RubyFocus

Efficient Rails Test-Driven Development — Week 3

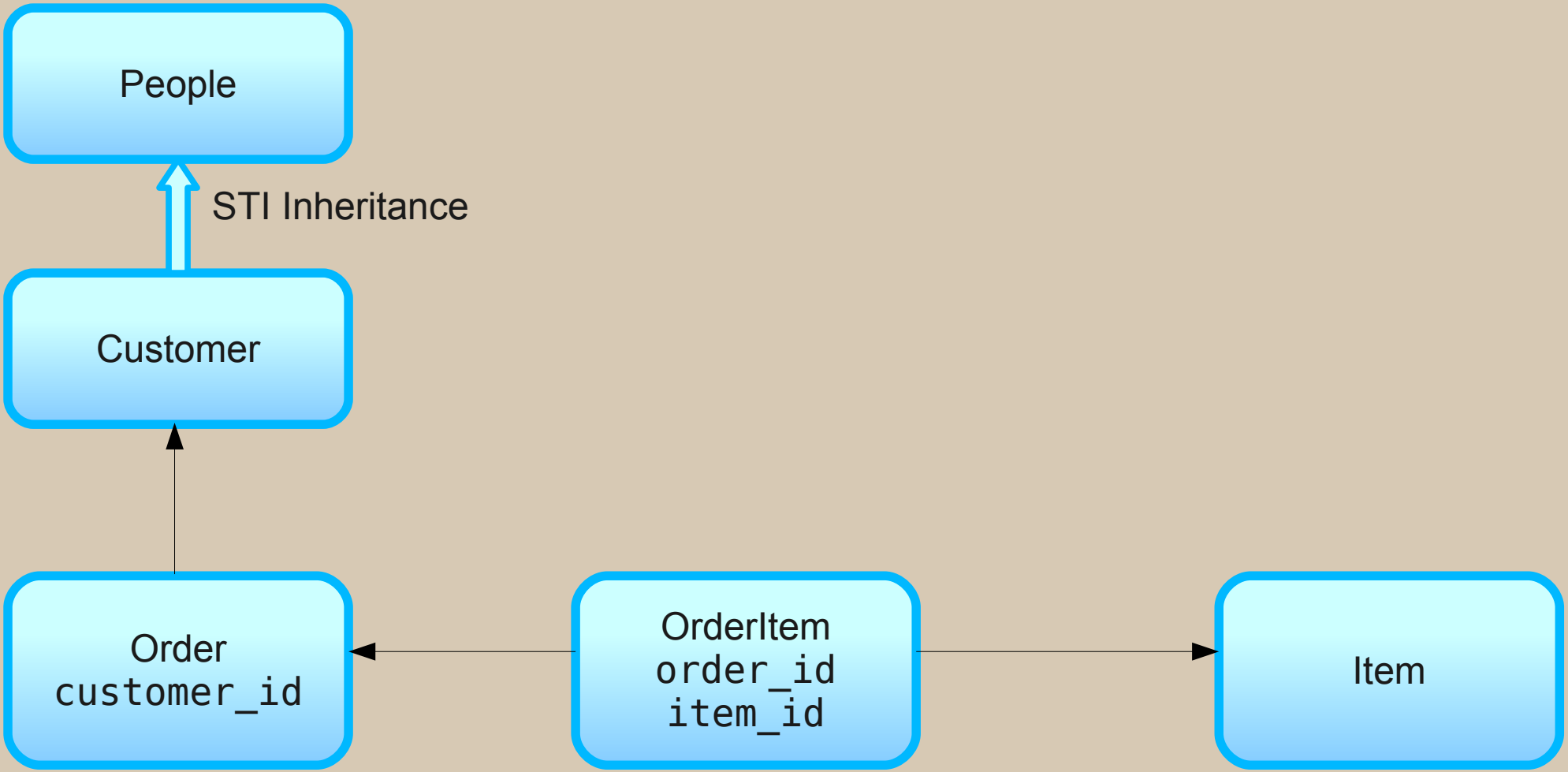
Wolfram Arnold
www.rubyfocus.biz

In collaboration with:
Sarah Allen, BlazingCloud.net
marakana.com

RSpec Subject

```
describe Address do

   it “must have a street” do
     a = Address.new
     a.should_not be_valid
     a.errors.on(:street).should_not be_nil

   end

   #subject { Address.new }  # Can be omitted if .new
                             # on same class as in describe

   it “must have a street” do
     should_not be_valid  # should is called on
                          # subject by default
     subject.errors.on(:street).should_not be_nil

   end


end
```

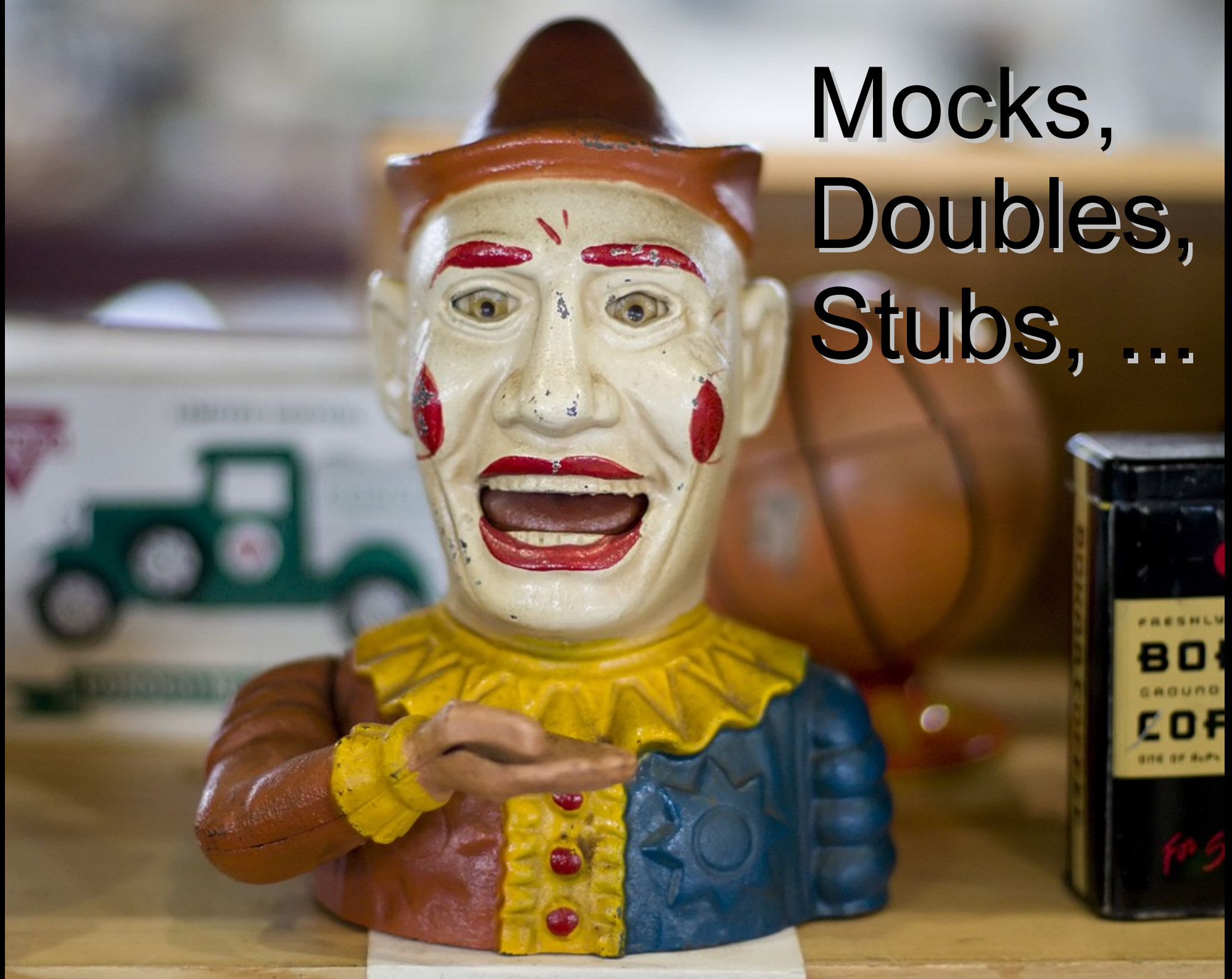
Homework


People
STI Inheritance
Customer
Order
customer_id
OrderItem
order_id
item_id
Item

Single-Table Inheritance

```
class Person < ActiveRecord::Base
end

class Customer < Person
end
```

Derived class inherits associations, named_scope's, validation rules, methods,
...


Mocks,
Doubles,
Stubs, ...

Object level

All three create a “mock” object.

mock(), stub(), double() at the Object level are *synonymous*

Name for error reporting

```
m = mock(“A Mock”)
m = stub(“A Mock”)
m = double(“A Mock”)
```

Using Mocks

RubyFocus

Mocks can have method stubs.

They can be called like methods.

Method stubs can return values.

Mocks can be set up with built-in method stubs.

```
m = mock("A Mock")
m.stub(:foo)
m.foo => nil


m.stub(:foo).
  and_return("hello")
m.foo => "hello"


m = mock("A Mock",
    :foo => "hello")
```

Message Expectations

Mocks can carry message expectations.

should_receive expects a single call by default

Message expectations can return values.

Can expect multiple calls.

```
m = mock("A Mock")

m.should_receive(:foo)

m.should_receive(:foo).
  and_return("hello")

m.should_receive(:foo).
  twice
m.should_receive(:foo).
  exactly(5).times
```

Argument Expectations

Regular expressions

Hash keys

Block

```
m = mock("A Mock")
m.should_receive(:foo).
  with(/ello/)


with(hash_including(
  :name => 'joe'))


with { |arg1, arg2|
  arg1.should == 'abc'
  arg2.should == 2
}
```

Partial Mocks

Replace a method on an existing class.

Add a method to an existing class.

```
jan1 =
  Time.civil(2010)


Time.stub!(:now).
  and_return(jan1)


Time.stub!(:jan1).
  and_return(jan1)
```

Dangers of Mocks

Problems

- Non-DRY
  - Simulated API vs. actual API
- Maintenance
  - Simulated API gets out of sync with actual API
  - Tedious to remove after “outside-in” phase
- Leads to testing implementation, not effect
- Demands on integration and exploratory testing higher with mocks.

Less value per line of test code!

So what are they good for?

- External services
  - API's
- System services
  - Time
  - I/O, Files, ...
- Sufficiently mature (!) internal API's
  - Slow queries
  - Queries with complicated data setup


Controllers
DISCH RELIEF VALVE
OPEN
CLOSED
Waterous
ENGINE COOLER
REEL DISCHARGE
DELUGE DISCHARGE
PASS. DISCHARGE
NO. 4 DISCHARGE
FRONT INLET
OUT-OPEN IN-CLOSED
PANEL LIGHT
NO. 2 CROSSLAY
Pierce
TANK FILL & RE-CIRCULATING LINE
PRIMER
CAUTION
DRIVER REAR DISCHARGE
TRANSFER VALVE
WARNING
THROTTLE
NO. 1 DISCHARGE
PASS. SIDE LARGE DIA. DISCHARGE
NO. 2 DISCHARGE
JOB NO.
GPM AT
RPM
GOVERNED SPEED
PUMP IN
DIRECT
MANUAL PUMP SHIFT
TANK TO PUMP

Controllers

RubyFocus

Controllers are pass-through entities

Mostly boilerplate—biz logic belongs in the model

Controllers are “dumb” or “skinny”

They follow a run-of-the mill pattern:

the Controller Formula

Controller RESTful Actions

- Display methods (“Read”)
  - GET: index, show, new, edit
- Update method
  - PUT
- Create method
  - POST
- Delete method
  - DELETE

REST?

RubyFocus

Representational State Transfer

All resource-based applications & API's need to do similar things, namely:

create, read, update, delete

It's a convention:

- no configuration, no ceremony
- superior to CORBA, SOAP, etc.

RESTful rsources in Rails

RubyFocus

map.resources :people (in config/routes.rb)

people_path, people_url “named route methods”

GET /people → “index” action

POST /people → “create” action

new_person_path, new_person_url

GET /people/new → “new” action

edit_person_path, edit_person_url

GET /people/:id/edit → “edit” action with ID

person_path, person_url

GET /people/:id → “show” action with ID

PUT /people/:id → “update” action with ID

DELETE /people/:id → “destroy” action with ID

Reads Test Pattern

- Make request (with id of record if a single record)
- Check Rendering
  - correct template
  - redirect
  - status code
  - content type (HTML, JSON, XML,...)
- Verify Variable Assignments
  - required by view

Read Formula

Find data, based on parameters

Assign variables

Render

How much test is too much?

