Exercice 1 – Fourmis

Q 1.1 Classe `Fourmi` (la base)
Les fourmis ont un `nom` (`String`) et un attribut `estReine` qui vaut `true` si l'instance est une reine et `false` sinon. Toutes les fourmis qui ne sont pas des reines sont des ouvrières.

Q 1.2 Donner le nom du fichier dans lequel développer la classe `Fourmi`.

Q 1.3 Donner le code de la classe `Fourmi`, déclarer les attributs puis implémenter les méthodes permettant de :
- construire une `Fourmi` avec un nom et un booléen indiquant s'il s'agit d'une reine,
- construire une `Fourmi` sans argument
  - dans 1% des cas, il s'agit d'une reine, sinon, c'est une ouvrière
  - son nom donne son type (reine/ouvrière) et un entier aléatoire tiré entre 0 et 999
- récupérer le nom de la fourmi, récupérer s'il s'agit d'une reine ou pas

Les signatures de méthodes sont volontairement NON données... A vous de vous débrouiller.

Q 1.4 Méthodes avancées : ajouter le code des méthodes permettant de :
- cloner une fourmi
- construire une chaine de caractères indiquant le nom et le type de la fourmi

Q 1.5 Que penser de l'usage de l'instruction `this(...)` dans le constructeur de fourmi sans argument ? Est-ce une bonne ou une mauvaise idée ? Pourquoi ?

Exercice 2 – Classe `Oeuf` et élargissement de la classe `Fourmi`

Nous allons maintenant avancer dans la simulation en permettant aux reines de pondre des œufs. Ces œufs pourront éclore pour donner des fourmis.

Q 2.1 Donner le code de la classe `Oeuf`. Un oeuf possède dès sa création un nom et un type de fourmi.
- Ecrire un constructeur à deux arguments,
- une méthode de création de chaine de caractères,
- une méthode `eclore` qui retourne une `Fourmi` initialisée avec le nom et le type présents dans l'œuf.

Q 2.2 Donner le code à ajouter dans la classe `Fourmi` pour que les reines puissent pondre un œuf
- méthode `pondre` sans argument,
- si la fourmi n'est pas une reine, afficher un message d'erreur et ne rien faire d'autre,
- sinon, tirer le type de l'œuf aléatoirement (1%/99%) et créer l'œuf avec le même nom que la reine, augmenté de `"_ouv"` ou `"_reine"` en fonction de son type.

Note : réfléchir à ce que doit retourner la méthode en général (signature) et à ce qu'il convient de faire dans le cas d'une ouvrière.

Q 2.3 Donner les instructions de compiler les classes `Fourmi` et `Oeuf`. Peut-on exécuter un des fichiers compilé ? Si oui, dire lequel et comment, sinon, dire pourquoi.

Exercice 3 – `main`

Q 3.1 Donner la signature d'une classe et d'une méthode `main` pour tester nos classes.

Q 3.2 Lors de l'exécution du code ci-dessous, combien d'instances ont été créée en tout ? A la fin de l'exécution du code combien en reste-t-il ? Justifier brièvement.

```
1 Fourmi f1 = new Fourmi("toto", false);   Fourmi f2 = new Fourmi();
2 Fourmi f3 = f1;                          Fourmi f4 = new Fourmi("Queen Mary II", true);
3 f2 = f4;                                 f1 = f4;
```

Q 3.3 Même question sur les instructions suivantes (question indépendante de la précédente)

```
1 Fourmi f5 = new Fourmi("Queen Elizabeth", true);        Fourmi f6 = f5.clone();
2 Oeuf o1 = f6.pondre();                                  Fourmi f7 = o1.eclore();
3 Fourmi f8 = f7;                    f7 = f5;             f5 = f6;
```

Q 3.4 Donner les instructions (création d'instances, manipulations, affichages) permettant de tester le bon fonctionnement des fourmis et des œufs.