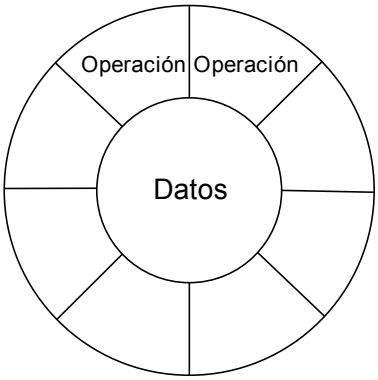
Clase práctica Objetos I

Paradigma de Objetos:
Objetos que colaboran entre sí enviándose mensajes

Objetos en Smalltalk

- Un objeto **expone** un **protocolo de mensajes**, que conforma el conjunto de operaciones que realiza el objeto. La forma de interactuar con un objeto es enviándole mensajes.
- Un objeto **oculta sus datos** propios, que sólo son accesibles desde las operaciones del objeto.



Objetos elementales

En Smalltalk todo es un objeto.
Muchos elementos que en otros lenguajes son valores de tipos de datos primitivos, están implementados en Smalltalk como objetos. Pertenecen a esta categoría, entre otros, los números, los strings, los caracteres, los booleanos y los arrays.
No obstante, hay ciertas construcciones sintácticas (del lenguaje) que denotan objetos. Estas construcciones, denominadas **literales**, permiten crear los **objetos elementales** de manera sencilla.
Éstos son algunos objetos elementales:

Objeto	**Descripción**
37	El entero 37
'Ser o no ser'	Un string con el texto "Ser o no ser"
2.71828	El punto flotante 2.71828
$p	El carácter p
true	El valor booleano *trae*
#(1 2 3)	Un array con los números 1, 2 y 3
#new	El símbolo *new*

- Un **símbolo** en Smalltalk es una secuencia de 1 o más caracteres que comienza con un #. A diferencia de los strings, no existen dos símbolos con la misma secuencia de caracteres.

Variables y asignación

- Una variable es un *puntero* o *referencia* a un objeto o instancia.

- Las variables son un nombre que empieza con minúscula (los identificadores que empiezan con mayúscula son variables globales, reservadas para el ambiente).
  Las variables en Smalltalk no tienen tipo. Ejemplos:

  anObject
  a
  x
  miConjunto
  trulala112
- La asignación de la referencia a un objeto a una variable se hace con el símbolo :=
  Sintaxis:

  variable := objeto

  Ejemplos:

  a := #(1 2 3)
  unNúmero := 23.3

- Las variables sin asignar apuntan al objeto elemental *nil.*

- **Variables globales**. Las variables globales son accesibles desde cualquier scope. Se identifican porque empiezan con mayúscula.

  A := 1

  ¡Atención!: no es una buena práctica usar variables globales para almacenar datos de programas. Tienen ciertos usos específicos relacionados con el ambiente (como mantener las referencias a las clases, que veremos luego).

Mensajes

- Cuando se le pide a un objeto que haga algo, se le está enviando un **mensaje**.
- Un mensaje tiene un **nombre de operación** o **selector** y un conjunto de **argumentos**.
- Sintáxis de la invocación de mensajes:

  objeto *mensaje*

Tipos de mensajes:

Tipo de mensaje	***Ejemplo***	***Selector***
unario	'pitufo' reverse	reverse
binarios	17 <= 14	<=
keyword	miCuenta transferir: 300 a: otraCuenta	transferir:a:

Nota: ¡Smalltalk distingue mayúsculas y minúsculas!

Precedencia de mensajes:

- primero **unarios**, depués **binarios** y por último **keyword**
- asociatividad de **izquierda a derecha**

Ejemplos:

Expresión	*Evaluación*
5 + 6 * 2	(5 + 6) * 2 = 22
#(6 5 4 3 2 1) at: 2 * 3	#(6 5 4 3 2 1) at: (2 * 3) = 1
'iglú de mijo' size * 4 between: 6 negated and: 3 factorial * 5	(('iglú de mijo' size) * 4) between: (6 negated) and: ((3 factorial) * 5) = false

"Todo es un objeto" (I)

- Los enteros son objetos

  7 + 4

  El mensaje "**+ 4**" es enviado al objeto **7**. Eventualmente, cuando finaliza la ejecución de la operación, se devuelve otro objeto, el objeto **11**.

