chalk ENRICHED WEB DESIGN & DEVELOPMENT FOR THE MASSES FOR HIRE!

– Jason Hummel Partner with Chalk

BIG IDEAS FOR SMALL PROJECTS

- Borrowing ideas from frameworks that can be used in small projects to keep your code cleaner

FRAMEWORKS

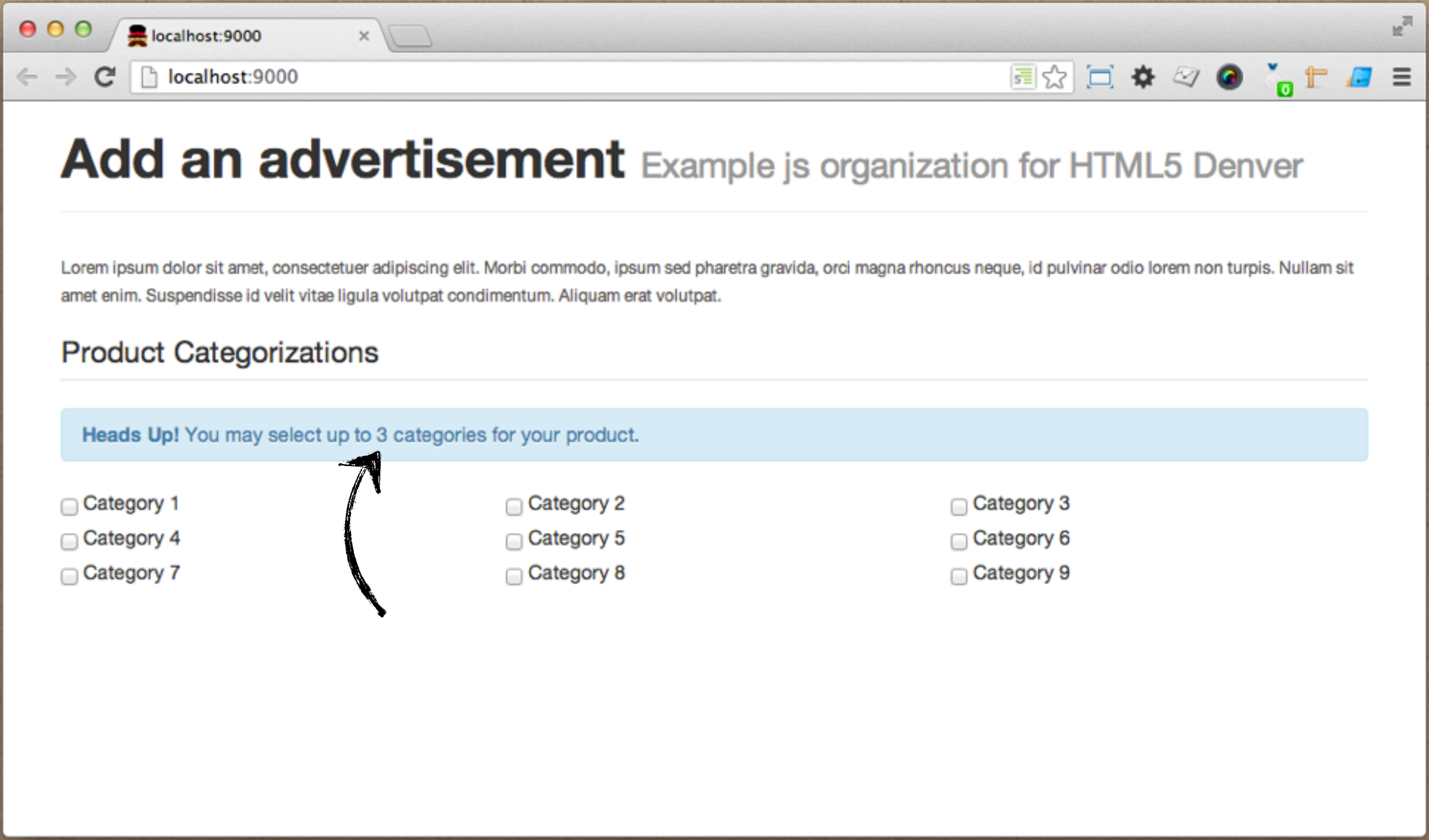
- Let us write highly maintainable code
- Modularize our codebase, separating our concerns
- Gives us structure to our programs
- Allows ease of testability of our code

- First, why do we use frameworks

WHY ABANDON THESE PRINCIPLES ON SMALL JOBS?

chalk

- Small jobs - adding behavior to an existing form for example
- Worked with teams designing huge complex systems using disciplined js techniques.
- A contact form is a mess, WHY?
    - Don't have the time, bigger fish to fry
    - Don't want to download big frameworks for simple interaction
- Doesn't have to be complicated, or use external libraries
- Note on jQuery, using it here because it's so common, and have a couple tricks. This can all be done with plain js however.



localhost:9000

Add an advertisement Example js organization for HTML5 Denver

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat.

