Chapter 02.코틀린 기초 - 09.인터페이스

인터페이스

1. 인터페이스 선언 방법

- 코틀린의 인터페이스는 `interface` 키워드를 사용해 정의할 수 있다

```
interface Cart
```

- 코틀린의 인터페이스 내부에는 추상 함수와 자바 8의 디폴트 메서드 처럼 구현을 가진 함수를 모두 정의할 수 있다

```
class Product(val name: String, val price: Int)

interface Cart {

    fun add(product: Product)

    fun rent() {
      /* ... */
    }

}
```

- 클래스에서 인터페이스를 구현할때는 `:` 을 붙이고 인터페이스의 이름을 적는다
- 상속과 다른 점은 `()` 생성자 호출이 없다는 것이다

```
class MyCart() : Cart {

  override fun add(product: Product) {
      /* ... */
  }

}
```

---

2. 인터페이스에 프로퍼티 선언

- 코틀린의 인터페이스는 프로퍼티가 존재할 수 있다

```
interface Cart {

    var coin: Int

    val weight: String
        get() = "20KG"


    /* ... */
}


class MyCart(override var coin: Int) : Cart {

  /* ... */
}
```

- MyCart를 구현해보자

```
interface Cart {

    var coin: Int

    val weight: String
        get() = "20KG"

    fun add(product: Product)

    fun rent() {
        if (coin > 0) {
            println("카트를 대여합니다")
        }
    }

}

class MyCart(override var coin: Int) : Cart {

    override fun add(product: Product) {
        if (coin <= 0) println("코인을 넣어주세요")
        else println("${product.name}이(가) 카트에 추가됐습니다")
    }

}

fun main() {
    val cart = MyCart(coin = 100)
    cart.rent()
    cart.add(Product(name = "장난감", price = 1000))
    // 카트를 대여합니다
    // 장난감이(가) 카트에 추가됐습니다

    val cart2 = MyCart(coin = 0)
    cart2.rent()
    cart2.add(Product(name = "장난감", price = 1000))
    // 코인을 넣어주세요
}
```

---

3. 인터페이스 상속

- 코틀린 인터페이스는 상위 인터페이스를 가질 수 있다

```
interface Wheel {

    fun roll()

}

interface Cart : Wheel {

    override fun roll() {
        println("카트가 굴러갑니다")
    }

}


fun main () {
    /* ... */
    cart.rent()
    cart.roll() // 추가
    cart.add(toy)
  // 카트가 굴러갑니다
}
```

---

4. 재정의 충돌 해결

- 클래스는 하나 이상의 인터페이스를 구현할 수 있다 Order엔 add가 구현이 있는 디폴트 함수이고 Cart는 abstract 함수이다
- 이때 동일한 시그니처를 가진 함수가 있는 경우 `super<인터페이스>` 를 사용해 호출할 수 있다

```
interface Order {
    fun add(product: Product) {
        println("${product.name} 주문이 완료되었습니다.")
    }
}


class MyCart(override var coin: Int) : Cart, Order {

    override fun add(product: Product) {
        if (coin <= 0) println("코인을 넣어주세요")
```

```
        else println("${product.name}이(가) 카트에 추가됐습니다")

        // 주문
        super<Order>.add(product)
        // 장난감 주문이 완료되었습니다.
    }
}
```

- 두 인터페이스에 구현을 가진 동일한 디폴트 함수를 사용할 경우 어떻게 될까?

```
interface Cart : Wheel {
  fun printId() = println("1234")
}


interface Order {
  fun printId() = println("5678")
}

// 컴파일 오류가 발생한다
class MyCart(override var coin: Int) : Cart, Order
```

- 하위 클래스에서 직접 구현하도록 컴파일 오류가 발생한다

```
class MyCart(override var coin: Int) : Cart, Order {


    override fun printId() {
        super<Cart>.printId()
//        super<Order>.printId()
    }
}
```

- 하위 클래스에서 직접 구현해서 인터페이스의 디폴트 함수를 사용하면된다