GÉNÉRICITÉ

FMS ACADEMY BY FMS
FACYLITIES MULTI SERVICES

PLAN

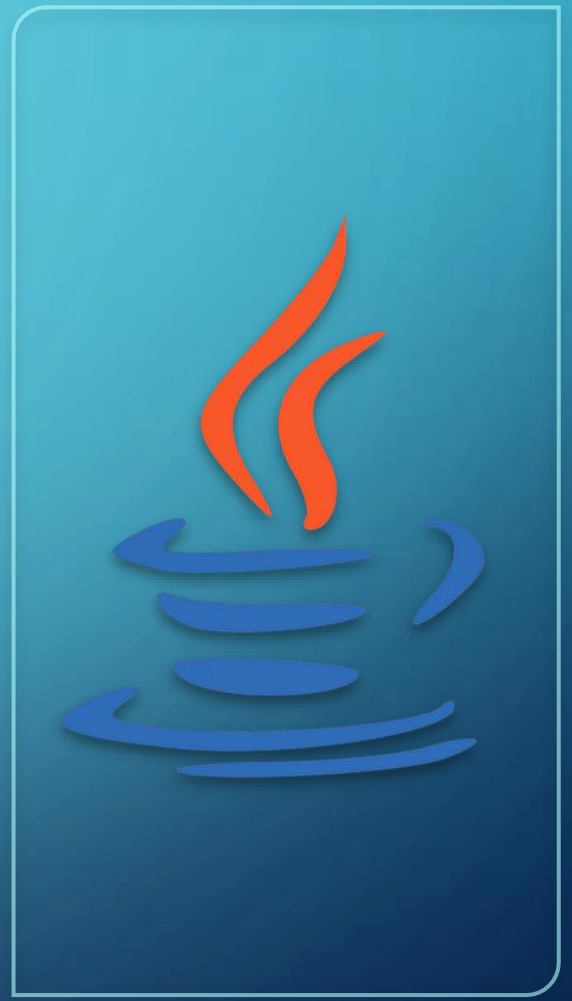
C’est quoi ?

Utiliser les types générique Java SE

La syntaxe

Opérateur diamant

Quelques types génériques



En *POO*, la généricité est un *concept* qui permet de *définir* des algorithmes identiques qui peuvent être utilisés sur de *multiples types* de données.

```java
import java.util.ArrayList;
import java.util.Date;
public class Demo {
    public static void main( String [] args ) {
     // Cette ligne produit un warning, mais pour le moment on l'ignore
      ArrayList collection = new ArrayList();
       // On y stocke nos chaînes de caractères
      collection.add( "Essai" );
      collection.add( "Cours Java" );
      collection.add( "Tutoriel Java" );
     // Mais rien nous interdit d'y mettre autre chose
      collection.add( new Date() );
     // Maintenant on parcourt la collection
     for ( Object value : collection ) {
         System.out.println( value ); } } }
```

NB : la ligne permettant de définir la collection produit un warning.

```java
import java.util.ArrayList;
import java.util.Date;
public class Demo {
    public static void main( String [] args ) {
     // Cette ligne produit un warning, mais pour le moment on l'ignore
      ArrayList<String> collection = new ArrayList<String>();
       // On y stocke nos chaînes de caractères
      collection.add( "Essai" );
      collection.add( "Cours Java" );
      collection.add( "Tutoriel Java" );
     // Mais rien nous interdit d'y mettre autre chose
     // collection.add( new Date() );
     // Maintenant on parcourt la collection
     // elle contient des instances de type string
     for ( Object value : collection ) {
         System.out.println( value ); } } }
```

Si nous utilisons la ***généricité***, nous pouvons indiquer que la collection ne peut contenir que des chaînes de caractères.

Et le programme sera donc plus fortement *typé* et plus *sûr*.

Utiliser les types générique Java SE

Le java SE propose un très grand nombre de classes génériques dans sa librairie, notamment les classes de *collections*.

Les différentes classes de collections java partagent certaines caractéristiques communes : elles implémentent donc des interfaces communes. Ces interfaces sont elles-mêmes génériques.

La syntaxe

L'exemple proposé précédent utilise la syntaxe traditionnelle. L'information de généricité doit être mentionnée sur la déclaration de la variable, mais aussi lors de son instanciation.

```
1 // On crée une collection de chaînes de caractères
2 ArrayList<String> collection = new ArrayList<String>();
```

Opérateur diamant

Depuis le java SE *7.0*, la syntaxe utilisée pour produire une instance basée sur un type générique a été simplifiée grâce à l'*opérateur diamant* <>.

```
1 // On crée une collection de chaînes de caractères
2 ArrayList<String> collection = new ArrayList<>();
```

QUELQUES TYPES GÉNÉRIQUES

On retrouve la *généricité* à différents endroits dans la librairie java SE.

Beaucoup de *collections* Java implémentent l'interface générique `java.util.Collection`.

```
Collection<E> List<E> Map<K,V> ….
```

Les conventions de nommage sur les noms des types

T : un type comme premier paramètre
E : un élément notamment dans les types de l'api collection
N : un nombre
S, U V : un type comme paramètre supplémentaire
K : la clé d'une map
V : la valeur d'une map

Ressources :

HTTPS://KOOR.FR/JAVA/TUTORIAL/JAVA_GENERICS_INTRODUCTION.WP

HTTPS://WWW.JMDOUDOUX.FR/JAVA/DEJ/CHAP-GENERIQUE.HTM