Chapter 02.코틀린 기초 - 08.상속

상속

1. 자바의 상속

- 객체지향 핵심 원칙 중 하나인 `상속`은 상속을 통해 기존 코드를 재사용하거나 확장할 수 있다
- 자바는 기본적으로 모든 클래스가 상속이 가능하나 상속에 따른 부작용이 발생할 경우를 대비해 `final` 키워드로 막을 수 있다
- 대표적으로 `System 클래스`

```
System.out.println("Hello World");

public final class System {
    /* ... */
}
```

- 이펙티브 자바의 아이템 중 `상속을 위한 설계와 문서를 작성하고 그렇지 않으면 상속을 금지하라` 라는 주제가 있는데 여기에는 여러가지 상속에 대한 문제점에 대해 나와있으며 결과적으로 상속을 목적으로 만든 클래스가 아니라면 `모두 final로 작성하는 것이 좋다`

2. 코틀린의 상속

- 자바의 모든 클래스의 조상은 `Object` 코틀린에서 모든 클래스의 조상은 `Any`
- Any에는 equals, hashCode, toString이 존재하고 모든 클래스로 자동 상속된다
- 코틀린의 클래스는 기본적으로 final 클래스와 같이 상속을 막고 꼭 필요한 경우 `open` 키워드로 상속을 허용할 수 있다

```
open class Dog
```

- 하위 클래스에서 상위 클래스를 확장하려면 클래스 뒤에 `:` 을 추가하고 상위 클래스를 작성합니다

```
class Bulldog : Dog()
```

- 함수나 프로퍼티를 재정의할때도 마찬가지로 `open` 키워드로 오버라이드에 대해 허용해야 한다

```
open class Dog {
    open var age: Int = 0

    open fun bark() {
        println("멍멍")
    }
}

class Bulldog : Dog() {

    override var age: Int = 0

    override fun bark() {
        println("컹컹")
    }
}

fun main() {
```

```
    val dog = Bulldog(age = 2)
    println(dog.age)
    dog.bark()
}
```

- 프로퍼티는 기본 생성자를 사용해 오버라이드할 수 있다

```
open class Dog(open var age: Int = 0) {

    open fun bark() {
        println("멍멍")
    }
}

class Bulldog(override var age: Int = 0) : Dog() {

    override fun bark() {
        println("컹컹")
    }
}

fun main() {
    val dog = Bulldog(age = 2)
    println(dog.age)
    dog.bark()
}
```

- `override` 된 함수나 프로퍼티는 기본적으로 open 되어 있으므로 하위 클래스에서 오버라이드를 막기 위해선 final을 앞에 붙인다

```
open class Bulldog(final override var age: Int = 0) : Dog() {

    final override fun bark() {
        println("컹컹")
    }
}

class ChildBulldog : Bulldog() {

    override var age: Int = 0 // 컴파일 오류

    override fun bark() {} // 컴파일 오류
}
```

- 하위 클래스에서 상위 클래스의 함수나 프로퍼티를 접근할땐 `super` 키워드를 사용한다

```
open class Bulldog(final override var age: Int = 0) : Dog() {

    final override fun bark() {
        super.bark()
    }
}
```

- 오늘은 코틀린의 상속에 대해서 살펴봤습니다. 기본이 final 클래스인 점으로 인해 실무에선 몇가지 문제점이 있는데 이후 `자바 프로젝트에 코틀린 도입해보기` 섹션에서 자세히 다룬다

---

3. 추상클래스

- 코틀린은 `abstract` 키워드를 사용해 추상클래스도 제공한다
- 이때 하위 클래스에서 구현해야하는 프로퍼티나 함수 또한 abstract 키워드를 사용한다

```
abstract class Developer {

    abstract var age: Int
    abstract fun code(language: String)

}

class BackendDeveloper(override var age: Int) : Developer() {

    override fun code(language: String) {
        println("I code with $language")
    }
}


fun main() {
    val backendDeveloper = BackendDeveloper(age = 20)
```

```
    println(backendDeveloper.age)
    backendDeveloper.code("Kotlin")
}
```