Interface par défaut

Cas d'usage, le comportement par défaut

Le premier cas d'usage est la mise en place d'un comportement par "défaut" pour les objets

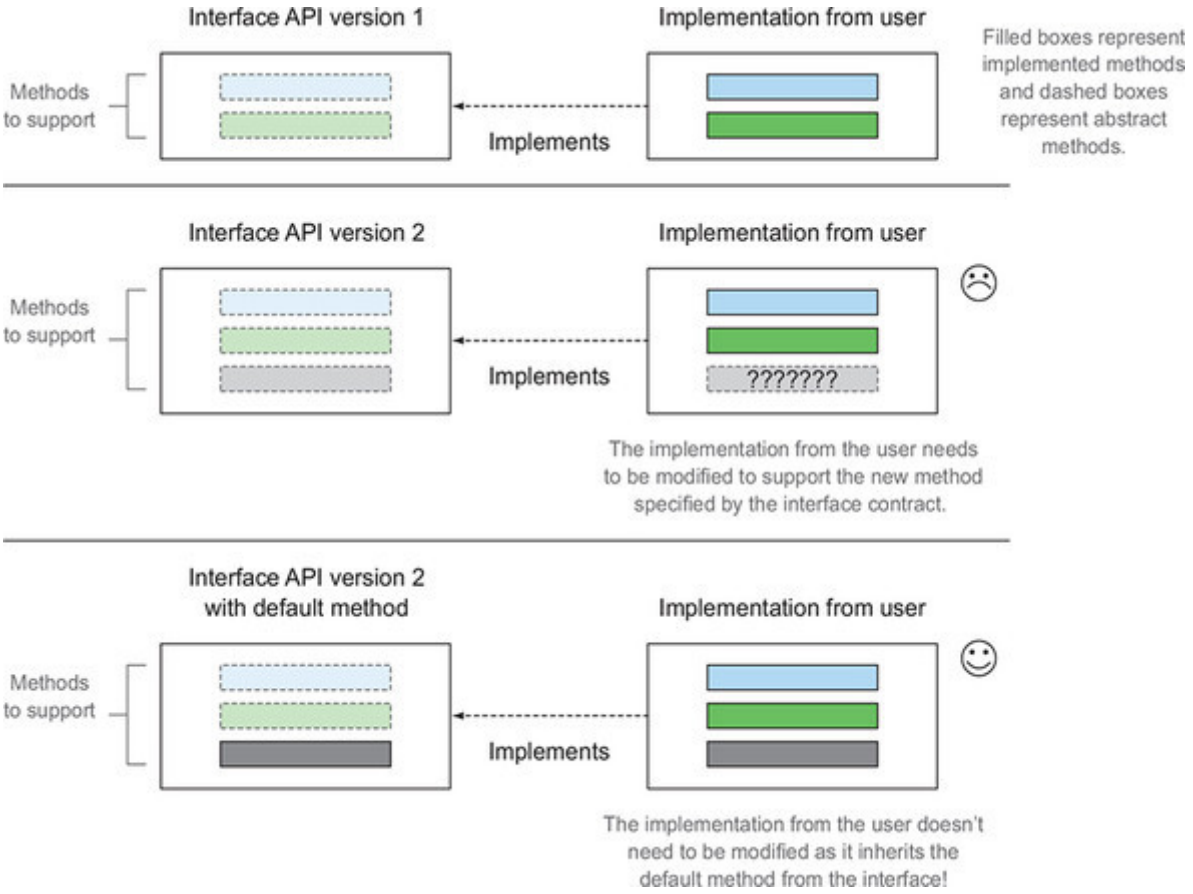
```
public interface MyInterface {

    // regular interface methods

    default void defaultMethod() {
        // default method implementation
    }
}
```

La raison pour laquelle la version Java 8 incluait des méthodes par défaut est assez évidente.

Dans une conception typique basée sur des abstractions, où une interface a une ou plusieurs implémentations, si une ou plusieurs méthodes sont ajoutées à l'interface, toutes les implémentations seront obligées de les implémenter également. Sinon, la conception s'effondrera.

Les méthodes d'interface par défaut sont un moyen efficace de traiter ce problème. Ils nous permettent d'ajouter de nouvelles méthodes à une interface qui sont automatiquement disponibles dans les implémentations. Par conséquent, nous n'avons pas besoin de modifier les classes d'implémentation.

De cette façon, la rétrocompatibilité est soigneusement préservée sans avoir à refactoriser les implémentations.



La problématique du diamand est soulevé ainsi:

```
public interface Vehicle {

    String getBrand();

    String speedUp();

    String slowDown();

    default String turnAlarmOn() {
        return "Turning the vehicle alarm on.";
    }

    default String turnAlarmOff() {
        return "Turning the vehicle alarm off.";
    }
}
public interface Alarm {

    default String turnAlarmOn() {
```

```
        return "Turning the alarm on.";
    }

    default String turnAlarmOff() {
        return "Turning the alarm off.";
    }
}
```

Il faut si on veux hériter des deux interface explicitement implémenté les méthodes quitte a rappeler les methodes supérierus

```
@Override
public String turnAlarmOn() {
    return Vehicle.super.turnAlarmOn();
}

@Override
public String turnAlarmOff() {
    return Vehicle.super.turnAlarmOff();
}
```

Methode statique

En plus de déclarer des méthodes par défaut dans les interfaces, Java 8 nous permet également de définir et d'implémenter des méthodes statiques dans les interfaces.

Puisque les méthodes statiques n'appartiennent pas à un objet particulier, elles ne font pas partie de l'API des classes implémentant l'interface ; par conséquent, ils doivent être appelés en utilisant le nom de l'interface précédant le nom de la méthode.

Pour comprendre le fonctionnement des méthodes statiques dans les interfaces, refactorisons l'interface Vehicle et ajoutons-y une méthode utilitaire statique :

```
public interface Vehicle {

    // regular / default interface methods
```

```
    static int getHorsePower(int rpm, int torque) {
        return (rpm * torque) / 5252;
    }
}
```

Revision #1
Created 8 June 2022 17:42:49 by ggpilou2
Updated 19 June 2022 10:52:23 by ggpilou2