Internet Programming
Week 3

Instructor:
Daniel Slack, P. Eng.
Applied Computer Science
University of Winnipeg

Announcments

- Why learn “Vanilla JS”
  - https://snipcart.com/blog/learn-vanilla-javascript-before-using-js-frameworks
- JavaScript objects (what we’re talking about today)
  - https://www.kirupa.com/html5/objects_classes_javascript.htm
- Jetbrains Suite (including Webstorm)
  - https://www.jetbrains.com/shop/eform/students

Reading

- Please read:
  - Chapter 4: Data Structures: Objects and Arrays
    - Datasets, Properties, Methods, Objects, Strings and their Properties, The Arguments Object, Strings and their Properties, The Math Object, The Global Object
    - The rest of Chapter 4 is up to you
  - Chapter 6: The Secret Life of Objects
  - Chapter 12: JavaScript and the Browser
    - Today’s lecture

Objects

- Start with an example
- e.g. a dog
  - A dog could have the following properties
    - Name (string)
    - Weight (integer)
    - Breed (string)
    - List of activities they enjoy (Array of strings)

Objects

- Example of creating an object

```
var fido = {
      name: “Fido”,
      weight: 40,
      breed: “Mixed”,
      loves: [“walks”, “fetching balls”]
};
```

- Properties are accessed with dot notation

```
if (fido.weight > 25)
      alert(“WOOF”);
else
      alert(“YIP”);
```

Objects

- Properties can also be accessed using a string with [] notation

```
var breed = fido[“breed”];
if (breed == “Mixed”) {
      alert(“Best in show”);
}
```

- Enumerate all an object’s properties

```
var prop;
for (prop in fido) {
      alert(“Fido has a “ + prop + “ property “);
}
```

Objects

- Properties can be added or deleted at any time
  - Add a property

```
fido.age = 5;
```

  - Delete a property

```
delete fido.age;
```

Passing Objects to Functions

- Arguments are passed by value
  - e.g. passing an integer results in the function receiving a copy
    - Thus, original variable cannot be modified by the function
- Same is true for objects
  - Sort of!

Passing Objects to Functions

- Variables assigned objects only contain a reference to the object
- Thus, function call copies reference
  - Any change to object property is made to original object

Object Behaviour

- Example of adding a method to an object

```
var fido = {
   name: “Fido”,
   weight: 40,
   breed: “Mixed”,
   loves: [“walks”, “fetching balls”];
   bark: function () {
         alert(“Woof woof!”);
   }
};
```

Defining Objects

- Was the dog object defined thus far useful?
- We didn't really create a reusable Dog object..
  - We created a fido object
  - What if we wanted to create a Lassie object?

Object Constructor

```
function Dog(name, breed, weight) {
   this.name = name;
   this.breed = breed;
   this.weight = weight;
   this.bark = function() {
        if (this.weight > 25) {
               alert(this.name + “ says Woof!”);
        } else {
               alert(this.name + “ says Yip!”);
        }
   };
}
```

Constructor Observations

- By convention, the constructor names are capitalized
- Property names and parameter names do not have to be the same
  - Usually are by convention
- Notice the syntax differs from object syntax

Using the Constructor

```
var fido = new Dog(“Fido”, “Mixed”, 38);
var tiny = new Dog(“Tiny”, “Chihauhua”, 8);
var clifford = new Dog(“Clifford”, “Bloodhound”, 65);

fido.bark();
tiny.bark();
clifford.bark();
```

Prototype-Based Language

- JavaScript is a class-free, object-oriented language
  - Uses prototypal inheritance
    - Means that behaviour reuse (inheritance) is performed via a processes of cloning existing objects
    - These objects serve as *prototypes*
  - Instead of classical inheritance
- Class-based object-oriented languages use classes and instances
- JavaScript only has objects
  - Since it is a prototype-based language

Prototype-Based Language

- JavaScript does not natively support the declaration of class hierarchies
- However:
  - JavaScript's prototype mechanism simplifies the process of adding properties and methods to all instances of an object

Prototype-Based Language

- In JavaScript you can add custom properties to specific objects

```
//Create an empty object “bicycle”
function Bicycle() {
}
//Create an instance of bicycle called roadbike
var roadbike = new Bicycle();
//Define a custom property, wheels, for roadbike only
roadbike.wheels = 2;
```

Prototype-Based Language

- A custom property added this way only exists for that instance of the object
  - Another instance of bicycle(), called mountainbike, for example, would return undefined to the call mountainbike.wheels.
- Sometimes this functionality is desired
  - Other times you may want all instances of an object to have the property

Prototype-Based Language

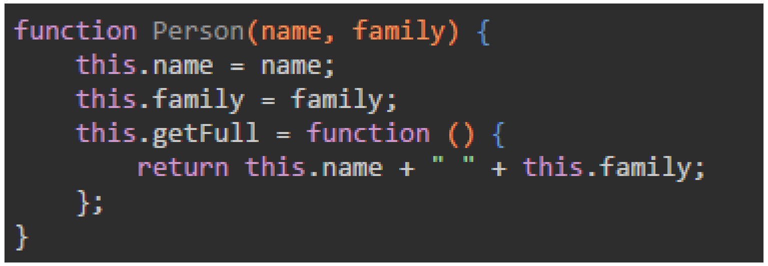
- The prototype object is used to add a property to all instances
- The prototype object allows you to quickly add a custom property to an object for all instances

```
//First,  create the “bicycle” object
function Bicycle () {
}
//Assign the wheels property to the object’s prototype
Bicycle.prototype.wheels = 2;
```

- Isn’t this the same as using a constructor?
- Which approach is better?

