Université Lille1 Licence mention Informatique Semestre 4

UE Programmation Orientée Objet
Examen première session – 26 mai 2012
2 heures - notes et polycopiés de cours/TD/TP autorisés
livres, calculatrices et appareils de communication interdits
dictionnaires de langue autorisés

Exercice 1 : London 2012

Sauf indication contraire, tous les types définis dans ce sujet appartiendront au paquetage `london2012`.

Les organisateurs des jeux olympiques LONDON 2012 souhaitent informatiser la gestion de cet événement. Dans ce sujet on s'intéresse en particulier aux épreuves[1] et à la gestion de leurs résultats.

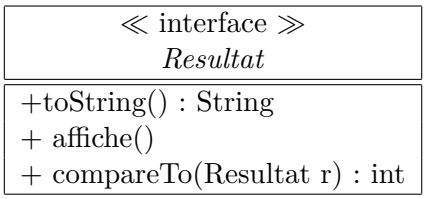
Athlètes Les jeux olympiques réunissent des athlètes issus de très nombreux pays. La liste de ces pays est fixée et connue à l'avance.

Q 1 . Comment proposez-vous de représenter les pays ? Donnez le code java correspondant à votre proposition en supposant qu'il n'y a que 3 pays impliqués : la France, la Belgique et l'Angleterre.

Les athlètes sont identifiés par leur nom, leur prénom et le pays pour lequel ils concourrent.

Q 2 . Donnez le code java d'une classe `Athlete` pour laquelle on disposera des méthodes accesseurs, d'une méthode `toString` reprenant les informations et une méthode `equals`, deux athlètes étant égaux s'ils ont même nom, prénom et nationalité.

Résultats Les résultats des athlètes sont représentés par des objets du type `Resultat` :



La méthode `toString` fournit le résultat sous forme de chaîne de caractères et la méthode `affiche` l'affiche sur la sortie standard.
Selon les épreuves la nature du résultat change. Il peut se mesurer :

- par un temps, c'est par exemple le cas des courses, une telle donnée est modélisée par une instance de la classe `ResultatTemps` ;
- par une distance, c'est par exemple le cas des lancers ou des sauts, une telle donnée est modélisée par une instance de la classe `ResultatDistance` ;
- par un nombre de points, comme c'est le cas en plongeon, en tir ou au décathlon, une telle donnée est modélisée par une instance de la classe `ResultatPoints` ;
- par un rang dans un classement, comme c'est le cas au tennis de table ou pour les sports de combat, une telle donnée est modélisée par une instance de la classe `ResultatRang`.

Les quatre classes de résultats présentées ci-dessus sont définies dans le paquetage `london2012.resultat`.
Les résultats de même nature sont comparables deux à deux, les classes ci-dessus implémentent donc également l'interface `Comparable` (cf. Annexe). La méthode `compareTo` qui définit la relation d'ordre permet alors de déterminer si un résultat est ou non meilleur qu'un autre.

[1] On ne s'occupera ici que des épreuves individuelles

Q 3 . Donnez un code java pour la classe `ResultatDistance`. Un tel résultat est caractérisé par la distance réalisée par l'athlète, cette donnée est représentée par un flottant.
Un tel résultat est meilleur qu'un autre si sa distance est plus grande.

Q 4 . Sachant que la déclaration et l'affectation suivante sont valides (elles compilent) :

```
T res;
res = new ResultatDistance(6.01f);
```

Quelles sont **toutes** les valeurs possibles pour le type `T` ?

Q 5 . Donnez un code java pour la classe `ResultatRang`. Un tel résultat est caractérisé par le rang de classement de l'athlète, cette donnée est représentée par un entier.
Un tel résultat est meilleur qu'un autre si son rang est inférieur.

Epreuves. Les épreuves sont modélisées par des instances de la classe `Epreuve`.

Ces objets permettent de gérer les informations concernant les athlètes participant à l'épreuve et leur résultat à celle-ci. Ces informations sont stockées dans une table de hachage dont les clés sont les athlètes et les valeurs les résultats.

Une épreuve peut-être ou non déclarée close (ou terminée). Lorsque c'est le cas les résultats sont définitifs.

