

# construction techniques for domain specific languages

**NEAL FORD** software architect / meme wrangler

**ThoughtWorks**

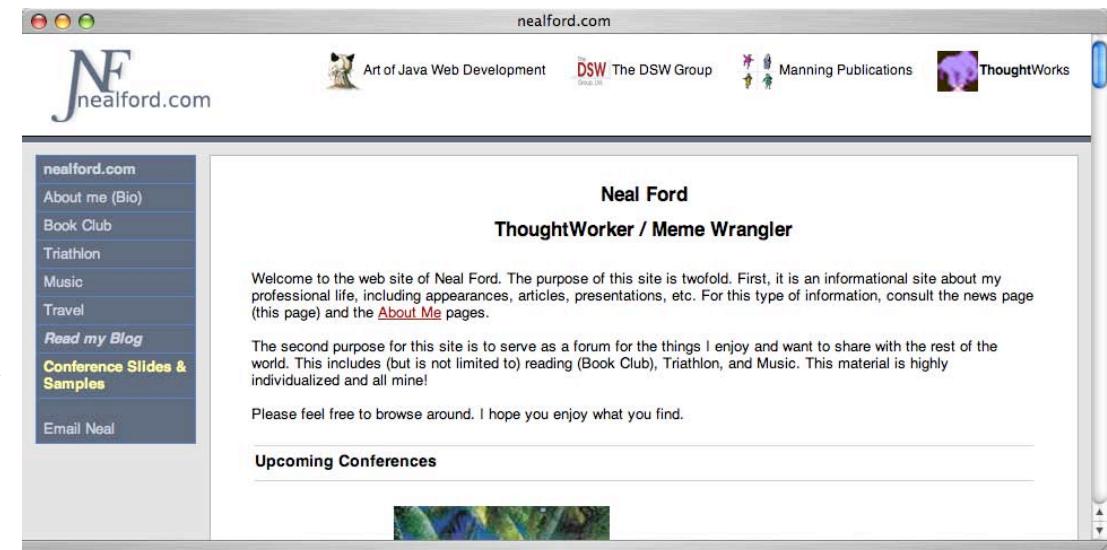
---

[nford@thoughtworks.com](mailto:nford@thoughtworks.com)  
3003 Summit Boulevard, Atlanta, GA 30319  
[www.nealford.com](http://www.nealford.com)  
[www.thoughtworks.com](http://www.thoughtworks.com)  
[memeagora.blogspot.com](http://memeagora.blogspot.com)

# housekeeping

ask questions anytime

download slides from  
nealford.com



download samples from [github.com/nealford](https://github.com/nealford)

