Programmation Objet en Java - Feuille de TP 3

Exercice 1. ($\sim$ 1 heure)

On souhaite définir un objet de type `Polygon` pour représenter un polygone comme une suite de points :

```
public record Point(int x, int y) { }
```

On utilisera un champ de type `ArrayList<Point>` pour stocker les points et il n'y a pas de limite sur leur nombre. Attention, un objet `Polygon` *n'est pas juste* une liste de points.

a) Doit-on utiliser un record ou une classe pour définir un `Polygon` ?

b) Doit-on utiliser le mot `final` pour déclarer la liste des points ? Est-ce que cela signifierait que l'on ne peut plus ajouter de points dans la liste ?

c) Écrire un constructeur qui crée un polygone sans point.

d) Écrire une méthode `add` qui permet d'ajouter un `Point` au polygone.

e) Écrire une méthode qui permet d'afficher un `Polygon`.

f) Écrire une classe `PolygonTest` pour tester vos polygones.

g) Ajouter une méthode `numberOfPoints` renvoyant le nombre de points du polygone.

h) Ajouter une méthode `contains` qui prend un `Point` en paramètre et renvoie `true` si c'est un point du polygone et `false` sinon.

i) Écrire une méthode `upperPoint` qui renvoie le point le plus "haut" du polygone (pour cette question, on supposera que le polygone contient au moins un point).

Que pourrait-on ajouter dans `Point` pour simplifier l'écriture de cette méthode?

j) Que va-t-il se passer si on exécute le code suivant?

```
Polygon p = new Polygon();
p.add(null);
System.out.println(p.upperPoint());
```

Corriger le problème.

k) Écrire une méthode `perimeter` qui renvoie le périmètre du polygone. Quelle méthode est-il naturel d'ajouter dans la classe `Point` pour faire cela ?

l) Écrire une méthode `smallestPerimeter` qui prend en paramètre une liste de polygones et renvoie celui dont le périmètre est le plus petit (si la liste est vide, on renvoie `null`). Quelle est la particularité de cette méthode ?

m) Les méthodes `upperPoint` et `perimeter` ne sont pas très efficaces... Essayer d'améliorer leur complexité.

n) On souhaite écrire une méthode `move(int dx, int dy)` permettant de déplacer un polygone dans le plan suivant le vecteur (dx, dy). Peut-on le faire avec les points déclarés comme des records ou faut-il une classe mutable ? La classe `Polygon` doit-elle être mutable ou non ? Est-ce que l'on a vraiment le choix ?

Exercice 2. ($\sim$ 45 minutes)

On va définir une classe `BookStore` pour stocker des livres avec un champ `catalog` qui est une liste d'objets `Book` (définis au TP précédent). Un livre pourra figurer plusieurs fois dans le catalogue. On verra plus tard comment utiliser d'autres classes pour s'assurer qu'un même livre ne figure qu'une seule fois ou pour tenir compte du nombre d'exemplaires.

a) On souhaite ajouter un champs `price` à `Book` pour indiquer son prix, ainsi qu'une méthode `setOnSale` qui permet de réduire ce prix de 20%. Est-ce possible avec la définition de actuelle de `Book`. Faire les changements nécessaires et rajouter cette méthode.

b) Écrire la classe `BookStore` avec un constructeur qui crée un catalogue vide et une méthode `add` qui permet d'ajouter un livre dans le catalogue.

c) Écrire une classe `TestBookStore` pour la tester : créez trois livres, un catalogue et mettez les livres dans le catalogue. Afficher le catalogue. Quelle(s) méthode(s) doit-on ajouter pour que l'affichage fonctionne bien ?

d) Écrire une méthode `booksByAuthor` qui prend un nom d'auteur en paramètre et renvoie une liste des livres écrits par cet auteur. Est-il vraiment nécessaire d'ajouter un accesseur pour l'auteur, dans `Book` ?

e) Écrire une méthode `authorBeginsWith` qui prend un caractère en paramètre et renvoie une liste des livres dont l'auteur commence par cette lettre (pensez à regarder dans la documentation de la classe `String`).

f) Que se passe-t-il avec le code ci-dessous ? Où se situe le problème ?

```
public static void main(String[] args){
    BookStore bookStore = new BookStore();
    bookStore.add(new Book("1984", "Orwell", 10));
    bookStore.add(new Book("Auteur inconnu", "", 1));
    bookStore.add(new Book("Contes chinois", null, 15));
    System.out.println(bookStore.authorBeginsWith('O'));
}
```

Corriger le problème en empêchant la construction d'un Book avec des champs `null`. Quel autre problème du même type pourrait se produire? Corriger le code si nécessaire.

