

TP 3 : Les interactions entre les joueurs et les pokemons !



Aujourd'hui notre but sera de continuer de construire les interactions entre les pokemons et leurs joueurs. Les joueurs pourront collectionner de la nourriture, qu'ils pourront donner à leurs pokemons. De plus, aujourd'hui nous allons finalement permettre aux usagers d'interagir avec le programme.

Lien utile : <https://pokemondb.net/pokedex/all> .

Exercice 1

Nous allons commencer par modifier la classe Pokemon. Chaque pokemon aura un score qui indiquera s'il a faim et s'il est loyal à son maître actuel. Plus tard, ces scores vont impacter la capacité du pokemon de lutter et de progresser.

- Nous allons donc premièrement définir ces attributs, qui seront très simples : des scores entre 0 et 100 (ce sont donc quel type d'attributs ?). Les attributs s'appelleront : `appetit`, `loyaute`.
- Réharmonisez vos constructeurs pour prendre en compte le fait qu'un pokemon est toujours créé avec : `appetit = 50`, `loyaute = 0`.
- Pour manipuler ces attributs vous aurez besoin de méthodes qui vous donnent les valeurs stockées par ces attributs (des méthodes `getAppetit()`, `getSatisfaction()`), ainsi que des méthodes qui permettent de monter ou de baisser la valeur d'un attribut par une valeur donnée en entrée : (par exemple `void baisserAppetit(int difference)`, `void monterAppetit(int difference)`, `void baisserLoyaute(int difference)`, `void monterLoyaute(int difference)`). Attention : il faut prendre en compte que les valeurs de l'appétit, de la satisfaction et de la loyauté d'un pokemon ne peuvent jamais sortir de l'intervalle 0-100 !
- Mettez à jour votre méthode `toString()` qui devra désormais également retourner les trois nouveaux attributs (au-delà des valeurs qu'elle rend déjà).
- Dans la classe `Joueur` modifiez les méthodes `capturerPokemon`, `libererPokemon` et `donnerNom` pour prendre en compte ces règles de base :
 - *Capturer* ou *libérer* un pokemon réduit sa loyauté à 0 et baisse son appetit à 10 (il refuse de manger).
 - La première fois qu'un nouveau maître *donne un nom* à un pokemon, sa loyauté monte par 10 points (ou elle est mise à 100 si le pokemon avait une loyauté supérieure à 90).

- Si le même maître *choisit après un autre nom* pour un pokemon qui a déjà un nom, sa loyauté baisse par 10 points (ou elle est mise à 0 si le pokemon avait une loyauté inférieure à 10).

Exercice 2

Nos pokemons devront manger. Créez une classe de base, qui s'appellera `Nourriture`. Lors du prochain TP, nous allons utiliser l'héritage pour construire un nombre d'Ingrédients qui hériteront de la classe `Nourriture`. Mais, pour le propos de ce TP, nous allons utiliser le(s) constructeur(s) de la classe `Nourriture` pour définir des divers types d'ingrédients, qui pourront nourrir les pokemons.

- Définissez une nouvelle classe `Nourriture` avec les attributs suivants :
 - Une variable appelée `apport`, qui stockera à quel point la nourriture soulage l'appétit d'un pokemon (quel type de variable sera celle-là, alors ?)
 - Une variable de type `String` qui stockera le nom de l'ingrédient.
 - Une variable appelée `compatibilites`, qui sera de type `String[]` : elle stockera les types de pokemon (plante, eau, etc.) qui aiment bien manger cette nourriture. Quelle taille maximale aura ce `String[]` ?
 - Une variable de type `int` appelée `frequence`, qui indique la fréquence avec laquelle on peut trouver des divers types de nourriture.
- Nous utiliserons un constructeur avec la signature :

```
Nourriture(int, String, String[], int)
```

Ce constructeur met les valeurs des attributs aux valeurs des paramètres en entrée (dans l'ordre mentionnée ci-dessus).

- Écrivez une méthode `toString()` qui affiche les attributs de chaque objet de la classe `Nourriture` dans le format :

```
[<nom> ; <apport> ; <frequence>/100 ; {<types compatibles separees par une vergule>}]
```

- Il va falloir qu'on puisse savoir si un certain pokemon est compatible avec une nourriture. Nous écrirons une méthode boolean `isCompatible(Pokemon pokemon)` qui retourne `true` si le type de pokemon est parmi les types nommés dans le tableau de compatibilités, et `false` autrement.

