Method References in Java8

3 min read · Mar 17, 2023

Srikanth Dannarapu Follow

Listen Share

Method references provide a shorthand notation for lambda expressions that only call a single method. Method references can be used to simplify the code and make it more readable. In this answer, we'll explore the different types of method references in Java 8 with examples.

There are four types of method references in Java 8:

1. Reference to a static method
2. Reference to an instance method of an object
3. Reference to an instance method of an arbitrary object of a particular type
4. Reference to a constructor

Let's explore each of these types in detail:

1. **Reference to a static method:** A reference to a static method can be created using the syntax `ClassName::methodName`. For example:

```
public class MathUtils {
    public static int multiply(int a, int b) {
        return a * b;
    }
}

public class Main {
    public static void main(String[] args) {
        IntBinaryOperator operator = MathUtils::multiply;
        int result = operator.applyAsInt(4, 5);
        System.out.println(result);
```

```
    }
}
```

In this example, we created a reference to the static method `multiply` in the `MathUtils` class using the `MathUtils::multiply` syntax. We then assigned this reference to an `IntBinaryOperator` variable, which is a functional interface that takes two integer parameters and returns an integer value. We called the `applyAsInt` method on this variable with the arguments 4 and 5 to get the result, which is 20.

2. Reference to an instance method of an object: A reference to an instance method of an object can be created using the syntax `object::methodName`. For example:

```
public class Person {
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}

public class Main {
    public static void main(String[] args) {
        Person person = new Person("John");
        Supplier<String> supplier = person::getName;
        String name = supplier.get();
        System.out.println(name);
```

```
    }
}
```

In this example, we created a reference to the `getName` method of the `Person` class instance `person` using the `person::getName` syntax. We then assigned this reference to a `Supplier` variable, which is a functional interface that takes no arguments and returns a value. We called the `get` method on this variable to get the name of the person, which is "John".

3. Reference to an instance method of an arbitrary object of a particular type: A reference to an instance method of an arbitrary object of a particular type can be created using the syntax `ClassName::methodName`. For example:

```
public class StringUtils {
    public boolean startsWithUppercase(String s) {
        return Character.isUpperCase(s.charAt(0));
    }
}

public class Main {
    public static void main(String[] args) {
        Predicate<String> predicate = StringUtils::startsWithUppercase;
        boolean result = predicate.test("Hello");
        System.out.println(result);
    }
}
```

In this example, we created a reference to the `startsWithUppercase` method of the `StringUtils` class using the `StringUtils::startsWithUppercase` syntax. We then assigned this reference to a `Predicate` variable, which is a functional interface that takes one argument of a specified type and returns a boolean value. We called the `test` method on this variable with the argument "Hello" to get the result, which is `true`.

4. Reference to a constructor: A reference to a constructor can be created using the syntax `ClassName::new`. For example:

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter and setter methods
}

public class Main {
    public static void main(String[] args) {
        BiFunction<String, Integer, Person> personConstructor = Person::new;
        Person person = personConstructor.apply("John", 25);
        System.out.println(person.getName());
        System.out.println(person.getAge());
    }
}
```

In this example, we created a reference to the `Person` constructor using the `Person::new` syntax. We then assigned this reference to a `BiFunction` variable, which is a functional interface that takes two arguments of specified types and returns a value of a specified type. We called the `apply` method on this variable with the arguments "John" and 25 to create a new `Person` instance. We then called the `getName` and `getAge` methods on this instance to get the name and age of the person, which are "John" and 25, respectively.

Note that the `BiFunction` interface is used in this example because the `Person` constructor takes two arguments. If the constructor took a different number of arguments, a different functional interface would be used. For example, a `Function` interface would be used if the constructor took one argument.

. . .

Thanks, before you go:

- 👏 Please clap for the story and follow the author 👉

- Please share your questions or insights in the comments section below. Let's help each other and become better Java developers.
- Let's connect on LinkedIn

Java8 Method References Java Lambda