Héritage, appel de constructeurs, visibilité

Exercice 1 - Robot

On souhaite dans un premier temps modéliser des combats de robots. Un robot possède un nom et un nombre de points de vie compris entre 10 (parfaitement en vie) et 0 (totalement mort). Tous les robots commencent avec 10 points de vie à leur création et perdent des points de vie lors de combats.

1. Écrire une classe `Robot` dans le package `fr.umlv.fight` avec un constructeur prenant le nom du robot et de telle sorte que le code suivant affiche "Robot" suivi du nom du robot.

```
var bob = new Robot("bob");
System.out.println(bob);  // affiche "Robot bob"
```

2. Ajouter une méthode `fire` qui prend en paramètre un robot et qui permet au robot courant de tirer sur le robot pris en paramètre. Les robots étant parfaits, ils atteignent toujours leur cible, donc un robot qui se fait tirer dessus va toujours perdre 2 points de vie.
   Ajouter, de plus, une méthode `isDead` qui permet de savoir si un robot est mort.
3. Écrire dans une nouvelle classe `fr.umlv.fight.Arena` une méthode `fight` qui simule le combat entre deux robots.
   Un combat se déroule de la façon suivante, le premier robot tire sur le second, si le second n'est pas mort, celui-ci réplique et ce jusqu'à la mort d'un des robots. Le robot restant en vie est déclaré vainqueur et renvoyé en valeur de retour de la méthode `fight`.
   Pour suivre le déroulement du combat, modifier la méthode `fire` pour qu'à la fin de celle-ci, soit affiché le message:

```
"Robot foo a été touché par le Robot bar !"
```

   Écrire le code de la méthode `fight` ainsi qu'un `main` de test d'un combat entre `D2R2` et `Data`.

Exercice 2 - Fighter

En fait, les combats entre `Robot` ne sont pas intéressants car le premier robot qui tire gagne toujours. L'association sportive de tir de robot en accord avec les principes des droits de l'homme décide d'organiser des combats entre combattants humains et robots, les humains étant plus intéressants car une fois sur deux ils ratent leur cible.
On souhaite écrire une classe `Fighter` qui représente des combattants humains qui ont aussi un nom, un nombre de points de vie mais qui, lorsqu'ils tirent, ratent une fois sur deux.
Pour modéliser le fait qu'un humain rate un tir sur deux, on va utiliser la classe `java.util.Random` qui implante un générateur pseudo aléatoire et sa méthode `nextBoolean`.
Pour que tous les humains semblent avoir des comportements différents, à la création d'un `Fighter` en plus du nom, on indiquera un nombre qui va servir de graine (`seed`) au générateur pseudo-aléatoire.

1. Expliquer comment marche un générateur pseudo aléatoire et qu'est ce que la graine d'un générateur pseudo aléatoire ?
2. Écrire la classe `Fighter` de telle sorte que:
   - La classe `Fighter` hérite de la classe `Robot`
   - Que la méthode d'affichage affiche "Fighter" suivi du nom du combattant.
   - Que lorsqu'un humain fait feu avec la méthode `fire`, il ait une chance sur deux de rater sa cible.

   Note: on suppose ici qu'il n'est pas interdit d'avoir des champs avec une visibilité autre que `private` ou de package et qu'un peu de copier-coller ne va pas ruiner votre carrière naissante.
3. Expliquer pourquoi il ne faut `jamais` qu'un champ soit déclaré avec une visibilité autre que `private` ou de package.
   Modifier le code en conséquence.
4. Il y a un bug dans l'implantation de la classe `Robot`, il est possible qu'un robot tire sur un robot mort ce qui est strictement interdit par le code de déontologie de l'association sportive de tir de robot.
   Faire en sorte que, si un robot tente de tirer sur un robot mort, une exception soit levée (on vous laisse deviner laquelle :)

5. On peut remarquer que la méthode `fire` de la classe `Fighter` a le même problème. Que pouvez vous en conclure sur le copier-coller ?
6. Pour éviter le copier-coller de code, l'idée est d'introduire une méthode `rollDice` dans `Robot` qui modélise le fait qu'un robot a une chance sur un d'atteindre sa cible tandis qu'un combattant a une chance sur deux.
   Dans ce cas, la méthode `fire` peut être écrite une seule fois dans `Robot` car la méthode `rollDice` aura un code différent pour `Robot` et `Fighter`.
   Modifier le code pour introduire la méthode `rollDice`. Quelle doit être la visibilité de cette méthode ?
7. Vérifier que l'on peut bien faire un combat d'humain avec le code suivant

```
var john = new Fighter("John", 1);
var jane = new Fighter("Jane", 2);
System.out.println(Arena.fight(john, jane) + " wins");
```

8. Si cela n'a pas déjà été fait, faîtes en sorte que pour un `Fighter`, l'affichage soit "Fighter" suivi de son nom.
   Note: de même le champs `name` doit rester `private` !
9. En conclusion, rappeler ce qu'est le sous-typage et ce qu'est le polymorphisme en utilisant comme exemple les codes que vous venez d'écrire.

© Université de Marne-la-Vallée