Practical Development of Web Applications with JavaScript and AngularJS

Unit 1. Intro to JavaScript. Prototyping. Responsive Web Design. Bootstrap Framework.

FARATA

Introducing Instructors

- Yakov Fain, Farata Systems
- Anton Moiseev, Farata Systems

http://faratasystems.com

Fast Paced Intro to JavaScript

What's HTML5?

HTML + JavaScript + CSS +
AJAX + HTML APIs +
Developer's Tools

Where to Run JavaScript?

- In Your Web browser
- In any JavaScript Engine, e.g. Google's V8, Oracle's Nashorn.

FARATA

JavaScript is Interpreted Language

JavaScript arrives to the place of execution as text.

There's no compiler helping developers with syntax errors.

Users may have different runtime environment.

Debugging JavaScript

- Firefox add-on FireBug
- Chrome Developer Tools - our choice
- Internet Explorer F12 Developer Tools
- Safari Develop
- Opera Dragonfly

FARATA

Variables

Declaring a variable (unless in <u>strict mode</u>: `'use strict';`) is optional: girlfriendName="Mary";

Variables declared *without* the keyword **var** are global.

Variables declared with **var** inside functions are local.

```
function calculateSalary() {
    var address = "123 Main Street";   // local String variable
    age = 25;                          // global Number variable
    var married = true;                // local boolean variable
    married = "don't remember";        // now it's String variable
}
```

Objects and Functions

Functions. Briefly.

You can declare a function that doesn't belong to any object and invoke it.

You can declare a function, assign it to a property of any object and invoke it there.

Functions can be objects

Functions have memory

Declaring and Invoking a Function

```
// Declaring
/*
 *  No var in arguments
 *  No data types in arguments
 *  No need to declare return type
*     even if a function return a value
 */
function calcTax(income, dependents) {
    // Do stuff here
}
```

```
//Invoking:

calcTax(50000, 2);
var myTax = calcTax(50000, 2);
```

```
//Declaring and invoking at the same time:

(function calcTax(income, dependents) {
    // Do stuff here
})();
```


FARATA

Function Expressions

```
//Assigning a function literal to a variable:

var doTax = function (income, dependents){
   // Do stuff here
}
```

```
//Invoking a function:

doTax(50000, 2);
```

Creating Objects

You can create objects with one of these methods:

1. Using object literals
2. Using `new Object()` notation
3. Create an object based on another object:
   `obj2=Object.create(obj1);`
4. Using constructor functions and a `new` operator

FARATA

Object Literals

An object is just a bunch of properties: key/value pairs.

```
var t = {};                    // create an instance of an empty object

var a = {someValue: 125};      // an instance with one property

// Store the data about Julia Roberts
var person = { lastName: "Roberts",
    firstName: "Julia",
    age: 42};
```

Object Methods in Literals

```
var person = {
    "last Name": "Roberts",
    firstName: "Julia",
    age: 42,
    makeAppointment: function () {
        alert("Yakov wants to see  " + this.firstName);
    }
};

person.makeAppointment();
```

Assigning a Function to an Object Property

Declaring and assigning:

```
myCustomer.doTax = function (income, dependents){
   // Do stuff here
}
```

Invoking:

```
myCustomer.doTax(50000, 2);
```

Constructor Functions

In Java, classes have constructors

```
class Tax {

     Tax(float income,
                 int dependents){
         // Do something here
     }
     …
    public static void main(
                    String[] args){

         // Creating 2 Tax objects
         Tax t1 = new Tax(50000f, 3);

         Tax t2 = new Tax(68000f, 1);
    }
}
```

In JavaScript, a function can serve as a constructor

```
function Tax(income, dependents){
         // Do something here
}
     …

// Creating 2 Tax objects
var t1 = new Tax(50000, 3);
var t2 = new Tax(68000, 1);
```

Name constructor functions with capital letters.

Walkthrough 1

- Start IntelliJ IDEA and create a new empty project according to the document *import_code_manual.pdf*. As of IDEA 14, steps 6-8 are nor needed.

- In IDEA Preferences set Google Chrome to be your default Web browser.

- Right-click on *w1.html* from the directory *walkthroughs* and select Open in Browser.

- In Chrome browser open Developer Tools from the Chrome's menu View | Developer.

- Select the tab Sources and click on a little triangle icon on the top left of the Developer Tools panel to see the file names. Open the JavaScript file *w1.js*.

- Put a breakpoint on the line that starts with `var p1` by clicking on the left of this line. Refresh the page. The browser's debugger will stop at this line.

- We want to monitor variables `p1` and `p2`. Add Watch Expressions `p1` and `p2` by clicking on the **+** sign on the right.

- Step through the code  and watch the content of object referenced by `p1` and `p2`. Step inside  the function `marryMe()`. Check the output in the Console view at the bottom.

JavaScript Object Notation (JSON)

```
{
    "fname": "Yakov",
    "lname": "Fain",
    "address": {
        "street": "123 Main St.",
        "city": "New York"
    }
}
```

Methods in Function Objects

doTaxes
is a property of
object Tax

```
function Tax(income, dependents){
    this.income=income;
    this.dependents=dependents;

    this.doTaxes=function() {
       return income*0.05 - dependents*500;
    }
}

// Creating  Tax objects
var t1 = new Tax(50000, 3);

console.log("Your tax is : " + t1.doTaxes());
```

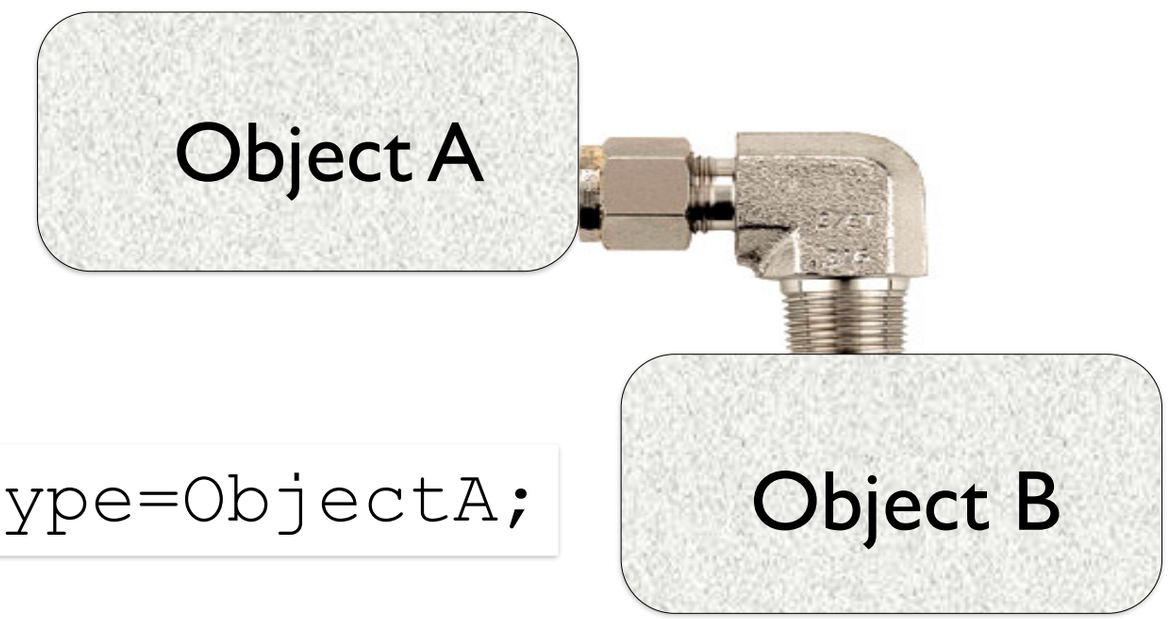
Assigning
anonymous
function to a property

Calling the method
doTaxes()

Prototypal Inheritance

In Java, you can define a blueprint first: class A, and another blueprint based on the first one: class B extends A. After that you can create instances of objects A and/or B.

