JS

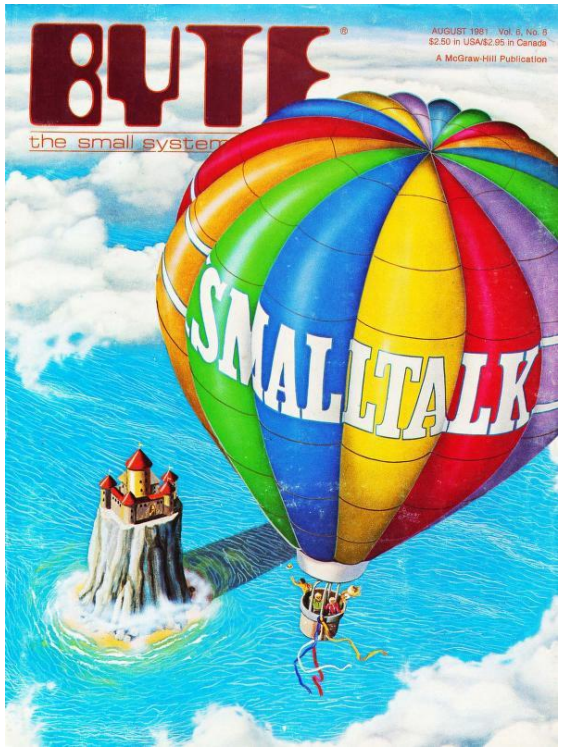
JavaScript

JavaScript (JS) is a lightweight interpreted or JIT-compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a prototype-based, multi-paradigm, dynamic language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles. Read more about JavaScript.

ES

ECMAScript

The standard for JavaScript is ECMAScript. As of 2012, all modern browsers fully support ECMAScript 5.1. Older browsers support at least ECMAScript 3. On June 17, 2015, ECMA International published the sixth major version of ECMAScript, which is officially called ECMAScript 2015, and was initially referred to as ECMAScript 6 or ES6. Since then, ECMAScript standards are on yearly release cycles. This documentation refers to the latest draft version, which is currently ECMAScript 2018.



Smalltalk y Javascript

Dos lenguajes OO bien particulares

Herramientas diferentes para pensar diferente

- Presentar los aspectos más particulares de dos lenguajes que se diferencian del resto
- Smalltalk
  - OO de pies a cabeza (escrito en Smalltalk)
  - Su ambiente que invita a un enfoque exploratorio de desarrollo
  - Fuente de muchas de las ideas que hoy vemos en otros lenguajes y ambientes
- JavaScript (ECMAScript)
  - Su naturaleza basada en prototipos

JavaScript (ECMAScript)

- Lenguaje de propósito general
- Dinámico
- Basado en objetos (con base en prototipos en lugar de clases)
- Multiparadigma
- Se adapta a una amplia variedad de estilos de programación
- Pensado originalmente para scripting de páginas web
- Con una fuerte adopción en el lado del servidor (NodeJS)

