TP 2 : Héritage - Polymorphisme - utilisation de Collections et de Stream

Exercice 1 : Des Portes

On propose de modéliser différents types de portes. Une porte de base peut être soit ouverte, soit fermée. On peut bien évidemment ouvrir et fermer une porte. En plus des portes classiques, on souhaite modéliser des portes à verrou automatique, des portes à code secret et des portes condamnées.

Une porte à verrou automatique fonctionne comme une porte classique, si ce n'est que lorsqu'on la ferme, elle se verrouille. Pour l'ouvrir, il faudra donc d'abord la déverrouiller. Bien entendu, lorsqu'elle est verrouillée, une telle porte ne peut pas s'ouvrir.

Les portes à code secret ne peuvent se déverrouiller qu'avec un code secret (un entier). Lorsque l'on ferme une telle porte, elle se verrouille automatiquement.

Les portes condamnées sont des portes fermées que l'on ne peut plus ouvrir.

1. Implantez les classes `Door`, `AutoLockDoor`, `SecretCodeDoor` et `ClosedDoor`.
2. Dans la classe `Door`, redéfinissez la méthode `toString()` de manière à afficher le nom de la classe suivi de son état (ouverte ou fermée). Pour cela vous utiliserez la méthode `getClass().getSimpleName()` de la classe `Object`.
3. Dans la classe `Door`, ajoutez une méthode static `printDoorList` permettant d'afficher une liste de portes. Pour cela, vous pouvez utiliser une boucle `for each`.
4. Vérifiez les spécifications fonctionnelles suivantes :
   (a) une porte verrouillée ne peut pas s'ouvrir
   (b) après avoir fermé une porte avec verrou, elle est bien verrouillée et ne peut donc pas s'ouvrir
   (c) lorsque l'on tente de déverrouiller une porte à code avec un mauvais code, elle reste verrouillée.
   (d) lorsque l'on tente de déverrouiller une porte à code avec le bon code, elle est bien déverrouillée et peut s'ouvrir.
   (e) lorsque l'on tente de déverrouiller une porte à code avec le bon code, elle est bien déverrouillée et peut s'ouvrir.
   (f) si on tente de déverrouiller une une porte à code sans indiquer de code, elle reste verrouillée.
   (g) une porte condamnée ne peut pas s'ouvrir.

   Depuis Java 8, vous pouvez également simplifier cette écriture en utilisant une expression lambda afin d'instancier l'interface fonctionnelle de type `Consumer<T>` à mettre en paramètre de la fonction `forEach` appliquée à la liste. Autrement dit, si l est la liste à afficher, votre fonction doit être de la forme : `l.forEach(expression lambda)`.
5. Dans une classe `Main`, effectuez quelques tests afin de vérifier que vos différentes classes respectent bien le cahier des charges.
6. Créez une `List<Door>` et ajoutez-y quelques portes (de différents types).
7. Testez votre méthode `printDoorList`.
8. Créez à présent une `List<AutoLockDoor>` et ajoutez-y des instances de `AutoLockDoor` et de `SecretCodeDoor`. Pouvez-vous utiliser `printDoorList` avec cette liste ? Assurez-vous de bien comprendre pourquoi.

9. Modifiez la signature de la fonction `printDoorList` afin de pouvoir l'utiliser avec des listes de n'importe quel type de porte. Testez.
10. À partir de votre liste de `Door`, créez des pipelines sur des `Stream`, afin d'obtenir :
    — le nombre de portes `AutoLockDoor` qui sont ouvertes,
    — la liste des portes à code secret, que vous aurez pris soin de fermer,
    — l'affichage de chaque porte que vous avez pu ouvrir.

Exercice 2 : Des Stylos

Le but de cet exercice est de modéliser différents types de stylos. Tous les stylos contiennent un cartouche, caractérisée par un niveau d'encre et une couleur. Tous les stylos peuvent écrire, et on peut changer leur cartouche (On supposera que la nouvelle cartouche ne provient jamais d'un autre stylo).

On souhaite définir les stylos suivants :

— `Highlighter` : à la différence des stylos classiques, ce nouveau type de stylo dispose en plus d'une méthode `highlight(...)` qui affiche le texte passé en paramètre en MAJUSCULES. Note : pour transformer une chaîne de caractères en majuscules vous pouvez utiliser la méthode toUpperCase() de la classe String.

— jetable : si on essaye de remplacer la cartouche d'un stylo jetable, la cartouche se vide et le stylo est alors inutilisable.

— Retractable : un tel stylo dispose d'une mine rétractable et deux nouvelles méthodes leadIn et leadOut permettant de rentrer ou sortir la mine. Lorsque la mine d'un stylo rétractable est rentrée, celui-ci ne peut évidemment pas écrire.

