

Concept. et prog. objet


\1 TP n°3 : exceptions

Dépublié

Détails

Écrit par stéphane Domas

Catégorie : M3105 - Concept. et prog, objet avancée (/index.php/menu-cours-s3/menu-mmi3-test)

 Publication : 23 octobre 2014

 Affichages : 1479

Préambule

- Comme abordé en cours, la classe `Humain` représente concept. On est donc censé se poser la question de si la classe doit être abstraite ou non.
- Si on applique la règle "utilitaire", à savoir "pas besoin de l'instancier dans l'application => abstraite", alors effectivement, la classe `Humain` est abstraite.
- Reprenez les sources du TP n°2 en renommant `TP2.java` par `TP3.java`
- Modifiez l'entête de la classe `Humain` pour la rendre abstraite.

1°/ Rencontre impossible = exception

- Dans les TP n°1 et 2, quand deux humains `h1` et `h2` sont pris dans la population, la rencontre ne peut donner un bébé que si les deux humains sont de sexes opposés.
- Si ce n'est pas le cas, la méthode de rencontre renvoie un objet `null`.
- Pour bien faire, il vaudrait mieux générer une exception car c'est un cas d'erreur.
- Pour cela, créez le fichier `BreedingForbiddenException.java` à partir du code suivant :

```
1  class BreedingForbiddenException extends Exception {
2
3      protected Humain[] source;
4
5      public BreedingForbiddenException(Humain h1, Humain h2) {
6          super("naissance impossible : "+h1.getNom()+" et "+h2.getNom()+" sont de meme sexe");
7          source = new Humain[2];
8          source[0] = h1;
9          source[1] = h2;
10     }
11
12     public Humain[] getHumain() {
13         return source;
14     }
15 }
```

- Modifiez les classes `Humain`, `Homme`, `Femme` et `TP3` afin d'utiliser l'exception définie ci-dessus lors de rencontres : s'il n'y a pas d'exception, on insère le bébé dans la population, sinon on affiche le message contenu dans l'exception.

2°/ Rencontre non productive = exception

- Si vous avez suivi à la lettre les indications de l'exercice 1, vous devriez avoir une exécution qui se termine sur `NullPointerException`.
- Cela vient simplement du fait que les deux humains sont de sexes opposés mais les conditions (sur le poids, age, ...) ne sont pas respectées ou bien le tirage aléatoire sur la fertilité a été négatif.
- Dans ces deux cas, il n'y a pas de nouvel humain et la valeur renvoyée est `null`, d'où l'exception.
- Pour éviter ce problème, on va :

- générer une `BreedingForbiddenException` quand les conditions sur le poids, age, ... ne sont pas respectées
- générer une `NoBreedingException` quand le tirage sur la fertilité est négatif.
- Pour le deuxième cas, créez un fichier `NoBreedingException.java` à partir du code suivant :

```

1  class NoBreedingException extends Exception {
2
3      protected Humain source;
4
5      public NoBreedingException(Humain h) {
6          super("rencontre improductive : "+h.getNom()+" n'est pas fertile");
7          source = h;
8      }
9
10     public Humain getHumain() {
11         return source;
12     }
13 }

```

- Dans `BreedingForbiddenException`, modifiez le code pour avoir un message différent selon le cas (NB: cela impose de redéfinir la méthode `getMessage()`) :
- - même sexe : le message est le même que dans l'exercice 1
 - conditions d'âge et/ou poids non respectées : message donnant les conditions non conformes. Par exemple : "naissance impossible : toto est trop jeune, tutu est trop vieille, tutu est trop gros"
- Modifiez `Humain`, `Homme`, `Femme` et `TP3` pour utiliser les deux classes d'exception.

3°/ Une super-classe pour les exceptions de rencontre.

- Créez le fichier `MeetingException.java` à partir du code suivant :

```

1  class MeetingException extends Exception {
2
3      protected Humain[] source;
4
5      public MeetingException(Humain h1, Humain h2) {
6          super("problème de rencontre");
7          source = new Humain[2];
8          source[0] = h1;
9          source[1] = h2;
10     }
11
12     public Humain[] getHumain() {
13         return source;
14     }
15 }

```

- Modifiez les deux classes d'exception pour qu'elles héritent de cette classe.

4°/ la propagation des exceptions

- Le coeur du jeu de la vie est pour l'instant dans `TP3`. Or, ce sont les humains d'une population que l'on manipule. On pourrait donc mettre une partie du code de la classe `TP3` dans la classe `Population`.
- Pour cela, modifiez la classe `Population` en ajoutant une méthode `rencontre` :

```
1 public Humain rencontre(int index1, int index2) throws BreedingForbiddenException, NoBreedingE
2     Humain h1 = getHumain(index1);
3     Humain h2 = getHumain(index2);
4     return h1.rencontre(h2);
5 }
```

- Modifiez la classe TP3 pour appeler cette méthode pour faire les rencontres.
- On remarque que la méthode ne fait pas de try/catch. Or, elle génère potentiellement des exceptions.
- Le fait d'ajouter `throws ...` dans l'entête va provoquer la propagation des éventuelles exceptions et c'est donc dans la méthode `main()` de TP3 que l'on doit faire le try/catch.