Rapport de Développement

Conception d'une application de Gestion d'Événements

Fait par : FEZE DJOUMESSI Fred
Encadrant : Dr. KUNGNE
Date : 26 mai 2025

Table des matières

1 Introduction

Ce rapport expose la conception d'une application de gestion d'événements reposant sur les principes de la programmation orientée objet (POO). Le système offre aux organisateurs la possibilité de planifier, consulter, enregistrer ou annuler des événements, tandis que les utilisateurs peuvent y participer via une fonctionnalité d'inscription. Le développement s'appuie sur le langage Java et met en œuvre divers fondements de la POO, notamment l'héritage, l'utilisation d'interfaces, la manipulation de collections, ainsi que l'application du patron de conception Singleton.

2 Objectifs du projet

— Offrir une fonctionnalité dédiée à la planification et à l'enregistrement d'activités publiques ou privées, telles que des concerts ou des colloques.

— Mettre en place un mécanisme d'adhésion et de désengagement des utilisateurs vis-à-vis des événements proposés.

— Structurer la gestion des profils d'organisateurs ainsi que le suivi des événements qu'ils supervisent.

— Intégrer un système de notifications permettant d'informer les utilisateurs en temps réel des mises à jour ou changements liés aux événements.

3 Technologies et outils

— Langage : Java 21

— Maven : gestionnaire de dépendances et d'exécution

— Jackson : bibliothèque de sérialisation JSON

— JUnit 5 : tests unitaires

— Swing : interface graphique

4 Modélisation des Classes

Voici un aperçu des principales classes et de leurs responsabilités :

4.1 Classe abstraite : Evenement

— Attributs : id, nom, date, lieu, capaciteMax.

— Méthodes : ajouterParticipant(), annuler(), afficherDetails().

4.2 Sous-classes

Conference

— Attributs spécifiques : theme, intervenants (List<Intervenant>).

Concert

— Attributs spécifiques : artiste, genreMusical.

4.3 Participant

— Attributs : id, nom, email.

4.4 Organisateur (hérite de Participant)

— Attribut : evenementsOrganises (List<Evenement>).

4.5 NotificationService (interface)

— Méthode : envoyerNotification(String message).

4.6 GestionEvenements (Singleton)

— Attribut : evenements (Map<String, Evenement>).

— Méthodes : ajouterEvenement(), supprimerEvenement(), rechercherEvenement().

4.7 Exceptions personnalisées

— CapaciteMaximaleAtteinteException : levée si on dépasse la capacité max.

— EvenementDejaExistantException : gère les doublons (amélioration possible).

5 Sérialisation JSON

La sérialisation des événements est réalisée avec Jackson, grâce à l'annotation polymorphique :

@JsonTypeInfo(use = JsonTypeInfo.Id.NAME, include = JsonTypeInfo.As.PROPERTY, property = "type") @JsonSubTypes(@JsonSubTypes.Type(value = Concert.class, name = "concert"), @JsonSubTypes.Type(value = Conference.class, name = "conference"))

5.1 Classe utilitaire EvenementSerializer

— sauvegarderEvenements(List, fichier) : écrit dans un fichier JSON.

— chargerEvenements(fichier) : lit un tableau d'événements avec leurs types.

5.2 Support des types Java modernes

Le module JavaTimeModule est utilisé pour gérer LocalDateTime.

6 Interface graphique

La classe EvenementGUI propose une interface complète pour :

— saisir un événement

— choisir le type (concert / conférence)

— afficher la liste

— sauvegarder en JSON

— annuler le dernier événement et envoyer les notifications

Elle utilise des composants :

— JComboBox, JTextField, JButton, JTextArea

— JOptionPane pour les erreurs

— SwingUtilities.invokeLater pour la gestion du thread GUI

7 Notifications asynchrones

7.1 Classe NotificationService

Utilise CompletableFuture.runAsync() pour simuler l'envoi de messages sans bloquer l'interface.

CompletableFuture.runAsync(() -> Thread.sleep(1000) ; System.out.println("Notification envoyée à " + p.getNom()) ;) ;

7.2 Intégration dans Evenement.annuler()

Lorsqu'un événement est annulé, tous les participants reçoivent une notification asynchrone simulée (console).

8 Design pattern

Un design pattern Observer simplifié est utilisé pour notifier automatiquement les participants lorsqu'un événement est annulé. Chaque Participant agit comme un Observer.

9 Tests unitaires

Des tests ont été créés pour :

— Ajouter un participant (succès / échec)

— Sérialisation / désérialisation

— Annulation et notifications

— Observer pattern

La commande suivante exécute les tests :
mvn test

10 Architecture et Design

10.1 Concepts utilisés

— **Héritage** : pour différencier les types d'événements.

— **Interface** : NotificationService permet d'implémenter différents canaux (mail, SMS. . .).

— **Singleton** : GestionEvenements assure une instance globale pour gérer les événements.

— **Collections Java** : List, Map utilisées pour gérer les données dynamiquement.

10.2 Structure du projet (Packages proposés)

— **modele** : classes métiers (Evenement, Participant, Organisateur...)
— **services** : NotificationService, GestionEvenements.
— **main / ui** : point d'entrée, interface console ou GUI.

11 Fonctionnalités de l'application

— Création et suppression d'un événement par un organisateur.
— Inscription et désinscription à un événement par un participant.
— Affichage des détails d'un événement.
— Notification des participants.
— Recherche d'événements par identifiant.

12 Difficultés rencontrées

1. Support graphique manquant en environnement non X11.
2. Erreur HeadlessException
3. Gestion du polymorphisme JSON

13 Perspectives d'amélioration

— Ajouter la persistance en base de données (ex :PostgreSQL)
— Créer un login pour les participants
— Intégrer un système de messagerie simulée
— Afficher les événements dans un tableau avec filtres
— Utiliser JavaFX pour une interface plus moderne

14 Conclusion

Ce projet est un exemple structuré et complet de developpement java pour construire une application modulaire, évolutive et réutilisable. Il applique les principes SOLID, intègre une architecture MVC simplifiée, met en oeuvre SWING, JSON, et la POO avancée. Il peut facilement être étendu avec une base de données ou une interface graphique pour devenir une application complète.