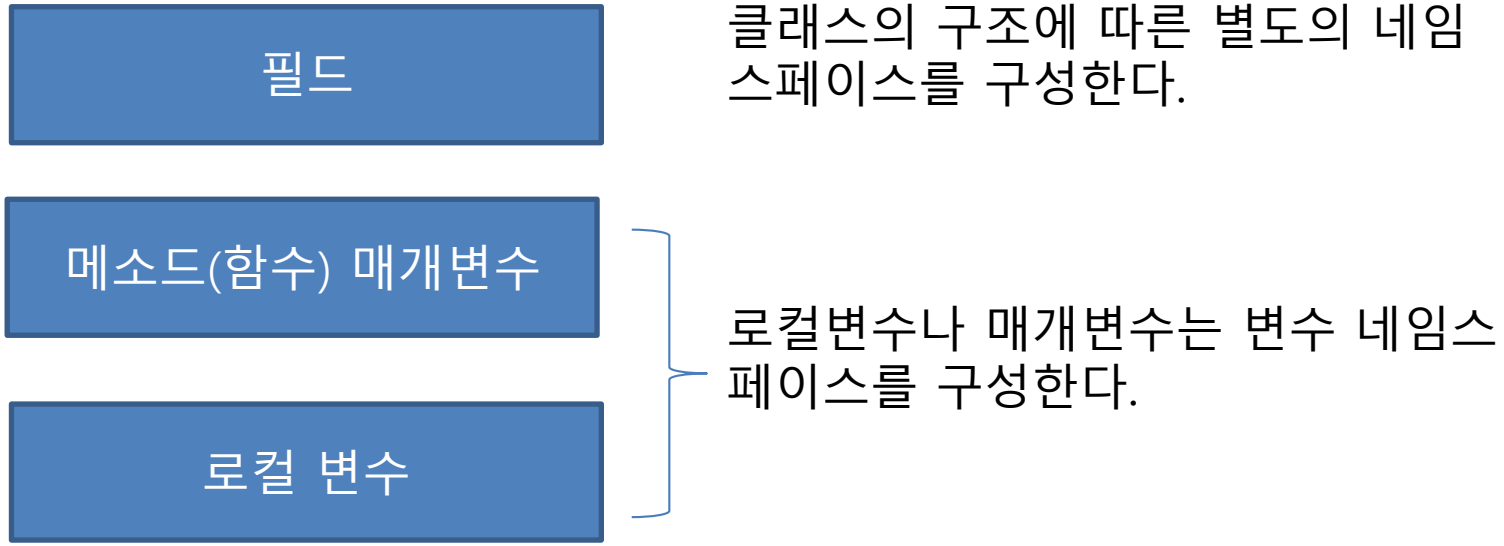
Scala
namespace 이해하기

myjlms99@gmail.com

Scope

변수 관리기준

스칼라 변수는 선언 된 위치에 따라 세 가지 범위로 분류됩니다. 필드, 메소드 매개 변수 및 로컬 변수입니다. 우리가 이것들을 하나씩 토론합시다.



필드 field

클래스, 객체, trait 등에 선언된 변수를 말하면 액세스 수정 자 유형에 따라 객체의 모든 메소드와 객체 외부에서 액세스 할 수 있고, var 및 val 키워드에 따라 변경 가능하거나 변경 불가능할 수 있다.

클래스에 정의된 필드는 기본 public이다

```
scala> class I {
     |    val x = 100
     | }
defined class I

scala> val i = new I
i: I = I@2ddb44da

scala> i.x
res59: Int = 100
```

메소드 매개변수 : 메소드

메소드가 호출 될 때마다 메소드 내부의 값을 전달하는 데 사용되는 변수

```
scala> class Sub {
     |     def minus(s1:Int, s2:Int) = {
     |         s1 - s2
     |     }
     | }
defined class Sub

scala> val sm = new Sub
sm: Sub = Sub@776522a0

scala> sm.minus(100,100)
res46: Int = 0
```

메소드 매개변수 : 클래스

일반 클래스를 정의할 때 사용되는 매개변수는 함수에서 바로 접근해서 사용할 수 있다.

