TP 4 : L'héritage, la nourriture et d'autres intéractions !

Aujourd'hui notre but sera d'utiliser l'héritage pour étendre la notion de nourriture. Puis, nous allons continuer créer des interactions entre les joueurs et leurs pokemons : les joueurs pourront nourrir et caresser leurs pokemons.

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

Préambule

On commence le TP comme d'habitude : on crée un nouveau projet qui s'appelle TP4, avec un package dedans qui s'appelle tp4. Dans ce package nous allons copier les classes du TP précédent (en harmonisant les références si besoin).

Exercice 1

La dernière fois nous avons créé de la nourriture pour les pokemons, toutefois sans leur donner la possibilité d'interagir avec la nourriture. Nous allons premièrement programmer une méthode qui nous permet faire un pokemon manger une nourriture.

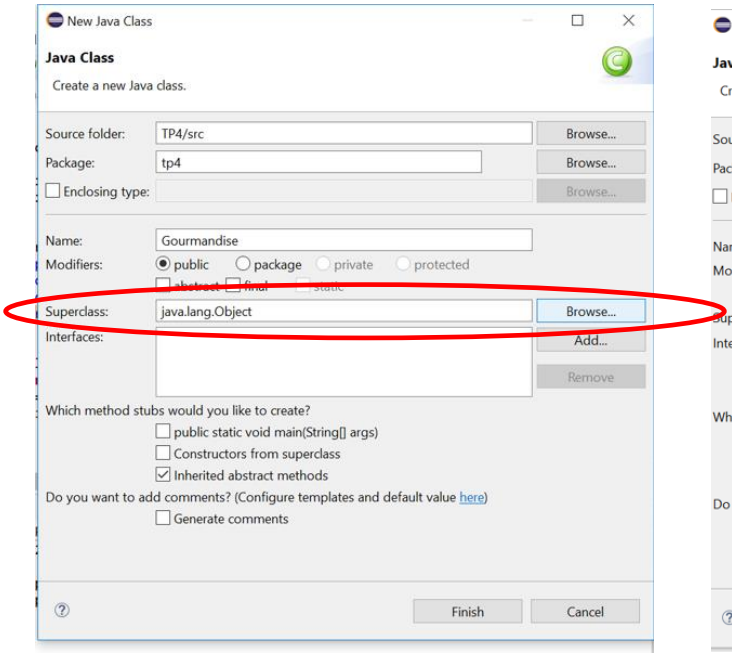
- Dans la classe `Pokemon`, écrivez une méthode avec la signature `void manger(Nourriture nourriture)`. Cette méthode vérifie si la nourriture n'est pas `null` et si elle est compatible avec le pokemon. Si c'est le cas, l'appétit du pokemon baissera par l'apport nutritionnel de la nourriture (ou deviendra 0 si l'apport est supérieur à l'appétit du pokemon).
- Dans la classe `ChasseAuxPokemons`, dans la méthode principale (`main`) :
    - Essayez de faire manger une nourriture à un pokemon qui est bien compatible avec la nourriture donnée
    - En affichant les détails du pokemon avant et après l'instruction qui le fait manger la nourriture, vérifiez que son appétit a bien baissé par l'apport nutritionnel de la nourriture
    - Faites le pokemon manger la même nourriture encore une fois et répétez cela jusqu'à faire son appétit baisser sous 0. Est-ce que l'appétit se maintient à 0 ?
    - Faites un autre pokemon essayer de manger une nourriture qui n'est pas compatible avec lui. Qu'est-ce qui se passe ?

