Polytech Lille GIS4
Contrôle de Programmation Par Objets

10 janvier 2020

> Tous documents papiers autorisés. Lire le sujet entièrement avant de répondre aux questions. Répondre en respectant l'ordre des questions et en rappelant leur numéro. Programmer vos solutions en Java. Vous pouvez utiliser ce qui a été spécifié dans une question même si vous n'y avez pas répondu. Ne vous préoccupez pas des packages ni des `import` de bibliothèques, et il ne doit pas être question de quelconque `main`.

Une entreprise souhaite se munir d'une plateforme pour organiser des réunions électroniques. De telles réunions permettent aux participants d'intervenir à distance sur des sujets variés et ceci pendant une certaine période.

1 Plateforme (4 points)

On part du diagramme de classes UML de la figure 1 :

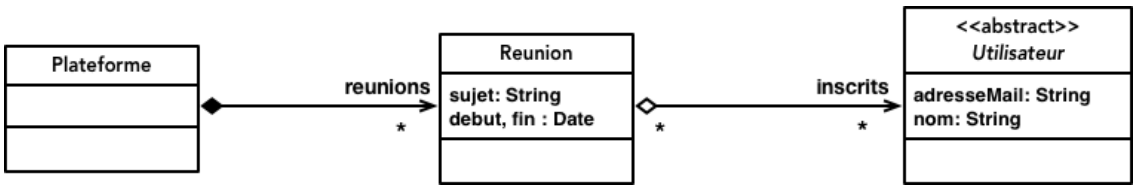

FIGURE 1 – Plateforme

- la plateforme (classe `Plateforme`) gère les réunions organisées par l'entreprise : association de rôle `reunions` vers la classe `Reunion`
- une réunion a un `sujet` (unique), une date de début et une date de fin (la classe `Date` est fournie ci-dessous pour l'ensemble du problème, il n'est pas demandé de la programmer)
- les participants à une réunion sont représentés par l'association de rôle `inscrits` vers la classe ***`Utilisateur`***. Cette classe est abstraite car il existe divers types d'utilisateurs comme on le verra par la suite.

On fait les choix suivants :

- on range les réunions de la plateforme dans une Map : $sujet \rightarrow Reunion$ qui doit permettre leur accès ordonné par `sujet`
- les `inscrits` à une réunion sont rangés dans un ensemble d'utilisateurs (`Set`) ordonnés par leur `adresseMail`.

Question

Programmer les classes de la figure 1 en respectant les choix précédents.

```
public class Date implements Comparable<Date> {
  public static Date today() // date du jour
  public int compareTo(Date d) // ordre chronologique
  public String toString()
}
```

2 Hiérarchie de classes d'utilisateurs (1,5 points)

On distingue en Figure 2 (mais cela doit rester extensible à d'autres classes d'utilisateurs) :

- les employés de l'entreprise (classe `Employe`) caractérisés par le `service` auquel ils appartiennent (par exemple : "Direction", "Technique", "RH", ...)
- les utilisateurs extérieurs à l'entreprise (classe `Exterieur`) caractérisés par :
  - le nom de leur société d'appartenance (attribut `societe`)
  - un contact au sein de l'entreprise : association de rôle `contact` entre `Exterieur` et `Employe`.

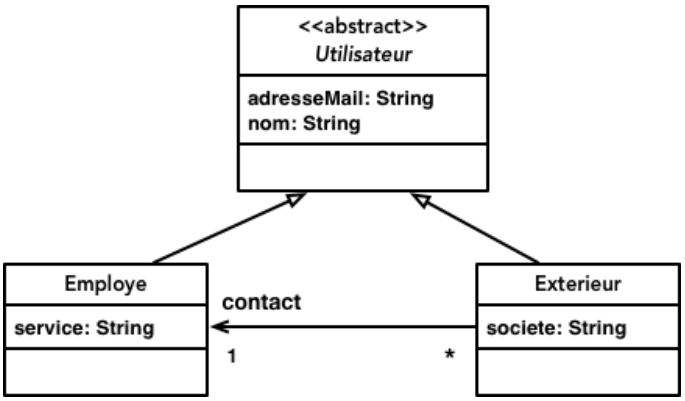

FIGURE 2 – Hiérarchie de classes d'utilisateurs

Question

Programmer la hiérarchie de classes de la figure 2.

