2024 -2025

TP 3 - JAVA

Exercice 1: classe de base

1. Créez un nouveau projet de type *Java application*.

Dans un package "IsiHeritage.java" créez une classe **Personne** avec les attributs (private? protected?...) :

- **nom** (chaîne de caractères),
- **prenom** (chaîne de caractères);
- **anNaissance** (entier primitif)

Dans la classe Personne créez:

- un constructeur permettant d'initialiser le nom, le prénom et la date de naissance
- d'autres constructeurs
- une méthode **calculerAge** permettant de retourner l'âge (à partir de 2024)
- une méthode **afficher** qui affiche sur la console *"Nom : xxx, Prénom : xxx, Age : xxx"*

2. Dans une autre classe, **Start**, placez votre point d'entrée ("**main**").

3. Créez 2 instances de la classe (Personne) en mettant en jeu leurs constructeurs.
   Invoquez ensuite les méthodes afficher() et calculerAge(), sur chacune des instances créées.

4. Essayez d'afficher et modifier les attributs des 2 instances à partir de la classe Start. Est-ce possible ?
5. Sinon, rajouter dans la classe Personne les setters et les getters nécessaires

Exercice 2 :sous classe

1. Créez une nouvelle classe **Programmeur** qui hérite de Personne.

Les champs propres à la classe Programmeur sont:

- **prime** : float
- **salaire**: float
- **pseudo** : chaîne de caractères

2. Les constructeurs de la classe Programmeur

- Programmeur() // Constructeur d'initialisation
- Programmeur(X, Y) // X, Y = salaire, prime
- Programmeur(X,Y,Z) // X,Y,Z= salaire, prime, pseudo
- Programmeur(TOUS LES PARAMETRES POSSIBLES)

3. Les méthodes de la classe Programmeur

- **afficher**() --> affiche les informations propres au Programmeur
- **getSalaire**() --> retourne le salaire + la prime

Questions

- Avez-vous remarqué quoique ce soit de particulier lors de la création des constructeurs? Expliquez
- Peut-on utiliser à la fois **super** et **this** dans une même méthode?
- Comment faire pour, à partir d'une instance de Programmeur, afficher les informations communes à toutes les instances de Personne?
- Comment faire pour créer une méthode afficher() dans Programmeur qui n'affiche que les informations propres aux instances de Programmeur?

4. Tout objet peut appeler la méthode **toString**

- A quoi sert cette méthode?
- Comment faire pour désormais utiliser toString pour renvoyer la chaine de caractère qui sera utile pour la méthode ***afficher()***?

5. Au lieu de valeurs rentrées en "dur", utilisez des valeurs tapées sur la console pour initialiser les attributs nom, prénom et année de naissance.

- Utilisez la classe **Scanner** pour réaliser cela.
- Sur la console, demandez à l'utilisateur de renseigner successivement : le nom, le prénom, l'année de naissance, le salaire, la prime, le pseudo
- Au final instanciez des objets Programmeur à l'aide des informations saisies sur la console. Utilisez en ce sens le constructeur le plus riche.
- Utilisez la classe start pour tester les 2 classes Personne et Programmeur

Affichage souhaité :

```
Entrez le nom : X
Entrez le prenom : Y
Entrez l'annee de naissance : 2004
Entrez le pseudo : dev
Entrez le salaire : 4500
Entrez la prime : 300

>>>>> BIENVENUE ! <<<<<
Nom : X
Prenom : Y
Pseudo : dev
Age : 20
Salaire total : 4800
```

Exercice 3:

I. Écrire une classe Véhicule:

1. La classe Vehicule a deux attributs : le nombre de roues **nbRoues** et le remplissage en passagers e**stComplet** (complet ou pas).

2. Créez un constructeur à un seul paramètre entier. A la création de l'objet, le remplissage ne doit pas être complet !).

3. Créez une méthode **roule**() qui se contente d'afficher le remplissage entre parenthèses, puis " Les **N** roues tournent." (***N*** *étant le nombre de roues*).

Un exemple d'exécution : `(false) Les 6 roues tournent`

4. Créez une procédure **freine()** qui permet d'afficher "`J'appuie sur le frein`.".

II. Écrire une classe Avion :

1. C**réez** une classe **Avion.** Bien évidemment Avion une sorte de **Véhicule**. Cette classe a un attribut : le nombre de réacteurs **nbReact**.

3. Créez le constructeur de cette classe (on suppose que par défaut, le remplissage n'est pas complet).

4. Créez une procédure **vole()** qui permet d'afficher "`Je vole`.".

5. Redéfinir la procédure **freine()** pour qu'elle affiche "`J'inverse les` **`N`** `réacteurs`" (***N*** *étant le nombre de réacteurs*) avant d'effectuer la procédure de freinage "normale" (c'est-à-dire celle de "Vehicule").

