Rapport Final - Système de Gestion d'Événements

Michele Kenmeugne

Mai 2025

Introduction

Ce document présente l'implémentation complète d'un système de gestion d'événements développé en Java suivant les principes de la POO. L'application permet la gestion de concerts et conférences avec inscription de participants et persistance des données.

1 Architecture Technique

1.1 Structure des Packages

— **src/main/java**
  — **Premier** : Classes métier (Evenement, Concert, Conference)
  — **Design_Pattern_logique_Metier** : Implémentation des patterns
  — **Exception** : Exceptions personnalisées
— **src/test/java** : Tests unitaires

1.2 Technologies Utilisées

— **Jackson** : Pour la sérialisation JSON
  — Format clé-valeur léger
  — Pas de support des commentaires
— **JUnit 5** : Framework de tests

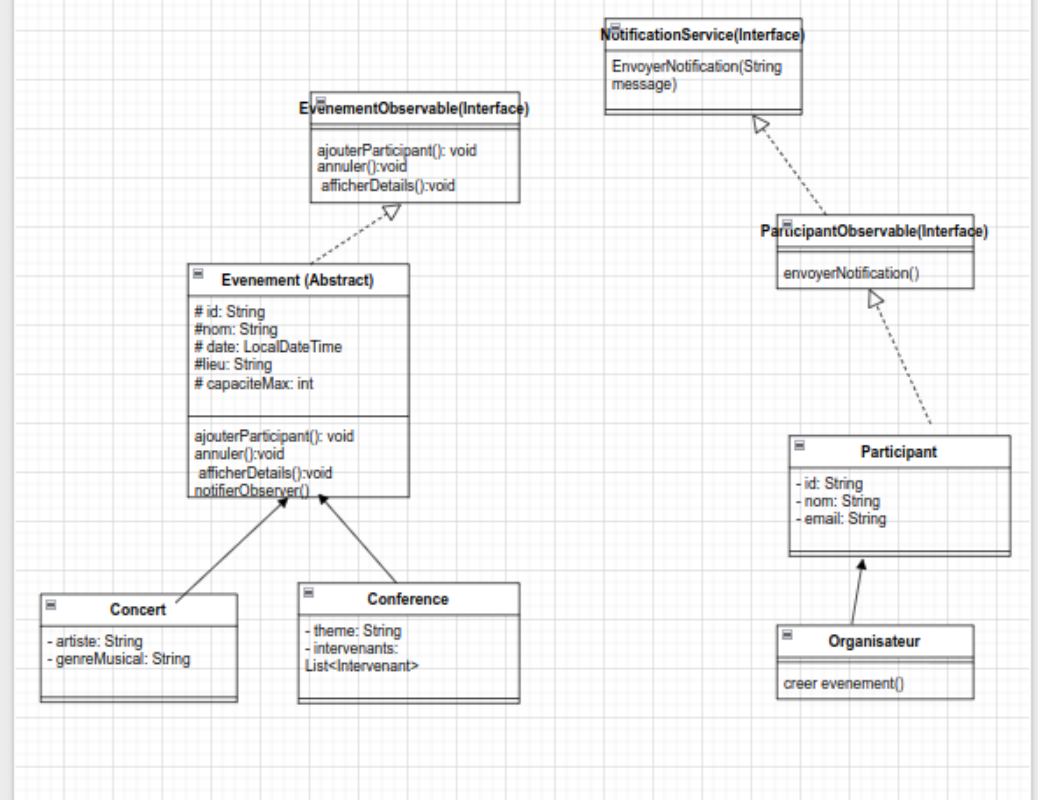

FIGURE 1 – Image

2 Design Patterns Implémentés

2.1 Singleton

— Garantit une seule instance de `GestionEvenements`

— Accès centralisé à la liste des événements

```
public class GestionEvenements {
    private static GestionEvenements instance;

    private GestionEvenements() {}

    public static GestionEvenements getInstance() {
        if (instance == null) {
            instance = new GestionEvenements();
        }
        return instance;
    }
}
```

2.2 Observer

— Analogie : Abonnement à un magazine
  — `Evenement` = Magazine
  — `ParticipantObserver` = Abonné
— Fonctionnement :
  — L'événement maintient une liste d'observateurs
  — Notification automatique lors des changements

```
public interface EvenementObservable {
    void ajouterParticipant(ParticipantObserver o);
    void notifierObservers(String message);
}
```

3 Persistance des Données

3.1 Sérialisation JSON

— Avantages :
  — Structure légère (clé-valeur)
  — Facile à lire/modifier
— Limites :
  — Désérialisation complexe pour les types polymorphiques

3.2 Exemple de Fichier JSON

```
{
  "type": "concert",
  "id": "c1",
  "nom": "Concert Zouk",
  "date": "2025-05-20 20:00:00",
  "lieu": "Yaound ",
  "capacity": 100,
  "artiste": "Fanny J"
}
```

4 Tests Unitaires

4.1 Stratégie de Test

— Tests isolés pour chaque fonctionnalité
— Validation des cas limites
— Tests d'intégration pour la sérialisation

4.2 Exemple de Test

```
@Test
void testAjoutParticipant() {
    Concert concert = new Concert(...);
    ParticipantObserver p = new ParticipantObserver("
   test@mail.com");

    assertDoesNotThrow(() -> concert.ajouterParticipant(p));
    assertThrows(CapaciteMaxAtteinteException.class,
        () -> concert.ajouterParticipant(p));
}
```

5 Gestion de Version

— Utilisation de Git avec une branche `master`
— Hébergement sur GitHub
— Workflow : Développement via des branches feature

Conclusion

Bilan

— Architecture modulaire et extensible
— séparation des responsabilités

Perspectives

— Interface graphique
— Gestion avancée des conflits
— Couverture de test complète

— FIN DU DOCUMENT —