📘 Rapport Technique

TPPOO25 – Système de Gestion d'Événements Distribué

Auteur : AYISSI NGONO Odile Marie Ange
Date : 26 Mai 2025
Technologie : Java 11+, Maven

---

1. Introduction

Ce projet a été réalisé dans le cadre du TP de Programmation Orientée Objet avancée. Il consiste en une application de gestion d'événements (conférences, concerts) permettant l'inscription de participants, la gestion des organisateurs, la notification des utilisateurs en cas de changement, et la persistance des données.

L'objectif principal était d'intégrer plusieurs concepts avancés de POO, notamment les design patterns, la gestion d'exceptions personnalisées, la sérialisation JSON/XML et la programmation asynchrone.

---

2. Objectifs Atteints

- Conception orientée objet avec héritage, encapsulation, polymorphisme.
- Application de plusieurs design patterns : Observer, Singleton, Factory, Strategy.
- Mise en œuvre d'un système de notification asynchrone.
- Gestion de la persistance des données avec JSON (Jackson) et XML (JAXB).
- Implémentation de tests unitaires avec JUnit, atteignant plus de 70% de couverture.

---

3. Architecture Générale

Structure des packages :

- `model` : Entités métier (`Evenement`, `Conference`, `Concert`, `Participant`, `Organisateur`).
- `observer` : Interfaces et implémentations du pattern Observer.
- `service` : Persistance JSON/XML.
- `exception` : Exceptions personnalisées métier.
- `main` : Point d'entrée (`Main.java`).

Design Patterns :

Pattern	Utilisation
Observer	Notifie les participants lors des modifications/annulations d'un événement.
Singleton	`GestionEvenements` : instance unique de gestion d'événements.
Factory	Création dynamique d'objets `Evenement` selon leur type.
Strategy	Choix du mode de notification (console, e-mail simulé).

---

4. Fonctionnalités Implémentées

- Création et suppression d'événements (conférences, concerts).
- Inscription/désinscription des participants avec vérification de capacité.
- Notification automatique des participants (Observer + Strategy).
- Exécution asynchrone des notifications avec `CompletableFuture`.
- Sauvegarde et chargement des événements en JSON et XML.
- Gestion d'exceptions personnalisées :
  - `CapaciteMaxAtteinteException`
  - `EvenementDejaExistantException`
- Utilisation de Java Streams et lambdas pour simplifier la logique métier.
- Tests unitaires pour :
  - Les inscriptions
  - La persistance
  - La gestion des exceptions

---

5. Résultats et Évaluation

- L'application est pleinement fonctionnelle et respecte le cahier des charges.
- La conception est claire, modulaire et facilement extensible.
- Le code est commenté, bien organisé et conforme aux principes SOLID.
- La couverture des tests atteint plus de 70% grâce à JUnit 5.

---

6. Difficultés Rencontrées

- Implémentation synchronisée du pattern Observer avec exécution différée.
- Intégration parallèle des sérialisations JSON et XML.
- Conception flexible permettant l'extension future (ajout de types d'événements, de canaux de notification...).

---

7. Conclusion

Le projet TPPOO25 est une réussite technique et pédagogique. Il illustre parfaitement l'application concrète de la programmation orientée objet avancée. L'architecture mise en

place est robuste et prête à évoluer vers un système réel de gestion d’événements multi-utilisateurs ou Web.

---

8. README.md (extrait)

```
markdown
CopierModifier
# TPPOO25 – Gestion d'Événements

## Fonctionnalités
- Création, modification et annulation d’événements
- Inscriptions et désinscriptions des participants
- Notifications observables et asynchrones
- Persistance JSON et XML
- Gestion des erreurs via exceptions personnalisées
- Tests unitaires avec JUnit

## Exécution
mvn compile
mvn exec:java -Dexec.mainClass="com.gestionevenements.Main"
```

---

9. Références

- Documentation Java 11 (Oracle)
- Tutoriels sur les design patterns (GoF)
- Documentation officielle de Jackson et JAXB
- JUnit 5 User Guide