JS

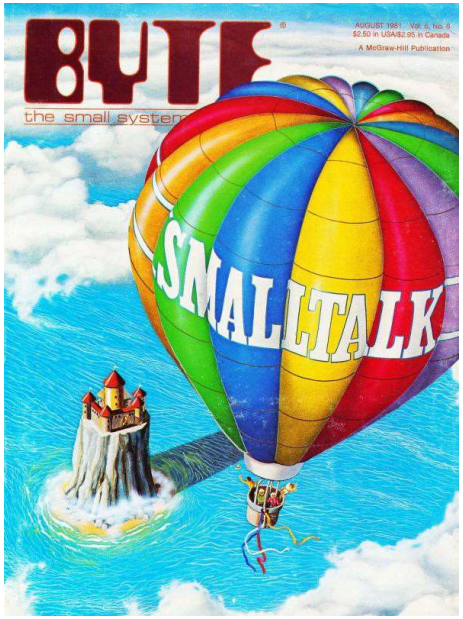
JavaScript

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.



JavaScript y Smalltalk

Dos lenguajes OO bien particulares

Objetivos

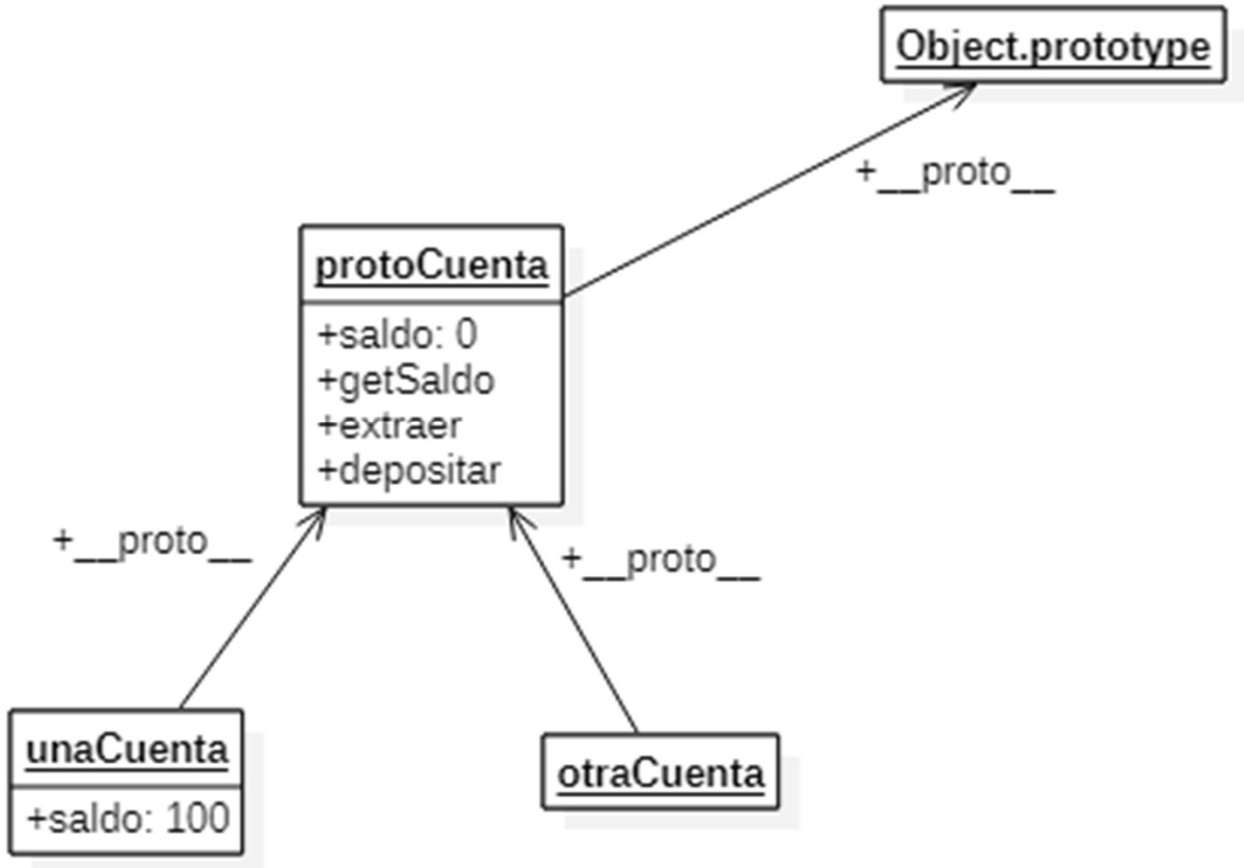
- Presentar los aspectos mas particulares de dos lenguajes que se diferencian del resto
- JavaScript (ECMAScript)
  - Su naturaleza basada en prototipos
- Smalltalk
  - OO de pies a cabeza (escrito en Smalltalk)
  - Su ambiente que invita a un enfoque exploratorio de desarrollo
  - Fuente de muchas de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

JavaScript (ECMAScript)

- 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 servidos (NodeJS)

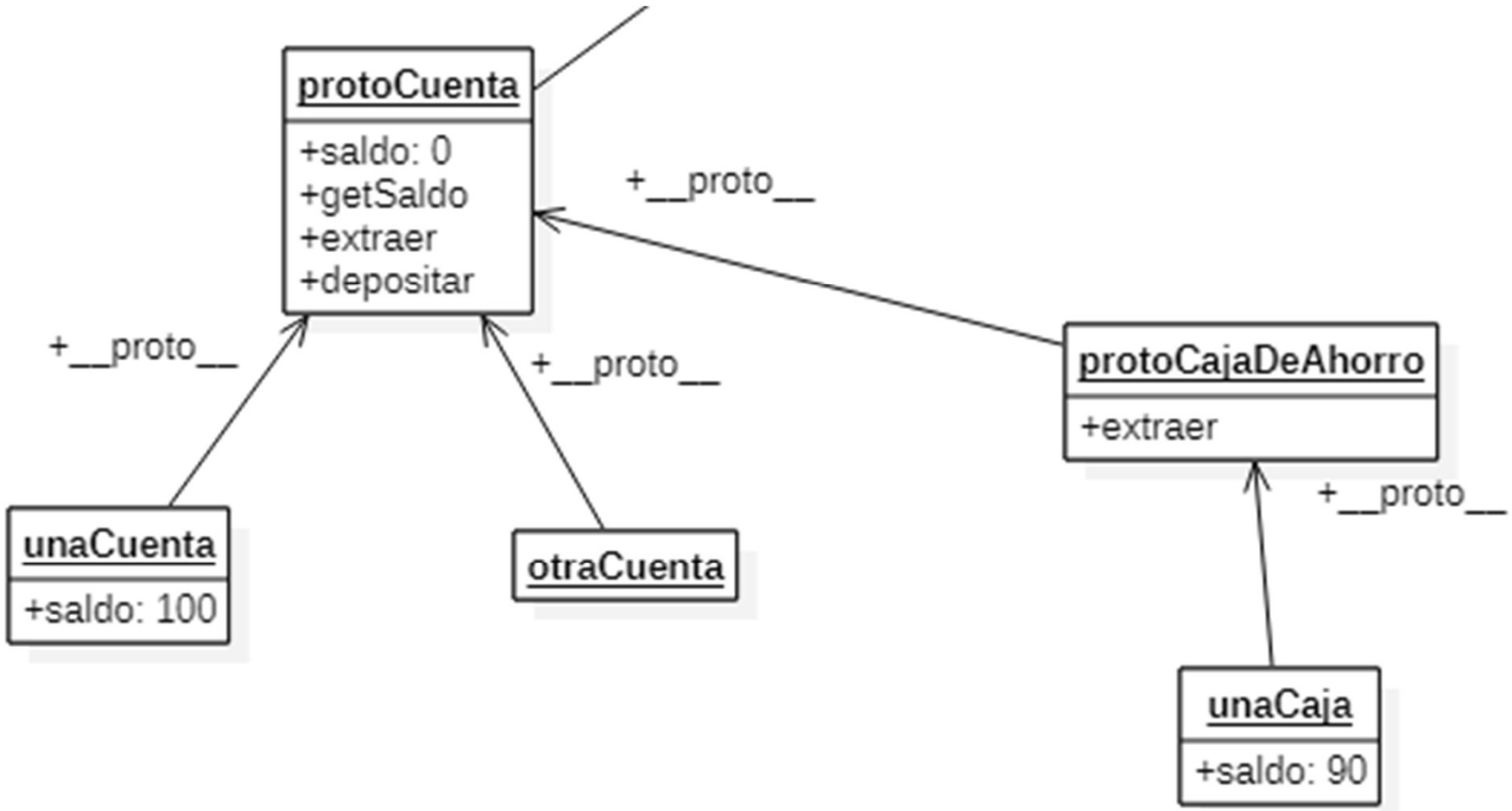
Prototipos

- En Javascript no tengo clases
- La forma mas simple de crear un objeto es mediante la notación literal (estilo JSON)
- Cada objeto puede tener su propio comportamiento (métodos)
- Los objetos heredan comportamiento, estructura y estado de otros (sus prototipos)
- Cualquier objeto puede servir como prototipo (del cual crear otros)
- Una vez que tengo un objeto (prototipo) puedo crear otros a partir de él con Object.create(…)