스칼라 클래스도 매개변수로 정의가 가능하고 이를 내부 메소드에서 바로 접근 가능

```
scala> class Add(s1:Int, s2:Int) {
     |     def plus = s1 + s2
     | }
defined class Add

scala> val a = new Add(10,10)
a: Add = Add@23eba65c

scala> a.plus
res49: Int = 20
```

로컬 매개변수

지역 변수는 함수, 메소드 내부에서 선언된 변수입니다. 메소드 내에서만 접근 할 수 있다. var 및 val 키워드를 사용

함수, 메소드 내에 정의된 변수는 외부에서 참조가 되지 않는다.

```
scala> def a = { val x = 100; println(x)}
a: Unit

scala> a
100

scala> a.x
<console>:21: error: value x is not a member of Unit
       a.x
         ^
```

매개변수를 로컬변수로 지정

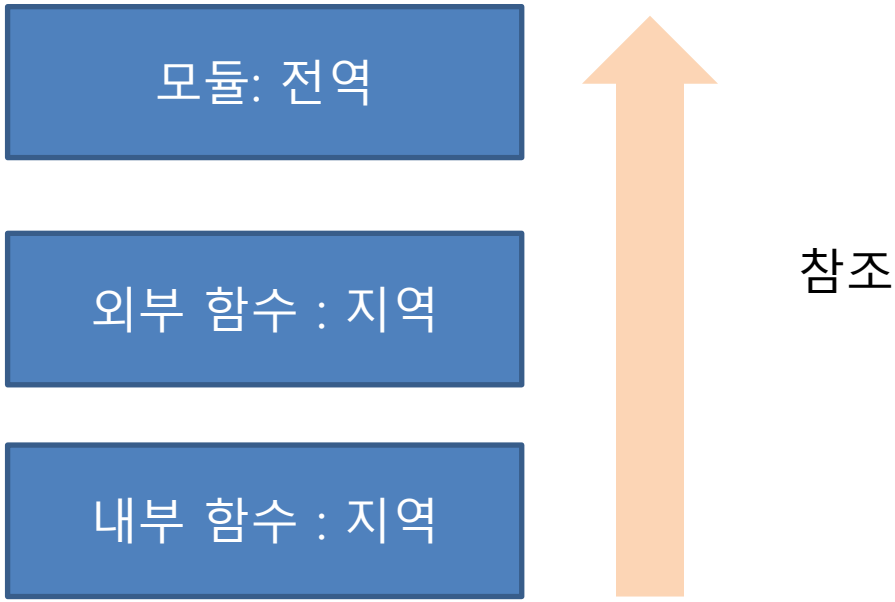
매개변수 이름으로 로컬변수로 지정하면 에러가 발생한다. 동일한 이름에 대한 체크를 해서 재정의를 불가하게 한다.

```
scala> def mul(x:Int) = {
     |    val x : Int = x*3
     | }
<console>:21: error: forward reference extends over definition of value x
         val x : Int = x*3
                       ^


scala> def mul(x:Int) = {
     |    val y = x * 3
     |    y
     | }
mul: (x: Int)Int

scala> mul(3)
res53: Int = 9
```

지역변수 네임스페이스

지역 네임스페이스 참조 기준

변수에 대한 참조는 자기 자신의 네임스페이스부터 검색하며 없으면 상위로 검색한다. 최종으로 없으면 예외를 발생시킨다.



모듈 네임스페이스 참조

함수를 매개변수 없이 정의했다. 내부 로직에 있는 변수는 모듈에서 찾아서 처리한다.

함수는 로컬 변수와 매개변수를 먼저 처리한다. 없으면 모듈을 검색한다.

```
scala> val x = 100
x: Int = 100

scala> val y = 100
y: Int = 100

scala> def add = {x+y}
add: Int

scala> add
res105: Int = 200
```

내부함수에서 외부 함수 참조

외부함수를 정의하고 내부함수는 매개변수 없이 외부함수에 정의된 로컬변수와 매개변수를 가지고 처리한다.

내부 함수에는 별도의 선언이 없으므로 모든 변수를 외부함수에서 검색해서 처리한다.

```
scala> def outer(x:Int, y:Int) = {
     |   val z = 100
     |    def inner = x+y+z
     |    inner
     | }
outer: (x: Int, y: Int)Int

scala> outer(10,20)
res106: Int = 130
```

내부함수에서 모듈 참조

외부함수를 정의하고 내부함수는 매개변수 없이 모듈에 정의된 로컬변수와 외부 함수 매개변수를 가지고 처리한다.

내부 함수에는 별도의 선언이 없으므로 모든 변수를 외부함수에서 1차로 검색하고 없으면 모듈에서 2차 검색한다.

```
scala> val z = 100
z: Int = 100

scala> def outer(x:Int, y:Int) = {
     |     def inner = x + y + z
     |     inner
     | }
outer: (x: Int, y: Int)Int

scala> outer(10,20)
res107: Int = 130
```

전역 변수 변경 처리

함수 내에서 모듈 변수 즉 전역 변수를 변경하려면 var로 지정된 변수일 때만 가능하다.