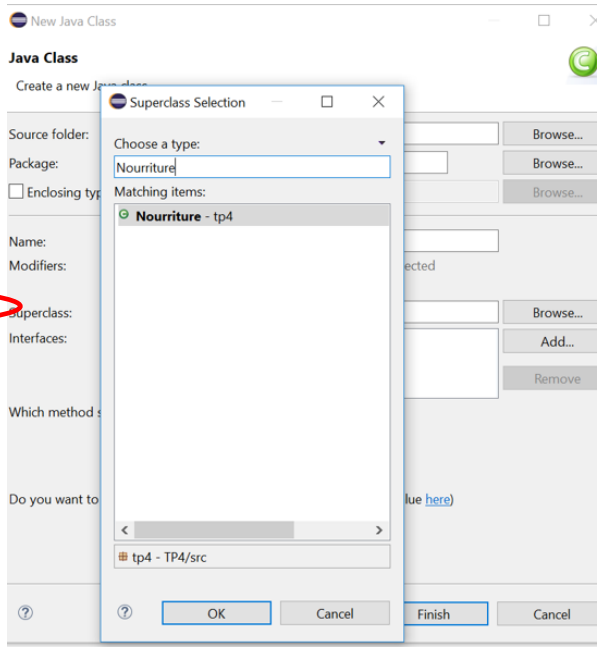
Exercice 2

Il y aura divers types de nourriture dans ce jeu. Le type le plus commun, c'est celui qui soulage seulement l'appétit d'un pokemon. Mais on aura également de la nourriture spéciale, qui changera également la loyauté des pokemons. Finalement, il y a également des types de nourriture qui peuvent modifier d'autres attributs des pokemons -- par exemple leurs niveaux.

Nous allons utiliser l'héritage pour créer deux sous-classes de `Nourriture` -- correspondant à la nourriture qui change la loyauté des pokemons, et à celle qui modifie leurs niveaux.

- Nous allons créer une nouvelle classe. Lorsque l'écran de création de la classe s'affiche, trouvez sur cet écran la superclasse de l'objet (mise par défaut à `Java.lang.Object`) -- voire les illustrations ci-dessous.





  Utilisez le bouton `Browse` pour arriver à l'écran suivant, notamment l'écran de sélection des superclasses. Mettez en tant que superclasse votre classe `Nourriture`. Eclipse vous affichera les classes possibles (dans ce cas-ci, seulement la classe `Nourriture` dans le package `tp4`). Faites un double-click sur la suggestion d'Éclipse.

  Nommez cette première classe `Gourmandise`. Faites click sur Finish.
  Pouvez-vous voir une indication de l'héritage indiqué dans le code de la classe `Gourmandise` ?

- Au-delà des attributs et méthodes qu'elle hérite de la classe `Nourriture`, les objets de type `Gourmandise` vont également avoir deux attributs de plus, de type entier : `apportLoyaute`. Ecrivez un constructeur pour cette classe, avec la signature `Gourmandise(int, String,`

`String[], int, int`). Ce constructeur utilise le constructeur de la classe `Nourriture` avec les premiers 4 paramètres du constructeur de la classe `Gourmandise` pour instancier les attributs hérités par la classe `Gourmandise` de la classe `Nourriture`. La dernière valeur sert à instancier l'attribut `apportLoyaute`.

- Dans la classe Nourriture modifiez votre méthode String toString() pour qu'elle retourne le texte : `"<nom>, <apport>, <frequence>/100, <liste de compatibilites, separees par une virgule>"`

- Ecrivez une méthode `String toString()` dans la classe `Gourmandise`, qui retourne : `"Gourmandise : <apport loyaute>, <nom>, <apport>, <frequence>/100, <liste de compatibilites, separees par une virgule>".` Quelle est la mention qu'il faut mettre avant d'écrire la méthode ?

- Ecrivez un getter pour l'attribut `apportLoyaute`.

- La méthode `genererMemeNourriture` de la classe `Nourriture` génère un objet de type `Nourriture`. Nous voulons une méthode `genererMemeNourriture` dans la classe `Gourmandise` qui génère une gourmandise, plutôt qu'une nourriture. Ecrivez une méthode `Gourmandise genererMemeNourriture(boolean generer)` qui génère une gourmandise avec les mêmes caractéristiques que la gourmandise actuelle (`this`).

- Dans la classe `ChasseAuxPokemons`, créez une nouvelle gourmandise `barreChocolatee` qui a le nom `"Barre chocolatee"`, un apport nutritionnel de 20, un apport de loyaute de 7 et une fréquence de 10. Vous pouvez choisir librement les compatibilités de cette nourriture. Affichez ses caractéristiques.

Exercice 3

Un autre type spécial de nourriture est une potion magique. Ce type de nourriture ne soulage pas la faim d'un pokemon, mais elle fait monter le niveau d'un pokemon par 1. Elle est compatible avec tout type de pokemon.

- Pour faciliter la programmation des compatibilités, nous allons introduire une constante tousLesTypesDePokemondans la classe Nourriture. Cette constante s'appellera tousLesTypesDePokemons. Elle sera une valeur publique et statique de type String[], qui stockera tous les types de pokemons possibles.
- Créez une nouvelle classe PotionMagique qui hérite de la classe Nourriture. Il n'y aura pas de nouvel attribut dans cette classe.
- Le constructeur de la classe PotionMagique aura la syntaxe PotionMagique(String nom, int frequence). Il appellera le constructeur de la classe Nourriture en mettant l'apport à 0 et les

compatibilités à la nouvelle constante créée dans la classe Nourriture, qui contient tous les types de pokemon possibles. Ecrivez ce constructeur.

- Ecrivez une méthode String toString() dans la classe PotionMagique, qui retourne "`Potion Magique : <nom>, 0, <frequence>/100, <liste de compatibilites, separees par une virgule>`".
- Finalement, écrivez une méthode `PotionMagique genererMemeNourriture()` qui retourne une potion magique avec les mêmes attributs que la potion magique actuelle `(this)`.
- Dans la classe `ChasseAuxPokemons`, créez une nouvelle potion magique `mojito` qui a le nom "`mojito`" et une fréquence de 2. Affichez ses caractéristiques.

Exercice 4

Dans cet exercice le but sera de pouvoir faire les pokemons manger des gourmandises et des potions magiques. Le but sera aussi de réaliser cela d'une façon qui permettra, lors de l'utilisation du polymorphisme, de faire exécuter une seule instruction, qui permettra à la fois de prendre en compte les propriétés d'une gourmandise (en augmentant la loyauté) et d'une potion magique (mise à niveau).

- Créez premièrement une méthode `void miseANiveau()` dans la classe `Pokemon`, qui fait juste augmenter le niveau d'un pokemon par 1. Cette mise à niveau est très simpliste, mais pour l'instant c'est ce qu'on peut faire avec les attributs actuels des pokemons.

- Actuellement la méthode manger de la classe `Pokemon` permet à celui-ci de manger une nourriture. Comme les gourmandises et les potions magiques sont des types de nourriture, cette méthode marcherait si on mettait en paramètre un objet de type `PotionMagique` ou un objet de type `Gourmandise`. Toutefois, la méthode manger ne fait que baisser l'appétit du pokemon par l'apport nutritionnel de la nourriture -- et donc cette méthode n'est pas utilisable dans le sens souhaité pour les gourmandises (il nous manque la modification de la loyauté du pokemon), ni pour les potions magiques (pas de mise à niveau).

  Une solution, si on veut toujours garder la méthode manger de la classe `Pokemon`, est de faire cette méthode appeler une autre méthode, qui sera mise dans la classe `Nourriture` (et puis modifiée dans les sousclasses). Nous allons voir comment faire cela.

  On commence par écrire une méthode `void estMangee(Pokemon pokemon)` dans la classe Nourriture. Cette méthode vérifie seulement si le paramètre en entrée n'est pas `null` et, si c'est le cas, alors l'appétit du pokemon sera modifié selon les règles évoquées dans le premier exercice.

- Regardez votre méthode manger dans la classe `Pokemon`. Comment pouvez-vous modifier le code de cette méthode tel qu'elle fonctionne exactement comme avant, mais cette fois-ci en appelant la méthode `estMangee` de la classe `Nourriture` ?

- Ecrivez des méthodes `void estMangee(Pokemon pokemon)` dans les sousclasses `Gourmandise` et `PotionMagique`. Dans la classe `Gourmandise` cette méthode doit s'assurer que l'appétit du pokemon baisse par l'apport nutritionnel de la nourriture, et que, s'il a un maître, sa loyauté monte par l'apport de loyauté de la gourmandise, toutefois sans dépasser 100. Dans la classe `PotionMagique` il faut juste augmenter le niveau du pokemon par 1 (en utilisant la méthode `miseANiveau`).

- Créez un tableau polymorphe qui contient la tartiflette, la ratatouille, la barre chocolatée et le mojito. En utilisant une boucle for, faites générer une nourriture de chaque type. Faites à un de vos pokemons qui n'a pas encore un maître manger chaque type de nourriture. Faites capturer le pokemon par un joueur. Qu'est-ce qui change ?

- (le petit extra) Essayez de vous imaginer comment le code changerait si on avait choisi de ne pas garder la méthode manger. Quel code aurait dû être modifié ? Attention : ne modifiez pas votre code directement dans votre répertoire pour TD4. Vous pouvez soit créer un nouveau projet à ce but, soit mettre les modifications sur papier.