

هو الحکیم

شرکت پوش داده نوین

کارگاه اصول برنامه نویسی

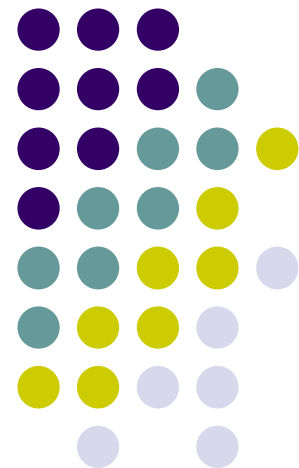
Hadoop & Spark(Scala)

ارائه کننده : امین نظارات

عضو هیئت علمی پیام نور

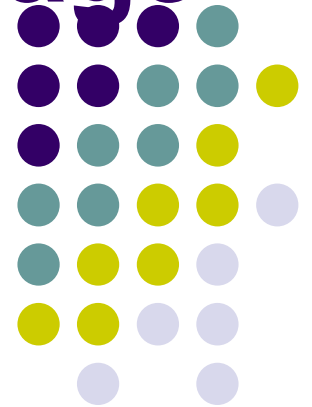
[aminnezarat@gmail.com](mailto:aminnezarat@gmail.com)

[www.hpclab.ir](http://www.hpclab.ir)



# Scala programming language

---





# Scala on JVM

- scalac compiles Scala to Java bytecode
  - (regular .class files)
- **Any Java class can be used from Scala**



شرکت خدمات مهندسی  
پیش داده نوین

# Origin



- Started at 2001 by Martin Odersky at EPFL ●  
Lausanne, Switzerland
- Scala 2.0 released in 2006 ●
- Current version 2.7 ●
  
- Twitter backend runs on Scala ●



شرکت خدمات مهندسی  
پدین داده نوین

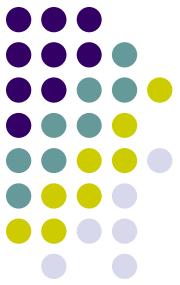
# Scala properties



- Object oriented ●
- Statically typed ●
- Functional & imperative ●



# Static typing



- Type checking done at compile time
- Type associated with variable, not value
  - Better tools possible
- More verbose code compared to dynamic language
- Can't add methods to class at runtime
  - No duck typing – really?



شرکت خدمات مهندسی  
پویا داد نوین

# Functional programming



- Functions are first class citizens
- Immutability
- Tuples
- Currying
- Recursion
- Monads



شرکت خدمات مهندسی  
پوشش داده نوین

# Introduction



*Demo of Scala interpreter*





# Variables & values, type inference

شرکت خدمات مهندسی  
پوشش داده نوین



```
// msg is mutable
```

```
// compiler error
```

```
var msg = "Hello"  
msg += " world"  
msg = 5;
```



# Variables & values, type



شرکت خدمات مهندسی  
پوشش داده نوین

## inference

```
// msg is immutable  
// compiler error
```

```
val msg = "Hello world"  
msg += " world"
```

```
// explicit type declaration val n : Int = 3  
var n2 : Int = 3
```



# Immutability



Why? ●

- Immutable objects are automatically thread-safe  
(you don't have to worry about object being changed by another thread)
- Compiler can reason better about immutable values -> optimization
- Steve Jenson from Twitter: *“Start with immutability, then use mutability where you find appropriate.”*



# Calling Java from Scala

Any Java class can be used seamlessly •

```
import java.io._  
val url = new URL("http://www.scala-lang.org")
```

*demo*



# Methods



```
def max(x : Int, y : Int) = if (x > y) x else y
```

// equivalent:

```
def neg(x : Int) : Int = -x
```

```
def neg(x : Int) : Int = { return -x; }
```



شرکت خدمات مهندسی  
پویا داد نوین

# Types



Int, Double, String, Char, Byte, BigInt, ... ●  
wrappers around Java types ●



# Lists



Lists are immutable (= contents cannot be changed) •

List[**String**] contains Strings •

```
val lst = List("b", "c", "d")
lst.head // "b"
lst.tail // List("c", "d")
val lst2 = "a" :: lst // cons operator
```



Nil = synonym for empty list ●

```
val l = 1 :: 2 :: 3 :: Nil
```

List concatenation ●

```
val l2 = List(1, 2, 3) ::: List(4, 5)
```





# Foreach



```
val list3 = List("mff", "cuni", "cz")
```

Following 3 calls are equivalent ●

```
list.foreach((s : String) => println(s))
```

```
list.foreach(s => println(s))
```

```
list.foreach(println)
```



