specs and Scala

Tips and tricks for a friendly DSL syntax


Menu
Tuesday 05 Ju
7/21/10
Venoge, N.V.
Sparkling Wine:
Charles de Fere Brut
Reserve, N.V.
Marques De Monistral
Starters
Main Course
Dessert
About…
specs tour
Implicit def
Restrict
Combine
Add , add
Be lazy
!&%$#>
Manifests
Resources

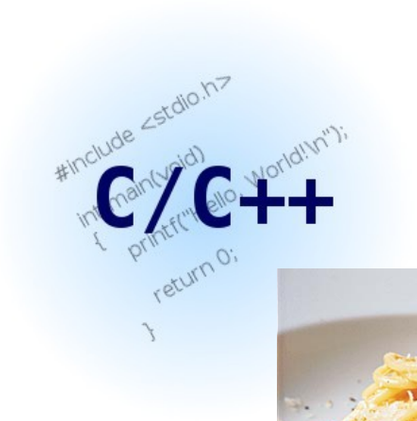
About...

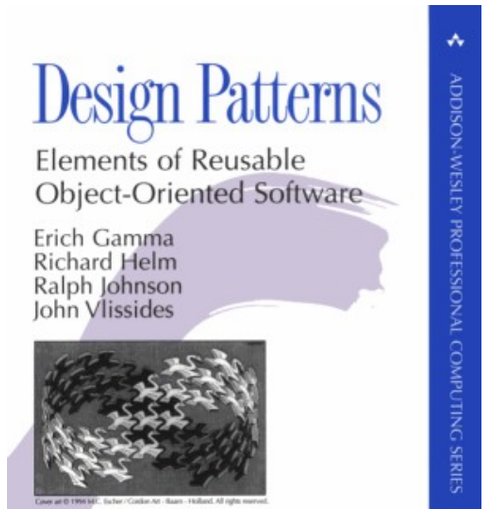

Atlantic Ocean
U.K.
North Sea
English Channel
BELG.
GERMANY
LUX.
Bay of Biscay
FRANCE
SWITZ
Nice
ANDORRA
SPAIN
Mediterranean Sea

About…

Photo: Tourism Australia

About...

