THE FUNCTIONAL
ARCHITECTURE STUDIO

Spark programming with Scala

1. Introduction to Scala

Habla Computing
info@hablapps.com
@hablapps

Index

- Object Oriented features
- Generic Programming features
- Implicits
- SBT (Simple Build Tool)
- Testing with ScalaTest

Spark programming with Scala

1. Object Oriented features

OO features
Goals

- Be able to apply OO fundamental aspects in an idiomatic way in Scala: classes, attributes, methods...
- Understand new concepts Scala introduces into this paradigm: objects, traits, case classes...
- Use shortcuts and other kind of syntactic sugar Scala provides

OO features
Outline

- Good old classes
- Single inheritance
- (Singleton & Companion) Objects
- Traits
- Case Classes
- Method invocation syntax
- The *apply* method

OO features
Good old classes (1/4)

```
class Bike(_cadence: Int, _speed: Int, _gear: Int) {
  var cadence: Int = _cadence
  var speed: Int = _speed
  var gear: Int = _gear
}

val b: Bike = new Bike(1, 2, 3)
b.cadence // resX: Int = 1
b.cadence = 100
b.cadence // resX: Int = 100
```

section2-oo/code/Bicicleta.scala

OO features
Good old classes (2/4)

```scala
class Bike(_cadence: Int, _speed: Int, _gear: Int) {
  val cadence: Int = _cadence
  val speed: Int = _speed
  val gear: Int = _gear
}

val b: Bike = new Bike(1, 2, 3)
b.cadence // resX: Int = 1
// b.cadence = 100 // Doesn't compile
// b.cadence will ALWAYS be 1
```

section2-oo/code/Bicicleta.scala

OO features
Good old classes (3/4)

```scala
class Bike(val cadence: Int, val speed: Int, val gear: Int)




val b: Bike = new Bike(1, 2, 3)
b.cadence // resX: Int = 1
// b.cadence = 100 // Doesn't compile
// b.cadence will ALWAYS be 1
```

section2-oo/code/Bicicleta.scala

OO features
Good old classes (4/4)

```
class Bike(val cadence: Int, val speed: Int, val gear: Int) {
  def slowDown(dec: Int): Bike =
    new Bike(cadence, speed-dec, gear)
}

val b: Bike = new Bike(1, 2, 3)
b.speed // resX: Int = 2
val b2: Bike = b.slowDown(2)
b2.speed // resX: Int = 0
b.speed // resX: Int = 2
```

section2-oo/code/Bicicleta.scala

OO features
Single inheritance

```
class MountainBike(
  val seatHeight: Int,
  cadence: Int,
  speed: Int,
  gear: Int) extends Bike(cadence, speed, gear)

val mb: MountainBike = new MountainBike(1, 2, 3, 4)
mb.seatHeight // resX: Int = 1
mb.speed // resX: Int = 3
mb.slowDown(2).speed // resX: Int = 1
```

section2-oo/code/Bicicleta.scala

OO features
Singleton Objects

```
object BikeFactory {
  val defaultSpeed: Int = 25

  def create(cadence: Int): Bike =
    new Bike(cadence, defaultSpeed, 200)
}

BikeFactory.defaultSpeed // resX: Int = 25
BikeFactory.create(10) // resX: Bike = Bike(10, 25, 200)
```

section2-oo/code/Bicicleta.scala

OO features
Companion Objects

```scala
object Bike {
  val defaultSpeed: Int = 25

  def create(cadence: Int): Bike =
    new Bike(cadence, defaultSpeed, 200)
}

Bike.defaultSpeed // resX: Int = 25
Bike.create(10) // resX: Bike = Bike(10, 25, 200)
```

section2-oo/code/Bicicleta.scala

OO features
Abstract classes

```
abstract class RoadBike(
    cadence: Int,
    speed: Int,
    gear: Int) extends Bike(cadence, speed, gear) {
  val seatHeight: Int
}

val b: RoadBike = new RoadBike(1, 2, 3) {
  val seatHeight = 5
}
```

