Функциональный интерфейс (functional interface) – это интерфейс у которого только один абстрактный метод. Функциональный интерфейс может содержать любое количество методов по умолчанию (*default*) или статических методов.

Функциональный интерфейс может, но не обязательно, быть помечен аннотацией `@FunctionalInterface` Наличие этой аннотации дает возможность компилятору контролировать “функциональность” интерфейса, т.е. наличие одного и только одного абстрактного метода. Наличие или отсутствие данной аннотации не делает интерфейс функциональным.

Функциональный интерфейс может быть реализован с помощью лямбда-выражения.

Внимание! Формально, любой интерфейс декларирующий единственный абстрактный метод - функциональный, но с точки зрения использования лямбда-выражении для реализации важно, чтобы интерфейс был функциональным и по смыслу. Это означает, что результат метода, объявленного в интерфейсе, должен зависеть только от исходных параметров и не зависеть от состояния класса имплементирующего интерфейс. Лямбда не имеет состояний! Например, Comparable мог бы считаться функциональным интерфейсом (единственный метод compareTo()), но он не является функциональным по смыслу, т.к. его результат зависит от переданного параметра и от класса в котором он реализован.

В JDK в пакете java.util.function собраны предопределенные функциональные интерфейсы. Давайте рассмотрим некоторые из них

java.util.function

Predicate<T>

Функциональный интерфейс Predicate<T> проверяет соблюдение некоторого условия. Если оно соблюдается, то возвращается значение true. В качестве параметра принимает объект типа T

```
boolean test(T t);
```

BiPredicate<T,U> тоже, что и Predicate<T> но, принимает два параметра

```
boolean test(T t U u);
```

Consumer<T>

Consumer<T> (потребитель) выполняет некоторое действие над объектом типа T, при этом ничего не возвращая

```
void accept(T t);
```

BiConsumer<T, U> выполняет некоторое действие над двумя объектами, при этом ничего не возвращая

```
void accept(T t, U u)
```

Supplier<T>

Supplier<T> (поставщик) не принимает никаких аргументов, но должен возвращать объект типа T

```
T get();
```

Function<T,R>

Функциональный интерфейс Function<T, R> представляет функцию перехода от объекта типа T к объекту типа R

```
R apply(T t);
```

BiFunction<T, U, R> представляет функцию которая принимает объект типа T, объект типа U и возвращает объект типа R

```
R apply(T t, U u);
```

UnaryOperator<T>

UnaryOperator<T> принимает в качестве параметра объект типа T, выполняет над ними операции и возвращает результат операций в виде объекта того же типа T

```
T apply(T t);
```

***Примечание**: фактически, UnaryOperator<T> это частный случай Function<T, R>, где тип результата <R> равен типу <T>*

BinaryOperator<T>

BinaryOperator<T> принимает в качестве параметра два объекта типа T, выполняет над ними бинарную операцию и возвращает ее результат также в виде объекта типа T

```
T apply(T t1, T t2);
```

***Примечание**: фактически, BinaryOperator<T> это частный случай Function<T, U, R>, где все три типа равны*

Остальные функциональные интерфейсы пакета java.util.function представляют вариации основных функциональных интерфейсов адаптированных для использования с примитивными значениями.