Scripting for Multimedia

LECTURE 10: CREATING JAVASCRIPT OBJECTS

Using object-oriented terminology

- object
- encapsulation, inheritance and polymorphism
- property and method

Understanding the JavaScript object-oriented caveat

- JS is a prototype-based, object-oriented language
  - Everything is an object
- JavaScript is not class-based
  - There is no *class* keyword in JS

Using the JavaScript object literal pattern

- Use the object literal syntax to create an object
- Object literals create an object from nothing
- No prototype object is associated with the created object

```
var car1 = {
    year: 2000,
    make: 'Ford',
    model: 'Fusion',
    getInfo: function () {
        return 'Vehicle: '+this.year+' '+this.make+' '+this.model;
    }
};
```

Using the JavaScript object literal pattern

- Use the object literal syntax to create an object
- Object literals create an object from nothing
- No prototype object is associated with the created object

```
var car2 = {
    year: 2010,
    make: 'BMW',
    model: 'Z4',
    getInfo: function () {
        return 'Vehicle: '+this.year+' '+this.make+' '+this.model;
    }
};
```

Using the JavaScript object literal pattern

- Assign an array to a property

```
var car1 = {
    year: 2000,
    make: 'Ford',
    model: 'Fusion',
    repairs: ['repair1', 'repair2', 'repair3'],
    getInfo: function () {
        return 'Vehicle: '+this.year+' '+this.make+' '+this.model;
    }
};
```

- This is one of the easiest ways to create an object

Creating dynamic objects by using the factory pattern

- JS has an Object type

```
function getVehicle(theYear, theMake, theModel) {
    var vehicle = new Object();
    vehicle.year = theYear;
    vehicle.make = theMake;
    vehicle.model = theModel;
    vehicle.getInfo: function () {
        return 'Vehicle: '+this.year+' '+this.make+' '+this.model;
    }
    return vehicle;
};
```

- The encapsulation of the code to create an object is commonly referred to as using the *factory pattern*

Creating dynamic objects by using the factory pattern

- You can create multiple instances of Object and add properties dynamically to each instance
  - The actual type is Object
- Although the getVehicle function encapsulates the object creation, the properties are all public

Creating a class

- There is no class keyword in JS
- Some problem exists in the following code

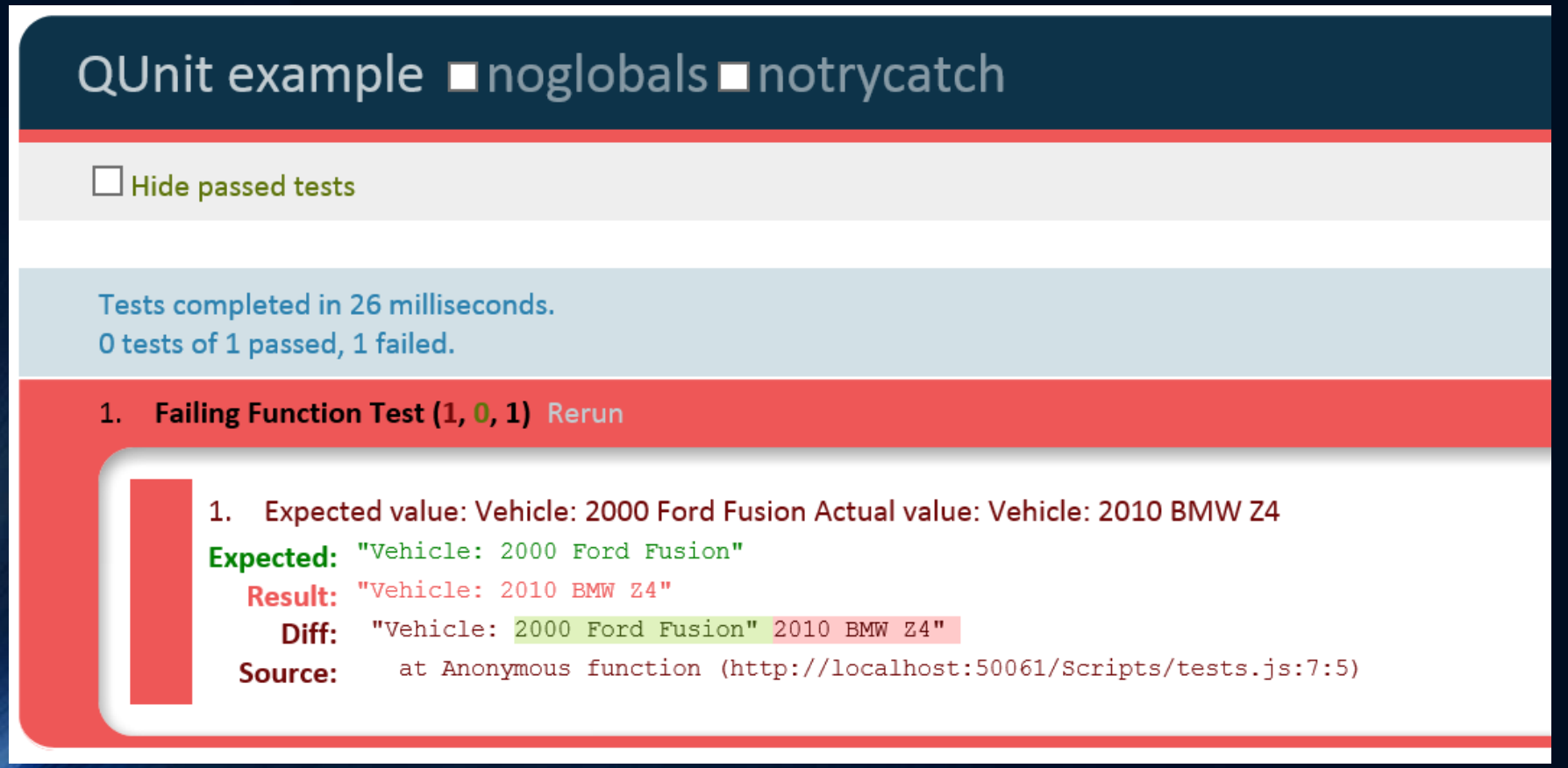
```
function getVehicle(theYear, theMake, theModel) {
    var vehicle = new Object();
    vehicle.year = theYear;
    vehicle.make = theMake;
    vehicle.model = theModel;
    vehicle.getInfo = function () {
        return 'Vehicle: '+this.year+' '+this.make+' '+this.model;
    };
    return vehicle;
};
```

Creating a class

- Test code

```
test ("Failing Function Test", function() {
    expect(1);
    Vehicle(2000, 'Ford', 'Fusion');
    Vehicle(2010, 'BMW', 'Z4');
    var expected = 'Vehicle: 2000 Ford Fusion';
    var actual = getInfo();
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
});
```

Creating a class



Creating a class

- To fix the problem:

```
function Vehicle(theYear, theMake, theModel) {
    var year = theYear;
    var make = theMake;
    var model = theModel;
    this.getInfo = function () {
        return 'Vehicle: '+year+' '+make+' '+model;
    }
};
```

Creating a class

- Test code

```
test ("Encapsulation Test", function() {
    expect(2);
    var car1 = new Vehicle(2000, 'Ford', 'Fusion');
    var car2 = new Vehicle(2010, 'BMW', 'Z4');
    var expected = 'Vehicle: 2000 Ford Fusion';
    var actual = car1.getInfo();
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
    expected = 2000;
    actual = year;
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
});
```

Privileged method

Creating a class

Tests completed in 35 milliseconds.
1 tests of 2 passed, 1 failed.

