Method References in Java with examples

Difficulty Level : Medium Last Updated : 16 Nov, 2021

Functional Interfaces in Java and Lambda Function are prerequisites required in order to grasp grip over method references in Java. As we all know that a method is a collection of statements that perform some specific task and return the result to the caller. A method can perform some specific task without returning anything. Methods allow us to reuse the code without retyping the code. In this article, we will see how to use methods as value.

In Java 8 we can use the method as if they were objects or primitive values, and we can treat them as a variable. The example shows the function as a variable in java:

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


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

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. A method reference is the shorthand syntax for a lambda expression that contains just one method call. Here's the general syntax of a

Generic syntax: Method reference

A. To refer to a method in an object

```
Object :: methodName
```

B. To print all elements in a list

Following is an illustration of a lambda expression that just calls a single method in its entire execution:

```
list.forEach(s -> System.out.println(s));
```

C. Shorthand to print all elements in a list

To make the code clear and compact, In the above example, one can turn lambda expression into a method reference:

```
list.forEach(System.out::println);
```

The method references can only be used to replace a single method of the lambda expression. A code is more clear and short if one uses a lambda expression rather than using an anonymous class and one can use method reference rather than using a single function lambda expression to achieve the same. In general, one doesn't have to pass arguments to method references.

The following example is about performing some operations on elements in the list and adding them. The operation to be performed on elements is a function argument and the caller can pass accordingly.

Illustration:

```
public int transformAndAdd(List<Integer> l,
                           Function<Integer, Integer> ops) {
    int result = 0;
    for (Integer s : l)
        result += f.apply(s);

    return results;
}

// Operations utility class
class OpsUtil {

    // Method 1
    // To half the variable
    public static Integer doHalf(Integer x) {
        return x / 2;
    }

    // Method 2
    // Square up the integer number
    public static Integer doSquare(Integer x) {

        return x * x;
    }
```

```
    ... many more operations...
}
```

Following are the ways to call the above method as depicted below as follows:

```
List<Integer> list = new ArrayList<>();
// Add some element to list
...
// Using an anonymous class
transformAndAdd(list, new Function<Integer, Integer>() {
    public Integer apply(Integer i) {
        // The method
        return OpsUtil.doHalf(i);
    }
});

// Using a lambda expression
tranformAndAdd(list, i -> OpsUtil.doHalf(i));

// Using a method reference
tranformAndAdd(list, OpsUtil::doHalf);
```

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:

Type 1: Reference to a static method

If a Lambda expression is like:

// If a lambda expression just call a static method of a class

(args) -> Class.staticMethod(args)

Then method reference is like:

// Shorthand if a lambda expression just call a static method of a class

Class::staticMethod

Example:

```
// Java Program to Illustrate How One can use
// Static method reference
// To Sort with Custom Comparator

// Importing required classes
import java.io.*;
import java.util.*;

// Class 1
// Helper class
// Object need to be sorted
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; }
}

// Class 2
```

```
// Main class
public class GFG {

    // Method 1
    // Static method to compare with name
    public static int compareByName(Person a, Person b)
    {
        return a.getName().compareTo(b.getName());
    }

    // Method 2
    // Static method to compare with age
    public static int compareByAge(Person a, Person b)
    {
        return a.getAge().compareTo(b.getAge());
    }

    // Method 3
    // Main driver method
    public static void main(String[] args)
    {

        // Creating an empty ArrayList of user-defined type
        // List of person
        List<Person> personList = new ArrayList<>();

        // Adding elements to above List
        // using add() method
        personList.add(new Person("vicky", 24));
        personList.add(new Person("poonam", 25));
        personList.add(new Person("sachin", 19));

        // Using static method reference to
        // sort array by name
        Collections.sort(personList, GFG::compareByName);

        // Display message only
        System.out.println("Sort by name :");

        // Using streams over above object of Person type
        personList.stream()
            .map(x -> x.getName())
            .forEach(System.out::println);

        // Now using static method reference
        // to sort array by age
        Collections.sort(personList, GFG::compareByAge);

        // Display message only
        System.out.println("Sort by age :");

        // Using streams over above object of Person type
        personList.stream()
            .map(x -> x.getName())
            .forEach(System.out::println);
    }
```

```
}
```

Output:

```
Sort by name :
poonam
sachin
vicky
Sort by age :
sachin
vicky
poonam
```