Prototype vs. Constructor

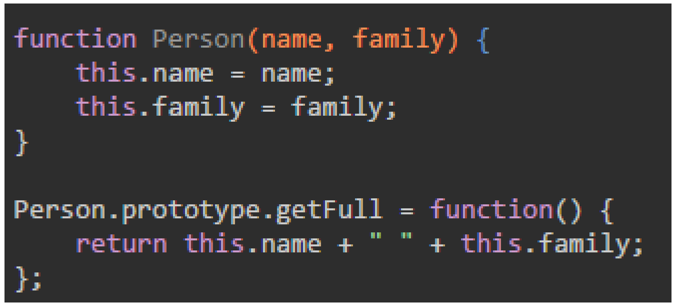
- Recall, functions are objects
- In the example below:
  - Binding a method using *this* provides the method to only that particular instance of the object
  - Since functions are objects: Adding another property
  - Any method attached via *this* will be re-declared for every new instance we create
    - Could affect memory usage of the application



```
function Person(name, family) {
    this.name = name;
    this.family = family;
    this.getFull = function () {
        return this.name + " " + this.family;
    };
}
```

Prototype vs. Construction

- Solution:
  - Use the object’s prototype
  - One single method (represented as a single object)
  - Object available to all instances via the object’s prototype
    - Only stored in memory once
    - Objects coming from the same constructor point to one common prototype object



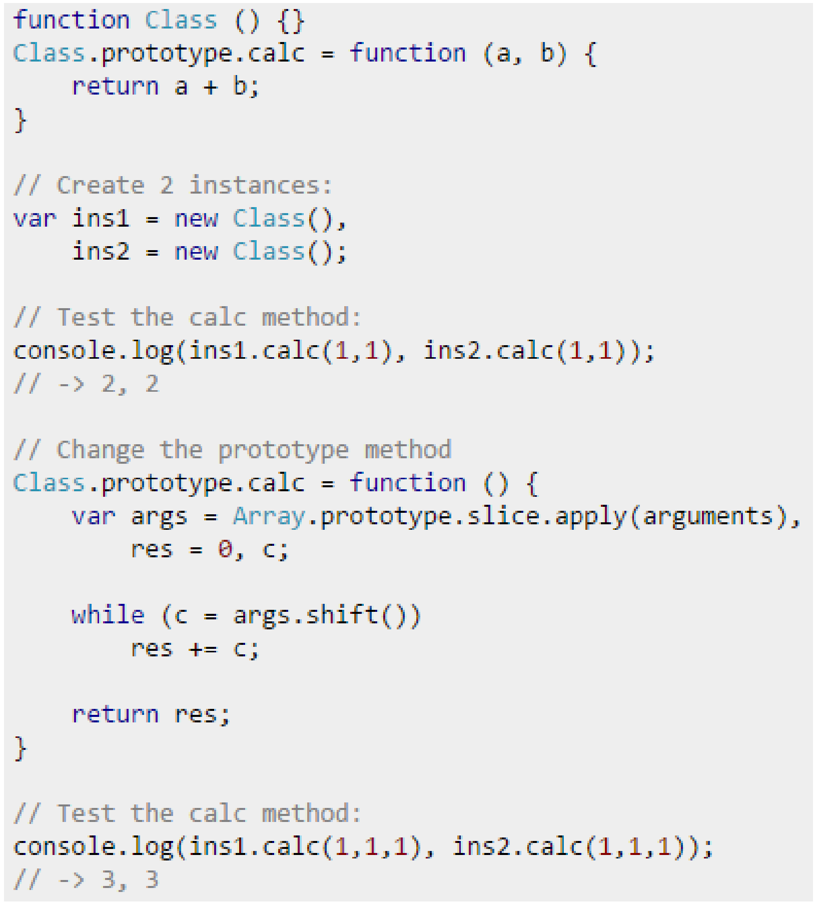
```
function Person(name, family) {
    this.name = name;
    this.family = family;
}

Person.prototype.getFull = function() {
    return this.name + " " + this.family;
};
```

http://thecodeship.com/web-development/methods-within-constructor-vs-prototype-in-javascript/

Another Example

- Methods that inherit via the prototype chain can be changed universally for all instances



```javascript
function Class () {}
Class.prototype.calc = function (a, b) {
    return a + b;
}

// Create 2 instances:
var ins1 = new Class(),
    ins2 = new Class();

// Test the calc method:
console.log(ins1.calc(1,1), ins2.calc(1,1));
// -> 2, 2

// Change the prototype method
Class.prototype.calc = function () {
    var args = Array.prototype.slice.apply(arguments),
        res = 0, c;

    while (c = args.shift())
        res += c;

    return res;
}

// Test the calc method:
console.log(ins1.calc(1,1,1), ins2.calc(1,1,1));
// -> 3, 3
```

http://stackoverflow.com/questions/4508313/advantages-of-using-prototype-vs-defining-methods-straight-in-the-constructor

Misc.: window Object

- window object plays a major role in your applications
  - Represents both
    - Global environment of your application
    - Main window for your app
  - Result: contains many core properties and methods
- Location: holds the URL of the page
  - If you change it, the browser retrieves the new URL
- Status: holds the string displayed in the status area of your browser

Misc. window Object

- onload: holds the function to be called when the browser loads
- alert method: Displays an alert
- document: holds the DOM
- prompt: like alert, except it gets information from the user
- open: opens a new browser window
- close: closes the window
- setTimeout: Invokes a handler after a specified interval
- setInterval: Invokes a handler on a specified interval, repeatedly

Misc. window Object

- Why not window.alert()?
  - The window object acts as your global environment
  - Names of any properties or methods from window are resolved
  - Any global variables you define are also put into the window namespace
    - Sometimes it is a good idea to specify the window object
      - window.onload

Misc. document Object

- Also contains many core properties and methods
- domain: domain of the server that is serving the document
- title: title of the document
- URL: the URL of the document
- getElementById: Gets an element by its id
- getElementsByTagName: Retrieves elements by their tag name
- getElementByClassName: Retrieves elements by their class name
- createElement: Creates elements for inclusion in DOM

Misc. element Objects

- Also contains many core properties and methods
- innerHTML: Contains elements inner HTML
- childElementCount: Gives the number of children
- firstChild: Reference to the object’s first child
- appendChild, insertBefore: Used to insert new elements into the DOM
- setAttribute, getAttribute: Used to set and get attributes like “src”, “class”, and “id”

Chapter 14

Events and Handlers

Playlist Example

- Nothing happens when I click “Add Song”
  - Browser knows you clicked the button
  - How do we add functionality when button is clicked?
- We need two things
  - JavaScript code with desired functionality
  - Associate JavaScript code with button click

Handling Events

- There are many events in the browsers
  - Button Clicks
  - Receiving requested data
  - Timers
- JavaScript can respond to events
  - Called handling events

Plan

1. Set up a handler to handle the user’s click on “Add Song”
2. Write the handler to get the song name input
3. Create a new element to hold the new song
4. Add the element to the page’s DOM

Access the “Add Song” Button

- How do we access the “Add Song” button?

```
var button = document.getElementById(“addButton”);
```

Adding a Handler

- How do we add a handler to the button click?
  1. Create a function

```
function handleButtonClick() {
    alert(“Button was clicked”);
}
```

  2. Use the button’s onclick property

```
Button.onclick = handleButtonClick;
```

Getting the Song Name

- How are we going to get the song name?
  - The user has typed the name in an input box
  - Any thing that happens in the page is reflected in the DOM
    - Text is there too
- Must get a reference to the input element

How Do We Add a Song to the Page

- We already created an empty list
  - List is present in the DOM
- Solution: Add a new item ever time user enters a song
- Browser will update the page

Create a New Element

- A new li element can be created with the following

```
var li = document.createElement(“li”);
```

- Note: This element is not in the DOM (yet)
  - We have to attach it!

Adding an Element to the DOM

- You need to know where to put an element before you can add it

Introducing the Event Model

- Recall how we have handled events thus far
  - Adding handler to onevent attribute in HTML code
  - Using traditional binding
    - Adding handler using onclick property in JavaScript file
- What did we discuss about calling more than one function when an event occurs?

