# Java-8-new-features

## **forEach() method in Iterable interface.**

## **Interface Default and Static Methods**

Before Java 8, interfaces could have only public abstract methods. It was not possible to add new functionality to the existing interface without forcing all implementing classes to create an implementation of the new methods, nor it was possible to create interface methods with an implementation.

Starting with Java 8, interfaces can have ***static*** and **default** methods that, despite being declared in an interface, have a defined behavior.

**2.1. Static Method**

Consider the following method of the interface (let’s call this interface *Vehicle*):

|  |  |
| --- | --- |
| 1  2  3 | static String producer() {      return "N&F Vehicles";  } |

The static *producer()* method is available only through and inside of an interface. It can’t be overridden by an implementing class.

To call it outside the interface the standard approach for static method call should be used:

|  |  |
| --- | --- |
|  | String producer = Vehicle.producer(); |

**2.2. Default Method**

Default methods are declared using the new ***default*keyword**. These are accessible through the instance of the implementing class and can be overridden.

Let’s add a *default* method to our *Vehicle*interface, which will also make a call to the *static* method of this interface:

|  |  |
| --- | --- |
|  | default String getOverview() {      return "ATV made by " + producer();  } |

Assume that this interface is implemented by the class *VehicleImpl.*For executing the *default* method an instance of this class should be created:

|  |  |
| --- | --- |
|  | Vehicle vehicle = new VehicleImpl();  String overview = vehicle.getOverview(); |

## **Functional Interfaces and Lambda Expressions.**

Lambda expression is a new feature which is introduced in Java 8. A lambda expression is an anonymous function. A function that doesn’t have a name and doesn’t belong to any class. The concept of lambda expression was first introduced in LISP programming language.

**Java Lambda Expression Syntax**

To create a lambda expression, we specify input parameters (if there are any) on the left side of the lambda operator ->, and place the expression or block of statements on the right side of lambda operator. For example, the lambda expression (x, y) -> x + y specifies that lambda expression takes two arguments x and y and returns the sum of these.

//Syntax of lambda expression

(parameter\_list) -> {function\_body}

**Lambda expression vs method in Java**

A method (or function) in Java has these main parts:  
1. Name  
2. Parameter list  
3. Body  
4. return type.

A lambda expression in Java has these main parts:  
Lambda expression **only has body and parameter list**.  
1. **No** name – function is anonymous so we don’t care about the name  
2. Parameter list  
3. Body – This is the main part of the function.  
4. **No** return type – The java 8 compiler is able to infer the return type by checking the code. you need not to mention it explicitly.

**Where to use the Lambdas in Java**

To use lambda expression, you need to either create your own functional interface or use the pre defined functional interface provided by Java. An interface with **only single abstract method** is called functional interface(or Single Abstract method interface), for example: Runnable, callable, ActionListener etc.

**To use function interface:**  
Pre Java 8: We create anonymous inner classes.  
Post Java 8: You can use lambda expression instead of anonymous inner classes.

# How to Fix the Target Type of This Expression Must be a Functional Interface Error?

This post talks about how to resolve "**the target type of this expression must be a functional interface**" error while trying to write a lambda expression.

Let's first get some background on what are **functional interfaces** and **lambda expressions** in Java; that will help you to get an idea why this error is coming.

**Functional Interface in Java**

A functional interface is an interface with **only one abstract method**. A functional interface is also known as SAM type where SAM stands for (Single Abstract Method).

* Refer this post to know more about functional interfaces - [Functional interfaces & lambda expression](http://netjs.blogspot.com/2015/06/functional-interfaces-and-lambda-expression-in-java-8.html)

**Lambda expression in Java**

Lambda expression is an instance of the functional interface and **provides implementation of the abstract method defined by the functional interface**. Thus the functional interface specifies its **target type**.

* Refer this post to know more about lambda expression - [Overview of lambda expressions](http://netjs.blogspot.com/2015/06/lambda-expression-in-java-8-overview.html)

As you can see Functional interface is an interface with only one abstract method and lambda expression provides implementation of the abstract method defined by the functional interface.

So *this error may come when you have a functional interface which has more than one abstract methods*. Let's see it with an **example**. Here I am trying to write a lambda block which counts the number of words in a string.

interface IMyFunc {

int func(String n);

int func1(String n1, String n2);

}

public class FuntionalIntError {

public static void main(String[] args) {

String inStr = "Lambdas are a new addition to Java";

// lamda block assigned to a functional interface reference

IMyFunc myFunc = (str) -> {

int c = 0;

char ch[]= new char[str.length()];

for(int i = 0; i < str.length(); i++){

ch[i] = str.charAt(i);

if(((i > 0) && (ch[i] != ' ') && (ch[i-1] == ' ')) ||

((ch[0] != ' ') && (i == 0)))

c++;

}

return c;

};

// calling the func method of that functional interface

System.out.println("The word count is " + myFunc.func(inStr));

}

}

This code will give me the same error "**The target type of this expression must be a functional interface**" because of the fact that functional interface **IMyFunc** has **two abstract methods** so lambda expression won't know which of these two methods it is implementing and what is its target type.

**How to resolve this error**

By now you may have guessed that resolving this error "*the target type of this expression must be a functional interface*" means functional interface should have only one abstract method. So we need to delete the second method.

interface IMyFunc {

int func(String n);

//int func1(String n1, String n2);

}

In order to *make sure that the functional interface you write has one and only one abstract method*, you can use **@FunctionalInterface** annotation with your functional interface.

@FunctionalInterface

interface IMyFunc {

int func(String n);

}

* Refer this post to know more about functional interface annotaton - [Functional interface annotation in Java 8](http://netjs.blogspot.com/2015/06/functional-interface-annotation-java-8.html)

With this annotation @FunctionalInterface in place, any attempt to add another abstract method to this functional interface will result in compiler error.

That's all for this topic **How to Fix The Target Type of This Expression Must be a Functional Interface Error**. If you have any doubt or any suggestions to make please drop a comment. Thanks!

# Reference:

<https://www.journaldev.com/2389/java-8-features-with-examples>

<https://www.javatpoint.com/java-8-features>

<https://dzone.com/articles/java-lambda-expressions-basics>

<http://www.baeldung.com/java-8-new-features>

<https://github.com/eugenp/tutorials/tree/master/core-java-8>