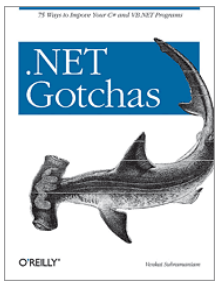
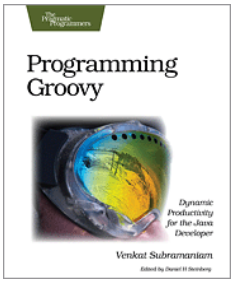
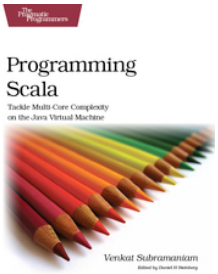
Building DSLs in Groovy

Venkat Subramaniam
venkats@AgileDeveloper.com







Communication

- How do we humans communicate?

First Moon Landing

Transcript...

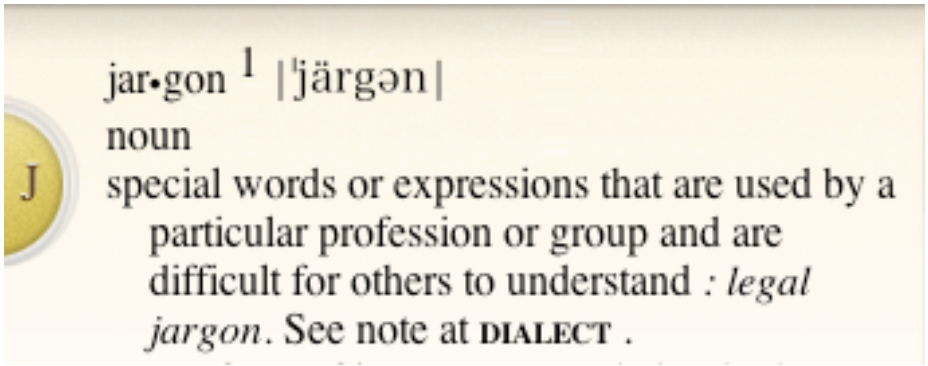
```
...
102:45:04 Aldrin: (Velocity) light's on.
102:45:08 Aldrin: 60 feet, down 2 1/2. (Pause) 2 forward. 2 forward. That's good.
102:45:17 Aldrin: 40 feet, down 2 1/2. Picking up some dust.
102:45:21 Aldrin: 30 feet, 2 1/2 down. (Garbled) shadow.
102:45:25 Aldrin: 4 forward. 4 forward. Drifting to the right a little. 20 feet, down a half.
102:45:31 Duke: 30 seconds (until the 'Bingo' call).
102:45:32 Aldrin: Drifting forward just a little bit; that's good. (Garbled) (Pause)
102:45:40 Aldrin: Contact Light.
102:45:43 Armstrong (on-board): Shutdown
102:45:44 Aldrin: Okay. Engine Stop.
102:45:45 Aldrin: ACA out of Detent.
102:45:46 Armstrong: Out of Detent. Auto.
102:45:47 Aldrin: Mode Control, both Auto. Descent Engine Command Override, Off. Engine Arm, Off. 413 is in.
102:45:57 Duke: We copy you down, Eagle.
102:45:58 Armstrong (on-board): Engine arm is off. (Pause) Houston, Tranquility Base here. The Eagle has landed.
...

First Landing on the Moon: Source http://history.nasa.gov/alsj/a11/a11.landing.html
```

Two things go through my mind:

1. Wow
2. I've the faintest idea what they're talking about!

Communication

- As human we communicate using Jargons



Aren't Jargons Bad?

- They're very bad for people who don't understand the domain or don't share the context
  - You'd be totally lost
- But, they're very good for people who share the context in the domain
  - The communication is highly effective
  - Terse, high signal—to—noise ratio
  - This is what professionals use in their respective fields

How do you use your App/ Computer?

- Typically you use keyword input files, configuration files, etc.
- If you're stuck at an airport, which of these two agents gives you the most confidence you'll be on your way—one that uses the keyboard, or the one that mouses around?

