Système de Gestion d'Événements

Rapport de Projet - Programmation Orientée Objet

Étudiant : HEUDEP DJANDJA BRIAN B
Classe : 3GI 2025
Projet : GestionEvents
Date : 25 Mai 2025

Application Java avec Design Patterns
Gestion d'événements (Conférences & Concerts)

École Nationale Supérieure Polytechnique de Yaoundé

Table des matières

1 Introduction

1.1 Contexte

Ce projet s'inscrit dans le cadre du cours de Programmation Orientée Objet (POO) et vise à mettre en pratique les concepts avancés de la POO à travers la conception d'un système de gestion d'événements distribué.

1.2 Objectifs

Le système développé permet de gérer différents types d'événements (conférences, concerts) avec les fonctionnalités suivantes :

- Inscription et gestion des participants
- Gestion des organisateurs et intervenants
- Système de notifications en temps réel
- Persistance des données au format JSON
- Interface utilisateur intuitive

1.3 Technologies Utilisées

- **Java 14+** : Langage principal de développement
- **JavaFX** : Interface graphique utilisateur
- **Jackson** : Sérialisation/désérialisation JSON
- **JUnit 5** : Tests unitaires
- **Maven** : Gestion des dépendances et build

2 Architecture du Système

2.1 Structure Modulaire

Le projet est organisé selon une architecture modulaire respectant les bonnes pratiques :

```
GestionEvents/
          src/main/java/
                models/          # Classes m tier
                service/         # Services de notification
                gestion/         # Logique principale
                persistence/     # Sauvegarde donn es
                observer/        # Pattern Observer
                exceptions/      # Exceptions personnalis es
                factory/         # Pattern Factory
          src/test/java/       # Tests unitaires
          src/main/resources/  # Fichiers de configuration
```

Listing 1 – Structure du projet

2.2 Classes Principales

2.2.1 Hiérarchie des Événements

- **Evenement** (abstraite) : Classe de base contenant les attributs communs
- **Conference** : Spécialisation pour les conférences avec thème et intervenants
- **Concert** : Spécialisation pour les concerts avec artiste et genre musical

2.2.2 Gestion des Utilisateurs

- **Participant** : Classe de base pour les utilisateurs
- **Organisateur** : Hérite de Participant, peut créer des événements
- **Intervenant** : Hérite de Participant, participe aux conférences

3 Design Patterns Implémentés

3.1 Singleton Pattern

GestionEvenements (Singleton)

Objectif : Garantir une seule instance de gestion des événements
Avantage : Centralisation et évitement des doublons
Usage : `GestionEvenements.getInstance()`

3.2 Observer Pattern

Système de Notifications

Objectif : Notification automatique des participants
Avantage : Couplage faible entre événements et participants
Usage : Notification automatique lors d'annulation d'événements

3.3 Factory Pattern

NotificationFactory

Objectif : Création flexible de services de notification
Avantage : Facilite l'ajout de nouveaux types (Email, SMS)
Usage : `NotificationFactory.creerService(TypeNotification.EMAIL)`

3.4 Strategy Pattern

Services de Notification

Objectif : Changement dynamique de méthode de notification
Avantage : Interface commune pour Email et SMS
Usage : Sélection du service selon le contexte

4 Choix de Conception

4.1 Architecture Modulaire

La séparation en packages distincts offre plusieurs avantages :

- **Maintenabilité :** Code organisé et facile à modifier
- **Testabilité :** Tests unitaires ciblés par module
- **Réutilisabilité :** Composants indépendants
- **Évolutivité :** Ajout facile de nouvelles fonctionnalités

4.2 Gestion des Exceptions

Des exceptions personnalisées ont été créées pour une gestion précise des erreurs :

- `CapaciteMaxAtteinteException` : Événement complet
- `EvenementDejaExistantException` : Évite les doublons
- `ParticipantDejaInscritException` : Inscriptions multiples
- `EvenementNonTrouveException` : Événement inexistant

4.3 Persistance JSON

Le choix de JSON plutôt qu'une base de données se justifie par :

- **Simplicité :** Mise en place rapide
- **Lisibilité :** Fichiers facilement modifiables
- **Portabilité :** Aucune dépendance externe
- **Suffisance :** Adapté à un projet éducatif

5 Principes SOLID

5.1 Single Responsibility Principle (SRP)

Chaque classe a une responsabilité unique et bien définie.

5.2 Open/Closed Principle (OCP)

Les classes sont ouvertes à l'extension (nouveaux types d'événements) mais fermées à la modification.

5.3 Liskov Substitution Principle (LSP)

Les sous-classes `Concert` et `Conference` peuvent remplacer la classe de base `Evenement`.

5.4 Interface Segregation Principle (ISP)

Utilisation d'interfaces spécifiques (`EvenementObservable`, `ParticipantObserver`).

5.5 Dependency Inversion Principle (DIP)

Dépendance vers les abstractions plutôt que les implémentations concrètes.

6 Fonctionnalités Implémentées

6.1 Gestion des Événements

- Création de conférences et concerts
- Modification des détails d'événements
- Annulation avec notification automatique
- Recherche et filtrage d'événements

6.2 Gestion des Participants

- Inscription à des événements
- Désinscription
- Gestion de la capacité maximale
- Historique des participations

6.3 Système de Notifications

- Notification par email
- Notification par SMS
- Notifications en temps réel
- Historique des notifications

7 Tests et Validation

7.1 Tests Unitaires

Des tests complets ont été implémentés avec JUnit 5 :

- Tests de la logique métier (`GestionEvenementsTest`)
- Tests des opérations sur les événements (`EvenementTest`)
- Tests de persistance (`GestionPersistanceTest`)
- Couverture de code supérieure à 70%

7.2 Validation des Fonctionnalités

- Inscription/désinscription de participants
- Gestion des exceptions personnalisées
- Sérialisation/désérialisation JSON
- Notifications en temps réel

8 Installation et Utilisation

8.1 Prérequis

- Java 14 ou supérieur
- Maven 3.6+
- IDE compatible (IntelliJ IDEA recommandé)

8.2 Installation

```
# Cloner le projet
git clone https://github.com/BrianBrusly/GestionEvents.git

# Construire le projet
mvn clean install

# Ex cuter les tests
mvn test

# G n rer le rapport de couverture
mvn test jacoco:report
```

Listing 2 – Commandes d'installation

9 Conclusion

Ce projet a permis de mettre en pratique de nombreux concepts avancés de la Programmation Orientée Objet :

9.1 Acquis Techniques

- Maîtrise des design patterns classiques
- Application des principes SOLID
- Gestion avancée des exceptions
- Sérialisation de données complexes
- Tests unitaires avec couverture de code

9.2 Compétences Développées

- Conception d'architecture modulaire
- Programmation événementielle
- Gestion de projet avec Maven
- Documentation technique

9.3 Perspectives d'Amélioration

- Interface web avec Spring Boot
- Intégration d'une base de données
- API REST pour les services externes
- Notifications push en temps réel
- Système de paiement pour les événements payants

Le système développé respecte les bonnes pratiques de développement et offre une base solide pour de futures extensions.

Projet réalisé par HEUDEP DJANDJA BRIAN B
3GI 2025 - École Nationale Supérieure Polytechnique de Yaoundé
25 Mai 2025