JavaScript for Java Developers

Yakov Fain

Farata Systems, USA

@yfain

DEVOXX™

the java™ community conference

Simple things that Java developers take for granted (a VM version or names of the class methods) aren't available in a tribe called *JavaScript Developers*.

About Myself

- Work for IT consultancy Farata Systems
- Java Champion
- Authored several technical books.
- Working on "Enterprise Web Development: From Desktop to Mobile", for O'Reilly. Read its current draft at EnterpriseWebBook.com
- Blogging at: yakovfain.com and blog.faratasystems.com

Why 17 year old JavaScript is Hot Again?

Hope to Write Once, Run Anywhere.

Where to Run JavaScript?

- In Your Web browser
- Google's V8 JavaScript Engine exists on the server (Node.js runs there)
- Java includes Java Scripting API and the utility *jrunscript*
- JDK SE 8 will include JavaScript engine Nashorn.

JavaScript arrives to the place of execution as text.

There's no compiler helping developers with syntax errors.

Users may have different runtime environment.

Imagine writing a Java program that must work in any version of JVM.

Selected IDEs Supporting JavaScript

- Eclipse for Java EE Developers with VJET Plugin
- WebStorm and IntelliJ IDEA
- Aptana Studio - Eclipse-based IDE
- NetBeans - Project Easel
- DreamWeaver
- Visual Studio

Debugging JavaScript

- Firefox: add-on FireBug
- Chrome: Developer Tools
- Internet Explorer: F12 Developer Tools
- Safari: the menu Develop
- Opera Dragonfly
- NetBeans 7.3 (see Project Easel)

JavaScript in the Web Page

Place `<script>` tags at the bottom of HTML page.

But some JavaScript frameworks want to live in the `<head>` section.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/h
<html>
<head>
    <title>Hello Ext</title>

    <link rel="stylesheet" type="text/css" href="extjs/resources/css/ext-all.css">
    <script type="text/javascript" src="extjs/ext-debug.js"></script>

    <script type="text/javascript" src="djn/djn-remote-call-support.js"></script>
    <script type="text/javascript" src="ejn/ejn-assert.js"></script>
    <script type="text/javascript" src="clear/direct/ServerConfig.js"></script>
    <script type="text/javascript" src="app/direct/ServerConfig.js"></script>
    <script type="text/javascript" src="app.js"></script>
</head>
<body></body>
</html>
```

index.html of a typical Ext JS application

Variables

Declaring a variable (unless in strict mode) 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.

Java classes can have *methods*. JavaScript has no classes as of yet.

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

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

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);      // you can pass more/less parameters
var myTax = calcTax(50000,2);
```

Declaring and invoking at the same time:

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

Function Expressions

Assigning a function literal to a variable:

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

Invoking a function:

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

Assigning a Function to the Object Property

Declaring:

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

Invoking:

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

Objects vs. Classes

In Java, you define a class and then create its instance(s).

In JavaScript, you 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

Object Literals

An object is an unordered 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};
```

Accessing Object Properties

```
var person = {
        “last name”: ”Roberts”,    //”last name” is a valid key
        firstName: ”Julia”,
              age: 42};
herName=person.lastName;           //  herName is undefined

herName=person[“last name”];        //   herName is “Roberts”

person.salutation=“Mrs.”;          // create and initialize a
                                   // new property salutation

// Display “Hello Mrs. Roberts”
alert(“Hello ”+ person.salutation + “ “ + herName);
```

Object Methods in Literals

```
var person = {
      “last Name”: ”Roberts”,
      firstName: ”Julia”,
      age: 42,

      makeAppointment: function(){
         alert(“Yakov wants to see  ” +
                              this.firstName);
      }
};

person.makeAppointment();
```

Nested Objects

```
var person = {
        lastName: ”Roberts”,
        firstName: ”Julia”,
        age: 42,

        makeAppointment: function(){
            alert(“Yakov wants to see you ” + firstName);
        },

        phones:{
            work: “212-555-1212”,
            cell:    “212-000-0000”
        }
};

var phone=person.phones.work          //  “212-555-1212”

herAddress=person.address;            //  herAddress=undefined

herAddress=person.address.street;     // throws  TypeError
```

JavaScript Object Notation (JSON)

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

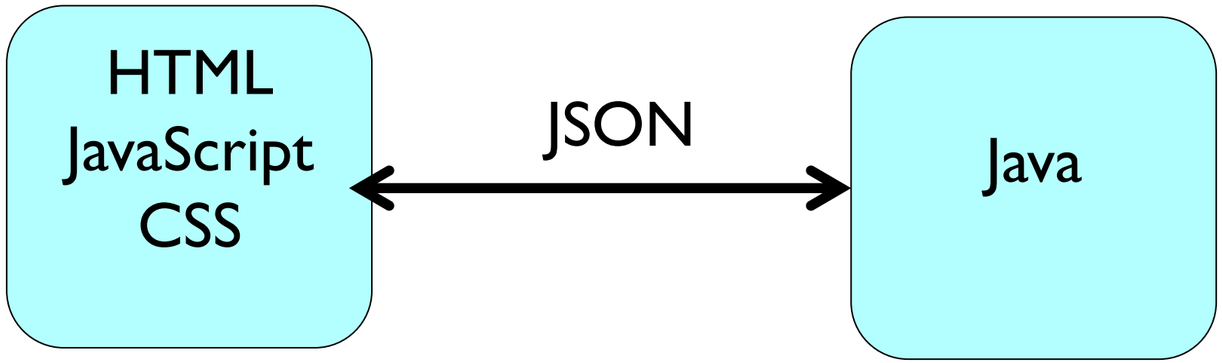
JSR 353 for JSON Processing in Java, final release - Apr 2013 .

Streaming and Object model APIs.

This JSR is targeted for Java SE 6 or higher and Java EE 7 or higher platforms.

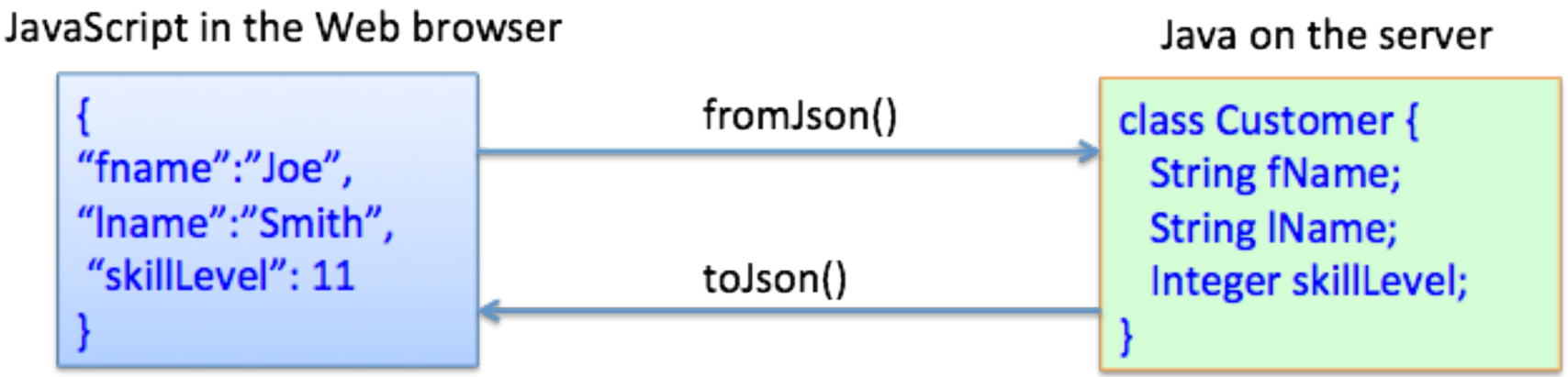
Expect Data Binding JSON-B (JSON text to/from Java Objects) two years later.

Separating JavaScript and Java



Try to remove any presentation code from your JSPs

Google's Gson (gson.jar)




```
// Parsing JSON in JavaScript

var customer=
       JSON.parse(myJSONString);
```

```
//Parsing JSON String in Java using gson

public  Customer createCustomerFromJson(
                                String jsonString){

    Gson myGson = new Gson();
    Customer cust = myGson.fromJson(jsonString,
                                    Customer.class);
    return cust;
}
```

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 play a role of 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);
```

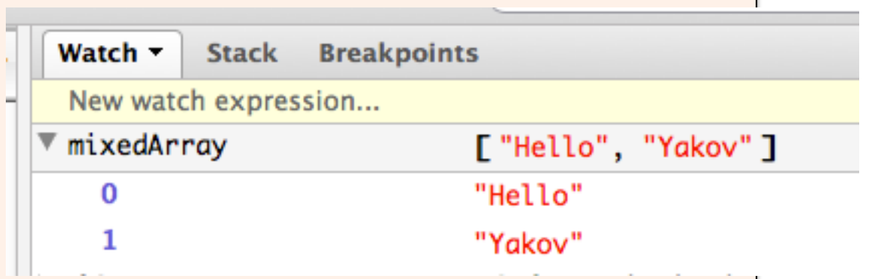
Name constructor functions with capital letters.

Array is a Grab Bag of Stuff

```
var emptyArray=[];

var states = [“NJ”, “NY”, “CT”, “FL”];

var states = new Array(4);  // size is optional
states[0]=“NJ”;
states[1]=“NY”;
states[2]=“CT”;
states[3]=“FL”;
```

This `mixedArray` will store two strings after the `promt()` will be executed during initialization:

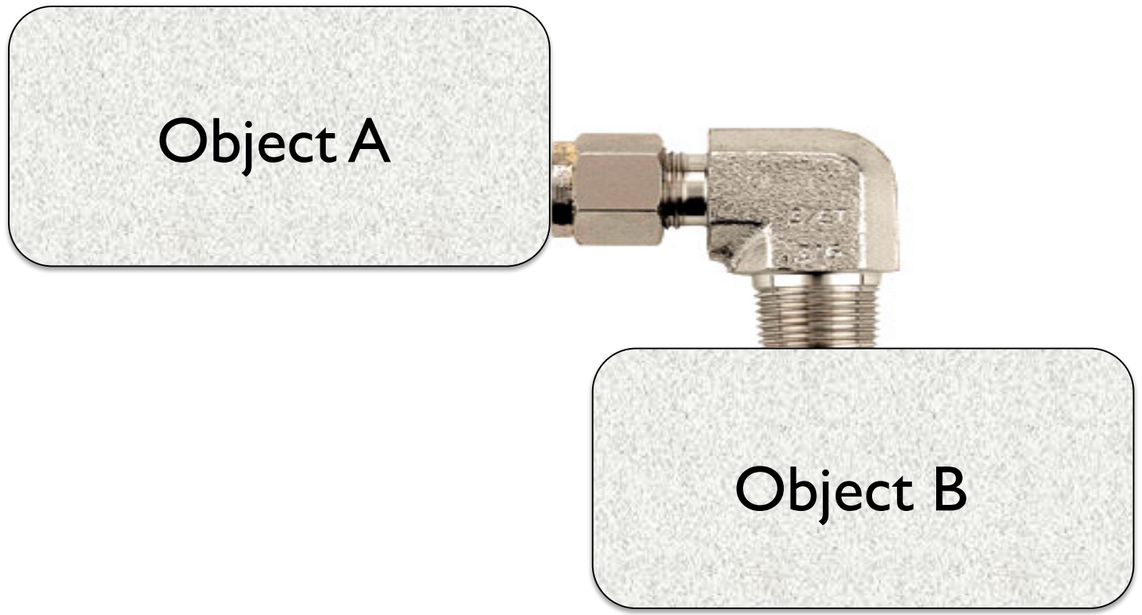
```
var mixedArray=[
    "Hello",
    prompt("Enter your name", ”Type your name here")
];

alert(mixedArray.join(","));
```



Prototypal Inheritance

In Java, you 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 A and/or B.

