Les classes, on en fait des wagons

POnGL, TD1, héritage, redéfinition, transtypage, liaison dynamique.

1. Présentation

On va construire une hiérarchie de classes représentant les éléments roulants d'une hypothétique compagnie ferroviaire. Les éléments décrits seront les suivants :

- La classe générale `EltTrain` regroupe tous les éléments de train (on peut imaginer qu'un train est une liste d'objets de classe `EltTrain`). Chaque élément possède un nom (type `String`) et a un certain poids.
- Une `Locomotive` est un élément de train pouvant tracter d'autres éléments dans une certaine limite de poids.
- Un `Wagon` est un élément de train pouvant accueillir un certain nombre de passagers.
- Un `WagonBar` est un wagon ne pouvant pas accueillir de passagers (les passagers doivent avoir un siège réservé dans un autre élément).
- Un `Autorail` est un élément de train pouvant tracter d'autres éléments dans une certaine limite de poids et pouvant accueillir un certain nombre de passagers.

On veut de plus avoir les comportements suivants :

- Il est impossible de créer un élément directement avec le constructeur de la classe `EltTrain` (on peut seulement passer par l'un des quatre autres types d'éléments).
- Les éléments accueillant des passagers offrent une méthode `boolean reserver(int n)` qui renvoie `true` et enregistre que `n` places supplémentaires sont réservées si la capacité maximale n'a pas été dépassée, et renvoie `false` sinon.
- Tous les éléments possèdent une méthode `int calculerPoids()` qui estime le poids total de l'élément, passagers compris (on comptera 75kg par passager).
- Chaque nouvel élément créé reçoit un nom, qui est différent des noms des autres éléments déjà créés.
- Les éléments moteurs (ceux qui peuvent tracter d'autres éléments) doivent être inspectés tous les 30 jours. Chacun est associé à un compteur indiquant le nombre de jours écoulés depuis la dernière inspection. Une méthode `boolean peutRouler()` indique si un élément moteur a été inspecté suffisamment récemment pour être autorisé à rouler, et une méthode `void inspectionOK()` remet à zéro le compteur.

2. Organisation des classes

Le but de cet exercice est de proposer une organisation des 5 classes précédentes, qui respecte les relations de spécialisation/généralisation suggérées par l'énoncé, et qui évite le code redondant.

- Décrivez les relations d'héritage entre les 5 classes précédentes, en précisant le cas échéant quelles classes sont abstraites (`abstract`). Pour une bonne organisation, il pourra être judicieux d'introduire de nouvelles classes, qui seront soit directement insérées dans l'arbre d'héritage, soit utilisées par délégation.
- Pour chaque classe, donnez la liste de ses attributs, en précisant leur portée (`public`, `private`, `protected`) et leurs éventuels qualificatifs supplémentaires (`final`, `static`).
- Écrivez les constructeurs de chacune des classes, ainsi que les méthodes `reserver(int)`, `calculerPoids()`, `peutRouler()` et `inspectionOK()`. N'hésitez pas à faire qu'une classe fille redéfinisse une méthode déjà définie dans une classe mère. En revanche, évitez d'écrire deux fois une méthode ayant le même comportement. Si vous ne pouvez pas éviter les redondances, une solution peut être de modifier ou d'enrichir l'organisation, du côté de l'héritage ou des délégations.

3. Que se passe-t-il ?

Considérons le programme ci-dessous. Indiquez quelles instructions sont correctes ou incorrectes. Pour les instructions incorrectes, précisez si l'erreur est détectée à la compilation ou à l'exécution. Pour les instructions correctes, précisez le résultat obtenu. Pour l'exécution, on ignorera l'effet des instructions incorrectes.

```
EltTrain e;
Wagon w;
WagonBar wb;
Loco l;
Autorail a;

e  = new EltTrain(100);
l  = new Loco(1000, 10000);
w  = new Wagon(1000, 20);
wb = new WagonBar(1000);
a  = new Autorail(1000, 5000, 10);
System.out.println(w.capacite);
System.out.println(wb.capacite);
System.out.println(e.nom);
System.out.println(wb.nom);
System.out.println(a.reserver());
System.out.println(w.reserver());
System.out.println(wb.reserver());
System.out.println(l.reserver());
wb = (WagonBar) w;
System.out.println(wb.capacite);
l = (Loco) a;
e = wb;
System.out.println(e.reserver());
System.out.println(e.capacite);
w = (Wagon) e;
System.out.println(w.reserver());
System.out.println((EltTrain) w).capacite);
System.out.println((WagonBar) w).capacite);
```

4. Gestion d'un train complet

Créez une classe `Train` représentant les trains. Un train sera caractérisé par une liste d'éléments `ArrayList<EltTrain>`. Définir des méthodes ayant les effets suivants :

1. Ajouter un élément à un train.
2. Calculer la capacité totale d'un train.
3. Calculer le nombre de places encore disponibles.
4. Calculer le poids total du train.
5. Vérifier la bonne formation du convoi :
   - Aucun élément accueillant des passagers ne doit avoir de surréservation.
   - Le poids total doit être inférieur aux capacités de traction des éléments tracteurs.
   - Les éléments tracteurs sont toujours aux extrémités, et ont le droit de rouler.

5. Petit bonus

On veut ajouter un nouveau type d'élément de train : le porte-conteneur, qui peut transporter une certaine quantité de marchandises. Détaillez les changements à apporter dans votre organisation.