Université [Nom de l'Université]

Département d'Informatique

Conservatoire Virtuel

Système de Gestion d'École de Musique

Projet de Programmation Orientée Objet

Réalisé par :

Hassen Ben Amor
Dalil Adimi

Encadré par :

Prof. [Nom du Superviseur]

Année Universitaire :
2024 – 2025

9 décembre 2025

Résumé

Le projet **Conservatoire Virtuel** est un système complet de gestion d'école de musique développé en Java. Cette application répond aux besoins opérationnels d'une école de musique privée, offrant des fonctionnalités pour la gestion des étudiants, des professeurs, des forfaits de cours, de la planification des leçons, des paiements et des examens officiels.

Le système a été conçu avec un fort accent sur les principes de programmation orientée objet, comprenant des classes abstraites, des interfaces, le polymorphisme et des constructeurs de copie. L'architecture suit les meilleures pratiques en conception logicielle, incluant la séparation des préoccupations, l'encapsulation et l'utilisation appropriée des modèles de conception.

Les fonctionnalités clés incluent un système de planification robuste avec prévention des conflits, un système de facturation flexible supportant différents types de services, et un module de gestion des examens avec contrôle de capacité et suivi des résultats.

Mots-clés : Java, POO, École de Musique, Système de Gestion, Planification, Facturation, Classes Abstraites, Interfaces, Polymorphisme

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte du Projet

Une école de musique privée appelée **Conservatoire Virtuel** nécessite un système d'information pour supporter ses activités internes. L'objectif est de concevoir et implémenter une application qui gère tous les aspects des opérations de l'école de manière efficace et professionnelle.

1.2 Objectifs du Projet

Les principaux objectifs de ce projet sont :

1. Concevoir un modèle de domaine complet pour une école de musique
2. Implémenter des concepts avancés de programmation orientée objet
3. Développer une application console fonctionnelle
4. Démontrer les bonnes pratiques d'ingénierie logicielle
5. Créer une documentation professionnelle incluant des diagrammes UML

1.3 Périmètre

Le système gère les aspects suivants :

— **Gestion des Personnes :** Étudiants et professeurs avec leurs attributs et relations
— **Gestion des Services :** Forfaits de cours, leçons individuelles et locations d'instruments
— **Planification :** Planification des leçons avec prévention des conflits et réservation de ressources
— **Gestion Financière :** Paiements, facturation et comptabilité
— **Gestion des Examens :** Examens officiels, inscription et résultats

1.4 Structure du Document

Ce rapport est organisé comme suit :

— **Chapitre 2 :** Analyse des Besoins
— **Chapitre 3 :** Conception du Système avec UML
— **Chapitre 4 :** Détails d'Implémentation
— **Chapitre 5 :** Discussion des Concepts POO
— **Chapitre 6 :** Tests et Résultats
— **Chapitre 7 :** Conclusion

Chapitre 2

Analyse des Besoins

2.1 Besoins Fonctionnels

2.1.1 Gestion des Étudiants

Un étudiant est identifié par un ID unique avec les attributs suivants :

— Nom, prénom
— Adresse, date de naissance, téléphone, email
— Niveau (Débutant/Intermédiaire/Avancé)
— Instruments préférés

Les étudiants peuvent s'inscrire à :

— Des Forfaits de Cours (N leçons avec dates de validité)
— Des leçons individuelles (facturées par leçon)

Le système suit :

— Les heures achetées
— Les heures restantes
— L'historique d'utilisation

2.1.2 Gestion des Professeurs

Chaque professeur possède :

— ID, nom, qualifications
— Spécialisations (instruments qu'ils peuvent enseigner)
— Tarif horaire
— Planning de disponibilité

2.1.3 Forfaits et Services

L'école propose :

— Forfaits musique (heures fixes)
— Forfaits illimités

— Leçons de groupe ou individuelles
— Leçons payantes uniques
— Location d'instruments
— Réservation de salles

2.1.4 Système de Planification

Le système permet de planifier :

— Leçons (professeur + étudiant + salle + instrument)
— Locations de salles
— Sessions d'examen

Pour chaque activité planifiée :

— Date et heure de début
— Durée
— Ressources assignées
— Statut (planifié/terminé/annulé)

Important : Les conflits de planification doivent être évités.

