Chapter 03.코틀린 고급 - 04.실드 클래스

실드 클래스

1. 실드 클래스를 쓰는 이유

- 하나의 상위 클래스 또는 인터페이스에서 하위 클래스 정의를 제한할 수 있는 방법
- 먼저 Developer라는 추상클래스를 상속받은 Backend, Frontend 클래스가 있다

```
abstract class Developer {

    abstract val name: String
    abstract fun code(language: String)

}

data class BackendDeveloper(override val name: String) : Developer() {

    override fun code(language: String) {
        println("저는 백엔드 개발자입니다 ${language}를 사용합니다")
    }
}

data class FrontendDeveloper(override val name: String) : Developer() {

    override fun code(language: String) {
        println("저는 프론트엔드 개발자입니다 ${language}를 사용합니다")
    }
}
```

- 그다음은 생성된 Developer 객체를 보관하는 DeveloperPool을 만든다

```
object DeveloperPool {
    val pool = mutableMapOf<String, Developer>()

    fun add(developer: Developer) = when (developer) {
        is BackendDeveloper -> pool[developer.name] = developer
        is FrontendDeveloper -> pool[developer.name] = developer
        else -> {
            println("지원하지않는 개발자입니다")
        }
    }

    fun get(name: String) = pool[name]
}


fun main() {
    val backendDeveloper = BackendDeveloper(name = "토니")
    DeveloperPool.add(backendDeveloper)

    val frontendDeveloper = FrontendDeveloper(name = "카즈야")
    DeveloperPool.add(frontendDeveloper)

    println(DeveloperPool.get("토니"))
    println(DeveloperPool.get("카즈야"))
}
```

- 겉보기엔 문제가 없어보인다

- 만약 `else` 를 삭제하면 무슨일이 발생할까?

```
object DeveloperPool {
    val pool = mutableMapOf<String, Developer>()

    fun add(developer: Developer) = when (developer) { // 컴파일 오류
        is BackendDeveloper -> pool[developer.name] = developer
        is FrontendDeveloper -> pool[developer.name] = developer
    }

    fun get(name: String) = pool[name]
}
```

- when식에서 컴파일 오류가 발생한다. 컴파일러 입장에선 Developer를 상속한 하위클래스가 몇개나 되는지 어떤 클래스가 상속받고 있는지 알 수 있는 방법이 없다

- 이때 abstract 키워드를 `sealed` 로 바꿔보자

```
sealed class Developer {

    abstract val name: String
    abstract fun code(language: String)

}
```

- when식에 else가 없음에도 오류가 발생하지 않는다
- 실드 클래스는 하위 클래스를 `제한 조건에 따라 정의해야하고` 이렇게 만들어진 실드 클래스의 하위 클래스는 `컴파일러가 컴파일 시점에 판단할 수 있다`
- 제한 조건으로 최신 코틀린 1.6 버전 기준 같은 패키지 또는 모듈 안에 있는 경우에만 하위 클래스를 정의할 수 있다. 그 이전에는 같은 파일내에 정의해야했다

---

2. 활용 예시

- 실드 클래스로 만들면 새로운 하위 클래스가 생길 경우 when식에서 새로운 하위 클래스에 대한 처리를 해야한다
- 새로운 하위 클래스인 OtherDeveloper를 추가한다

```
object OtherDeveloper : Developer() {

    override val name: String = "익명"

    override fun code(language: String) {
        TODO("Not yet implemented")
    }

}
```

- when식에서 컴파일 오류 발생

```
object DeveloperPool {
    val pool = mutableMapOf<String, Developer>()

    fun add(developer: Developer) = when (developer) { // 컴파일 오류
        is BackendDeveloper -> pool[developer.name] = developer
        is FrontendDeveloper -> pool[developer.name] = developer
    }

    fun get(name: String) = pool[name]
}
```

- 아래와 같이 처리 가능

```
object DeveloperPool {
    val pool = mutableMapOf<String, Developer>()

    fun add(developer: Developer) = when (developer) {
        is BackendDeveloper -> pool[developer.name] = developer
        is FrontendDeveloper -> pool[developer.name] = developer
        is OtherDeveloper -> println("지원하지않는 개발자종류입니다")
    }

    fun get(name: String) = pool[name]
}
```

- else를 제거할 수 있으므로 버그가 줄어든다
- when식을 else로 바꾸고 AndroidDeveloper를 추가하고 실수로 Developer.pool에서 처리를 누락했을경우

```
data class AndroidDeveloper(override val name: String) : Developer() {

    override fun code(language: String) {
        println("저는 안드로이드 개발자입니다 ${language}를 사용합니다")
    }
}
```

```
object DeveloperPool {
    val pool = mutableMapOf<String, Developer>()

    fun add(developer: Developer) = when (developer) {
        is BackendDeveloper -> pool[developer.name] = developer
        is FrontendDeveloper -> pool[developer.name] = developer
        else -> println("지원하지않는 개발자종류입니다")
    }

    fun get(name: String) = pool[name]
}

fun main() {
  val androidDeveloper = AndroidDeveloper(name="안드로")
  DeveloperPool.add(androidDeveloper)
}
```

- 의도치 않은 결과가 발생 (OtherDeveloper만 `println("지원하지않는 개발자종류입니다")` 를 출력하려했음)
- 아래와 같이 모든 하위 클래스를 추가하여 이러한 버그를 줄일 수 있다

```
fun add(developer: Developer) = when (developer) {
        is BackendDeveloper-> pool[developer.name] = developer
        is FrontendDeveloper -> pool[developer.name] = developer
        is AndroidDeveloper -> pool[developer.name] = developer
        is OtherDeveloper -> println("지원하지않는 개발자종류입니다")
    }
```