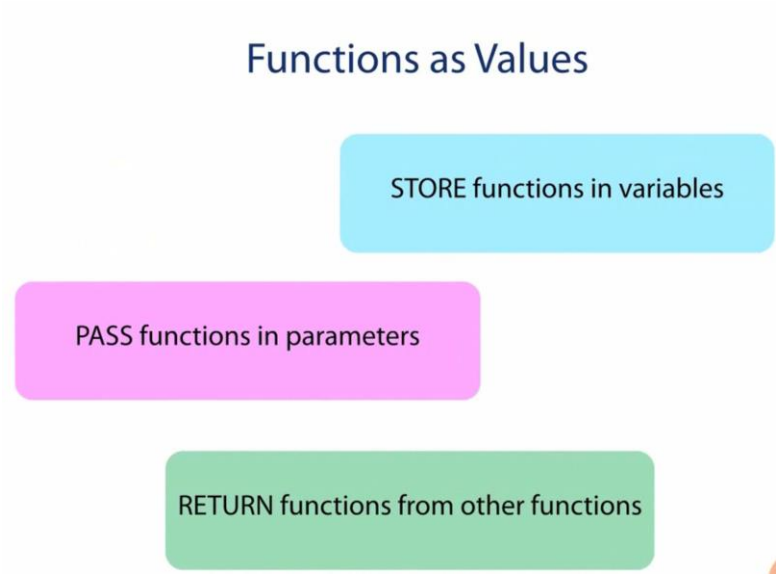
# Java Functional Programming




## Imperative Programming

Programs are written in a series of instructions that tell the computer what to do (and how).

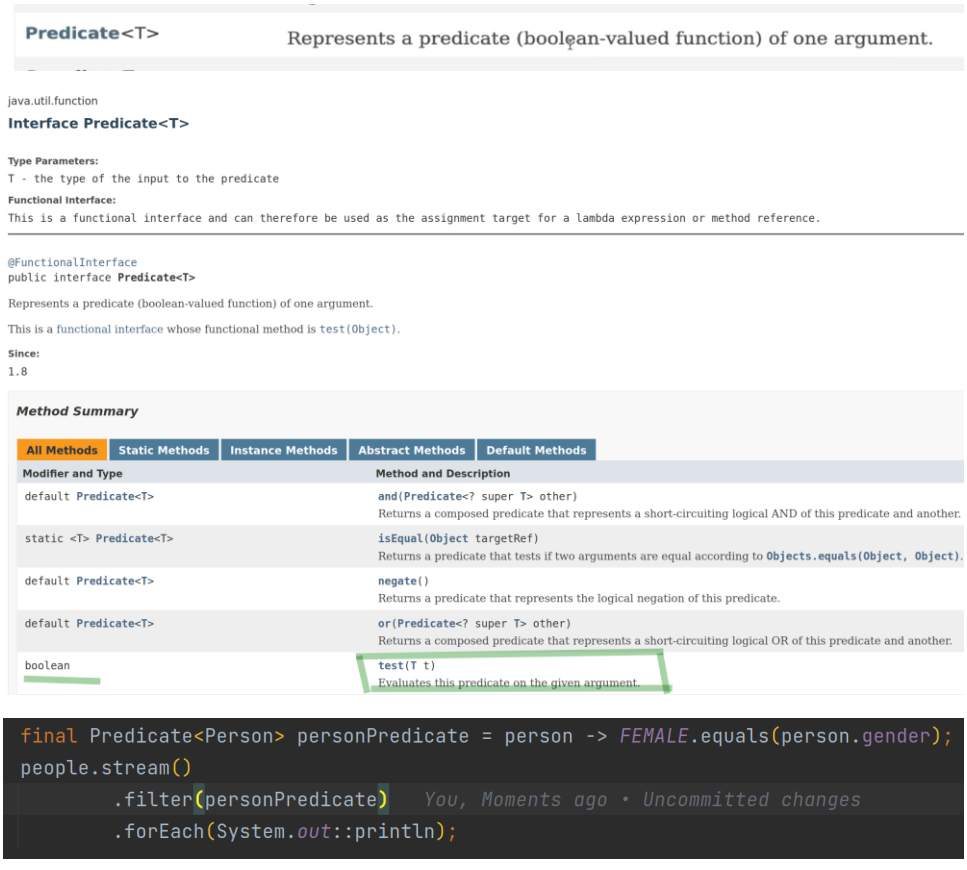
## Declarative Programming

A programming paradigm that describes what the program should do, without specifying how it should be done.

## Functional interfaces

## Predicate Interface

In Java, a predicate is a function that takes one or more arguments and returns a boolean value.



**Predicate<T>** — Represents a predicate (boolean-valued function) of one argument.

java.util.function

**Interface Predicate<T>**

**Type Parameters:**
`T` - the type of the input to the predicate

**Functional Interface:**
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

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

Represents a predicate (boolean-valued function) of one argument.

This is a functional interface whose functional method is `test(Object)`.

