Chapter 03.코틀린 고급 - 06.제네릭

제네릭

1. 코틀린의 제네릭

- 제네릭 선언 방법에 대해 알아본다
- 코틀린의 클래스는 자바와 마찬가지로 `타입 파라미터`를 가질 수 있다

```
class MyGenerics<T>(val t: T)
```

- 제네릭을 사용한 클래스의 인스턴스를 만들려면 `타입 아규먼트` 를 제공하면 된다

```
fun main() {
    val generics = MyGenerics<String>("테스트")
}
```

- 또한 인자를 통해 타입 추론이 가능한 경우 타입 아규먼트를 생략할 수 있다

```
val generics = MyGenerics("테스트")
```

- 타입 추론이 가능하기 때문에 변수의 타입에 타입 아규먼트를 추가해도되고 그렇지 않은 경우 타입 아규먼트를 생성자에서 추가해도 된다

```
val list1: MutableList<String> = mutableListOf()
val list2 = mutableListOf<String>()
```

- 어떤 타입이 들어올지 알수 없지만 안전하게 사용하고 싶은 경우 스타 프로젝션 구문을 제공한다

```
val list: List<*> = listOf<String>("테스트")
```

---

2. 변성

- 제네렉에서 파라미터화된 타입이 서로 어떤 관계에 있는 지 설명하는 개념
- 변성은 크게 공변성, 반공변성 그리고 무공변성으로 나뉜다
- 이펙티브 자바에선 공변성과 반공변성을 설명할때 `PECS` 규칙을 언급한다
- *PECS는 Producer-Extends, Consumer-Super의 약자입니다*
  - 공변성은 자바 제네릭의 `extends` 키워드이고 코틀린에선 `out`
  - 반공변성은 자바 제네릭의 `super` 키워드이고 코틀린에선 `in`

2.1 공변성

- 공변성을 쓰지 않아 문제가 되는 케이스

```
class MyGenerics<T>(val t: T)

fun main() {
    val generics = MyGenerics<String>("테스트")

    val charGenerics : MyGenerics<CharSequence> = generics // 컴파일 오류
}
```

- 현재 코드에서 실수로 String이 아닌 타입을 넣을 경우를 막으므로 의미 있는 컴파일 오류지만 특정 상황에선 공변성이 필요할 수 있다
- 공변성 키워드인 `out` 을 사용해서 해결한다

```
class MyGenerics<out T>(val t: T)

fun main() {
    val generics = MyGenerics<String>("테스트")

    val anygenerics : MyGenerics<CharSequence> = generics
}
```

- 공변성은 CharSeqeunce가 String의 상위 타입일때 MyGenerics<CharSequence>가 MyGenerics<String>의 상위 타입이므로 공변성이다

2.2 반공변성

- 반공변성을 쓰지 않아 문제가 되는 케이스

```
class Bag<T> {

    fun saveAll(
        to: MutableList<T>,
        from: MutableList<T>,
    ) {
        to.addAll(from)
    }
}


fun main() {
    val bag = Bag<String>()

    // in이 없다면 컴파일 오류
    bag.saveAll(mutableListOf<CharSequence>(""), mutableListOf<String>(""))

}
```

- 이때 반공변성 키워드인 `in`을 사용해 상위 타입도 전달 받을 수 있도록 한다

```
class Bag<T> {

    fun saveAll(
        to: MutableList<in T>,
        from: MutableList<T>,
    ) {
        to.addAll(from)
    }
}


fun main() {
     val bag = Bag<String>()

    bag.saveAll(mutableListOf<CharSequence>(""), mutableListOf<String>(""))

}
```

- 공변성과는 반대로 String이 CharSequence의 하위 타입일때 Bag<String>가 Bag<CharSequence>의 상위 타입이 되므로 반공변성이다

- 기본은 무공변성으로써 in, out 어떤 것도 지정하지 않은 경우
    - String은 CharSequence의 하위 타입이지만 MyGenerics<String>은 MyGenerics<CharSequence>와 아무 관계도 아니다 이게 무공변성이다