Java Method References

Last Updated : 15 Jul, 2025

In Java, a method is a collection of statements that perform some specific task and return the result to the caller. A **method reference** is the shorthand syntax for a **lambda expression** that contains just one method call. In general, one does not have to pass arguments to method references.

Why Use Method References?

Method references are used for the following reasons, which are listed below:

- Method references enhance readability, which makes the code easier to understand.
- It supports a functional programming style that works well with streams and collections.
- Reusability increases because we can directly use the existing methods.

Note: Functional Interfaces in Java and Lambda Function are prerequisites required in order to grasp a grip over Method References in Java.

Example:

```java
// Using Method Reference

import java.util.Arrays;

public class Geeks

{

    // Method

    public static void print(String s) {

        System.out.println(s);

    }

```

Output

```
Geek1
Geek2
Geek3
```

Explanation: In the above example, we are using method reference to print items. The print method is a static method which is used to print the names. In the main method we created an array of names and printing each one by calling the print method directly.

Key Benefits of Method References

The key benefits of method references are listed below:

- **Improved Readability:** Method references simplify the code by removing boilerplate syntax.
- **Reusability:** Existing methods can be directly reused, enhancing modularity.
- **Functional Programming Support:** They work seamlessly with functional interfaces and lambdas.

Function as a Variable

In Java 8 we can use the method as if they were objects or primitive values, and we can treat them as a variable.

```
// This square function is a variable getSquare.
Function<Integer, Integer> getSquare = i -> i * i ;

// Pass function as an argument to another function easily
SomeFunction(a, b, getSquare) ;
```

Sometimes, a lambda expression only calls an existing method. In those cases, it looks clear to refer to the existing method by name. The method references can do this, they are compact, easy-to-read as compared to lambda expressions.

Generic Syntax for Method References

Aspect	Syntax
Refer to a method in an object	Object :: methodName
Print all elements in a list	list.forEach(s -> System.out.println(s));
Shorthand to print all elements in a list	list.forEach(System.out::println);

Types of Method References

There are four type method references that are as follows:

1. Static Method Reference
2. Instance Method Reference of a particular object
3. Instance Method Reference of an arbitrary object of a particular type
4. Constructor Reference

To look into all these types we will consider a common example of sorting with a comparator which is as follows:

1. Reference to a Static Method

A static method lets us use a method from a class without writing extra code. It is a shorter way to write a lambda that just calls that static method.

Syntax:

```
 // Lambda expression
(args) -> Class.staticMethod(args);
// Method reference
Class::staticMethod;
```

Example:

```
// Reference to a static method

import java.io.*;
import java.util.*;


class Person

{

    private String name;

    private Integer age;


    // Constructor

    public Person(String name, int age)

    {

        // This keyword refers to current instance itself

        this.name = name;

        this.age = age;

    }


    // Getter-setters

    public Integer getAge() { return age; }

    public String getName() { return name; }

}


// Driver class

public class Geeks

{
    // Static method to compare with name

    public static int compareByName(Person a, Person b) {

        return a.getName().compareTo(b.getName());

    }



    // Static method to compare with age

    public static int compareByAge(Person a, Person b) {

        return a.getAge().compareTo(b.getAge());

    }



    // Main driver method

    public static void main(String[] args) {



        // Creating an empty ArrayList of user-defined type

        // List of person
```

Output

```
Sort by Name :
Poonam
Sachin
Vicky

Sort by Age :
Sachin
Vicky
Poonam
```

Explanation: This example shows how to use static method references to sort items. We have a person class with attributes like name and age and there are two methods to compare people by name and by age. In the main method we created a list of people and sorting them by name and then sort them by age and then printing the name again.

2. Reference to an Instance Method of a Particular Object

This type of method means using a method from a certain object which we already have. We do not need to write another function to call that particular method we can just simply refer to it directly.

