8253jang@daum.net
장희정

람다식에서 메소드 참조와 생성자 참조란?

자바 8에서는 람다식(Lambda Expressions)을 도입하면서, 코드가 더욱 간결해졌다. 람다식에서 더 나아가 메소드 참조(Method Reference)와 생성자 참조(Constructor Reference)를 사용하면 **기존 메소드를 더욱 직관적으로 사용할 수 있다.**

1. 메소드 참조 (Method Reference)란?

메소드 참조는 **이미 존재하는 메소드를 람다식보다 더 간결하게 표현하는 방법**으로 람다식에서 불필요한 코드를 줄이고, 메소드 자체를 참조해서 호출할 수 있다.

☞ 기본문법

클래스이름::메소드이름
객체이름::메소드이름

:: 기호를 사용해서 메소드를 참조한다.

☞ 메소드 참조의 종류

종류	람다식	메소드 참조
정적 메소드 참조	(x) -> 클래스.메소드(x)	클래스::메소드
인스턴스 메소드 참조	(x) -> 객체.메소드(x)	객체::메소드
특정 객체의 인스턴스 메소드 참조	(x, y) -> x.메소드(y)	클래스::메소드
생성자 참조	(x) -> new 생성자(x)	클래스::new

☞ 메소드 참조 예제

예) 정적 메소드 참조

```
package lamda.methodRef;
import java.util.function.Function;

class Utils {
	public static int square(int x) {
		return x * x;
	}
}
```

```
public class MethodReferenceExample01 {
    public static void main(String[] args) {
        //기존방식
        Function<Integer, Integer>  beforeSquare =  new Function<Integer, Integer>() {
                @Override
                public Integer apply(Integer t) {
                  return Utils.square(t);
                }
        };

        // 람다식 방식
        Function<Integer, Integer> lambdaSquare = x -> Utils.square(x);

        // 메소드 참조 방식
        Function<Integer, Integer> methodRefSquare = Utils::square;

        System.out.println(beforeSquare.apply(5)); // 25
        System.out.println(lambdaSquare.apply(5)); // 25
        System.out.println(methodRefSquare.apply(5)); // 25
    }
}
```

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

Function<T, R> 는 입력 타입 T를 받아서 결과 타입 R을 반환하는 함수형 인터페이스

Function<Integer, Integer>는 하나의 정수를 입력 받아 정수를 반환하는 함수형 인터페이스이다.
lambdaSquare 는 람다식이고, methodRefSquare 는 메소드 참조방법이다.
Utils::square를 사용하면 더 간결하게 square() 메소드를 호출할 수 있다.

예) 인스턴스 메소드 참조

```
package lamda.methodRef;
import java.util.function.Function;

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

public class InstanceMethodReference02 {
	public static void main(String[] args) {
		StringUtils utils = new StringUtils();

		//기존방식
		Function<String, Integer> beforeLength =new Function<String, Integer>() {
			@Override
			public Integer apply(String t) {
				return utils.getLength(t);
			}
		};

		// 람다식 방식
		Function<String, Integer> lambdaLength = s -> utils.getLength(s);

		// 메소드 참조 방식
		Function<String, Integer> methodRefLength = utils::getLength;

		System.out.println(beforeLength.apply("Hello")); // 5
		System.out.println(lambdaLength.apply("Hello")); // 5
		System.out.println(methodRefLength.apply("Hello")); // 5
	}
}
```

Function<String, Integer>는 문자열을 입력 받아 길이를 반환하는 함수형 인터페이스이다.
utils.getLength(s)를 람다식으로 쓰면 복잡하지만, utils::getLength 로 단순화 가능하다.

예) 특정 객체의 인스턴스 메소드 참조

```
package lamda.methodRef;
import java.util.function.BiFunction;

public class SpecificInstanceMethod03 {
    public static void main(String[] args) {
    //기존방식
  BiFunction<String, String, Boolean> beforeEquals = new BiFunction<String, String, Boolean>() {
          @Override
          public Boolean apply(String t, String u) {
              return t.equals(u);
           }
     };

     // 람다식방식
      BiFunction<String, String, Boolean> lambdaEquals = (s1, s2) -> s1.equals(s2);

     //메소드 참조방식
      BiFunction<String, String, Boolean> methodRefEquals = String::equals;

       System.out.println(beforeEquals.apply("heejung", "heejung")); // true
       System.out.println(lambdaEquals.apply("Java", "Java")); // true
       System.out.println(methodRefEquals.apply("Java", "Python")); // false
    }
}
```

```
@FunctionalInterface
public interface BiFunction<T, U, R> {
    R apply(T t, U u);
}
```