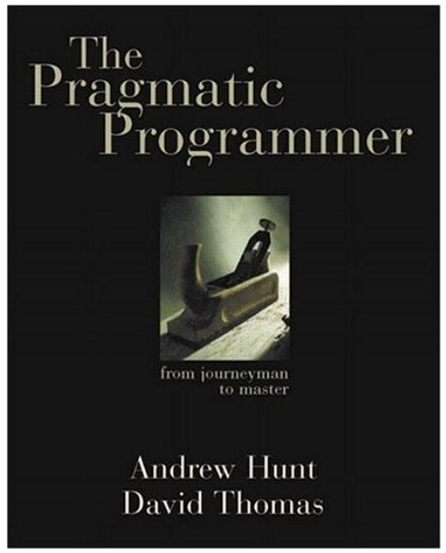

#include <stdio.h>
int main(void)
{
printf("Hello, World!\n");
return 0;
}
C/C++


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES


UNIFIED
MODELING
LANGUAGE
UML
TM


Java™

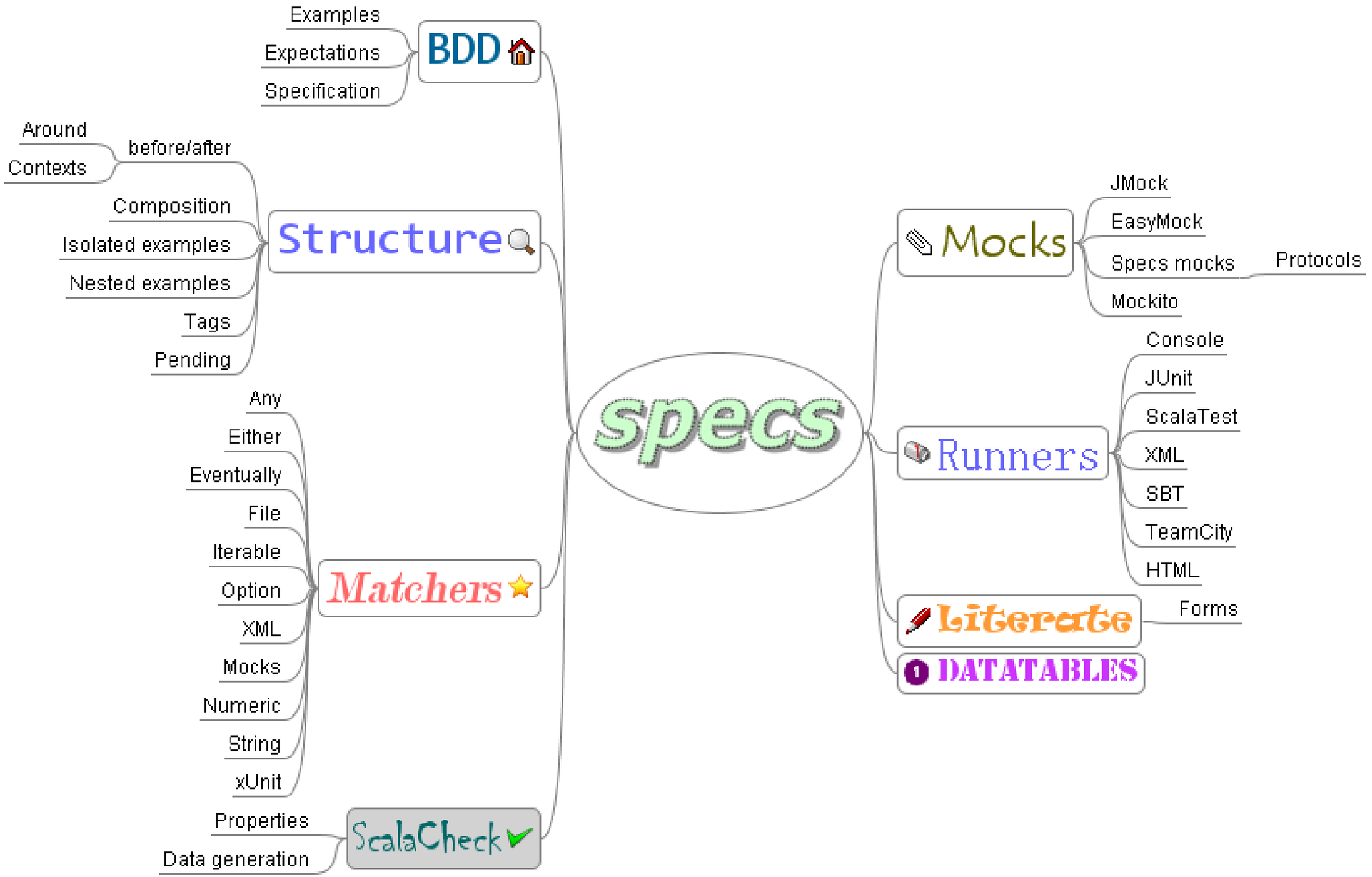

J2EE
JAVA 2 PLATFORM
ENTERPRICE EDITION


The
Pragmatic
Programmer
from journeyman
to master
Andrew Hunt
David Thomas


Groovy


Scala

specs tour

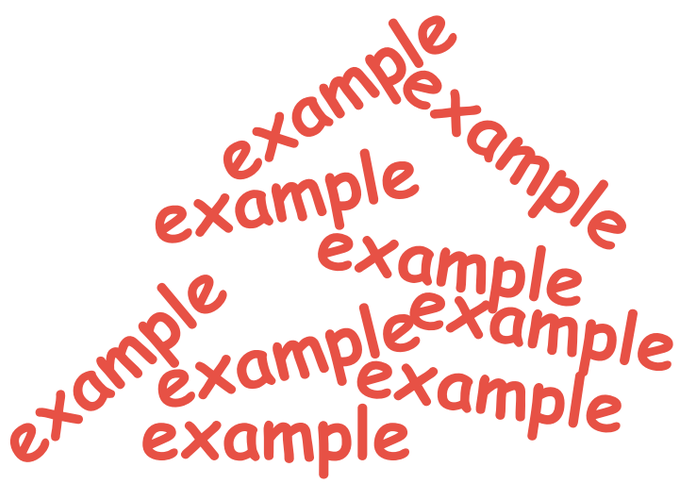


specs tour

~~Test~~

Specification

specify



specs tour

```
import IncredibleStringReverser._

class ReverserSpec extends Specification {
  "a reversed empty string must be empty" in {
    reverse("") must_== ""
  }
  "a reversed empty string must really *be empty*" in {
    reverse("") must be empty
  }
  "a reversed string must be reversed abc -> cba" in {
    reverse("abc") must be_==("cba")
  }
  "a longer string must also be reversed. Whoops!" in {
    reverse("abcdef") must be_==("xxxxx")
  }
}
```

specs tour

```
"a reversed empty string must be empty" in {
  reverse("") must be empty
}
```

specs tour

