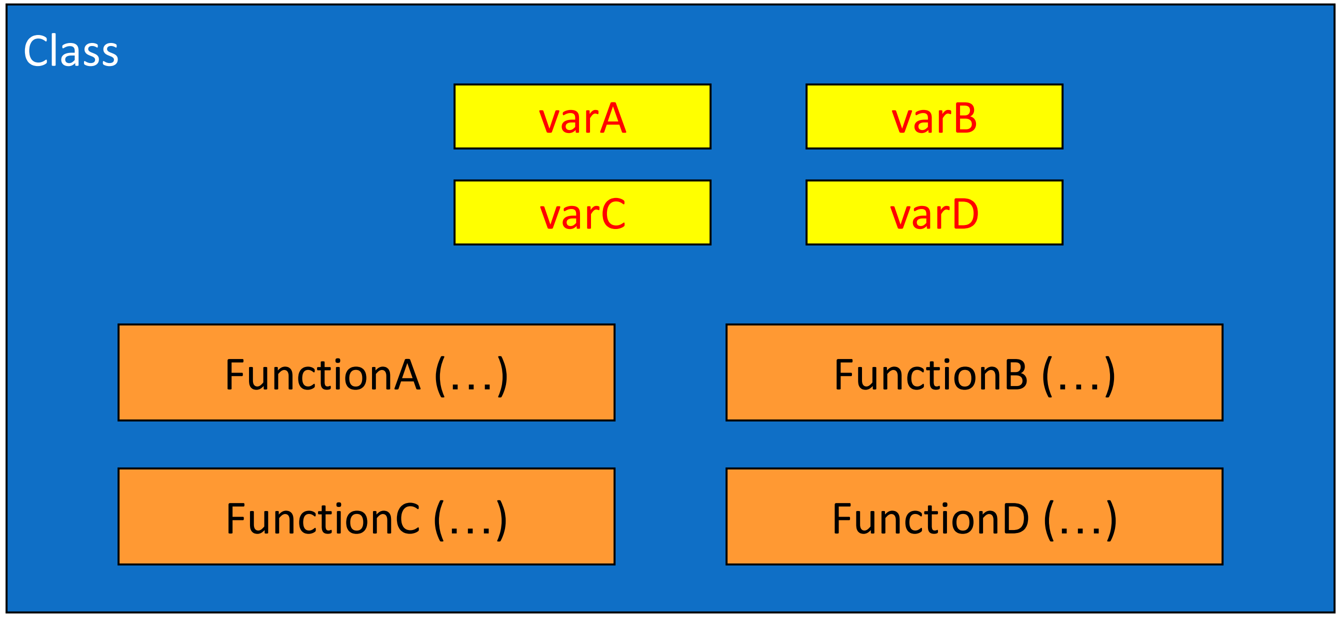
JavaScript Objects

B Nagaraju
http://nbende.wordpress.com

Understanding JavaScript Objects

- One of the most important features of JavaScript, enables modular programs.
- Objects are based on Classes, variables, functions, statements are contained in a structure called class.



JavaScript Objects

- In JS objects are from 4 different domains:
  - Those built into JavaScript
  - Those from the Browser Object Model
  - Those from the Document Object Model
  - Custom objects from the developer

Class and Object

- You can instantiate an object from a class by using the constructor function.
- JavaScript is said to be an Object-based programming language.
- What is property?
  - A variable belongs to the object.
- What is method?
  - It is a function belongs to the object.
  - Function-Valued Properties

Creating Instances of Objects

- You can use the “new” operator to create instances of objects of a particular class or object type.
  - Variable = new objectType(parameters)
- This objectType() is called constructor.
- E.g. Date is a predefined object type.
  - Assign the current date and time to objA
    - objA = new Date()
  - Assign another date and time to objB
    - objB = new Date(99,23,5) ← 23 May, 99

Objects and Classes

- Fields Can Be Added On-the-Fly
  - Adding a new property (field) is a simple matter of assigning a value to one
  - If the field doesn’t already exist when you try to assign to it, JavaScript will create it automatically.
  - For instance:

```
var test = new Object();
test.field1 = "Value 1";    // Create field1 property
test.field2 = 7;                        // Create field2 property
```

Objects and Classes – Literal Notation

- You Can Use Literal Notation
  - You can create objects using a shorthand "literal" notation of the form

```
{ field1:val1, field2:val2, ... , fieldN:valN }
```