Un mínimo de sintaxis

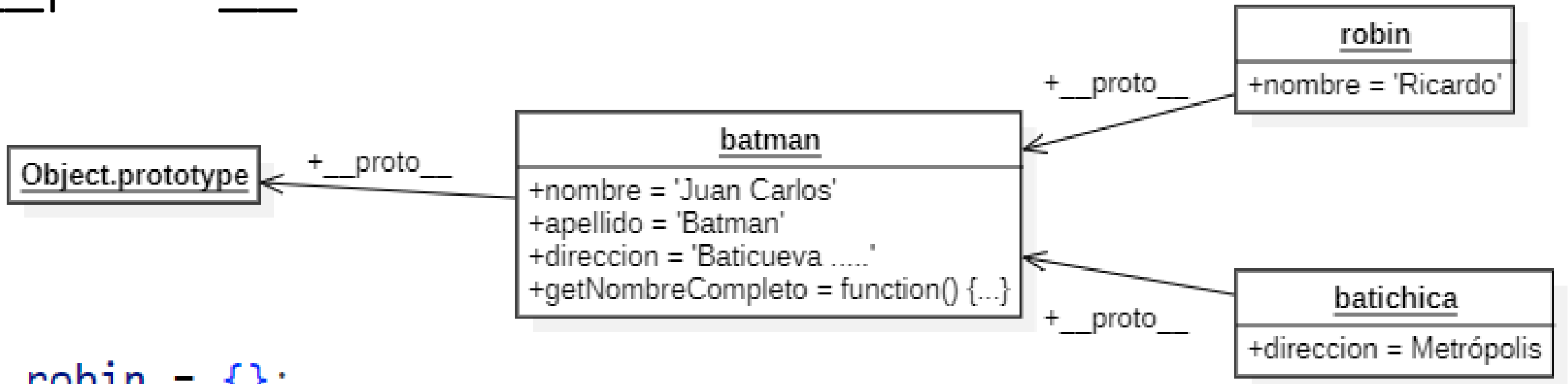
```
sumar = function(a,b) {return a + b}

sumar(1,2)

function restar(a, b) {return a-b}

batman = { nombre: 'Juan Carlos', apellido: 'Batman', edad: 53 };

batman.edad = 44

batman.direccion = "Baticueva 14, entre Gallos y Medianoche"

batman.getNombreCompleto = function()
            {return this.nombre + " " + this.apellido}

batman.getNombreCompleto()
```

```
batman = {
    nombre : "Juan Carlos",
    apellido : "Batman",
    direccion : "Baticueva - C. Gótica",

    getNombreCompleto() {
        return this.nombre + ' ' + this.apellido;
    }
};

batman.nombre; // 'Juan Carlos'

batman.getNombreCompleto(); // 'Juan Carlos Batman'
```

