Examen Ecrit

Tous documents sur support papier autorisés. Durée 2h.
L'ensemble des réponses sera à donner sur les feuilles d'énoncé. Ne pas dégrapher les feuilles.

Question 1. Ecrire une interface `IActiviteSportive`.

On souhaite représenter des activités sportives (patinage, luge, bobsleigh, voile, surf, aviron, etc.) qui disposent : (1) d'une méthode `getNom` permettant de connaître leur nom sous forme d'une chaîne de caractères, (2) d'une méthode `getCotisation` permettant de connaître le montant annuel de la cotisation à la fédération sportive correspondante (faux sinon), (3) d'une méthode `disciplineOlympique` retournant vrai si la discipline est inscrite au programme des jeux olympiques. Ecrivez le code Java de l'interface représentant les objets disposant de ces méthodes.

Question 2. Compléter l'interface `IActiviteSportive`.

On rappelle l'interface existant en Java pour représenter les objets comparables : `public Interface Comparable<T> {int compareTo(T o)}`. Comme l'indique la documentation, dans une implémentation, la méthode `compareTo` doit retourner *a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.* Compléter le code de l'interface `IActiviteSportive` (indiquez les modifications sans tout recopier) de manière à ce qu'elle spécialise l'interface `Comparable`, pour représenter le fait que les activités sportives sont comparables entre elles d'après leur nom (avec une méthode `défaut`). Ajustez le paramètre `T` au problème et implémentez la méthode `compareTo`.

Question 3. Implémenter une interface.
(1) Ecrivez une classe `ActiviteSportive` qui implémente l'interface `IActiviteSportive`. N'écrivez que les attributs et les méthodes nécessaires pour implémenter l'interface. (2) Puis écrivez l'en-tête d'une classe `SportGlace` qui est sous-classe de `ActiviteSportive` (uniquement l'en-tête). Pour la suite, leurs constructeurs et accesseurs sont tous supposés exister avec les conventions classiques.

Question 4. Utiliser une classe de l'API et écrire une structure de données
Nous introduisons la classe générique `TreeSet<T>` de l'API standard de Java. Cette classe permet de représenter des ensembles à l'aide d'une structure interne arborescente. Un ensemble est une collection d'objets sans doublons (pour savoir si des objets sont identiques, la méthode `compareTo` de la classe des objets stockés dans l'ensemble est appelée). `TreeSet` dispose entre autres des méthodes suivantes :

- un constructeur sans paramètre pour créer un ensemble vide
- s.size() retourne le nombre d'éléments
- s.add(e) — Adds e to s if e is not already present.
- s.contains(e) — Returns true if this set contains e.
- s1.retainAll(s2) — transforms s1 into the intersection of s1 and s2. (The intersection of two sets is the set containing only the elements common to both sets.)

Utilisez la classe `TreeSet<T>` pour mettre en place une classe générique `CarteMultisports`. Une carte multi-sports appartient à une personne (dont le nom sera stocké comme attribut). Elle contient l'ensemble des activités sportives auxquelles participe la personne. On désire représenter des cartes multi-sports contenant un ensemble de sports hétérogènes (comme $\{luge, VTT\}$) ou homogènes (d'une même catégorie comme $\{luge, curling\}$ qui sont des instance d'une même classe, ici `SportGlace`, la classe en question devant être obligatoirement une classe d'activités sportives). Dans cette question, écrivez (1) l'entête de la classe générique `CarteMultisports` (2) ses attributs en les initialisant.

Question 5. Ecrire une assertion
Ecrivez dans la classe `CarteMultisports` une méthode permettant d'ajouter un sport à la carte. Ecrivez une assertion vérifiant qu'à la fin de cette méthode l'ensemble de sports contient le sport que l'on a tenté d'ajouter. Ecrivez une assertion vérifiant la manière dont la taille (nombre d'éléments) de l'ensemble de sports a évolué (analysez tout d'abord de quelle manière il a pu évoluer).

Question 6. Comprendre les streams et ajouter une méthode manipulant la structure
En supposant que l'ensemble des activités de la carte se nomme `listeActivites`. On peut écrire la méthode suivante basée sur les `streams`.

```
public double cotisDisOlymp(){
    return listeActivites
      .stream()
      .filter(act -> act.disciplineOlympique())
      .mapToDouble(act -> act.getCotisation())
      .sum();
}
```

Expliquez ce qu'elle fait et réécrivez cette méthode avec une itération classique (`for` ou `while`).

Question 7. Ecrire une exception

Nous nous intéressons à présent à écrire une méthode statique `activitesCommunes` prenant en paramètre deux cartes et retournant l'ensemble constitué des activités communes. Cette méthode ne doit modifier aucune des cartes passées en paramètre. Sa pré-condition indique que l'intersection ne peut être effectué que si les deux cartes sont différentes, sinon une exception est signalée. (1) Ecrivez une classe d'exception pour représenter cette erreur, puis (2) écrivez la méthode `activitesCommunes` pour qu'elle déclare et signale cette exception lorsque la pré-condition n'est pas remplie.

Question 8. Ecrire un main

Ecrivez un main : (1) Créez trois sports de glace (2) Créez deux cartes multi-sports (3) Appelez la méthode `activitesCommunes` d'une manière normale et d'une manière qui provoque le signalement d'exception (4) Capturez la ou les exceptions qui risquent de se produire dans le programme.