아이템 42. 익명 클래스보다는 람다를 사용하라

결론

- **람다는 작은 함수 객체를 아주 쉽게 표현할 수 있다.**
- **람다를 사용하지 못하는 경우 및 사용하는 것이 적절하지 않은 경우에만 익명 클래스를 사용하자.**

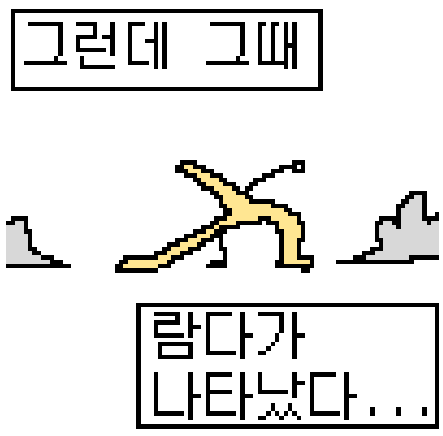
과거

함수 타입을 표현해야 할 때, **추상 메서드를 하나만 담은 인터페이스 또는 드물게는 추상 클래스**를 사용했다.

- 이런 인터페이스의 **인스턴스를 함수 객체(function object)**라고 하여, 특정 함수나 동작을 나타내는데 사용했다.
- JDK 1.1이 등장하며 함수 객체를 만드는 주요 수단은 익명 클래스(Item24)가 되었다.

```
Collections.sort(words, new Comparator<String>() {
  public int compare(String s1, String s2) {
    return Integer.compare(s1.length(), s2.length());
  }
});
```

람다식의 등장



Java 8이 되며 추상 메서드 하나짜리 인터페이스는 **특별한 대우**를 받기 시작했다 !

- (함수형) **인터페이스들의 인스턴스를 람다식(lambda expression)을 사용해 만들 수 있게 된 것.**
- 익명 클래스와 개념은 비슷하지만 코드는 훨씬 간결하다 !

```
Collections.sort(words, (s1, s2) -> Integer.compare(s1.length(), s2.length()));
```

- 예제에서 볼 수 있듯, 기존 코드에 비해 **타입에 대한 정보도 적다.**
  - lambda 반환 타입 - Comparator<String>
  - 매개변수(s1, s2) 타입 - String
  - 반환 값 타입 - int

- 이 정보는 **컴파일러가 문맥을 살피며 대신 타입을 추론해준 것**
  - 컴파일러가 타입을 결정하지 못하는 경우도 존재한다 → 프로그래머가 직접 명시해야한다.
  - 이런 경우는 대부분 제네릭의 로타입 사용에서 발생한다.
    - 로타입 제네릭 메서드와 람다식을 같이 활용할 때, 컴파일러가 타입추론 하기가 힘들다.

```
// Generic raw
List words = Arrays.asList("어,,,", "이게", "왜", "되지...?");
Collections.sort(words, (s1, s2) -> Integer.compare(s1.length(), s2.length())); // ERROR
```

```
// 무조건 raw type을 사용해야겠다면...
Collections.sort(words, (s1, s2) -> Integer.compare(((String)s1).length(), ((String)s2).length()));

// Generic
List<String> words = Arrays.asList(new String[]{"어,,,", "이게", "왜", "되지...?"});
Collections.sort(words, (s1, s2) -> Integer.compare(s1.length(), s2.length()));
```

- 계속 나오던 `Collections.sort` **코드는 비교자 생성 메서드와 List 인터페이스에 추가된 sort 메서드를 이용**하면 더 간결히 짧게 표현할 수 있다.
  - 또한 위 예제에서 로 타입으로 발생했던 문제도 해결할 수는 있다. (그래도 로 타입은 사용하지 말자...)

```
List<String> words = Arrays.asList(new String[]{"어,,,", "이게", "왜", "되지...?"});
// 이전 방법
Collections.sort(words, (s1, s2) -> Integer.compare(s1.length(), s2.length()));

// 비교자 생성 메서드 사용
Collections.sort(words, Comparator.comparing(String::length));

// List.sort 메서드까지 사용 사용
words.sort(Comparator.comparing(String::length));
```

람다식의 다른 예제

람다를 활용해 더 나아진 또 다른 예를 보자.