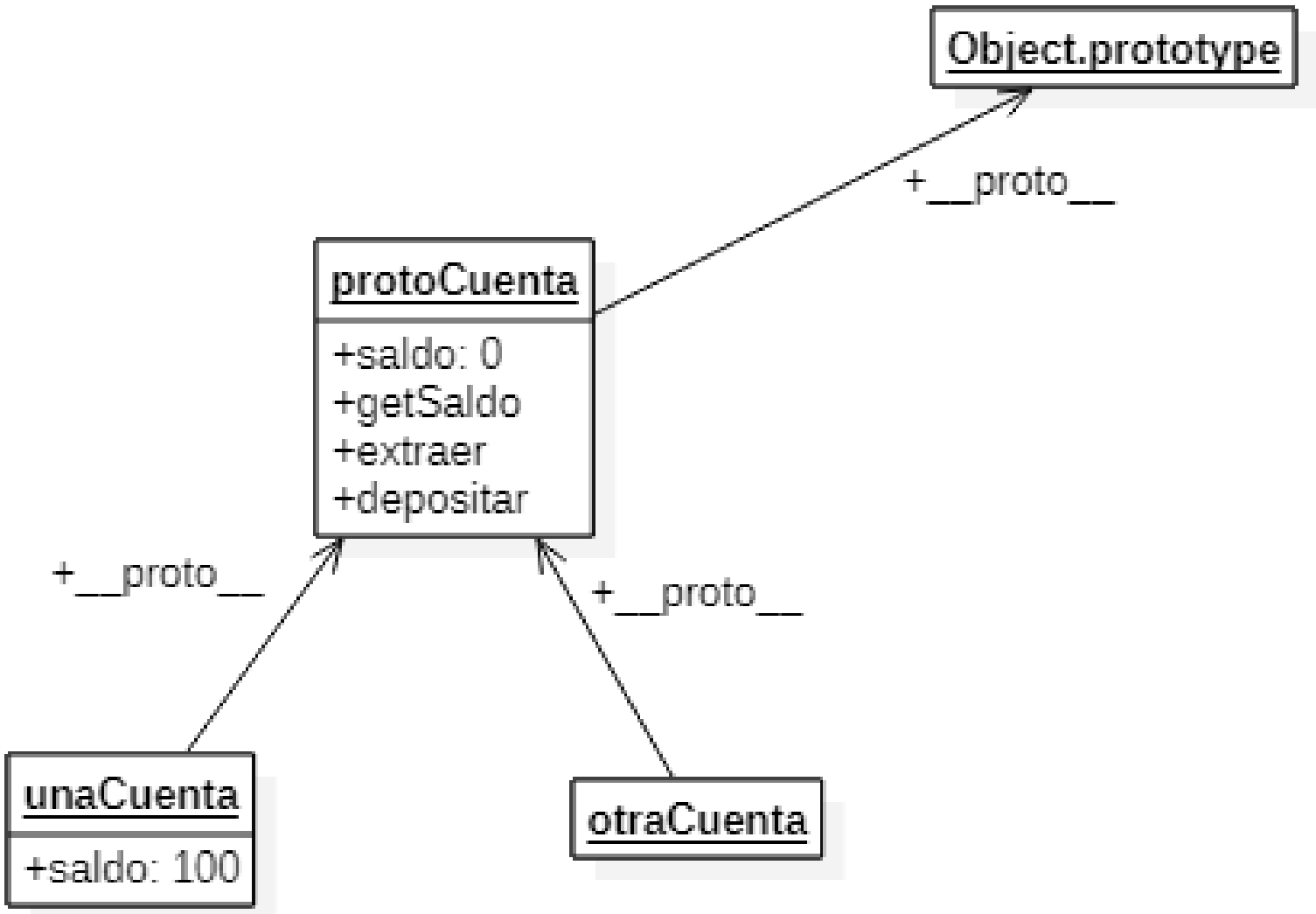
Prototipos

- En Javascript no tengo clases
- La forma más simple de crear un objeto es mediante la notación literal (estilo JSON)
- Cada objeto puede tener su propio comportamiento (métodos)
- Los objetos heredan comportamiento y estado de otros (sus prototipos)
- Cualquier objeto puede servir como prototipo de otro
- Puedo cambiar el prototipo de un objeto (y así su comportamiento y estado)
- Termino armando cadenas de delegación

__proto__

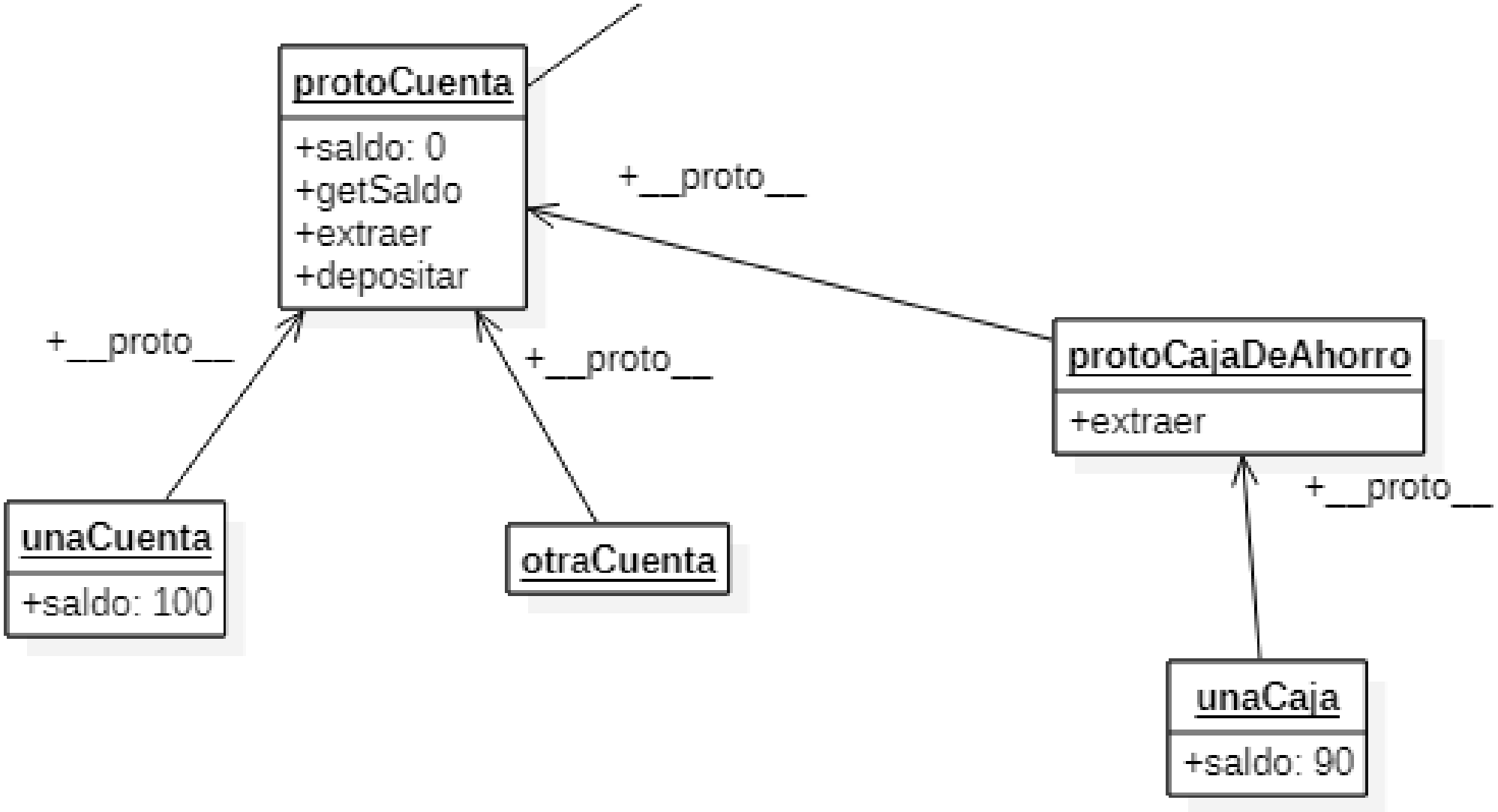



```
robin = {};
robin.__proto__ = batman;
robin.nombre = 'Ricardo';
robin.getNombreCompleto(); // 'Ricardo Batman';

batichica = Object.create(batman);
batichica.direccion = 'Metrópolis';
```


