TP : FoodFast

Bienvenue dans l'équipe de développement de **FoodFast**, une startup innovante qui révolutionne la livraison de repas. Suite à une croissance rapide, FoodFast doit améliorer son système de gestion des commandes et des livraisons pour offrir un service plus performant aux restaurants partenaires et aux clients finaux.

L'équipe technique est mobilisée pour concevoir une solution à la hauteur de ces nouveaux défis. Vous faites partie de cette équipe et votre mission est de créer un prototype qui pourra ensuite être intégré au système de production.

Objectif du Projet

FoodFast reçoit de plus en plus de commandes et doit s'assurer que chaque commande est livrée dans les meilleures conditions. Vous allez développer un prototype pour simuler l'envoi et la gestion des commandes entre plusieurs restaurants et une plateforme de livraison centralisée. Votre solution devra évoluer au fur et à mesure des exigences pour être plus robuste et flexible.

Votre première tâche est de concevoir une solution simple. Un restaurant doit pouvoir générer une commande en fonction d'un délai d'attente aléatoire et l'envoyer à une plateforme de livraison. La plateforme de livraison reçoit les commandes et les gère pour les dispatcher.

Question 1 : Créer les classes **Application.java**, **Restaurant.java**, **DeliveryPlateform.java, Order.java** et **Dish.java**

La classe Dish.java doit contenir un nom et d'une taille (enum S,M et L)

La classe Order.java doit contenir le restaurant de préparation, des plats commandés (Dish.java) et de leurs quantités, d'un montant initial et d'un lieu de livraison (dans ce cas un String est suffisant).

La classe Restaurant, doit contenir un nom et une méthode prepareOrder qui génère et retourne l'order (vous pouvez ajouter un appel à la méthode Thread.sleep(new Random().nexInt(3000)) pour simuler un temps d'attente.

La classe DeliveryPlateform qui contient une méthode "void delivery(Order order)" simulant la livraison d'une commande.

La classe Application.java, qui dans la méthode main, instancie une plateforme de livraison, un restaurant, et effectue quelques commandes.

Écrivez les tests unitaires associées aux méthodes.

Question 2 :

Après avoir observé la première version, l'architecte de FoodFast remarque que la solution manque de réactivité et propose d'introduire un modèle Observer. Ce modèle permettra à la plateforme d'être informée en temps réel des commandes générées par les restaurants, sans que le restaurant ait besoin de connaître les détails internes de la plateforme.
Classe à modifier :

1. **Restaurant** : notifier la plateforme dès qu'une commande est générée.

Classe à modifier :

1. **DeliveryPlatform** : s'abonner pour recevoir les notifications des commandes en temps réel.

Implémentez le pattern Observer afin que le restaurant notifie la plateforme de livraison (https://en.wikipedia.org/wiki/Observer_pattern)

Écrivez les tests unitaires associées aux méthodes.

Question 3 :

FoodFast collabore avec plusieurs restaurants, et chaque restaurant doit pouvoir s'abonner à la plateforme de livraison. Votre prototype doit donc maintenant gérer les commandes provenant de plusieurs sources de manière concurrente.
Classe à modifier :

1. **DeliveryPlatform** : gérer plusieurs restaurants, chacun pouvant s'abonner à la plateforme.

Question 4 :

Avec l'augmentation du volume de commandes, l'équipe technique remarque des doublons dans les commandes, ce qui cause des erreurs. Il est donc crucial de garantir que chaque commande soit unique pour éviter tout bug.
Classe à modifier :

1. **DeliveryPlatform** : assurer que chaque commande reçue soit unique.

Comment pourriez-vous gérer l'unicité des commandes et éviter les doublons ?

Écrivez les tests unitaires associées aux méthodes.

Question 5 :

Avec la croissance de FoodFast, le système doit devenir plus évolutif. Il est désormais nécessaire de passer à une architecture plus souple, capable de supporter des dizaines de restaurants, de plateforme de livraison et des centaines de commandes en même temps. L'architecte propose d'introduire un EventBus pour gérer cette complexité. Le développement de ce composant sera itératif sur les prochaines questions.

1) Dans un premier temps, créer un nouveau composant **EventBus.java.** Ce composant doit avoir une référence à une liste unique de **Subscriber**. **Subscriber.java** est une interface possédant une méthode **void handleEvent(Order order)**. Il faut aussi une méthode **void subscribe(Subscriber subscriber)** qui permet d'ajouter un subscriber.

2) Ecrivez les tests associés

Question 6 :

Pour le moment, l'**EventBus** ne gère que les commandes, et les besoins font que le composant doit prendre en charge plusieurs types d'événements.
En effet, les restaurants souhaitent connaître l'état d'avancement de la livraison de la commande
Créer la classe de type enum **EventType.java**, contenant **ORDER_PREPARED** et **DELIVERY**.

1) Créez une interface **Event.java**, et les classes **OrderEvent.java**, qui encapsule un objet Order (ajouter un uuid comme identifiant unique se ce n'est pas déjà fait), et **DeliveryEvent.java** qui encapsule un order, un nouveau statut et un numéro de livraison (uuid).
   Les statuts de livraisons sont IN_DELIVERY et DELIVERED. La méthode simulant la livraison envoie dans un premier temps un event au statut IN_DELIVERY, et après un temps d'attente entre 2 et 15 secondes envoie un nouvel événement au statut DELIVERED.
2) Adaptez la structure de données et les méthodes de l'**EventBus** afin de prendre en charge les événements **Event.java**.
3) Adaptez les tests
4) Ajouter un test permettant de vérifier que les 2 types d'évènements sont gérés par l'eventbus:
   - La plateforme de livraison est notifiée des commandes devant être livrées
   - les restaurants doivent afficher les infos de commandes et de livraison quand leurs commandes sont livrées (et uniquement les leurs).

