Method Referencing Java (8)

Introduction

In Java, we can use references to objects, either by creating new objects:

**List list = new ArrayList();**

**store(new ArrayList());**

Or by using existing objects:

**List list2 = list;**

**isFull(list2);**

**But what about a reference to a method?**

If we only use a method of an object in another method, we still have to pass the full object as an argument. Wouldn't it be more practical to just pass the method as an argument? For example:

**isFull(list.size);**

**In Java 8, thanks to lambda expressions, we can do something like this. We can use methods as if they were objects, or primitive values.**

**Java 8 Method Reference**

A method reference is the shorthand syntax for a lambda expression that executes just ONE method. Here's the general syntax of a method reference:

**Object :: methodName**

We know that we can use lambda expressions instead of using an anonymous class. But sometimes, the lambda expression is really just a call to some method, for example:

**Consumer<String> c = s -> System.out.println(s);**

To make the code clearer, you can turn that lambda expression into a method reference:

**Consumer<String> c = System.out::println;**

In a method reference, you place the object (or class) that contains the method before the :: operator and the name of the method after it without arguments.

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

Java provides a new feature called method reference in Java 8. Method reference is used to refer method of functional interface. It is compact and easy form of lambda expression. Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference. In this tutorial, we are explaining method reference concept in detail.

**Four types of method references**

1. Method reference to an instance method of an object – object::instanceMethod

2. Method reference to a static method of a class – Class::staticMethod

3. Method reference to an instance method of an arbitrary object of a particular type – Class::instanceMethod

4. Method reference to a constructor – Class::new

Method reference is a shorthand notation of a lambda expression to call a method. For example:

If your lambda expression is like this:

* **str -> System.out.println(str)**

then you can replace it with a method reference like this:

* **System.out::println**

The :: operator is used in method reference to separate the class or object from the method name(we will learn this with the help of examples).

**Example-**

**1. Method reference to an instance method of an object**

@FunctionalInterface

interface MyInterface{

void display();

}

public class Example {

public void myMethod(){

System.out.println("Instance Method");

}

public static void main(String[] args) {

Example obj = new Example();

// Method reference using the object of the class

MyInterface ref = obj::myMethod;

// Calling the method of functional interface

ref.display();

}

}

Output:

Instance Method

## 2. Method reference to a static method of a class

import java.util.function.BiFunction;

class Multiplication{

public static int multiply(int a, int b){

return a\*b;

}

}

public class Example {

public static void main(String[] args) {

BiFunction<Integer, Integer, Integer> product = Multiplication::multiply;

int pr = product.apply(11, 5);

System.out.println("Product of given number is: "+pr);

}

}

Output:

Product of given number is: 55

## Method reference to an instance method of an arbitrary object of a particular type

import java.util.Arrays;

public class Example {

public static void main(String[] args) {

String[] stringArray = { "Steve", "Rick", "Aditya", "Negan", "Lucy", "Sansa", "Jon"};

/\* Method reference to an instance method of an arbitrary

\* object of a particular type

\*/

Arrays.sort(stringArray, String::compareToIgnoreCase);

for(String str: stringArray){

System.out.println(str);

}

}

}

Output:

Aditya

Jon

Lucy

Negan

Rick

Sansa

Steve

## Method reference to a constructor

@FunctionalInterface

interface MyInterface{

Hello display(String say);

}

class Hello{

public Hello(String say){

System.out.print(say);

}

}

public class Example {

public static void main(String[] args) {

//Method reference to a constructor

MyInterface ref = Hello::new;

ref.display("Hello World!");

}

}

Output:

Hello World!

JavaTpoint examples-

## 1) Reference to a Static Method

You can refer to static method defined in the class. Following is the syntax and example which describe the process of referring static method in Java.

Syntax

1. ContainingClass::staticMethodName

### **Example 1**

In the following example, we have defined a functional interface and referring a static method to it's functional method say().

1. **interface** Sayable{
2. **void** say();
3. }
4. **public** **class** MethodReference {
5. **public** **static** **void** saySomething(){
6. System.out.println("Hello, this is static method.");
7. }
8. **public** **static** **void** main(String[] args) {
9. // Referring static method
10. Sayable sayable = MethodReference::saySomething;
11. // Calling interface method
12. sayable.say();
13. }
14. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=MethodReference)

Output:

Hello, this is static method.

### **Example 2**

In the following example, we are using predefined functional interface Runnable to refer static method.

1. **public** **class** MethodReference2 {
2. **public** **static** **void** ThreadStatus(){
3. System.out.println("Thread is running...");
4. }
5. **public** **static** **void** main(String[] args) {
6. Thread t2=**new** Thread(MethodReference2::ThreadStatus);
7. t2.start();
8. }
9. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=MethodReference2)

Output:

Thread is running...

