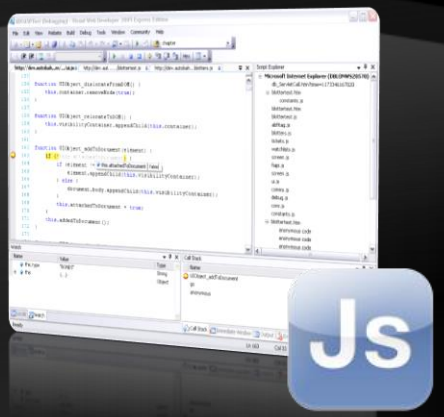
Telerik Academy



Classical OOP in JavaScript

The way of Object-oriented Ninja

Telerik Software Academy
Learning & Development Team
http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

Object-oriented Design

Telerik Academy

Object-oriented Programming

- OOP means that the application/program is constructed as a set of objects
  - Each object has its purpose
  - Each object can hold other objects
- JavaScript is prototype-oriented language
  - Uses prototypes to define its properties
  - Does not have definition for class or constructor
    - ECMAScript 1.6 introduces classes

Telerik Academy

OOP in JavaScript

Telerik Academy

OOP in JavaScript

- JavaScript is dynamic language
  - No such things as types and polymorphism
- JavaScript is also highly expressive language
  - Most things can be achieved in many ways
- That is why JavaScript has many ways to support OOP
  - Classical/Functional, Prototypal
  - Each has its advantages and drawbacks
  - Usage depends on the case

Telerik Academy

Classical OOP

Classical OOP

- JavaScript uses functions to create objects
  - It has no definition for class or constructor
- Functions play the role of object constructors
  - Create/initiate object by calling the function with the "new" keyword

```
function Person(){}
var gosho = new Person(); //instance of Person
var maria = new Person(); //another instance of Person
```

Telerik Academy

Creating Objects

- When using a function as an object constructor it is executed when called with new

```
function Person(){}
var personGosho = new Person(); //instance of Person
var personMaria = new Person(); //instance of Person
```

  - Each of the instances is independent
    - They have their own state and behavior
- Function constructors can take parameters to give instances different state

Creating Objects

- Function constructor with parameters
  - Just a regular function with parameters, invoked with new

```
function Person(name,age){
  console.log("Name: " + name + ", Age: " + age);
}
var personGosho = new Person("Georgi",23);
//logs:
//Name: Georgi, Age: 23
var personMaria = new Person("Maria",18);
//logs:
//Name: Maria, Age: 18
```

Telerik Academy

Function Constructors

Live Demo

Telerik Academy

Prototypes

Telerik Academy

The prototype Object

- JavaScript is prototype-oriented programming language
  - Every object has a hidden property prototype
  - Its kind of its parent object
- Prototypes have properties available to all instances
  - The object type is the parent of all objects
    - Every object inherits object
    - All objects has `toString()` method

Telerik Academy

The prototype Object (2)

- When adding properties to a prototype, all instances will have these properties

```
//adding a repeat method to the String type
String.prototype.repeat = function (count) {
  var str,
      pattern,
      i;
  pattern = String(this);
  if (!count) {
    return pattern;
  }
  str = '';
  for (i = 0; i < count; i += 1) {
    str += pattern;
  }
  return str;
};
```

The prototype Object (2)

- When adding properties to a prototype, all instances will have these properties

```
//adding a repeat method to the String type
String.prototype.repeat = function (count) {
  var str,
      pattern,
      i;
  pattern = String(this);
  if (!count) {
    return pattern;
  }
  str = '';
  for (i = 0; i < count; i += 1) {
    str += pattern;
  }
  return str;
};
```

Add method to all strings

The prototype Object (2)

- When adding properties to a prototype, all instances will have these properties

```
//adding a repeat method to the String type
String.prototype.repeat = function (count) {
  var str,
      pattern,
      i;
  pattern = String(this);
  if (!count) {
    return pattern;
  }
  str = '';
  for (i = 0; i < count; i += 1) {
    str += pattern;
  }
  return str;
};
```

Add method to all strings

Here this means the string

The prototype Object (2)

- When adding properties to a prototype, all instances will have these properties

```
//adding a repeat method to the String type
String.prototype.repeat = function (count) {
  var str,
      pattern,
      i;
  pattern = String(this);
  if (!count) {
    return pattern;
  }
  str = '';
  for (i = 0; i < count; i += 1) {
    str += pattern;
  }
  return str;
};
```

Add method to all strings

Here this means the string

```
//use it with:
'-'.repeat(25);
```

Telerik Academy

Prototypes

Live Demo

Object Members

- Objects can also define custom state
  - Custom properties that only instances of this type have
- Use the keyword `this`
  - To attach properties to object

```
function Person(name,age){
  this.name = name;
  this.age = age;
}
var personMaria = new Person("Maria",18);
console.log(personMaria.name);
```

