Scala companion object 이해하기

myjlms99@gmail.com

Object 이해하기

오브젝트 특징

오브젝트는 클래스와 인스턴스가 하나인 싱글턴을 만든다.

클래스의 정적 속성과 메소드가 없으므로 오브젝트를 이용해서 인스턴스를 만들지 않고 실행이 가능하다.

별도의 메인함수를 지정해서 실제 실행하는 입구로 만들 수 있다.

싱글턴이므로 기본생성자나 매개변수를 지정할 수 없다.

싱글턴 객체 생성

Object를 이용해서 생성하면 하나의 인스턴스를 가진 싱글턴 인스턴스가 만들어진다.

내부에 정의된 부분을 인스턴스로 직접 사용하는 것과 동일

```
scala> object AB {
     |     var x : Int = 0
     |     def getX = x
     | }
defined object AB

scala> AB.x
res78: Int = 0

scala> AB.x = 100
AB.x: Int = 100

scala> AB.getX
res79: Int = 100
```

추상클래스 상속하기

추상클래스를 object에서 상속하고 추상 속성을 재정의해서 apply로 처리

```
abstract class ABC {
   val x : Int
}
object ABC extends ABC {
   val x : Int = 100
   def apply() = x
}

// Exiting paste mode, now interpreting.

defined class ABC
defined object ABC

scala> ABC()
res98: Int = 100
```

Companion object

Static 처리 필요

scala 언어는 class 내부에 정의된 모든 것은 인스턴스에서 사용하는 필드와 메소드만을 제공한다.

클래스로 접근해서 처리할 수 있는 필드와 메소드 처리가 필요한 경우 Companion object를 만들어야 한다.

동일한 모듈에 class와 object를 정의해야 하고 apply 메소드를 이용해서 생성 메소드를 작성할 수 있다.

Companion 객체 내부에 전달된 companion 클래스의 인스턴스를 이용해서 private에도 접근이 가능하다.

컴패니언 객체

클래스를 접근해서 정적 속성과 메소드를 처리하기 위해서는 class와 object를 가지 정의하면 실제 클래스 정적 처리처럼 실행

Class와 object를 동시 정의

Class 명으로 접근해서 처리

인스턴스에서는 정적 처리 불가

```
class A

object A {
   var x : Int = 0
   def setX(a : Int) = x =a
}

// Exiting paste mode, now interpreting.

defined class A
defined object A

scala> A.x
res80: Int = 0

scala> A.setX(10)

scala> A.x
res82: Int = 10

scala> val a = new A
a: A = A@1dd67363

scala> a.x
<console>:14: error: value x is not a member of A
       a.x
         ^
```

생성자 변경

두 가지의 생성자 처리

class와 object를 두개를 지정해서 컴패니언 구조를 만들어지면 실제 이를 가지고 new와 apply로 객체를 생성할 수 있다.

Class와 object를 동시 정의

두 가지 방식으로 객체 생성

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

class ABCD(val name: String)
object ABCD {
   def apply(name:String) = new ABCD(name)
}

// Exiting paste mode, now interpreting.

defined class ABCD
defined object ABCD

scala> new ABCD("Moon")
res96: ABCD = ABCD@3d0496cd

scala> ABCD("Dahl")
res97: ABCD = ABCD@6e9fffdc
```

Apply로 생성자 변경

컴패니언 객체에 apply 메소드로 생성자를 변경해서 컴패니언 클래스 내의 보호 필드에 값을 변경했다. 외부에서는 접근하지 못한다.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

class My(val name : String) {
   private var extra = ""
}
object My {
   def apply(n:String, e:String) = {
      val m = new My(n)
      m.extra = e
m
   }
   def apply(n:String) = new My(n)
}

// Exiting paste mode, now interpreting.

defined class My
defined object My

scala> val m = My("dahl","moon")
m: My = My@18e5987e

scala> m.name
res138: String = dahl

scala> m.extra
<console>:14: error: value extra is not a member of My
       m.extra
         ^
```

