코틀린을 활용한 안드로이드 앱 개발

코틀린 객체지향 프로그래밍 이해
- Inheritance, Abstract

한국기술교육대학교
온라인평생교육원

학습내용

1. 상속의 이해
2. 추상 클래스와 인터페이스의 차이
3. 접근 제한자의 이해

학습목표

1. 상속에 대해 이해할 수 있다.
2. 추상 클래스와 인터페이스의 차이를 구분할 수 있다.
3. 접근 제한자에 대해 이해할 수 있다.

1. 상속의 이해

1 상속(Inheritance)이란?

상속(Inheritance)이란?

- 클래스를 선언할 때 코드에 명시적으로 상위 클래스를 선언하지 않으면 기본으로 Any의 서브 클래스

```
class Shape {
    var x: Int = 0
    var y: Int = 0
    var name: String = "Rect"

    fun draw() {
        println("draw $name : location : $x, $y")
    }
}
fun main(args: Array<String>) {
    val obj1: Any = Shape()
    val obj2: Any = Shape()
    val obj3 = obj1
    println("obj1.equals(obj2) is ${obj1.equals(obj2)}")
    println("obj1.equals(obj3) is ${obj1.equals(obj3)}")
}
```

상속(Inheritance)이란?

- 코틀린에서 클래스는 기본이 final로 선언되어 상속 관계로 하위 클래스를 만들 수 없음
- 상속이 가능하게 하려면 클래스 선언 부분에 open 예약어로 선언해야 함

```
open class Shape {

}
```

- 상속 관계의 표현은 클래스 선언부분에 콜론(:)을 이용하며 콜론 오른쪽에 상위 클래스 생성자 호출구문을 명시

```
class Rect: Shape() {

}
```

1. 상속의 이해

2 오버라이드

- 함수를 선언하면 기본으로 final이 적용
- 함수의 오버라이드를 허용하려면 open 예약어로 명시
- override 예약어를 이용하여 이 함수는 상위 함수를 재정의 한 것임을 명시적으로 선언해야 함

```
open class Shape {
    //.....................
    open fun print() {
        println("$name : location : $x, $y")
    }
}
```

```
class Circle: Shape() {
    //.....................
    override fun print() {
        println("$name : location : $x, $y ... radius : $r ")
    }
}
```

- 오버라이드된 상위 클래스의 멤버에 접근하고자 하는 경우에는 super 예약어를 이용

```
open class Super {
    open var x: Int = 10
    open fun someFun(){
        println("Suer... someFun()")
    }
}
class Sub : Super() {
    override var x: Int = 20
    override fun someFun() {
        super.someFun()
        println("Sub... ${super.x} .... $x")
    }
}
```

1. 상속의 이해

3 형변형(캐스팅)

- 기초 데이터 타입의 캐스팅은 자동 형변형이 안 됨
  - 함수에 의해서만 형변형이 가능

```
val data1: Int = 10
val data2: Double = data1.toDouble()
```

- 상속관계에서 스마트 캐스팅(자동 형변형)
- 하위 타입의 객체가 상위 타입으로 형변형은 스마트 캐스팅

```
open class Super

class Sub1: Super()

//.....
val obj1: Super = Sub1()

val obj2: Sub1 = Super()//error
```

- as를 이용한 캐스팅은 상속관계에 의한 객체의 명시적 캐스팅
- 상위 타입의 객체를 하위 타입으로 캐스팅 하고자 할 때 사용

```
val obj3: Super = Sub1()
val obj4: Sub1 = obj3 as Sub1
obj4.superFun()
obj4.subFun1()
```

2. 추상 클래스와 인터페이스의 차이

1 추상 클래스

- 추상함수(Abstract method)는 미완성 함수 혹은 실행영역을 가지지 않는 함수
- 추상클래스(Abstract Class)는 추상 함수를 가지는 클래스

```
abstract class AbstractTest1 {
    fun myFun1() {
        //.....
    }
    abstract fun myFun2()
}
```