شرکت خدمات مهندسی  
پویا داد نوین

# For comprehensions



```
for (s <- list)
  println(s)
```

```
for (s <- list if s.length() == 4)
  println(s)
```

for just calls foreach •



شرکت خدمات مهندسی

پارس داده نوین

# Arrays



Lists are immutable, arrays are mutable ●

```
val a = Array("Java", "rocks")  
a(0) = "Scala";
```



# Covariance



Lists are covariant, arrays are invariant •

```
val array : Array[Any] = Array(1, 2, 3); // compiler error
```

```
val list : List[Any] = List(1, 2, 3); // ok
```



شرکت خدمات مهندسی

پارس داده نوین

# Arrays



```
val greets = new Array[String](2)
    greets(0) = "Hello"
    greets(1) = "world!\n"
    for (i <- 0 to 1)
        print(greets(i))
```



# Arrays are no special type

`greet.apply(i) === greet(i)`

`=== greet.update(i, greet(i) = "Hi"  
"Hi")`

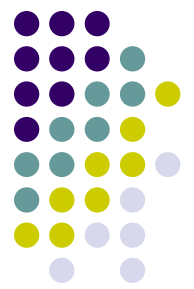
Any class that defines `apply` / `update` can be used like this •



شرکت خدمات مهندسی  
پویستر داده نوین

# Every operation is a method

# call



“to” is not a keyword ●

for (i <- 0 **to** 2) print(greets(i)) ●

0 to 2 === 0.to(2) ●

$x - 1$  ===  $x.-(1)$  ●

map containsKey 'a' === ●

map.containsKey('a')



# Associativity



If method name ends with colon, the method ●  
is invoked on the right operand

```
val list = List("b", "c")  
"a" :: list      ===      list.::("a")
```





# Performance



- Scala treats everything as objects
- no primitive types, no arrays

- So this comes with a cost, right?
- Usually not, the scalac compiler uses Java primitive types and arrays where possible



شرکت خدمات مهندسی

پارس داده نوین

# Anonymous functions



```
val l = new List("mff", "cuni", "cz")  
l.filter(s => s.length == 4)
```

```
val l = List[Person](new Person(...), ...)  
l.sort((p1, p2) => p1.lastName < p2.lastName)
```



شرکت خدمات مهندسی  
پویا داد نوین

# Currying



Function with only some arguments specified •  
=

function expecting the rest of the arguments

Common concept in functional languages •



شرکت خدمات مهندسی  
پویا داد نوین

# Currying



```
// Does n divide m?  
def nDividesM(m : Int)(n : Int) = (n % m == 0)  
  
// Currying,  
// isEven is of type (Int) => Boolean  
val isEven = nDividesM(2)_  
  
println(isEven(4))  
println(isEven(5))
```



# Tuples



Sequence of elements with different types •

```
(10, List('c', 'm'), "cache");
```

type of this expression is Tuple3[Int, List[Char],  
String] •



# Tuples



```
def divMod(x : Int, y : Int) = (x / y, x % y)
```

```
val dm = divMod(10, 3)    // Tuple2[Int, Int]
                           dm._1      // 3
                           dm._2      // 1
```

```
val (d, m) = divMod(10, 3);
// 3 1 println(d + " " + m);
```



شرکت خدمات مهندسی  
پریش دادنه نوین

# Pattern matching



- Like switch statement
- But much more powerful



# Pattern matching



```
def flatten(list: List[Any]) : List[Any] =  
    list match {  
        case (x: List[Any]) :: xs =>  
            flatten(x) ::: flatten(xs)  
        case x :: xs => x :: flatten(xs)  
        case Nil => Nil  
    }
```

```
val nested = List(1, List(2, 3), 4);  
val flat = flatten(nested); // List(1, 2, 3, 4)
```





شرکت خدمات مهندسی  
بویشا داده نوین

# Classes



```
/** A Person class.  
 * Constructor parameters become  
 * public members of the class.*/  
class Person(val name: String, var age: Int) {  
    if (age < 0) {  
        throw ...  
    }  
}  
  
var p = new Person("Peter", 21);  
    p.age += 1;
```



شرکت خدمات مهندسی  
پویا داد نوین

# Objects



- Scala's way for "statics"
- not quite – see next slide
- (in Scala, there is no *static* keyword)

- "Companion object" for a class
- = object with same name as the class

*demo*



# Objects



```
// we declare singleton object "Person"
// this is a companion object of class Person
    object Person {
        def defaultName() = "nobody"
    }
class Person(val name: String, var age: Int) {
    def getName() : String = name
}

// surprise, Person is really an object
val singleton : Person = Person;
```



# Case classes



Implicitly override toString, equals, hashCode •  
take object's structure into account •

```
abstract class Expr  
case class Number(n: Int) extends Expr  
case class Sum(e1: Expr, e2: Expr) extends Expr
```

```
// true thanks to overridden equals  
Sum(Number(1), Number(2)) ==  
Sum(Number(1), Number(2))
```



# Case classes



Needed if we want to pattern match on class hierarchies

```
def eval(e: Expr): Int = e match {  
    case Number(n) => n  
    case Sum(l, r) => eval(l) + eval(r)  
}
```



شرکت خدمات مهندسی  
پویا داده نوین

# Exceptions



```
object Main {  
  def main(args: Array[String]) {  
    try {  
      val elems = args.map(Integer.parseInt)  
      println("Sum is: " + elems.foldRight(0) (_ + _))  
    }  
    catch {  
      case e: NumberFormatException =>  
        println("Usage: scala Main <n1> <n2> ... ")  
    }  
  }  
}
```



شرکت خدمات مهندسی  
پویا داد نوین

# Traits





شرکت خدمات مهندسی  
پویا داد نوین

# Traits



- Like Java interfaces
- But can contain implementations and fields

```
trait Pet {  
  var age: Int = 0  
  def greet(): String = {  
    return "Hi"  
  }  
}
```





# Extending traits



```
class Dog extends Pet {  
  override def greet() = "Woof"  
}
```

```
trait ExclamatoryGreeter extends Pet {  
  override def greet() = super.greet() + " !"  
}
```



# Traits - mixins

Traits can be “mixed in” at instation time ●

```
trait ExclamatoryGreeter extends Pet {  
  override def greet() = super.greet() + " !"  
}
```

```
val pet = new Dog with ExclamatoryGreeter  
// woof ! println(pet.greet())
```

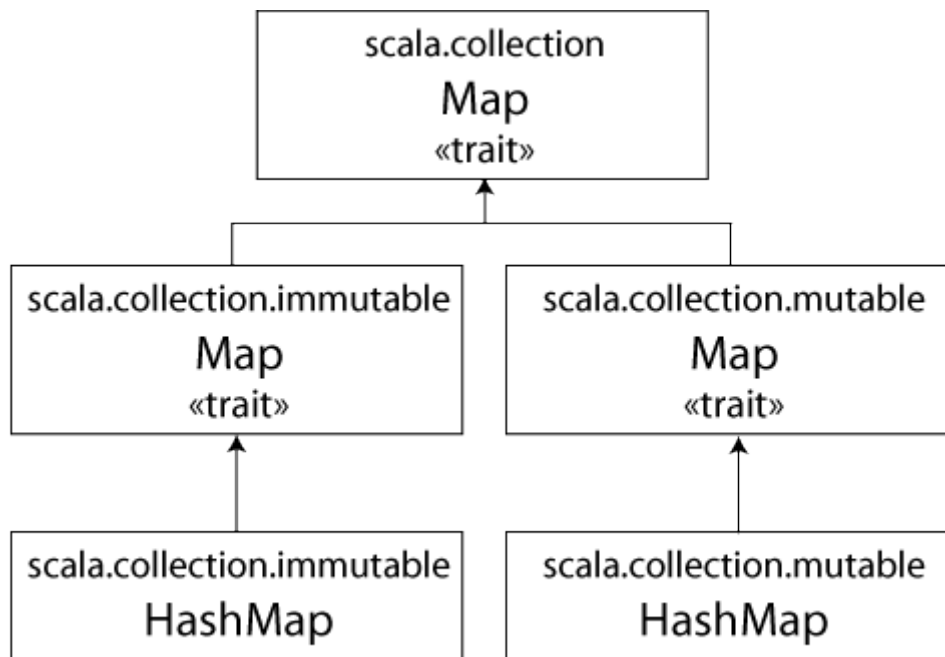


# Traits – common use

```
trait Ordered[A] {  
    // abstract method  
    def compare(that: A): Int  
  
    def < (that: A): Boolean = (this compare that) < 0  
    def > (that: A): Boolean = (this compare that) > 0  
}  
  
class Health(val value : Int) extends Ordered[Health]  
    {  
    override def compare(other : Health) = {  
        this.value - other.value; }  
    def isCritical() = ...  
    }
```



# Maps and Sets





# Map – simple example

```
import scala.collection._
```

```
val cache = new mutable.HashMap[String, String];  
cache += "foo" -> "bar";
```

```
val c = cache("foo");
```

