# Nom : Seck

# Prenoms : Cheick Omar Nouredine

# Nom : KABORE

# PRENOM :BASNKOWINDE ALIX LARISSA

# UVBF/Génie Logiciel pure developer L2 2024


# Rapport de projet : Gestion de réservation de billets de concert

## 1. Introduction

Dans ce projet, nous avons développé une application Java pour la gestion des réservations de billets de concerts. Le but principal était de permettre aux utilisateurs de consulter des événements, de réserver des billets, et d'offrir une interface pour la gestion des événements par un administrateur.
L'application utilise les principes de la programmation orientée objet (POO) pour structurer les entités, les utilisateurs et les processus de réservation. En plus de la programmation en Java, nous avons utilisé **Git** pour la gestion de version et la collaboration, ainsi que **GitHub** pour le dépôt de code. Ce rapport décrit en détail les étapes du développement, les méthodes et classes utilisées, ainsi que le processus d'implémentation du projet.

---

## 2. Processus de développement

### 2.1 Conception du projet et structure

Nous avons commencé par une analyse du cahier des charges, définissant les différentes fonctionnalités attendues de l'application. Le premier objectif était de concevoir une structure de classes qui modéliserait correctement les différents éléments : les événements, les utilisateurs et le système de réservation.

La structure principale du projet se décompose en plusieurs classes :

- **Evenement** : Représente un concert ou un événement musical.
- **Utilisateur** : Représente un utilisateur qui peut effectuer des réservations.
- **Reservation** : Représente une réservation effectuée par un utilisateur pour un événement.

- **GestionEvenements** : Gère la création, la suppression et l'affichage des événements.
- **GestionReservations** : Gère les réservations des utilisateurs.

Chaque classe est responsable de la gestion d’un aspect spécifique du projet, facilitant ainsi l’extension et la maintenance du code.

---

## 2.2 Implémentation des classes et explication du code

### Classe `Evenement`

La classe `Evenement` est l'une des classes principales et est utilisée pour représenter un concert ou un spectacle.

- **Attributs** :
  - `String nom` : Le nom de l'événement (par exemple, "Concert de Coldplay").
  - `String date` : La date de l'événement (par exemple, "25 décembre 2024").
  - `String lieu` : Lieu de l'événement (par exemple, "Stade du 4AOUT").
  - `int nbBilletsDisponibles` : Le nombre de billets disponibles pour cet événement.

L'objectif principal de cette classe est de stocker et de gérer les informations relatives à un concert. Chaque instance de la classe `Evenement` représente un concert spécifique.

Voici un extrait de code pour cette classe :

```
public class Evenement {
    private String nom;
    private String date;
    private String lieu;
    private int nbBilletsDisponibles;

    public Evenement(String nom, String date, String lieu, int
nbBilletsDisponibles) {
        this.nom = nom;
        this.date = date;
        this.lieu = lieu;
        this.nbBilletsDisponibles = nbBilletsDisponibles;
    }

    public String getNom() {
        return nom;
    }

    public String getDate() {
        return date;
    }

    public String getLieu() {
        return lieu;
    }

    public int getNbBilletsDisponibles() {
        return nbBilletsDisponibles;
    }

    public void setNbBilletsDisponibles(int nbBilletsDisponibles) {
        this.nbBilletsDisponibles = nbBilletsDisponibles;
```

```
    }
}
```

- **Constructeur** : Le constructeur initialise chaque événement avec son nom, sa date, son lieu et le nombre de billets disponibles.
- **Méthodes getter** : Ces méthodes permettent d'accéder aux informations sur l'événement, comme son nom, la date, ou encore le lieu.
- **Méthode setter** : La méthode `setNbBilletsDisponibles` permet de mettre à jour le nombre de billets restants après une réservation.

### Classe `Utilisateur`

Cette classe représente un utilisateur de l'application, et elle contient des informations telles que son nom ainsi que les réservations qu'il a effectuées.

- **Attributs** :
  - `String nom` : Le nom de l'utilisateur.
  - `ArrayList<Reservation> reservations` : Une liste des réservations effectuées par l'utilisateur.

Voici un exemple du code :

```
public class Utilisateur {
    private String nom;
    private ArrayList<Reservation> reservations;

    public Utilisateur(String nom) {
        this.nom = nom;
        this.reservations = new ArrayList<>();
    }

    public String getNom() {
        return nom;
    }

    public void ajouterReservation(Reservation reservation) {
        reservations.add(reservation);
    }

    public void afficherReservations() {
        for (Reservation reservation : reservations) {
            System.out.println("Événement : " +
reservation.getEvenement().getNom() +
                               ", Nombre de billets : " +
reservation.getNbBillets());
        }
    }
}
```

- **Constructeur** : L'utilisateur est initialisé avec un nom et une liste vide de réservations.
- **ajouterReservation()** : Cette méthode permet d'ajouter une nouvelle réservation à la liste de l'utilisateur.
- **afficherReservations()** : Cette méthode affiche toutes les réservations effectuées par l'utilisateur, en précisant l'événement et le nombre de billets réservés.