Telerik Academy

Object Members (2)

- Property values can be either variables or functions
  - Functions are called methods

```
function Person(name,age){
  this.name = name;
  this.age = age;
  this.sayHello = function(){
    console.log("My name is " + this.name +
                " and I am " + this.age + "-years old");
  }
}
var maria = new Person("Maria",18);
maria.sayHello();
```

Telerik Academy

Object Members

Live Demo

Attaching Methods

- Attaching methods inside the object constructor is a tricky operation
  - Its is slow
  - Every object has a function with the same functionality, yet different instance
  - Having the function constructor

```
function Person(name, age){
  this.introduce = function(){
    return 'Name: ' + name +
           ', Age: ' + age;
  };
}
```

Attaching Methods

- Attaching methods inside the object constructor is a tricky operation
  - Its is slow
  - Every object has a function with the same functionality, yet different instance
  - Having the function constructor

```
function Person(name, age){
  this.introduce = function(){
    return 'Name: ' + name +
           ', Age: ' + age;
  };
}
```

And the code

```
var p1 = new Person();
var p2 = new Person();
console.log (p1 === p2);
```

Telerik Academy

Attaching Methods

- Attaching methods inside the object constructor is a tricky operation
  - Its is slow
  - Every object has a function with the same functionality, yet different instance
  - Having the function constructor

```
function Person(name, age){
  this.introduce = function(){
    return 'Name: ' + name +
           ', Age: ' + age;
  };
}
```

And the code

```
var p1 = new Person();
var p2 = new Person();
console.log (p1 === p2);
```

Logs 'false'

Telerik Academy

Different Method Instances

Live Demo

Telerik Academy

Better Method Attachment

- Instead of attaching the methods to this in the constructor

```
function Person(name,age){
  //…
  this.sayHello = function(){
    //…
  }
}
```

Telerik Academy

Better Method Attachment

- Instead of attaching the methods to this in the constructor

```
function Person(name,age){
  //…
  this.sayHello = function(){
    //…
  }
}
```

- Attach them to the prototype of the constructor

```
function Person(name,age){
}
Person.prototype.sayHello =
          function(){
            //…
          }
```

Telerik Academy

Better Method Attachment

- Instead of attaching the methods to this in the constructor

```
function Person(name,age){
  //…
  this.sayHello = function(){
    //…
  }
}
```

- Attach them to the prototype of the constructor

```
function Person(name,age){
}
Person.prototype.sayHello =
          function(){
            //…
          }
```

- And each method is created exactly once

Telerik Academy

Attaching Methods to the Prototype

Live Demo

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
- Attaching to prototype

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages ✓
- Attaching to prototype

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages ✓
- Attaching to prototype
  - Using JavaScript as it is meant ✓

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages ✓
- Attaching to prototype
  - Using JavaScript as it is meant ✓

JavaScript is NO other language

It should be treated like a first class language

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages ✓
  - Hidden data ✓
- Attaching to prototype
  - Using JavaScript as it is meant ✓

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages 
  - Hidden data 
- Attaching to prototype
  - Using JavaScript as it is meant 
  - No hidden data 

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages 
  - Hidden data 
- Attaching to prototype
  - Using JavaScript as it is meant 
  - No hidden data 

Hidden data is not such a big problem

Prefix "hidden" data with _(underscore) and be done with it

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages 
  - Hidden data 
  - Not good performance 
- Attaching to prototype
  - Using JavaScript as it is meant 
  - No hidden data 

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages 
  - Hidden data 
  - Not good performance 
- Attaching to prototype
  - Using JavaScript as it is meant 
  - No hidden data 
  - A way better performance 

Telerik Academy

Pros and Cons When Attaching Methods

- Attaching to this
  - Code closer to other languages 
  - Hidden data 
  - Not good performance 
- Attaching to prototype
  - Using JavaScript as it is meant 
  - No hidden data 
  - A way better performance 

Performance is a big deal!

It should be taken into serious consideration

Telerik Academy

The this Object

Telerik Academy

The this Object

- this is a special kind of object
  - It is available everywhere in JavaScript
    - Yet it has a different meaning
- The this object can have two different values
  - The parent scope
    - The value of this of the containing scope
    - If none of the parents is object, its value is window
  - A concrete object
    - When using the new operator

Telerik Academy

this in Function Scope

- When executed over a function, without the new operator
  - this refers to the parent scope

```
function Person(name) {
  this.name = name;
  this.getName = function getPersonName() {
    return this.name;
  }
}
var p = new Person("Gosho");
var getName = p.getName;
console.log(p.getName()); //Gosho
console.log(getName()); //undefined
```

