Interface *vs.* Classe Abstraite

BUT Informatique et Licence Professionnelle

V. Deslandres

IUT Univ. Lyon1

Sommaire

- 5 pages de présentation des différences entre **interface** et **classe abstraite**
  - En Java
- Le reste est un Quizz de 12 questions, avec les réponses
  - Récompense à la fin ☺

DEFINITIONS : interface

Interface (d’une classe) :

- C’est l’ensemble des **signatures des méthodes** d’une classe ou un framework
- Définit ce qu’un objet (ou package) peut faire (pas comment il le fait)

DEFINITIONS : interface

Interface (en tant qu'entité isolée en Java) :

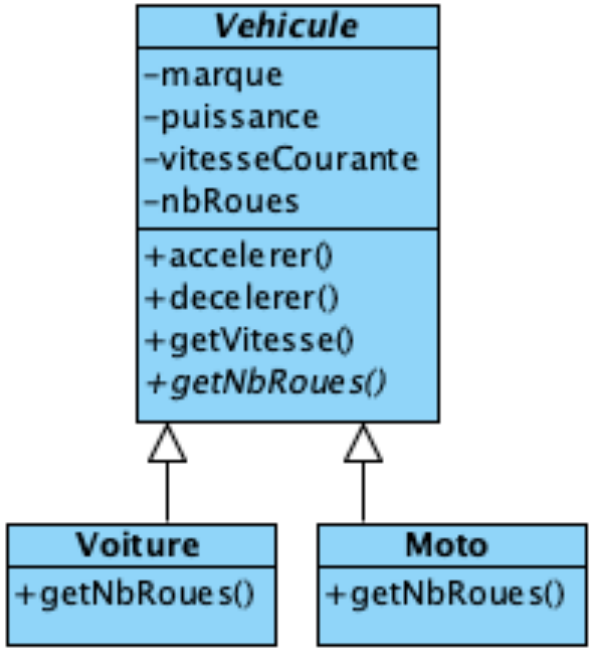
- Définit un ensemble de **méthodes** (sans leur implémentation*)
- Implémentée dans un autre objet (ou plusieurs autres objets)
- **N'a pas d'attribut**, que des constantes `static final`
- Toutes ses méthodes sont implicitement `public`
- Ne **peut pas être instanciée**
- Une classe peut implémenter **plusieurs interfaces**
- On peut élaborer une **arborescence d'interfaces** avec l'héritage
- Synonymes: services, contrat des méthodes d'une classe

* *Sauf depuis Java8, avec le mot-clef default : code par défaut d'une méthode pour permettre la compatibilité ascendante avec du legacy code d'interfaces de l'API Java qui ont été modifiées). Ajout aussi de méthodes statiques.*

DEFINITION : Classe abstraite

- Classe abstraite
  - Une classe **comme une autre** qui possède **au moins** une méthode abstraite
  - Peut définir des constructeurs, mais ne peut pas être instanciée
    - Les constructeurs sont utilisés par les sous-classes : `super(param);`
  - Une classe quelconque ne peut hériter **que d'une seule classe abstraite**

- Exemple : une classe Vehicule abstraite
  - Méthodes concrètes : *accelerer(), decelerer(),* etc.
  - Méthodes **abstraites** : *getNbRoues()*



Même principe, des objectifs différents

- Les deux entités reposent sur des méthodes ABSTRAITES
- Leurs objectifs sont différents :
  - Une classe abstraite définit une partie **commune** à une arborescence d'objets
    - Les méthodes abstraites sont spécialisées dans les sous-classes
    - Cela permet de définir des comportements spécifiques
  - Une interface expose un contrat type, un ensemble de services qui seront réalisés par les classes qui l'implémenteront
    - Couplage faible : une classe cliente utilise des objets implémentant une interface `Sortable` et sait donc qu'elle peut trier ces objets. Peu importe comment le tri est effectué.
    - Analogie : la prise de courant, son contrat = fournir du courant alternatif; comment ? pour faire quoi ? Peu importe.

Utilisation des interfaces en POO

- Un des avantages de la programmation orienté objet est le **polymorphisme**
  - En plus de **l'encapsulation** et **l'héritage**
- Le **polymorphisme** permet d'exploiter les interfaces très facilement.
- En effet très souvent, on définit un type de référence comme étant une interface, qu'on instancie ensuite du type réel requis.
  - Ex.: `List<Personne> maListe;` // *List* est une interface en Java
  - On peut ensuite instancier maListe en tant qu' `ArrayList` ou `Stack`, selon les besoins : `maListe = new Stack<Personne>;`
  - Mais on sait que l'on pourra utiliser `contains(unePersonne)` par exemple, sur toute la liste, qu'elle soit implémentée sous forme de tableau ou de pile.

