

# Rapport TP Final POO

## Application de gestion d'événements

Ngapgou Farelle  
3<sup>ème</sup> année Génie Informatique  
ENSPY

Mai 2025

### 1. Introduction

Ce projet a pour objectif de développer une application de gestion d'événements (concerts, conférences, etc.) permettant à un organisateur de créer, modifier, rechercher et supprimer des événements, avec un système de persistance des données. L'objectif est de nous familiariser avec des concepts avancés de la POO, à savoir les design patterns (Observer, Singleton, Factory), les tests unitaires en Java, la programmation asynchrone, ainsi que les concepts d'héritage, d'encapsulation, des streams et des lambdas.

Ce rapport a pour but de décrire le fonctionnement de notre système tout en justifiant nos choix de conception.

### 2. Principe de fonctionnement

#### 2.1 Pour qui est notre système ?

Au vu de l'énoncé du TP, la cible du système n'a pas été explicitement définie. Nous avons donc décidé de concevoir un système accessible à toute personne possédant l'ID de l'événement. Ainsi, cette personne peut modifier, supprimer, ajouter un participant, désinscrire un participant, rechercher un événement, etc.

**Pourquoi ce choix ?** L'objectif du TP étant de se familiariser avec certains concepts avancés de la POO, il a semblé judicieux de commencer par un système simple et fonctionnel plutôt que de concevoir un système complexe dans lequel la plupart des concepts ne seraient pas maîtrisés.

#### 2.2 Fonctionnalités de notre système

Notre système propose plusieurs fonctionnalités regroupées en deux paquets :

- **Événement** : actions relatives aux événements
  - Créer un événement
  - Modifier un événement

- Supprimer un événement
- Rechercher un événement
- **Participant** : actions relatives aux participants d'un événement
  - Inscrire un participant
  - Désinscrire un participant

### 3. Architecture logicielle

L'application est structurée autour des entités suivantes :

- **Événement** (classe abstraite) : classe de base pour tous les types d'événements.
- **Concert** et **Conférence** : classes dérivées d'Événement avec des attributs spécifiques.
- **Organisateur** : personne créant l'événement.
- **GestionEvenements** : classe Singleton centralisant les opérations.

Les classes communiquent entre elles via des objets bien encapsulés.

