Organizing Code in Web Apps

SWE 432, Fall 2016
Design and Implementation of Software for the Web
Today

- Some basics on how and why to organize code (SWE!)
- Closures
- Classes
- Modules

For further reading:
“Wow back in my day before ES6 we didn’t have your fancy modules”
Spaghetti Code
```javascript
window.onload = function () {
  eqCtl = document.getElementById('eq');
currNumberCtl = document.getElementById('currNumber');
}

var eqCtl,
currNumberCtl,
operator,
operatorSet = false,
equalsPressed = false,
lastNumber = null;

function add(x,y) {
  return x + y;
}

function subtract(x, y) {
  return x - y;
}

function multiply(x, y) {
  return x * y;
}

function divide(x, y) {
  if (y == 0) {
    alert("Can't divide by 0");
    return 0;
  }
  return x / y;
}

function setVal(val) {
  currNumberCtl.innerHTML = val;
}

function setEquation(val) {
  eqCtl.innerHTML = val;
}

function clearNumbers() {
  lastNumber = null;
  equalsPressed = operatorSet = false;
  setVal('0');
  setEquation('');
}

function setOperator(newOperator) {
  if (newOperator == '=') {
    equalsPressed = true;
    calculate();
    setEquation('');
    return;
  }
  if (!equalsPressed) calculate();
  equalsPressed = false;
  operator = newOperator;
  operatorSet = true;
  lastNumber = parseFloat(currNumberCtl.innerHTML);
  var eqText = (eqCtl.innerHTML == '') ? lastNumber + ' ' + operator + ' ' : eqCtl.innerHTML + ' ' + operator + ' ';
  setEquation(eqText);
}

function numberClick(e) {
  var button = (e.target) ? e.target : e.srcElement;
  if (operatorSet == true || currNumberCtl.innerHTML == '0') {
    setVal('');
    operatorSet = false;
  }
  setVal(currNumberCtl.innerHTML + button.innerHTML);
  setEquation(eqCtl.innerHTML + button.innerHTML);
}

function calculate() {
  if (!operator || lastNumber == null) return;
  var currNumber = parseFloat(currNumberCtl.innerHTML);
  var eqText = (eqCtl.innerHTML == '') ? currNumber + ' ' + operator + ' ' : eqCtl.innerHTML + ' ' + operator + ' ';
  setEquation(eqText);
}

function numberClick(e) {
  var button = (e.target) ? e.target : e.srcElement;
  if (operatorSet == true || currNumberCtl.innerHTML == '0') {
    setVal('');
    operatorSet = false;
  }
  setVal(currNumberCtl.innerHTML + button.innerHTML);
  setEquation(eqCtl.innerHTML + button.innerHTML);
}
```

...aka big ball of mud aka shanty town code

Brian Foote and Joe Yoder
Bad Code “Smells”

- Tons of not-very related functions in the same file
- No/bad comments
- Hard to understand
- Lots of nested functions

```javascript
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + ' to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename,
              if (err) console.log)
          }).bind(this))
        }
      })
    });
  }
});
```
Design Goals

• Within a component
  • Cohesive
  • Complete
  • Convenient
  • Clear
  • Consistent
• Between components
  • Low coupling
Cohesion and Coupling

- Cohesion is a property or characteristic of an individual unit
- Coupling is a property of a collection of units
- High cohesion GOOD, high coupling BAD
- Design for change:
  - Reduce interdependency (coupling): You don't want a change in one unit to ripple throughout your system
  - Group functionality (cohesion): Easier to find things, intuitive metaphor aids understanding
Design for Reuse

• Why?
  • Don’t duplicate existing functionality
  • Avoid repeated effort

• How?
  • Make it easy to extract a single component:
    • Low coupling between components
    • Have high cohesion with
Design for Change

• Why?
  • Want to be able to add new features
  • Want to be able to easily **maintain** existing software
    • Adapt to new environments
    • Support new configurations

• How?
  • Low **coupling** - prevents unintended side effects
  • High **cohesion** - easier to find things
Organizing Code

How do we structure things to achieve good organization?

<table>
<thead>
<tr>
<th>Individual Pieces of Functional Components</th>
<th>Java</th>
<th>Javascript</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td></td>
<td>Classes</td>
</tr>
<tr>
<td>Entire libraries</td>
<td>Packages</td>
<td>Modules</td>
</tr>
</tbody>
</table>
Closures

• Closures are expressions that work with variables in a specific context
• Closures contain a function, and its needed state
  • Closure is that function and a **stack frame** that is allocated when a function starts executing and **not freed** after the function returns
Closures & Stack Frames

• What is a stack frame?
  • Variables created by function in its execution
  • Maintained by environment executing code

```javascript
function a() {
  var x = 5, z = 3;
  b(x);
}
function b(y) {
  console.log(y);
}
a();
```

Contents of memory:

<table>
<thead>
<tr>
<th>Stack frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>a: x: 5</td>
</tr>
<tr>
<td>z: 3</td>
</tr>
</tbody>
</table>

Function called: stack frame created
Closures & Stack Frames

• What is a stack frame?
  • Variables created by function in its execution
  • Maintained by environment executing code

```javascript
function a() {
  var x = 5, z = 3;
  b(x);
}
function b(y) {
  console.log(y);
}
a();
```

Contents of memory:

- **Stack frame**
  - `b: y: 5`
  - `a: x: 5`
  - `z: 3`

Function called: new stack frame created
Closures & Stack Frames

• What is a stack frame?
  • Variables created by function in its execution
  • Maintained by environment executing code

```javascript
function a() {
    var x = 5, z = 3;
    b(x);
}

function b(y) {
    console.log(y);
}

a();
```

Contents of memory:

```
a:  x: 5
   z: 3
```

Stack frame

Function returned: stack frame popped
Closures

• Closures are expressions that work with variables in a specific context
• Closures contain a function, and its needed state
  • Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
• That state just refers to that state by name (sees updates)

```javascript
var x = 1;
function f() {
  var y = 2;
  return function() {
    console.log(x + y);
    y++;
  }
}
var g = f();
g();  // 1+2 is 3
g();  // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them. It “closes up” those references.
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}

var g = f();
g(); // 1+2 is 3

g(); // 1+3 is 4

**Closures**

[Diagram showing the closure concept with variables and functions.]
Closures

```javascript
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}

var g = f();
g(); // 1+2 is 3

g(); // 1+3 is 4
```
```javascript
var x = 1;
function f() {
  var y = 2;
  return function() {
    console.log(x + y);
    y++;
  };
}

var g = f();
g(); // 1+2 is 3

var x = 1;

var g = f();
g(); // 1+3 is 4
```
Modules

• We can do it with closures!
• Define a function
  • Variables/functions defined in that function are “private”
  • Return an object - every member of that object is public!
• Remember: Closures have access to the outer function’s variables even after it returns
LaToza/Bell

GMU SWE 432 Fall 2016

Modules with Closures

```javascript
var facultyAPI = (function(){
    var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza", section:1}];

    return {
    getFaculty : function(i)
    {
    return faculty[i].name + " ("+faculty[i].section +")";
    }
    }
})(());

console.log(facultyAPI.getFaculty(0));
```

This works because inner functions have visibility to all variables of outer functions!
Closures gone awry

```javascript
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = function() { return i; };
}
```

What is the output of `funcs[0]()`?

> 5

Why?

Closures retain a *pointer* to their needed state!
Closures under control

Solution: IIFE - Immediately-Invoked Function Expression

```javascript
function makeFunction(n)
{
    return function(){
        return n;
    };
}
for (var i = 0; i < 5; i++) {
    funcs[i] = makeFunction(i);
}
```

Why does it work?

Each time the anonymous function is called, it will create a `new variable n`, rather than reusing the same variable `i`.

Shortcut syntax:

```javascript
var funcs = [];
for (var i = 0; i < 5; i++) {
    funcs[i] = (function(n) {
        return function() {
            return n;
        }
    })(i);
}
```
Exercise: Closures

var facultyAPI = (function(){
    var faculty = [{name:"Prof Bell", section: 2}, {name:"Prof LaToza", section:1}];

    return {
        getFaculty : function(i)
        {
            return faculty[i].name + " ("+faculty[i].section +")";
        }
    }
})();

https://jsfiddle.net/hkcq5vpa/

console.log(facultyAPI.getFaculty(0));

Here’s our simple closure. Add a new function to create a new faculty, then call `getFaculty` to view their formatted name.
Classes

A small correction:

Remember... There's no Class!

Our Object

var profJon = {
  firstName: "Jonathan",
  lastName: "Bell",
  teaches: "SWE 432",
  office: "ENGR 4322",
  fullName: function(){
    return this.firstName + " " + this.lastName;
  }
};

profJon.officeHours = "Tuesdays 10:30-12:00"

Lazily creates a new property and sets it

delete profJon.office;

Deletes a property

Lecture 4, JavaScript
Classes

• ES6 introduces the `class` keyword
• Mainly just syntax - still not like Java Classes

