

# 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 IMovable {
    public void moveTo (Point position);
    public void moveTo (int x_position, int y_position);
}
```

Usage :

```
// En implémentant l'interface IMovable, la classe Rectangle doit
nécessairement
// implémenter et définir le contenu des méthodes inscrites dans l'interface
IMovable
public class Rectangle implements IMovable {
    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).



Exemple : 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

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).

The screenshot shows the Java API documentation for the `JTextField` class. The page is titled "Class JTextField" and lists its inheritance hierarchy: `java.lang.Object`, `java.awt.Component`, `java.awt.Container`, `javax.swing.JComponent`, `javax.swing.text.JTextComponent`, and `javax.swing.JTextField`. It also lists "All Implemented Interfaces": `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `Scrollable`, and `SwingConstants`. Under "Direct Known Subclasses", it lists `DefaultTreeCellEditor`, `DefaultTextField`, `JFormattedTextField`, and `JPasswordField`. The source code snippet shows `public class JTextField extends JTextComponent implements SwingConstants`. Red annotations with arrows point to these elements: "Classe abstraite" points to the class name; "Les ascendants successifs de la classe JTextField." points to the inheritance list; "Les interfaces implémentées par JTextField ou ses ascendants et dont il hérite." points to the implemented interfaces; "Les descendants de JTextField" points to the subclasses; and "En fait, JTextField n'implémente en direct qu'une seule interface. Les autres sont héritées." points to the `implements` clause in the code snippet.

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

---

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 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 equ except that it is unsynchronized.)

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

Permanent link:  
<https://wiki.siochaptalqper.fr/doku.php?id=bloc2:prog:poo:interfaces&rev=1710431137>

Last update: **2024/03/14 16:45**

