Nicolas BERNE
2023/2024
DE3

Génie Logiciel – DE3
TD 6 Noté

Consignes

*Le rendu de ce TD sera fait sur mootse sous forme d'une archive unique avec pour nom de fichier : prenom_NOM.zip est attendu. Il comportera vos documents sous un format lisible de type PDF et code source avec l'arborescence complète de vos fichiers *.py (attention à ne pas fournir un format xml généré par votre outil de modélisation ou les fichiers de configuration de votre IDE).*

Par faciliter de compréhension on parlera de la méthode __init__() de Python comme constructeur.

Le TD contient deux parties. Les consignes sont les suivantes :

- *La première vous fera travailler avec le design pattern « Itérateur ». Il vous est demandé de répondre aux questions et de ne pas résoudre le problème d'une autre façon.*
- *La deuxième partie vous fera travailler par groupe de 2 et selon la parité un groupe de 3 afin de confronter vos idées et d'avoir un esprit critique sur vos réalisations respectives.*

Nicolas BERNE
2023/2024
DE3

Partie 1 – Itérateur (2h30 – 15 points)

Le but de ce patron de conception est de cacher l'implémentation sous-jacente afin de manipuler des objets de façons identiques alors que l'implémentation est différente. L'intérêt est de pouvoir travailler sur des objets complexes (plus que ceux d'un exercice) en n'ayant pas besoin de connaître l'implémentation. En n'étant pas liés à l'implémentation, celle-ci peut aussi évoluer sans risque de casser le code appelant.

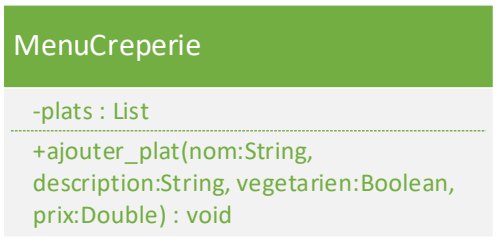
Implémentation 1

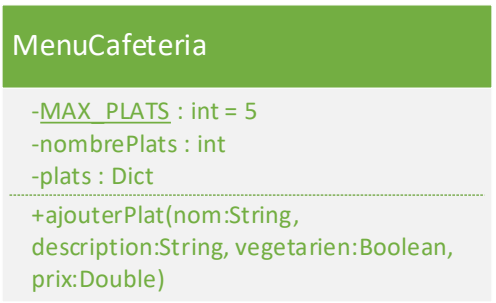
Créer un premier dossier (Impl1) pour stocker cette première implémentation

1. Définir une classe `Plat` qui comporte les propriétés suivantes :
    - nom
    - description
    - vegetarien
    - prix

Cette classe de base comportera un « constructeur » avec la méthode spéciale `__init__()`

2. Définir deux classes « MenuCreperie » et « MenuBrasserie » :





On utilisera le nom du plat comme clé pour le dictionnaire. Le nombre de menus de la cafétaria ne pourra pas dépasser 5.

La crêperie devra proposer les plats suivants qui seront ajoutés dans le constructeur :

Nom	Prix	Végétarien	Description
Galette Annie	11.90	N	Jambon blanc, emmental
Galette Jeannette	13.50	N	Jambon blanc, œuf, emmental
Galette Paulette	14.50	N	Jambon blanc, œuf, emmental, champignons à la crème
Galette Germaine	12.90	O	Légumes au choix : fondue de poireaux, épinards, ratatouille Maison selon la saison ou champignons à la crème

La cafétéria proposera les plats suivants qui seront ajoutés dans le constructeur :

Nom	Prix	Végétarien	Description
Salade du Forez	12.50	N	Saucisson chaud lyonnais artisanal, Râpées de pomme de terre maison
Salade bergère	12.50	N	Fourme fondue, Crottin de chèvre chaud
Salade bien-être	12.50	O	Assortiment de crudités

3. Créer une classe `Serveuse` avec une méthode `afficher_menu()` qui permet d'afficher tous les plats de la cafétéria et de la crêperie. Vous utiliserez l'écriture sur la sortie standard. Créer également une classe qui permet d'appeler cette dernière méthode.
   Utiliser la construction « `for i in range len(ma_liste)` » pour parcourir une liste et « `for cle in mon_dic` » pour par parcourir les clés du dictionnaire.
   Créer une classe `TestServeuse` pour permettre d'appeler la méthode.

4. Vrai ou Faux :
   - MenuCreperie et MenuCafeteria sont des implémentations concrètes pour une interface non définie.
   - La Serveuse n'implémente pas l'API Python Waitresse.
   - Si MenuCafeteria utilise un Set alors le code de Serveuse devra être revu.
   - L'implémentation de Serveuse respecte la notion d'encapsulation.
   - Nous avons du code dupliqué et une troisième implémentation imposerait une nouvelle boucle.
   - L'implémentation n'utilise pas un métalangage (XML/JSON) afin d'assurer l'interopérabilité des menus.
5. Quelles sont les défauts de cette première implémentation ?

Implémentation 2

Créer un dossier (Impl2) pour stocker cette deuxième implémentation.

1. Nous souhaitons encapsuler l'itération, pour cela introduisons l'interface `Iterateur` qui propose deux méthodes : `encore()` et `suivant()`. Pour ce faire utiliser une classe abstraite Python. La première méthode renverra `true` s'il reste encore un élément dans la structure de données parcourue `false` sinon, `suivant`() renverra une donnée, ici un plat.

2. En héritant de notre classe abstraite précédente implémenter une classe `IterateurMenuCafeteria` qui est un itérateur concret et qui permet d'itérer sur le menu de la Cafeteria. Cette classe comportera deux propriétés : une liste de plats et l'indice de la liste pour savoir quel plat renvoyer.

3. Modifier `MenuCafeteria` pour insérer une méthode (`creer_iterateur`) qui permet de créer cet itérateur et de le retourner au client.

4. Réaliser la même opération pour la crêperie avec la création d'un itérateur.

5. Modifier le code de `Serveuse` pour prendre en compte ces modifications. Celle-ci devra utiliser nos itérateurs maisons désormais.

6. Proposer un diagramme de classe reprenant l'ensemble des classes de votre code source.

Implémentation 3

Créer un dossier (Impl3) pour stocker cette troisième implémentation.

Contrairement à d'autres langages, Python n'utilise pas deux méthodes pour définir un Itérateur mais utilise la structure suivante avec l'utilisation d'une méthode spéciale `__next__` :

```
from collections.abc import Iterator

class CustomIt(Iterator):

    def __init__(self, my_list):
        self.my_list = my_list
        self.position = 0

    def __next__(self):
        if self.position < len(self.my_list):
            i = self.position
            self.position += 1
            return self.my_list[i]
        else:
            raise StopIteration

it = CustomIt([0,1,3])
for elem in it:
    print(elem)
```

Il faut hériter de la classe Iterator et implémenter la méthode spéciale `__next__` l'arrêt se fait en levant une exception plutôt qu'en implémentant une méthode `encore()`.

1. En se basant sur l'exemple donné, proposer une première implémentation en définissant un `IterateurMenuCafeteria` et `IterateurMenuCreperie` qui héritent de classe `Iterator` et permettent de parcourir votre liste ou le dictionnaire.
2. Proposer le diagramme de classe correspondant.

Implémentation 4

Créer un dossier (Impl4) pour stocker cette quatrième implémentation

1. Au lieu de créer notre propre itérateur, nous pouvons utiliser l'itérateur fourni avec le type liste. Il n'est alors plus nécessaire d'avoir une classe `IterateurMenuCafeteria` la méthode `creer_iterateur` pouvant être créer automatiquement à partir d'une liste sans opération manuelle. Idem nous pouvons utiliser l'itérateur d'un dictionnaire sur les valeurs car les clés ne nous intéressent pas. Il n'est alors plus nécessaire d'avoir une classe `IterateurMenuCreperie`. Proposer alors une solution qui utilise les itérateurs par défaut.

Partie 2 – Travail en binôme (0h30 – 5 points)

Avec votre binôme, comparer le travail réalisé pendant 10 minutes. Répondez alors chacun aux questions suivantes :

- Quels sont les points que vous avez réalisés différemment ? Pourquoi ?
- Si vous deviez reprendre le travail réaliser maintenant comment aborderiez-vous le TD ?
- Si vous deviez mettre une note pour la réalisation de votre binôme quelle serait-elle ? Pourquoi ?