- Termino armando cadenas de delegación


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
```

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);
        }
    }
};
```

Smalltalk

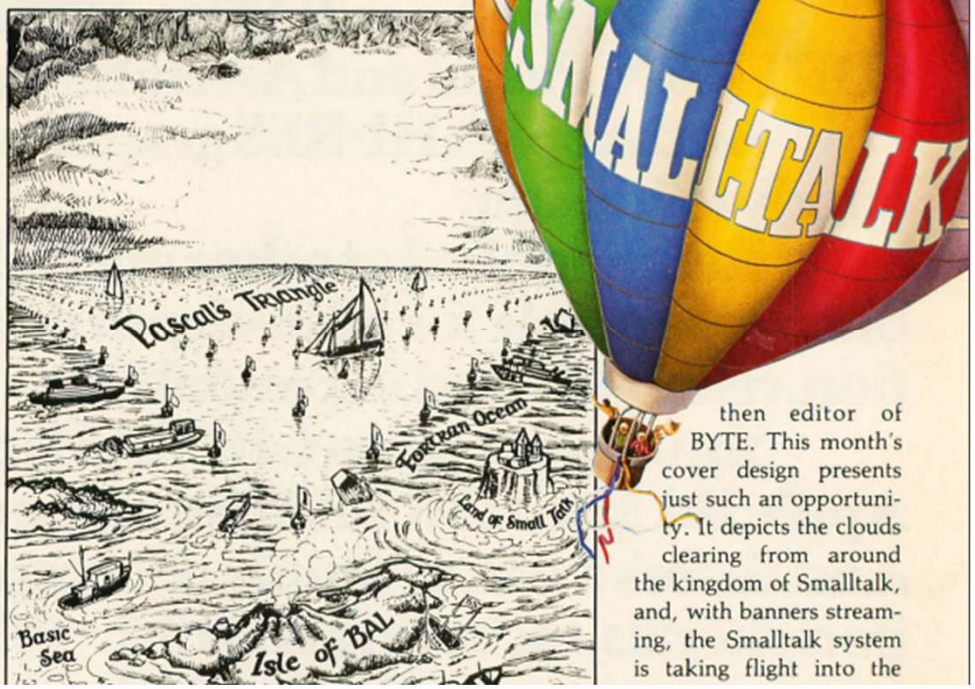
https://www.youtube.com/watch?v=2u70CgBr-OI

Introducing the Smalltalk-80 System

Adele Goldberg
Manager, Learning Research Group
Xerox Palo Alto Research Center
3333 Coyote Hill Rd
Palo Alto CA 94304

It is rare when one can indulge in one's prejudices with relative impunity, poking a bit of good humored fun to make a point.

