

UNIVERSIT  DE TOURS
FACULT  DES SCIENCES ET TECHNIQUES

Projet de Design Patterns (Refactoring)

Auteurs :

Abderrahim BENMELOUKA

Adama BAYO

Alexis BOYER

Hakim AMRAOUI

Hakim BEGHAMI

Huma ZUHAIR

R f rent :

Thierry BROUARD

30 avril 2021

Table des matières

1	Plusieurs possibilités de traitement	2
1.1	Intitulé	2
1.2	Réponse	2
1.2.1	Diagramme de classes	2
2	Parcours des collections par des iterators	3
2.1	Intitulé	3
2.2	Réponse	3
3	Plusieurs possibilités de tri	3
3.1	Intitulé	3
3.2	Réponse	3
3.2.1	Diagramme de classes	3
4	Possibilité d'anonymat des données	3
4.1	Intitulé	3
4.2	Réponse	4
4.2.1	Diagramme de classes	4
5	Utilisation des Factory	5
5.1	Intitulé	5
5.2	Réponse	5
5.2.1	Diagramme de classes	5
6	Interface graphique	5
6.1	Intitulé	5
6.2	Réponse	5
6.2.1	Aperçu de l'interface	6

1 Plusieurs possibilités de traitement

1.1 Intitulé

La sortie actuelle est en HTML, sur la console. C'est pas hyper pratique. Il faut trouver ici un moyen d'utiliser un objet chargé, par exemple, d'enregistrer cela dans un fichier HTML. Une autre possibilité serait d'ouvrir un fichier type Excel, et d'aller écrire les informations dans les bonnes cases. Un autre exemple d'objets pourrait être réservé au débogage, en écrivant les détails des structures de données sur la console. Il y a donc plusieurs possibilités de traitement, qui doivent pouvoir être échangées facilement.

1.2 Réponse

Nous généralisons et simplifions la méthode d'affichage trouvée dans `TEAMSProcessor.java`. Pour y parvenir, nous utiliserons le patron de conception de stratégie afin d'ajouter de la modularité à notre code.

Pour appliquer ce DP, nous allons ajouter un attribut statique à `TEAMSProcessor` de type `Display`. `Display` étant une classe abstraite de stratégie DP, sera la superclasse de toutes nos classes métiers qui feront l'affichage comme `DisplayHTML` qui produira et ouvrira un fichier HTML ou `DisplayExcel` qui produira et ouvrira un fichier Excel 2007.

Nous avons également supprimé la méthode `getHTMLCode()` de la classe `People` et placé toute la logique métier de l'affichage HTML dans la classe `DisplayHTML`.

Pour `DisplayExcel`, une bibliothèque externe (Apache POI) est importée via maven pour générer le fichier.

Dans `MainController`, l'afficheur choisi est injecté dans `TEAMSProcessor` à l'aide de la méthode `setDisplayer()`.

Ce DP nous permet d'ajouter autant de possibilités de traitement que nécessaire, il confère au code une certaine modularité tout en assurant sa robustesse.

1.2.1 Diagramme de classes

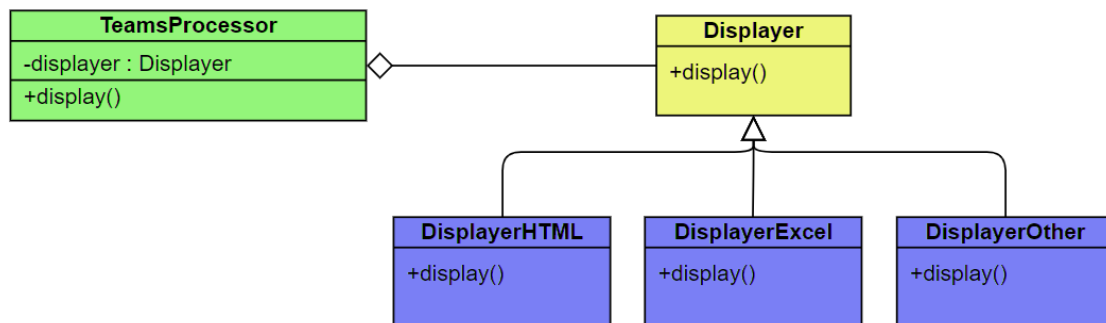


FIGURE 1 – Diagramme de classe pour le DP Strategy (Displayer)

Note : La classe `DisplayOther` ne sert qu'à montrer la modularité de ce DP, elle n'est pas présente dans le code.

2 Parcours des collections par des iterators

2.1 Intitulé

Les parcours des collections doivent se faire avec un `Iterator` (TP5).

2.2 Réponse

Nous remplaçons les boucles `Enhanced For` par l'`Iterator` DP partout où il y a une boucle de ce type. Assez simple comme concept.

3 Plusieurs possibilités de tri

3.1 Intitulé

Les données sont, par défaut, triées par durée de connexion croissante. Il sera possible d'avoir d'autres options, par ex. par nom d'utilisateur, ou par identifiant.

3.2 Réponse

En effet, le tri par défaut se fait par durée de connexion ascendante. Nous pouvons changer cela en utilisant le patron de conception `Stratégie` comme dans la première question. Cette fois en ajoutant un attribut statique de type `Sorter` et une méthode `sort()` qui appellera la méthode `sort()` du champ ajouté dans la classe `TEAMSPProcessor`. La classe abstraite `Sorter` est une superclasse qui contient une seule méthode `sort()`. Ses enfants remplacent cette méthode, d'où les différentes implémentations de tri.

Dans `MainController`, le trieur choisi est injecté dans `TEAMSPProcessor` à l'aide de la méthode `setSorter()`.

Comme pour la première question, ce DP est modulaire et extensible.

3.2.1 Diagramme de classes

4 Possibilité d'anonymat des données

4.1 Intitulé

Les données peuvent être anonymisées, ie. Les noms, voir les id, ne s'affichent pas. Cela peut se faire à deux niveaux : les données ne sont pas générées, ou bien elles ne sont pas affichées (neutralisation au niveau du CSS). On peut aussi avoir « sans planning » (juste la liste des connectés).

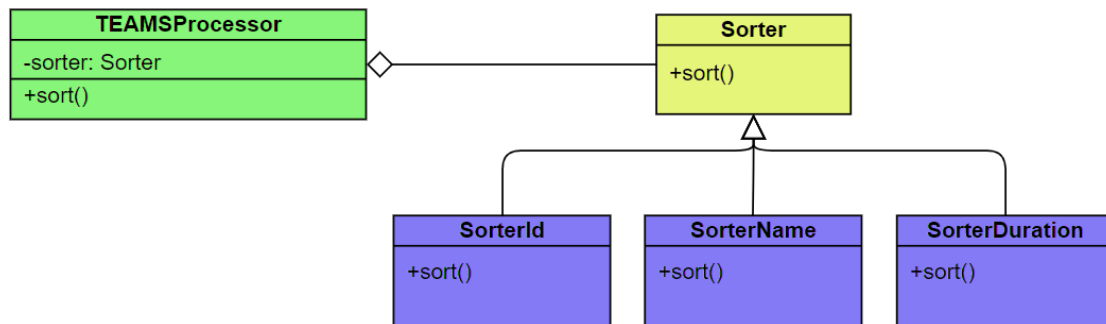


FIGURE 2 – Diagramme de classe pour le DP Strategy (Sorter)

4.2 Réponse

À l'instant, toutes les données sont affichées via HTML ou Excel. Nous allons changer cela en utilisant les DP Strategy et Decorator. Nous ajoutons un champ Extracteur statique dans la classe **TEAMSPProcessor**. Ce champ va contenir les données à extraire.

Nous créons d'abord la classe **Extractor** qui contienne elle-même un attribut **Extractor**, un constructeur pour définir cet attribut et une méthode `getData()`. Ensuite, nous créons trois classes qui héritent de **Extractor** (**ExtractorId**, **ExtractorName** et **ExtractorTime**). Ces classes définissent la méthode `getData()` qui renverra un objet **List** contenant les champs à afficher (nom et planning par exemple).

Dans **MainController**, l'extracteur choisi est fabriqué en utilisant le DP Decorator et injecté dans **TEAMSPProcessor** à l'aide de la méthode `setExtractor()`.

Ce DP fermera le code à la modification et l'ouvrira à l'extension.

4.2.1 Diagramme de classes

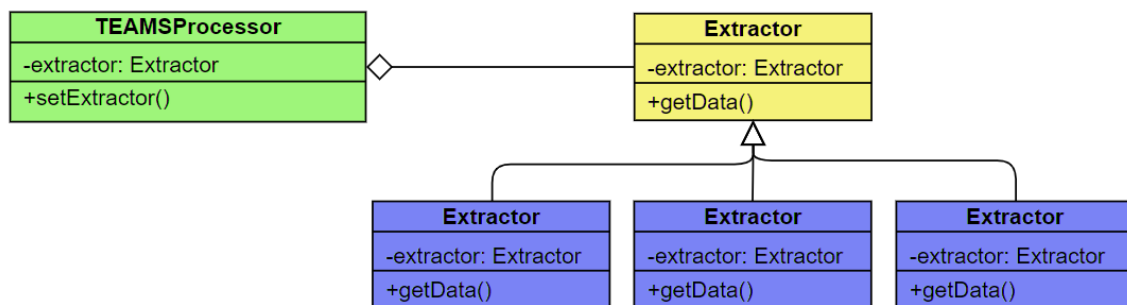


FIGURE 3 – Diagramme de classe pour le DP Strategy et Decorator (Extractor)

5 Utilisation des Factory

5.1 Intitulé

Autant que possible, les créations d'objets devraient reposer sur des Factory (TP4) (il y a plusieurs patterns dans cette famille).

5.2 Réponse

Nous avons décidé d'utiliser le Factory DP de base (pas abstract Factory). Le but de Factory DP est de choisir la classe correcte au moment de l'exécution, ce qui la rend très appropriée à appliquer dans notre MainController. Nous créons trois Factory pour les classes récemment créées (Displayeur, Sorter et Extractor), ces Factory ont une seule méthode statique qui retourne leur objet correspondant en utilisant les cases à cocher et les boutons radio de l'interface graphique comme arguments. Dans MainController, tout ce que nous devons faire est de passer ces cases à cocher et boutons radio à ces Factory pour créer l'objet Displayeur/Sorter/Extractor, puis définir cet objet comme attribut TEAMSPProcessor.

5.2.1 Diagramme de classes

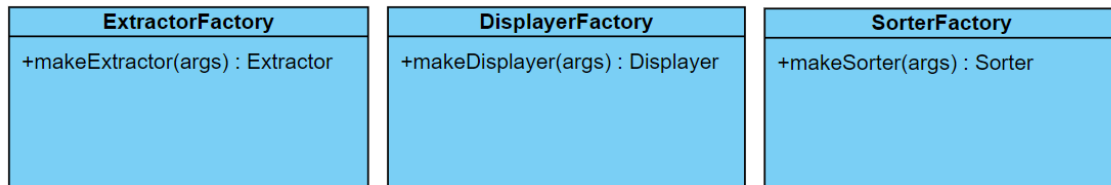


FIGURE 4 – Diagramme de classe pour les Factory

6 Interface graphique

6.1 Intitulé

Il n'y a pas, pour le moment, à proprement parler, d'interface utilisateur. Bien que celle-ci soit minimale, elle doit permettre, selon un MVC, de saisir : le fichier à traiter (en drag n drop), l'heure de début du cours, l'heure de fin (on va considérer que le cours n'est pas à cheval sur 2 jours), l'objet de la réunion. Une fois validé, le fichier est traité. Voir également le scénario type en dernière page.

6.2 Réponse

Nous avons opté pour la création d'une interface dans SceneBuilder. Cette interface a :

- un drag n drop pour saisir le fichier csv.
- les données sur le fichier d'entrée.

- des champs de texte pour saisir le nom du cours et limiter la période analysée ainsi que spécifier le nom du fichier de sortie.
- des boutons radio pour choisir le type de sortie (HTML ou Excel) et la méthode de tri (par id, nom ou durée).
- des cases à cocher pour anonymiser la sortie des données.
- et un sélecteur de dossier pour choisir le dossier de sortie (s’il n’est pas spécifié, un dossier par défaut dans le répertoire du projet est utilisé à la place).

Dans le MainController, nous récupérerons ces objets et validons les entrées et les utilisons dans la création des Factory et l’initialisation du TEAMSPprocessor.

6.2.1 Aperçu de l’interface

FIGURE 5 – Interface graphique