**Java8**

* Interface method are by default public and abstract and var are by default public static and final.

**Features:**

* Lambda expression
* Functional interface(used to call lambda expression)
* Interface static and default method.
* Predicate ,function and consumer (Predefined functional interface)
* Method reference and constructor reference by using (:: operator)
* Stream API
* Date and Time Api(Joda api)

**Lambda Expression(anonymous function ,no name ,no modifier and no return type):**

* Used to enable functional programming and parallel processing.

**Functional interface:**

* Functional interface contains only one abstract method eg(Runabble,callable,comparable)
* It is used to invoke lambda expression
* @FunctionalInterface annotation used to declare interface as functional interface.
* It contains any no of static and default method.

Collection:

List: insertion order preserved, and duplication allowed.

Set: duplicate not allowed and order is not preserved.

Map: key value pair insertion order not preserved.

Comparator: customized sorting (compare method abstract) asc is default sorting order

**Interface static and default method:**

**Default Method:** default method is defined in interface using default keyword and the implementing class can directly call or override that method.

**Note:**

Incase of multiple inheritance if two interface has same default method signature than complier will throw error to solve this either override the method and give your own implementation or in overridden method specify the interface whose method you want to use by (interfacename.super.methodname)

|  |  |
| --- | --- |
| **Interface** | **Abstract class** |
| All var are public static and final we cannot declare instance var | we can declare instance var which are req to the child class |
| It never talks about state of object | It talks |
| We cannot declare constructor | We can declare |
| Functional interface with default method can refer lambda exp | It cannot refer lambda exp |
| Inside interface we cannot override object class method | We can |
| Can not declare instance and static blocks | We can |

**Static Method:** purpose : just to define general utility methods (defined using static keyword)

Static method can be called by using interface Name only, interface static method is not available by default in implemented class so there is no concept of overriding exact same method you can define no issue.

Note: As static method we can define in interface so we can write main method in interface and we can run interface also.

**Predefined Functional Interface: (Present in java.util.Function )**

* **Predicate (I) :** test abstract method is present which return Boolean value function

**Eg:** public Boolean test(T t)

**Predicate joining:** p1.and(p2) and p1.or(p2) where p1 and p2 are two different predicate .To reverse the value of predicate use p1.negate().

**interface Predicate<T>**

**{**

public Boolean test(T t)

**}**

**Function Functional Interface** : To apply certain operation and can return any type of value.

**interface Function<T,R> T:input param ,R: return type**

**{**

**R apply(T t) ;**

**}**

**Function chaining:** f1.andThen(f2) first f1 will be applied than f2 .to reverse order we can use f1.compose(f2) .these are default method in function interface.

**Function interface static method Identity : Function.Identity return exact same output as input.**

**Consumer Functional interface:**

**interface Consumer <T>**

**{**

**Void accept(T t)**

**}**

**Consumer chaining :** andThen method

C1.andThen(c2).andThen(c3).accept(obj)

**Supplier Functional interface:**

**interface Supplier<R> R: return type**

**{**

**R get();**

**}**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Properties** | **Predicate** | **Function** | **Consumer** | **Supplier** |
| **use** | To take some input and perform some conditional checks | To take some input and perform req operation and return null or any value | To consume some input and perform req operation but don’t return anything | To supply some val based on our req |
| **Abstract method** | public boolean test (T t) | public R apply(T t) | Public void accept(T t) | Public R get() |
| **Default method** | and ,or ,negate | andThen ,compose | andThen() | - |
| **Static method** | isEqual | Identity | **-** | **-** |

**Method Reference and constructor reference (::) :**

**Method reference** : to use method reference only one condition is there the referring method and referred method should have same argument.

Eg: if static method then (classname:: methodname)

If nonstatic(instance ) then (obj reference :: methodName)

**Constructor reference** : className:: new

**Streams: (Process object from the collection)**

**Filter:** if we want to filter elements from collection based on some boolean condition.

**Eg: Stream s1 =**collection.stream.filter(i->i%2===0)

**Map:** if we want to create separate new object for every collection obj based on some function.

Eg: **Stream s1 =**collection.stream.map(i->i\*2)

Stream s=collection.stream() --- stream method is a default method introduced in collection interface

Stream is an interface present in java.util.stream package

Public Stream filter (Predicate <T>) any boolean valued function.

Public Stream map(Function<T,R> f)

After getting Stream we need to do two steps :

* Configuration – By using filter or map method.
* Processing – can be done by various methods.

Streams Methods:

* Collect() : method collects elements from the stream and add it to the specific collection.
* Count(): no of element present in stream . return type is long.
* Sorted() : sort based on default or customized sorting order (Comparator compare method).
* Min and max() : list should be sorted to apply min and max method.

Min(Comparator c) ,max(Comparator c).

Eg: l.stream().min(I1,I2 ->I1.compareTo(I2)).get();

* forEach() : ( not equivalent to foreach loop but more or less it is same )

It does not return any value and it can take lambda exp or method ref (::) as argument and apply that lambda exp for each element present in stream.

* toArray() : to copy elements of steam into specific array.

Eg: Integer [] arr=l.stream.toArray(Integer[]:: new)

* reduce () : a stream reduces to single resultant value, for example, maximum, minimum, sum, product, etc. *reduce* operation applies a binary operator to each element in the stream where the first argument to the operator is the return value of the previous application and second argument is the current stream element.

Eg: l.stream.reduce(lambda exp)

**Stream can also be applied on group of values and array, to work on this Stream contains one method of(). Eg: Stream.of()**

* **Group of value:** Stream s =Stream.of(9,99,99999,9999) after that we can apply any stream method .
* **For array:** Double [] d= {10.2,12,3,34.5}

Stream<Double> s= Stream.of(d)

**Optional class :** To reduce null pointer check optional class is introduced.  By using Optional, we can specify alternate values to return or alternate code to run.  If a value is present, **isPresent()** will return true and **get()** will return the value

**Collections**: