Exercices de Patterns (9)

Un salon de thé propose à ses clients un choix de différents parfums : de Earl Grey, Darjeeling, ... à Verveine ou Menthe. Pour gérer les commandes des clients, ils utilisaient des classes `Parfum` pour les différents thés et `Table` pour les emplacements au sein du salon. Ils ont vite constatés que leur logiciel créait énormément de petits objets tels que la table qui avait commandé et les parfums désirés, avec du code tel que `new Parfum("Camomille").servirThé(3, new Table(7));`

Ils ont donc décidé de créer des classes supplémentaires `Carte` et `Salle` et de réutiliser des objets prédéfinis. Après tout, la carte n'est pas souvent modifiée et il est très rare que l'on ajoute ou supprime des tables.

Quel pattern utilisez-vous dans cette solution ? Quelle est la correspondance entre les classes de cette application et les classes théoriques ? Même question pour les méthodes.

Le design arrêté est le suivant :

- L'interface `CommandeThé` déclare la méthode `void servirThé(int quantité, Parfum parfum, Table table)`.
- La classe `Parfum` stocke le nom du parfum (`String parfum`) et son prix (`double prix`). Le constructeur initialise les champs dans cet ordre. La classe se termine par la définition des getters correspondant aux champs.
- Afin de définir et de réutiliser les parfums, une classe `Carte` est créée. Elle stocke une `Map` de parfums dont les clés sont les noms de ceux-ci. De plus elle définit les méthodes `void ajouterParfum(Parfum parfum)`, `void supprimerParfum(Parfum parfum)` et `Parfum getParfum(String parfum)`.
- La classe `Table` garde le numéro de table et le nombre de places de cette table. Elle définit un constructeur et les deux getters.
- La classe `Salle` garde un tableau de `Table` organisé comme suit : à l'indice correspondant au numéro de la table. Elle définit une méthode d'initialisation qui prend en argument un objet de type `Properties`.
- La classe `SalonDeThé`
  - Implémente l'interface `CommandeThé`.
  - stocke dans la `Map chiffre`, le chiffre d'affaire réalisé pour chaque table.
  - Elle garde également une `Carte` et une `Salle`.
  - Son constructeur prend en paramètre la `Carte`, la `Salle` ainsi que deux strings correspondants à des noms de fichiers .properties :
    - Le premier liste les parfums à la carte sous la forme `nom=prix`.
    - Le second correspond aux tables de la `Salle` et ne contient que la seule propriété table avec comme valeur la liste des nombres de places de chaque table placés dans l'ordre des numéros des tables.
  - Elle définit également une méthode `void prendreCommande(String parfum, int quantité, int numéroDeTable)`. Cette méthode recherche le parfum et la table avant d'appeler la méthode `servirThé`. Elle met également à jour le `chiffre`. Au cas où le parfum ou la table ne serait pas trouvé, elle lance une exception appropriée (`ParfumInexistantException` ou `TableInexistanteException`).
  - La méthode `servirThé` peut ici se contenter de faire un `System.out.println`. Plus généralement, elle pourrait par exemple notifier les observateurs liés à une interface graphique chargée de simuler le service à l'aide d'une animation.
- La classe principale `MainThé` définit la méthode main suivante (par exemple, `prendreCommande` et `chiffre` doivent y être remplacé par des appels de méthodes sur le salon de thé que cette méthode définit) :

```
public static void main(String[] args) {
  try {
    prendreCommande("Thé vert", 2, 5);
  } catch (ParfumInexistantException e) {
    System.out.println("Le parfum demandé n'existe pas");
  } catch (TableInexistanteException e) {
    System.out.println("Numéro de table incorrect");
  }
  try {
    prendreCommande("Camomille", 1, 1);
  } catch (ParfumInexistantException e) {
```

```
    System.out.println("Le parfum demandé n'existe pas");
  } catch (TableInexistanteException e) {
    System.out.println("Numéro de table incorrect");
  }
  try {
    prendreCommande("Menthe", 1, 1);
  } catch (ParfumInexistantException e) {
    System.out.println("Le parfum demandé n'existe pas");
  } catch (TableInexistanteException e) {
    System.out.println("Numéro de table incorrect");
  }
  try {
    prendreCommande("Earl Grey", 1, 1);
  } catch (ParfumInexistantException e) {
    System.out.println("Le parfum demandé n'existe pas");
  } catch (TableInexistanteException e) {
    System.out.println("Numéro de table incorrect");
  }
  for (Table table : chiffre.keySet()) {
    System.out.println("Table " + table.getNuméroDeTable() + " : " +
                       chiffre.get(table) + " €");
  }
}
```