BiFunction<T, U, R>는 입력 타입 두 개 T, U를 받아서 결과 타입 R을 반환하는 함수형 인터페이스(주로 두 값을 조합해서 새로운 값을 만들 때 사용)

BiFunction<String, String, Boolean>는 두 개의 문자열을 비교해서 Boolean 값을 반환한다.
"Java".equals("Java")를 메소드 참조(String::equals)로 변환 가능하다.

8253jang@daum.net
장희정

2. 생성자 참조 (Constructor Reference)란?

생성자 참조는 **클래스의 생성자를 람다식보다 간결하게 표현하는 방법**이다.

객체를 생성할 때 **new 키워드를 직접 사용하지 않고, 생성자를 참조해서 객체를 만들 수 있다.**

☞기본문법

클래스이름::new

::new를 사용해서 생성자를 참조한다.

☞ 생성자 참조 예제

예) 기본 생성자 참조

```
package lamda.constructorRef;
import java.util.function.Supplier;

class Person {
    public Person() {
        System.out.println("새로운 Person 객체 생성!");
    }
}

public class ConstructorReference01 {
    public static void main(String[] args) {
        //기존방식
        Supplier<Person> beforePerson = new Supplier<Person>() {
            @Override
            public Person get() {
                return new Person();
            }
        };

        // 람다식 방식
        Supplier<Person> lambdaPerson = () -> new Person();

        // 생성자 참조 방식
        Supplier<Person> methodRefPerson = Person::new;
```

```
        Person p1 = beforePerson.get();// 새로운 Person 객체 생성
        Person p2 = lambdaPerson.get(); // 새로운 Person 객체 생성
        Person p3 = methodRefPerson.get(); // 새로운 Person 객체 생성
    }
}
```

```
@FunctionalInterface
public interface Supplier<T> {
    T get();
}
```

Supplier<T> 입력 값이 없음 , 어떤 연산을 해서 **값을 만들어서 반환하는 역할**만 함.
주로 매번 새로운 객체나 값을 **공급**해야 할 때 사용
스트림에서 generate() 메소드와 함께 자주 쓰임

Supplier<T>는 매개변수 없이 객체를 생성하는 함수형 인터페이스이다.
() -> new Person() 대신 Person::new로 더 간결하게 표현 가능하다.

예) 매개변수가 있는 생성자 참조

```
package lamda.constructorRef;
import java.util.function.Function;

class User {
    String name;
    public User(String name) {
        this.name = name;
    }

    public void printName() {
        System.out.println("User 이름: " + name);
    }
}

public class ConstructorReferenceWithParams02 {
    public static void main(String[] args) {
```

```
        // 기존방식
         Function<String, User> beforeUser = new Function<String, User>() {
                @Override
                public User apply(String name) {
                        return new User(name);
                }
        };

        // 람다식 방식
        Function<String, User> lambdaUser = name -> new User(name);

        // 생성자 참조 방식
        Function<String, User> methodRefUser = User::new;

        User user0 = lambdaUser.apply("heejung");
        User user1 = lambdaUser.apply("Alice");
        User user2 = methodRefUser.apply("Bob");

        user0.printName(); // User 이름: heejung
        user1.printName(); // User 이름: Alice
        user2.printName(); // User 이름: Bob
    }
}
```

Function<String, User>는 문자열을 입력 받아 User 객체를 생성하는 인터페이스이다.
name -> new User(name) 대신 User::new를 사용하면 코드가 더 직관적이다.

3. 메소드 참조와 생성자 참조를 언제 사용하면 좋을까?

메소드 참조는 언제 사용하면 좋을까?

- 기존의 메소드를 람다식보다 간결하게 표현할 때.
- 반복적으로 같은 메소드를 호출해야 할 때.
- 코드 가독성을 높이고 싶을 때.

생성자 참조는 언제 사용하면 좋을까?

- 객체 생성을 람다식보다 더 직관적으로 표현하고 싶을 때.
- 생성자를 인자로 전달해야 할 때.

- 특정 인터페이스(Supplier, Function 등)에서 new 를 사용하고 싶을 때