TP 02

Exercice 1 : Gestion des concurrents

On souhaite organiser une compétition de roller avec au maximum 100 participants. Cette compétition repose sur deux épreuves :

— une épreuve artistique pour laquelle un nombre de points est accordé selon les figures réalisées ;
— une épreuve de vitesse pour laquelle le temps de parcours est chronométré.

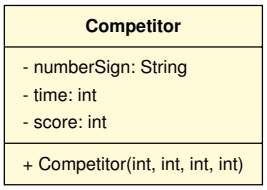
Un concurrent est caractérisé par un numéro de dossard, un score compris entre 0 et 50 pour l'épreuve artistique ainsi qu'un temps exprimé en secondes pour l'épreuve de vitesse.

Q1. Écrivez la classe `Competitor` munie de trois attributs et d'un constructeur paramétré répondant à la signature suivante, tout en respectant les contraintes énoncées :

— le numéro de dossard de type entier doit être converti en une chaîne de caractères de la forme $NoXX$ (où XX désigne le numéro du dossard) pour renseigner l'attribut `numberSign` ;
— le temps fourni en minutes et en secondes devra être converti en secondes pour renseigner l'attribut `time` ;
— le numéro de dossard doit être compris entre 1 et 100, le score entre 0 et 50 et les nombres de minutes et de secondes entre 0 et 60. Si un des paramètres est invalide (en dehors de son domaine de valeurs), l'attribut `numberSign` restera `null`.

```
Competitor(int numberSign, int score, int min, int sec)
```

Autrement dit, votre classe doit respecter la structure suivante :



Q2. Écrivez la méthode `display` qui renvoie le concurrent sous la forme : `[No10,12 points,180 s]` ou `[<invalide>, 12 points, 180 s]` le cas échéant.

```
String display()
```

Q3. Ajoutez une méthode principale `main`, qui crée un tableau d'inscrits et les affiche. Ce tableau doit pouvoir contenir jusqu'à 100 concurrents, mais ne contient que les valeurs suivantes : `(1,45,15,20)`, `(2,32,12,45)`, `(5,12,13,59)`, `(12,12,15,70)` et `(32,75,15,20)`. L'affichage doit mettre un concurrent par ligne pour obtenir le résultat suivant (Attention à la gestion des cas invalides pour lesquels la méthode d'affichage ne doit pas être invoquée) :

```
[No1,45 points,920 s]
[No2,32 points,765 s]
[No5,12 points,839 s]
```

Q4. Écrivez une méthode `equals(...)` basée sur les attributs `numberSign` et `score`.

Q5. Récupérez le fichier `UseCompetitor.java` sous Moodle et copiez-le dans votre répertoire `tp02`. Exécutez-le pour déterminer si votre méthode `equals` fonctionnement correctement.

Q6. Ajoutez à la classe `Competitor` une méthode `isFaster(...)` qui détermine si un compétiteur passé en paramètre est plus rapide.

```
boolean isFaster(Competitor other)
```

Q7. Complétez votre méthode principale pour afficher tous les compétiteurs plus rapides que le premier.

Exercice 2 : Jeu de dé

On souhaite réaliser un jeu de dé relativement simple : un joueur lance un dé jusqu'à atteindre un total cumulé de 20. Son score correspond au nombre de lancers nécessaires pour y arriver. Un dé est caractérisé par son nombre de faces et la valeur qu'il renvoie. Un joueur est caractérisé par son nom, son nombre de lancers et le cumul des valeurs obtenues lors des lancers.

Q1. Écrivez la classe `Dice` constituée de trois attributs de visibilité privée (le nombre de faces `numberSides`, le générateur aléatoire `rand`, la valeur courante du dé `value`) et répondant aux signatures suivantes :

Class `Dice`

	`Dice(int)` Crée un dé caractérisé par son nombre de faces.
`void`	`roll()` Effectue un lancer et modifie la valeur du dé.
`String`	`toString()` Renvoie la valeur du dé sous forme textuelle.

Votre constructeur doit gérer le cas où le nombre de faces fourni est strictement positif, sinon le nombre de faces vaudra par défaut 1. On notera de plus que la valeur initiale que porte un dé est déterminée grâce à un lancer.

Q2. Écrivez la classe `DicePlayer` caractérisée par un nom `name`, un total cumulé `totalValue` ainsi qu'un nombre de lancers `nbDiceRolls`. Munissez cette classe d'un constructeur prenant le nom du joueur en paramètre.

```
DicePlayer(String name)
```

Q3. Écrivez la méthode `play` qui prend en paramètre le dé utilisé pour jouer, ajoute au cumul le résultat du lancer (au singulier) et qui incrémente le nombre de lancers effectués par le joueur.

```
void play(Dice dice6)
```

Q4. Munissez cette classe `DicePlayer` d'une méthode `toString` permettant l'affichage suivant : "`Alice: 22 points en 8 coups.`"

```
public String toString()
```

Q5. Écrivez une classe `OneDicePlayerGame` implémentant le jeu désiré, afin qu'un joueur lance un dé jusqu'à obtenir au moins 20 points et affiche le résultat.

Exercice 3 : Jeu à deux joueurs

On souhaite disposer d'une version qui permet à deux joueurs de s'affronter à ce jeu de dé.

Q1. Écrivez une classe `TwoDicePlayerGame` caractérisée par deux joueurs, munie d'un constructeur les prenant en paramètre.

Q2. On souhaite déterminer qui est le vainqueur, en sachant que le gagnant est le joueur ayant effectué le moins de lancers pour atteindre l'objectif. En cas d'égalité, c'est-à-dire en cas de nombre de lancers équivalents pour dépasser le seuil, les cumuls sont comparés pour déterminer le vainqueur.

Q2.1. Ajoutez à la classe `DicePlayer` une méthode `isWinning` dans ce but.

Q2.2. Ajoutez à la classe `TwoDicePlayerGame` une méthode `winner` qui respecte la signature suivante:

```
DicePlayer winner()
```

Q3. Ajouter une méthode principale à cette classe afin de dérouler une partie et afficher le nom du vainqueur. Un joueur effectue tous ses jets de dés à la suite (jusqu'à atteindre le seuil visé), avant de laisser lancer l'autre joueur.

```
Le gagnant est Alice: 25 points en 5 coups.
```

Exercice 4 : Jeu à plusieurs 👍

On souhaite étendre ce jeu, dans une classe `NDicePlayerGame`, à un nombre quelconque de joueurs stockés dans un tableau de `DicePlayer`.

Q1. Écrivez la classe `NDicePlayerGame` avec un constructeur qui prend en paramètre le nombre de joueurs à créer et les initialise avec un nom correspondant à leur numéro d'ordre.

Q2. Écrivez une méthode `winner` renvoyant le ou les vainqueurs du jeu (ceux ayant le nombre de lancers le plus faible, peu importe la valeur totale atteinte).

```
DicePlayer[] winner()
```

Q3. Ajoutez une méthode principale, qui initialise une partie à partir d'un nombre de joueurs récupéré en paramètre du programme principal. Chaque joueur effectue tous ses lancers avant de laisser la place au suivant. Le programme termine en affichant les vainqueurs.

```
Les vainqueurs:
0: 23 points en 8 coups.
3: 20 points en 8 coups.
```