  - The former, uses direct queries

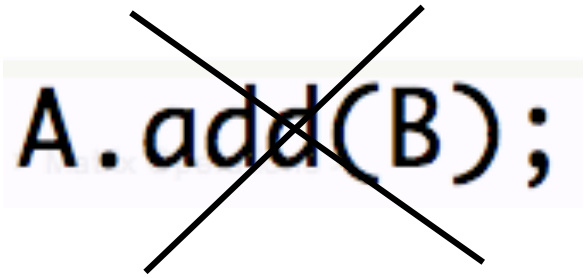
Productivity is Key

- A GUI makes an average user productive, but often may slowdown an expert user or a user very fluent with your application and its domain.
- You don't want to make the novice productive at the expense of making an expert counterproductive.

Naked Objects

- Richard Pawson and Robert Matthews introduced the concept of Naked Objects
- They auto-build a thin object-oriented user interface (OOUI) from a domain model
- The OOUI exposes the behavior of domain objects for direct interaction by the users.

How does an expert express Matrix Multiplication?




```
A + B
```


Goal of a DSL

- The goal of DSLs is to provide a highly effective interface for your users to interact with your application.
- The interface may be graphical or textual.

But?!

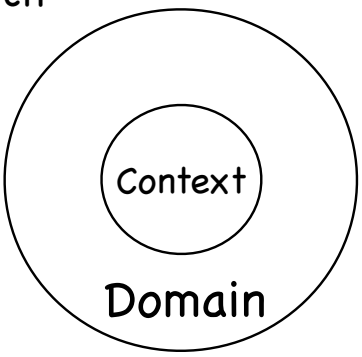
what's a DSL?

Domain Specific Language

- A language targeted at a particular type of problem
- Syntax focuses on intended domain/problem
- Unlike a general purpose language, you can't use it for all kinds of stuff (or don't want to)
- Limited in scope and capability
- Hence, they're small and simple—that's good

Domain and Context

- What's Domain?
  - It is an area of sphere or knowledge, influence, or activity
- What's Context?
  - A logical framework within which we evolve models
- What's Bounded Context?
  - Defines a logical boundary of relevance within the context



What is it really?

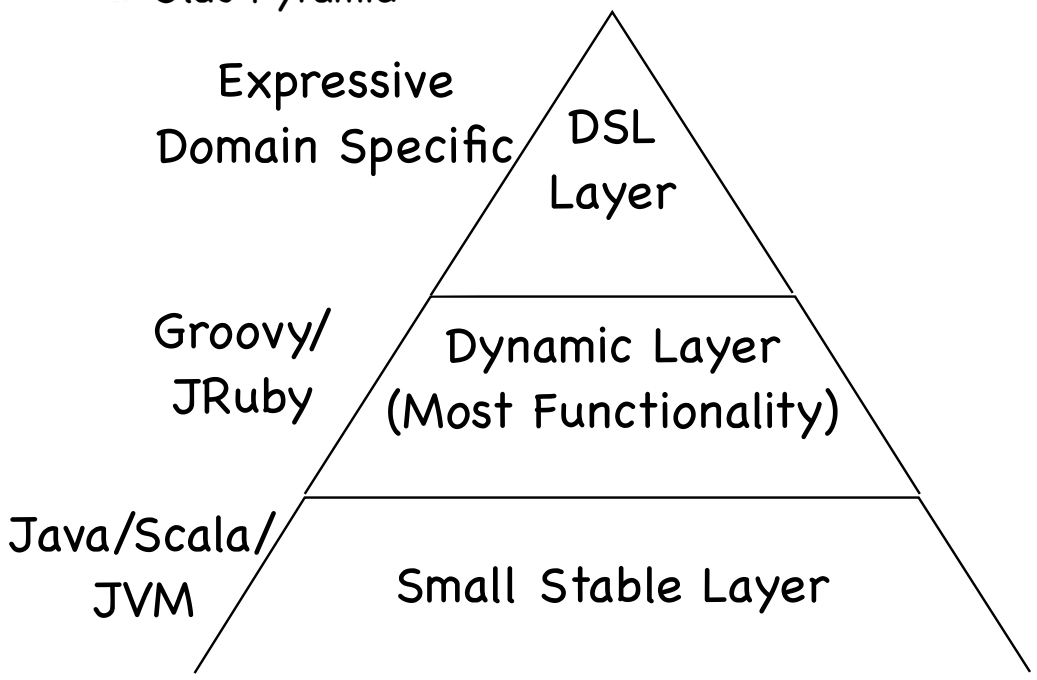
- It's a ubiquitous language—only narrow within a domain
- Domain folks understand it well
- Highly expressive and easy to use

Where does it fit?

