Java to Scala

[Scala for Java Developers] http://www.scala-lang.org/docu/files/ScalaTutorial.pdf (http://www.scala-lang.org/docu/files/ScalaTutorial.pdf)

In [1]:
```
import java.util.{Date, Locale}
import java.text.DateFormat._
```

```
import java.util.{Date, Locale}
import java.text.DateFormat._
```

In [5]:
```
println(getDateInstance(LONG,Locale.UK).format(new Date()))
```

```
29 May 2017
```

In [3]:
```
println(getDateInstance(LONG,Locale.US) format new Date)
```

```
May 29, 2017
```

In [4]:
```
println(getDateInstance(LONG,Locale.GERMANY) format new Date)
```

```
29. Mai 2017
```

CLASSES

In [6]:
```
class Person(name:String, age:Int){
  def name =name
  def age = age //fun is also first class Citizen//name is defining again
}
```

```
Main.scala:24: overloaded method name needs result type
  def name =name
           ^
Main.scala:25: overloaded method age needs result type
  def age = age
           ^
```

In [6]:
```
class Person(name:String, age:Int){
  def name :String =name
  def age:Int = age
}
```

```
Main.scala:24: ambiguous reference to overloaded definition,
both method name in class Person of type => String
and  value name in class Person of type String
match expected type String
  def name :String =name
                    ^
Main.scala:25: ambiguous reference to overloaded definition,
both method age in class Person of type => Int
and  value age in class Person of type Int
match expected type Int
  def age:Int = age
                ^
```

In [6]:
```
class Person(name, age){
  def name():String =name
  def age() :Int = age
}
```

```
Failure(")":1:14 ..."name, age)")
```

In [6]:
```
class Person(name, age){
}
```

```
Failure(")":1:14 ..."name, age)")
```

In [11]:
```
class Person(name:String, age:Int){
  def name():Int =name
  def age() :Int = age
}
```

```
defined class Person
```

In [14]:
```
val p = new Person("b",30)
```

```
p: Person = cmd10$$user$Person@5a0661fa
```

In [15]:
```
p name
```

```
java.lang.reflect.InvocationTargetException
  sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.j
ava:57)

sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccess
orImpl.java:43)
  java.lang.reflect.Method.invoke(Method.java:606)

ammonite.Interpreter$$anonfun$evaluate$1$$anonfun$apply$9.apply(Interp
reter.scala:325)
  ammonite.Interpreter$.evaluating(Interpreter.scala:291)

ammonite.Interpreter$$anonfun$evaluate$1.apply(Interpreter.scala:325)

ammonite.Interpreter$$anonfun$evaluate$1.apply(Interpreter.scala:324)
  ammonite.InterpreterAction$$anon$1.apply(Interpreter.scala:57)
```

In [10]:
```
p age
```

```
res9: Int = 30
```

In [24]:
```
class Person(name:String, age:Int){
  val _name = name
  def name():String =name
  def age() :Int = age
}
```

```
defined class Person
```

In [20]:
```
val p = new Person("buddy",30)
```

```
p: Person = cmd18$$user$Person@7b435b90
```

In [21]:
```
p name
```

```
res20: String = "buddy"
```

In [22]:
```
p _name
```

```
res21: String = "buddy"
```

In [28]:
```
//anything under {} primary constructor
```

In [29]:
```
println(p)
```

```
cmd18$$user$Person@7b435b90
```

In [29]:
```
class Person(name:String, age:Int)
val p = new Person("buddy",30)
p.name //variable
```

```
Main.scala:30: value name is not a member of $user.this.Person
p.name //variable
  ^
```

In [30]:
```
case class Person(name:String, age:Int)//automatic obj class - static sin
val p = new Person("buddy",30)
p.name //variable
```

```
defined class Person
p: $user.Person = Person("buddy", 30)
res29_2: String = "buddy"
```

In [31]:
```
println(p)
```

```
Person(buddy,30)
```

In [33]:
```
case class Person(name:String, age:Int)//getter setter already implemente
val p = Person("buddy",30)//new not reqd, Person.Apply() method instead
p.name //variable
println(p)
```

```
Person(buddy,30)
```

```
defined class Person
p: $user.Person = Person("buddy", 30)
res32_2: String = "buddy"
```

In [34]:
```
class Person(name:String, age:Int){
  def name():String =name
  def age() :Int = age
  override def toString()={
    "Person with name "+name+" at an age of "+age
  }
}
val p=new Person("ABC",45)
```

```
defined class Person
p: $user.Person = Person with name ABC at an age of 45
```

Pattern Matching / Decomposition

pattern maching - crieteria -type,vale, reg expn, decomposition itself

In [35]:
```
abstract class Shape
case class Square(s:Int) extends Shape
case class Rectangle(l:Int,b:Int) extends Shape
def printObj(x:Shape) = x match{
  case Square(s) => println("Square: "+s)
  case Rectangle(x,y)=>println("Rectangle: "+x+"::"+y)
}
```

```
defined class Shape
defined class Square
defined class Rectangle
defined function printObj
```

In [36]:
```
printObj(Square(3)) //type cast down to Square from Shape and assign 3 to
printObj(Rectangle(4,5))//para are decomposed, accessing the variable. Po
```

```
Square: 3
Rectangle: 4::5
```

In [39]:
```
abstract class Shape
case class Square(s:Int,t:Int) extends Shape
case class Rectangle(l:Int,b:Int) extends Shape
def printObj(x:Shape) = x match{
  case Square(s,t) => println("Square: "+s+" and t: "+t)
  case Rectangle(x,y)=>println("Rectangle: "+x+"::"+y)
}
```

```
defined class Shape
defined class Square
defined class Rectangle
defined function printObj
```

In [40]:
```
printObj(Square(3,6))
printObj(Rectangle(4,5))
```

```
Square: 3 and t: 6
Rectangle: 4::5
```

In [40]:
```
abstract class Shape
case class Square(s:Int) extends Shape
case class Square(s:Int,t:Int) extends Shape
case class Rectangle(l:Int,b:Int) extends Shape
def printObj(x:Shape) = x match{
  case Square(s) => println("Square: "+s)
  case Square(s,t) => println("Square: "+s+" and t: "+t)
  case Rectangle(x,y)=>println("Rectangle: "+x+"::"+y)
}
```

```
Main.scala:28: Square is already defined as case class Square
              abstract class Shape ; case class Square(s:Int) extends
Shape ; case class Square(s:Int,t:Int) extends Shape ; case class Rect
angle(l:Int,b:Int) extends Shape ; def printObj(x:Shape) = x match{

^
Main.scala:29: wrong number of arguments for pattern $user.this.Square
(s: Int,t: Int)
  case Square(s) => println("Square: "+s)
             ^
```

In [1]:
```
class Person(name:String, age:Int)
val p= new Person("H",20)
```

```
defined class Person
p: $user.Person = cmd0$$user$Person@2d77f2cb
```

In [1]:
```
p name
```

```
Main.scala:25: value name is not a member of cmd1.this.$ref$cmd0.Perso
n
p name
  ^
```

In [8]:
```
case class Person(name:String, age:Int)
val p= new Person("H",20)
p name
```

```
defined class Person
p: $user.Person = Person("H", 20)
res7_2: String = "H"
```

In [7]:
```
p.name
p.age
```

```
res6_0: String = "H"
res6_1: Int = 20
```

In [5]:
```
p age
```

```
res4: Int = 20
```

In [8]:
```
p name
p age
```

```
Main.scala:26: type mismatch;
 found   : cmd8.this.$ref$cmd7.Person
 required: Int
p age
^
```

In [ ]:
```
//getter n setter
//overriden toString fun
//companion class - type of class - no need of new operator
//singleton factory
//obj - immediate target for
//extracting value for para - to local variables
//class - companion case obj
//case expn matching
//boiler plate - created automatically - msg accepts String msg only.
```

In [ ]:
```

```