Scala
변수
이해하기

myjlms99@gmail.com

리터럴, 표현식

Literal

숫자, 문자, 텍스트 등 소스코드에 바로 사용하는 데이터

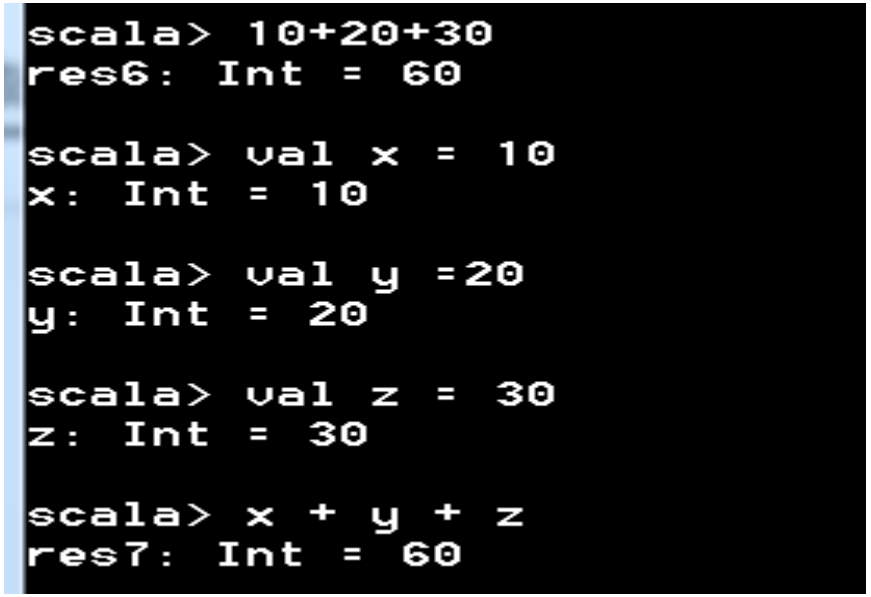
스칼라 내부 지원하는 값을 표현하는 클래스는 리터럴로 바로 사용 가능

```
scala> 1
res41: Int = 1

scala> "Hello"
res42: String = Hello

scala> 'c'
res43: Char = c
```

expression

리터럴과 연산자 등이 하나로 묶어 하나의 결과값을 나타내는 식으로 표현된 것

표현식은 평가가 되면 값인 리터럴로 반환된다.



Expression block

표현식 블록은 한줄 이상의 코드를 중괄호인 brace({ })로 묶어서 처리한다.

표현식 블록에 마지막은 반환해야 할 값을 처리한다.

```
scala> { val a = 10
     |   val b = 100
     |    a*b
     | }
res13: Int = 1000
```

변수란

Variable

하나의 식별자이면서 특정한 값을 특정 네임스페이스에서 관리한다

```
val  식별자 : 타입 = 표현식
```

```
var  식별자 : 타입 = 표현식
```

val로 변수 지정

일반적인 변수는 변수명과 타입을 정의하고 값을 할당해야 한다.

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

scala> x.getClass
res60: Class[Int] = int
```

var로 변수 지정

var로 변수를 정의할 때도 변수명과 타입을 정의하고 값을 할당해야 한다.

```
scala> var y : Int = 100
y: Int = 100

scala> y.getClass
res61: Class[Int] = int
```

변수 명명 규칙

명명 규칙

명명규칙에 대한 관행은 첫글자를 소문자나 대문자로 지정하고 두번째 단어는 카멜 표기법을 사용한다.

- 변수나 함수는 소문자부터 관례상 시작한다
- 상수는 전부 대문자부터 시작한다
- 클래스, object, trait도 대문자부터 시작한다.

변수 명명 규칙 1

첫번째에는 무조건 문자가 와야 한다.

하나의 문자 다음에는 아무것도 없거나 하나 이상의 문자 또는 숫자가 뒤따라온다

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

scala> val a1 = 200
a1: Int = 200

scala> val 1a = 300
<console>:1: error: Invalid literal number
       val 1a = 300
           ^
```

변수 명명 규칙 2

문자, 숫자, 기호를 조합하는 규칙을 사용하지만 중간에 언더스코어로 구분을 줄 수 있다.

하나의 문자 뒤에 문자,숫자, 기호가 오고 언더스코어(_)를 붙이고 다시 문자,숫자, 기호가 온다.

```
scala> val a_1_a = 300
a_1_a: Int = 300

scala> val a_a_1 = 400
a_a_1: Int = 400
```

변수 명명 규칙 3

연산기호만을 조합해서 변수로 지정할 수 있다.

