

# Les interfaces

## Principes

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

Une interface décrit les signatures d'un certain nombre de **méthodes**, et possiblement aussi 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
nécessairement
// 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 se distingue de l'héritage en le rendant plus riche : **une classe peut implémenter plusieurs interfaces alors qu'elle ne peut hériter qu'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 `javax.swing.JTextField`. The page includes a navigation bar with tabs for OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. Below the navigation bar, there are links for PREV CLASS, NEXT CLASS, FRAMES, NO FRAMES, and ALL CLASSES. The main content area is divided into several sections:

- Class Hierarchy:** A tree view showing the inheritance path: `java.lang.Object`, `java.awt.Component`, `java.awt.Container`, `javax.swing.JComponent`, `javax.swing.text.JTextComponent`, and `javax.swing.JTextField`. A red box labeled "Classe abstraite" has arrows pointing to `JComponent`, `JTextComponent`, and `JTextField`.
- All Implemented Interfaces:** A list of interfaces: `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `Scrollable`, and `SwingConstants`. A red box labeled "Les interfaces implémentées par JTextField ou ses ascendants et dont il hérite." has an arrow pointing to this list.
- Direct Known Subclasses:** A list of subclasses: `DefaultTreeCellEditor.DefaultTextField`, `JFormattedTextField`, and `JPasswordField`. A red box labeled "Les descendants de JTextField" has an arrow pointing to this list.
- Class Declaration:** The code snippet: 

```
public class JTextField extends JTextComponent implements SwingConstants
```

. A red box labeled "En fait, JTextField n'implémente en direct qu'une seule interface. Les autres sont héritées." has an arrow pointing to `implements SwingConstants`.

At the bottom of the page, there is a brief description: "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 Use Text Fields in The Java Tutorial."

The screenshot shows the Oracle Java API documentation for the `ArrayList` class. The page includes navigation tabs (OVERVIEW, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, HELP) and a breadcrumb trail (PREV CLASS, NEXT CLASS, FRAMES, NO FRAMES, ALL CLASSES). The main content area lists the class hierarchy: `compact1, compact2, compact3`, `java.util`, **Class `ArrayList<E>`**, `java.lang.Object`, `java.util.AbstractCollection<E>`, `java.util.AbstractList<E>`, and `java.util.ArrayList<E>`. It also lists implemented interfaces (`Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, `RandomAccess`) and direct known subclasses (`AttributeList`, `RoleList`, `RoleUnresolvedList`). The source code snippet shows `public class ArrayList<E>` extending `AbstractList<E>` and implementing `List<E>`, `RandomAccess`, `Cloneable`, and `Serializable`. Annotations with red arrows and boxes explain: 'Classe abstraite' points to `AbstractList`; 'Les ascendants successifs de la classe ArrayList.' points to the hierarchy; 'Les interfaces implémentées par ArrayList ou ses ascendants et dont il hérite.' points to the interface list; 'Les descendants de ArrayList' points to the subclasses; and 'Il en implémente 3 et en hérite donc de 3' points to the `ArrayList` class declaration.

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=1710430314>

Last update: 2024/03/14 16:31

