Exercice : un agenda

On se propose de concevoir un agenda, contenant une liste d'événements. Les fonctionnalités attendues sont les suivantes :

Spécifications :

L'agenda contient des événements (`Event`). Pour un événement, on doit pouvoir définir son titre, son instant de début (`LocalDateTime`) et sa durée (`Duration`).

En plus des événements simples (qui se produisent une seule fois) on veut pouvoir créer des événements répétitifs (`RepetitiveEvent`), qui se répètent indéfiniment.

- Les événements répétitifs ont une fréquence de répétition : ils peuvent être répétés chaque jour (`ChronoUnit.DAYS`), chaque semaine (`ChronoUnit.WEEKS`), ou chaque mois (`ChronoUnit.MONTHS`).
- Pour un événement répétitif, on peut définir des exceptions, c'est-à-dire des dates (`LocalDate`) où une répétition planifiée ne se produira pas (par exemple, un événement se produit tous les jours sauf le 1/1/2021).

On veut également représenter une autre forme d'événements : Des événements répétitifs qui ne se répètent pas indéfiniment, mais pour lesquels on peut fixer la terminaison (`FixedTerminationEvent`) :

- Ces événements peuvent se répéter un nombre prédéfini de fois (par exemple répéter 3 x un événement hebdomadaire), ou, de manière équivalente,
- se répéter jusqu'à une date donnée (par exemple, répéter un événement mensuel jusqu'au 31/12/2021).

Figure 1 : les événements peuvent s'étendre sur plusieurs jours

On doit pouvoir :

- Ajouter de nouveaux événements de n'importe quel type à l'agenda,
- Trouver tous les événements qui se produisent à une date donnée. Attention, on doit prendre en compte le fait qu'un événement peut s'étendre sur plusieurs jours. Ainsi, dans l'exemple de la Figure 1, les événements `Evénement1`, `Evénement2` et `Evénement5` se produisent le mercredi 6. On doit également prendre en compte les événements répétitifs dans ce calcul.

Ces spécifications donnent lieu au diagramme de classe UML illustré en Figure 2.

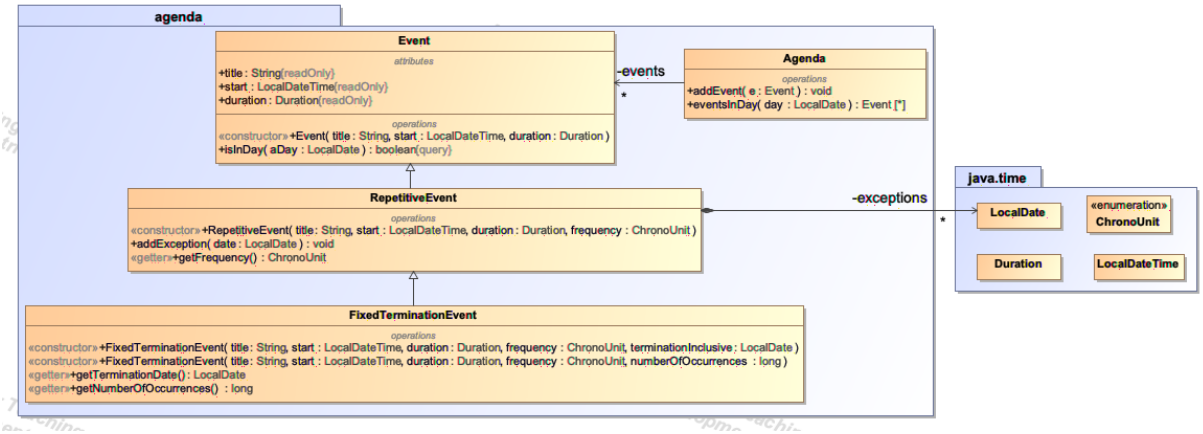

Figure 2 : Modèle UML

Travail à réaliser

A partir du point de départ disponible sur https://github.com/bastide/agenda-empty , implémenter ce diagramme UML de manière à faire passer les tests unitaires inclus.

Les tests unitaires fournis ne doivent pas être modifiés. Vous pouvez ajouter d'autres classes de test si nécessaire.

Éléments de solution

Cet exercice fait une utilisation intensive des classes du package `java.time` pour représenter et manipuler les données temporelles (dates, durées…). Quelques exemples de code :

```
// Une date, ex : 1° novembre 2020
LocalDate d1 = LocalDate.of(2020, 11, 1);
// Date et heure, 1° novembre 2020, 22:30
LocalDateTime dt = LocalDateTime.of(2020, 11, 1, 22, 30);
// Une durée : 120 minutes
Duration m = Duration.ofMinutes(120);
// Ajouter des durées à une date
LocalDate d2 = d1.minus(1, ChronoUnit.WEEKS)
// Le nombre de jours entre deux dates
long j = ChronoUnit.DAYS.between(d1, d2); // -7 jours
```