하나 또는 그 이상이 연산 기호

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

scala> val --- = 400
---: Int = 400
```

변수 명명 규칙 4

키워드나 기존에 사용하는 이름을 변수로 지정할 때 사용

하나 또는 그 이상의 문자를 한 쌍의 역인용부호(backquote `)로 표시

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

scala> `def`
res8: Int = 100
```

타입 지정 규칙

Type

작업하는 데이터의 종류로 데이터의 정의 또는 분류를 의미하며 모든 데이터는 특정 타입에 대응하며 이 데이터를 처리하는 클래스를 가지고 있다.

val 변수명 다음에 타입을 지정한다.

```
scala> val x : Int = 1
x: Int = 1

scala> var y : Int = 2
y: Int = 2

scala> y = y + x
y: Int = 3
```

타입 확인하기

변수가 만들어지면 어떤 타입이 들어있는 지를 확인하기 위해서는 getClass 메소드로 확인할 수 있다.

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

scala> a.getClass
res14: Class[Int] = int
```

클래스나 trait로 타입 지정

클래스나 trait를 변수명 다음에 지정해서 타입을 확정할 수 있다.

```
scala> trait AAA {
     |     def print = println("AAA")
     | }
defined trait AAA

scala> class BBB extends AAA
defined class BBB

scala> val a : AAA = new BBB
a: AAA = BBB@dd909bb

scala> a.print
AAA

scala> val b : BBB = new BBB
b: BBB = BBB@3eb53655

scala> b.print
AAA
```

Trait가 타입이 되려면

trait가 정의되고 상속이 되거나 trait를 이용해서 익명이 객체를 만들때 사용된다.

```
scala> trait NoType {
     |    def print = println("NoType")
     | }
defined trait NoType
```

```
scala> val a:  NoType = NoType
<console>:12: error: not found: value NoType
       val a:  NoType = NoType
                        ^


scala> val a:  NoType = new NoType { val c = 0 }
a: NoType = $anon$1@5af2f934
```

Object도 타입이 아니다.

object도 하나의 싱글턴 인스턴스이므로 타입이 될 수 없다.

```
scala> object Type {
     |     val a = "Type"
     | }
defined object Type

scala> val a : Type = Type
<console>:12: error: not found: type Type
       val a : Type = Type
               ^

scala> val a :Type = new Type { val b = 1}
<console>:11: error: not found: type Type
       val a :Type = new Type { val b = 1}
              ^
<console>:11: error: not found: type Type
       val a :Type = new Type { val b = 1}
                        ^

scala> Type.a
res11: String = Type
```

함수 타입 지정하기

함수 타입 지정 규칙

함수 타입은 함수의 시그너처 타입만을 가지고 정의한다.

(함수 매개변수의 타입) => 함수결과값 타입

함수도 변수에 할당할 수 있다.

함수를 정의하고 변수에 할당하려면 타입과 결과값이 동일해야 한다.

```
scala> def add(x:Int, y:Int) : Int = x+y
add: (x: Int, y: Int)Int

scala> val a : (Int, Int) => Int = add
a: (Int, Int) => Int = $$Lambda$1525/1692284740@5353b529

scala> a(10,10)
res15: Int = 20

scala> add(10,10)
res16: Int = 20
```

인자없고 반환값 없는 함수 할당

인자가 없고 반환값이 없는 lambda 함수를 변수에 할당한다. 이를 실행하면 lambda 함수가 실행된다.

```
scala> val b = () => println("no Parameter, no Return")
b: () => Unit = $$Lambda$1529/464268759@1289e4ea

scala> b
res17: () => Unit = $$Lambda$1529/464268759@1289e4ea

scala> b()
no Parameter, no Return
```

Assignments

재할당 가능 여부

val로 정의하면 한번 할당하면 재할당이 불가하다. Var 정의해서 재할당을 해야 한다.

재할당될 때도 동일한 타입 일 경우 할당이 된다.

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

scala> d = 200
<console>:20: error: reassignment to val
       d = 200
         ^

scala> var e : Int = 100
e: Int = 100

scala> e = 300
e: Int = 300
```

여러 변수를 한번에 할당

여러 변수를 지정할 때는 괄호를 사용해서 묶고 데이터도 괄호로 묶어서 보내주면 된다.

```
scala> val (a:Int, b:Double) = (12, 3.14)
a: Int = 12
b: Double = 3.14

scala> a
res44: Int = 12

scala> b
res45: Double = 3.14

scala> val (c, d) = (12, 3.14)
c: Int = 12
d: Double = 3.14
```

타입 추론 /
타입 상향 전환

변수 지정할 때 타입 추론

정적 타입을 체크하기 위해서는 변수에 타입을 정의해야 하지만 변수에 정의된 값을 보고 추론 기능도 지원을 해서 값을 보고 타입을 정한다.