The rest of Map and Set interface looks as you would expect •



# ListBuffer



- ListBuffer[T] is a mutable List ●
- Like Java's ArrayList<T> ●

```
import scala.collection.mutable._  
  
val list = new ListBuffer[String]  
    list += "vicky"  
    list += "Christina"  
  
val str = list(0)
```



شرکت خدمات مهندسی  
پیش داده نوین

# Option



Like “Maybe” in Haskell ●

Example – 3 state Boolean ●

```
var sure : Option[Boolean] = Some(false);  
        sure = Some(true);  
        sure = None;
```

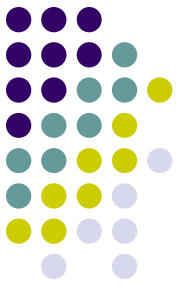


شرکت خدمات مهندسی  
پارس داد نوین

# Actors







Concurrency using threads is hard ●  
Shared state – locks, race conditions, deadlocks ●

Solution – **message passing + no shared state** ●

Inspired by Erlang language ●

Erlang used at Ericsson since 1987, open source ●  
since 1998

Facebook chat backend runs on Erlang ●



# What is an actor



- Actor is an object that receives messages
- Actor has a *mailbox* – queue of incoming messages
- Message send is by default *asynchronous*
- Sending a message to an actor immediately returns



شرکت خدمات مهندسی  
پارس داده نوین

# Actors – trivial example



We define messages ●

```
case object MsgPing
```

```
case object MsgPong
```

```
case object MsgStop
```



# Actors – trivial example



```
class Ping(count: Int, pong: Actor) extends Actor {
    def act() {
        var pingsSent = 0
        println("Ping: sending ping " + pingsSent)
        pong ! MsgPing; pingsSent += 1
        while(true) {
            receive {
                case MsgPong =>
                    if (pingsSent < count) {
                        if (pingsSent % 1000 == 0)
                            println("Ping: sending ping " + pingsSent)
                        pong ! MsgPing; pingsSent += 1
                    } else {
                        println("Ping: sending stop")
                        pong ! MsgStop
                        exit()
                    }
            }
        }
    }
}
```



# Actors – trivial example



```
class Pong extends Actor {  
    def act() {  
        var pongCount = 0  
        while(true) {  
            receive {  
                case MsgPing =>  
                    if (pongCount % 1000 == 0)  
                        println("Pong: replying " + pongCount)  
                        sender ! MsgPong  
                        pongCount += 1  
                case MsgStop =>  
                    println("Pong: stop")  
                    exit()  
            }  
        }  
    }  
}
```



# Actors – trivial example

```
val pong = new Pong
val ping = new Ping(100000, pong)
ping.start
pong.start
```

// any following code here is not  
blocked by the actors, each Actor  
(Ping, Pong) runs in his own thread



# Actors – what else is available?

- actor ! message - asynchronous send ●
- synchronous send actor !? message ●  
(awaits reply)
- actor !! message - asynchronous, returns ●  
*future* object
- future object can be used later to get the result ●



# Creating “keywords”

From actors example, it seems that Scala has built-in keywords like *receive { }* or *!* Not true – actors are implemented as a library

We already know that

`pong ! MsgPing` is equivalent to  
`pong.!(MsgPing)` // *!* is a method of Actor class





# Creating “keywords”

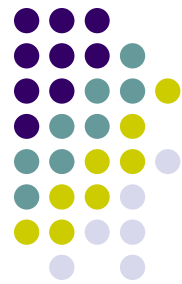


- Moreover, receive is just a method of Actor class
- Method arguments can be passed in curly braces
- Ability to create DSL-like languages

```
receive {  
  case MsgPong =>  
    ...  
}
```



# Creating keywords - lock in



شرکت خدمات مهندسی

پیش داده نوین

# Java

```
String x = "No"
```

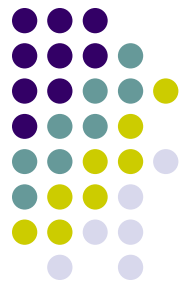
```
l.lock();  
    try {  
        x = "Yes"  
    } finally {  
        l.unlock();  
    }
```



# Creating keywords - lock in

شرکت خدمات مهندسی  
پارس داده نوین

## Scala



```
var x = "No"
```

```
lock(1) {  
  x = "Yes"  
}
```



# Lock “keyword” implementation



Lock “keyword” is really an ordinary method •

```
// f is a function (piece of code) returning
// Unit (ie. void)
def lock(l : Lock)(f : => Unit) = {
    l.lock();
    try {
        // call f
        f
    } finally {
        l.unlock();
    }
}
```



شرکت خدمات مهندسی  
پریش دادنه نوین

# Parallelism





شرکت خدمات مهندسی  
پریش دادنه نوین

# Parallelism



What about parallelMap, parallelReduce etc. ●  
?