1. **Encapsulation Test (1, 1, 2)** Rerun

1. Expected value: Vehicle: 2000 Ford Fusion Actual value: Vehicle: 2000 Ford Fusion

Expected: `"Vehicle: 2000 Ford Fusion"`

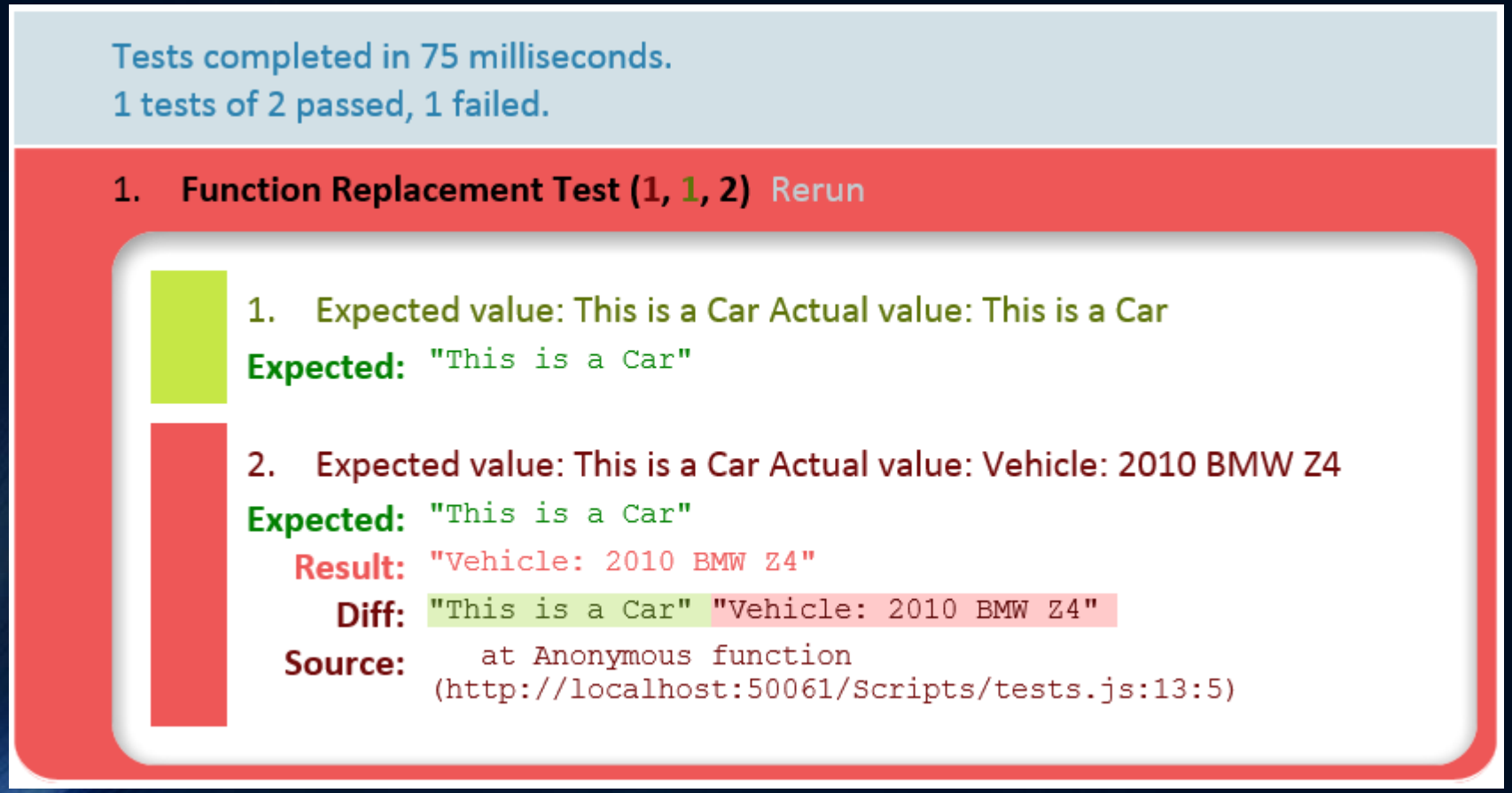
2. Died on test #2: 'year' is undefined - { "description": "'year' is undefined", "number": -2146823279, "stack": "ReferenceError: 'year' is undefined at Anonymous function (http://localhost:50061/Scripts/tests.js:9:5) at Test.prototype.run (http://localhost:50061/Scripts/qunit.js:102:4) at Anonymous function (http://localhost:50061/Scripts/qunit.js:232:5) at process (http://localhost:50061/Scripts/qunit.js:869:4) at Anonymous function (http://localhost:50061/Scripts/qunit.js:408:5)" }