정적 타입을 체크하므로 추론된 타입에 대해서는 변경이 불가하다.

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

scala> x = 10.d
<console>:20: error: value d is not a member of Int
       x = 10.d
              ^
```

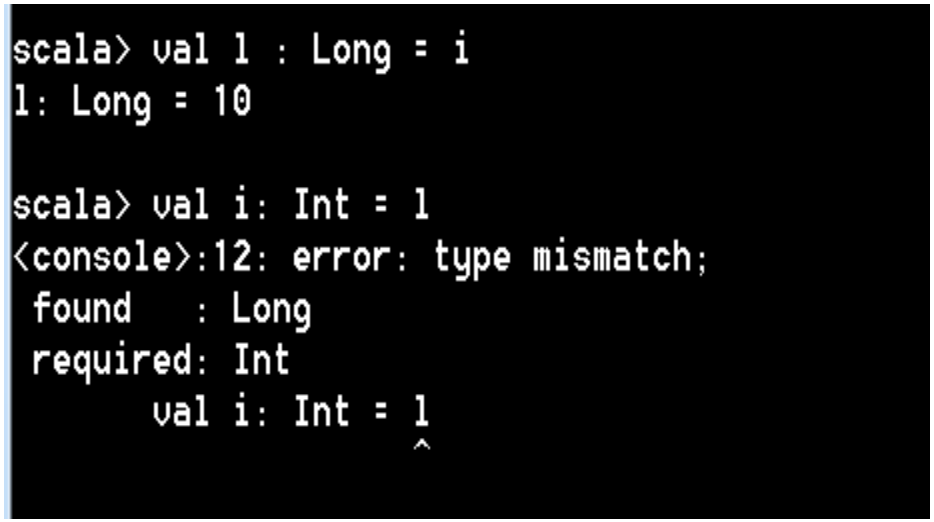
상향전환 : upconverted

숫자형 데이터 타입일 경우는 다른 타입이라도 상향 타입을 지정할 경우는 자동으로 변환된다.

```
scala> val b: Byte = 10
b: Byte = 10

scala> val s:Short = b
s: Short = 10

scala> val i : Int = s
i: Int = 10

scala> val d : Double = i
d: Double = 10.0

scala> val f : Float = i
f: Float = 10.0

scala> val l : Long = i
l: Long = 10
```

하향전환 : downconverted

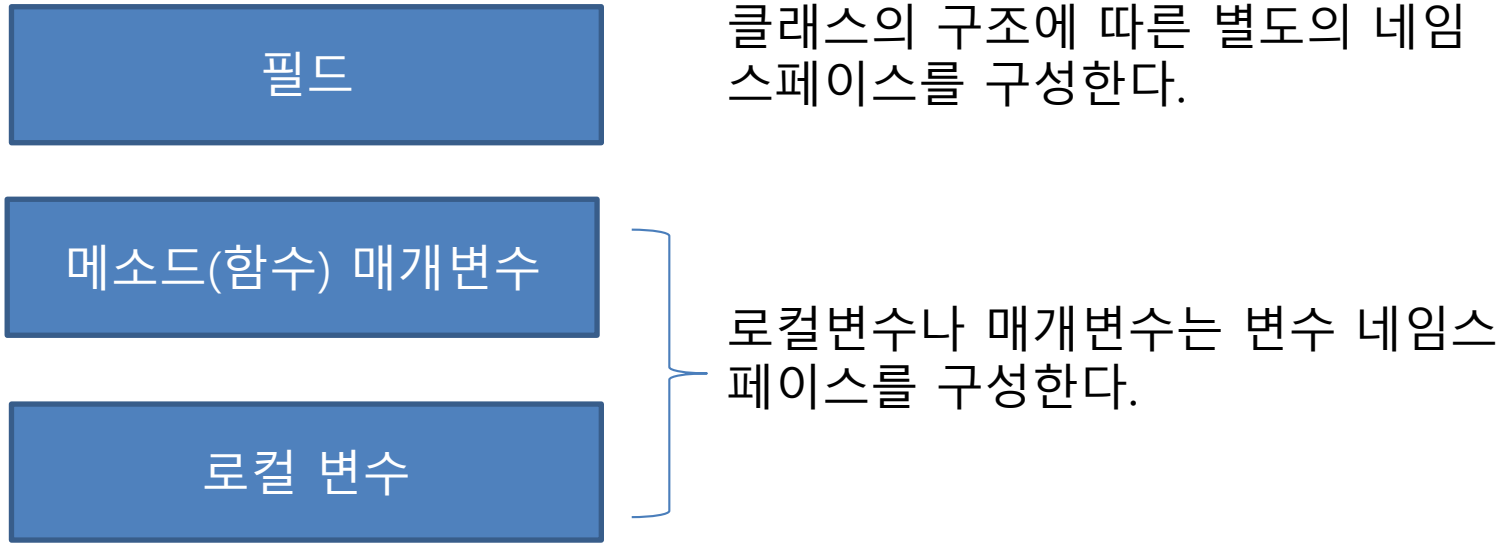
숫자형 데이터 타입도 상향전환은 되지만 더 큰 타입에서 작은 타입으로는 전환이 되지 않는 것을 볼 수 있다.



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
```

