Système de Gestion d'Événements Distribué

Rapport de Projet Java Swing

PASSO DENNY

26 mai 2025

Table des matières

Chapitre 1

Introduction

Ce projet a pour objectif de mettre en œuvre un système de gestion d'événements (conférences, concerts, etc.) en Java, en appliquant les concepts avancés de la Programmation Orientée Objet (POO) : héritage, polymorphisme, design patterns (Observer, Singleton), gestion des exceptions personnalisées, collections génériques, sérialisation JSON, programmation événementielle et asynchrone.

Chapitre 2

Modélisation des Classes

2.1 Diagramme UML

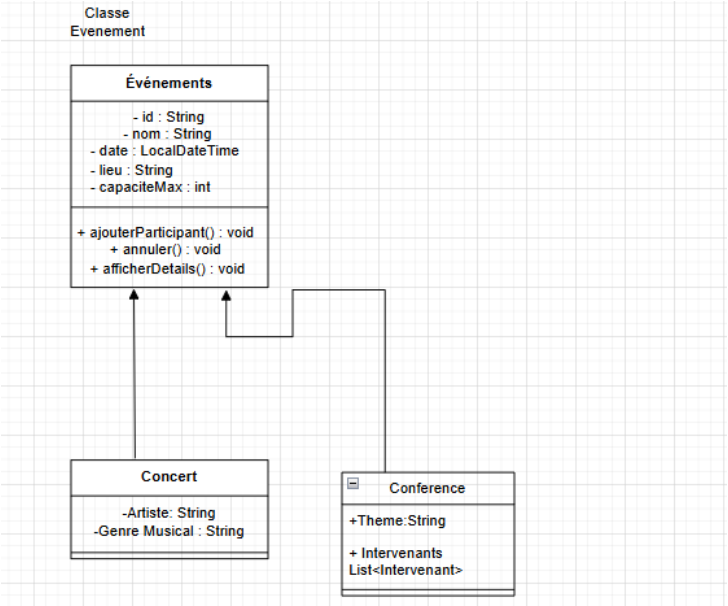

FIGURE 2.1 – Diagramme de classes principal : Evenement, Concert, Conference

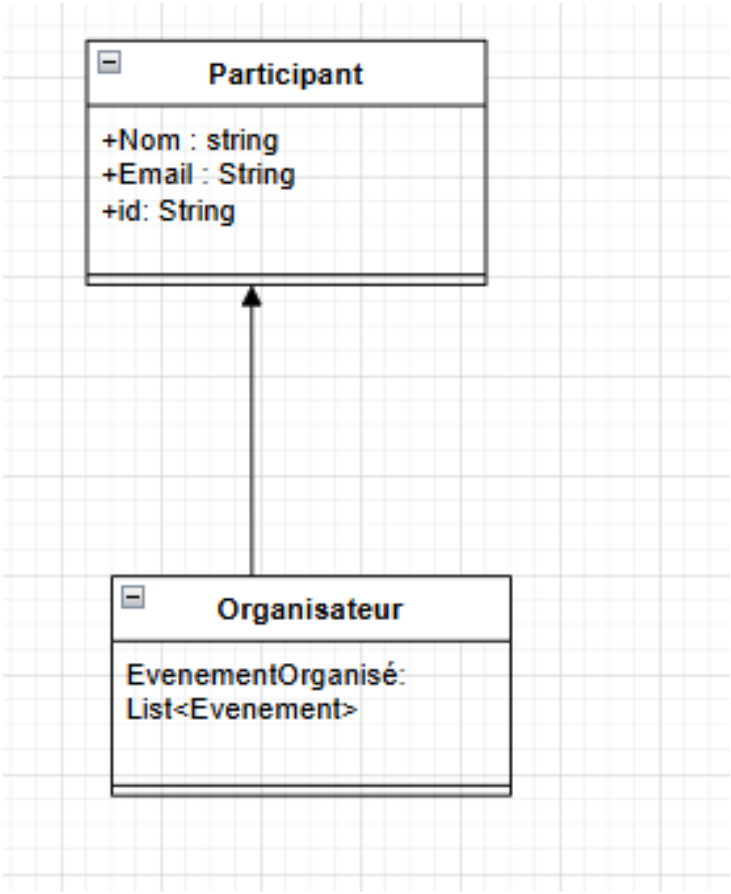

FIGURE 2.2 – Diagramme de classes principal : Participants et Organisateurs

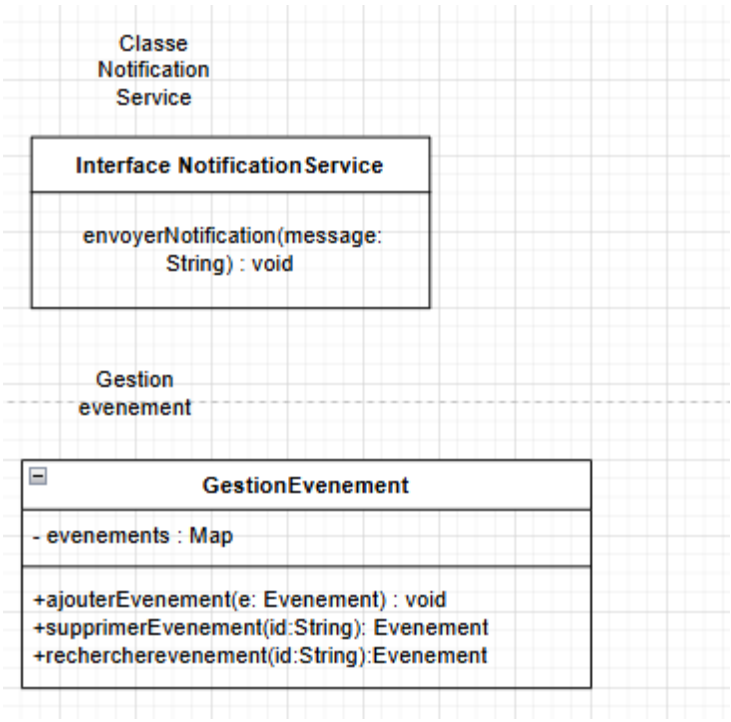

FIGURE 2.3 – Diagramme de classes principal : Participants et Organisateurs

2.2 Extraits de code – Modélisation

Classe abstraite Evenement

```java
public abstract class Evenement {
    protected String id;
    protected String nom;
    protected LocalDateTime date;
    protected String lieu;
    protected int capaciteMax;
    protected List<Participant> participants = new ArrayList<>();
    protected boolean annule = false;

    public abstract void afficherDetails();
    public abstract void annuler();
    public boolean ajouterParticipant(Participant p) throws
        CapaciteMaxAtteinteException, ParticipantDejaExistant {
        if (participants.size() >= capaciteMax) throw new
            CapaciteMaxAtteinteException();
        if (participants.stream().anyMatch(pa -> pa.getId().equals(p.
            getId()))) throw new ParticipantDejaExistant();
        return participants.add(p);
    }
}
```

