Web Engineering:
Task: Cart creation

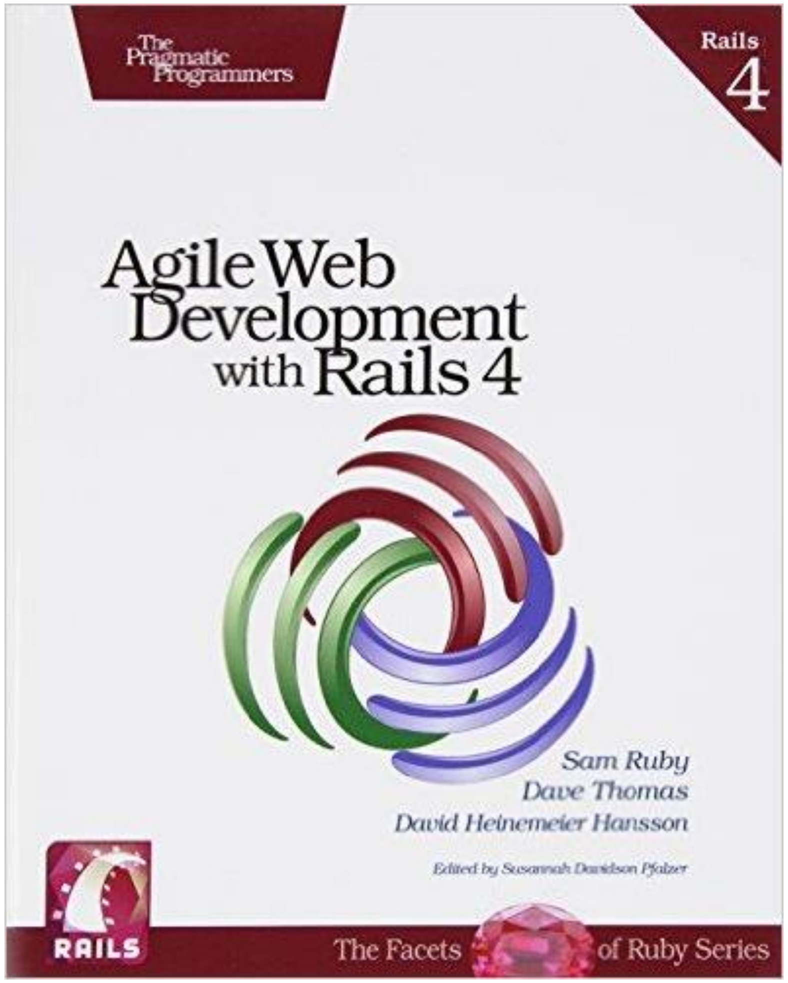
The University of Aizu
Quarter 2, AY 2018

Outline

- Finding a cart
- Connection products to carts
- Adding a button

Literature

- Agile Web Development with Rails 4 (1st edition) by Sam Ruby, Dave Thomas and Devid Hansson, The Pragmatic Bookshelf, 2013.
  - Chapter 9.


The Pragmatic Programmers
Rails 4
Agile Web Development with Rails 4
Sam Ruby
Dave Thomas
David Heinemeier Hansson
Edited by Susannah Davidson Pfalzer
RAILS
The Facets of Ruby Series

Finding a cart

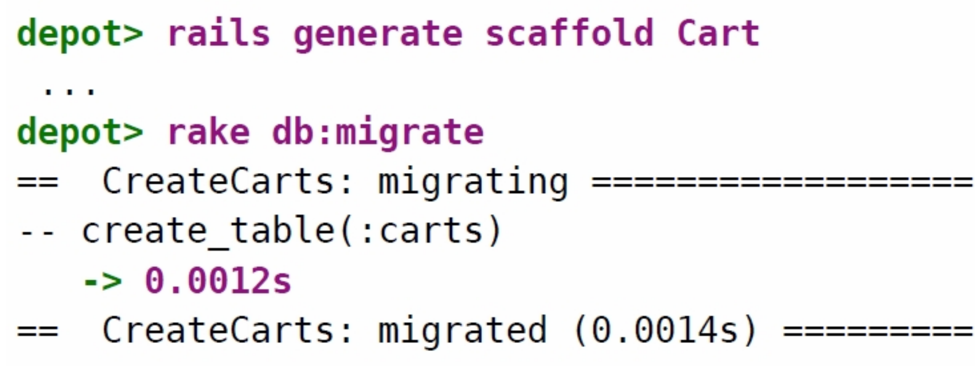
- Now, our application has the ability to display a catalog containing all our products.
- We would like to add the selling functionality.
- The convention is that each item selected will be added to a virtual shopping cart.
- When buyers are finished selecting goods, they will proceed to our site's checkout, where they'll pay for the stuff in the carts.

Finding a cart

- Our application will need to keep track of all the items added to the cart by buyers.
- To implement this, we will keep a cart in the database and store its unique identifier, *cart.id*, in the session.
- Every time, a request comes in, we can recover the identity from the session and use it to find the cart in the database.

Finding a cart

- To create a cart, we need to do the following:



```
depot> rails generate scaffold Cart
 ...
depot> rake db:migrate
==  CreateCarts: migrating ==================
-- create_table(:carts)
   -> 0.0012s
==  CreateCarts: migrated (0.0014s) =========
```

- Rails makes the current sessions look like a hash to the controller.
  - We have to store the *id* of the cart in the session by indexing it with the symbol *:cart_id* (see the next slide)

Finding a cart

Download rails40/depot_f/app/controllers/concerns/current_cart.rb

```ruby
module CurrentCart
  extend ActiveSupport::Concern

  private

    def set_cart
      @cart = Cart.find(session[:cart_id])
    rescue ActiveRecord::RecordNotFound
      @cart = Cart.create
      session[:cart_id] = @cart.id
    end
end
```

