메서드 참조

소유자	종수 김
태그	

메서드 참조가 필요한 이유

특정 상황에서 람다를 좀 더 편리하게 사용할 수 있는 메서드 참조(Method Reference)

```
package methodref;

import java.util.function.BinaryOperator;

public class MethodRefStartV1 {
    public static void main(String[] args) {
        BinaryOperator<Integer> add1 = (x, y) -> x + y;
        BinaryOperator<Integer> add2 = (x, y) -> x + y;

        Integer result1 = add1.apply(1, 2);
        System.out.println("result1 = " + result1);

        Integer result2 = add2.apply(1, 2);
        System.out.println("result2 = " + result2);
    }
}
```

- 동일한 기능을 하는 람다.
    - 코드가 중복되어 유지보수가 불편
    - 덧셈 로직이 변경된다면 많은 곳에 변경이 일어남.
- 동일한 부분을 메서드로 추출

```
package methodref;

import java.util.function.BinaryOperator;
```

```
public class MethodRefStartV2 {
    public static void main(String[] args) {
        BinaryOperator<Integer> add1 = (x, y) -> add(x,y);
        BinaryOperator<Integer> add2 = (x, y) -> add(x,y);

        Integer result1 = add1.apply(1, 2);
        System.out.println("result1 = " + result1);

        Integer result2 = add2.apply(1, 2);
        System.out.println("result2 = " + result2);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

- 코드 중복을 해결.
  - 덧셈 로직을 별도의 메서드로
  - 람다는 add() 메서드를 호출.
  - 로직이 한 곳으로 모여 유지보수가 쉬어짐.
- 람다를 작성할 때마다 (x,y) -> add(x,y)라는 코드가 들어가야함.
  - 매개변수를 전달하는 부분이 장황

```
package methodref;

import java.util.function.BinaryOperator;

public class MethodRefStartV3 {
    public static void main(String[] args) {
        BinaryOperator<Integer> add1 = MethodRefStartV3::add;
        BinaryOperator<Integer> add2 = MethodRefStartV3::add;

        Integer result1 = add1.apply(1, 2);
        System.out.println("result1 = " + result1);
```

```
        Integer result2 = add2.apply(1, 2);
        System.out.println("result2 = " + result2);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

- 메서드 참조 문법인, 클래스명::메서드명 을 사용하여 (x,y) → add(x,y) 람다를 더욱 간단하게 표현.
  MethodRefStartV3::add = (x,y) → add(x,y)

메서드 참조의 장점

1. 메서드 참조를 사용하면 코드가 더욱 간결해지고, 가독성 향상
2. 더 이상 매개변수를 명시적으로 작성할 필요가 없음.
    a. 컴파일러가 자동으로 매개변수를 매칭
3. 별도의 로직 분리와 함께 재사용성 역시 높아짐.

메서드 참조란 ?

‘이미 정의된 메서드를 그대로 참조하여 람다 표현식을 더 간결하게 작성하는 방법’

(x,y) → add(x,y)라는 람다는 사실상 매개변수 x,y를 그대로 add메서드에 전달하는 것이므로,
클래스명::메서드명 형태의 메서드 참조로 간단히 표현할 수 있음.

- 불필요한 매개변수 선언 없이 코드가 깔끔해지고, 가독성도 높아짐.
- 매개변수를 그대로 다른 메서드의 매개변수로 넘기는 경우 사용 가능.

정리

메서드 참조는 이미 정의된 메서드를 람다로 변환하여 더욱 간결하게 사용할 수 있도록 해주는 문법적 편의 기능.

람다 매개변수를 넘기거나, 람다식을 작성하는 것 대신 직관적으로 확인 가능.

- 람다를 작성할 때, 이미 정의된 메서드를 그대로 호출하는 경우. 메서드 참조를 사용하여 더욱 직관적이고 간결한 코드 작성 가능.

메서드 참조 - 1 시작

‘이미 정의된 메서드를 람다처럼 간결하게 표현’할 수 있게 해주는 문법

즉, 람다 내부에서 단순히 어떤 메서드(정적, 인스턴스, 생성자 등)를 호출만 하고있을 경우, 다음과 같은 형태로 메서드 참조 사용 가능

```
(x, y) → 클래스명.메서드명(x, y) // 기존 람다
클래스명::메서드명 // 메서드 참조
```

람다와 메서드 참조는 동등하게 동작.

