amU Faculté des sciences Aix Marseille Université

Année universitaire 2024-2025

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☐ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☒ 1er semestre ☐ 2ème semestre ☐ Session 2 Durée de l'épreuve : 2h
Examen de : L3 Nom du diplôme : Licence informatique : parcours math-info
Code du module : SIN5U34 Libellé du module : Initiation au Génie logiciel
Calculatrices autorisées : NON Documents autorisés : OUI, notes de Cours, supports de cours

---

1 Gestion de groupes de coupe du monde

Le but de cet exercice est d'écrire un programme qui calcule le classement des groupes de la coupe du monde de football. Dans cet exercice, nous implémenterons seulement les trois premiers des 8 critères du règlement de la coupe du monde :

- Le plus grand nombre de points sur tous les matchs (règles des coupes du monde jusqu'à 1990 : victoire = 2 points, match nul = 1 point) ;
- La différence de buts sur tous les matchs (différences de buts = buts marqués - buts encaissés) ;
- Le plus grand nombre de buts marqués sur tous les matchs.

En d'autres termes, on regarde en premier critère les nombres de points pour départager deux équipes. Si les deux équipes ont le même nombre de points, on regarde en second critère les différences de buts des deux équipes. Si les deux équipes ont la même différence de buts, on regarde en troisième critère les nombres de buts marqués. Vous trouvez ci-dessous un exemple de classement respectant l'ordre :

Pays	Victoires	Nuls	Défaites	Points	Buts pour	Buts contre	Diff. buts
USA	2	1	0	5	7	3	4
Italie	2	1	0	5	6	2	4
Chili	0	1	2	1	2	4	-2
France	0	1	2	1	1	7	-6

Nous souhaitons pouvoir classer les pays de la façon suivante :
<MINTED>
Les classes `Team`, `Group` et `Match` sont déjà implémentées à l'exception de la méthode `sort` de la classe `Group`.
<MINTED>
<MINTED>
<MINTED>

Question 1 : Quelles parties du code doivent être modifiées pour ajouter une nouvelle méthode de calcul des points qui donne 3 points pour une victoire (et toujours 1 point pour un nul et 0 pour une défaite) ? Quel principe SOLID n'est pas respecté d'après vous ? Justifiez votre réponse.

Afin de permettre de choisir entre les deux méthodes de calcul des points (2 ou 3 points pour une victoire), on introduit l'interface suivante :
<MINTED>
La méthode de calcul des points ne sera plus dans la classe `Match` (plus de méthode `pointsFor` dans la classe `Match`) mais dans une classe implémentant `ScoreFunction` qui sera utilisé dans `Group`.
Afin de créer des objets calculant les points d'une équipe (qui seront des instances de classes implémentant `ScoreFunction`), on utilisera l'interface suivante qui sera implémentée par deux classes `PointsFunctionOldRuleFactory` (calcul avec 2 points pour les victoires) et `PointsFunctionNewRuleFactory` (calcul avec 3 points pour les victoires) :
<MINTED>
L'objectif est de permettre l'exécution du code suivant :
<MINTED>

Question 2 : Écrivez le code des classes `PointsFunctionOldRuleFactory` et `PointsFunctionNewRuleFactory` ainsi que le code de la ou les classe(s) des objets créés par les deux versions de la méthode `createPointsFunction`.

On considère l'interface `Comparator<T>` ci-dessous qui permet de comparer deux éléments de type `T`.
<MINTED>
On souhaite créer une classe `ScoreComparator` qui implémente l'interface `Comparator<Team>` afin de pouvoir comparer les équipes.
Cette classe devra posséder :

- un constructeur qui prend un tableau de matchs (de type `Match[]`) et une instance de `ScoreFunction` en paramètre ;
- implémente la méthode `compare` de sorte à retourner pour deux équipes t_1 et t_2 la valeur de la formule suivante :

$$\sum_{m \in M} (s(t_2, m) - s(t_1, m))$$

avec m le tableau de matchs et s la fonction calculée par la méthode `score` l'instance de `ScoreFunction` donnée en paramètre au constructeur.

L'objectif est de permettre l'exécution du code suivant :
<MINTED>

Question 3 : Écrivez le code de la classe `ScoreComparator`.

On souhaite pouvoir composer plusieurs critères de comparaisons. Pour cela, on va introduire une classe `CompositeComparator<T>` qui implémentera l'interface `Comparator<T>`. Cette classe aura une référence d'une liste de `Comparator<T>` et implémentera la méthode `compare(T t1, T t2)` de façon à retourner la valeur `c.compare(t1, t2)` du premier comparateur `c` dans la liste donnant une comparaison différente de `0` (ou `0` si tous les comparateurs donne une comparaison à 0).

Question 4 : Écrivez le code de la classe `CompositeComparator<T>`.

Il reste à écrire l'implémentation de la méthode `void sort()` de la classe `Group` qui trie le tableau `teams` en considérant les matchs du tableau `matches` et en composant les fonctions de score suivantes dans cet ordre :

- nombre de points avec l'ancienne règle des points (2 points par victoire et 1 par nul) ;
- différence de buts (`goalDifferenceFor`) ;
- nombre de buts marqués (`goalCountFor`).

Vous pouvez utiliser la méthode statique `Arrays.sort(T[] array, Comparator<T> comparator)` de Java qui trie le tableau `array` en respectant le comparateur `comparator`.

Question 5 : Écrivez le code de la méthode `sort` de la classe `Group`.

On souhaite pouvoir classer les groupes selon la nouvelle règle de calcul des points (3 points par victoire et 1 par nul).

Question 6 : Décrivez une réorganisation du code permettant de changer la règle de calcul des points en modifiant uniquement une seule ligne de la méthode `main`.