- You're most likely to use multiple languages to build your application—Neal Ford's vision for Polyglot Programmer.
- Martin Fowler argues that rather than using one big general purpose language, applications will use several small and limited DSLs—Languages Oriented Programming.
- Ola Bini argues at http://ola-bini.blogspot.com/2008/01/language-explorations.html that "three language layers will emerge in large applications"

Where does it fit?

- Ola's Pyramid



What's Need to Build 'em?

- Objects + Meta Programming (Aspects) + lots of Patience + 

Types of DSLs

- External DSLs
- Internal DSLs

External DSL

- You define a new language
- You then parse the commands to take actions
- You have the liberty to chose the syntax you like
- You're in full control—that's good... and bad
- Can be lot of fun ;)
- One of my first jobs was to maintain lex and Yacc...

External DSL Examples

```
A:hover { color: #FF0000; text-decoration: none; }
```

- CSS is an example of an external DSL!

External DSL Examples

```
GCC=cxx -I.

all : clean compile

compile : myprog

myprog: MyProg.cc  Util.o
        $(GCC) -o myprog MyProg.cc Util.o

Util.o : Util.h Util.cc
        $(GCC) -c Util.cc

clean :
        /bin/rm -f myprog Util.o
```

- Good old Makefile is an example of external DSL

External DSL Examples

```
<project name="AnExampleProject" default="jarit" basedir=".">
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="distrib"  location="distrib"/>

  <target name="compile" description="compile your Java code from src into build" >
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="jarit" depends="compile" description="jar it up" >
    <jar jarfile="${distrib}/AnExampleProject.jar" basedir="${build}"/>
  </target>
</project>
```

- Ant is an example of external DSL

Internal DSL

- Also known as Embedded DSLs
- Built on top of a host language
- Constructs of your host language are construed to form a DSL
- You don't need to write an separate parser—that's less work
- Commands tactfully map over to constructs like methods, etc.
- You can enjoy the host language's flexibility, but you get bitten by its limitations and idiosyncrasies

Internal DSL Examples

```
ORIGINAL = 'input.dat'
BACKUP = 'input.dat.bak'

task :default => BACKUP

file BACKUP => ORIGINAL do |task|
 cp task.prerequisites[0], task.name
end
```

- Rake is an example of internal DSL!
  - It is pure Ruby

Internal DSL Examples

```
#slightly modified version of example from http://gant.codehaus.org/
includeTargets << gant.targets.Clean
cleanPattern << [ '**/*~' ,  '**/*.bak' ]
cleanDirectory << 'build'

target (stuff : 'A target to do some stuff') {
  println 'Stuff'
  depends clean
  echo message : 'A default message from Ant'
  otherStuff()
}

target (otherStuff : 'A target to do some other stuff') {
  println 'OtherStuff'
  echo message : 'Another message from Ant'
  clean()
}

setDefaultTarget stuff
```

- Gant is an example of internal DSL!
  - It is pure Groovy, wrapper around Ant

Internal DSL Examples

```
//transferMoney.story
scenario 'transfer money', {
  given 'account numbers 123456789 and 123456788'
  when 'tranfer $50 from 123456789 to 123456788'
  then 'balance of 123456789 is $50 less'
  and
  then 'balance of 123456788 is $50 more'
  and
  then 'transaction has been logged...'
}
```

- easyb is an example of internal DSL
- It is a Behavior Driven Design Tool and uses Groovy for expressing stories and behavior in a DSL form

External or Internal

- It depends on your application
- Certain (dynamic) languages are better suited for internal DSLs
  - Smalltalk, Lisp, Ruby, Groovy, …
- Other languages are better suited for external DSL and offer good parsing libraries or capabilities
  - C++, Java, C#, …

External or Internal

- External
- +You decide syntax
- -You write parser for it
- + A tad easier to design
- + Once you write a parser, validation is a breeze

- Internal
- - You decide within limits of host language
- + Language takes care of parsing
- - You gotta find your way around host
- - Hard to validate at the moment, good research potential

Are all Languages Suited for Internal DSLs?

