JavaScript for C# Developers

Module 3: Object Oriented JavaScript

Shawn Wildermuth
Wilder Minds
wilderminds.com

pluralsight
hardcore developer training

Agenda

- **Object Oriented JavaScript**
  - Dynamic Objects
  - "Classes" in JavaScript
  - Inheritance in JavaScript
  - Interfaces?
  - Reflection
  - Extension Methods
  - Patterns for JavaScript Objects

Dynamic Objects

- **C# Supports Object and Collection Initializers**

```
// C#
var cust = new Customer()
{
  Name = "Shawn",
  CompanyName" = "Wilder Minds",
  Address = new Address()
  {
    StreetAddress = "123 Main Street",
    CityTown = "Atlanta",
    StateProvince = "Georgia",
    PostalCode = "12345",
    Country = "USA"
  }
};
```

Dynamic Objects

- **Anonymous Types are Closer to JavaScript Objects**

```
// C#
var cust = new {
  Name = "Shawn",
  CompanyName" = "Wilder Minds",
  Address = new {
    StreetAddress = "123 Main Street",
    CityTown = "Atlanta",
    StateProvince = "Georgia",
    PostalCode = "12345",
    Country = "USA"
  }
};
```

Dynamic Objects

- **Simple Object Creation**
  - Follows Object Instantiation

```
// JavaScript
var cust = {
  name: "Shawn",
  "company name": "Wilder Minds",
  address: {
    streetAddress: "123 Main Street",
    cityTown: "Atlanta",
    stateProvince: "Georgia",
    postalCode: "12345",
    country: "USA"
  }
};
```

Dynamic Objects

- **Property Syntaxes**
  - Dot syntax
  - Bracket Syntax

```
// JavaScript
var name = cust.name;
var name2 = cust["name"];

var company = cust."company name";  // NOPE
var company2 = cust["company name"];

var addr = cust.address;
var city = addr.cityState;
var city2 = cust.address.cityState;
```

Dynamic Objects

- **Malleability**
  - In .NET, dynamic + ExpandoObject gives you this behavior

```
// C#
dynamic foo = new ExpandoObject();
foo.Name = "Shawn";
foo.CompanyName = "Wilder Minds";
foo.Phone = "404-555-1212";

var p = foo.Phone;
```

Dynamic Objects

- **JavaScript Can Add Members on the Fly Too**

```
// JavaScript
var cust = {
  name: "Shawn",
  "company name": "Wilder Minds"
};

cust.phone = "404-555-1212";
cust.call = function () {
  var toCall = this.phone;
};
```

"Classes" in JavaScript

- **In .NET, Class is Standard Unit of Work**
  - Containers for data, code and behavior

```
// C#
class Customer
{
  string Name { get; }
  string Company  { get; }

  Customer(string name, string company = "")
  {
    _name = name;
    _company = company;
  }
}
```

"Classes" in JavaScript

- **No such thing as a "Class" in JavaScript**
  - But you can mimic them with some effort

```
// JavaScript
function Customer(name, company) {
  this.name = name;
  this.company = company;
}

var cust = new Customer("Shawn", "Wilder Minds");
var name = cust.name;
```

"Classes" in JavaScript

- **Member Functions Work Fine**

```
// JavaScript
function Customer(name, company) {
  this.name = name;
  this.company = company;
  this.sendEmail = function (email) { ... };
}

var cust = new Customer("Shawn", "Wilder Minds");
cust.sendEmail("shawn@foo.com");
```

"Classes" in JavaScript

- **Need Everything be Public?**
  - Nope…closures to the rescue!

```
// JavaScript
function Customer(name, company) {
  // public
  this.name = name;
  this.company = company;

  // non-public (e.g. private)
  var mailServer = "mail.google.com";

  this.sendEmail = function (email) {
    sendMailViaServer(mailServer);
  };
}
```

"Classes" in JavaScript

- **What about Properties?**
  - Special Syntax…only use as necessary

```
// JavaScript
function Customer(name) {
  var _name = name;

  Object.defineProperty(this, "name", {
    get: function () { return _name; }
  });
}

var cust = new Customer("Shawn");
var name = cust.name; // readonly
```

"Classes" in JavaScript

- **What about Properties?**
  - Setter is similar

```
// JavaScript
function Customer(name) {
  var _name = name;

  Object.defineProperty(this, "name", {
    get: function () { return _name; },
    set: function (value) { _name = value; }
  });
}
```

"Classes" in JavaScript

- **Static Members**

```
// JavaScript
function Customer(name, company) {
  this.name = name;
  this.company = company;
}

Customer.mailServer = "mail.google.com";

var cust = new Customer();
var svr = cust.mailServer; // NOPE
svr = Customer.mailServer; // YUP
```

The Prototype Object

- **Centerpiece of the object story in JavaScript**
  - Each 'type' has a prototype object
  - Just an object (e.g. can add properties to it)
  - All instances of an 'type' shares the members of the prototype

