* Extending a Class

在scala中你继承一个class，与java中相似，用extends关键字：

class Employee **extends** Person {  
 var salary = 0.0  
 ...  
}

和java中类似，你可以在子类中指定新的field和method，也可以重写父类的方法。你也可以声明class为final，这样他就不能被继承。也可以将method和field声明为final，这样他们就不能被重写。

NOTE：与java不同的是，java中final field是不可变的，类似scala中的val

* Overriding Methods

在scala中你必须使用override关键字来重写一个非abstract方法（如果重写的方法是abstract，可以省略override）

DEMO：

class Person {  
 ...  
 **override** def toString = s"${getClass.getName}[name=$name]"//override不能省略  
}

override关键字可以在一下情况下给出很多有用的错误信息，比如：

1. 当你需要重写method的名字写错的时候
2. 当你不小心把参数类型写错的时候
3. 当父类方法与子类方法冲突的时候

调用父类的方法和java一样，用super关键字：

class Employee extends Person {  
 ...  
 override def toString = s"${**super.**toString}[salary=$salary]"  
}

super.toString调用父类的toString方法，即Person.toString

* Type Checks and Casts

想检验一个实例是否属于某个class，可以用isInstanceOf方法。如果检验成功，你可以用asInstanceOf方法将引用转换。

if (p.isInstanceOf[Employee]) {  
 val s = p.asInstanceOf[Employee] // s has type Employee  
 ...  
}

如果p为指向class Employee或者它子类的实例（例如 Manager），p.isInstanceOf[Employee]返回true。如果p为null，p.isInstanceOf[Employee]返回false，p.asInstanceOf[Employee]返回null。

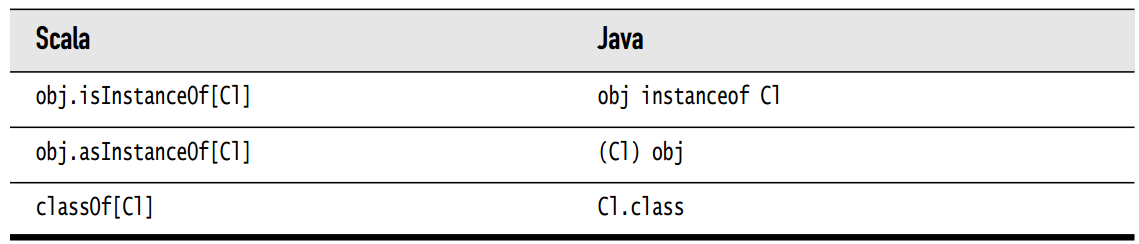
如果p不是一个Employee，那么p.asInstanceOf[Employee]会抛出异常。

如果你想检验p指向的是Employee的实例，而不是它的子类，可以用：

if (p.getClass == classOf[Employee])

classOf方法定义在scala.Predef object中，始终被导入。

下表显示了Scala和Java类型检查和强制转换之间的对应关系：



但是，模式匹配通常是使用类型检查和强制转换的更好选择，后面我会专门讲解scala中的模式匹配，例如：

p match {  
 case s: Employee => ... // Process s as an Employee  
 case \_ => ... // p wasn’t an Employee  
}

* Protected Fields and Methods

与java中类似，你可以将一个field或者method声明为protected，这样，这个field或者method可以在该类的子类中访问，但是其他位置却不可以。

与java不同的是：在java中protected声明的成员变量在该类所属的packet中可以访问protected修饰的成员，但是在scala中却不可以。同样也有一个protected[this]，它可以限制只能在当前的实例中访问，和private[this]类似。

* Superclass Construction

回想我在讲class的时候说过，一个class可以有一个primary constructor和多个auxiliary constructor，所有的auxiliary constructor必须调用前一个auxiliary constructor或者primary constructor。因此，auxiliary constructor不可能直接调用父类的构造函数。

子类的auxiliaryconstructor最终调用子类的primary constructor。只有子类的primary constructor才可以调用父类的构造函数。

回想之前所讲的，primary constructor和class的定义是交织在一起的，所以父类构造函数的调用也与此类似。DEMO：

class Employee(name: String, age: Int, val salary : Double) extends Person(name, age)

这样把class定义和构造函数交织在一起，使得代码非常简洁。我们可以发现，Employee有三个参数，name, age和salary，其中两个来自父类。

在java中，相同的实现会显得繁琐：

public class Employee extends Person { // Java  
 private double salary;  
 public Employee(String name, int age, double salary) {  
 super(name, age);  
 this.salary = salary;  
 }  
}

在scala中可以继承java 的类，它的primary constructor必须调用java 类的一个构造方法。Demo：

class PathWriter(p: Path, cs: Charset) extends java.io.PrintWriter(Files.newBufferedWriter(p, cs))

* Overriding Fields

回想scala类中说过，scala中一个field默认为pirvate，同时scala会自动生成它的getter和setter。你可以在子类中通过一个相同名字的val field来重写父类中的val field或者无参def。子类中会提供一个private field和一个public getter。

