez pas que ce sont des instances d'`Employe` qui ont *aussi* les attributs communs à tous les `Employe`s.

De plus, pour obtenir l'affichage de vérification (cf. plus loin), il faudra définir des `toString()` en vous appuyant sur celui fourni dans la super-classe `Employe`.

Exercice 2A - Lire les données

Vous compléterez la méthode `testPartie_A()`, partiellement fournie dans `Run_G_Ava`.

Les données à lire se trouvent dans le fichier fourni `GroundStaff.csv` qui est un fichier texte où les éléments de chaque ligne sont séparés par un point-virgule (comme d'habitude pour un `.csv`).

La structure du fichier est très simple : une ligne d'en-tête à ignorer puis des lignes où se trouvent, comme l'indique l'en-tête :

- le nom,
- le genre, sous la forme H ou F, correspondant respectivement à une valeur `false` ou `true` de l'attribut,
- le canton ou département de résidence,
- le type de personnel au sol, soit une chaîne valant *DeService* ou *Admin*,
- l'attribut donnant le nombre d'années déjà travaillées.

Ensuite, selon le type de personnel au sol, on aura :

- Pour les personnels de service, la spécialité (une chaîne de caractères)
- Pour les personnels administratifs, on a choisi de donner 3 informations distinctes dans le fichier :
  - le nom du bâtiment (une lettre),
  - le numéro d'étage (pas plus de 9 étages),
  - le numéro de bureau à proprement parler (pas plus de 99 bureaux par étage).

Toutefois comme l'attribut, la référence du bureau, est une chaîne de caractères unique, on l'obtient en concaténant les 3 informations lues. Par exemple si on lit F comme nom du bâtiment, 1 comme étage et 09 comme numéro de bureau, la référence du bureau est tout simplement F109 !

Les données lues seront stockées dans une liste d'instances de `AuSol` par la fonction `lireDonneesPartieA()` (à compléter) appelée dans `testPartieA()` dans `Run_G_Ava`.

Pour vérifier que les données sont bien lues, une méthode `vérificationLecture()` est fournie et même déjà appelée dans `testPartieA()`. Cf. Annexe Partie A pour ces sorties.

Exercice 3A – Utiliser la liste pour un récapitulatif

L'objectif du récapitulatif est de lister pour tous les personnels au sol le nombre d'années leur restant à travailler.

Pour obtenir ce récapitulatif, vous commencerez par définir autant de méthodes-fonctions `calculerAnneesRestantes()` que strictement nécessaire tout en respectant les principes de la programmation objet. Ces méthodes calculeront le nombre d'années restant à travailler pour une instance donnée d'une classe concernée sachant que :

- Pour les personnels `AuSol` en général, le mode de calcul standard est donné par la différence entre le nombre d'annuités de base et le nombre d'années déjà travaillées. Le nombre d'annuités de base est le même pour tout le monde (47 en 2050) et il est déjà défini comme constante publique dans la classe `Employe`. Pour l'utiliser, il faut préfixer le nom de la constante par `Employe`. comme ceci : `Employe.NB_ANNUITES_BASE`

- Mais, pour les personnels `DeService`, Geneva _ Avionics Alliance a décidé d'octroyer des années de compensation pour la pénibilité du travail qui seront *soustraites* du nombre d'années standard, nombre qui est susceptible de changer à terme.
  Par exemple, si un bagagiste doit travailler encore 10 ans, en fait `calculerAnneesRestantes()` renverra 8 pour lui car il a 2 ans de compensation.

  Dans ce travail nous ne considérerons que les 3 cas des employés *mécanicien*, *sécurité* ou *bagagiste*. Mais pour être général vous devrez définir vous-même une structure de données qui permette d'associer à une spécialité le nombre d'années de compensation selon la tabelle ci-dessous :

spécialité	années octroyées
mécanicien	3
sécurité	1
bagagiste	2
…	…

  Cette structure de données peut être, à choix, un tableau (à trois éléments), une liste ou une `HashMap` (cette dernière option étant celle qui est recommandée). Si vous n'arrivez pas à définir une structure de données, codez « en dur » les valeurs (mais au prix d'une perte de points).

