

Lenguajes y Paradigmas de Programación

Programación Orientada a Objetos y Scheme

Índice

- Ideas fundamentales de la POO
- POO en MzScheme
- Objetos y clases
- Herencia
- Interfaces
- Funciones de bajo nivel en MzScheme
- Implementación de POO en Scheme

POO

- La Programación Orientada a Objetos es un paradigma de programación que explota en los 80 pero nace a partir de ideas a finales de los 60 y 70
- Smalltalk como lenguaje paradigmática de POO
- Alan Kay es el creador del término “Object-Oriented”
- Artículo de Alan Kay: “The Early History of Smalltalk”, ACM SIGPLAN, March 1993

Alan Kay

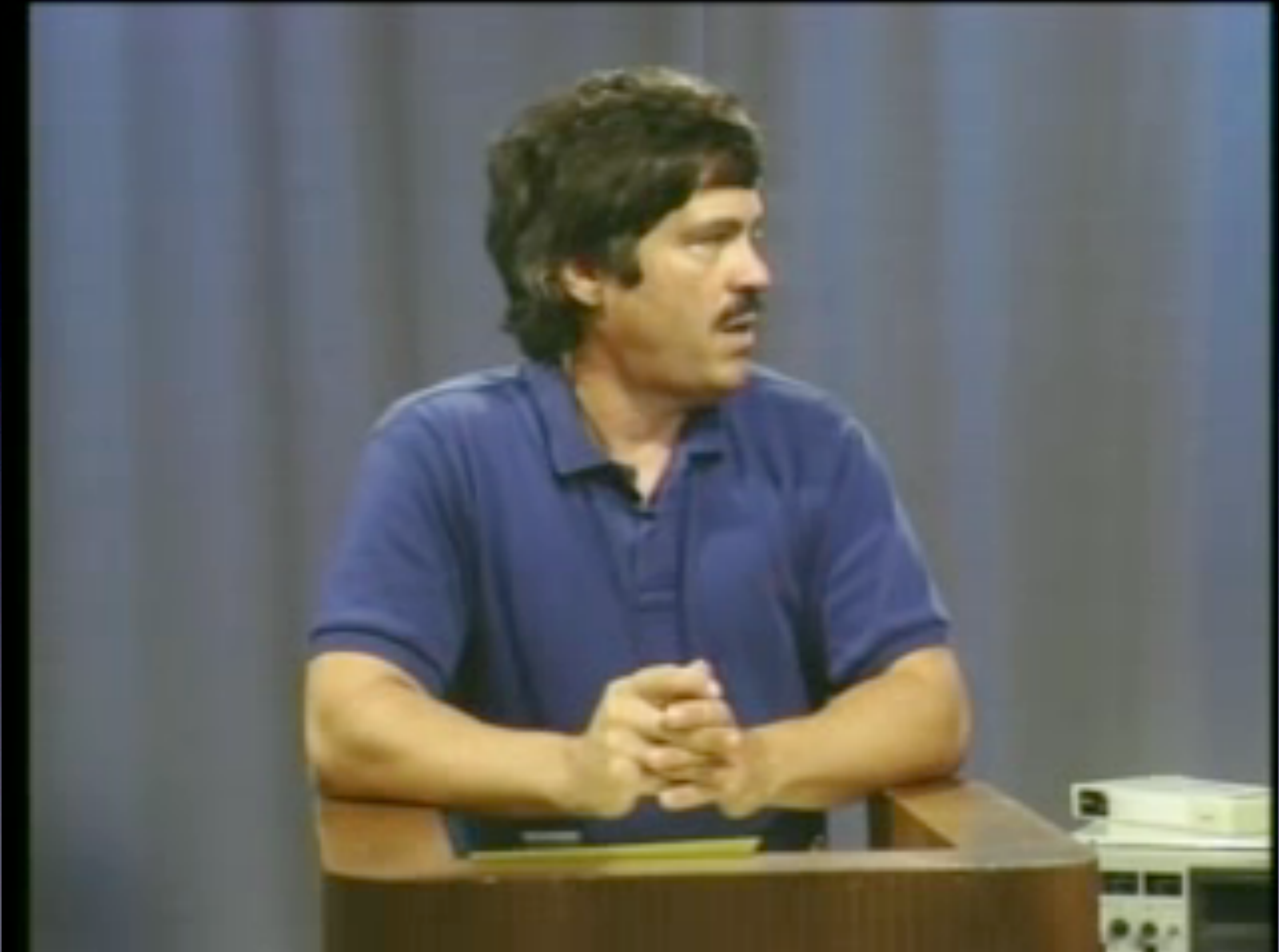
“I invented the term Object-Oriented and I can tell you I did not have C++ in mind.”