2.1.5 Gestion des Examens

L'école organise des examens officiels :

— Les examens ont un nom, un instrument, une date et une capacité
— Les étudiants peuvent s'inscrire
— Les résultats sont enregistrés : réussi/échoué et score optionnel

2.1.6 Paiements et Facturation

Le système suit :

— Les paiements
— Les soldes impayés
— Les prix des forfaits, locations et leçons

2.2 Règles Métier

1. Les heures de forfait expirent à la date de fin du forfait
2. Si une leçon est annulée moins de 24 heures avant, elle est comptée comme consommée
3. L'inscription aux examens est fermée lorsque la capacité maximale est atteinte
4. Une leçon de groupe consomme une heure de chaque participant

Chapitre 3

Conception du Système

3.1 Vue d'Ensemble de l'Architecture

Le système suit une architecture en couches :

1. **Couche Présentation :** Interface utilisateur console (`ConservatoireApp`)
2. **Couche Service :** Logique métier (`SchedulingService`, `PaymentService`, `ExamService`)
3. **Couche Repository :** Stockage des données (`DataRepository`)
4. **Couche Modèle :** Entités du domaine (Person, Service, ScheduledActivity, etc.)

3.2 Structure des Packages

```
com.music.school/
 model/
    person/       (Student, Teacher)
    service/      (CoursePackage, IndividualLesson, InstrumentRental)
    scheduling/   (Lesson, RoomBooking)
    resource/     (Room, Instrument)
    exam/         (Exam, ExamRegistration)
    billing/      (Invoice, Payment)
 interfaces/       (Schedulable, Billable, Bookable)
 enums/            (Level, ActivityStatus, PaymentStatus, etc.)
 service/          (Services métier)
 repository/       (Stockage de données)
 data/             (Initialisation des données de test)
```

FIGURE 3.1 – Structure des Packages du Projet

3.3 Diagramme de Classes

3.3.1 Hiérarchie Person

TABLE 3.1 – Hiérarchie de la Classe Person

Classe	Type	Parent	Attributs Clés
Person	Abstraite	-	id, firstName, lastName, email, dateOfBirth
Student	Concrète	Person	level, preferredInstruments, packageHours
Teacher	Concrète	Person	hourlyRate, specializations, availability