Question 7 :

Des erreurs peuvent survenir, telles qu'un restaurant échouant à envoyer une commande ou la plateforme de livraison qui ne traite pas correctement une commande. Il est essentiel de gérer ces erreurs pour garantir la stabilité du système.
Créer les erreurs liées à la préparation et à la livraison de la commande et envoyées les de façons aléatoires dans le code et implémenter la gestion d'erreur.

1. Créez deux classes d'exception personnalisées : **OrderPreparationException** et **DeliveryProcessingException**.

2. Simulez ces erreurs en introduisant des cas aléatoires dans les méthodes correspondantes. Par exemple, une probabilité de 20 % qu'une commande échoue à être préparée ou livrée.

3. Implémentez un mécanisme pour gérer ces exceptions:

- Le système ne doit pas s'arrêter en cas d'erreur.
- Les erreurs sont enregistrées dans un **service de gestion d'erreurs** (ex. : ErrorManagementService).

Question 8 :

FoodFast souhaite conserver un journal des événements importants pour faciliter la maintenance et l'audit.
Ajoutez une classe Logger.java pour enregistrer :

1. Les commandes reçues par la plateforme.
2. Les livraisons effectuées.

- Implémentez le principe du **Singleton** pour que le journal soit partagé dans tout le système.
- Chaque entrée doit contenir une horodatation (timestamp) et une description de l'événement.

Testez le bon fonctionnement du logger.

Question 9 :

FoodFast souhaite envoyer des notifications aux clients lorsque leur commande est en cours de préparation ou livrée.

1. Ajoutez une classe NotificationService avec une méthode sendNotification(String message).
2. Ajoutez une classe Customer (nom, prénom, adresse de livraison, numéro de téléphone).
3. Modifiez le traitement des événements OrderEvent et DeliveryEvent pour inclure les notifications automatiques à envoyer aux clients concernés.

Testez que :

1. Les notifications sont envoyées au bon moment.
2. Les bons clients reçoivent les notifications appropriées.

Question 10 :

Afin d'améliorer l'observabilité du système, l'architecte travaillant sur le projet souhaite sauvegarder en base l'intégralité des events en base de données.

1) Configurez une connexion JDBC avec une base comme PostgreSQL (Annexe 1)

2) Sauvegarder chaque événement dans une table events avec les informations suivantes
    a) Identifiant unique.
    b) Type de l'événement (Order, Delivery …).
    c) Date et heure de l'événement.
3) Développez une méthode permettant de rejouer les événements en erreur

Test :
Vérifiez que les données sont correctement sauvegardées dans la base.

Question 11:
Chaque commande livrée représente non seulement une réussite pour les restaurants, mais aussi une occasion de satisfaire pleinement un client, le service de facturation devient l'élément clé pour garantir que chaque transaction soit correctement enregistrée et traitée. Grâce à ce nouveau service de facturation, chaque livraison sera désormais accompagnée de la création automatique d'une facture, assurant ainsi un suivi rigoureux et transparent des transactions.

Créer un service de livraison qui émet une facture au statut CREATED lors du passage d'une commande et qui doit passer à CLOSED après la livraison

Question 12:
Les clients ou restaurants peuvent annuler des commandes ou demander des retours après livraison, ce qui nécessite de gérer ces situations spéciales dans le système.

1. **Classes à modifier :**

    • Order : Ajoutez un attribut pour l'état de la commande, qui peut être PENDING, COMPLETED, ou CANCELLED.

    • DeliveryPlatform : Ajoutez une méthode cancelOrder(Order order) pour annuler une commande et la retirer du système, avec une gestion de l'état correspondant.

2. **Test unitaire à implémenter :**

    • Créez un test pour vérifier qu'une commande peut être annulée avant ou après sa livraison et que l'état de la commande est mis à jour en conséquence. Vous devrez également tester que la commande annulée n'est pas livrée.

Annexe 1:
Connection à une base de données:

0. Démarrer une base de données dans docker

docker run --name my-postgres -e POSTGRES_USER=admin -e POSTGRES_PASSWORD=password -e POSTGRES_DB=mydatabase -p 5432:5432 -d postgres:latest pour lancer un docker)

1. Ajouter la dépendance PostgreSQL dans Maven

Ajoutez la dépendance du driver PostgreSQL dans le fichier pom.xml :
<dependencies>

```
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.6.0</version>
    </dependency>
</dependencies>
```

2. Configurer les propriétés de connexion

Ajoutez un fichier application.properties ou db.properties dans le répertoire src/main/resources :

```
db.url=jdbc:postgresql://localhost:5432/your_database_name
db.username=your_username
db.password=your_password
db.driver=org.postgresql.Driver
```

3. Charger les propriétés

Utilisez un utilitaire comme java.util.Properties pour charger les informations :

```
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

public class DatabaseConfig {
    public static Properties loadProperties() throws IOException {
        Properties properties = new Properties();
        try (FileInputStream input = new FileInputStream("src/main/resources/db.properties")) {
            properties.load(input);
        }
        return properties;
    }
}
```

4. Établir une connexion à la base de données

Utilisez DriverManager pour établir la connexion :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class DatabaseConnection {
    public static Connection getConnection() {
        try {
            Properties properties = DatabaseConfig.loadProperties();
            String url = properties.getProperty("db.url");
            String username = properties.getProperty("db.username");
            String password = properties.getProperty("db.password");

            // Charger le driver (optionnel avec JDBC 4+)
            Class.forName(properties.getProperty("db.driver"));
```

```
            // Établir la connexion
            return DriverManager.getConnection(url, username, password);
        } catch (IOException | SQLException | ClassNotFoundException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to connect to the database");
        }
    }
}
```