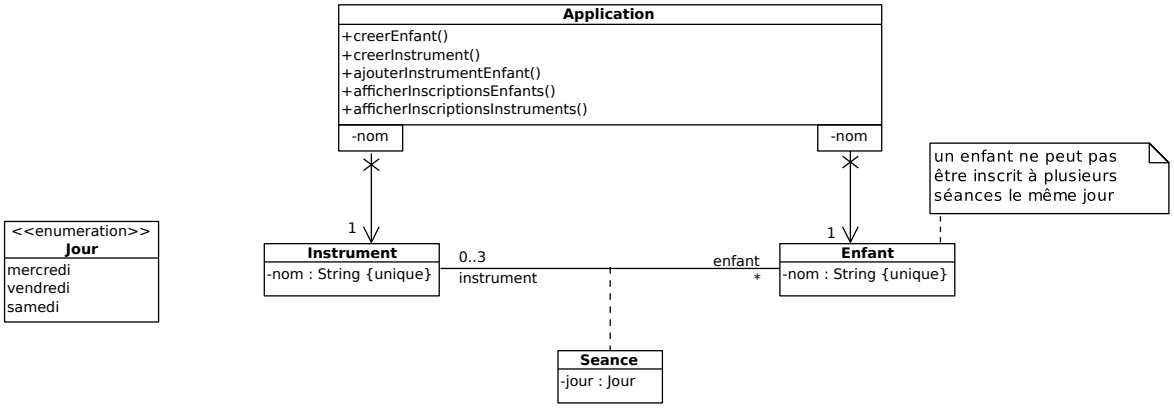
TP – Découverte d'instruments

Inscription à des séances de découverte d'instruments

Une école de musique organise des séances de découverte d'instruments pour des enfants. Les séances sont uniquement les mercredi, vendredi et samedi. Un enfant a au plus trois inscriptions, on ne peut pas choisir plusieurs fois le même instrument et on ne peut pas participer à plusieurs séances le même jour.

On veut connaître :
— pour chaque enfant les instruments choisis ainsi que le jour de la séance correspondant ;
— pour chaque instrument et pour chaque jour, la liste des enfants inscrits.



Remarques

— On rappelle que la contrainte « `{unique}` » signifie que l'on ne peut pas avoir plusieurs instruments (plusieurs enfants) avec le même nom.
— On rappelle que deux objets ne peuvent pas être liés par deux liens de la même association. Aussi, un enfant ne peut être lié deux fois au même instrument (et ce même pour des séances différentes).
— Pour exprimer qu'un enfant ne peut pas participer à plusieurs séances le même jour, une note associée à la classe `Enfant` précise cette contrainte.
— Les associations qualifiées sont implémentées en Java en utilisant la collection (plus précisément un tableau associatif ou dictionnaire) abstraite `Map` et son implémentation concrète `HashMap`.
— Les collections `HashSet` et `HashMap` sont brièvement décrites en Annexes A et B.

Scénarios des cas d'utilisation

`creerInstrument`

1. Le système indique à l'utilisateur·rice les noms de tous les instruments enregistrés ;
2. L'utilisateur·rice saisit le nom du nouvel instrument ;
3. Le système vérifie qu'il n'existe pas un instrument de ce nom ;
4. Le système crée l'instrument ;
5. Le système confirme la création de l'instrument.

`creerEnfant`

1. Le système indique à l'utilisateur·rice les noms de tous les enfants inscrits ;
2. L'utilisateur·rice saisit le nom du nouvel enfant ;
3. Le système vérifie qu'il n'existe pas un enfant du même nom ;
4. Le système crée l'enfant ;
5. Le système confirme la création de l'enfant.