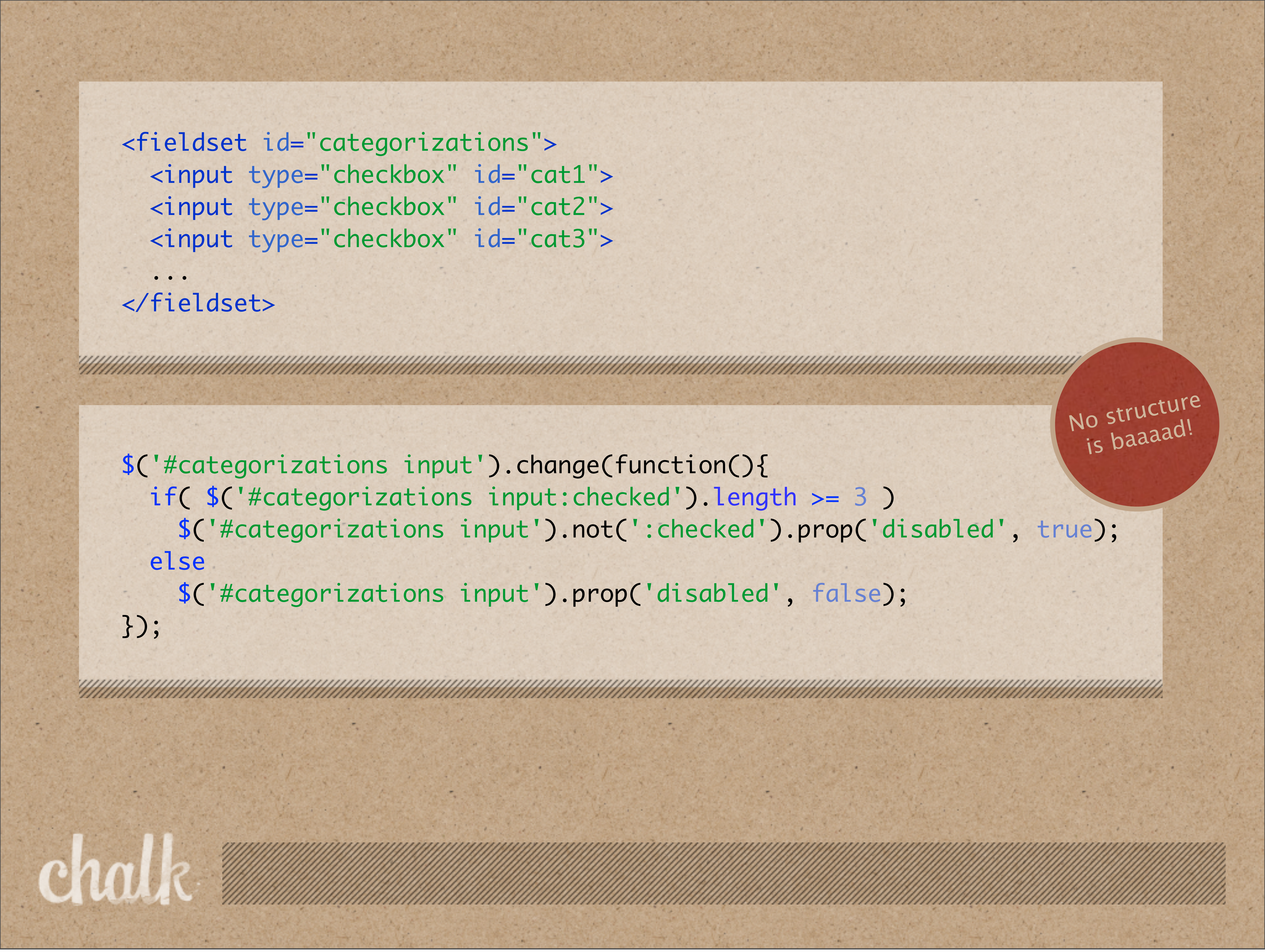
Product Categorizations

Heads Up! You may select up to 3 categories for your product.

☐ Category 1	☐ Category 2	☐ Category 3
☐ Category 4	☐ Category 5	☐ Category 6
☐ Category 7	☐ Category 8	☐ Category 9

chalk

- Taken from recent project, allows companies to post advertisements in a magazine targeting a vertical market, like a focused yellow pags
- Can list a product in up to three categories per issue.

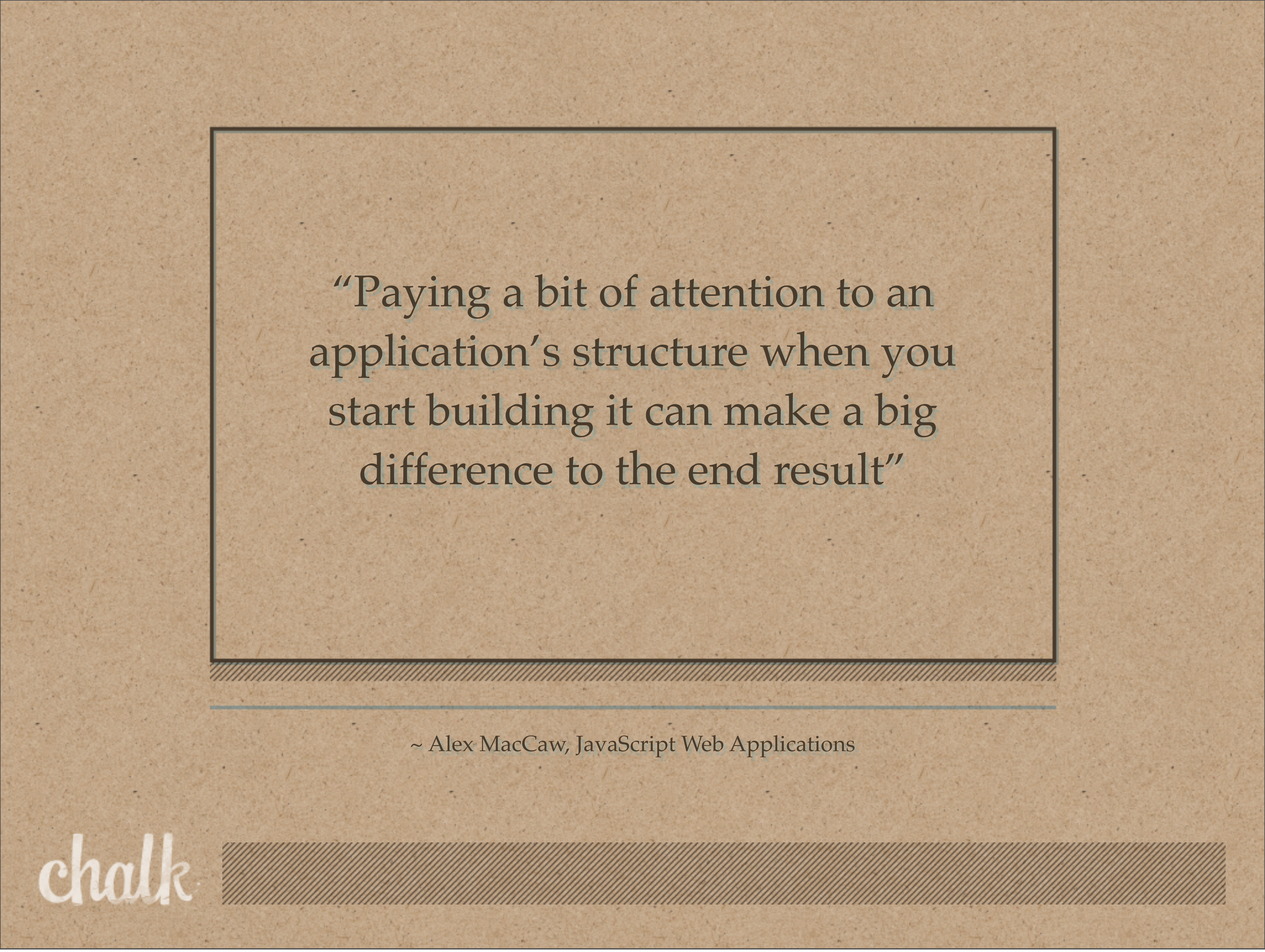


```html
<fieldset id="categorizations">
  <input type="checkbox" id="cat1">
  <input type="checkbox" id="cat2">
  <input type="checkbox" id="cat3">
  ...
</fieldset>
```

```javascript
$('#categorizations input').change(function(){
  if( $('#categorizations input:checked').length >= 3 )
    $('#categorizations input').not(':checked').prop('disabled', true);
  else
    $('#categorizations input').prop('disabled', false);
});
```

No structure is baaaad!

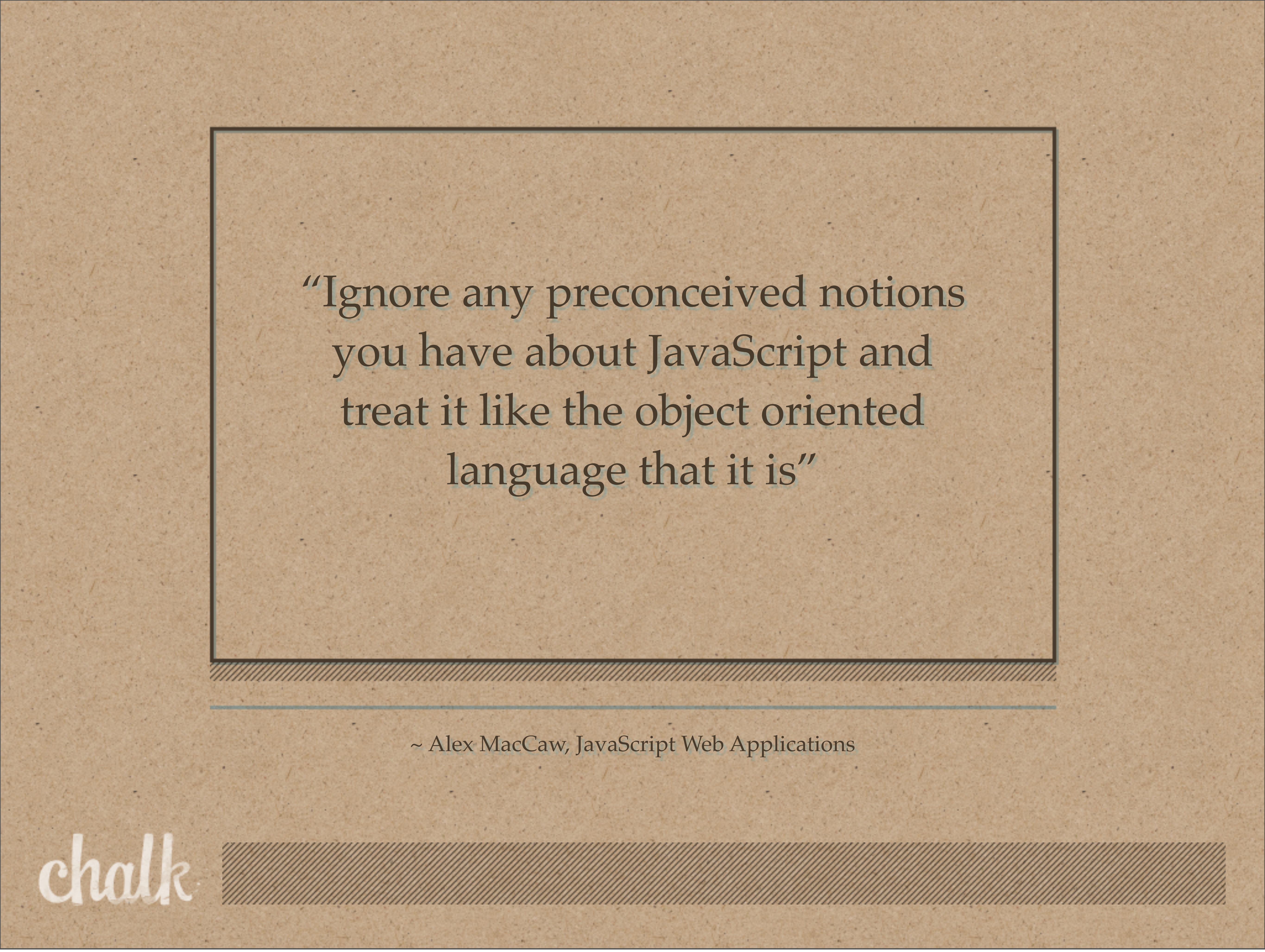
- First run at this, and what ends up happening with little projects
- Simplified HTML
- Might be asking, what's wrong with this – it's only five lines of code
    - Adding additional functionality becomes more difficult and tangled
    - Can't use the functionality elsewhere, like on product integrations (another part of the form)
    - Inefficient selectors, jumping in the pool over and over
    - Will grow into an unmaintainable mess as features are added.