### **Example 3**

You can also use predefined functional interface to refer methods. In the following example, we are using BiFunction interface and using it's apply() method.

1. **import** java.util.function.BiFunction;
2. **class** Arithmetic{
3. **public** **static** **int** add(**int** a, **int** b){
4. **return** a+b;
5. }
6. }
7. **public** **class** MethodReference3 {
8. **public** **static** **void** main(String[] args) {
9. BiFunction<Integer, Integer, Integer>adder = Arithmetic::add;
10. **int** result = adder.apply(10, 20);
11. System.out.println(result);
12. }
13. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=MethodReference3)

Output:

30

### **Example 4**

You can also override static methods by referring methods. In the following example, we have defined and overloaded three add methods.

1. **import** java.util.function.BiFunction;
2. **class** Arithmetic{
3. **public** **static** **int** add(**int** a, **int** b){
4. **return** a+b;
5. }
6. **public** **static** **float** add(**int** a, **float** b){
7. **return** a+b;
8. }
9. **public** **static** **float** add(**float** a, **float** b){
10. **return** a+b;
11. }
12. }
13. **public** **class** MethodReference4 {
14. **public** **static** **void** main(String[] args) {
15. BiFunction<Integer, Integer, Integer>adder1 = Arithmetic::add;
16. BiFunction<Integer, Float, Float>adder2 = Arithmetic::add;
17. BiFunction<Float, Float, Float>adder3 = Arithmetic::add;
18. **int** result1 = adder1.apply(10, 20);
19. **float** result2 = adder2.apply(10, 20.0f);
20. **float** result3 = adder3.apply(10.0f, 20.0f);
21. System.out.println(result1);
22. System.out.println(result2);
23. System.out.println(result3);
24. }
25. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=MethodReference4)

Output:

30

30.0

30.0

## 2) Reference to an Instance Method

like static methods, you can refer instance methods also. In the following example, we are describing the process of referring the instance method.

Syntax

1. containingObject::instanceMethodName

### **Example 1**

In the following example, we are referring non-static methods. You can refer methods by class object and anonymous object.

1. **interface** Sayable{
2. **void** say();
3. }
4. **public** **class** InstanceMethodReference {
5. **public** **void** saySomething(){
6. System.out.println("Hello, this is non-static method.");
7. }
8. **public** **static** **void** main(String[] args) {
9. InstanceMethodReference methodReference = **new** InstanceMethodReference(); // Creating object
10. // Referring non-static method using reference
11. Sayable sayable = methodReference::saySomething;
12. // Calling interface method
13. sayable.say();
14. // Referring non-static method using anonymous object
15. Sayable sayable2 = **new** InstanceMethodReference()::saySomething; // You can use anonymous object also
16. // Calling interface method
17. sayable2.say();
18. }
19. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=InstanceMethodReference)

Output:

Hello, this is non-static method.

Hello, this is non-static method.

### **Example 2**

In the following example, we are referring instance (non-static) method. Runnable interface contains only one abstract method. So, we can use it as functional interface.

1. **public** **class** InstanceMethodReference2 {
2. **public** **void** printnMsg(){
3. System.out.println("Hello, this is instance method");
4. }
5. **public** **static** **void** main(String[] args) {
6. Thread t2=**new** Thread(**new** InstanceMethodReference2()::printnMsg);
7. t2.start();
8. }
9. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=InstanceMethodReference2)

Output:

Hello, this is instance method

### **Example 3**

In the following example, we are using BiFunction interface. It is a predefined interface and contains a functional method apply(). Here, we are referring add method to apply method.

1. **import** java.util.function.BiFunction;
2. **class** Arithmetic{
3. **public** **int** add(**int** a, **int** b){
4. **return** a+b;
5. }
6. }
7. **public** **class** InstanceMethodReference3 {
8. **public** **static** **void** main(String[] args) {
9. BiFunction<Integer, Integer, Integer>adder = **new** Arithmetic()::add;
10. **int** result = adder.apply(10, 20);
11. System.out.println(result);
12. }
13. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=InstanceMethodReference3)

Output:

30

## 3) Reference to a Constructor

You can refer a constructor by using the new keyword. Here, we are referring constructor with the help of functional interface.

Syntax

1. ClassName::**new**

### **Example**

1. **interface** Messageable{
2. Message getMessage(String msg);
3. }
4. **class** Message{
5. Message(String msg){
6. System.out.print(msg);
7. }
8. }
9. **public** **class** ConstructorReference {
10. **public** **static** **void** main(String[] args) {
11. Messageable hello = Message::**new**;
12. hello.getMessage("Hello");
13. }
14. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=ConstructorReference)

Output:

Hello