Scala type class pattern

myjlms99@gmail.com

암묵적과 명시적 처리 알아보기

매개변수 처리 기준

명시적일 경우는 함수에 초기값을 주지만 암묵적인 경우 implicit로 지정한 값을 내부적으로 처리한다.

```
scala> def add(x:Int, y:Int=10) = x+y
add: (x: Int, y: Int)Int

scala> add(10)
res0: Int = 20

scala> def addI(x:Int)(implicit y:Int) = x+y
addI: (x: Int)(implicit y: Int)Int

scala> addI(10)(10)
res1: Int = 20

scala> addI(10)
<console>:13: error: could not find implicit value for parameter y: Int
       addI(10)
            ^


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

scala> addI(10)
res3: Int = 20
```

명시적 trait 상속 처리

Object 이해하기

object 키워드를 이용해서 하나의 싱글턴 객체를 만들어서 처리한다. 하나의 클래스와 하나의 인스턴스만이 만들어진다.

```
scala> object AC {
     |    val a = (x:Int, y:Int) => x+y
     | }
defined object AC

scala> AC
res8: AC.type = AC$@37d43b9b

scala> AC.a
res9: (Int, Int) => Int = AC$$$Lambda$1275/283282839@5331a22c

scala> AC.a(5,5)
res10: Int = 10

scala> AC.getClass
res11: Class[_ <: AC.type] = class AC$
```

Trait를 object로 상속해서 처리

trait에 추상 메소드를 지정하고 이를 상속해서 구현한 두 개의 object를 만들고 실행한다.

```
scala> trait NumberLike[T] {
     |    def plus(x:T, y:T) : T
     | }
defined trait NumberLike

scala> object NumberLikeDouble extends NumberLike[Double] {
     |    def plus(x: Double, y : Double) : Double = x+y
     | }
defined object NumberLikeDouble

scala> object NumberLikeInt extends NumberLike[Int] {
     |    def plus(x:Int, y:Int) : Int = x+y
     | }
defined object NumberLikeInt

scala> NumberLikeDouble.plus(10,10)
res12: Double = 20.0

scala> NumberLikeInt.plus(10,10)
res13: Int = 20
```

Trait 용 동반 객체 생성

Trait와 동반객체 정의

trait와 동반 객체에 특별한 타입매개변수 없이 정의하고 동일한 이름으로 호출해서 object 내의 메소드를 호출해도 처리가 된다.

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

trait Adder {
   def plus(x:Int, y:Int) : Int
}
object Adder {
   def plus(x:Int, y:Int) : Int = x+y
}

// Exiting paste mode, now interpreting.

defined trait Adder
defined object Adder

scala> Adder.plus(10,10)
res50: Int = 20
```

타입 매개변수를 가진 Trait와 동반 객체

trait와 동반 객체를 정의하고 내부에 동반 객체의 익명 인스턴스를 만들어 처리한다.

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

trait Show[A] {
   def show(a:A) : String
}
object Show {
   val intCanShow : Show[Int] = new Show[Int] {
      def show(int:Int) : String = s"int $int "
}
}
```

```
defined trait Show
defined object Show

scala> Show.intCanShow
res3: Show[Int] = Show$$anon$1@6530a1de


scala> Show.intCanShow.show(20)
res5: String = "int 20 "
```

익명 객체 만들기

Trait 이용한 익명 객체

trait를 정의하고 이를 이용해서 실제 new로 인스턴스를 만들때 구현된 블록을 제공하면 익명 인스턴스가 만들어진다.

```
scala> trait A {
     |    def show : Unit
     | }
defined trait A

scala> val a = new A { def show = println("anno ") }
a: A = $anon$1@16bbb59c

scala> a.show
anno

scala> a.getClass
res7: Class[_ <: A] = class $anon$1
```

암묵적 제너릭 클래스 정의하기

암묵적 처리 클래스 정의

인터페이스를 trait으로 정의하고 두개의 클래스를 암묵적으로 정의한다.

```
scala> trait Monoid[A] {
     |    def zero: A
     |    def plus(a:A, b:A) :A
     | }
defined trait Monoid

scala> implicit object IntMonoid extends Monoid[Int] {
     |    override def zero:Int = 0
     |    override def plus(a:Int, b:Int) : Int = a+b
     | }
defined object IntMonoid

scala> implicit object StringMonoid extends Monoid[String] {
     |    override def zero:String = ""
     |    override def plus(a:String, b:String) : String = a.concat(b)
     | }
defined object StringMonoid
```

함수 정의 및 정수처리

함수를 정의할 때 암묵적으로 trait에 연결하면 이 함수를 실행할 때 내부의 암묵적 메소드를 호출할 수 있다.

```
scala> def sum[A](values:Seq[A])(implicit ev:Monoid[A]) : A = values.foldLeft(ev
.zero)(ev.plus)
sum: [A](values: Seq[A])(implicit ev: Monoid[A])A

scala> sum(List(0,0))
res126: Int = 0

scala> sum(List(1,1))
res127: Int = 2
```

문자열 처리

암묵적으로 문자열을 호출하면 문자열 처리에 필요한 메소드가 호출되어 처리되는 것을 알 수 있다.

```
scala> sum(Seq("Hello","world"))
res129: String = Helloworld

scala> sum(List("Hello","world"))
res130: String = Helloworld

scala>
```

함수에 암묵적 매개변수 지정하기

암묵적 매개변수 지정

함수 내부에 명시적인 매개변수를 지정도 할 수 있지만 암묵적이 매개변수도 지정해서 사용할 수 있다.

```
scala> def pEx(a :Int) = a
pEx: (a: Int)Int

scala> pEx(10)
res21: Int = 10
```

```
scala> def pIm(implicit i :Int) = i
pIm: (implicit i: Int)Int

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

scala> pIm
res23: Int = 10
```

암묵적 매개변수 두개 지정

함수 내의 암묵적 매개변수가 하나인데 2개를 지정하면 실제 값을 제대로 찾을 수가 없어 오류를 발생한다.

```
scala> def pTwo(implicit i :Int) = print(i)
pTwo: (implicit i: Int)Unit

scala> implicit val v = 2
v: Int = 2

scala> implicit val v2 = 20
v2: Int = 20

scala> p
<console>:29: error: ambiguous implicit values:
 both value v of type => Int
 and value v2 of type => Int
 match expected type Int
       p
       ^
```

함수에는 하나만 암묵적 매개변수만 존재

함수를 정의할 때 제일 앞에 하나만 암묵적 매개변수를 지정해서 사용할 수 있다.

```
scala> def ppTwo(implicit i :Int, a:Int) = print(i)
ppTwo: (implicit i: Int, implicit a: Int)Unit

scala> def ppTwo(implicit i :Int, a:Int) = print(a,i)
ppTwo: (implicit i: Int, implicit a: Int)Unit

scala> def ppTwo(implicit i :Int, implicit a:Int) = print(a,i)
<console>:1: error: identifier expected but 'implicit' found.
       def ppTwo(implicit i :Int, implicit a:Int) = print(a,i)
                                  ^


scala> def ppTwo(i :Int, implicit a:Int) = print(a,i)
<console>:1: error: identifier expected but 'implicit' found.
       def ppTwo(i :Int, implicit a:Int) = print(a,i)
                         ^
```

부분함수를 암묵적으로 지정하기

함수 매개변수 목록을 이용해서 실제 들어오는 매개변수를 나눠서 처리할 수 있다. 이때도 실제 암묵적 매개변수 지정은 하나만 있어야 한다.

```
scala> def mul(x:Int)(implicit y:Int) = x * y
mul: (x: Int)(implicit y: Int)Int

scala> mul(3)
<console>:13: error: could not find implicit value for parameter y: Int
       mul(3)
           ^


scala> implicit val z:Int = 10
z: Int = 10

scala> mul(3)
res1: Int = 30
```