— SwitchPen : ces stylos ont tous une mine verte qui permettent d'écrire un texte en substituant toutes les occurrences d'un caractère `c1` par un caractère `c2`. Par exemple, si `pen` est un `SwitchPen`, alors `pen.write(«something very smart»,'m','?')` produira l'affichage `so?ething very s?art`. Si on ne précise pas les deux caractères, un `SwitchPen` changera les `'a'` en `'*'`.

— MultiColorPen : les stylos multicolores sont des stylos rétractables dont la particularité de pouvoir écrire de plusieurs couleurs (comme les stylos 4 couleurs usuels). On peut construire des crayons multicolores avec autant de couleurs que l'on souhaite (au moins une). Un stylo multicolore peut avoir au maximum une mine de sortie.

1. Implantez les différentes classes.
2. Effectuez différents tests afin de vous assurer que votre implantation respecte bien le cahier des charges.

Exercice 3 : Une course de véhicules

Les différents éléments de la course

On souhaite simuler des courses de véhicules. Toutes les courses s'effectuent en ligne droite, un circuit est uniquement caractérisé par :

- sa longueur (qui correspond à la position de la ligne d'arrivée, la ligne de départ correspondant à la position 0),
- la liste des véhicules qui participent.

En outre, un circuit contient un seul constructeur permettant de construire un circuit de la longueur voulue. Un circuit peut :

- ajouter un véhicule participant à la course,
- initialiser la course en positionnant tous les véhicules sur la ligne de départ,
- lancer la course (cf. déroulement de la course),
- afficher l'état de la course à un instant T (i.e. affiche chaque véhicule),
- savoir si la course est terminée (i.e. si quelqu'un a franchi la ligne d'arrivée),
- désigner aléatoirement un véhicule (cf. déroulement de la course).

Un véhicule est caractérisé par :

- sa position,
- sa vitesse courante,
- la vitesse max qu'il peut atteindre,
- son état (i.e. démarré ou HS).

À sa création, un véhicule n'est pas HS. Un véhicule peut :

- accélérer et décélérer (ce qui augmente ou diminue sa vitesse de 1),
- se positionner sur la ligne de départ (i.e. en position 0, vitesse 0),
- s'arrêter (i.e. met directement sa vitesse à 0),
- être mis hors service (le véhicule est arrêté, et passe en mode HS),
- afficher son type (cf. section suivante)
- afficher son trajet,
- avancer : s'il n'est pas HS, sa position augmente de la valeur de sa vitesse. Par exemple, si le véhicule est en position 3 et que sa vitesse est de 2, alors la méthode avancer déplacera le véhicule en position 5),
- donner sa position,
- effectuer une action spéciale.

Un skateboard est un véhicule qui possède une vitesse max de 4, son action spéciale est d'accélérer.

Un tank est un véhicule qui possède une vitesse max de 2. À sa création, un tank possède un missile (et un seul), son action spéciale est d'attaquer un autre véhicule. Si le véhicule attaqué est un tank, la cible s'arrête ; sinon, la cible est HS. Dans tous les cas, le tank qui a tiré s'arrête (il pourra repartir à l'étape suivante) et n'a plus de missile.

Une voiture est un véhicule qui possède une vitesse max de 8, lorsqu'elle accélère, sa vitesse augmente de 2. Son action spéciale est de décélérer.

Déroulement de la course

Le déroulement de la course sera affiché en mode texte. À chaque étape, le trajet de chaque véhicule sera représenté de la manière suivante : un véhicule en position p affichera p tirets `'-'` suivis du caractère décrivant son type entre crochets :

— `''[*]''` pour un véhicule HS
— `''[S]''` pour les skateboards non HS,
— `''[T]''` pour les tanks non HS,
— `''[C]''` pour les voitures non HS.

À chaque étape, tous les véhicules affichent leur trajet, accélèrent, avancent. À la fin de chaque étape, le circuit choisit un véhicule au hasard, qui sera le seul à effectuer son action spéciale. Pour cela, un entier correspondant à la position d'un véhicule est choisi aléatoirement (i.e. un entier k compris entre 0 et la taille de la liste). Le véhicule se situant à la position k dans la liste est le véhicule choisi.

La course se termine lorsqu'un véhicule a franchi la ligne d'arrivée.

Note : La méthode `rand.nextInt(N)` (où rand est un objet de la classe `java.util.Random`) permet d'obtenir un entier aléatoire compris entre 0 et N-1.

Exemple d'affichage d'une étape de la course :

```
0:-------[V]
1:---[T]
2:--[T]
3:----[*]
4:-----------[S]
```