Formation Java 17

Fonctions du JDK

Sommaire

Interface Fonctionnelle

Les expressions lambda ne sont applicables que sur des interfaces dites “fonctionnelles”.

Il s’agit d’interface avec **une seule méthode abstraite**.

Les interfaces existantes du JDK qui n’ont qu’une seule méthode abstraite peut-être vue comme des interfaces fonctionnelles

```
public interface Runnable {

    public abstract void run();

}
```

Interface Fonctionnelle

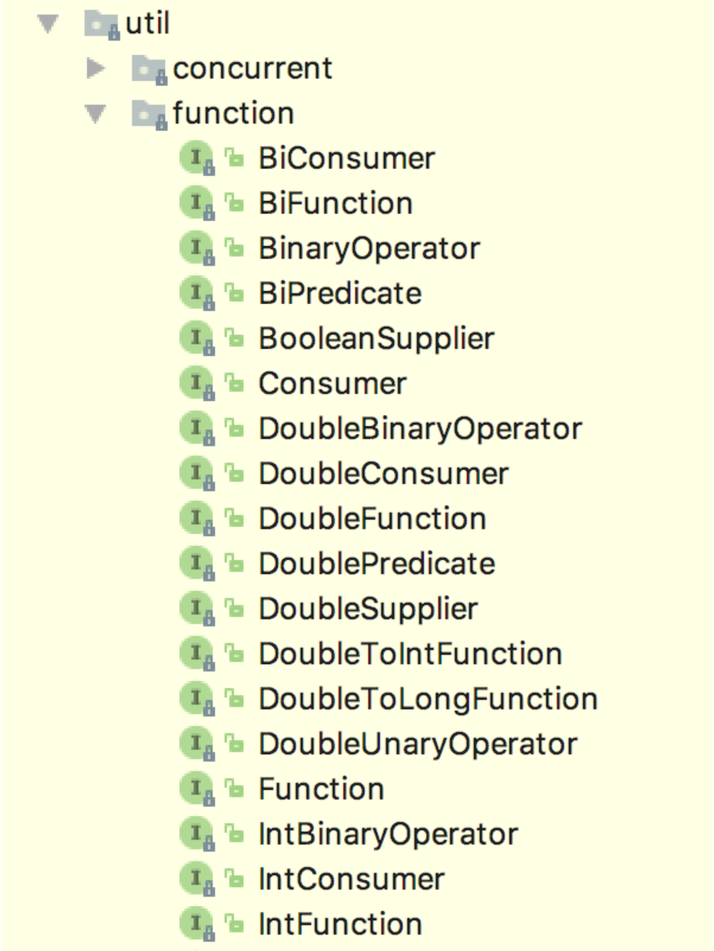
L'annotation **@FunctionalInterface** permet de vérifier à la compilation qu'une interface est bien fonctionnelle au sens Java 8 (elle ne contient qu'une seule méthode abstraite).

```
@FunctionalInterface
public interface Runnable {

   public abstract void run();

}
```

package java.util.function

Java 8 fournit des interfaces fonctionnelles usuelles dans le package **java.util.function**.

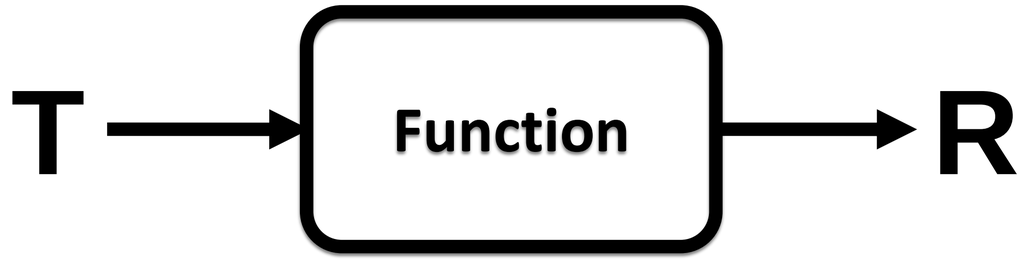
Function<T,R>
BiFunction<T,U,R>
Consumer<T>
Supplier<T>
...



