# **Java 8**

* From java 1.8 version we can define static method and default method inside an interface.
* We can also define main method inside an interface
* Ex-
* Interface interf{
* Psvm(String args[]){
* System.out.println(“Inside Inteface”);
* }
* }
* **Functional Interface**
* Functinal interface is the interface which contain only one abstract method and any number of default and static methods.
* Predefined Functional interface are Predicate , Function ,Supplier , Consumer – these predefined functional interface are present in **java.util.function**
* **Predefined Functional Interface – Predicate**
* Predicate interface is the boolean value returned interface which will perform comparison operation.
* Introduced from java 1.8 version
* Contain abstract method **test,** test function takes the input value and perform comparison and return Boolean value
* **Example -**
* public class Test{

psvm(String args[]){

Predicate<Integer> p = I -> I>10;

sop(p.test(100));//true

sop(p.test(5));//false

}

}

public class Test{

psvm(String args[]){

Predicate<String> p = s -> s.length() > 5;

sop(p.test("abcdef"));//true

sop(p.test("abc"));//false

}

}

-----Check whether a collection is empty or not

public class Test{

psvm(String args[]){

Predicate<Collection> p = c -> c.isEmpty();

ArrayList<Integer> arr1 = new ArrayList<>();

arr1.add(10)

sop(p.test(arr1));//false

ArrayList<Integer> arr2 = new ArrayList<>();

sop(p.test(arr2));//true

}

}

* **Example 2:- Display names which are starts with ‘K’**

String[] names =["Kajal","katrina","Madhuri","Shradha","kareena"];

Predicate<String> startsWithK = s -> s.charAt(0) == 'K';

for(String s : names){

if(startsWithK.test(s)){

sop("name starts with K =" + s);

}

}

* Example 3:- **Remove null values and empty string from the given list**

String[] strarr = {"apple","","orange","cucumber","","lemon",null};

List<String> arr = new ArrayList<>();

Predicate<String> p = s -> s != null && s.length() != 0;

for(String s: strarr){

if(p.test(s)){

arr.add(s);

}

}

System.out.println(arr.size());

* **Predefined Functional Interface - Function**
* Function can return any type.
* It has an abstract method “apply”
* Example :-

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

System.out.println(f.apply("Riya"));

* **Method Reference :: double colon**
* **Method reference is used for code reusability**
* Example 1 :-
* interface Interef{
* public void m1();
* }
* public class Test {
* public static void m2(){
* System.out.println("Method reference....reuse code");
* }
* public static void main(String args[]){
* /\*
* \* m1() method is the interfaec Interef method
* \* and m1() has the same line of code as m2() in Test class
* \* so we re use the code of m2() for m1()
* \*
* \* \*/
* **Interef i = Test::m2;**
* i.m1();
* }
* }
* **Ouptput**:-
* Method reference....reuse code
* Only
* Only condition is that referring method and referrer method should have same argument list
* Syntax:-
* **For Static**:-
* Classname::method name
* **For Instance method:-**
* Object ref::method name

Example 2:-

* Create Thread Class by Implementing Runnable Interface
* Runnable Interface is the Functional Interface which has only abstract method run()
* **Streams**
* **In Java, two packages are having streams**
* **Java.io = java.io package streams are used for performing**

**input or output operation in file system**

* **Java.util = java.util package streams are used for performing operation in collection**
* **Collection = Collection is the group of objects represented in a single entity.**
* **Streams = If we want to process objects from the collection,then we should go fo streams**

**For ex – suppose a collection contains 100 Student object and I need to prepare a list of Students are having percentage > 80%. That means I need to process each Student object marks property.**