  - For example, the following gives equivalent values to `object1` and `object2`

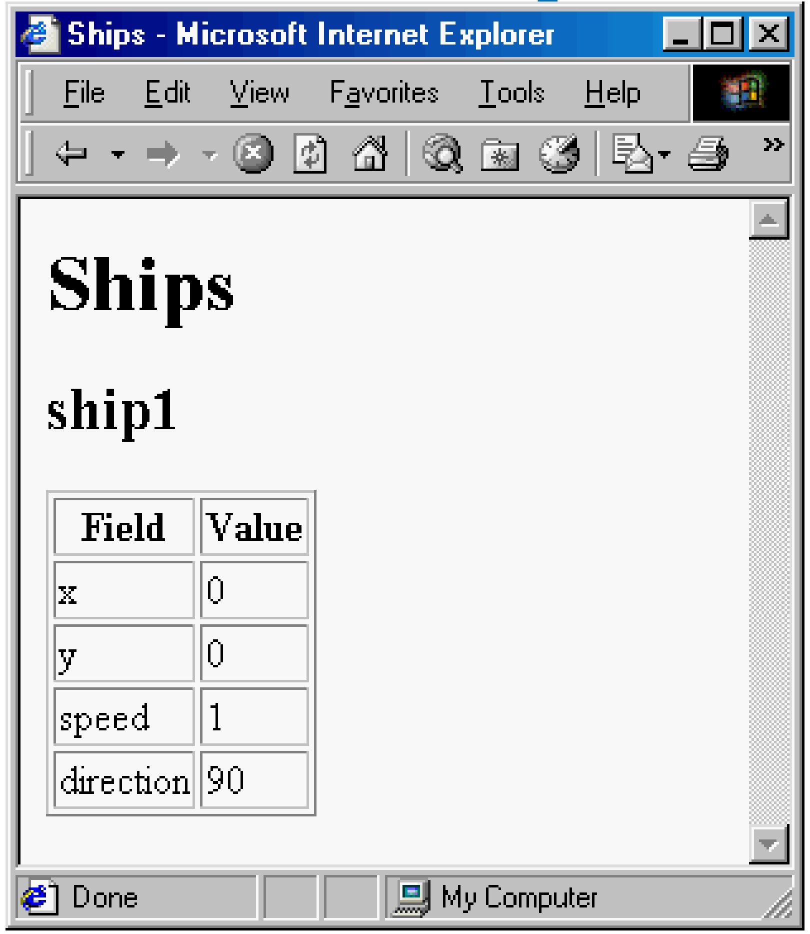
```
var object1 = new Object();
var object2 = new Object();
object1.x = 3;
object1.y = 4;
object1.z = 5;
object2 = { x:3, y:4, z:5 };
```

Objects and Classes - Constructor

- A "Constructor" is Just a Function that Assigns to "this"
  - JavaScript does not have an exact equivalent to Java/.Net class definition
  - The closest you get is when you define a function that assigns values to properties in the `this` reference
  - Calling this function using `new` binds `this` to a new `Object`
  - For example, following is a simple constructor for a `Ship` class

```
function Ship(x, y, speed, direction) {
            this.x = x;
this.y = y;
this.speed = speed;
this.direction = direction;
}
```

Constructor, Example


Ships - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Done
My Computer

Ships

ship1

Field	Value
x	0
y	0
speed	1
direction	90

```
var ship1 = new Ship(0, 0, 1, 90);
```