메서드 참조의 4가지 유형

1. 정적 메서드 참조
2. 특정 객체의 인스턴스 메서드 참조
3. 생성자 참조
4. 임의 객체의 인스턴스 메서드 참조

1. 정적 메서드 참조

- 설명: 이름 그대로 정적(static) 메서드를 참조한다.
- 문법: `클래스명::메서드명`
- 예: `Math::max`, `Integer::parseInt` 등

2. 특정 객체의 인스턴스 메서드 참조

- 설명: 이름 그대로 특정 객체의 인스턴스 메서드를 참조한다.
- 문법: `객체명::인스턴스메서드명`
- 예: `person::introduce`, `person::getName` 등

3. 생성자 참조

- 설명: 이름 그대로 생성자를 참조한다.
- 문법: `클래스명::new`
- 예: `Person::new`

---

4. 임의 객체의 인스턴스 메서드 참조

- 설명: 첫 번째 매개변수(또는 해당 람다가 받을 대상)가 그 메서드를 호출하는 객체가 된다.
- 문법: `클래스명::인스턴스메서드명`
- 예: `Person::introduce`, 같은 람다: `(Person p) -> p.introduce()`

- Person

```
package methodref;

public class Person {
    private String name;
```

```
    public Person() {
        this("Unknown");
    }

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

    //정적 메서드
    public static String greeting() {
        return "Hello";
    }

    //정적 메서드 + 매개변수
    public static String greeting(String name) {
        return "Hello" + name;
    }

    //인스턴스 메서드
    public String introduce() {
        return "I am " + name;
    }

    //인스턴스 메서드 + 매개변수
    public String introduce(int number) {
        return "I am " + name + ", my Number is " + number;
    }

    public String getName() {
        return name;
    }


}
```

1. 정적 메서드 참조

```
Supplier<String> staticMethod1 = () -> Person.greeting();
System.out.println("staticMethod1.get() = " + staticMethod1.get());
Supplier<String> staticMethod2 = Person::greeting;
System.out.println("staticMethod2.get() = " + staticMethod2.get());
```

2. 특정 객체의 인스턴스 메서드 참조

```
//2. 특정 객체의 인스턴스 참조
Person person = new Person("Kim");
Supplier<String> instanceMethod1 = () -> person.introduce();
System.out.println("instanceMethod1.get() = " + instanceMethod1.get());
Supplier<String> instanceMethod2 = person::introduce;
System.out.println("instanceMethod2.get() = " + instanceMethod2.get());
```

3. 생성자 참조

```
//3. 생성자 참조
Supplier<Person> newPerson1 = () -> new Person();
System.out.println("newPerson1.get() = " + newPerson1.get());
Supplier<Person> newPerson2 = Person::new;
System.out.println("newPerson2.get() = " + newPerson2.get());
```

1. Person::greeting

- 정적 메서드를 참조한 예시이다.
- `() -> Person.greeting()`을 `Person::greeting`으로 간단히 표현했다.

2. person::introduce

- 특정 객체(`person`)의 인스턴스 메서드를 참조한 예시이다.
- `() -> person.introduce()`를 `person::introduce`로 간략화했다.

3. Person::new

- 생성자를 참조한 예시이다.
- `() -> new Person()`을 `Person::new`로 대체했다.

4. 직관적 이해가 안될 것. 추 후 설명

메서드 참조에서 ()를 사용하지 않는 이유

- 메서드 참조의 문법 뒤에 메서드 명 뒤에는 ()가 없음.
- ()는 메서드를 즉시 호출한다는 의미를 가짐. 여기서 ()가 없는 것은 메서드 참조를 하는 시점에는 메서드를 호출하는게 아니라, 단순히 메서드의 이름으로 해당 메서드 참조만 한다는 뜻.
    - 해당 메서드의 실제 호출 시점은 함수형 인터페이스를 통해서 이후에 이루어진다.

메서드 참조 2 - 매개변수1

매개변수가 있을 때 메서드를 참조하는 방법.

```
package methodref;

import java.util.function.Function;
import java.util.function.Supplier;

public class MethodRefEx3 {
    public static void main(String[] args) {
        //1. 정적 메서드 참조
        Function<String, String> staticMethod1 = (name) -> Person.greeting(nam
        Function<String, String> staticMethod2 = Person::greeting;

        System.out.println(staticMethod1.apply("kim"));
        System.out.println(staticMethod2.apply("kim"));

        //2. 특정 객체의 인스턴스 참조
        Person person = new Person("Kim");
        Function<Integer, String> instanceMethod1 = (n) -> person.introduce(n);
        System.out.println("instanceMethod1.get() = " + instanceMethod1.apply(1
        Function<Integer, String> instanceMethod2 = person::introduce;
        System.out.println("instanceMethod2.get() = " + instanceMethod2.apply(

        //3. 생성자 참조
        Function<String, Person> newPerson1 = (name) -> new Person(name);
        System.out.println("newPerson1.get() = " + newPerson1.apply("name"));
        Function<String, Person> newPerson2 = Person::new;
        System.out.println("newPerson2.get() = " + newPerson2.apply("name"));
```

```
    }
}
```

- 메서드 참조는 매개변수를 생략함.
    - 매개변수가 여러개라면 순서대로 전달.
- 함수형 인터페이스의 시그니처(매개변수와 반환 타입)가 이미 정해져있고, 컴파일러가 그 시그니처를 바탕으로 메서드 참조와 연결해주기 때문에 명시적으로 매개변수를 작성하지 않아도 자동으로 추론되어 호출.

