Modélisation UML : Gestion d'une école de musique

ali.zainoul.az@gmail.com

13 juin 2025

Contexte

L'école de musique enseigne des instruments, emploie des professeurs, accueille des élèves, gère des cours et organise des concerts. L'ensemble des mini-TPs ci-dessous permet de modéliser progressivement ce système à l'aide d'un **diagramme de classes UML** cohérent et complet.

Mini-TP 1 — Héritage : Les personnes

Objectif : Comprendre l'héritage et la factorisation de comportements.
Sujet : Créez une classe abstraite `Person` avec un attribut `name` et une méthode `getFullName()`. Deux sous-classes héritent de cette classe : `Student` et `Teacher`.
Relation UML couverte : Héritage

Mini-TP 2 — Association simple : Élève et instrument

Objectif : Représenter une relation entre deux objets existants.
Sujet : Un `Student` peut apprendre un instrument. Un `Instrument` a un nom et une famille (cordes, vents, etc.). Créez une association 1→1 entre `Student` et `Instrument`, avec possibilité pour plusieurs élèves d'apprendre le même instrument.
Relation UML couverte : Association

Mini-TP 3 — Agrégation : Professeur et Cours

Objectif : Montrer une relation "partie de" sans possession forte.
Sujet : Un `Teacher` peut donner plusieurs `Course`, mais un `Course` peut continuer même si le professeur change.
Relation UML couverte : Agrégation (le `Teacher` agrège des `Course`, sans en être propriétaire)

Mini-TP 4 — Composition : Cours et Planning

Objectif : Montrer une relation "partie-tout" avec cycle de vie partagé.
Sujet : Un `Course` contient un `Schedule` (jour, heure, salle). Le `Schedule` n'existe que dans le contexte de ce `Course`.
Relation UML couverte : Composition

Mini-TP 5 — Implémentation : Interface Playable

Objectif : Utiliser une interface et modéliser la réalisation.
Sujet : Une interface `Playable` définit la méthode `playNote(note: str)`. Les classes `Student` et `Teacher` l'implémentent si elles jouent d'un instrument.
Relation UML couverte : Réalisation

Mini-TP 6 — Dépendance : Concert et invités

Objectif : Représenter une dépendance temporaire (dans une méthode ou un constructeur).
Sujet : La classe `Concert` utilise temporairement un `Student` ou un `Teacher` via la méthode `invite(Person)`. Il n'y a pas de lien d'attribut ou de relation forte.
Relation UML couverte : Dépendance

Mini-projet final — Intégration

Sujet : Regroupez tous les mini-TPs dans un **grand diagramme de classes UML cohérent**. Implémentez ensuite ce modèle orienté objet dans un mini-programme.

Livrables attendus

— Un **diagramme de classes UML** annoté ;
— Le **code** dans le langage POO de votre choix ;
— Un **fichier d'explication** (Markdown ou texte) motivant les choix de conception.

Diagramme de classes UML (textuel)

Person (abstract)
– name : String
– getFullName() : String

Student ⟶ Person
– instruments : List<Instrument>
– getFullName()
– playNote(note : String)
– chooseInstrument(instrument : Instrument)

Teacher ⟶ Person
– getFullName()
– playNote(note : String)

Instrument
– name : String
– family : String

Course
– title : String

Schedule
– day : String
– time : String
– room : String

Concert
– invite(person : Person)

Playable (interface)
– playNote(note : String)

Diagramme UML en notation Mermaid

```
classDiagram

class Playable {
  <<interface>>
  +playNote(note: String)
}

class Person {
  <<abstract>>
  - String name
  + getFullName(): String
}

class Student {
  - List<Instrument> instruments
  + getFullName(): String
  + playNote(note: String)
  + chooseInstrument(instrument: Instrument)
}

class Teacher {
  + getFullName(): String
  + playNote(note: String)
}

class Instrument {
  - String name
  - String family
}

class Course {
  - String title
}

class Schedule {
  - String day
  - String time
  - String room
}

class Concert {
  + invite(person: Person)
}

Person <|-- Student
Person <|-- Teacher
```

```
Playable <|.. Student
Playable <|.. Teacher
Student "1" <--> "N" Instrument
Teacher o-- Course
Course *-- Schedule
Person ..> Concert
```