In JavaScript,an object can inherit from other object via a `prototype` property.




```
ObjectB.prototype=ObjectA;
```

Who's Your Daddy?

Person

```
// Constructor function Person
function Person(name, title){
      this.name=name;
      this.title=title;
      this.subordinates=[];
}
```

Employee

```
// Constructor function Employee

function Employee(name, title){
      this.name=name;
      this.title=title;
}
```

Who's Your Daddy?

Person

```
// Constructor function Person
function Person(name, title){
      this.name=name;
      this.title=title;
      this.subordinates=[];
}
```

Employee

```
// Constructor function Employee

function Employee(name, title){
      this.name=name;
      this.title=title;
}
```

Let's make an Employee a "subclass" of a Person:

```
Employee.prototype = new Person();
```

```
var emp = new Employee("Mary", "Specialist");
```

If a JS engine won't find a property in Employee, it'll keep looking in its prototype chain – Person and Object .

Mozilla has introduced a property __proto__, but it'll become official only in ECMA 6.

Where to Declare Functions

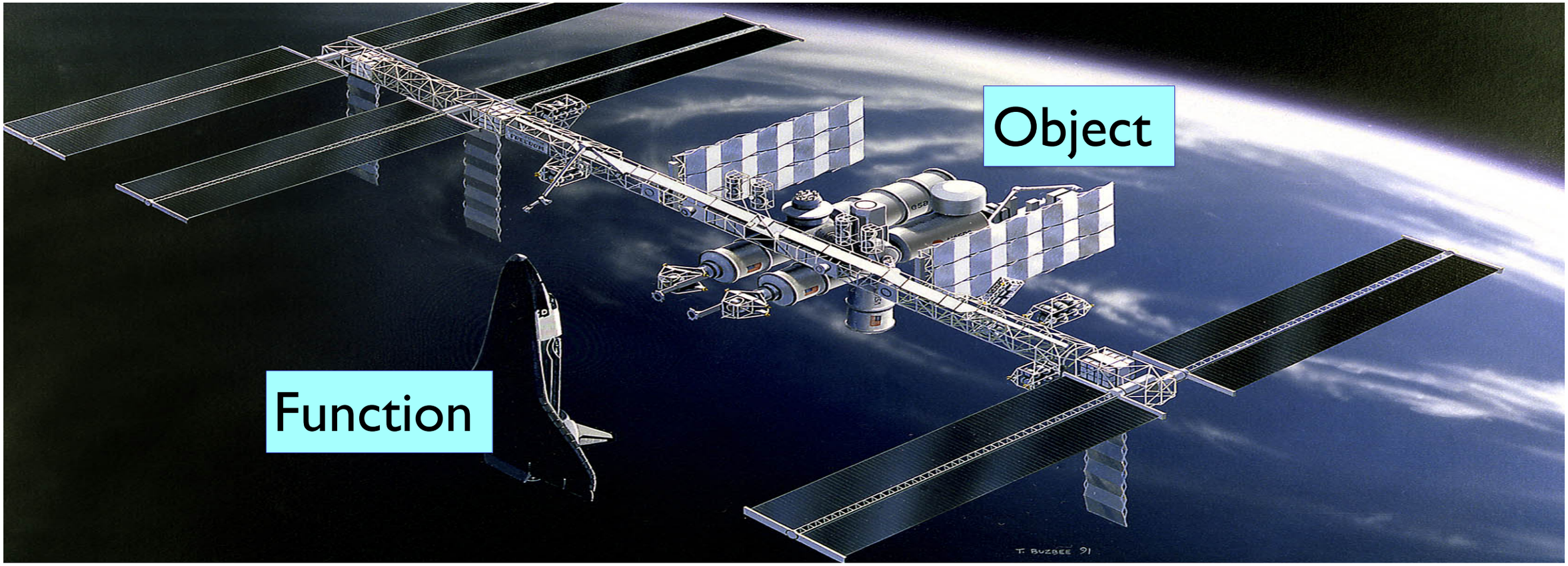
Declare functions on the prototype to avoid code duplication

```
function Person(name, title){
    this.name=name;
    this.title=title;
    this.subordinates=[];
}

Person.prototype.addSubordinate=function(person){
        this.subordinates.push(person);
}

// The code of addSubordinate() won't be duplicated in each instance

var p1=new Person("Joe", "President");
var p2 =new Person("Mary", "CTO");

p1.addSubordinate(p2);
```

Private Variables in Function Objects

```
function Tax(income, dependents){
    this.income=income;
    this.dependents=dependents;

    var mafiaTaxDeduction= 300;   // a private variable of the Tax object

    this.doTaxes=function() {
       return income*0.05 - dependents*500 - mafiaTaxDeduction;
    }
}


// Creating  Tax objects

var t1 = new Tax(50000, 3);

console.log("Your tax is : " + t1.doTaxes());

console.log("Your mafiaTaxDeduction is : " + t1.mafiaTaxDeduction);  // Undefined
```

A function can operate in any object by using call() or apply().



Delegation in action

Every Function Object Has Methods apply() and call()

- apply() – Allows calling any function on any object.
  Parameters are given as an array.

- call() – Allows calling any function on any object.
  Parameters are given as a comma-separated list.

