Un peu de design et des entrées-sorties

Exercice 1 - Panier électronique

Voici le code du package `fr.umlv.shopping` permettant 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. Les tests sont dans le package `fr.umlv.shopping.test`.
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.

1. Modifier le code fourni pour que l'on puisse ajouter/retirer aussi bien des livres que des jeux vidéos et des cartes cadeau au panier.
2. Vérifier que la suppression fonctionne bien dans tous les cas.
3. Faire en sorte de minimiser la duplication de code.

Exercice 2 - Entrées-sorties

On veut maintenant pouvoir sauvegarder les articles d'un panier d'achats dans un fichier sous forme textuelle en respectant un format particulier, afin de pouvoir les recharger en mémoire ultérieurement.

On retient le format de sauvegarde suivant (qui est volontairement un peu naïf). Chacun des articles est représenté sur une seule ligne de texte. Cette ligne débute par une chaîne de caractères précisant de quel type de contenu numérique il s'agit (`"B"` pour un `Book`, `"G"` pour un `VideoGame` ou `"P"` pour une carte `PrePaid`). Ensuite, séparés par un délimiteur (une chaîne de caractères, disons `"#"`), les différentes caractéristiques de chacun de ces articles.

1. En fonction de l'architecture de classes, classes abstraites et/ou interfaces que vous avez mis en place pour l'exercice précédent, écrire les méthodes `toTextFormat()` nécessaires pour que le code suivant produise l'affichage indiqué en commentaire. Attention, vous devez encore essayer d'avoir du code générique en minimisant la duplication de code. En particulier, on souhaiterait que tout type de média dispose de cette méthode `toTextFormat()` qui retourne une `String`.

```
public class SaverLoader {
  // ...
  public static void main(String[] args) {
    var sdb = new Book("S. de Beauvoir", "Mémoires d'une jeune fille rangée", 990);
    System.out.println(sdb.toTextFormat());
    // B#990#Mémoires d'une jeune fille rangée#S. de Beauvoir
    var zelda = new VideoGame("The legend of Zelda", VideoGame.Console.WII, 4950);
    System.out.println(zelda.toTextFormat());
    // G#4950#The legend of Zelda#WII
    var pp100 = new PrePaid(10000, 10);
    System.out.println(pp100.toTextFormat());
    // P#10000#10
  }
}
```

Des constantes pour le délimiteur (dièse) et pour les types d'articles (B, G, P) peuvent être définies dans une classe `SaverLoader`:

```
public class SaverLoader {
    static final String SEPARATOR = "#";
    static final String BOOK_TYPE = "B";
    static final String VIDEO_GAME_TYPE = "G";
```

```
    static final String PREPAID_TYPE = "P";
    ...
```

2. Écrivez maintenant dans la classe `SaverLoader` une méthode `saveInTextFormat` qui prend en argument une liste d'articles numériques et un flot d'écriture de caractères de type `BufferedWriter`; cette méthode écrit dans le flot les chaînes de caractères produites par les méthodes `toTextFormat` des différents articles de la liste, en les séparant par des retours à la ligne. Ainsi, les articles créés à la question précédente doivent pouvoir sauvegardés dans un fichier par le code suivant (pensez à créer le chemin du fichier de sauvegarde):

```
var list = List.of(sdb, zelda, pp100);

Path saveFilePath = ... ; // nom du fichier de sauvegarde: "saveFile.txt"
try(var writer = Files.newBufferedWriter(saveFilePath,
                                         StandardCharsets.UTF-8,
                                         StandardOpenOption.CREATE)) {
  SaverLoader.saveInTextFormat(list, writer);
}
```

   Vérifier ce qui a été écrit dans le fichier!

3. Pour relire ce fichier, ou un autre formaté de la même manière, il nous faut maintenant une méthode `loadFromTextFormat` qui prend en paramètre un flot de lecture de caractères et retourne une liste d'articles numériques construits à partir des indications lues dans le flot. Écrire cette méthode dans la classe `SaverLoader`.
   Vous pourrez utiliser la méthode `br.readLine()` pour lire le flot ligne par ligne, et la méthode `split()` de la classe `String` pour découper une ligne suivant un séparateur donné passé en argument.
   Testez-la sur le fichier créé précédemment, en utilisant le `try-with-ressources`.
4. Le fichier ci-dessous contient des articles qui ont été sauvegardés en utilisant l'encodage ISO-8859-15, extension de l'encodage Latin-1 qui inclut par exemple le symbole €. Enregistrez ce fichier (en faisant clic-droit enregistrer-sous...) et réutilisez le code que vous avez écrit à l'exercice précédent pour charger les articles de ce fichier.

Exercice 3 - Réductions

Il arrive que certains clients commandent des articles en plusieurs exemplaires (au moins deux). Dans ce cas, on applique une réductions des 5% à tous les articles concernés.

1. Comment peut-on faire pour gérer les multiplicités des articles de façon efficace?
2. Modifier le code pour que le calcul du prix du panier prenne en compte cette réduction.
3. Pour aller plus loin: utiliser un `Stream` pour le calcul du prix d'un panier. Vous verrez l'année prochaine comment optimiser également l'affichage.
