Before learning Lambda expression, we should know functional interface.

**What is this functional interface?**

An Interface that contains exactly one abstract method is known as functional interface. It can have any number of default and static methods but can contain only one abstract method. It can also declare methods of object class (like equals, hashCode, clone..etc.).

Any interface with a **SAM** (Single Abstract Method) is a functional interface, and its implementation may be treated as lambda expressions.

A functional interface can extend another interface only when it does not have any abstract method. It throws compile time error.

It's recommended that all functional interfaces have an informative @FunctionalInterface annotation. This clearly communicates the purpose of the interface, and also allows a compiler to generate an error if the annotated interface does not satisfy the conditions.

Example of Java Functional interfaces

@FunctionalInterface  
public interface Runnable {

public abstract void run();  
}

@FunctionalInterface

public interface Comparable<T> {

public int compareTo(T o);

}

@FunctionalInterface  
public interface Comparator<T> {

int compare(T o1, T o2);

}

@FunctionalInterface  
public interface Callable<V> {

call() throws Exception;

….  
}

Our own functional interface

@FunctionalInterface

public interface MyFunctionalInterface{

    public void m1();

    public static void m2(){

        System.out.println("Example0001FunctionalInterface.m2()");

    }

    public static void m3(){

        System.out.println("Example0001FunctionalInterface.m3()");

    }

    public default void m4(){

        System.out.println("Example0001FunctionalInterface.m4()");

    }

    public default void m5(){

        System.out.println("Example0001FunctionalInterface.m5()");

    }

}