**Java 8 features:-**

* Static and Default methods in interfaces
* Functional interfaces
* Lambda Expressions
* Optionals
* Collection API
* LocaleDateTime API

Static and Default Methods:-

Java 8 introduced static and default methods in interfaceso that we can handle the multiple inheritance implementation problem.

And allow developers to add some implementation (logic) such that it allows developers to use that globally.

Ex1:-

static int add(int a, int b) {  
 return a + b;  
}

Ex2:-

// with generics  
default int byIdByGenerics(List<T> data) {  
 for (T d : data) {  
 System.*out*.println(d);  
 }  
 return 0;  
}

Static Method :-

With the help of Static methods we can add some default implementation so that we can use this across the application.

Ex:-

static int add(int a, int b) {  
 return a + b;  
}

We can call (or) invoke static method by **Referring to The Interface**.

Ex:-

int add = StaticAndDefaultInterface.*add*(11, 19);  
System.*out*.println(add);

NOTE:- We know Static methods can’t be overridden

Default Method:-

Default Methods are also same as static method but developer can override the implentation of it whenever it is necessary.

Ex:-

default int add1(int a, int b) {  
 return a + b;  
}

We can call (or) invoke Default method by creating object of the class which implements the interface.

Ex:-

// invoking default method  
StaticAndDefaultInterfaceImpl ss = new StaticAndDefaultInterfaceImpl();  
int i = ss.add1(11, 19);  
System.*out*.println(i);

Note:- Default methods can be overridden. as like normal way of overriding (run time polymorphism).

Ex:-

@Override  
public int add1(int a, int b) {  
 return a \* b;  
}

Lambda Expressions:-

1

Lambda Syntax:-

()->{};

Where

* **()** is known as anonymous function or nameless function
* -> is a arrow which points to the implementation.
* {} inside this where the implementation (or) logic will be added.

Note:- if there is only one line then no need to add {} and return statement.

Advantage of lambda:-

* + No need to provide data types and return type.
  + Simple syntax.

**Purpose:-**

* + No need to implement interface to override the abstract method.
  + Simple syntax, easy readability.
  + No need to create object(or) Instance of the class to invoke (or) call the abstract method of an Functional Interface.
  + We can make an interface as a functional interface by adding @FunctionalInteface annotation but this annotation is optional but if added it will e much more readable to the developer.

Ex:-

@FunctionalInterface // optional

public interface \_FunctionalInterface {

abstract String appendString(String s, int n);

}

Invocation of Abstract method.

Ex:-

\_FunctionalInterface a = (s, n) -> {  
 if (s.startsWith("a")) return s;  
 return "";  
};

String string = a.appendString("abc", 10);