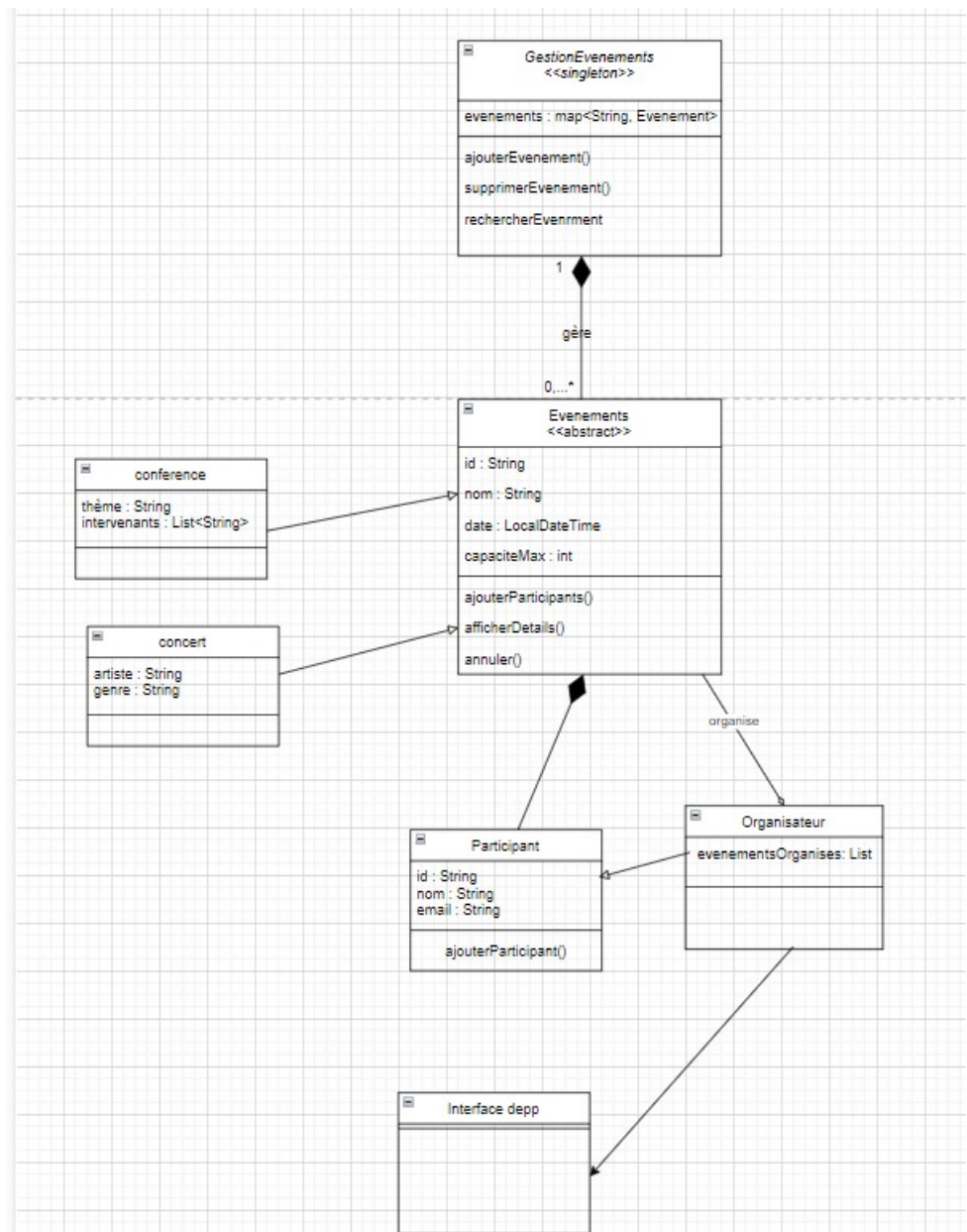
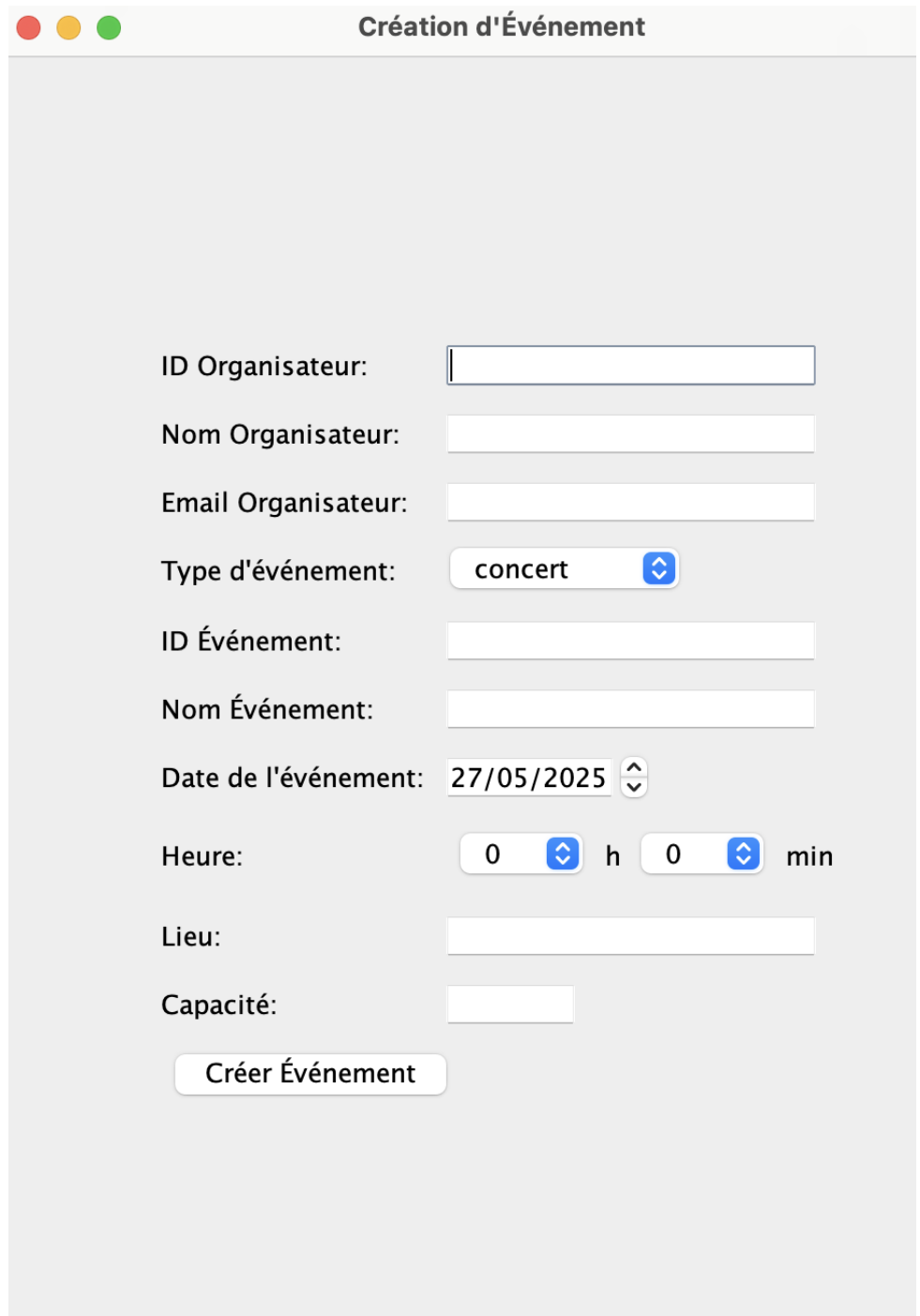


Diagramme UML :

## 4. Cas d'utilisation et concepts de la POO

### 4.1 Création d'un événement

L'utilisateur saisit les données de l'organisateur et les informations de l'événement. Les champs sont validés, une instance est créée puis ajoutée au gestionnaire. On utilise ici la sérialisation et la désérialisation : une fois les informations entrées, on récupère la liste des événements depuis le fichier JSON, on ajoute le nouvel événement créé ainsi que l'organisateur comme premier participant, puis on sauvegarde à nouveau dans le fichier JSON. Voici



Création d'Événement

ID Organisateur:

Nom Organisateur:

Email Organisateur:

Type d'événement: concert

ID Événement:

Nom Événement:

Date de l'événement: 27/05/2025

Heure: 0  h 0  min

Lieu:

Capacité:


Créer Événement

la fiche pour l'inscription :

Selon le choix de l'utilisateur, les champs manquants s'ajoutent automatiquement.

## 4.2 Modification d'un événement

L'utilisateur saisit les nouvelles valeurs demandées. On resauvegarde le fichier JSON en mémoire, en utilisant la sérialisation. On applique également le design pattern Observer,




The screenshot shows a window titled "Modifier Évén" with a standard macOS-style title bar (red, yellow, green buttons). The window contains four text input fields labeled "ID :", "Nom :", "Lieu :", and "Capacité :". Below these fields are two buttons: "Modifier" and "Quitter".

puisque les participants doivent être notifiés des modifications.

### 4.3 Recherche d'un événement

Aucun concept spécifique n'est implémenté ici, on réalise une recherche classique en uti-

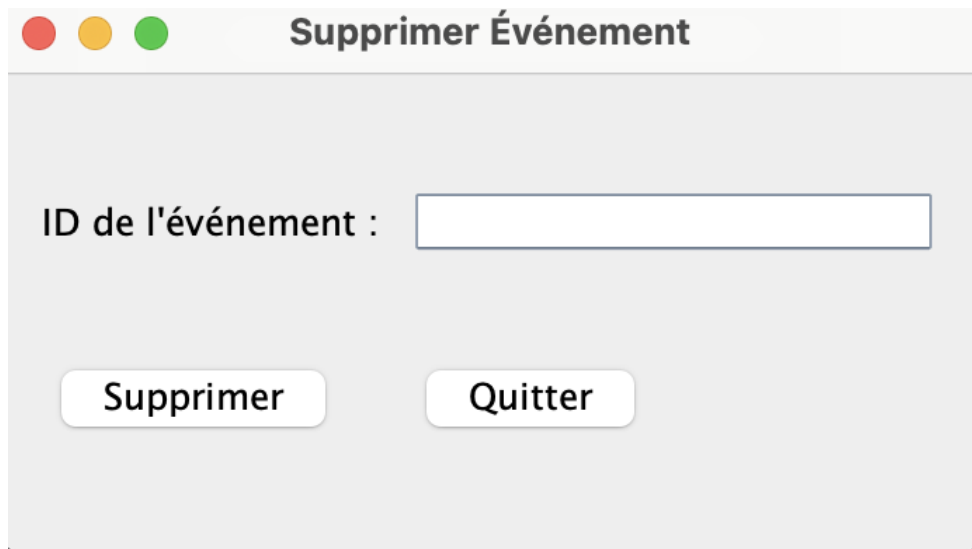


The screenshot shows a window titled "Rechercher Événement" with a standard macOS-style title bar. It features a text input field labeled "ID de l'événement :". Below this field is a large, empty rectangular box, likely for displaying search results. At the bottom of the window are two buttons: "Rechercher" and "Quitter".

lisant l'ID de l'événement.

### 4.4 Suppression d'un événement

L'utilisateur entre l'ID de l'événement. On resauvegarde ensuite le fichier JSON. On utilise la sérialisation ainsi que le design pattern Observer pour notifier les participants des

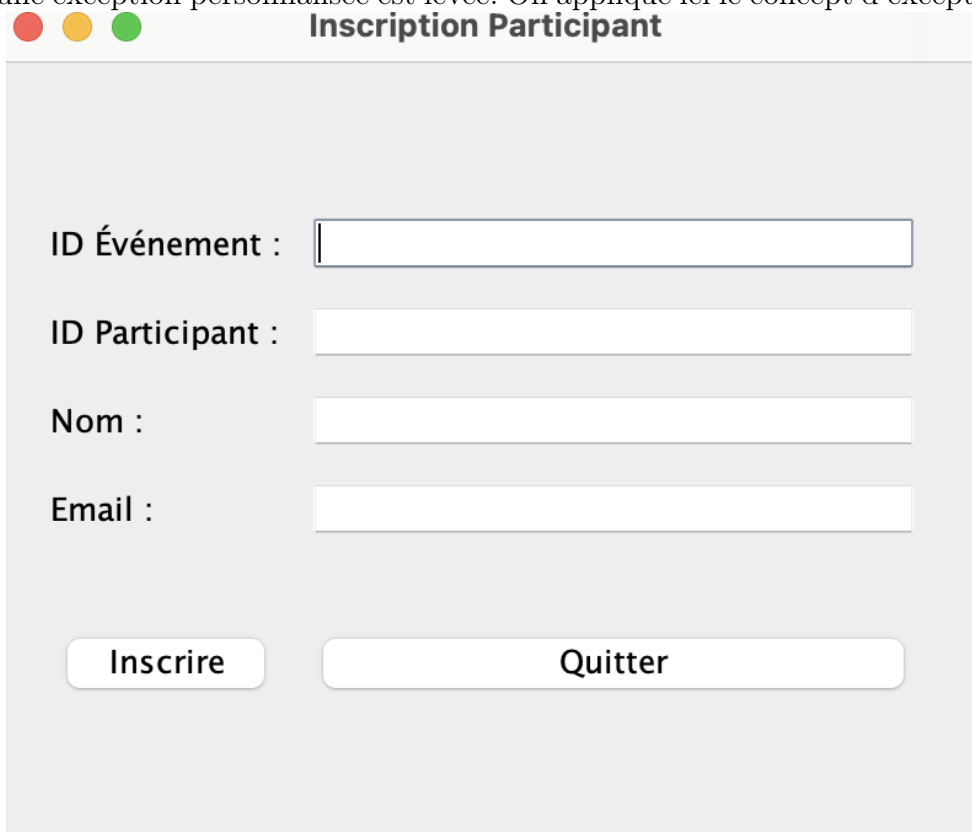


The screenshot shows a Java Swing window titled "Supprimer Événement". It has a standard macOS-style title bar with red, yellow, and green window control buttons. The main content area is light gray and contains a label "ID de l'événement :" followed by a white text input field. Below the input field, there are two white buttons with rounded corners: "Supprimer" on the left and "Quitter" on the right.

modifications.

#### 4.5 Ajout d'un participant

Le participant est ajouté à la liste si la capacité maximale n'est pas dépassée. En cas de dépassement, une exception personnalisée est levée. On applique ici le concept d'exception

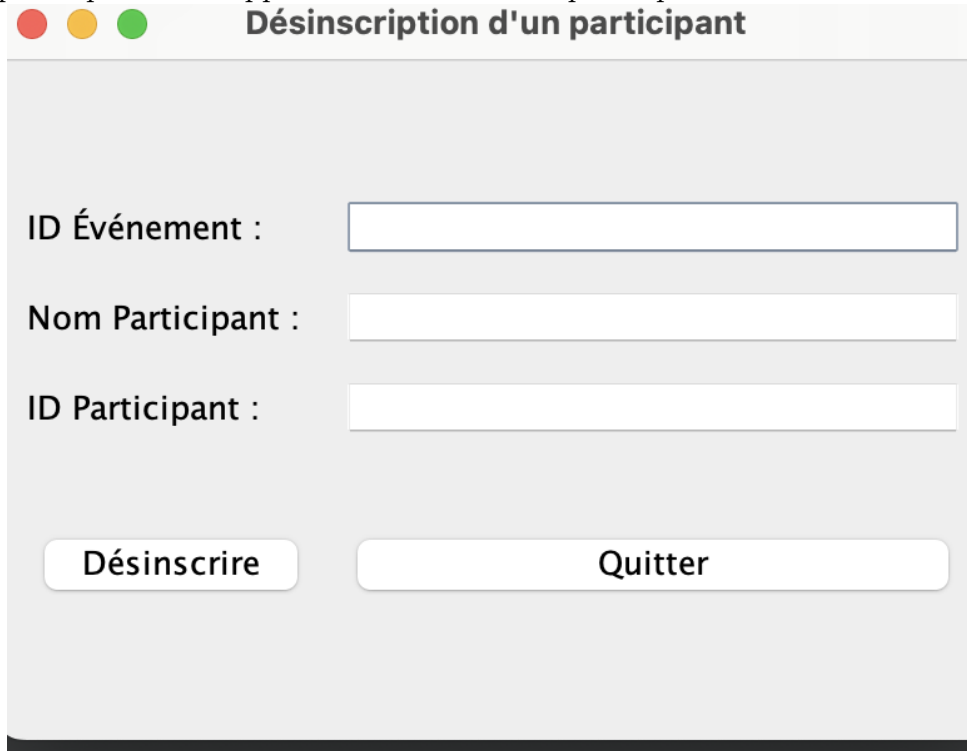


The screenshot shows a Java Swing window titled "Inscription Participant". It has a standard macOS-style title bar with red, yellow, and green window control buttons. The main content area is light gray and contains four labels with corresponding text input fields: "ID Événement :", "ID Participant :", "Nom :", and "Email :". At the bottom, there are two white buttons with rounded corners: "Inscrire" on the left and "Quitter" on the right.

personnalisée.

## 4.6 Suppression d'un participant

Le participant est supprimé de la liste des participants de l'événement s'il y était



The screenshot shows a Java Swing window titled "Désinscription d'un participant". The window has a standard title bar with three colored buttons (red, yellow, green). The main content area is light gray and contains three text input fields. The first field is labeled "ID Événement :", the second "Nom Participant :", and the third "ID Participant :". Below these fields are two buttons: "Désinscrire" and "Quitter".

inscrit.

## 5. tests

Pour les tests , on a utiliser JUnit . la première classe à tester la principale a été celle de GestionEvenement car elle contient les méthodes principale qui inclut les concept de la POO . Toutes les méthodes de cette classe ont été testé et ces test ont permis de ce rendre compte des erreurs relatives à certaines fonction et on a pu apporter des correction nécessaire

## 6. Avantages de notre conception

- Séparation claire des responsabilités.
- Réutilisabilité du code (ex. : les types d'événements peuvent être étendus).
- Utilisation de standards modernes (JSON, POO).
- Bonne maintenabilité.

## 7. Limites et pistes d'amélioration

- Augmenter les tests unitaires.
- Implémenter la programmation asynchrone pour simuler l'envoi des mails en temps réels.
- Étendre le système aux personnes ne possédant pas l'ID de l'événement.
- Ajouter une authentification des utilisateurs.

## 8. Conclusion

Ce projet a permis de mettre en pratique les principes de la programmation orientée objet en Java, en construisant une application modulaire, maintenable et fonctionnelle avec persistance des données. Il peut être facilement enrichi par de nouvelles fonctionnalités.