g) Écrire une méthode `contains(Book b)` dans la classe `BookStore` qui renvoie `true` si le livre en argument figure dans le catalogue et `false` sinon. Tester la méthode `contains` dans `Test` avec le code suivant.

```
public static void main(String[] args){
    BookStore bookStore = new BookStore();
    Book b1 = new Book("Les metamorphoses", "Ovide", 4);
    Book b2 = new Book("Cent ans de solitude", "Gabriel Garcia Marquez", 10);
    Book b3 = new Book(new String("Les metamorphoses"), "Ovide", 4);
    bookStore.add(b1);
    bookStore.add(b2);
    System.out.println(bookStore.contains(b3));
}
```

Est-ce que c'est le résultat que vous attendiez ? Pourquoi ?

h) Que faudrait-il faire pour que la méthode `contains` fonctionne correctement ? Commenter la méthode `contains` pour l'instant.

Exercice 3. Fabriquer des chaînes de caractères efficacement

a) Modifier la méthode d'affichage d'un catalogue pour que chaque livre soit affiché sur une ligne séparée et qu'elle indique le nombre de livres dans le catalogue.

b) En Java, les chaînes de caractères ne sont pas mutables. Donc, si l'on souhaite modifier une chaîne (rajouter des caractères, par exemple), il faut la reconstruire entièrement. C'est ce que fait l'opérateur `+`. Malheureusement, cet opérateur est très coûteux, notamment lorsqu'il est utilisé plusieurs fois sur la même chaîne. On préférera donc utiliser un objet de la classe `StringBuilder`.

Lire la documentation de cette classe et modifier le code pour utiliser un `StringBuilder` et ainsi éviter l'utilisation des `+` dans la boucle.

Exercice 4. (Optionnel)

Le but de cet exercice est de tester les énigmes de sabliers du type : "On dispose de deux sabliers, l'un de x minutes et l'autre de y minutes ; comment mesurer exactement z minutes".

a) Un sablier est caractérisé par le temps total qu'il peut mesurer mais, pour cet exercice, il sera plus simple de le décrire comme un couple temps écoulé (`elapsedTime`) et temps restant (`timeLeft`), qui permet de décrire l'état du sablier (et pas juste son temps total).

Écrire la classe `Hourglass` qui représente un sablier, avec un constructeur qui prend uniquement en paramètre le temps total qu'il peut mesurer. On remarquera qu'initialement, un sablier n'a plus de temps restant.

b) Écrire une méthode d'affichage pour le sablier. Elle devra indiquer le temps écoulé par rapport au temps total.

c) Écrire une classe `Enigma` qui contient deux sabliers avec le constructeur correspondant, ainsi qu'une méthode d'affichage permettant de connaître l'état des deux sabliers.

d) Écrire trois méthodes `turnH1`, `turnH2` et `turnBoth` qui permettent respectivement de retourner le premier, le deuxième ou les deux sabliers, **puis d'attendre jusqu'à ce que l'un des deux sabliers (au moins) soit vide**. Essayer de ne pas trop dupliquer de code.

Contrainte : vous n'avez besoin d'aucun *accesseur* pour cet exercice.

Indication : séparer le retournement des sabliers et l'écoulement du temps.

e) On souhaite pouvoir connaître le temps total écoulé depuis que l'on a commencé à retourner les sabliers. Ajouter le nécessaire à la classe `Enigma`.

f) Vous pouvez, par exemple, tester votre code sur l'exemple suivant.

Attention, il s'agit de la solution de l'énigme "On dispose de deux sablier, l'un de 7 minutes et l'autre de 4 minutes; comment mesurer exactement 9 minutes, à partir du moment où l'on retourne les sabliers". Si vous voulez chercher la solution par vous même, attendez pour lire ce qui suit.

```
Enigma enigma = new Enigma(new Hourglass(7), new Hourglass(4));
System.out.println(enigma);
// H1 (Temps ecoule : 7/7), H2 (Temps ecoule : 4/4), total : 0 minutes

enigma.turnBoth();
System.out.println(enigma);
// H1 (Temps ecoule : 4/7), H2 (Temps ecoule : 4/4), total : 4 minutes

enigma.turnH2();
System.out.println(enigma);
// H1 (Temps ecoule : 7/7), H2 (Temps ecoule : 3/4), total : 7 minutes

enigma.turnH1();
System.out.println(enigma);
// H1 (Temps ecoule : 1/7), H2 (Temps ecoule : 4/4), total : 8 minutes

enigma.turnBoth();
System.out.println(enigma);
// H1 (Temps ecoule : 7/7), H2 (Temps ecoule : 1/4), total : 9 minutes
```