- Not really
- Low Ceremony + Metaprogramming is essential
- Scala, for example, has low ceremony, but lacks metaprogramming capabilities. So, not so easy to write internal DSLs with it
- Groovy, Ruby, ... shine due to high essence, low ceremony, and metaprogramming

Characteristics of a DSL

- What are some of the qualities of a DSL?
  - Context Driven
    - Extremely context sensitive
  - Fluent Interfaces
    - Provides for a humane, easy to understand interface

Context Driven

- DSL is extremely context sensitive or context driven
- A lot of things are knows from the context
- Without context, we may be confused or misled
- With context, there is less clutter, terse, but readable

Context Driven ...

- I heard my friend Neal Ford yell out

  Quad Venti Latte with 2 Extra Shots

- What's he talking about?

- He's using the Starbucks DSL

- Nowhere did he mention the word coffee, but he sure got a good one at a high price ;)
- That's context driven

Context Driven ...

- I heard my friend Scott Davis mutter

  place a $ in a GString

- What's he talking about?
- You don't want to know
- He sure was talking about embedding variables into the Groovy's mutable string class
- That's also context driven :)

```
today = new Date()
str = "Today is ${today}"
println str
println str.class.superclass
```

```
Today is Wed Sep 05 04:35:28 MDT 2007
class groovy.lang.GString
```


Fluent Interface

- I was going to call this hygienic interface
- I like the name Eric Evans and Martin Fowler promote–Fluent Interface/Humane Interface
- It's designed to be readable and to flow naturally
- Not easy to design–you put in more effort to create it, so it is easier for users to use

Code With No Context

```
Mailer mailer = new Mailer();
mailer.from("johndoe@example.com");
mailer.to(new String[]{"janedoe@example.com", "bobdoe@example.com"});
mailer.subject("refactor to fluency and context");
mailer.body("Hello, ...");
mailer.send();
```

- What do you do with mailer object after call to send?
- Is the order important?
- Do I have to create an object each time?
- ...
- Can I just get my work done, can this be simpler?

Code With Context

```
Mailer.send {
  from    'johndoe@example.com'
  to      'janedoe@example.com', 'bobdoe@example.com'
  subject 'refactor to fluency and context'
  body    'Hello, ...'
}
```

Let's Look at them both

```
Mailer mailer = new Mailer();
mailer.from("johndoe@example.com");
mailer.to(new String[]{"janedoe@example.com", "bobdoe@example.com"});
mailer.subject("refactor to fluency and context");
mailer.body("Hello, ...");
mailer.send();
```

```
Mailer.send {
  from    'johndoe@example.com'
  to      'janedoe@example.com', 'bobdoe@example.com'
  subject 'refactor to fluency and context'
  body    'Hello, ...'
}
```

We'll see later how to build these...

How do you write a Loop?

- Find total of numbers from 1 to 10

```
int total = 0;
for (int i = 1; i <= 10; i++)
{
  total += i;
}
```

- Why does the Java loop look like this?

Fluent Loops (Groovy)

```
total = 0
for (i in 1..10)
{
  total += i
}

// OR

total = 0
1.upto(10) { total += it }

// OR

total = 0
11.times { total += it }
```

How to get Fluency?

- You can achieve fluency in Java as well
- You can use **method chaining** with a little compromise

A Little Compromise

- Command Query Separation was promoted by Bertrand Meyer
  - Command methods—mutators or modifiers—affect object state and don't return anything
  - Query methods—don't affect object state and return useful information
- You compromise a little to be fluent (kind of like words bent to rhyme in a poem)
- Command methods return self or a promoter for a good flow

Fluent Interface in EasyMock

```
expect(alarm.raise()).andReturn(true);
expect(alarm.raise()).andThrow(new InvalidStateException());
```

Fluent Interface in JSONObject Library

```
JSONObject json = new JSONObject()
  .accumulate("key1", "value1")
  .accumulate("key2", "value2")
  .accumulate("key3", "value3");
```