section2-oo/code/Bicicleta.scala

OO features

Traits

```scala
trait Engine {
  val engineDisplacement: Int
  val rpm: Int = 5000
}

val engine1: Engine = new Engine {
  val engineDisplacement: Int = 2000
}

val engine2: Engine = new Engine {
  val engineDisplacement: Int = 2000
  override val rpm: Int = 4000
}
```

section2-oo/code/Bicicleta.scala

OO features
Multiple inheritance

```
trait Engine {
  val engineDisplacement: Int
  val rpm: Int = 5000
}

class MotorBike(
  cadence: Int,
  speed: Int,
  gear: Int,
  override val engineDisplacement: Int)
    extends Bike(cadence, speed, gear) with Engine
```

section2-oo/code/Bicicleta.scala

OO features
Case Classes

```
case class Bike(cadence: Int, speed: Int, gear: Int) {
  def slowDown(dec: Int): Bike =
    this.copy(speed = speed-dec)
}

val bike = Bike(1, 2, 3) // apply method in companion object

val quickBike = bike.copy(speed = 100)

val Bike(c, s, g) = quickBike // unapply
// c: Int = 1
// s: Int = 100
// g: Int = 3
```

OO features
Methods invocation (1/2)

```scala
val bike = Bike(1, 100, 3)

// def slowDown(dec: Int): Bike
bike.slowDown(10) // Bike(1, 90, 3)
bike.slowDown(dec = 10)
bike slowDown 10
bike slowDown {
  val read = readLine
  read.toInt + 5
}

// This syntax is widely used in Scala, especially
// with "operator-like" methods like the following
3 + 4 // In fact, syntactic sugar for 3.+(4)
```

OO features
Methods invocation (2/2)

```scala
// All methods ending in `:` will execute in the opposite
// direction when using the "operator syntax" invocation

val l: List[Int] = List(1, 2, 3, 4, 5)

// def ::(x: A): List[A]
l.::(0) // List(0, 1, 2, 3, 4, 5)
0 :: l

val l2: List[Int] = List(6, 7, 8)

// def :::(prefix: List[A]): List[A]
l2.:::(l1) // List(1, 2, 3, 4, 5, 6, 7, 8)
l1 ::: l2
```

section4-azucar/code/Lista.scala

OO features

The apply method

```
object Bike {
  def apply(cadence: Int): Bike =
    new Bike(cadence, 100, 200)
}

val b1 = Bike.apply(1) // Bike(1, 100, 200)
val b2 = Bike(1) // Bike(1, 100, 200)
```

OO features

Exercises

OO features
Takeaways & further reading

- Takeaways
  - *Scala as a better Java*
  - Thanks to *objects*, we don't need to use *static* modifier anymore
  - Multiple inheritance capabilities using traits
- Further reading
  - Multiple inheritance resolution Linearization
  - Dependency injection Cake Pattern

Spark programming with Scala

2. Generic Programming features

Generic Programming features
Goals

- Understand what type params are and how to use them to build generic code

Características de PG

Outline

- Generic classes
- Polymorphic methods

Generic Programming features
Generic classes (1/2)

```scala
class IntWrapper(_value: => Int) {
  lazy val value = _value
}
```

```scala
describe("IntWrapper") {
  it("should contain an integer") {
    new IntWrapper(3).value shouldBe 3
  }
}
```

section6-genericidad/code/Lista.scala

Generic Programming features
Generic classes (2/2)

```scala
class Wrapper[A](_value: => A) {
  lazy val value = _value
}
```

```scala
describe("Wrapper") {
  it("should contain any A") {
    new Wrapper(3).value shouldBe 3
    new Wrapper("hola").value shouldBe "hola"
    new Wrapper(true).value shouldBe true
  }
}
```