```javascript
function Faculty(first, last, teaches, office)
{
    this.firstName = first;
    this.lastName = last;
    this.teaches = teaches;
    this.office = office;
    this.fullName = function()
    {
        return this.firstName + " " + this.lastName;
    }
}

Old

var profJon = new Faculty("Jonathan", "Bell", "SWE432", "ENGR 4322");

New

class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
}

var profJon = new Faculty("Jonathan", "Bell", "SWE432", "ENGR 4322");
```
**Classes - Extends**

`extends` allows an object created by a class to be linked to a "super" class. Can (but don’t have to) add parent constructor.

```java
class Faculty {
    constructor(first, last, teaches, office) {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
}

class CoolFaculty extends Faculty {
    fullname() {
        return "The really cool " + super.fullname();
    }
}
```
Classes - static

**static** declarations in a class work like in Java

class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
    static formatFacultyName(f) {
        return f.firstName + " " + f.lastName;
    }
}

```java
class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
    static formatFacultyName(f) {
        return f.firstName + " " + f.lastName;
    }
}
```
Modules (ES6)

• With ES6, there is finally language support for modules
• Module must be defined in its own JS file
• Modules `export` declarations
  • Publicly exposes functions as part of module interface
• Code `import` modules (and optionally only parts of them)
  • Specify module by path to the file
Modules (ES6) - Export Syntax

```javascript
var faculty = [{name: "Prof Bell", section: 2}, {name: "Prof LaToza", section: 1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1, 2, 3];
var faculty = [{name: "Prof Bell", section: 2}, {name: "Prof LaToza", section: 1}];
var someVar = [1, 2, 3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
export {getFaculty as aliasForFunction, someVar};

export default function getFaculty(i) {...
```

Label each declaration with "export"

Or name all of the exports at once

Can rename exports too

Default export
Modules (ES6) - Import Syntax

• Import specific exports, binding them to the same name
  
  ```javascript
  import { getFaculty, someVar } from "myModule";
  getFaculty()...
  ```

• Import specific exports, binding them to a new name
  
  ```javascript
  import { getFaculty as aliasForFaculty } from "myModule";
  aliasForFaculty()...
  ```

• Import default export, binding to specified name
  
  ```javascript
  import theThing from "myModule";
  theThing()... -> calls getFaculty()
  ```

• Import all exports, binding to specified name
  
  ```javascript
  import * as facModule from "myModule";
  facModule.getFaculty()...
  ```
Patterns for using/creating libraries

- Try to reuse as much as possible!
- Name your module in all lower case, with hyphens
- Include:
  - README.md
  - keywords, description, and license in package.json (from npm init)
- Strive for high cohesion, low coupling
  - Separate models from views
  - How much code to put in a single module?
- Cascades (see jQuery)
Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):
  ```javascript
  str.replace("k","R").toUpperCase().substr(0,4);
  ```
- Example (jQuery):
  ```javascript
  $("#wrapper")
  .fadeOut()
  .html("Welcome")
  .fadeIn();
  ```
Demo: Modules

Not yet supported by any browser!
Closures Exercise

• Work from our example before of the Faculty Closure API to create a Class API (with Closures).

• Private fields:
  • Faculty API
  • List of students (students are objects with names, section numbers, and partners [which are students])

• Public functions:
  • Add a student to the class
  • Retrieve the name of the student’s faculty

https://jsfiddle.net/hkcq5vpa/1/
https://jsfiddle.net/hkcq5vpa/3/
Exit-Ticket Activity

Go to socrative.com and select “Student Login”

Class: SWE432001 (Prof LaToza) or SWE432002 (Prof Bell)
ID is your @gmu.edu email

1: How well did you understand today's material
2: What did you learn in today's class?
For question 3: What happens when the user clicks on the 4th button on this page and why?

```javascript
var nodes = document.getElementsByTagName('button');
for (var i = 0; i < nodes.length; i++) {
  nodes[i].addEventListener('click', function() {
    console.log('You clicked element #' + i);
  });
}
```