POO: Conceptos

Objeto

En el paradigma orientado a objetos, un objeto es una entidad que tiene una identidad propia, una estructura interna y una serie de comportamientos asociados. Una característica fundamental de los objetos es que tienen una memoria interna (llamada "estado interno" o "atributos") que almacena información específica para ese objeto, y una serie de métodos (también llamadas operaciones) que definen el comportamiento que se puede realizar con ese objeto. Los objetos son autónomos y pueden interactuar entre sí a través de la comunicación de mensajes, lo que permite crear sistemas complejos y flexibles. Los objetos pueden ser instanciados desde una clase, que define la estructura y el comportamiento de los objetos, y pueden ser utilizados para modelar conceptos y entidades del mundo real, como personas, objetos físicos, conceptos abstractos, etc.

En Smalltalk, todo es un objeto. No hay forma de crear, dentro del lenguaje, una entidad que no sea un objeto. Se compone de una memoria privada y un conjunto de métodos. La memoria privada de un objeto es accesible únicamente por ese objeto, lo que garantiza la encapsulación y la seguridad de los datos. Ningún objeto puede leer o modificar los valores de memoria de otro objeto, brindando integridad de la información. En cuanto a los métodos, la naturaleza de las operaciones depende del tipo de objeto que se está representando. Por ejemplo, los objetos que representan números calculan funciones aritméticas, mientras que los objetos que representan strings realizan operaciones de manipulación de cadenas de caracteres.

Mensaje

Un mensaje es una solicitud para que un objeto realice una de sus operaciones. Los mensajes se envían a objetos, a quienes se los llama objeto receptor. Un mensaje especifica qué operación se desea ejecutar, pero no cómo debe llevarse a cabo. El receptor, el objeto al que se envió el mensaje, determina cómo llevar a cabo la operación solicitada.

El conjunto de mensajes a los que un objeto puede responder se denomina su interfaz con el resto del sistema. La única forma de interactuar con un objeto es a través de su interfaz. Una propiedad crucial de un objeto es que su memoria privada solo puede ser manipulada por sus propias operaciones. Una propiedad crucial de los mensajes es que son la única forma de invocar las operaciones de un objeto. Estas propiedades aseguran que la implementación de un objeto no puede depender de los detalles internos de otros objetos, solo de los mensajes a los que responden.

Los mensajes aseguran la modularidad del sistema porque especifican el tipo de operación deseada, pero no cómo debe llevarse a cabo esa operación. Por ejemplo, para interactuar con cualquier objeto, uno solo necesita saber a qué mensajes responde, no cómo está implementado.

Métodos

Los métodos son la especificación de cómo los objetos responden a los mensajes. Un método describe cómo un objeto llevará a cabo una de sus operaciones. Los métodos acceden a la estructura interna del objeto y también a enviar mensajes a sí mismo y otros objetos. Un método es la codificación de cómo se realizan las operaciones a las que responderán los objetos.

Pueden tener ninguno o muchos argumentos. Los argumentos son pseudo-variables, ya que se puede acceder a su valor pero de ninguna manera su valor puede ser modificado por una asignación.

Al finalizar la ejecución del método, siempre retornan un valor, puede estar explícito por medio del uso de una expresión de retorno (^), o bien no estar, en cuyo caso retorna el objeto receptor sobre quien se está ejecutando el método.

Clase

Una clase es una entidad que define un tipo de objeto y establece sus características y comportamientos. Define la estructura y el comportamiento de un objeto, incluyendo sus atributos (datos) y métodos (funciones u operaciones). Los objetos individuales descritos por una clase se llaman instancias. Cada objeto en el sistema es una instancia de una clase.

Cada clase tiene un nombre que describe el tipo de componente que representan sus instancias. Los nombres de las clases se escriben siempre en mayúsculas, y estos nombres son parte del lenguaje. Por ejemplo, la clase cuyas instancias representan una tira de caracteres se denomina String. La clase cuyas instancias representan un punto en el plano cartesiano se denominan Point. La clase cuyas instancias representan números flotantes se denomina Float.

Las clases están compuestas por **atributos** (memoria privada que utilizarán los objetos o instancias creadas a partir de dicha clase) y **comportamientos** (operaciones que puede realizar el objeto o la instancia). Los atributos se denominan variables de instancia, y cada variable de instancia definida en la clase se convertirán en la memoria privada de los objetos creados a partir de esa clase, cada variable puede referenciar a un único objeto, y dicho objeto es su valor asignado.

