JavaScript for C# Developers

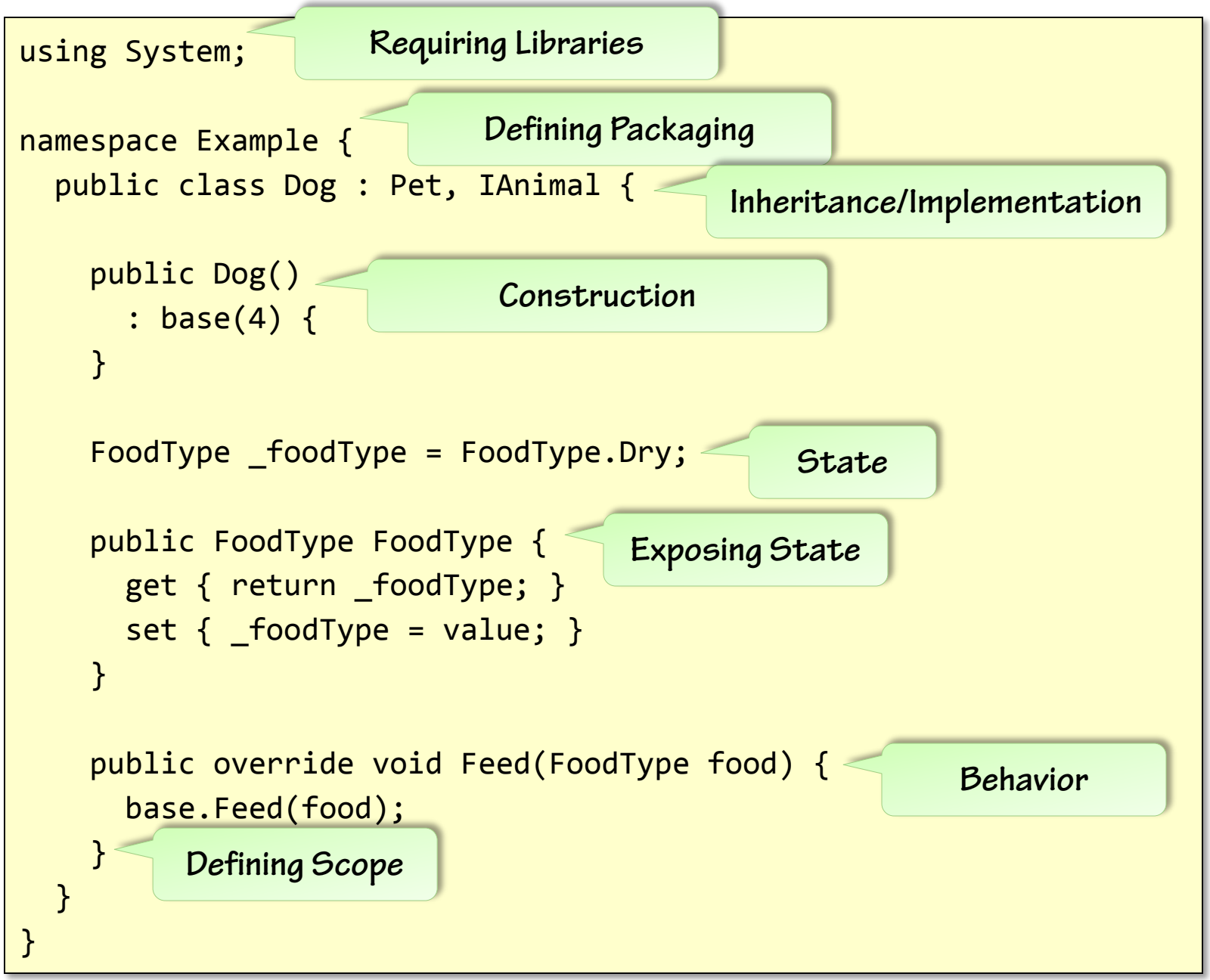
Module 1: JavaScript Basics

Shawn Wildermuth
Wilder Minds
wilderminds.com

pluralsight
hardcore developer training

Agenda

- **JavaScript and C#**
  - Runtime Environments
  - Comparing Languages
  - Strong and Loose Typing
  - Dynamic Typing
  - Language Basics
  - Types


using System;
Requiring Libraries
namespace Example {
Defining Packaging
public class Dog : Pet, IAnimal {
Inheritance/Implementation
public Dog()
: base(4) {
}
Construction
FoodType _foodType = FoodType.Dry;
State
public FoodType FoodType {
get { return _foodType; }
set { _foodType = value; }
}
Exposing State
public override void Feed(FoodType food) {
base.Feed(food);
}
Behavior
}
Defining Scope
}

Comparing Runtime Environments

- **C# and .NET**
  - Common Language Runtime (CLR) provides services:
    - Memory Management (e.g. Garbage Collection)
    - Just-in-Time Compilation
    - Common Type System
    - (et al.)