**Since:**
1.8

***Method Summary***

All Methods | Static Methods | Instance Methods | Abstract Methods | Default Methods

| Modifier and Type | Method and Description |
|---|---|
| default `Predicate<T>` | **`and(Predicate<? super T> other)`**<br>Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another. |
| static `<T> Predicate<T>` | **`isEqual(Object targetRef)`**<br>Returns a predicate that tests if two arguments are equal according to **`Objects.equals(Object, Object)`**. |
| default `Predicate<T>` | **`negate()`**<br>Returns a predicate that represents the logical negation of this predicate. |
| default `Predicate<T>` | **`or(Predicate<? super T> other)`**<br>Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another. |
| boolean | **`test(T t)`**<br>Evaluates this predicate on the given argument. |

```java
final Predicate<Person> personPredicate = person -> FEMALE.equals(person.gender);
people.stream()
        .filter(personPredicate)    You, Moments ago • Uncommitted changes
        .forEach(System.out::println);
```

## Fuction Interface

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

Represents a function that accepts one argument and produces a result.

This is a functional interface whose functional method is `apply(Object)`.

**Since:**
1.8

***Method Summary***

All Methods | Static Methods | Instance Methods | Abstract Methods | Default Methods

| Modifier and Type | Method and Description |
|---|---|
| default `<V> Function<T,V>` | **`andThen(Function<? super R,? extends V> after)`**<br>Returns a composed function that first applies this function to its input, and then applies the `after` function to the result. |
| `R` | **`apply(T t)`**<br>Applies this function to the given argument. |
| default `<V> Function<V,R>` | **`compose(Function<? super V,? extends T> before)`**<br>Returns a composed function that first applies the `before` function to its input, and then applies this function to the result. |
| static `<T> Function<T,T>` | **`identity()`**<br>Returns a function that always returns its input argument. |

***Method Detail***

**apply**

`R apply(T t)`

Applies this function to the given argument.

**Parameters:**
`t` - the function argument

**Returns:**
the function result

```java
public static void main(String[] args) {
    int increment = incrementByOne( number: 0);
    System.out.println(increment);    You, Moments ago • Uncommitted changes

    final Integer incrementByOneLambda = incrementByOneFunction.apply( t: 1);
    System.out.println(incrementByOneLambda);
}

1 usage
static Function<Integer, Integer> incrementByOneFunction = number -> number + 1;
```

## Chaining Lambdas (andThen())

The output of a lambda can be used as the input of another lambda.

```java
2 usages
static Function<Integer, Integer> incrementByOneFunction = number -> number + 1;

2 usages
static Function<Integer, Integer> multiplyBy10Function = number -> number * 10;
```

```java
Function<Integer, Integer> addByOneAndThenMultiplyBy10 = incrementByOneFunction.andThen(multiplyBy10Function);
int incrementAndMultiply = addByOneAndThenMultiplyBy10.apply( t: 4);
System.out.println(incrementAndMultiply);
```

## Bifunction

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

Represents a function that accepts two arguments and produces a result. This is the two-arity specialization of `Function`.

This is a functional interface whose functional method is `apply(Object, Object)`.

**Since:**
1.8

**See Also:**
`Function`

***Method Summary***

All Methods | Instance Methods | Abstract Methods | Default Methods

| Modifier and Type | Method and Description |
|---|---|
| `default <V> BiFunction<T,U,V>` | `andThen(Function<? super R,? extends V> after)`<br>Returns a composed function that first applies this function to its input, and then applies the after function to the result. |
| `R` | `apply(T t, U u)`<br>Applies this function to the given arguments. |

```java
static BiFunction<Integer, Integer, Integer> incrementByOneAndMultiplyBiFunction =
        (numberToIncrementByOne, numberToMultiplyBy)
                -> (numberToIncrementByOne + 1) * numberToMultiplyBy;
```

```java
// BiFunction
int incrementAndMultiplyBiFunction = incrementByOneAndMultiplyBiFunction.apply( t: 1,  u: 2);
System.out.println(incrementAndMultiplyBiFunction);
```

## Consumer

java.util.function

**Interface Consumer<T>**

**Type Parameters:**
`T` - the type of the input to the operation

**All Known Subinterfaces:**
`Stream.Builder<T>`

**Functional Interface:**
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

---

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

Represents an operation that accepts a single input argument and returns no result. Unlike most other functional interfaces, `Consumer` is expected to operate via side-effects.

This is a functional interface whose functional method is `accept(Object)`.

**Since:**
1.8

***Method Summary***

All Methods | Instance Methods | Abstract Methods | Default Methods

| Modifier and Type | Method and Description |
|---|---|
| void | `accept(T t)`<br>Performs this operation on the given argument. |
| default `Consumer<T>` | `andThen(Consumer<? super T> after)`<br>Returns a composed `Consumer` that performs, in sequence, this operation followed by the `after` operation. |

```java
static Consumer<Customer> greetCustomerConsumer = customer ->
        System.out.println("Hello " + customer.customerName +
        ", thanks for registering phone number: " +
        customer.customerPhoneNumber);
```

```java
// Consumer   You, Moments ago • Uncommitted changes
final Customer jack = new Customer( customerName: "Jack",  customerPhoneNumber: "555869588");
greetCustomerConsumer.accept(jack);
```

## BiConsumer