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




```
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;
}
```

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

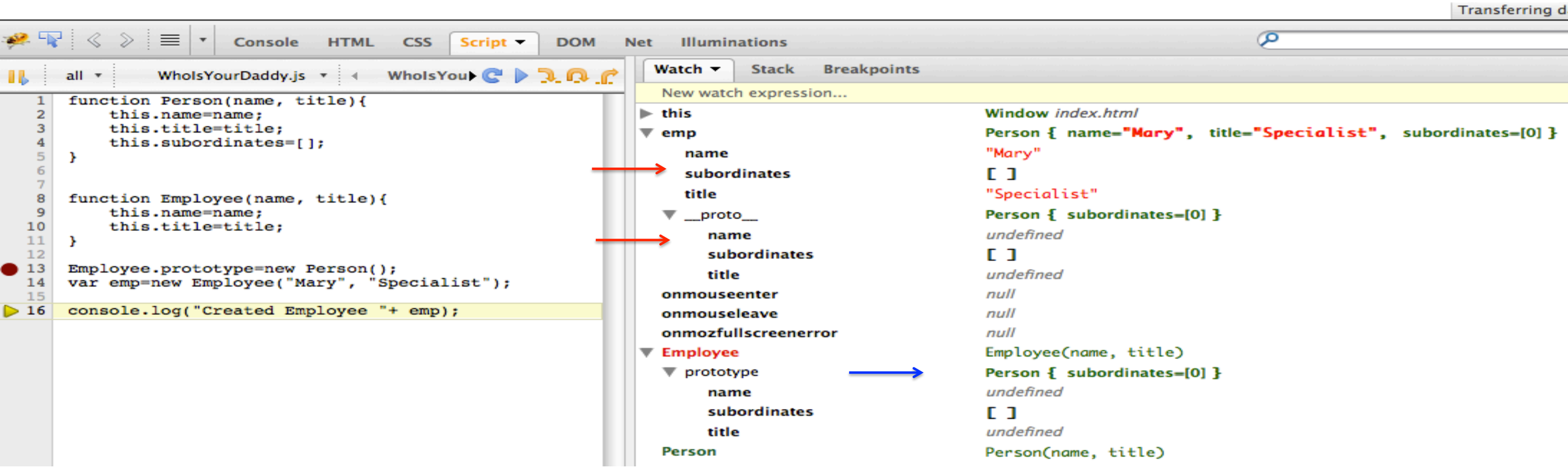
```
Employee.prototype = new Person();
```

```
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.

Watching Employee and Person in Firebug



→ Person is a prototype of all Employees

→ Redundant definitions of name and title

Objects can have properties and methods, right?

A function is an object, right?

Hence…

Functions can have properties and methods!

Methods in Function Objects

doTaxes
is a property of
object Tax

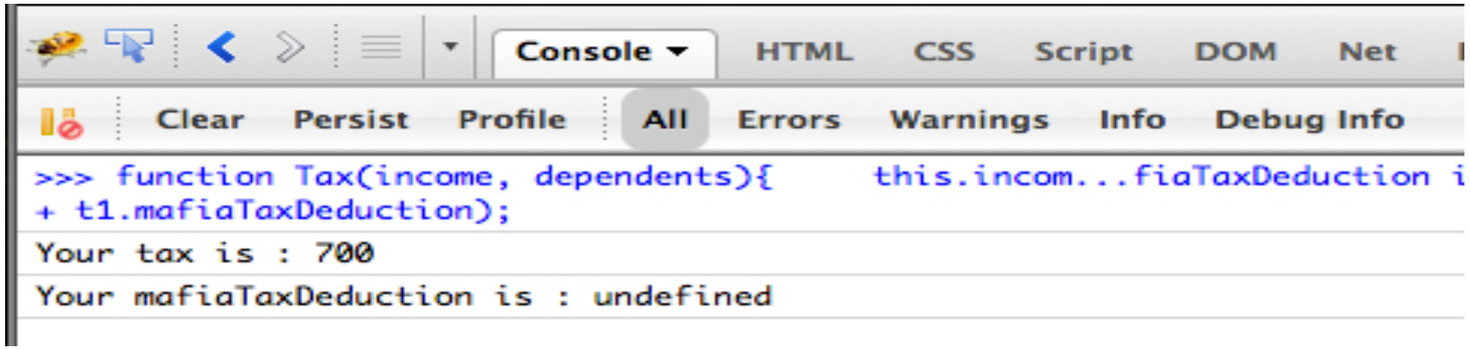
```
function Tax(income, dependents){
    this.income=income;
    this.dependents=dependents;

    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 an anonymous function to the object's property doTaxes

Calling the method doTaxes

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
```



Where Not to Declare Methods

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

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

// The code of addSubordinate() is duplicated in each instance
var p1=new Person("Joe", "President");
var p2 =new Person("Mary", "CTO");

    p1.addSubordinate(p2);
```

Where to Declare Methods

```
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() is not duplicated in each instance
var p1=new Person("Joe", "President");
var p2 =new Person("Mary", "CTO");

p1.addSubordinate(p2);
```

Declaring methods in the function prototype prevents code duplication.

Method Overriding

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

Person.prototype.addSubordinate=function(person){

this.subordinates.push(person);
}

var p1=new Person("Joe", "President");
    p1.addSubordinate=function(person){
      // some other code goes here
}
```

Overriding happens **after** the object was created.

Override toString() to return a String representation of your object.
By default toString() prints [object Object] .

Function Overloading Made Easy

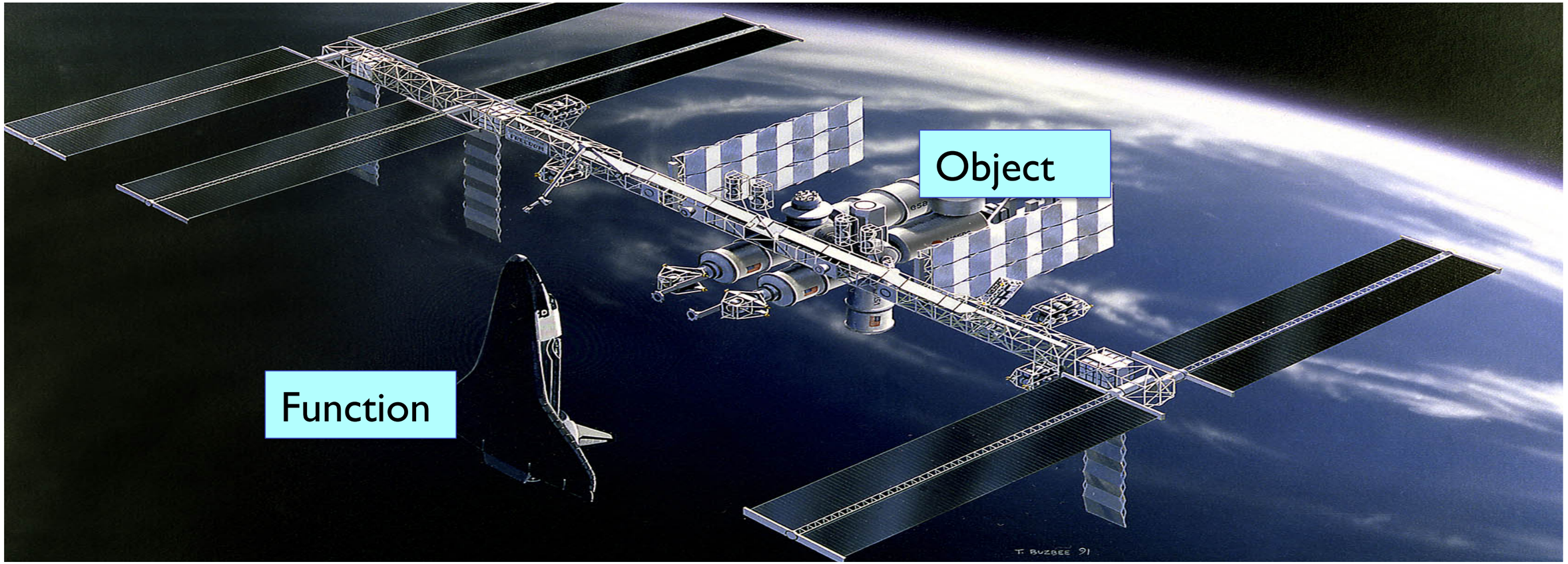
Java

```
class Tax{
  …
  float calcTax(int descendents,
                              float income){

    // by default, calculate tax for New York
     calcTax( descendents, income,  “NY”);
  }

  float calcTax(int descendents,
          float income, String state){

   // Your code for calculating tax goes here

   }
…
}
```

JavaScript

```
function calcTax(descendents, income, state){

    if (!state){
      state="NY”;
      console.log(“Using default state: NY");
    }
   // …
}

calcTax(2,50000); // missing 3rd parameter
```

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



Delegation in action

Every Function Object Has These Functions

- 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`):

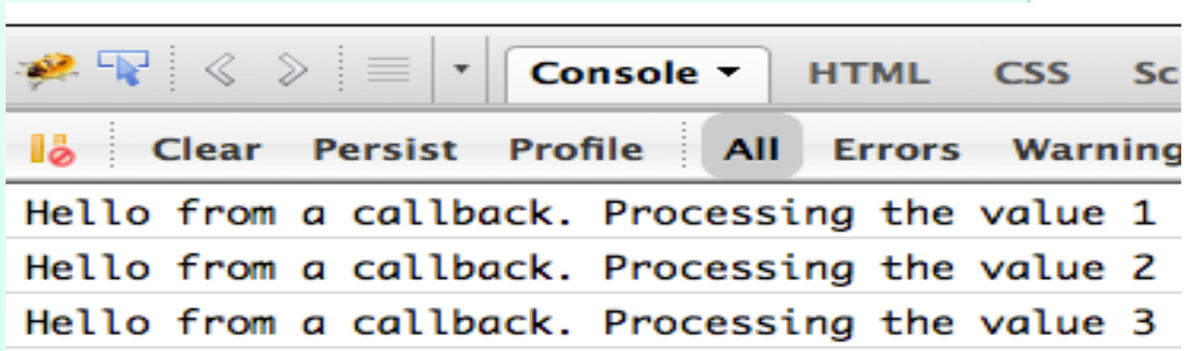
```
xyz.call(myTaxObject, 50000, 3);

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]);

    }
}
```



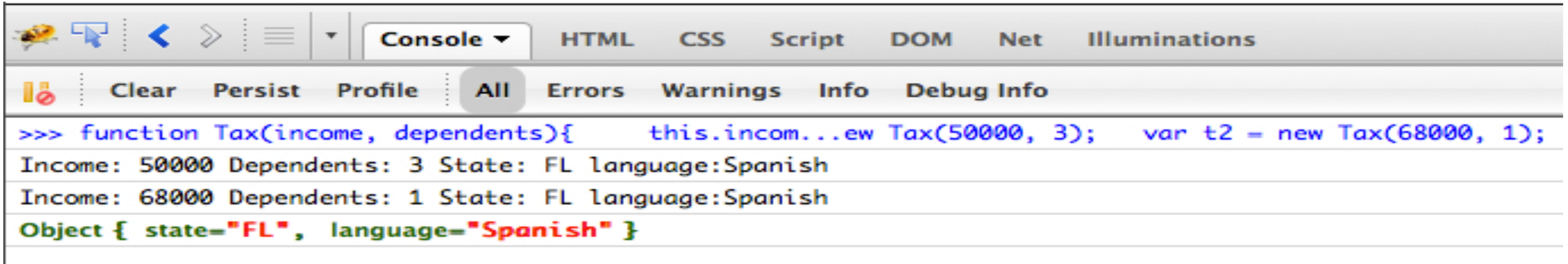
2. Invoke a function providing an array and a function
   to be called for every element of the array

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

Function Properties as Static Variables

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

    console.log("Income: " + this.income + " Dependents: "+ this.dependents
         + " State: " + Tax.defaults.state + " language:" + Tax.defaults.language);
}

Tax.defaults={
     state:"FL",
     language:"Spanish"
};

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

Function properties can serve as cache for some values for better performance (e.g. for browser's DOM elements).



Anonymous Classes vs. Callbacks

Java Swing and inner classes

```
myButton.addActionListener(
     new ActionListener() {
        public void
          actionPerformed(ActionEvent e) {
          // Process the button click event
        }
      }
);
```

JavaFX and inner classes

```
myButton.setOnAction(new
           EventHandler<ActionEvent>({
            public void handle(ActionEvent event){
                // Process the button click
            }
        });
```

JavaScript anonymous functions as callbacks

```
myButton.addEventListener("click",
     function(){
        //Process the button click event
     }
);
```

Scope: this

```
var taxDeduction=300;        // global var

var myTaxObject = {

    taxDeduction: 400,     // object property

    doTaxes: function() {
         this.taxDeduction += 100;

         var mafiaSpecial= function(){
           console.log( “Will deduct " + this.taxDeduction);
         }

         mafiaSpecial();  // invoking as a function
    }
}

myTaxObject.doTaxes();  //invoking method doTaxes
```

300

What value of taxDeduction will this code print?

Scope: Who's this now?

```
var taxDeduction=300;        // global var

var myTaxObject = {

    taxDeduction: 400,          // object property

    doTaxes: function() {
         var thisOfMyTaxObject=this;
         thisOfMyTaxObject.taxDeduction += 100;

         var mafiaSpecial= function(){
           console.log( "Will deduct " + thisOfMyTaxObject.taxDeduction);
         }

         mafiaSpecial(); // invoking a function without obj. in front
                         // makes it use global this ☹
    }
}

myTaxObject.doTaxes();  //invoking method doTaxes
```

500

What value of taxDeduction will this code print?

Scope: this again

```
var taxDeduction=300;              // global var

var myTaxObject = {

    taxDeduction: 400,             // object property

    doTaxes: function() {
         this.taxDeduction += 100;

         var mafiaSpecial= function(){
           console.log( “Will deduct " + this.taxDeduction);
         }

         mafiaSpecial.call(this);
    }
}

myTaxObject.doTaxes();  //invoking method doTaxes
```

500

What value of taxDeduction will this code print?

Scope: this once more

```
var taxDeduction=300;        // global var

var myTaxObject = {

    taxDeduction: 400,          // object property

    doTaxes: function() {
         this.taxDeduction += 100;

         var mafiaSpecial= function(){
           console.log( “Will deduct " + this.taxDeduction);
         }

         this.mafiaSpecial();
    }
}

myTaxObject.doTaxes();  //invoking method doTaxes
```

What value of taxDeduction will this code print?

Scope: this once more

```
var taxDeduction=300;        // global var

var myTaxObject = {

    taxDeduction: 400,          // object property

    doTaxes: function() {
         this.taxDeduction += 100;

         var mafiaSpecial= function(){
           console.log( “Will deduct " + this.taxDeduction);
         }

         this.mafiaSpecial();
    }
}

myTaxObject.doTaxes();  //invoking method doTaxes
```

Type Error:

this.mafiaSpecial is not a function

What value of taxDeduction will this code print?

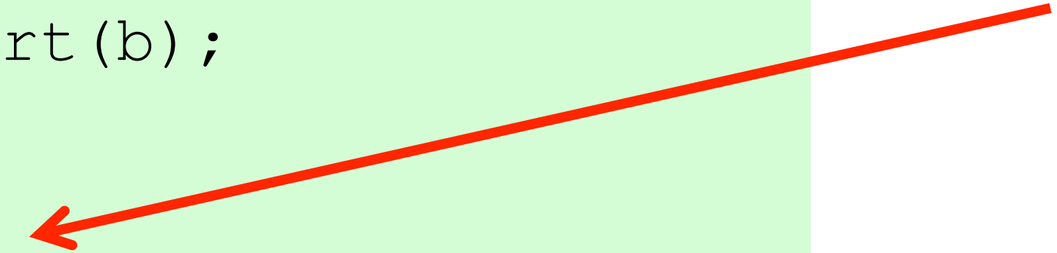
What this code will display?

```
function test () {
    var a=1;

    if(a>0) {
        var b = 5;
    }

    alert(b);
}
```

What this code will display?

```
function test () {
    var a=1;

    if(a>0) {
        var b = 5;
    }

    alert(b);
}

test();
```

Nothing until you invoke it



What this code will display?

```
function test () {
    var a=1;

    if(a>0) {
        var b = 5;
    }

    alert(b);
}

test();
```

What this code will display?

```
function test () {
    var a=1;

    if(a>0) {
        var b = 5;
    }

    alert(b);
}

test();
```

5

Hoisting in action

Closures

Closure is a function call with memory.

Larry Ullman

Closure is a function call with strings attached.

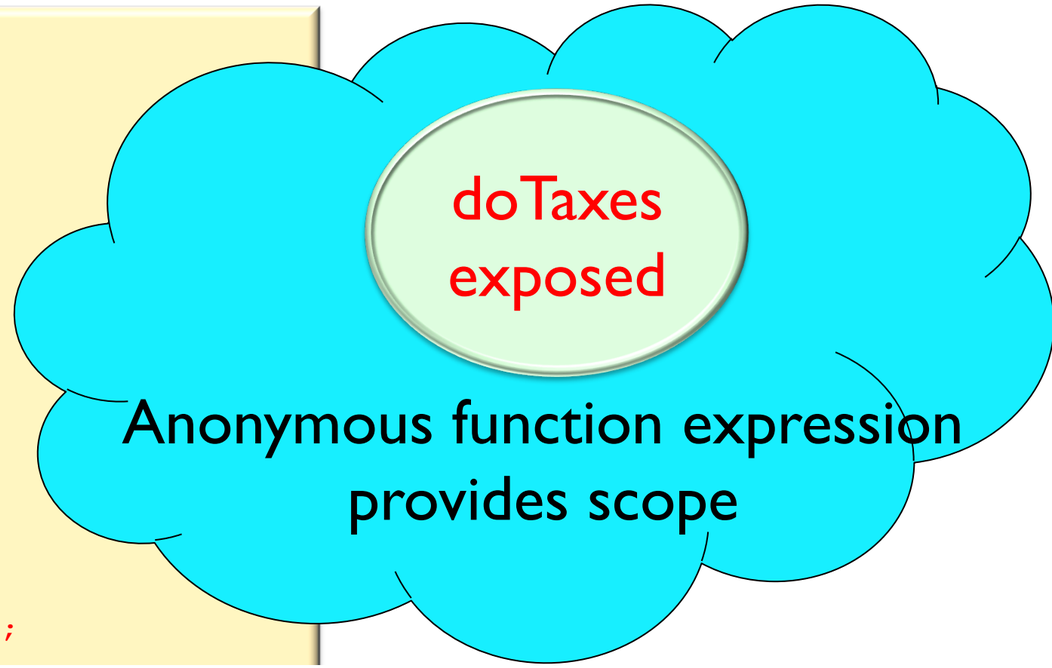
Yours Truly

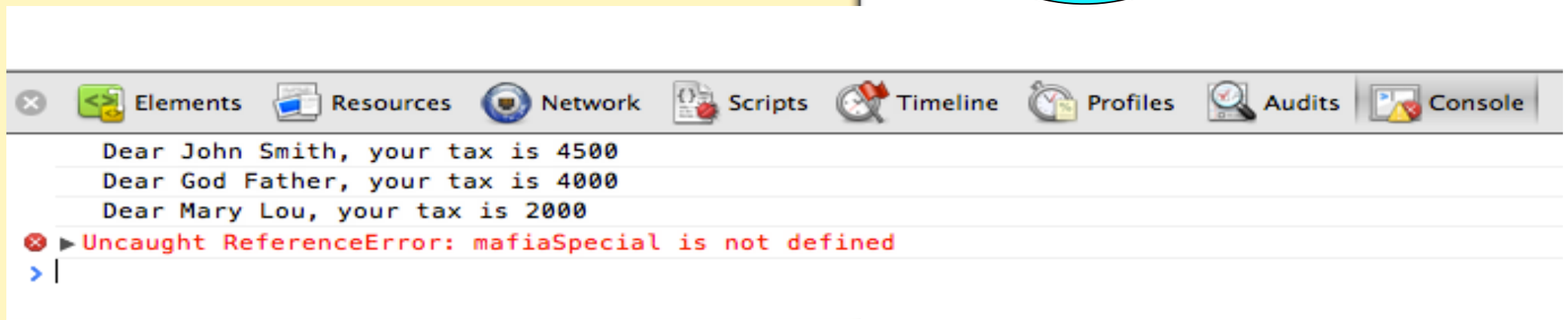
Original image url: http://bit.ly/MYFaXD

JDK 8 will support closures (JSR 335 or Project Lambda).

Controlling Exposure with Closures

```
(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 remembers its context: taxDeduction=500

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*

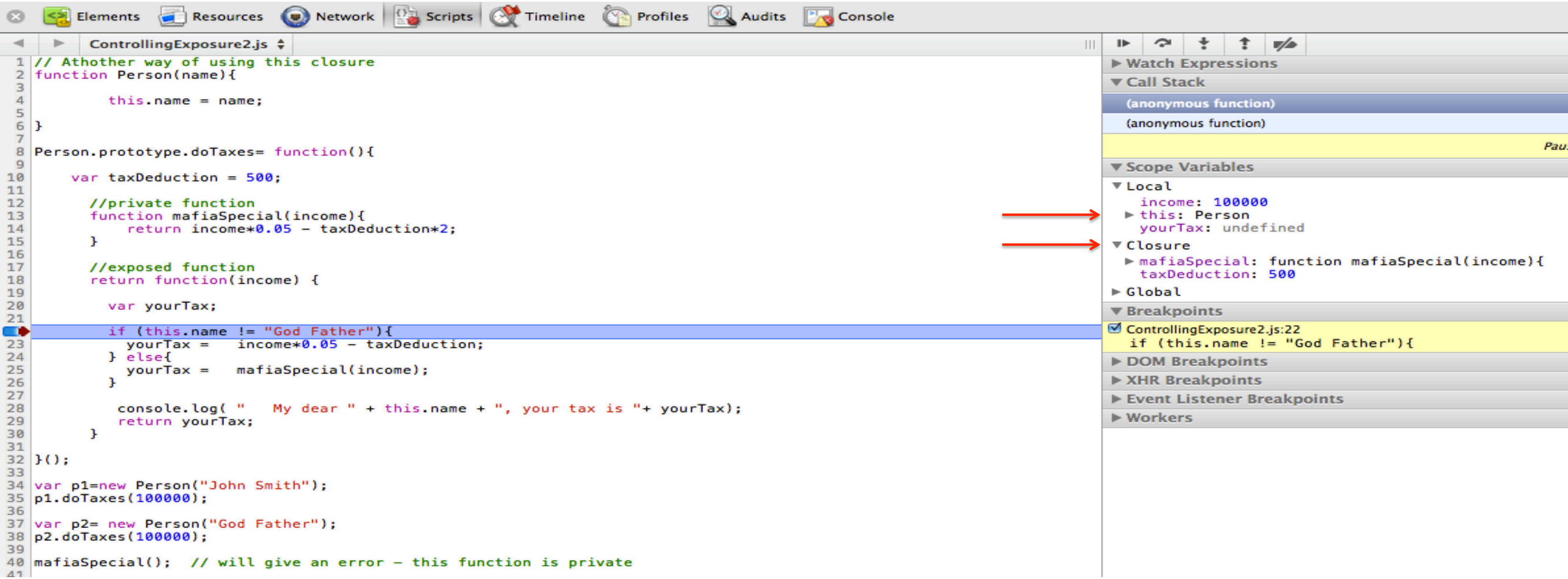
Returning Closure 2

```
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 Closure in Chrome



Mixins

- In Java, subclasses vertically inherit functionality from their superclasses. Multiple inheritance is not supported.

- JavaScript supports *mixins* – pieces of functionality that can be reused by any objects

Borrowing Mixin's Methods

Mixins are code fragments that can be added to objects without using inheritance.

```
var TaxMixin = function () {};

TaxMixin.prototype = {

  mafiaSpecial: function(originalTax){
    console.log("Mafia special:" +
                    (originalTax - 1000));
  },

  drugCartelSpecial: function(originalTax)
{
     console.log("Drug Cartel special:" +
                    (originalTax - 3000));
  }

};
```

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

};
```

Can this be done?

```
var yourTax = new Tax(50000, "NY");

yourTax.mafiaSpecial(10000);
```

Blending TaxMixin into Tax

```
function blend( mainDish, spices ) {

  for ( var methodName in spices.prototype ) {
      mainDish.prototype[methodName] = spices.prototype[methodName];
  }
}

// Blend the spices with the main dish
blend( Tax, TaxMixin );

var yourTax = new Tax(50000, "NY");
yourTax.mafiaSpecial(10000);
```

JavaScript in a Web Browser

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

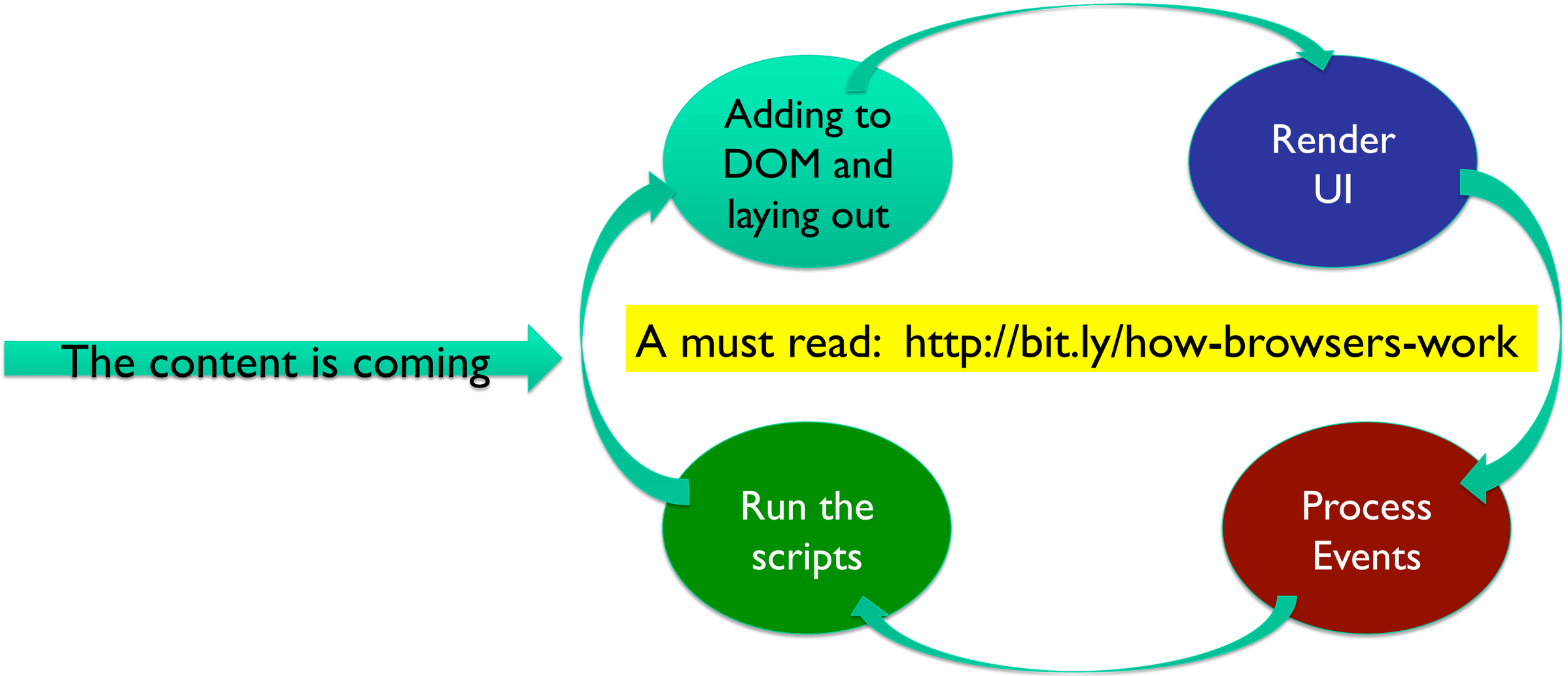
opener - a reference to the parent window that opened the current popup window

