TP – Premières classes et objets
Algorithmique – Programmation FIP (ING39)

V. Aponte, P. Courtieu

Septembre 2022

Exercice 1 : réecriture d'une application procédurale en style orienté objet

Le but de cet exercice est d'apréhender les différences entre les styles de programmation procédurale et orienté objet. Ce sera l'occasion de découvrir un code plus facile à structurer et à garder concis. Nous vous proposons une ébauche de solution, à vous de la compléter. Dans cet exercice vous aurez besoin du projet `TpCaisseEnregistreuse`.

Question 1 : réecriture version 1

Le code procédural se trouve dans le paquetage `sourcesSujet`. Il s'agit d'une application qui simule le comportement d'une caisse enregistreuse qui « lit » les noms et prix d'articles qui défilent sur son tapis. La caisse enregistre (ajoute) ces articles dans un ticket. Lorsqu'il n'y a plus d'article à ajouter au ticket, le ticket de caisse et le total final sont affichés.

Le code procédural est volontairement un peu naïf : il n'utilise que des types de base et des tableaux. Les prix sont dans un tableau de prix, et les noms des d'articles sont dans un tableau de **`String`**. Les deux tableaux sont des « miroirs » l'un de l'autre : les cases de même numéro concernent le même article.

Prenez le temps de comprendre comment marche cette version, et pourquoi elle est un peu « low-tech ».

La réécriture en code objet consiste à n'avoir qu'un seul tableau contenant des « paires » (nom,prix), c'est-à-dire des objets contenant à la fois le prix et le nom d'un article.

Nous aurons donc :

— la classe `Article` qui contiendra un prix et un nom et dans laquelle on aura les méthodes d'accès à ces deux valeurs (+ constructeur + une méthode statique de lecture au clavier, voir plus bas) ;

— la classe `Ticket` qui contiendra une liste d'`Article` et dans laquelle on mettra les méthodes de manipulation de cette liste (ajout, retrait,etc, voir plus bas).

— un classe `CaisseEnregistreuse` avec une méthode `main` pour démarrer le programme.

Un exemple d'utilisation des deux classes `Article` et `Ticket` est donné ci-dessous. Vous retrouverez cet exemple dans la classes `TestClasses.java`.

```
Article clou = new Article("clou",0.3);
Article marteau = new Article("marteau",17);
Article a = Article.lireArticle(); /* méthode statique*/
System.out.println(a.toString());
System.out.println(clou.toString());
// on crée maintenant un Ticket et on ajoute les 2 articles dedans.
Ticket t1 = new Ticket();
t1.ajouter(clou);
t1.ajouter(marteau);
System.out.println("Ticket␣t1:␣");
t1.affiche();
```

Vous trouverez une ébauche de solution dans le paquetage `question1`. On détaille les deux classes et la classe principale ci-dessous :

La classe `Article`

Elle modélise les données internes (données par des variables d'instance) d'un article, à savoir son nom et son prix, et définit des opérations pour agir sur ces données.

- les variables d'instance sont **privées** et les méthodes sont **publiques** : pouvez vous expliquer quelle est leur visibilité ?
- il y a un **constructeur** permettant de créer un nouvel objet avec des valeurs initiales reçues en paramètre. Ex : `new Article("clé douille 34mm", 12.0)`. Pouvez vous expliquer les particularités d'un constructeur ? S'agit-il d'une méthode comme les autres ? Combien de sortes de constructeur ?
- vous devez définir des méthodes pour obtenir les valeurs des variables d'instance *getters*, et pour les modifier *setters*. Pouvez vous expliquer pourquoi avons nous besoin de définir des getters ? Ne peut-on pas tout simplement accéder par exemple au prix de l'article clou par `clou.prix` ?
- Notez qu'il y a également une méthode **statique**, autrement dit, qui n'a pas accès aux variables d'instance de l'objet courant. En théorie, elle pourrait être placée ailleurs que dans cette classe, par exemple, dans une classe avec un main. Pouvez vous expliquer l'intérêt de la placer ici ?
- Enfin, la méthode `toString()` est une méthode prédéfinie par défaut sur tout objet. Ici, nous donnons une rédéfinition.
- Expliquez le fonctionnement de la méthode `affiche()`.

Avant de passer à la suite de l'exercice, testez votre classe. Utilisez pour cela la classe `TesteClasses`, qui contient déjà plusieurs créations d'objets et appels de méthodes.

1. Expliquez les affichages de son exécution.
2. Commentez la méthode `toString()`, puis invoquez-la. Observez les changements de comportement.
3. Complétez avec des invocations sur vos méthodes afin de tester leur comportement.

