

Labo 6 - Faria/Aubry

1. Modèle de données et relations entre les entités

Le projet simule un système d'administration scolaire avec des groupes d'étudiants, des professeurs, et des leçons. Voici les principaux choix de conception liés aux entités :

Classe `Personne`

- **Description** : Classe abstraite servant de base pour les autres entités (`Etudiant` et `Professeur`).
- **Attributs** : `nom` , `prenom` .
- **Méthodes** : Redéfinition de `toString()` dans les sous-classes pour fournir des informations spécifiques.
- **Objectif** : Centraliser les informations communes à toutes les personnes.

Classe `Etudiant`

- **Description** : Hérite de `Personne` , ajoute un `matricule` et une référence à un `Groupe` .
- **Attributs** : `matricule` , `groupe` .
- **Méthodes** :
 - Redéfinition de `toString()` pour afficher le nom, prénom, matricule et groupe associé.
- **Objectif** : Représenter un étudiant avec un identifiant unique et un groupe.

Classe `Professeur`

- **Description** : Hérite de `Personne` , ajoute un champ `abreviation` (ex : "PDO" pour un professeur de POO).
- **Attributs** : `abreviation` , `lecons` .
- **Méthodes** :
 - `abreviation()` : Renvoie l'abréviation du professeur.
 - `horaire()` : Affiche l'emploi du temps du professeur basé sur ses leçons.
 - Redéfinition de `toString()` pour afficher le nom, prénom et l'abréviation du professeur.
- **Objectif** : Représenter un professeur avec des leçons et un emploi du temps.

Classe `Groupe`

- **Description** : Représente un groupe d'étudiants avec une orientation, un trimestre, et un numéro.
- **Attributs** : `orientation` , `trimestre` , `numero` , `etudiants` , `lecons` .
- **Méthodes** :
 - `ajouterEtudiant(Etudiant etudiant)` : Ajoute un étudiant au groupe.

- `ajouterLecon(Lecon lecon)` : Ajoute une leçon au groupe.
- `getNom()` : Retourne le nom du groupe sous la forme de "orientation-trimestre-numero".
- `afficherHoraire()` : Affiche l'emploi du temps du groupe en utilisant la classe `HoraireUtils`.
- **Objectif** : Gérer les étudiants et les leçons d'un groupe spécifique.

2. Gestion des leçons et emploi du temps

Classe `Lecon`

- **Description** : Représente une leçon dans un emploi du temps. Contient des informations comme la matière, le jour, la période de début, la durée, la salle et le professeur.
- **Attributs** : `matiere`, `jourSemaine`, `periodeDebut`, `duree`, `salle`, `professeur`.
- **Objectif** : Permet de gérer et d'afficher les informations sur une leçon spécifique.

Association entre `Professeur` et `Lecon`

- Chaque leçon est associée à un professeur via l'attribut `professeur` dans la classe `Lecon`.
- La méthode `horaire()` de `Professeur` permet d'extraire les leçons attribuées à un professeur et de générer son emploi du temps.

3. Affichage de l'emploi du temps

Classe `HoraireUtils`

- **Description** : Utilitaire pour afficher un emploi du temps sous forme de grille.
- **Méthodes** :
 - `afficherHoraire(List<Lecon> lecons, String titre)` : Génère une grille horaire à partir des leçons et d'un titre fourni. La grille affiche les matières et les salles de cours en fonction des jours et périodes.
- **Objectif** : Centraliser la logique d'affichage des horaires pour les professeurs et les groupes d'étudiants.

Fonctionnement de la méthode `afficherHoraire`

- La méthode génère une grille de 5 jours (Lun-Ven) et 11 périodes horaires. Pour chaque leçon, elle place la matière, la salle et l'abréviation du professeur dans la grille, prenant en compte la durée des leçons.
- Le titre, le séparateur et les horaires sont formatés et affichés proprement dans le résultat final.


4. Organisation des collections

Listes (`ArrayList`)

- **Description** : Les données, comme les étudiants, les leçons et les personnes, sont stockées dans des `ArrayList` . Cela permet une gestion dynamique des objets et une grande flexibilité pour l'ajout et la suppression d'éléments.
- **Objectif** : Fournir une structure de données efficace pour gérer les entités et leurs relations.

5. Principe de séparation des responsabilités

Le design suit le principe de séparation des responsabilités :

- **Les** `Personne` **et ses sous-classes** gèrent les informations personnelles des individus (étudiants et professeurs).
- **La classe** `Lecon` gère les informations relatives aux leçons.
- **La classe** `Professeur` gère l'emploi du temps des professeurs via la méthode `horaire()` .
- **La classe** `Groupe` gère les étudiants d'un groupe ainsi que les leçons qui lui sont associées.
- **La classe** `HoraireUtils` centralise la logique d'affichage des horaires.
- **La classe**  `Main` centralise l'exécution du programme, instanciant les objets nécessaires et coordonnant les interactions.

6. Méthodes d'affichage et présentation des données

Méthode `toString()`

- Chaque classe redéfinit la méthode `toString()` pour fournir une représentation textuelle des objets.
- Par exemple, dans la classe `Etudiant` , `toString()` affiche son nom, prénom, matricule et groupe (si applicable), ce qui facilite l'affichage dans la console.

Gestion de l'affichage de l'emploi du temps

- L'emploi du temps des professeurs est géré par la méthode `horaire()` de la classe `Professeur` , qui filtre les leçons en fonction du professeur et génère un affichage structuré de son emploi du temps.

7. Flexibilité et extensibilité

Le système est conçu de manière à faciliter l'extension :

- **Ajout de nouvelles classes** : De nouvelles entités (comme des assistants ou des administrateurs) peuvent être ajoutées facilement en créant des sous-classes de `Personne` .
- **Modification de l'emploi du temps** : Le système peut être étendu pour gérer des conflits d'horaires ou d'autres aspects liés aux leçons et groupes.

8. Points d'amélioration possibles

- **Gestion des erreurs** : Il n'y a pas de gestion explicite des erreurs, par exemple, pour vérifier que les étudiants sont correctement ajoutés à un groupe ou pour valider les horaires des leçons.
- **Tests unitaires** : L'ajout de tests unitaires plus précis pour valider les méthodes critiques (ajout d'étudiants, gestion des leçons, etc.) pourrait améliorer la robustesse du programme.