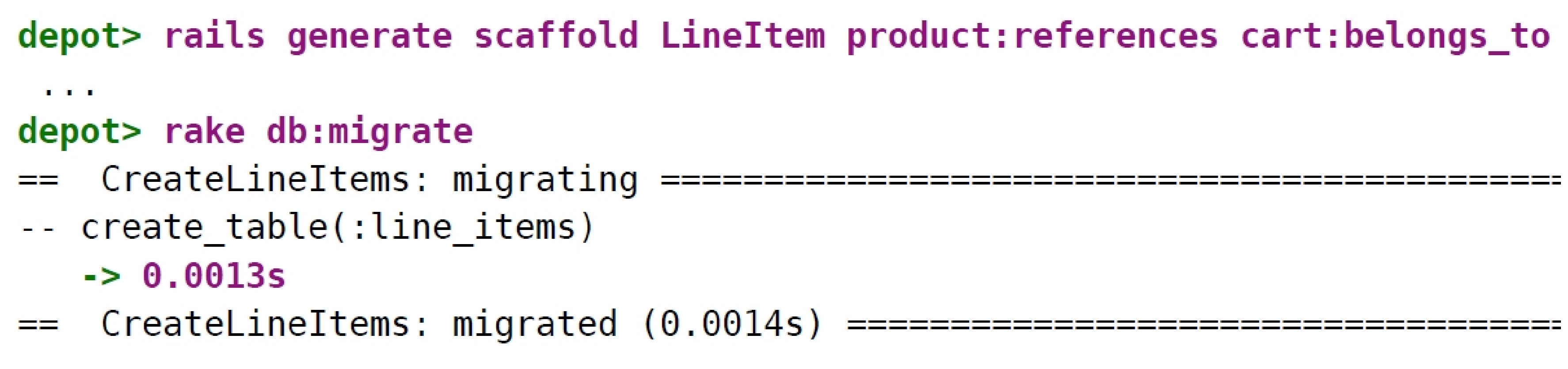
- The *set_cart* starts by *getting* the *:cart_id* from the session object and then attempts to find a cart corrersponding to this *id*.

Comments on the previous slide

- If such a cart record is not found, then this method will proceed to create a new *Cart*, store the *id* of the created cart into the session, and then return the new cart.
- Note, that we place the *set_cart* method in the *CurrentCart module* and mark it as private. This allows us to share common code between controllers and furthermore prevents Rails from making it available as an action on the controller.

Connecting products to carts

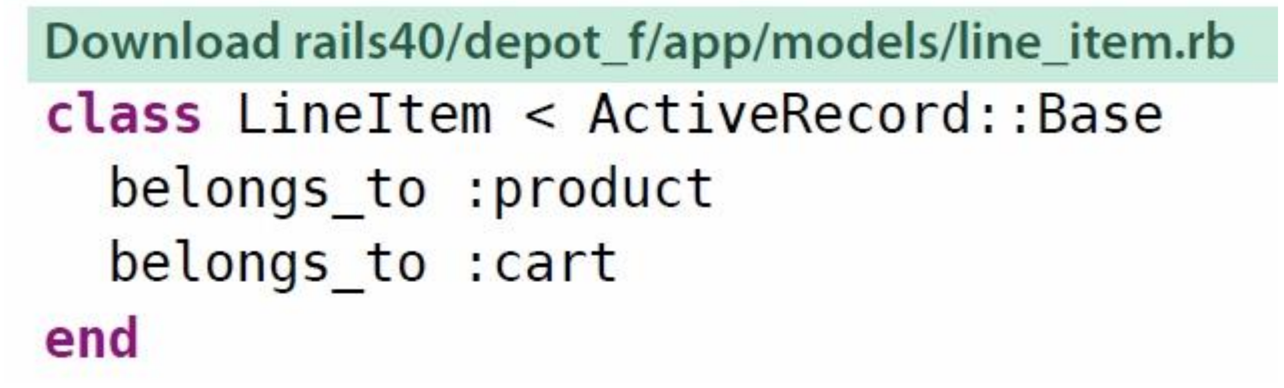
- A cart contains a set of products.
- We will generate the Rails models and populate the migrations to create the corresponding tables.



```
depot> rails generate scaffold LineItem product:references cart:belongs_to
 ...
depot> rake db:migrate
==  CreateLineItems: migrating ===========================================
-- create_table(:line_items)
   -> 0.0013s
==  CreateLineItems: migrated (0.0014s) ==================================
```

- The database has a place to store the references between line items, carts, and products.

Connecting products to carts



Download rails40/depot_f/app/models/line_item.rb

```ruby
class LineItem < ActiveRecord::Base
  belongs_to :product
  belongs_to :cart
end
```

- A generated definition of the *LineItem* class includes the definitions of these relationships.
- This class specifyes links in the opposite direction, from the line item to the carts and products tables.

Connecting products to carts

- At the model level, there is no differences between a simple reference and "belong to" relationship.
- *belons_to* tells Rails that rows in the *line_items* table are children rows in *carts* and *products* tables.
  - No line item can exist unless the corresponding cart and product rows exist.

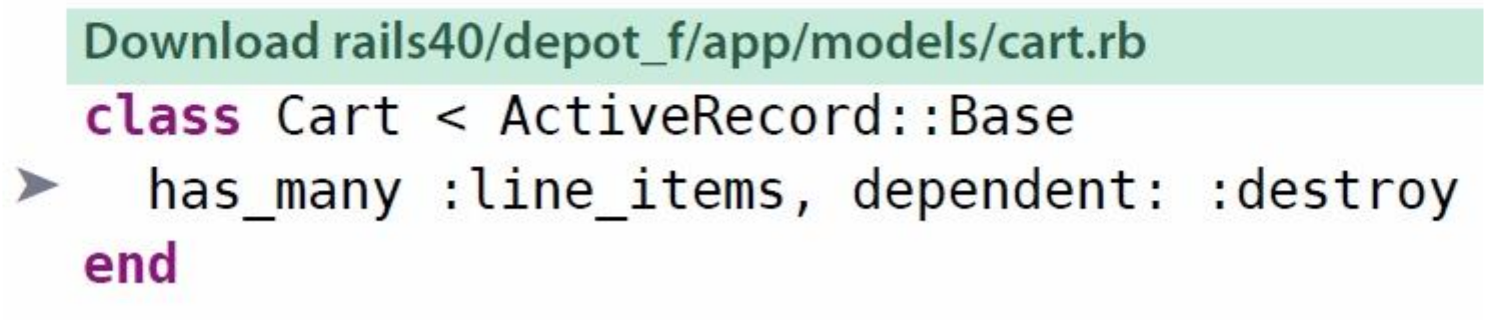
Connecting products to carts

- An easy way to remember where to put *belong_to* declarations:
  - If a table has foreign keys, the corresponding model should have a *belong_to* for each.
- These declaration add navigation capabilities to the model objects.
  - Now, we can retrieve its Product and display the book's title:

```
li = LineItem.find(...)
puts "This line item is for #{li.product.title}"
```

Connecting products to carts

- We need to add some declarations to our model files that specify their inverse relations.



