

# Les interfaces

## Principes

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
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 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 `javax.swing.JTextField`. Red arrows and boxes highlight key aspects:

- Classe abstraite**: Points to the `JTextField` class name.
- Les ascendants successifs de la classe JTextField.**: Points to the inheritance chain: `java.lang.Object`, `java.awt.Component`, `java.awt.Container`, `javax.swing.JComponent`, `javax.swing.text.JTextComponent`, and `javax.swing.JTextField`.
- Les interfaces implémentées par JTextField ou ses ascendants et dont il hérite.**: Points to the "All Implemented Interfaces" list: `ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `Scrollable`, and `SwingConstants`.
- Les descendants de JTextField**: Points to the "Direct Known Subclasses" list: `DefaultTreeCellEditor.DefaultTextField`, `JFormattedTextField`, and `JPasswordField`.
- En fait, JTextField n'implémente en direct qu'une seule interface. Les autres sont héritées.**: Points to the `implements SwingConstants` line in the class declaration.

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

`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<E>` class. Red arrows and boxes highlight key aspects of its design:

- Classe abstraite**: Points to the `java.util.ArrayList<E>` class name.
- Les ascendants successifs de la classe ArrayList.**: Points to the inheritance chain: `java.lang.Object`, `java.util.AbstractCollection<E>`, `java.util.AbstractList<E>`, and `java.util.ArrayList<E>`.
- Les interfaces implémentées par ArrayList ou ses ascendants et dont il hérite.**: Points to the "All Implemented Interfaces" section, which lists `Serializable`, `Cloneable`, `Iterable<E>`, `Collection<E>`, `List<E>`, and `RandomAccess`.
- Les descendants de ArrayList**: Points to the "Direct Known Subclasses" section, which lists `AttributeList`, `RoleList`, and `RoleUnresolvedList`.
- Il en implémente 3 et en hérite donc de 3**: Points to the `implements List<E>, RandomAccess, Cloneable, Serializable` line in the class declaration.

The class declaration is shown as follows:

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

The description below the declaration states: "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 equivalent 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=1710429889>

Last update: **2024/03/14 16:24**

