Héritage et appel des constructeurs (Lab Caseine no 21112)

L'objectif de cet exercice est de s'entraîner à l'encapsulation vis à vis de l'héritage.

Nous allons considérer une classe `Humain`. Dans cet exemple, un `Humain` n'a qu'un `age` qui doit être positif ou nul.

Nous considérerons ensuite une classe `Bebe`. Un `Bebe` est un `Humain` qui a entre 0 et 6 ans et pour lequel on compte les `dents`: il en a entre 0 et 28.

Le diagramme UML de classes est présenté ci-dessous:

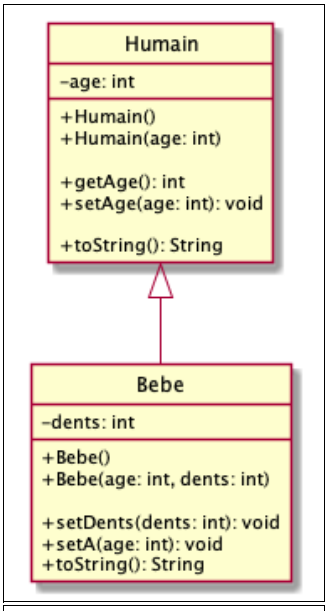

Figure 0.1. Diagramme UML de classes.

> Les tests ne fonctionnent pas dans un premier temps. Ils fonctionneront lorsqu'un minimum de méthodes auront été remplies.

La classe Humain

Encapsulation de l'attribut

Nous allons remplir la classe `Humain` petit à petit.

Question 1

> Définissez l'attribut `age` de type `int` dans la classe `Humain`.

L'attriut `age` de la classe `Humain` doit toujours être $>= 0$. Cela doit être assuré par le principe d'encapsulation.

Question 2

> Commencez par écrire l'accesseur de l'attribut `age`

Question 3

> Ecrivez le mutateur (modificateur / *getter*) de l'attribut `age`.

> L'attribut `age` ne peut être changé que si le paramètre `age` du mutateur est positif ou nul. Si ce n'est pas le cas, l'attribut garde sa valeur initiale.

> Vous pouvez à présent lancer un premier test.

Les constructeurs

La classe `Humain` possède 2 constructeurs. Le constructeur qui prend un paramètre `age` a deux fonctions:

- Initialiser l'attribut `age` tout en s'assurant du principe d'encapsulation, mais sans réécrire du code existant
- Afficher pour l'utilisateur l'état de l'instance après sa création. Ainsi, la deuxième ligne de ce constructeur devra être: `System.out.println(this);`

Question 4

Ecrire le constructeur qui prend un paramètre.

Question 5

A ce stade, votre projet ne compile plus. Pourquoi ?

- [x] *Cela vien du fait qu'il y a maintenant un constructeur avec un paramètre*
- [ ] *Quelque chose ne va pas dans le constructeur que vous venez de créé*
- [ ] *Cela vien de l'accesseur et du modificateur*
- [ ] *C'est incompréhensible !*

Résultat: Votre réponse est juste.

La prochaine étape de cet exercice est de compléter la classe `Bebe` qui hérite de `Humain`. La classe `Bebe` n'a pas encore de constructeur (vous devrez les écrire). Java crée donc implicitement et silencieusement un constructeur par défaut pour `Bebe`. Or ce constructeur par défaut appelle automatiquement le constructeur par défaut de la classe `Humain`.

Tant que vous n'aviez pas créé de constructeur avec un paramètre dans la classe `Humain`, Java avait également créé implicitement et silencieusement un constructeur par défaut pour `Humain`.

Or, maintenant que vous avez créé un constructeur avec un paramètre, le constructeur implicite par défaut de Java a disparu.... D'où le problème de compilation de votre classe `Bebe`.

Le constructeur par défaut, c'est-à-dire le constructeur qui ne prend pas de paramètre ne doit surtout pas ré-écrire du code.

