**TP 2 : Les joueurs aux pokemons !**

Dans ce TP nous continuerons à travailler sur la relation entre un pokemon et un joueur. Nous allons implémenter plusieurs constructeurs, nous permettrons aux joueurs de nommer leurs pokemons et de capturer et libérer des pokemons qu'ils possèdent. Pour un aspect plus social, nous donnerons également la possibilité à deux joueurs de s'échanger des pokemons.

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

## Préparation avant de commencer le TP

Nous allons continuer de programmer le jeu aux pokemons. Nous avons un choix : est-ce qu'on veut continuer à modifier le projet Java commencé au TP précédent ou est-ce qu'on veut commencer un nouveau projet ?

Pour notre projet nous allons sauvegarder le travail de chaque TP dans un nouveau projet.

0. Pour commencer, ouvrez Eclipse et créez un nouveau projet Java, qui s'appellera TP2. Dedans, créez un package tp2.
   Dans votre package explorer (à gauche dans la vue standard d'Eclipse) détaillez votre TP1 et sélectionnez les deux classes de ce TP. En Windows faites CTRL + C (ou en utilisant click gauche copiez les deux fichiers). Puis mettez vous sur le package tp2 du projet TP2 et collez les deux fichiers.
   Nous allons désormais travailler exclusivement sur le code du TP2.

1. Ouvrez la classe ChasseAuxPokemons. Regardez la première ligne de ce fichier : remarquez le fait que ce qu'on vient de faire -- copier/coller les fichiers -- a changé le package dans lequel on travaille.
   Dans la classe ChasseAuxPokemons enlevez tout le code sauf la déclaration des trois pokemons et les trois lignes de System.out.println des trois pokemons.

2. Dans la classe Pokemon regardez la méthode String toString().
   - Changez son nom à String sePresente().
   - Faites exécuter le code écrit dans la méthode main. Qu'est-ce qui se passe avec l'affichage des pokemons ? Pourquoi ?

- Faites afficher le texte renvoyé par la méthode sePresente().

3. La méthode String toString() est utilisée de façon standard pour afficher l'état actuel des objets de la classe. Nous allons donc écrire une nouvelle méthode qui retournera les valeurs des attributs de la classe dans le format suivant :

   [Nom : <valeur du nom du pokemon> ; Type : <le type du pokemon> ; Niveau : <le niveau du pokemon> ; Diurne : <true si diurne, false sinon>]

4. À présent, nous voudrions ajouter des joueurs au jeu des pokemons. Nous avons besoin d'une nouvelle classe Joueur. Les attributs caractérisant les objets de cette classe seront :
   Un nom de type String
   Un prénom de type String
   Un attribut age de type int
   Un tableau contenant leurs pokemons. La taille de ce tableau sera limitée à 5.
   - Implémentez cette classe et ajouter un constructeur avec la signature :

     Joueur (String, String, int, Pokemon[])

     Ce constructeur assigne à chaque attribut la valeur correspondante mise en paramètre.

   - Implémentez un deuxième constructeur, qui aura la signature
     Joueur (String, String, int)

     Ce deuxième constructeur instancie les trois premiers attributs aux valeurs données en paramètre et instancie les cinq éléments du tableau de pokemons à null.
   - Implémentez des getters pour les attributs nom et prenom.

5. Dans la classe Pokemon :
   - Ajoutez un attribut nomDonne du type String et un attribut monJoueur du type Joueur.
   - Ajouter un deuxième constructeur à la classe Pokemon avec la signature : Pokemon(String, String, int, boolean, String, Joueur) , qui prendra en entrée, en ordre, les valeurs des attributs : **nom**, **type, niveau, diurne, nomDonne, monJoueur**.
   - Chaque pokemon est né en liberté, alors initialement les attributs nomDonne et monJoueur ne seront pas initialisés. Si on veut initialiser un attribut, sans lui donner une valeur, nous pouvons lui attribuer la valeur null (par exemple nomDonne = null;). Ainsi, l'object fera référence à une location vide.
     Adaptez ainsi l'ancien constructeur de la classe Pokemon (avec la signature Pokemon(String, String, int, boolean)) en initialisant dedans les attributs nomDonne et monJoueur à null. Pour éviter la déduplication du code nous allons appeler le constructeur avec plus de paramètres dans celui avec moins.

- Modifiez la méthode toString() de la classe Pokemon pour y ajouter les valeurs des attributs nomDonne et monJoueur dans le même format que les autres attributs.
- Modifiez la méthode sePresente() de la classe Pokemon. Le texte qui sera affiché sur l'écran sera désormais :
  - Si le pokemon a un maître, et si nomDonne n'est pas mis à null alors le texte affiché sera : <nomDonne du pokemon> est un pokemon de genre <nom du pokemon>, du type <type du pokemon>, qui a le niveau <niveau>. Ce pokemon appartient a <le nom et prenom du joueur monJoueur>.
  - Si le pokemon a un maître, mais nomDonne est mis à null alors le texte affiché sera : Voici un pokemon du genre <nom du pokemon>, du type <type du pokemon>, qui a le niveau <niveau>. Ce pokemon appartient a < le nom et prenom du joueur monJoueur >.
  - Si le pokemon n'a pas de maître, alors le texte affiché sera : Voici un pokemon du genre <nom du pokemon>, du type <type du pokemon>, qui a le niveau <niveau>. Ce pokemon n'a pas encore de maitre.