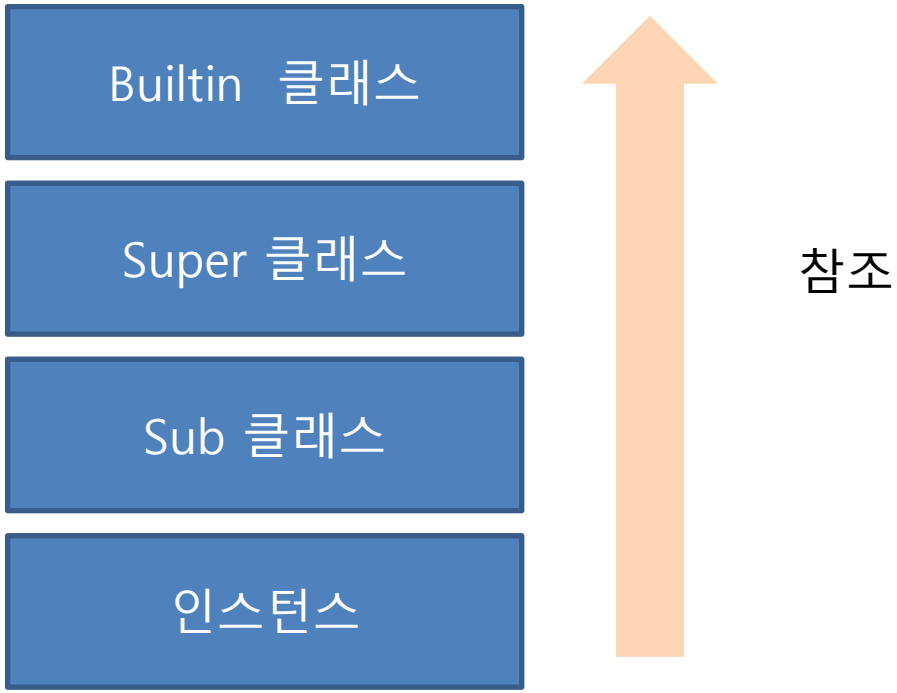
Var 로 지정된 변수만 재할당이 되므로 내부 값이 변경될 수 있다.

```
scala> val z = 300
z: Int = 300

scala> def add = { z = z+300 }
<console>:13: error: reassignment to val
       def add = { z = z+300 }
                   ^


scala> var a = 300
a: Int = 300

scala> def add = { a = a+ 300 }
add: Unit

scala> add

scala> a
res109: Int = 600
```

필드 네임스페이스

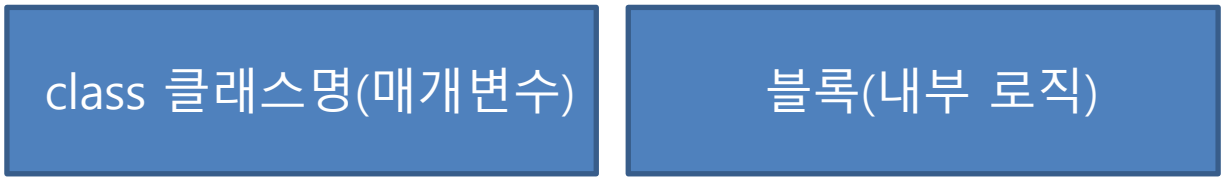
필드 네임스페이스 참조 기준

인스턴스부터 자기 자신을 참조하고 없으면 상위 클래스를 검색한다. 검색결과가 없으면 최종적으로 예외를 발생한다.



클래스 정의

클래스를 정의할 때 내부 블록에 정의된 것을 객체 네임스페이스로 선언된다. 외부에서 클래스의 내부는 인스턴스 없이 참조가 불가능하다.



클래스 정의 후 인스턴스로 내부 접근

하나의 클래스를 정의하고 인스턴스를 만들어 필드를 접근하면 그 값을 조회할 수 있다.

```
scala> class AA {
     |    val a = 10
     | }
defined class AA

scala> val aa = new AA
aa: AA = AA@2e97ac7d

scala> aa.a
res110: Int = 10

scala> aa.getClass
res111: Class[_ <: AA] = class AA
```

슈퍼클래스 상속하고 참조하기

하나의 클래스를 정의하고 인스턴스를 만들어 필드를 접근하면 그 값을 조회할 수 있다.