`ajouterInstrumentEnfant`

1. Le système propose à l'utilisateur·rice les noms des enfants enregistrés qui peuvent encore s'inscrire à un instrument (ce sont les enfants pour lesquels il reste au moins une séance de libre, et qui ne sont pas déjà inscrits à tous les instruments) ;
2. L'utilisateur·rice saisit le nom de l'enfant ;
3. Le système vérifie qu'il existe un enfant de ce nom et qu'elle·il n'est pas déjà inscrit·e à trois séances de découverte ;
4. Le système propose à l'utilisateur·rice les noms des instruments pour lesquels l'enfant n'est pas encore inscrit ;
5. L'utilisateur·rice saisit le nom de l'instrument ;
6. Le système vérifie que l'instrument existe et que l'enfant n'est pas déjà inscrit à une séance de cet instrument ;
7. Le système propose les jours d'inscription (mercredi, vendredi ou samedi) encore possibles pour l'enfant ;
8. L'utilisateur·rice saisit un jour ;
9. Le système vérifie que ce jour existe et que l'enfant n'a pas déjà une séance ce même jour ;
10. Le système lie l'enfant à son instrument pour le jour souhaité ;
11. Le système confirme la nouvelle inscription.

`afficherInscriptionsEnfants`

Pour chaque enfant, le système affiche les noms des instruments et les jours des séances. Les lignes suivantes sont un exemple de résultat attendu :

```
===== Liste des inscriptions par enfant =====
· Eric
  - [vendredi] Tuba
  - [samedi] Triangle
  - [mercredi] Piano

· Jean
  - [samedi] Piano

· Nina
  - [samedi] Batterie
```

`afficherInscriptionsInstruments`

Pour chaque instrument le système affiche pour chaque jour les noms des enfants inscrits. Les lignes suivantes sont un exemple de résultat attendu :

```
===== Liste des inscriptions par instrument =====
· Piano
  - [samedi] Jean
  - [mercredi] Eric

· Batterie
  - [samedi] Nina

· Tuba
  - [vendredi] Eric

· Triangle
  - [samedi] Eric
```

Objectif

Dans le cadre de ce travail, il vous est demandé d'implanter un système logiciel pour gérer l'inscription d'enfants à des séances de découverte d'un instrument de musique.

Environnement de développement

Il vous est fourni un squelette de code configuré pour l'environnement de développement IntelliJ IDEA.

Vous utiliserez la version 11 de Java. La documentation en ligne est celle qui fait référence. La documentation de la bibliothèque standard est disponible à l'URL https://docs.oracle.com/en/java/javase/11/docs/api/index.html. En particulier la documentation des classes énumérées se trouve à l'URL https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Enum.html.

Contexte de réalisation

Votre application sera interactive. *Toutes les entrées/sorties* (lectures de données entrées par l'utilisateur·rice et affichage de données à l'utilisateur·rice) se font dans la classe `CLI` (Command Line Interface, ou interface en ligne de commande).

Un mécanisme de sérialisation (classe `Persisteur`) est fourni afin de rendre vos objets persistants. Les états des objets sont stockés dans un fichier (flux binaire). Pour qu'un objet puisse être sérialisé, il est *nécessaire* que sa classe implémente l'interface `java.io.Serializable`.

Attention

À chaque fois que vous ajoutez/supprimez des attributs ou des classes, vous devez supprimer le fichier de sérialisation situé dans le dossier `persistence/` de votre projet. Si vous ne souhaitez pas supprimer le fichier, vous devez incrémenter l'attribut `serialVersionUID` de la classe modifiée.

Gestion des erreurs et des contraintes

Lors de la réalisation de chacune des fonctionnalités demandées, les cas d'erreur donnent lieu à un message à destination de l'utilisateur. Toute opération de mise à jour donne lieu à un message de confirmation.

Travail attendu

1. Transformez – sur le diagramme de classe fourni – la classe association en classe « normale » (cf. polycopié de cours).
2. Récupérez, extrayez l'archive `.zip` puis ouvrez avec IntelliJ le projet contenant le squelette de l'application qui vous est fourni :
   — La classe `CLI` est responsable des interactions avec l'utilisateur·rice. Cette classe est également celle qui lance un menu proposant les différentes fonctionnalités de l'application.
   — La classe `Persisteur` réalise le mécanisme de sérialisation.
   — La classe `Jour` est une classe énumérée (mercredi, jeudi, vendredi).
   — La classe `Application` qui contient les méthodes à implanter a été initialisée.
   — Les méthodes `creerInstrument()` et `creerEnfant()` sont déjà implantées.
   — La méthode `afficherInscriptionsEnfants()` n'affiche que le nom des enfants : elle est donc à compléter.
3. Complétez les classes `Instrument`, `Enfant` et `Seance` avec leurs attributs. Ajoutez également les « getters » et « setters » correspondants.
4. Implantez la méthode `ajouterInstrumentEnfant()`. Vous porterez une attention particulière aux contraintes et commenterez explicitement les portions de codes portant sur leur mise en œuvre.
5. Complétez/Implantez les deux méthodes d'affichage : `afficherInscriptionsEnfants()` et `afficherInscriptionsInstruments()`.

Vous testerez l'ensemble de votre application. Toutes les situations générant des erreurs doivent être traitées par des messages à destination de l'utilisateur·rice.

A Utilisation de la collection `HashSet<E>`

La documentation de la collection `HashSet<E>` est consultable à l'URL https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashSet.html. La documentation en ligne est celle qui fait référence.

La classe `HashSet<E>` est un type particulier de collection d'éléments `E`. Les éléments ne sont pas ordonnés. Un élément ne peut être présent qu'une seule fois dans la collection. En ce sens, une instance de `HashSet<E>` correspond à un ensemble au sens mathématique d'éléments `E`.

Opérations usuelles

`boolean add(E e)` Ajoute un élément à l'ensemble s'il n'est pas déjà présent. Retourne vrai si l'élément n'était pas présent.

`boolean contains(Object o)` Retourne vrai si l'objet appartient à l'ensemble.

`boolean remove(Object o)` Supprime un élément de l'ensemble s'il y était présent.

`void clear()` Supprime tous les éléments de l'ensemble.

`int size()` Retourne le nombre d'éléments de l'ensemble.

`boolean isEmpty()` Retourne vrai si l'ensemble est vide.

Exemple d'utilisation

```
1 Set<Activite> lesActivites = new HashSet<>();
2
3 // parcours de toutes les activités de l'ensemble
4 for (Activite act: lesActivites) {
5         String nomAct = act.getNomActivite();
6         // ...
7 }
```

B Utilisation de la collection `HashMap<K, V>`

La documentation de la classe `HashMap<K, V>` est consultable à l'URL https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/HashMap.html. La documentation en ligne est celle qui fait référence.

La classe `HashMap<K, V>` contient des couples (clef, valeur). Dans une instance donnée chaque clef est unique, tandis que les valeurs peuvent apparaître plusieurs fois. C'est un type particulier de collection d'éléments `V`, où chaque élément est associé à une clef discriminante `K`. On nomme aussi ce type de collection indexée « tableau associatif » ou « dictionnaire ».

Opérations usuelles

`V put(K key, V value)` Ajoute un objet `value` identifié par la clef `key` si la clef n'est pas déjà présente. Sinon modifie la valeur associée à cette clef et retourne la valeur précédemment associée.

`V get(Object key)` Retourne l'objet associé à la clef `key`. Retourne `null` si la clef n'est pas présente.

`V remove(Object key)` Supprime la clef et la valeur associée si la clef est présente.

`void clear()` Supprime tous les couples du tableau associatif.

`int size()` Retourne le nombre de couples présents dans le tableau associatif.

`boolean containsValue(Object value)` Retourne vrai si la valeur `value` est présente dans le tableau associatif.

`boolean containsKey(Object key)` Retourne vrai si la clef `key` est associée à un élément.

`Set<K> keySet()` Retourne une vue ensemble contenant toutes les clefs. Attention la suppression d'un élément `key` de cet ensemble supprime le couple de clef `key` du tableau associatif.

`Collection<V> values()` Retourne une vue collection contenant toutes les valeurs. Attention la suppression d'un élément `val` de cette collection supprime l'entrée pour cette valeur `val` du tableau associatif.

Exemple d'utilisation

```
1  Map<String, Activite> lesActivitesParNom = new HashMap<>();
2
3  Activite act;  // = ...
4
5  // ajout d'une activité avec sa clef dans le tableau associatif
6  lesActivitesParNom.put("ski", act);
7
8  // accès à une valeur à partir de la valeur de sa clef
9  act = lesActivitesParNom.get("ski");
10
11 // parcours de toutes les activités du tableau associatif
12 for (Activite act: lesActivitesParNom.values()) {
13         String nomAct = act.getNomActivite();
14         // ...
15 }
```