Rappel : `List<Personne> maListe = new ArrayList<>;`

Type de référence

Type réel

Quizz : comptez-vos points !

1. En Java, il n’est pas possible de combiner **abstract** et **final** dans la déclaration d’une classe

2. Les méthodes d’un interface sont abstraites, même si on omet le mot-clef **abstract**

3. Les méthodes d’un interface sont toujours **public**, même sans le mot-clef explicite

Quizz (suite)

4. Une interface est toujours **publique**, même sans le mot-clef explicitement mentionné

5. On peut créer des sous-interface qui **extends** une interface

6. Une interface doit toujours comporter **au moins une méthode**

Quizz (suite)

7. Une interface **peut définir des attributs** qui seront instanciés dans les classes de réalisation

8. Une **interface** peut définir des constructeurs qui seront implémentés dans les classes de réalisation

9. Une **classe abstraite** peut définir des constructeurs qui seront utilisés dans ses sous-classes

Quizz (fin)

10. Quel est **l'avantage** pour une classe de dépendre d'une **interface** plutôt que d'une classe concrète ?

11. **Comment s'appelle** ce type de dépendance ?

12. Quand on implémente une **interface**, on ne peut pas implémenter une méthode **à vide**, toutes les méthodes doivent exposer un code de réalisation

Réponses

1. En Java, il n’est pas possible de combiner **abstract** et **final** dans la déclaration d’une classe

VRAI - Cela n’aurait aucun sens puisqu’on ne peut pas instancier une classe abstraite, elle « n’existe » que par ses sous-classes.

2. Les méthodes d’un interface sont abstraites, même si on omet le mot-clef **abstract**

VRAI

3. Les méthodes d’un interface sont toujours **public**, même sans le mot-clef explicite

VRAI

4. Une interface est toujours **publique**, même sans le mot-clef explicitement mentionné

FAUX – Sans **public**, la portée de l'interface est limitée au package

5. On peut créer des sous-interface qui **extends** une interface

VRAI, l’héritage d’interfaces existe et est même souvent utilisée

6. FAUX, ça s’appelle une *interface de marquage (Java 1)*.

Ex.: **Cloneable.** La méthode **clone()** se trouve dans Object. **Cloneable** n’a aucune méthode, elle signifie juste que la classe implémente effectivement **clone().**

Sans implémenter Cloneable(), l’utilisation de clone() peut compiler mais faire des erreurs lors de l’exécution (fonctionnement de la MVJ non standard).

7. Une interface **peut définir des attributs** qui seront instanciés dans les classes de réalisation

FAUX – Les seuls attributs possibles sont des constantes (**static final**)

8. Une **interface** peut définir des constructeurs qui seront implémentés dans les classes de réalisation

FAUX, aucun constructeur puisqu’une interface ne sera jamais instanciée.

9. Une **classe abstraite** peut définir des constructeurs qui seront utilisés dans ses sous-classes

VRAI, avec le mot-clef **super**

10. Quel est **l'avantage** pour une classe de dépendre d'une **interface** plutôt que d'une classe concrète ?

Cela limite l'interaction avec l'interface. Comme la classe ne dépend que des **signatures** des méthodes, peu importe si le code d'implémentation évolue, tout ce que la classe sait, c'est que les méthodes pourront être appelées.

11. **Comment s’appelle** ce type de dépendance ?

On parle de **couplage faible** quand une classe dépend d’un interface ou d’une classe abstraite.

12. Quand on implémente une **interface**, on ne peut pas implémenter une méthode **à vide**, toutes les méthodes doivent exposer un code de réalisation

FAUX, on peut implémenter une méthode et qu'elle **ne fasse rien**.

Ex. en GUI: *WindowAdapter*, qui implémente toutes les méthodes de *WindowsListener* à vide.

L'objectif est de permettre aux classes de réalisation de *WindowAdapter* de n'implémenter **que les méthodes dont elle a besoin**, pas les 10 méthodes de l'interface *WindowsListener*.

Résultats

- **Moins de 7 points** : mieux vaut revoir le cours !

- **Entre 8 et 10 points** : pas mal !

- **Plus de 11 points** : bravo, vous avez droit à une glace