- 추상 클래스는 객체 생성이 불가
- 추상 클래스를 상속받아 하위 클래스를 작성하고 하위 클래스를 객체 생성해서 이용할 수 있음

2 인터페이스

- 인터페이스는 추상 함수를 선언함을 주 목적으로 사용
- 인터페이스는 객체생성이 불가
- 클래스에서 인터페이스를 구현해 이용

```
interface MyInterface {
    var data1: String
    fun myFun1() {
        //.....
    }
    fun myFun2()
}

class MyClass: MyInterface {
    override var data1: String = "hello"
    override fun myFun2() {
        //.....
    }
}
```

2. 추상 클래스와 인터페이스의 차이

2 인터페이스

클래스에서 여러 인터페이스 구현

```
interface MyInterface4 {
    fun myFun4()
}
interface MyInterface5 {
    fun myFun5()
}
class MyClass4: MyInterface4, MyInterface5 {
    override fun myFun4() { }
    override fun myFun5() { }
}
```

클래스에서 상속과 인터페이스 혼용

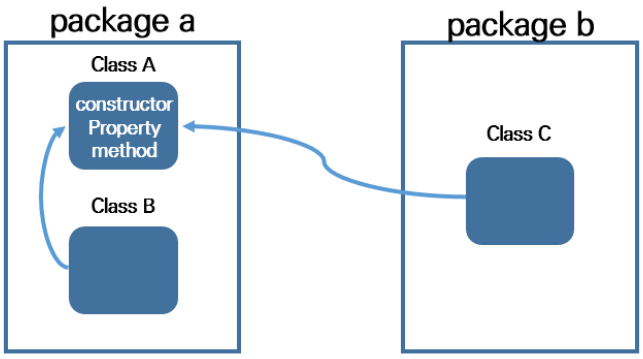
```
interface MyInterface6 {  fun myFun6()    }
interface MyInterface7 {  fun myFun7()    }
open class Super { }
class Sub: Super(), MyInterface6, MyInterface7 {
    override fun myFun6() { }
    override fun myFun7() { }
}
```

3. 접근 제한자의 이해

1 접근 제한자란?

접근 제한자(Visibility Modifier)

- 외부에서 클래스, 생성자, 프로퍼티, 함수 등을 이용할 때 접근의 범위를 지정
- public, internal, protected, private
- 코틀린에서 기본 접근 제한자는 public




```
public class User {
    public constructor(){}

    public val name: String = "kkang"

    public fun myFun(){

    }
}
```

3. 접근 제한자의 이해

2 접근 제한자와 접근 방법

Top-Level 구성요소의 접근범위

- public : (Default) 만약 접근제한자가 명시적으로 선언되지 않는다면 자동으로 public이 적용
  - public은 접근제한이 없다는 의미이기 때문에 어느 곳에서나 접근이 가능
- private : 동일 file 내에서만 접근이 가능
- internal : 같은 module 내에 어디서나 접근이 가능
- protected : top-level에서는 사용 불가능

클래스 멤버의 접근범위

- public : (Default) 만약 접근제한자가 명시적으로 선언되지 않는다면 자동으로 public이 적용
  - public은 접근제한이 없다는 의미이기 때문에 어느 곳에서나 접근이 가능
- private : 동일 클래스내에서만 접근 가능
- protected : private + 서브 클래스에서 사용 가능
- internal : 같은 모듈에 선언된 클래스에서 사용 가능

▪ 정리하기

1. 상속의 이해

Inheritance

- 상속이 가능하게 하려면 클래스 선언 부분에 open 예약어로 선언
- 상속 관계의 표현은 클래스 선언부분에 콜론(:)을 이용, 콜론 오른쪽에 상위 클래스 생성자 호출구문을 명시
- 함수의 오버라이드를 허용하려면 open 예약어로 명시
- override 예약어를 이용하여 이 함수는 상위 함수를 재정의 한 것임을 명시적 선언

2. 추상 클래스와 인터페이스의 차이

Abstract

- 추상클래스(Abstract Class)는 추상 함수를 가지는 클래스

Interface

- 인터페이스는 추상 함수를 선언함을 주 목적으로 사용

▪ 정리하기

3. 접근 제한자의 이해

- public, internal, protected, private 접근제한자 제공
- 코틀린에서 기본 접근 제한자는 public