Héritage : Conference et Concert

```java
package Tp_Poo2_final_v2.Models;

import java.time.LocalDateTime;
import java.util.List;

public  class Conference extends Evenement{
    List <Intervenant> intervenants;
    String theme;
    public Conference() {

    }

    public Conference(String id, String nom, LocalDateTime date, String
        lieu, int capaciteMax,List<Intervenant> intervenants,String theme
        ) {
        super(id, nom, date, lieu, capaciteMax);
        this.intervenants = intervenants;
        this.theme = theme;
    }
    @Override
    protected void afficherDetailsSpecifiques() {
        System.out.println(" Theme : " + theme);
        System.out.println(" Intervenants : ");
        for (Intervenant i : intervenants) {
            System.out.println(" - " + i.getNom());
        }
    }
    @Override
    protected String getTypeEvenement() {
```

```
28          return "Conf rence";
29      }
30      // Getters utiles (facultatif)
31      public List<Intervenant> getIntervenants() {
32          return intervenants;
33      }
34
35      public void setIntervenants(List<Intervenant> intervenants) {
36          this.intervenants = intervenants;
37      }
38
39      public void setTheme(String theme) {
40          this.theme = theme;
41      }
42
43      public String getTheme() {
44          return theme;
45      }
46
47  }
48
49
50  package Tp_Poo2_final_v2.Models;
51
52  import java.time.LocalDateTime;
53
54  public class Concert extends Evenement{
55      String artiste;
56      String genre;
57      public Concert() {
58      }
59      public Concert(String id, String nom, LocalDateTime date, String
            lieu, int capaciteMax, String artiste, String genre) {
60          super(id, nom, date, lieu, capaciteMax);
61          this.artiste = artiste;
62          this.genre = genre;
63      }
64      @Override
65      protected void afficherDetailsSpecifiques() {
66          System.out.println(" Artiste : " + artiste);
67          System.out.println(" Genre : " + genre);
68      }
69      @Override
70      protected String getTypeEvenement() {
71          return "Concert";
72      }
73      // Getters utiles (facultatif)
74      public String getArtiste() {
75          return artiste;
76      }
77
78      public void setArtiste(String artiste) {
79          this.artiste = artiste;
80      }
81
82      public String getGenre() {
83          return genre;
84      }
```

```
85
86      public void setGenre(String genre) {
87          this.genre = genre;
88      }
89  }
90
91  }
```

Participant et Organisateur

```
1   package Tp_Poo2_final_v2.Models;
2
3   import Tp_Poo2_final_v2.Observers.ParticipantObserver;
4
5   public class Participant implements ParticipantObserver {
6
7       private String id;
8       private String nom;
9       private String email;
10
11      public Participant(String id, String nom, String email) {
12          this.id = id;
13          this.nom = nom;
14          this.email = email;
15      }
16
17      public Participant() {
18          this.id = "";
19          this.nom = "";
20          this.email = "";
21      }
22
23      public String getId() {
24          return id;
25      }
26
27      public String getNom() {
28          return nom;
29      }
30
31      public String getEmail() {
32          return email;
33      }
34
35      public void setId(String id) {
36          this.id = id;
37      }
38
39      public void setNom(String nom) {
40          this.nom = nom;
41      }
42
43      public void setEmail(String email) {
44          this.email = email;
45      }
46
```

```
47      @Override
48      public void notifier(String message) {
49          System.out.println(" Notification pour " + this.nom + " : " +
                message);
50      }
51  }
52
53
54  package Tp_Poo2_final_v2.Models;
55
56  import java.util.ArrayList;
57  import java.util.List;
58
59  public class Organisateur extends Participant {
60
61          private List<Evenement> evenementsOrganises;
62
63          public Organisateur(String id, String nom, String email) {
64              super(id, nom, email);
65              this.evenementsOrganises = new ArrayList<>();
66          }
67
68          public void ajouterEvenementOrganise(Evenement evenement) {
69              this.evenementsOrganises.add(evenement);
70          }
71
72          public List<Evenement> getEvenementsOrganises() {
73              return evenementsOrganises;
74          }
75      }
76
77
78
79  }
```

Chapitre 3

Respect des principes SOLID

L'architecture du projet respecte les principes SOLID :
— **S** (Single Responsibility Principle) : Chaque classe a une responsabilité unique.
— **O** (Open/Closed Principle) : Les classes sont ouvertes à l'extension mais fermées à la modification.
— **L** (Liskov Substitution Principle) : Les sous-classes peuvent remplacer leurs super-classes sans altérer la logique.
— **I** (Interface Segregation Principle) : Les interfaces sont spécifiques, évitant des méthodes inutiles.
— **D** (Dependency Inversion Principle) : Les classes dépendent d'abstractions et non de classes concrètes.