"Paying a bit of attention to an application's structure when you start building it can make a big difference to the end result"

~ Alex MacCaw, JavaScript Web Applications

- Was talking about large applications, but why not apply it smaller projects
  - smaller projects can become large projects
  - if you are diligent on small projects, larger projects become easier to handle.
  - Start writing tests on small projects. Easiest way to get started



"Ignore any preconceived notions you have about JavaScript and treat it like the object oriented language that it is"

~ Alex MacCaw, JavaScript Web Applications

- How do we start off our projects with structure?
- What do we use

INDEPENDENT COMPONENTS

chalk

- Borrows from frameworks
  - called controllers, components, views, etc.
  - keep rendering, event handling and updates in one place

```
<fieldset id="categorizations">
  <input type="checkbox" id="cat1">
  <input type="checkbox" id="cat2">
  <input type="checkbox" id="cat3">
  ...
</fieldset>
```

```
$('#categorizations input').change(function(){
  if( $('#categorizations input:checked').length >= 3 )
    $('#categorizations input').not(':checked').prop('disabled', true);
  else
    $('#categorizations input').prop('disabled', false);
});
```

chalk

- Here’s what we had before
    - Create an object that can encapsulate this behavior

```
var CheckboxGroupController = function(view) {
  this.view = $(view);
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};


CheckboxGroupController.prototype.limit = function() {
  var checked = this.inputs.filter(':checked').length >= 3;
  this.view.find('input:not(:checked)').prop('disabled', checked);
}


new CheckboxGroupController('#categorizations');
```

chalk

- Not going to get into js oop and prototypical inheritence. basically...
    - create a constructor that takes a string or jquery object and saves it.
    - create a method that checks if 3 or more inputs are checked, then disable all the rest
    - listen for change event on any inputs
    - using delegation, one event handler no matter how many inputs
    - create a new instance, passing in a selector that surrounds our inputs

```
var CheckboxGroupController = function(view) {
  this.view = $(view);
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};


CheckboxGroupController.prototype.limit = function() {
  var checked = this.inputs.filter(':checked').length >= 3;
  this.view.find('input:not(:checked)').prop('disabled', checked);
}


new CheckboxGroupController('#categorizations');
```

Binds to the correct context

- This technique makes you jump through a hoop or two to keep your context.
- jQuery makes ‘this’ refer to the input element instead of our object
- use the jQuery proxy method to keep ‘this’ referring to our component object
- otherwise this.inputs wouldn’t exist in our limit function

EASIER TO TEST

chalk

- We're passing in a view. We can pass in a string in our tests and jQuery will parse it into a dom object.
- Doesn't need to exist on the page.
- Faster! doesn't need to start up an entire application dom

Encapsulated Behavior

- You always know this object is in charge of its own behavior
- No one else should be telling this object what to do
- Concerns are separated/single responsibility

EXTERNAL API

- We can now call the limit method if need be. Even if someone hasn’t clicked a checkbox. May come in handy in the future.
- Will talk shortly about a better way to use the API

```
var CheckboxGroupController = function(view) {
  this.view = $(view);
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};


CheckboxGroupController.prototype.limit = function() {
  var checked = this.inputs.filter(':checked').length >= 3;
  this.view.find('input:not(:checked)').prop('disabled', checked);
}


new CheckboxGroupController('#categorizations');
```

chalk

– Example of programmatically testing the number of checkboxes

```
var CheckboxGroupController = function(view) {
  this.view = $(view);
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};

CheckboxGroupController.prototype.limit = function() {
  var checked = this.inputs.filter(':checked').length >= 3;
  this.view.find('input:not(:checked)').prop('disabled', checked);
}

var c = new CheckboxGroupController('#categorizations');
c.limit();
```

chalk

- save component to variable
- can call methods
- This is good but we can do better. Several things that can be cleaned up.

BREAK APART METHOD BEHAVIOR

Where it makes sense

chalk

- single responsibility.
- Could come in handy in the future to be able to disable all checkboxes, without checking how many were checked.

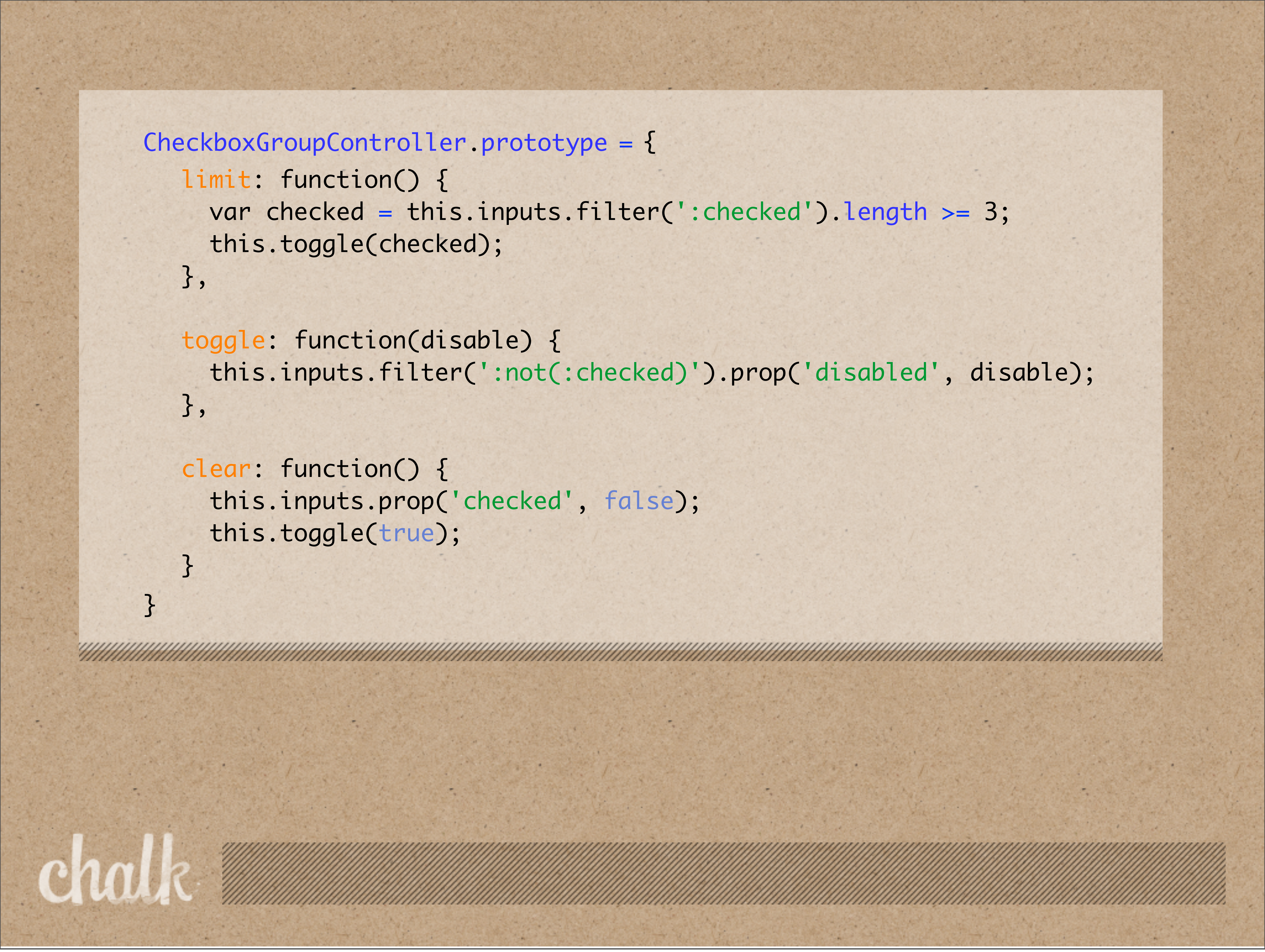
```
var CheckboxGroupController = function(view) {
  this.view = $(view);
  this.inputs = this.view.find('input');
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};

CheckboxGroupController.prototype.limit = function() {
  var checked = this.inputs.filter(':checked').length >= 3;
  this.view.find('input:not(:checked)').prop('disabled', checked);
}


new CheckboxGroupController('#categorizations');
```

```
CheckboxGroupController.prototype.limit = function() {
  var checked = this.inputs.filter(':checked').length >= 3;
  this.view.find('input:not(:checked)').prop('disabled', checked);
}
```

Method is comparing checked fields and performing the disabling action



```
CheckboxGroupController.prototype = {
  limit: function() {
    var checked = this.inputs.filter(':checked').length >= 3;
    this.toggle(checked);
  },

  toggle: function(disable) {
    this.inputs.filter(':not(:checked)').prop('disabled', disable);
  },

  clear: function() {
    this.inputs.prop('checked', false);
    this.toggle(true);
  }
}
```

- Can call toggle to disable or enable all unchecked boxes. Added clear to uncheck all boxes and set everything to disabled.
- will use clear in a second
- can still call limit() with the same result as before.
- Now we can more easily unit test specific behavior.
- create a new instance with a string of inputs, assert that limit, toggle, clear are doing what they're supposed to.

ALLOW COMPONENT TO BE CONFIGURED

- Can still improve our component
- What if we want to change the max number of categories to 2 or 4
- What if we want to have similar behavior elsewhre, but with a different number of max checkboxes.
- Adding configuration will make our component more generic. Allow it to be reused

```
var CheckboxGroupController = function(view) {
  this.view = $(view);
  this.inputs = this.view.find('input');
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};
```

chalk

- Current implementation
- passing in a selector or jquery object,
- saving all the inputs
- setting up event handler

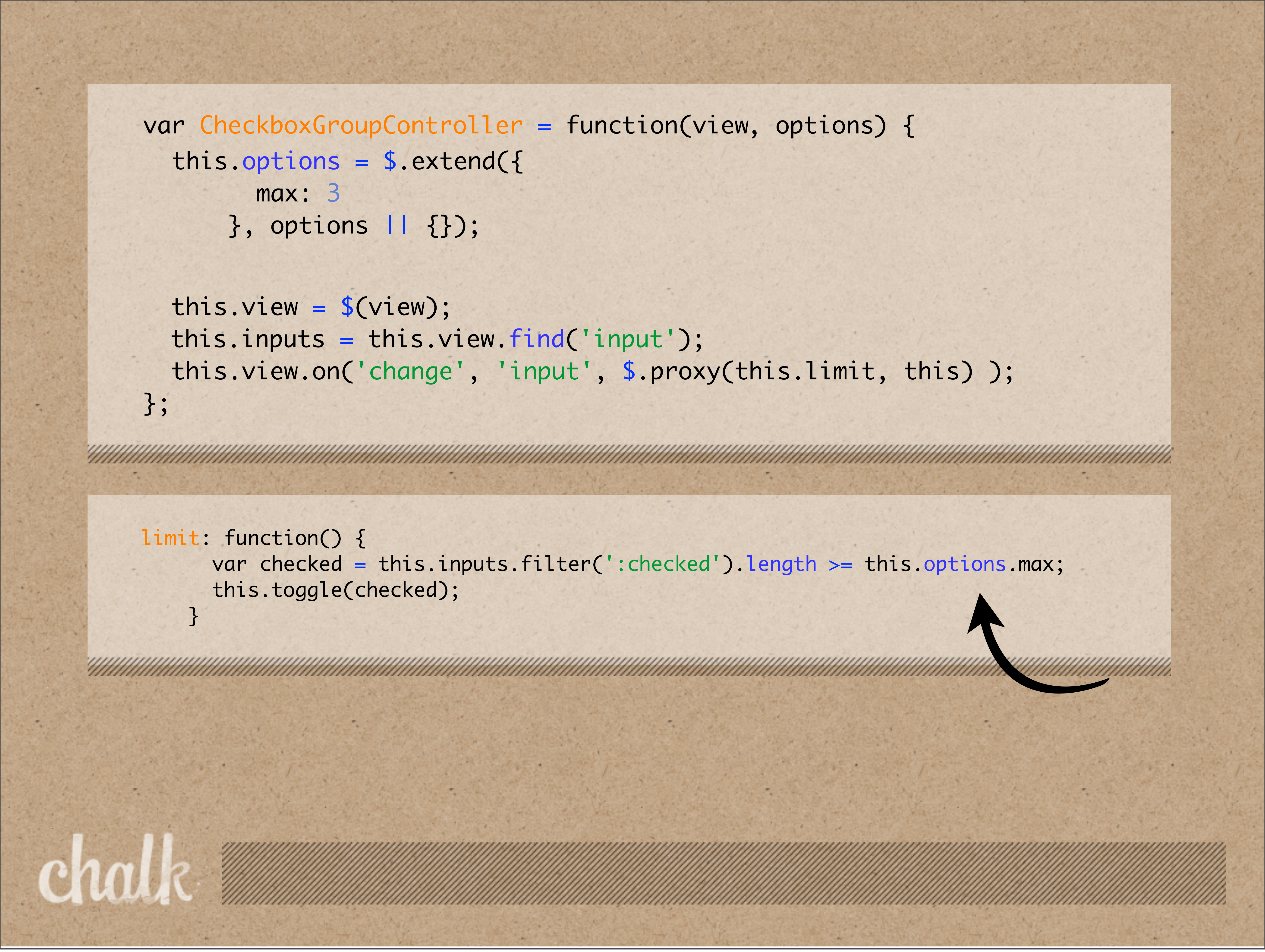
```
var CheckboxGroupController = function(view, options) {
  this.view = $(view);
  this.inputs = this.view.find('input');
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};
```

- adding an ‘options’ argument that will let us pass in a configuration object

```
var CheckboxGroupController = function(view, options) {
  this.options = $.extend({
        max: 3
      }, options || {});


  this.view = $(view);
  this.inputs = this.view.find('input');
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};
```

create an options property with defaults.
- $.extend will override the properties in the first object if the same property is on the options object
- if no object is passed in it will use the empty object, which will not override anything, since it doesn't have any of the same properties as the default object



```
var CheckboxGroupController = function(view, options) {
  this.options = $.extend({
        max: 3
      }, options || {});

  this.view = $(view);
  this.inputs = this.view.find('input');
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};
```

```
limit: function() {
      var checked = this.inputs.filter(':checked').length >= this.options.max;
      this.toggle(checked);
    }
```

- Now we can use our options object in our methods.
- The property can be different per instance

```
var CheckboxGroupController = function(view, options) {
  this.options = $.extend({
    max: 3
  }, options || {});

  this.view = $(view);
  this.inputs = this.view.find('input');
  this.view.on('change', 'input', $.proxy(this.limit, this) );
};

CheckboxGroupController.prototype = {
  limit: function() {
    var checked = this.inputs.filter(':checked').length >= this.options.max;
    this.toggle(checked);
  },

  toggle: function(disable) {
    this.inputs.filter(':not(:checked)').prop('disabled', disable);
  },

  clear: function() {
    this.inputs.prop('checked', false);
    this.toggle(true);
  }
}

new CheckboxGroupController('#categorizations', {max: 2});
```

chalk

Put it all together
- notice the configuration object being passed in when instantiating the object
- but we went from 5 lines to 20 something lines of code, is it worth it?

MORE EFFICIENT

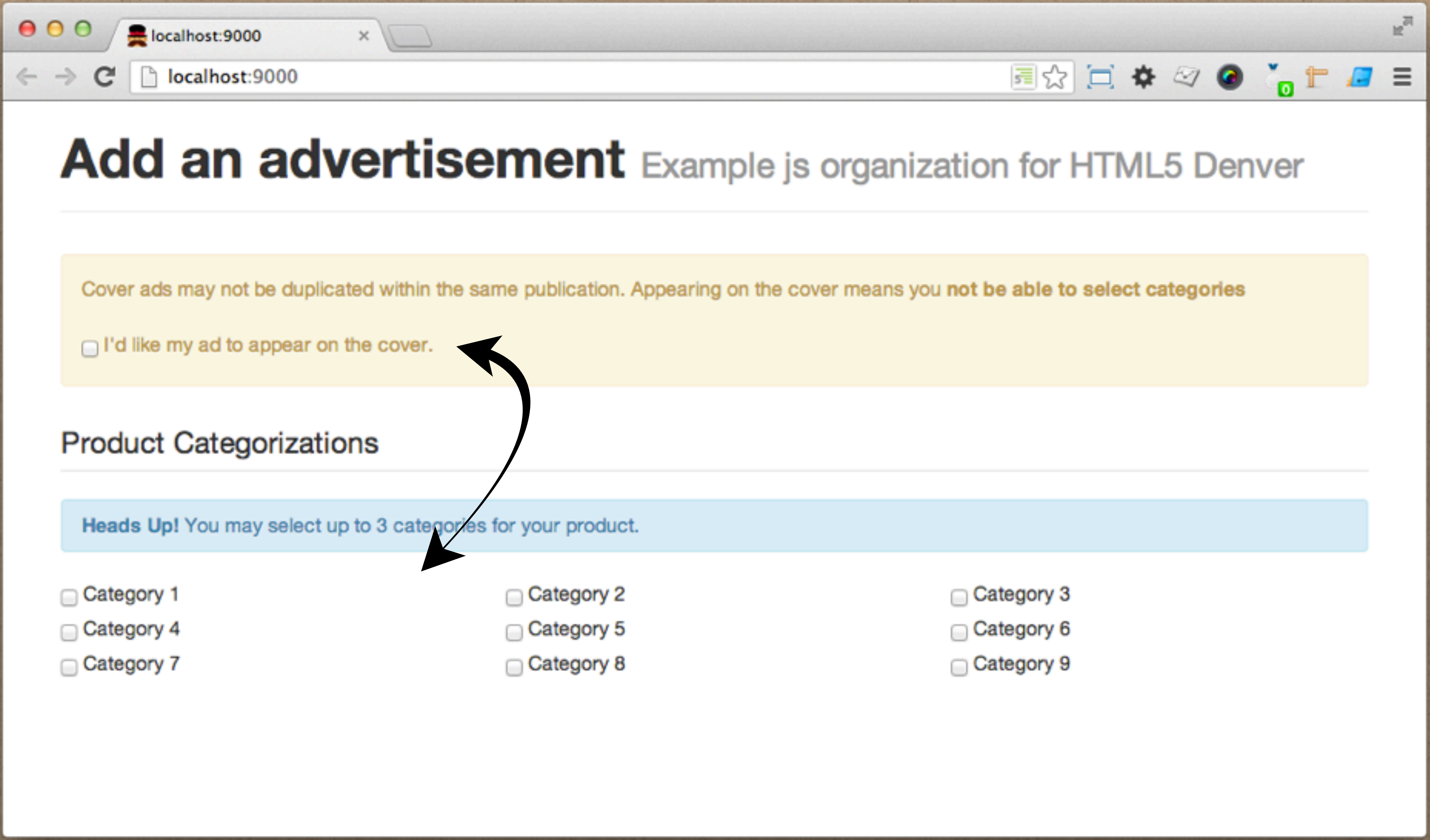
- No longer jumping in the pool multiple times to get the inputs

REUSABLE

- can configure the module for reuse elsewhere with similar functionality

TESTABLE

- methods can be independently tested and they’re fast



Add an advertisement Example js organization for HTML5 Denver

Cover ads may not be duplicated within the same publication. Appearing on the cover means you **not be able to select categories**

☐ I'd like my ad to appear on the cover.

Product Categorizations

Heads Up! You may select up to 3 categories for your product.

☐ Category 1	☐ Category 2	☐ Category 3
☐ Category 4	☐ Category 5	☐ Category 6
☐ Category 7	☐ Category 8	☐ Category 9

chalk

- What happens when we have behavior that crosses components
- Checking the first box means your ad is on the cover, you don't pick categories
- checking the box should uncheck and disable the checkboxes
- already have this method - called clear on our checkboxgroup component

```
var c = new CheckboxGroupController('#categorizations');


var CheckboxGroupDisabler = function(view) {
  this.view = $(view);
  this.input = this.view.find('input');

  this.view.on('change', 'input', $.proxy(this.disable, this) );
};

CheckboxGroupDisabler.prototype.disable = function() {
  this.input.prop('checked') ?
    c.clear() && c.toggle(true) :
    c.toggle(false);
};

new CheckboxGroupDisabler('#disabler');
```

Coupling is baaaad!

chalk

- first pass
- wait till end.... This works, but it’s got a problem

DON'T CROSS ~~THE STREAMS~~ Components

chalk

- The CheckboxGroup instance MUST exist an must be called ‘c’ or we’ll get an error.
- Should be able to add/remove individual components without causing errors.
- One component shouldn’t need to cross over into another components functionality
- Components only care about themselves