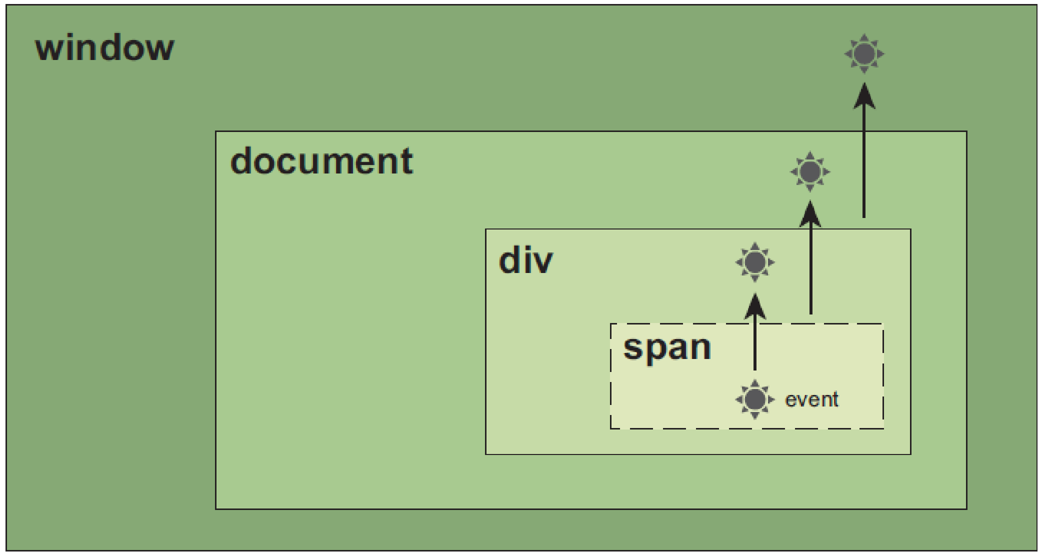
Introducing the Event Model

- Problem can be solved using an event model
- The event model describes how events interact with objects
  - IE event model
    - Supported by IE and Opera
  - W3C event model
    - Supported by other major browsers

Introducing the Event Model

- event bubbling: an event is initiated at the bottom of the object tree and rises to the top of the hierarchy

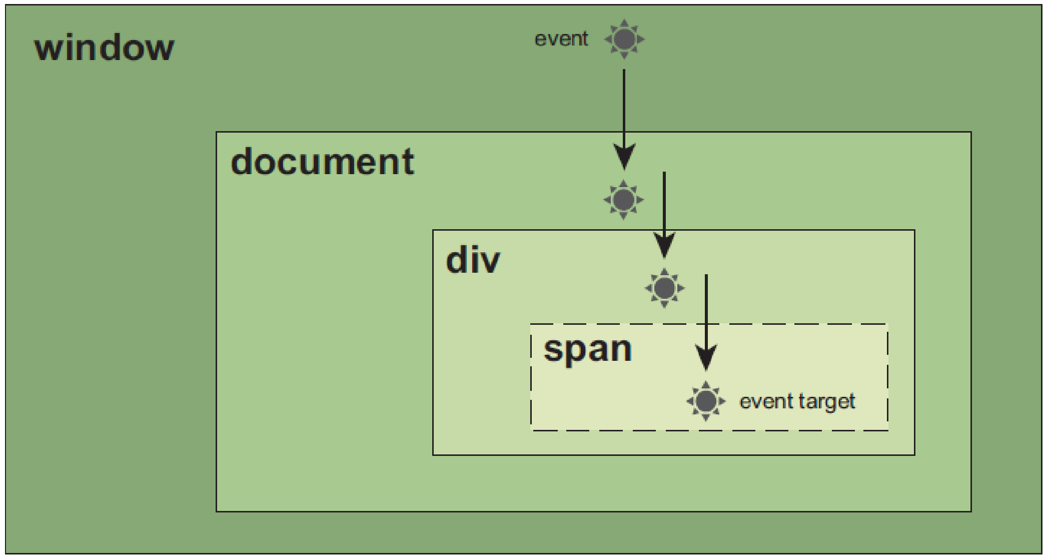
Event bubbling in the IE event model **Figure 15-8**



Introducing the Event Model

- event capturing: events are initiated at the top of the object hierarchy and drop down the object tree to the lowest object

Event capturing Figure 15-9



Introducing the Event Model

- W3C event model: an event is split into three phases
  - A **capture phase** as the event moves down the object hierarchy
  - A **target phase** in which the event reaches the object from which the event originated
  - A **bubbling phase** in which the event moves back up the object hierarchy

Attaching and Listening for Events

- To run a function:
  - Create an event listener
    - Detects when a particular event has reached an object in the document
- W3C Event Model:

```
object.addEventListener(event, function, capture)
```

  - *object* is the object receiving the event
  - *event* is a text string naming the event
  - *function* is a function that runs in response to the event
  - *capture* is a Boolean value
    - True: listen for event during capture phase
    - False: listen for event during bubbling phase

Introducing the Event Model

- Model allows you to remove event handlers from objects
  - The W3C event model uses the removeEventListener method

```
object.removeEventListener (event, function, capture)
```

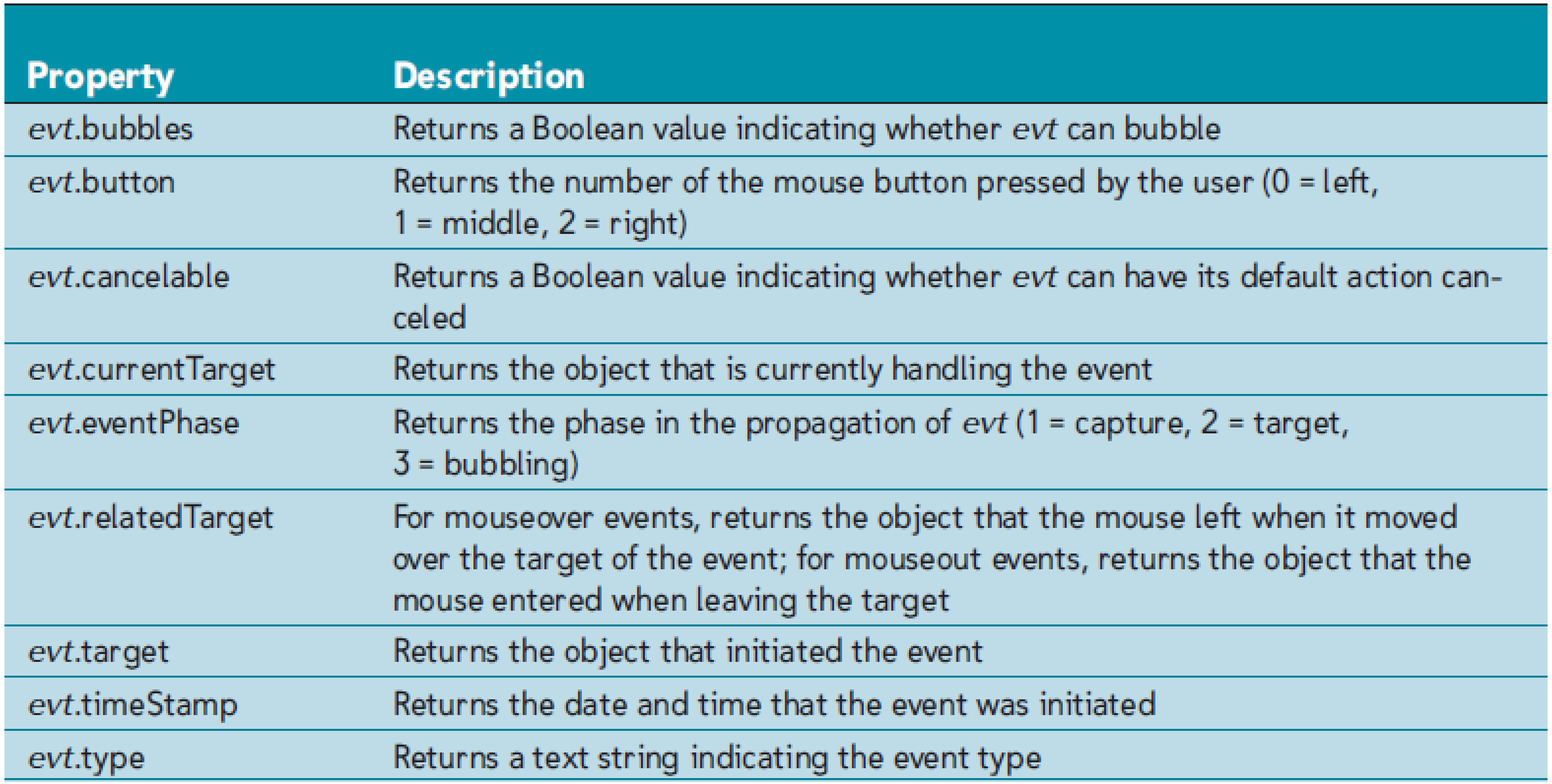
Introducing the Event Object

- You now know how events are propagated and handled in the two models
- Next, we need to know how to get information about the event
  - e.g. You may want to know the location of the mouse for a mouse event
- This type of information is stored in an event object

The W3C Event Object

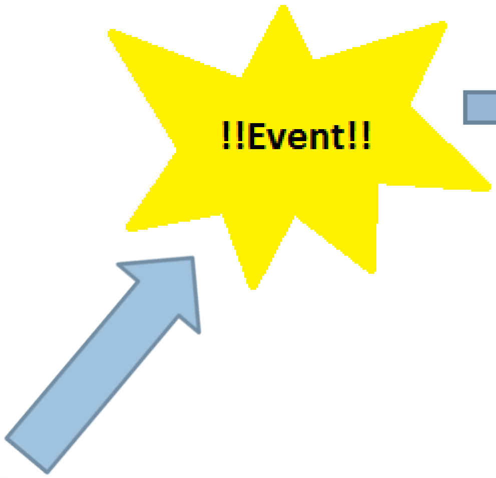
- The event object is inserted as a parameter of the function responding to the event
  - Standard practice is to name the parameter “e” or “evt”
- The object that initiated an event is returned using the target property

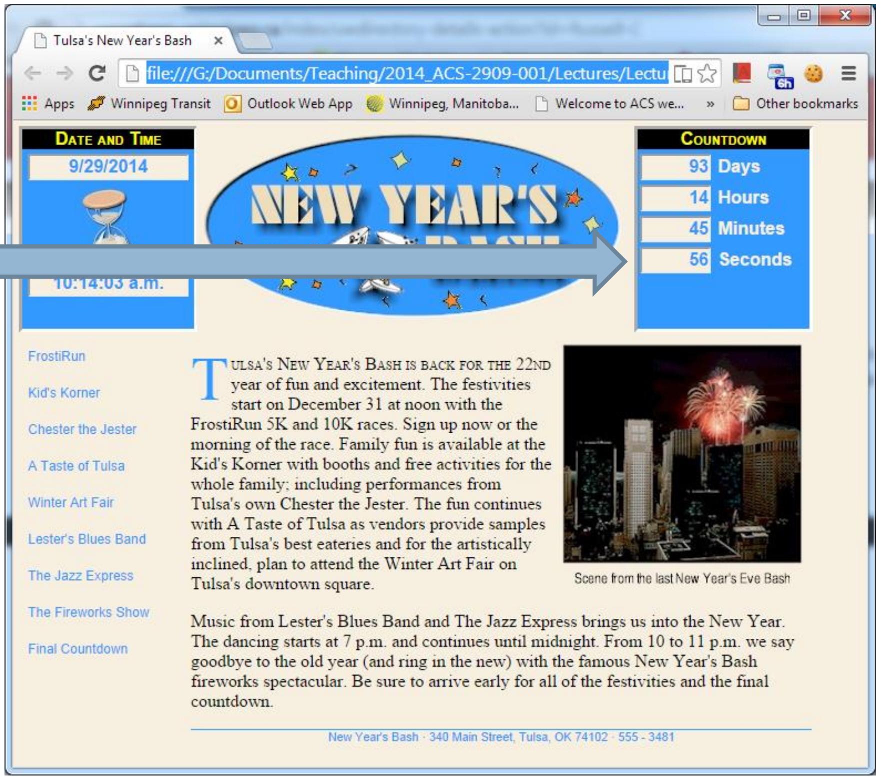
The W3C Event Object



Property	Description
evt.bubbles	Returns a Boolean value indicating whether *evt* can bubble
evt.button	Returns the number of the mouse button pressed by the user (0 = left, 1 = middle, 2 = right)
evt.cancelable	Returns a Boolean value indicating whether *evt* can have its default action canceled
evt.currentTarget	Returns the object that is currently handling the event
evt.eventPhase	Returns the phase in the propagation of *evt* (1 = capture, 2 = target, 3 = bubbling)
evt.relatedTarget	For mouseover events, returns the object that the mouse left when it moved over the target of the event; for mouseout events, returns the object that the mouse entered when leaving the target
evt.target	Returns the object that initiated the event
evt.timeStamp	Returns the date and time that the event was initiated
evt.type	Returns a text string indicating the event type

Example





DATE AND TIME

9/29/2014

10:14:03 a.m.

