Programmation Par Objets et Java - TD 2

Création de classes

Exercice 1 : Création de classes et objets

Créez une classe complète `Livre` avec les attributs `titre`, `auteur` et `annee` (constructeur, getters, setters). Lancement d'exception si les chaînes passées aux setters sont vides.

Utiliser le code suivant pour tester votre classe, avec et sans `toString()` dans la classe `Livre`.

```
public class Main {
    public static void main(String[] args) {
        Livre livre1 = new Livre("1984", "George Orwell", 1949);
        System.out.println(livre1);
    }
}
```

Exercice 2 : Héritage

Créez une classe `Roman` qui hérite de `Livre` et ajoutez un attribut `genre`. Surchargez `toString()` pour inclure le genre.

Exercice 3 : Implémentation d'une interface

Créez une interface `Empruntable` avec une méthode `emprunter()`. Faites implémenter cette interface par `Livre` et `Roman`.

Exercice 4 : Comparaison d'objets (`equals`)

Tester le code suivant :

```
Livre l1 = new Livre("Dune", "Frank Herbert", 1965);
Livre l2 = new Livre("Dune", "Frank Herbert", 1965);
System.out.println(l1.equals(l2)); // Doit retourner false
```

Redéfinissez `equals()` dans `Livre` pour comparer deux livres selon leur titre et auteur. Relancez le test. Attention : `equals` est une méthode de la classe `Object`.

Exercice 5 : Recherche dans une `ArrayList`

Créez une `ArrayList<Livre>` et ajoutez-y plusieurs livres. Vérifiez qu'une recherche avec `indexOf()` fonctionne bien après la surcharge de `equals()`.

Test :

```
ArrayList<Livre> bibliotheque = new ArrayList<>();
bibliotheque.add(new Livre("Le Petit Prince", "Antoine de Saint-Exupéry",
1943));
System.out.println(bibliotheque.indexOf(new Livre("Le Petit Prince",
"Antoine de Saint-Exupéry", 1943))); // Doit retourner un index valide
```

Exercice 6 : Recherche dans une `Map`

Nous allons gérer le nombre d'exemplaires de livres dans une bibliothèque. Pour cela, on utilise la classe `HashMap<Livre, Integer>` avec ses méthodes `put` et `get`. Voir https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

Testez ce qui suit :

```
HashMap<Livre, Integer> exemplaires = new HashMap<>();
exemplaires.put(new Livre("Le Petit Prince", "Antoine de Saint-Exupéry",
1952), 5);
exemplaires.put(new Livre("Le Petit Prince", "Antoine de Saint-Exupéry",
1952), 8);

for (Livre l : exemplaires.keySet()) {
   System.out.println(l + " : " + exemplaires.get(l) + " exemplaires");
}
```

Vous obtenez 2 instances dans le tableau associatif, une avec 5 exemplaires, une avec 8. Cela montre que les clés sont "mal" gérées (gérées par adresse !).

Puis surchargez `public int hashCode()` dans `Livre` et retestez.

Exercice 7 : tri

Essayez de trier une liste de livres avec le code suivant :

```
Collections.sort(bibliotheque);
```

Vous obtenez un message d'erreur. Comprendre ce qu'il se passe et corriger en conséquence. Voir https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#sort-java.util.List-

Exercice 8 : Refactoring

Si vous avez produit un code de qualité, vous avez plusieurs classes `Ex1.java`, `Ex2.java`, etc. et des classes `Livre.java`, `Roman.java`, `Empruntable.java`, etc. Réorganisez votres code : laisser les classes exercices dans td2, mais mettre les classes métiers dans *td2/modele*. Cela va nécessiter des imports et la mention `package` dans les classes métier.

Exercice 9 : Modélisation d'un portefeuille de devises

Vous devez modéliser un portefeuille de devises. Ce portefeuille peut contenir un certain nombre de devises, stockées dans une liste ; à chaque devise correspond un montant et le nom de la monnaie (Euros, Roubles, etc.). On peut ajouter des devises à ce portefeuille ou en sortir, afficher le contenu du portefeuille et le trier. Il n'y a qu'une seule entrée de chaque devise dans le portefeuille. On ne peut sortir des Euros que s'il y a déjà des Euros dans le portefeuille, et seulement s'il y en a suffisamment.

À faire :

- Déterminer les attributs de la classe `Devise` ;
- Déterminer les attributs et méthodes à écrire pour la classe `Portefeuille` ;
- Écrire cette classe ;
- En fonction des besoins apparus dans l'écriture de la classe `Portefeuille`, écrire toutes les méthodes de la classe `Devise`.

Exercice 10 : Gestion d'une clientèle

Le but de l'exercice est de gérer une clientèle composée à la fois de clients et de clients privilégiés. Un client est identifié par son nom et son prénom ; on connaît également le chiffre d'affaires réalisé avec notre société. Lorsque ce chiffre d'affaires devient supérieur à 1000 Euros, le client devient alors un client privilégié, qui a droit à un pourcentage de réduction sur chaque achat.

Il y a en fait 3 types de clients privilégiés :

- **Bon Client** : chiffre d'affaires > 1000 Euros, réduction de 5% ;
- **Client Exceptionnel** : chiffre d'affaires > 3000 Euros, réduction de 15% ;
- **V.I.C.** : chiffre d'affaires > 10000 Euros, réduction de 30%.

Le but est bien évidemment d'utiliser l'héritage ! Pour simplifier, il est uniquement possible d'ajouter du chiffre d'affaires aux clients.

À faire :

- Créer la classe `Client` (constructeurs, méthodes d'accès, etc. ;
- Créer la classe `ClientPrivilégié` ;
- Créer la classe `Clientele` (constructeur, méthodes `add(Client c)`, `addCA(int num, float chiffre)`, `affiche()`) ;

- Faire une classe servant de programme principal pour utiliser ces classes : menu permettant de créer, modifier, supprimer un client, etc. Après chaque opération, la clientèle doit être triée de façon décroissante selon le CA du client ;
- Si ce n'est pas déjà fait, réfléchir à un mécanisme permettant de faire muter un client en client privilégié lorsque cela est nécessaire.