Creating a class

- The Vehicle function is a constructor function

```
test ("Function Replacement Test", function() {
    expect(2);
    var car1 = new Vehicle(2000, 'Ford', 'Fusion');
    var car2 = new Vehicle(2010, 'BMW', 'Z4');
    car1.getInfo = function() {
        return 'This is a Car';
    };
    var expected = 'This is a Car';
    var actual = car1.getInfo();
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
    expected = 'This is a Car';
    actual = car2.getInfo();
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
});
```

Creating a class



Using the prototype property

- In JS, everything (including function) is an Object type, which has a prototype property
- You can replace the function shared by all instances using prototype pattern

```
function Vehicle(theYear, theMake, theModel) {
    this.year = theYear;
    this.make = theMake;
    this.model = theModel;
}
Vehicle.prototype.getInfo = function () {
    return 'Vehicle: '+year+' '+make+' '+model;
}
```

Using the prototype property

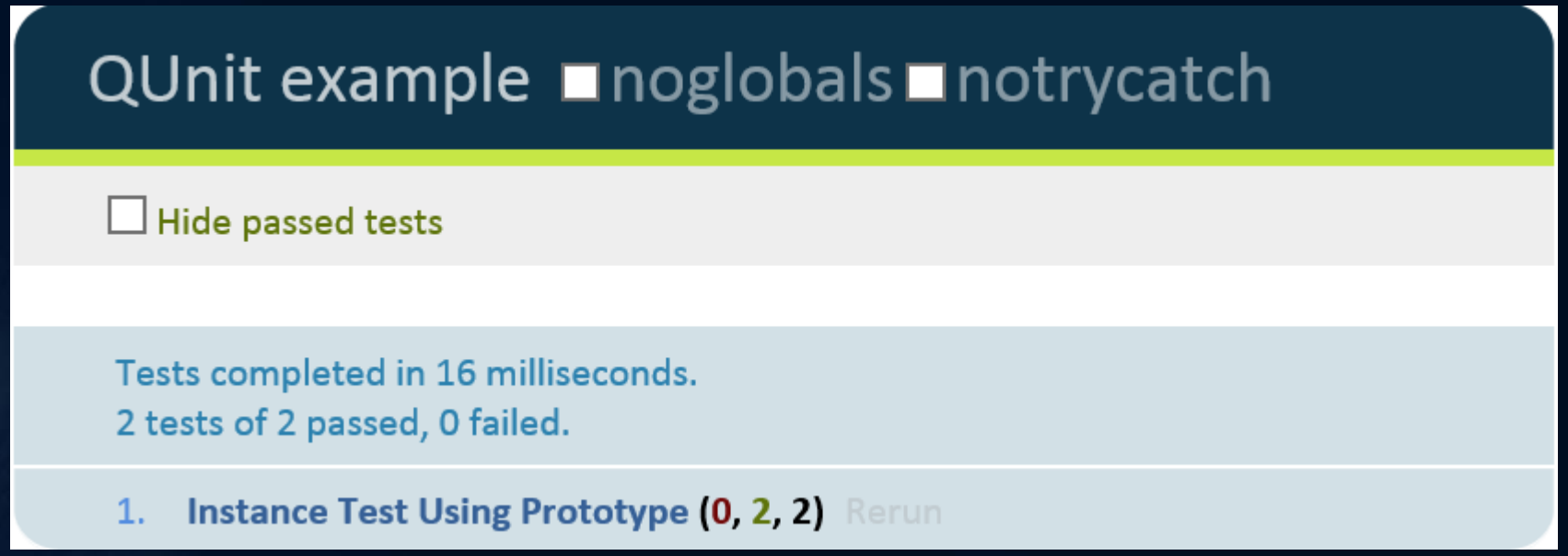
- The prototype is an object containing properties and methods available to all instances
- It is specified externally to the constructor function
  - It doesn't have access to private variables
  - You must expose the data for the prototype

