JA

1. Lambda Expressions
2. Functional Interface
3. Default Method and Static Method
4. Predefined Functional Interface

Predicate

Function

Consumer

Supplier

Etc.

1. Double Colon Operator (::)

Method reference

Constructor Reference

1. Streams
2. Date and Time API
3. Optional Class

etc.

1. Lambda Expressions :

What is Lambda Expression :

It is an anonymous function

* Name less
* Without return type
* Without modifiers

Functional Interface : SAM Single Abstract Method (The Job of functional interface is to invoke lambda Expression) and any number of Default and Static Method

Runnable 🡺 run

Comparable 🡺compareTo()

Comparator 🡺compare()

ActionListener🡺actionPerfomed()

Callable 🡺call()

Predicates : In over program wherever some conditional checks are there haply use predicates.

Predicate<Integer> p = i -> i % 2 == 0;

System.***out***.println(p.test(5));

System.***out***.println(p.test(6));

Predicate<String> p1 = (k)->k.length()>5;

String s1 ="hellosdlfj";

System.***out***.println(p1.test(s1));

**Predicate,Function ,Consumer,Supplier :**

**Predicate** : Take some input perform some conditional check and return s boolean value.

test()

**Function** : Take some input perform some operation return the result which is not be a Boolean type.

apply()

**Consumer** : Accept some input and perform required operation and not required to return anything.

accept()

**Supplier** : Just supply my required objects and it won’t take any input .

get();

**package com.test.java8;**

**import java.util.Date;**

**import java.util.function.Function;**

**import java.util.function.Predicate;**

**import java.util.function.Supplier;**

**public class Test {**

**public static void main(String[] args) {**

**// Function<Integer, Integer> f = i -> i \* i;**

**// System.out.println("The square of 5 is :" + f.apply(5));**

**// System.out.println("The square of 5 is :" + f.apply(6));**

**//**

**// Predicate<Integer> p = i -> i % 2 == 0;**

**// System.out.println(p.test(5));**

**// System.out.println(p.test(6));**

**//**

**// Predicate<String> p1 = (k)->k.length()>5;**

**// String s1 ="hellosdlfj";**

**// System.out.println(p1.test(s1));**

**int x[] = {0,5,10,15,20,25,30};**

**Predicate<Integer> p2 = l -> l%2 ==0;**

**Predicate<Integer> p3 = l -> l > 10;**

**//and(),or(),negate()**

**// System.out.println("The number which are even and greater than 10 are");**

**// for(int x1:x) {**

**// if(p2.and(p3).test(x1)) {**

**// System.out.println(x1);**

**// }**

**// }**

**// System.out.println("The number which are not even are");**

**// for(int x1:x) {**

**// if(p2.negate().test(x1)) {**

**// System.out.println(x1);**

**// }**

**// }**

**//When this result need not we boolean type according to my business operations like Integer or Boolean or any type**

**//then go with Function.**

**// Function<Integer, Integer> f=i->i\*i;**

**// System.out.println(f.apply(10));**

**// Supplier<Date> s = ()->new Date();**

**// System.out.println(s.get());**

**// System.out.println(s.get());**

**//i want to supplier to generate otp**

**Supplier<String> s =()->{**

**String otp = "";**

**for (int i = 0; i < 6; i++) {**

**// 0<= x <1**

**//0 is min**

**//0.99999**

**//Requirement is to generate number 0 to nine**

**//9.9999**

**otp = otp + (int)(Math.random()\*10);**

**}**

**return otp;**

**};**

**System.out.println(s.get());**

**}**

**}**

**Streams :**

**Collection and Stream :**

If you want to represent a group of object as single entity then we should go for collection.

If we want to process objects from the collection then we should go for Streams.

Stream is an interface which is present java.util.Stream package

**filter :**

filter(Predicate)

input 🡪 10 elelments

output 🡪 o to 10 or <=10

**map:**

map(Function)

input : 10 elements

output : 10

**count :**

**stream()**

**filter()**

**map()**

**collect()**

**count()**

**sorted() 🡪**Default natural sorting order

**sorted(Comparator)🡪**customize sorting order

**min():**

**max():**

**toArray(): 🡪**To convert stream of object into array