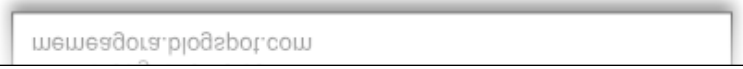
building dsl's in static & dynamic languages

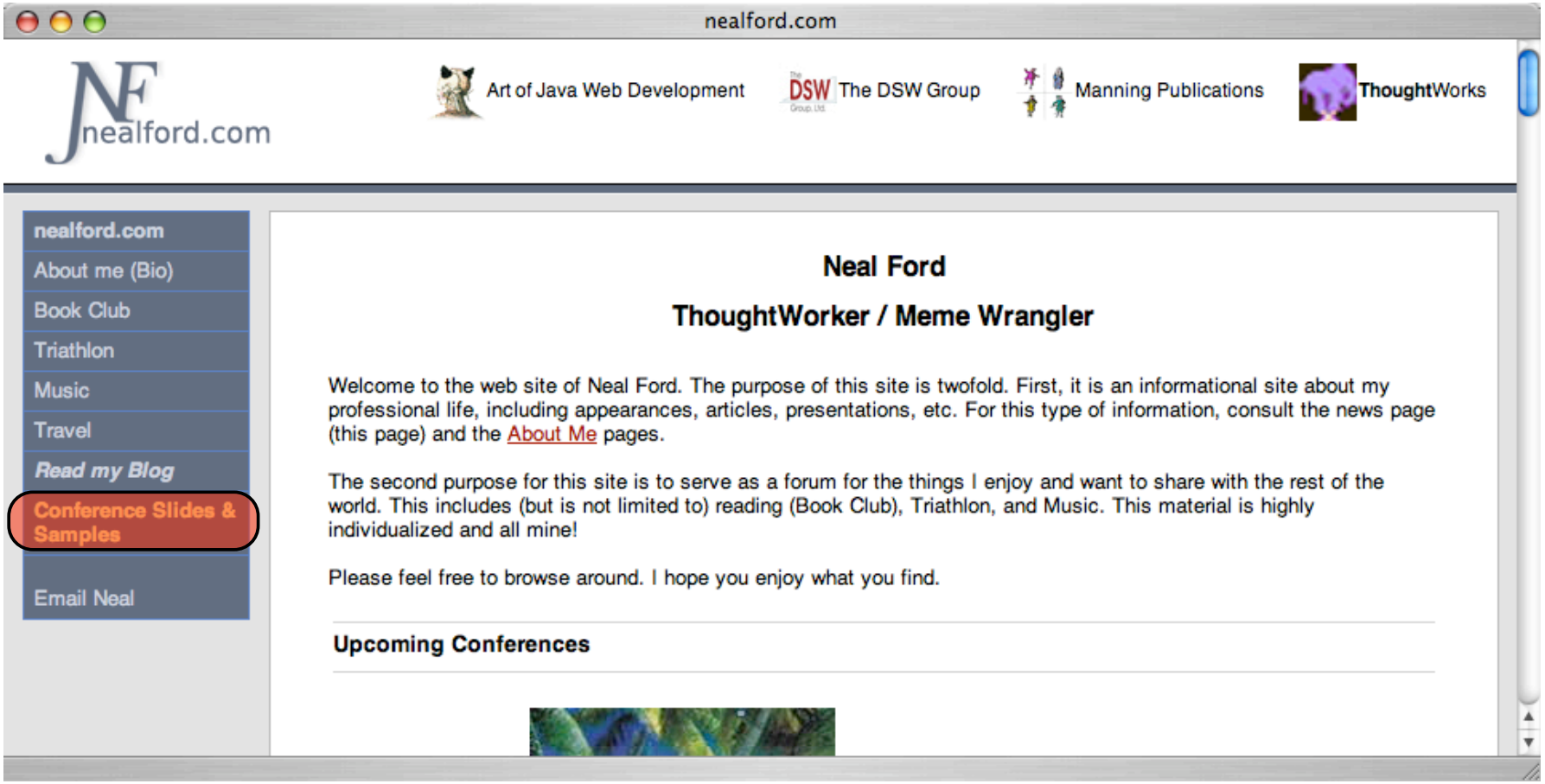
NEAL FORD thoughtworker / meme wrangler

ThoughtWorks

14 Wall St, Suite 2019, New York, NY 10005

nford@thoughtworks.com
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com




nealford.com
nealford.com
Art of Java Web Development
The DSW Group
Manning Publications
ThoughtWorks
nealford.com
About me (Bio)
Book Club
Triathlon
Music
Travel
Read my Blog
Conference Slides & Samples
Email Neal
Neal Ford
ThoughtWorker / Meme Wrangler
Welcome to the web site of Neal Ford. The purpose of this site is twofold. First, it is an informational site about my professional life, including appearances, articles, presentations, etc. For this type of information, consult the news page (this page) and the About Me pages.
The second purpose for this site is to serve as a forum for the things I enjoy and want to share with the rest of the world. This includes (but is not limited to) reading (Book Club), Triathlon, and Music. This material is highly individualized and all mine!
Please feel free to browse around. I hope you enjoy what you find.
Upcoming Conferences

what i cover

motivation

types of dsls

building dsls in:

- java
- groovy
- ruby

external dsls

best practices

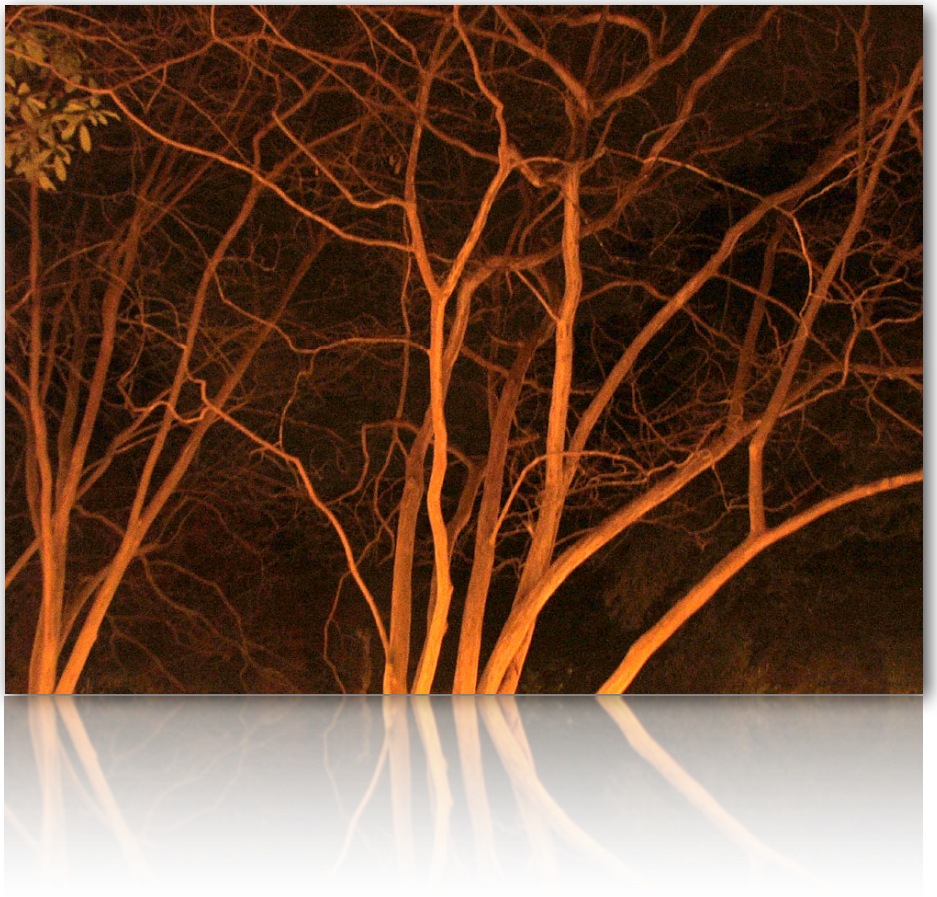
burning questions

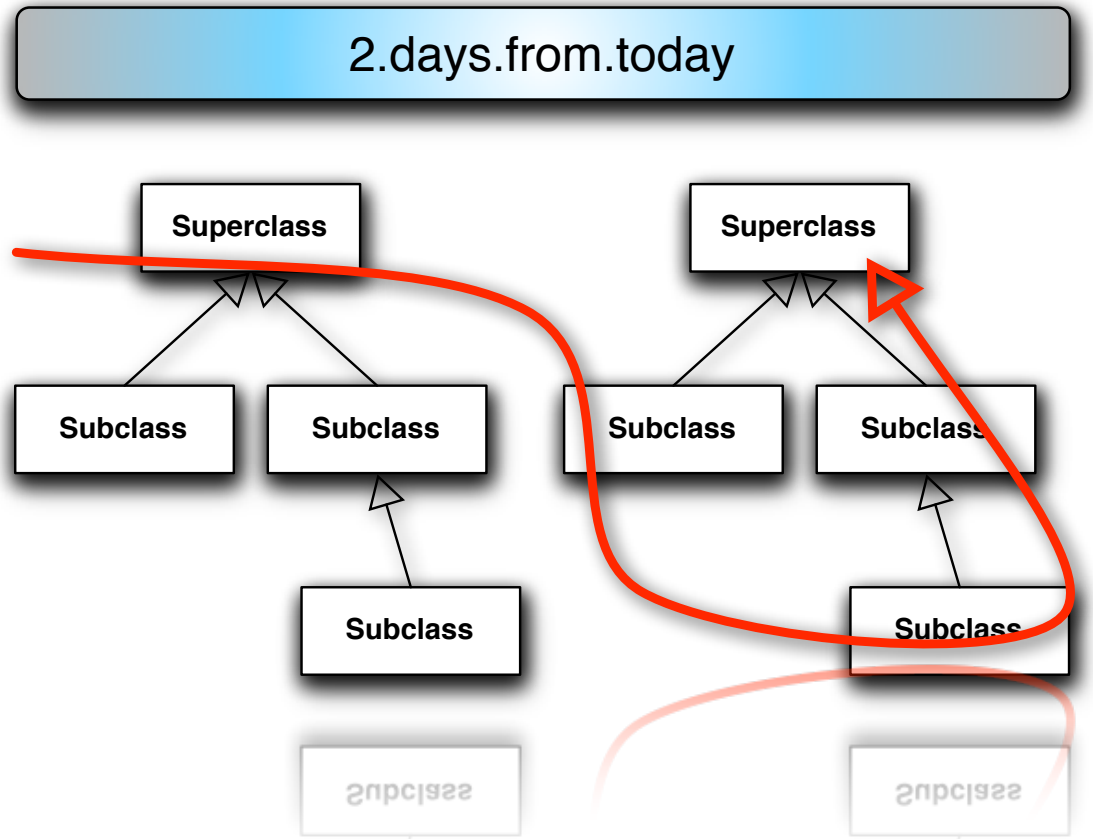
why is there so much xml mixed in with my java code?

why do things like aspects exist?

why won't everyone shut up already about ruby on rails?

is there an evolutionary step beyond object-oriented programming?





objects, aspects, generics become the building blocks for dsls

declarative vs imperative code

```
class Customer < ActiveRecord::Base
  validates_presence_of :name, :sales_description, :logo_image_url
  validates_numericality_of :account_balance
  validates_uniqueness_of :name
  validates_format_of :logo_image_url,
                      :with => %r{\.(gif|jpg|png)}i,
                      :message => "must be a URL for a GIF, JPG, or PNG image"
  has_many :orders
```

why dsls?

"Iced Decaf Triple Grande Vanilla Skim with whip latte."

"Scattered, smothered, covered"

"Route 66, swinging, easy on the chorus, extra solo at the coda, and bump at the end"

"OMFG D00d Bob is t3h UBER 1337 R0XX0RZ LOL"

waffle house hash brown language:
scattered, smothered, covered, chunked, topped, diced, peppered, & capped

Every non-trivial human behavior has a domain specific language.

including your job

all businesses have their own dsl

why developers tend to stay within domains

how non-developers talk about work

nomenclature

coined by martin fowler

domain specific language

> a limited form of computer language
> designed for a specific class of problems

language oriented programming

> general style of development which operates
> about the idea of building software around a
> set of domain specific languages

api: explicit context

```
Coffee latte = new Coffee(Size.VENTI);
latte.setFatContent(FatContent.NON_FAT);
latte.setWhip(Whip.NONE);
latte.setFoam(Foam.NONE);
latte.setTemperature(Temp.EXTRA_HOT);
latte.setStrength(5);
```

dsl: implicit context

```
Venti half-caf, non-fat, extra
hot, no foam, no whip latte
```

once context is established, repeating it over and over is just noise

types: internal

aka “embedded”

sit atop a base language

must follow syntax rules

why dynamic languages tend to make better bases

types: external

build your own language

must be able to lex and parse your language

let your imagination run wild!

internal dsl's

fluent interface

treat lines of code as sentences

example: jmock expectation

```
public void testOneSubscriberReceivesAMessage() {
    Mock mockSubscriber = mock(Subscriber.class);
    Publisher publisher = new Publisher();
    publisher.add((Subscriber) mockSubscriber.proxy());
    final String message = "message";
    // expectations
    mockSubscriber.expects(once()).
            method("receive").with(eq(message));

    publisher.publish(message);
}
```

car api

```
Car car = new CarImpl();
MarketingDescription desc = new
    MarketingDescriptionImpl();
desc.setType("Box");
desc.setSubType("Insulated");
desc.setAttribute("length", "50.5");
desc.setAttribute("ladder", "yes");
desc.setAttribute("lining type", "cork");
car.setDescription(desc);
```

car fluent interface

```
Car car = Car.describedAs()
            .box()
            .length(12)
            .includes(Equipment.LADDER)
            .has(Lining.CORK);
```

as simple as writing **`set`** methods that return **`this`**

method chaining

Make modifier methods return the host object so that multiple modifiers can be invoked in a single expression.

example: logging

writing a fluent interface around log4j
properties file creation

the target

```
# Set root logger level to DEBUG and its only appender to A1.
log4j.rootLogger=DEBUG, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-4r [%t] %-5p %c %x - %m%n
```

dsl syntax

```
Log l = new Log().
        withRoot(DEBUG).
        usingAppender(CONSOLE).
        formattedWith(PATTERN_LAYOUT).
        withPatternOf("%-4r [%t] %-5p %c %x - %m%n");
```

```
public class Log {
    public static String DEBUG = "DEBUG";
    public static String CONSOLE = "org.apache.log4j.ConsoleAppender";
    public static String PATTERN_LAYOUT = "org.apache.log4j.PatternLayout";

    public Log() {
        _logProperties = new Properties();
    }

    public Log withRoot(String root) {
        _logProperties.put("log4j.rootLogger", root);
        return this;
    }

    public Log usingAppender(String appender) {
        _internalAppender = "A1";
        String rootLogger = (String) _logProperties.get("log4j.rootLogger");
        _logProperties.put("log4j.rootLogger", rootLogger + ", " + _internalAppender);
        _logProperties.put("log4j.appender." + _internalAppender, appender);
        return this;
    }

    public Log formattedWith(String layout) {
        _logProperties.put("log4j.appender." + _internalAppender + ".layout", layout);
        return this;
    }
```

expression builder

a layer that provides a fluent interface over a regular api

wrapping api's

wrap existing api's in fluent interfaces to improve readability

example: wrapping ibatis

ibatis xml configuration

```
<sqlMap namespace="cache_event">
  <select id="matching-this" parameterClass="common.CacheEventImpl"
      resultClass="common.CacheEventImpl">
    select
      id              AS id,
      sequence_number AS sequenceNumber,
      event_type      AS eventType,
      source          AS source,
      event_time      as eventTime,
      receive_time    AS receiveTime,
      process_time    AS processTime,
      load_time       AS loadTime,
      status          AS status,
      sid             AS sid,
      id1             AS id1,
      id2             AS id2
    from cache_event
    where sequence_number = #sequenceNumber#
    </select>
```