Using the prototype property

- Test code

```
test ("Instance Test Using Prototype", function() {
    expect(2);
    var car1 = new Vehicle(2000, 'Ford', 'Fusion');
    var car2 = new Vehicle(2010, 'BMW', 'Z4');
    Vehicle.prototype.getInfo = function () {
        return 'Car: '+this.year+' '+this.make+' '+this.model
    }
    var expected = 'Car: 2000 Ford Fusion';
    var actual = car1.getInfo();
    equal(actual, expected, 'Expected value: '+expected+    '
          Actual value: ' + actual);
    expected = 'Car: 2010 BMW Z4';
    actual = car2.getInfo();
    equal(actual, expected, 'Expected value: '+expected+    '
          Actual value: ' + actual);
});
```

Using the prototype property



Using the prototype property

- Remember that
  - the prototype is defined externally to the constructor function
  - and all properties must public

✓ Quick check

You want to add a method to all instances of Vehicle. How do you do this?

Debating the prototype/private compromise

- Create a getter function for each object to retrieve the private data
  - Keep it as small as possible

```
function Vehicle(theYear, theMake, theModel) {
    var year = theYear;
    var make = theMake;
    var model = theModel;
    this.getYear = function () {return year;};
    this.getMake = function () {return make;};
    this.getModel = function () {return model};
}
Vehicle.prototype.getInfo = function () {
    return 'Vehicle: '+ this.getYear()+' '+this.getMake()+'
'+this.getModel();
}
```

Debating the prototype/private compromise

- Test code

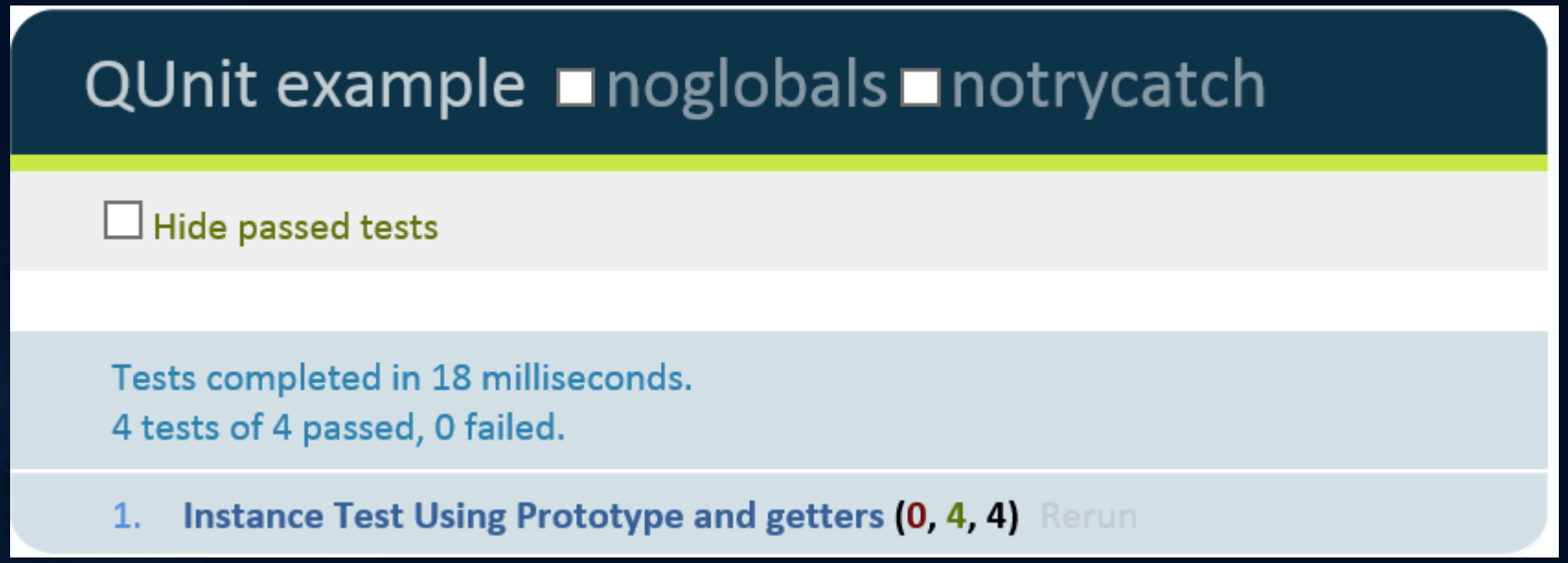
```
test ("Instance Test Using Prototype and getters",
function() {
    expect(4);
    var car1 = new Vehicle(2000, 'Ford', 'Fusion');
    var car2 = new Vehicle(2010, 'BMW', 'Z4');
    var expected = 'Vehicle: 2000 Ford Fusion';
    var actual = car1.getInfo();
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
    expected = 'Vehicle: 2010 BMW Z4';
    actual = car2.getInfo();
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
```