  Une fois la structure de données définie et initialisée avec les valeurs demandées, vous devrez l'utiliser dans votre méthode.

Enfin vous afficherez la liste des employés avec le nombre d'années leur restant à travailler dans `testPartieA()` appelée dans `Run_G_Ava`. Si la fonction renvoie une valeur nulle (0), cela signifie que l'employé a atteint la retraite.

Résultats attendus sur deux colonnes pour économiser le papier :

```
Les années restant à travailler pour nos collaborateurs au sol
Maurice Ritter : 36                     Valentina Lima : 21
Käte Zobel : 27                         Hans-Martin Lübs : 37
Natali Hellwig : 37                     Camilla Nordio : 30
Pauline Guyot : 26                      Milena Nogueira : retraite !
Luc Bertin : 8                          Morris Brennan : 37
Rafael Pinto : 42                       Stéphane Fabre : 29
Paulina Volterra : 27                   Beranger Suter : 32
Nicolas Da Silva : retraite !           Lili Voegeli : 42
Benjamin Pages : 42                     Lou Baumann : retraite !
Katharina Schenk : 33                   Sophie Wenger : 24
Tomas Hörle : 39                        Sonja Liechti : retraite !
Patricia Blanchet : 42                  Charline Leuenberger : 39
Gustavo Offredi : retraite !            Ludovic Baumann : 32
Hugues Masson : 26                      Alessio Sutter : 2
Susan Dos Santos : 3                    Luc Ackermann : 16
Marina Almeida : 2                      Morgane Lehmann : 17
```

Partie B – les employés navigants

Tout se passe désormais à partir de `testPartie_B()`. Dans cette partie la lecture des données est ***déjà*** effectuée. Vous utiliserez donc la liste `airCrew` déjà remplie pour vous. C'est un `ArrayList<Navigant>`.

Un appel à `vérificationLecture()` est déjà programmé mais pour retrouver l'affichage de l'Annexe Partie B, vous devrez ajouter certains `toString()`. Notez que dans la classe `Pilote`, une constante `DATE_FORMATTER` est fournie pour faciliter le formatage de la date d'obtention du brevet.

Exercice 1B – proportion des genres

Il s'agit de calculer à partir de la liste `airCrew` la proportion d'hommes et de femmes (simplification probablement discutable en 2050) et les afficher.
On rappelle à toutes fins utiles qu'une proportion en pourcentage se calcule en divisant le nombre considéré par le nombre total et en multipliant par 100.
Pour arrondir votre résultat à l'affichage, utilisez la fonction fournie `arrondi()`.

Complétez la méthode `exo1B_afficherProportions()` (dans `Run_G_Ava`).

Résultats attendus :

```
Parmi les 19 membres du personnel navigant,
il y a 52,6% de femmes et 47,4% d'hommes selon leur état-civil actuel
```

Exercice 2B – Le meilleur (avant la fin)

On souhaite connaître le pilote le plus expérimenté, c'est-à-dire celui qui a le plus grand nombre d'heures de vol. Il faudra pour cela ***impérativement*** compléter la définition de la fonction fournie `pilotePlusExperimente()` qui renvoie un pilote (évidemment). Le résultat de cette fonction sera utilisée dans `exo2B_afficherPlusExperimente()` qui procédera à l'affichage demandé (ci-dessous).

Comme l'exercice 5B (plus loin) implique une modification de la liste, vous n'obtiendrez pas le même résultat si jamais vous exécutiez la méthode `exo2B_afficherPlusExperimente()` après avoir modifié la liste, plutôt qu'avant. Donc les résultats attendus donnent les deux possibilités mais vous n'en produirez qu'une !

Résultats attendus :

(*avant modification, résultat normal*)

```
Le pilote le plus expérimenté est Jean Ferrard avec 1099 heures de vol
```

(*si jamais après modification*)

```
Le pilote le plus expérimenté est Jean Ferrard avec 1126 heures de vol
```

Exercice 3B – Revalidation des brevets

Tous les 5 ans, les pilotes doivent suivre une formation et des tests en vue de valider à nouveau leur brevet.

