Modular OO JS

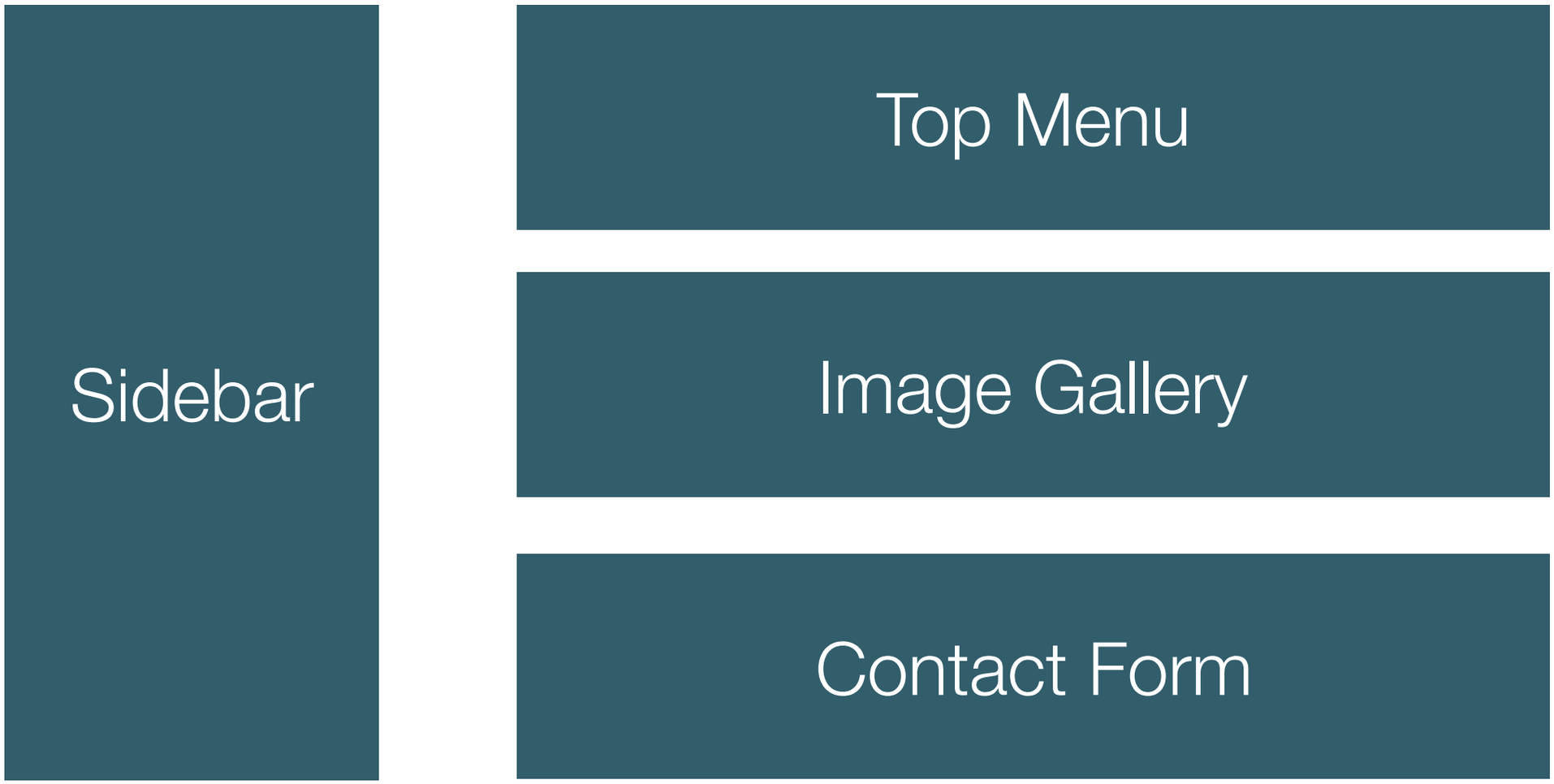
Ynon Perek

JS Problems

- Everything's global
- (So far) we can only write global functions
- => Hard to write large apps

What We Want

- Components Based Architecture



Requirements for Components

- Write once, use many
- Accessible via Interface
- Share between projects / pages
- Easy to test
- Depend on other components

A Simple JS Class (is just a function)

```
function Person(name, color) {
  this.name = name;
  this.favoriteColor = color;
}

var p = new Person('joe', 'blue');
var q = new Person('jane', 'purple');

console.log('joe likes ', p.favoriteColor);
```

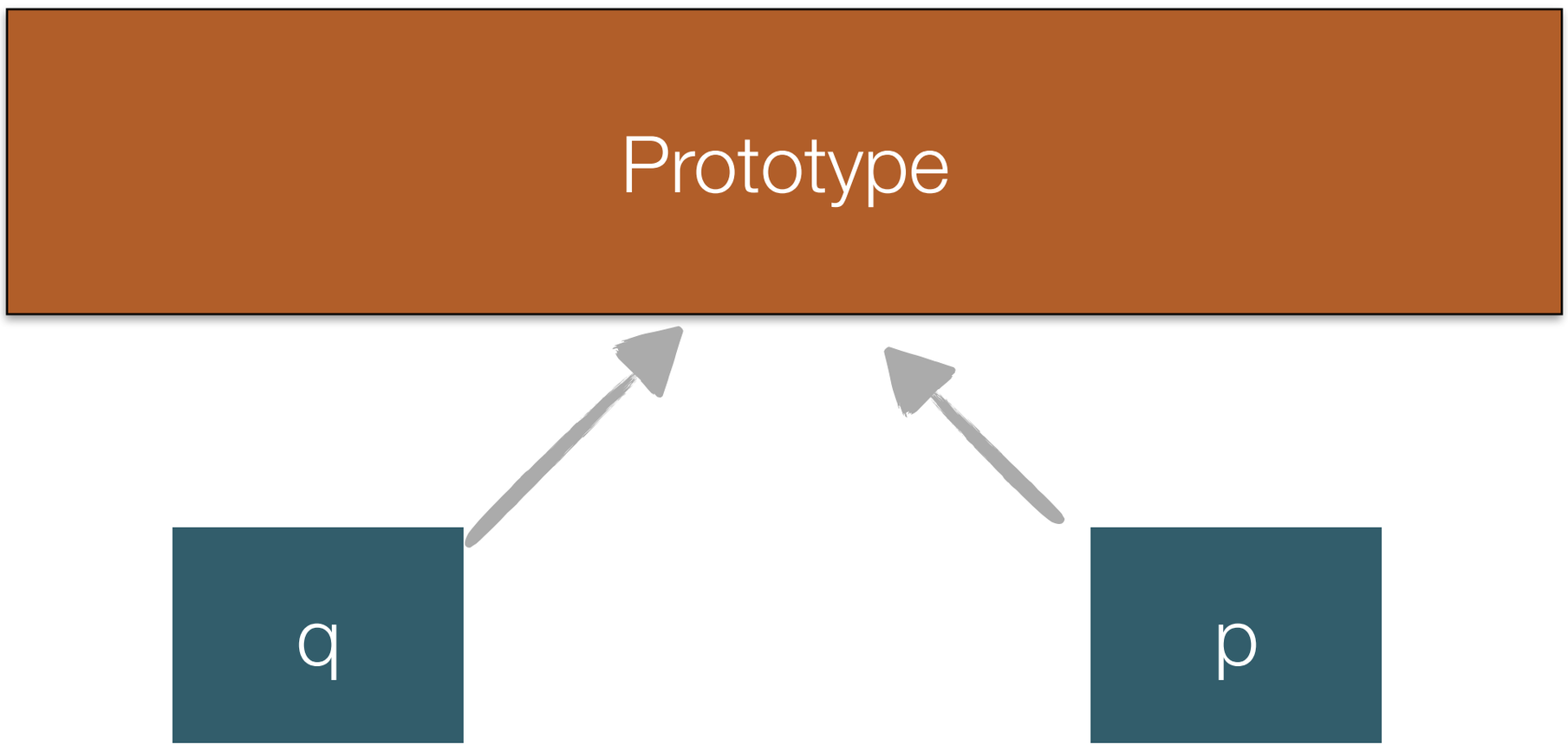
How did it work?