sqlmap

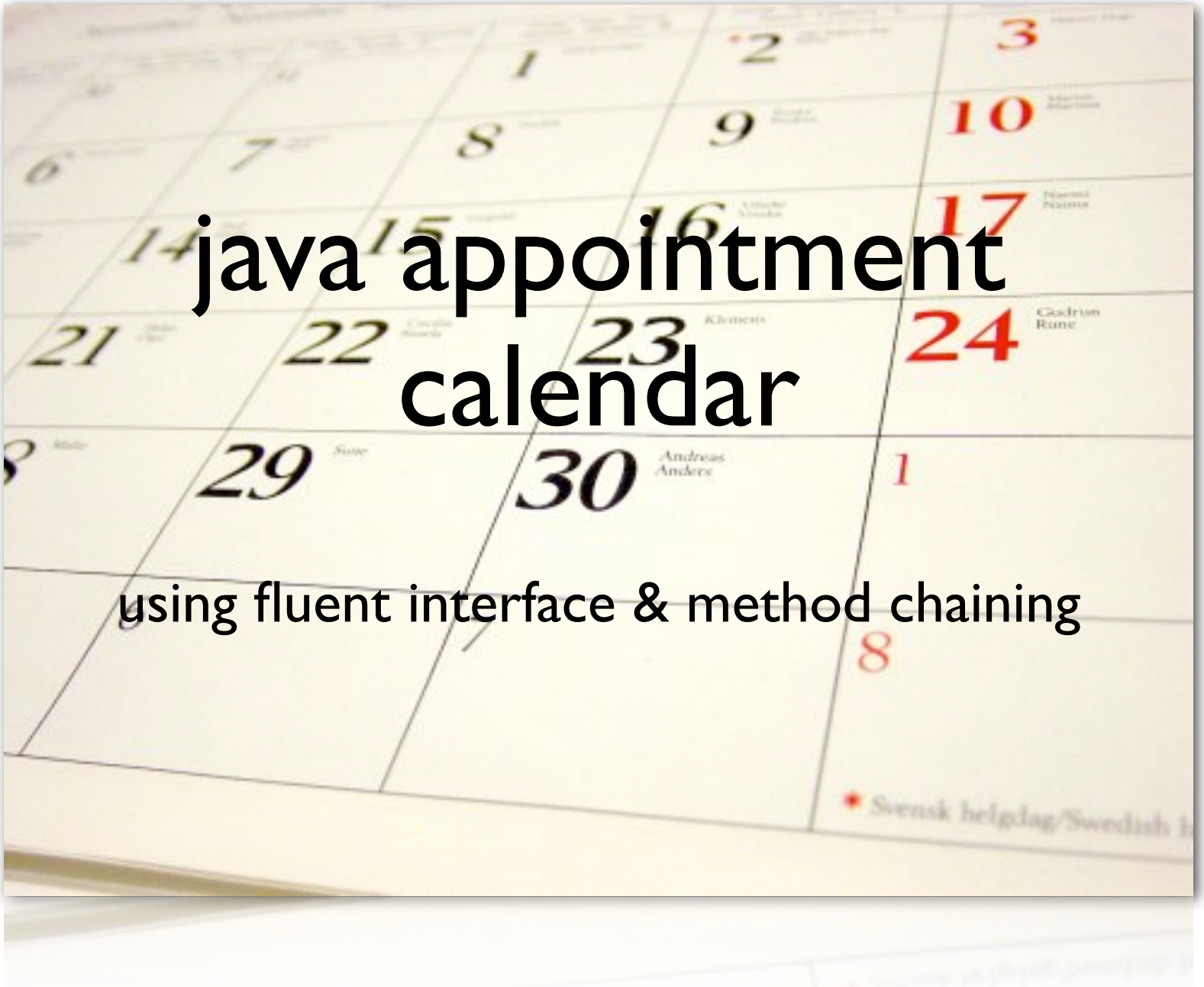
```
private SqlMapClient usePersistenceOperationNamed() {
    if (sqlMap == null) {
        if (resource == null || resource.length() == 0)
            resource = "common/data/SqlMapConfig.xml";
        try {
            Reader reader = Resources.getResourceAsReader(resource);
            sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage());
        }
    }
    return sqlMap;
}
```

```
public void addEvent(CacheEvent event)  {
    try {
        usePersistenceOperationNamed().insert("with-this-new", event);
    } catch (SQLException thatIndicatesPersistenceFailure) {
        throw new RuntimeException(thatIndicatesPersistenceFailure.getMessage());
    }
}
```

```
return (CacheEvent) usePersistenceOperationNamed().
                    queryForObject("matching-this",
                    new CacheEventImpl(sequenceNumber));
```

```
return (List<CacheEvent>) usePersistenceOperationNamed().
                    queryForList("matching-the-entire-list");
```

```
result = usePersistenceOperationNamed().
                    update("to-update-matching-this", event);
```


java appointment
calendar
using fluent interface & method chaining

```
public CalendarDemoChained() {
    Calendar fourPM = Calendar.getInstance();
    fourPM.set(Calendar.HOUR_OF_DAY, 16);
    Calendar fivePM = Calendar.getInstance();
    fivePM.set(Calendar.HOUR_OF_DAY, 17);

    AppointmentCalendarChained calendar =
            new AppointmentCalendarChained();

    calendar.add("dentist").
            from(fourPM).
            to(fivePM).
            at("123 main street");
    calendar.add("birthday party").at(fourPM);
    displayAppointments(calendar);
}
```

```
public class Appointment {
    private String _name;
    private String _location;
    private Calendar _startTime;
    private Calendar _endTime;

    public Appointment(String name) {
        this._name = name;
    }

    public Appointment at(String location) {
        _location = location;
        return this;
    }

    public Appointment from(Calendar startTime) {
        _startTime = startTime;
        return this;
    }

    public Appointment to(Calendar endTime) {
        _endTime = endTime;
        return this;
    }

    . . .
```

```
public class AppointmentCalendarChained {
    private List<Appointment> appointments;

    public AppointmentCalendarChained() {
        appointments = new ArrayList<Appointment>();
    }

    public List<Appointment> getAppointments() {
        return appointments;
    }

    public Appointment add(String name) {
        Appointment appt = new Appointment(name);
        appointments.add(appt);
        Database.persistAppointment(appt);
        return appt;
    }

}
```

the finishing problem

when does the call "finish"?

```
calendar.add("dentist").
         from(fourPM).
         to(fivePM).
         at("123 main street");
```

```
calendar.add("dentist").
         from(fourPM).
         to(fivePM).
         at("123 main street").save();
```

how can we make sure things happen at the right time?

method invocation

chained method calls

```
e().i().e().i().o()
```

nested method calls

```
e(i(e(i(o()))))
```

```
class OldMcDonald {
    def song = ""

    def e(def chain) {
        song += "e"
        this
    }

    def i(def chain) {
        song += "i"
        this
    }

    def o() {
        song += "o"
        this
    }

    def chained() {
        e().i().e().i().o()
    }

    def nested() {
        e(i(e(i(o()))))
    }
}
```

```
import groovy.util.GroovyTestCase

class TestFarmer extends GroovyTestCase {
    def farmer

    void setUp() {
        farmer = new OldMcDonald()
    }

    void test_chained() {
        farmer.chained()
        assertEquals "eieio", farmer.song
    }

    void test_Nested() {
        farmer.nested()
        assertEquals "oieie", farmer.song
    }
}
```

mitigating the finishing problem

```
calendar.add(
            new Appointment("Conference Call")
                .from(fourPM)
                .to(fivePM)
                .at("555-123-4321"));
```

build fluent interfaces with a mixture of chained and nested method invocations

use method chaining
for stateless object
construction

use nested methods to
control completion

good citizenship?

