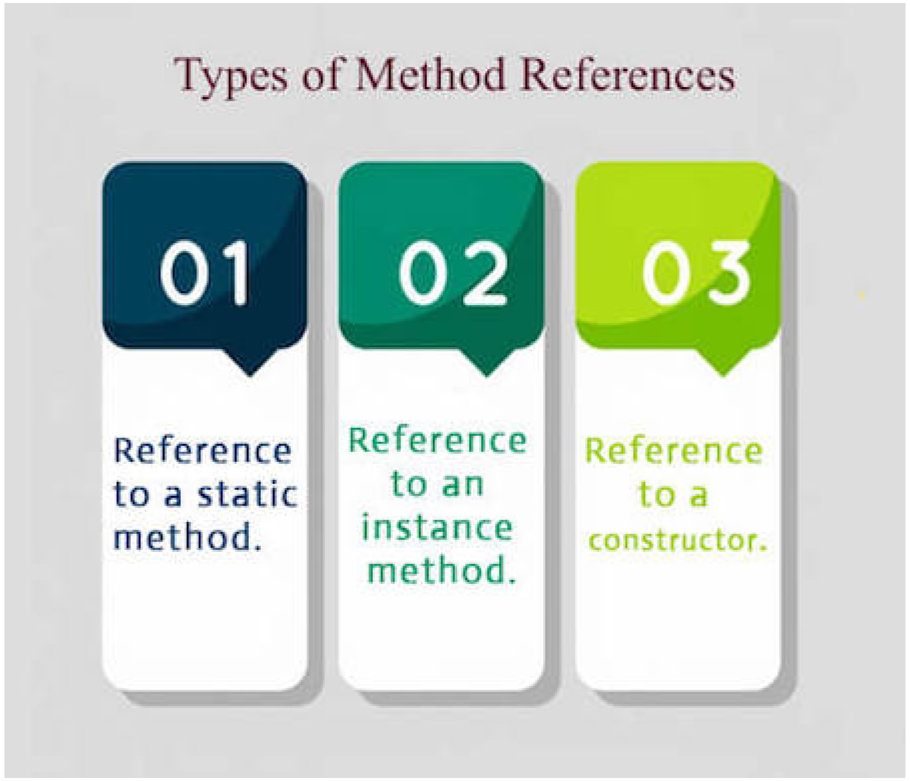
Java Method References

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.

Types of Method References

There are following types of method references in java:

1. Reference to a static method.
2. Reference to an instance method.
3. Reference to a constructor.


Types of Method References
01
Reference to a static method.
02
Reference to an instance method.
03
Reference to a constructor.

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

```
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().

```
interface Sayable{
    void say();
}
public class MethodReference {
    public static void saySomething(){
        System.out.println("Hello, this is static method.");
    }
    public static void main(String[] args) {
        // Referring static method
        Sayable sayable = MethodReference::saySomething;
        // Calling interface method
        sayable.say();
    }
}
```

Output:

```
Hello, this is static method.
```

Example 2

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

```
public class MethodReference2 {
    public static void ThreadStatus(){
        System.out.println("Thread is running...");
    }
    public static void main(String[] args) {
        Thread t2=new Thread(MethodReference2::ThreadStatus);
        t2.start();
    }
}
```

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.

```
import java.util.function.BiFunction;
class Arithmetic{
public static int add(int a, int b){
return a+b;
}
}
public class MethodReference3 {
public static void main(String[] args) {
BiFunction<Integer, Integer, Integer>adder = Arithmetic::add;
int result = adder.apply(10, 20);
System.out.println(result);
}
}
```

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.

```
import java.util.function.BiFunction;
class Arithmetic{
public static int add(int a, int b){
return a+b;
}
public static float add(int a, float b){
return a+b;
}
public static float add(float a, float b){
return a+b;
}
}
public class MethodReference4 {
```

```
public static void main(String[] args) {
BiFunction<Integer, Integer, Integer>adder1 = Arithmetic::add;
BiFunction<Integer, Float, Float>adder2 = Arithmetic::add;
BiFunction<Float, Float, Float>adder3 = Arithmetic::add;
int result1 = adder1.apply(10, 20);
float result2 = adder2.apply(10, 20.0f);
float result3 = adder3.apply(10.0f, 20.0f);
System.out.println(result1);
System.out.println(result2);
System.out.println(result3);
}
}
```

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

```
containingObject::instanceMethodName
```

Example 1

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

```
interface Sayable{
    void say();
}
public class InstanceMethodReference {
    public void saySomething(){
        System.out.println("Hello, this is non-static method.");
    }
    public static void main(String[] args) {
        InstanceMethodReference methodReference = new InstanceMethodReference(); // Creating object
        // Referring non-static method using reference
            Sayable sayable = methodReference::saySomething;
        // Calling interface method
            sayable.say();
            // Referring non-static method using anonymous object
            Sayable sayable2 = new InstanceMethodReference()::saySomething; // You can use anonymous object also
            // Calling interface method
            sayable2.say();
    }
}
```

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.

```
public class InstanceMethodReference2 {
    public void printnMsg(){
        System.out.println("Hello, this is instance method");
    }
    public static void main(String[] args) {
    Thread t2=new Thread(new InstanceMethodReference2()::printnMsg);
        t2.start();
    }
}
```

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.

```
import java.util.function.BiFunction;
class Arithmetic{
public int add(int a, int b){
return a+b;
}
}
public class InstanceMethodReference3 {
public static void main(String[] args) {
BiFunction<Integer, Integer, Integer>adder = new Arithmetic()::add;
int result = adder.apply(10, 20);
System.out.println(result);
}
}
```

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

```
ClassName::new
```

Example

```
interface Messageable{
    Message getMessage(String msg);
}
class Message{
    Message(String msg){
        System.out.print(msg);
    }
}
public class ConstructorReference {
    public static void main(String[] args) {
        Messageable hello = Message::new;
        hello.getMessage("Hello");
    }
}
```

Output:

```
Hello
```