Type 2: Reference to an instance method of a particular object

If a Lambda expression is like:

```
// If a lambda expression just call a default method of an object
(args) -> obj.instanceMethod(args)
```

Then method reference is like:

```
// Shorthand if a lambda expression just call a default method of an object
obj::instanceMethod
```

Example:

```
// Java Program to Illustrate How One can use
```

```
// Static method reference
// To Sort with Custom Comparator
// But using object method reference

// Importing required classes
import java.io.*;
import java.util.*;

// Class 1
// Helper class
// Object need to be sorted
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; }
}

// Class 2
// Helper class
// Comparator class
class ComparisonProvider {

    // Method 1
    // To compare with name
    public int compareByName(Person a, Person b)
    {
        return a.getName().compareTo(b.getName());
    }

    // Method 2
    // To compare with age
    public int compareByAge(Person a, Person b)
    {
        return a.getAge().compareTo(b.getAge());
    }
}

// Class 3
// Main class
public class GFG {

    // Main driver method
```

```
    public static void main(String[] args)
    {
        // Creating an empty ArrayList of user-defined type
        // List of person
        List<Person> personList = new ArrayList<>();

        // Adding elements to above object
        // using add() method
        personList.add(new Person("vicky", 24));
        personList.add(new Person("poonam", 25));
        personList.add(new Person("sachin", 19));

        // A comparator class with multiple
        // comaparator methods
        ComparisonProvider comparator
            = new ComparisonProvider();

        // Now using instance method reference
        // to sort array by name
        Collections.sort(personList,
                         comparator::compareByName);

        // Display message only
        System.out.println("Sort by name :");

        // Using streams
        personList.stream()
            .map(x -> x.getName())
            .forEach(System.out::println);

        // Using instance method reference
        // to sort array by age
        Collections.sort(personList,
                         comparator::compareByAge);

        // Display message only
        System.out.println("Sort by age :");

        personList.stream()
            .map(x -> x.getName())
            .forEach(System.out::println);
    }
}
```

Output:

```
Sort by name :
poonam
sachin
vicky
```

```
Sort by age :
sachin
vicky
poonam
```

Type 3: Reference to an instance method of an arbitrary object of a particular type

If a Lambda expression is like:

// If a lambda expression just call an instance method of a ObjectType

(obj, args) -> obj.instanceMethod(args)

Then method reference is like:

// Shorthand if a lambda expression just call an instance method of a ObjectType

ObjectType::instanceMethod

Example:

```
// Java Program to Illustrate how One can use
// Instance type method reference to
// sort with custom comparator

// Importing required classes
import java.io.*;
import java.util.*;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating an empty ArrayList of user defined type
```

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

        // Adding elements to above object of List
        // using add() method
        personList.add("vicky");
        personList.add("poonam");
        personList.add("sachin");

        // Method reference to String type
        Collections.sort(personList,
                         String::compareToIgnoreCase);

        // Printing the elements(names) on console
        personList.forEach(System.out::println);
    }
}
```

Output:

```
poonam
sachin
vicky
```

Type 4: Constructor method reference

If a Lambda expression is like:

// If a lambda expression just create an object
(args) -> new ClassName(args)

Then method reference is like:

// Shorthand if a lambda expression just create an object
ClassName::new

Example:

```
// Java Program to Illustrate How We can Use
// constructor method reference

// Importing required classes
import java.io.*;
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;
    }
}

public class GFG {

    // Get List of objects of given
    // length and Supplier
    public static <T> List<T>
    getObjectList(int length,
                  Supplier<T> objectSupply)
    {
```

```
            List<T> list = new ArrayList<>();

            for (int i = 0; i < length; i++)
                list.add(objectSupply.get());
            return list;
        }

        public static void main(String[] args)
        {

            // Get 10 person by supplying
            // person supplier, Supplier is
            // created by person constructor
            // reference
            List<Person> personList
                = getObjectList(5, Person::new);

            // Print names of personList
            personList.stream()
                .map(x -> x.getName())
                .forEach(System.out::println);
        }
    }
```

Output:

```
vzskgmu
iupltfx
kocsipj
lyvhxsp
hbdphyv
```

Conclusion: *As mentioned above, if a lambda expression only calls an existing method then using method reference can make code more readable and clear. There are many more thing s we can do with Java8 Lambda and Method References while using Java streams.*