메서드 참조 3 - 임의 객체의 인스턴스 메서드 참조

메서드 참조의 4가지 유형 중 - 임의 객체의 인스턴스 메서드 참조.

```
package methodref;

import java.util.function.Function;

public class MethodRefEx4 {
    public static void main(String[] args) {
        // 4. 임의 객체의 인스턴스 메서드 참조(특정 타입의)
        Person person1 = new Person("Kim");
        Person person2 = new Person("Park");
        Person person3 = new Person("Lee");

        // 람다
        Function<Person, String> fun1 = person -> person.introduce();
        System.out.println("person1.introduce = " + fun1.apply(person1));
        System.out.println("person2.introduce = " + fun1.apply(person2));
        System.out.println("person3.introduce = " + fun1.apply(person3));

        // 메서드 참조, 타입이 첫 번째 매개변수가 됨,
        // 그리고 첫 번째 매개변수의 메서드를 호출, 나머지는 순서대로 매개변수에 전달.
        Function<Person, String> fun2 = Person::introduce; // 타입::인스턴스메서드
        System.out.println("person1.introduce = " + fun2.apply(person1));
        System.out.println("person2.introduce = " + fun2.apply(person2));
        System.out.println("person3.introduce = " + fun2.apply(person3));
```

```
    }
}
```

```
Function<Person, String> fun1 = person -> person.introduce();
```

- 이 람다는 Person 타입을 매개변수로 받음. 그리고 매개변수로 넘겨받은 person 인스턴스의 introduce() 인스턴스 메서드를 호출

람다 실행

```
System.out.println("person1.introduce = " + fun1.apply(person1));
System.out.println("person2.introduce = " + fun1.apply(person2));
System.out.println("person3.introduce = " + fun1.apply(person3));
```

- 앞서 정의한 람다는 `Function<Person, String>` 함수형 인터페이스를 사용한다. 따라서 `Person` 타입의 인스턴스를 인자로 받고, `String`을 반환한다.
- 코드에서 보면 `apply()`에 `person1`, `person2`, `person3`을 각각 전달한 것을 확인할 수 있다.
  - `person1`을 람다에 전달하면 `person1.introduce()`가 호출된다.
  - `person2`를 람다에 전달하면 `person2.introduce()`가 호출된다.
  - `person3`을 람다에 전달하면 `person3.introduce()`가 호출된다.

- 매개변수로 지정한 특정 타입의 객체에 대해 동일한 메서드를 호출하는 패턴.
- 매개변수로 지정한 특정 타입의 임의 객체의 인스턴스 메서드를 참조.
- Reference to an instance method of an arbitary object of a particular type
- 클래스명::인스턴스메서드

`Person::introduce`와 같이 선언하면 다음과 같은 람다가 된다.

```
Person::introduce
1. 왼쪽에 지정한 클래스를 람다의 첫 번째 매개변수로 사용한다.
(Person person)

2. 오른쪽에 지정한 '인스턴스 메서드'를 첫 번째 매개변수를 통해 호출한다.
(Person person) -> person.introduce()
```

정리

1. 정적 메서드 참조 : 클래스명::클래스메서드(Person::greeting)

2. 특정 객체의 인스턴스 메서드 참조 : 객체명::인스턴스메서드(person::introduce)
3. 생성자 참조 : 클래스명::new(Person::new)
4. 임의 객체의 인스턴스 메서드 참조 : 클래스명::인스턴스메서드(Person::introduce)

2,4가 비슷할 수 있으나 기능은 완전히 다름.

- 둘의 문법이 다름. 인스턴스 메서드를 호출하지만, 하나는 객체명 하나의 클래스명을 사용
  객체명::인스턴스메서드(person::introduce)
  클래스명::인스턴스메서드(Person::introduce)
- 특정 객체의 인스턴스 메서드 참조
  - 메서드 참조를 선언할 때 부터 이름 그대로 특정 객체(인스턴스)를 지정해야 함. 선언 시점부터 이미 인스턴스가 지정되어 있음. 람다를 실행하는 시점에 인스턴스 변경 불가능
- 임의 객체의 인스턴스 메서드 참조
  - 메서드 참조를 선언할 때는 어떤 객체(인스턴스)가 대상이 될 지 모름.
  - 선언 시점에 호출할 인스턴스를 지정하지 않음. 호출 대상을 매개변수로 선언해두고, 실행 시점에 호출할 인스턴스를 받음. 실행 시점이 되어야 어떤 객체가 호출 될 지 알 수 있는 것. = 임의 객체의 인스턴스 메서드 참조
- **메서드 참조나, 람다를 정의하는 시점에 호출할 대상 인스턴스가 고정되는 것인지 아닌지.**