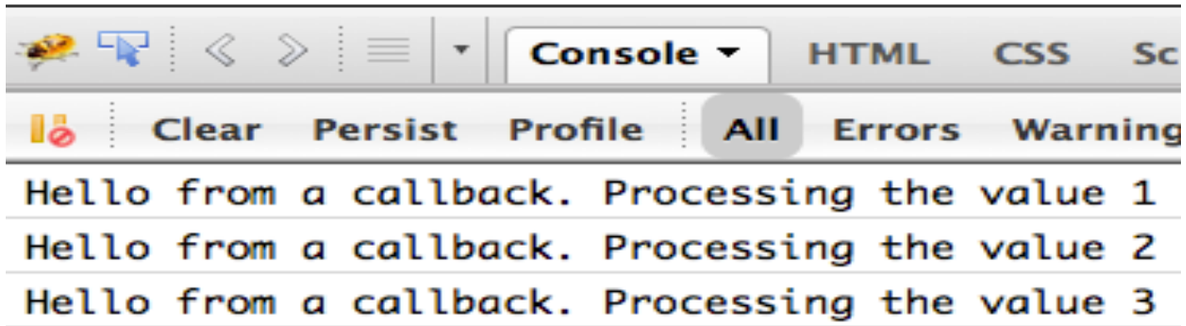
With `apply()` and `call()` you to call an arbitrary function `xyz()` *as if* this function was declared as a method on a specified object (e.g. `myTaxObject`):

```
xyz.call(myTaxObject, 50000, 3);

xyz.apply(myTaxObject, [50000, 3]);
```

Passing a Callback to a Function

1. Declare a function that invokes a given function to process an array

```
function applyHandlersToArray(someArray, someCallBackFunction){
    for (var i = 0; i < someArray.length; i++)

        // Invoke the callback
        someCallBackFunction.call(this, someArray[i]);

    }
}
```



2. Invoke a function providing an array and a function to be called for every element of the array

```
applyHandlersToArray([1,2,3], function(data){
    console.log("Hello from a callback. Processing the value " + data)}
);
```

Closures

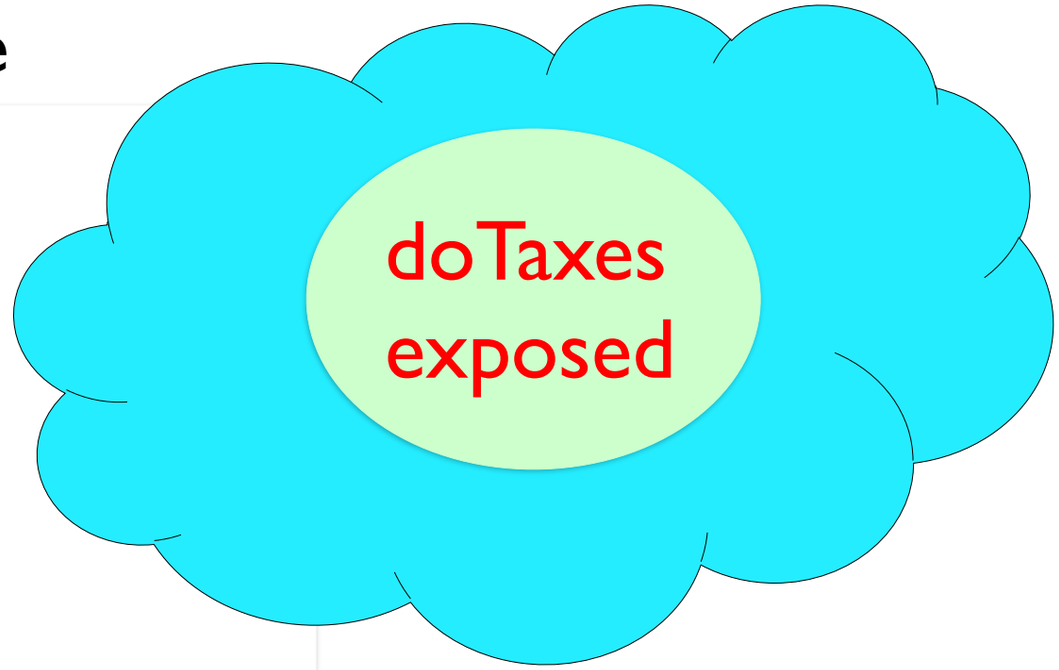
Closure is a function call with strings attached.
A function “remembers” the context it was declared in.

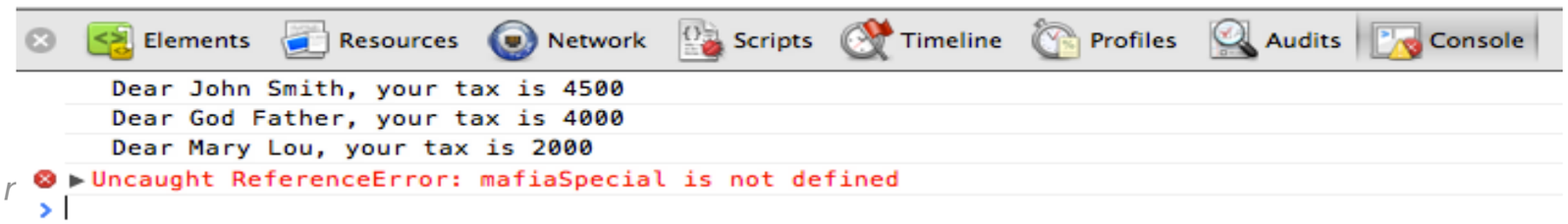
Original image url: http://bit.ly/MYFaXD

Controlling Exposure with Closures

Anonymous function expression creates a scope

```
(function () {                  // this is an anonymous function expression

    var taxDeduction = 500;  // private context to remember

    //exposed closure
    this.doTaxes = function (income, customerName) {

        var yourTax;

        if (customerName != "God Father") {
            yourTax = income * 0.05 - taxDeduction;
        } else {
            yourTax = mafiaSpecial(income);
        }

        console.log("   Dear " + customerName + ", your tax is " + yourTax);
        return yourTax;
    }

    //private function
    function mafiaSpecial(income) {
        return income * 0.05 - taxDeduction * 2;
    }

})();    // Self-invoked function

// calling doTaxes() in the global scope.
doTaxes(100000, "John Smith");  // The closure r

doTaxes(100000, "God Father");
setTimeout(doTaxes(50000, "Mary Lou"), 2);   // Call in 2 seconds
mafiaSpecial();                              // an error - this function is private
```





Returning Closure 1

Transferring data from 127.0.0.1...

Console HTML CSS Script DOM Net Cookies Illuminations cssUpdater

static test2.js prepareTaxes < test2.js

```
1  function prepareTaxes (studentDeductionAmount) {
2
3      return function (income) {
4
5          return income*0.05 - studentDeductionAmount;
6
7      };
8
9  }
10
11 var doTaxes = prepareTaxes(300);    // returns a function that remembers student deduction
12 var yourTaxIs = doTaxes(10000);     // calls a function to calculate tax
13 console.log("Your Tax is " + yourTaxIs); // Returns 10000*0.05 - 500 = 200
```

Watch Stack Breakpoints

New watch expression...

doTaxes	function()
studentDeductionAmount	300
▶ this	Window *index.html*
▼ Closure Scope	Closure Scope { toStri
▶ Window	Window *index.html*

1. Call prepareTaxes(300) that returns **another function** that remembers 300.
2. Call that *returned* function with arg 1000, which will use remembered 300.