슈퍼 클래스는 extends로 상속해서 서브 클래스로 인스턴스를 만들고 슈퍼 클래스 내의 필드를 검색한다.

```
scala> class AA {
     |     val a = 10
     | }
defined class AA

scala> class BB extends AA
defined class BB

scala> val bb = new BB
bb: BB = BB@2db4f8d3

scala> bb.a
res114: Int = 10
```

trait 상속하고 참조하기

하나의 trait를 정의하고 인스턴스를 만들어 필드를 접근하면 그 값을 조회할 수 있다.

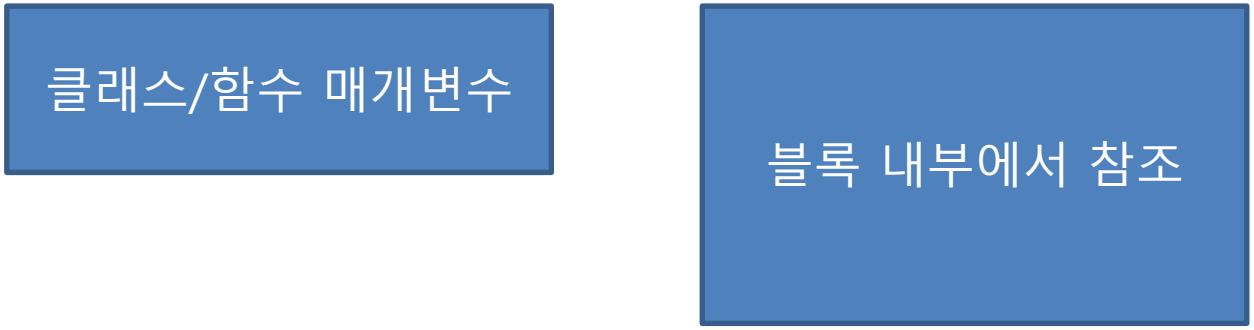
슈퍼 클래스 대신 trait를 사용해도 처리하는 것은 동일하다.

```
scala> trait Able {
     |     val a = 100
     | }
defined trait Able

scala> class CC extends Able
defined class CC

scala> val cc   = new CC
cc: CC = CC@696840fb

scala> cc.a
res118: Int = 100
```

매개변수 네임스페이스

매개변수 네임스페이스

매개변수는 클래스나 함수에서 정의할 수 있다. 매개변수는 해당 클래스나 함수의 블록 내에서만 참조가 제한된다.



클래스 매개변수를 필드에 할당

클래스 정의할 때 val,var로 정의되지 않는 정의는 필드가 아닌 매개변수로 인식한다.

클래스에도 매개변수 지정이 가능하다. 내부 블록 내에서 참조해서 사용이 가능

```
scala> class AA(n:Int) {
     |    val a = n
     | }
defined class AA

scala> val aa = new AA(10)
aa: AA = AA@15442140

scala> aa.n
<console>:14: error: value n is not a member of AA
       aa.n
          ^


scala> aa.a
res116: Int = 10
```

클래스 매개변수를 메소드 처리

클래스 정의할 때 매개변수를 지정했다. 이 매개변수는 메소드 블록에서 호출해서 사용이 가능하다

메소드 블록에서 매개변수를 참조해서 사용이 가능

```
scala> class AA(n:Int) {
     |     def getN = n
     | }
defined class AA

scala> val aa = new AA(10)
aa: AA = AA@3cf198b8

scala> aa.getN
res117: Int = 10
```

메소드 내의 참조 방식

메소드는 필드가 없으면 모듈 내부 참조

모듈 변수와 클래스를 정의하고 메소드에 모듈 변수를 검색해서 반환하게 했다. 모듈에 있는 변수를 조회해서 처리한다.

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

val a = 800
class AAA {
   def getA = a
}

// Exiting paste mode, now interpreting.

a: Int = 800
defined class AAA

scala> val a = new AAA
a: AAA = AAA@79c1af45

scala> a.getA
res128: Int = 800
```

메소드는 필드가 있면 클래스 참조

모듈 변수와 클래스 필드가 동일할 경우 메소드에서 특정 필드를 참조하면 내부에 정의된 필드를 읽고 처리한다.

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

val a = 99
class AAA {
   val a = 77
   def getA = a
}

// Exiting paste mode, now interpreting.

a: Int = 99
defined class AAA

scala> val a = new AAA
a: AAA = AAA@33a6ce4a

scala> a.a
res129: Int = 77

scala> a.getA
res130: Int = 77
```

Companion object 내에서 companion class 참조

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
```