# agenda

an extended example

definitions

techniques for supplying implicit context

lots of language techniques in java, groovy, and  
ruby

dsl variants

# secret compartments

a company that supplies secret compartments

written in java, utilizing left-over java-powered  
toasters

expensive to re-flash the toasters

general behavior

configuration



a word about  
patterns

# Mrs. H

1. close the door

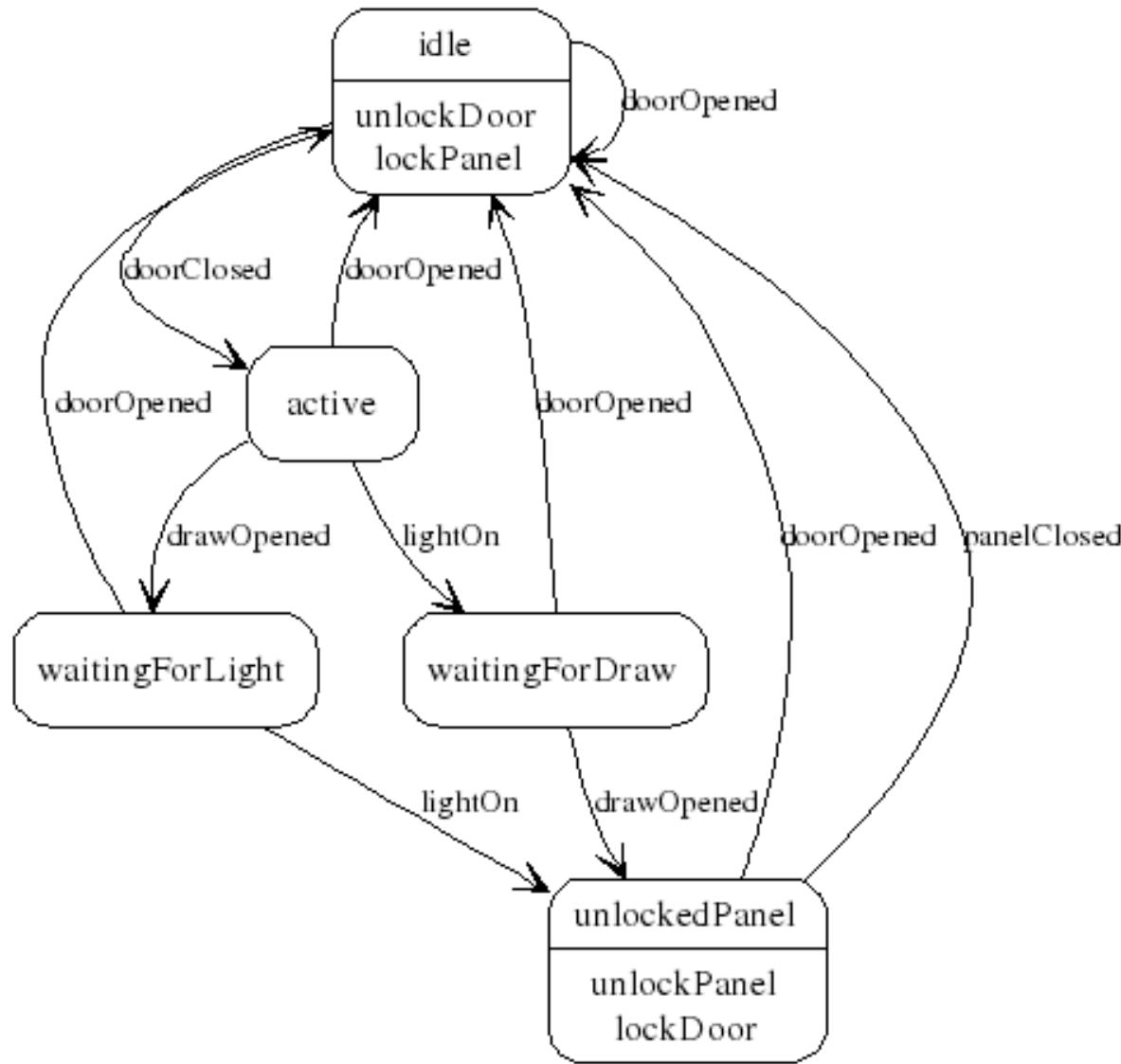
2. open the second draw in her chest

3. turn her bedside light on

must be in sequence

missed steps requires restart of the sequence

# state diagram



# events

```
class AbstractEvent
    private String name, code;

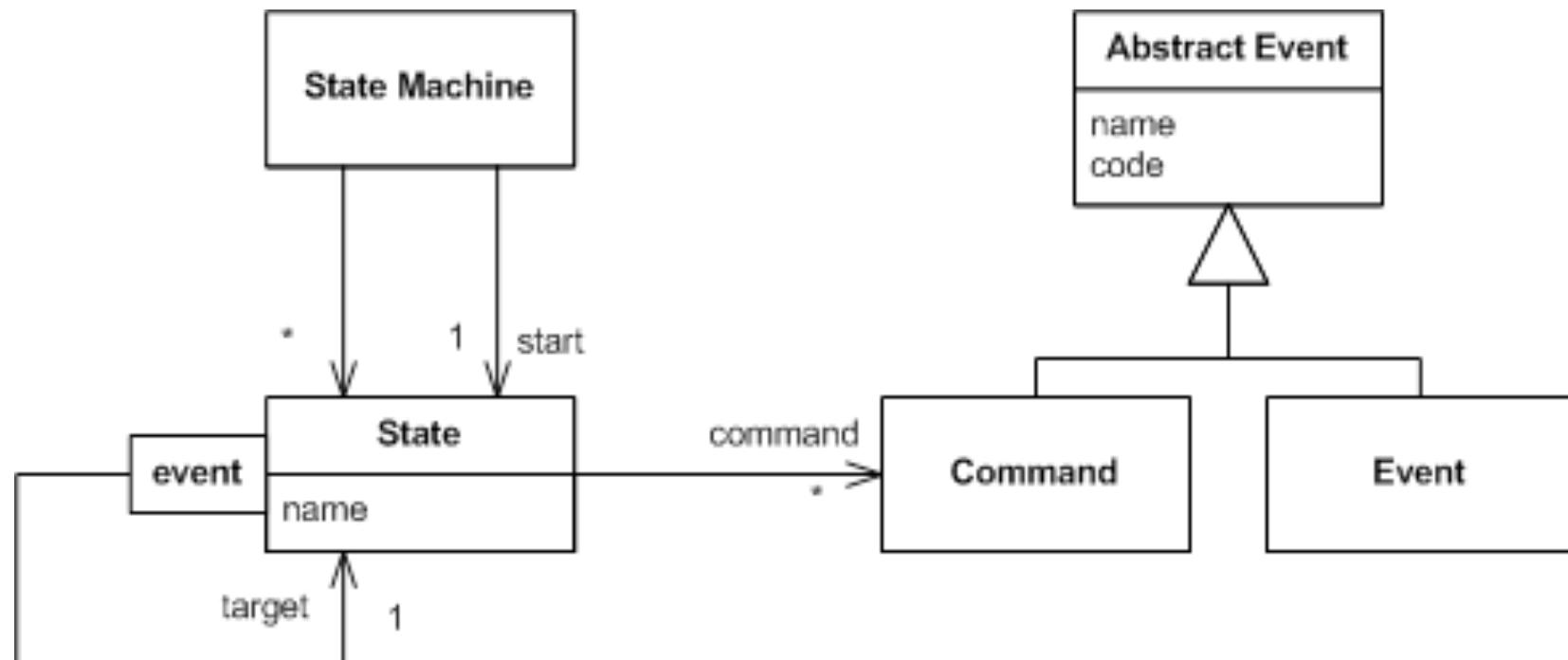
    public AbstractEvent(String name, String code) {
        this.name = name;
        this.code = code;
    }

    public String getCode() { return code; }
    public String getName() { return name; }
}

public class Command extends AbstractEvent {}

public class Event extends AbstractEvent {}
```

# states



# states

```
class State...
    private String name;
    private Map<Event, State> transitions = new HashMap<Event, State>();
    private Map<String, State> transitionsByCode = new HashMap<String, State>();
    private List<Command> commands = new ArrayList<Command>();

    public void addTransition(Event event, State targetState) {
        transitions.put(event, targetState);
        transitionsByCode.put(event.getCode(), targetState);
    }
```

# state machine

```
class StateMachine
    private State start;

    public StateMachine(State start) {
        this.start = start;
    }

    public Collection<State> getStates() {
        List<State> result = new ArrayList<State>();
        gatherForwards(result, start);
        return result;
    }

    private void gatherForwards(Collection<State> result, State start) {
        if (result.contains(start)) return;
        else {
            result.add(start);
            for (State next : start.getAllTargets()) gatherForwards(result, next);
            return;
        }
    }
}
```

# configuration

```
Event doorClosed = new Event("doorClosed", "D1CL");
Event drawOpened = new Event("drawOpened", "D20P");
Event lightOn = new Event("lightOn", "L10N");
Event doorOpened = new Event("doorOpened", "D10P");
Event panelClosed = new Event("panelClosed", "PNCL");

Command unlockPanelCmd = new Command("unlockPanel", "PNUL");
Command lockPanelCmd = new Command("lockPanel", "PNLK");
Command lockDoorCmd = new Command("lockDoor", "D1LK");
Command unlockDoorCmd = new Command("unlockDoor", "D1UL");

State idle = new State("idle");
State activeState = new State("active");
State waitingForLightState = new State("waitingForLight");
State waitingForDrawState = new State("waitingForDraw");
State unlockedPanelState = new State("unlockedPanel");
```

# configuration

```
StateMachine machine = new StateMachine(idle);

idle.addTransition(doorClosed, activeState);
idle.addCommand(unlockDoorCmd);
idle.addCommand(clockPanelCmd);

activeState.addTransition(drawOpened, waitingForLightState);
activeState.addTransition(lightOn, waitingForDrawState);

waitingForLightState.addTransition(lightOn, unlockedPanelState);

waitingForDrawState.addTransition(drawOpened, unlockedPanelState);

unlockedPanelState.addCommand(unlockPanelCmd);
unlockedPanelState.addCommand(clockDoorCmd);
unlockedPanelState.addTransition(panelClosed, idle);

machine.addResetEvents(doorOpened);
```

# alternate configuration: XML

```
<stateMachine start = "idle">
    <event name="doorClosed" code="D1CL"/>
    <event name="drawOpened" code="D2OP"/>
    <event name="lightOn" code="L10N"/>
    <event name="doorOpened" code="D10P"/>
    <event name="panelClosed" code="PNCL"/>

    <command name="unlockPanel" code="PNUL"/>
    <command name="lockPanel" code="PNLK"/>
    <command name="lockDoor" code="D1LK"/>
    <command name="unlockDoor" code="D1UL"/>

    <state name="idle">
        <transition event="doorClosed" target="active"/>
        <action command="unlockDoor"/>
        <action command="lockPanel"/>
    </state>

    <state name="active">
        <transition event="drawOpened" target="waitingForLight"/>
        <transition event="lightOn" target="waitForDraw"/>
    </state>

    <state name="waitingForLight">
        <transition event="lightOn" target="unlockedPanel"/>
    </state>

    <state name="waitForDraw">
        <transition event="drawOpened" target="unlockedPanel"/>
    </state>

    <state name="unlockedPanel">
        <action command="unlockPanel"/>
        <action command="lockDoor"/>
        <transition event="panelClosed" target="idle"/>
    </state>

    <resetEvent name = "doorOpened"/>
</stateMachine>
```

# external configuration

late binding

no recompiling toaster code to change  
configuration

more expressive

declarative

ubiquitous

# another representation

```
events
  doorClosed  D1CL
  drawOpened  D20P
  lightOn     L10N
  doorOpened  D10P
  panelClosed PNCL
end

resetEvents
  doorOpened
end

commands
  unlockPanel PNUL
  lockPanel   PNLK
  lockDoor    D1LK
  unlockDoor  D1UL
end

state idle
  actions {unlockDoor lockPanel}
  doorClosed => active
end

state active
  drawOpened => waitingForLight
  lightOn    => waitingForDraw
end

state waitingForLight
  lightOn => unlockedPanel
end

state waitingForDraw
  drawOpened => unlockedPanel
end

state unlockedPanel
  actions {unlockPanel lockDoor}
  panelClosed => idle
end
```

# characteristics of DSLs

very narrow focus

no control structures, exceptions, types, etc.

what's the difference between DSL and XML?

DSL written with a custom parser (ANTLR)

configuration file == DSL

# another alternative

```
event :doorClosed, "D1CL"
event :drawOpened, "D2OP"
event :lightOn, "L1ON"
event :doorOpened, "D1OP"
event :panelClosed, "PNCL"

command :unlockPanel, "PNUL"
command :lockPanel, "PNLK"
command :lockDoor, "D1LK"
command :unlockDoor, "D1UL"

resetEvents :doorOpened

state :idle do
  actions :unlockDoor, :lockPanel
  transitions :doorClosed => :active
end
```

```
state :active do
  transitions :drawOpened => :waitingForLight,
              :lightOn => :waitingForDraw
end

state :waitingForLight do
  transitions :lightOn => :unlockedPanel
end

state :waitingForDraw do
  transitions :drawOpened => :unlockedPanel
end

state :unlockedPanel do
  actions :unlockPanel, :lockDoor
  transitions :panelClosed => :idle
end
```



# types of DSLs

internal (or embedded):

*DSL expressed within the syntax of a general purpose language*

*stylized use of language for a domain specific purpose*

external:

*separate language to the main programming language*

# is this a DSL?

```
Event doorClosed = new Event("doorClosed", "D1CL");
Event drawOpened = new Event("drawOpened", "D2OP");
Event lightOn = new Event("lightOn", "L10N");
Event doorOpened = new Event("doorOpened", "D10P");
Event panelClosed = new Event("panelClosed", "PNCL");

Command unlockPanelCmd = new Command("unlockPanel", "PNUL");
Command lockPanelCmd = new Command("lockPanel", "PNLK");
Command lockDoorCmd = new Command("lockDoor", "D1LK");
Command unlockDoorCmd = new Command("unlockDoor", "D1UL");

State idle = new State("idle");
State activeState = new State("active");
State waitingForLightState = new State("waitingForLight");
State waitingForDrawState = new State("waitingForDraw");
State unlockedPanelState = new State("unlockedPanel");
```

# OK, what about this one?

```
public class BasicStateMachine extends StateMachineBuilder {

    Events doorClosed, drawOpened, lightOn, panelClosed;
    Commands unlockPanel, lockPanel, lockDoor, unlockDoor;
    States idle, active, waitingForLight, waitingForDraw, unlockedPanel;
    ResetEvents doorOpened;

    protected void defineStateMachine() {
        doorClosed. define("D1CL");
        drawOpened. define("D2OP");
        lightOn. define("L1ON");
        panelClosed.define("PNCL");

        doorOpened. define("D1OP");

        unlockPanel.define("PNUL");
        lockPanel. define("PNLK");
        lockDoor. define("D1LK");
        unlockDoor. define("D1UL");
```

# OK, what about *this* one?

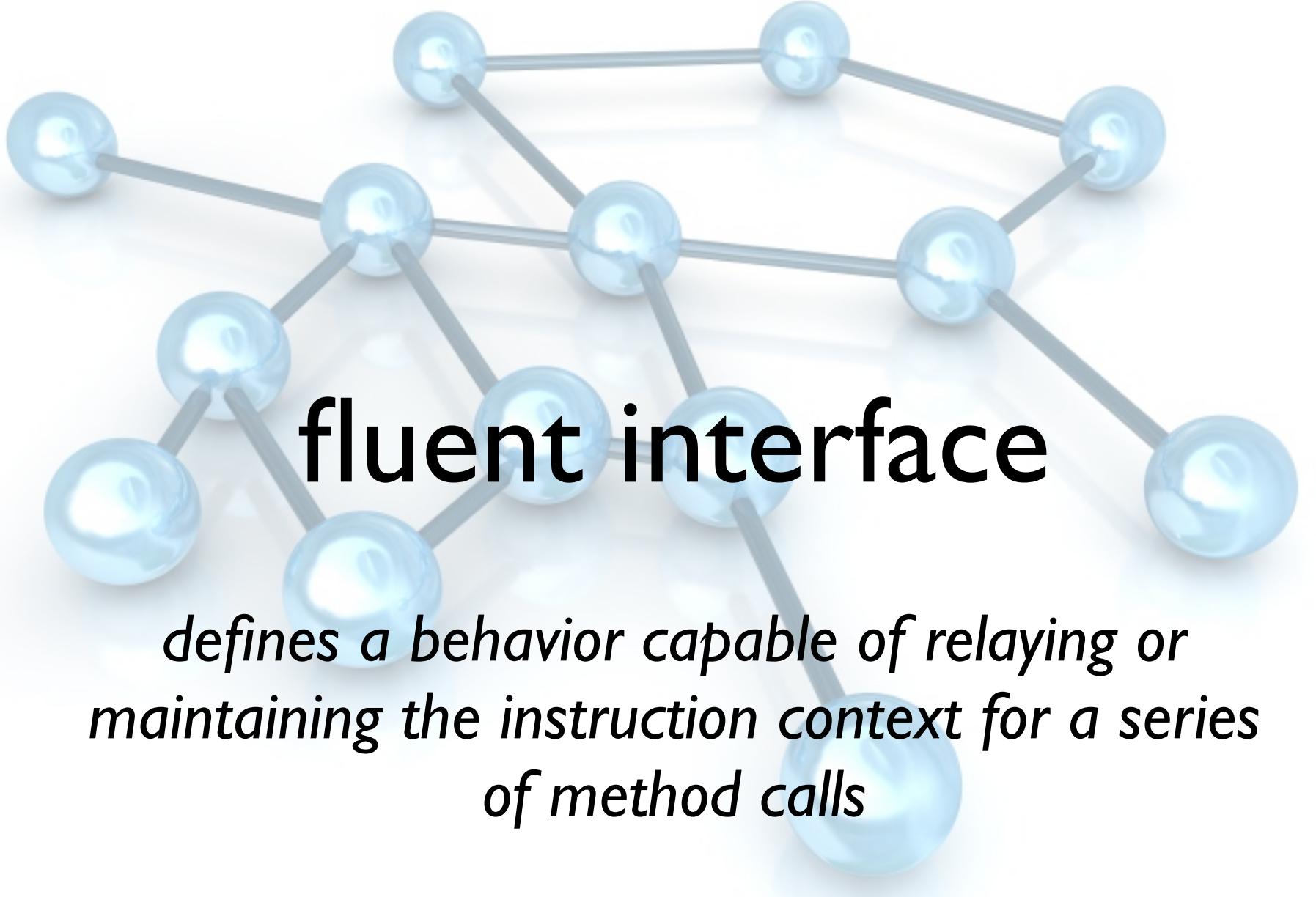
```
.actions(unlockDoor, lockPanel)
    .transition(doorClosed).to(active)
    ;

active
    .transition(drawOpened).to(waitingForLight)
    .transition(lightOn).  to(waitingForDraw)
    ;

waitingForLight
    .transition(lightOn).to(unlockedPanel)
    ;

waitingForDraw
    .transition(drawOpened).to(unlockedPanel)
    ;

unlockedPanel
    .actions(unlockPanel, lockDoor)
    .transition(panelClosed).to(idle)
    ;
}
```



# fluent interface

*defines a behavior capable of relaying or maintaining the instruction context for a series of method calls*

# domain specific language (noun)

*a computer programming language of limited expressiveness focused on a particular domain*

# 5 key elements

computer programming language

language nature

domain focus

limited expressiveness

*not Turing complete*

# DSL or not DSL?

regular expressions?

XSLT?

serialized data structures (i.e., properties file)?

struts-config?

Starbucks?

**context**

# calendars



A stack of various calendar pages showing different designs and colors.

The visible calendar pages include:

- January: A grid-based calendar page.
- February: A grid-based calendar page.
- March: A grid-based calendar page.
- April: A grid-based calendar page.
- May: A grid-based calendar page.
- June: A grid-based calendar page.
- July: A grid-based calendar page.
- August: A grid-based calendar page.
- September: A grid-based calendar page.
- October: A grid-based calendar page.
- November: A grid-based calendar page.
- December: A grid-based calendar page.