building objects with chained methods can create "bad citizens"

validation

- on dependent method calls
- in the nested **add()** method

imposes order semantics

allows invalid state


Groovy

dynamic building blocks

closures

open classes

dynamic typing

looser syntax rules than java

closures as containers

```
use (StringCategory) {
    expected.each { key, value ->
        assertEquals value, key.camelize()
    }
}
```

call a new method on String

open classes via categories

```
class StringCategory {

    static String camelize(String self) {
        def newName = self.split("_").collect() {
            it.substring(0, 1).toUpperCase() +
                it.substring(1, it.length())
        }.join()
        newName.substring(0, 1).toLowerCase() +
            newName.substring(1, newName.length())
    }
}
```

```
import groovy.util.GroovyTestCase

class TestStringCategory extends GroovyTestCase {
    def expected = ["event_map" : "eventMap",
            "name" : "name", "test_date" : "testDate",
            "test_string_with_lots_of_breaks" :
            "testStringWithLotsOfBreaks" ]

    void test_Camelize() {
        use (StringCategory) {
            expected.each { key, value ->
                assertEquals value, key.camelize()
            }
        }
    }
}
```

expando metaclass

```
metaClass = new ExpandoMetaClass(Integer.class,true)
metaClass.getLbs << {->
    "${delegate} pounds"
}
metaClass.initialize()
i = 1;
println i              // => 1
println i.class        // => java.lang.Integer
println i.lbs          // => 1 pounds
```

time spans

```
2.days.fromToday.at(4.pm)
```

returns a java.util.Calendar for the proper date

integer with time support

```
class IntegerWithTimeSupport {
    static Calendar getFromToday(Integer self) {
        def target = Calendar.instance
        target.roll(Calendar.DAY_OF_MONTH, self)
        return target
    }

    static Integer getAm(Integer self) {
        self == 12 ? 0 : self
    }

    static Integer getPm(Integer self) {
        self == 12 ? 12 : self + 12
    }

    static Integer getDays(Integer self) {
        self
    }
}
```

calendar support

```
package com.nealford.comf.time_dsl;

class CalendarDsl {
static Calendar at(Calendar self, Integer time) {
    self.set(Calendar.HOUR_OF_DAY, time)
    return self
}

    static Integer getMonth(GregorianCalendar self) {
        self.get(Calendar.MONTH) + 1
    }

    static Integer getYear(GregorianCalendar self) {
        self.get(Calendar.YEAR)
    }

    static String getHour(GregorianCalendar self) {
        def hour = self.get(Calendar.HOUR_OF_DAY)
        (hour > 12 ? hour - 12 : hour) + ' ' + (hour > 12 ? "PM" : "AM")
    }

    static void setHour(GregorianCalendar self,
                        Integer targetHour) {
        self.set(Calendar.HOUR_OF_DAY, targetHour)
    }
}
```

time span demo

```
use (CalendarDsl, IntegerWithTimeSupport) {
    def t = Calendar.instance

    def c =  2.days.fromToday.at(4.pm)

    println "Today:${t.month}/${t.day}/${t.year} at ${t.hour}"
    println "Target:${c.month}/${c.day}/${c.year} at ${c.hour}"
}
```

who returns what?

```
2.days.fromToday.at(4.pm)
```

2	**Integer**
days	**Integer**
fromToday	**Calendar**
at	**Calendar**
4.pm	**Integer**

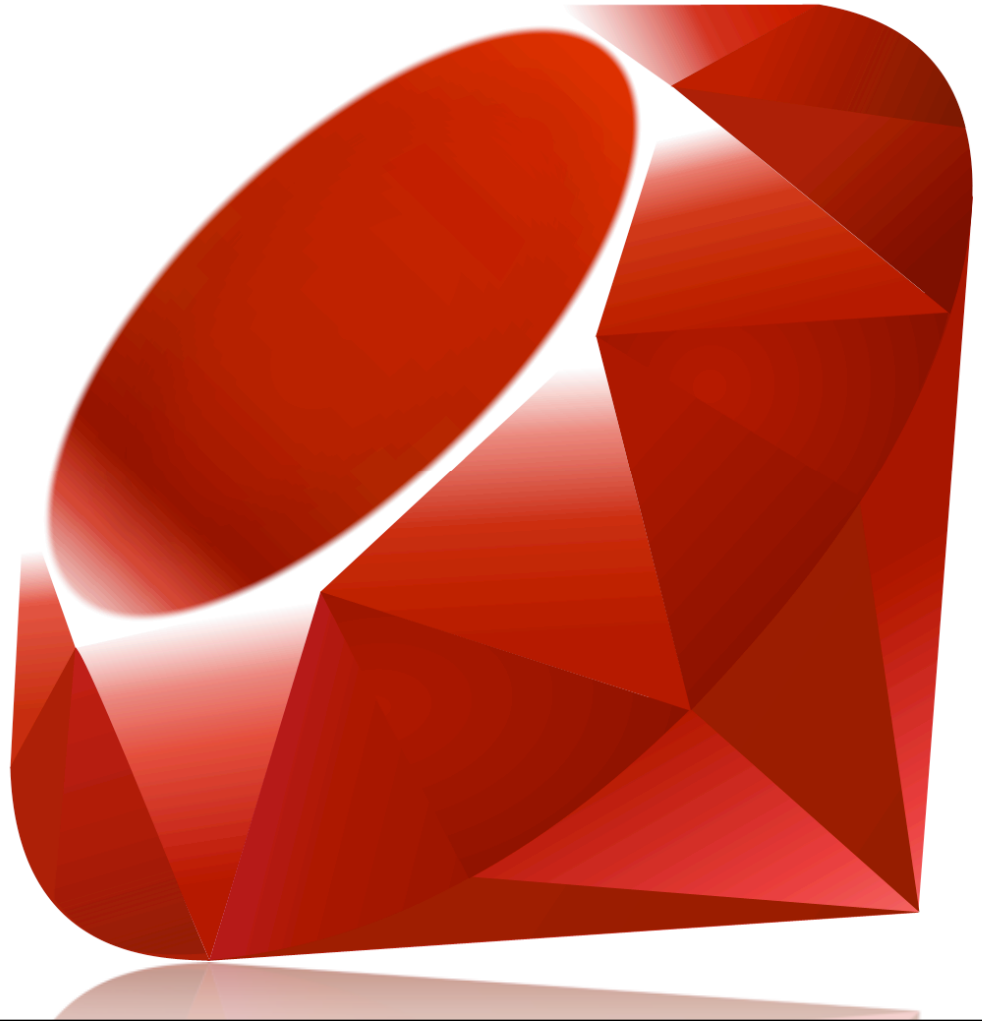
appointment calendar

```
public AppointmentCalendarDemo() {
    def calendar = new AppointmentCalendar()

    use (IntegerWithTimeSupport) {
        calendar.add new Appointment("Dentist").from(4.pm)
        calendar.add new Appointment("Conference call").
            from(5.pm).to(6.pm).at("555-123-4321")
    }

    calendar.print()
}
```

testing

```groovy
package com.nealford.comf.time_dsl;

import groovy.util.GroovyTestCase
import java.util.Calendar

class IntegerWithTimeSupportTest extends GroovyTestCase {
    void test_From_today() {
        use (IntegerWithTimeSupport) {
            def seven_days_from_now = Calendar.instance
            seven_days_from_now.roll(Calendar.DAY_OF_MONTH, 7)
            assertEquals seven_days_from_now.get(Calendar.DAY_OF_MONTH),
                7.fromToday.get(Calendar.DAY_OF_MONTH)
        }
    }

    void test_Am() {
        use (IntegerWithTimeSupport) {
            assertEquals 7, 7.am
            assertEquals 1, 1.am
            assertEquals 0, 12.am
        }
    }

    void test_Pm() {
        use (IntegerWithTimeSupport) {
            assertEquals 19, 7.pm
            assertEquals 12, 12.pm
            assertEquals 13, 1.pm
        }
    }
    . . .
```

ruby



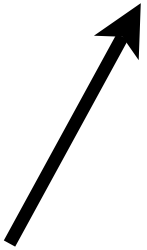
open classes

```
class Integer
  def am(*args)
    self == 12 ? 0 : self
  end

  def pm(*args)
    self == 12 ? 12 : self + 12
  end
end
```

calendar in ruby

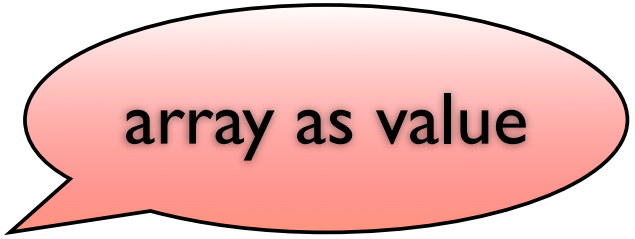
```
calendar.add a('conference call').on('may 3')
calendar.add a('Dentist Appointment').at(4.pm).on :friday
```

added an expression builder



support for days of the week

using named parameters




```
verify :method => :post, :only => [ :destroy, :create, :update ],
       :redirect_to => { :action => :list }
```

```
def verify(list_of_options_as_hash)

end
```

literal collection expression

Form language expressions using literal collection syntax

switching contexts

declarative advantages

declarative code reveals semantic intent

imperative code reveals implementation

dsl's allow you to specify intent without coupling in implementation

```
class Farm < Model
  has_many :cows
end

class Cow < Model
  has_one  :farm
  property :name, String
  property :age, Integer
end
```

```
def self.property(name, type)
  setup
  class_name = self.name.sub(/.*::/, '')
  eval(%{
    @@field_names[:#{class_name}][:#{name}] = type

    def #{name}
      @fields[:#{name}]
    end

    def #{name}=(value)
      @fields[:#{name}] = value
    end
  })
end
```