Not present in Scala library yet ☹ ●

Have to implement own versions ●



شرکت خدمات مهندسی  
پویا داد نوین

# Little more advanced





# What exactly is the List?

List is an abstract class with 2 descendant case classes:

Nil

::

What gets called for List(1, 2, 3) ?

```
object List {  
  // * means variable arguments  
  def apply[A](xs: A*): List[A] = xs.toList
```





# scala.Seq



- scala.Seq is the supertype that defines methods like:  
filter, fold, map, reduce, take, contains, ...
- List, Array, Maps... descend from Seq



# Yield, iterators

- Syntax sugar for returning iterator object
- Iterators allow to iterate over a sequence of elements. They have `hasNext()` and `next()` methods.
- Lazy evaluation
  - when `olderThan21` is called, the for loop is **not** executed

```
def olderThan21(xs: Iterator[Person]): Iterator[String] =  
  {  
    for (p <- xs if p.age > 21) yield p.getName()  
  }
```



# Matching generic arguments?



Will this compile? ●

```
def genMatch(list: List[Any]) : String =  
    list match {  
        case (x: List[Int]) => "ints"  
        case (x: List[String]) => "strings"  
    }
```



# Matching generic arguments?



JVM has no runtime support for generics  
(compiler uses erasure)

```
def genMatch(list: List[Any]) : String =  
    list match {  
    // warning: type argument is unchecked  
    case (x: List[Int]) => "ints"  
    // error: unreachable code  
    case (x: List[String]) => "strings"  
    }
```



# Adding methods to classes



- Possible in dynamic languages (even at runtime)
- Possible using Extension methods in C#  
● (just syntax sugar for static methods)
- How to do it in Scala?



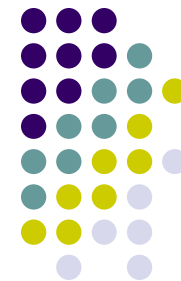
# “Adding methods” to classes

ScalaTest test framework ●  
map should have value 7

// legal scala code

We want to be able to call map.should ●  
map does not have a “should” method ●

Solution – wrapper object ●  
class Wrapper(wrappedObject : Any) {  
    def should() ...  
}



# “Adding methods” to classes

```
class Wrapper(wrappedObject : Any) {  
    def added() { ... }  
}
```

Define implicit conversion method Any -> ●  
Wrapper

```
implicit def wrap(o : Any) = new Wrapper(o)
```

object.added()      compiles as  
wrap(object).added()



# “Adding methods” - demo

```
class CollectionWrapper[T](wrappedCollection : java.util.Collection[T]) {  
    def join(delim : String) : String = {  
        val iter = wrappedCollection.iterator();  
        val buffer = new StringBuffer(iter.next().toString());  
        while (iter.hasNext()) buffer.append(delim).append(iter.next().toString());  
        return buffer.toString();  
    }  
}  
  
implicit def wrapCollection[T](o : java.util.Collection[T]) = new  
    CollectionWrapper(o)  
  
var javaList = new java.util.ArrayList[String]();  
// same as wrapCollection(javaList).join("-")        println(javaList.join("-"));
```





# Structural types



`{ val length : Int }` ●

any object that has length field ●

`{ def length() : Int }` ●

any object that has length() method ●

Duck typing ●

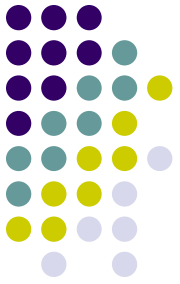
Invoking methods on the object uses ●

reflection - slower



شرکت خدمات مهندسی  
پویا داد نوین

# Traits – diamond inheritance?





```
import scala.xml._
```

```
val df = java.text.DateFormat.getDateInstance()  
val dateString = df.format(new java.util.Date())
```

```
def theDate(name: String) =  
  <dateMsg addressedTo={ name }>  
    Hello, { name }! Today is { dateString }  
  </dateMsg>;
```

```
println(theDate("Amin Nezarat").toString())
```



شرکت مهندسی شبکه  
پویا داده نوین



# آرزوی سلامتی و پیروزی

## پرسش؟

[info@pdnsoft.com](mailto:info@pdnsoft.com)