Chapitre 4

Justification des choix d'implémentation

Modélisation orientée objet

L'héritage permet de factoriser les comportements communs tout en permettant l'extension facile à d'autres types d'événements. Les collections génériques garantissent la flexibilité et la robustesse.

Design Patterns

- **Singleton** pour `GestionEvenement` : gestion centralisée des événements.
- **Observer** pour la notification : notification automatique des participants lors de l'annulation d'un événement.
- **Factory** (si utilisé) : création d'objets selon le type d'événement choisi dynamiquement.
- **Strategy** (si utilisé) : changement dynamique de la stratégie de notification ou de recherche.

Gestion des exceptions personnalisées

Les exceptions spécifiques permettent une gestion fine des erreurs et une meilleure expérience utilisateur.

Sérialisation JSON

Jackson garantit la portabilité, la simplicité et la rapidité de lecture/écriture.

Interface graphique Swing

Swing, enrichi par FlatLaf et des effets graphiques modernes, permet une expérience utilisateur agréable et multiplateforme.

Programmation événementielle et asynchrone

Swing pour la gestion des événements utilisateur, notifications asynchrones via `CompletableFuture`.

Chapitre 5

Design Patterns

5.1 Singleton – GestionEvenement

```java
package Tp_Poo2_final_v2.Services;

import Tp_Poo2_final_v2.Exceptions.ParticipantDejaExistant;
import Tp_Poo2_final_v2.Models.Concert;
import Tp_Poo2_final_v2.Models.Evenement;

import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class GestionEvenement {

    //  Singleton
    private static GestionEvenement instance;

    //  Donn es internes
    private Map<String, Evenement> evenements;

    //  Constructeur priv
    private GestionEvenement() {
        this.evenements = new HashMap<>();
    }

    //  Point  d  accs  global
    public static GestionEvenement getInstance() {
        if (instance == null) {
            instance = new GestionEvenement();
        }
        return instance;
    }

    //  M thodes de gestion
    public Map<String, Evenement> getEvenements() {
        return Collections.unmodifiableMap(evenements); // lecture seule
    }

    public void chargerDepuisFichier(Map<String, Evenement>
       nouvellesDonnees) {
        evenements.clear();
```

```
40          evenements.putAll(nouvellesDonnees);
41      }
42
43      public Evenement getEvenement(String id) {
44          return evenements.get(id);
45      }
46
47      public void ajouterEvenement(Evenement evenement) throws Exception {
48          if(evenements.containsKey(evenement.getId())){
49              throw new ParticipantDejaExistant("L'ID de l'  vnement  est
                    d  j  utilis .");
50          }
51          evenements.put(evenement.getId(), evenement);
52          System.out.println("  vnement  ajout  : " + evenement.getNom
                ());
53      }
54
55      public void supprimerEvenement(String id) {
56          evenements.remove(id);
57      }
58
59      public void modifierEvenement(Evenement evenement) {
60          evenements.replace(evenement.getId(), evenement);
61      }
62
63      public void rechercherEvenementParNom(String nom) {
64          boolean trouve = false;
65          for (Evenement e : evenements.values()) {
66              if (e.getNom().equals(nom)) {
67                  System.out.println("Evenement trouv  :\n");
68                  e.afficherDetails();
69                  trouve = true;
70
71              }
72
73          }
74          if(!trouve){
75              System.out.println(" Aucun  vnement  trouv  avec le nom :
                    " + nom);
76          }
77      }
78
79      public void rechercherEvenementParId(String id) {
80          Evenement e = evenements.get(id);
81          if (e != null) {
82              System.out.println("Evenement trouv  :\n");
83              e.afficherDetails();
84          } else {
85              System.out.println(" Aucun  vnement  trouv  avec l'ID : "
                    + id);
86          }
87      }
88      public void afficherTousLesEvenements() {
89          if (evenements.isEmpty()) {
90              System.out.println(" Aucun  vnement  enregistr .");
91              return;
92          }
93          System.out.println(" Liste des  vnements  :");
```

```
94          for (Evenement e : evenements.values()) {
95              System.out.println("      ----------------------------");
96              e.afficherDetails();
97          }
98      }
99
100     public void afficherConcertsParis() {
101         System.out.println("Concerts    Paris :");
102         evenements.values().stream()
103             .filter(e -> e instanceof Concert && e.getLieu().
                    equalsIgnoreCase("Paris"))
104             .forEach(Evenement::afficherDetails);
105     }
106
107     public void rechercherEvenementsParTypeEtVille(Class<? extends
            Evenement> type, String ville) {
108         List<Evenement> resultats = evenements.values().stream()
109             .filter(e -> type.isInstance(e) && e.getLieu().
                    equalsIgnoreCase(ville))
110             .toList();
111
112         if (resultats.isEmpty()) {
113             System.out.println("Aucun    vnement    de type " + type.
                    getSimpleName() + " trouv      " + ville + ".");
114         } else {
115             System.out.println("   vnements    trouv s :");
116             resultats.forEach(Evenement::afficherDetails);
117         }
118     }
119 }
```

5.2 Observer – Notification des participants

```
1  public interface EvenementObservable {
2      void ajouterObserver(ParticipantObserver obs);
3      void notifierObservers(String message);
4  }
5
6  public interface ParticipantObserver {
7      void recevoirNotification(String message);
8  }
```

Chapitre 6

Gestion des Exceptions

```
1  public class CapaciteMaxAtteinteException extends Exception {
2      public CapaciteMaxAtteinteException() { super("Capacit  maximale
           atteinte."); }
3  }
4
5  public class EvenementDejaExistantException extends Exception {
6      public EvenementDejaExistantException() { super("  vnement   d  j
           existant."); }
7  }
8
9  public class ParticipantDejaExistant extends Exception {
10     public ParticipantDejaExistant() { super("Participant  d  j  inscrit
           ."); }
11 }
```

Chapitre 7

Sérialisation/Désérialisation JSON

```java
public class SerialisationJson {
    public static void sauvegarderEvenements(List<Evenement> evenements,
        String fichier) throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        mapper.writerWithDefaultPrettyPrinter().writeValue(new File(
            fichier), evenements);
    }
    public static List<Evenement> chargerEvenements(String fichier)
        throws IOException {
        ObjectMapper mapper = new ObjectMapper();
        return Arrays.asList(mapper.readValue(new File(fichier),
            Evenement[].class));
    }
}
```

Chapitre 8

Interface Graphique (Swing)

8.1 Presentation du logiciel :

Il s'agit juste de la presentation du logiciel lorsque ca s'ouvre ;

8.1.1 Execution

FIGURE 8.1 – Presentation Logiciel

8.2 Ajout d'un participant

ici une fois apres avoir cliqué sur le bouton ajouter participant ca nous renvoie vers une boite de dialogue qui nous demande de remplir des informations et ainsi de valider l'ajout dans un evenement non annulé et existante

8.2.1 exécution

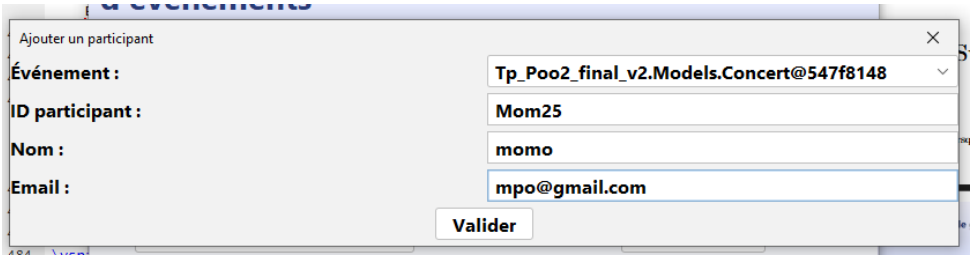

FIGURE 8.2 – Ajout Participant

8.3 Ajouter L'evenement

ici nous ajoutons l'evenement qui sera sauvegarder dans le fichier evenements.json

