Introduction to Functional Interface in Java

It was first introduced in java 8. A functional interface can be defined as an interface with a single abstract method. This means functional interfaces in java provide only a single basic functionality. However, a functional interface can contain static and default methods, in addition to a single abstract method. java.util.function.Function, java.util.function.Predicate, UnaryOperator, BinaryOperator, Supplier, Consumer are examples of built-in functional interfaces in java.

Syntax of Functional Interface

```
public interface MyFunctionalInterface(){
// abstract method
public void functionalMethod();
}
```

From the above syntax, we can see that the interface named MyFunctionalInterface contains only a single unimplemented method; that's why it can be considered as a functional interface. It can also contain static and default methods like the one shown below:

```
public interface MyFunctionalInterface(){
public default void defaultMethod(){
// default method logic goes here
}
public static void staticMethod(){
// static method logic goes here
}
// abstract method
public void functionalMethod();
}
```

Also, a functional interface can be implemented using Lambda expression like the following:

```
MyFunctionalInterface functionalInterface = () ->{
// basic functionality logic goes here
}
```

We can also use an annotation or declare an interface as a functional interface. Here is how a functional interface can be declared using an annotation:

```
MyFunctionalInterface functionalInterface = () ->{
// basic functionality logic goes here
}
```

@FunctionalInterface was **introduced in java 8** and is used for compiler level error in case an interface breaks rules of a functional interface. Declaring an interface using @FunctionalInterface annotation makes an interface functional, and if more than one abstract method is used, it will generate a compilation error.

Important Points Regarding Functional Interface

- Only one abstract method is allowed in a function interface. If @FunctionalInterface annotation is not used with a function interface, then more than one abstract method can be declared, but in that case, that interface will be regarded as non-functional.
- Use of @FunctionalInterface annotation is optional; it is only used for compiler level checking.
- A Functional interface can contain any number of static and default methods.
- Overriding methods from the parent class do not break the rules of a functional interface.

Example:

```
@FunctionalInterface
public interface MyFunctionalInterface(){
// abstract method
public void functionalMethod();
@Override
public boolean equals(Object object);
//method overridden from parent class
}
```

Since the above interface overrides a method from the parent class and does not declare multiple abstract methods, it can be considered as a functional interface.

Examples to Implement Functional Interface

Example #1

In this example, we will show how built-in function interface java.util.function.function interface is used. Here is the declaration of the Function interface.

Interface:

```
package java.util.function;
public interface Function<T,R>{
public <R> apply(T inputparams);
}
```

Code:

```
import java.util.function.*;
public class FunctionalInterfaceDemo implements
Function<Integer, Integer>{
@Override
public Integer apply (Integer n){
```

```
return n*n;
}
public static void main (String args[]){
FunctionalInterfaceDemo demo = new FunctionalInterfaceDemo
();
Integer sqroot= demo.apply(12);
System.out.println("Square root of 12 is " + sqroot);
}
}
```

Example #2

In this example, we will see how these interfaces are created using lambda expressions.

Code:

```
public class FunctionalInterfaceDemo{
public static void main (String args[]){
// creating functional Interface instance
Runnable r = () -> {System.out.println ("Executing
Thread........");};
new Thread(r).start();
}
}
```

Example #3

In this example, we will see the use of another built-in interface consumer to iterate a List.

Code:

```
import java.util.function.*;
import java.util.*;
public class FunctionalInterfaceDemo{
public static void main (String args[]){
List<String> list = new ArrayList<String>();
list.add("One");
list.add("Two");
```

```java
list.add("Three");
list.add("Four");
list.add("Five");
list.add("Six");
// Iterate arraylist using consumer
list.forEach(new Consumer<String>(){
@Override
public void accept(String item){
System.out.println(item);
}
});
}
}
```