Lorsque l'on ne connaît pas l'âge d'un `Humain`, on lui donne l'âge moyen de 50 ans.

Question 6

Ecrire le constructeur par défaut.

La méthode `toString()`

Vous pouvez à présent dé-commenter les première lignes du `main` de la classe `Programme`.

Question 7

Qu'est-il affiché lors de la construction d'un `Humain` ?

- [ ] *rien*
- [ ] *Humain (age: 50)*
- [x] *uga.Humain@65ae6ba4 (avec une autre adresse après lee @)*

Résultat: Votre réponse est juste.

Question 8

Pourquoi ?

- [ ] *Cela est dû à la recherche des instances dans la JVM.*
- [ ] *Cela est dû au fait que l'on appelle System.out.println() dans un constructeur*
- [x] *Cela est dû à l'implémentation de la méthode toString() de la classe Object.*

Résultat: Votre réponse est juste.

En effet, la conversion automatique d'une référence en `String` passe par la méthode `toString()`. Cette méthode est implémentée par défaut dans la classe `Object`. Comme toute classe hérite de `Object` `Humain`, qui ne redéfinie pas encore la méthode `toString()`, utilise la méthode `toString()` de la classe `Object` dont il hérite implicitement et automatiquement.

On souhaite à présent que la méthode `toString()` affiche: `Humain (age : 50)`.

Question 9

Re-définissez la méthode `toString()`.

Question 10

Vérifiez que les tests correspondants aux questions précédentes sont tous positifs avant de passer à la suite.

La classe Bebe

Encapsulation des attributs

Un bébé est un `Humain` entre 0 et 6 ans et qui a entre 0 et 28 dents.

Lorsque l'on ne connaît pas le nombre de dents d'un bébé, ou lorsque le paramètre du mutateur de `dents` n'est pas correct, on lui affecte un nombre moyen de 12 dents.

Question 11

Créez un attribut `dents` et encapsulez-le.

Question 12

Comme l'âge d'un `Bebe` n'a pas les même propriétés que l'âge d'un `Humain`, redéfinissez le mutateur (modificateur, *setter*) de l'attribut `age`.

Attention ! L'attribut `age` de `Humain` doit rester `private`...

Question 13

Définissez les constructeurs de `Bebe` comme indiqués dans le diagramme UML de classes.

Le constructeur avec des paramètres devra également avoir pour dernière ligne `System.out.println(this)`.

Le constructeur de la super classe est appelée par `super(...)`

Cette instruction doit être la **première** instruction du constructeur.

Question 14

Votre constructeur doit appeler le constructeur de la super-classe. Le constructeur de la classe `Humain` appelle la méthode `setAge(age: int):void`.
A votre avis, la méthode `setAge(age: int):void` qui sera exécutée dans l'appel au constructeur de `Humain` depuis le constructeur de `Bebe` sera la méthode:

- [ ] *de Humain*
- [x] *de Bebe*
- [ ] *Aucune des 2*

Résultat: Votre réponse est juste.

En Java, la méthode appelée sur une instance est toujours la méthode la plus spécifique. C'est-à-dire que même dans le constructeur de `Humain`, c'est la méthode `setAge()` de `Bebe` qui sera appelée... ... Tout comme la méthode `toString()` de `Bebe` dès que vous l'aurez re-définie.

Question 15

Avant de décommenter la deuxième partie du code de la méthode `main` de la classe `Programme`, essayer de deviner ce qui sera écrit à l'écran.
Dé-commentez la deuxième partie du code de la méthode `main` de la classe `Programme` et vérifiez vos hypothèses.

Question 16

Re-définissez la méthode `toString()` pour qu'elle écrive:
`Bebe (age : 3, dents : 12).`

Cette méthode ne devra pas appeler `super.toString()`.

Question 17

Que s'affiche-t-il à présent à l'exécution de la méhtode `main` de la classe `Programme` ? Pourquoi ?

[Fermer]