R6.A.06

TP - Semaine 4

L'exemple pour ce TP est un programme pour calculer et imprimer un relevé des frais d'un client dans un club vidéo. le programme est informé des films qu'un client a loués et pour combien de temps.
Il calcule ensuite les frais, qui dépendent de la durée de location du film, et identifie le type de film.
Il y a 3 sortes de film : réguliers, jeunesse et nouveautés.
En plus de calculer les frais, le relevé calcule également les points de fréquence de location (fidélité) qui varient selon que le film est une nouvelle sortie.
Plusieurs classes représente les éléments video :

- la classe `Movie` est juste une classe de données
- la classe `Rental` (location) représente un client louant un film.
- La classe `Customer` représente un client du magasin. Comme les autres classes elle a des données et des accesseurs.
  Le client a aussi la méthode qui produit le relevé.

Le code de la version originale de cette petite application est sur la zone moodle.

Répondre aux question suivantes en donnant des explications détaillées

1. Quelle est votre impression sur la conception de ce programme ?
2. Si les usagers souhaitent avoir le relevé en HTML, est-ce simple à ajouter ?
3. Si les usagers souhaitent changer la classification des films, est-ce facile à faire ?
4. Quels sont les principaux défauts de ce programme ?

Le travail demandé est détaillé dans les 13 points qui suivent.

1) Préliminaires : modernisation (version plus récente de Java)
Modifier légèrement le programme pour qu'il utilise un `ArrayList` au lieu d'un vecteur et remplacer le `while` par un `for` (each) au lieu d'utiliser un itérateur avec `Enumeration` dans la méthode `statement` de `Customer.`

2) Première étape de refactoring :
Construire un ensemble solide de tests pour cette portion de code.
Les tests sont essentiels car même si on suit une démarche structurée pour le refactoring on n'est pas à l'abri de faire des erreurs. Le relevé (statement) produit une String, donc créer quelques clients, donner à chaque client des locations de films divers et générer les String du relevé.
Comparer la String produite avec celles que vous pouvez vérifier à la main.
Pour cela faire une classe de tests automatiques.

3) Decomposition et redistribution de la méthode Statement
La première phase est de découper cette longue méthode et de mettre les morceaux dans de meilleures classes.
Première étape : trouver un groupe logique de code et utiliser ***Extract Method*** .
Le morceau évident est l'instruction swich.
1) Regarder dans le fragment les variables locales dans la méthode et les paramètres. Ce fragment

utilise *each* et *thisAmount* : *thisAmount* est modifié par la méthode.
Une variable non modifiée peut être passée en paramètre ; pour les modifiées : s'il n'y en qu'une on peut la retourner. La variable temporaire est initialisée à 0 à chaque tour de la boucle et n'est pas modifiée avant le switch dont on peut juste l'affecter au résultat

- extraire le fragment et le mettre dans une méthode de signature :
  - private int amountFor(Rental each) :
- compiler et tester

4) Renommer des variables
Dans la nouvelle méthode renommer les paramètres et variables :
each> aRental ; thisAmount> result
compiler tester à nouveau

5) La nouvelle méthode `amountFor` utilise des informations du rental passé en paramètre mais pas d'info de Customer, il faut déplacer son code vers la classe Rental avec ***Move Method***. On va la renommer `getCharge()` et supprimer le paramètre. Dans Customer le corps de `amountFor()` délèguera le calcul à la nouvelle méthode. Il faut adapter les changements induits.

6) Il faut rechercher tous les endroits où on utilisait l'ancienne méthode. Par exemple :

- `thisAmounts = amountFor (each) -` devient `thisAmount=each.getCharge()`

7) En fait on peut supprimer la variable `thisAmount` dans statement : on applique ***Replace Temp with Query*** . Compiler et tester.

8) Appliquer ***ExtractMethod*** pour extraire de statement le calcul des *frequent renter points*
Déplacer le calcul dans une méthode `int getFrequentRenterPoints()` dans la classe `Rental` et appeler cette méthode dans `statement`.

9) Dessiner le diagramme de classes pour cette nouvelle version du programme et faire un diagramme de séquence de la nouvelle méthode `statement`.

10) Enlever les variables temporaires. On a deux variables dans statement qui sont utilisées pour obtenir un total pour la location. On va utiliser ***ReplaceTemp with Query*** pour remplacer `totalAmount` et `frequentRentalPoints` par des méthodes « query » , qui vont faire les calculs en reprenant la boucle

1. remplacer `totalAmount` avec getTotalCharge().

```
private double getTotalCharge() {
   double results = 0 ;
   for (Rental each : rentals) {
        results += each.getCharge() ;
   }
   return results ;
}
```

2. Faire la même chose pour remplacer `frequentRenterPoints` avec
   - int `getTotalFrequentRenterPoints`()

11) Replacing the conditional logic on Price Code with Polymorphism .

- On va s'occuper du switch dans `getCharge()` de la classe `Rental`. C'est une mauvaise idée d'avoir un switch basé sur un attribut d'un autre objet. Ceci implique de déplacer `getCharge()` dans la classe `Movie` et lui mettre un paramètre :

```
double getCharge(int daysRented)
```

Ensuite dans `Rental`, la méthode `getCharge()` retourne `movie.getCharge(daysRented)`

- Faire la même chose avec le calcul de « frequent renter point »

12) Héritage

On a plusieurs types de movie qui ont des façons différentes pour répondre aux mêmes questions.
On pourrait avoir 3 sous-classes de films, chacune a sa propre version pour `getCharge()`.
Cela permet de remplacer le switch en utilisant le polymorphisme mais cela ne va pas marcher parce qu'un film peut changer de classification au cours de sa vie ;
Il faut utiliser le pattern State (***Replace Type Code with State/Strategy***):
`Movie` a un état de type `Price`. Les 3 sous-classes de `Price` sont `RegularPrice, ChildrenPrice, NewReleasePrice` qui implémentent `getCharge()`
Faire le code correspondant au pattern State et les modifications nécessaires pour que le programme fonctionne. Compiler et tester.

13) Terminer par produire le diagramme de classes finale

A rendre

Déposer

- le code source de l'application,
- les classes de test produites
- les diagrammes UML demandés.