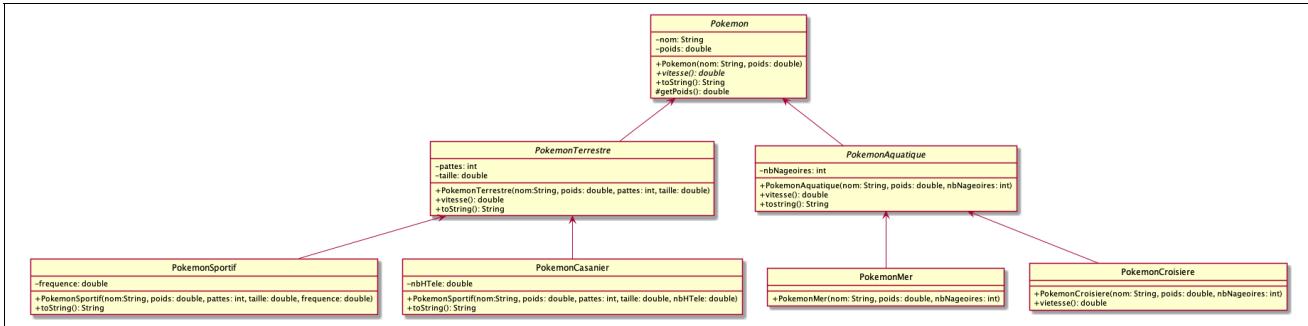


Une implémentation de Pokémons

Le but de cet exercice est d'implémenter la hiérarchie de Pokémons du TD.

Lors du TD, nous sommes arrivés au diagramme UML suivant:



N'hésitez pas à visualiser l'image dans une nouvelle fenêtre et à zoomer sur les différentes parties. (bouton droit sur l'image -> afficher l'image...)

Question 1

La classe Pokemon est

- une classe instanciable
- une classe abstraite
- une interface
- une énumération

Vérifiez votre réponse

Résultat: Votre réponse est juste.

La classe Pokemon contient des attributs, ce n'est donc pas une interface.

Tous les pokemons ont une vitesse, il est donc naturel de déclarer une méthode `+ vitesse(): double` dans cette classe. cependant, dans la classe Pokemon elle-même, on ne sait pas comment l'implémenter. Cette méthode sera donc **abstraite**.

Comme la méthode `+ vitesse(): double` est abstraite, la classe Pokemon **doit être abstraite**.

Question 2

Une méthode `# getPoids(): double` a été ajouté à la classe Pokemon. Pourquoi à votre avis ?

- Parce qu'il faut systématiquement mettre des accesseurs et des modificateurs pour tous les attributs
- Parce qu'on a besoin du poids dans le calcul de la vitesse des PokemonTerrestres
- Parce qu'on a besoin du poids dans le calcul de la vitesse des PokemonAquatiques
- Parce qu'on a besoin du poids dans la méthode `+toString(): String` des PokemonSportifs
- Parce qu'on a besoin du poids dans la méthode `+toString(): String` des PokemonCasaniers
- Parce qu'on a besoin du poids dans la méthode `+toString(): String` des PokemonMers
- Parce qu'on a besoin du poids dans la méthode `+toString(): String` des PokemonCroisières

Vérifiez votre réponse

Résultat: Votre réponse est juste.

Lorsqu'ils ne sont pas utiles, les accesseurs et surtout les modificateurs ne doivent pas être implémentés. En effet, d'après le principe d'encapsulation, il se peut que les attributs ne doivent pas être modifiés. Si la classe gère elle-même ses attributs et ses méthodes, certains n'ont plus besoin d'être visibles par des classes extérieures. On notera d'ailleurs ici que le modificateur d'accès de l'accesseur `# getPoids(): double` est `protected`.

Les méthodes `+toString(): String` des classes PokemonSportif, PokemonCasanier, PokemonMer et PokemonCroisiere n'ont pas besoin d'accéder au poids du pokémon. En effet, cette information sera écrite dans la méthode `+toString(): String` de la classe Pokemon elle-même.

Question 3

Les classes PokemonTerrestre et PokemonAquatique sont **abstraites** parce que

- elles contiennent chacune une/des méthode(s) abstraite(s)
- il existe dans la nature de nombreux PokemonTerrestre et PokemonAquatique
- c'est une lubie de l'enseignant, en fait elles ne devraient pas être abstraites
- on ne devrait pas pouvoir instancier des PokemonTerrestre et des PokemonAquatique.

Vérifiez votre réponse

Résultat: Votre réponse est juste.

Les classes `PokemonTerrestre` et `PokemonAquatique` ne contiennent aucune méthode abstraite. Elles devraient donc techniquement pouvoir être instanciées. Cependant, elles ont été construites pour factoriser du code commun aux classes `PokemonSportif` et `PokemonCasquier` d'une part et aux classes `PokemonMer` et `PokemonCroisiere` d'autre part. Elles n'ont pas de réalité autre que techniques. En d'autres termes, "Il n'existe aucun `PokemonTerrestre` et `PokemonAquatique` dans la vraie vie."... Il n'y aurait donc aucun sens à instancier un `PokemonTerrestre` ou un `PokemonAquatique`. C'est pourquoi ces 2 classes sont abstraites.

Question 4

Tel que le code est décrit dans le diagramme UML de classes, on ne peut pas appeler la méthode `+ vitesse(): double` sur une instance de `PokemonMer`.

- vrai
 faux

Vérifiez votre réponse

Résultat: Votre réponse est juste.

En fait, la vitesse des `PokemonMer` est calculée dans la classe `PokemonAquatique`. On peut donc appeler la méthode `+ vitesse(): double` sur une instance de `+ vitesse(): double` qui hérite de l'implémentation de toutes les méthodes de sa classe mère.

Question 5

Implémentez les classes `Pokemon`, `PokemonTerrestre`, `PokemonAquatique`, `PokemonSportif`, `PokemonCasquier`, `PokemonMer` et `PokemonCroisiere`.

Les tests Caseine pour ces classes ne vérifient que les signatures des méthodes et les déclarations de classes et d'attributs. Une fois que vous avez 20/20 à ces tests, vous **devez** vérifier que votre code est correct grâce au `main` fourni et solliciter vos enseignants pour vérifier que ce code est également écrit dans la bonne classe.

Si votre code n'est pas complet, ou s'il ne compile pas (y compris le `main`), les tests Caseine renverrons, comme à chaque fois qu'un test ne compile pas, 0/20.

Ci-dessous est le résultat de la sortie standard que vous êtes supposé obtenir en exécutant la méthode `main`:

```
1 Je suis le pokemon Pikachu mon poids est de 18.0kg, ma vitesse est de 5.1 km/h j'ai 2 pattes, ma taille est de 0.85m ma fréquence ca
2 Je suis le pokemon Salameche mon poids est de 12.0kg, ma vitesse est de 3.9000000000000004 km/h j'ai 2 pattes, ma taille est de 0.65
3 Je suis le pokemon Rondoudou mon poids est de 45.0kg, ma vitesse est de 3.6 km/h j'ai 2 nageoires
4 Je suis le pokemon Bulbizarre mon poids est de 15.0kg, ma vitesse est de 0.8999999999999999 km/h j'ai 3 nageoires
5 La vitesse moyenne des pokemons est: 3.375
6
```

[Fermer]