section6-genericidad/code/Lista.scala

Generic Programming features
Polymorphic methods

```
class Wrapper[A](_value: => A) {
  lazy val value = _value

  def map[B](f: A => B): Wrapper[B] =
    new Wrapper[B](f(value))
}

describe("Wrapper") {
  it("should change its content with the method `map`") {
    new Wrapper(3).map(_ > 0).value shouldBe true
    new Wrapper("hola").map(_.length).value shouldBe 4
    new Wrapper(true).map(identity).value shouldBe true
  }
}
```

section6-genericidad/code/Lista.scala

Generic Programming features

Exercises

Generic Programming features
Takeaways & further reading

- Takeaways
  - Genericity in Scala es very powerful, we just saw a bit of its multiple usages
- Further reading
  - Higher-kind generics
  - Learn generic collections: `List`, `Option`, `Map`, `Set`, etc.
  - The Type Astronaut's Guide to Shapeless

Spark programming with Scala

3. Implicits

Implicits
Goals

- Understand what implicits are and what kind of implicits there are:
  - Implicit arguments
  - Implicit conversions
- See all the syntax associated with implicits and all the possibilities they offer
- Find out what each of them are useful for

Implicits
Outline

- Implicit params
- Implicit conversions
- Restricted implicit conversions
- Derived implicits
- Implicit resolution mechanism
- Exercise

Implicits
Implicit params

```scala
def post(data: Array[Byte])
    (implicit uri: String, port: Int) =
  s"Posting to $uri on port $port"

post(myData)("215.15.46.26", 9000)
// res0: String = Posting to 215.15.46.26 on port 9000

implicit val URI: String = "192.168.0.1"
implicit val PORT: Int = 8080

post(myData)
// res1: String = Posting to 192.168.0.1 on port 8080
post(myData)(implicitly, 9000)
// res2: String = Posting to 192.168.0.1 on port 9000
```

section5-implicitos/code/Implicitos.scala

Implicits
Implicit conversions (1/3)

```scala
import scala.language.implicitConversions

implicit def doubleToInt(d: Double): Int = d.toInt

val myNumber: Int = 243.53
// res0: Int = 243

// Watch out! This behavior could be dangerous
```

Implicits
Implicit conversions (2/3)

```scala
class RichInt(i: Int) {
  def factorial: Int = ???
  def squared: Int = math.pow(i, 2)
  def exp(e: Int): Int = math.pow(i, e)
}

implicit def intToRichInt(i: Int): RichInt =
  new RichInt(i)

5.factorial
// res0: Int = 120
5.squared
// res1: Int = 25
2 exp 10
// res2: Int = 1024
```

section5-implicitos/code/Implicitos.scala

Implicits
Implicit conversions (3/3)

```scala
implicit class RichInt(i: Int) {
  def factorial: Int = ???
  def squared: Int = math.pow(i, 2)
  def exp(e: Int): Int = math.pow(i, e)
}
```

```scala
5.factorial
// res0: Int = 120
5.squared
// res1: Int = 25
2 exp 10
// res2: Int = 1024
```

section5-implicitos/code/Implicitos.scala

Implicits

Restricted implicit conversions

```
implicit def doubleRDDToDoubleRDDFunctions(rdd: RDD[Double]) =
  new DoubleRDDFunctions(rdd)
```

Derived implicits

```
implicit def numericRDDToDoubleRDDFunctions[T](rdd: RDD[T])
    (implicit num: Numeric[T]): DoubleRDDFunctions =
  new DoubleRDDFunctions(rdd.map(x => num.toDouble(x)))
```

Implicits

Other Spark implicit conversions

```
implicit def rddToPairRDDFunctions[K, V](rdd: RDD[(K, V)])
    (implicit kt: ClassTag[K],
              vt: ClassTag[V],
              ord: Ordering[K] = null): PairRDDFunctions[K, V] =
  new PairRDDFunctions(rdd)

implicit def rddToOrderedRDDFunctions[
    K: Ordering: ClassTag,
    V: ClassTag](rdd: RDD[(K, V)]) =
  new OrderedRDDFunctions[K, V, (K, V)](rdd)
```

