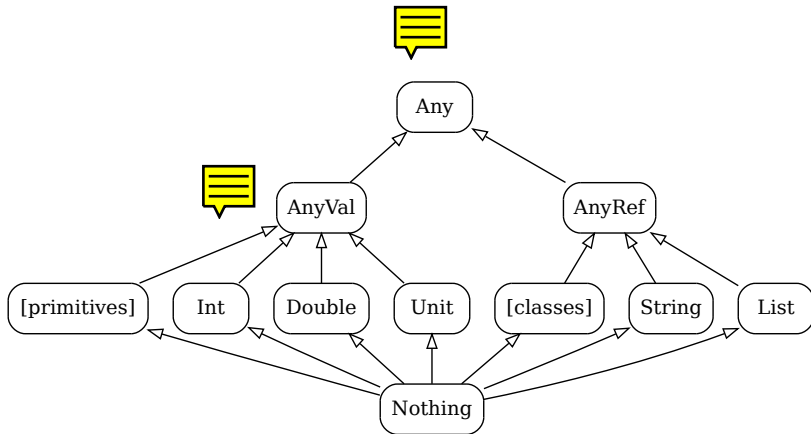


OOP en Scala

Jean-Luc Falcone

Janvier 2020

Hiérarchie des types



Déclaration

java

```
public class Car extends Vehicle implements Motorized {  
    //...  
}
```

scala

```
class Car extends Vehicle with Motorized {  
    //...  
}
```

Constructeur (java)

```
public class Point {  
  
    private final double x;  
    private final double y;  
  
    public Point( double xx, double yy ) {  
        System.out.println( "Building point..." );  
        x = xx;  
        y = yy;  
    }  
  
    double distance( Point p ) {  
        double d2 = (p.x-x)*(p.x-x) + (p.y-y)*(p.y-y);  
        return Math.sqrt( d2 );  
    }  
}
```

Constructeur (Scala)



```
class Point( x: Double, y: Double ) {  
  println( "Building point..." )  
  
  def distance( p: Point ) = {  
    val d2 = (p.x-x)*(p.x-x) + (p.y-y)*(p.y-y)  
    math.sqrt( d2 )  
  }  
}  
  
val p = new Point( 1, 0.5 )
```

Accesseurs (java)

```
public class Point {  
  
    private final double x;  
    private final double y;  
  
    /* ... */  
  
    double getX() { return x; }  
    double getY() { return y; }  
}
```

Accesseurs (scala)



```
class Point( xx: Double, yy: Double ) {  
  val x = xx  
  val y = yy  
  /* ... */  
}
```

Accesseurs (scala)



```
class Point( val x: Double, val y: Double ) {  
  /* ... */  
}
```


Accesseurs (scala)

```
class Rank( val position: Double, var player: Player ) {  
  /* ... */  
}
```

Uniform access principle

- Si un champ `val` est public, le **getter** correspondant est généré
- Si un champ `var` est public, les **getter** et **setter** sont générés

Evaluation paresseuse (java)

```
public class DataSample {  
    private double[] data = null;  
    private double avg = 0.0;  
    private boolean avgOK = false;  
    private double[] getData() {  
        if( data == null ) data = downloadData();  
        return data;  
    }  
    public double getAverage() {  
        if( ! avgOK ) {  
            avg = computeAvg( getData() );  
            avgOK = true;  
        }  
        return avg;  
    }  
}
```

Evaluation paresseuse (scala)

```
public class DataSample {  
  
    private lazy val data = downloadData()  
  
    lazy val average = computeAvg( data )  
  
}
```

Allègement de la syntaxe

```
val p = new Point( 1.5, -0.1 )
```

```
val q = new Point( 0.5, 0.25 )
```

```
val d1 = p.distance(q)
```

```
val d2 = p distance q
```

Opérateurs symboliques

```
class OrderItem( val item: Item, number: Int )

class Item( val name: String, val price: Double ) {
  def *( num: Int ) = new OrderItem( this, num )
}

val apple = new Item( "Apple" )

val order1 = apple.*(3)
val order2 = apple * 3
```

Opérateurs Symboliques

Haute lisibilité

```
val order = apple*2 + orange*3 + banana  
order >> customer
```

Opérateurs Symboliques

Haute lisibilité

```
val order = apple*2 + orange*3 + banana  
order >> customer
```

Attention aux abus

```
val x = ( data1 /+ data2 ) !# filter  
x ||| "Error during processing"
```


Redéfinir une méthode (java)

```
public class Point {  
  
    private final double x;  
    private final double y;  
  
    @Override  
    public boolean equals( object that ) { /*...*/ }  
  
}
```

Redéfinir une méthode (scala)

```
class Point( val x: Double, val y: Double ) {  
  
    override def equals( that: Any ) = { /*...*/ }  
  
}
```

Egalité

```
java
```

```
// Egalité par reference ...ou valeur
```

```
p == q
```

```
// Egalité par valeur
```

```
p.equals(q)
```