6. Redéfinir la procédure **roule**() pour qu'elle affiche "`Je pousse les réacteurs`." seulement si l'avion est complet, avant, dans tous les cas, d'effectuer la procédure de roulage "normale". N'y aura-t-il pas un problème de compilation ? Lequel ? Pour le résoudre :

7. Ajoutez dans **Vehicule** un accesseur estComplet (et servez-vous en !).

8. Ajouter dans **Vehicule** une méthode **invComplet()** qui inverse l'état de remplissage à chaque appel (*complet* <--> *non complet*).

III. Écrire une classe Utilisation :

1. Comme d'habitude, cette classe ne sera pas instanciée et ne sert qu'à contenir une procédure essai *(pratique à utiliser)* pour mettre en œuvre les 2 classes précédentes. Cette procédure devra :

2. Déclarer/créer un objet **Vehicule** à 6 roues, puis le faire rouler et freiner.

3. Déclarer/créer un objet Avion à 4 roues et 2 réacteurs, le rendre complet, puis le faire rouler, voler, et freiner.

4. Déclarer une référence de type **Vehicule** et lui affecter un nouvel objet Avion à 8 roues et 4 réacteurs, et le faire rouler.
Peut-on le faire voler ? Peut-on l'affecter directement à une nouvelle référence d'Avion ? Écrire ce qu'il faut pour que la nouvelle référence d'Avion pointe sur l'objet Avion précédemment créé, et le faire voler et freiner.

5. L'exécution de la procédure essai devrait donner les affichages suivants :

```
(false) Les 6 roues tournent.
J'appuie sur le frein.
Je pousse les réacteurs.
(true) Les 4 roues tournent.
Je vole.
J'inverse les 2 réacteurs.
J'appuie sur le frein.
(false) Les 8 roues tournent.
Je vole.
J'inverse les 4 réacteurs.
J'appuie sur le frein.
```

IV. Comptage : ajouter dans la classe **Vehicule**:

1. Un nouvel attribut : le nombre d'objets **Vehicule** créés (on ne tiendra pas compte des éventuels objets détruits).

2. L'initialisation de cet attribut à 0
3. L'incrémentation de cet attribut. À quel endroit ?
N'y a-t-il pas un problème ? D'ailleurs, combien y aura-t-il de compteurs si on crée 3 objets ? Et combien vaudront-ils ?
Il nous faudrait un seul compteur pour toute la classe, donc un attribut "de classe" donc **un attribut statique**.

4 Comment peut-on initialiser un tel attribut ? Grâce à un "bloc statique" qui se situe en dehors de toute méthode *(typiquement entre les attributs et les constructeurs)* :
static { aNombre = 0; } qui peut contenir plusieurs instructions. *Pourquoi this.aNombre = 0; n'a-t-il pas de sens ?*

5 Comment une autre classe pourra-t-elle accéder à cet attribut privé ? Ajouter l'accesseur qui convient.

6 Ajouter aussi dans la classe Utilisation des affichages du nombre d'objets **Vehicule**, aux 4 endroits appropriés.

V. Constante : ajouter dans la classe **Vehicule** :

1 Un nouvel attribut constant : le nombre minimum de roues que le constructeur pourra utiliser pour initialiser l'attribut. Pour signaler un attribut constant, on fait précéder son type du mot final, comme pour les paramètres. D'autre part, un attribut constant (appelé ' une constante') s'écrit toujours entièrement en majuscules, éventuellement séparées par un _ , ici MIN_ROUES par exemple.

2 Y a-t-il un intérêt à dupliquer cette constante dans tous les objets de la classe **Vehicule**?
3 Une constante reçoit généralement sa valeur dès sa création en ajoutant = *valeur* à la fin de la déclaration. En résumé :
private static final int NOM_CONSTANTE = *valeur*;

4 Si la constante peut avoir un intérêt en dehors de la classe, il est même permis de la déclarer publique, puisqu'il n'y a aucun risque qu'elle soit modifiée. *Que faudra-t-il écrire pour utiliser cette constante depuis une autre classe ?*
À partir du moment où une constante a été déclarée, on doit toujours l'utiliser dans le programme et ne plus utiliser directement la valeur qu'elle contient ; ainsi, si on décide de changer cette valeur, la modification du programme sera aisée.
Pourquoi un simple remplacer tout *dans un logiciel de traitement de texte ne conviendrait-il pas ici ?*

5 Ajouter ce qu'il faut dans le code pour s'assurer qu'aucun **Vehicule** ne pourra avoir un nombre de roues inférieur à MIN_ROUES.