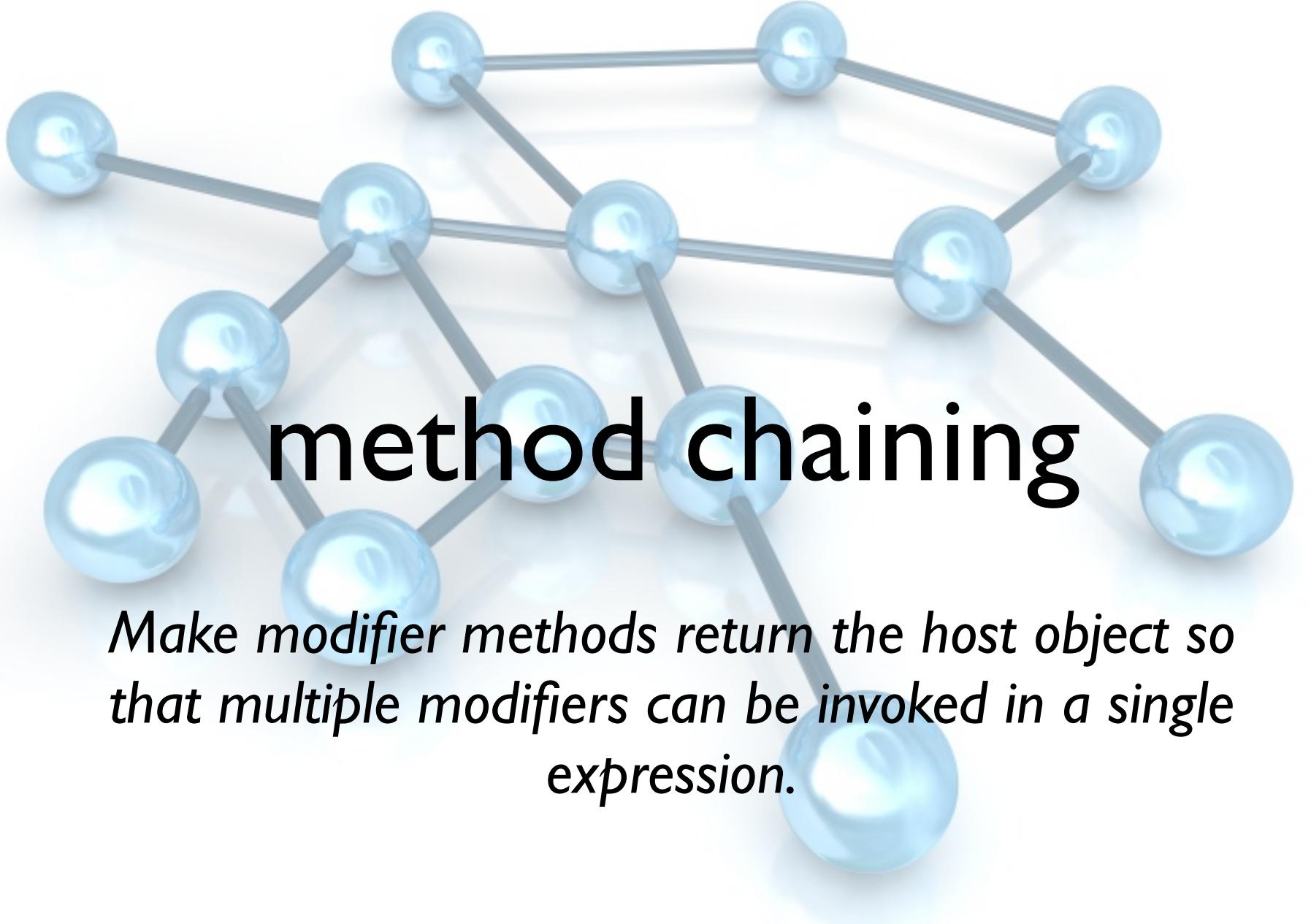
# the goal syntax

```
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);
```



# method chaining

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

# appointments

```
public class Appointment {  
    private String _name;  
    private String _location;  
    private Calendar _startTime;  
    private Calendar _endTime;  
  
    public Appointment at(String location) {  
        _location = location;  
        return this;  
    }  
  
    public Appointment at(Calendar startTime) {  
        _startTime = startTime;  
        return this;  
    }  
  
    public Appointment from(Calendar startTime) {  
        _startTime = startTime;  
        return this;  
    }  
}
```

# appointment calendar

```
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);  
  
        return appt;  
    }  
}
```

# run it!

```
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);  
}
```

```
Appointment:dentist, location:123 main street, Start time:16, End time: 17  
Appointment:birthday party, Start time:16
```

# but!...

```
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);  
        System.out.println(appt.toString());  
        return appt;  
    }  
}
```

# OOPS

```
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);  
}
```

```
Exception in thread "main" java.lang.NullPointerException  
at com.nealford.dsl.calendar.stopping.Appointment.toString(Appointment.java:42)  
at com.nealford.dsl.calendar.stopping.AppointmentCalendarChained.add(AppointmentCalen...  
at com.nealford.dsl.calendar.stopping.CalendarDemoChained.<init>(CalendarDemoChai...  
at com.nealford.dsl.calendar.stopping.CalendarDemoChained.main(CalendarDemoChaine...
```

# finishing problem

```
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);  
        System.out.println(appt.toString());  
        return appt;  
    }  
}
```

# appointment calendar redux

```
public class AppointmentCalendar {  
    private List<Appointment> appointments;  
  
    public AppointmentCalendar() {  
        appointments = new ArrayList<Appointment>();  
    }  
  
    public AppointmentCalendar add(Appointment appt) {  
        appointments.add(appt);  
        return this;  
    }  
  
    public List<Appointment> getAppointments() {  
        return appointments;  
    }  
}
```

# wrapper via parameter

```
calendar.add(  
    new Appointment("Dentist")  
    .at(fourPM));  
calendar.add(  
    new Appointment("Conference Call")  
    .from(fourPM)  
    .to(fivePM)  
    .at("555-123-4321"));  
calendar.add(  
    new Appointment("birthday party")  
    .from(fourPM)  
    .to(fivePM))  
    .add(  
        new Appointment("Doctor")  
        .at("123 Main St"));  
calendar.add(  
    new Appointment("No Fluff, Just Stuff")  
    .at(fourPM));  
displayAppointments(calendar);
```

# wrapping context

method chaining

wrapping via parameters

*functional specification*

# the target

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

    AppointmentCalendar calendar = new AppointmentCalendar();
    calendar.add(new Appointment() {{
        name("dentist");
        from(fourPM);
        to(fivePM);
        at("123 main street");
    }});

    calendar.add(new Appointment() {{
        name("birthday party");
        at(fourPM);
    }});

    displayAppointments(calendar);
}
```

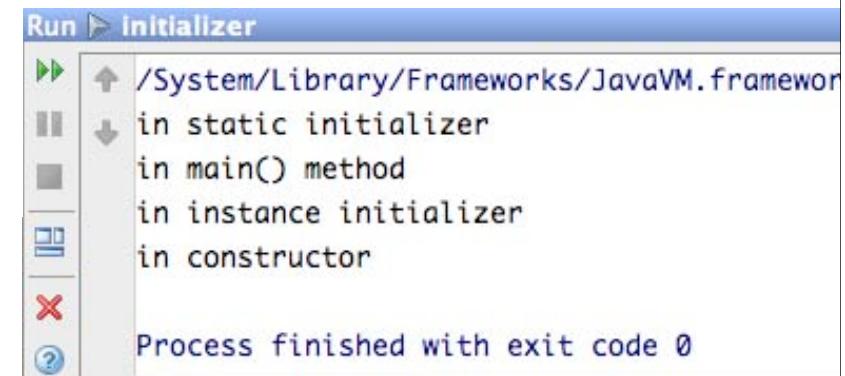
# initializers

```
public class InitializerDemo {  
    public InitializerDemo() {  
        out.println("in constructor");  
    }  
  
    static {  
        out.println("in static initializer");  
    }  
  
    {  
        out.println("in instance initializer");  
    }  
  
    public static void main(String[] args) {  
        out.println("in main() method");  
        new InitializerDemo();  
    }  
}
```

2

4

1  
3



# the target

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

    AppointmentCalendar calendar = new AppointmentCalendar();
    calendar.add(new Appointment() {{
        name("dentist");
        from(fourPM);
        to(fivePM);
        at("123 main street");
    }});

    calendar.add(new Appointment() {{
        name("birthday party");
        at(fourPM);
    }});

    displayAppointments(calendar);
}
```

# changes to Appointment

default constructor

nothing else!

functional specification allows you to reuse  
existing classes with minor impact

fluentizing existing classes...

...with ugly syntax!

# jmock expectations

```
@RunWith(JMock.class)
public class ClassifierWithMockTest {
    Mockery context = new JUnit4Mockery() {{
        setImposteriser(ClassImposteriser.INSTANCE);
    }];

    @Test public void external_factors() {
        final Finder facts = context.mock(Finder.class);
        Classifier7 c = new Classifier7(42, facts);
        final Set<Integer> expected =
            new HashSet(Arrays.asList(1, 2, 3, 6, 7, 21, 14, 42));
        context.checking(new Expectations() {{
            one(facts).factors(); will(returnValue(expected));
       }});
        assertThat(c.sumOfFactors(), is(1 + 2 + 3 + 6 + 7 + 21 + 14 + 42));
        context.assertIsSatisfied();
    }
}
```

# limits in java

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

    AppointmentCalendar calendar = new AppointmentCalendar();
    calendar.add(new Appointment() {{
        name("dentist");
        from(fourPM);
        to(fivePM);
        at("123 main street");
    }});

    calendar.add(new Appointment() {{
        name("birthday party");
        at(fourPM);
    }});

    displayAppointments(calendar);
}
```



# support for DSLs

dynamic typing

closures for context

two different open class techniques:

categories

expando metaclass

# groovy calendars

```
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")  
}
```

# categories

```
class IntegerWithTimeSupport {  
    static Integer getAm(Integer self) {  
        self == 12 ? 0 : self  
    }  
  
    static Integer getPm(Integer self) {  
        self == 12 ? 12 : self + 12  
    }  
  
    static Calendar getFromToday(Integer self) {  
        def target = Calendar.instance  
        target.roll(Calendar.DAY_OF_MONTH, self)  
        return target  
    }  
}
```

# am / pm

```
@Test void am_returns_number() {
    use (IntegerWithTimeSupport) {
        1.upto(11) { i ->
            assertThat(i.am, is(i))
        }
        assertThat(12.am, is(0))
    }
}

@Test void pm_returns_correct_number() {
    use (IntegerWithTimeSupport) {
        1.upto(11) { i ->
            assertThat(i.pm, is(i + 12))
        }
        assertThat(12.pm, is(12))
    }
}
```

# categories

```
class IntegerWithTimeSupport {  
    static Integer getAm(Integer self) {  
        self == 12 ? 0 : self  
    }  
  
    static Integer getPm(Integer self) {  
        self == 12 ? 12 : self + 12  
    }  
  
    static Calendar getFromToday(Integer self) {  
        def target = Calendar.instance  
        target.roll(Calendar.DAY_OF_MONTH, self)  
        return target  
    }  
}
```

# fromToday

```
@Test void from_today_gets_correct_date() {
    use (IntegerWithTimeSupport) {
        def expected = Calendar.instance
        expected.roll(Calendar.DAY_OF_MONTH, 4)
        assertThat(4.fromToday.DAY_OF_MONTH, is(expected.DAY_OF_MONTH))
    }
}

@Test void from_today_responds_to_negative_days() {
    use (IntegerWithTimeSupport) {
        def expected = Calendar.instance
        expected.roll(Calendar.DAY_OF_MONTH, -4)
        assertThat((-4).fromToday.DAY_OF_MONTH, is(expected.DAY_OF_MONTH))
    }
}
```

# categories

```
class IntegerWithTimeSupport {  
    static Integer getAm(Integer self) {  
        self == 12 ? 0 : self  
    }  
  
    static Integer getPm(Integer self) {  
        self == 12 ? 12 : self + 12  
    }  
  
    static Calendar getFromToday(Integer self) {  
        def target = Calendar.instance  
        target.roll(Calendar.DAY_OF_MONTH, self)  
        return target  
    }  
}
```

# expando target syntax

```
def calendar = new AppointmentCalendar()  
  
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()
```

# expando metaclass

```
Integer.metaClass.getAm = { ->
    delegate == 12 ? 0 : delegate
}
```

```
Integer.metaClass.getPm = { ->
    delegate == 12 ? 12 : delegate + 12
}
```

```
Integer.metaClass.getFromToday = { ->
    def target = Calendar.instance
    target.roll(Calendar.DAY_OF_MONTH, delegate)
    target
}
```

# expando tests

```
class TestIntegerMeta {  
  
    @Test void am() {  
        0.upto(11) {  
            assertThat it, is(it.am)  
        }  
    }  
  
    @Test void pm() {  
        0.upto(11) {  
            assertThat it+12, is(it.pm)  
        }  
    }  
  
    @Test void from_today() {  
        0.upto(100) {  
            def c = Calendar.instance  
            c.roll(Calendar.DAY_OF_MONTH, it)  
            assertThat it.fromToday.DAY_OF_MONTH, is(c.DAY_OF_MONTH)  
        }  
    }  
}
```

# category vs. expando

```
def calendar = new AppointmentCalendar()

use (IntegerWithTimeSupport) {
    calendar.add new Appointment("Dentist")
        .from(4.pm)
    calendar.add new Appointment("Conference call")
        .from(5.pm)
        .to(7.pm)
        .at("555-123-4321")
}

calendar.print()

def calendar = new AppointmentCalendar()

    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()
```



behavior driven development testing  
framework

written in groovy

mimics several ruby BDD DSLs

```
def scenario(scenarioDescription, scenarioClosure)

scenario "a valid person has been entered", {

    when "filling out the person form with a first and last name", {
        selenium.open("http://acme.racing.net/greport/personracereport.html")
        selenium.type("fname", "Britney")
        selenium.type("lname", "Smith")
    }
    def when(whenDescription, closure = {})

    and "the submit link has been clicked", {
        selenium.click("submit")
    }

    then "the report should have a list of races for that person", {
        selenium.waitForPageToLoad("5000")
        values = ["Mclean 1/2 Marathon", "Reston 5K", "Herndon 10K", "Leesburg 10K"]
        for(i in 0..<values.size()){
            selenium.getText("//table//tr[${(i+3)}]/td").shouldEqual values[i]
        }
    }
}
```

```
description "This is how a Queue must work"

before "initialize the queue for each spec", {
    queue = new Queue()
}

it "should dequeue item just enqueued", {
    queue.enqueue(2)
    queue.dequeue().shouldBe(2)
}

it "should throw an exception when null is enqueued", {
    ensureThrows(RuntimeException.class) {
        queue.enqueue(null)
    }
}

it "should dequeue items in same order enqueued", {
    [1..5].each {val ->
        queue.enqueue(val)
    }
    [1..5].each {val ->
        queue.dequeue().shouldBe(val)
    }
}
```

```
it "should dequeue items in same order enqueued", {
    [1..5].each {val ->
        queue.enqueue(val)
    }
    [1..5].each {val ->
        queue.dequeue().shouldBe(val)
    }
}

def it(spec, closure) {
    stepStack.startStep(listener, BehaviorStepType.IT, spec)
    closure.delegate = new EnsuringDelegate()
    try {
        if (beforeIt != null) {
            beforeIt()
        }
        listener.gotResult(new Result(Result.SUCCEEDED))
        use(BehaviorCategory) {
            closure()
        }
        if (afterIt != null) {
            afterIt()
        }
    } catch (Throwable ex) {
        listener.gotResult(new Result(ex))
    }
    stepStack.stopStep(listener)
}
```

## ExampleStory

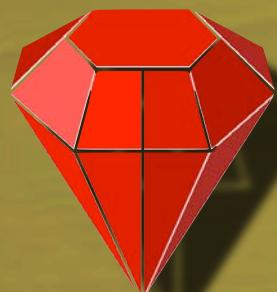
```
it "should dequeue items in same order enqueueued", {  
    [1..5].each {val ->  
        queue.enqueue(val)  
    }  
    [1..5].each {val ->  
        queue.dequeue().shouldBe(val)  
    }  
}
```

## SpecificationKeywords

```
use(BehaviorCategory) {  
    closure()  
}
```

## BehaviorCategory

```
static void shouldBe(Object self, value) {  
    shouldBe(self, value, null)  
}
```



**recipes**

# the goal

```
recipe = Recipe.new "Spicy bread"  
recipe.add 200.grams.of Flour  
recipe.add 1.lb.of Nutmeg
```

# open classes

```
class Numeric
  def gram
    self
  end
  alias_method :grams, :gram

  def pound
    self * 453.59237
  end
  alias_method :pounds, :pound
  alias_method :lb, :pound
  alias_method :lbs, :pound
end
```

# recipe redux

```
recipe = Recipe.new "Spicy bread"  
recipe.add 200.grams.of Flour  
recipe.add 1.lb.of Nutmeg
```

# of

```
class Numeric
  def of ingredient
    if ingredient.kind_of? String
      ingredient = Ingredient.new(ingredient)
    end
    ingredient.quantity = self
    ingredient
  end
end
```

# who returns what?

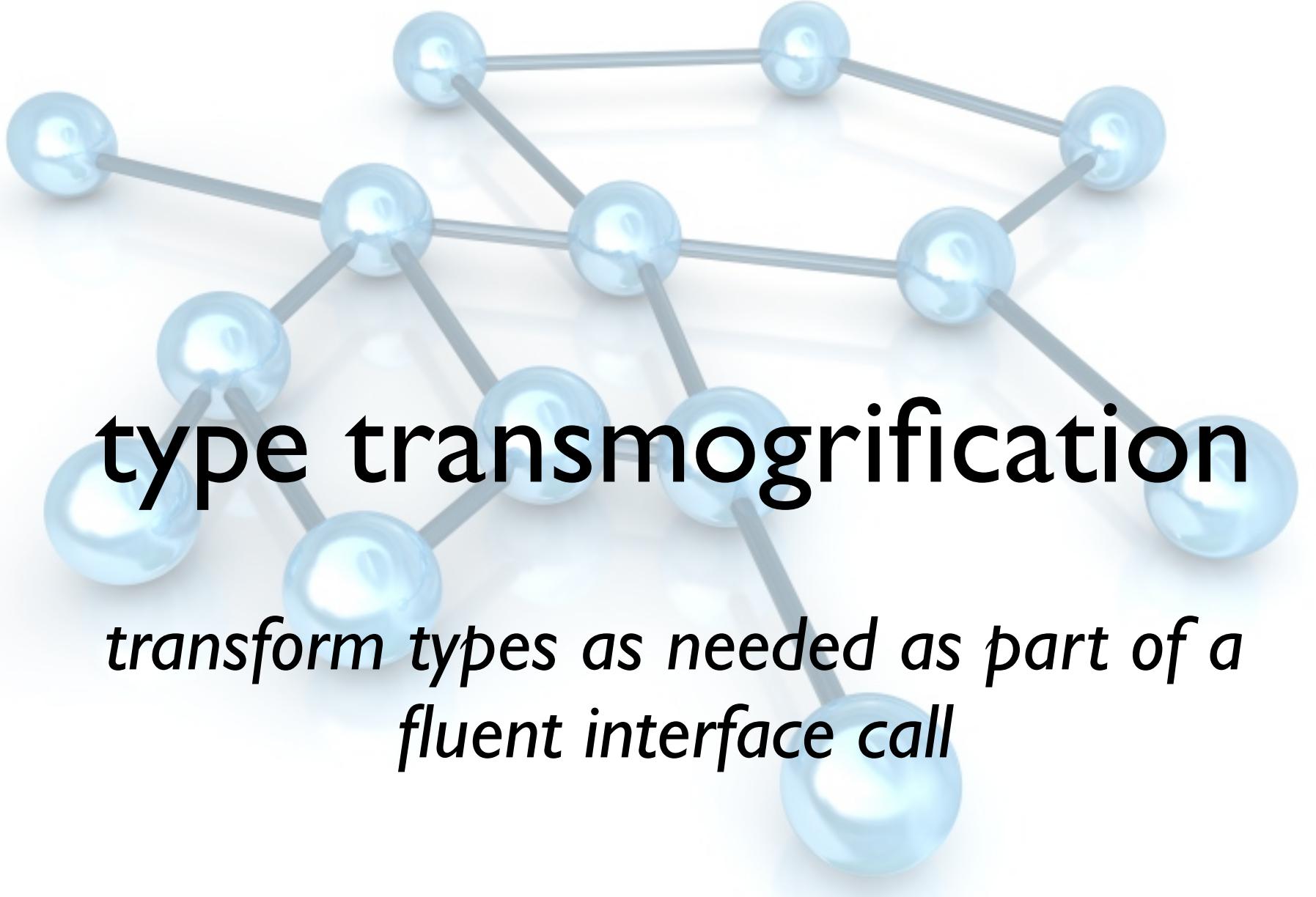
Numeric

Ingredient

1. pound.of("Flour")

Integer

Ingredient



# type transmogrification

*transform types as needed as part of a fluent interface call*

# killing noise characters

```
recipe.add 200.grams.of Flour  
recipe.add 1.lb.of Nutmeg
```

# const\_missing

```
class Object
  def self.const_missing(sym)
    eval "Ingredient.new(sym.to_s)"
  end
end
```



# `constant_missing` factory

*using “missings” as factories to  
create types*

# ingredient factory

yikes!



```
class Object
  def self.const_missing(sym)
    Ingredient.new(sym.to_s)
  end
end
```

# mix it in

```
module IngredientBuilder
  def self.append_features(target)
    def target.const_missing(name)
      Ingredient.new(name.to_s)
    end
    super
  end
end
```

# safer const factories

```
class TestIngredients < Test::Unit::TestCase
  include IngredientBuilder

  def test_ingredient_factory
    i = Flour
    assert i.kind_of? Ingredient
    assert_equal(i.name, "Flour")
  end
```

# smarter const factories

```
module SmartIngredientBuilder
  @@INGREDIENTS = {
    "Flour" => "Flour", "Fluor" => "Flour", "Flower" => "Flour",
    "Flur" => "Flour", "Nutmeg" => "Nutmeg", "Knutmeg" => "Nutmeg"
  }
  def self.append_features(target)
    def target.const_missing(name)
      i = @@INGREDIENTS.keys.find do |val|
        name.to_s == val
      end
      return Ingredient.new(@@INGREDIENTS[i]) unless i.nil?
      raise "No such ingredient"
    end
    super
  end
end
```

```
class TestSmartIngredients < Test::Unit::TestCase
  include SmartIngredientBuilder

  def test_correct_spelling
    i = Flour
    assert i.kind_of? Ingredient
    assert_equal(i.name, "Flour")
  end

  def test_misspelling
    i = Flower
    assert i.kind_of? Ingredient
    assert_equal(i.name, "Flour")
  end

  def test_missing_ingredient
    assert_raise(RuntimeError) {
      i = BakingSoda
    }
  end
end
```

# shotgun approach to open classes

*don't provide universe-wide access to the whacky  
stuff you've implemented for your **dsl***

**control your context**

# context

implicit context tersifies **dsl's**

# context

```
def test_verbose_syntax
  recipe = Recipe.new "Milky Gravy"
  recipe.add 1.lb.of Flour
  recipe.add 200.grams.of Milk
  recipe.add 1.gram.of Nutmeg
  assert_equal 3, recipe.ingredients.size
end
```

```
def test_consists_of
  recipe = Recipe.new "Milky Gravy"
  recipe.consists_of {
    add 1.lb.of Flour
    add 200.grams.of Milk
    add 1.gram.of Nutmeg
  }
  assert_equal 3, recipe.ingredients.size
end
```

# add context

```
def consists_of &block  
  instance_eval &block  
end
```

evaluates ruby code by switching *self* to the  
instance of the object calling `instance_eval`

# context

```
def test_consists_of
  recipe = Recipe.new "Milky Gravy"
  recipe.consists_of {
    add 1.lb.of Flour
    add 200.grams.of Milk
    add 1.gram.of Nutmeg
  }
  assert_equal 3, recipe.ingredients.size
end
```

# expression builder

building a simple language for recipes allows  
you to build other stuff underneath

for example, a nutrition profile

# recipe nutrition profile

```
def nutrition_profile
  profile = NutritionProfile.new
  ingredients.each { |i|
    foo = NutritionProfileDatabase.get_profile_for(i)
    add_to profile, NutritionProfileDatabase.get_profile_for(i)
  }
  profile
end
```

# nutrition profile

```
class NutritionProfile
  attr_accessor :protein, :lipid, :sugars, :calcium, :sodium

  def initialize(protein=0, lipid=0, sugars=0, calcium=0, sodium=0)
    @protein, @lipid, @sugars = protein, lipid, sugars
    @calcium, @sodium = calcium, sodium
  end

  def to_s()
    "\tProtein: " + @protein.to_s + "\n"
    "\tLipid: " + @lipid.to_s + "\n"
    "\tSugars: " + @sugars.to_s + "\n"
    "\tCalcium: " + @calcium.to_s + "\n"
    "\tSodium: " + @sodium.to_s
  end
end
```

# testing profile

```
def test_nutrition_profile_for_recipe
  recipe = Recipe.new
  expected = []
  << 2.lbs.of(Flour) << 1.gram.of(Nutmeg)
  expected.each { |i| recipe.add i}
  protein = 11.5 + 5.84
  lipid = 1.45 + 36.31
  sugar = 1.12 + 28.49
  calcium = 20 + 184
  sodium = 2 + 16
  expected_profile = recipe.nutrition_profile
  assert_equal expected_profile.protein, protein
  assert_equal expected_profile.lipid, lipid
  assert_equal expected_profile.sugars, sugar
  assert_equal expected_profile.calcium, calcium
  assert_equal expected_profile.sodium, sodium
end
```

# profile target

```
ingredient "flour" has Protein=11.5, Lipid=1.45, Sugars=1.12, Calcium=20, Sodium=0  
ingredient "nutmeg" has Protein=5.84, Lipid=36.31, Sugars=28.49, Calcium=184, Sodium=16  
ingredient "milk" has Protein=3.22, Lipid=3.25, Sugars=5.26, Calcium=113, Sodium=40
```

# what is this?

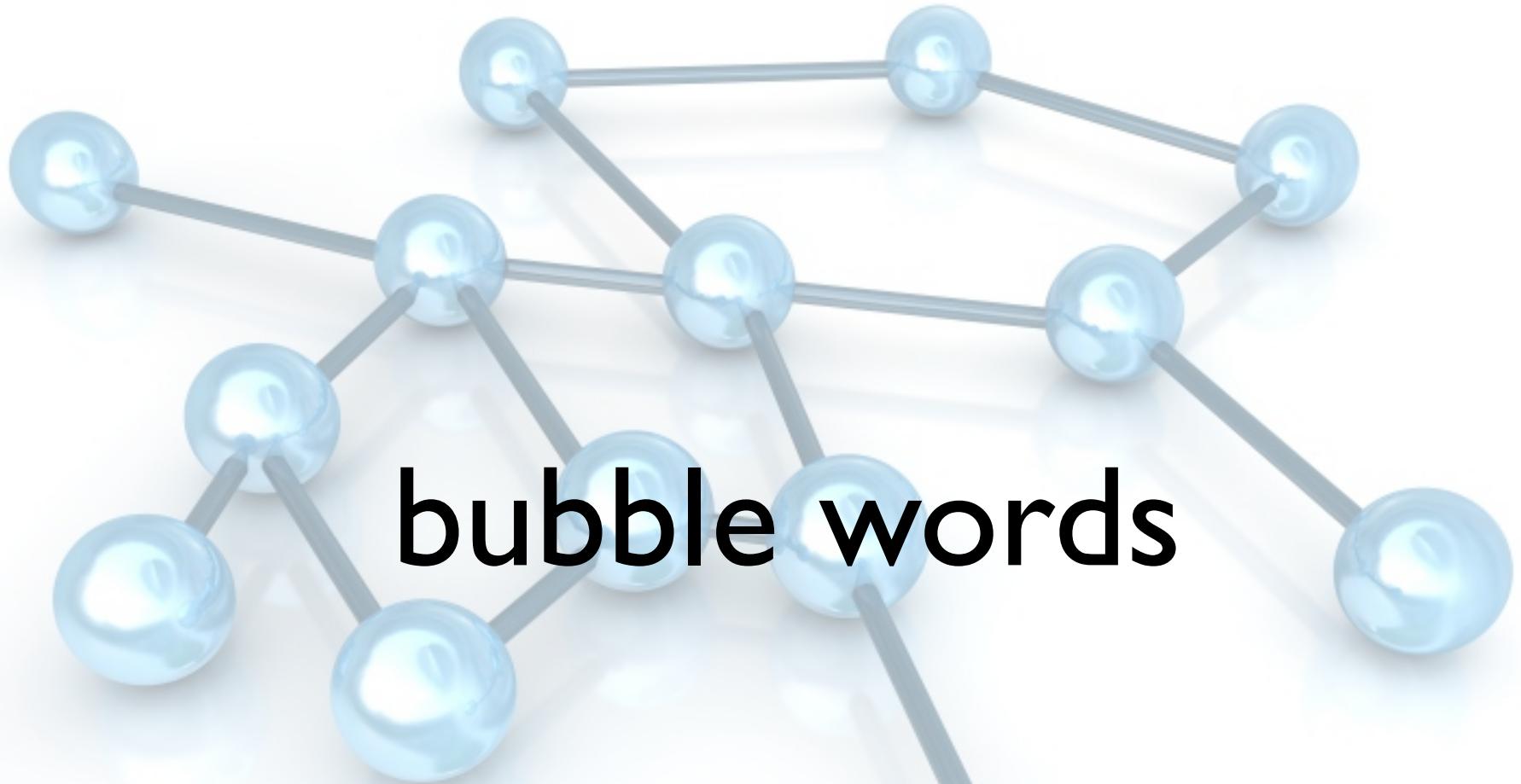
method

“bubble” word

ingredient "flour" has Protein=11.5, Lipid=1.45, ...

1st parameter

2nd parameter(s)



## bubble words

*terms in a dsl that don't contribute  
to the definition but rather to the  
readability*

```
class NutritionProfileDefinition
  class << self
    def const_missing(sym)
      sym.to_s.downcase
    end
  end

  def ingredient(name, ingredients)
    NutritionProfile.create_from_hash name, ingredients
  end

  def process_definition(definition)
    t = polish_text(definition)
    instance_eval polish_text(definition)
  end

  def polish_text(definition_line)
    polished_text = definition_line.clone
    polished_text.gsub!(/=/, '=>')
    polished_text.sub!(/and /, '')
    polished_text.sub!(/has /, ',')
    polished_text
  end

end
```

```
def test_polish_text
  test_text = "ingredient \"flour\" has Protein=11.5, Lipid=1.45, Sugars=1.12, Calcium=20, and Sodium=0"
  expected = 'ingredient "flour" ,Protein=>11.5, Lipid=>1.45, Sugars=>1.12, Calcium=>20, Sodium=>0'
  assert_equal expected, NutritionProfileDefinition.new.polish_text(test_text)
end
```

```
def polish_text(definition_line)
  polished_text = definition_line.clone
  polished_text.gsub!(/=/, '=>')
  polished_text.gsub!(/and /, '')
  polished_text.gsub!(/has /, ',')
  polished_text
end
```