**Stream methods are -**

**Example :- 1 . filter()**

* **ArrayList<Integer> l1 = new ArrayList<Integer>();**
* l1.add(0);l1.add(10);l1.add(5);l1.add(20);l1.add(15);
* System.out.println(l1);
* /\*
* \* Finding Even Numbers from list
* \* without using Streams
* \*/
* ArrayList<Integer> l2 = new ArrayList<>();
* for(Integer i: l1){
* if(i%2 == 0){
* l2.add(i);
* }
* }
* System.out.println("Even Num List without streams= "+ l2);
* /\*
* \*
* \* Finding Even Numbers from list
* \* with Streams (java.util) package
* \*
* \*/
* ArrayList<Integer> l3 = (ArrayList<Integer>) **l1.stream().filter(I -> I%2==0).collect(Collectors.toList());**
* System.out.println("List of even num with Stream = "+l3);
* **Example :- 2. map()**
* /\*
* \* Stream map
* \* map function is used to create a different set of collection or object from the
* \* existing collection
* \*
* \* suppose we want to double the value of the collection
* \*
* \*/
* ArrayList<Integer> l4 = (ArrayList<Integer>) **l1.stream().map(I -> I\*2).collect(Collectors.toList()**);
* System.out.println("Double value of list using Stream map = "+l4);
* **NOTE :-**
* /\*
* \* **Stream is an interface present in java.util.stream package**
* \* here l1 is the ArrayList but it can be any collection object
* \* **stream() is present inside the Collection interface as the default method**
* \*
* \*/
* **Stream s = l1.stream();**
* System.out.println(s);

