코틀린을 활용한 안드로이드 앱 개발

코틀린 객체지향 프로그래밍 이해
- Data, Object, Companion class

한국기술교육대학교
온라인평생교육원

학습내용

1. 데이터 클래스에 대한 이해
2. 오브젝트 클래스에 대한 이해
3. 컴패니언 클래스에 대한 이해

학습목표

1. 데이터 클래스에 대해 이해할 수 있다.
2. 오브젝트 클래스에 대해 이해할 수 있다.
3. 컴패니언 클래스에 대해 이해할 수 있다.

1. 데이터 클래스에 대한 이해

1 데이터 클래스란?

- VO(Value-Object) 클래스
- data라는 예약어로 선언되는 클래스
- 주생성자가 선언되어야 하며 주생성자의 매개변수는 최소 하나 이상이 선언되어 있어야 함
- 모든 주생성자의 매개변수는 var 혹은 val로 선언되어야 함
- 데이터 클래스는 abstract, open, sealed, inner 등의 예약어를 추가할 수 없음

```
data class User(val name: String, val age: Int)
```

```
data class User1()//error
data class User2(name: String)//error
data abstract class User3(val name: String)//error
```

1. 데이터 클래스에 대한 이해

2 데이터 클래스의 함수

Equals()

- 객체의 데이터가 같은지에 대한 비교

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

data class User(val name: String, val age: Int)

fun main(args: Array<String>) {

    var product=Product("prod1",100)
    var product1=Product("prod1",100)
    println(product.equals(product1))

    var user=User("kkang",30)
    var user1=User("kkang",30)
    println(user.equals(user1))
}
```

```
data class User(val name: String, val age: Int)
data class Person(val name: String, val age: Int)

fun main(args: Array<String>) {
    val user=User("kkang", 20)
    val person=Person("kkang", 20)
    println(user.equals(person)) //false
}
```

1. 데이터 클래스에 대한 이해

2 데이터 클래스의 함수

- equals 함수에 의한 값의 비교는 주생성자에 선언된 프로퍼티 값만을 비교

```
data class User(val name: String, val age: Int){
    var email: String = "a@a.com"
}
fun main(args: Array<String>) {
    val user = User("kkang", 20)

    val user1 = User("kkang", 20)
    user1.email = "b@b.com"
    println(user.equals(user1))
}
```

toString()

- 데이터 클래스의 데이터를 문자열로 반환하는 함수

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

data class User(val name: String, val age: Int){
    var email: String = "a@a.com"
}

fun main(args: Array<String>) {

    var product=Product("prod1",100)
    println(product.toString())

    var user=User("kkang",30)
    println(user.toString())
}
```

1. 데이터 클래스에 대한 이해

2 데이터 클래스의 함수

실행결과

```
fourteen_four.Product@5e2de80c
User(name=kkang, age=30)
```

componentN()

- 클래스 프로퍼티 값을 획득

```
data class User(val name: String, val age: Int)

fun main(args: Array<String>) {

    var user=User("kkang",30)

    println(user.component1()) //kkang
    println(user.component2()) //30
}
```

1. 데이터 클래스에 대한 이해

2 데이터 클래스의 함수

```
data class User(val name: String, val age: Int)

fun main(args: Array<String>) {
    var user=User(age=30, name="kkang")
    val (name, age) = user

    println("name : $name, age: $age") //name : kkang, age: 30
}
```

2. 오브젝트 클래스에 대한 이해

1 오브젝트를 이용한 익명 내부 클래스 정의

- object { } 형태로 클래스를 선언
- 클래스명이 없지만 선언과 동시에 객체가 생성
- object 클래스에는 생성자를 추가할 수 없음

```
val obj1=object {
    var no1: Int = 10
    fun myFun() {

    }
}

class Outer {
    val obj2 = object {
        var no2: Int = 0
        fun myFun() {

        }
    }
}
```

2. 오브젝트 클래스에 대한 이해

2 타입 명시로 오브젝트 이용

- object 클래스를 만들 때 다른 클래스를 상속 받거나 인터페이스를 구현

```
interface SomeInterface {
    fun interfaceFun()
}
open class SomeClass {
    fun someClassFun(){
        println("someClassFun....")
    }
}
class Outer {
    val myInner: SomeClass =object : SomeClass(), SomeInterface {
        override fun interfaceFun() {
            println("interfaceFun....")
        }
    }
}
fun main(args: Array<String>) {
    val obj=Outer()
    obj.myInner.someClassFun()
}
```

3 오브젝트 선언

- val obj = object { }
- object 클래스명 { }
- 클래스명과 동일한 이름의 객체까지 같이 생성
- object 클래스명 { }은 객체생성 구문

2. 오브젝트 클래스에 대한 이해

4 오브젝트 선언

```
class NormalClass {
    fun myFun(){ }
}
object ObjectClass {
    fun myFun() { }
}
fun main(args: Array<String>) {
    val obj1: NormalClass = NormalClass()
    val obj2: NormalClass = NormalClass()
    obj1.myFun()

    val obj3: ObjectClass = ObjectClass()//error

    ObjectClass.myFun()
}
```

3. 컴패니언 클래스에 대한 이해

1 companion 예약어

- object 클래스의 일종
- object 클래스의 멤버를 outer 클래스의 static 멤버처럼 이용하고자 할 때 사용

```
class Outer {
    object NestedClass {
        val no: Int = 0
        fun myFun() { }
    }
}
fun main(args: Array<String>) {
    val obj=Outer()
    obj.NestedClass.no//error

    Outer.NestedClass.no
    Outer.NestedClass.myFun()
}
```

```
class Outer {
    companion object NestedClass {
        val no: Int = 0
        fun myFun() { }
    }
    fun myFun(){
        no
        myFun()
    }
}
fun main(args: Array<String>) {
    Outer.NestedClass.no
    Outer.NestedClass.myFun()

    Outer.no
    Outer.myFun()
}
```

▪ 정리하기

1. 데이터 클래스에 대한 이해

Data class

- data라는 예약어로 선언되는 클래스
- 주생성자가 선언되어야 하며 주생성자의 매개변수는 최소 하나 이상 선언 필수
- equals(), toString(), componentN() 등의 함수 도움으로 데이터 핸들링 가능

2. 오브젝트 클래스에 대한 이해

Object class

- object { } 형태로 클래스를 선언
- 클래스명이 없지만 선언과 동시에 객체가 생성

3. 컴패니언 클래스에 대한 이해

Companion class

- object 클래스의 멤버를 outter 클래스의 static 멤버처럼 이용하고자 할 때 사용