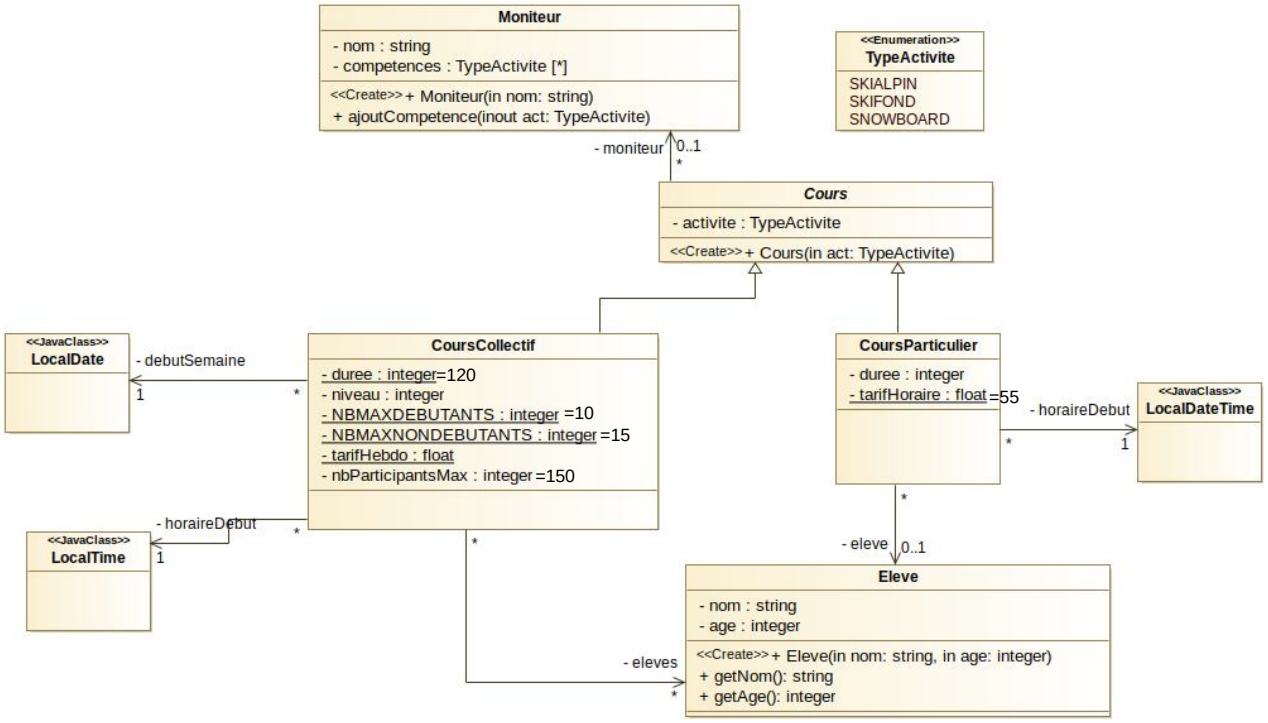
Examen Modélisation et Programmation par Objets

Documents autorisés : une feuille A4 recto-verso. Le barème est donné à titre indicatif et pourra être modifié.

On s'intéresse au système de planification et réservation de cours d'une école de ski et de snowboard. Une première modélisation incomplète des cours est fournie ci-dessous, ainsi que des éléments de code.



Un cours est dispensé par un moniteur. Un moniteur a des compétences choisies parmi les types d'activités proposées par l'école de ski : ici le ski alpin, le ski de fond et le snowboard. Un cours concerne une activité, choisie parmi le ski alpin, le ski de fond et le snowboard. Il existe deux sortes de cours (`Cours` est une classe abstraite) : les cours collectifs et les cours particuliers.

Un cours collectif a lieu sur 6 jours consécutifs. Le premier jour est représenté par une `LocalDate` (association `debutSemaine`). Un cours collectif a un horaire de début de cours (association `horaireDebut`) et une duree de 120 minutes (attribut `duree`). Il correspond à un niveau de difficulté dans l'activité, ici représenté par un entier, de 1 à 10 (attribut `niveau`). Un cours a des élèves inscrits (association vers la classe `Eleve`). Il y a un nombre maximal d'élèves dans un cours (attribut `nbParticipantsMax`). Dans un cours de difficulté 1 ou 2, il y a au maximum 10 élèves (constante `NBMAXDEBUTANTS`) et dans les autres cours il y a au max 15 élèves (constante `NBMAXNONDEBUTANTS`). Le tarif hebdomadaire pour un cours collectif est fixé à 150 euros (attribut `tarifHebdo`).

