05

# 05-3. Reflection

다양한 기법

# 리플렉션

## 리플렉션 이해

- 리플렉션은 런타임시 프로그램의 구조(객체, 함수, 프로퍼티)를 분석해 내는 기법
- kotlin-reflect.jar 라는 라이브러리에서 제공

## 클래스 타입과 레퍼런스

- 런타임시 동적인 클래스 분석
- 클래스에 대한 정보를 클래스 Reference , 클래스 Reference가 대입되는 곳은 클래스 타입
- 클래스 타입은 KClass<*> 로 표현하며 이곳에 대입되는 클래스 Reference는 "클래스명::class" 로 표현

```
val myVal: KClass<*> = String::class

fun myFun(arg: KClass<*>){

}
```

# 리플렉션

## 클래스 타입과 레퍼런스

- KClass<*> 은 모든 타입의 클래스 Reference 도 대입이 가능
- 특정 타입의 클래스 Reference만 대입되어야 한다면 Kclass<클래스명>
- 자바 클래스의 Reference는 .java 를 추가

```
val myVal: KClass<String> = String::class

//val myVal2: KClass<String> = Double::class//error

val myVal3: Class<*> = String::class.java
```

# 리플렉션

---

**클래스 정보 분석**

- val isAbstract: Boolean : 클래스 Reference 가 abstract 로 선언되었는지 판단
- val isCompanion: Boolean : 클래스 Reference 가 companion 로 선언되었는지 판단
- val isData: Boolean : 클래스 Reference 가 data 로 선언되었는지 판단
- val isFinal: Boolean : 클래스 Reference 가 final 로 선언되었는지 판단
- val isInner: Boolean : 클래스 Reference 가 inner 로 선언되었는지 판단
- val isOpen: Boolean : 클래스 Reference 가 open 로 선언되었는지 판단
- val isSealed: Boolean : 클래스 Reference 가 sealed 로 선언되었는지 판단

**생성자 분석**

- val constructors: Collection<KFunction<T>> : 모든 생성자 정보
- val <T : Any> KClass<T>.primaryConstructor: KFunction<T>? : 주생성자 정보

# 리플렉션

---

**클래스 프로퍼티 분석**

- val <T : Any> KClass<T>.declaredMemberProperties: Collection<KProperty1<T, *>> : 확장 프로퍼티를 제외한 클래스에 선언된 모든 프로퍼티 리턴
- val <T : Any> KClass<T>.memberProperties: Collection<KProperty1<T, *>> : 확장 프로퍼티를 제외한 클래스와 상위 클래스에 선언된 모든 프로퍼티 리턴
- val <T : Any> KClass<T>.declaredMemberExtensionProperties: Collection<KProperty2<T, *, *>> : 클래스에 선언된 확장 프로퍼티 리턴
- val <T : Any> KClass<T>.memberExtensionProperties: Collection<KProperty2<T, *, *>> : 상위 클래스 및 현 클래스의 확장 프로퍼티 리턴

**클래스 함수 분석**

- val KClass<*>.declaredMemberFunctions: Collection<KFunction<*>> : 확장 함수를 제외한 클래스에 선언된 모든 함수 리턴
- val KClass<*>.memberFunctions: Collection<KFunction<*>> : 확장 함수를 제외한 클래스와 상위 클래스에 선언된 모든 함수 리턴
- val KClass<*>.declaredMemberExtensionFunctions: Collection<KFunction<*>> : 클래스에 선언된 확장 함수 리턴
- val KClass<*>.memberExtensionFunctions: Collection<KFunction<*>> : 상위 클래스 및 현 클래스의 확장 함수 리턴

# 리플렉션

## 함수 레퍼런스 분석

- 함수 Reference는 "::함수명" 을 이용, 함수는 KFunction<*> 타입으로 이용

```
fun myFun(){ }

class MyClass {
    fun myFun2() { }
}

val funReference: KFunction<*> = ::myFun

val funReference2: KFunction<*> = MyClass::myFun2
```

- val name: String : 함수이름
- val parameters: List<KParameter> : 함수에 선언된 모든 매개변수
- val returnType: KType : 함수의 리턴 타입

# 리플렉션

## 프로퍼티 레퍼런스 분석

- 프로퍼티 Reference는 "::프로퍼티명", 타입은 KProperty<*> 와 KMutableProperty<*>

```
val myVal: Int = 10

var myVar: Int = 10

val referenceVal1: KProperty<*> = ::myVal

val referenceVal2: KProperty<*> = ::myVar

//val referenceVal3: KMutableProperty<*> = ::myVal//error

val referenceVal4: KMutableProperty<*> = ::myVar

fun main(args: Array<String>) {
    println(::myVal.get())

    ::myVar.set(30)
    println(::myVar.get())
}
```

# 리플렉션

---

**Kproperty**

- val getter: Getter<R> : get() 함수의 정보
- val isConst: Boolean : const으로 선언된건지 판단
- val isLateinit: Boolean : lastinit으로 선언된건지 판단
- val isAbstract: Boolean : abstract로 선언된건지 판단
- val isFinal: Boolean : finale로 선언된건지 판단
- val isOpen: Boolean : open 으로 선언된건지 판단
- val name: String : 프로퍼티 이름 획득
- val returnType: KType : 프로퍼티 타입 획득
- fun call(vararg args: Any?): R : 프로퍼티 get() 호출

**KMutableProperty**

- val setter: Setter<R> : set() 에 대한 정보

# 감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare