* Simple Classes and Parameterless Methods

Demo:

class Counter {  
 private var value = 0 // You *must* initialize the field  
 def increment() { value += 1 } // Methods are public by default  
 def current() = value  
}

在scala里面，一个scala源文件，可以包含多个类，他们之间可以相互访问。

* Properties with Getters and Setters

在Java中

public class Person { // This is Java

private int age;

public int getAge() { return age; }

public void setAge(int age) { this.age = age; }

}

我们创建一个类Person，他有一个属性age，有自己的getter和setter，任何人都可以调用setAge（21），那么他就永远21岁了，我们可以通过改变setAge方法，

public void setAge(int newValue) { if (newValue > age) age = newValue; }那么他就不能变年轻了。Getter 和 setter比pulic要好，因为他能根据具体情况来get和set。

Scala为每个属性提供getter和setter方法，这里是一个public field:

class Person {  
var age = 0  
}

Scala在JVM中生成这样一个类，它带有一个private的age field，它具有getter和setter方法。这两个方法都是public，因为我们没有将age设为private。如果设为private字段，那么getter和setter方法都为private。

在scala中，getter和setter方法是age和age\_=，举个例子：

println(fred.age) // Calls the *method* fred.age()  
fred.age = 21 // Calls fred.age\_=(21)

在scala中虽然getter和setter不叫getXxx和setXxx，但是，他们实现了相同的功能。

class Person {  
 private var privateAge = 0 // Make private and rename  
 def age = privateAge  
 def age\_=(newValue: Int) {  
 if (newValue > privateAge) privateAge = newValue; // Can’t get younger  
 }  
}

当我们用的时候，仍然是fred.age,但是fred不能变年轻了：

Val fred = new Person

fred.age = 30

fred.age = 21

println(fred.age) //30

你可能会感觉奇怪，scala为每个field生成getter和setter方法，但是我们也可以进行控制：

1. 如果field是private，getter 和setter就是private
2. 如果field是val，只会生成getter
3. 如果你不想要getter或者setter，可以将field声明为private[this]

* Properties with Only Getters

如果你只想要一个只读的field，只需要getter，不需要setter。那么使用val：

class Message {  
val timeStamp = java.time.Instant.now  
...  
}

scala编译器会产生一个private final field，同时带有一个public getter方法没有setter方法的Java类。

总结一下，你可以有四种方法：

1. Var foo： Scala会生成getter和setter
2. Val foo： scala 只会生成 getter
3. 你可以自己定义foo和foo\_=方法
4. 你可以只定义foo方法

在scala中没有write-only的field。Scala类中的field，和java中的类的field有所不同，它是一个private field，带有getter（val field）方法或者带有getter和setter方法（var field）

* Object-Private Field

和java一样，在scala中，class的所有对象都可以访问类的私有field。

class Counter {  
 private var value = 0  
 def increment() { value += 1 }  
 def isLess(other : Counter) = value < **other.**value  
 // Can access private field of other object  
}

Scala可以有更严格的访问限制符：private[this]

private[this] var value = 0 // Accessing *someObject*.value is not allowed

现在，只能在当前的object中访问value，而不能在Couter的其他object中访问。Class-privatescala会生成private getter和setter方法，而object-private field，不会生成getter和setter方法。

NOTE：

Scala 允许我们对特定的类授予访问权限，private[ClassName]指明只有指定类的方法可以访问给定的field。这些方法将被公开，因为JVM没有细粒度的访问控制系统。

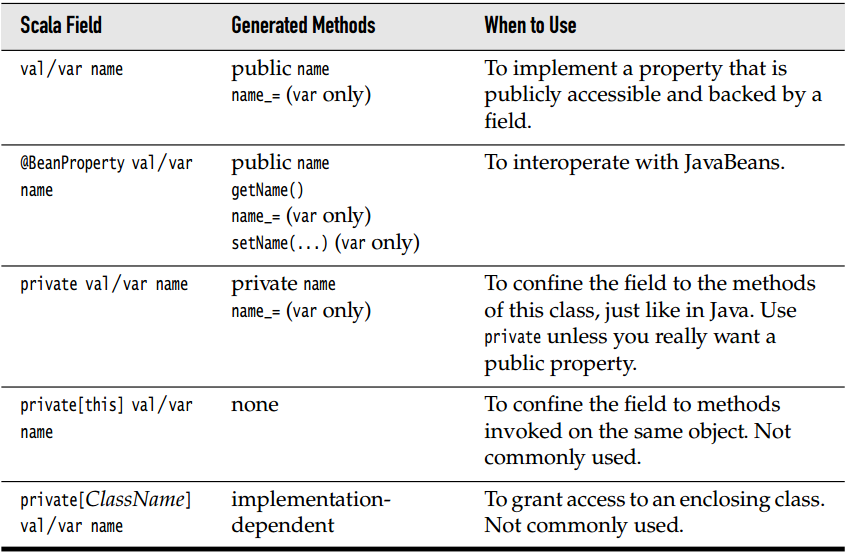
* Bean Peoperties

Scala为我们定义的field提供getter和setter方法，然而这些方法名并不是Java工具所期望的。当用@BeanProperty注解一个Scala字段时，这些方法是自动生成的。例如：

import scala.beans.BeanProperty  
class Person {  
@BeanProperty var name: String = \_  
}

会生成以下四个方法：

generates *four* methods:  
1. name: String  
2. name\_=(newValue: String): Unit  
3. getName(): String  
4. setName(newValue: String): Unit



Val var 都是生成的getter和setter都是public

Private val var生成的getter和setter是private，这个field仅限于这个类及这个类的对象。

Private[this] var val那么这个field仅限于这个object

Private[ClassName] 授权某个类对这个field有访问权限