With this statement, Carl Helmers opened his remarks in the "About the Cover" section of the August 1978 issue of BYTE. The issue was a special on the language Pascal, so Helmers took the opportunity to present Pascal's triangle as drawn by artist Robert Tinney. The primary allegory of the cover was the inversion of the Bermuda Triangle myth to show smooth waters within the area labeled "Pascal's



then editor of BYTE. This month's cover design presents just such an opportunity. It depicts the clouds clearing from around the kingdom of Smalltalk, and, with banners streaming, the Smalltalk system is taking flight into the

https://www.tech-insider.org/star/research/acrobat/8108.pdf

Smalltalk

- Lenguaje OO puro – todo es un objeto (¡ incluso las clases !)
- Tipado dinámicamente
- Propone una estrategia exploratoria (construccionista) al desarrollo de software
- El ambiente es tan importante como el lenguaje
  - Está implementado en Smalltalk
  - Ricas librerías de clases (fuentes de inspiración y ejemplos)
  - Todo su código fuente disponible y modificable
  - Tiene su propio compilador, debugger, editor, inspector, perfilador, etc.
  - Es extensible
- Sintaxis minimalista (con sustento en su foco educativo)
- Fuente de inspiración de casi todo lo que vino después (en OO)

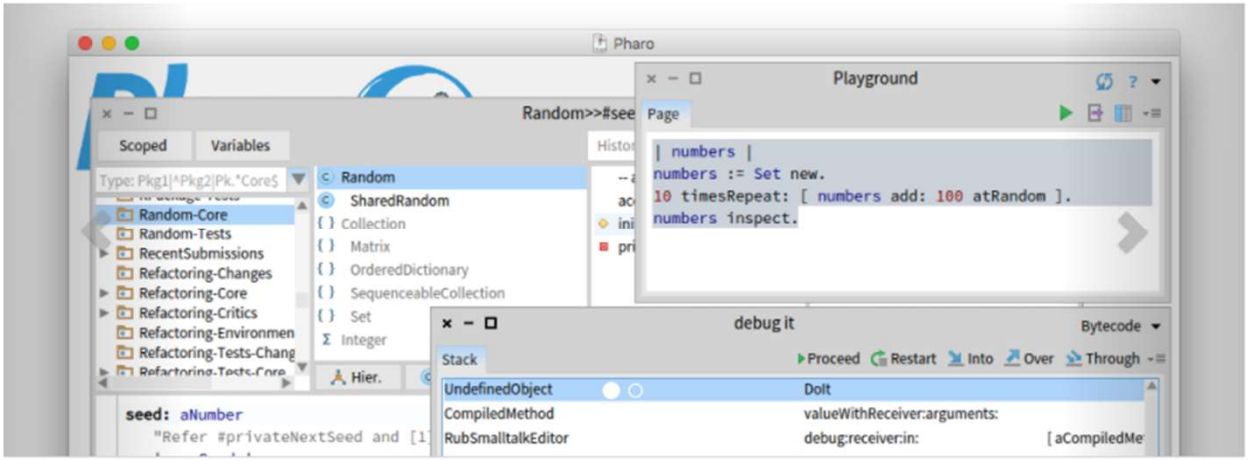
Veamos (¡ en vivo !)

- La sintaxis
- La imagen de objetos y la maquina virtual
- Las clases también son objetos
- Las estructuras de control como objetos
- Los iteradores de colecciones
- Programar, testear, depurar, ejecutar … todo se mezcla

Pharo

The immersive programming experience

Pharo is a pure object-oriented programming language *and* a powerful environment, focused on simplicity and immediate feedback (think IDE and OS rolled into one).



Discover | Download | Learn

Cinefiloos

- Para explorar la implementación de referencia de Cinefiloos, evalúen la siguiente expresión en un playground de Pharo

```
Metacello new
    baseline: 'Cinefiloos';
    repository: 'bitbucket://lifia-oop/practicas-objetos-1’;
    onConflictUseLoaded;
    load.
```