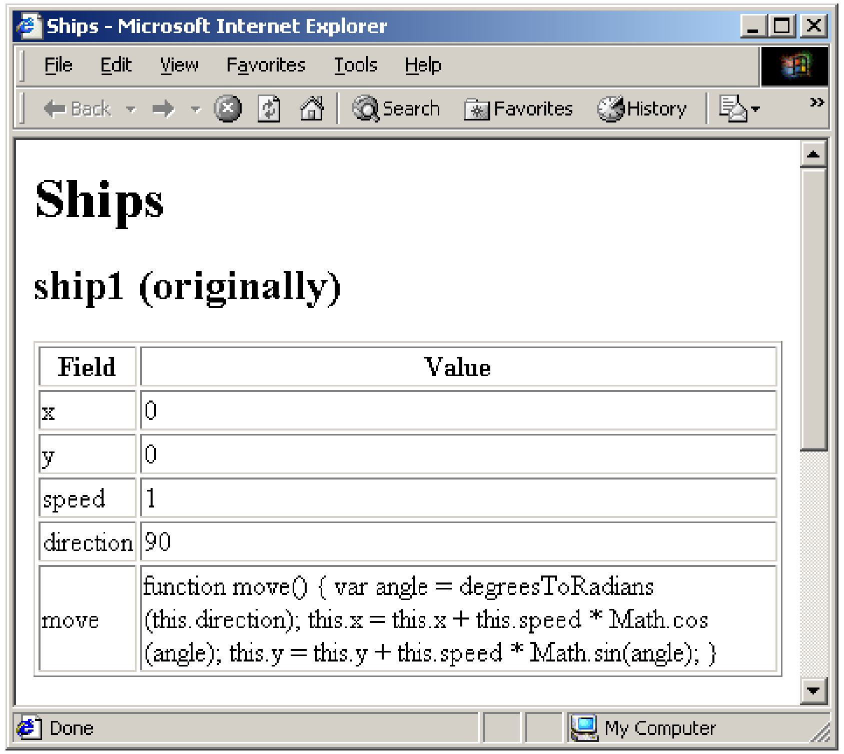
Class Methods, Example

```
 function degreesToRadians(degrees) {
  return(degrees * Math.PI / 180.0);
}

function move() {
  var angle = degreesToRadians(this.direction);
  this.x = this.x + this.speed * Math.cos(angle);
  this.y = this.y + this.speed * Math.sin(angle);
}
function Ship(x, y, speed, direction) {
  this.x = x;
  this.y = y;
  this.speed = speed;
  this.direction = direction;
  this.move = move;
}
```

Class Methods, Result

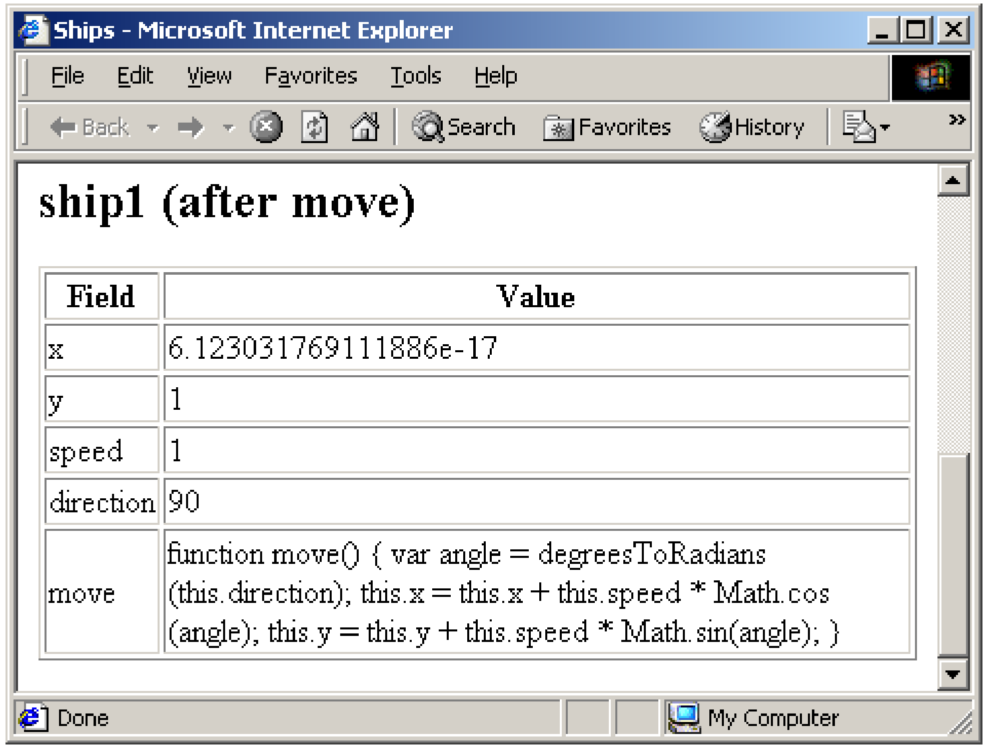
```
var ship1 = new Ship(0, 0, 1, 90);
ship1.move();
```


Ships - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Done
My Computer

Ships

ship1 (originally)

Field	Value
x	0
y	0
speed	1
direction	90
move	function move() { var angle = degreesToRadians (this.direction); this.x = this.x + this.speed * Math.cos (angle); this.y = this.y + this.speed * Math.sin(angle); }


Ships - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Done
My Computer

ship1 (after move)

Field	Value
x	6.123031769111886e-17
y	1
speed	1
direction	90
move	function move() { var angle = degreesToRadians (this.direction); this.x = this.x + this.speed * Math.cos (angle); this.y = this.y + this.speed * Math.sin(angle); }