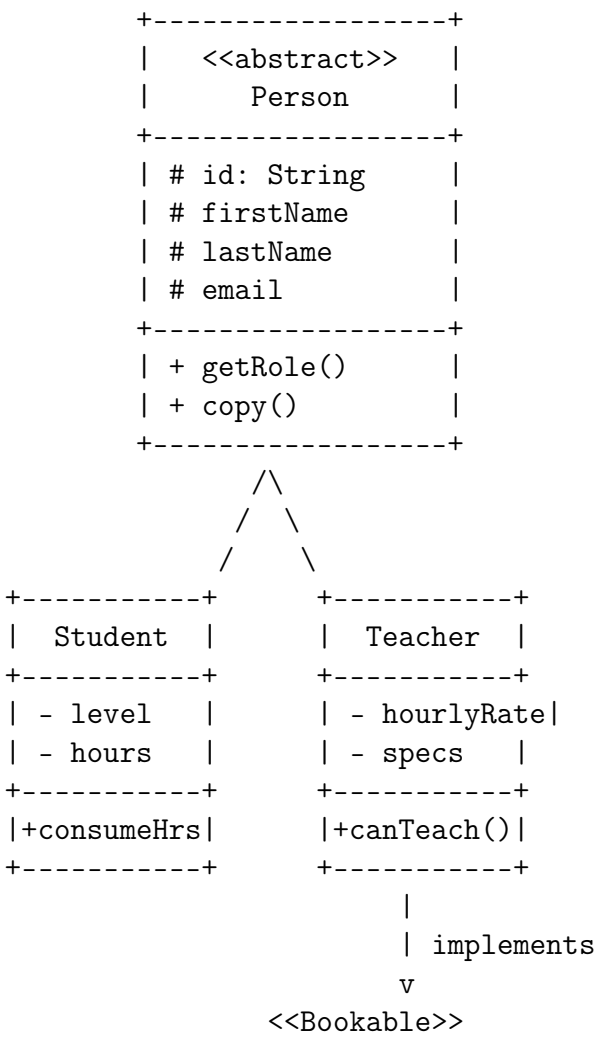

FIGURE 3.2 – Diagramme de Classes - Hiérarchie Person

3.3.2 Hiérarchie Service

TABLE 3.2 – Hiérarchie de la Classe Service

Classe	Type	Interface	Attributs Clés
Service	Abstraite	Billable	id, name, price, studentId, paid
CoursePackage	Concrète	Billable	totalHours, usedHours, instrument
IndividualLesson	Concrète	Billable	instrument, durationMinutes
InstrumentRental	Concrète	Billable	dailyRate, depositAmount

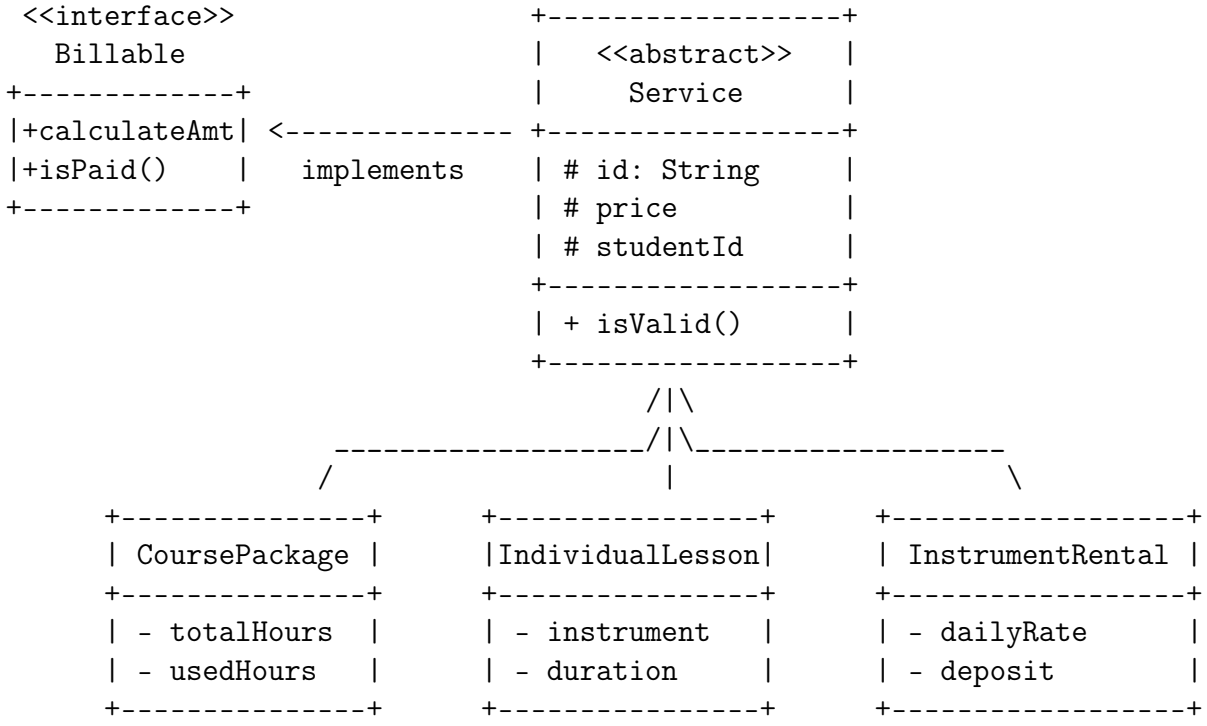

FIGURE 3.3 – Diagramme de Classes - Hiérarchie Service

3.3.3 Hiérarchie ScheduledActivity

TABLE 3.3 – Hiérarchie de la Classe ScheduledActivity

Classe	Type	Interface	Attributs Clés
ScheduledActivity	Abstraite	Schedulable	id, dateTime, duration, status, roomId
Lesson	Concrète	Schedulable	teacherId, studentIds, instrument
RoomBooking	Concrète	Schedulable	studentId, hourlyRate, purpose

3.3.4 Résumé Complet des Classes

TABLE 3.4 – Résumé de Toutes les Classes

Classe	Type	Parent	Interfaces
Person	Abstraite	-	-
Student	Concrète	Person	-
Teacher	Concrète	Person	Bookable
Service	Abstraite	-	Billable
CoursePackage	Concrète	Service	Billable
IndividualLesson	Concrète	Service	Billable
InstrumentRental	Concrète	Service	Billable
ScheduledActivity	Abstraite	-	Schedulable
Lesson	Concrète	ScheduledActivity	Schedulable
RoomBooking	Concrète	ScheduledActivity	Schedulable
Room	Concrète	-	Bookable
Instrument	Concrète	-	Bookable
Exam	Concrète	-	-
Invoice	Concrète	-	-
Payment	Concrète	-	-

3.4 Diagramme de Séquence : Planification d'une Leçon

Le processus de planification d'une leçon suit ces étapes :

TABLE 3.5 – Diagramme de Séquence - Étapes de Planification

Étape	De	Vers	Message
1	Utilisateur	SchedulingService	scheduleLesson()
2	SchedulingService	DataRepository	getTeacher(id)
3	SchedulingService	Teacher	canTeach(instrument)
4	SchedulingService	DataRepository	checkConflicts()
5	SchedulingService	Teacher	isAvailableAt()
6	SchedulingService	Room	isAvailableAt()
7	SchedulingService	Teacher	addBooking()
8	SchedulingService	Room	addBooking()
9	SchedulingService	DataRepository	addScheduledActivity()
10	SchedulingService	Utilisateur	return Lesson

