

Les interfaces

Principes

Si on considère une **classe abstraite** qui ne décrit que des méthodes abstraites (aucun attribut, aucune méthode non abstraites), on aboutit à la notion d'**interface**.

Une interface décrit les signatures d'un certain nombre de **méthodes**, ainsi que de **constantes**, dont on souhaite imposer l'existence dans certaines classes ;

Exemple :

```
// une interface qui décrit les comportements d'un objet déplaçable
public interface Movable {
    public void moveTo (Point position);
    public void moveTo (int x_position, int y_position);
}
```

Usage :

```
// En implémentant l'interface Movable, la classe Rectangle doit
necessarily
// implémenter et définir le contenu des méthodes inscrites dans l'interface
Movable
public class Rectangle implements Movable {
    public void moveTo (Point position) {
        ...
    }
    public void moveTo (int x_position, int y_position) {
        ...
    }
}
```

Cela ressemble beaucoup à une hybridation entre :

- héritage (**implements** en lieu et place de **extends**) ;
- et classe abstraite.

Pourtant, ce n'est ni vraiment l'un, ni vraiment l'autre.

Règles de construction

- Le mécanisme sous-jacent est différent d'un simple cas particulier de classe abstraite. Il se distingue aussi franchement de l'héritage en le rendant plus riche : **une classe peut implémenter plusieurs interfaces alors qu'une classe ne peut pas hériter plus d'une fois** (héritage simple) ;
- **Implémentation d'interface et extension par héritage peuvent se combiner** ;

- Par définition, **les méthodes d'une interface sont abstraites** ;
- Une interface est perçue par l'environnement comme un type à part entière. Il est donc possible de déclarer une variable d'un type d'interface ;

Bénéfices

Une interface permet de définir un ensemble de services « contractuels » dont on veut être certain qu'une classe les fournira. La classe est libre de l'implémentation (comment est réalisé le service) mais pas du contrat (la surface d'échange : paramètres et retour). En Java, typer une donnée **List** (qui est une Interface disponible dans le JDK) permet d'accepter différentes sortes de collections (celles qui implémentent **List**) et de les traiter indistinctement par le fait que les fonctionnalités de base de ces collections sont les mêmes. Comme le mécanisme d'interface est absolument indépendant de l'héritage, il est possible d'implémenter une même interface dans des classes distinctes qui ne partagent rien (pas de filiation, pas d'ADN commun), mais ont pourtant des comportements similaires.

Illustrations : API Java

The screenshot shows the Java API documentation for the `JTextField` class. The top navigation bar includes links for Overview, Package, Class (which is selected), Use, Tree, Deprecated, Index, and Help. Below the navigation bar, there are links for Prev Class, Next Class, Frames, No Frames, and All Classes. Further down are links for Summary, Nested, Field, Constr, Method, Detail, Field, Constr, and Method.

The main content area is titled "javax.swing" and "Class JTextField". It lists the class's inheritance path: `java.lang.Object`, `java.awt.Component`, `java.awt.Container`, `javax.swing.JComponent`, `javax.swing.text.JTextField`, and finally `javax.swing.JTextField`. A red box highlights the word "Abstract" in the inheritance chain, with a callout "Classe abstraite".

Below the inheritance tree, under "All Implemented Interfaces:", are listed `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `Scrollable`, and `SwingConstants`. A red box highlights this section, with a callout "Les interfaces implementées par JTextField ou ses descendants et dont il hérite".

Under "Direct Known Subclasses:", `DefaultTreeCellEditor.DefaultTextField`, `JFormattedTextField`, and `JPasswordField` are listed. A red box highlights this section, with a callout "Les descendants de JTextField".

At the bottom, the class definition is shown:

```
public class JTextField
extends JComponent
implements SwingConstants
```

A red box highlights the implements clause, with a callout "En fait, JTextField n'implémente en direct qu'une seule interface. Les autres sont héritées."

The page footer contains the text: "JTextField is a lightweight component that allows the editing of a single line of text. For information on and examples of using text fields, see How to Fields in The Java Tutorial."

docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY NESTED FIELD CONSTR METHOD DETAIL FIELD CONSTR METHOD

compact1, compact2, compact3
java.util

Class ArrayList<E>

java.lang.Object
 java.util.AbstractCollection<E>
 java.util.AbstractList<E>
 java.util.ArrayList<E>

All Implemented Interfaces:
Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:
Attributelist, Rolelist, RoleUnresolvedList

Classe abstraite

Les ascendants successifs de la classe ArrayList.

Les interfaces implémentées par ArrayList ou ses ascendants et dont il hérite.

Les descendants de ArrayList

Il en implemente 3 et en hérite donc de 3

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

Resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to the Vector class, except that it is unsynchronized.)

Les Classes abstraites et interfaces sont largement employées dans la conception des bibliothèques graphiques (les composants graphiques SWING, par exemple) et des bibliothèques de classes techniques (les collections, par exemple).

From:
<https://wiki.siochaptalqper.fr/> - Wiki SIO Chaptal



Permanent link:
<https://wiki.siochaptalqper.fr/doku.php?id=bloc2:prog:poo:interfaces&rev=1710429963>

Last update: 2024/03/14 16:26