USE EVENTS FOR COMMUNICATION

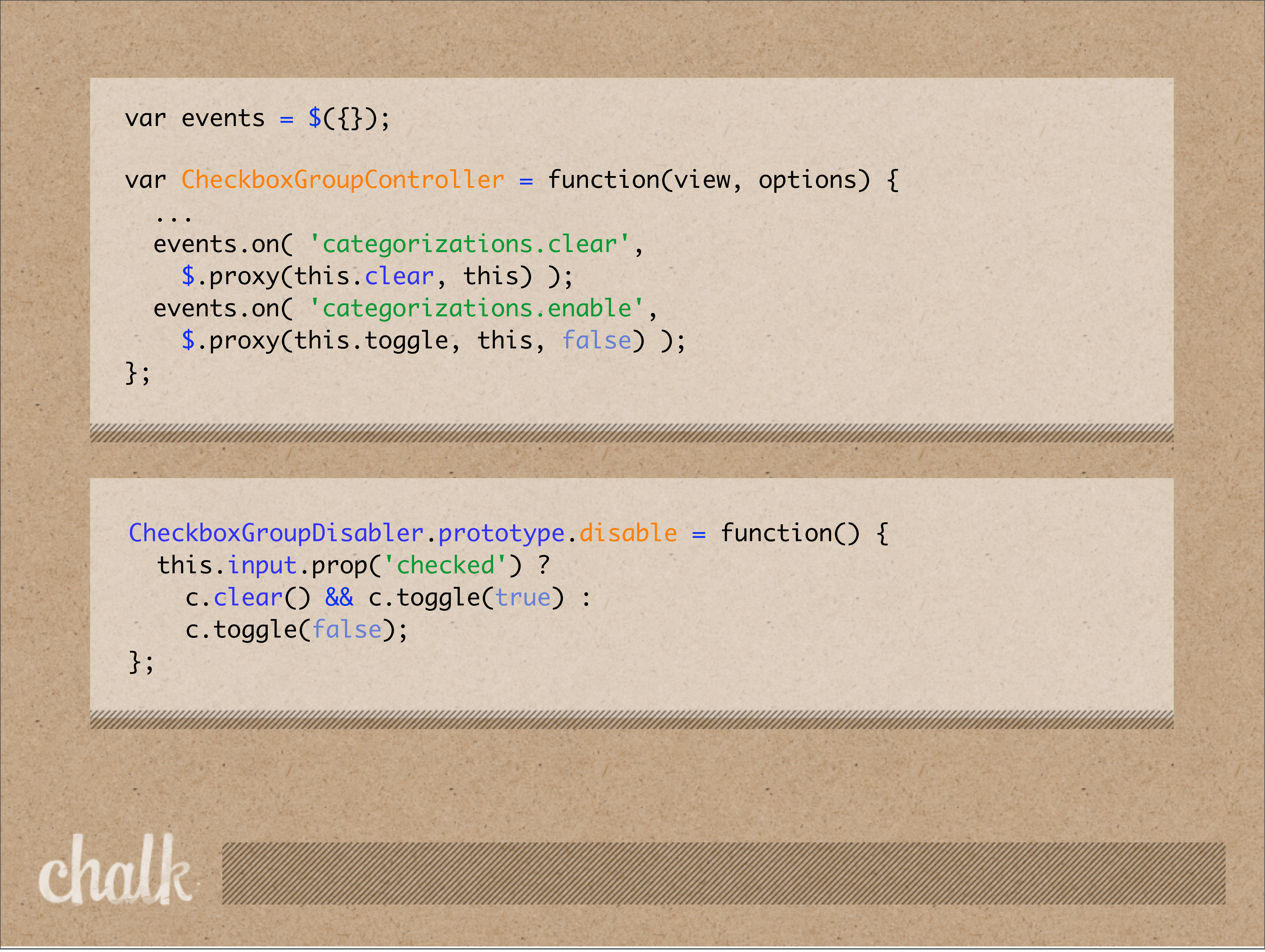
- We can solve this by using events
- Spine, Backbone, etc. have their own event implementations, let you listen for events happening to your data, or trigger events when something happens
- Standalone libraries exist like eventemitter.js
- If you have jquery you can create your own event emitter

$({})

- wrap a primative object in the jquery function

JQUERY'S ENTIRE EVENT API IN A FEW CHARACTERS

- can use all of jquery's event methods without having to pass in a DOM object (which we don't need in our case)
- on, trigger, off, etc.



```
var events = $({});

var CheckboxGroupController = function(view, options) {
  ...
  events.on( 'categorizations.clear',
    $.proxy(this.clear, this) );
  events.on( 'categorizations.enable',
    $.proxy(this.toggle, this, false) );
};
```

```
CheckboxGroupDisabler.prototype.disable = function() {
  this.input.prop('checked') ?
    c.clear() && c.toggle(true) :
    c.toggle(false);
};
```

... go through upper box
– save our jquery object to a variable that both components can access
– Going to have our checkbox group listen for events. if it gets one, perform the methods.
– Behavior is still internal
... bottom
– this is what we were doing when our top checkbox button was checked. Let’s change it to use events.

```
var events = $({});

var CheckboxGroupController = function(view, options) {
  ...
  events.on( 'categorizations.clear',
    $.proxy(this.clear, this) );
  events.on( 'categorizations.enable',
    $.proxy(this.toggle, this, false) );
};
```

```
CheckboxGroupDisabler.prototype.disable = function() {
  this.input.prop('checked') ?
    events.trigger('categorizations.clear') :
    events.trigger('categorizations.enable');
};
```

chalk

- The checkbox now just triggers events.
- Not concerned about what happens after the events have been triggered.
- Again easily testable! Not setting up intertwined relationships between models. Fast!

SINGLE RESPONSIBILITY

chalk

- Every component for itself
- easily testable - don’t have to look at the dom, just check that an event was fired.

LOOSELY COUPLED

- Can remove or add modules without js errors.
- Keeps the system maintainable. Can continue adding new modules
- No more jQuery soup!

chalk

hummel@wearechalk.com

@jhummel

- Thanks!
- email me or reach me on twitter, would love feedback