```
Specification "ReverserSpec"

  + a reversed empty string must be empty
  + a reversed empty string must really *be empty*
  + a reversed string must be reversed abc -> cba
  x a longer string must also be reversed. Whoops!
    'fedcba' is not equal to 'xxxxx'
(ReverserSpec.scala:17)

Total for specification "ReverserSpec":
Finished in 0 second, 140 ms
4 examples, 4 expectations, 1 failure, 0 error
```

specs tour

repetitio
n repetitio
n repetitio
n repetitio
n repetitio
n repetitio
n

organiz
e

System
Under
Specificatio
n

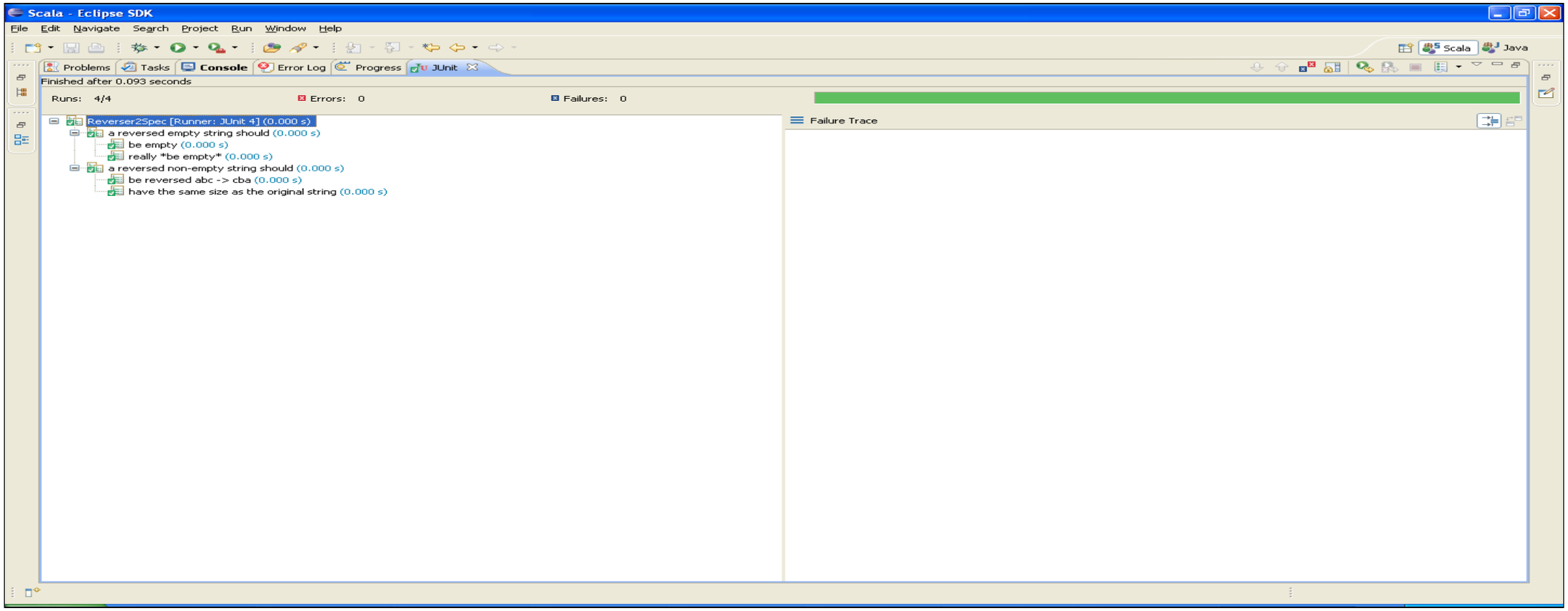
specs tour

```
class Reverser2Spec extends Specification {
  "a reversed empty string" should {
    val reversed = reverse("")
    "be empty" in {
      reversed must_== ""
    }
    "really *be empty*" in {
      reversed must be empty
    }
  }
  "a reversed non-empty string" should {
    val reversed = reverse("abc")
    "be reversed abc -> cba" in {
      reversed must be_==("cba")
    }
    "have the same size as the original string" in {
      reversed must have size(3)
    }
  }
}
```

