# JAVA 8 FEATURES

## Inner Class:

The class which is declared as a member of another class is known as inner class or nested class.

There are three types of inner classes.

1. Static inner class
2. Member or non-static inner class
3. Anonymous inner class

**Static inner class: -**

A class which is declared as a static member of another class is known as static inner class.

➔ To access static member of inner class we use

```
OuterClassName.InnerClass.staticmember
OuterClass.InnerClass.innerMethod()
```

➔To access non-static member of inner class we need to create an object of inner class by using class name of outer class and by using inner class reference we will call the non-static member.

```
Outerclassname.innerclassname var = new Outerclass.innerClass();

Var.nonstatimethod()
```

**Member class or non-static inner class: -**

A class which is declare as a non-static member of another class is known as non-static inner class.

⇨ To access the non-static member or member class we need to create an object of inner class by using the object reference of outer class.

```
Ex:- OuterClass var =   new OuterClass();
     OuterClass.Innerclass var2 = var.new InnerClass();
        Var2.innermethod().
```

**Difference between comparable and comparator: -**

| **Comparable** | **Comparator** |
|---|---|
| 1. From lang packag0e<br>2. public int compareTo(Object o1)<br>3. Single comparing parameter<br>4. Implemented in comparing class | 1. From util package<br>2. Public int compareTo(Object o1, Object o2)<br>3. Multiple comparing parameter.<br>4. Implemented in external class |

**Anonymous inner class: -**

Anonymous inner class which does not have a class declaration and it is declared / initialize it as local member of main method.

- ➔ Anonymous class is used for creating an object of implementing class of interface and abstract class, and anonymous class is used for providing the implementation of abstract methods of interface and abstract class.
- ➔ For providing the implementation anonymous class will be created as the implementing class or extending class of interface and abstract class.
- ➔ Since anonymous class doesn't have a name while creating an object of anonymous class. We will use the name of parent interface or abstract class while calling a constructor of anonymous class.

Ex:-

```
@Comparator<Student> com = new Comparator<Student>(){
        @Override
        Public int compare(Studento1, Student o2){
                Return o1.getName().compareToIgnoreCase(o2.getName());
        }
};
```

In this example we are creating an object of implementing class comparator interface by using Anonymous implementation and inside class block of anonymous class we are overriding compare method from comparator interface.

The dot class file is created for the anonymous class will be ($) followed by number of anonymous class is created in the project $.class.

NOTE: -

Other then providing implementation to the abstract methods of abstract class. An abstract class anonymous class can also be used for creating an object for abstract class with only concrete method from parent class or interface.

**Java 8 update :**

- ➔ **Update of interface**

  In java 8 update static method and default non-static method are introduced and both of these methods are concrete methods in interface.

  The default method created inside the interface the method signature of the default method should not match the non-static method signature of the object.class

**Lambda Expression: -**

It is introduced in java 1.8 update and it is used to provide implementation to the abstract method of functional interface and abstract with one abstract method.

While providing the implementation to the abstract method. We need to provide the number of arguments inside the parenthesis of lambda expression which is accepted by the abstract method as a parameter.

And inside the block of lambda expression we provide the line of implementation which is suppose to be return as implementation to the abstract method after providing the implementation to the abstract method by using lambda expression. Lambda expression returns the object of implementing class of a specific functional interface of abstract class.

**Syntax: -**

```
()->{}
```

- ➔ Lambda parenthesis/parameter (()): - As a parameter of lambda we provide same number of arguments that is accepted by the abstract method as a parameter.
- ➔ Lambda token (->): - Lambda token is a symbol which represents the use of lambda expression in the program. Lambda token is mandatory for using lambda expression.
- ➔ Lambda block ({}): - Inside the block of lambda expression, we write line of code which is suppose to be used as a implementation of the abstract method.

On the basis of parameter and implementation of lambda expression, java internally decides which abstract method we are trying to implement.

**Stream API: -**

This api was introduced in java 8 update. It is used to perform multiple operations on collection object.

**Stream interface: -**

This interface contains method to perform operation on collection objects and also contains the method to manage collection object. This interface is from java.util.stream package.

**Stream method : -**

This method is from collection interface. It is a default method which is used to convert current collection type object to stream type object. It has a default parameter and returns an object of implementing class of stream interface.

**Note: -** Stream object can contain multiple data and this object as a behavior of passing the data which is contained inside it one at a time.

**Filter method/ filter(): -**

Filter is a method from stream interface. It is used to filter the data from stream object to filter the data filter method sends the data to the test method of Predicate interface whose object is provided to the filter method as an argument.

Filter method savevs the data for which test methpd from Predicate returns true.

Filter method returns stream object contained / containing filter datat and filter method accepts an object of implementing class of Predicate interface as an argument.

**Predicate: -** Predicate is a functional interface from java.util.function package. It as an abstract method called test. Test method as object type parameter and Boolean as return type.

**Collect method / collect (): -**

Collect is method from stream interface. This method is used to pass the stream of data to the methods of Collectors class.

**Collectors class: -**

It is a class from java.util.stream package and it as methods like toList(), toSet() etc., which accept multiple data and returns collection type object containing those data.

Example of using stream API to filter list containing integer data.

List<Integer> l = list.stream().filter(e -> e%2 ==0).collect(collectors.toList());

For each method / forEach (): -

This method is from iterable interface. It is a default method which as an implementation for each loop / enhanced for loop.

ForEach method accepts an object of consumer interface as an argument and for the data type implementated for each loop inside forEach method it refers to the current iterable object type and the implementation of this loop is provided from accept method from consumer interface.

Reference.forEach(e->System.out.println(e));

**Consumer interface: -** It is a functional interface from java.util..function package. It as an abstract method called accept.

Accept method as parameter of object type and has void as a return type.

Ex:- list.forEach(e->sop(e));

**Function interface: -**

Function is a functional interface from java.utill.function package and it as an abstract method called apply. Apply method accepts object type argument and returns object type value.

**Supplier interface: -**

It is a functional interface from java.util.function package. It was introduced in java 8 and as an abstract method called as get, this get method has default parameter, however it has return type of Object.