FARATA

Returning Closure 2

```
Person.prototype.doTaxes = function () {

    var taxDeduction = 500;

    //private function
    function mafiaSpecial(income) {
        return income * 0.05 - taxDeduction * 2;
    }

    //exposed function is returned as a value to the caller
    return function (income) {

        var yourTax;

        if (this.name != "God Father") {
            yourTax = income * 0.05 - taxDeduction;
        } else {
            yourTax = mafiaSpecial(income);
        }

        console.log("   My dear " + this.name + ", your tax is " + yourTax);
        return yourTax;
    }
}();
```

```
function Person(name){

    this.name = name;

}
```

```
//Using closure
var p1=new Person("John Smith");
p1.doTaxes(100000);

var p2=new Person("God Father");
p2.doTaxes(100000);
```

Monitoring Closures in Chrome

Elements | Resources | Network | Scripts | Timeline | Profiles | Audits | Console

ControllingExposure2.js

```
1  // Athother way of using this closure
2  function Person(name){
3
4          this.name = name;
5
6  }
7
8  Person.prototype.doTaxes= function(){
9
10     var taxDeduction = 500;
11
12       //private function
13       function mafiaSpecial(income){
14           return income*0.05 - taxDeduction*2;
15       }
16
17       //exposed function
18       return function(income) {
19
20         var yourTax;
21
22         if (this.name != "God Father"){
23           yourTax =   income*0.05 - taxDeduction;
24         } else{
25           yourTax =   mafiaSpecial(income);
26         }
27
28          console.log( "   My dear " + this.name + ", your tax is "+ yourTax);
29          return yourTax;
30       }
31
32 }();
33
34 var p1=new Person("John Smith");
35 p1.doTaxes(100000);
36
37 var p2= new Person("God Father");
38 p2.doTaxes(100000);
39
40 mafiaSpecial();  // will give an error - this function is private
```

Watch Expressions

Call Stack

(anonymous function)

(anonymous function)

Pau.

Scope Variables

Local

income: 100000

this: Person

yourTax: undefined

Closure

mafiaSpecial: function mafiaSpecial(income){

taxDeduction: 500

Global

Breakpoints

ControllingExposure2.js:22

if (this.name != "God Father"){

DOM Breakpoints

XHR Breakpoints

Event Listener Breakpoints

Workers

FARATA

Walkthrough 2

- In IntelliJ IDEA right-click on the file *w2_1.html* from the folder *walkthroughs* and select Open in Browser.
- Open Chrome Developer Tools from the Chrome's menu View | Developer.
- Select the tab Sources and hit a little triangle on the top left of the Developer Tools panel to see the file names. Refresh the page. Open and review the *w2_1.js*. This is an *exposing closure* example.
- Review the output in the Console tab.
- Repeat the above with *w2_2.html* and *w2_2.js* (this is a *returning closure* example).

AJAX

Requesting data and updating an HTML element without the page refresh

```
function loadData(dataUrl, target) {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', dataUrl, true);
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
            if((xhr.status >=200 && xhr.status <300) ||
                xhr.status===304){

                target.innerHTML += xhr.responseText;
            } else {

                console.log(xhr.statusText);
            }
        }
    }
    xhr.send();
}
```

JavaScript in a Web Browser

Tipically we place <script> tags at the bottom of HTML page.

Some Properties of the `window` Object

location - an information about the window's current location

document - a reference to the Document object that provides access to all HTML elements in a Web page

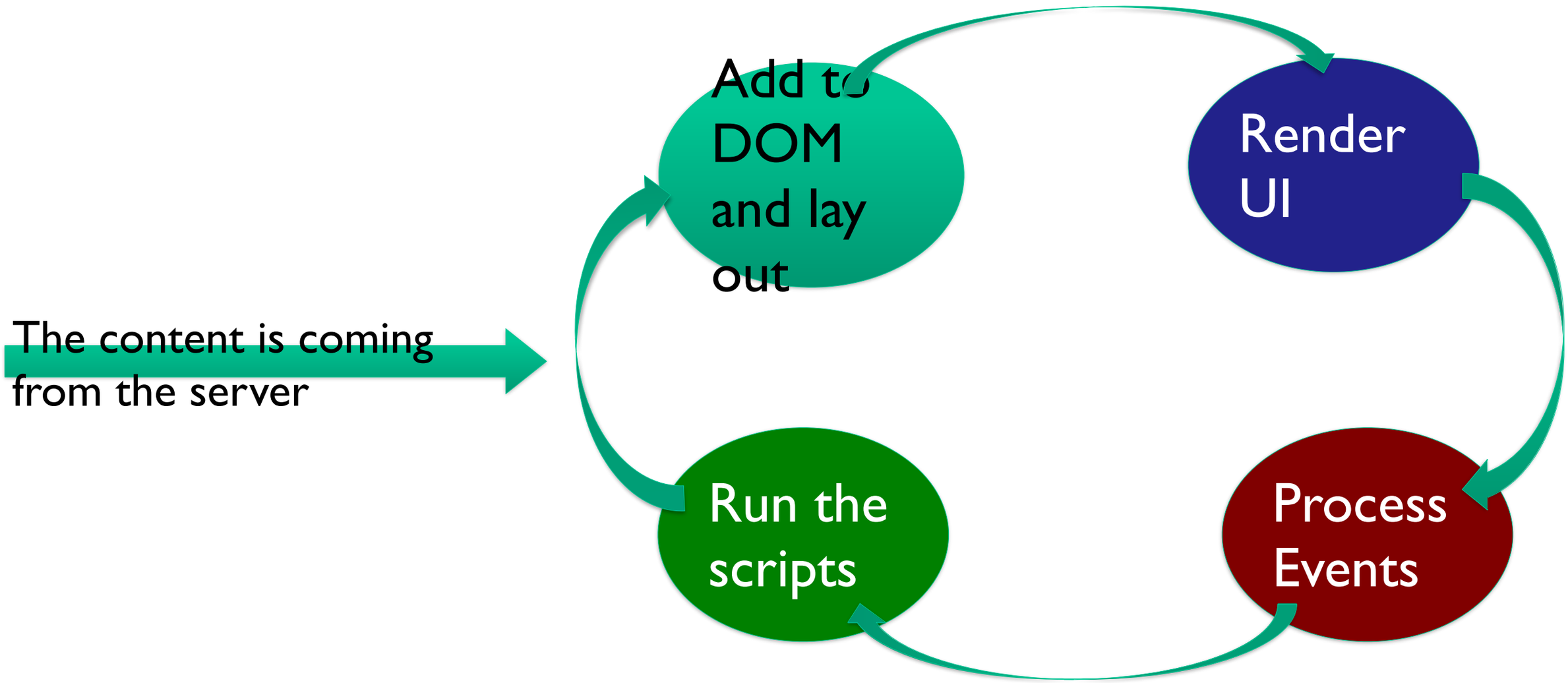
opener - a reference to the parent window that opened the current popup window

parent - a reference to the parent of a frame or iFrame

cookie - a reference to all name/value pairs of cookies in the document

```
location=''http://yahoo.com";        // redirects your window to Yahoo's home page

location.reload();                   // reloads the current page

var childWindow = open("xyz.html"); // opens a new browser child

childWindow.moveTo(200,300);         // moves the window to x=200, y=300

childWindow.close();                 // closes the window
```

Web Browser's Circle



Working with DOM

document.getElementById(id) – get a reference to HTML element by unique identifier

document.getElementsByTagName(tagname) - get a reference to one or more elements by tag name, like a reference to all elements.

document.getElementsByName(name) - get a reference to all elements that have specified value in their name attribute.

document.getElementsByClassName(className) – get a reference to all elements to use specified CSS class.

Selectors API

```
document.querySelector(cssSelector)     - Find the first element that matches
                                          a CSS selector string.

document.querySelectorAll(cssSelector) - Find all elements by a CSS selector string.
```

These methods allows using more complex CSS selector strings than `getElementById(id)` or `getElementsByTagName(tname)`.

Selecting an element by ID and changing its value

```
<html>
   <body>
   The employee of the month is <span id="emp">...</span>

    <script>
          function setEmployeeOfTheMonth(){
                var mySpan = document.getElementById("emp");

                mySpan.firstChild.nodeValue= “John Smith”;
          }
     </script>
  </body>
</html>
```

Using Styles

HTML use CSS class selectors, and you can control styles programmatically in JavaScript.

The styles can be either embedded in your HTML page in using <style> tag or loaded from the external .css file using the <link> tag:

```
<head>
    <link rel="stylesheet" type="text/css" href="mystyles.css">
</head>
<body>
      <div id="billingInfo" class="niceStyle">….</div>
</body>
```

```
.niceStyle{
          font-family: Verdana;
          font-size:large;
          font-style:italic;
          color:gray;
          background-color:green;
  }
```

Changing Styles in JavaScript

To find elements that use specific class selectors use getElementsByClassName(), which returns a NodeList of matching elements.

```
document.getElementsByClassName("niceStyle");
```

To change the selector on the HTML element, use the attribute className:

```
document.getElementsByClassName("niceStyle")[0].className=“badStyle”;
```

Web Browser's Events

Browser dispatches events: load, mousemove, click, keydown etc.

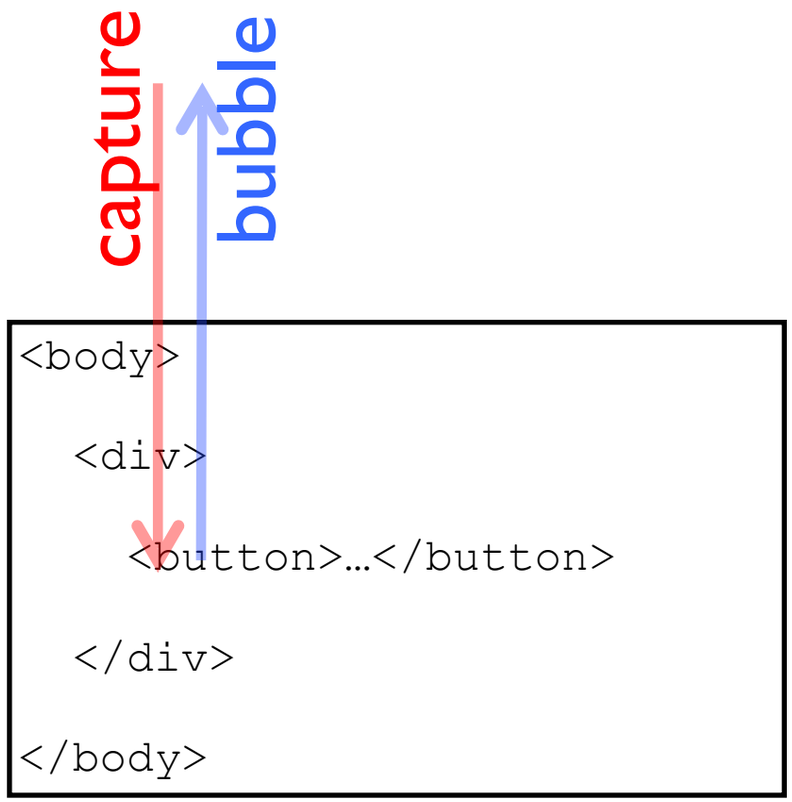
An event handler (a.k.a. listener) is a function you want to be called as a response to this event.

```
//Declaring an event handler in HTML
<button id="myButton" onclick="myFunctionHandler(event)">
          Click Me
</button>

//Declaring an event handler in JavaScript
<script>
  window.onload=function(){ … }
  myButton.click=myFunctionHandler;
</script>
```

Events Phases

Capturing phase is when event object travels to the target from the top most container.

Bubbling phase is when event object from target up through all enclosing containers.



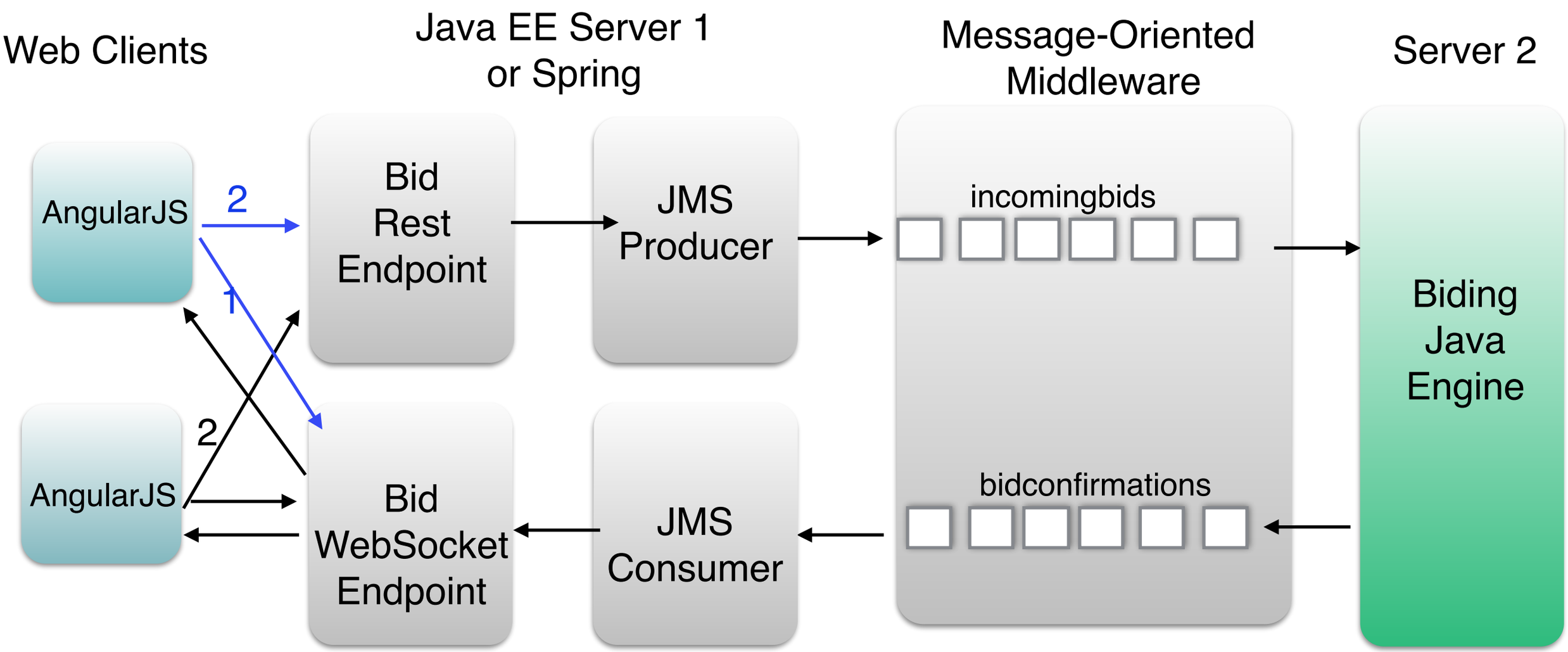
Registering an event listener in JavaScript

```
// bubbling phase - third arg is false - it's false by default
myButton.addEventListener("click", myHandler, false);
```

