Features

1. Lambda Expression
2. Functional Interface

Lambda expression

What is Lambda expression?

It is anonymous function. A function which don’t have name, access modifier, return type is called anonymous function. Lambda expression is an anonymous function. The type of the lambda expression is a functional interface.

Benefits of lambda expression

1. To enable Functional programming
2. To write more readable, maintainable and concise code
3. To use API easily and effectively.
4. To enable parallel processing

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

Functional Interface

What is Functional Interface?

If an interface contains only one abstract method, we can say it as functional interface.

Runnable -> run(), callable -> call are examples for functional interface. Also functional interface have any number of default and static methods. The restriction will only apply for abstract methods only.

Use of @FunctionalInterface annotation

To indicate the interface as functional interface. It is not a mandatory one, but it is a additional option define the functional interface.

Advantage of *Functional* interface annotation

The *functional* interface annotation is an optional one.

When we add another abstract method inside the functional interface the error will thrown at the implementation level(Lambda Expression) If we have @FunctionalInterface annotation compiler will notify at interface level.

It helps to avoid the impact of future modification

Functional Interface Inheritance

If the parent interface is a functional interface and it extends to child functional interface, and child interface doesn’t have any abstract methods then both parent and child are proper functional interfaces. Since both interface have single abstract methods(We declare abstract method at parent level and same method will available for extended interface)

If the parent interface is a functional interface and it extends to child functional interface, and child interface has one abstract method with same as parent then both parent and child are proper functional interfaces. Since both interface abstract methods(We declare abstract method at parent level and same method override for extended interface)

If the parent interface is a functional interface and it extends to child functional interface, and child interface has one abstract method with different name then child will not be the functional interface. Since it has two abstract methods (We declare abstract method at parent level and same method will available for extended interface also child interface have multiple abstract methods so it won’t be functional interface)

If the parent interface is a functional interface and it extends to child interface(Not functional interface), then child interface can have any number abstract, default and static methods.

If the interface is created with 0 abstract methods, then the compiler error will throw.

Why functional interface should contain only one abstract method?

The purpose of the functional interface is to invoke lambda expression. If there is more than one abstract method then the compiler cannot identify, which method can invoke lambda expression. Then the compiler will throw, “Incompatible type : Interface is not a functional interface” error.

In general, the abstract keyword is not used to define the abstract method inside the interface. Why?

In *the* interface when we define abstract method *implicitly* it considered as Abstract. We don't need to specify *the* abstract keyword