Programar consiste en crear clases, crear instancias a partir de dichas clases (objetos) y especificar secuencias de intercambio de mensajes entre estos objetos.

Literales

Algunos objetos pueden ser referidos por expresiones literales. Dado que el valor de una expresión literal siempre es el mismo objeto, estas expresiones también se llaman constantes literales. Algunos tipos de objetos que se pueden crear a partir de literales son:

- Números
- Caracteres
- Strings
- Simbolos
- Arreglos de otros literales

Los **números** son objetos que representan valores numéricos y responden a mensajes que calculan resultados matemáticos, lógicos, entre otros. La representación literal de un número es una secuencia de dígitos que puede ser precedida por un signo menos y/o seguida por un punto decimal y otra secuencia de dígitos. Se pueden especificar también en otra base que no sea la decimal, por medio de poner la base seguida de la letra “r” y luego el número.

```
583
45.78
12/7
16rFF
```

Los **caracteres** son objetos que representan los símbolos individuales de un alfabeto. Una expresión literal de carácter consiste en un signo de pesos seguido de cualquier carácter.

```
$c
$F
```

Los **strings** son objetos que representan secuencias o cadenas de caracteres. Las cadenas responden a mensajes que acceden a caracteres individuales, reemplazan subcadenas y realizan comparaciones con otras cadenas. La representación literal de una cadena es una secuencia de caracteres delimitada por comillas simples.

```
‘Hola mundo’.
```

Los **símbolos** son objetos que representan cadenas de caracteres utilizadas para nombres en el sistema. La representación literal de un símbolo es una secuencia de caracteres alfanuméricos precedida por el signo numeral (#). Cada símbolo es único y nunca habrá dos símbolos con los mismos caracteres. Esto hace posible comparar símbolos de manera eficiente en el sistema.

```
#factorial
#min:
```

```
#detect:ifNone:
#between:and:
```

Un **arreglo** es un objeto con una estructura de datos simple cuyo contenido puede ser referenciado por un índice entero de uno a un número que es el tamaño del arreglo. Los arreglos responden a mensajes solicitando acceso a su contenido. La representación literal de un arreglo es una secuencia de otros literales: números, caracteres, cadenas, símbolos y arreglos, delimitados por paréntesis y precedidos por el signo numeral. Los otros literales están separados por espacios. Los símbolos y arreglos incrustados dentro de otro arreglo no deben estar precedidos por el signo numeral (#).

```
#(1 2 3)
#('hola' 45 800.5)
#(45 (30 25) factorial)
```

Variables de instancia

Una variable de instancia es una variable declarada dentro de una clase y se asocia a cada objeto que se crea a partir de esa clase. Estas variables se inicializan cuando se crea un objeto y se pueden modificar en cada objeto individualmente.

Relacionando conceptos, cuando hablamos de clases, decimos que las clases definen atributos y comportamientos que tendrán los objetos creados a partir de ella. Esos atributos son las variables de instancia.

También, cuando hablamos de que un objeto tiene una memoria privada, dicha memoria interna está definida por variables de instancia. Estas variables de instancia tienen un nombre, cada una hace referencia a un único objeto y por medio de ese nombre se puede acceder al objeto referenciado por la variable de instancia. Cada objeto tiene su propia memoria privada y los cambios en variables de instancia de un objeto no afectan a otros objetos.

Pseudo-variables

Una pseudo-variable es un identificador que se refiere a un objeto. De esta manera, es similar a una variable. Sin embargo, una pseudo-variable difiere en que su valor no puede ser cambiado con una expresión de asignación. Su valor es constante y siempre se referirán a los mismos objetos.

nil Hace referencia a un objeto indefinido, un objeto de la clase UndefinedObject. Este objeto indefinido es utilizado como el valor de una variable cuando ningún

otro objeto es apropiado. Por ejemplo, las variables que no han sido inicializadas apuntarán a nil.

true Se refiere a un objeto que representa el valor lógico verdadero. Refiere a una instancia creada a partir de la clase True.

false Se refiere a un objeto que representa el valor lógico falso. Refiere a una instancia creada a partir de la clase False.

self Se refiere al objeto receptor, quien está enviando el mensaje a ser ejecutado. Esta variable puede ser utilizada por ejemplo para operaciones recursivas, como el cálculo del factorial. Esta variable no hace referencia a instancias de una clase en particular, sino que puede referenciar a instancias de cualquier clase.

super Se refiere al igual que self al objeto receptor del mensaje, pero se diferencia en que la búsqueda del método a ejecutar comenzará en la superclase en lugar de hacerlo en la propia clase del objeto receptor. De esta forma se habilita a un método poder acceder a métodos definidos en la superclase incluso cuando estén redefinidos.

Mensajes

Un mensaje es un mecanismo que permite que un objeto interactúe con otros objetos o con sí mismo. Los mensajes representan la interacción entre objetos. Son quienes solicitan la realización de una operación al receptor del mensaje, el receptor procesa el mensaje, realiza la acción solicitada y devuelve una respuesta.

Una expresión de mensaje se compone de tres elementos:

```
receptor selector argumentos
```

El receptor y los argumentos son un objeto, que puede ser escrito de forma literal o bien ser una expresión que retorne un objeto.

El selector es el nombre de la operación que se quiere realizar sobre el objeto receptor. Y los argumentos son datos necesarios para llevar a cabo la operación.

El selector es quien determina qué operación se realizará sobre el objeto receptor y los argumentos son otros objetos que están involucrados en la realización de la operación.

Tipos de mensajes

Dependiendo del tipo de operación y la cantidad de argumentos que necesite dicha operación para poder ser procesada, existen tres tipos de mensajes, mensajes unarios, binarios y de palabra clave. A continuación se describe cada uno de ellos y se dan ejemplos de la sintaxis de los mismos.

Mensajes Unarios

Son mensajes que no tienen argumentos. En estos mensajes sólo interviene el objeto receptor y no necesita de ningún otro objeto para poder responder a la operación solicitada.

Formato:

```
objetoReceptor mensaje
```

Ejemplos:

`5 factorial`	El objeto receptor es el número `5`, un número entero creado como un literal. El mensaje está dado por el selector `#factorial`
`tita sin`	El objeto receptor es la variable `tita` y el mensaje está dado por el selector `#sin`
`alumno nombre`	El objeto receptor es la variable `alumno` y el mensaje está dado por el selector `#nombre`
`#(45 10.0 $t) first`	El objeto receptor es un arreglo creado de forma literal. El mensaje está dado por el selector `#first`

En todos los ejemplos, el objeto receptor debe contener un objeto acorde al mensaje que está recibiendo para poder responder satisfactoriamente. Por ejemplo, la variable `tita` debe contener un valor numérico para poder responder al mensaje `#sin`, si no fuera el caso responderá con un “Message does not understand” (mensaje no comprendido), que significa que el objeto receptor no puede responder al mensaje recibido.

Mensajes de palabra clave

Son mensajes en los que se puede enviar uno o más argumentos. El selector de un mensaje de palabras clave está compuesto por una o más palabras clave, y cada palabra clave precedida por un argumento. Una palabra clave es un identificador simple seguido de dos puntos. Existirá una palabra clave por cada argumento que sea

necesario. El nombre del método será todas las palabras claves concatenadas, teniendo en cuenta que los dos puntos ":" son también parte del nombre.

Formato:

```
objetoReceptor claveUno: argUno claveDos: argDos claveTres: argTres …
```

Ejemplos:

`index max: limit`	El objeto receptor es la variable `index`, el mensaje contiene un argumento y está dado por el selector `#max:` y el argumento es la variable `limit`
`3 between: 2 and: 5`	El objeto receptor es el número `3`, creado como un literal, el mensaje contiene dos argumentos y está dado por el selector `#between:max:` y los argumentos son el número `2` y `5` creados como literales.
`#(45 10.0 $t) at: pos`	El objeto receptor es el arreglo creado a partir de una expresión literal, el mensaje está dado por el selector `#at:` y el argumento recibido es la variable `pos`
`col at: 4 put: 45.03`	El objeto receptor es la variable `col`, el mensaje contiene dos argumentos y está dado por el selector `#at:put:` y los argumentos son el valor literal `4` y el valor literal `45.03`

Cuando se quiere referenciar al selector de un mensaje de múltiples palabras claves, se debe hacer por medio de concatenar dichas palabras clave. De los ejemplos anteriores se puede resaltar los selectores `#between:and:` y `#at:put:`.

Mensajes Binarios

Los mensajes binarios son un tipo de mensaje especial que toman un solo argumento.

El selector de un mensaje binario no puede ser creado con cualquier carácter, sino que debe estar compuesto por uno o dos caracteres no alfanuméricos, con la única restricción de que el segundo carácter no puede ser un signo menos (-).

Los selectores binarios tienden a ser utilizados para mensajes aritméticos, pero pueden ser utilizados para múltiples propósitos.