```
def process_definition(definition)
  instance_eval polish_text(definition)
end
```

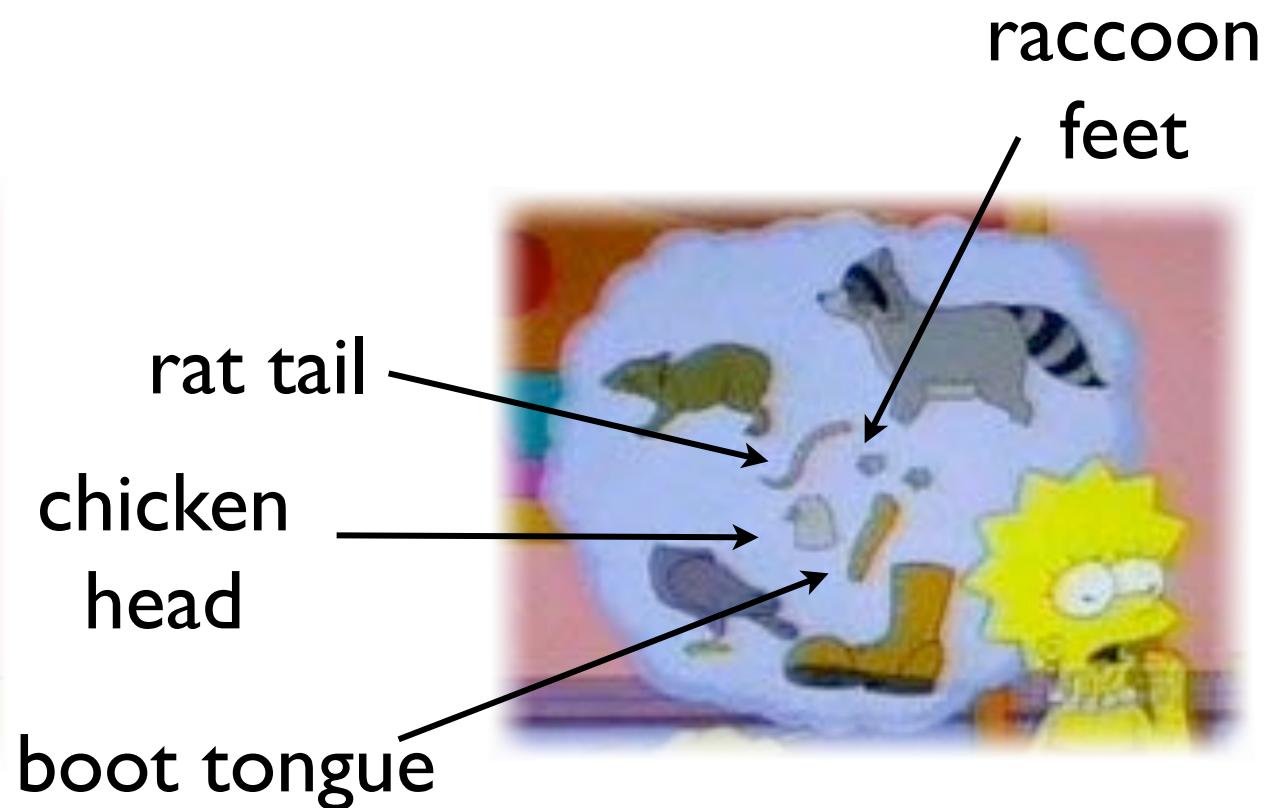
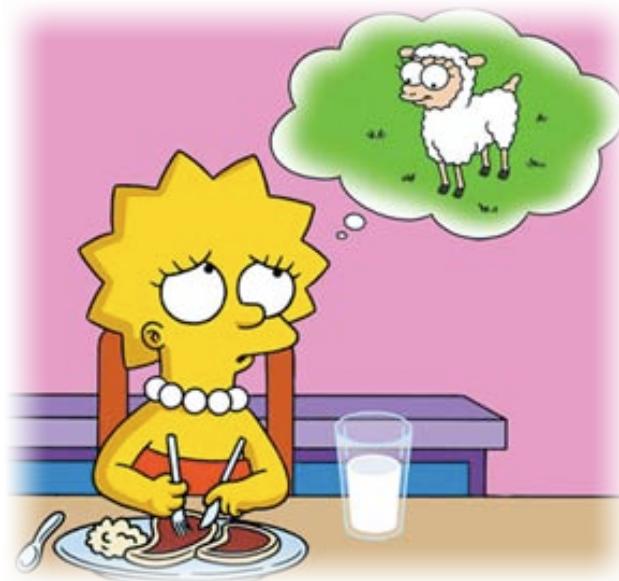
```
'ingredient "flour" ,Protein=>11.5, Lipid=>1.45,
```

```
def ingredient(name, ingredients)
  NutritionProfile.create_from_hash name, ingredients
end
```

polish vs.  
preprocess vs.  
parse



=



# polish

simple string  
substitutions to  
convert *nearly ruby*  
to *actual ruby*



# pre-process

load strings and  
modify to coerce  
them into ruby code



# business natural languages

term defined by jay fields ([www.jayfields.com](http://www.jayfields.com))

use natural language to represent business logic

**bnl** is a **dsl**, but not all **dsl**'s are **bni**'s

# example

employee John Jones

compensate \$2500 for each deal closed in the past 30 days

compensate \$500 for each active deal that closed more than 365 days ago

compensate 5% of gross profits if gross profits are greater than \$1,000,000

compensate 3% of gross profits if gross profits are greater than \$2,000,000

compensate 1% of gross profits if gross profits are greater than \$3,000,000

# process\_payroll.rb

```
Dir[File.dirname(__FILE__) + "/*.bnl"].each do |bnl_file|
  vocabulary = CompensationVocabulary.new(File.basename(bnl_file, '.bnl'))
  compensation = CompensationParser.parse(File.read(bnl_file), vocabulary)
  puts "#{compensation.name} compensation: #{compensation.amount}"
end
```

# vocabulary.rb

```
module Vocabulary

  def phrase(name, &block)
    define_method :"_#{name.to_s.gsub(" ", "_")}", block
  end

end
```

# compensation\_vocabulary.rb

```
class CompensationVocabulary
  extend Vocabulary

  def initialize(data_for)
    @data_for = data_for
  end

  phrase "active deal that closed more than 365 days ago!" do
    SalesInfo.send(@data_for).year_old_deals.to_s
  end

  phrase "are greater than" do
    " > "
  end

  phrase "deal closed in the past 30 days!" do
    SalesInfo.send(@data_for).deals_this_month.to_s
  end

  phrase "for each" do
    "*"
  end
```

# compensation\_parser.rb

```
class CompensationParser

  class << self
    def parse(script, vocabulary)
      root = Root.new(vocabulary)
      script.split(/\n/).each { |line| root.process(preprocess(line)) }
      root
    end

    def preprocess(line)
      line.chomp!
      line.delete!('$')
      line.gsub!(/(\d+)%/, '\1percent')
      line.gsub!(/\s/, '_')
      "_#{line.downcase}!"
    end
  end
end
```

```

class Compensation

  def initialize(vocabulary)
    @phrase, @compensation_logic = '', ''
    @vocabulary = vocabulary
  end

  def method_missing(sym, *args)
    @phrase = reduce(@phrase + sym.to_s)
    if @phrase.any? && sym.to_s =~ /!$/
      raise NoMethodError.new("#{@phrase} not found")
    end
    self
  end

  def reduce(phrase)
    case
      when phrase =~ /^_d+[(percent)|!]*$/ 
        append(extract_number(phrase))
      when @vocabulary.respond_to?(phrase)
        append(@vocabulary.send(phrase))
      else phrase
    end
  end

  def append(piece)
    @compensation_logic += piece
    ""
  end

  def extract_number(string)
    string.gsub(/(\d+)percent$/, '0.0\1').delete('_!')
  end

  def amount
    instance_eval(@compensation_logic) || 0
  end
end

```



**parse**

parse strings (and files) into  
your own language



**context**



Bringing the simplicity of Ruby  
to syntactic analysis.

a DSL for writing *parser expression grammars*

describes a formal language in terms of a set of rules for recognizing strings in the language

makes it easy to write limited grammars in ruby

without going all the way to BNF

```
grammar Arithmetic
    rule additive
        multitive '+' additive / multitive
    end

    rule multitive
        primary '*' multitive / primary
    end

    rule primary
        '(' additive ')' / number
    end

    rule number
        [1-9] [0-9]*
    end
end
```

```
grammar Arithmetic
  rule additive
    multitive '+' additive {
      def value
        multitive.value + additive.value
      end
    }
  /
  multitive
end

# other rules below ...
end
```

usage

inline  
grammar  
rules

```
Treetop.load "arithmetic"

parser = ArithmeticParser.new
if parser.parse('1+1')
  puts 'success'
else
  puts 'failure'
end
```

# summarizing DSLs

limited computer languages

used for:

configuration

fluent interfaces

“little” languages

# summarizing DSLs

context is king!

readability matters...a lot

build solutions by composition, not elaboration

start with the end

evolutionary, not revolutionary

? , S

please fill out the session evaluations  
samples at [github.com/nealford](https://github.com/nealford)



This work is licensed under the Creative Commons  
Attribution-Share Alike 3.0 License.

<http://creativecommons.org/licenses/by-sa/3.0/us/>

NEAL FORD software architect / meme wrangler

ThoughtWorks

nford@thoughtworks.com  
3003 Summit Boulevard, Atlanta, GA 30319  
[www.nealford.com](http://www.nealford.com)  
[www.thoughtworks.com](http://www.thoughtworks.com)  
[memeagora.blogspot.com](http://memeagora.blogspot.com)

# resources



<http://martinfowler.com/bliki/DomainSpecificLanguage.html>  
<http://martinfowler.com/articles/languageWorkbench.html>



[http://www.theserverside.com/news/thread.tss?thread\\_id=46674](http://www.theserverside.com/news/thread.tss?thread_id=46674)



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