Egalité

java

// Egalité par reference ...ou valeur

p == q

// Egalité par valeur

p.equals(q)

scala

// Egalité par reference

p.eq(q)

p eq q

// Egalité par valeur

p.equals(q)

p equals q

p == q

Case class

```
case class Point( x: Double, y: Double )
```

Code généré

- *Getters*
- *equals, hashCode, toString*
- *copy*
- Methode *factory*
- Extracteur



Extracteur

```
def whereis( p: Point ) = p match {  
  case Point(0,0) => "Origin"  
  case Point(0,y) if y > 0 => "North"  
  case Point(x,y) =>  
    "Somewhere: " + x + ";" + y  
}
```

Méthode copy

```
val p = Point( 1.5, 0.5 )
```

```
val q = p.copy( x=0 )
```

```
val r = q.copy( y=0 )
```

Interface (java)

```
public interface Motorized {  
    public void start();  
    public boolean isStarted();  
}
```

```
public class Car implements Motorized {  
    private boolean started = false;  
    public void start() {  
        /* Start */  
        started = true;  
    }  
    public boolean isStarted() { return started; }  
}
```


Trait (scala)

```
trait Motorized {  
  def start(): Unit  
  def isStarted: Boolean  
}  
  
class Car extends Motorized {  
  private var started = false  
  def start() {  
    /*Start*/  
    started = true  
  }  
  def isStarted = started  
}
```

Mixin (scala)

```
trait Motorized {  
  private var started = false  
  protected doStart(): Unit  
  def start() {  
    doStart()  
    started = true  
  }  
  def isStarted = started  
}
```

```
class Car extends Motorized {  
  def doStart = /* Start */  
}
```

Mixine (Java)

```
public interface TimeClient {  
    // ...  
    static public ZoneId getZoneId (String zoneString) {  
        try {  
            return ZoneId.of(zoneString);  
        } catch (DateTimeException e) {  
            System.err.println("Invalid time zone: " + zoneString  
                + "; using default time zone instead.");  
            return ZoneId.systemDefault();  
        }  
    }  
  
    default public ZonedDateTime getZonedDateTime(String zoneString) {  
        return ZonedDateTime.of(getLocalDateTime(), getZoneId(zoneString));  
    }  
}
```

Classe "statique" (java)

```
public class Water {  
  
    public final static double g = 9.81;  
    public final static double density = 1000;  
  
    public static double pressure( double h ) {  
        return waterDensity * g * h;  
    }  
  
}
```

Objet (scala)

```
object Water {  
  
  val g = 9.81  
  val density = 1000.0  
  
  def pressure( h: Double ) = density * g * h  
  
}
```

Singleton (java)

```
public class ItemPriceComparator
implements Comparator<Item> {
    public int compareTo( Item i1, Item i2 ) {
        if( i1.getPrice() < i2.getPrice() ){
            return -1;
        }
        if( i1.getPrice() > i2.getPrice() ) {
            return 1;
        }
        return 0;
    }
}
```

Singleton (scala)

```
object ItemPriceComparator extends Comparator[Item] {  
  def compareTo( i1: Item, i2: Item ) =  
    if( i1.price < i2.price ) -1  
    else if( i1.price < i2.price ) 1  
    else 0  
}
```

Static factory (java)

```
public class Username {  
    private String uname;  
    private Username( String u ) {  
        uname = u;  
    }  
    public static Username make( String u ) {  
        check(u);  
        new Username( u );  
    }  
    private static void check( u ) { /* ... */ }  
}
```


Companion object (scala)

```
class Username private( username: String )

object Username {
  def make( u: String ): UserName = {
    check(u)
    new Username(u)
  }
  private def check( u: String ): Unit =
    ???
}
```

Import

Package import (java)

```
package com.example.foo;  
  
import com.example.bar.Bar;  
import com.example.bar.Ops;  
import com.example.baz.*;
```

Import

Package import (java)

```
package com.example.foo;  
  
import com.example.bar.Bar;  
import com.example.bar.Ops;  
import com.example.baz.*;
```

Package import (scala)

```
package com.example  
package foo  
  
import com.example.bar.{Bar,Ops}  
import com.example.baz._
```

Import (Scala)

```
package com.example
package foo {
  import java.util.{List => JList}

  case class Foo( i: Int, s: String )

  def isEmpty( f: Foo ): Boolean = {
    import f._
    i == 0 && s.size == 0
  }
}
```

String interpolation

Création de String (java)

```
String s1 = x + " " + y + " " + z
```

```
String s2 = "Foo(" + f.i + ", " + f.s + ")";
```

```
String s3 = usd + " $"
```

String interpolation

Création de String (java)

```
String s1 = x + " " + y + " " + z  
String s2 = "Foo(" + f.i + ", " + f.s + ")";  
String s3 = usd + " $"
```

Création de String (scala)

```
val s1 = s"$x $y $z"  
val s2 = s"Foo(${f.i}, ${f.s})"  
val s3 = s"$usd $"
```