On souhaite connaître la liste des pilotes dont le brevet date de 5 ans ou plus. Pour cela il faudra calculer « l'âge » du brevet à l'aide de la fonction `age` fournie qui prend en paramètre une instance de `LocalDate`. Une constante `LIMITE_BREVET` est définie dans `Employe` (pour l'utiliser, écrivez `Employe.LIMITE_BREVET`).

La méthode de test `exo3B_afficherRevalidations()` est fournie et vous devrez compléter la fonction `listerRevalidations()` qui renvoie une `ArrayList<Pilote>`.

Résultats attendus :

```
Liste des pilotes devant valider à nouveau leur brevet :
Luisa Vitali - brevet en date du 06.06.2018
Ferdinand Rivat - brevet en date du 04.04.2018
Jean Ferrard - brevet en date du 13.11.2017
```

Exercice 4B – répartition selon les lieux de résidence

Geneva _ Avionics Alliance souhaite avoir une vue d'ensemble des lieux de résidence de son personnel navigant.

La forme demandée du résultat est une `HashMap` associant cantons (et départements de France voisine) avec le nombre d'employés navigants résidant dans ce lieu. D'autres formes (listes parallèles par exemple) sont acceptées mais moins valorisées dans l'évaluation.

Comme il est plus simple de construire la `HashMap` demandée quand on connaît la liste des clés possibles, c'est-à-dire les lieux de résidence, une liste `LISTE_LIEUX` et un tableau `TAB_LIEUX` sont fournis (des constantes déjà définies, dans `Run_G_Ava`) et vous pouvez utiliser l'une ou l'autre. Mais il y aura un bonus si vous arrivez à vous passer de cette liste de lieux déjà établie.

Complétez la méthode `exo4B_afficherRepartition()` afin d'obtenir ce qui suit.

Résultats attendus :

```
Liste des cantons/départements de résidence
[1, 74, GE, VD]

Répartition
1 : 3
74 : 5
GE : 8
VD : 3
```

Exercice 5B – Mise à jour des heures de vol

L'objectif de cette question de mettre à jour le nombre des heures de vol de chaque personnel navigant à partir d'un tableau hebdomadaire (à deux dimensions) fourni (cf. ci-dessous).

Le tableau à deux dimensions donne pour chaque membre du personnel navigant le nombre d'heures volées pour chaque jour de la semaine (il y en 7 car les avions volent aussi le week-end !). Au besoin il y a dans `Run_G_Ava` une constante `NB_JOURS_SEMAINE`.

Il est déjà automatiquement lu et affiché dans la méthode `exo5B_effectuerMiseAJour`()

```
Tableau hebdomadaire
-------------------------------------------------------------
| lundi  | mardi  |mercredi| jeudi  |vendredi| samedi |dimanche|
-------------------------------------------------------------
|      6 |      0 |      0 |      6 |      2 |      2 |      2 |
|     -1 |     -1 |      0 |      4 |      3 |     -1 |     -1 |
|      8 |      0 |      7 |      0 |     -2 |      2 |      7 |
|      6 |      0 |      4 |      4 |      0 |      4 |      0 |
|      4 |      0 |      7 |      6 |      0 |     -1 |     -1 |
|      0 |      6 |     -2 |      0 |      6 |      0 |      8 |
|      7 |      0 |     -3 |     -3 |     -3 |      0 |      3 |
|      3 |      4 |     -2 |      0 |      4 |      5 |      8 |
|      8 |      0 |      3 |      0 |      0 |      2 |      8 |
|      0 |     -1 |     -1 |     -1 |     -1 |     -1 |     -1 |
|      4 |     -1 |     -1 |      5 |      7 |      0 |      0 |
|      8 |      5 |      5 |      1 |     -1 |      0 |      3 |
|     -1 |     -3 |      7 |      5 |      0 |      4 |      4 |
|     -1 |     -1 |     -1 |     -1 |     -1 |     -1 |     -1 |
|     -1 |      6 |      5 |      0 |      4 |      3 |      4 |
|      2 |      3 |      0 |      8 |     -1 |      3 |      5 |
|      4 |     -1 |      6 |      4 |      0 |      7 |      6 |
|      0 |      3 |      8 |      0 |      4 |      3 |      7 |
|     -3 |     -3 |     -3 |      4 |      0 |      4 |      5 |
-------------------------------------------------------------
```

