Today’s a real treat! Brought to you by...

Follow along: @HMC_DSLs

Warning: What we learn today may be useful for something other than DSLs.
Is this a DSL?

REST APIs

POST statuses/update

Updates the authenticating user's current status, also known as Tweeting.

For each update attempt, the update text is compared with the authenticating user's recent Tweets. Any attempt that would result in duplication will be blocked, resulting in a 403 error. Therefore, a user cannot submit the same status twice in a row.

While not rate limited by the API a user is limited in the number of Tweets they can create at a time. If the number of updates posted by the user reaches the current allowed limit this method will return an HTTP 403 error.

About Geo

- Any geo-tagging parameters in the update will be ignored if geo_enabled for the user is false (this is the default setting for all users unless the user has enabled geolocation in their settings)
ScalaTest = Traits for DSLs

Dialect: test-driven development

import org.scalatest.FunSuite
import scala.collection.mutable.Stack

class ExampleFunSuite extends FunSuite {

  test("pop is invoked on a non-empty stack") {

    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    val oldSize = stack.size
    val result = stack.pop()
    assert(result === 2)
    assert(stack.size === oldSize - 1)
  }
}

examples taken from scalatest.org
import org.scalatest.FunSpec
import scala.collection.mutable.Stack

class ExampleFunSpec extends FunSpec {

describe("A Stack") {

  it("should pop values in last-in-first-out order") {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
    assert(stack.pop() === 2)
    assert(stack.pop() === 1)
  }

  ...

}

examples taken from scalatest.org
ScalaTest = Traits for DSLs

Dialect: functional testing

```scala
import org.scalatest.FeatureSpec
import org.scalatest.GivenWhenThen
import scala.collection.mutable.Stack

class ExampleFeatureSpec extends FeatureSpec with GivenWhenThen {

  feature("The user can pop an element off the top of the stack") {

    info("As a programmer")
    info("I want to be able to pop items off the stack")
    info("So that I can get them in last-in-first-out order")

    scenario("pop is invoked on a non-empty stack") {

      given("a non-empty stack")
      val stack = new Stack[Int]
      stack.push(1)
      stack.push(2)
      val oldSize = stack.size

      when("when pop is invoked on the stack")
      val result = stack.pop()

      then("the most recently pushed element should be returned")
      assert(result === 2)

      and("the stack should have one less item than before")
      assert(stack.size === oldSize - 1)

    }
  }
}
```

examples taken from scalatest.org
Traits for Extensibility

(see GitHub repo)
Is this a DSL?

- Everyone speaks with an English accent.
- Verity multiplies every number by 10.
- Lambert divides every number by 3.
- Never say “mattress” to Lambert; say “dog kennel” instead.

So, if Verity wants to say, “This mattress costs £90” to Lambert, it will come out as “This dog kennel costs £900” (in an English accent).
val sentence = "This mattress costs 90 pounds"

class Person {
  def say(s: String): String = s
  def @:(s: String) = s
}

trait MSEmployee extends Person {
  override def say(s: String) = super.say("(In an English accent) " + s)
}

trait TimesTen extends Person {
  override def say(s: String): String =
    super.say("\d+".replaceAllIn(s, m=>(m.toString.toInt * 10).toString))
}

trait DivThree extends Person {
  override def say(s: String): String =
    super.say("\d+".replaceAllIn(s, m=>(m.toString.toInt / 3).toString))
}

trait DogKennels extends Person {
  override def @:(s: String) = super.@:(s.replace("mattress", "dog kennel"))
}

object MrVerity extends Person with MSEmployee with TimesTen
object MrLambert extends Person with MSEmployee with DivThree with DogKennels

MrVerity say sentence @: MrLambert