NEW YEAR'S BASH

COUNTDOWN

93 Days

14 Hours

45 Minutes

56 Seconds

FrostiRun

Kid's Korner

Chester the Jester

A Taste of Tulsa

Winter Art Fair

Lester's Blues Band

The Jazz Express

The Fireworks Show

Final Countdown

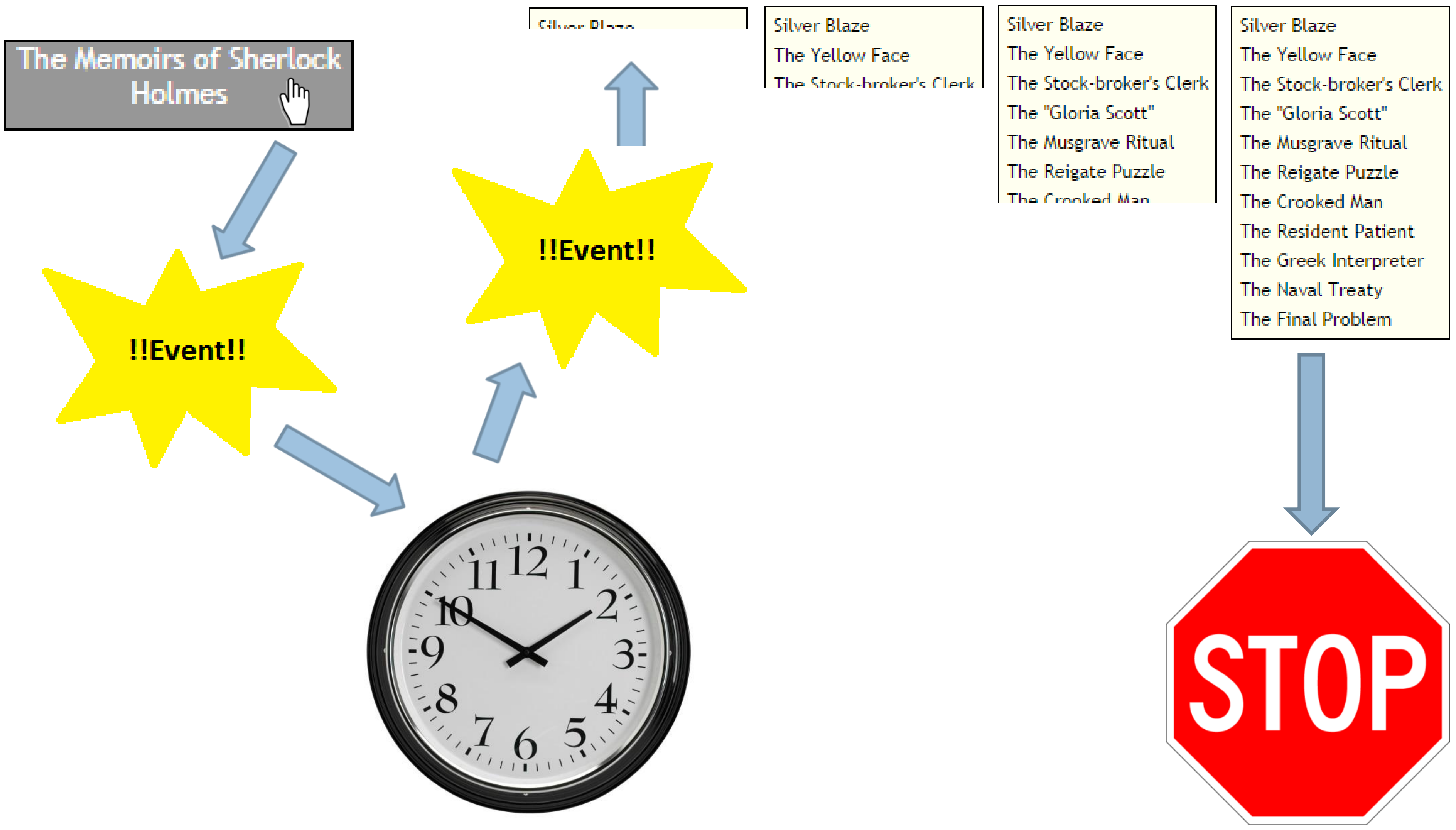
TULSA'S NEW YEAR'S BASH IS BACK FOR THE 22ND year of fun and excitement. The festivities start on December 31 at noon with the FrostiRun 5K and 10K races. Sign up now or the morning of the race. Family fun is available at the Kid's Korner with booths and free activities for the whole family; including performances from Tulsa's own Chester the Jester. The fun continues with A Taste of Tulsa as vendors provide samples from Tulsa's best eateries and for the artistically inclined, plan to attend the Winter Art Fair on Tulsa's downtown square.

Scene from the last New Year's Eve Bash

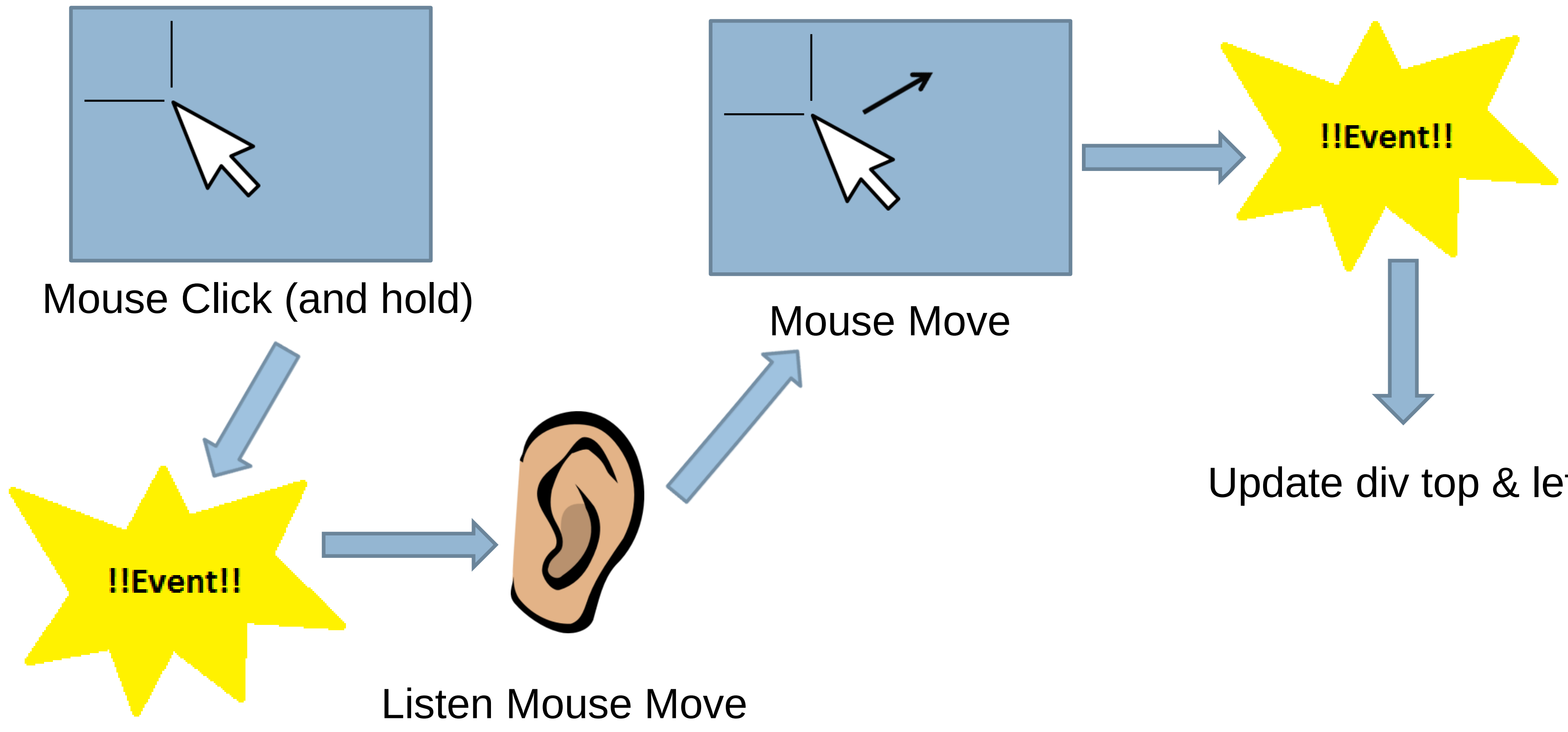
Music from Lester's Blues Band and The Jazz Express brings us into the New Year. The dancing starts at 7 p.m. and continues until midnight. From 10 to 11 p.m. we say goodbye to the old year (and ring in the new) with the famous New Year's Bash fireworks spectacular. Be sure to arrive early for all of the festivities and the final countdown.