변수 평가 기준

지연 평가

lazy val로 변수를 정의하고 블록으로 값을 할당하면 정의할 때 할당이 없고 이 변수를 호출할 때 실제 값이 할당되는 것을 알 수 있다.

```
scala> lazy val y = {println("Lazy"); 100}
y: Int = <lazy>

scala> y
Lazy
res52: Int = 100
```

한번 호출되면 항상 동일한 값을 가지고 처리된다.

즉시 평가/지연 평가

var나 val 로 정의한 변수는 즉시 평가되어 변수가 확정 된다. Lazy 키워드를 이용하면 호출될 때 평가된다.

```
scala> object ABC {
     |    println(" val evaluation")
     |    val x = 100
     |    println(" val x " + x)
     |    lazy val y = 300
     |    def getY = { println(" Lazy"); y }
     | }
defined object ABC

scala> ABC.x
 val evaluation
 val x 100
res50: Int = 100

scala> ABC.getY
 Lazy
res51: Int = 300
```

Lazy로 지정한 것은 object 가 실행할 대 실제 값이 평가되지 않는다.

필드 접근 제어

private 지정하기

클래스나 object 내에서만 사용하는 필드를 만들기 위해 필드 정의 앞에 private을 사용한다.

```
scala> object MUL {
     |    private val x = 100
     |    private val y = 200
     |    def mul = x * y
     | }
defined object MUL

scala> MUL.x
<console>:21: error: value x in object MUL cannot be accessed in object MUL
       MUL.x
           ^


scala> MUL.mul
res57: Int = 20000
```

private으로 정의하면 외부에서 접근할 수 없다.

private[this] 사용하기 1

동일한 객체에서만 접근해서 사용하기 위해서는 private[this]를 정의하면 다른 인스턴스에서는 직접 접근할 수 없다.

```
scala> class DIV(a:Int, b:Int) {
     |    private[this] val x = a
     |    private[this] val y = b
     |    def divX(obj:DIV) = x / obj.x
     | }
<console>:25: error: value x is not a member of DIV
         def divX(obj:DIV) = x / obj.x
                                    ^
```

private[this] 사용하기 2

다른 인스턴스에서 필드를 접근하기 위해 별도의 필드를 접근하는 메소드가 필요하다.

```
scala> class DIV(a:Int, b:Int) {
     |    private[this] val x = a
     |    private[this] val y = b
     |    def divX(obj:DIV) = x / obj.getX
     |    def getX = x
     | }
defined class DIV

scala> val d = new DIV(10,10)
d: DIV = DIV@4d127509

scala> val d2 = new DIV(2,2)
d2: DIV = DIV@4a7d2281

scala> d.divX(d2)
res58: Int = 5
```

추상 필드/ 구현 필드

추상필드

trait나 추상클래스에 추상 필드를 지정할 수 있다. 이를 구현되는 클래스에 구현 클래스로 재정의해야 한다.