Debating the prototype/private compromise

- Test code (Cont.)

```
....
Vehicle.prototype.getInfo = function () {
    return 'Car Year: '+this.getYear()+' Make: '+
        this.getMake()+' Model: '+this.getModel();
};
expected = 'Car Year: 2000 Make: Ford Model: Fusion';
actual = car1.getInfo();
equal(actual, expected, 'Expected value: '+expected+
      ' Actual value: ' + actual);
expected = 'Car Year: 2010 Make: BMW Model: Z4';
actual = car2.getInfo();
equal(actual, expected, 'Expected value: '+expected+
      ' Actual value: ' + actual);
});
```

Debating the prototype/private compromise



Debating the prototype/private compromise

- Remember to create only getter methods as needed and to keep them small and concise

✓ Quick check

How can you expose private data as read-only?

Implementing namespaces

- Global namespace pollution
- JS doesn't have a namespace keyword

```
var vehicleCount = 5;
var vehicles = new Array();
function Car() {}
function Truck() {}
var repair = {
    description: 'changed spark plugs',
    cost: 100
}
```

```
var myApp = {};

myApp.vehicleCount = 5;

myApp.vehicles = new Array();
myApp.Car = function () {}
myApp.Truck = function () {}

myApp.repair = {
    description: 'changed
spark plugs',
    cost: 100
}
```

Implementing namespaces

- A namespace is created by creating an object
  - All its members are globally accessible
- You can also create a namespace if it doesn't already exist

```
var myApp = myApp || {};
```

```
var myApp = {};

myApp.vehicleCount = 5;

myApp.vehicles = new Array();
myApp.Car = function () {}
myApp.Truck = function () {}

myApp.repair = {
    description: 'changed
spark plugs',
    cost: 100
}
```

Implementing namespaces

- You can make some member of the namespace private and some public

```
(function () {
    this.myApp = this.myApp || {};
    var ns = this.myApp;
    var vehicleCount = 5;
    var vehicles = new Array();
    ns.Car = function () {}
    ns.Truck = function () {}
    var repair = {
        description: 'changed spark plugs',
        cost: 100
    };
}());
```

Implementing namespaces

- An IIFE (immediately invoked function expression) is an anonymous function expression that has a set of parentheses at the end of it
  - The anonymous function expression is wrapped in parentheses to tell the JS interpreter that the function isn't only being defined but also being executed when the file is loaded

Implementing namespaces

- Create a sub-namespace

```
(function () {
    this.myApp = this.myApp || {};
    var rootNs = this.myApp;
    rootNs.billing = rootNs.billing || {};
    var ns = rootNs.billing;

    var taxRate= .05;
    ns.Invoice = function () {}
}());
```