- Item 34에서의 Operation 예제

```
public enum Operation {
    PLUS("+") {
        public double apply(double x, double y) {
            return x + y;
        }
    },
    MINUS("-") {
        public double apply(double x, double y) {
            return x - y;
        }
    },,
    TIMES("*") {
        public double apply(double x, double y) {
            return x * y;
        }
    },,
    DIVIDE("/") {
        public double apply(double x, double y) {
            return x / y;
        }
    },
    ;

    private final String symbol;

    Operation(String symbol) {
        this.symbol = symbol;
    }

    public abstract double apply(double x, double y);
}
```

- 추상 메서드를 선언하고 이를 각 열거 타입마다 재정의함으로써, 각 타입마다 메서드가 다르게 작동하도록 구현할 수 있다.
- 여기에 람다를 사용해 더 간단히 만들어보자.

```
public enum Operation {
    PLUS("+", (x, y) -> x + y),
    MINUS("-", (x, y) -> x - y),
    TIMES("*", (x, y) -> x * y),
    DIVIDE("/", (x, y) -> x / y);

    private final String symbol;
    private final DoubleBinaryOperator op;
```

```
    Operation(String symbol, DoubleBinaryOperator op) {
        this.symbol = symbol;
        this.op = op;
    }

    public double apply(double x, double y) {
        return op.applyAsDouble(x, y);
    }
}
```

- `java.util.function`은 다양한 함수 인터페이스(Item 44)를 제공하는 패키지
  - `DoubleBinaryOperator` 는 double 타입인수 2개를 받아 double 타입 결과를 반환한다.

그럼, 람다식이 무조건 좋은 것인가?

그렇지 많은 않다. 람다가 주는 이점이 많음에도 람다의 한계, 일반 클래스 몸체와 차이점이 존재한다.

람다는 간결하게 사용될 때만 사용하자.

- 람다는 이름도 없고, 문서화도 못한다.
- 즉, 코드 자체로 동작이 명확히 설명되지 않거나 코드 줄 수가 많아지면 람다를 사용하지 말아야 한다.
- 람다는 한 줄일 때 가장 좋고, 세 줄 안에 끝내는 것이 가장 좋다.

람다는 함수형 인터페이스에서만 사용된다.

- 람다는 함수형 인터페이스의 인스턴스를 만들 때만 사용된다.
- 추상 클래스의 인스턴스를 만들 때는 사용할 수 없고, 이런 경우에는 익명 클래스를 사용해야 한다.

람다는 자신을 참조할 수 없다.

- 익명 클래스에서의 this는 익명 클래스의 인스턴스 자신을 가리킨다.
- 반면, 람다에서의 this 키워드는 바깥 인스턴스를 가리킨다.
- 함수 객체가 자신을 참조해야하는 경우, 익명 클래스를 사용해야만 한다.

```
public interface FunctionalInterface {
    int NUM = 10;
    void printThis();
}

// Some class
public class SomeClass {
    int NUM = 999;

    public void print() {
        FunctionalInterface f1 = new FunctionalInterface(){
            @Override
            public void printThis() {
                System.out.println(this.NUM); // 10

            }
        };

        FunctionalInterface f2 = () -> System.out.println(this.NUM); // 999
        f1.printThis();
        f2.printThis();
    }
}
```

람다는 직렬화를 해서는 안된다.

- 람다도 익명 클래스처럼 직렬화 형태가 구현별로 다를 수 있다.
- 따라서 람다를 직렬화하는 일을 극히 삼가야 한다. (익명 클래스도 마찬가지)
- 직렬화해야만 하는 함수 객체가 있다면 private static 중첩 클래스(item24)의 인스턴스를 사용하자.