- Todos los argumentos y los objetos respuesta de los mensajes son objetos
- Los mensajes son objetos

Nota: los enteros, así como otros valores numéricos, son *inmutables*, es decir no cambian nunca. Otros objetos como las cuentas bancarias son *mutables*.

Efecto y *respuesta de enviar un mensaje*

- En Smalltalk, el resultado normal de mandar un mensaje a un objeto es obtener otro objeto como respuesta.
- Enviar un mensaje tienen dos consecuencias:
  1) Algo ocurre, es decir, la operación tiene un *efecto*
  2) Un objeto se *devuelve* como respuesta

Effect vs. return

- En ciertos mensajes es más importante el **efecto** que producen que el objeto que devuelven como resultado

  Ejemplo:

  ```
  7 storeOn: someFile
  ```

  El mensaje tiene el **efecto** de guardar el objeto 7 en algún archivo en el disco al que hace referencia *someFile*. El efecto es el propósito del mensaje.

- En otros mensajes el efecto no tiene importancia y lo que realmente interesa es el **resultado.**

  Ejemplo:

  ```
  7 factorial
  ```

  En este caso lo que nos interesa es lo que **devuelve** el mensaje.

Print it vs. Do it

- Al evaluar una expresión en Smalltalk se puede explicitar si sólo interesa el efecto de un mensaje o si también interesa la respuesta.

  Print it: el efecto del mensaje ocurre **y** Smalltalk muestra la respuesta en la pantalla
  Do it: el efecto del mensaje ocurre pero Smalltalk descarta el objeto obtenido como respuesta

  Ejemplos:

a := #(1 2 3) copy.	Crea un arreglo con los numeros 1, 2 y 3.
a at: 1 put: 5.	Pone en la primera posición de A el entero 5. Nos importa el **efecto**
a at: 1.	Pide la primera posición de A. Nos importa lo que **devuelve**

Nota: Los literales de Arrays generan arreglos inmutables. Por eso usamos copy, que crea otro arreglo con el mismo contenido, pero mutable.

Cascada de mensajes:

- Para enviar varios mensajes al mismo objeto.
- Se separan con **;**

Sintaxis:

```
objeto  mensaje1;
        mensaje2;
        ...;
        mensajeN.
```

Ejemplos:

```
Transcript cr;
        show: 'Ser ';
        show: 'o no ';
        show: 'ser...';
        cr.
```

```
'salt' copy at: 1 put: $m; yourself.
```

yourself pide al objeto receptor que se devuelva a sí mismo.

Definición de Objetos en Smalltalk

Vimos cómo se opera con objetos.
Programar en Smalltalk consiste en definir nuevos objetos. Estas definiciones de objetos son las clases.

Clases

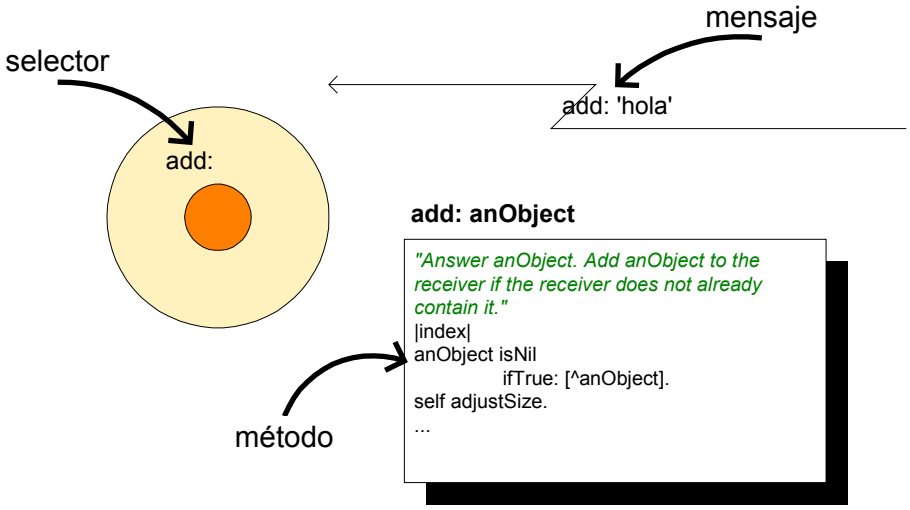
- Los objetos se crean a partir de una clase.
- La clase **define** el comportamiento de los objetos de esa clase. Los objetos construidos a partir de la misma clase tienen el mismo *comportamiento*
- La clase también define los datos propios de los objetos de esa clase.
- A los objetos de una clase A se los llama **instancias** de A.
- Los datos propios de un objeto se almacenan en *variables de instancia*.
- Los datos compartidos por todas las instancias de una clase se almacenan en *variables de clase*. Son accesibles por cualquier operación de la clase y de la instancia.

"Todo es un objeto" (II)

- Las clases son objetos. **String** es una variable global que apunta al objeto que define la clase String.
- Reciben mensajes como cualquier objeto.
- **new** es un mensaje que pueden recibir las clases y cuyo objetivo es pedir que se *construya* una nueva instancia (i.e. construyen un objeto de la clase receptora).
- Los mensajes sin el receptor son instancias de la clase ***Message***.
- Los mensajes, incluyendo el receptor, son instancias de la clase ***MessageSend***.

Métodos y mensajes

- Un **método** es el código que define el comportamiento del objeto cuando recibe un **mensaje**.
- El nombre de un método se llama *selector* (en otros lenguajes *signatura*).
- El selector sirve para encontrar la implementación (método) correspondiente a un mensaje.



Ejemplo de un método

```
replaceLastBy: anObject
        "Reemplaza el último elemento por anObject"
|last|

last := self size.
self at: last put: anObject.
^self
```

- La primera línea contiene el selector más un nombre para los argumentos.
- La segunda línea contiene un comentario que describe el método.
- La siguiente línea declara una variable local. Se pueden declarar varias variables locales, separadas por espacios.
- **self** se refiere al objeto receptor del mensaje.
- Los puntos son separadores de instrucciones.
- ^ indica el objeto que se devuelve. finaliza el método.

Todos los métodos devuelven algo. Por defecto, **self**.

Objeto	Mensaje	Selector	Efecto o respuesta
MiCuenta	MiCuenta withdraw: 250	withdraw:	Se realiza una extracción en MiCuenta
195	195 - 34	-	Devuelve 161
'hola mundo!'	'hola mundo!' size	size	Devuelve 11
MiConjunto	MiConjunto add: 'hola'	add:	Agrega el objeto 'hola' a MiConjunto

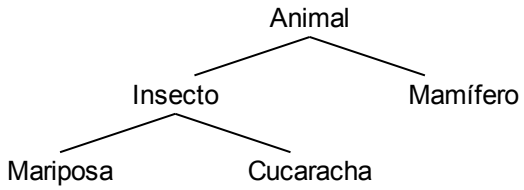
- Los "nombres" de los métodos que ve el usuario de un objeto pueden denominarse:
  - protocolo
  - comportamiento
  - interfaz
  - servicios
  - *public member functions* (C++)
- Las variables de instancia también pueden ser llamadas:
  - atributos
  - características
  - memoria
  - estado
  - *private member data* (C++)

Métodos de instancia y de clase

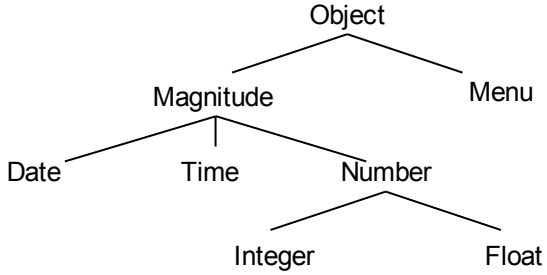
- Las clases son objetos. **Hay métodos de clase, que se definen de manera similar**. Por ejemplo, new.
- Puede haber datos propios de cada clase. Estos se almacenan en **variables de instancia de clase**, y son sólo accesibles por métodos de clase.
- Puede haber datos compartidos por todas las instancias de una clase. Éstos se almacenan en **variables de clase**, y son accesibles por los métodos de instancia y de clase.

Herencia

- Se pueden estructurar las clases diciendo que una es un tipo especial (*kind of*) de otra.



- Una ***Mariposa*** es una *subclase* de ***Insecto***. ***Insecto*** es la *superclase* de ***Mariposa*** y de ***Cucaracha***
- Se dice que ***Mariposa*** *hereda de* ***Insecto***.
- La relación entre una clase y una subclase es la relación **Es-un** (también **A-Kind-Of** en otros dominios como IA).
- Idea: si conecto de alguna manera a las mariposas con los insectos, se establece una agrupación mental que me permite *reusar* conocimiento que ya tengo sobre los insectos y aplicarlos a las mariposas. Todo lo que es cierto sobre los insectos, automáticamente también es cierto para las mariposas (6 patas, distribución de las patas, metamorfosis, etc.)



- ***Float*** es un tipo especial de ***Number***
- Todo lo que es cierto sobre los números (se pueden multiplicar, etc) es cierto también para los puntos flotantes y para los enteros.
- A su vez todo es subclase de ***Object*** en Smalltalk.
- ***Magnitude*** tiene el comportamiento común entre fechas, horas y números (relación de orden total).

Intuición:
- los objetos de una subclase tienen *más* cualidades que los objetos de su superclase
- los objetos de una subclase son más específicos que los de su superclase, los objetos de una superclase son más generales que los de sus subclases.

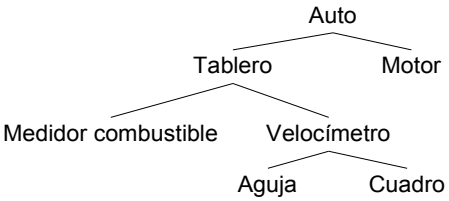
Sintaxis en Smalltalk para definir una subclase (a mano):

```
Insecto subclass: #Mariposa ...
```

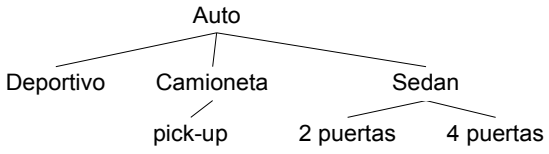
Herencia y mensajes

- Smalltalk ejecuta el primer método, con el selector correspondiente al mensaje, que encuentra en la jerarquía empezando desde la clase a la cual pertenece el objeto y subiendo por las superclases.
- Esta búsqueda (que se denomina *method lookup*) se hace en tiempo de ejecución (*binding dinámico/late binding/dispatch dinámico*).
- De este modo una subclase **hereda el comportamiento** de sus superclases.
- En otros lenguajes OO tipados, las clases determinan el tipo de sus instancias. Hay reglas de verificación de tipos que dependen de la jerarquía de clases. No es el caso del Smalltalk, porque no es tipado.

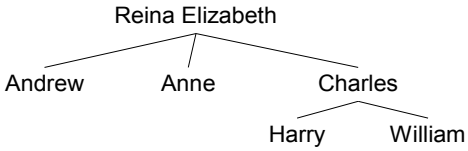
Ejercicio (Jerarquías)



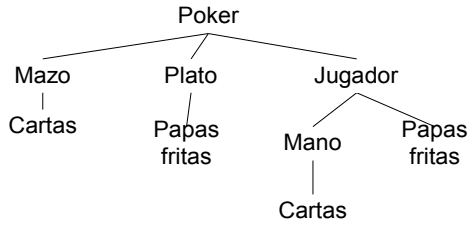
jerarquía 1



jerarquía 2



jerarquía 3



jerarquía 4

Para cada una de estas jerarquías, decidir si es razonable decir que representa una relación de herencia.

Discusión:

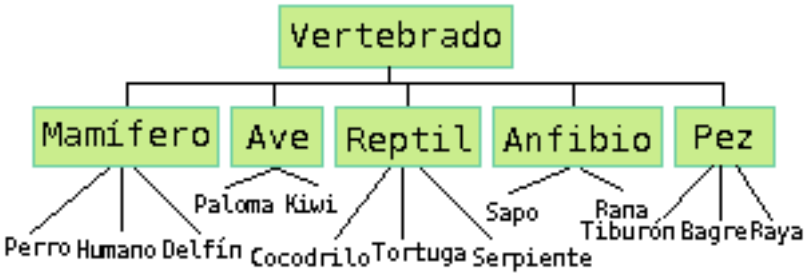
Jerarquía 1:
- Definitivamente no es herencia: los motores no son casos particulares de autos.
- Es una jerarquía de partes (*part-of*, *aggregation*, *assembly*, *whole-part*, *composite*, *has-a*).
- Este tipo de jerarquías es todavía más importante que la de herencia (en algunos casos se lo toma como elemento principal de los lenguajes de objetos).

