Rapport de Conception

Projet : Système de Gestion d'Événements Distribué

Nom : Mvogo David

Objectif

Ce projet a été réalisé dans le but de concevoir une application simple, évolutive et bien structurée permettant de gérer différents types d'événements, tels que des conférences et des concerts. À travers ce système, j'ai cherché à mettre en pratique les principes de conception orientée objet tout en intégrant des fonctionnalités modernes comme la sérialisation et les tests unitaires.

Modélisation orientée objet

J'ai adopté une structure hiérarchique avec une classe abstraite `Evenement` représentant les propriétés communes à tous les événements (nom, date, lieu, etc.), et deux sous-classes `Conference` et `Concert` pour en détailler les spécificités. Cela permet de factoriser le code tout en conservant la souplesse nécessaire à l'extension.

Gestion des participants

Chaque participant est représenté par un objet `Participant`. Il peut s'inscrire ou se désinscrire d'un événement, sous réserve de la capacité maximale. Pour notifier un participant lorsqu'un événement est annulé, j'ai utilisé le pattern Observer, en rendant `Participant` capable de réagir automatiquement aux changements d'état d'un événement.

Gestion des événements

Tous les événements sont centralisés dans un gestionnaire singleton `GestionEvenements`. Cela permet de garantir une cohérence globale dans l'enregistrement et la récupération des événements, peu importe leur nature.

Exceptions personnalisées

Pour capturer proprement les erreurs, j'ai défini deux exceptions personnalisées :

`CapaciteMaxAtteinteException` pour signaler une tentative d'inscription hors limite, et `EvenementDejaExistantException` pour empêcher l'ajout d'un événement en double. Cela renforce la robustesse du système.

Sérialisation JSON

La sérialisation a été implémentée avec la bibliothèque Jackson, permettant d'écrire et de lire les événements au format JSON. La sauvegarde fonctionne parfaitement. En revanche, j'ai rencontré des difficultés techniques lors de la désérialisation liées à la gestion du polymorphisme (`Evenement` étant une classe abstraite). Ce problème, bien compris, pourra être résolu dans un environnement Maven ou avec une meilleure gestion du classpath.

Utilisation de Streams et Lambdas

Pour exploiter les collections d'événements et de participants, j'ai utilisé les flux (`Stream`) de Java. Cela m'a permis de trier, filtrer ou compter les éléments de façon fluide, tout en rendant le code plus lisible.

Tests unitaires

Les principales fonctionnalités ont été testées avec JUnit : inscription d'un participant, gestion des exceptions, ajout d'événements, et test de la sérialisation JSON. Ces tests m'ont permis de valider le comportement attendu des classes développées. Seule la désérialisation reste à fiabiliser dans un cadre technique plus stable.

Conclusion

Ce projet m'a permis de mettre en pratique des concepts essentiels du développement logiciel moderne, d'intégrer des outils de qualité comme JUnit et Jackson, et de réfléchir à la conception logicielle de manière plus rigoureuse. Malgré un léger blocage sur la désérialisation, l'ensemble du système est fonctionnel, bien structuré et prêt à évoluer.