specs tour

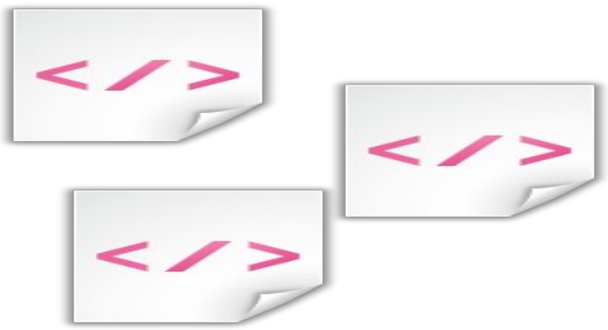
```
"a reversed empty string" should {
   val reversed = reverse("")
   …
}

"a reversed non-empty string" should {
   val reversed = reverse("abc")
   …
}
```

specs tour



specs

tour



Be specific

Matchers

specs tour

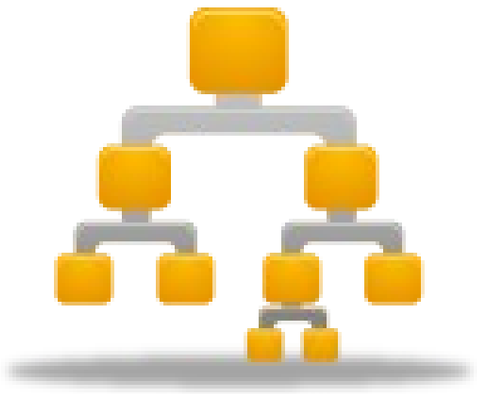
```
trait HelloWorldSnippet {
  def hello(s: String)=
          <div class="txt">Hello {s}</div>
}
```

```
"the snippet must output a div element" >> {
  hello("Eric") must \\(<div/>)
}
"it must have an attribute class=txt" >> {
  hello("You") must \\(<div/>, "class" -> "txt")
}
```

specs tour

```
hello("Eric") must \\(<div/>)
```

specs tour



Be exhaustive

Properties

specs tour

```
class Reverser3Spec extends Specification with ScalaCheck {

  "reverse must preserve the length of a string" verifies {
    s: String => reverse(s).size == s.size
  }
  "reverse applied twice must return the same string" verifies {
    s: String => reverse(reverse(s)) == s
  }
  "reverse 2 concatenated strings must return the reversed second
   string concatenated with the reversed first one" verifies {
      (s1: String, s2: String) => reverse(s1 + s2) ==
                                  reverse(s2) + reverse(s1)
  }
  def center(s: String) = s(s.size / 2)
  "keep the same 'center' character - Whoops!" verifies {
    s: String => s.isEmpty || center(reverse(s)) == center(s)
  }
}
```

specs tour

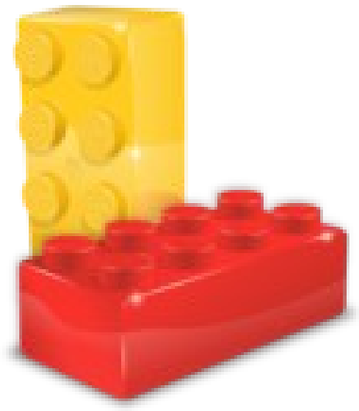
```
"reverse applied twice must return " +
"the same string" verifies {

  s: String => reverse(reverse(s)) == s

}
```

specs tour

```
x keep the same 'center' character - Whoops!
  A counter-example is 'bc'
  (after 1 try - shrinked ('bcab' -> 'bc'))
```

specs

tour



Isolat
e

Mocks

specs tour

```
/**
 * A simple Observable-Observer implementation
 */
trait Observable {
  private var observers: List[Observer] = Nil

  def add(o: Observer) =
            observers = o :: observers

  def changed(event: String) =
     observers foreach (_.notify(event))
}
trait Observer {
  def notify(event: String)
}
```

specs tour

```
class ObservableSpec extends Specification with Mockito {
  val observer = mock[Observer]
  val observable = new Observable { add(observer) }

  "An observable notifies its observers if changed" >> {
    observable.changed("event")
    there was one(observer).notify("event")
  }
  "Each change event is notified" >> {
    observable.changed("event1")
    observable.changed("event2")
    there was two(observer).notify(startWith("event"))
  }
}
```

specs tour

```
"An observable notifies its observers if " +
"changed" >> {
  observable.changed("event")

  there was one(observer).notify("event")
}
```

Scala DSL

How does it work?

Scala DSL

The best tool in the box

```
"This is ok" in {
   1 + 1
}
// is the same as
"This is ok".in(1 + 1)
```

"In" *method on String?!*

Scala DSL

The best tool in the box

```
class Example(description: String) {
  def in(e: Any) = e
}

implicit def forExample(d: String) = {
  new Example(d)
}
```

Scala DSL

Naming

```
forExample("this works") in {
  1 + 1
}
```

Scala DSL

Some examples

```
"This is true" in {
  true must beTrue
}
```

```
3.seconds
3 seconds
3 times { i => println(i) }
```

About…

specs tour

Implicit def

Restrict

Combine

Add , add

Be lazy

!&%$#>

Manifests

Resources

Scala DSL

Restric

Scala DSL

Restric
t

```
class Example(description: String) {
  def in(e: Any) = expectations
  def tag(t: String) = this
}
```

Scala DSL

Restric t

```
"this is a" tag ("really?")
```

Scala DSL

Restrict

```
class ExampleDesc(description: String) {
  def in(e: Any): Example = new
    Example(description, e)
  }
}

class Example(d: String, e: Any) {
  def tag(t: String): Example = this
}
```

Scala DSL

Combine

Scala DSL

Combine

```
trait Matcher[-T] {
  def apply(y: =>T): (Boolean, String, String)
  def not: Matcher[T]
  def when(condition: =>Boolean): Matcher[T]
  def unless(condition: =>Boolean): Matcher[T]
  def orSkip: Matcher[T]
  def or[S <: T](m: Matcher[S]): Matcher[S]
  def xor[S <: T](m: Matcher[S]): Matcher[S]
  def and[S <: T](m: Matcher[S]): Matcher[S]
  def ^^[S <: T, U](f: S => U): Matcher[U]
  def toIterable: Matcher[Iterable[T]]
}
```

