Rapport de TP final
Application de Gestion d'Événements

FONGANG NDE BRYAN
Encadrant : Dr W. Kungne

Mai 2025

Table des matières

1 Énoncé

Dans le cadre du TP final de Programmation Orientée Objet, il nous a été demandé de concevoir une application Java permettant de gérer différents types d'événements : conférences, concerts, etc. Cette application devait mettre en œuvre des concepts avancés tels que :

— L'héritage, le polymorphisme, les interfaces,
— Les patrons de conception (Observer, Singleton, Factory...),
— La gestion des exceptions personnalisées,
— La persistance via la sérialisation JSON,
— L'asynchronisme (envoi différé de notifications),
— Une interface graphique via JavaFX,
— Des tests unitaires avec JUnit.

L'application devait permettre l'ajout, la modification, la suppression d'événements, la gestion des participants, ainsi que l'envoi de notifications dynamiques en temps réel.

2 Diagramme UML

Le diagramme UMLde classe metier suivant illustre les principales classes, leurs relations ainsi que les interactions principales du système :

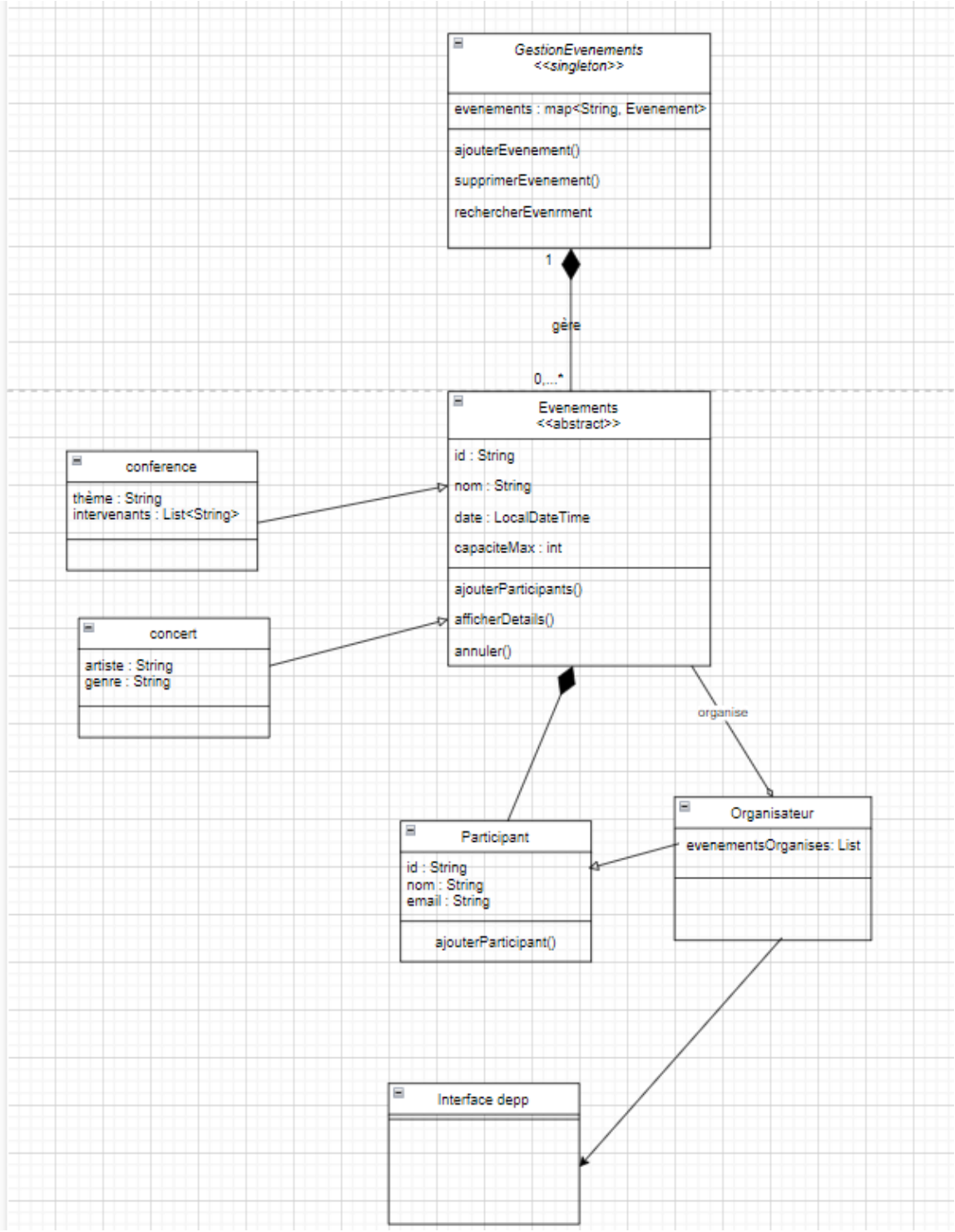


FIGURE 1 – Diagramme UML du système de gestion d'événements

Ce diagramme modélise notamment :

— Une classe abstraite `Evenement`, spécialisée par `Concert` et `Conference`,

— Un `Participant` qui peut être observateur d'un événement,
— Un `Organisateur` qui hérite de `Participant` et gère plusieurs événements,
— Un gestionnaire global `GestionEvenements` (Singleton),
— Le service de notification `NotificationService`.

