UE Conception Orientée Objet

Projet: Compétitions Sportives

Les compétitions

Une compétition sportive est définie par un ensemble de matchs (`Match`) joués entre compétiteurs (`Competitor`). La compétition est responsable de l'organisation de ses matchs. Autrement dit, c'est elle qui fait *jouer* ses matchs. La méthode `play()` déroule donc une compétition jusqu'à son terme, c'est à dire lorsque tous les matchs ont été joués et qu'un vainqueur est désigné. Organiser une compétition consiste donc à faire jouer successivement chaque match de la compétition selon les règles suivantes:

- il y a forcément un gagnant à chaque match. Considérant que chaque compétiteur a les mêmes chances de remporter un match que son adversaire, le gagnant d'un match est obtenu de manière aléatoire et marque un point. Il n'est donc pas demandé de prendre en compte les différents types de match, les différentes façons de gagner un match, ni de gérer les situations d'égalité. Il n'est pas non plus demandé de gérer le déroulement simultané de plusieurs matchs (il n'y a pas de notion de *calendrier*).
- les championnats (`League`) se jouent en matchs aller/retour. Chaque compétiteur rencontre donc 2 fois chacun des autres compétiteurs du championnat. A l'issue du championnat, le vainqueur est donc le compétiteur ayant cumulé le plus grand nombre de victoires.
- les tournois (`Tournament`) à élimination directe se déroulent sur plusieurs tours. Seuls les compétiteurs ayant gagné leur match lors d'un tour participent au tour suivant. Les vainqueurs de chaque match se rencontrent ainsi entre eux jusqu'à ce qu'il n'en reste plus qu'un, déclaré alors vainqueur du tournoi. Ce dernier a donc obtenu autant de victoires que le nombre de tours joués... L'organisation de tels tournois nécessite un nombre de compétiteurs puissance de 2.

Mise en oeuvre

▷ Dans la *première version* de l'application, il y a donc deux types de compétition et un seul type de match (celui dont le gagnant est choisi aléatoirement). Par la suite, on doit cependant avoir la possibilité d'envisager d'autres types de compétitions, voire même d'autres types de matchs (matchs "pondérés" où un compétiteur part favori, matchs truqués, etc.).

▷ Une compétition est terminée lorsque l'ensemble de ses matchs a été joué. Lorsque tous les matchs ont été joués, le classement (`ranking()`) de la compétition peut donc être établi pour afficher les compétiteurs et leur nombre de points (victoires).

≪ abstract ≫ *Competition*
- match : Match - final competitors : List<Competitor>
≪constructor≫ + Competition(List<Competitor>) ≪methods≫ + play() # *play*(List<Competitor>) # playMatch(c1 : Competitor, c2 : Competitor) + ranking() : Map<Competitor, Integer>

Ci-contre, un extrait du diagramme UML de la classe `Competition`. Le constructeur prend en paramètre une Liste de compétiteurs. Il faut donc au préalable créer les différents compétiteurs du jeu pour qu'ils puissent participer à la compétition.

L'appel de la **méthode** `play()` permet d'organiser les matchs entre les compétiteurs et de les "jouer" (`playMatch()`).

Les Masters

On ajoute un nouveau type de compétition, les `Master`, qui font s'affronter les compétiteurs en deux phases. La compétition commence par une phase de poules à l'issue de laquelle certains compétiteurs sont sélectionnés pour disputer la phase finale du tournoi. Les poules sont organisées sour forme de championnat tandis que la phase finale se déroule sous la forme d'un tournoi à élimination directe.

La méthode de sélection des compétiteurs retenus pour disputer la phase finale du Master est précisée à la création du Master, et peut varier d'un Master à un autre. Par exemple, une première compétition de type Master peut répartir 16 joueurs en 4 poules de 4 joueurs et ne conserver que les meilleurs de chaque poule pour la seconde phase. Un second Master peut lui considérer 24 compétiteurs répartis en 3 groupes de 8, pour ne sélectionner pour la phase finale que les 2 premiers de chaque poule auxquels s'ajoutent les 2 meilleurs troisièmes, toutes poules confondues. Bien entendu, d'autres méthodes pour sélectionner les compétiteurs entre la première et la seconde phase peuvent être envisagées : deux premiers de chaque groupe, meilleurs seconds, repêchage du dernier de chaque poule, etc. Quelle que soit la méthode de sélection utilisée, le nombre de compétiteurs accédant en phase finale doit être une puissance de 2.

Le vainqueur du Master est le compétiteur finissant vainqueur de la phase finale. Ce n'est donc pas nécessairement le compétiteur qui a le plus de victoires qui remporte le tournoi, puisque un compétiteur peut terminer premier de sa poule en ayant moins de points que le premier d'une autre poule. L'affichage du classement d'un Master fait donc apparaitre le classement de chacune des poules, ainsi que le classement de la phase finale.

Q 1 . Proposez une conception, un programme et la suite de tests associée qui reprennent l'ensemble des fonctionnalités présentées dans ce sujet. Plusieurs méthodes de sélection (au choix) devront être proposées dans votre conception.

Annexes - La classe `util.MapUtil`

Cette classe utilitaire propose une méthode permettant de classer une Map<Clé, Valeur> dans l'ordre décroissant des valeurs, à condition que ces valeurs soient des entiers. Le fichier est disponible sur le portail.

```
package util;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

public class MapUtil {

  public static <K, V extends Comparable<? super V>> Map<K, V> sortByDescendingValue(Map<K, V> map) {

    List<Entry<K, V>> sortedEntries = new ArrayList<Entry<K, V>>(map.entrySet());

    Collections.sort(sortedEntries, new Comparator<Entry<K, V>>() {
      @Override
      public int compare(Entry<K, V> e1, Entry<K, V> e2) {
        return e2.getValue().compareTo(e1.getValue());
      }
    });

      Map<K, V> result = new LinkedHashMap<>();
      for (Entry<K, V> entry : sortedEntries) {
          result.put(entry.getKey(), entry.getValue());
      }

      return result;
  }
}
```

Un exemple de trace d'exécution

```
Blastoff vs Drift --> Blastoff wins!
Blastoff vs Lynx --> Lynx wins!
Blastoff vs Catalyst --> Catalyst wins!
Blastoff vs Raven --> Raven wins!
Blastoff vs Midas --> Midas wins!
Drift vs Blastoff --> Blastoff wins!
Drift vs Lynx --> Lynx wins!
Drift vs Catalyst --> Drift wins!
Drift vs Raven --> Drift wins!
Drift vs Midas --> Midas wins!
Lynx vs Blastoff --> Lynx wins!
Lynx vs Drift --> Drift wins!
Lynx vs Catalyst --> Lynx wins!
Lynx vs Raven --> Lynx wins!
Lynx vs Midas --> Lynx wins!
Catalyst vs Blastoff --> Catalyst wins!
Catalyst vs Drift --> Drift wins!
Catalyst vs Lynx --> Lynx wins!
Catalyst vs Raven --> Catalyst wins!
Catalyst vs Midas --> Midas wins!
Raven vs Blastoff --> Raven wins!
Raven vs Drift --> Drift wins!
Raven vs Lynx --> Raven wins!
Raven vs Catalyst --> Raven wins!
Raven vs Midas --> Raven wins!
Midas vs Blastoff --> Blastoff wins!
Midas vs Drift --> Midas wins!
Midas vs Lynx --> Lynx wins!
Midas vs Catalyst --> Catalyst wins!
Midas vs Raven --> Raven wins!

*** Ranking ***
Lynx - 8
Raven - 6
Drift - 5
Catalyst - 4
Midas - 4
Blastoff - 3
```