Scala DSL

Some examples

```
def beFalse = beTrue.not

condition must beTrue or beFalse

condition must beTrue.unless(condition2)

def trimmedSize(i: Int) = size(i) ^^ (_.trim)

string must have trimmedSize(3)
```

Scala DSL

Add, add,

Scala DSL

Repeated parameters

```
include(spec1)
include(spec1, spec2)
```

```
1 must beOneOf(1, 2, 3)
```

Scala DSL

Use and pass

```
result + 1      // result = ?
result.pp + 1 // print and pass
```

Scala DSL

Use and pass

```
Customers.findBy(_.age >= 18)
   aka "adult customers"
   must haveSize(3)
```

```
> adult customers 'Bob, Lee'
  doesn't have size 3
```

Scala DSL

this.type

```
class BigSpec extends Spec {
  // I want to "tag" the
  // slow spec as slow
  include(slowSpec, fastSpec)
}
```

Scala DSL

this.type

```
class BigSpec extends Spec {
  include(slow tag "slow", fast)
}
```

Scala DSL

this.type

```
class Spec extends Tagged {
   def include(other: Spec*) = …
}

trait Tagged {
   def tag(t: String): ?
}
```

Scala DSL

this.type

```
// store the tag and return this
def tag(t: String): this.type = this
```

Scala DSL

Configuration

```
val prop = forAll { (s: String) =>
  reverse(reverse(s)) == s
}

"With the default configuration" in {
  prop must pass
}
```

Scala DSL

Configuration

```
"With a 3 tests ok" in {
  prop must pass(set(minTestsOk -> 3))
}
```

```
"With 3 tests ok - in the console" in {
  prop must pass(display(minTestsOk -> 3))
}
```

Scala DSL

Implicit parameters

```
def pass(implicit p: Params) = {
   new Matcher[Prop] {
      def apply(prop: =>Prop) = check(prop)(p)
   }
}

implicit val defaultParams = new Params
```

Scala DSL

Be

Scala DSL

Be Lazy

```
"This should not explode" in {
  error("boom")
}
```

Scala DSL

Be

lazy

```
class ExampleDesc(d: String) {
  def in(e: =>Any) =
    new Example(description, e)
}

class Example(d: String, e: =>Any)
```

Scala DSL

Be

Lazy

```
def in(e: =>Any) = …
```

Scala DSL

By name parameters

- Evaluate once!

```
class EqualMatcher[T](x: =>T) extends Matcher[T] {
  def apply(y: =>T) = {

    val (a, b) = (x, y)

     (a == b, a + " is equal to " + b,
              a + " is not equal to " + b)
  }
}
```

Scala DSL

By name parameters

- With varargs?

```
// not legal!!
def method[T](params: =>T*)

method(1., 2., math.pow(100, 100))
```

Scala DSL

By name parameters

- With varargs!

```
implicit def lazyfy[T](value: =>T) =
  new LazyParameter(() => value)

class LazyParameter[T](value: () => T) {
  lazy val v = value()
  def get() = v
}
```

Scala DSL

By name parameters

- With varargs!

```
def method[T](params: LazyParameter[T]*)
```

About…

specs tour

Implicit def

Restrict

Combine

Add , add

Be lazy

!&%$#>

Manifests

Resources

Scala DSL

Operators

```
class ExampleDesc(d: String) {
  def >>(e: =>Any) = new Example(d, e)
}
```

```
"When I setup the system" >> {
  "And a customer is entered" >> {
    "if he has a discount" >> {}
    "if he doesn't have a discount" >> {}
  }
}
```