Les méthodes accessibles pour des épreuves permettent :

- (méthode `toString`) de connaitre sa dénomination ("`100m haies féminin`", "`Tennis de table masculin`", etc.),
- (méthode `ajouteAthlete`) l'ajout d'un participant à l'épreuve,
- (méthode `estClose`) de savoir si une épreuve est terminée ou non,
- (méthode `cloture`) la cloture d'une épreuve,
- (méthode `getRecordOlymique`) de connaître le record olympique en vigueur **avant** le déroulement de l'épreuve, cette donnée est de type `Resultat`,
- (méthode `fixeResultat`) l'ajout ou mise à jour d'un résultat pour un participant de l'épreuve, rien ne se passe si le participant n'était pas inscrit à l'épreuve, une exception `IllegalStateException` est déclenchée si l'épreuve a été déclarée close,
- (méthode `getResultat`) de connaître le résultat d'un athlète à l'épreuve, la valeur `null` est renvoyée si l'athlète ne participe pas à l'épreuve ou si aucun résultat n'a été ajouté pour lui,

A la construction d'une épreuve, seule sa dénomination et le record olympique en vigueur (lorsqu'il a un sens[2]) sont communiqués.

Q 6 . Qu'est-il nécessaire d'ajouter dans les définitions faites lors des questions précédentes pour permettre la bonne gestion des résultats (soyez précis et concis) ?
Donnez le code java correspondant à votre proposition.

Q 7 . Donnez un code java pour la classe `Epreuve` satisfaisant le cahier des charges ci-dessus.

Q 8 . On suppose définies une référence `epreuve` de type `Epreuve` et une référence `athlete` de type `Athlete`.
Donnez les lignes de code java qui permettent de fixer le résultat distance de cet athlète pour cette épreuve à la valeur `2.43f`, un message `"épreuve close"` est affiché si l'épreuve est close.

On souhaite ajouter quelques fonctionnalités à la classe `Epreuve` afin de gérer les médaillés.

Q 9 . Le vainqueur d'une épreuve est celui qui a obtenu le meilleur résultat. La médaille d'or lui est alors décernée[3].
Donnez la javadoc et le code java d'une méthode `getMedailleOr` qui a pour résultat l'athlete vainqueur de l'épreuve une fois qu'elle est déclarée close, dans le cas contraire une exception `IllegalStateException` est levée.

[2] pour les épreuves dont le résultat est de type rang, cette notion n'a pas réellement de sens, sa valeur sera alors mise à `null`

[3] On suppose que le règlement sportif empêche la possibilité d'ex-aequo

Pour la suite on supposera que des méthodes similaires `getMedailleArgent` et `getMedailleBronze`[4] ont été définies.

Les Jeux Les jeux olympiques sont gérés par la classe `JeuxOlympiques` dont chaque instance est caractérisée par l'année du déroulement de ces jeux (un `int`), la ville organisatrice (un `String`) et la liste des objets `Epreuve` qui la composent (de type `List<Epreuve>`).

Q 10 . Donnez le code java d'une méthode `nbRecordsBattus` qui indique le nombre d'épreuves terminées pour lesquelles le record olympique a été battu par le vainqueur au cours de l'épreuve.

Q 11 . Un classement entre les pays participant est réalisé. Pour cela on attribue un nombre de points pour chaque médaille d'or, d'argent ou de bronze, par exemple 10, 3 et 1. Ces valeurs sont définies par des constantes dans la classe `JeuxOlympiques`.

Donnez le code java de déclarations de ces constantes ainsi que celui d'une méthode `getScore` dont la javadoc est :

```
/**
 * calcule le score d'un pays en fonction du nombre de médailles obtenues
 * par ses athlètes au cours des épreuves terminées de ces jeux
 * @param pays le pays concerné
 * @return le score de ce pays
 */
```

Annexe

java.lang
public interface Comparable<T>

compareTo

```
int compareTo(T o)
```

Parameters : `o` - the object to be compared.

Returns : a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Throws : `ClassCastException` - if the specified object's type prevents it from being compared to this object.

[4] On ne considèrera pas les épreuves ou plusieurs médailles de bronze sont décernées