- Test anything where the code deviates from defaults, e.g. redirect vs. straight up render
- These tests are not strictly necessary:
  - response.should be_success
  - response.should render_template('new')
- Test anything required for the application to proceed without error
  - Speficially variable assignments
- Do test error handling code!

form_for's automagic powers

form_for @person do |f| ... end

When @person is new

→ <form action="people" method="post">

→ PeopleController#create

→ uses people_path method to generate URL

When @person exists already

→ <form action="people" method="put">

→ PeopleController#update

→ uses person_path method to generate URL

Create/Update Test Pattern

RubyFocus

- Make request with form fields to be created/upd'd
- Verify Variable Assignments
- Verify Check Success
  - Rendering
- Verify Failure/Error Case
  - Rendering
  - Variables
- Verify HTTP Verb protection

Create/Update Formula

Update: Find record from parameters

Create: Instantiate new model object

Assign form fields parameters to model object

- This should be a single line
- It is a pattern, the “Controller Formula”

Save

Handle success—typically a redirect

Handle failure—typically a render

Destroy Test Pattern

Make request with id of record to be destroyed

Rendering

Typically no need to check for success or failure

- Invalid item referenced → Same behavior as read
- Database deletion failed → System not user error

Destroy Formula

RubyFocus

Find record from parameters

Destroy

Redirect

How much is enough?

Notice: No view testing so far.

Emphasize behavior over display.

Check that the application handles *errors* correctly

Test views only for things that could go wrong badly

- incorrect form URL
- incorrect names on complicated forms, because they impact parameter representation

View Testing

RubyFocus

- RSpec controllers do *not* render views (by default)
- Test form urls, any logic and input names
- Understand CSS selector syntax
- View test requires set up of variables
  - another reason why there should only be very few variables between controller and view

A note on RJS

RJS lets you render a JS response using “RJS”

Built on Prototype JS framework

Hard to test

Largely superseded by

- RSpec tested controllers responding in JSON
- JS tests mocking the server response
- jQuery's Ajax library very helpful

RSpec Scaffold & Mocks

RSpec Scaffold mocks models

Their philosophy is outside-in development

I'm not a fan of this because:

- It causes gaps in coverage
- Mocks need to be maintained
- Outside-in remains unproven beyond textbook examples
- When taken to the extreme, it can become un-Agile by over-specifying the application


Nested
Attributes

One Form—Multiple Models

DB schema should not limit view/form layout

View/form layout should not impose DB schema

A single form to manipulate associated rows of multiple tables.

→ Nested Attributes

View & Model Collaboration

RubyFocus

Model:

accepts_nested_attributes_for

View:

form_for

fields_for

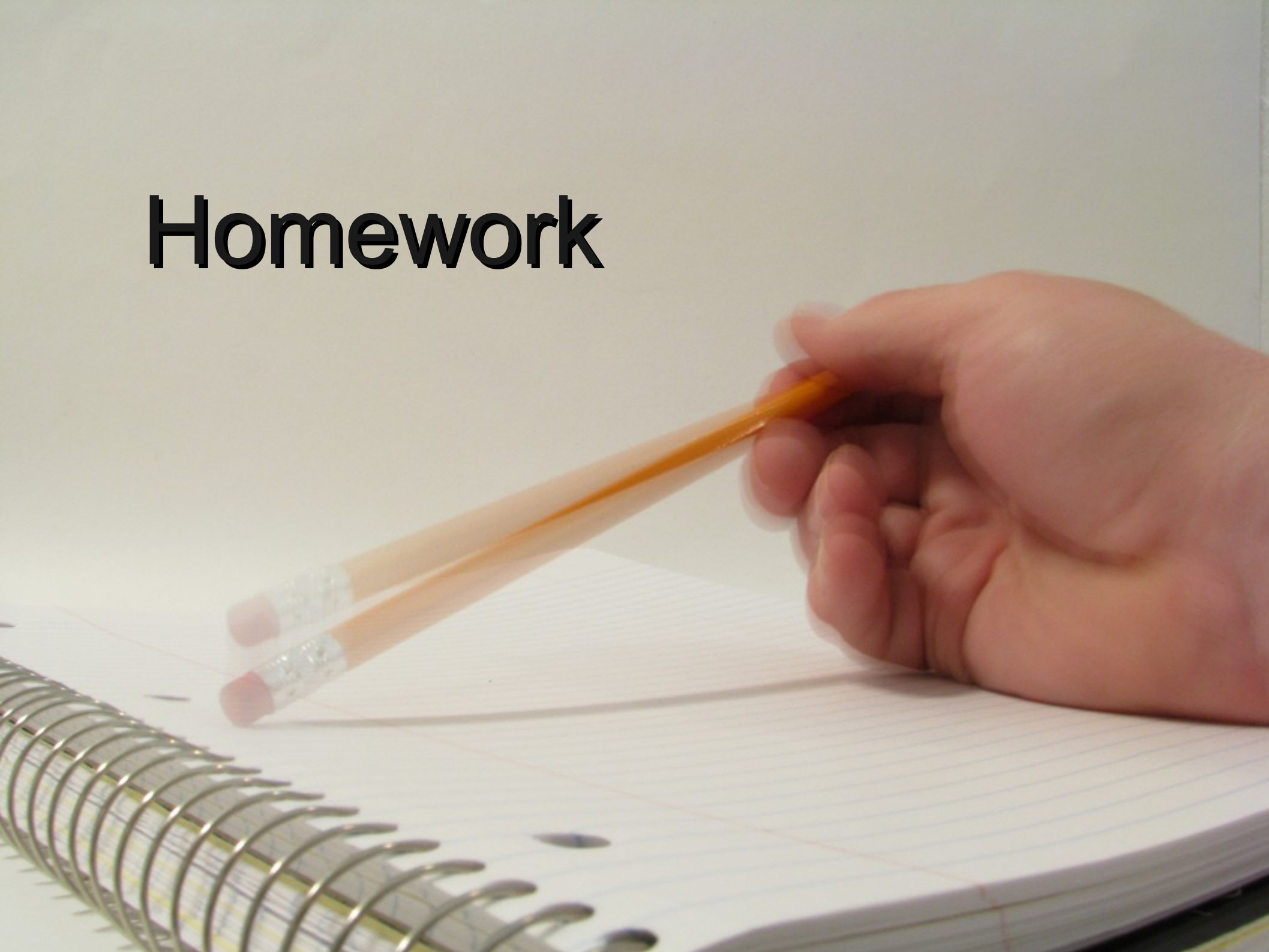
Controller is unaware! Strictly pass-through

Person.new(params[:person])

accepts_nested_attributes_for

```
class Person < ActiveRecord::Base
   has_many :addresses
   accepts_nested_attributes_for :person
end


Person.create(:first_name => “Joe”,
              :last_name  => “Smith”,
              :addresses_attributes => [ {
                :street => “123 Main,
                :city   => “San Francisco”,
                :zip    => “94103”,
                :state  => “CA” } ]
```

Homework



Homework

As a person, I can enter or update my name along with 2 addresses on the same form.

Extra credit:

As a customer, I can order multiple items from a list of items.

On the order form, I can click “add one” to add an item.

On the order form, I can click “remove” next to each item already on the order.

Recommended Reading

RubyFocus

Nested Attributes API Docs: http://bit.ly/7cnt0

Form for (with nest'd attr's) API Docs:
http://bit.ly/9DAscq

My presentation on nested attributes:
http://blip.tv/file/3957941

RSpec book chapter 14 (on Mocks)

RSpec book chapters 1.5, 10, 11 (on BDD)

RSpec book chapter 20 (outside-in development)

Flickr Photo Attribute Credit

Matroshka nested dolls
http://www.flickr.com/photos/shereen84/2511071028/sizes/l/

Notebook with pencil
http://www.flickr.com/photos/tomsaint/2987926396/sizes/l/

Danger Sign
http://www.flickr.com/photos/lwr/132854217/sizes/l/

Control Panel
http://www.flickr.com/photos/900hp/3961052527/sizes/l/

Mocking Clown
http://www.flickr.com/photos/bcostin/134531379/sizes/l/