Jerarquía 2:
- Claro ejemplo de herencia.
- Las Pick-ups son un tipo especial de Camionetas, que a su vez son Automóviles.

Jerarquía 3:
- Charles no *es un* Reina Elizabeth.
- El problema fundamental es que la herencia es entre *clases* y en la jerarquía 3 hay *objetos.*

Jerarquía 4:
- Tampoco es herencia.
- Otro ejemplo de *aggregation.*
- En este caso, un mismo nodo aparece más de una vez.

Ejemplo: herencia en Smalltalk



Perro new deciAlgo	new crea una nueva Perro. Smalltalk responde 'No tengo nada para decir...'
	Buscamos el método deciAlgo en Perro
	Buscamos el método deciAlgo en Mamifero
	Buscamos el método deciAlgo en Vertebrado -> ahí está la implementación
Delfín new deciAlgo	Responde '¡Gracias por los pescados!'
Vertebrado new deciAlgo	Sigue respondiendo 'No tengo nada para decir...'

Ejercicios:

- Agregar una clase ***Loro*** y hacer que diga siempre la misma frase aprendida.

```
deciAlgo
        self display: frase
```

- Hacer que Humano diga "Mi nombre es _____"

```
Humano superclass: Mamifero

deciAlgo
        self display: 'Mi nombre es ', nombre
```

- Hacer que Humano diga "Mi nombre es...." , después de lo que dicen los vertebrados.

```
deciAlgo
        super deciAlgo.
    self display: 'Mi nombre es ', nombre
```

super altera el orden en que se búscan los métodos, comenzando a buscar por la superclase de la clase donde está definido el método que hace referencia a super.

Predecir el resultado de:

'iglú de mijo' class	ByteString
Kiwi new class superclass	Ave
(2/7) class superclass superclass	ArithmeticValue
P := Paloma new. P isKindOf: Vertebrado P isMemberOf: Vertebrado P isMemberOf: Paloma	 true false true
K := Kiwi new. K class == Kiwi K respondsTo: #deciAlgo K respondsTo: #vola	 true true false
Paloma allInstances size	1

isKindOf: dice si un objeto es instancia de la clase o alguna subclase
isMemberOf: dice si el objeto es exactamente de esa clase

Creación e Inicialización de objetos

- Un objeto o instancia se crea enviándole un mensaje a la clase correpondiente.

  Ejemplo:
  ```
  Perro new.
  ```

- Se pueden definir métodos que creen instancias y reciban parámetros.

  Ejemplo:
  ```
  Humano conNombre: 'Pepe'
  ```

```
Humano conNombre: 'Pepe'   (metodo de clase)

    ^ (self new) nombre: 'Pepe'.
```

- Para inicializar instancias se estila crear un método de instancia (por ej. initialize) al que llaman los métodos de clase.

Re-uso:

Desafío:

- Escribir un método *pepe* en una clase AAA con el código:

  "... un monton de código importante..."
  Transcript show: 'To be or not to be'.
  "... otro monton de codigo super importante..."

- Crear una subclase BBB que en el método *pepe* haga exactamente lo mismo que AAA, pero en lugar de escribir 'To be or not to be', escriba 'My kingdom for a horse'.
- No debe quedar código repetido.

```
AAA >> pepe
        "...un monton de código importante..."
        Transacript show: (self miFrase).
    "... otro monton de codigo super importante..."

AAA >> miFrase
        ^ 'To be or not to be'

BBB superclass: AAA

BBB >> miFrase
        ^ 'My kingdom for a horse'
```

Smalltalk

Imagen:

- Contiene todos los objetos que se crearon y se usaron, así como todos los objetos que son críticos para Smalltalk.
- Cuando se carga una imagen, todos los objetos vuelven a estar disponibles
- En VisualWorks, las imágenes son archivos con extensión **.im**. La imagen por defecto se llama **visualnc.im** y está en el subdirectorio images del directorio de instalación de VisualWorks.
- Hay que generar copias de la imagen.
- Smalltalk distingue mayúsculas y minúsculas.
- Es importante respetar la convención de nombres de variables y de métodos.
- Las variables no tienen tipo pero los objetos son fuertemente tipados.
- Todas las clases (como Delfín) son instancias de otra clase, que se llama la *metaclase*. Cada clase es la única instancia de su metaclase. Todas las metaclases son instancias de la misma clase, ***Metaclass***.