- JS provides many ways to call a function:
  - Globally: `func()`
  - From an object: `btn.addEventListener()`
- Now we learned a new one:
  - new `func()`

What new does

- Create a new JS object
- Pass that object to the function as “this”
- If the function doesn’t have an explicit return, automatically return that new object
- Set new object’s prototype to function.prototype

JavaScript Prototypes



JavaScript Prototypes

- Searching a property on an object first checks that object
- If property wasn't found, JS searched in its prototype
- And recursively up the prototype chain

Adding Methods To Person

```
function Person(name, color) {
  this.name = name;
  this.favoriteColor = color;
}

Person.prototype.hello = function() {
  console.log(`Hi! My name is ${this.name}`);
  console.log(`My favorite color is ${this.favoriteColor}`);
};

var p = new Person('joe', 'blue');
var q = new Person('jane', 'purple');

console.log('joe likes ', p.favoriteColor);
```

Private Members / Methods

- JavaScript doesn't support access control to members of a class

- We use _ prefix to mark a field as "private" by convention

```
function Person(name, color) {
  this.name = name;
  this._favoriteColor = color;
}
```

Type Safety

- JavaScript doesn't support static types for variables
- We sometimes use typeof inside our methods

```
function Person() {
}

Person.prototype.countTill = function(num) {
  if (typeof num !== 'number') {
    throw new Error(`${num} is not a number`);
  }

  // continue counting...
}
```

Object Oriented Lab #1

- Write a class called summer so the following code works

```
var s = new Summer();
var t = new Summer();

s.add(10, 20);
s.add(30);

t.add(5, 7);
t.add(5);

// prints 60
console.log(s.total());

// prints 17
console.log(t.total());
```

Object Oriented Lab #2

- Add to the starter code here: http://codepen.io/ynonp/pen/amzWzg

- A new class called Race which can take several cars and return which one is the fastest, so the code in the starter works and returns the correct answer

Q & A

Components

- A components is a class that works on the DOM
- It adds:
    - a base DOM element to work on
    - Event handling
- No special syntax

Components: How

- Take a DOM node in the constructor
- Write initial structure to DOM
- Register event handlers
- Implement methods to interact with other components

Components: Demo

```
function Person($el, name, color) {
  $el.html(`
    <div class="person">
      <span style="color: ${color}">${name}</span>
    </div>
  `);

  this.$el = $el;
  this.name = name;
  this.color = color;
}
```

Event Handling

- By default event handlers are called automatically by the browser in response to events
- Argument is the event object
- “this” is the DOM element created the event
- Demo

Event Handling + Components

- To use methods as event handlers, we need to “bind” the methods to the object

- This is done automatically in most languages

```
MyButton.prototype.registerEvents = function() {
  this.$el.on('click', this.handleClick.bind(this));
};
```

Demo: Components + Events

- We'll write a click counter component:
  - Shows a button and a text panel with a number
  - Every click on the button increases the number
  - Has click() method that increases value
  - Can increase value externally

Components Lab

- Write a component for multiple synced text boxes:
  - Show 5 <input /> elements
  - Provide “clear” method that allows external code to clear value in all boxes
  - When user changes value in one <input>, new value should be copied to all other inputs
  - Provide an “add” method that adds a new <input> to the box

Q & A

Sharing Code Between Components / Classes

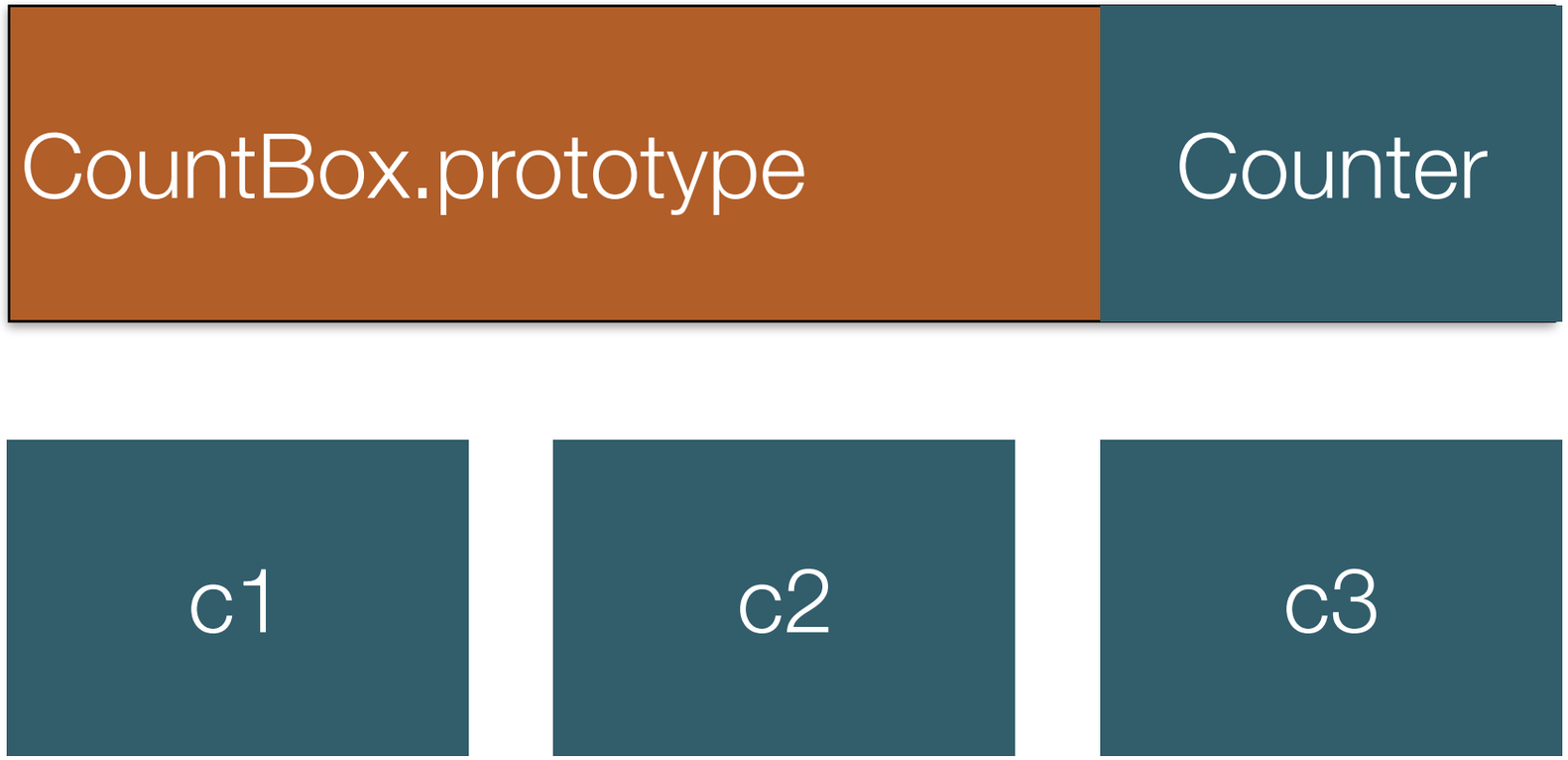
- **Prototypical Inheritance**: Components' prototypes have the same prototype