Fluent Interface in Guice

```
bind(Alerter.class).
  to(DialogAlerter.class).
  in(Scopes.SINGLETON);
```

Fluency in Rails/Active Records

```ruby
class State < ActiveRecord::Base
  validates_uniqueness_of :code
  validates_numericality_of :year_into_statehood
  validates_inclusion_of :year_into_statehood, :in => 1776..1959

  validates_presence_of :name
  validates_presence_of :code
  validates_presence_of :year_into_statehood
end
```


Fluency in Grails/GORM

```groovy
class State
{
  String twoLetterCode

  static constraints = {
    twoLetterCode size: 2..2, unique: true, blank: false
  }
}
```

Let's Order Pizza

- Voice on line (VOL): Thanks for calling Joe's
- You: Ya, A Large Thin Crust
- VOL: Toppings?
- You: Olives, Onions, Bell Pepper
- VOL: Address
- You: 101 Main St.,...
- VOL: Card?
- You: Visa 1234-1234-1234-1234
- VOL: 20 minutes-Thank you

Java Pizza Order

```
PizzaShop joes_pizza = new PizzaShop();
joes_pizza.setSize(Size.LARGE);
joes_pizza.setCrust(Crust.THIN);
joes_pizza.setTopping("Olives");
joes_pizza.setTopping("Onions");
joes_pizza.setTopping("Bell Pepper");
joes_pizza.setAddress("101 Main St., ...");
int time = joes_pizza.setCard(CardType.VISA, "1234-1234-1234-1234");
System.out.printf("Pizza will arrive in %d minutes\n", time);
```

- Not fluent
- Very noisy—how many times do you have to say joes_pizza?
- Can we make it fluent?

with

- Some languages have a with keyword
- within the scope, methods are invoked on an implicit object
- Here is an example from JavaScript

```
// JavaScript accessing Java API in Java 6

lst = new java.util.ArrayList();

with(lst)
{
  add(1);
  add(2);
  println(lst.size());
}
```


with in Groovy?

- Groovy has with keyword which is synonym for identity method

```
lst = []

lst.identity {
  add(1)
  add(2)
  println size
}
```

How about Fluent Java?

```
int time = new PizzaShop()
           .setSize(Size.LARGE)
           .setCrust(Crust.THIN)
           .setTopping("Olives")
           .setTopping("Onions")
           .setTopping("Bell Pepper")
           .setAddress("101 Main St., ...")
           .setCard(CardType.VISA, "1234-1234-1234-1234");
System.out.printf("Pizza will arrive in %d minutes\n", time);
```

- Less clutter
- There is a context in each call
- Each set method (except setCard()) returns this (PizzaShop)
- Can we make this a bit more readable?

A bit more Fluent...

```
int time = PizzaShop.orderPizza()
           .ofSize(Size.LARGE)
           .withCrust(Crust.THIN)
           .withTopping("Olives")
           .withTopping("Onions")
           .withTopping("Bell Pepper")
           .deliverToAddress("101 Main St., ...")
           .chargingCard(CardType.VISA, "1234-1234-1234-1234");
System.out.printf("Pizza will arrive in %d minutes\n", time);
```

- Method names make sense in the flow, but may not make sense alone
- What's a catch?

Ordered vs. Unordered

- Some DSLs may require a certain order in the flow
  - You can force ordering by tactfully defining return types that will allow method chaining in only certain order
    - Not easy
- Others may not require ordering
- It depends on the particular problem
- Ideally, not requiring an order is great, but may not always be possible

How to Enforce Sequence?

```
int time = PizzaShop.orderPizza(
          new PizzaShop()
          .ofSize(Size.LARGE)
          .withCrust(Crust.THIN)
          .withTopping("Olives")
          .withTopping("Onions")
          .withTopping("Bell Pepper")
          .deliverToAddress("101 Main St., ...")
          .chargingCard(CardType.VISA, "1234-1234-1234-1234")
          );
System.out.printf("Pizza will arrive in %d minutes\n", time);
```

Closures Make It Easier

```
time = PizzaShop.orderPizza { pizzaShop ->
   pizzaShop.ofSize(Size.LARGE)
   .withCrust(Crust.THIN)
   .withTopping("Olives")
   .withTopping("Onions")
   .withTopping("Bell Pepper")
   .deliverToAddress("101 Main St., ...")
   .chargingCard(CardType.VISA, "1234-1234-1234-1234")
}
```

```
class PizzaShop
{
  private PizzaShop() {}

  public static int orderPizza(closure)
  {
    def shop = new PizzaShop()

    // Allow user to work with PizzaShop
    closure(shop)

    // process order after user is done
```


OK, but...