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
```

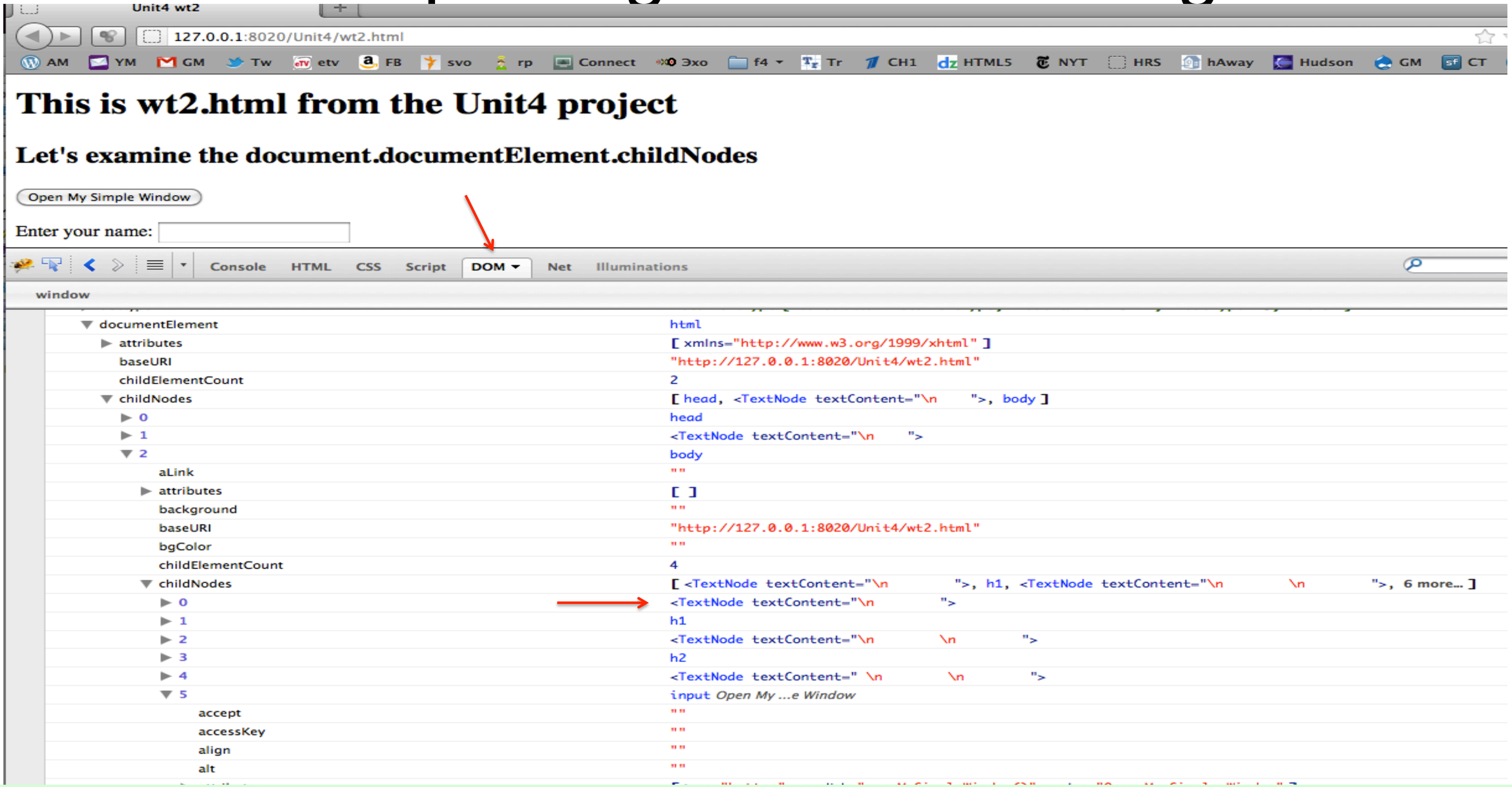
Web Browser's Circle



Document Object Model (DOM)

- Every element of an HTML page is loaded into DOM.
- Each DOM's element has a reference to its children and siblings.
- Find an element in DOM (query the DOM) and then manipulate it with JavaScript.

Inspecting DOM in Firebug



Script minification tools remove all these useless text nodes and reduce the file size.

Working with DOM

`document.write(text)` – adds the specifies text to the page. It's bad idea - other HTML content may be still arriving.

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

`document.getElementsByTagName(tname)` - 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.

ECMAScript 5 Adds 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 their counterparts like or `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 specified 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=finction(){ … }
  myButton.click=myFunctionHandler;
</script>
```

Events Phases

JavaScript target = Swing source

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

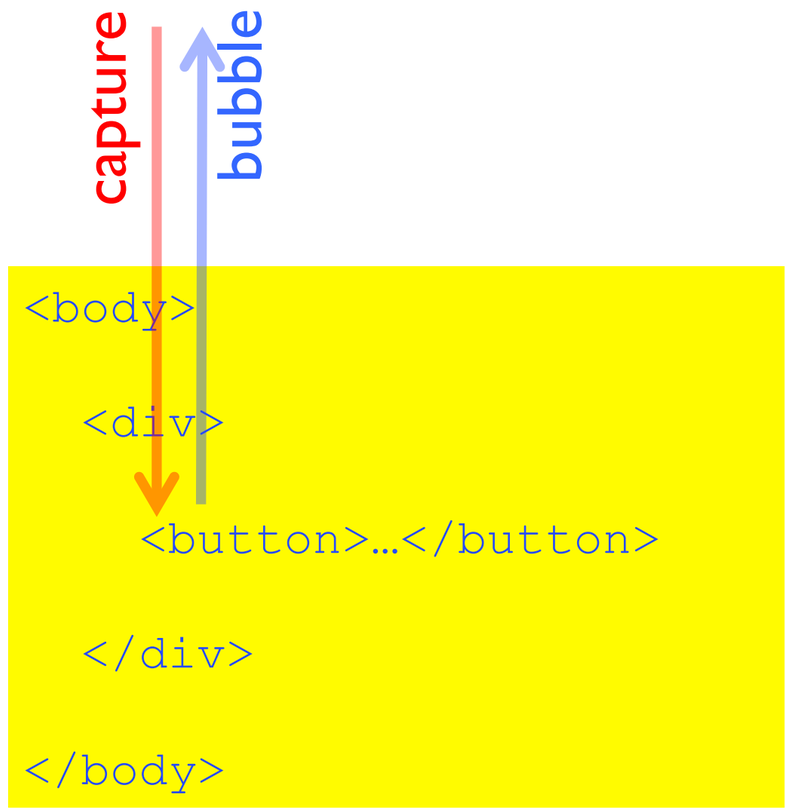
Bubbling phase is when event object from target up through all enclosing containers.

Registering an event listener in JavaScript

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

Removing an event listener:

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



Download this slide deck at http://bit.ly/SF0Wqf

Q&A

DEVOXX™
the java™ community conference