- **Delegation**: Shared methods are copied into both components' prototypes

Delegation

```
var Counter = {
  reset: function() {
    this._value = 0;
  },
  inc: function() {
    this._value++;
  }
};

function CountBox($el) {
  this.reset();
  // ...
}

Object.assign(CountBox.prototype, Counter);
```

Delegation



Prototypical Inheritance

```
function BaseComponent($el) {
  this.$el = $el;
}

BaseComponent.clear = function() {
  this.$el.empty();
};

function MyComponent($el) {
  BaseComponent.call(this, $el);
}

MyComponent.prototype = Object.assign(
  {},
  BaseComponent.prototype);
```

Demo: Prototypical Inheritance

- Build two static pages both as components
- Add buttons to hide/show pages
- Use a single base class to implement the buttons

Lab: Sharing Code

- Change inheritance from previous demo to delegation
- Which one is more appropriate for the situation?

Lab: Catch Red! Game

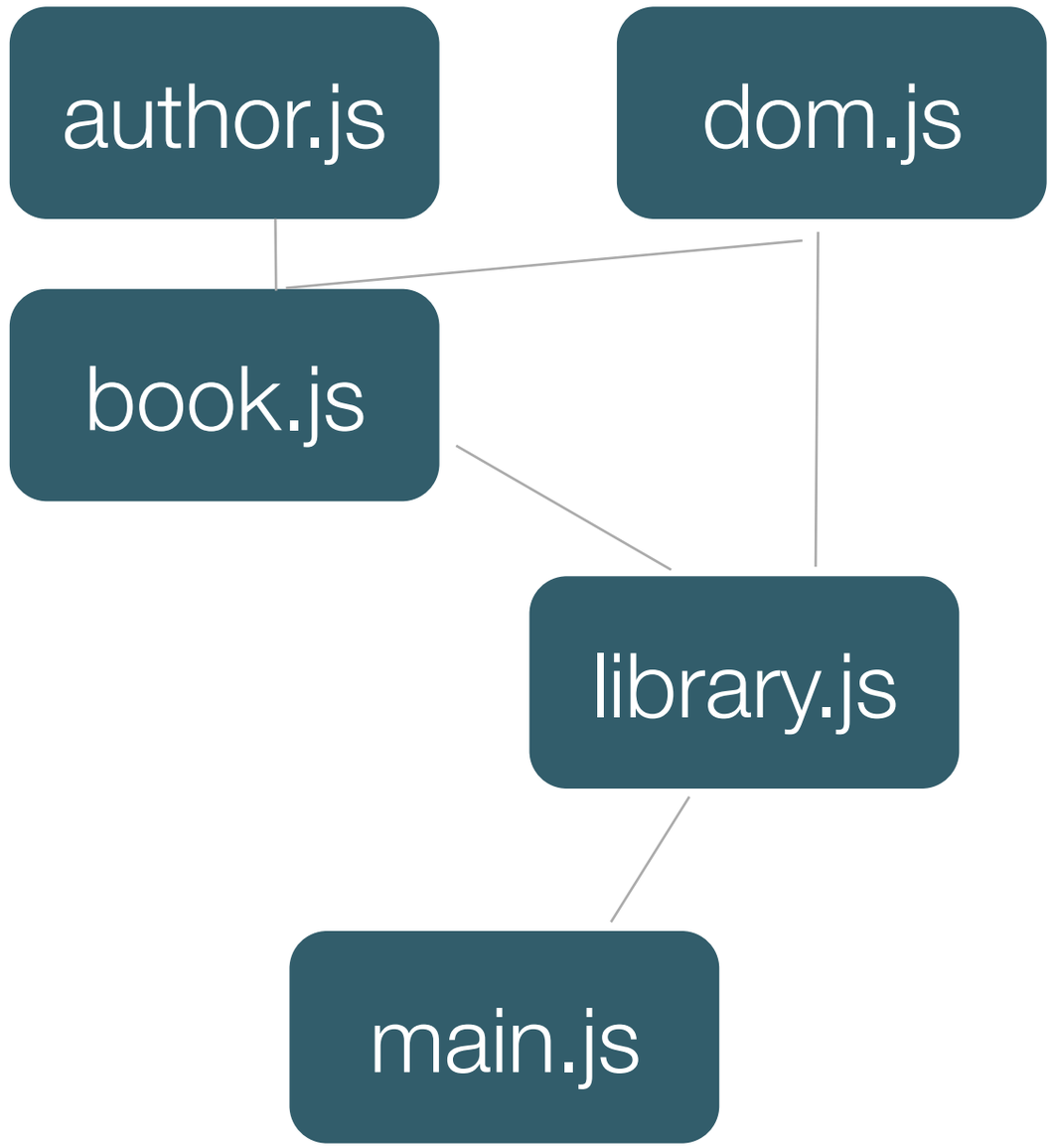
- Implement Catch Red game in an object oriented manner
  - Use a “Game” class to handle game logic
  - Use a “Player” class to store player data (name and score)
  - Allow user to change player name via a form
- Game should call Player’s API to increase score
- Game should provide #newGame method

ES6 Classes

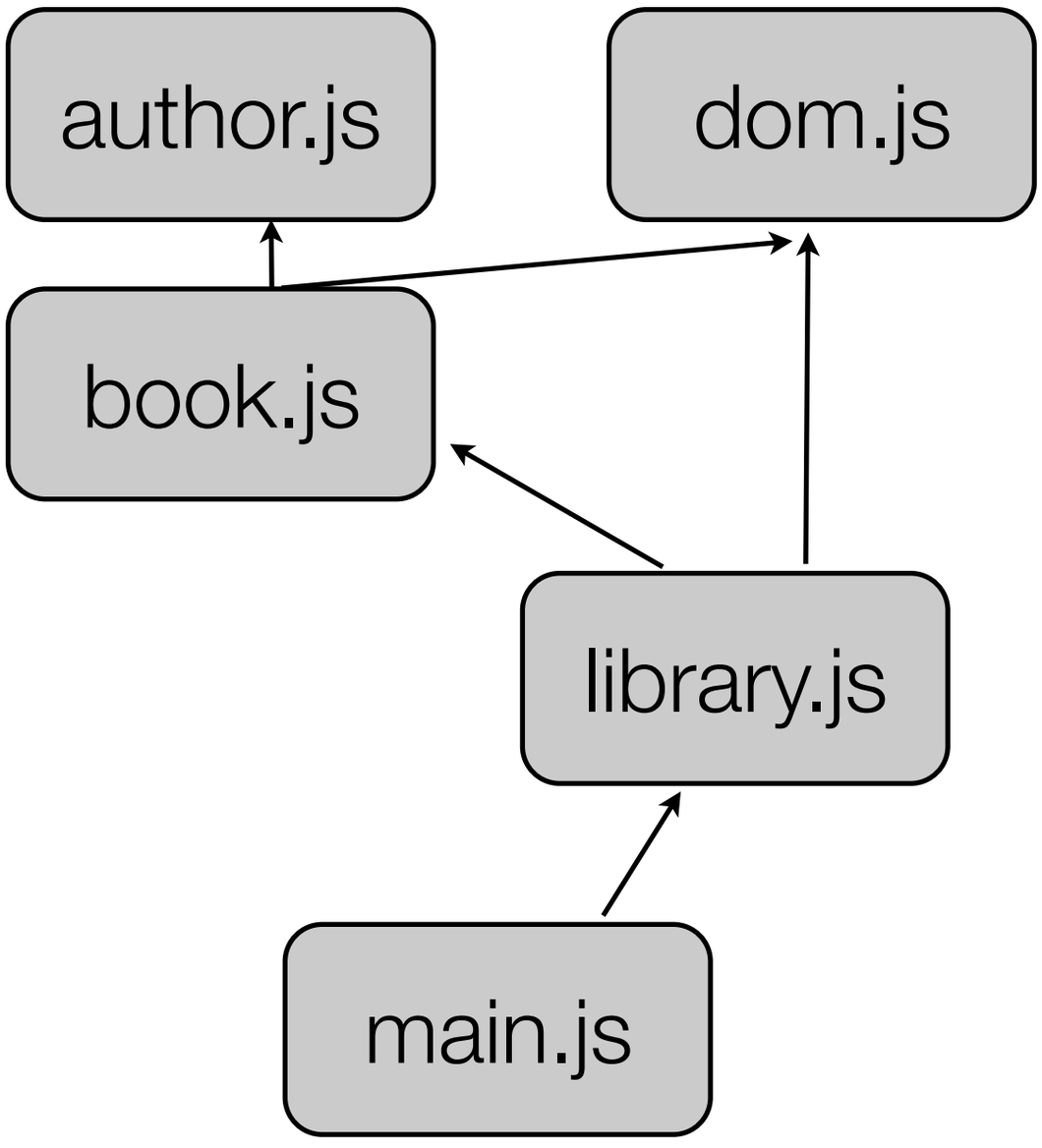
- ES6 added syntactic sugar for declaring classes which work on all major new browsers

```
class Person {
  constructor(name) {
    this.name = name;
  }

  hello(other) {
    console.log(`Hello! I am ${name}`);
    console.log(`And you are ${other.name}`);
  }
}

var p = new Person('joe');
var q = new Person('jane');

p.hello(q);
```

