R5.A.8 Qualité de développement

TD-TP5

Exercice 1 : Observer

Le pattern Observer permet à un objet de réagir aux comportements d'un autre sans pour autant les lier fortement.

1) En utilisant l'interface java.util.Observer et la classe java.util.Observable de l'API Java (même si elles sont deprecated), écrire un petit système de log capable d'envoyer des mails (écrire dans un fichier) et d'afficher à l'écran les informations envoyées.

Vous devez définir :

- une classe **`LogBase`** pour l'objet à observer qui préviendra ses observateurs quand il recevra un message.

- une classe **`FileSaver`** dont les instances observeront l'objet de type `LogBase` et en réaction écriront le message dans un fichier

- une classe **`SendMail`** dont les instances observeront l'objet de type `LogBase` et en réaction enverront un message (on pourra se contenter d'afficher un message à l'écran)

2) Quel pattern pourrait-on mettre à contribution pour avoir une seule instance de la classe LogBase accessible dans notre application ?

Exercice 2 : Bridge

L'objectif de cet exercice est d'appliquer le pattern Bridge sur la petite application proposée.

La classe **`BaseList`** représente des listes et les implémente dans un `ArrayList`, et les classes **`NumberedList`** et **`OrnamentedList`** sont deux autres représentations des listes.

Pour l'instant ces représentations sont séparées ; on doit ajouter explicitement des éléments dans un objet NumberedList ou un objet OrnamentedList.

On aimerait que les 3 classes BaseList, NumberedList et OrnamentedList partagent toutes une même implémentation, c'est-à-dire le même ArrayList qui s'imprimera différemment selon l'objet utilisé.

Le pattern Bridge vous permet de séparer l'abstraction et l'implémentation, et plusieurs abstractions peuvent partager une même implémentation.

1) Modifier le code donné en appliquant les principes de Bridge.
2) Réaliser le diagramme de classes final (par rétro-conception ou non)

Ci- dessous la trace d'exécution que votre solution devra fournir.

Trace de la solution à obtenir

```
Adding elements to the list.
```

```
Creating an OrnamentedList object.

Creating an NumberedList object.

Printing out first list (BaseList)
      One
      Two
      Three
      Four

Printing out second list (OrnamentedList)
      + One
      + Two
      + Three
      + Four

Printing our third list (NumberedList)
      1. One
      2. Two
      3. Three
      4. Four
```