Objects and Classes - Arrays

- Arrays
  - For the most part, you can use arrays in JavaScript a lot like Java/.Net arrays.
    - Here are a few examples:

```
   var squares = new Array(5);
for(var i=0; i<squares.length; i++) {
   vals[i] = i * i;
}
// Or, in one fell swoop:
var squares = new Array(0, 1, 4, 9, 16);
var array1 = new Array("fee", "fie", "fo", "fum");
// Literal Array notation for creating an array.
var array2 = [ "fee", "fie", "fo", "fum" ];
```

  - Behind the scenes, however, JavaScript simply represents arrays as objects with numbered fields
    - You can access named fields using either `object.field` or `object["field"]`, but numbered fields only via `object[fieldNumber]`

Build-In JavaScript Objects

Object	Description
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Error	Returns run-time error information
Function	Creates new function objects
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers.
String	Contains methods and properties for manipulating text strings

Scope

- Scope is a place where variables are defined and can be accessed
- JavaScript has only two types of scopes
  - Global scope and function scope
    - Global scope is the same for the whole web page
    - Function scope is different for every function
  - Everything outside of a function scope is inside of the global scope

```
if(true){
    var sum = 1+2;
}
console.log(sum);
```

ECMAScript 6 Way of Working with Scopes

- ECMAScript 6 introduces a new way to handle scopes:
  - The key word '`let`'
- `let` is much like var
  - Creates a variable
- But, `let` creates a block scope

```
if(false){
  var x = 5;
  let y = 6;
}
alert(x); //prints undefined
alert(y); //throws error
```

Prototypes and Prototype Inheritance

prototype is a property of functions and of objects that are created by constructor functions.

```
function Vehicle(wheels, engine) {
    this.wheels = wheels;
    this.engine = engine;
}
```

Using Prototypes to Add Properties and Methods

```
var testVehicle = new Vehicle(2, false);
Vehicle.prototype.color = "red";
var testColor = testVehicle.color;
```

You can even add properties and methods to predefined objects.

For example, you can define a Trim method on the String prototype object, and all the strings in your script will inherit the method.

```
String.prototype.trim = function()
{
   // Replace leading and trailing spaces with the empty string
   return this.replace(/(^\s*)|(\s*$)/g, "");
}
var s = "    leading and trailing spaces    ";
// Displays "    leading and trailing spaces     (35)"
window.alert(s + " (" + s.length + ")");
// Remove the leading and trailing spaces
s = s.trim();
// Displays "leading and trailing spaces (27)"
window.alert(s + " (" + s.length + ")");
```

Using Prototypes to Derive One Object from Another with Object.create

The prototype Object can be used to derive one object from another. For example, you can use the Object.create function to derive a new object Bicycle using the prototype of the Vehicle object we defined earlier (plus any new properties you need).

```
var bicycle =
Object.create(Object.getPrototypeOf(Vehicle), {
"pedals" :{value: true} });
```

The bicycle object has the properties wheels, engine, color, and pedals, and its prototype is Vehicle.prototype. The JavaScript engine finds the pedals property on bicycle, and it looks up the prototype chain to find the wheels, engine, and color properties on Vehicle.

Polymorphism

- The ability to call the same method on different objects and have each of them respond in their own way is called polymorphism.
- In OOP, we think of objects that are linked through inheritance has the same methods (override methods) and that the method being called up, is the method associated with the object and not the type of referance.
- This should not be a problem in Java Script as references (variables) in JavaScript is not type-set. We can assign any type of data to a variable in Javascript, and Javascript will know the object a variable refer to if it exists.

Closures

- A first-class function that binds to variables that are defined in its execution environment.
  - **free variable**: A variable referred to by a function that is not one of its parameters or local variables.
    - **bound variable**: A free variable that is given a fixed value when "closed over" by a function's environment.
- A *closure* occurs when a function is defined and it attaches itself to the free variables from the surrounding environment to "close" up those stray references.

Closures in JS

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        var z = 3;
        print(x + y + z);
    };
    y = 10;
}
var g = f();
g();                // 1+10+3 is 14
```

- a function closes over free variables as it is declared
  - grabs references to the names, not values (sees updates)

Module pattern

```
(function(params) {
    statements;
})(params);
```

- declares and immediately calls an anonymous function
  - used to create a new **scope** and **closure** around it
  - can help to avoid declaring global variables/functions
  - used by JavaScript libraries to keep global namespace clean

- Thanks