8.4 Modifier l'evenement

ici nous modifions les informations l'evenement qui sera sauvegarder dans le fichier evenements.json

8.5 Supprimer L'evenement

ici nous supprimons l'evenement en utilisant certaines fonctions propres à la classe Collection

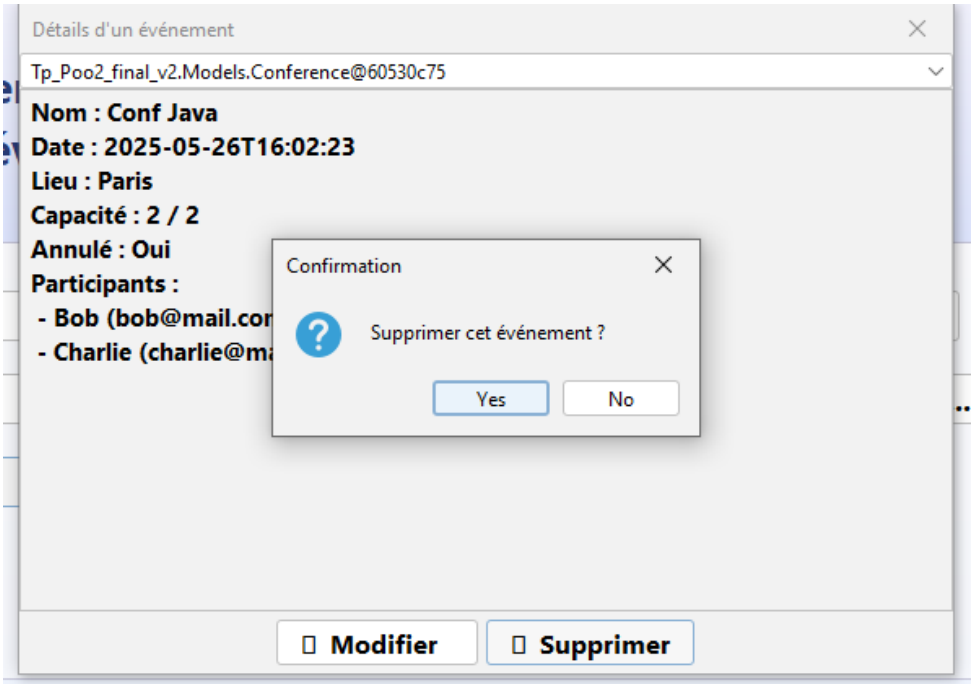

FIGURE 8.5 – suppression

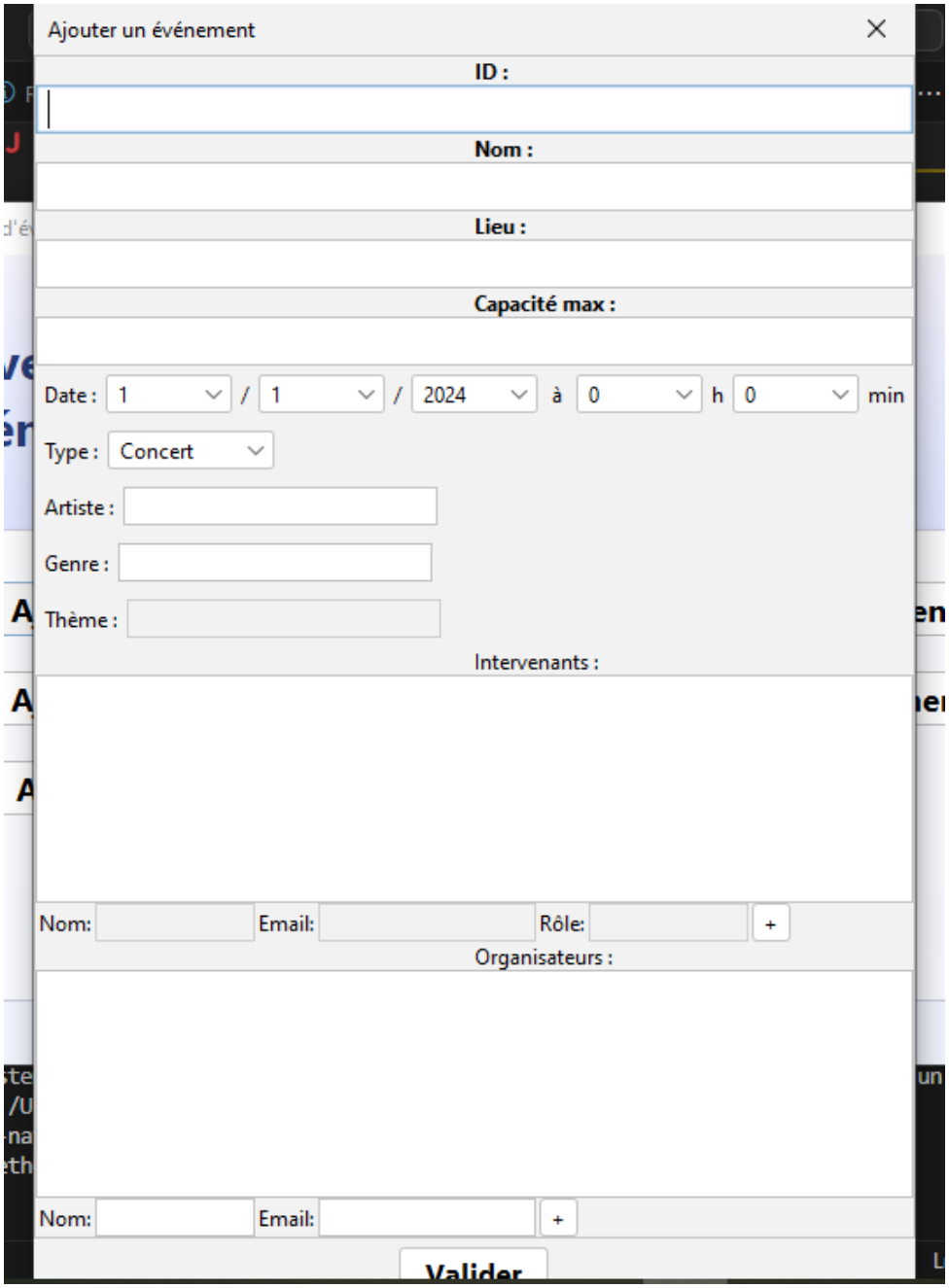

FIGURE 8.3 – Ajout Evenement

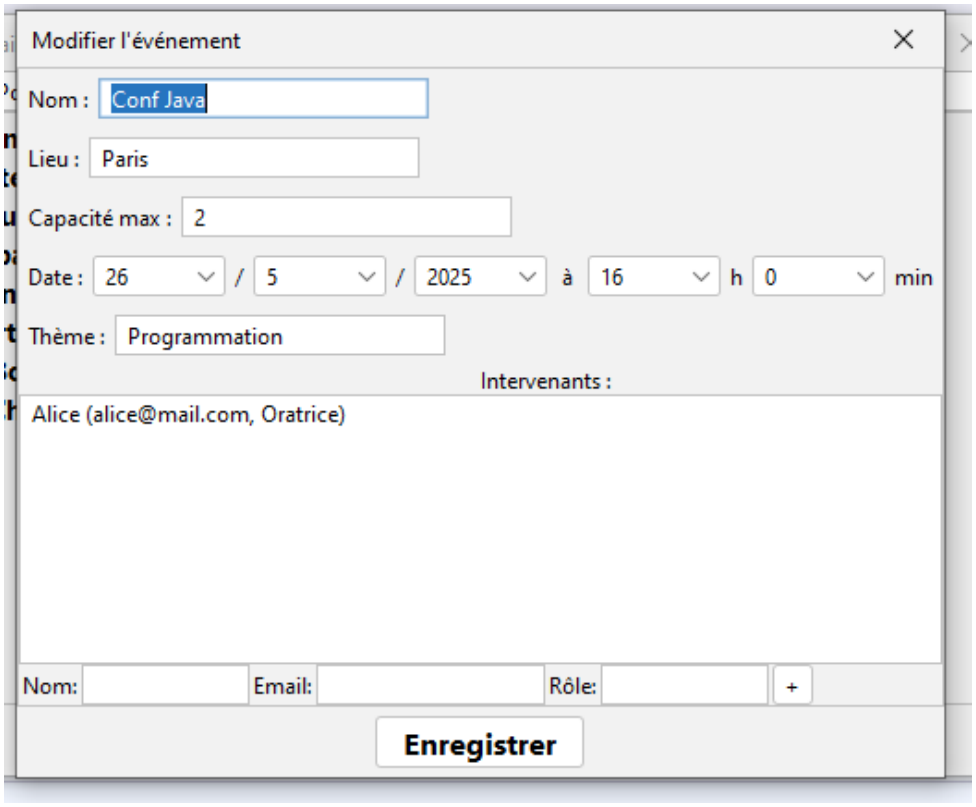

FIGURE 8.4 – Modification

Chapitre 9

Programmation Asynchrone (Bonus)

La programmation asynchrone est un style de programmation où certaines opérations (comme l'accès à un fichier, une base de données, ou le réseau) s'exécutent en arrière-plan, sans bloquer l'exécution du reste du programme. Cela permet à l'application de rester réactive (par exemple, l'interface graphique ne se fige pas pendant un chargement).

En Java, on utilise par exemple :

Threads ou ExecutorService CompletableFuture (API moderne pour gérer des tâches asynchrones) Les événements Swing (gestion asynchrone des actions utilisateur)

