RAPPORT DE PROJET FINAL
Programmation Orientée Objet

NKOLO ATANGANA Stacy Julie

26 mai 2025

Résumé

Ce projet vise à implémenter une application de gestion d'événements en utilisant JavaFX pour l'interface graphique et Jackson pour la sérialisation JSON. Les concepts clés de POO (héritage, encapsulation, dépendances fonctionnelles) sont mis en œuvre pour structurer l'application. Le rapport détaille l'architecture, les fonctionnalités, les défis rencontrés et les pistes d'amélioration.

Table des matières

1 Introduction

Les différents modules et librairies utilisés dans ce projet ont été choisis pour leur efficacité :

— **JavaFX** : Pour l'interface graphique interactive.
— **Jackson** : Pour la sérialisation/désérialisation JSON des données (événements, utilisateurs).

Bien que le projet présente des lacunes dues à mon manque d'expérience, il intègre plusieurs concepts vus en classe, tels que :

— L'**héritage** entre classes (ex : `Evenement` → `Concert`).
— Les **dépendances fonctionnelles** entre les interfaces graphiques.

2 Structure du Projet

2.1 Organisation des fichiers

— `src/sac/` : Dossier principal contenant le code source.
  — `MainApplication.java` : Point d'entrée de l'application.
  — `interfaces/` : Classes UI (accueil, menu, listes d'événements).
  — `composants/` : Classes persistantes (`Evenement`, `Concert`, `Artiste`).
  — `mecanismes/` : Interfaces/classes utilitaires (sans attributs).
  — `serialisation/` : Gestion de la sérialisation JSON.
  — `ressources/` : Images (backgrounds, etc.).

2.2 Diagramme de classes (Description)

La hiérarchie des classes repose sur :

— Classe mère `Evenement` avec des sous-classes spécialisées (`Concert`, `Exposition`, etc.)
— Relations de dépendance entre les classes UI (`MenuUI` → `AccueilUI`)

3 Fonctionnalités Implémentées

3.1 Menu principal

— **Consulter les événements** :
  — Inscription/désinscription des participants.
  — Affichage des détails (date, lieu, artistes).
— **Organiser un événement** :
  — Réservé aux organisateurs (identifiant requis).
  — Accès à une liste personnelle d'événements + liste publique.

3.2 Gestion des modifications

— Un organisateur peut modifier ses événements.
— Les participants inscrits reçoivent une notification.
— Mise à jour en temps réel des listes (via sérialisation JSON).

4 Concepts de POO Utilisés

4.1 Héritage

Exemple avec la classe mère `Evenement` et ses sous-classes :

```
public class Evenement {
    protected String titre;
    // ...
}

public class Concert extends Evenement {
    private List<Artiste> artistes;
    // ...
}
```

4.2 Dépendances fonctionnelles

Les classes UI dépendent les unes des autres pour la navigation :

```
public class MenuUI {
    private AccueilUI accueil; // R f rence    la fen tre
    pr c dente
    // ...
}
```

5 Difficultés Rencontrées et Solutions

5.1 Sérialisation JSON

- **Problème** : Gestion des exceptions lors de la lecture/écriture des fichiers JSON.
- **Solution** : Utilisation de `ObjectMapper` (Jackson) avec gestion des erreurs :

```
try {
    objectMapper.writeValue(new File("data.json"), evenements);
} catch (IOException e) {
    System.err.println("Erreur de s rialisation : " + e.
    getMessage());
}
```

5.2 Interface graphique

- **Problème** : Synchronisation entre les fenêtres JavaFX.
- **Solution** : Utilisation de `ObservableList` pour mettre à jour les listes dynamiquement.

6 Tests Unitaires avec JUnit

6.1 Configuration

Ajoutez JUnit 5 dans `pom.xml` :

```xml
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.9.0</version>
    <scope>test</scope>
</dependency>
```

6.2 Exemple de test

Test de la classe `Evenement` :

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class EvenementTest {
    @Test
    public void testAjoutParticipant() {
        Evenement event = new Concert("Concert Test");
        event.ajouterParticipant("Stacy");
        assertTrue(event.getParticipants().contains("Stacy"));
    }
}
```

7 Améliorations Possibles

— **Base de données** : Remplacer les fichiers JSON par une base SQLite/MySQL.
— **Notifications** : Implémenter un système d'e-mails (ex : avec JavaMail).
— **Internationalisation** : Support multilingue (properties files).

8 Conclusion

Ce projet a permis de maîtriser plusieurs concepts avancés de POO, notamment :
— La sérialisation/désérialisation JSON.
— La gestion d'interfaces graphiques complexes avec JavaFX.
— L'organisation d'un projet en couches (UI, données, mécanismes).
Les compétences acquises seront réutilisables dans de futurs projets, notamment la gestion d'erreurs et l'architecture modulaire.

A Annexes

A.1 Exemple de fichier JSON

```json
{
  "evenements": [
    {
      "type": "Concert",
      "titre": "Festival Jazz",
      "artistes": ["Artiste1", "Artiste2"]
    }
  ]
```

```
}
```