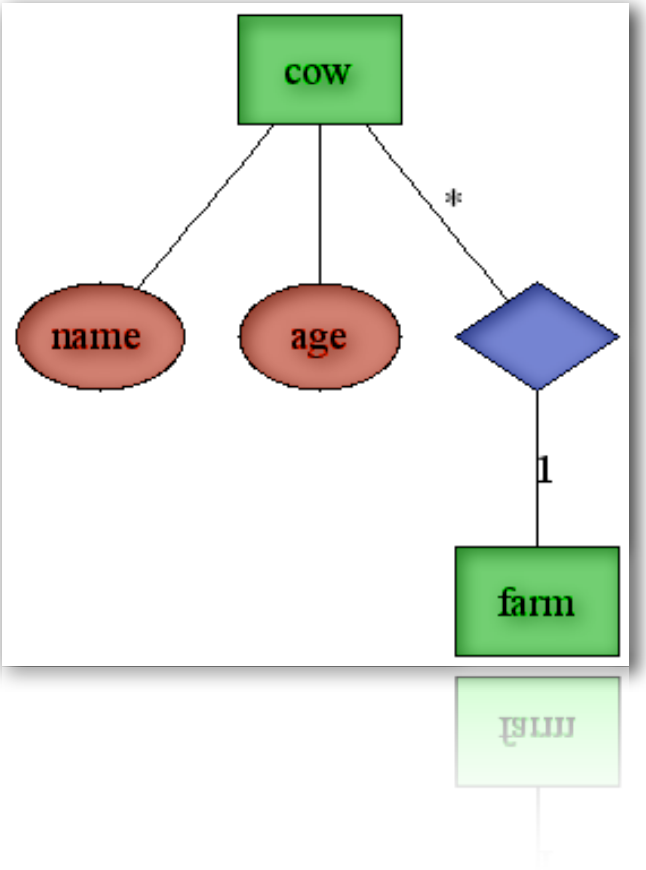
```ruby
def create_table
  class_name = self.class.name.sub(/.*::/, "")
  sql_string = "create table #{class_name} ("

  first = true
  @fields.each do |name, type|
    sql_string += ", " if !first
    first = false
    sql_string += "#{name} #{type}"

  end
  sql_string += ")"
end
```

```
class Farm < Model
  has_many :cows
end

class Cow < Model
  has_one  :farm
  property :name, String
  property :age, Integer
end
```

```
create table Farm (cows Integer)
create table Cow (age Integer, name String, farm Integer)
```

```
graph ER {
  node [shape=box style=filled fillcolor="#00ff005f"];
    farm;
    cow;
  node [shape=ellipse style=filled fillcolor="#ff00005f"];
    {node [label="name"] cow_name;}
    {node [label="age"] cow_age;}
  node [shape=diamond style=filled fillcolor="#0000ff5f"];
    {node [label=""]
      cow_farm;}

  cow -- cow_name;
  cow -- cow_age;

  cow -- cow_farm [label="*"];
  cow_farm -- farm [label="1"];
}
```

```
class Farm < Model
  has_many :cows
end

class Cow < Model
  has_one  :farm
  property :name, String
  property :age, Integer
end
```




```
create table Farm (cows Integer)
create table Cow (age Integer, name String, farm Integer)
```