Implicits

Resolution mechanism

1. Current scope
    a. Implicits defined in current scope
    b. Implicits imported
2. Associated types
    a. Companion object of the type
    b. Companion objects of params' types
    c. Companion objects of type params

Implicits
Takeaways & further reading

- Takeaways
  - Implicits are a very powerful tool that allows us to save a lot of code:
    - Avoiding setting arguments manually
    - Automatic wrapping
  - Watch out for dangerous implicit conversions
- Further reading
  - Type class design pattern
  - Implicits resolution mechanism

Implicits
Exercises

Spark programming with Scala

4. SBT (Simple Build Tool)

SBT
Outline

- Directory structure
- Build definition (provided)
- Using plugins (assembly)
- Specifying SBT version
- Common tasks
- Exercise: Launch a Spark app

SBT

Directory structure

```
src/
  main/
    resources/
       <files to include in main jar here>
    scala/
       <main Scala sources>
    java/
       <main Java sources>
  test/
    resources
       <files to include in test jar here>
    scala/
       <test Scala sources>
    java/
       <test Java sources>
build.sbt
project/
  build.properties
  plugins.sbt
  SparkSubmit.scala
target/
```

SBT
Specifying SBT version

```
project/
  build.properties
  plugins.sbt
  SparkSubmit.scala
```

```
sbt.version=0.13.15
```

SBT
Using plugins

```
project/
  build.properties
  plugins.sbt
  SparkSubmit.scala
```

```
addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.14.5")

addSbtPlugin("com.github.saurfang" % "sbt-spark-submit" % "0.0.4")
```

SBT

Using plugins (assembly)

```
> assembly
[info] ...
[info] Packaging <repo>/target/scala-2.XX/<name>-assembly-<version>.jar ...
[info] ...
```

Setting	Description
assemblyJarName in assembly := “name.jar”	Set a custom name for the jar to be generated.
test in assembly := {}	Skip the tests classes during assembly.
mainClass in assembly := Some(“com.example.Main”)	Set an explicit main class.

SBT
Using plugins (spark-submit)

```
> sparkSubmit --class org.hablapps.SparkPi --master local[*] -- 40
[info] ...
[info] Running org.apache.spark.deploy.SparkSubmit \
  --class org.hablapps.SparkPi \
  --master local[*] <repo>/target/scala-2.XX/<name>-assembly-<version>.jar 40
[info] ...
```

Arguments	Description
`--class` `<main-class>`	The entry point for your application (e.g. `org.apache.spark.examples.SparkPi`)
`--master` `<master>`	The *master URL* for the cluster (e.g. `spark://23.195.26.187:7077`)
`--conf` `<key>=<value>`	Arbitrary Spark configuration property in key=value format.
`--num-executors` `<number>`	Set the number of executors to be used.
`--executor-memory` `<size>`	Set the memory allocated for each executor.

SBT

Using plugins (spark-submit)

build.sbt

```
sparkSubmitJar := assembly.value.getAbsolutePath

SparkSubmit.settings
```

SBT

Using plugins (spark-submit)

```
project/
  build.properties
  plugins.sbt
  SparkSubmit.scala
```

```
import sbtsparksubmit.SparkSubmitPlugin.autoImport._

object SparkSubmit {
  lazy val settings = SparkSubmitSetting(
    SparkSubmitSetting("sparkPi",
      Seq("--class", "org.hablapps.SparkPi",
          "--master", "local[*]"),
      Seq("40")
    )
  )
}
```

SBT
Build definition

build.sbt

```
name := "hello"

organization := "com.example"

scalaVersion := "2.12.1"

version := "0.1.0-SNAPSHOT"

libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-sql" % "2.1.1",
  "com.datastax.spark" %% "spark-cassandra-connector" % "2.0.0")

scalacOptions ++= Seq(
  "-unchecked",
  "-deprecation",
  "-feature",
  "-language:higherKinds")
```