타입 클래스 패턴 순서

타입 클래스란

타입 클래스는 타입이 객체의 클래스를 정의하는 것과 같은 방식으로 타입의 클래스를 정의합니다. 스칼라에서 타입 클래스는 적어도 하나의 타입 변수를 가진 특성(trait)을 의미합니다.

```
scala> trait A[T] {
     |   def method(x:T) : String
     | }
defined trait A
```

타입 클래스 패턴 정의하는 법

암묵적으로 처리하기 위한 타입 클래스 패턴 정의 순서

타입클래스 정의 : 특정 기능을 구현하기 위한 일반적인 특징을 표현하는 것

타입클래스의 인스턴스 정의 : 표준 타입에 대한 구현을 미리 작성해 두거나 우리가 쓰고자 하는 커스텀 타입에 대한 구현을 정의 해 놓은 object

타입클래스의 인터페이스 정의 : 인터페이스는 사용자들이 사용하게 될 메소드를 정의 하는 것

명시적으로 클래스 정의

동일하 메소드가 있는 두 개의 클래스일 경우 다른 메소드가 추가할 필요가 있는 경우 수정을 해야한다.

```
scala> final case class A(fn: String, ln: String)
defined class A

scala> object AP {
     |    def print(x: A) = s"Hi, ${x.fn}, ${x.ln} "
     | }
defined object AP

scala> AP.print(A("Dahl","Moon"))
res13: String = "Hi, Dahl, Moon "
```

```
scala> final case class B(name:String)
defined class B

scala> object BP {
     |    def print(x: B) = s"Hi, ${x.name} "
     | }
defined object BP

scala> BP.print(B("lala"))
res14: String = "Hi, lala "
```

암묵적 타입 클래스와 객체 생성

trait에 추상 메소드를 정의하고 동반객체를 이용해서 내부에 로직을 넣는다.

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

trait Formatable[A] {
   def format(v:A) : String
}
object Formatable {
   implicit val s = new Formatable[String] {
      def format(i:String) = i
   }
   implicit val i = new Formatable[Int] {
      def format(i:Int) = i.toString
   }
}

// Exiting paste mode, now interpreting.

defined trait Formatable
defined object Formatable
```

인터페이스 생성 및 활용

인터페이스 메소드 내부에 암묵적으로 타입 클래스를 지정하고 format 메소드를 호출하면 타입에 맞춰 처리된다.

```
scala> object Format {
     |    def format[A](i:A)(implicit form : Formatable[A]) : String = {
     |       form.format(i)
     |    }
     | }
defined object Format

scala> Format.format("Hello")
res51: String = Hello

scala> Format.format(10)
res52: String = 10

scala> Forat.format(11.1)
<console>:28: error: not found: value Forat
       Forat.format(11.1)
       ^
```

타입 클래스를 상속해서 구성하기

타입을 지정하고 암묵적 동반객체 지정

trait를 통해 타입객체를 만들고 내부적으로 자동으로 연결될 object를 암묵적으로 지정한다.

타입 클래스 지정

암묵적으로 연결 객체 지정

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

trait Incrementer[T] {
   def inc(x:T) : T
}
implicit object IntIncrementer extends Incrementer[Int] {
   println("IntIncrementer ")
   def inc(x : Int):Int = x+1
}
implicit object DoubleIncrementer extends Incrementer[Double] {
   println(" DoubleInclrementer ")
   def inc(x : Double): Double = x + 10
}
```

함수를 인터페이스로 사용하기

함수 정의시 실제 실행한 타입클래스를 암묵적으로 지정하면 함수에 들어오는 타입에 따라 메소드가 호출되어 처리된다.

함수에 암묵적 타입클래스 지정

함수 실행하면 타입에 맞는 object가 자동으로 연결된다

```
scala> def imcrement[T](x:T)(implicit ev: Incrementer[T]) = ev.inc(x)
imcrement: [T](x: T)(implicit ev: Incrementer[T])T

scala> imcrement(10.0)
 DoubleInclrementer
res18: Double = 20.0

scala> imcrement(10)
IntIncrementer
res19: Int = 11
```