“Smalltalk is not only NOT its syntax or the class library, it is not even about classes. I'm sorry that I long ago coined the term *objects* for this topic because it gets many people to focus on the lesser idea. The big idea is *messaging*.”

Alan Kay

“Smalltalk's design--and existence--is due to the insight that everything we can describe can be represented by the recursive composition of a **single kind of behavioral building block** that **hides its combination of state and process inside itself** and can be dealt with only through the **exchange of messages.**”

El comienzo: Ivan Sutherland y SketchPad



¿Interesados en Smalltalk?

Visitar

<http://www.squeak.org/>

http://swiki.agro.uba.ar/small_land

<http://www.squeakland.org>

Lenguajes OO

- Smalltalk, Java, Ruby, Python, C#, C++ (si lo usamos con prudencia), ...

Del paradigma procedural al OO

- Programación procedural: estado abstracto (tipos de datos y barrera de abstracción) + funciones
- Siguiendo paso: agrupar estado y funciones en una única entidad
- El “objeto” es esta entidad

Objeto

- Un objeto contiene un estado (atributos o variables de instancia) y un conjunto de funciones (métodos) que implementan las funcionalidades soportadas
- Al ejecutar un método, el objeto modifica su estado
- Pedimos a un objeto que ejecute un método lanzándole un mensaje

Características de la POO

- Objetos (creados/instanciados en tiempo de ejecución) y clases (plantillas estáticas/tiempo de compilación)
- Los objetos agrupan estado y conducta (métodos)
- Los métodos se invocan mediante mensajes
- Dispatch dinámico: cuando una operación es invocada sobre un objeto, el propio objeto determina qué código se ejecuta. Dos objetos con la misma interfaz pueden tener implementaciones distintas.
- Herencia: las clases se pueden definir utilizando otras clases como plantillas y modificando sus métodos y/o variables de instancia.

Clases

- Una clase es la plantilla que sirve para definir los objetos
- En una clase se define los elementos que componen el objeto (sus atributos o campos) y sus métodos
- En una clase también se pueden definir variables (variables de clase) compartidas por todos los objetos de esa clase

Estructuras en Scheme

```
(define-struct persona
  (nombre apellidos fecha-nacimiento nif))

(define (edad persona fecha-actual)
  (let ((año-nacimiento
        (string->number
         (substring
          (persona-fecha-nacimiento
           persona) 0 4))))
    (año-actual (string->number
                 (substring fecha-actual 0 4))))
    (- año-actual año-nacimiento)))
```

POO en Scheme

```
(require (lib "class.ss"))

(define persona%
  (class object%
    (init-field nombre nif)
    (field (apellidos null)
           (fecha-nacimiento null))

    (define/public (di-hola)
      (printf "Hola, soy ~a~%" nombre))

    (super-new)))

(define p1
  (new persona% (nif '212121232)
               (nombre "Pepito")))

(send p1 di-hola)
```

Sintaxis class

```
(class <super-clase>
  (init-field <var1> ... <varn>)
  (field (<var1> <valor1>) ...
        (<varn> <valorn>))

  (define/public (<metodo> <args>)
    <cuerpo>)

  (super-new <args>)
  <sentencias-inicializacion>)
```

Sintaxis new y paso de mensajes

```
(new <clase> (<arg1> <val1>) ...  
            (<argn> <valn>))
```

```
(send <objeto> <metodo> <arg1> ... <argn>)
```

Herencia

- Al definir una clase hay que indicar la superclase que extiende
- La superclase por defecto es `Object`
- La clase hereda los métodos y atributos de la superclase: los objetos de la clase pueden ejecutar los métodos de la clase y los de la superclase

Invocación

- Todos los campos y métodos definidos en una clase pueden usarse directamente en los métodos de esa clase
- Los métodos de la superclase pueden invocarse desde el propio objeto utilizando (send **this** <metodo>)
- Los métodos y campos de la superclase pueden incluirse en el ámbito de la clase usando las palabras claves `inherit` y `inherit-fields`