### Classe `Reservation`

La classe `Reservation` modélise la réservation qu'un utilisateur effectue pour un événement donné.

- **Attributs** :
  - `Utilisateur utilisateur` : L'utilisateur qui a effectué la réservation.
  - `Evenement evenement` : L'événement pour lequel la réservation a été faite.
  - `int nbBillets` : Le nombre de billets réservés.

```
public class Reservation {
    private Utilisateur utilisateur;
    private Evenement evenement;
    private int nbBillets;

    public Reservation(Utilisateur utilisateur, Evenement evenement, int
nbBillets) {
        this.utilisateur = utilisateur;
        this.evenement = evenement;
        this.nbBillets = nbBillets;
    }

    public Evenement getEvenement() {
        return evenement;
    }

    public int getNbBillets() {
        return nbBillets;
    }

    public void afficherDetails() {
        System.out.println("Utilisateur : " + utilisateur.getNom() +
                           ", Événement : " + evenement.getNom() +
                           ", Billets : " + nbBillets);
    }
}
```

- **Constructeur** : La réservation est liée à un utilisateur, un événement, et un certain nombre de billets.
- **afficherDetails()** : Cette méthode permet d’afficher les détails de la réservation (nom de l’utilisateur, événement, nombre de billets).

---

## 2.3 Gestion des événements et des réservations

Nous avons ensuite créé deux classes supplémentaires pour gérer les événements et les réservations de manière plus globale.

### Classe `GestionEvenements`

Cette classe permet à l’administrateur de gérer la liste des événements disponibles (ajout, suppression, affichage).

```
public class GestionEvenements {
    private ArrayList<Evenement> evenements = new ArrayList<>();

    public void ajouterEvenement(Evenement e) {
        evenements.add(e);
```

```
        System.out.println("Événement ajouté : " + e.getNom());
    }

    public void supprimerEvenement(Evenement e) {
        evenements.remove(e);
        System.out.println("Événement supprimé : " + e.getNom());
    }

    public void afficherEvenements() {
        for (Evenement e : evenements) {
            System.out.println(e.getNom() + " - " + e.getDate() + " - " +
e.getLieu());
        }
    }
}
```

- **ajouterEvenement()** : Cette méthode permet d'ajouter un nouvel événement à la liste des événements.
- **supprimerEvenement()** : Permet de retirer un événement spécifique de la liste.
- **afficherEvenements()** : Affiche tous les événements avec leur nom, date et lieu.

### Classe `GestionReservations`

Elle gère les réservations, notamment la création de nouvelles réservations et la vérification de la disponibilité des billets.

```
public class GestionReservations {
    public void reserverBillet(Utilisateur utilisateur, Evenement evenement, int
nbBillets) {
        if (evenement.getNbBilletsDisponibles() >= nbBillets) {
            Reservation reservation = new Reservation(utilisateur, evenement,
nbBillets);
            utilisateur.ajouterReservation(reservation);

evenement.setNbBilletsDisponibles(evenement.getNbBilletsDisponibles() -
nbBillets);
            System.out.println("Réservation réussie pour " +
utilisateur.getNom());
        } else {
            System.out.println("Pas assez de billets disponibles.");
        }
    }
}
```

- **reserverBillet()** : Cette méthode permet de réserver des billets pour un utilisateur. Elle vérifie si le nombre de billets demandés est disponible avant de créer la réservation et de mettre à jour les billets restants.

---

## 2.4 Interface utilisateur et gestion des erreurs

Pour faciliter l'interaction avec l'utilisateur, nous avons implémenté une interface graphique basique utilisant **Swing**. Les utilisateurs peuvent ainsi voir une liste des événements et réserver des billets en quelques clics.

Nous avons également mis en place des gestionnaires d'erreurs pour prévenir des problèmes tels que la réservation d'un nombre de billets supérieur à ce qui est disponible.

# 3. Conclusion

Ce projet a permis de comprendre les principes fondamentaux de la POO en Java, tout en développant une application fonctionnelle de gestion de réservation de billets de concerts. L'application peut être étendue à d'autres types d'événements et améliorée avec une gestion plus complexe des utilisateurs et des paiements en ligne.

Nous avons utilisé des méthodes de gestion de version avec **Git** et **GitHub**, ce qui nous a aidés à suivre les changements de code et à collaborer efficacement. Le projet pourrait être étendu à l'avenir avec plus de fonctionnalités, comme la possibilité de personnaliser l'interface ou d’intégrer une base de données pour une gestion plus robuste des événements et des utilisateurs.