Exercice 3

Le but de cet exercice sera de générer des objets de type nourriture, d'afficher leurs caractéristiques et puis de tester leur compatibilité avec les trois pokemons qu'on a déjà créés. Le code ci-dessous sera dans la méthode `public static void main`.

- Créez un type de nourriture qui s'appelle "tartiflette", avec un apport de 35. Ce type de nourriture sera compatible avec les prochains types de pokemons : Dragon, Feu, Combat, Normal, Eau, Electrique. Il a une fréquence de 20.
- Créez un type de nourriture qui s'appelle "ratatouille", avec un apport de 10. Il est compatible avec les prochains types de pokemons : Plante, Eau, Vol, Feu, Normal, Electrique, Combat et a la fréquence de 50.
- Déclarez une variable `typesDeNourriture` de type `Nourriture[]` dans laquelle vous allez stocker la tartiflette (sur la première position) et la ratatouille (sur la deuxième position). Utilisez l'instanciation vue en CM et TD, notamment :

```
Nourriture[] typesDeNourriture = new Nourriture[] {<l'objet tartiflette>, <l'objet ratatouille>};
```

- Faites afficher les caractéristiques de ces deux types de nourriture.
- Testez la compatibilité de ces deux types de nourriture avec les pokemons `piplup` et `rowlet`.

Exercice 4

Nous allons vouloir générer de la nourriture aléatoirement. A ce propos, nous aurons besoin premièrement d'une méthode dans la classe `Nourriture`, qui rendra un objet de cette classe `Nourriture` (ou `null`, en fonction d'un paramètre en entrée).

- Dans la classe `Nourriture` écrivez une méthode avec la signature `Nourriture genererMemeNourriture(boolean generer)`. Si le paramètre en entrée de cette méthode est `false`, alors la méthode rendra `null`. Si le paramètre est `true`, alors on rendra un nouvel objet de type `Nourriture` avec les mêmes attributs que l'objet actuel (`this`).
- Pour générer de la nourriture aléatoirement il va falloir pouvoir générer un nombre aléatoire entre 0 et 100. La classe `Math` nous donne cette possibilité en Java avec la méthode `random()`, qui rend une valeur de type `double` entre 0 et 1. Utilisez l'instruction : `System.out.println(Math.random())` ; pour afficher un de ces nombres aléatoires.
- Pour notre jeu nous voulons obtenir une valeur entre 0 et 100. Pour faire cela il faudra multiplier la valeur aléatoire obtenue de la méthode `random()` par 100. Faites maintenant générer et afficher une telle valeur.
- Faites en sorte que, si la valeur aléatoire que vous générez est inférieure à la valeur de l'attribut `frequence` d'une tartiflette, alors on affiche un message nous informant de ce fait.

- Changez votre code : si la valeur aléatoire est inférieure à la valeur de l'attribut `frequence` d'une tartiflette, utilisez votre méthode `genererMemeNourriture` de la classe `Nourriture` pour rendre un objet avec les mêmes paramètres que la tartiflette. Affichez cet objet.
- Dans la classe `ChasseAuxPokemons`, dans la méthode principale, nous allons faire une extension de notre code pour obtenir une façon d'itérer 10 fois un essai de générer une valeur aléatoire, de vérifier si l'aléa généré est inférieur à la valeur de la fréquence de la tartiflette et de la ratatouille -- si c'est le cas, on génère des nouveaux objets avec les mêmes paramètres, sinon, on génère `null`. A chaque itération on affiche : le nombre de l'itération et les objets créés.

[Le petit extra]

- Au lieu d'itérer votre code 10 fois, nous allons l'itérer un nombre de fois qui sera pris en entrée du paramètre `args` de la méthode `main`. Notamment, nous allons utiliser la syntaxe `Integer.valueOf(args[0])` ; pour transformer la valeur `String` en entrée (`args[0]`) dans un entier, et mettre cette valeur comme le nombre d'itérations de la boucle qui génère la nourriture.
- Faites exécuter votre code pour un nombre variable d'itérations, en changeant les paramètres de l'exécution