Removing an event listener:

```
myButton.removeEventListener("click", myHandler, false);
```

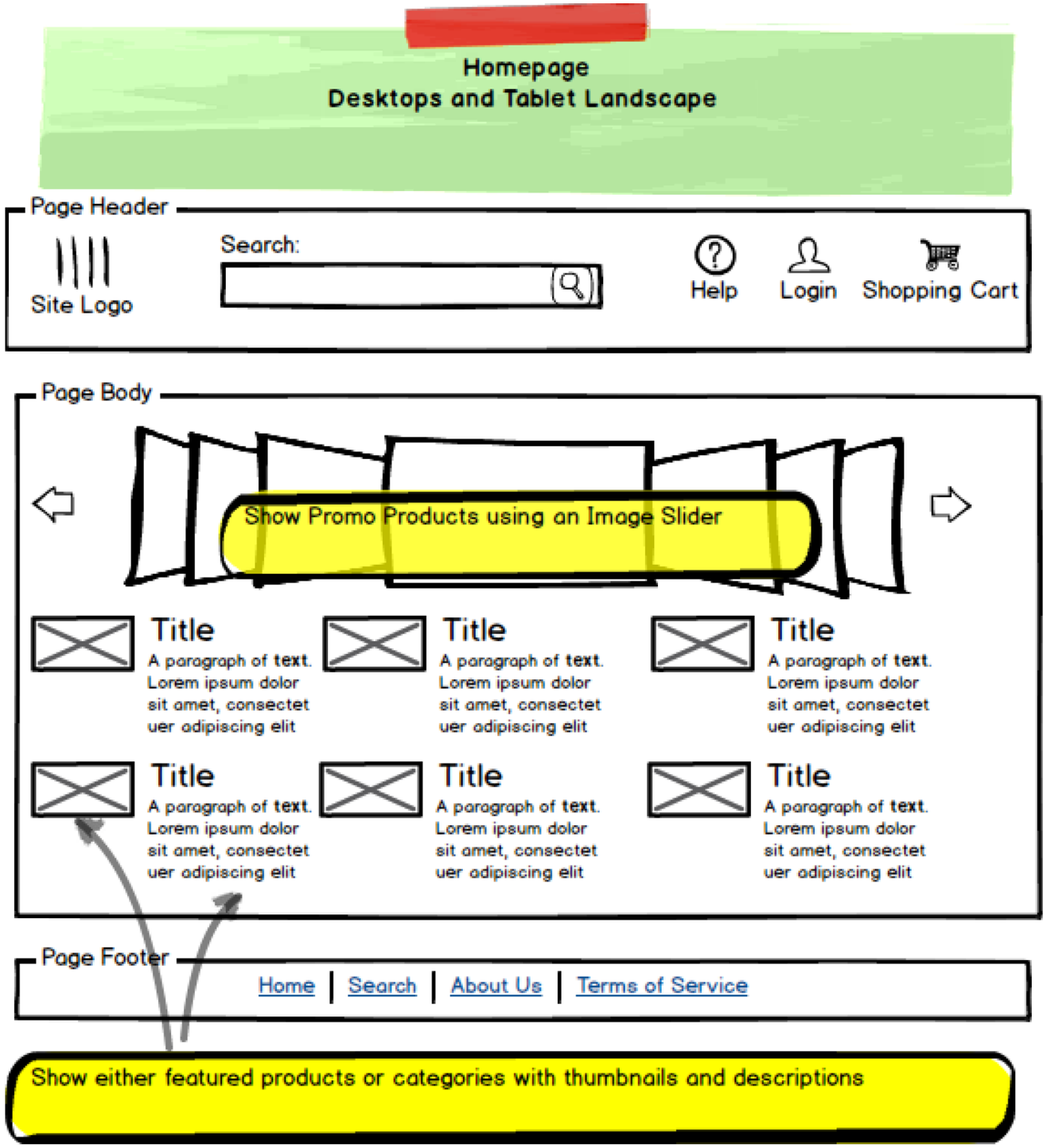
Sample Online Auction Architecture: AngularJS, REST API, WebSockets, and JMS



In this master class we'll develop just the Web client for the auction

Online Auction Mockups

We use Balsamiq Mockups



Walkthrough 3

- Start working with the **ABC Auction** prototype. In IntelliJ IDEA open in browser the file v0/*index.html*. This version has the HTML elements required by the mockup, but there is no layout yet.

- Review the code of the file *index.html*.