Function<T,R>

Une fonction prend en paramètre un objet et retourne un autre type d'objet.

```
@FunctionalInterface
public interface Function<T, R> {

   R apply(T t);

}
```

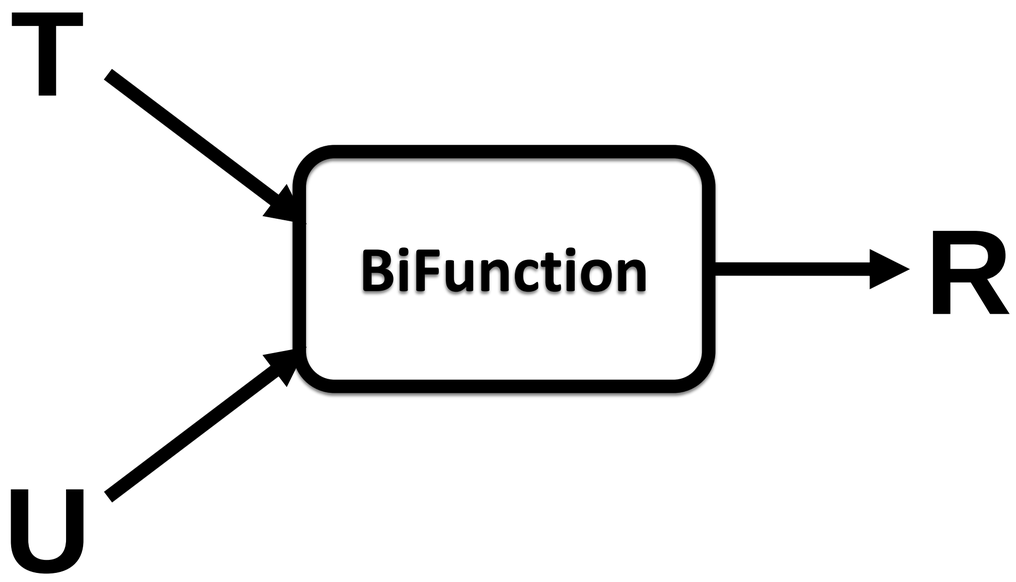


Exemple: le mapper qui transforme un objet en un autre objet
Si T=Compte et R=Double :
Function<Compte, Double> mapper = t -> t.getSolde();

BiFunction<T,U,R>

Une bi-fonction prend en paramètre 2 objets de types quelconque et retourne un objet d'un autre type quelconque.

```
@FunctionalInterface
public interface BiFunction<T, U, R> {

   R apply(T t, U u);

}
```

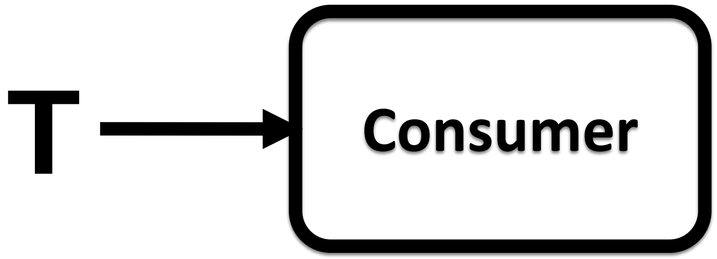


Exemple: une addition
T, U, R= Double
BiFunction<Double, Double, Double> addition = (t, u) -> t+u;

Consumer<T>

Un consumer représente une fonction qui consomme un objet.
Elle prend en paramètre 1 objet et ne retourne rien.

```
@FunctionalInterface
public interface Consumer<T> {

    void accept(T t);
}
```

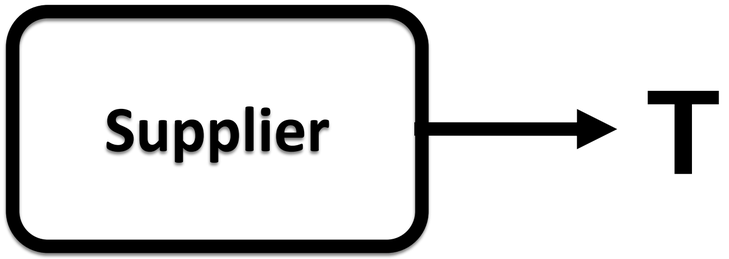


Exemple: une méthode qui écrit une donnée dans un fichier ou en base de données.
T = Compte
Consumer<Compte> consumer = t -> entityManager.persist(t);

Supplier<T>

Un consumer représente une fonction qui fournit un objet mais ne prend pas de paramètre.

```
@FunctionalInterface
public interface Supplier<T> {

    T get();
}
```



Exemple: une méthode qui fournit une instance d'objet, un singleton, etc.

T=Compte

Supplier<Compte> supplier = () -> new Compte();

Predicate<T>

Un prédicat représente une fonction qui teste un objet et retourne un booléen.

```
@FunctionalInterface
public interface Predicate<T> {

    boolean test(T t);

}
```

Exemple: une méthode qui fait du filtrage.
T=Compte
Predicate<Compte> predicat = t -> t.getSolde()>=0;

Travaux Pratiques