New Year's Bash · 340 Main Street, Tulsa, OK 74102 · 555 - 3481

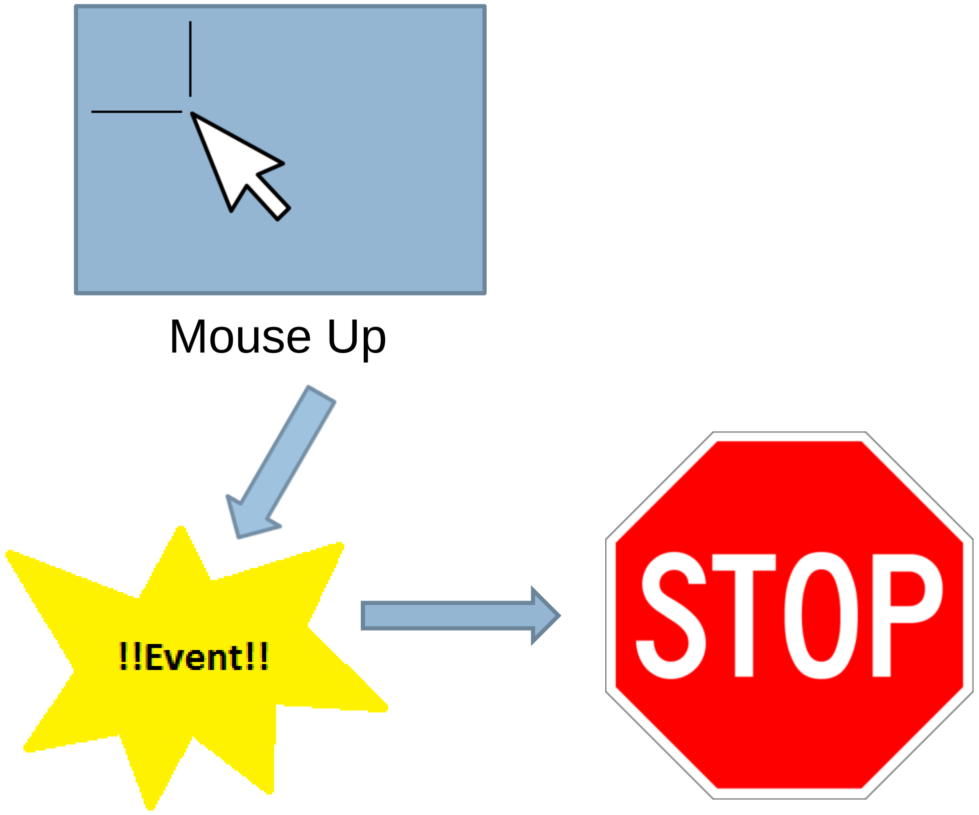
Example


The Memoirs of Sherlock Holmes
!!Event!!
!!Event!!
Silver Blaze
Silver Blaze
The Yellow Face
The Stock-broker's Clerk
Silver Blaze
The Yellow Face
The Stock-broker's Clerk
The "Gloria Scott"
The Musgrave Ritual
The Reigate Puzzle
The Crooked Man
Silver Blaze
The Yellow Face
The Stock-broker's Clerk
The "Gloria Scott"
The Musgrave Ritual
The Reigate Puzzle
The Crooked Man
The Resident Patient
The Greek Interpreter
The Naval Treaty
The Final Problem
STOP

Example


Mouse Click (and hold)
!!Event!!
Listen Mouse Move
Mouse Move
!!Event!!
Update div top & left

Example


Mouse Up
!!Event!!
STOP