Demo：

class Person(val name: String) {  
 override def toString = s"${getClass.getName}[name=$name]"  
}  
class SecretAgent(codename: String) extends Person(codename) {  
 **override val** name = "secret" // Don’t want to reveal name . . .  
 **override val** toString = "secret" // . . . or class name  
}

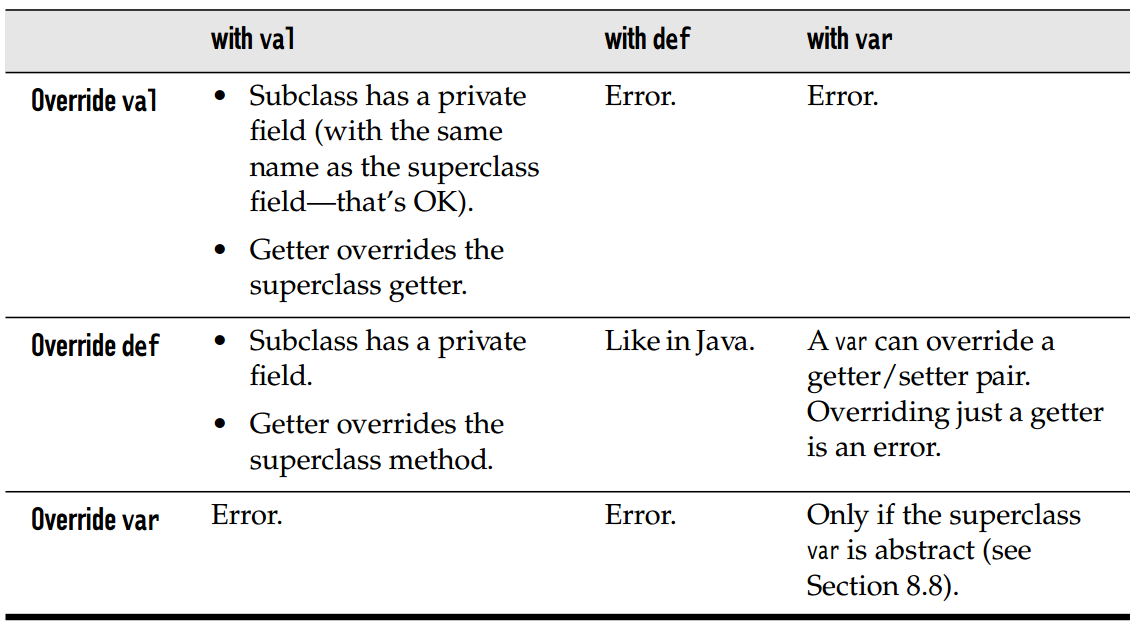
更常见的情况是使用val覆盖抽象的def，如下所示：

abstract class Person { // See Section 8.8 for abstract classes  
 def id: Int // Each person has an ID that is computed in some way  
 ...  
}

class Student(**override val** id: Int) extends Person  
// A student ID is simply provided in the constructor

下面是重写要注意的几点：

1. Def只能override def
2. Val 只能override val 或者无参的def
3. Var可以override 一个 abstract var



用val来override val好理解，用var来override var也好理解。其中有一点比较容易迷惑，Override def with var: A var can override a getter/setter pair. Overriding just a getter is an error.

那么如何理解这句话呢？

abstract class Person {

def id: Int

}

class Student extends Person{

override var id = 9527 //Error: method id\_= overrides nothing

}

在scala中定义了一个var变量，会自动生成getter和setter方法。

那么如果父类中只定义了一个方法def id: Int，用来生成用户的id，而子类用var变量重写这个方法的话override var id = 9527，编译会报错method id\_=overrides nothing，从报错中可以看出来，由于scala会为var变量自动生成了一个setter方法（id\_），但是父类并没有这个方法，所以是无法重写的。

abstract class Person {

def id: Int

def id\_=(value: Int) //父类必须有set方法

}

class Student extends Person{

override var id = 9527 //为var变量自动生成get和set方法

}

NOTE：

在前面class的部分我说可以用var来声明field，因为我们总是改变主意，重新实现一个getter/setter对。但是你要考虑，其他程序员继承你这个class的时候，他们不能通过getter和setter重写var，只能调用你写好的getter和setter。换句话说，如果你提供了一个var，那么所有的子类都被插入这个字段了。

* Anonymous Subclasses

和在Java中一样，如果想包含一个包含定义或覆盖的块，则创建一个匿名子类的实例，例如：

val alien = new Person("Fred") {

def greeting = "Greetings, Earthling! My name is Fred."

}

从技术上讲，这创建了一个结构类型的对象。类型表示为Person {def greeting：String}。 您可以将此类型用作参数类型：

def meet(p: Person{def greeting: String}) {  
 println(s"${p.name} says: ${p.greeting}")  
}