Java 8

1.Lambda Expression

2.Fuctional Interface

3.Default and static method

4.Predefined functional Interface

- Predicate

- Function

- Consumer

- Supplier

5.Streams

6.Date and Time api

7. Optional Class

**1. Lambda Expression:-**

-> It is Anonymous function which is having no name, no returnType,no modifiers.

It Provide The implementation of Functional interface.

It is applicable for only functional Interface.

Lambda Expression doesnot generate separate .class file

Eg 1.

() -> System.out.println(“Hello” );

Eg 2.

(**int** a,int b)->System.out.println(a+” ”+b);

(a, b)->System.out.println(a+” ”+b);

Eg 3.

**public** **int** m1(**int** a){

**return** a\*a;

}

* (**int** a) -> { **return** a\*a; } -- If return statement then use curlibrace

(int a) -> a\*a; -- remove return curlibrase not mandetory

(a) -> a\*a; -- datatype guess by compiler

a -> a\*a; -- if one parameter then open close bracket not need.

**2. Functional Interface:-**

-> Inteface contain only one abstract method .

It can contain static ,default method.

Runnable == > run

Comparable == > compareTo

Comparetor ==> compare

* Lambda Expression with fuctional interafe.

Eg. 1 –

**interface** Test2{

**public** **void** m1();

}

**class** Test3 **implements** Test2{

@Override

**public** **void** m1() {

System.***out***.println("Hii");

}

}

**public** **class** Test{

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

Test3 t = **new** Test3();

t.m1();

}

}

**Convert into lambda**:-

**interface** Test2{

**public** **void** m1();

}

**public** **class** Test{

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

Test2 t = ()-> System.***out***.println("Hii");

t.m1();

}

}

**Eg.2**

**interface** Test2 {

**public** **void** m1(**int** a,**int** b);

}

**public** **class** Test {

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

Test2 t = (a,b)->System.***out***.println(a+b);

t.m1(2, 6);

}

}

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

Eg.3

**interface** Test2 {

**public** **int** m1(**int** a);

}

**class** Test3 **implements** Test2{

@Override

**public** **int** m1(**int** a) {

**return** a\*a;

}

}

**public** **class** Test {

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

Test3 t= **new** Test3();

**int** m1 = t.m1(4);

System.***out***.println(m1);

}

}

Convert :-

**interface** Test2 {

**public** **int** m1(**int** a);

}

**public** **class** Test {

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

Test2 t= (a)-> a\*a;

**int** m1 = t.m1(4);

System.***out***.println(m1);

}

}

**Lambda Expression with Thread**

Eg.1

**class** Test1 **implements** Runnable{

**public** **void** run() {

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

System.***out***.println("child");

}

}

}

**public** **class** Test {

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

Test1 t= **new** Test1();

Thread th = **new** Thread(t);

th.start();

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

System.***out***.println("main");

}

} }

**Conver into lambad**:-

**public** **class** Test {

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

Runnable t=()->{

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

System.***out***.println("child");

}

};

Thread th = **new** Thread(t);

th.start();

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

System.***out***.println("main");

}

}

}

**Lambda Expression with Collection**

Eg.1

**class** MyComparator **implements** Comparator<Integer>{

@Override

**public** **int** compare(Integer o1, Integer o2) {

**if**(o1 > o2){

**return** 1;

}**else** **if**(o1 < o2){

**return** -1;

}

**return** 0;

}

}

**public** **class** Test {

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

ArrayList al = **new** ArrayList();

al.add(3);

al.add(5);

al.add(0);

al.add(2);

Collections.*sort*(al,**new** MyComparator());

System.***out***.println(al);

}

}

Convert:-

**public** **class** Test {

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

ArrayList al = **new** ArrayList();

al.add(3);

al.add(5);

al.add(0);

al.add(2);

Comparator<Integer> c = (a1,a2)->(a1 > a2)?1:(a1 < a2)?-1:0;

Collections.*sort*(al,c);

System.***out***.println(al);

}}

Eg 2 Sort data Using Lambda

**class** Student{

**int** rollNo;

String name;

**public** Student(**int** rollNo, String name) {

**super**();

**this**.rollNo = rollNo;

**this**.name = name;

}

@Override

**public** String toString() {

**return** "Student [rollNo=" + rollNo + ", name=" + name + "]";

}

}

**public** **class** Test {

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

ArrayList<Student> al = **new** ArrayList<Student>();

al.add(**new** Student(2,"Rahul"));

al.add(**new** Student(1,"Divya"));

al.add(**new** Student(4,"Gacfd"));

System.***out***.println(al);

//Collections.sort(al,(a,b)->(a.rollNo > b.rollNo)?1:(a.rollNo < b.rollNo)?-1:0);

Collections.*sort*(al,(a,b)->a.name.compareTo(b.name));

System.***out***.println(al);

}

}

**Remove Duplicate using key**

List<Student> collect = al.stream().collect(Collectors.*toMap*(Student::getRollNo, s->s,(existing,replacement)->existing)).values().stream().collect(Collectors.*toList*());

collect.forEach(i->{

System.***out***.println(i);

});

2] **Default Method and Abstarct Method:-**

**Default:-**

- Without effecting implementation classes if we want to add new method then go for default.