3 Choix des outils et technologies

Langages et Frameworks

— **Java 11** : langage principal, supportant les fonctionnalités POO modernes.
— **JavaFX** : bibliothèque graphique pour la conception de l'interface utilisateur.

IDE et Outils de développement

— **IntelliJ IDEA** : environnement utilisé pour le développement et les tests.
— **JUnit 5** : utilisé pour écrire les tests unitaires.
— **Git / GitHub** : versionnage du code et partage du dépôt.

Organisation du code

Le projet est structuré selon une architecture MVC simplifiée, répartie en plusieurs packages :

— `model` : contient les entités métiers.
— `controller` : logiques de contrôle et communication avec l'interface.
— `utils` : gestion de la persistance, notifications et exceptions.

Persistance

Les événements sont enregistrés en JSON dans un fichier local. La sérialisation est réalisée manuellement (sans bibliothèques externes comme Jackson), afin de conserver un contrôle précis sur le format et le processus de lecture/écriture.

Tests unitaires

Les tests couvrent :
— L'inscription/désinscription des participants,
— La détection des erreurs (capacité maximale, événement existant),
— La sérialisation/désérialisation des événements.
Tous les tests ont été validés avec succès, garantissant une robustesse du système.

4 Maquette de l'interface

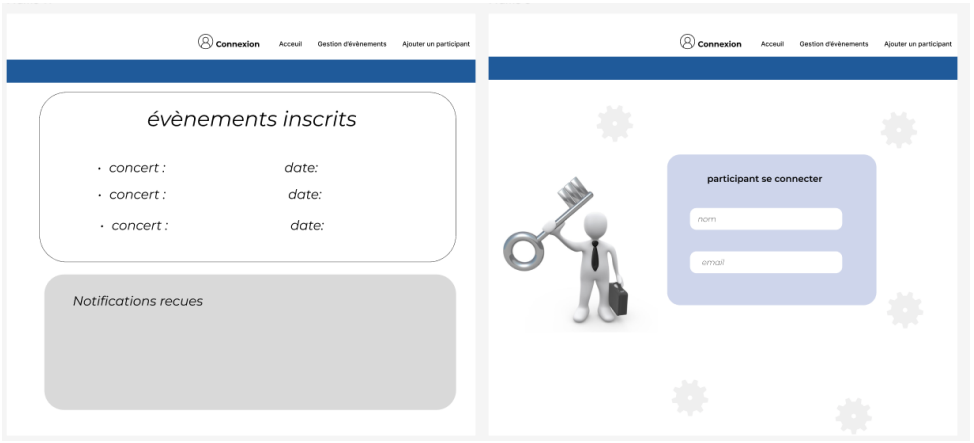
L'interface graphique a été conçue avec JavaFX, en respectant un agencement en plusieurs vues (onglets) :

— Un onglet d'accueil avec les actualités et les événements à venir,

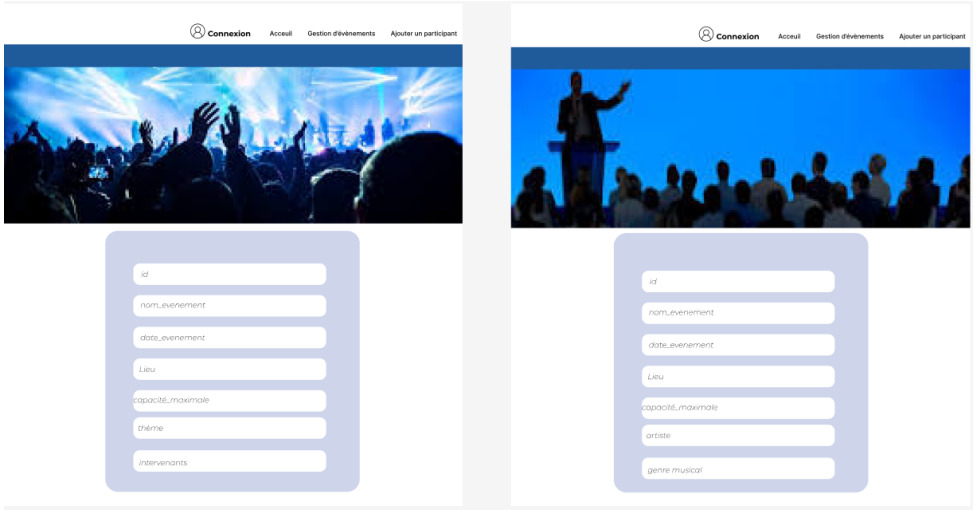
— Une section pour la gestion des événements (ajout, suppression, modification),

— Une section participant avec la possibilité de s'inscrire à des événements et recevoir des notifications.

Les figures suivantes montrent des captures de la maquette :



(a) Page participant



(b) Interface gestionnaire

FIGURE 2 – Aperçu de la maquette JavaFX