Comparing Runtime Environments

- **JavaScript Engines**
  - Depends on Browser
    - (e.g. V8 in Chrome, Chakra in IE, *Monkey in Firefox)
    - Services
      - Memory Management
      - Just-in-Time Compilation (for the most part)
      - Type System
      - (et al.)

Comparing Runtime Environments

- **JavaScript and CLR**
  - The services are similar
    - Should be able to just write code and trust the environment
    - Garbage Collection is Good Enough (for the most part)
    - JIT gives you performance
      - Though JS JIT'ing is based on what browser you're running in

Comparing the Languages

C#	JavaScript
• Strongly-Typed	• Loosely-typed
• Static	• Dynamic
• Classical Inheritance	• Prototypal
• Classes	• Functions
• Constructors	• Functions
• Methods	• Functions

* From Douglas Crawford's "Classical Inheritance in JavaScript" (http://shawnw.me/jsinheritance)

Strong and Loose Typing

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

Variable Declaration

Strong and Loose Typing

- **Strong Typing**
  - Types are defined by names and typically static structure
  - Compiler does the checks for you
  - Compiler can infer the type when necessary

```
// C#
var x = 0; // Infers type to be int

bool isInt = x is int; // true

x = new object(); // Compilation Error
```

Strong and Loose Typing

- **Loose Typing**
  - Types are typically defined by structure not by identity
  - Runtime checks
  - Type is dynamic

```
// JavaScript
var x = 0; // creates variable x that holds a number

var isNumber = typeof x == "number"; // Works but limited

x = new Object(); // no problem, redefines x with new type
```

Strong and Loose Typing

- **Strong Typing and OOP**
  - Inheritance and implementations are part of type identity

```
// C#
class Dog : Animal, IMoveable {...}
class Car : Vehicle, IMoveable {...}

// Accepts Any Animal (base class has Feed Method)
void Feed(Animal ani) { ani.Feed(); }

// Accepts any object that implements Interface
void Move(IMoveable object) { object.Move(); }
```

Strong and Loose Typing

- **Loose Typing and OOP**
  - Type is less important, but shape is important
  - Classic Duck Typing

```
// JavaScript

// Accepts any object
// (must implement Feed function)
function Feed(ani) { ani.Feed(); }

// Accepts any object
// (must implement Move method)
function Move(object) { object.Move(); }
```

Dynamic Typing

- **Dynamic Typing Can be Powerful**
  - "With much power comes much responsibility"

```
// JavaScript
var x = {
  name: "Shawn",
  city: "Atlanta"
};

x.phone = "404-555-1212";
x.makeCall = function () {
  callSomeone(this.phone);
};
```

Language Basics

- **Global Scope**
  - Objects at root are 'global'

```
// C#
public class MyApp
{
  static void SomeFunction(int x, int y)
  {
  }

  static void Main()
  {
    var x = 1;
    SomeFunction(5, x):
  }
}
```

Language Basics

- **Type Coalescing**
  - JavaScript Wants to Coalesce Values

```
// JavaScript
"test " + "me"     // test me
"test " + 1        // test 1
"test " + true     // test true
"test " + (1 == 1) // test true
100 + "25"         // 10025
```

Language Basics

- **Most Operators Identical to .NET, except…**
  - Equality/NotEqual (==, !=)
    - Determines equality with coalescing (if necessary)

```
// JavaScript
"hello" == "hello"; // true
1 == 1;             // true
1 == "1";           // true
```

Language Basics

- **JavaScript's Identically Equality Operators (===, !==)**
  - Similar to .Equal()
    - Determines equality without coalescing

```
// JavaScript
"hello" == "hello";       // true
1 == 1;                   // true
1 == "1";                 // true
1 === "1";                // false
1 !== "1";                // true
1 === 1.00000000000001;   // false
1 === 1.000000000000001; // true
```

