# 제네릭(Generics)

## 코틀린 기초강좌

강사: 배정만

# 학습요약

- 학습목표: 제네릭 학습
- 핵심키워드: Generics, Variance, Covariant, Contravariant
- 학습내용: 제네릭의 개념과 원리를 배웁니다.

# 제네릭<T>

- 제네릭은 저장하거나 사용하는 타입에 대한 자리 표시자(타입 파라미터)를 포함하는 클래스, 인터페이스 및 함수입니다.
- 제네릭은 레퍼런스 타입입니다.

# 자바의 제네릭

- 자바에서 와일드카드는 까다롭다.
  - ? extends Number
  - ? super Long
- List<String>은 List<Object>의 하위 형식이 아님
  - Java의 제네릭 형식은 불변(Invariant)
- 유연성을 높이기 위해 제네릭과 와일드 카드를 같이 사용해야함

# 자바의 제네릭 - 예제

```
interface Collection<E> ... {
  void addAll(Collection<E> items);
}

void copyAll(Collection<Object> to, Collection<String> from) {
  to.addAll(from);
 // Collection<String>은 Collection<Object>의 하위 타입이 아님
}

interface Collection<E> ... {
  void addAll(Collection<? extends E> items);
}
```

- <? extends E>
  - Collection<String>은
    Collection<? extends Object>의 하위 타입이 된다.
  - 공변을 보장 (상위 호환)
- <? super E>
  - List<? super String>은
    List<Object>의 상위 타입이 된다.
  - 반공변을 보장 (하위호환)

# 공변성 (Covariance)

- 선언한 만큼은 읽을 수 있다.
- 원래 지정된 것보다 더 많이 파생된 형식을 사용할 수 있음 (생산자, extend, out) // 하위 호환

```
interface Source<out T> {
    fun nextT(): T
}

fun demo(strs: Source<String>) {
    val objects: Source<Any> = strs
    // T 가 out 파라미터이기 때문에 괜찮다.
    ...
}
```

# 반공변성 (Contravariance)

- 다 채우지 않더라도 기록할 수 있다.
- 원래 지정된 것보다 더 제네릭한(덜 파생적인) 형식을 사용할 수 있음 (소비자, super, in) // 상위 호환

```
interface Comparable<in T> {
    operator fun compareTo(other: T): Int
}

fun demo(x: Comparable<Number>) {
    x.compareTo(1.0) //

    val y: Comparable<Double> = x // OK!
}
```

# 타입 투영 (Type projections)

- 타입 파라미터 T를 선언하고 사용부의 하위 타입 지정에 대한 문제를 방지하는 것이 매우 편리함
- 일부 클래스는 실제로 T를 반환하도록 제한 될 수 없음
- 이로인한 캐스트 오류 발생 가능

# 사용부의 타입 투영 (Type projections)

```
fun copy(from: Array<Any>, to: Array<Any>) {
    assert(from.size == to.size)
    for (i in from.indices)
        to[i] = from[i]
}
```

```
val ints: Array<Int> = arrayOf(1, 2, 3)
val any = Array<Any>(3) { "" }
copy(ints, any)
//   ^ Int타입의 배열 클래스이지만 Any가 추가됨
```

```
fun copy(from: Array<out Any>, to: Array<Any>) { ... }
```

# 제네릭 함수

```
fun <T> singletonList(item: T): List<T> {
    ...
}

fun <T> T.basicToString() : String {
    // extension function
    …
}
```

# 제네릭 함수

```
val l = singletonList<Int>(1)

val l = singletonList(1)
```

- 제네릭 함수를 호출하려면 함수의 이름 뒤에 호출 사이트에서 유형 인수를 지정함
- 타입 인수가 컨텍스트에서 유추 될 수 있으면 생략 할 수 있음

# 제네릭 제약: 상한선

```
fun <T : Comparable<T>> sort(list: List<T>) {  ... }

sort(listOf(1, 2, 3))
sort(listOf(HashMap<Int, String>())) // 에러
```

- 자바의 상속(extends) 관계에 있는 경우로 제한
- Int 는 Comparable<Int>의 하위타입
- HashMap<Int, String> 은Comparable<HashMap<>>의 하위타입이 아님