SBT
Common tasks

`clean`	Deletes all generated files (in the target directory).
`compile`	Compiles the main sources (in src/main/scala and src/main/java directories).
`test`	Compiles and runs all tests.
`testOnly <class>`	Compiles and run just `<class>` test.
`console`	Starts the Scala interpreter with a classpath including the compiled sources and all dependencies. To return to sbt, type :quit, Ctrl+D (Unix), or Ctrl+Z (Windows).
`run`	Runs the main class for the project in the same virtual machine as sbt.
`package`	Creates a jar file containing the files in src/main/resources and the classes compiled from src/main/scala and src/main/java.
`reload`	Reloads the build definition (build.sbt, project/*.scala, project/*.sbt files). Needed if you change the build definition.
`~<task>`	Make a `<task>` run automatically when one or more source files change.

Spark programming with Scala

5. ScalaTest

ScalaTest
Outline

- FunSpec basic structure
- Matchers
- BeforeAndAfter
- Spark bindings & techniques
- Exercise: Test a Spark app

ScalaTest
Goals

- Learn how to define simple tests with ScalaTest
- Take a close look at the Matchers DSL
- Understand how to allocate and clean resources using BeforeAndAfter
- Play with some of the spark plugins related with ScalaTest (spark-testing-base)

ScalaTest
FunSpec

```scala
import org.scalatest.FunSpec

class SetSpec extends FunSpec {

  describe("A Set") {
    describe("when empty") {
      it("should have size 0") {
        assert(Set.empty.size == 0)
      }

      it("should produce NoSuchElementException when head is invoked") {
        assertThrows[NoSuchElementException] {
          Set.empty.head
        }
      }
    }
  }
}
```

section2-oo/code/Bicicleta.scala

ScalaTest
Matchers (Common matchers I)

shouldBe	Checks that two expressions are equal
should not be	Opposite of `shouldBe`
should have length	Checks that the object has a certain length. This object can be any collection or any other type that has a length
should have size	Checks that the object has a certain size. This object can be any collection or any other type that has a size
should be [>,<,<=,>=]	Checks whether a value is greater than, less than, greater than or equal, or less than or equal to another value
should startWith endWith include	Checks whether a `String` starts with, ends with, or includes a substring

section2-oo/code/Bicicleta.scala

ScalaTest
Matchers (Common matchers II)

`shouldBe a [T]`	Checks that an object is an instance of a particular class or trait
`shouldBe empty`	Checks whether an object is empty
`should contain`	Checks whether a collection contains a particular element
`shouldBe sorted`	Checks whether the elements of “sortable” objects are in sorted order
`shouldNot compile`	Checks that a snippet of code does not compile
`shouldNot typeCheck`	Checks that a snippet of code does not compile because of a type error

ScalaTest
Matchers (Common matchers III)

st

`should compile`	Checks that a snippet of code compiles
`shouldBe defined`	Checks that an `Option` is defined
`inside(<CC>) { case ... => <matcher> }`	Allows you to make assertions after a pattern match
`should matchPattern`	Checks that an expression matches a pattern
`an [Exception] should be thrownBy`	Checks whether an expression throws an expected `Exception`
`the [Exception] thrownBy`	Captures the exception to be inspected afterwards
`noException should be thrownBy`	Checks that no exception is thrown by some expression

ScalaTest
Matchers (Custom matchers)

```
import org.scalatest._
import matchers._

trait CustomMatchers {

  class FileEndsWithExtensionMatcher(expectedExtension: String)
      extends Matcher[java.io.File] {

    def apply(left: java.io.File): MatchResult = {
      val name = left.getName
      MatchResult(
        name.endsWith(expectedExtension),
        s"""File $name did not end with extension "$expectedExtension"""",
        s"""File $name ended with extension "$expectedExtension""""
      )
    }
  }

  def endWithExtension(expectedExtension: String) =
    new FileEndsWithExtensionMatcher(expectedExtension)
}

object CustomMatchers extends CustomMatchers
```

