Scala

My title here

Carlos Tomé Cortiñas, Renate Eilers and Matthew Swart

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Table of Contents

History

- 2001
- École Polytechnique Fédérale de Lausanne (EPFL) by Martin Odersky

Companies (Not sure)

- Twitter
- Linkedin
- The Guardian
- FourSquare
- Sony
- etc.

Projects

- PlayFramework
- Akka

Scala (Still need to add a picture)

- Object oriented
- Functional
-

REPL

```
Welcome to Scala 2.12.0 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_111).
Type in expressions for evaluation. Or try :help.

scala> 5 + 5
res0: Int = 10

scala>  (1,2,3,4,5,6,7,8,9,10)
res1: (Int, Int, Int, Int, Int, Int, Int, Int, Int, Int) = (1,2,3,4,5,6,7,8,9,10
)

scala> (true,"It is not true",1,'p')
res2: (Boolean, String, Int, Char) = (true,It is not true,1,p)

scala> (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23)
<console>:1: error: too many elements for tuple: 23, allowed: 22
(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23)
^

scala> List(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,27
)
res3: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
, 18, 19, 20, 21, 22, 23, 24, 25, 27)

scala> List(true,"It is not true",1,'p')
res4: List[Any] = List(true, It is not true, 1, p)

scala> val countries = Map("Carlos" -> "Spain", "Renate" -> "The Netherlands", "
Matthew" -> "The Netherlands")
countries: scala.collection.immutable.Map[String,String] = Map(Carlos -> Spain,
Renate -> The Netherlands, Matthew -> The Netherlands)

scala> countries("Matthew")
res5: String = The Netherlands

scala>
```

