Programmation Objet en Java - Feuille de TD 3

Le but de ce TD est d'écrire des classes pour représenter un magasin de vêtements de sport.

Exercice 1.

On souhaite définir un type `Clothing` qui modélise des vêtements de sport. Chaque vêtement a une `category` de type `String`. Un vêtement a aussi une marque (`brand`) de type `String` et deux champs `size` (un `int`) et `price` (un `int`) pour la taille et le prix. Les tailles devront être comprises entre 1 et 5 et le prix doit être positif ou nul. Les champs ne seront pas modifiés.

Écrire un record `Clothing` avec le constructeur compact qui teste les pré-conditions. Écrire une deuxième constructeur qui prend en argument la catégorie, la marque et le prix et crée le vêtement en taille 1. Pourquoi prend-on une record plutôt qu'une classe ?

Le code suivant devra fonctionner :

```
public static void main(String[] args) {
  var polo = new Clothing("polo", "Colmar", 3, 40);
  System.out.println(polo);
  var polo2 = new Clothing("polo", "Colmar", 40);
  System.out.println(polo2);
}
```

et donner

```
Clothing[category=polo, brand=Colmar, size=3, price=40]
Clothing[category=polo, brand=Colmar, size=1, price=40]
```

Exercice 2. Écrire une classe `SportsShop` qui modélise un magasin de sport. Un magasin aura un champ `name` de type `String` et il contiendra une liste d'articles qui seront des vêtements de sport. Les articles pourront figurer plusieurs fois dans la liste. Écrire une méthode `add` pour ajouter un vêtement dans le magasin. On pourra ajouter plusieurs fois un même vêtement dans la liste. Pourquoi prend-on une classe plutôt qu'un record pour `SportsShop` ?

Exercice 3. Écrire une méthode `toString` dans `SportsShop` qui permet d'afficher le magasin. On affichera le nom du magasin sur une ligne puis chaque article sur une ligne. Il ne devra pas y avoir de passage à la ligne à la fin. On devra utiliser un `StringBuilder`. Le code suivant devra donc fonctionner :

```
public static void main(String[] args) {
  var polo = new Clothing("polo", "Colmar", 2, 40);
  var shirt1 = new Clothing("tshirt", "Burton", 4, 50);
  var shirt2 = new Clothing("tshirt", "Burton", 4, 50);
  var shop1 = new SportsShop("Italie2");
  shop1.add(polo);
  shop1.add(shirt1);
  shop1.add(shirt2);
  System.out.println(shop1);
}
```

et donner

```
Italie2
Clothing[category=polo, brand=Colmar, size=2, price=40]
Clothing[category=tshirt, brand=Burton, size=4, price=50]
Clothing[category=tshirt, brand=Burton, size=4, price=50]
```

Exercice 4. Écrire une méthode `public int totalPrice()` qui calcule le prix total de tous les vêtements du magasin. La méthode devra renvoyer 0 si le magasin est vide.

```
System.out.println(shop1.price());
// 140
```

Exercice 5. Un magasin souhaite proposer des vêtements en solde. Pour cela, on écrira une méthode `onSale` qui renvoie une liste non modifiable des articles du magasin qui vont être soldés. Les articles soldés seront les vêtements dont la taille est supérieure ou égale à 3.

```
System.out.println(shop1.onSale());
// [[Clothing[brand=Burton, size=4, price=50],
//  Clothing[brand=Burton, size=4, price=50]]
```

Exercice 6. Écrire une méthode `isIncluded(SportsShop shop1, SportsShop shop2)` qui teste si tous les articles du magasin `shop1` sont aussi des articles du magasin `shop2`. Quelle est la particularité de cette méthode ?

Exercice 7.

Écrire une méthode `sameItems(SportsShop shop1, SportsShop shop2)` qui teste si les magasins `shop1` et `shop2` contiennent les mêmes articles, sans tenir compte des éventuelles répétitions.

```
var shop2 = new SportsShop("Jaude");
shop2.add(shirt2);
shop2.add(polo);
System.out.println(SportsShop.sameItems(shop1, shop2));
// true
```