Modules



Using <script> Tags




```
<script src="dom.js"></script>
<script src="author.js"></script>
<script src="book.js"></script>
<script src="library.js"></script>
<script src="main.js"></script>
```

Disadvantages Of Script Tags

- No Hierarchy in modules
- No optional modules
- Too many globals

Enter Module Loaders

- ES6 provides new syntax to declare and use modules
- No browser supports it yet, and we have tons of legacy ways to declare and use modules
- A “module loader” is a tool that bundles together modules so browsers will be able to understand them. Think of it as a build tool for JS

Declaring a module

- Modules export one or more objects
- Some modules have default exports:
    - export one object, importer needs to select a name
- Some modules have normal exports:
    - export multiple objects, importer needs to specify which to import

Person Module

in file src/person.js

```
export default class Person {
  constructor(name) {
    this.name = name;
  }

  hello(other) {
    console.log(`Hello! I am ${this.name}`);
    console.log(`And you are ${other.name}`);
  }
}
```

Using Person Module

in file src/main.js

```
import Person from 'person';

const p = new Person('joe');
const q = new Person('jane');

p.hello(q);
```

Testing Person Module

in file spec/person-spec.js

```
import Person from 'person';

describe('Person', function() {
  describe('#constructor', function() {

    it('should store the name', function() {
      const p = new Person('joe');
      expect(p.name).toEqual('joe');
    });

  });
});
```

Bundling the Modules

- Our module bundler is called webpack
- To bundle all modules into a single dist/bundle.js file we run (from project directory):
- `node node_modules/webpack/bin/webpack.js -d`

Bundler Advantages

- Offer real separation between modules
- Can have dependencies for modules
- Can run code before bundling (e.g. babel)
- No need to list all modules in the HTML file

Lab

- Modify Catch Red! game to use modules architecture

Q & A

Single Page Architecture

- Traditional web applications use server side routing to change current page

home.php

about.php

contact.php

Single Page Architecture

- Recently application developers started to change into client side routing, mainly for performance

Single Page Architecture

- Each page is a component with its own HTML, CSS and JS

- Can use Webpack to separate files, or write all in one file

- Routing is done by detecting hash change events, or a#click events

- Many frameworks were created to provide easier developer experience for such apps

Demo: Let’s build a single page application

Q & A

Thanks For Listening

- Read more: www.tocode.co.il
- Talk to me: ynon@tocode.co.il
- Photos from: http://123rf.com