Un cours particulier concerne un unique élève. Il commence un certain jour à une certaine heure (association `horaireDebut` vers la classe Java `LocalDateTime`). Il a une certaine durée (attribut `duree`). Son coût est fonction de sa durée, le tarif horaire est fixé à 55 euros (attribut `tarifHoraire`).

Programme 1 : *Enumération TypeActivite*

```
public enum TypeActivite {
        SKIALPIN, SKIFOND, SNOWBOARD;
}
```

Programme 2 : *Classe Moniteur*

```
public class Moniteur {
        private String nom;
        private ArrayList<TypeActivite> competences;

        public Moniteur(String nom) {
                this.nom = nom;
                competences=new ArrayList<>();
        }

        public void ajoutCompetence(TypeActivite act) {
                competences.add(act);
        }
}
```

Programme 3 : *Classe abstraite Cours*

```
public abstract class Cours {
        private TypeActivite activite;
        private Moniteur moniteur;

        public Cours(TypeActivite act) {
                this.activite=act;
        }
}
```

Programme 4 : *Classe CoursCollectif*

```
public class CoursCollectif extends Cours {
        private LocalDate debutSemaine;
        private LocalTime horaireDebut;
        private static int duree=120;// en minutes
        private int niveau;// niveau de 1 à 10
        private static final int NBMAXDEBUTANTS=10;
        private static final int NBMAXNONDEBUTANTS=15;
        private static float tarifHebdo=150;
        private int nbParticipantsMax;
        private ArrayList<Eleve> eleves;
}
```

Programme 5 : *Classe CoursParticulier*

```
public class CoursParticulier extends Cours {
        private LocalDateTime horaireDebut;
        private int duree;// en minutes
        private static float tarifHoraire=55;
        private Eleve eleve;
}
```

Programme 6 : *Classe Eleve*

```
public class Eleve {
        private String nom;
        private int age;
```

```
        public Eleve(String nom, int age) {
                this.nom = nom;
                this.age=age;
        }

        public String getNom() {
                return nom;
        }

        public int getAge() {
                return age;
        }
}
```

Question 1. (1 point) Dans la classe `Moniteur`, écrivez en Java la méthode `aPourCompétence`, qui prend en paramètre un type d'activité, et qui retourne vrai si et seulement si ce type d'activité fait partie des compétences du moniteur.

Question 2. (1 point) Dans la classe `Cours`, ajoutez en Java une méthode qui permet d'affecter au cours un moniteur donné. S'il y a déjà un moniteur, celui-ci est remplacé par le nouveau moniteur. Un moniteur ne peut être affecté à un cours que si le cours concerne une activité pour laquelle le moniteur est compétent.

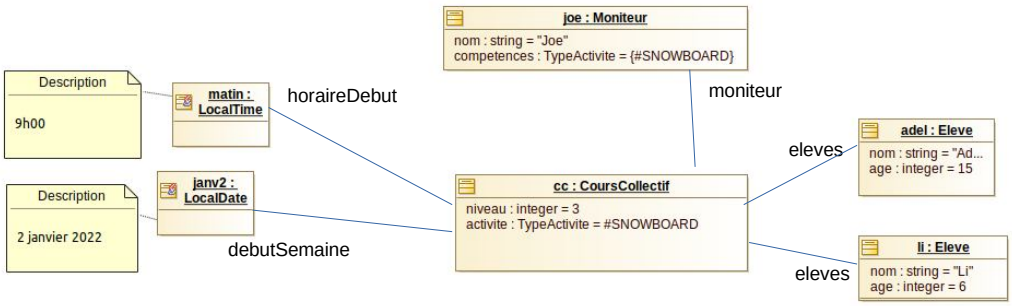
Question 3. (1,5 points) Dans la classe `CoursCollectif`, écrivez en Java le constructeur paramétré, qui permet de positionner le type d'activité (reçu en paramètre), le début de la semaine de cours et son horaire (reçus en paramètre), le niveau du cours (reçu en paramètre), les élèves participants au cours (aucun élève à la création), et le nombre de participants maximum pour le cours (10 pour les cours de niveau 1 et 2, et 15 pour les autres cours). A la création du cours, le moniteur n'est pas connu.

Question 4. (1,5 points) Dans la classe `CoursCollectif`, écrivez en Java une méthode qui calcule la moyenne d'âge des élèves.