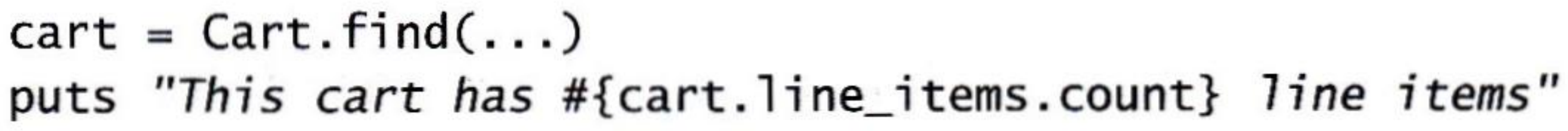
Download rails40/depot_f/app/models/cart.rb

```
class Cart < ActiveRecord::Base
➤   has_many :line_items, dependent: :destroy
end
```

- The part *has_many :line_items* , says that a cart has many associated line items. These are linked to the cart because each line item contains a reference to its cart's *id*.
- The part *dependent : :destroy* indicates that the existence of the line items is dependent on the existence of the cart.

Connecting products to carts

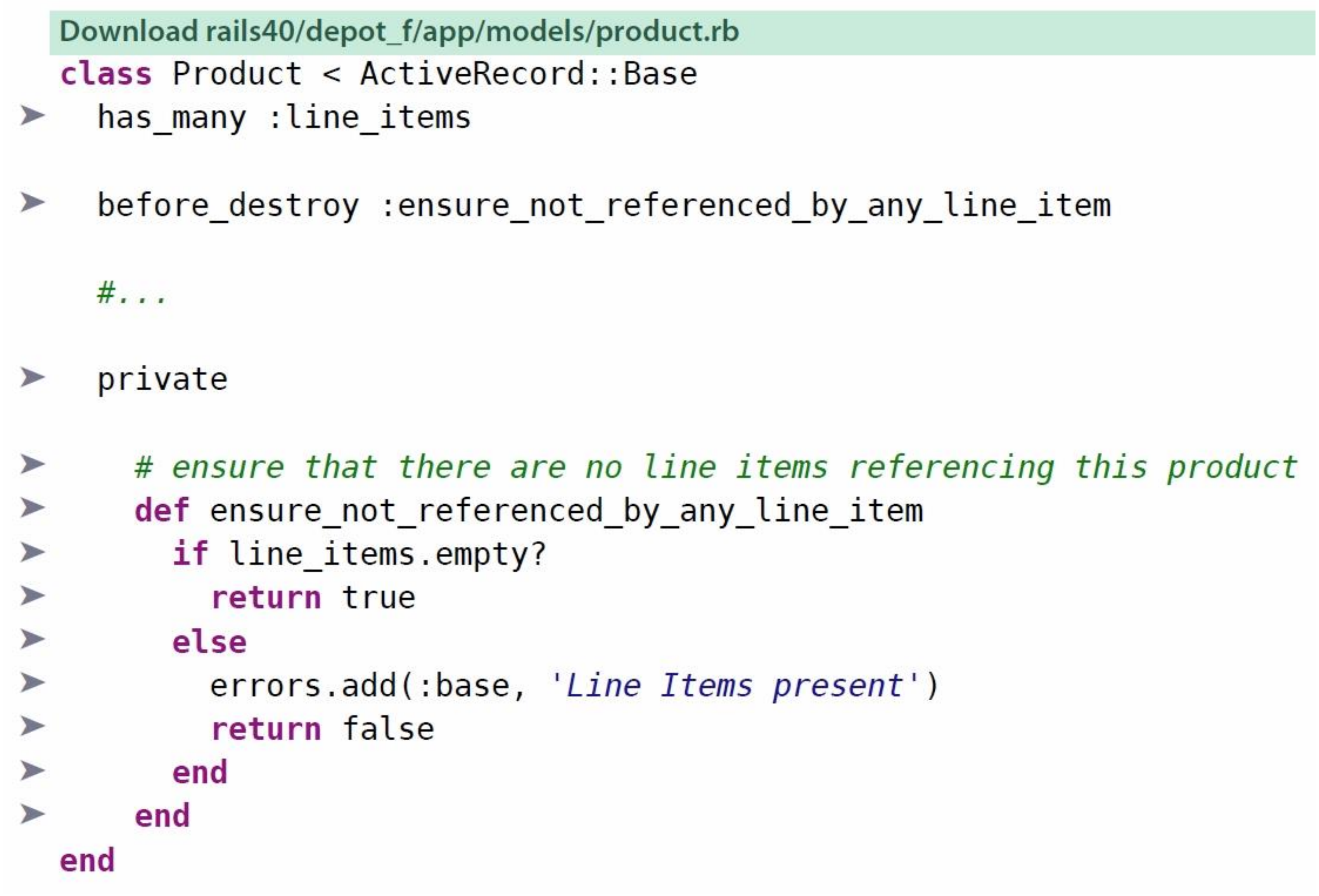
- Cart is declared to have many line items, so we can reference them (as a collection) from a cart object:



```
cart = Cart.find(...)
puts "This cart has #{cart.line_items.count} line items"
```

- For completeness, we should add *has_many* directive to our Product model.
- If we have lots of carts, each prodcut might have many line items referencing it.
- This time, we will make use of validation code to prevent removal of products that referenced by line items (see the next slide).

Connecting products to carts



Download rails40/depot_f/app/models/product.rb

```
  class Product < ActiveRecord::Base
➤   has_many :line_items

➤   before_destroy :ensure_not_referenced_by_any_line_item

    #...

➤   private

➤     # ensure that there are no line items referencing this product
➤     def ensure_not_referenced_by_any_line_item
➤       if line_items.empty?
➤         return true
➤       else
➤         errors.add(:base, 'Line Items present')
➤         return false
➤       end
➤     end
  end
```

Comments on the previous slide

- We declared that a product has many line items and define a *hook* method named *ensure_not_referenced_by_any_line_item.*
- A *hook* method is a method that Rails calls automatically at a given point in an object's life.
- The method will be called before Rails attempts to destroy a row in the datadabe.
- If the *hook* method returns false, the row will not be destroyed.

