CS 520 Advanced Programming Languages
Fall Semester, 2009
Doc 22 Scala Type Parameterization
Dec 8, 2009

Reference

Programming in Scala, Odersky, Spoon, Venners, Artima Press, 2008

Reading

Programming in Scala, Odersky, Spoon, Venners, Artima Press, 2008
Chapters 19

Questions

Equality

a eq b

- Returns true if a and b reference the same object

a == b

- Returns true if a and b
- final method
- Uses equals

a equals b

- Same as ==
- Implement this to change meaning ==

Explain This

```
class Test(var z:String) {
      println(z)
      z = "cat"
      println(z)
      println(toString)

      override def toString = "z " + z
}
```

```
val a = new Test(a)
println(a.z)
```

Output

a

a

z cat

2 cat

cat

javap -c ClassName

Shows the byte code for the given class

aload_0	Load the pointer to the argument list
aload_1	Load first argument
aload_2	Load second argument

Constructor Bytecode

```
0: aload_0
1: aload_1
2: putfield  #12; //Field z:Ljava/lang/String;
5: aload_0
6: invokespecial      #17; //Method java/lang/Object."<init>":()V
9: getstatic#23; //Field scala/Predef$.MODULE$:Lscala/Predef$;
12:      aload_1
13:      invokevirtual#27; //Method scala/Predef$.println:(Ljava/lang/Object;)V
16:      aload_0
17:      ldc  #29; //String cat
19:      invokevirtual#32; //Method z_$eq:(Ljava/lang/String;)V
22:      getstatic#23; //Field scala/Predef$.MODULE$:Lscala/Predef$;
25:      aload_1
26:      invokevirtual#27; //Method scala/Predef$.println:(Ljava/lang/Object;)V
29:      getstatic#23; //Field scala/Predef$.MODULE$:Lscala/Predef$;
32:      aload_0
33:      invokevirtual#36; //Method toString:()Ljava/lang/String;
36:      invokevirtual#27; //Method scala/Predef$.println:(Ljava/lang/Object;)V
39:      return
```

Other Methods Generated

```
public void z_$eq(java.lang.String);
  Code:
    0: aload_0
    1: aload_1
    2: putfield  #12; //Field z:Ljava/lang/String;
    5: return

public java.lang.String z();
  Code:
    0: aload_0
    1: getfield  #12; //Field z:Ljava/lang/String;
    4: areturn
```

Multiple Inheritance Issues

```
trait Mom {
        val a = 3
        var b = 3
}

trait Dad {
        val a = 5
        var b = 5
}

class Kid extends Dad with Mom {
}
```

```
error: error overriding value a in trait Dad$class of type Int;
 value a in trait Mom$class of type Int needs `override' modifier
        class Kid extends Dad with Mom {
```

Use Abstract Fields

```
trait Mom {
	val a: Int
	var b: Int
}

trait Dad {
	val a: Int
	var b: Int
}

class Kid extends Dad with Mom {
	val a = 3
	var b = 5
}
```

Use Private Fields

```
trait Mom {
	private val a = 3
	private var b = 3
}

trait Dad {
	private val a = 5
	private var b = 5
}
```

```
class Kid extends Dad with Mom {
	val a = 3
	var b = 5
}
```

Type Parameterization

A problem

```
class Foo[T](initial: T) {
        private var value = initial
        def get = value
        def set(x: T) {value = x}
}
```

