# Paradigmas de Objetos

La principal abstracción de nuestro nuevo paradigma son los objetos, que representan distintos tipos de entidades.

Un objeto es algo que puedo representar a través de una idea, un concepto. Tiene entidad.

Un sistema en el paradigma orientado a objetos es: “Un sistema es un conjunto de objetos que se envían mensajes para alcanzar un determinado objetivo”

¿Cuál era la definición de sistema según la Teoría General de Sistemas? “Conjunto de partes que se relacionan para un objetivo común” Bueno, al menos sabemos que vamos con una definición congruente...

## Mensaje

¿Cómo interactúan los objetos entre sí? A través de los mensajes. ¿Cómo envío un mensaje a otro objeto?

* primero lo tengo que conocer (más adelante veremos cómo)
* cuando lo conozco, sé qué mensajes le puedo mandar
* ¿se cómo lo hace? no, no quiero saberlo, porque saberlo me ata a él (en términos de diseño decimos “me acopla”) y si yo asumo ciertas cosas en base a cómo lo hace, eso aumenta la posibilidad de tener que modificar cosas si el objeto al que le mando el mensaje cambia.

## Ambiente

La palabra ambiente es conocida en otras tecnologías como

* imagen, en Smalltalk
* virtual machine, para Java y otras tecnologías similares

## Mensaje y método

Mensaje es lo que el objeto emisor le envía como orden al receptor. El emisor no se entera de cómo se resuelve el mensaje, sólo lo pide (el qué y no el cómo, ¿a qué suena eso?). El receptor recibe el mensaje y se ejecuta un método (porción de código).

### Method lookup

La estrategia que utilizan los lenguajes para resolver dónde está el código de un método al enviar un mensaje recibe el nombre de method lookup. En nuestro primer caso (el más simple) el código a ejecutar de un mensaje se busca en un método del objeto receptor de dicho mensaje.

# Breve introducción a Wollok

Wollok es un lenguaje de programación con fines didácticos

* no obliga al desarrollador a definir tipos para los objetos
* es interpretado
* no necesitas tener un programa editor de texto y una línea de comando para correr los programas/tests sino que provee un entorno integrado de desarrollo o IDE donde trabajar.

## Accessors

Como vimos en el párrafo Estado de un objeto, no es aconsejable que quien use a pepita manipule directamente sus variables, sino que debe enviarle mensajes que accedan o modifiquen las referencias a los objetos que conoce. Para ello existe cierto tipo de mensajes llamados accessors, cuyo fin es publicar la referencia (getter) o modificar dicha referencia (setter).

El getter se define como un método de una sola línea, que devuelve algo.

El setter no, es un método que modifica el estado interno del objeto, no tiene necesidad de devolver nada.

## Referencias en Wollok

Wollok tiene dos maneras de definir referencias

* variables (var): es una referencia que puede cambiar el objeto al que apunta
* valores (const): es una referencia fija a un objeto, que no puede cambiar (una vez inicializada no es válida la operación de asignación)

## Relaciones bidireccionales

Mantener una relación bidireccional

* es cómodo, porque cualquiera de los dos objetos puede enviar un mensaje a otro
* pero requiere mantener ambas referencias sincronizadas, para evitar inconsistencias

Una alternativa puede ser que uno de los objetos sea el responsable de actualizar la otra referencia:

## Identidad

En un sistema con objetos, cada objeto sabe que es él y ningún otro objeto más, por eso es el responsable de responder ante un mensaje. En el ambiente cada objeto tiene su propia identidad. En un sistema orientado a objetos es frecuente tener diferentes referencias y querer determinar si estamos hablando de un objeto u otro, entonces la definición de identidad nos dice que: *Dos referencias son idénticas si apuntan al mismo objeto.*

# Introducción al Polimorfismo

El polimorfismo es una herramienta para agrupar objetos que tienen diferente comportamiento (distinto código) pero la misma interfaz. Entonces puedo enviarles mensajes sin saber cómo lo implementan, e incluso sin saber a qué objeto particular le estoy hablando: puedo cambiar la referencia sin tener que hacer cambios.

# Objetos Básicos

## Números

Los números en Wollok se representan como objetos inmutables, esto quiere decir que

* una vez inicializado un número, su estado interno no cambia
* la suma de 1 + 2 resulta en un nuevo número que representa al 3.

## Números con decimales

Es posible configurar la forma de trabajar con números que tengan decimales. En particular, se puede cambiar

* la cantidad de decimales máxima que puede admitir, por defecto son cinco
* y qué hacer en caso de recibir un número que excede la máxima cantidad de decimales permitido, por defecto lo redondea hacia arriba
* la forma de imprimir un número, por defecto eliminando los ceros no representativos

## Booleanos

Hay dos objetos booleanos representados con los literales “true” y “false”. Al igual que los números también son objetos inmutables, la expresión (true || false) devuelve un nuevo objeto true.

## Strings

Las cadenas de caracteres se delimitan con una o dos comillas.

const unString = "hola"

const otroString = 'mundo'

También son objetos inmutables (al concatenar “hola” y “mundo” tenemos un nuevo String “holamundo”).

## Fechas

Una fecha es un objeto inmutable que representa un día, mes y año (sin horas ni minutos). Se crean de dos maneras posibles:

const hoy = new Date()

// toma la fecha del día

const unDiaCualquiera = new Date(30, 6, 1973)

// se ingresa en formato día, mes y año

## Lambdas: objetos bloque

Al igual que en otras tecnologías, Wollok nos permite instanciar un objeto que representa un bloque de código, de manera de:

* poder generar referencias a dichos bloques
* pasarlos como parámetro
* elegir en qué momento ejecutar una porción de código: en un momento instanciamos un bloque de código y tiempo después
  + lo ejecutamos si se cumple una condición x,
  + ante un evento e
  + bien lo ejecutamos n veces hasta que se cumpla una condición: por ejemplo, para reprocesar una instrucción hasta que no haya errores.

### Comparación con el paradigma funcional

Más allá de la diferencia en la sintaxis, podemos hacer una comparación entre los bloques de código en Objetos y las expresiones lambdas de Haskell:

* ambas sirven para abstraer comportamiento que no interesa reutilizar en otro contexto, por eso no tienen nombre
* las expresiones lambda de Haskell solo devuelven un valor, mientras que en objetos podemos retornar un valor, o tener efecto colateral (en ese caso puede no importar el valor resultante)
* ambas son particularmente útiles para evitar hacer tareas repetitivas y subir el grado de declaratividad. Tomando los ejemplos de soluciones con map, filter y fold, tienen un menor grado de conocimiento del algoritmo y del orden que la misma solución con recursividad (en el caso de funcional) o un forEach (en el caso de objetos).

# Colecciones

La colección nos permite representar un conjunto de objetos relacionados. A primera vista una colección es un conjunto de objetos. Si la vemos con más precisión nos damos cuenta que es más preciso pensarla como un conjunto de referencias: los elementos no están dentro de la colección, sino que la colección los conoce.

## Tipos de colecciones en Wollok

A primera vista, podemos diferenciar dos diferentes tipos de colecciones en Wollok:

* los conjuntos, que modelan al conjunto matemático: no hay orden en los elementos y no puede haber elementos repetidos. Se definen mediante el literal #{ }
* las listas, en donde los elementos tienen un orden y puede haber elementos repetidos. Se definen mediante el literal [ ].