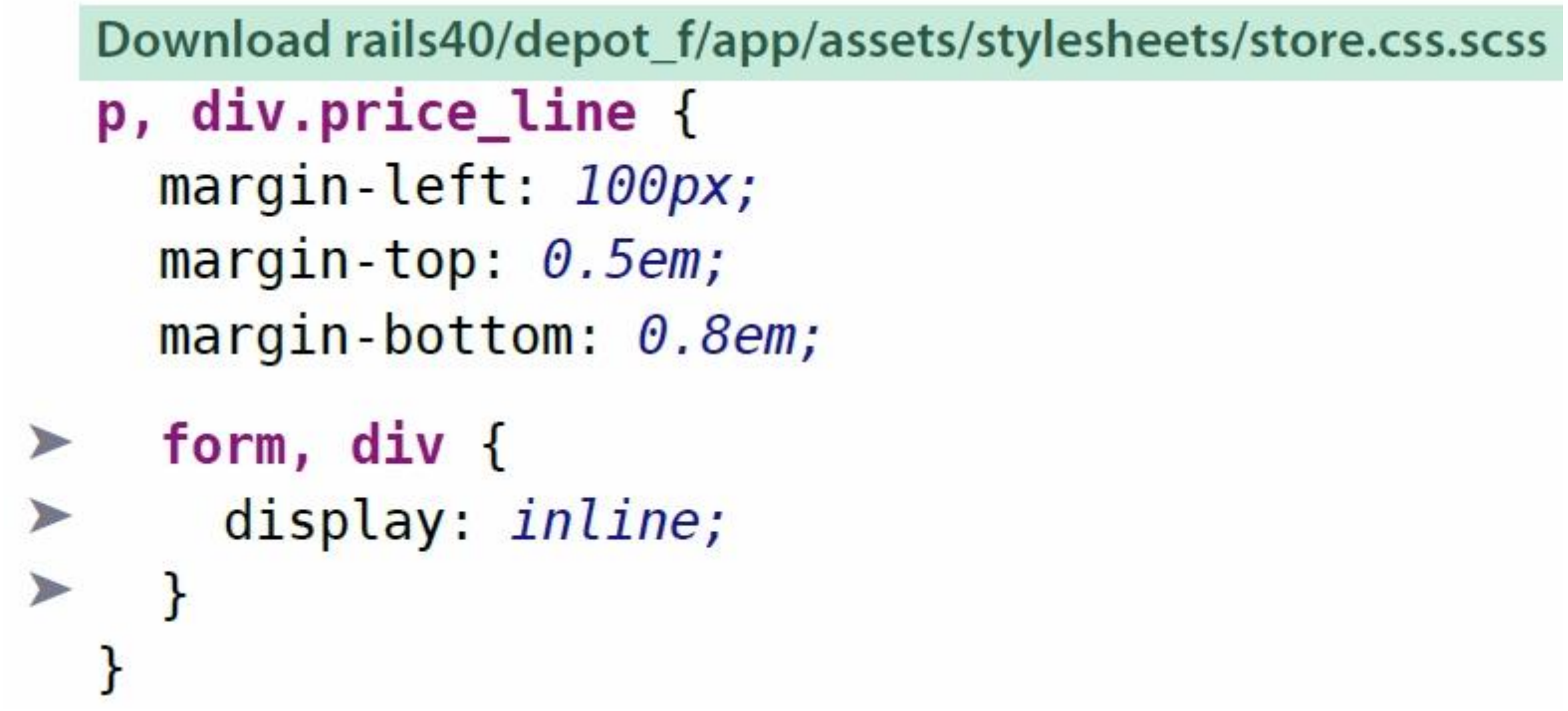
Adding a button

- It is time to add <u>*Add to Cart*</u> button to each product.

Download rails40/depot_f/app/views/store/index.html.erb

```
<% if notice %>
<p id="notice"><%= notice %></p>
<% end %>

<h1>Your Pragmatic Catalog</h1>

<% cache ['store', Product.latest] do %>
  <% @products.each do |product| %>
    <% cache ['entry', product] do %>
      <div class="entry">
        <%= image_tag(product.image_url) %>
        <h3><%= product.title %></h3>
        <%= sanitize(product.description) %>
        <div class="price_line">
          <span class="price"><%= number_to_currency(product.price) %></span>
          <%= button_to 'Add to Cart', line_items_path(product_id: product) %>
        </div>
      </div>
    <% end %>
  <% end %>
<% end %>
```

