software studio

this & new : nasty effects

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2013.
confused javascript

wants to be a prototyping language
› no classes, prototype chain

wants to be a standard OO language
› instead of cloning operator, has pseudo constructor

consequence
› some strange rules
› easy to mess up
this is dynamically scoped
› in evaluating e.m(), this is bound to value of e inside m
› but reverts to global environment in calls that m makes

```javascript
var counter = {
  val: 0,
  inc: function() {
    this.val += 1; return this.val;
  }
}
counter.inc(); // 1
counter.inc(); // 2
```
where *this* fails

a point ADT:

```javascript
var Point = function (x, y) {
    this.x = function () {return x;}
    this.y = function () {return y;}
    mag = function () {return Math.sqrt(x*x+y*y);}
    this.unit = function () {return new Point (x/mag(), y/mag());}
}
```

playing with it:

```javascript
> p = new Point(3,4)
Point {x: function, y: function, unit: function}
> p.unit().y()
0.8
```
where this fails

with defaulting:

```javascript
var Point = function (x, y) {
  this.x = function () {return x ? x : 0;}
  this.y = function () {return y ? y : 0;}
  mag = function () {return Math.sqrt(this.x() * this.x() + this.y() * this.y());}
  this.unit = function () {return new Point (this.x()/mag(), this.y()/mag());}
}
```

playing with it:

```javascript
> p = new Point(2)
Point {x: function, y: function, unit: function}
> p.unit().y()
TypeError: Object [object global] has no method 'x'
```

fixed:

```javascript
var Point = function (x, y) {
  var that = this;
  this.x = function () {return x ? x : 0;}
  this.y = function () {return y ? y : 0;}
  mag = function () {return Math.sqrt(that.x() * that.x() + that.y() * that.y());}
  this.unit = function () {that = this;
                           return new Point (this.x()/mag(), this.y()/mag());}
}
```
another *this* failure

abstract out logging:

```javascript
> f = function (reporter) {
  for (var i = 0; i < 4; i++)
    reporter("Step " + i);
}
> f(console.log);
TypeError: Illegal invocation
```

a workaround:

```javascript
> r = function (s) {console.log(s);}  
> f(r)
Step 0
Step 1
Step 2
Step 3
undefined
```
forgetting to use “new”

```javascript
var Point = function (x, y) {
    this.x = function () {return x;}
    this.y = function () {return y;}
}

> x = 3
3
> var p = Point(1,2)
undefined
> p.x()
TypeError
> x
function () {return x;}
> p
undefined
```

a remedy:

```javascript
var Point = function (x, y) {
    if (!(this instanceof Point)) return new Point(x, y);
    this.x = function () {return x;}
    this.y = function () {return y;}
}
```
so this or that?

how to make an ADT
› with *this* & *new*
› with closures alone

<table>
<thead>
<tr>
<th></th>
<th>this &amp; new</th>
<th>closures</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>instanceof</code></td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><code>extend prototype</code></td>
<td>yes, but can’t see rep</td>
<td>not so easily</td>
</tr>
<tr>
<td><code>avoid nasties</code></td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>
directly selecting prototype

choose a prototype for each new object?

Object.create = function (o) {
    var F = function () {};
    F.prototype = o;
    return new F();
}

> color = {bits: 24}
Object
> red = Object.create(color)
Object.create.F
> red.r = 255; red.g = 0; red.b = 0;
0
> red.bits
24

Object.create: in ECMA 5, and implemented in most browsers