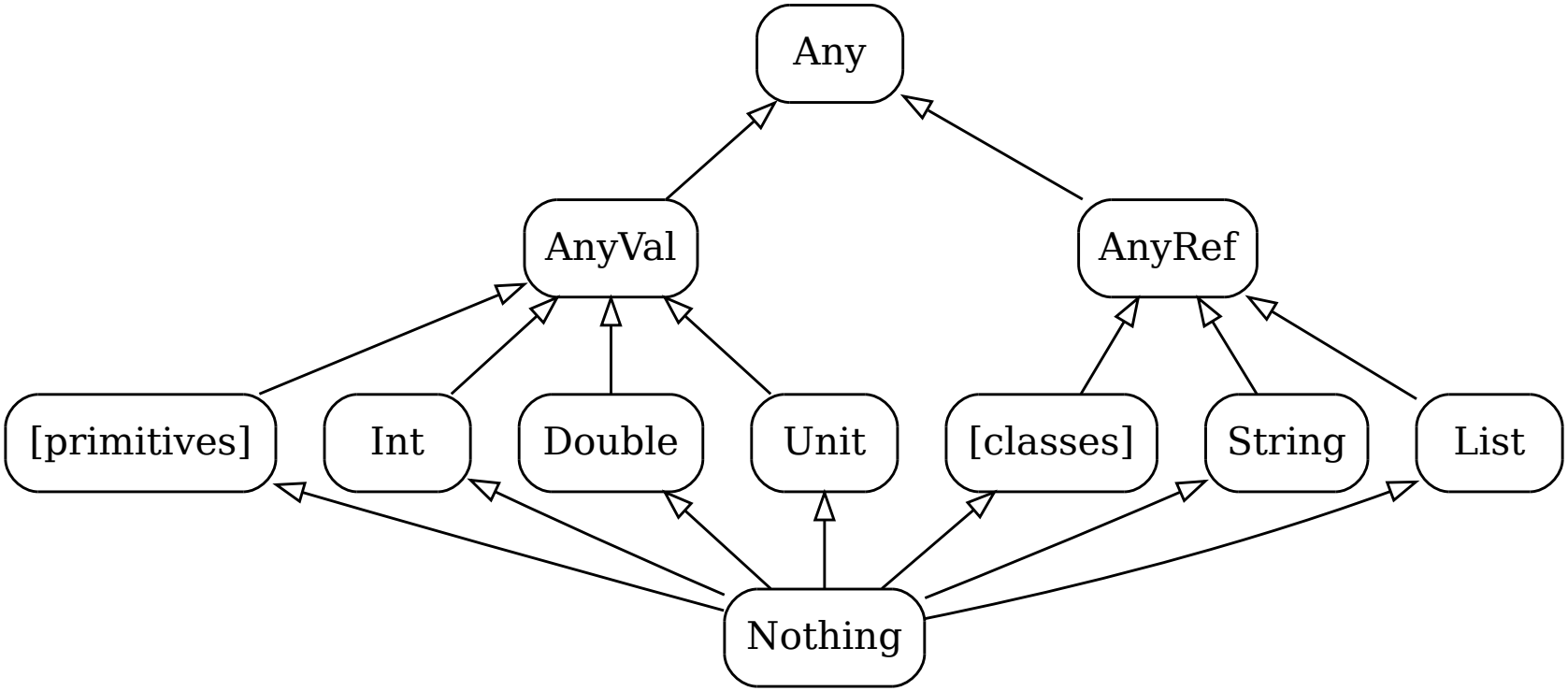
Programmation Orientée objet

Jean-Luc Falcone

HEPIA - 2013

Hiérarchie des types



Déclaration

java

```java
public class Car extends Vehicle implements Motor {
  //...
}
```

scala

```scala
class Car extends Vehicle with Motor {
  //...
}
```

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( xx: Double, yy: Double ) {
  println( "Building point..." )

  private val x = xx
  private val y = yy

  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)

```scala
class Point( xx: Double, yy: Double ) {

  private val x = xx
  private val y = yy

  /* ... */

  def getX() = x
  def getY() = y
}
```

Accesseurs (scala)

```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)

```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
```

Surcharge des opérateurs

```
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
```

Surcharge des opérateurs

Haute lisibilité

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

Surcharge des opérateurs

Haute lisibilité

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

Attention aux abus

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

Surcharger de méthode (java)

```java
public class Point {

  private final double x;
  private final double y;

  @Override
  public boolean equals( object that ) { /*...*/ }

}
```

Surcharger de méthode (scala)

```scala
class Point( val x: Double, val y: Double ) {

  override def equals( that: Any ) = { /*...*/ }

}
```

Egalité

java

```
// Comparaison de reference
p == q
// Comparaison de valeur
p.equals(q)
```

Egalité

java

```
// Comparaison de reference
p == q
// Comparaison de valeur
p.equals(q)
```

scala

```
// Comparaison de reference
p.eq(q)
p eq q
// Comparaison de 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 Motor {
  public void start();
  public boolean isStarted();
}

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

Trait (scala)

```scala
trait Motor {
  def start(): Unit
  def isStarted: Boolean
}

class Car extends Motor {
  private var started = false
  def start() {
    /*Start*/
    started = true
  }
  def isStarted = started
}
```

Mixin (scala)

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

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

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)

```scala
object Water {

  val g = 9.81
  val density = 1000.0

  def pressure( h: Double ) = density * g * h

}
```

Singleton (java)

```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)

```scala
case class Username private( username: String )

object Username {
  def make( u: String ) = {
    check(u)
    Username(u)
  }
  private def check( u: String ) { /* ... */}
}
```