Method References

In this lesson, you will learn about method references and explore its types.

We'll cover the following

- Four kinds of method references
  - 1. Static methods
  - 2. Instance method of a particular object
  - 3. Instance method of an arbitrary object
  - 4. Constructor references

Method references, as the name suggests are the references to a method. They are similar to object references. As we can have reference to an object, we can have reference to a method as well.

Similar to an object reference, we can now pass behavior as parameters. But, you might be wondering what the difference between a method reference and lambda expressions is. There is no difference. Method references are shortened versions of lambda expressions that call a specific method.

Let's say you have a `Consumer` as defined below:

```
Consumer<String> consumer = s -> System.out.println(s);
```

This can be written as:

```
Consumer<String> consumer = System.out::println;
```

Let's see one more example. Consider we have a Function<T,R> functional interface as defined below:

```
Function<Person, Integer>  function = p  -> p.getAge();
```

This can be written as:

```
Function<Person, Integer>  function = Person::getAge;
```

Four kinds of method references

There are four kinds of method references.

1. Static methods

The syntax to use static methods as method reference is `ClassName::MethodName`.

In the below example, we have a method `getLength()` which returns the length of the `String`. We have written a lambda expression using a method reference to fetch the length of the string.

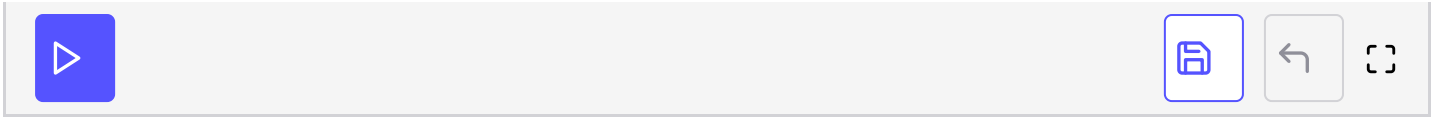
```
import java.util.ArrayList;
import java.util.List;

public class StreamDemo {

    public static int getLength(String str){
        return str.length();
    }

    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("done");
        list.add("word");
        list.add("practice");
        list.add("fake");

        // Code without using method reference.
        list.stream()
                .mapToInt(str -> StreamDemo.getLength(str))
                .forEach(System.out::println);

        // Code with method reference.
        list.stream()
                .mapToInt(StreamDemo::getLength)
                .forEach(System.out::println);

    }
}
```

2. Instance method of a particular object

The syntax to use the instance method as a method reference is `ReferenceVariable::MethodName`

We will look at the same example as above, but, this time, the `getLength()` method is not static.

```
import java.util.ArrayList;
import java.util.List;

public class StreamDemo {

    public int getLength(String str) {
        return str.length();
    }

    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("done");
        list.add("word");
        list.add("practice");
        list.add("fake");

        StreamDemo demo = new StreamDemo();
        // Code without instance method reference.
        list.stream()
                .mapToInt(str -> demo.getLength(str))
                .forEach(System.out::println);

        // Code with instance method reference.
        list.stream()
                .mapToInt(demo::getLength)
                .forEach(System.out::println);

    }
}
```

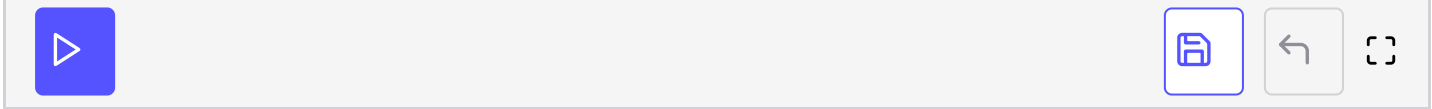


3. Instance method of an arbitrary object

This type of method reference does not require the object of the referenced class. We can directly use the class name in the method reference.

```
import java.util.ArrayList;
import java.util.List;

public class StreamDemo {

    public int getLength(String str) {
        return str.length();
    }

    public static void main(String[] args) {
        List<Employee> list = new ArrayList<>();
        list.add(new Employee("Dave", 23, 20000));
        list.add(new Employee("Joe", 18, 40000));
        list.add(new Employee("Ryan", 54, 100000));
        list.add(new Employee("Iyan", 5, 34000));
        list.add(new Employee("Ray", 63, 54000));
```

```
        // Code without using method reference
        int totalSalary1 = list.stream()
                .mapToInt(emp -> emp.getSalary())

                .sum();

        // Code with method reference
        int totalSalary = list.stream()
                .mapToInt(Employee::getSalary)
                .sum();

        System.out.println("The total salary is " + totalSalary);
    }
}


class Employee {
    String name;
    int age;
    int salary;

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
                ", salary=" + salary +
                '}';
    }
}
```



4. Constructor references

We can refer to a constructor in the same way we reference a static method. The only difference is that we need to use a new keyword.

```
import java.util.ArrayList;
```

```
import java.util.List;
import java.util.stream.Collectors;

public class StreamDemo {

    public int getLength(String str) {
        return str.length();
    }

    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Dave");
        list.add("Joe");
        list.add("Ryan");
        list.add("Iyan");
        list.add("Ray");

        // Code without constructor reference
        list.stream()
                .map(name -> new Employee(name))
                .forEach(System.out::println);

        // Code with constructor reference
        list.stream()
                .map(Employee::new)
                .forEach(System.out::println);
    }
}


class Employee {
    String name;
    int age;
    int salary;

    Employee(String name) {
        this.name = name;
    }

    Employee(String name, int age, int salary) {
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public int getSalary() {
        return salary;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "name='" + name + '\'' +
                ", age=" + age +
```

```
                ", salary=" + salary +
                '}';
    }

}
```




In the next lesson, we will explore the `Optional` class in Java 8.