external dsls

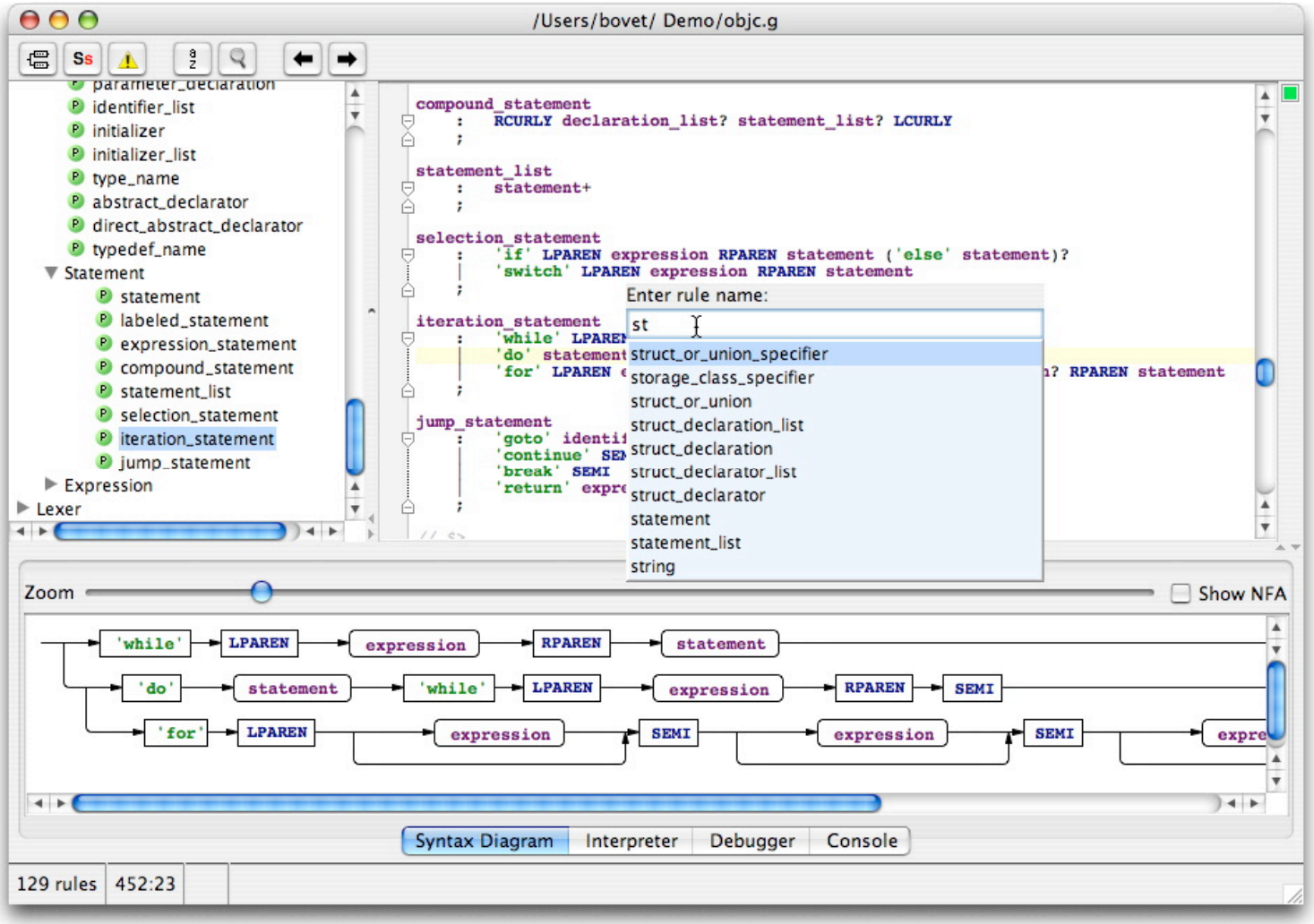
roll your own language

create your own parser & lexer

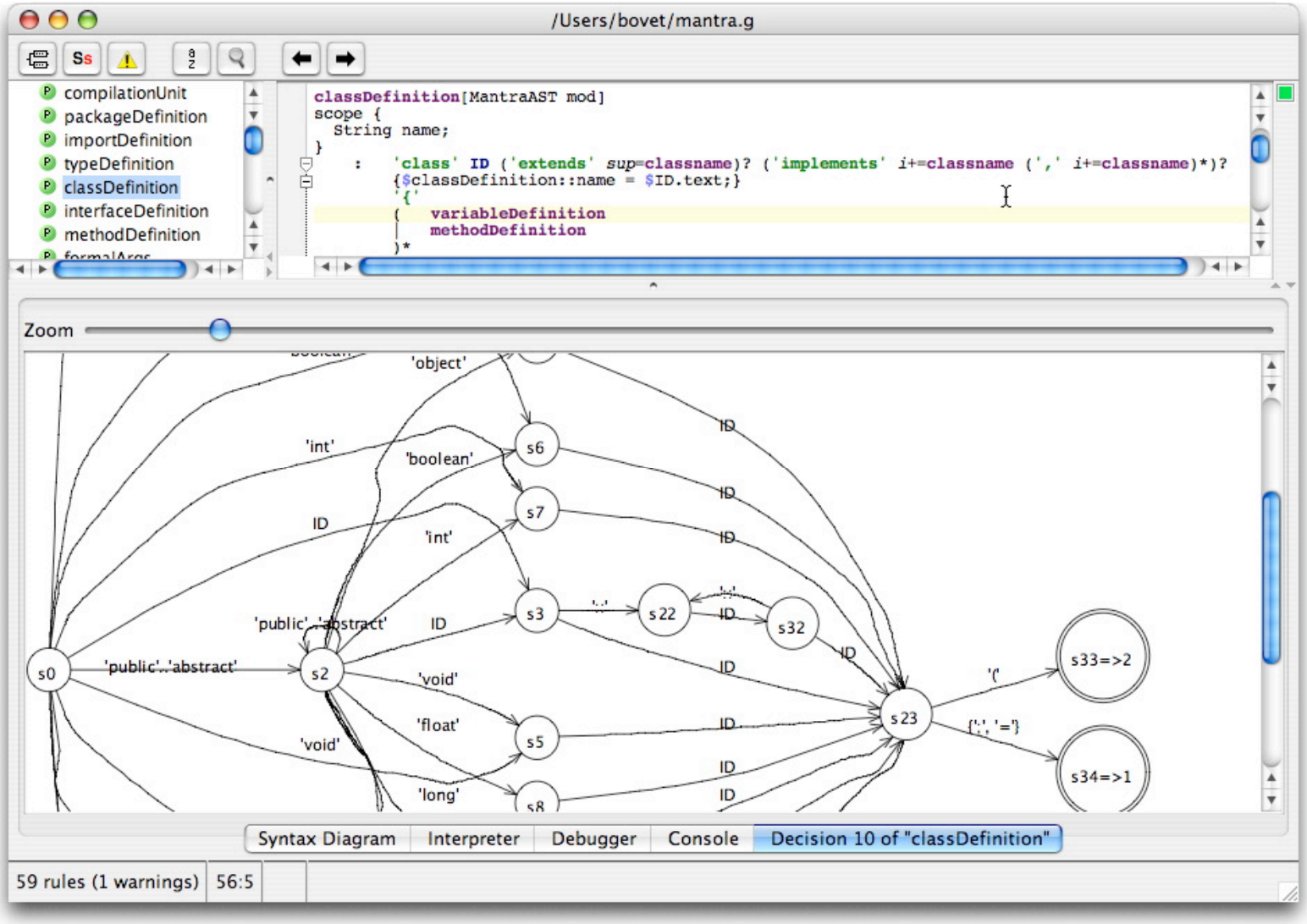
lex/yacc

www.antlr.org

antlrworks


/Users/bovet/ Demo/objc.g
identifier_list
initializer
initializer_list
type_name
abstract_declarator
direct_abstract_declarator
typedef_name
Statement
statement
labeled_statement
expression_statement
compound_statement
statement_list
selection_statement
iteration_statement
jump_statement
Expression
Lexer
compound_statement
: RCURLY declaration_list? statement_list? LCURLY
;
statement_list
: statement+
;
selection_statement
: 'if' LPAREN expression RPAREN statement ('else' statement)?
| 'switch' LPAREN expression RPAREN statement
;
Enter rule name:
st
struct_or_union_specifier
storage_class_specifier
struct_or_union
struct_declaration_list
struct_declaration
struct_declarator_list
struct_declarator
statement
statement_list
string
Zoom
Show NFA
'while'
LPAREN
expression
RPAREN
statement
'do'
statement
'while'
LPAREN
expression
RPAREN
SEMI
'for'
LPAREN
expression
SEMI
expression
SEMI
Syntax Diagram
Interpreter
Debugger
Console
129 rules
452:23

antlrworks


/Users/bovet/mantra.g
compilationUnit
packageDefinition
importDefinition
typeDefinition
classDefinition
interfaceDefinition
methodDefinition
classDefinition[MantraAST mod]
scope {
String name;
}
: 'class' ID ('extends' sup=classname)? ('implements' i+=classname (',' i+=classname)*)?
{$classDefinition::name = $ID.text;}
'{'
(variableDefinition
| methodDefinition
)*
Zoom
'object'
'int'
'boolean'
ID
'public'..'abstract'
'void'
'float'
'long'
s0
s2
s3
s5
s6
s7
s8
s22
s23
s32
s33=>2
s34=>1
Syntax Diagram
Interpreter
Debugger
Console
Decision 10 of "classDefinition"
59 rules (1 warnings)
56:5

language workbenches

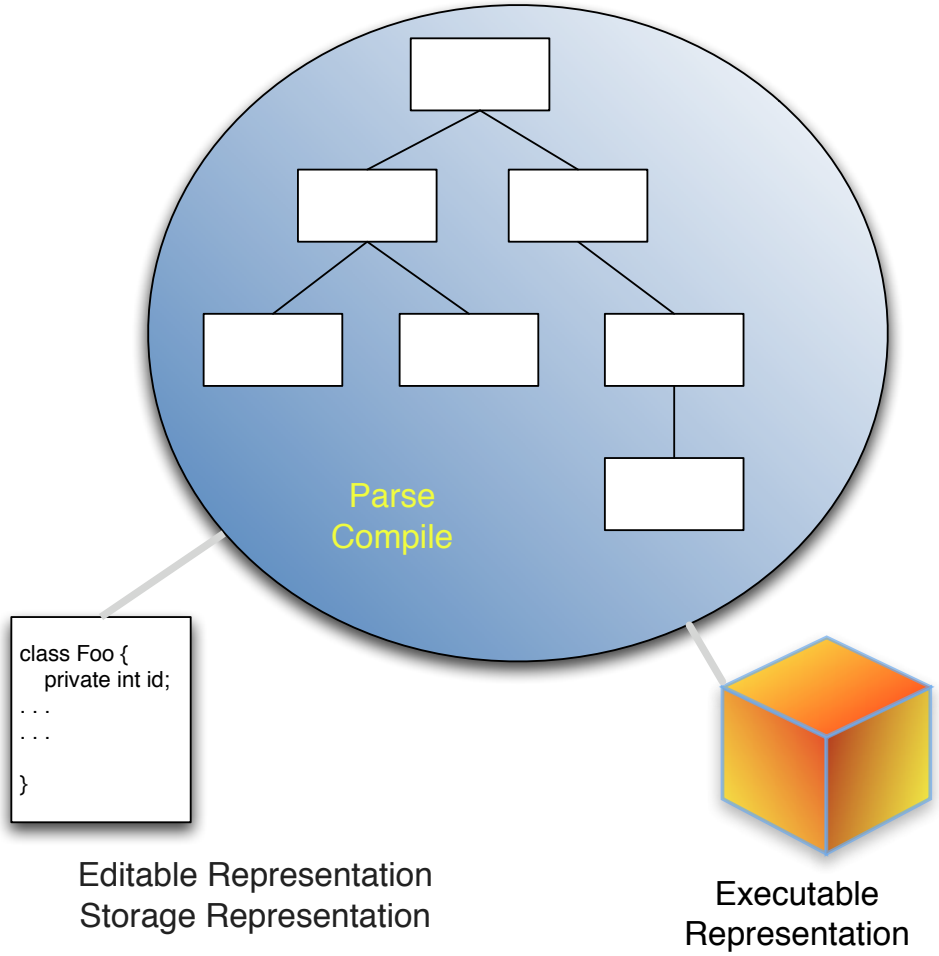
a tool that supports language oriented programming

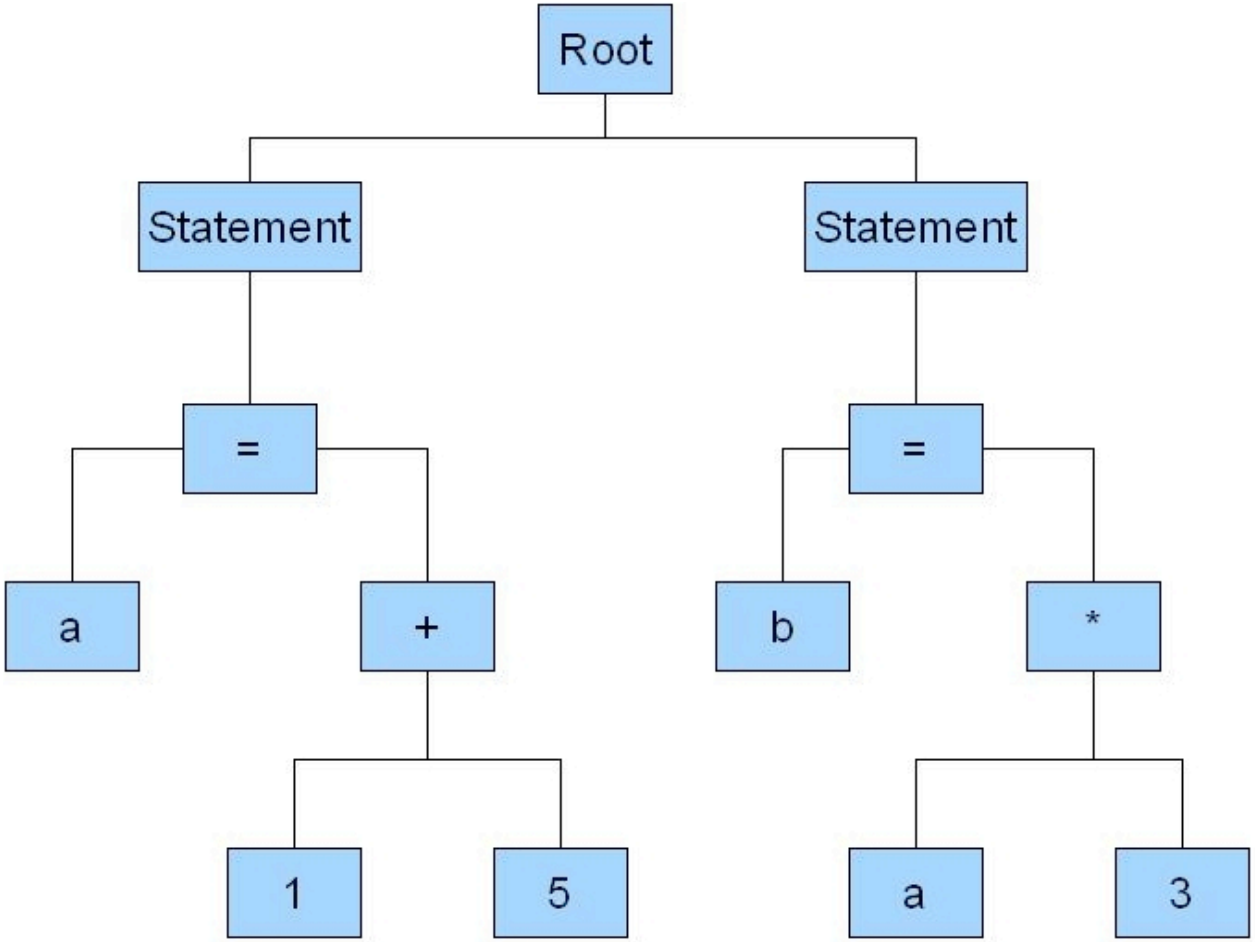
intentional software (charles simonyi)

software factories (microsoft)

mps (jetbrains)