ScalaTest

FunSpec + Matchers

```scala
import org.scalatest.{FunSpec, Matchers}

class SetSpec extends FunSpec with Matchers {

  describe("A Set") {
    describe("when empty") {
      it("should have size 0") {
        // assert(Set.empty.size == 0)
        Set.empty.size shouldBe 0
        Set.empty should have size 0
        Set.empty shouldBe empty
      }

      it("should produce NoSuchElementException when head is invoked") {
        // assertThrows[NoSuchElementException] {
        a [NoSuchElementException] should be thrownBy {
          Set.empty.head
        }
      }
    }
  }
}
```

section2-oo/code/Bicicleta.scala

ScalaTest
BeforeAndAfter

```scala
trait LocalSparkContext extends BeforeAndAfterAll { this: Suite =>

  @transient private var _sc: SparkContext = _
  def sc = _sc

  val conf = new SparkConf().setMaster("local[*]").setAppName("test")

  override def beforeAll() {
    _sc = new SparkContext(conf)
    super.beforeAll()
  }

  override def afterAll() {
    // We clear the driver port so that we don't try and bind to the same port on
    // restart.
    sc.stop()
    System.clearProperty("spark.driver.port")
    _sc = null
    super.afterAll()
  }
}
```

section2-oo/code/Bicicleta.scala

ScalaTest
BeforeAndAfter

```
class SimpleSparkTest extends FunSpec with Matchers with LocalSparkContext {

  describe("A RDD[Int]") {
    it("should calculate the sum correctly if it's not empty") {
      sc.parallelize(1 :: 2 :: 3 :: 4 :: Nil).sum shouldBe 10
    }
    it("should calculate the sum correctly if it's empty") {
      sc.parallelize(List.empty[Int]).sum shouldBe 0
    }
  }

}
```

section2-oo/code/Bicicleta.scala

ScalaTest
Spark bindings (spark-testing-base)

```
libraryDependencies +=
  "com.holdenkarau" %% "spark-testing-base" % "2.1.0_0.6.0" % "test"

parallelExecution in Test := false
```

```
import com.holdenkarau.spark.testing.SharedSparkContext

class SimpleSparkTest extends FunSpec with Matchers with SharedSparkContext {

  describe("A RDD[Int]") {
    it("should calculate the sum correctly if it's not empty") {
      sc.parallelize(1 :: 2 :: 3 :: 4 :: Nil).sum shouldBe 10
    }
    it("should calculate the sum correctly if it's empty") {
      sc.parallelize(List.empty[Int]).sum shouldBe 0
    }
  }
}
```

ScalaTest

Spark bindings (spark-testing-base)

```
import com.holdenkarau.spark.testing.RDDGenerator

class RDDsCheck extends FunSuite with with SharedSparkContext with Checkers {
  test("map should not change number of elements") {
    val property =
      forAll(RDDGenerator.genRDD[String](sc)(Arbitrary.arbitrary[String])) {
        rdd => rdd.map(_.length).count() == rdd.count()
       }

    check(property)
  }
}
```

Implicits
Exercises

SBT

Directory structure

```
src/
  main/
    resources/
       <files to include in main jar here>
    scala/
       <main Scala sources>
    java/
       <main Java sources>
  test/
    resources
       <files to include in test jar here>
    scala/
       <test Scala sources>
    java/
       <test Java sources>
build.sbt
project/
  build.properties
  plugins.sbt
  SparkSubmit.scala
target/
```

SBT
Specifying SBT version

```
project/
  build.properties
  plugins.sbt
  SparkSubmit.scala
```

```
sbt.version=0.13.15
```