Question 5. (2 points) Dans les classes `Cours`, `CoursCollectif`, et `CoursParticulier`, implémentez en Java les méthodes suivantes :

- `inscrire` : pour tous les cours, on peut inscrire un élève pris en paramètre et cette méthode retourne un booléen qui vaut vrai si et seulement si l'inscription a été possible. Pour les cours particuliers, l'inscription est possible si aucun élève n'était déjà inscrit, et dans ce cas, positionne l'unique élève du cours à l'élève reçu en paramètre. Pour les cours collectifs, l'inscription est possible si le nombre d'élève inscrits est strictement inférieur à la capacité maximale du cours. Cette capacité maximale est de 10 pour les cours de niveau 1 et 2, et de 15 pour les autres cours.
- `tarif` : pour tous les cours, on peut obtenir le tarif pour une inscription au cours. Pour les cours particuliers, le tarif est proportionnel à la durée, avec un coût horaire de 55 euros. Pour les cours collectifs, le tarif est forfaitaire, et fixé à 150 euros.

Question 6. (2 points) Donnez en Java le code permettant de mettre en place les instances et liens illustrés sur le diagramme d'objets ci-dessous. Vous supposerez disposer d'une variable :
`LocalDate janv2=LocalDate.of(2022, 01, 02);`
et d'une variable :
`LocalTime matin=LocalTime.of(9,0);`.



Question 7. Les cours particuliers évoluent et peuvent maintenant accueillir jusqu'à 3 élèves. On décide donc que les élèves ne sont plus représentés spécifiquement dans les classes `CoursCollectif` et `CoursParticulier` mais sont factorisés dans la classe `Cours`. On déplace également l'attribut représentant le nombre maximal d'élèves d'un cours de la classe `CoursCollectif` à la classe `Cours`.

Suite à cette modification :

a- (1,5 points) Donnez les modifications à apporter aux constructeurs des classes `Cours` et `CoursCollectif`, et donnez en Java un constructeur paramétré pertinent pour la classe `CoursParticulier`.

b- (1 points) Expliquez les modifications à apporter à ce que vous aviez implémenté pour la méthode `inscrire` à la question 5.

Question 8. (3,5 points) On ajoute au système une classe `PlanningSaison`. Cette classe référence tous les cours planifiés pour une saison, indexés par le numéro de la semaine pendant laquelle le cours a lieu : à partir d'une instance de `PlanningSaison` et d'un numéro de semaine, on obtient les cours prévus pour cette semaine. On supposera que le numéro de la semaine d'un cours est donné par la méthode `numSemaine()` introduite abstraite dans la classe `Cours`. Pour un même numéro de semaine, il y a plusieurs cours planifiés dans `PlanningSaison`. Un cours est planifié dans au plus une instance de `PlanningSaison`.

a- Modélisez dans un diagramme de classe UML la classe `PlanningSaison` (sans attribut ni méthode) et sa relation avec la classe `Cours`.

b- Proposez une implémentation en Java de cette relation.

c- Implémentez en Java dans la classe `PlanningSaison` une méthode qui ajoute un cours donné aux cours déjà planifiés.

Question 9. (2 points) On prépare une évolution du système dans laquelle on introduira d'autres cours récurrents que les cours collectifs actuels sur une semaine, comme par exemple des cours ayant lieu tous les dimanche hors vacances scolaires pour les locaux. On veut alors introduire une interface `IActiviteRecurrente`. Une activité récurrente doit fournir sa date et son heure de début, sa fréquence (à choisir parmi quotidienne, hebdomadaire ou mensuelle) et son nombre d'occurrences. Introduisez une telle interface en Java et expliquez comment modifier la classe `CoursCollectif` de manière à ce qu'elle l'implémente. On suppose disposer de l'énumération :
`public enum Frequence{quotidienne, hebdomadaire,mensuelle;}`.

Question 10. (3 points) Proposez un diagramme de classes permettant de modéliser les éléments suivants. L'école de ski gère un planning saisonnier. Elle emploie plusieurs moniteurs pour ses différents activités. L'école de ski maintient à jour la liste de ses clients. Un client a juste un nom et un numéro pour simplifier. Un client peut effectuer des inscriptions à des cours. On conserve la date à laquelle l'inscription a été réalisée. L'inscription peut être réalisée avec une assurance annulation ou pas. Un même client peut faire plusieurs inscriptions, pour différents cours, pour différents élèves. Par exemple un parent (qui serait ici un client) peut faire une inscription pour chacun de ses enfants. On modélisera les classes et les énumérations nécessaires, avec leurs attributs, et les relations que ces classes et énumérations entretiennent. On NE modélisera PAS la navigabilité des associations, ni les méthodes ni les accesseurs.