M2103-- TP3

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 .

Préambule

Vous devriez entrer dans la salle de TP avec un programme correct et complet pour le dernier TP (le TP2). Je souligne encore une fois : c'est votre responsabilité d'avoir un tel programme, en s'assurant que le code fourni pour la correction marche pour votre machine.

Créez un nouveau projet TP3 et dedans un package tp3. Dans le package tp3 copiez le code (correct) du TP 2 en utilisant les instructions données lors du TP2.

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.

1. Ajoutez deux attributs de type `int`, qui auront des scores entre 0 et 100. Les attributs s'appelleront : `appetit`, `loyaute`.

2. Réharmonisez vos constructeurs pour prendre en compte le fait qu'un pokemon est toujours créé avec : `appetit = 50, loyaute = 0`.

3. Ecrivez des getters pour ces deux attributs.

4. Ecrivez une méthode `void baisserAppetit(int difference)` qui baisse la valeur actuellement stockée par l'attribut `appetit` par l'entier en entrée (`difference`). Puis écrivez une méthode `void monterAppetit(int difference)`, qui fait l'inverse, en montant la

valeur stockée par `appetit` par la valeur `difference`. Attention : l'appétit d'un pokemon doit rester dans l'interval 0-100 !

5. Ecrivez deux méthodes `void baisserLoyaute(int difference)`, `void monterLoyaute(int difference)`) qui sont les équivalents des méthodes ci-dessus mais pour l'attribut `loyaute`.

6. Mettez à jour votre méthode `toString()` qui devra désormais également retourner les deux nouveaux attributs (au-delà des valeurs qu'elle retourne déjà).

7. Les attributs `appetit` et `loyaute` seront influencés par l'état d'esprit du pokemon. 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 loyaute monte par 10 points (ou elle est mise à 100 si le pokemon avait une loyaute 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 loyaute 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.

1. Créez une nouvelle classe `Nourriture` dans le package tp3.
2. Les objets de type `Nourriture` auront les attributs suivants :
    - Une variable appelée `apport`, qui stockera à quel point la nourriture soulage l'appétit d'un pokemon (`apport` sera de quel type alors ?)
    - Une variable nom de type `String` qui stockera le nom de la nourriture.
    - Une variable `compatibilites` de type `String[]`, qui stockera les types de pokemon (plante, eau, etc.) qui aiment bien manger cette nourriture. Quelle sera la taille maximale de ce `String[]` ?
    - Une variable de type `int` appellée `frequence`, qui indique la fréquence avec laquelle on peut trouver des divers types de nourriture.

3. Nous utiliserons un constructeur avec la signature :
    ```
    Nourriture(int, String, String[], int)
    ```

Ce constructeur initialise chaque attribut à la valeur donnée en paramètre (dans l'ordre mentionnée ci-dessus).

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

   ```
   Nourriture [<nom> ; <apport> ; <frequence>/100 ; {<types compatibles separes par une virgule>}]
   ```

5. Il va falloir qu'on puisse savoir si un certain pokemon est compatible avec une nourriture. Ecrivez 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` de la classe `ChasseAuxPokemons`.

1. 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`.

2. 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`.

3. Déclarez une variable `diversesNourritures` 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[] diversesNourritures = new Nourriture[]{<l'objet tartiflette>, <l'objet ratatouille>};
   ```

4. Faites afficher les caractéristiques de ces deux types de nourriture.

5. Testez la compatibilité de ces deux types de nourriture avec les pokemons `piplup` et `rowlet`.

Exercice 4

Nos joueurs pourront rencontrer de la nourriture aléatoirement dans leur jeu. Nous allons vouloir donc pouvoir générer de la nourriture. A ce propos, nous aurons besoin premièrement d'une méthode dans la classe `Nourriture`, qui retournera un objet de cette classe (ou `null`, en fonction d'un paramètre en entrée). Le code dans cet exercice sera pour la plupart dans la classe `ChasseAuxPokemons`, mais aussi dans la classe `Nourriture`.

1. 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 retournera `null`. Si le paramètre est `true`, alors elle retournera un nouvel objet de type `Nourriture` avec les mêmes attributs que l'objet actuel (`this`).

   Attention : on ne retourne pas `this`, on retourne un objet avec les mêmes attributs !

2. 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 retourne une valeur de type double entre 0 et 1. Dans la méthode `main` de la classe `ChasseAuxPokemons`, utilisez l'instruction : `System.out.println(Math.random())` pour afficher un de ces nombres aléatoires.

3. 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 dans la méthode `main`.

4. Faites en sort que, si la valeur aléatoire que vous générez est inférieure à la valeur de l'attribut `frequence` de l'objet `tartiflette` que vous avez généré, alors on affiche un message nous informant de ce fait.

5. 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 retourner un objet avec les mêmes paramètres que la tartiflette. Affichez cet objet.

6. 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