compilation since cs-101



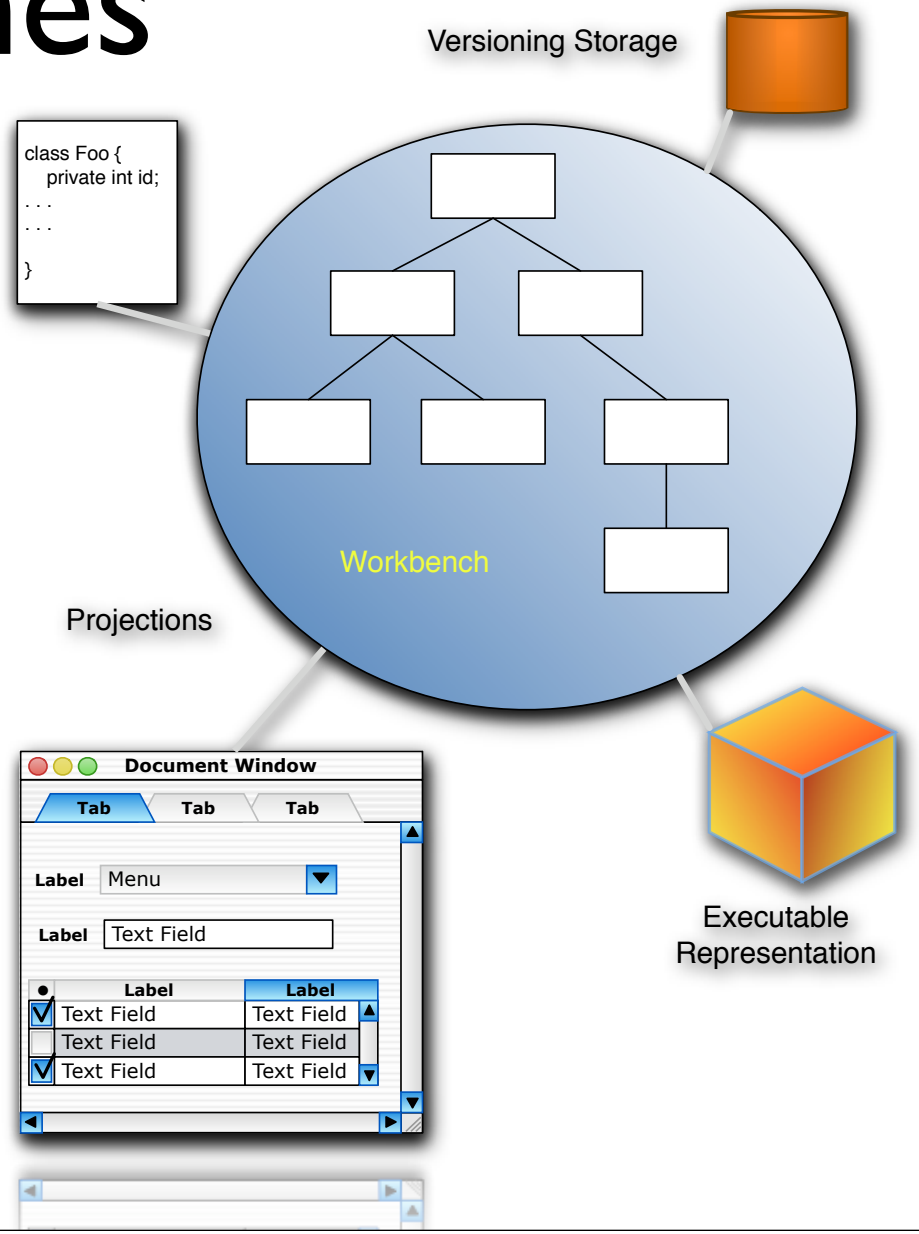

Root
Statement
Statement
=
=
a
+
b
*
1
5
a
3

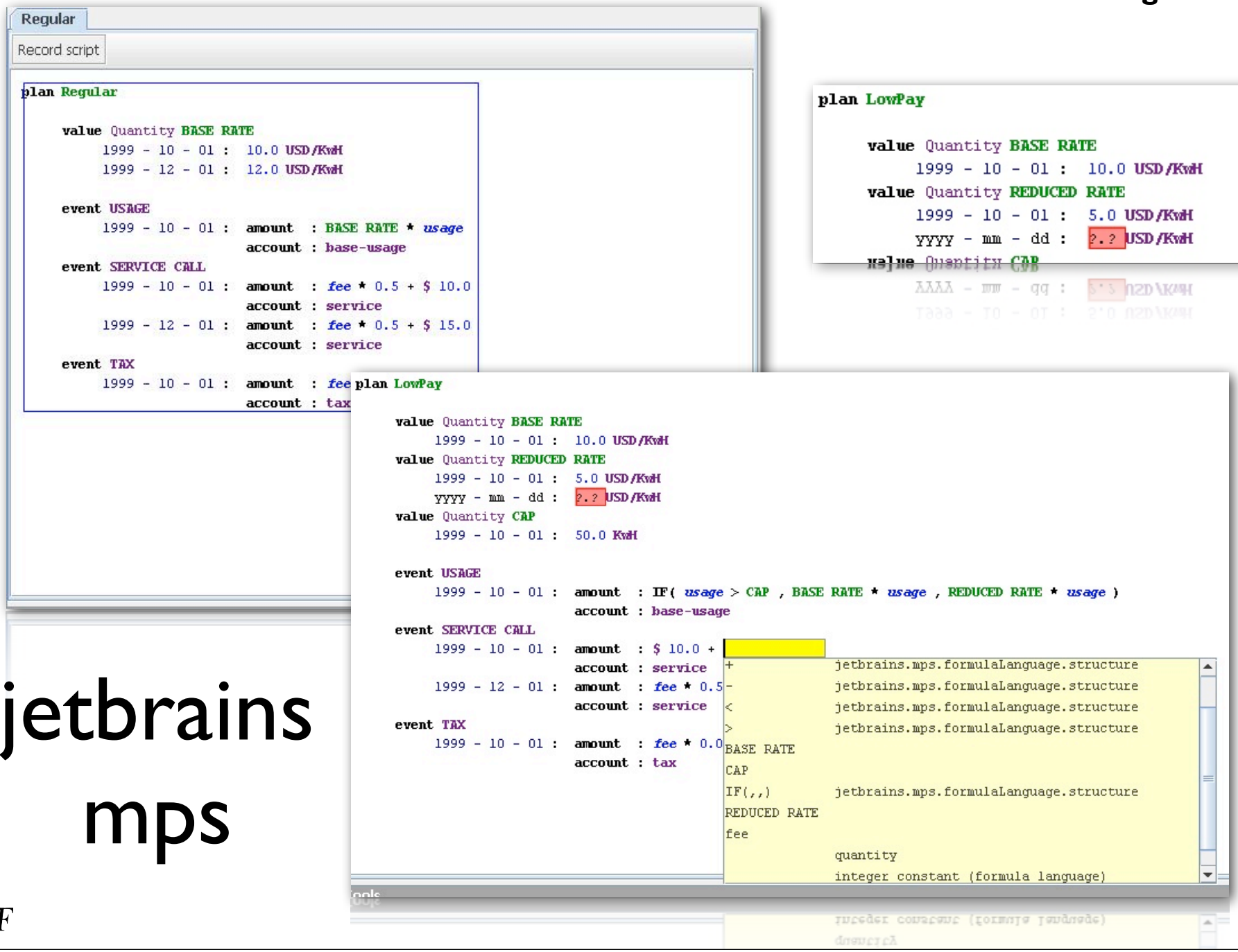
“post-  ide’s”

first java ide to edit the abstract syntax directly

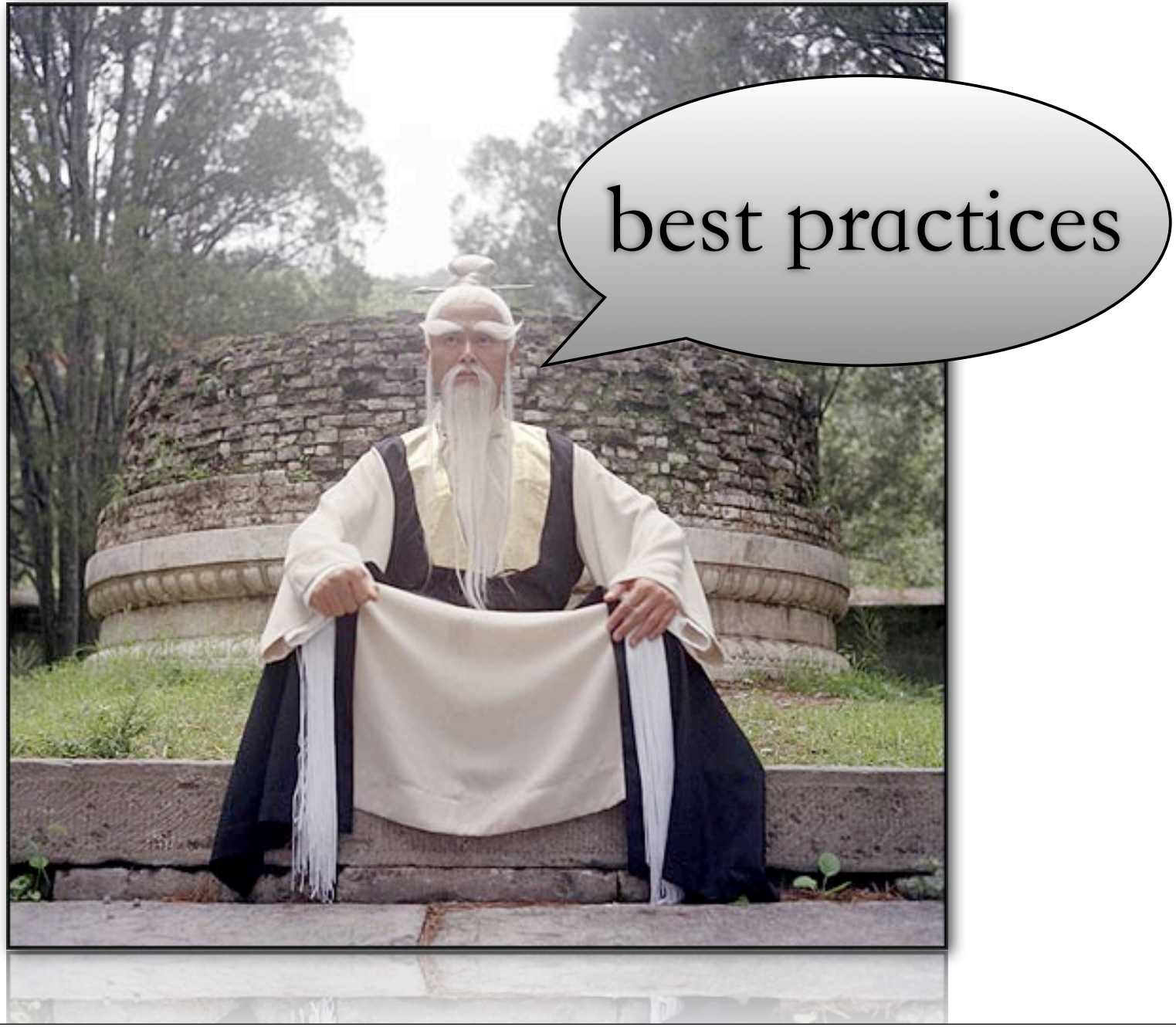
enables refactoring

workbenches





jetbrains mps


best practices

envision the perfect result

what is the ideal dsl syntax?

build towards it

the rake napkin

test,

test,

test

narrow the problem domain

keep your dsl as cohesive as possible

create solutions by composing dsls...

...not creating more complicated languages

jetbrain's use of mps

resources

http://martinfowler.com/bliki/DomainSpecificLanguage.html

http://martinfowler.com/articles/languageWorkbench.html

http://www.theserverside.com/news/thread.tss?thread_id=46674

http://homepages.cwi.nl/~arie/papers/dslbib/

upcoming book on building internal dsl's in ruby

zak tamsen, jeremy stell-smith,

dan manges, neal ford

questions?

please fill out the session evaluations
slides & samples available at nealford.com

NEAL FORD thoughtworker / meme wrangler

ThoughtWorks

14 Wall St, Suite 2019, New York, NY 10005

nford@thoughtworks.com
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com