Classes Are Dead - Long Live Classes!

by Adrian Sieber

adriansieber.com

Short Recap:
From ES5 Functions to ES2015 Classes

Constructor function with prototypical OOP

```
function Color (options) {
  this.red = options.red
  this.green = options.green
  this.blue = options.blue
}

Color.prototype.toString () {
  return 'rgb(' +
    this.red + ', ' + this.green + ', ' + this.blue +
  ')'
}
```

```
var color = new Color({red: 250, green: 35, blue: 129})
console.log(color)
```

```
'rgb(250, 35, 129)'
```

+ ES2015 Syntax Sugar

```
function Color (options = {}) {
  Object.assign(this, options)
}

Color.prototype.toString () {
  return `rgb(${this.red}, ${this.green}, ${this.blue})`
}
```

```
const color = new Color({red: 250, green: 35, blue: 129})
console.log(color)
```

```
'rgb(250, 35, 129)'
```

With ES2015 Class Syntax

```
class Color {
  constructor (options) {
    Object.assign(this, options)
  }

  toString () {
    return `rgb(${this.red}, ${this.green}, ${this.blue})
  }
}
```

```
const color = new Color({red: 250, green: 35, blue: 129})
console.log(color)
```

```
'rgb(250, 35, 129)'
```

Just a new Syntax for old-school constructor functions

```
console.log(typeof Color)
```

```
'function'
```

Inheritance

```
class BitColor extends Color {
  constructor (options = {}) {
    super(options)
    this.depth = options.depth

    const maxValue = 2 ** this.depth
    const outOfRange = this.red > maxValue ||
      this.green > maxValue ||
      this.blue > maxValue ||
    if (outOfRange) throw new RangeError()
  }
}
```

```
const bitColor = new BitColor(
  {red: 250, green: 35, blue: 129, depth: 5}
)
```

```
RangeError
    at new BitColor (/Users/adrian/Talks/BitColor.js:19:2
```

Static Methods

```
class Color {
  ...
  static fromRgbArray (rgbArray) {
    if (!Array.isArray(rgbArray)) throw new TypeError()
    return new Color({
      red: rgbArray[0],
      green: rgbArray[1],
      blue: rgbArray[2],
    })
  }
  ...
}
```

```
const color = Color.fromRgbArray([250, 35, 129])
```

Getter

```
class Color {
  ...
  get luminosity () {
   return 0.21 * this.red +
      0.72 * this.green +
      0.07 * this.blue
  }
}
```

```
console.log(color.red)
console.log(color.luminosity)
```

DON'T USE `color.getLuminosity()` !!!

Crappy code is perfectly OK, but don't write crappy APIs!

Setter

```
class Color {
  ...
  set red (redValue) {
   this._red = redValue
  }
  setRed (redValue) {
   this._red = redValue
   return this
  }
  ...
}
```

```
color.red = 90
```

```
color
  .setRed(32)
  .setBlue(88)
  .setAlpha(0.2)
```

Setters and Getters allow Ducktyping

```
const logoColor = new Color({
  red: 250,
  green: 35,
  blue: 129,
})

const textColorObject = {
  hue: 250,
  saturation: 0.7,
  luminosity: 0.5,
}

function logHue (color) {
  console.log(`The hue is ${color.hue}`)
}

logHue(logoColor) // The hue is 334
logHue(textColorObject) // The hue is 250
```

Type check anti pattern:

```
if (color instanceof Color) {
  return color.red
}
```

Instead use:

```
if (color.hasOwnProperty('red')) {
  return color.red
}
```

Let me introduce you to datatypes.js

github.com/datatypesjs

> A collection of various datatypes for JavaScript with an uniform interface.

Already Available

Geometry / Graphics

- point - Class for points in 3D space.
- vector - Class for 3D vectors.
- face - Class for 2-faces to use as facets of polyhedrons and polygon meshes.
- matrix - Class for 4x4 row-major matrices for transformations in 3D.

Time

- duration - ISO 8601 compatible duration class.
- interval - ISO 8601 based interval class.
- moment - ISO 8601 based moment and instant class.

Structure

- One class per file
- Class name must be the same as file name

```
cat project/Color.js
```

```
export default class Color {
  ...
}
```

One object as constructor argument

```
const color = new Color({red: 250, green: 35, blue: 129})
```

~~`new Color(250, 35, 129)`~~

~~`new Color('#FA2381')`~~

```
const color = Color.fromHexString('#FA2381')
```

~~`new Color([250, 35, 129])`~~

```
const color = Color.fromRgbArray([250, 35, 129])
```

I don't care how, just try!

```
const color = Color.from(pixel)
```

- For inhomogeneous database entries
- Data from the internet
- User input

No private methods => Move to module scope

```
function getChannelMax (options = {}) {
  return Math.max(options.red, options.green, options.blu
}

function getChannelMin (options = {}) {
  return Math.min(options.red, options.green, options.blu
}

export default class Color {
  constructor (options = {}) {
    Object.assign(this, options)
  }

  get lightness () {
    return (
      getChannelMax(this) +
      getChannelMin(this)
    ) / 2
  }
}
```

Special Properties

`toString()`

```
class Color {
  ...

  function toString () {
    return `rgb(${this.toArray().join(', ')})`
  }
}
```

```
const logoColor = new Color({
  red: 250,
  green: 35,
  blue: 129,
})

console.log(`The color ${logoColor} is beautiful!`)
// The color rgb(250,35,129) is beautiful!
```

`toJSON()`

```
class Color {
  ...

  function toJSON () {
    return {
      red: this.red,
      green: this.green,
      blue: this.blue,
    }
  }
}
```

```
const color = new Color({red: 250, green: 35, blue: 129})
console.log(JSON.stringify(color))
```

```
{"red": 250, "green": 35, "blue": 129}
```

Performance

> Premature optimization is the root of all evil

- Best algorithms?
- Caching?
- Streaming?
- Async?
- Multi threaded (web worker)?
- Multi process?

Developers are the largest performance bottle neck

- Search for best dependency
- Waste time reading docs / dependency code
- Use modules incorrectly
- Don't understand legacy code

Drawbacks

Typos don't trigger an error

```
person.setFulName('John Doe')
```

vs.

```
person.fulName = 'John Doe'
```

But:

- IDE autocompletion can help
- Can be prevented with `Object.seal(person)`

Not limited to JavaScript

Datatypes.php

```
class Color {
  public function __construct (hashmap) {
    $this->red = $hashmap['red'];
    $this->green = $hashmap['green'];
    $this->blue = $hashmap['blue'];
  }
  ...
  public function __toString () {
    return "rgb($this->red, $this->green, $this->blue)";
  }
}
```

```
$color = new Color(
  ['red' => 250, 'green' => 35, 'blue' => 129]
)
```

My Dream:

All Objected Oriented Languages

- Datatypes.**rb**
- Datatypes.**py**
- Datatypes.**java**
- Datatypes.**swift**
- Datatypes.**go**
- ...

My Dream:

All the things class-ified

```
const cart = new Cart()
cart.addItem(tshirt)
```

```
const email = new Email()
email.sendTo(user)
```

```
const document = new Document()
document.render()
```

```
const image = new Image()
image.detectFaces()
```

```
const rocket = new Rocket()
rocket.launch()
```

Project: github.com/datatypesjs

My homepage: adriansieber.com

My startup's homepage: feram.co