Programmation Objet en Java - Feuille de TD 5

Exercice 1. Voici le code d'une classe qui représente un boîte permettant de stocker jusqu'à deux mots. En plus de l'affichage, elle dispose de cinq méthodes publiques :

- `isEmpty` et `isFull` qui permettent de tester si la boîte est vide ou pleine.
- `add` qui permet d'ajouter un mot (les caractères doivent être des lettres) dans la boîte s'il reste de la place.
- `getAWord` qui permet de voir (sans le retirer) un mot dans la boîte (peu importe lequel) si elle en contient au moins un.
- `remove` qui prend en paramètre un nombre de mots à retirer de la boîte et en retire autant si c'est possible.

Ajouter ce qu'il faut dans le code de cette classe pour signaler et empêcher toute mauvaise utilisation de la boîte.

```
public class WordBox {
  private String word1; // exceptionnellement le champ n'est pas final et peut être null
  private String word2; // exceptionnellement le champ n'est pas final et peut être null

  public boolean isEmpty() {
    return word1 == null && word2 == null;
  }

  public boolean isFull() {
    return word1 != null && word2 != null;
  }

  private static boolean isAlpha(String word) { ... }

  public void add(String word) {
    if (word1 == null) {
      word1 = word;
      return;
    }
    if (word2 == null) {
      word2 = word;
      return;
    }
  }

  public void remove(int quantity) {
    if (word1 != null) {
      word1 = null;
      if (quantity == 1) {
        return;
      }
    }
    word2 = null;
  }

  public String getAWord() {
    if (word1 != null) {
      return word1;
    }
    if (word2 != null) {
      return word2;
    }
  }

  @Override
  public String toString() {
        ...
  }
}
```

Exercice 2.

Le package `fr.but.shopping` permet de modéliser des livres numériques (avec un titre, un auteur et un prix) ainsi qu'un panier électronique dans lequel on peut ajouter ou retirer des livres. Il est possible d'afficher le panier et de calculer son prix.

La boutique qui vend les livres se diversifie et vend désormais d'autres types de contenu numérique : des jeux-vidéo et des cartes cadeau dont on vous fournit le code. Les jeux vidéo ont un titre, un type de console et un prix. Les cartes cadeau ont une valeur (entière) et une durée de validité en semaines. Leur prix est calculé en fonction de ces 2 paramètres.

Le but de l'exercice est de transformer le code fourni pour que l'on puisse également ajouter ces nouveaux types de contenu au panier. Et en réalité, la boutique prévoit déjà de vendre de la musique et des films à l'avenir (dans un format qui reste à définir), on aimerait donc que l'ajout de nouveaux média se fasse aisément par la suite.

a) On voudrait pouvoir ajouter/retirer aussi bien des livres que des jeux vidéos et des cartes cadeau au panier. Comment faire cela ?

b) Modifier le code en conséquence, de façon à pouvoir tous les ajouter au panier.

c) Ajouter également de quoi faire en sorte que le code fourni ne permette pas de créer des objets qui risquent de provoquer des exceptions involontaires s'ils sont utilisés correctement. Ajouter aussi tous les `requireNonNull` sur les arguments objets des méthodes publiques.

```java
public record Book(String author, String title, int price) {
  private static String firstUpperCase(String s) {
    return Character.toUpperCase(s.charAt(0)) + s.substring(1, s.length());
  }

  @Override
  public String toString() {
    return firstUpperCase(title) + ", de " + author;
  }
}
```

```java
public record PrePaid(int value, int validTime) {
  @Override
  public String toString() {
    return "Carte cadeau (" + value + "), durée: " + validTime + " semaines";
  }
}
```

```java
public enum Console {
  PS4, PS5, XBOX, WII, DS, SWITCH
}
```

```java
public record VideoGame(String title, Console console, int price) {

  @Override
  public String toString() {
    return title + ", sur " + console.name();
  }
}
```

```java
public class ShoppingCart {
  private final ArrayList<Book> cart = new ArrayList<>();

  public void add(Book book) {
    cart.add(book);
  }

  public void remove(Book book) {
    cart.remove(book);
  }

  public int price() {
    var sum = 0;
```

```
  for (var book : cart) {
    sum += book.price();
  }
  return sum;
}
```

```
  @Override
  public String toString() {
    var sb = new StringBuilder();
    sb.append("--- Shopping cart ---\n");
    for (var book : cart) {
      sb.append(book).append("\n");
    }
    return sb.toString();
  }
}
```