Telerik Academy

this in Function Scope

- When executed over a function, without the new operator
  - this refers to the parent scope

```
function Person(name) {
  this.name = name;
  this.getName = function getPersonName() {
    return this.name;
  }
}
var p = new Person("Gosho");
var getName = p.getName;
console.log(p.getName()); //Gosho
console.log(getName()); //undefined
```

Here this means the Person object

this in Function Scope

- When executed over a function, without the new operator
  - this refers to the parent scope

```
function Person(name) {
  this.name = name;
  this.getName = function getPersonName() {
    return this.name;
  }
}
var p = new Person("Gosho");
var getName = p.getName;
console.log(p.getName()); //Gosho
console.log(getName()); //undefined
```

Here this means the Person object

Here this means its parent scope (window)

Telerik Academy

The this function object

Live Demo

Function Constructors

- JavaScript cannot limit function to be used only as constructors
  - JavaScript was meant for a simple UI purposes

```
function Person(name) {
  var self = this;
  self.name = name;
  self.getName = function getPersonName() {
    return self.name;
  }
}
var p = Person("Peter");
```

Telerik Academy

Function Constructors

- JavaScript cannot limit function to be used only as constructors
  - JavaScript was meant for a simple UI purposes

```
function Person(name) {
  var self = this;
  self.name = name;
  self.getName = function getPersonName() {
    return self.name;
  }
}
var p = Person("Peter");
```

What will be the value of `this`?

Telerik Academy

Function Constructors (2)

- The only way to mark something as contructor is to name it PascalCase
  - And hope that the user of you code will be so nice to call PascalCase-named functions with new

Telerik Academy

Invoking Function Constructors Without new

Live Demo

Function Constructor Fix

- John Resig (jQuery) designed a simple way to check if the function is not used as constructor:

```
function Person(name, age) {
  if (!(this instanceof arguments.callee)) {
    return new Person(name, age);
  }
  this._name = name;
  this._age = age;
}
```

Telerik Academy

Function Constructor Fix

- John Resig (jQuery) designed a simple way to check if the function is not used as constructor:

```
function Person(name, age) {
  if (!(this instanceof arguments.callee)) {
    return new Person(name, age);
  }
  this._name = name;
  this._age = age;
}
```

If this is not of type the function call the function with new

Telerik Academy

John Resig Constructor Fix

Live Demo

Telerik Academy

Function Constructors with Modules

Telerik Academy

Constructors with Modules

- Function constructors can be put inside a module
  - Introduces a better abstraction of the code
  - Allows to hide constants and functions
- In JavaScript functions are first-class objects, so they can be easily returned by a module

```
var Person = (function () {
  function Person(name) {
    //…
  }
  Person.prototype.walk = function (distance){ /*...*/ };
  return Person;
}());
```

Telerik Academy

Function Constructors with Modules

Live Demo

Telerik Academy

Hidden functions

What to do when we want to hide something?

Hidden Funcitons

- When a function constructor is wrapped inside a module:
  - The module can contain hidden functions
  - The function constructor can use these hidden functions
- Yet, to use these functions as object methods, we should use `apply` or `call`

Telerik Academy

Hidden Functions: Example

- Using hidden functions

```
var Rect = (function () {
  function validatePosition() {
    //…
  }
  function Rect(x, y, width, height) {
    var isPositionValid = validatePosition.call(this);
    if (!isPositionValid) {
      throw new Error('Invalid Rect position');
    }
  }

  Rect.prototype = { /* … */};
  return Rect;
}());
```

Telerik Academy

Hidden Functions: Example

- Using hidden functions

```
var Rect = (function () {
  function validatePosition() {
    //…
  }
  function Rect(x, y, width, height) {
    var isPositionValid = validatePosition.call(this);
    if (!isPositionValid) {
      throw new Error('Invalid Rect position');
    }
  }

  Rect.prototype = { /* … */};
  return Rect;
}());
```

This is not exposed from the module

Telerik Academy

Hidden Functions: Example

- Using hidden functions

```
var Rect = (function () {
  function validatePosition() {
    //…
  }
  function Rect(x, y, width, height) {
    var isPositionValid = validatePosition.call(this);
    if (!isPositionValid) {
      throw new Error('Invalid Rec
    }
  }

  Rect.prototype = { /* … */};
  return Rect;
}());
```

This is not exposed from the module

Use call() to invoke the function over this

Telerik Academy

Hidden Functions

Live Demo

Telerik Academy

Classical OOP in JavaScript

http://academy.telerik.com

Telerik Academy

Homework

1. Create a module for drawing shapes using Canvas. Implement the following shapes:
   - Rect, by given position (X, Y) and size (Width, Height)
   - Circle, by given center position (X, Y) and radius (R)
   - Line, by given from (X1, Y1) and to (X2, Y2) positions