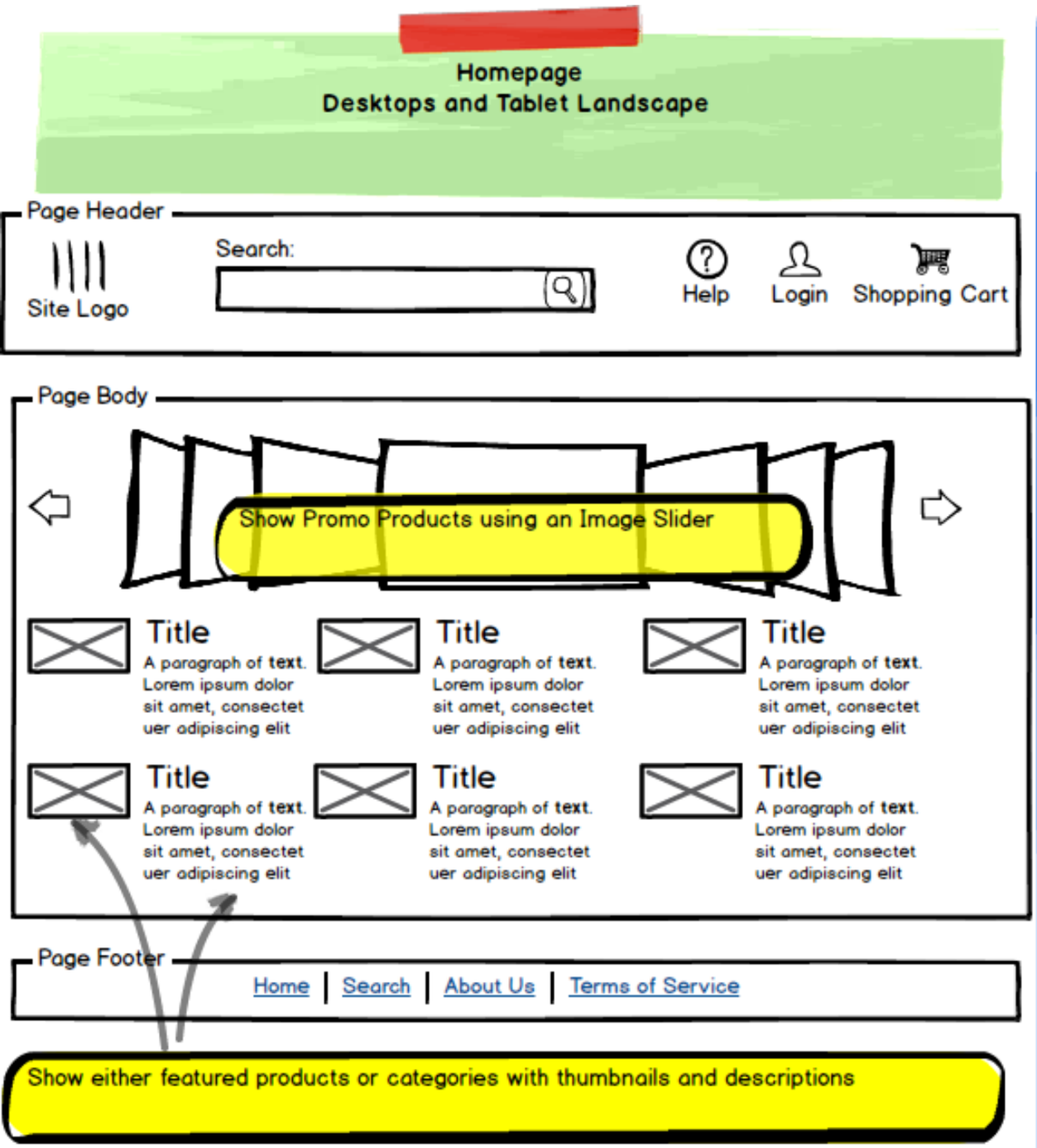
- In IntelliJ IDEA open the code of v1/*index.html*. This version adds CSS file for layout.

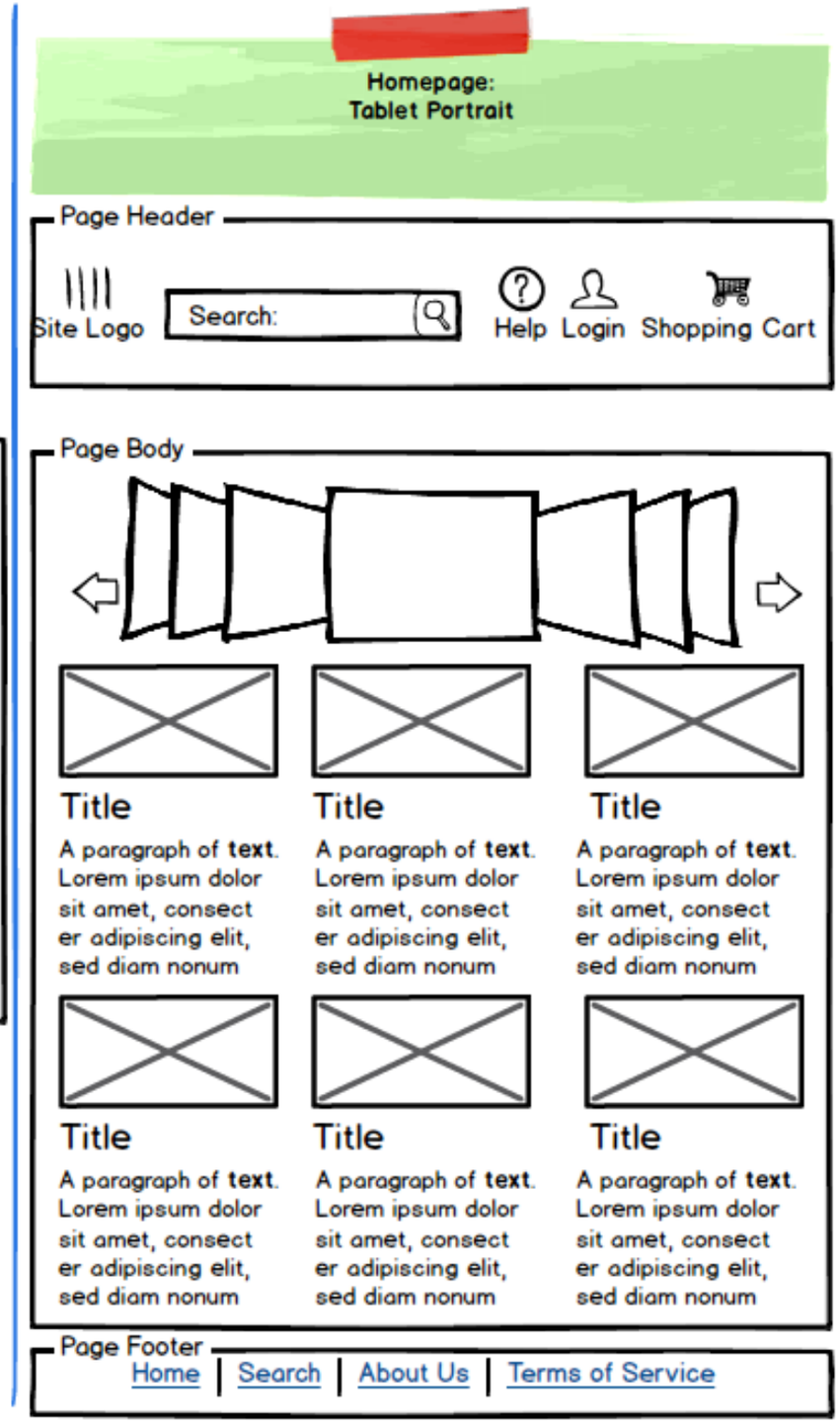
- Review the code of CSS. Note the line 6 of *index.html* that includes the CSS file:
  <link rel="stylesheet" href="styles.css">