Adding a button



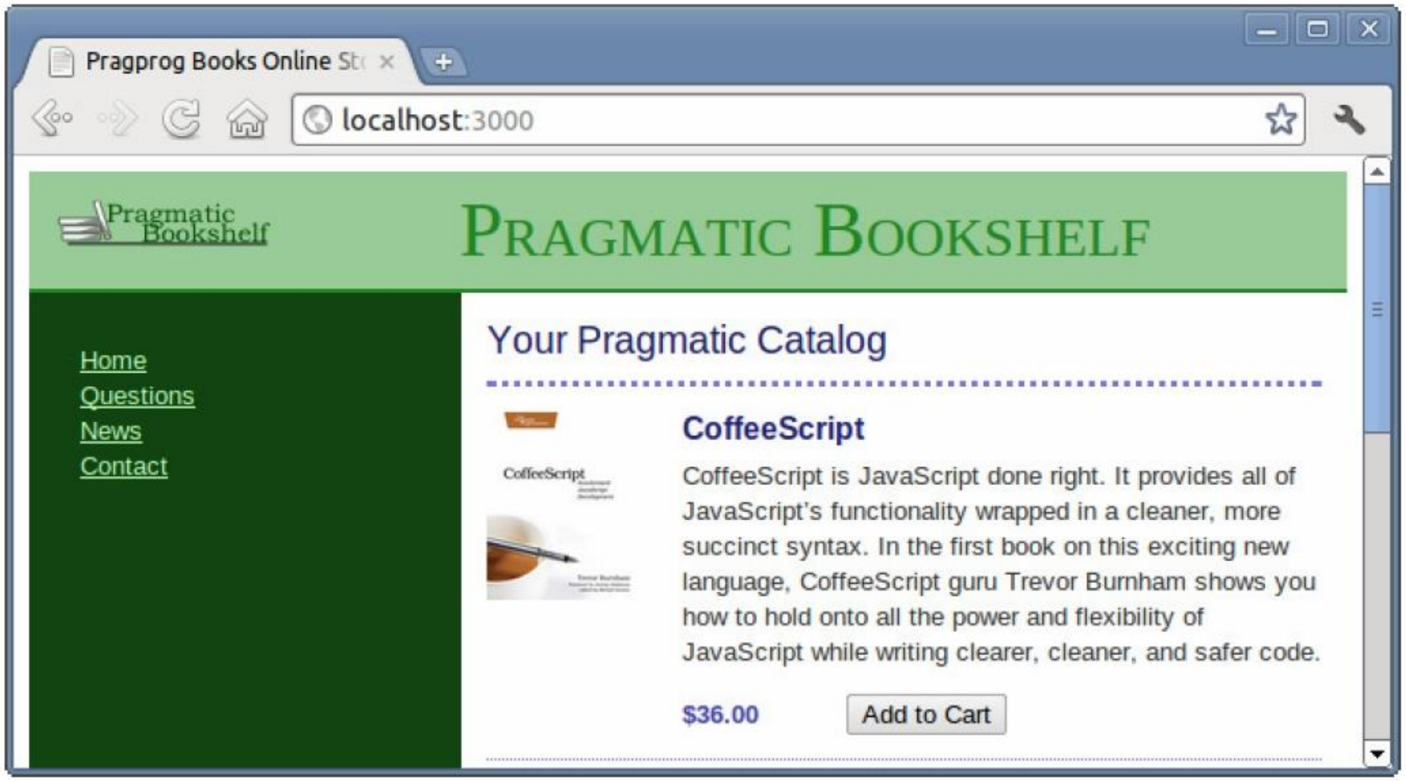
Download rails40/depot_f/app/assets/stylesheets/store.css.scss

```
p, div.price_line {
  margin-left: 100px;
  margin-top: 0.5em;
  margin-bottom: 0.8em;

➤   form, div {
➤     display: inline;
➤   }
}
```

- This is one formatting issue using CSS:
  - *bottom_to* creates an HTML form. That form contains an HTML div. Both of these normally block elements which appear on the next line. To place them next to the price , we need this CSS

Adding a button

- The final result is here:



Pragprog Books Online St

localhost:3000

Pragmatic Bookshelf

PRAGMATIC BOOKSHELF

Home
Questions
News
Contact

Your Pragmatic Catalog

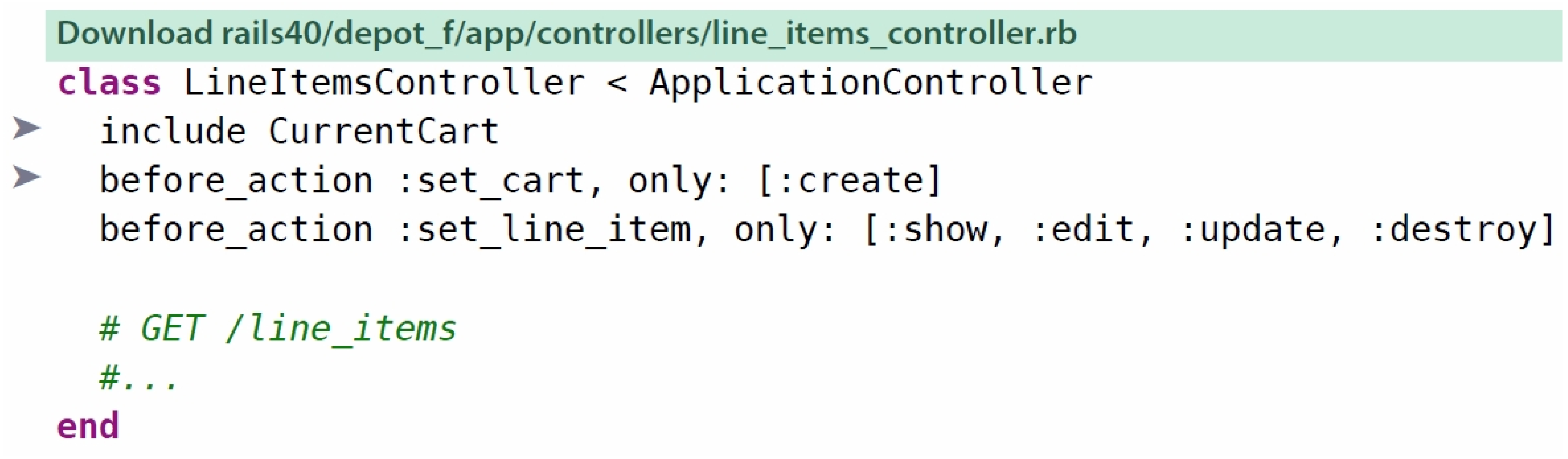
CoffeeScript

CoffeeScript is JavaScript done right. It provides all of JavaScript's functionality wrapped in a cleaner, more succinct syntax. In the first book on this exciting new language, CoffeeScript guru Trevor Burnham shows you how to hold onto all the power and flexibility of JavaScript while writing clearer, cleaner, and safer code.

$36.00 Add to Cart

Creating a cart

- We will modify the *LineItemsController* to find the shopping cart for the current session, add the selected product to the that cart and display the cart contends.
- We will use *CurrentCart* (see slide 7) to find or create a cart in the session.