Implementing inheritance

- "Is a" relationship allows code reuse
- Vehicle example
  - base class

```
var Vehicle = (function () {
    function Vehicle(year, make, model) {
        this.year = year;
        this.make = make;
        this.model = model;
    }
    Vehicle.prototype.getInfo = funtion () {
        return this.year+' '+this.make+' '+this.model;
    };
    Vehicle.prototype.startEngine = function () {
        return 'Vroom';
    }
}) ();
```

Implementing inheritance

- The class is wrapped in an IIFE
- The wrapper encapsulates the function and the Vehicle prototype
- Data is public

Implementing inheritance

- To create Vehicle objects, use the *new* keyword with the Vehicle variable

```
test ("Vehicle Inheritance Test", function() {
    expect(2);
    var v = new Vehicle(2012, 'Toyota', 'Rav4');
    var actual = v.getInfo();
    var expected = '2012 Toyota Rav4';
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
    actual = v.startEngine();
    expected = 'Vroom';
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
});
```

Implementing inheritance

- Now create child classes for Car and Boat that inherit from Vehicle
  - Write an IIFE but pass Vehicle into the IFFE

```
var Car = (function (parent) {

}) (Vehicle);
```

- Vehicle here is the *Vehicle* variable, not Vehicle function
- Vehicle is passed into the IIFE and is available inside the IIFE as *parent*

Implementing inheritance

- Now create child classes for Car and Boat that inherit from Vehicle
  - Write an IIFE but pass Vehicle into the IFFE

```
var Car = (function (parent) {

}) (Vehicle);
```

- The functions for Car can be added inside the IIFE
- Inside the function, add any additional properties
- In the function, call the parent class's constructor for Car to allocate memory slots for *year*, *make* and *model*

Implementing inheritance

- Call the parent constructor function

```
var Car = (function (parent) {
    function Car(year, make, model) {
        this.wheelQuantity = 4;
        parent.call(this, year, make, model);
    }
    return Car;
}) (Vehicle);
```

- The *this* object is the Car object, so the call to the parent constructor function creates year, make, and model on the Car object

Implementing inheritance

- Set up inheritance
  - getInfo and startEngine are not inherited

```
var Car = (function (parent) {
    Car.prototype = new Vehicle();
    Car.prototype.constructor = Car;
    function Car(year, make, model) {
        this.wheelQuantity = 4;
        parent.call(this, year, make, model);
    }
    return Car;
}) (Vehicle);
```

- The inheritance is accomplished by changing the Car prototype object to be a new Vehicle object
  - The prototype is the object that is cloned to create the new object
  - By default, the prototype is of type Object

Implementing inheritance

- You can add more methods into Car

```
var Car = (function (parent) {
    Car.prototype = new Vehicle();
    Car.prototype.constructor = Car;
    function Car(year, make, model) {
        this.wheelQuantity = 4;
        parent.call(this, year, make, model);
    }
    return Car;
    Car.prototype.getInfo = function () {
        return 'Vehicle Type: Car
            '+parent.prototype.getInfo.call(this);
    }
}) (Vehicle);
```

Implementing inheritance

- Test code

```
test ("Car Inheritance Test", function() {
    expect(6);
    var c = new Car(2012, 'Toyota', 'Rav4');
    var actual = c.year();
    var expected = '2012';
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
    actual = c.make();
    expected = 'Toyota';
    equal(actual, expected, 'Expected value: '+expected+
          ' Actual value: ' + actual);
    actual = c.model();
    expected = 'Rav4';
```

Implementing inheritance

- Test code (Cont.)

```
        equal(actual, expected, 'Expected value: '+expected+
              ' Actual value: ' + actual);
        actual = c.wheelQuantity();
        expected = 4;
        equal(actual, expected, 'Expected value: '+expected+
              ' Actual value: ' + actual);
        actual = c.getInfo();
        expected = 'Vehicle Type: Car 2012 Toyota Rav4';
        equal(actual, expected, 'Expected value: '+expected+
              ' Actual value: ' + actual);
        actual = c.startEngine();
        expected = 'Vroom';
        equal(actual, expected, 'Expected value: '+expected+
              ' Actual value: ' + actual);
    });
```