Functional Interface:

* If an interface has single abstract method that interface, we can call it as functional interface.
* We can write lambda expression only if there is a functional interface.
* Lambda expression and functional expression they work together

To indicate that it is functional interface to compiler and developer it is recommended to use an annotation ( @Functional Interface)

Syntax :

@FunctionalInterface

Interface Demo{

void display();

}

public class lauch {

public static void main( String[] args) {

}

}

From java 8 “-> “operator is introduced knows as arrow operator / lambda operator, used to write lambda expressions.

Lambda expression is an anonymous method, means it is a method with no name.

1. To write lambda expression we use “->“
2. It consists of 2 parts
3. To the left of lambda operator we write parameters
4. To the right of lambda operator we write body
5. In left side for parameters data type is optional
6. Right side if implementation (or) body has only one statement then { } can be omitted
7. Left side if parameter is single then () and parameter data type declaration both are optional.
8. Data type declaration is optional for parameters on the left side.
9. Right side i.e body if it is single line implementation and we have omitted {} braces then return statement should not be used.
10. You can write return statement only if you used {} braces

Syntax:

(parameters) -> {

// body of the method

}

We can omit the braces if the method body has only single statement.

Eg: () -> System.out.println(“hello”);

As it has no name lambda expression cannot be called, they are dependent on functional interface. lambda expression and functional interface both are dependent on each other.

Eg: Functional\_Interface

// go through the code

Eg: Fuctional\_Interface\_With\_Optional\_Paramaters\_DataTypes

// go through the code

Eg: Functional\_Interface\_With\_Single\_Parameter\_SingleLine\_Body

There is no need to use implements in functional interface

How a lambda expression is executed

1. Create the reference of the interface and write the lambda expression there
2. And with that reference call the interface method, then a call be will be made to the lambda expression during execution.

Similar to this we have anonymous inner classes

* Java anonymous inner class is an inner class without the name and for which only single object is created
* It is useful when we want to implement the class (or) interface without actually having the subclass.
* It should be used If you have to override the method of class or interface

Anonymous inner class can be created in two ways

1. Class (may be abstract or concrete)
2. Interface

Eg: Anonymous\_Class\_Using\_Interface

// go through the code

In anonymous inner class there is non-restriction that we need to implement only single abstract method like functional interface. We can implement any no of abstract methods.

There is no need of implements keyword in the class, the compiler automatically understands that it is Anonymous inner class.

Eg: Length\_Of\_String\_InnerClass

// go through the code

Eg: Length\_Of\_String\_Using\_Lambda\_Expression

// go through the code

Note: In Java lambda expression, if there is only one statement, you may or may not use return keyword. You must use return keyword when lambda expression contains multiple statements.

Eg: Lambda\_Expression

Eg: Lambda\_Expression\_Eg2

// go through the code