Méthodes statiques dans les interfaces (Java >= 8)

J. Sam, J.-C. Chappelier

version 1.0 de novembre 2015

1 Méthodes statiques d'interfaces

Avant Java 8, il était possible de définir des constantes dans une interface. Il s'agit de variables qui sont *de facto* `final` et `static`.

Il est désormais possible de définir aussi des *méthodes* statiques :

```
package mesUtitaires;

interface Utilitaires {
    static double  distance (Point p1, Point p2) {
        return Math.sqrt(p1.x() * p1.x() + p2.y() * p2.y());
    }
}
```

Les interfaces sont donc désormais aussi un moyen de regrouper une collection d'utilitaires.

La syntaxe pour la définition de méthodes statiques dans une interface est identique à celle utilisée pour de tels membres dans les classes.

Pour utiliser une méthode statique d'interface dans une classe, il est bien sûr possible de mettre en place un lien `implements`, comme dans l'exemple suivant :

```
class Habitation implements Utilitaires {
    private final double FAIBLE_DISTANCE = 5.0;
    private Point position;

    public boolean procheCentre(Point centreVille) {
        return Utilitaires.distance(position, centreVille)
               < FAIBLE_DISTANCE;
    }
}
```

Ce n'est cependant pas très naturel car il n'existe pas de lien *fonctionnel* entre la classe est l'interface : la classe ne se doit pas d'implémenter des fonctionnalités édictées par l'interface [1]. On peut dans ce cas, avantageusement recourir à la notion d'importation statique présentée ci-dessous.

2 Importation statique

Le mot-clef `import` [2] peut aussi être suivi du mot-clef `static` pour importer les noms des membres statiques de classes (ou interfaces) sans devoir nommer à chaque fois la classe (ou l'interface) en question . On nomme cela **importation statique**.

```
import static mesUtilitaires.Utilitaires.distance;
class Habitation {
    private final double FAIBLE_DISTANCE = 5.0;
    private Point position;
    public boolean procheCentre(Point centreVille) {
        return distance(position, centreVille) < FAIBLE_DISTANCE;
    }
}
```

L'astérisque peut également être utilisée pour importer la totalité des membres statiques ;

```
import static mesUtilitaires.Utilitaires.*;
```

Attention, ce type d'importation n'a rien à voir avec les importations classiquement faites avec les paquetages !

L'importation statique de membres de même noms depuis des classes ou interfaces différentes est source de **conflit**.

Le code suivant par exemple :

```
import static mesUtilitaires.Utilitaires.distance;
import static utilitairesGeometriques.Geometrie.distance;

class Habitation {
    private final double FAIBLE_DISTANCE = 5.0;
    private Point position;
    public boolean procheCentre(Point centreVille) {
        return distance(position, centreVille) < FAIBLE_DISTANCE;
    }
}
```

1. ce qui est le rôle naturel des interfaces
2. vu dans le complément sur les paquetages

fera émettre le message `reference to distance is ambiguous` au compilateur

Dans cet exemple, il existe deux interfaces distinctes, `Utilitaires` et `Geometrie` qui contiennent toutes deux une méthode statique nommée `distance`.

Il faudrait donc dans ce cas explicitement lever l'ambiguïté :

```
import static mesUtilitaires.Utilitaires.distance;
import static utilitairesGeometriques.Geometrie.distance;

class Habitation {
    private final double FAIBLE_DISTANCE = 5.0;
    private Point position;
    public boolean procheCentre(Point centreVille) {
        return mesUtilitaires.Utilitaires.distance(position, centreVille)
               < FAIBLE_DISTANCE;
    }
}
```

Ce document vous a présenté l'essentiel des particularités liées à la définition et à l'utilisation de méthodes statiques dans un interface.

Avec cette nouveauté de Java 8, les interfaces, puisque non instantiables en tant que telles, deviennent sans doute le moyen le plus naturel de regrouper des méthodes utilitaires.