메서드 참조 4 - 활용

임의 객체의 인스턴스 메서드 참조가 실제 활용되는 예.

```
package methodref;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;

public class MethodRefEx5 {
    public static void main(String[] args) {
        List<Person> personList = List.of(
                new Person("Kim"),
                new Person("Park"),
```

```
            new Person("Lee")
        );

//      List<String> list = personList.stream()
//              .map(Person::introduce)
//              .toList();
//
//      System.out.println(list);

        List<String> result1 = mapPersonToString(personList, p -> p.introduce());
        System.out.println(result1);
        List<String> result2 = mapPersonToString(personList, Person::introduce)
        System.out.println(result2);


        List<String> upperResult1 = mapStringToString(result1, s -> s.toUpperCas
        System.out.println(upperResult1);
        List<String> upperResult2 = mapStringToString(result1, String::toUpperCa
        System.out.println(upperResult2);
    }

    static List<String> mapPersonToString(List<Person> personList, Function<P
        List<String> result = new ArrayList<>();
        for (Person person : personList) {
            result.add(fun.apply(person));
        }
        return result;
    }

    static List<String> mapStringToString(List<String> strings, Function<String,
        List<String> result = new ArrayList<>();
        for (String string : strings) {
            result.add(fun.apply(string));
        }
        return result;
    }
}
```

- Person::introduce / String::toUpperCase / 클래스::인스턴스메서드 형식이 임의 객체의 인스턴스 메서드 참조.
- 람다 대신에 메서드 참조를 사용한 덕분에 코드가 더 간결해지고 의도가 더 명확히 드러남.

메서드 참조 5 - 활용 2

스트림에 메서드 참조 활용

```
package methodref;

import lambda.lambda5.mystream.MyStreamV3;

import java.util.List;

public class MethodRefEx6 {
    public static void main(String[] args) {
        List<Person> personList = List.of(
                new Person("Kim"),
                new Person("Park"),
                new Person("Lee")
        );

        List<String> result1 = MyStreamV3.of(personList)
                .map(person -> person.introduce())
                .map(str -> str.toUpperCase())
                .toList();

        System.out.println(result1);

        List<String> result2 = MyStreamV3.of(personList)
                .map(Person::introduce)
                .map(String::toUpperCase)
                .toList();

        System.out.println(result2);
    }
}
```

- 람다로 가능하지만, 메서드 참조를 통해 더욱 직관적 표현이 가능한 것.

메서드 참조 6 - 매개변수 2

임의 객체의 인스턴스 메서드 참조에서 매개변수가 늘어난다면 ?

```
package methodref;

import java.util.function.BiFunction;

// 매개변수 추가
public class MethodRefEx7 {
    public static void main(String[] args) {
        // 4. 임의 객체의 인스턴스 메서드 참조(특정 타입의)
        Person person = new Person("Kim");

        // 람다
        BiFunction<Person, Integer, String> fun1 = (Person p, Integer num) -> p.ir
        System.out.println("person.introduceWithNumber: " + fun1.apply(person,

        // 메서드 참조, 타입이 첫 번째 매개변수, 그리고 첫 번쨔 매개변수의 메서드를 호출
        // 나머지는 순서대로 매개변수에 전달
        BiFunction<Person, Integer, String> fun2 = Person::introduceWithNumbe
        System.out.println("person.introduceWithNumber: " + fun2.apply(person,
    }
}
```

- fun1은 람다를 사용하여 p.introduceWithNumber(number)를 호출.
- fun2는 Person::introduceWithNumber(number)로, 첫 번째 매개변수 Person이 메서드를 호출하는 객체가 되고 두 번째 매개변수가 Integer로 introduceWithNumber의 실제 인자로 전달.
    - 첫 번째 이후 매개변수는 모두 순서대로 실제 인자로 전달.
- 임의 객체의 인스턴스 메서드 참조는 함수형 인터페이스의 시그니처에 따라
    - 첫 번째 인자를 호출 대상 객체로
    - 나머지 인자들을 순서대로 해당 메서드의 매개변수로 전달

정리

메서드 참조의 필요성

- 람다에서 이미 정의된 메서드를 단순히 호출하기만 하는 경우 메서드 참조로 더 간결하게 표현

1. 간결성
2. 가독성
3. 유연성
4. 재사용성

메서드 참조의 4가지 유형

1. 정적 메서드 참조
2. 특정 객체의 인스턴스 메서드 참조
3. 생성자 참조
4. 임의 객체의 인스턴스 메서드 참조