Syntax:

```
// Lambda expression
(args) -> obj.instanceMethod(args);
// Method reference
obj::instanceMethod;
```

Example:

```java
// Reference to an Instance Method of

// a Particular Object

import java.io.*;

import java.util.*;



class Person {



    // Attributes of a person

    private String name;

    private Integer age;
    // Constructor

    public Person(String name, int age)

    {

        // This keyword refers to current object itself

        this.name = name;

        this.age = age;

    }



    // Getter-setter methods

    public Integer getAge() { return age; }

    public String getName() { return name; }

}



// Helper class

// Comparator class

class ComparisonProvider

{

    // To compare with name

    public int compareByName(Person a, Person b) {

        return a.getName().compareTo(b.getName());

    }
    // To compare with age
    public int compareByAge(Person a, Person b) {
        return a.getAge().compareTo(b.getAge());
    }
}


// Main class
public class Geeks
{
    public static void main(String[] args)
    {
        // Creating an empty ArrayList of user-defined type
        // List of person
```

Output

```
Sort by Name :
Poonam
Sachin
Vicky

Sort by Age :
Sachin
Vicky
Poonam
```

Explanation: This example show how to use an instance method reference to sort a list of people. We have created a Person class with name and age and we also created a ComparisonProvider class, it has methods to compare people by name or age. In the main method we created a list of people and we are using the

ComparisonProvider instance to sort and print the names first by name, then by age.

3. Reference to an Instance Method of an Arbitrary Object of a Particular Type

It means calling a method on any object that belongs to a certain group or class, not just one specific object. It helps us write less code when we want to do the same thing for many objects.

Syntax:

```
// Lambda expression
(obj, args) -> obj.instanceMethod(args);
// Method reference
ObjectType::instanceMethod;
```

Example:

```
// Reference to an Instance Method of an

// Arbitrary Object of a Particular Type

import java.io.*;

import java.util.*;



public class Geeks

{

    public static void main(String[] args)

    {

        // Creating an empty ArrayList of user defined type

        // List of person

        List<String> personList = new ArrayList<>();
```

Output

```
Poonam
Sachin
Vicky
```

Explanation: This example show how to use a method reference to sort a list of names. We created a list of names and then sorting them ignoring uppercase or lowercase with the help of compareToIgnoreCase method of the String class and then we are printing the sorted names

4. Constructor Method Reference

It lets us quickly create a new object without writing extra code. It is a shortcut to call the class new method.

Syntax:

```
// Lambda expression
(args) -> new ClassName(args);
// Method reference
ClassName::new;
```

Example:

```
// Java Program to Illustrate How We can Use

// constructor method reference



// Importing required classes

import java.io.*;
```

```java
import java.nio.charset.Charset;
import java.util.*;
import java.util.function.*;


// Object need to be sorted
class Person {
    private String name;
    private Integer age;


    // Constructor
    public Person()
    {
        Random ran = new Random();


        // Assigning a random value
        // to name
        this.name
            = ran
                  .ints(97, 122 + 1)
                  .limit(7)
                  .collect(StringBuilder::new,
                           StringBuilder::appendCodePoint,
                           StringBuilder::append)
```

```
                .toString();
    }


    public Integer getAge()
    {
        return age;
    }
    public String getName()
    {
        return name;
```

Output

```
ilvxzcv
vdixqbs
lmcfzpj
dxnyqej
zeqejcn
```

Explanation: This example show how to use a constructor method reference to create objects. We have created a Person class and it gives each person a random name. The getObjectList method creates a list of objects by using a supplier, which means it uses the Person constructor to make new Person objects. In the main method we created a list of people and then we are printing their random names.

Common Use Cases

There are some common cases where we use Method References in Java as mentioned below:

- **Iterating over collections:** Simplifying operations like printing or processing elements.

- **Stream API operations:** Enhancing readability in filtering, mapping, and reducing operations.
- **Custom utilities:** Using predefined methods for frequently used tasks like sorting and comparisons.