- You may not be able to deviate from standard API conventions
- What if it is existing API or there are other reasons to follow conventions?
- You can use a Façade or expression builder
  - It sits on top of existing traditional API and provides context and fluency
  - For example, Groovy has a number of these builders built-in

Groovy Builder: XML

Fluency to build XML

```
import groovy.xml.MarkupBuilder

map = ['C++' : 'Stroustrup', 'Java' : 'Gosling', 'C#' : 'Hejlsberg']

markupBldr = new MarkupBuilder()

xml = markupBldr.languages {
  for(entry in map)
  {
    language(name : entry.key) {
      author (entry.value)
    }
  }
}
```


Groovy Builder: Swing

Fluency to build Swing GUI

```
setLabel = { lbl -> lbl.setText(new Date().toString()) }

swingBldr = new groovy.swing.SwingBuilder()


frame = swingBldr.frame(
  title : "Example",
  size:[200, 100],
  layout: new java.awt.FlowLayout(),
  defaultCloseOperation:javax.swing.WindowConstants.EXIT_ON_CLOSE) {

  myLabel = label(text: "Hello")
  button(text : 'Click', actionPerformed: {  setLabel(myLabel) })
}

frame.show()
```

Creating DSLs

- You may use tools like Antlr for working with DSLs
- JetBrains is working on Meta Programming System (MPS)
- Languages like Ruby and Groovy may it easier to create (internal) DSLs
- What makes Groovy attractive?
  - Classes are always open
  - Closures and Builders
  - Categories and ExpandoMetaClass
  - To some extent operator overloading
  - Parenthesis are almost optional (a bit lacking)

Creating a DSL in Groovy

file:order

```
orderPizza {
size 'large'
toppings 'Olives'
toppings 'Onions'
toppings 'Bell Pepper'
deliverTo '101 Main St.,...'
chargeCard '1234-1234-1234-1234'
}
```

See next page

file:process.groovy

```
def defs = new File('defs.groovy').text
def dsl = new File('order').text

def script = defs + dsl
def shell = new GroovyShell()
shell.evaluate(script)
```

output

```
Here is your order:
deliverTo : 101 Main St.,...
chargeCard : 1234-1234-1234-1234
size : large
toppings : ["Olives", "Onions", "Bell Pepper"]
Pizza will arrive in 25 minutes
```

Creating a DSL in Groovy

contents of file:defs.groovy

```groovy
ordering = false
orderInfo = [toppings : []]

def methodMissing(String name, args)
{
  if (ordering)
  {
    if (name == 'toppings')
    {
      orderInfo[name] << args[0]
    }
    else
    {
      orderInfo[name] = args[0]
    }
  }
  else
  {
    // silenty ignore or throw exception...
  }
}
```

```groovy
def orderPizza(closure)
{
  ordering = true
  closure()
  ordering = false

  println "Here is your order:"
  orderInfo.each { key, value ->
    println "${key} : ${value}"
    }

  println "Pizza will arrive in 25 minutes"
}
```


transmogrification

- How to deal with fluency, context, and ordering
- You can tactfully return different interfaces so only correct sequence of calls will fit in
- Return an appropriate interface from your methods and let method chaining take over

Using Categories

```
use(DatesUtil.class)
{
  println 2.days.ago.at(4.30)

  println 20.days.ago.at(16.30)
}
```

Type conversions

---

2 => int

.days => int

.ago => Calendar

.at => Date

```
class DatesUtil
{
  public static int getDays(Integer self)
  {
    self
  }

  public static Calendar getAgo(Integer self)
  {
    def today = Calendar.instance
    today.add(Calendar.DAY_OF_MONTH, -self)

    today
  }

  public static Date at(Calendar self, Double time)
  {
    def timeDbl = time.doubleValue()
    def hours = (int)timeDbl
    def minutes = (int)((timeDbl - hours) * 100)

    self.set(Calendar.HOUR_OF_DAY, hours)
    self.set(Calendar.MINUTE, minutes)
    self.time
  }
}
```