컴패니언 객체를 통한 컴패니언 클래스 접근

companion class필드 참조 예외 발생

클래스에 필드를 접근하려고 object에서 메소드에서 정의하면 실제 모듈 변수로 접근해서 처리된다.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

class AA {
   private val aa = 88
}
object AA {
   def getAA = aa
}

// Exiting paste mode, now interpreting.

defined class AA
defined object AA

scala> val aa = new AA
aa: AA = AA@4976f724

scala> aa.aa
<console>:14: error: value aa in class AA cannot be accessed in AA
       aa.aa
          ^

scala> AA.getAA
res133: Aa = Aa@16a97ac0
```

companion class 내의 필드 참조

컴패니언 클래스의 멤버를 접근을 컴패니언 객체에서만 처리할 때는 private 멤버인 속성을 접근 처리한다.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

class AA {
   private val aa = 99
}
object AA {
   def getAA(aa:AA) = aa.aa
}

// Exiting paste mode, now interpreting.

defined class AA
defined object AA

scala> val aa = new AA
aa: AA = AA@314b789f

scala> AA.getAA(aa)
res135: Int = 99
```

컴패니언 클래스 내의 보호 메소드도 컴패니언 객체에서 점 연산으로 접근 처리 => 실제 클래스의 보호속성은 외부에서는 접근이 불가

companion class 내의 메소드 접근을 통한 필드 참조

컴패니언 클래스의 멤버를 접근을 컴패니언 객체에서만 처리할 때는 private 멤버인 메소드를 만들어서 접근 처리한다.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

class AA {
   private val aa = 99
   private def getAA_ = aa
}
object AA {
   def getAA(aa: AA) = aa.getAA_
}

// Exiting paste mode, now interpreting.

defined class AA
defined object AA

scala> val aa = new AA
aa: AA = AA@5bc2f30c

scala> AA.getAA(aa)
res134: Int = 99
```

컴패니언 클래스 내의 보호 메소드도 컴패니언 객체에서 점 연산으로 접근 처리 => 실제 클래스의 보호속성은 외부에서는 접근이 불가

companion class 내의 메소드 참조

클래스를 정의하고 컴패니언 object를 만들때 상속으로 처리한다.

Console println으로 하면 현재 창에 결과가 출력된다.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

class Foo {
   def foo = 8
}
object Foo extends Foo {
   def main(args : Array[String]) = {
      Console println foo
   }
}

// Exiting paste mode, now interpreting.

defined class Foo
defined object Foo

scala> Foo.main(null)
8
```

정적 클래스 필드와 메소드

정적 필드와 메소드 사용하기

클래스와 object를 정의한다. Object에 있는 정적 필드를 이용해서 인스턴스를 생성하고 메소드로 조회한다.

컴패니언 객체 내의 필드와 메소드를 이용해서 정적 처리

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

class MM(var size:String)
object MM {
   val SIZE_S = "small"
   val SIZE_L = "large"
   def getMM(mm:MM) = mm.size
}

// Exiting paste mode, now interpreting.

defined class MM
defined object MM

scala> val ms = new MM(MM.SIZE_S)
ms: MM = MM@4bbc76d0

scala> MM.getMM(ms)
res141: String = small
```

컴패니언 클래스에서 컴패니언 객체의 멤버 접근하기

컴패니언 객체 멤버 접근하기

클래스에서 object에 정의된 속성을 참조하는 메소드를 정의할 때 import 문을 사용해서 꼭 내부에서 호출한다.

```
scala> :pa
// Entering paste mode (ctrl-D to finish)

object CO {
   val aco = " Co"
}
class CO {
   import CO._
   def getACO = aco
}

// Exiting paste mode, now interpreting.

defined object CO
defined class CO

scala> val co = new CO
co: CO = CO@34110da6

scala> co.getACO
res142: String = " Co"
```