UE : Programmation Orientée Objet (I4AEIL11)

Institut National des Sciences Appliquées de Toulouse
Yannick Pencolé, yannick.pencole@laas.fr

—

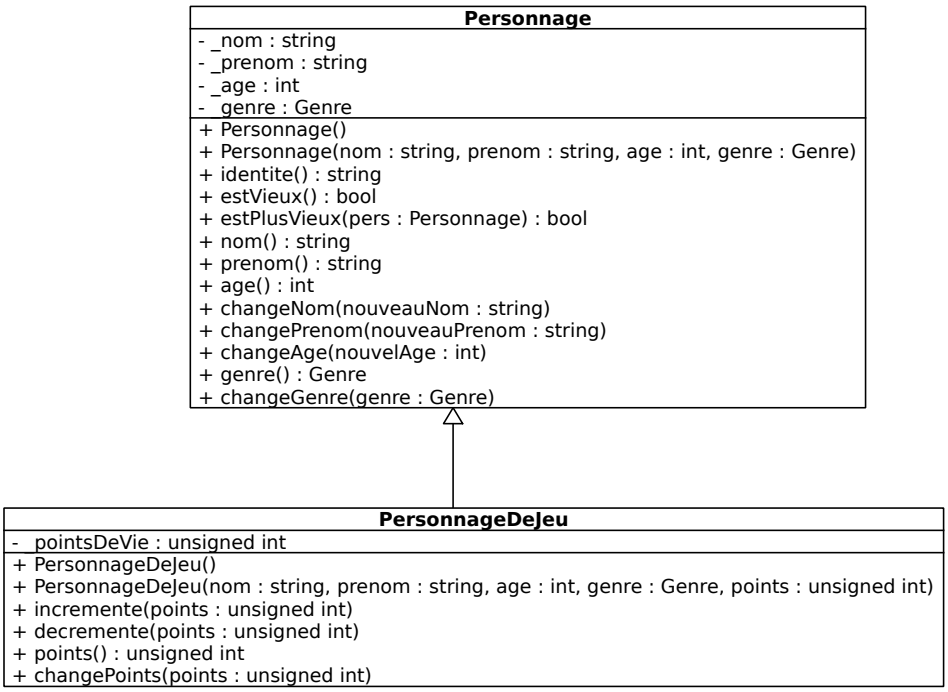
TD 2: Héritage, Polymorphisme, Diagrammes UML

(12 février 2024)

Ce TD s'appuie sur la classe `Personnage` que vous avez développée dans le TD1. Si vous n'avez pas eu le temps de terminer, vous pouvez partir de la solution qui se trouve sur moodle.

1 Héritage

Nous souhaitons procéder à la mise en œuvre d'un programme de gestion de plusieurs types de `Personnage` en vue de les exploiter dans des jeux (jeux vidéos, jeux de rôles,...). Un personnage de jeu a en général des points de vie (tant qu'il en a, il est vivant, sinon il meurt).



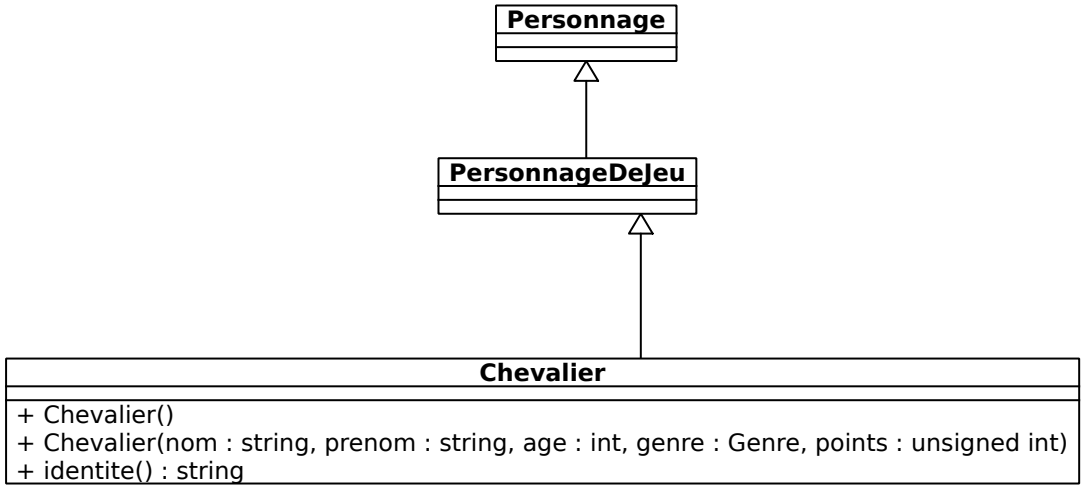
1. Créez une classe `PersonnageDeJeu` qui **dérive** de la classe `Personnage` et respecte le diagramme de classe UML ci-dessus. Le constructeur par défaut de cette classe

construira un personnage dont les noms, prénoms sont vides, l'âge à 0, le genre est féminin et les points de vie sont à 0. La méthode `incremente` incrémentera le nombre de points de vie. La méthode `decremente` décrémentera le nombre de points de vie. `points` sera l'accesseur pour les points de vie et `changePoints` son mutateur. Mettez également en œuvre le destructeur de `PersonnageDeJeu`. La classe `Personnage` dispose de combien de méthodes ? La classe `PersonnageDeJeu` dispose de combien de méthodes ?

2. En vue de tester votre classe, vous pouvez mettre en œuvre le scénario suivant dans votre programme principal :
   (a) Créez le `PersonnageDeJeu` Frodon Saquet, masculin, 65 ans, 30 points de vie.
   (b) Affichez son nom, son prénom, son genre, son âge et ses points.
   (c) Décrémentez de 3 son nombre de points de vie et réaffichez ses points.
   (d) Incrémentez de 6 son nombre de points de vie et réaffichez ses points.
   (e) Remplacez le prénom du personnage par `Galadriel`
   (f) Remplacez le nom du personnage par `Dame de Lorien`
   (g) Remplacez son genre par féminin
   (h) Remplacez son âge par 1040 (oui elle ne fait pas son âge).
   (i) Remplacez ses points de vie par 30000.
   (j) Réaffichez son nom, son prénom, son genre, son âge et ses points.
3. Parmi les personnages de jeux on peut en distinguer plusieurs types. Nous voulons mettre en œuvre un type `Chevalier`. De part sa noblesse d'âme et de coeur (certains le disent prétentieux...), le chevalier n'accepte pas son identité de Personnage. Pour un `Chevalier`, son `identite()` devra être de la forme :

   ```
   Moi, preux chevalier [prenom], je suis là pour vous servir.
   ```

   Mettez en œuvre la classe `Chevalier` (voir diagramme UML) et spécialisez la méthode `identite()` du preux Chevalier (ne pas oublier de virtualiser la méthode `identite` de `Personnage` et les destructeurs). En quoi la méthode `identite()` de la classe `Chevalier` **spécialise** la méthode `identite()` de `Personnage` ?



4. Pour tester le polymorphisme de la méthode `identite`, dans votre programme, initialisez un vecteur de pointeurs `vector<Personnage *>`. Créez un `PersonnageDeJeu` et puis ajoutez-le dans le vecteur (avec `push_back`). Créez un `Chevalier` et ajoutez-le également dans le vecteur. Pourquoi avez-vous le droit de stocker des `PersonnageDeJeu` et des `Chevalier` dans ce vecteur de `Personnage` ? Enfin, en mettant en œuvre une itération sur ce vecteur de `Personnage`, affichez l'identité de chacun des `Personnage`. Que devez-vous obtenir ? En quoi ce test illustre le **polymorphisme**, un des piliers de la programmation orientée objet ?