Figure 1

Type system

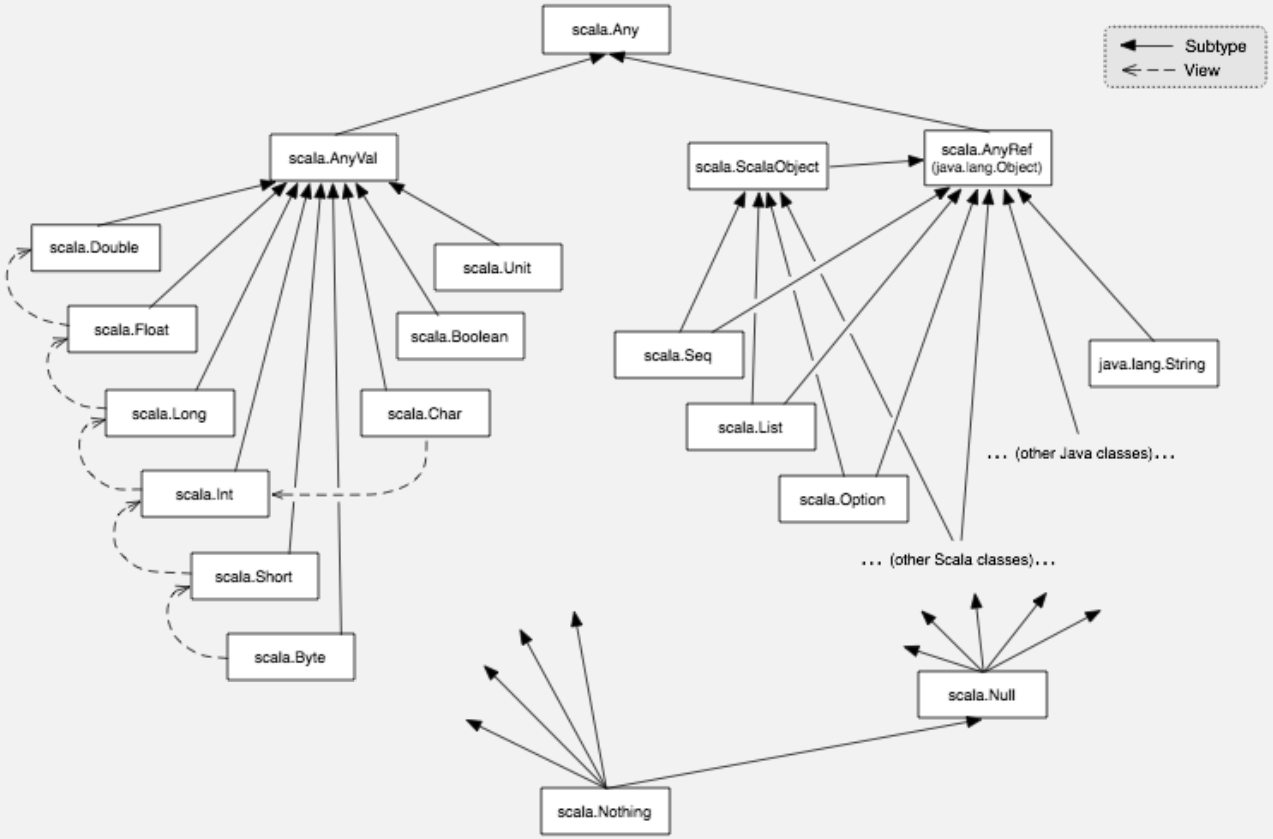

Figure 2

Classes

```
class Dog(name : String){
  def sound : Unit = println("Woof!")
  def movement : String = "Walk"
}
```

```
class Dog{
  private String name;
  public Dog(String name){
    this.name = name;
  }
  public void sound(){
    System.out.println("Woof!")
  }
  public String movement(){
    return("Walk")
  }
```

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]

Abstract classes

Object

Examples.

Classes

	Class	Abstract class	Object	Trait
Inherentence				
Composition				
Parameters				

Traits

Introduction

Look a lot like class definitions Difference with classes:

- No class parameters

```
class Point(x : Int, y : Int)
```

```
trait Point(x : Int, y : Int) // Does not compile
```

- `super` calls are dynamically bound
- Classes do not inherit traits: traits are mixed in

An example

Example with queue

Mixin traits

Extend queue example with double/increment Linearization

GADTs in Scala

- simpe Expr language in Scala using case classes
- pattern matching (show using eval function)

Anonymous function

```
scala> values.map(x => x + 1)
res3: List[Int] = List(2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
```

Function types

A => B is an abbrevication for the class scala.Function1[A,B]
package scala

```
trait Function1[A,B]{
  def apply(x : A) : B
}

traits Function2...Function22
```

Anonymous function

Anonymous function (x : Int) => x + 1

This will be expanded to (Same syntax as Java)

```
new Function1[Int,Int]{
    def apply(x : Int) : Int = x + 1
}
```

Apply method

Currying

Currying - Anonymous function

```
var f = (x : Int) => (y : Int) => x + 1
f: Int => (Int => Int) = $$Lambda$1113/551797833@2c58c

scala> f(1)
res0: Int => Int = $$Lambda$1130/1813375175@56380231

scala> f(1)(2)
res1: Int = 2
```


Case Classes

```
case class Person(name : String) extends Animal{
  def noise = "I am a person"
}
case class Tiger() extends Animal{
  override def noise : String = "Grr"
}

case class Frog() extends Animal{
  def noise : String = "CROAK"
}
```

Pattern Match

```
object AnimalNoise{
  def mkSound(animal : Animal) : Unit =
    animal match {
      case Frog() => println(Frog().noise)
      case Person(name) => println(name)
      case x => println(x.noise)
    }
}
```

Pattern Match

```
object pattern{
  def all(allTypes : Any) : Unit = {
    allTypes match{
      case (x,1) => println("(x,1)")
      case (x,y) => println("(x,y)")
      case x : String => println(x)
      case Tiger() => println(Tiger().noise)
      case 1 => println("One")
      case true => print("True")
    }
  }
}
```

Evaluation

```
class LazyMethod(values : () => List[List[String]]) {
    def heavyComputation : List[String] = {
      for(value <- values; n <- value)
        for(value1 <- values; n1 <- value)
          for(value2 <- values; n2 <- value)
            for(value3 <- values; n3 <- value)
              for(value4 <- values; n4 <- value)
    }
}

...
```

To conclude

Universiteit Utrecht

[Faculty of Science
Information and Computing
Sciences]