L'***ordre*** des employés entre le tableau fourni et la liste des personnels navigants est ***rigoureusement*** le ***même*** : ils sont parallèles selon la terminologie informatique. Vous devez vous appuyer sur cela pour réussir l'exercice.

Notez que certaines données sont négatives car elles correspondent à des cas particuliers pour lesquels des constantes ont été définies dans `Run_G_Ava` :

- -1 pour un jour où l'employé en congé ;
- -2 pour un jour où un employé est absent pour raison de santé ;
- -3 pour un jour pour un employé absent pour raisons administratives (formation par ex.)

Il ne faudra évidemment pas additionner les nombres négatifs !

On veut également un décompte des jours correspondant aux trois cas particuliers.

En revanche il est normal d'avoir des valeurs à zéro : il s'agit de jours d'astreinte où l'employé doit être présent ou disponible (par exemple pour remplacer un collègue) mais n'aboutissent à des heures de vol. Ces jours ne requièrent aucun traitement particulier.

Il y a plusieurs façons d'aborder le problème.

Par exemple, il est possible de tout traiter dans une unique double boucle imbriquée qui parcourt les lignes du tableau à deux dimensions.
Pour une ligne on va additionner les heures volées et décompter les cas particuliers.
Une fois calculée la somme des heures volées, il suffit de mettre à jour l'attribut du `Navigant` correspondant à la ligne. Vous aurez alors besoin de définir un accesseur en écriture, appelé aussi *setter* (rappel : c'est n'importe quelle procédure qui prend en paramètre une valeur pour changer celle d'un attribut).

Autre possibilité : créer un tableau ou liste des heures à ajouter en parcourant le tableau à deux dimensions et effectuer ensuite les ajouts dans un parcours de la liste des `Navigant`s.

Pour obtenir l'affichage ci-dessous, il faudra compléter l'appel à la méthode `afficherStatHebdo()`.
L'appel actuellement fourni est `afficherStatHebdo(airCrew, 0, 0, 0);` Il faudra remplacer les `0` avec vos compteurs pour les jours correspondant aux trois cas particuliers.

Si les deux colonnes de chiffres sont identiques, c'est que vous n'avez pas (encore) modifié la liste.

Résultats attendus :

```
Evolution des heures de vol
Semaine du lundi 6 au dimanche 12 juin 2050
Congés : 26 jours; Santé : 3 jours; Formation : 7 jours
-------------------------------------
|   Noah Hunziker |   982  ->  1000 |
|   Thomas Arnold |  1011  ->  1018 |
|      Bryan Jost |   950  ->   974 |
|  Valérie Blaser |   885  ->   903 |
|    Blanche Roth |  1035  ->  1052 |
|   Luc Ackermann |   780  ->   800 |
|      Yves Egger |  1012  ->  1022 |
| Morgane Lehmann |   924  ->   948 |
|    Luisa Vitali |  1050  ->  1071 |
|    Davide Negri |  1049  ->  1049 |
|    Giulia Rossi |   922  ->   938 |
|   Denise Vitale |  1039  ->  1061 |
| Ferdinand Rivat |  1072  ->  1092 |
|    Giulia Villa |   941  ->   941 |
|    Aurora Leone |   671  ->   693 |
| Azzurra Sartori |   556  ->   577 |
|    Jean Ferrard |  1099  ->  1126 |
| Samuel Esposito |   929  ->   954 |
| Martine Laurent |   904  ->   917 |
-------------------------------------
```