5. Le `Chevalier` n'est toujours pas satisfait, il veut que l'on surcharge la méthode `identite` avec un nouveau paramètre `string interlocuteur`.

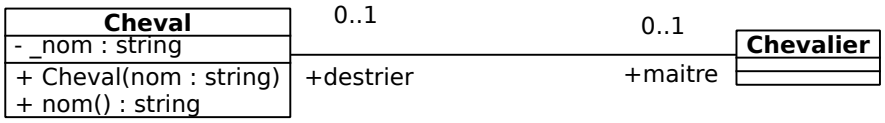
   ```
   Moi, preux chevalier [prenom], je suis là pour vour servir
                        [interlocuteur].
   ```

   Pour tester par exemple, dans le `main`, créez le chevalier Lancelot du Lac et affichez `lancelot.identite("Belle Guenièvre")`. En quoi la méthode `identite(string interlocuteur)` de la classe `Chevalier` est une **surcharge** de la méthode `identite()` de `Chevalier` ?

6. Un autre type de personnage de jeu se distingue, c'est le `Magicien`. Sa puissance magique fait qu'à chaque fois que l'on incrémente ses points de vie avec `incremente`, cet incrément est doublé (ainsi `gandalf.incremente(3)` fait qu'il a en fait 6 points de vie en plus. En faisant comme pour la classe `Chevalier`, dérivez la classe `Magicien` de `PersonnageDeJeu` et spécialisez la méthode `incremente`. Attention, pour cela il faut que la classe `Magicien` puisse avoir accès à l'attribut `_pointsDeVie` de `PersonnageDeJeu` en le passant en mode protégé. De la même façon que pour `identite`, testez le polymorphisme de `incremente`.

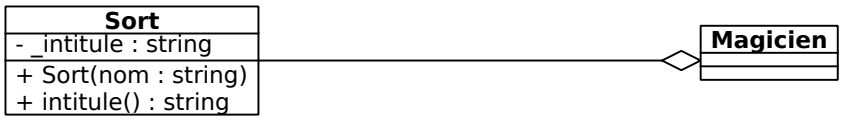
2 Association/Agrégation/Composition

1. Un `Chevalier` ne serait rien sans son fidèle destrier. Complétez votre programme en créant une classe `Cheval` (voir diagramme UML). Mettez en œuvre l'association en définissant les attributs nécessaires à ajouter aux classes `Cheval` et `Chevalier`. Instanciez une telle association entre un `Cheval` et un `Chevalier` dans votre programme principal.



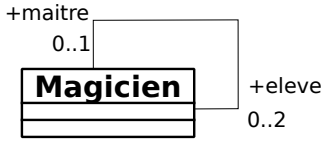
2. Un sort est souvent une recette associée à une phrase que le magicien prononce afin d'exercer un pouvoir magique sur les autres. Un bon magicien doit connaître de nombreux sorts. Parmi les sorts, on peut citer des sorts comme *rendre invisible*, *produire des boules de feux*, *nécromancie*,... Complétez votre programme en créant une classe

`Sort` (voir diagramme UML). Mettez en œuvre l'agrégation en définissant les attributs nécessaires à ajouter aux classes `Magicien` et `Sort`. Instanciez des sorts et un magicien pour tester votre agrégation.



3. On trouve également souvent un autre type de personnage, c'est le `Nain` dont la particularité est d'avoir une barbe (peu importe son Genre). Dérivez la classe `Nain` de la classe `PersonnageDeJeu` et créez la classe `Barbe` (voir le diagramme UML) pour mettre en œuvre la relation de composition suivante. La méthode `changeAge` de la classe `Personnage` doit être spécialisée car la longueur de la barbe du nain dépend de son âge (la longueur de barbe d'un nain est de `10 + age/10` selon les textes anciens...).

4. Mettez en œuvre l'association suivante :



5. En reprenant toutes les classes que vous avez mis en œuvre dans les TD1 et TD2, dessinez sur papier libre le diagramme UML de votre mise en œuvre.