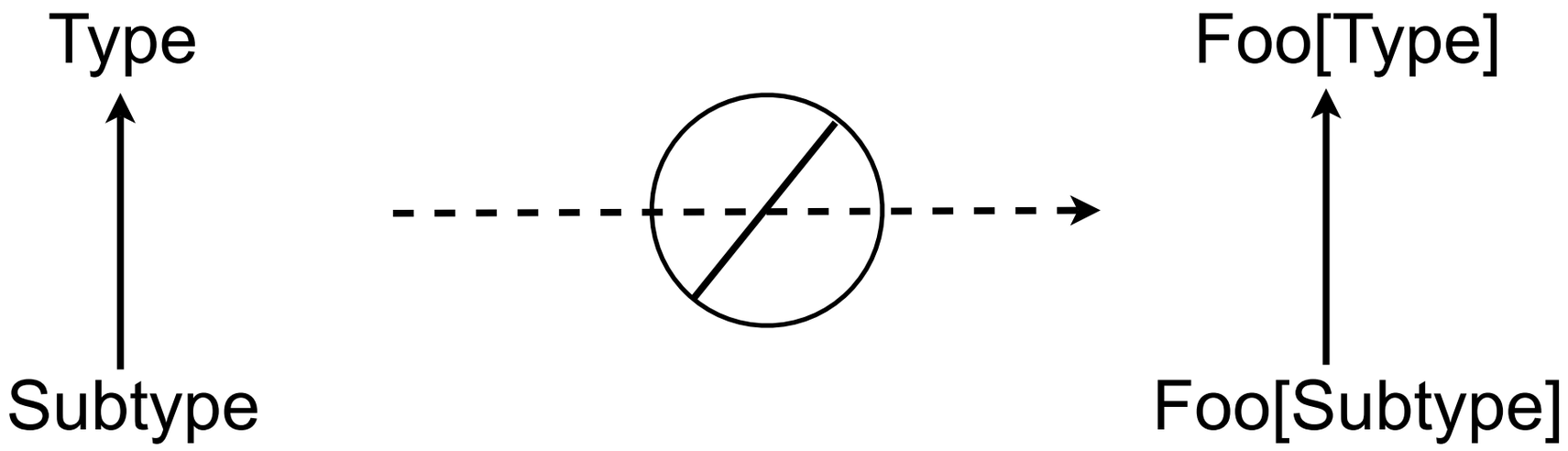
```
val a: Foo[Any] = new Foo[Any]("ok")

//Compile error
val b: Foo[Any] = new Foo[String]("ok")
```

The Issue

```
class Foo[T](initial: T) {
        private var value = initial
        def get = value
        def set(x: T) {value = x}
}
```

```
val a: Foo[String] = new Foo[String]("ok")
val b: Foo[Any] = a
b.set(1)
```


If we allow "val b: Foo[Any] = a" then we have a problem with b.set(1). "b.set(1)" should be legal as 1 is of type any. However we are really dealing with a reference to a. In a "value" is of type String and "b.set(1)" is trying to set "value" to a string


Type
Subtype
Foo[Type]
Foo[Subtype]

Default

```
class Bar[T](initial: T) {
      val value: Any = initial
}
```

```
var a = new Bar[AnyVal](1)
var b = new Bar[Float](12.0F)
a = b              //Compile error
b = a              //Compile error
```

Covariant

```
class Bar[+T](initial: T) {
      val value: T = initial
}
```

```
var a: Bar[Any] = new Bar[Any]("ok")
var b: Bar[String] = new Bar[String]("ok")
a = b
b = a            //Compile error
```

Compiler Checks for Possible Errors

```
class Bar[+T](initial: T) {
      var value: T = initial
}
```

```
error: covariant type T occurs in
contravariant position in type T of
parameter of setter value_=
            var value: T = initial
```

Upperbounds

```
class Bar[+T <: AnyVal](initial: T) {
      val value: T = initial
}
```

```
var a = new Bar[AnyVal](1)
val b = new Bar[Float](12.0F)
a = b
```

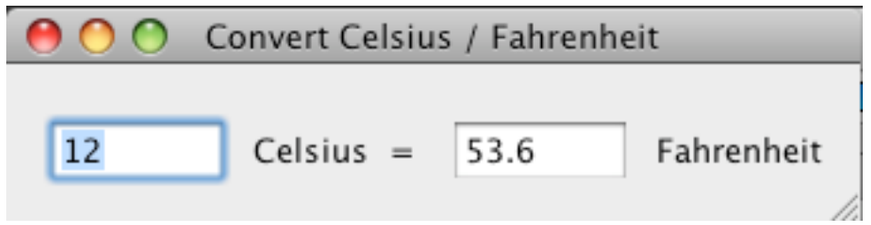
Contravariant

```
class Bar[-T](initial: T) {
      val value: Any = initial
}
```

```
var a = new Bar[AnyVal](1)
var b = new Bar[Float](12.0F)
a = b              //Compile error
b = a
```

GUI

Sample GUI




```
import swing._
import event._

object CelsiusConverter2 extends SimpleGUIApplication {
    val ui = new FlowPanel {
        val celsius = new TextField { columns = 5 }
        val fahrenheit = new TextField { columns = 5 }
        contents.append(celsius, new Label(" Celsius  =  "), fahrenheit, new Label(" Fahrenheit"))
        border = Swing.EmptyBorder(15, 10, 10, 10)

        listenTo(celsius)
        reactions += {
            case EditDone(celsius) =>
                val c = celsius.text.toFloat
                val f = c * 9 / 5 + 32
                fahrenheit.text = f.toString
        }
    }
    def top = new MainFrame {
        title = "Convert Celsius / Fahrenheit"
        contents = ui
    }
}
```


From Scala source distribution