Types

- **Primitives**
  - JavaScript has basic types
    - "Value Types"
      - boolean
      - string
      - number
    - "Reference Type"
      - object
    - "Delegate Type"
      - function
    - Special
      - "undefined"

Types

- **Type Detection**
  - typeof operator

```
// JavaScript
var x = 1;
var typeName = typeof x; // "number"
```

Types

- Special Types
  - null
  - undefined

```
// JavaScript
var a = null; // "null"
var b = c;    // "undefined"
```

Types

- "Value Types"

```
// JavaScript
var a = typeof 1;              // "number"
var b = typeof 1.0;            // "number"
var c = typeof true;           // "boolean"
var d = typeof false;          // "boolean"
var e = typeof "hello world"; // "string"
```

Types

- **The Number Type**
  - Holds IEEE-754 format
    - (This format prone to rounding errors)
    - Integers and Floating Point Numbers

```
// JavaScript
var a = 1;          // integer
var b = 1.5;        // floating point
var c = 070;        // integer (in octal)
var d = 0xffff;     // integer (in hex)
var e = 1.34e6;     // Scientific Notation (1340000)
var f = 10.0;       // integer (optimization)
```

Types

- **number**
  - Special Values

```
// JavaScript
var a = Number.MIN_VALUE;
var b = Number.MAX_VALUE;
var c = Number.POSITIVE_INFINITY;
var d = Number.NEGATIVE_INFINITY;

var fail = 10/"zero";       // Not a Number (NaN)
var test1 = NaN == NaN;     // false, huh?
var test2 = isNaN(NaN);     // true
var test3 = isNaN(fail);    // true
var test4 = isNaN(10);      // false
var test5 = isNaN("10");    // false
var test6 = isNaN("fail"); // true
```

Types

- **The Boolean Type**
  - *true* and *false* only (e.g. not a number)
    - Flow-control statements apply coalescing to boolean

```
// JavaScript
if (true) {}    // true
if (false) {}   // false
if ("hello") {} // true
if ("") {}      // false
if (25) {}      // true
if (0) {}       // false
if (10/0) {}    // false (NaN)

var a = null;
if (a) {}       // false
if (c) {}       // false (undefined)
```

Types

- **The String Type**
  - Immutable (like .NET)

```
// JavaScript
var s = "String";        // Simple Strings
var t = 'String';        // Either delimiter

var u = "One" + "Two";   // Immutable
var single = u[3];       // 'T'

log(u.length);           // 6
var d = "בָּרוּךְ";          // Unicode
log(d.length);           // 8 (count of 8 bits)
```

Types

- "Reference Types"

```
// JavaScript
var a = new Object();           // "object"
var b = new Array();            // "object"
```

Types

- **Object Literals**
  - Shortcut for creating data structures

```
// JavaScript
var data = {
  name: "hello",
  "some value": 1
};

var more = {
  "moniker": "more data",
  height: 6.0,
  subData: {
    option: true,
    flavor: "vanilla"
  }
};
```

Types

- Array Literals
  - Shortcut for creating collections

```
// JavaScript
var array = [ "hello", "goodbye" ];

var coll = [{
  "moniker": "more data",
  height: 6.0,
  subData: {
    option: true,
    flavor: "vanilla"
  }
}];
```

Types

- **Array**
  - Untyped Collection

```
// JavaScript
var c = [];

c.length;                       // Um, yeah…

c.push({});                     // add to end
var obj = c.pop()               // remove from end
c.shift();                      // remove from beginning
c.unshift({});                  // add to beginning

var where = c.indexOf(obj); // positional access
where = c.lastIndexOf(obj);

// etc. slice, splice, concat
```

Types

- **Functions**
  - Yup, a data type
  - Just like Func<>

```
// JavaScript
var f = function (arg1, arg2) {
};
f(1,2);

var o = {
  name: "Shawn",
  sing: function (song) {
  }
};
o.sing("happy birthday");
```

Summary

- **JavaScript and .NET**
  - While there is much in common, there are definite differences
  - The differences between Strong and Loose typing are key
  - The basics of JavaScript are important to expecting the right behavior
  - The type system differences aren't too surprising