Scala DSL

Operators

```
// contexts
"A full stack" ->-(fullStack) should { … }

// matchers
xml must \\(<node/>)

// repl matcher
> "This is the expected result"
```

Scala DSL DataTables

```
/**
 * Fit-like table in Scala?
 *
 *   | a | b | c |
 *   | 1 | 2 | 3 |
 *   | 2 | 2 | 4 |
 *
 */
```

Use ‘!‘ to separate cells and
‘|‘ to separate rows

Scala DSL

DataTable

```
class DataTablesSpec extends Specification
with DataTables {
   "lots of examples" >> {
      "a" | "b" | "a + b" |
       1  !  2  !    3    |
       2  !  2  !    4    |
       2  !  3  !    4    |> { (a, b, c) =>
          a + b must_== c
      }
   }
}
```

Scala DSL DataTables

```
// the 'play' operator |>
2   !   3   !   4     |> { (a, b, c) =>
```

Scala DSL

Class manifests

```
// How to avoid
// throwA(classOf[FailureException])
(1 must_== 2) must
              throwA[FailureException]
```

Scala DSL

Class manifests

```
import scala.reflect._

def throwA[T](implicit m: ClassManifest[T]) =
  new Matcher[Any] {
  // use m.erasure
  }
```

Scala DSL

Make a guess

```
"When I setup the system" >> {
  "And a customer is entered" >> {
    "if he has a discount" >> {}
    "if he doesn't have a discount" >> {}
  }
}
```

⇨ Discover leaves without executing?

Scala DSL

Make a guess

```
def in[T : Manifest](e: =>T) = {
  expectationsAre(e)

  if (manifest[T].erasure == getClass)
    hasNestedExamples = true
  this
}
```

7/21/10

About…

specs tour

Implicit def

Restrict

Combine

Add , add

Be lazy

!&%$#>

Manifests

Resources

Scala DSL

http://code.google.com/p/specs
http://etorreborre.blogspot.com/oscon-201

Scala DSL

Bag of tricks

Scala DSL

I want a pony

```
/**
 *   how to write:
 *   I want a pony ??
 */
```

```
/**
 *   2 solutions:
 *
 *   I.want(a.pony) => ScalaTest
 *   (I.want(a)).pony => specs
 */
```

Scala DSL

Some examples

```
list must have size(3)
list must be empty

list must have size(2) or have size(3)

(((list must have).size(2)).or(have)).size(3)
```

- Fancy but difficult to reason about

Scala DSL

I want a pony

```
trait Word; class AWord extends Word

class Result[T](v: T)
class Expectable[T](v: T) {
  def want(a: Word) = new Result[T](v)
}
implicit def theValue[T](v: T): Expectable[T] = …
implicit def ponyable[T](r: Result[T]) {
  def pony = // check the result
}
val a = new Aword; val I = 1
I want a pony
```

Scala DSL

apply

```
object DBContext extends Specification {
  val setup = new SpecContext {
    beforeExample(deleteUsersTable)
    aroundExpectations(inDatabaseSession(_))
  }
}

object RepositorySpecification extends Specification {
  // equivalent to: DBContext.setup.apply(this)
  DBContext.setup(this)
  "A Users repository" can { /*...*/ }
}
```

object.apply(object) ▫
verb(object)

Scala DSL

Varianc

Know the rules!

```
trait Matcher[-T] { def apply(y: =>T): Boolean }
class EqualMatcher(x: Any) extends Matcher[Any] {
  def apply(y: =>Any) = x == y
}
class HelloMatcher extends Matcher[String] {
  def apply(x: =>String) = x == "hello"
}
val beHello = new HelloMatcher
val equalToHello = new EqualMatcher("hello")

"hello" must beHello
"hello" must equalToHello
```

Scala DSL

typing

```
implicit def forExample(d: String): Example
```

"An implicit conversion without explicit result type is visible only in the text following its own definition"