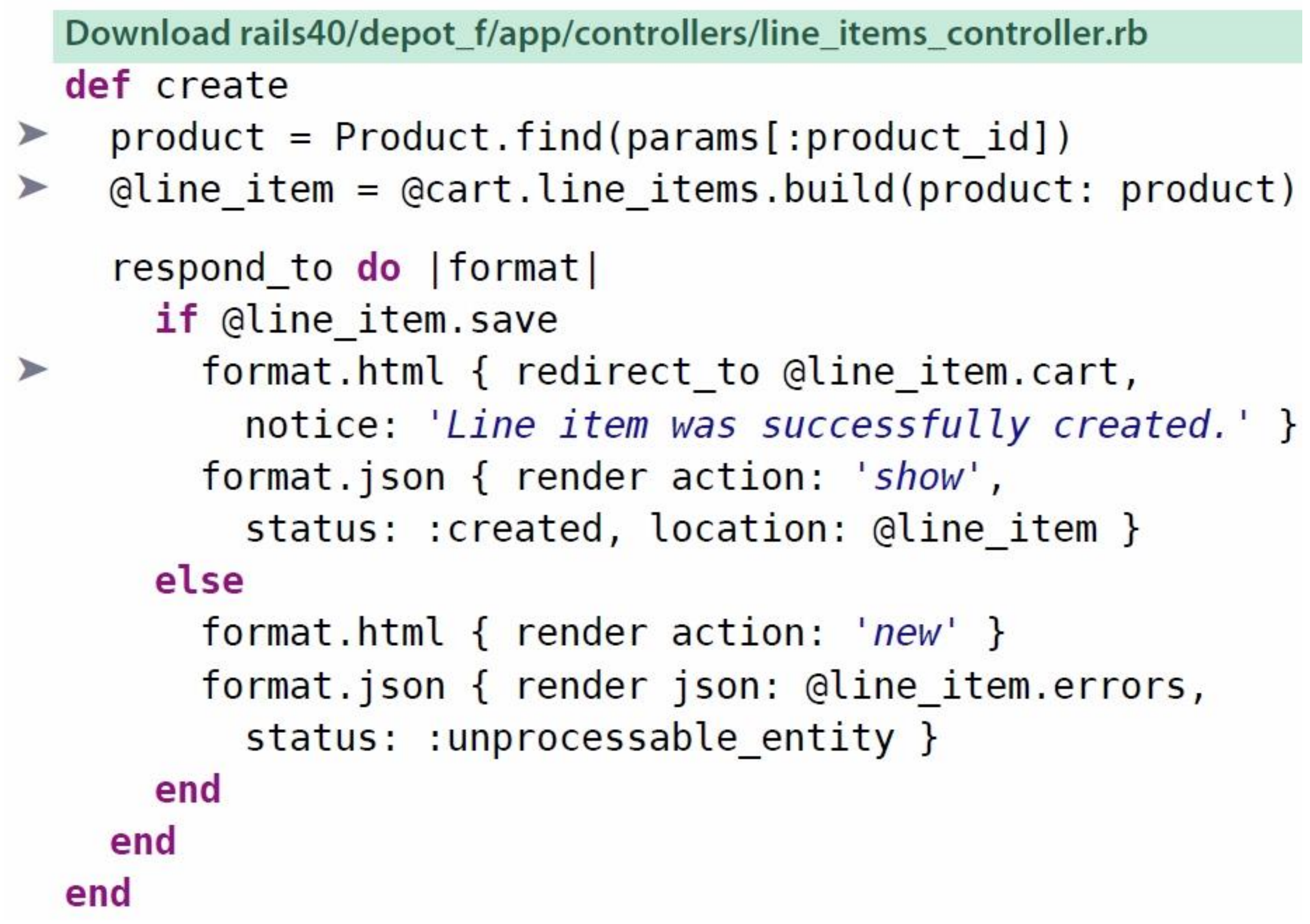
Download rails40/depot_f/app/controllers/line_items_controller.rb

```
  class LineItemsController < ApplicationController
➤   include CurrentCart
➤   before_action :set_cart, only: [:create]
    before_action :set_line_item, only: [:show, :edit, :update, :destroy]

    # GET /line_items
    #...
  end
```

Creating a cart

- We need to modify a few lines of code in the *create* method in *app/controllers/line_items_controller.rb*
- See the next slide

The create method



Download rails40/depot_f/app/controllers/line_items_controller.rb

```
  def create
➤   product = Product.find(params[:product_id])
➤   @line_item = @cart.line_items.build(product: product)

    respond_to do |format|
      if @line_item.save
➤       format.html { redirect_to @line_item.cart,
          notice: 'Line item was successfully created.' }
        format.json { render action: 'show',
          status: :created, location: @line_item }
      else
        format.html { render action: 'new' }
        format.json { render json: @line_item.errors,
          status: :unprocessable_entity }
      end
    end
  end
```

Comments on the previous slide

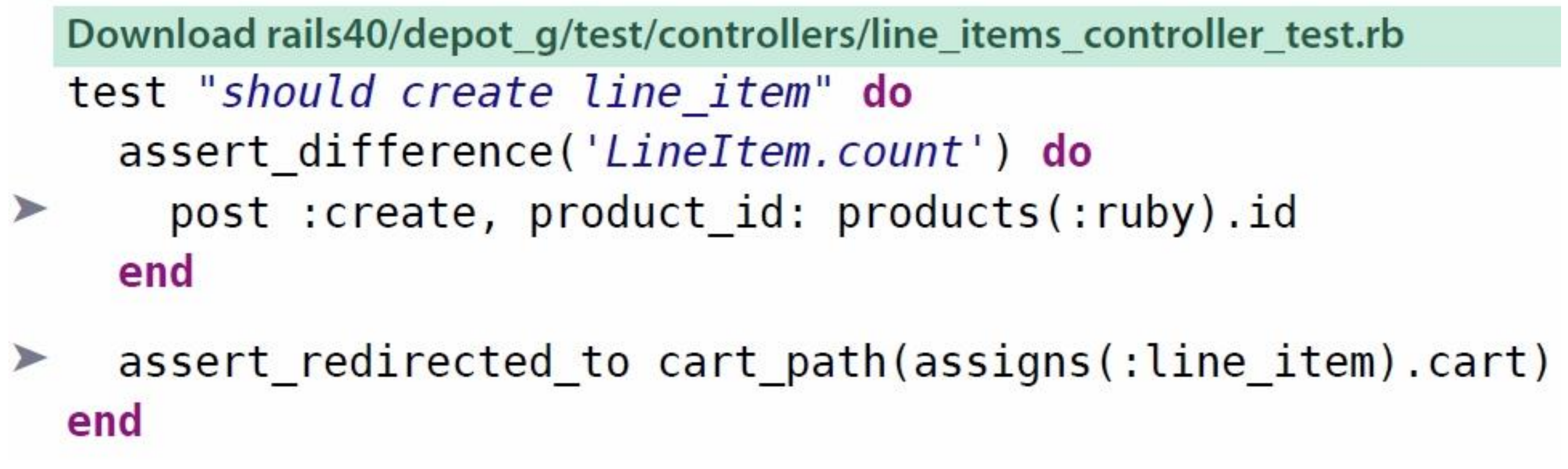
- We use the *params* object to get the *:prodect_id* parameter from the request.
  - The *params* object is important inside Rails applications: It holds all of parameters passed in a browser request.
  - We store the result in a local variable because there is no need to make this available to the view.
- We then pass that product we found into *@cart.line_items.build.*
  - This causes a new line item relationship to be build between *@cart* object and the *product.*

