TP - Gestion de Commandes pour une Pizzeria

Dans ce TP, vous allez modéliser un système de gestion de commandes pour une pizzeria en utilisant les concepts avancés de la Programmation Orientée Objet (POO) : **héritage**, **polymorphisme**, **composition**, **agrégation**, et **factory**. Vous travaillerez également avec des fichiers JSON pour simuler un menu et des commandes de clients.

Objectifs :

- Modéliser des produits (pizzas, boissons) et des commandes.
- Gérer des pizzas avec des ingrédients (composition).
- Traiter des commandes avec agrégation de produits.
- Utiliser une factory pour créer des produits à partir de fichiers JSON.
- Utiliser des **générateurs** et le **pattern matching** pour traiter les commandes efficacement.

Étape 1 : Modélisation des Produits et Composition

Objectif :

Modéliser les **pizzas** et **boissons**.

Tâches :

1. **Créer la classe `Ingredient`** :
   - Attributs : `name`.
   - Exemple : un ingrédient pourrait être “cheese”, “tomato”, etc.
2. **Créer la classe `Pizza`** :
   - Une pizza est composée de plusieurs ingrédients.
   - Attributs : `name`, `size` (small, medium, large), `price`.
   - Méthode : `add_ingredient(ingredient)` qui ajoute un ingrédient à la pizza.
3. **Créer la classe `Drink`** :
   - Attributs : `name`, `volume` (en ml), `price`.

Exemple de sortie attendue :

```
pizza = Pizza("Margherita", "medium", price=20)
pizza.add_ingredient(Ingredient("Cheese"))
pizza.add_ingredient(Ingredient("Tomato"))

drink = Drink("Coca-Cola", 500, 2)
```

Étape 2 : Gestion des Commandes (Agrégation)

Objectif :

Créer une classe `Order` qui contient plusieurs produits et permet de calculer le total d'une commande.

Tâches :

1. **Créer la classe `Order`** :
   - Attributs : `order_id`, `products` (liste de pizzas ou boissons).
   - Méthode : `add_product(product)` pour ajouter un produit à la commande.
   - Méthode : `total()` pour calculer le prix total de la commande.

Exemple de sortie attendue :

```
order = Order(1)
order.add_product(pizza)
order.add_product(drink)
print(f"Total order price: {order.total()} euros")
```

Étape 3 : Factory et Pattern Matching pour la Création des Produits

Objectif :

Utiliser une **factory** pour générer des objets `Pizza` ou `Drink` à partir d'instructions dans un fichier `commands.json` et utiliser le **pattern matching** pour traiter ces instructions.

Fichiers fournis :

- **menu.json** : Contient la liste des boissons disponibles.
- **commandes.json** : Contient les commandes à traiter.

Tâches :

1. **Factory `ProductFactory`** :
   - Créer une méthode `create_product` qui utilise le pattern matching pour déterminer s'il faut créer une pizza ou une boisson.
   - Les ingrédients des pizzas peuvent être prédéfinis ou générés à partir du fichier.
2. **Lecture des commandes** :
   - Lire le fichier `commands.json` et utiliser la factory pour générer les produits.

Structure du fichier `menu.json` :

```
{
    "drinks": [
        {"name": "Coca-Cola", "volume": 500, "price": 2},
        {"name": "Water", "volume": 500, "price": 1}
    ]
}
```

Structure du fichier `commandes.json` :

```
[
   {
      "order_id": 1,
      "items": [
         "Pesto",
         "Sprite",
         "Orange Juice",
         "Coca-Cola",
         "Pesto",
         "Vegetarian",
         "Coca-Cola",
         "Pesto"
      ]
   }
]
```

Bonus : Utilisation des Générateurs/Création d'un watcher

Objectif :

Traiter les commandes à l'heure d'arrivée. Surveiller le dossier d'input. À chaque fois que l'on ajoute un fichier de commande, il faut afficher sur la console : - Nombre d'items et prix de la commande.

Tâche :

Créer une fonction `read_orders(file)` qui renvoie une commande à chaque appel à l'aide du mot-clé `yield`.

Livrables :

- Le code des classes `Ingredient`, `Pizza`, `Drink`, `Order` et `ProductFactory`.
- La gestion des fichiers JSON pour le menu et les commandes.
- Une démonstration de la création et du traitement des commandes.