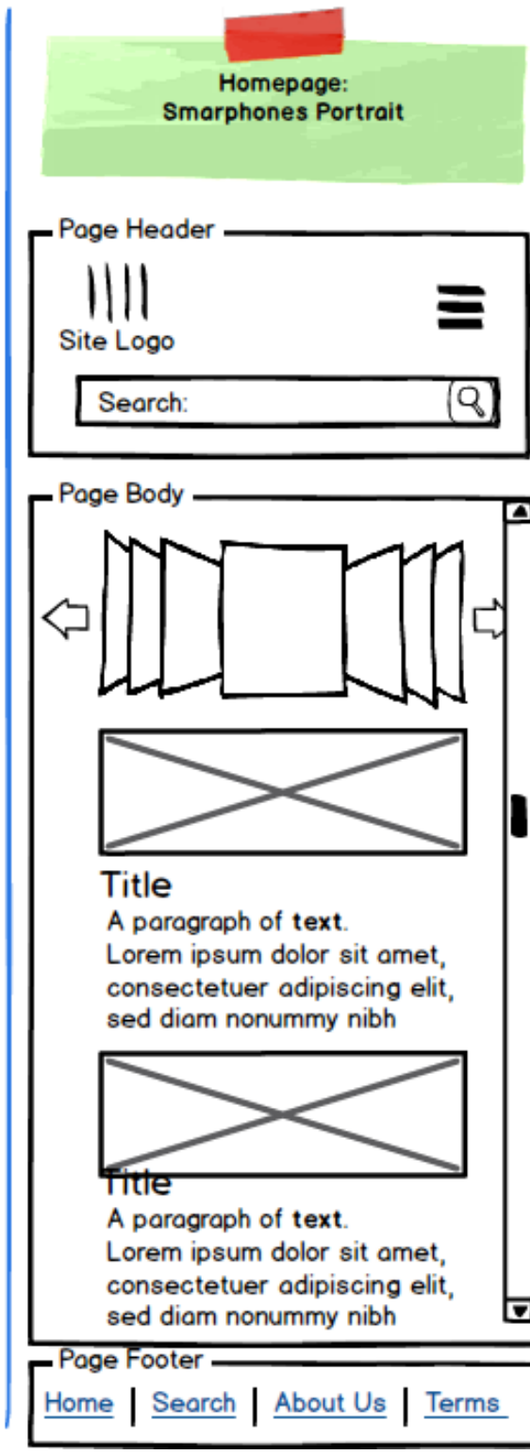
- Grab the lower right corner of the browser's window and drag it to the left to make the window smaller. The browser adds scrollbars, and shows **partial** page content.

- How the UI will look on mobile devices with smaller screens?

Back to Mockups. Mobile First?







Responsive Web Design (RWD)

- Separate versions for desktop and mobile?
- How many versions of the UI to create?
- Can we have a single HTML version for all devices?
- Try Boston Globe changing the window width: http://www.bostonglobe.com/
- BYOD - bring your own device.

CSS Media Queries

- Specify the layout in a CSS file based on the viewport width:

```
@media screen and (max-width:768px) …

@media screen and (max-width:640px) …
```

- How many breakpoints I need? Set the breakpoint when your screen design breaks.

- HTTP header has attribute User-Agent to identify the client, but there could be thousands of possible values there (see http://useragentstring.com).

- The showcase of Web sites that use Media Queries is here: http://mediaqueri.es.

Walkthrough 4

- Open in browser the file *walkthroughs/w4.html*. Change the window width and observe the change of the layout.

- Review the code of the file *css/w4_style.css*. Note the properties `float:left` and `float:none`.

RWD Pros and Cons

- RWD is good for publishing sites. Mobile frameworks can be a better choice for interactive apps.
- RWD allows to have a single app code base.
- Mobile versions of an app may need limited functionality and specific navigation.
- RWD means larger traffic (heavy CSS) – no good for slower connections.
- Mobile frameworks offer more native look of the UI.

FARATA

Walkthrough 5

- Introducing media queries into ABC Auction. Open in Web browser *v2/index.html* .
- Drag the browser's window to make its width smaller. Observe changes in the layout.
- Review the code of the file *v2/style.css*.
- Open in the browser *v3/index.html*. It added images in the *assets* folder.

Twitter Bootstrap Framework

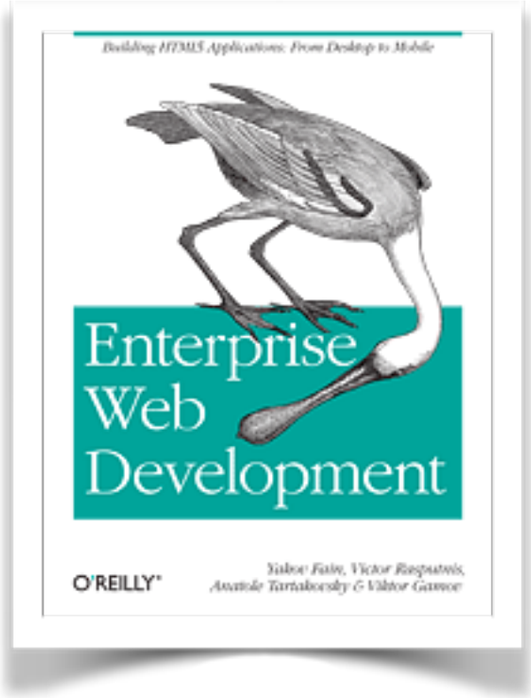
- Bootstrap is a lightweight framework with nicely styled responsive HTML components.
- It's available for download at http://getbootstrap.com/.
- You can download only a subset of components by using the menu *Customize*. But get the entire library at http://getbootstrap.com/getting-started/#download.
- You'll need to unzip the .zip file add to your project the bootstrap.min.css(113Kb) and bootstrap.min.js (36Kb).
- You'll need to download compressed jQuery library from http://jquery.com and add to your project jQuery-2-1-1.min.js (84KB).

Walkthrough 6

- We'll add images and introduce the Bootstrap framework into the online auction.
- Review the content of the folders *v4/assets/img* and *v4/assets/js*.
- Open the file *v4/index.html* in Web browser.
- Resize the browser's window to make its width smaller. Observe the responsive page layout behavior.
- Review the code. Note the three **div** sections in the index.html styled with Bootstrap's classes *navbar*, *container*, and *footer*. The classes *row* and *col-md-4* are needed for the responsive fluid grid layout (see http://getbootstrap.com/css).
- Make the browser's window narrower. Note the behavior of the navigation toolbar.
- Open in the browser *v5/index.html*. It adds the Bootstrap's Carousel component (look for the CSS class selector `carousel` in index.html, line 46). You can read about the Carousel at http://bit.ly/1dgQWLa.

Additional Materials

- Read the following chapters from our book “Enterprise Web Development”:
  Intro to JavaScript (bonus chapter): http://oreil.ly/1i2S5e6
  Chapter 1 (Prototyping), Chapter2 (AJAX), Ch10 (Responsive Web Design).



- Watch our video about Git/GitHub: http://bit.ly/1iDpOKp
- Watch a video tutorial on basics of Bootstrap framework:
  http://bit.ly/1fsNrWd
- Bootstrap Grid System: http://getbootstrap.com/css
- HTML templating with Handlebars: http://handlebarsjs.com
- Customizing Bootstrap: http://bit.ly/1qljzu6
- How Web Browsers Work: http://bit.ly/how-browsers-work

Project Review

- Using HTML, JavaScript, and Bootstrap framework develop a prototype of the Search Results Web page based on the provided mockup in the file named *SearchResults.png*. Attach the click event handler to the button Search so it'll show the content of the Search Results page.

- Our Auction is a single-page app, so click on the button should use an AJAX call to load the content .

- Use the provided JSON files *featured-product.json* and *search-results.json* to populate the Home and Search Results pages.

- .Review a proposed solution at http://farata.github.io/modernwebdev-showcase/homework2 . Note the use of the Handlebars templates.