Comments on the previous slide

- We save the resulting line item into instance variable named @*line_item*.
- The reminder of this method takes care of handling errors.
- Now, we need to modify one more thing:
  - Once the line item is created, we want to redirect you to the cart instead of back to the line item itself.
  - Since the line item object knows how to find the cart object, all we need to do is add .*cart* to the method call.

Creating a cart

- As we changed the function of our controller, we know that we will need to update the corresponding functional test.
- We need to pass a product *id* on the call to create and change what we expect for the target of the redirect.
- We do this by updating the following file:



Download rails40/depot_g/test/controllers/line_items_controller_test.rb

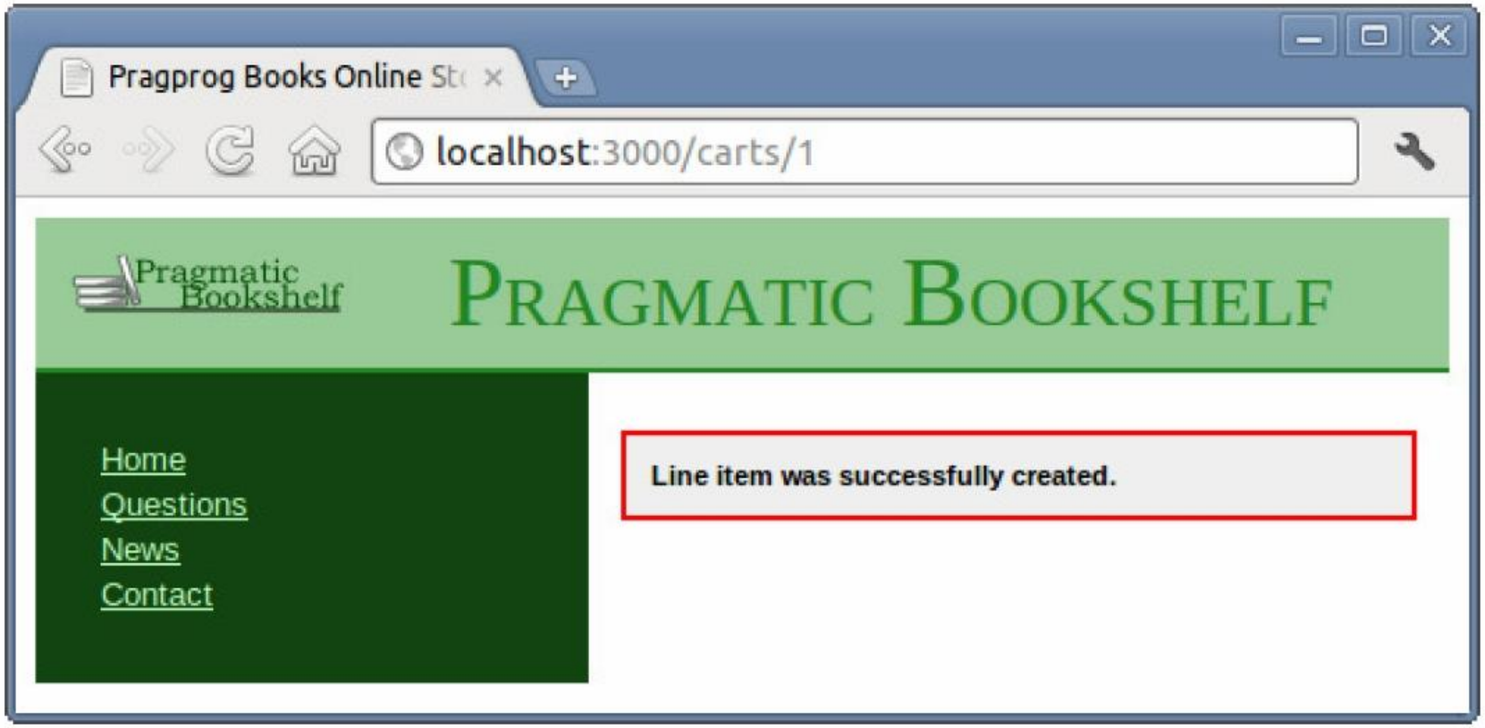
```
  test "should create line_item" do
    assert_difference('LineItem.count') do
➤     post :create, product_id: products(:ruby).id
    end

➤   assert_redirected_to cart_path(assigns(:line_item).cart)
  end
```

Creating a cart

- We now rerun the functional tests:

depot> rake test test/controllers/line_items_controller_test.rb

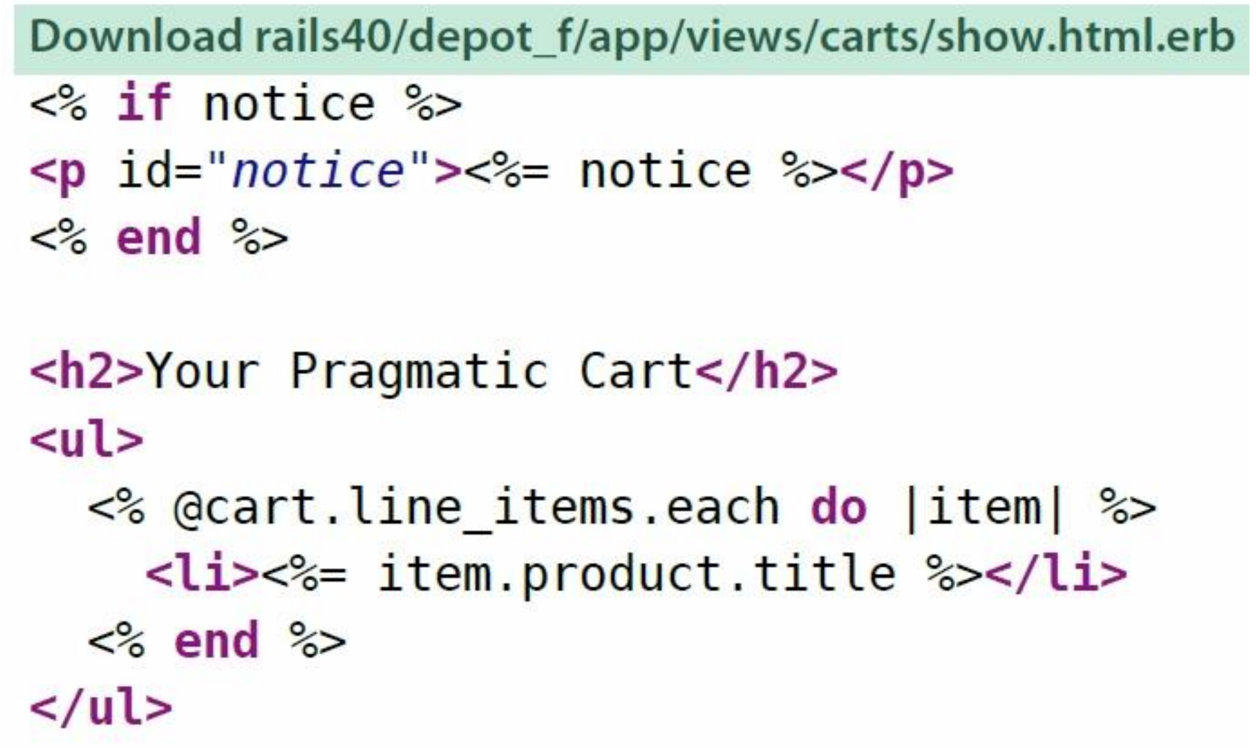
- All things are working fine and we can try *Add to Cart* button in our browser.



Line item was successfully created.

Creating a cart

- (See the previous slide) We did not provide any attributes, so the view does not have anything to show.
- We prepare the trivial template:

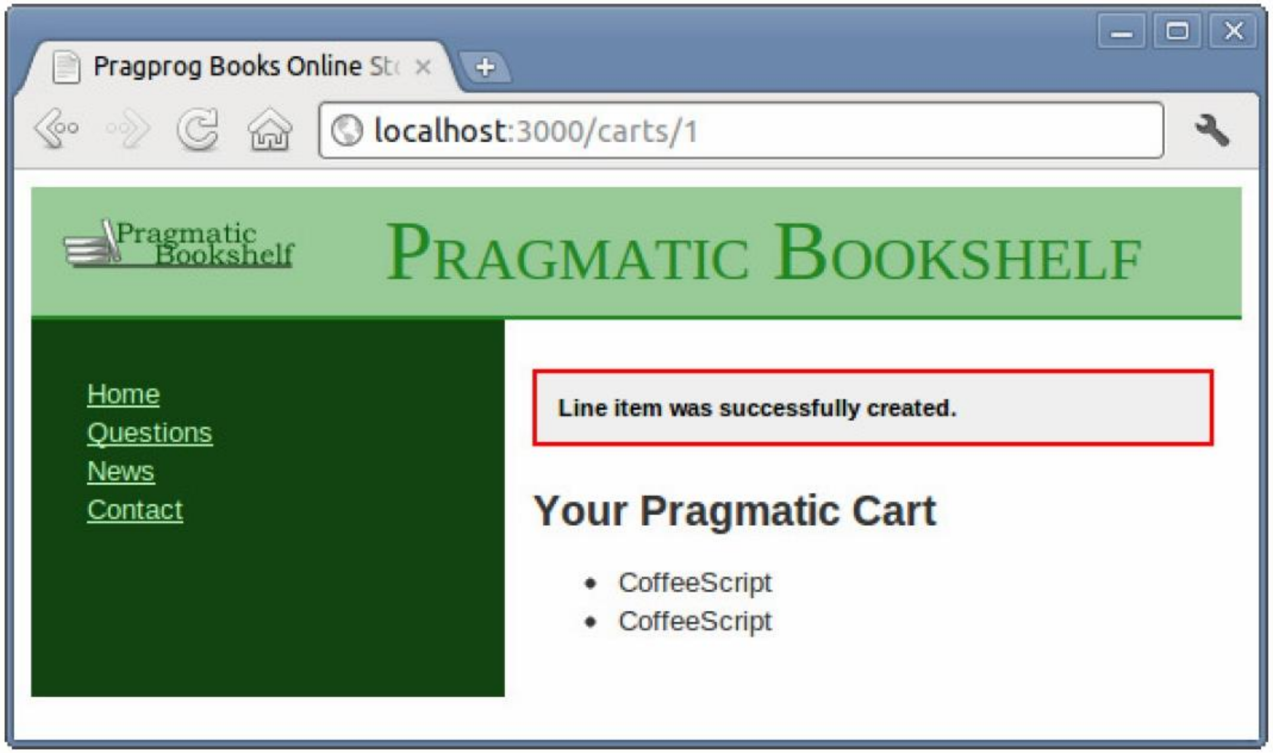


Download rails40/depot_f/app/views/carts/show.html.erb

```
<% if notice %>
<p id="notice"><%= notice %></p>
<% end %>

<h2>Your Pragmatic Cart</h2>
<ul>
  <% @cart.line_items.each do |item| %>
    <li><%= item.product.title %></li>
  <% end %>
</ul>
```

Creating a cart

- After reloading the page, we will see:



Line item was successfully created.

Your Pragmatic Cart

- CoffeeScript
- CoffeeScript

- Real shopping carts do not show the separate lines for the same product, prodcut line once with a quantity of 2 (in our case).
- We need to improve it!

What we just did

- We created a *Cart* object and were able to successfully locate the same cart in subsequent requests using a session object.
- We added a private method in the base class for all of our controllers making it accessible to all of our controllers.
- We created relationships between carts and line items and relationships between line items and products, and we were able to navigate using these relationships.
- We added a button that caused a product to be posted to a cart, causing a new line item to be created.