Object.prototype
+__proto__
protoCuenta
+saldo: 0
+getSaldo
+extraer
+depositar
+__proto__
+__proto__
unaCuenta
+saldo: 100
otraCuenta

```
> protoCuenta = { saldo: 0}
◂ ▶ {saldo: 0}
> protoCuenta.getSaldo = function() { return this.saldo }
◂ ƒ () { return this.saldo }
> protoCuenta.depositar = function(monto) { this.saldo = this.saldo + monto }
◂ ƒ (monto) { this.saldo = this.saldo + monto }
> protoCuenta.extraer = function(monto) { this.saldo = this.saldo - monto }
◂ ƒ (monto) { this.saldo = this.saldo - monto }
> unaCuenta = Object.create(protoCuenta)
◂ ▶ {}
> unaCuenta.getSaldo()
◂ 0
> unaCuenta.depositar(100)
◂ undefined
> unaCuenta.getSaldo()
◂ 100
```

Prototipos


protoCuenta
+saldo: 0
+getSaldo
+extraer
+depositar
+__proto__
+__proto__
+__proto__
protoCajaDeAhorro
+extraer
+__proto__
unaCuenta
+saldo: 100
otraCuenta
unaCaja
+saldo: 90

```
> protoCajaDeAhorro = Object.create(protoCuenta)
< ▶ {}

> protoCajaDeAhorro.extraer = function(monto) {
      if (monto < this.saldo) {
          this.saldo = this.saldo - monto;
      }
  }
< ƒ (monto) {
      if (monto < this.saldo) {
          this.saldo = this.saldo - monto;
      }
  }
```

Prototipos y “herencia”

```
> unaCajaDeAhorro = Object.create(protoCajaDeAhorro)
< ▶ {}

> unaCajaDeAhorro.getSaldo()
< 0

> unaCajaDeAhorro.extraer(10)
< undefined

> unaCajaDeAhorro.getSaldo()
< 0

> unaCajaDeAhorro.depositar(100)
< undefined

> unaCajaDeAhorro.extraer(10)
< undefined

> unaCajaDeAhorro.getSaldo()
< 90
```

