Lemda Expression

Characteristics of lambda expression

1. Lambda Expression can take any number of arguments

Ex: () -> System.out.println(“Hello”);

(s) -> s.length();

(a,b) -> system.out.println(a+b);

1. If multiple parameter presents then these parameter should be separated with comma(,)

Ex: (a,b) -> sop(a,b);

1. If only parameter available then parenthesis are optional

Ex: (s) -> s.length(); == s -> s.length();

1. Usally we can specify the type of parameter of compiler expect the type based on context then we can remove type[ type infreence]

Ex: (int a, int b) -> sop(a+b); == (a,b) -> sop(a+b);

1. Similar to the method body lambda expression can contains any number of statements. If multiple statements are then it should be under the curly braces.

Ex: () -> {

----

--

}

If body contains only one statements than curly braces are optional

1. If lambda expression return something then we can remove return keywords

Ex: s -> s.length();

**Functional Interface (**Single abstract method**)**

A interface which contains single abstract method is called functional interface.

Runnalble 🡪 contains only run() method

Callable 🡪 contains only call() method

ActionListener 🡪 contains only actionPerformed() method

Comparable 🡪 contains only compareTo() method

Ex: interface interf{

public void m1();

default void m2(){

}

Public static void m3(){  
 --

}

}

It is a functional interfaces

: interface interf{

public void m1();

public void m2();

}

It is not a functional interface because it contains two abstract method

**@FunctionalInterface**: will check that the interface should contains only one abstract method.

@FuncitonlInterface

Public interface interf1{

public void m1(); //single abstract method

default void m2(); Functional interface



static void m3();

}

@FuncitonlInterface

Public interface interf1{

Unexception functional interface

Multiple non overriding method present in functional interface

public void m1(); //single abstract method

public void m2();//one more abstract method Functional interface



}

@FuncitonlInterface

Unexception @FunctionalInterface

No abstract method found in functional interface

Public interface interf1{

//No method



}

**@FunctionalInterface with respect to interface**

**Case 1**. If an interface extends functional interface and child interface doesnot contains any SAM then chile interface is always functional interface.

Ex:

@FunctionInterface

Public interface P{

It is correct

Public void m1();

}



@FunctionalInterface

Public interface Q extends P{

}

Case 2:

If the child interface we can define excetly same parent interface abstract method

Ex:

@FunctionInterface

Public interface P{

It is correct

Public void m1();

}



@FunctionalInterface

Public interface Q extends P{

Public void m1();

}

Case 3:

In the child interface we can not define any new abstract method otherwise we will get compile time error.

Ex:

@FunctionInterface

Public interface P{

We will get compile time error

Public void m1();

}



@FunctionalInterface

Public interface Q extends P{

Public void m2();

}

Example 1

Public interface interf{

Public void add(int a, int b);

}

Class Test{

Public static void main(String[] args){

Interf I = (a, b) -> System.out.println(“The sum is ”+(a+b));

i.add(10,2);

i.add(20,40);

}

}

Example 2;

public interface interf{

public int getLength(String s);

}

public class Test(){

public static void main(String[] args){

Interf i = s -> s.length();

i.getLength("Hello");

}

}

Example with Runnable

--------------------------

class ThreadDemo{

public static void main(String argh[]){

Runnable r = ()->{

for(int i=0; i<5;i++){

Sop("Chile Thread");

}

}

Thread t = new Thread(r);

t.start();

for(int i=0; i<5;i++){

Sop("Main Thread");

}

}

}

Functional Interface

* it should contains exactly only one abstract method(Single abstract method)
* it can contain any number of default and static method
* it is act as type of Lambda expression
* it can be used to invoke Lambda expression.