Improving JavaScript "Classes"

- **Sharing a Function**
  - That way each instance doesn't have its own copy

```
// JavaScript
function Customer(name, company) {
  this.name = name;
  this.company = company;
}

// Works but no access to private/member data
Customer.send = function (email) { ... };
```

Improving JavaScript "Classes"

- **Sharing a Function**
  - That way each instance doesn't have its own copy

```
// JavaScript
function Customer(name, company) {
  this.name = name;
  this.company = company;
}

// Gives access to each instance of Customer
Customer.prototype.send = function (email) { ... };

var cust = new Customer("Shawn");
cust.send("shawn@foo.com"); // WORKS
```

Improving JavaScript "Classes"

- Same works for sharing data across instances

```
// JavaScript
function Customer(name, company) {
  this.name = name;
  this.company = company;
}

// Works but no access to private/member data
Customer.prototype.mailServer = "mail.google.com";
Customer.prototype.sendMail = function (msg) {
  var svr = this.mailServer;
};

var cust = new Customer("Shawn", "Wilder Minds");
cust.sendMail("Hey buddy");
```

Inheritance in JavaScript "Classes"

- **Basic Inheritance with the Prototype object**
  - The basic is-a relationship

```
// JavaScript
function Animal(foodType) {
  this.foodType = foodType;
}

Animal.prototype.feed = function () {
  alert("Fed the animal: " + this.foodType);
};

var a = new Animal("None");
a.feed();                          // "None"
var test = a instanceof Animal; // true
```

Inheritance in JavaScript "Classes"

- **Basic Inheritance with the Prototype object**
  - The basic is-a relationship

```
// JavaScript
function Cow(color) {
  this.color = color;
}

// Inheritance Magic
Cow.prototype = new Animal("Hay");

var c = new Cow("White");
c.feed();                          // "Hay"
var test = c instanceof Animal; // true
var test2 = c instanceof Cow;   // true
```

Inheritance in JavaScript "Classes"

- **Can Fake Abstract Classes**
  - With some caveats

```
// JavaScript
var Animal = {
  foodType: "None",
  feed: function () {
    log("Fed the animal: " + this.foodType);
  }
};

var a = new Animal(); // Fails (not a constructor)
```

Inheritance in JavaScript "Classes"

- **Can Fake Abstract Classes**
  - With some caveats

```
// JavaScript
function Cow(color) {
  this.color = color;
  this.foodType = "Hay";
}

// Inheritance Magic
Cow.prototype = Animal;

var c = new Cow("White");
c.feed();                          // "Hay"
var test = c instanceof Animal; // error
var test2 = c instanceof Cow;   // true
```

Inheritance in JavaScript "Classes"

- **What Else?**
  - Private
    - via closures and local variables
  - Protected
    - not supported
  - Overloaded Constructors
    - No, but no overloaded functions so same

What About Interfaces?

- **Interfaces Aren't Necessary**
  - Get Comfortable with Duck Typing

```
// JavaScript
function sendEmail(r) {
  var to = r.email;
  var name = r.name;
}

var Owner = {
  name: "Shawn",
  email: "shawn@foo.com",
  phone: "404-555-1212"
};

sendEmail(Owner); // works
```

What About Interfaces?

- **Interfaces Aren't Necessary**
  - Get Comfortable with Duck Typing

```
// JavaScript
function sendEmail(r) {
  var to = r.email;
  var name = r.name;
}

function Customer(name, email) {
  this.name = name;
  this.email = email;
  this.balance = 0;
}
var c = new Customer("Bob", "bob@foo.com");
sendEmail(c); // also works
```

Object Reflection

- **Enumerating Members**
  - Simplest Version of Reflection

```
// JavaScript
var cust = {
  name: "Shawn",
  "company name": "Wilder Minds",
  sendEmail: function() { ... }
};

for (var prop in cust) {
  alert(prop);       // property name
  alert(cust[prop]) // property value
}
```

Object Reflection

- **Detecting Properties**

```
// JavaScript
var c = new Customer();

var has = c.hasOwnProperty("name");
var isEnum = c.propertyIsEnumerable("name");
```

Extension Methods

- **Prototype Can Be Used to Add Extension Methods**
  - Add to Existing Type's Prototype

```
// JavaScript
Array.prototype.calculateCount = function() {
  return this.length;
};

var a = ["one", "two"];

var count = a.calculateCount();
```

Object Patterns

- **Many Patterns Exists for Object Creation**
  - Prototype Pattern (seen here)
  - Module Pattern
  - Revealing Prototype Pattern
  - Revealing Module Pattern
  - Etc.

- **Dan Wahlin's great "Structuring JavaScript" course for more**

Summary

- **OOP in JavaScript**
  - Objects and "Classes" represent the basic data structures in JavaScript
  - You can use your OOP skills in JavaScript
  - "Classes" aren't real, but they can model real-world relationships
  - Not all features of C# Classes are supported
  - But many features aren't necessary