- It is use in Interface

**interface** Test1{

**public** **void** m1();

**public** **void** m2();

**default** **void** m3(){

System.***out***.println("df");

}

}**class** Test2 **implements** Test1{

**public** **void** m1() {

}

**public** **void** m2() {

}

}

**public** **class** Test {

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

}

}

**Static :-**

To define utility methods with static

Utitity :- not related to class, not related to object

Eg. **interface** Test1{

**public** **static** **void** m1() {

System.***out***.println("df");

}

}

**public** **class** Test **implements** Test1{

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

Test1.*m1*();

}

}

4] **Predefined finction :**

**1. Predicate** :-

Whenever checking condition is required then we use predicate.

It contain one abstract method:-

Public abstract Boolean test(T t);

It return true or false

Eg .

**public** **boolean** test(Integer i){

**if**(i%2==0){

**return** **true**;

}**else**{

**return** **false**;

}

}

Convert into lambda :-

(i)->i%2==0;

Eg.2

**import** java.util.function.\*;

**public** **class** Test {

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

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

System.***out***.println(p.test(10));

}

}

Eg.3

// length of string is greater than 5 oe not

**public** **class** Test {

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

String s="Rahul";

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

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

}

}

Eg.4

// length of string is greater than 5 oe not

**public** **class** Test {

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

String s[]={"rahul","divya","nikhil","ram"};

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

**for**(String s1:s){

**if**(p1.test(s1)){

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

}

}

}

}

Eg.5

// print employee salary greater than 3000

**class** Employee{

**int** salary;

String name;

**public** Employee(**int** salary, String name) {

**super**();

**this**.salary = salary;

**this**.name = name;

}

}

**public** **class** Test {

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

ArrayList<Employee> al = **new** ArrayList();

al.add(**new** Employee(3000, "Rahul"));

al.add(**new** Employee(2000, "shiva"));

al.add(**new** Employee(5000, "divya"));

Predicate<Employee> p1 = (i)->i.salary>=3000;

**for**(Employee e:al){

**if**(p1.test(e)){

System.***out***.println(e.name);

}

}

}

}

Ex. StartWith of int array

**int** a[]={2,24,3,6};

Predicate<Integer> p= i->i.toString().startsWith("2");

List<Integer> collect = Arrays.*stream*(a).boxed().filter(p).collect(Collectors.*toList*());

System.***out***.println(collect);

2] **function:-**

If operation we want to preform the go for function.

It contain apply(T,R) method. (T id input type & R is Return type)

Eg.1

**public** **class** Test {

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

**int** a=20;

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

System.***out***.println(f.apply(a));

}

}

Eg.2

//I went to return length of String

**public** **class** Test {

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

String s ="rahul";

Function<String,Integer> f= (i)->i.length();

// String is input and integer is return type

System.***out***.println(f.apply(s));

}

}

Eg.3

//Convert String into uppercase

**public** **class** Test {

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

String s ="rahul";

Function<String,String> f= (i)->i.toUpperCase();

// String is input and String is return type

System.***out***.println(f.apply(s));

}

}

3] **Consumer** :-

It is used for oply prent message

It contain accept(T t) method ..It return nothing

Eg.1

**public** **class** Test {

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

String s="Divya";

Consumer<String> c = (i)->System.***out***.println(i);

c.accept(s);

}

}

4] **Stream** :-

Collection:- If we want to represent group of object into single entity then we use

Collection.

Stream :- If we want to process of object from collection then use stream.

It contain Stream(); 🡪 java.util.stream pkg.

Method:-

Filter(Predicate)

Map(function)

Collect

Count

Sorted

Sorted(Comparator)

Min()

Max()

Foreach()

toArray()

Stream.of()

1] **Filter** :- To filter element based on some codition then go for filter

Eg.1 . I want Even no.

**public** **class** Test {

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

ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

List<Integer> collect = al.stream().filter(i->i%2==0).collect( Collectors.*toList*());

System.***out***.println(collect);

}

2] **Map** :- If we want to do some operation gthen gor for map();

Eg 1.

//Every Student can we add 5 marks

ArrayList<Integer> marks = **new** ArrayList();

marks.add(0);

marks.add(10);

marks.add(11);

marks.add(15);

marks.add(20);

List<Integer> collect = marks.stream().map(i->i+5).collect(Collectors.*toList*());

System.***out***.println(collect);

3] Count :- If we want to how many object int list then go for count()

Eg.1

//count() count fail student

ArrayList<Integer> marks = **new** ArrayList();

marks.add(45);

marks.add(40);

marks.add(11);

marks.add(15);

marks.add(20);

**long** count = marks.stream().filter(i -> i < 35).count();

System.***out***.println(count);

4] **Collect** :- It collect the result.

5] **Sorted** :- Sort date according to natural sorting

Eg. 1

//sorted() I want to sort list

ArrayList<Integer> marks = **new** ArrayList();

marks.add(45);

marks.add(40);

marks.add(11);

marks.add(15);

marks.add(20);

List<Integer> collect = marks.stream().sorted().collect(Collectors.*toList*());

System.***out***.println(collect);

Eg.1 //Descending

ArrayList<Integer> marks = **new** ArrayList();

marks.add(45);

marks.add(40);

marks.add(11);

marks.add(15);

marks.add(20);

//List<Integer> collect = marks.stream().sorted((i1,i2)->(i1<i2)?1:(i1>i2)?-1:0).collect(Collectors.toList());

List<Integer> collect = marks.stream().sorted((i1,i2)->-i1.compareTo(i2)) .collect(Collectors.toList());

System.out.println(collect);

Eg.3 // natural sort

ArrayList<String> al = **new** ArrayList();

al.add("kajal");

al.add("sunny");

al.add("prabhas");

al.add("anushka");

al.add("mallika");

List<String> collect = al.stream().sorted().collect(Collectors.*toList*());

System.***out***.println(collect);

Eg.4 // custome descending

ArrayList<String> al = **new** ArrayList();

al.add("kajal");

al.add("sunny");

al.add("prabhas");

al.add("anushka");

al.add("mallika");

List<String> collect = al.stream().sorted((i1,i2)->-i1.compareTo(i2)).

collect(Collectors.*toList*());

System.***out***.println(collect);

Eg.5

// list to increasing order

ArrayList<String> al = **new** ArrayList();

al.add("kajal");

al.add("sunny");

al.add("prabhas gjh");

al.add("anushka");

al.add("mallika");

Comparator<String> c= (s1,s2)->{

**int** l1=s1.length();

**int** l2 = s2.length();

**if**(l1<l2){

**return** -1;

}**else** **if**(l1>l2){

**return** 1;

}**else**{

**return** 0;

}

};

List<String> collect = al.stream().sorted(c).collect(Collectors.*toList*());

System.***out***.println(collect);

6] **min() and max()** :- it contain comparator parametor

Eg.1

//min() and max():-

ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

Integer min = al.stream().min((i1,i2)->i1.compareTo(i2)).get();

Integer max = al.stream().max((i1,i2)->i1.compareTo(i2)).get();

System.***out***.println(min);

Eg. Sum

ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

**long** sum = al.stream().mapToLong(i->i).sum();

System.***out***.println(sum);

7] **forEach**() :- It is used to travers element.

Eg.1

//ForEach element process into list u

ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

al.stream().forEach(System.***out***::println);

Eg.2

//Sqare of each element

ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

al.stream().forEach(i->System.***out***.println("square of "+i+"=="+(i\*i)));

7] **toArray**() :- Convert Stream into Array

Eg.1 //toArray :- Stream convert into array

ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

Integer[] array = al.stream().toArray(Integer[]::**new**);

**for**(Integer i :array){

System.***out***.println(i);

};

8] **Stream.of()** :- Convert array into stream

Eg.1 ArrayList<Integer> al = **new** ArrayList();

al.add(0);

al.add(10);

al.add(11);

al.add(15);

al.add(20);

Integer[] array = al.stream().toArray(Integer[]::**new**);

Stream.*of*(array).forEach(System.***out***::println);

// create Stream and travers

/\*Stream s=Stream.of(99,999,9999,99999);

s.forEach(System.out::println);\*/

9**] Date and Time**:-

/////Date and Time..java.time.\* pkg

LocalDate date = LocalDate.*now*();

System.***out***.println(date);

LocalTime time = LocalTime.*now*();

System.***out***.println(time);

**int** dd = date.getDayOfMonth();

System.***out***.println(dd);

**int** mm = date.getMonthValue();

System.***out***.println(mm);

**int** yyyy =date.getYear();

System.***out***.println(yyyy);

**int** hh = time.getHour();

System.***out***.println(hh);

**int** ss = time.getMinute();

System.***out***.println(ss+"");

//System.out.println(dd+"-"+mm+"-"+yyyy+" "+hh+":"+ss);

System.***out***.printf("%d-%d-%d %d:%d", dd,mm,yyyy,hh,ss);

String s=dd+"-"+mm+"-"+yyyy+" "+hh+":"+ss;

// System.out.println(s);

// System.out.printf("%d:%d", hh,ss); //for formate

System.***out***.println("--------"+date.minusMonths(1));

System.***out***.println(date.plusDays(2));