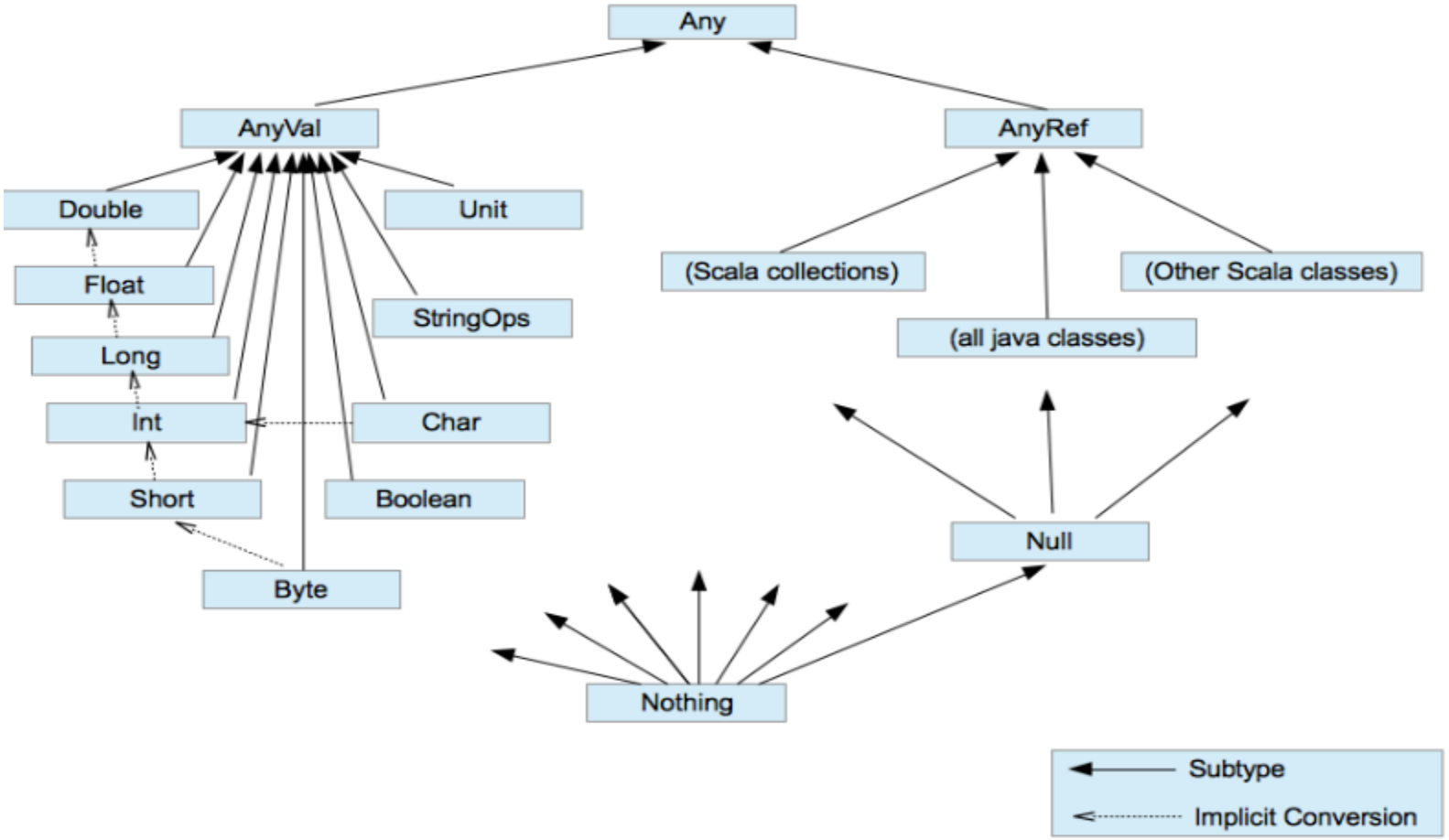
Trait에 필드에 타입까지만 정하면 추상필드가 된다.

```
scala> trait AAA {
     |     val x : Int
     |     var y : Int
     | }
defined trait AAA

scala> class ADD(a:Int, b:Int) {
     |     val x = a
     |     val y = b
     | }
defined class ADD

scala> val a = new ADD(10,100)
a: ADD = ADD@1fce6419

scala> a.x
res54: Int = 10

scala> a.y
res55: Int = 100
```

내장 타입 계층 구조

타입 계층 구조

최상위 구조와 최하위 구조가 있다.



최상위 계층에 하위타입 할당

변수에 타입을 지정할 는 것은 정의된 타입과 그 하위 타입의 인스턴스를 할당하면 처리될 수 있는 구조를 만든다.

```
scala> val a : Any = 1
a: Any = 1

scala> val b : AnyVal = 1
b: AnyVal = 1

scala> val c : AnyRef = "String"
c: AnyRef = String
```

내장 타입 변환

내장 타입 변환

내장 클래스들은 대부분 내장 타입으로 전환하는 메소드를 제공한다.

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

scala> val b = a.toString
b: String = 100

scala> val c = a.toD
toDegrees   toDouble

scala> val c = a.toDouble
c: Double = 100.0
```