Formato:

```
objetoReceptor mensaje argumento
```

Ejemplos:

`3 + 4`	El objeto receptor es número `3` creado como un literal, el mensaje está dado por el selector `#+` y el argumento es el valor literal `4`
`total - 1`	El objeto receptor es la variable `total`, el mensaje está dado por el selector `#-` y el argumento es el valor literal `1`
`total <= max`	El objeto receptor es la variable `total`, el mensaje está dado por el selector `#<=` y el argumento es la variable `max`
`'Hola' == 'Chau'`	El objeto receptor es el string `'Hola'` creado como un literal, el mensaje está dado por el selector `#==` y el argumento es otro string creado como literal `'Chau'`

Valores retornados

Los mensajes brindan una comunicación bidireccional, ya que los mensajes contienen toda la información para que el objeto receptor pueda realizar la operación deseada y siempre se espera un resultado por la operación realizada.

En palabras más técnicas, el selector y los argumentos transmiten información al objeto receptor sobre qué operación y tipo de respuesta se busca. El objeto receptor procesa la operación con la información recibida y transmite la respuesta retornando un objeto que se convierte en el valor obtenido por el envío de la expresión del mensaje.

Si la expresión de mensaje está asignada a una variable, el resultado de esa expresión será el nuevo objeto al que haga referencia la variable. Si la expresión de mensaje es parte de un argumento en un mensaje de palabra clave o binario, el resultado de esa expresión será el objeto referenciado por la variable utilizada como argumento.

Ejemplos:

`sum := 3 + 4`	La expresión `3 + 4` compuesta por el objeto receptor `3`, el selector `#+` (mensaje binario) y el argumento `4`, dicha expresión retornará el valor `7` como resultado de hacer la operación de sumar. El objeto `7` será el nuevo valor de la variable `sum`
`5 between: 3 factorial and: 80`	Prestemos atención primero a la expresión `3 factorial` interactuando dentro de un mensaje de palabra clave. Aislando este mensaje unario tenemos que el objeto

	receptor es el número `3` y el mensaje es `#factorial`, el resultado de este mensaje es el número `6`. Tomando la expresión completa `5 between: 3 factorial and: 80` tenemos que el objeto receptor es el número `5`, el mensaje esta dado por el selector `#between:and:` y los argumentos son: el número `6` (resultado del factorial) y el número `80`.

El objeto receptor siempre retornará un valor ante la ejecución del mensaje, incluso cuando este valor retornado no sea necesario de utilizar. Devolver un valor indica que la respuesta al mensaje está completa. Algunos mensajes están destinados únicamente a informar al receptor de algo y en esos casos la respuesta no se utilizará pero existirá de todos modos.

Un mensaje puede tener un valor de retorno explícito por medio del uso del "^", o bien, si no existe explícitamente se retornará el objeto receptor.

Precedencia en la ejecución de mensajes

Cuando el objeto receptor o los argumentos de un mensaje son otras expresiones (mensajes) aparece el problema de quien se ejecuta primero.

Los mensaje unarios toman precedencia sobre los mensajes binarios. Los mensajes binarios tienen precedencia sobre los mensajes de palabra clave. Para alterar la precedencia se utilizan paréntesis. Cuando existen mensajes con el mismo nivel se toma la regla de ejecución de izquierda a derecha, es decir se ejecuta primero el mensaje que está a la izquierda.

Ejemplos

`1.5 tan rounded`	Tenemos el objeto receptor `1.5` y dos mensajes unarios: `#tan` y `#rounded`. Ambos son mensajes unarios y por lo tanto tienen el mismo peso en cuanto a la precedencia. La ejecución se realiza de izquierda a derecha, por lo tanto se ejecuta primero el mensaje `#tan` y a ese resultado será el objeto receptor que recibirá el mensaje `#rounded`
`index + offset * 2`	Tenemos el objeto receptor `index` y dos mensajes binarios: `#+` y `#*`. Ambos son mensajes binarios y por lo tanto tienen el mismo peso en cuanto a la precedencia. Nuevamente la ejecución se realiza de izquierda a derecha, por lo que en primer lugar se realiza la operación de suma `index + offset` y su resultado luego será utilizado como objeto receptor del mensaje `#*` quien recibe como argumento al número `2`.

Notar el segundo caso de mensaje binario, donde el seguimiento de las reglas de precedencia no corresponde con la precedencia matemática, ya que la expresión `index + offset * 2` en Smalltalk primero se suma y luego se multiplica, y en una expresión matemática debería primero hacer la multiplicación y luego la suma. Para alterar esta situación se utilizan paréntesis: `index + (offset * 2)`.

Los mensaje unarios toman precedencia sobre los mensajes binarios. