Université Bretagne Sud
IUT de Vannes
Département Informatique

BUT 3 – R5.A.08.Qualité de Dev
TP : Patrons de conception Bridge et Command

Exercice 1 : 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.

```
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
```

A rendre : l’ensemble des classes (fichiers .java) annotées (par des commentaires) par le rôle de chacune dans le patron Bridge et un diagramme de classe (fichier PNG)

Exercice 2 : Command

Cet exercice consiste à créer un système de contrôle pour l'automatisation de la maison (comme les lumières, les ventilateurs, etc.) en utilisant le patron Command.

Système d'automatisation de la maison

Imaginez que vous construisez un système d'automatisation de la maison qui peut contrôler divers appareils (par exemple, les lumières, les ventilateurs, les portes de garage). Chaque appareil doit prendre en charge des commandes de base telles que l'allumage/extinction et l'augmentation/diminution de la vitesse (pour les ventilateurs). Vous devez concevoir un système en utilisant le patron de conception Commande pour implémenter les fonctionnalités suivantes :

1. Allumer/éteindre les lumières.
2. Allumer/éteindre les ventilateurs et augmenter/diminuer la vitesse du ventilateur.
3. Ouvrir/fermer les portes de garage.

Exigences

1. Interface Commande : Créez une interface `Commande` qui définit une méthode `executer()` pour exécuter une commande.
2. Classes de Commandes Concrètes : Implémentez des classes de commandes concrètes pour chaque fonctionnalité :
    - `CommandeAllumerLumiere` et `CommandeEteindreLumiere` pour allumer et éteindre les lumières.
    - `CommandeAllumerVentilateur`, `CommandeEteindreVentilateur`, `CommandeAugmenterVitesseVentilateur` et `CommandeDiminuerVitesseVentilateur` pour contrôler les ventilateurs.
    - `CommandeOuvrirPorteGarage` et `CommandeFermerPorteGarage` pour manipuler les portes de garage.
3. Classes Récepteurs : Implémentez des classes représentant les appareils :
    - Classe `Lumiere` avec les méthodes `allumer()` et `eteindre()`.
    - Classe `Ventilateur` avec les méthodes `allumer()`, `eteindre()`, `augmenterVitesse()` et `diminuerVitesse()`.
    - Classe `PorteGarage` avec les méthodes `ouvrir()` et `fermer()`.
4. Classe Invocateur : Créez une classe `Telecommande` qui agit comme un invocateur. Elle doit avoir des méthodes pour définir une commande et l'exécuter.
5. Code Client : Montrez comment utiliser la `Telecommande` pour exécuter les commandes sur les appareils.

Défi Bonus

- Fonctionnalité Annuler : Étendez le système pour prendre en charge les opérations d'annulation pour chaque commande.
- Commandes Macro : Créez une `MacroCommande` qui peut exécuter plusieurs commandes séquentiellement.

Rendre sur Moodle le code et le diagramme de classe correspondant.