**Example :- 3. count()**

* **/\***
* \* **Processing by count() method**
* \* count() returns the number of elements present in stream
* \* its return type is long
* \*
* \*/
* ArrayList<String> strarr =new ArrayList<>();
* strarr.add("pen");
* strarr.add("pencil");
* strarr.add("eraser");
* strarr.add("note book");
* **long count = strarr.stream().filter(S ->** **S.length()>=4).count();**
* System.out.println("Length of stream =" + count)
* **Example : 4. sorted()**
* **/\***
* \* sorted() method - sort element inside Stream
* \* **sorted()** => for default natural sorting i.e ascending order
* \* **sorted(Comparator c**) => for customized sorting
* \*
* \*/
* System.out.println("Before sorting"+ l1);
* List<Integer> ls = **l1.stream().sorted().collect(Collectors.toList());**
* System.out.println("After sorting - natural sort ="+ls);
* //**sorted(Comparator c) => customized sorting - we need to create list**
* **//of descending order**
* List<Integer> ls1 = **l1.stream().sorted((i1,i2) -> -i1.compareTo(i2)).collect(Collectors.toList());**
* System.out.println("Sorted in descending order customized ="+ls1);//here - (-ve) sign will reverse the sorting order
* **Example** :- 5.**min() and max()**
* **I)** min(Comparator c) => return the minimum value according to the Comparator specified
* II) max(Comparator c) => return the maximum value according to the Comparator specified
* /\*
* \* min(Comparator c) and max(Comparator c)
* \*
* \*/
* **Integer min = l1.stream().min((i1,i2) -> i1.compareTo(i2)).get();**
* System.out.println("Minimum value of Collection = "+min);
* //for max(Comparator c)
* **Integer max = l1.stream().max((i1,i2) -> i1.compareTo(i2)).get();**
* System.out.println("Max value inside the collection +"+max);
* **Example :- 6. forEach()**
* **/\***
* \* forEach()
* \*
* \*/
* List<String> strArr = new ArrayList<>();
* strArr.add("ABC");strArr.add("CCC");strArr.add("DDD");strArr.add("GGG");
* **strArr.stream().forEach(S -> System.out.println(S));**
* **//with method reference**
* **strArr.stream().forEach(System.out::println);**
* **Example 7 : toArray()**
* **/\***
* \* toArray()
* \* **we use the toArray() method to copy the element of the Stream inside the**
* \* **array**
* \*
* \*/
* ArrayList<Integer> sArray = new ArrayList<>();
* sArray.add(0);
* sArray.add(10);
* sArray.add(5);
* sArray.add(20);
* sArray.add(2);
* sArray.add(25);
* **Integer[] arr = sArray.stream().toArray(Integer[]::new)**;
* for(Integer x: arr){
* System.out.println(x);
* }
* **Example 8: Stream.of() - used for non collection**
* 1.For group of values
* **/\***
* \* Stream.of()
* \* We can also apply Stream for group of values and array
* \*
* \*/
* **Stream<Integer> st = Stream.of(9,99,999,9999,999999);**
* st.forEach(System.out::println);
* 2.For Arrays
* //2.For Arrays
* Double[] d = {10.1,10.2,10.3,10.4};
* //Convert array of double values into Stream
* **Stream<Double> sd = Stream.of(d);**
* sd.forEach(System.out::println);
* **Date and Time API**
* Upto java 1.7 version Date,Calender,TimeStamp ...etc calsses are not recommended to use in program due to inefficiency
* So, in Java 1.8 version Date and Time api is itroduced and it is also known as **JODA Time AP**I as it is developed by **joda.org**
* **Example 1:-**
* **Package – java.time.\***
* **/\***
* \*
* \* JODA Time API
* \*
* \*/
* **LocalDate date = LocalDate.now();**
* System.out.println(date);
* **LocalTime time = LocalTime.now();**
* System.out.println(time);
* **Example 2:-**
* int dd= date.getDayOfMonth();
* int mm = date.getMonthValue();
* int yyyy = date.getYear();
* System.out.printf("%d-%d-%d",dd,mm,yyyy);
* **Example 3:-**
* int hour = time.getHour();
* int minute = time.getMinute();
* int sec = time.getSecond();
* int nsec = time.getNano();
* System.out.println("");
* System.out.printf("%d:%d:%d:%d", hour,minute,sec,nsec);
* **Example 4:- LocalDateTime Class**
* LocalDateTime dt = LocalDateTime.now();
* System.out.println(dt);
* int dd1 = dt.getDayOfMonth();
* int mm1 = dt.getMonthValue();
* int yy1 = dt.getYear();
* System.out.println("########");
* System.out.println("");
* System.out.printf("%d-%d-%d",dd1,mm1,yy1);
* int hour1 = dt.getHour();
* int minute1 = dt.getMinute();
* int sec1 = dt.getSecond();
* int nsec1= dt.getNano();
* System.out.println("");
* System.out.printf("%d:%d:%d:%d", hour1,minute1,sec1,nsec);
* **Example 5: - Customized Date and Time**
* **LocalDateTime cLdt = LocalDateTime.of(2020,Month.JULY,30,12,15,00);**
* System.out.println(cLdt);
* System.out.println(**"After Six months = "+ cLdt.plusMonths(6));**
* System.out.println("**Before six months = "+ cLdt.minusMonths(6));**
* **Example 6:- Use of Period Class**
* **/\***
* \* Getting age with Period
* \*
* \*
* \*/
* LocalDate birthday = LocalDate.of(1993, 7, 30);
* LocalDate today = LocalDate.now();
* **Period p = Period.between(birthday, today);**
* System.out.printf("Age is %d yrs %d months %d days",p.getYears(),p.getMonths(),p.getDays());
* **Example 7 :- Year Class**
* /\*
* \* Year Class
* \*
* \* Finding whether a year is leap yr or not
* \*
* \*/
* Scanner sc = new Scanner(System.in);
* System.out.println("Enter Year");
* int n = sc.nextInt();
* Year y = Year.of(n);
* if(y.isLeap()){
* System.out.printf("%d is Leap year",n);
* }
* else{
* System.out.printf("%d is not Leap year",n);
* }
* **Example 8: ZoneId**
* /\*
* \* ZoneId
* \*
* \*/
* ZoneId zone= ZoneId.systemDefault();
* System.out.println(zone);
* ZoneId la = ZoneId.of("America/Los Angeles");
* ZonedDateTime zt = ZonedDateTime.now(la);
* System.out.println(zt);
* **LocalDate** – getDayOfMonth() , getMonthValue() , getYear()
* **LocalDateTime** – getHour() ,getMinute() ,getSecond() , getNano()