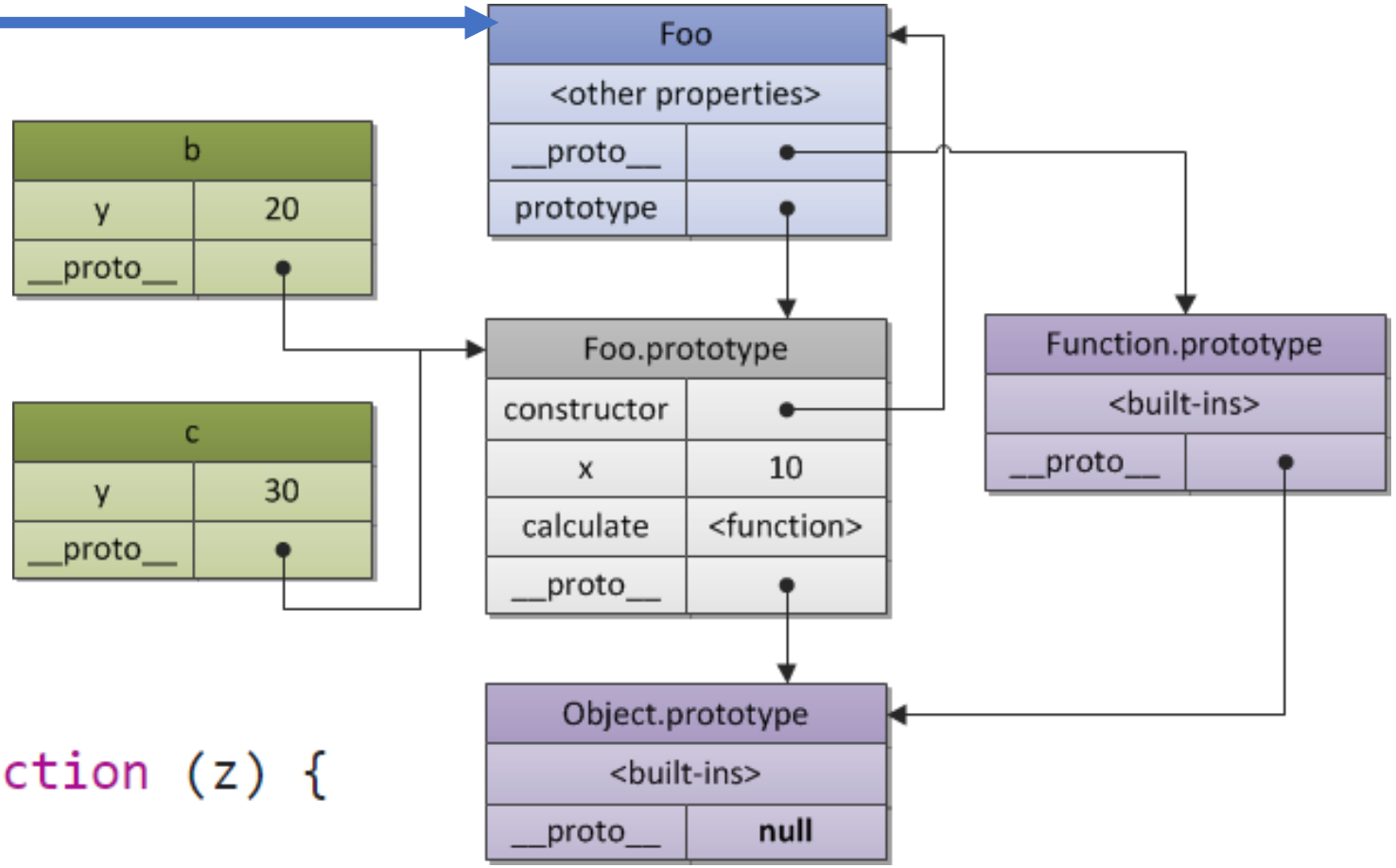
Funciones que son objetos y constructores

```
function Persona(nombre, apellido) {
    this.nombre = nombre,
    this.apellido = apellido
}

juan = new Persona("juan","gomez")
```

¿Qué prototipo tienen esas personas?

¿Cómo lo uso para agregar datos comportamiento que hereden todas?

```
function Foo(y) {
  this.y = y;
}

Foo.prototype.x = 10;

Foo.prototype.calculate = function (z) {
  return this.x + this.y + z;
};

var b = new Foo(20);
var c = new Foo(30);
```



ES6 – Clases como azúcar sintáctico

```
class Cuenta {

    constructor() {
        this.saldo = 0;
    }

    getSaldo() {
        return this.saldo;
    };

    depositar(monto) {
        this.saldo = this.saldo + monto;
    };

    extraer(monto) {
        this.saldo = this.saldo - monto;
    };

};
```

```
class CajaDeAhorro extends Cuenta {

    extraer(monto) {
        if (monto < this.saldo) {
            super.extraer(monto);
        }
    }
};
```