Using ExpandoMetaClass

```
Integer.metaClass.getDays = {->
    delegate
  }

Integer.metaClass.getAgo = {->
  def today = Calendar.instance
  today.add(Calendar.DAY_OF_MONTH, -delegate)

  today
  }

GregorianCalendar.metaClass.at = {Double time ->
  def timeDbl = time.doubleValue()
  def hours = (int)timeDbl
  def minutes = (int)((timeDbl - hours) * 100)

  delegate.set(Calendar.HOUR_OF_DAY, hours)
  delegate.set(Calendar.MINUTE, minutes)
  delegate.time
}

println 2.days.ago.at(4.30)

println 20.days.ago.at(16.30)
```

Using EMC

```
// Allows you to enhance hierarchy instead of a specific class
ExpandoMetaClass.enableGlobally()


//GregorianCalendar.metaClass.at = {Double time ->

Calendar.metaClass.at = {Double time ->
  ...
```


Categories vs. ExpandoMetaClass

- Categories allows you to tactically enhance a class
- ExpandoMetaClass is far reaching, global in nature
- You may not want to affect a class globally
- Categories provide controlled flexibility

Run Your DSL within a Context

- Running your DSL out in the open is not good: Unpredictable, unsettling
- Run it within the context of an object

Running in a context

```
players Jim, Jake, Jill
Jim 9
Jake 14
Jill 12
winner
```

scores

See next page

```
GameScore.process(new File('scores').text)
```

```
Winner is Jake with score 14
```

Output

Running in a Context

```
class GameScore
{
  def scores = [:]

  def players(String[] names)
  {
    names.each { scores."$it" = 0 }
  }

  def getWinner()
  {
    def highScore = 0
    def winner = ""

    scores.each { name, score ->
      if (score > highScore)
      {
        highScore = score
        winner = name
      }
    }

    println "Winner is $winner with score $highScore"
  }
```


Running in a Context

```
  def methodMissing(String name, args)
  {
    if (scores.containsKey(name) && args.length == 1)
    {
      scores."$name" = args[0]
    }
    else
    {
      throw new IllegalArgumentException("Invalid command")
    }
  }

  def propertyMissing(String name) { name }

  def static process(dslString)
  {
    def script = """
      new GameScore().with {
        $dslString
      }
    """

    new GroovyShell().evaluate(script)
  }
}
```

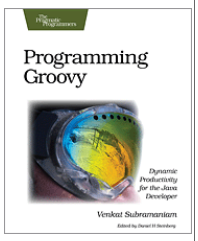
References

- http://martinfowler.com/bliki/DomainSpecificLanguage.html
- http://docs.codehaus.org/display/GROOVY/Writing+Domain-Specific+Languages
- http://docs.codehaus.org/display/GROOVY/Gant
- http://groovy.codehaus.org/ExpandoMetaClass
- http://www.onboard.jetbrains.com/is1/articles/04/10/lop/
- http://www.jetbrains.com/mps
- Martin Fowler's book work in progress: http://martinfowler.com/dslwip
- A Chapter on creating DSL in Groovy: "Programming Groovy: Dynamic Productivity for Java Developers" by Venkat Subramaniam (Pragmatic Bookshelf).

You can download examples and slides from
http://www.agiledeveloper.com - download

References

- A number of references from the following URL
- http://martinfowler.com/bliki/DomainSpecificLanguage.html
- http://docs.codehaus.org/display/GROOVY/Writing+Domain-Specific+Languages



- http://docs.codehaus.org/display/GROOVY/Gant
- http://groovy.codehaus.org/ExpandoMetaClass
- http://www.onboard.jetbrains.com/is1/articles/04/10/lop/
- http://www.jetbrains.com/mps

You can download examples and slides from
http://www.agiledeveloper.com - download

Thank You!

Please fill in your session evaluations

You can download examples and slides from
http://www.agiledeveloper.com - download