La classe `Ticket`

Elle sert à définir l'objet contenant une seule donnée : la liste de tous les articles lors du passage d'un seul client, ainsi que les opérations qu'on peut effectuer sur cette liste. Pour simplfier ces traiements nous utiliserons la classe prédéfinie `ArrayList` de la bibliothèque `Collection` que nous aborderons la semaine prochaine. Allez voir la documentation de cette classe sur le site d'oracle. Vous aurez au minimum besoin de comprendre les méthodes `size()`, `get(i)`, `add(article)`, `remove(i)`.

- la seule variable d'instance est une liste d'articles (`ArrayList<Article>`) toujours vide au départ.
- pas de constructeur explicite ici. Expliquez comment se passe l'initialisation d'un nouvel objet.
- vous devez définir les opérations sur la liste. Au minimum : ajout d'un nouvel article, suppression d'un article, calcul du total à payer, affichage de tous les articles.
- La méthode `toString()` est redéfinie convenablement pour toutes les classes de la bibliothèque `Collection`. La méthode `toString()` d'un ticket doit simplement afficher les articles de la liste sur des lignes différentes. On peut donc se contenter de faire appel à `toString()` sur la liste. Ce qui est d'ailleurs proposé en commentaire. Nous donnons néanmoins une implantation alternative, qui affiche en prime les numéros de lignes du ticket (pratique pour décider quel numéro d'article enlever). Elle vous permet surtout de comprendre comment l'implantation de la bibliothèque marche. Prenez donc le temps de comprendre l'implantation fournie. Vous aurez souvent à écrire ce genre de boucle dans une semaine !
- Ajoutez dans le main de `TesteClasses` la déclaration de deux tickets avec appels à toutes vos méthodes.

La classe `CaisseEnregistreuse`

C'est la classe qui orchestre l'exécution. Notez que la méthode main y est très, très courte...

Question 2

On souhaite apporter une extension au programme de la question 1. On voudrait considérer l'achat groupé en plusieurs exemplaires d'un même produit. Ainsi, sur une ligne du ticket on aura, non seulement le nom et le prix d'un

produit, mais aussi le nombre d'exemplaires achetés.

- Une solution consisterait à ajouter une quantité (`int`) dans la classe `Article`. Cette classe devrait du coup s'appeler `LigneTicket`.
- Une autre solution consisterait à laisser la classe `Article` représenter un seul article, et avoir une classe `LigneTicket` qui contiendrait une paire (article,quantité).

Dans les deux cas, la classe `Ticket` contiendrait une liste de `LigneTicket`. Implantez la solution que vous préférez.

Exercice 2 : Les classes Heure et Vol

Cet exercice est à faire à la maison si vous n'avez pas eu le temps en séance.

Question 1

Écrivez une classe `Heure` permettant de représenter un horaire défini par deux entiers : heures et minutes.

- Ajoutez un constructeur pour initialiser les attributs d'une heure et les méthodes accesseur nécessaires.
- Ajoutez une méthode `toString()` et une méthode `afficher()`.
- Ajoutez une méthode `boolean estAvant(Heure h)` qui teste si l'heure courante est antérieure à h (cela permettra d'écrire quelque-chose comme : `h1.estAvant(h)`), et une méthode `int dureeIntervalle(Heure h)` qui renvoie la durée en minutes de l'intervalle entre l'heure courante et h.
- Écrivez une méthode main qui déclare et initialise un tableau avec 4 objets `Heure`. Votre programme devra déterminer si le tableau est trié, et si c'est le cas, calculer la somme des intervalles (en minutes) entre deux heures consécutives. Exemple : si le tableau contient les heures 12h30, 14h, 15h, 15h15, alors les intervalles entre ces heures sont : 1h30 + 1h + 0h15, ce qui correspond à 165 minutes.
- Modifiez votre constructeur de la classe `Heure` pour qu'on ne puisse jamais créer un objet avec un horaire impossible (heure négative ou au-delà de 23h59).

Question 2

Ecrivez une classe `Vol` qui est caractérisée par les nom des deux villes de départ et d'arrivée, et par les horaires de départ et d'arrivée du vol. Vous utiliserez la classe `Heure` pour représenter les heures de départ et d'arrivée.

- Ajoutez un constructeur pour initialiser les attributs d'un vol et les méthodes accesseur nécessaires, ainsi qu'une méthode `toString()`.
- Ajoutez une méthode `departAvant(Vol v)` qui teste si le vol courant a un horaire de départ avant le vol v, et une méthode `duree()` qui donne la durée en minutes d'un vol.
- Ajoutez dans votre méthode `main` un tableau de 4 vols et déterminez si ce tableau est trié par horaire de départ. Calculez ensuite la durée cumulée de tous les vols dans ce tableau. Affichez ensuite tous les vols se trouvant dans le tableau.