- Dans la méthode principale (main) de la classe ChasseAuxPokemons, regardez les instructions que vous avez utilisées pour créer vos trois pokemons (Piplup, Rowlet et Totodile). Est-ce que ces instructions vont compiler ? Pourquoi (ou pourquoi pas) ? Quel sera l'affichage des instructions que vous avez ensuite utilisées pour « afficher » ces pokemons ?
- Nous voulons pouvoir accéder aux attributs nomDonne et monJoueur dans la classe Pokemon pour les modifier au besoin (un pokemon peut avoir un maître, puis en changer). Pour cela il faut avoir une méthode qui retourne la valeur stockée en ce moment par chacune de ces variables, et une méthode qui change la valeur de chacune de ces variables. Créez les méthodes suivantes :
  - String getNomDonne() – qui retourne la valeur actuellement stockée par nomDonne
  - Joueur getMonJoueur() – qui retourne la valeur actuellement stockée par monJoueur
  - void setNomDonne(String) – qui remplace la valeur actuellement stockée dans nomDonne par la valeur mise en entrée
  - void setMonJoueur(Joueur) – qui remplace la valeur actuellement stockée par monJoueur par le joueur donné en entrée.

6. Maintenant nous allons passer à la classe Joueur.
   - Dans le constructeur avec la signature Joueur(String, String, int, Pokemon[]) nous avons assigné a this.pokemons la valeur en entrée de type Pokemon[]. Nous allons s'assurer que chacun de ces pokemons aura la valeur de MonJoueur mise au joueur actuel (this). Modifiez donc le constructeur tel que cela soit réalisé.
   - Nous allons écrire une méthode privée avec la signature int trouverPokemon(Pokemon pokemon) qui aura le fonctionnement suivant : nous allons chercher dans le tableau this.pokemons jusqu'à trouver le pokemon en paramètre de la méthode. Si on trouve ce pokemon, la méthode rend sa position dans le tableau. Si on ne trouve pas ce pokemon,

on retourne la valeur -1 (qui fonctionne comme une valeur d'erreur, indiquant que le pokemon n'existe pas dans le tableau).

- Comment peut-on utiliser cette méthode pour chercher si on a une place disponible dans le tableau ?

7. Encore dans la classe Joueur, nous allons créer les interactions suivantes entre les joueurs et les pokemons :
    - **Capturer un pokemon** : pour l'instant cette méthode permettra d'ajouter un pokemon au tableau de pokemons d'un joueur (si ce dernier a moins de cinq pokemons à ce moment-là). Si le joueur a déjà cinq pokemons, il y aura un message d'erreur qui demandera au joueur de renoncer à l'un de ses pokemons.

      La signature de la méthode qui capture des pokemons sera : `void capturer (Pokemon)`
      Attention : il n'est pas possible de capturer un pokemon qui appartient à un autre joueur.

      De plus, lorsqu'un pokemon libre est capturé par un joueur, ce dernier devient son maître, et donc l'attribut `monJoueur` est mis à jour de façon correspondante (voir la méthode `setMonJoueur`).

    - **Libérer un pokemon** : cette méthode-ci permettra à un joueur de libérer un pokemon qui est déjà présent dans son tableau d'au plus 5 pokemons. La signature de cette méthode sera : `void liberer (Pokemon)`. Un pokemon libéré aura son `nomDonne` et son `monJoueur` remis à `null`. Il redeviendra capturable. Attention : il faut bien prendre en compte que, lorsque le joueur capture un autre pokemon pour remplacer un qu'il a libéré, il faut trouver la ou les position(s) libre(s) du tableau de pokemons.

    - **Nommer un pokemon** : un joueur peut donner des noms aux pokemons dans sa collection. Ceci changera l'attribut `nomDonne` de ce pokemon. Quelle sera la signature de cette méthode (qui fera partie de la classe Joueur) ?

8. (Le petit extra) Si vous avez déjà fini l'exercice précédent, utilisez vos méthodes pour écrire du code dans la classe Joueur qui vous permettrait de faire les actions suivantes :
    - **Donner ses pokemons** : un joueur pourrait vouloir donner un de ses pokemons à un autre joueur (sans faire un échange). Écrivez une méthode qui permet cela, sans oublier de faire les vérifications nécéssaires sur le joueur qui reçoit le pokemon (sa collection ne doit pas excéder cinq pokémons) et sans oublier de mettre à jour les informations du pokemon donné.

    - **Échanger des pokemons** : deux joueurs peuvent échanger des pokemons. Écrivez une méthode qui effectue l'échange des pokemons entre deux joueurs.