

## 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.** Concentrons-nous sur la structure de base de la classe `Competitor`.

**Q1.1.** Écrivez la classe `Competitor` munie de trois attributs et d'un constructeur répondant à la signature suivante, tout en respectant les contraintes précédemment 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` alors que les autres attributs `time` et `score` sont affectés normalement.

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

Autrement dit, votre classe doit respecter la structure suivante :

Competitor
- numberSign : String
- time : int
- score : int
+ Competitor(int, int, int, int)

**Q1.2.** Écrivez la méthode `toString` qui retourne le concurrent en une chaîne de caractères respectant la forme : `(<numberSign>,<score> points, <time> s)`. En cas d'attribut `numberSign` de valeur `null`, la méthode retourne `(-invalide-, <score> points, <time> s)`.

```
public String toString()
```

**Q1.3.** Créez une classe `UseCompetitor` contenant 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). Ajoutez les *getters* que vous jugerez nécessaires.

```
(No1, 45 points, 920 s)
(No2, 32 points, 765 s)
(No5, 12 points, 839 s)
```

**Q2.** Travaillons sur l'égalité entre instances de `Competitor`.

**Q2.1.** Écrivez une méthode `equals` dans votre classe `Competitor`, basée sur les attributs `numberSign` et `score` uniquement. Veillez à respecter la structure standard de cette méthode.

```
boolean equals(Competitor other)
```

**Q2.2.** Récupérez le fichier `UseCompetitor2.java` sous Moodle et copiez-le dans votre répertoire `tp02`. Exécutez-le pour déterminer si votre méthode `equals` fonctionne correctement.

**Q3.** Cherchons enfin à identifier qui sont les plus rapides !

**Q3.1.** Ajoutez à la classe `Competitor` une méthode `isFaster` qui détermine si un compétiteur passé en paramètre est plus rapide.

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

**Q3.2.** Complétez votre méthode principale pour afficher tous les compétiteurs plus rapides que qu'un concurrent initialisé avec les valeurs : (7, 42, 13, 50).

## Exercice 2 : Jeu de dés

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.** Concentrons-nous d'abord sur un dé et son utilisation.

**Q1.1.** 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 :

Dice
- numberSides : int - rand : Random - value : int
+ Dice(int) roll() : void ...

### Class Dice

	<code>Dice(int)</code> Crée un dé caractérisé par son nombre de faces.
<code>void</code>	<code>roll()</code> Effectue un lancer et modifie la valeur du dé.
<code>String</code>	<code>toString()</code> 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.

**Q1.2.** Écrivez une classe `UseDice`, munie d'un programme principal, exécutant le scénario suivant :

- création d'un dé à 6 faces
- faire 20 lancers dont vous afficherez les résultats au fur et à mesure (tous doivent être compris entre 1 et 6 inclus)
- création d'un dé à 10 faces
- faire 20 lancers dont vous afficherez les résultats au fur et à mesure (tous doivent être compris entre 1 et 10 inclus)

**Q2.** Intéressons-nous maintenant au joueur.

**Q2.1.** 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
- name : String - nbDiceRolls : int - totalValue : int
+ DicePlayer(String) ...

**Q2.2.** Ajoutez à votre classe les *getters* pour les trois attributs.

**Q2.3.** Ajoutez une méthode `toString` permettant l'affichage suivant : "<name>: <totalScore> points en <nbDiceRolls> coups."

**Q2.4.** Écrivez la méthode `play` qui prend en paramètre le dé utilisé pour jouer, ajoute au cumul le résultat du lancer et qui incrémente le nombre de lancers effectués par le joueur. Cette méthode correspond à un unique lancer.

```
void play(Dice aDice)
```

**Q2.5.** Ajoutez une méthode `playUntil` qui permet au joueur de lancer le dé fournit en paramètre, jusqu'à ce que `totalValue` excède l'objectif passé lui-aussi en paramètre.

```
void playUntil(Dice aDice, int objective)
```

**Q2.6.** Écrivez une classe `OneDicePlayerGame` implémentant le jeu désiré dans une méthode principale, afin qu'un joueur lance un dé jusqu'à obtenir au moins 20 points et affiche le résultat.