f

Pain-free inheritance with F.js

JavaScript Inheritance

- JS is prototype-based
  - Existing objects are used as a blueprint, or prototype, for new objects
  - Results in a prototype chain that ends with Object
  - Method and property lookup happens by traversing the prototype chain

```
1  // Class: Person
2  function Person() {}
3
4  // Methods every Person should have
5  Person.prototype.a = function() { return "a"; };
6  Person.prototype.b = function() { return "b"; };
7
8  // Class: Child
9  function Child() {}
10
11 // inherit from Person
12 Child.prototype = new Person();
13
14 // Methods every Child should have
15 Child.prototype.c = function() { return "c"; };
16
17 // Create an instance of a child
18 var kid = new Child();
19
20 // Calls to kid.a actually call Person.prototype.a
21 kid.a(); // "a"
```

```
1  // Class: Person
2  function Person() {}
3  Person.prototype.a = function() { return "a"; };
4
5  // Class: Child
6  function Child() {}
7  Child.prototype = new Person();
8  Child.prototype.a = function() {
9      // Call the parent method by referring to its prototype
10     var parentReturnVal = Person.prototype.a.apply(this, arguments);
11     return parentReturnVal+"à";
12 };
13
14 // Create an instance of a child
15 var kid = new Child();
16
17 kid.a(); // "aà"
```

JavaScript Inheritance:
Superclass Methods

- Must know parent class and directly refer to its prototype
- Awkward syntax
- Doesn't work if immediate parent doesn't implement the method

JavaScript Inheritance

It's painful.

The syntax is ugly.

It's hard to understand from a classical background.

Inheritance with Class

- Class is F's inheritance model
  - Simplified syntax
  - Works the same way under the hood
  - Gives you a few things for free

```
1  // Class: Person
2  var Person = new Class({
3      a: function() { return 'a'; },
4      b: function() { return 'b'; }
5  });
6
7  // Class: Child
8  var Child = new Class({
9      extend: Person,
10     c: function() { return "c"; };
11 });
12
13 // Create an instance of a child
14 var kid = new Child();
15
16 // Calls to kid.a actually call Person.prototype.a
17 kid.a();
```

```
1  // Class: Person
2  var Person = new Class({
3      a: function() { return 'a'; }
4  });
5
6  // Class: Child
7  var Child = new Class({
8      extend: Person,
9      a: function() {
10         var parentReturnVal = this.inherited(arguments);
11         return parentReturnVal+"à";
12     };
13 });
14
15 // Create an instance of a child
16 var kid = new Child();
17
18 kid.a(); // "aà"
```

Inheritance with Class:
Superclass Methods

- Do not need to know parent class
- Simple syntax
- Works if immediate parent does not implement called method

```
1 var Person = new Class({
2     toString: 'Person'
3 });
4
5 var person = new Person();
6
7 person.toString(); // "Person"
8
9 alert(person+' says hi!'); // "Person says hi!"
```

Inheritance with Class:
The toString Method

- Used heavily by F, recommended for better stacktraces
- Can be defined as a method or a string
- Inherits from parent if not defined

```
var Person = new Class({
    construct: function() { console.log('Building a Person...'); },
    destruct: function() { console.log('Destroying a Person...'); }
});

var Child = new Class({
    extend: Person,
    construct: function() { console.log('Building a Child...'); },
    destruct: function() { console.log('Destroying a Child...'); }
});

// Create an instance of a child
var kid = new Child();
// "Building a Person..."
// "Building a Child..."

// Destroy the instance
kid.destruct();
// "Destroying a Child..."
// "Destroying a Person..."
```

Inheritance with Class:
Constructors & Destructors

- Defined with `construct` and `destruct`
- Chained automatically: parent constructors first, child destructors first
- `construct` is always passed an empty object if no arguments are provided