COLLÈGE DE
BOIS-DE-BOULOGNE

Intra

SIM – H22 – Programmation orientée objet et Structures de données

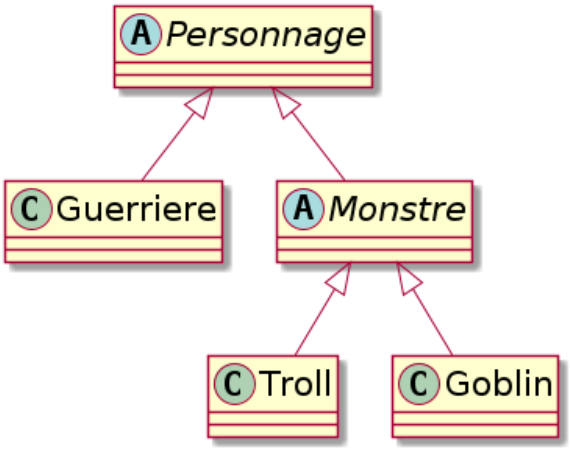
Pierre Prades, Georges Côté, Nicolas Hurtubise

Donjons et Dragons

Vous devez coder un programme qui simule le jeu D&D (Dungeons & Dragons), dans une version très simplifiée.

Le programme consiste à simuler un combat entre un groupe de valeureux personnages et une horde de monstres.

Vous devrez reproduire la structure de classes suivantes :



Les classes `Personnage` et `Monstre` sont des classes **abstraites**.

En plus des classes pour la hiérarchie de `Personnage`s, vous devez créer une classe `Main` pour tester votre code.

Classes à coder

Une classe *abstraite* Personnage, qui

- A un *attribut* pour son nom
- A un *attribut* pour son nombre de points de vie
- A un *attribut* de type `De` (classe fournie en annexe) qu'il ou elle peut utiliser pour prendre des décisions au hasard
- A une *méthode* `getPointsDegats()`, qui est abstraite (elle sera redéfinie dans les différentes sous-classes)
- A une *méthode* `estVivant()` qui retourne vrai si le nombre de points de vie est `> 0`
- A une *méthode* `attaquer(Personnage cible)` pour attaquer un autre Personnage (ie. lui causer un certain nombre de points de dégâts).

  En plus de modifier le nombre de points de vie de la cible, cette méthode doit afficher dans la console :

  `{nom du personnage} attaque {nom de la cible}!`

  Notez que si on appelle `attaquer(...)` avec un personnage déjà mort, rien ne devrait se passer
- A une *méthode* `recevoirDegats(int nbDegats)` pour recevoir un certain nombre de points de dégâts, qui diminue son nombre de points de vies

  Notez qu'il devrait être impossible d'avoir un nombre négatif de points de vie
- A une *méthode* `toString()` qui permet d'afficher

  `{Nom}: {nb points de vies} points de vie`

  Par exemple :

  `Victoria: 10 points de vie`

La classe `Personnage` a un constructeur qui doit recevoir le nom et le nombre de points de vies avec lequel commencer

Tous les personnages commencent avec leur propre objet `De` déjà instancié avec un `new De()`.

Une classe Guerrière, qui

- Doit redéfinir la méthode `getPointsDegats()` avec la logique suivante :

  D'habitude, son nombre de points de dégâts est de 3.

  Dans le cas où il lui reste 5 points de vie ou moins, elle entre dans une fureur : son nombre de points de dégâts est alors de 6.

➢ Dans votre `main()`, créez un tableau de 4 Guerrières avec dans l'ordre :

- Victoria : 10 points de vie
- Melanie : 15 points de vie
- Geri : 7 points de vie
- Emma : 18 points de vie

Affichez chacune d'entre-elles sur la console avec leur `toString()`

Une classe *abstraite* Monstre, qui

- Possède une méthode `rireMachiavelique()` qui affiche sur la console :

  `{nom du personnage}: Mouahahaha!`

- Doit redéfinir la méthode `getPointsDegats()` avec la logique suivante :

  Le monstre utilise son propre `De` pour le rouler avec 6 faces.

  Si le nombre obtenu est plus petit ou égal à 2, l'attaque est manquée, il fait 0 points de dégâts.

  Sinon, il fait 3 points de dégâts.

- Doit redéfinir la méthode `attaquer(Personnage cible)` avec la logique suivante :

  Attaquer la cible comme d'habitude (exactement comme le ferait la classe parent Personnage), *puis*

  Rouler un Dé à 3 faces (oui oui, je vous jure!) : si on obtient 3, on appelle la méthode `rireMachiavelique()`, sinon rien à faire de plus.

Une classe Goblin, qui

- Redéfinit la méthode : `rireMachiavelique()` pour afficher plutôt :

  ```
  Gnia! Gnia! Gnia!
  ```

- Commence *toujours* avec 9 points de vie.

  On ne doit **pas** passer le nombre de points de vie dans le constructeur de `Goblin`.

➢ Dans votre `main()`, ajoutez un Goblin nommé `"Twado"` pour tester

Faites-le attaquer la dernière guerrière de votre tableau de guerrières.

Ensuite, faites en sorte que chaque guerrière du tableau attaque *Twado*

Une classe Troll, qui

- A un attribut `niveau`, qui détermine sa force d'attaque :

  La méthode `getPointsDegats()` d'un `Troll` utilise la formule : `niveau / 2 + 3`

- Commence *toujours* avec 12 points de vie.

  On ne doit **pas** passer le nombre de points de vie dans le constructeur de `Troll`.

Notez : on doit passer le niveau du `Troll` dans le constructeur

➢ Dans votre `main()`, à la suite de ce que vous avez déjà, créez un tableau de 4 monstres, avec dans l'ordre :

- Joshua : un Troll de niveau 2
- Jareth : un Goblin
- Nevergonna Giveyouup : un Troll de niveau 7
- Sardius : un Goblin

Faites en sorte que chaque monstre attaque chaque guerrière, une à la fois.

Ajoutez une **méthode `static` à votre classe `Main`** pour afficher l'état d'un `Personnage` reçu en paramètre.

Cette méthode devrait afficher un emoji pour l'état du personnage, suivi de son `toString()` sur la même ligne.

L'emoji est :

- `:)` si le personnage est vivant
- `x(` si le personnage est mort

On aurait par exemple :

```
:) Joshua: 4 points de vie
```

Ou encore :

```
x( Joshua: 0 points de vie
```

➢ D'abord, utilisez cette méthode depuis votre `main()` pour afficher l'état des 4 guerrières.

Utilisez ensuite la méthode pour afficher l'état de *Twado* le Goblin, puis ensuite pour l'état des 4 monstres du tableau.

Précisions

La remise du code se fait **sur Léa**. Il ne doit **PAS** y avoir de `Scanner` dans votre code.

Écrivez de la JavaDoc seulement pour la méthode `attaquer(Personnage cible)`

Barème

- /5 Classe principale `Main`
  - Les classes sont testées comme demandé
  - La méthode pour afficher un personnage est correcte
- /20 Hiérarchie de classes bien définie, fonctionne comme demandé
  - /6 Classe `Personnage`
  - /4 Classe `Guerriere`
  - /5 Classe Monstre
  - /3 Classe Troll
  - /2 Classe Goblin
- /5 Qualité générale du code
  - JavaDoc pour la méthode `attaquer(Personnage cible)`
  - Respect des normes de codage
  - Respect de l'indentation
  - Respect de l'encapsulation