OBJECT ORIENTATED PROGRAMMING

in Javascript

CLASS DEFINITION AND INSTANTIATION

```
// Define a new class
function MyClassName(val1, val2, ...) {
      // constructor method
      this.property1 = val1;
      this.property2 = val2;
}


// Add methods to class
MyClassName.prototype.methodName = function(val1, val2) {
   // do something with scope of this
   this.property1 = val1;
   this.property2 = val2;
}


// Add class variables or methods
MyClassName.CONSTANT_NAME = “Constant Variable”;


// Instantiate new instance of class
var myInstance = new MyClassName(1, 2, ...);
myInstance.property1 = 30;
myInstance.methodName(“VAL 1”, “VAL 2”);
```

Subclassing

```
// Define a new super class
function ParentClass(val1, val2, ...) {
     // constructor method
     this.property1 = val1;
     this.property2 = val2;
}


// Define a new sub class

function SubClass(val1, val2, ...) {
     // constructor method
     ParentClass.call(this, val1, val2, ...);
}
SubClass.prototype = new ParentClass();

SubClass.prototype.newMethod = function(val1) {
     this.newProperty = val1;
}
```

Scope Issues

```
// Define a new class
function MyClassName() {
     // constructor method
     this.addListeners();
}

// Add methods to class
MyClassName.prototype.addListeners = function() {
   // Add event listeners with JQuery using instance method
   $("a#link1").click(this.doSomething);
}
MyClassName.prototype.doSomething = function(e) {
   // this is not in scope of the instance of class
   console.log( "this is   " + typeof(this) );
   console.log( "target is " + typeof(e.target) );
}
```

Scope Issues

```
// Define a new class
function MyClassName() {
     // constructor method
     this.addListeners();
}

// Add methods to class
MyClassName.prototype.addListeners = function() {
   // Add event listeners with JQuery using instance method
   $(“a#link1”).click(this.doSomething);

   // Add event listener with inline function declaring scope
   var self = this;
   $(“a#link2”).click(function(e) {
      // variable self is passed into handler
      console.log( “self is   ” + typeof(self) );
      console.log( “target is ” + typeof(e.target) );
   });
}
MyClassName.prototype.doSomething = function(e) {
   // this is not in scope of the instance of class
   console.log( “this is   ” + typeof(this) );
   console.log( “target is ” + typeof(e.target) );
}
```

Loading a class in a separate file

```
// Define a self running anonymous function
function(window) {

   // create class
   function MyClassName() {
         // constructor method
   }

   // add class to main scope of window
   window.MyClassName = MyClassName;

} (window);
```

```
<script type="text/javascript" src= "classes/MyClassName.js"></script>
<script type="text/javascript">

   var instance = new MyClassName();

</script>
```

Controller Objects / “Singletons”

```
var myController = {

   property1: “value”,
   property2: “value”,

   arrayValues: [ “val1”, “val2”, “val3” ],

   methodName: function(value1, value2, ...) {
      // this has scope as myController
      // but myController will always have the correct reference
      myController.property1 = value1;
      myController.property2 = value2;
   }

};


myController.property1 = “Some New Value”;

myController.methodName(“val1”, “val2”);

console.log(“Array Value: ” + myController.arrayValues[1];
```