9.1 Notification différée

```
1 CompletableFuture.runAsync(() -> {
2     try {
3         Thread.sleep(2000);
4         participant.recevoirNotification("Votre    vnement    a    t
              annul   !");
5     } catch (InterruptedException ignored) {}
6 });
```

Chapitre 10

Recherche et Streams

10.1 Exemple de recherche avancée

```
List<Evenement> resultats = evenements.stream()
    .filter(ev -> ev.getNom().toLowerCase().contains(query) ||
                  ev.getLieu().toLowerCase().contains(query))
    .filter(ev -> type.equals("Tous") || ev.getType().equals(type))
    .collect(Collectors.toList());
```

Chapitre 11

Tests et Validation

11.1 Exemple de test JUnit

```java
@Test
public void testAjoutParticipant() throws Exception {
    Evenement conf = new Conference(...);
    Participant p = new Participant("1", "Alice", "alice@mail.com");
    conf.ajouterParticipant(p);
    assertTrue(conf.getParticipants().contains(p));
}
```

Chapitre 12

Difficulté Rencontre

J'ai ete en face d'un probleme majeur lors de la realisation de mon projet qui etait la serialisation et la deserialisation dans un fichier json et l'installation des dependances jackson et junit, c'est l'etape qui m'a donné beaucoup de fil à retordre. ainsi que le polymorphisme pour les classes abstraites

Chapitre 13

Solution Envisagées

nous prevoyons pour cette application à la rendre interactives et plus bien designé. Egalement nous projettons l'ajout de certaines fonctionnalités comme les envois par sms ou par email.

Chapitre 14

Conclusion

Ce projet nous a permis de mettre en pratique l'ensemble des concepts avancés de la POO en Java, de la conception UML à l'implémentation, en passant par la gestion des exceptions, la persistance, les design patterns et la programmation événementielle. L'interface graphique moderne et la robustesse du code rendent l'application agréable à utiliser et fiable.

Chapitre 15

Annexes

Structure du projet

Voir le fichier `README.md`.

Exemples de tests

Voir le dossier `src/test/java/`.

Exemples de fichiers JSON

Voir `evenements.json`.