Précision : il n'est pas demandé de programmer de méthodes à ce niveau.

3 Inscription aux réunions (4 points)

Un utilisateur ne peut s'inscrire à une réunion que si elle est ouverte, c'est-à-dire que la date du jour (cf. `today()` de la classe `Date`) est comprise entre ses dates de début et de fin, sinon une exception `DateException` doit être provoquée. En plus, un utilisateur `Exterieur` ne peut s'inscrire à une réunion que si son `contact` est lui-même inscrit, sinon une exception `InscriptionException` doit être provoquée.

Questions

Programmer les méthodes suivantes dans la classe `Reunion` pour répondre à ces spécifications :

1. `boolean ouverte()` : qui renvoie `true` si la date du jour est comprise entre ses dates de début et de fin, `false` sinon
2. `void inscrire(Employe x) throws DateException`
3. `void inscrire(Exterieur x) throws DateException, InscriptionException`

Précision :
— programmer les classes d'exception
— il peut être nécessaire de programmer d'autres méthodes dans les classes concernées.

4 Affichage des réunions en cours (4 points)

Question

Programmer dans la classe `Plateforme` une méthode `'void etat()'` et toutes les méthodes nécessaires dans les classes concernées pour afficher l'état des réunions ouvertes, ordonnées par sujet, et avec les informations suivantes :
— sujet de la réunion
— dates de début et de fin de la réunion
— `inscrits` à la réunion : `adresseMail`, `nom`, et :
  — dans le cas d'un `Employe` : son `service`
  — dans le cas d'un `Exterieur` : sa `societe` d'appartenance et l'`adresseMail` de son `contact`.

5 Information aux utilisateurs (3,5 points)

On donne la primitive d'envoi de mail `"send"` suivante, fournie en `static` par la classe `Plateforme`. Elle provoque une exception `MailException` si le destinataire n'est pas joignable (adresse mail erronée, ...) :

```
public class Plateforme {
    public static void send (
        String adresseMailDestinataire,
        String subject,
        String body
       )
       throws MailException
 }
```

5.1 Question

Dans la classe `Reunion` programmer une méthode `'void informer(String body)'` qui :
— envoie à tous ses `inscrits` un mail formé comme suit :
  — `subject` : `sujet` de la réunion
  — `body` : passé en paramètre.
— enregistre dans une variable d'instance `'Set<Utilisateur> unreachable'` de la réunion ses `inscrits` non joignables.

5.2 Question

Dans la classe `Plateforme` programmer par streaming et lambdas une méthode `'void alerteFin()'` qui informe (en utilisant la méthode `'void informer(String body)'` de la question 5.1) tous les inscrits aux réunions qui prennent fin à la date du jour avec :
`body : "Attention la réunion se termine aujourd'hui!"`.

6 Administration de la plateforme (3 points)

Question

Dans la classe `Plateforme` programmer une méthode `'void administration()'` qui :

1. rassemble en un seul `'Set<Utilisateur>'` tous les utilisateurs non joignables enregistrés par toutes les réunions (dans leur `'Set<Utilisateur> unreachable'`)
2. les traite de la façon suivante :
   — range les `Employe` dans une variable `'Set<Employe> unreachableEmployees'` de la plateforme (problèmes à résoudre en interne)
   — pour les `Exterieur` : prévient leur `contact` au sein de la société par mail :
      — `subject : "extérieur non joignable"`
      — `body` : nom de l'`Exterieur` concerné.