

JAVA 8 - a revolutionary release of the development platform which brings some major tweaks and upgrades to the Java programming language -including enhanced JavaScript engine, new APIs for date and time manipulation, improved and faster JVM, and the lambda expressions, stream API, etc.



Many of you may be aware of these upgrades, but few who are wondering, what is it and how it works, we have a brief explanation with a simple example 😊 follows.

1. Lambda expressions – The tech giant’s most anticipated upgrade for programming language.

What is lambda expression?

- To put in simple words, *expresses instances of functional interfaces, lambda expressions implement the only abstract function and therefore implement functional interfaces*

Syntax:

```
Lambda operator -> body
```

Now, we have another question, what is functional interface?

- *an interface that contains only one abstract method, they can have only one functionality to exhibit and it can have any number of default methods*

First we will see how to create functional interface.

Ex: *creating function interface*

```
@FunctionalInterface
public interface MyFuntionalInterface {

    void arithmeticOp(int operand);

    /**
     * Some default methods
     */
}
```

@FunctionalInterface annotation is used to ensure that the functional interface can't have more than one abstract method. In case more than one abstract methods are present, the compiler flags an 'Unexpected @FunctionalInterface annotation' message. However, it is not mandatory to use this annotation.

Now, will see how we can use this functional interface in our program.

```
.....
FunctionalInterface mFI_ADD_Func = (a) -> a + a;

System.out.println("Add the value : " + mFI_ADD_Func.arithmeticOp(4));

FunctionalInterface mFI_MUL_Func = (a) -> a * a;

System.out.println("Multiply the value : " + mFI_MUL_Func.arithmeticOp(4));

FunctionalInterface divideWithPrint_Func = (a) -> {
    System.out.println("Divide the value From lambda :");
    return a / a;
};
System.out.println(" --> " + divideWithPrint_Func.arithmeticOp(4));
.....
```

Ref : [GIT - lambda with on demand functions in java 8](#)

So, Lambda is nothing but function on demand which is not belongs to any class and can access variables declared outside its scope (not advisable).

Functionalities of Lambda

Exs : [GIT - lambda examples](#)

- ✚ Enable to treat functionality as a method argument, or code as data.
- ✚ A function that can be created without belonging to any class.
- ✚ A lambda expression can be passed around as if it was an object and executed on demand
- ✚ A lambda operator can have “**O**” - zero parameters, “**(a)**” - one parameter, “**(a, b, c)**” multiple parameters, Lambda expressions are just like functions and they accept parameters just like functions (parameters depends on abstract method in the functional interface designed)

Now, will see how the same interface in Java 7 works

```
.....
MyFunctionalInterface mFI_ADD_Func = new MyFunctionalInterface() {
    @Override
    public int arithmeticOp(int a) {
        return a + a;
    }
};

System.out.println("Java7 impl - Add the value : " + mFI_ADD_Func.arithmeticOp(4));

MyFunctionalInterface mFI_MUL_Func = new MyFunctionalInterface() {
    @Override
    public int arithmeticOp(int a) {
        return a * a;
    }
};

System.out.println("Java7 impl - Mul the value : " + mFI_MUL_Func.arithmeticOp(4));
.....
```

Ref : [GIT - On demand functions in java 7](#)

Yes, in Java7 too, we can have an on-demand functionality, but it cost us to create an Anonymous **InnerClass/ or new classes to implement**

sinterface on function,  Java7, but stills does the works but on cost of bit ugly code design and it will compile with three more class files.

So, the Magic, **lambda expression**



Java8's cool Dude...! Allow us to write on demand functions with, beautiful code design.

Some of functional interfaces in Java - *Runnable, ActionListener, Comparable and some java8 util.function package has Predicate, BinaryOperator, and Function, etc.*

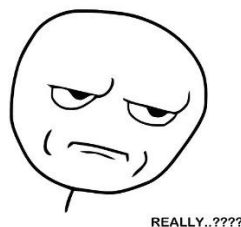
We can make functional interface as a generic type (above all the java's functional interface are generic type) and write our code with desired data types and various functions. (Ex: Function as name tells we can apply any functionality to it.)

Cool right?

Here it is, can we have hands on to implement lambda, which will be the best practice and will also have experience on functionality?



We can skip if you guys want to move on to next topic and can do the hands on later. But



Skip, skip, skip

Use case – have an Employee class with firstName, lastName, age and salary.

Conditions:

- ✚ Sort by first name
- ✚ Sort by last name first letter
- ✚ Sort by last name last letter
- ✚ Sort by age and,
- ✚ Sort by salary

It is easy with lambda,

Hints - use `Collections.sort()` for sorting with `Comparable/ comparator` user interface.

Ref : [Git - exercise](#)

Java7

vs

Java8 (With lambda & other upgrades)



That's how the transformation design happened from java7 -> java8, still we have more API's & upgrade to know about.