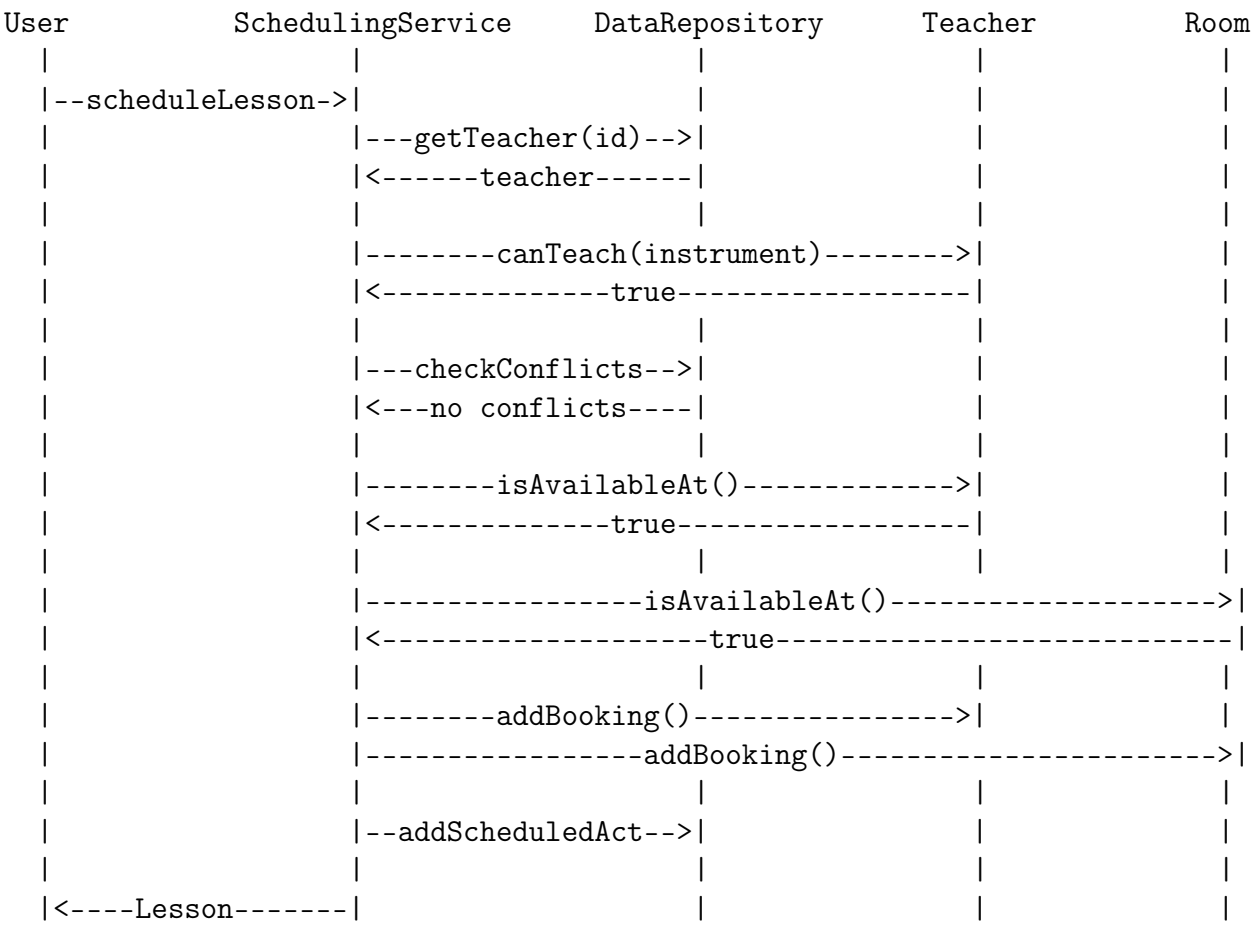

FIGURE 3.4 – Diagramme de Séquence - Planification d'une Leçon

3.5 Diagramme d'Activité : Inscription à un Examen

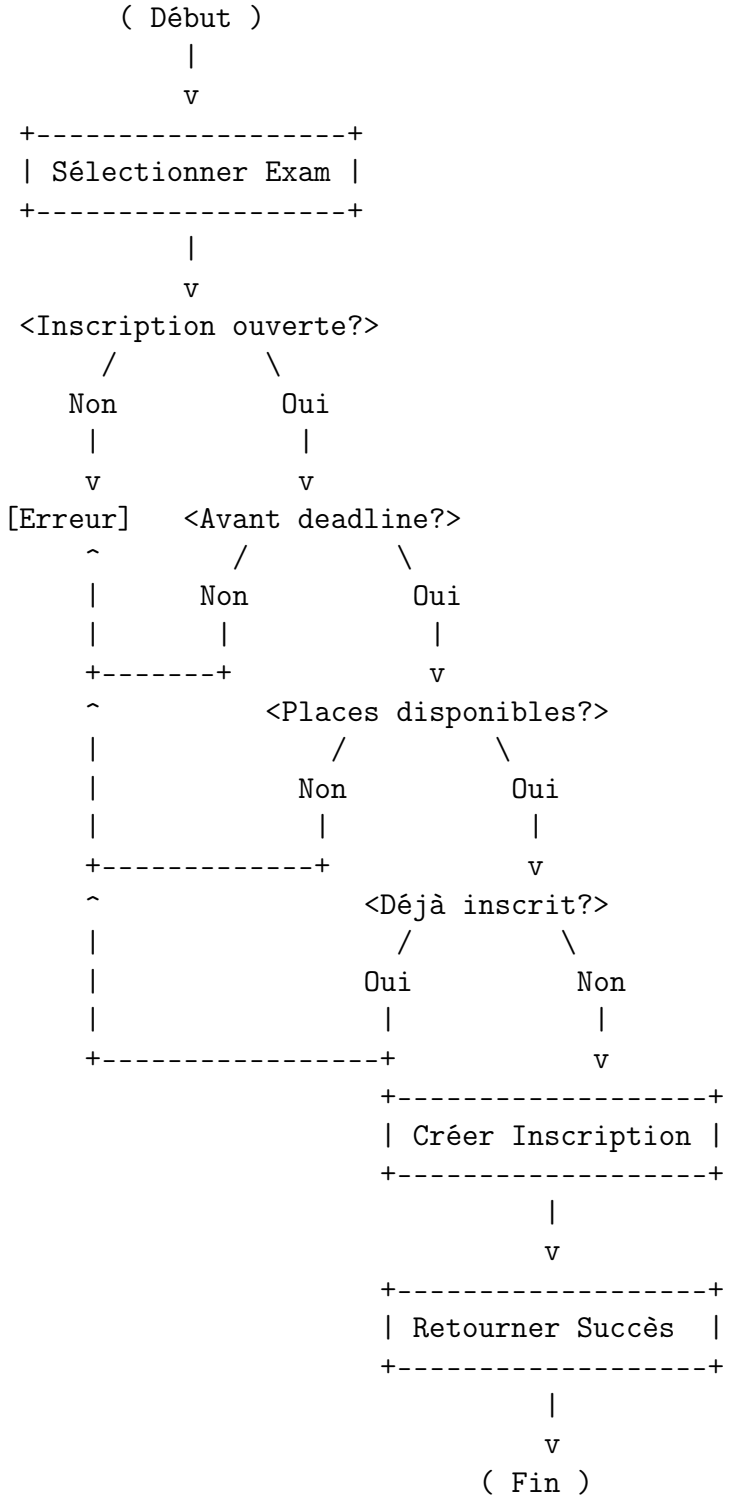

FIGURE 3.5 – Diagramme d'Activité - Processus d'Inscription à un Examen

Chapitre 4

Détails d'Implémentation

4.1 Technologies Utilisées

— **Langage :** Java 17
— **Outil de Build :** Maven 3.x
— **IDE :** IntelliJ IDEA / VS Code

4.2 Détails Clés d'Implémentation

4.2.1 Classe Abstraite Person

```java
public abstract class Person {
    protected String id;
    protected String firstName;
    protected String lastName;
    protected String email;
    protected LocalDate dateOfBirth;

    // Constructeur de copie
    protected Person(Person other) {
        this.id = other.id;
        this.firstName = other.firstName;
        this.lastName = other.lastName;
        this.email = other.email;
        this.dateOfBirth = other.dateOfBirth;
    }

    // Methodes abstraites
    protected abstract String getIdPrefix();
    public abstract String getRole();
    public abstract Person copy();

    public String getFullName() {
        return firstName + " " + lastName;
    }
}
```

Listing 4.1 – Classe Abstraite Person

4.2.2 Interface Schedulable

```
public interface Schedulable {
    LocalDateTime getScheduledDateTime();
    Duration getDuration();
    ActivityStatus getStatus();

    default LocalDateTime getEndDateTime() {
        return getScheduledDateTime().plus(getDuration());
    }

    default boolean conflictsWith(Schedulable other) {
        LocalDateTime thisStart = this.getScheduledDateTime();
        LocalDateTime thisEnd = this.getEndDateTime();
        LocalDateTime otherStart = other.getScheduledDateTime();
        LocalDateTime otherEnd = other.getEndDateTime();

        return thisStart.isBefore(otherEnd) &&
               thisEnd.isAfter(otherStart);
    }

    default boolean canCancelWithoutPenalty() {
        return LocalDateTime.now().plusHours(24)
            .isBefore(getScheduledDateTime());
    }
}
```

Listing 4.2 – Interface Schedulable

4.2.3 Détection des Conflits

```
public List<String> checkSchedulingConflicts(
        String teacherId, List<String> studentIds,
        String roomId, LocalDateTime dateTime,
        int durationMinutes) {

    List<String> conflicts = new ArrayList<>();

    // Verifier les conflits du professeur
    for (Lesson lesson : repository.getTeacherLessons(teacherId)) {
        if (tempSchedulable.conflictsWith(lesson)) {
            conflicts.add("Le professeur a une autre lecon");
        }
    }

    // Verifier les conflits de salle
    for (ScheduledActivity activity :
            repository.getRoomActivities(roomId)) {
        if (tempSchedulable.conflictsWith(activity)) {
            conflicts.add("La salle est deja reservee");
        }
    }

    return conflicts;
}
```

Listing 4.3 – Détection des Conflits de Planification

4.2.4 Exemple de Constructeur de Copie

```java
public class Student extends Person {
    private Level level;
    private List<String> preferredInstruments;
    private Map<String, Integer> packageHours;

    // Constructeur de copie
    public Student(Student other) {
        super(other);  // Appel au constructeur parent
        this.level = other.level;
        // Copie profonde des collections
        this.preferredInstruments =
            new ArrayList<>(other.preferredInstruments);
        this.packageHours =
            new HashMap<>(other.packageHours);
    }

    @Override
    public Person copy() {
        return new Student(this);
    }
}
```

Listing 4.4 – Constructeur de Copie de Student

Chapitre 5

Concepts de Programmation Orientée Objet

5.1 Classes Abstraites

Le système implémente trois classes abstraites :

TABLE 5.1 – Résumé des Classes Abstraites

Classe Abstraite	Étendue Par	Méthodes Abstraites
Person	Student, Teacher	getIdPrefix(), getRole copy()
Service	CoursePackage, IndividualLesson, InstrumentRental	getIdPrefix(), isValid copy()
ScheduledActivity	Lesson, RoomBooking	getIdPrefix(), getActivit Type(), consumesLesso Hours()

5.2 Interfaces

Trois interfaces définissent des capacités indépendantes :

TABLE 5.2 – Résumé des Interfaces

Interface	Implémentée Par	Méthodes Clés
Schedulable	ScheduledActivity, Lesson, RoomBooking	getScheduledDateTime(), conflictsWith()
Billable	Service et sous-classes	calculateAmount(), getBillingDescription()
Bookable	Teacher, Room, Instrument	isAvailableAt(), addBooking()

5.3 Polymorphisme

5.3.1 Collections Polymorphiques

```java
// Liste de Person contenant Student et Teacher
List<Person> people = new ArrayList<>();
people.addAll(repository.getAllStudents());
people.addAll(repository.getAllTeachers());

for (Person person : people) {
    // Dispatch dynamique - sortie differente pour chaque type
    System.out.println(person.getRole());
}
```

Listing 5.1 – Collections Polymorphiques

5.3.2 Appels de Méthodes Polymorphiques

```java
// Tous les services implementent Billable
List<Billable> billables = new ArrayList<>();
billables.addAll(packages);      // CoursePackage
billables.addAll(lessons);       // IndividualLesson
billables.addAll(rentals);       // InstrumentRental

for (Billable b : billables) {
    // calculateAmount() varie selon le type
    total = total.add(b.calculateAmount());
}
```

Listing 5.2 – Facturation Polymorphique

5.4 Constructeurs de Copie

Les constructeurs de copie sont implémentés dans toutes les classes principales :

- Student, Teacher
- CoursePackage, IndividualLesson, InstrumentRental
- Lesson, RoomBooking
- Room, Instrument
- Exam, Invoice, Payment

5.5 Encapsulation

- Tous les champs sont `private` ou `protected`
- Les getters retournent des copies défensives des collections mutables
- Les setters valident les données d'entrée

```java
public List<String> getPreferredInstruments() {
    return new ArrayList<>(preferredInstruments);
}
```

Listing 5.3 – Copie Défensive

Chapitre 6

Tests et Résultats

6.1 Données de Test

Le système inclut des données de test complètes :

— **8 Étudiants** (dont 3 mineurs)
— **5 Professeurs** (Piano, Violon, Guitare, Batterie, Flûte)
— **8 Salles** (Studios, Salles de pratique, Salle d'ensemble)
— **7 Instruments** à louer
— **5 Forfaits de Cours**
— **4 Examens à Venir**

6.2 Captures d'Écran de l'Application

6.2.1 Menu Principal

```
+============================================+
|              MENU PRINCIPAL                |
+============================================+
|  1. Gerer Etudiants et Professeurs         |
|  2. Gerer Forfaits et Lecons               |
|  3. Gerer Planification et Reservations    |
|  4. Gerer Paiements et Facturation         |
|  5. Gerer Examens et Resultats             |
|  6. Demonstrer Concepts POO                |
|  0. Quitter                                |
+============================================+
```

FIGURE 6.1 – Menu Principal de l'Application

6.2.2 Liste des Étudiants

```
=== TOUS LES ETUDIANTS (8) ===
ID              Nom                  Niveau         Heures
-----------------------------------------------------------------------
STU-A1B2C3D4    Alice Moreau         Intermediaire  8
STU-E5F6G7H8    Thomas Leroy         Debutant       6
STU-I9J0K1L2    Emma Dubois          Avance         Illimite
```

FIGURE 6.2 – Affichage de la Liste des Étudiants

6.2.3 Détection des Conflits

```
X Erreur: Conflits de planification detectes:
  - Le professeur a une autre lecon a 2025-01-15T10:00
  - La salle est deja reservee a 2025-01-15T10:00
```

FIGURE 6.3 – Message d'Erreur de Conflit

Chapitre 7

Conclusion

7.1 Résumé

Le projet **Conservatoire Virtuel** implémente avec succès un système complet de gestion d'école de musique répondant à toutes les exigences spécifiées.

7.2 Checklist des Exigences POO

TABLE 7.1 – Exigences POO Satisfaites

Exigence	**Statut**	**Implémentation**
Classes Abstraites (2+)	✓	Person, Service, ScheduledActivity
Interfaces (2+)	✓	Schedulable, Billable, Bookable
Polymorphisme	✓	Collections et appels de méthodes
Constructeurs de Copie	✓	Toutes les classes principales
Héritage	✓	Multiples hiérarchies
Composition	✓	Teacher-TimeRange, Exam-Registration
Encapsulation	✓	Champs privés, copies défensives

7.3 Améliorations Futures

— Intégration base de données

— Interface graphique utilisateur

— API REST

— Notifications par email

Annexe A

Comment Exécuter

```
# Naviguer vers le repertoire du projet
cd java-project

# Compiler et executer avec Maven
mvn clean compile
mvn exec:java -Dexec.mainClass="com.music.school.ConservatoireApp"

# Ou packager et executer comme JAR
mvn package
java -jar target/conservatoire-virtuel-1.0.0.jar
```

Listing A.1 – Commandes d'Exécution

Bibliographie

[1] Oracle Corporation, *Java SE 17 Documentation*, 2021.

[2] Bertrand Meyer, *Object-Oriented Software Construction*, Prentice Hall, 2nd Edition, 1997.

[3] Robert C. Martin, *Clean Code : A Handbook of Agile Software Craftsmanship*, Prentice Hall, 2008.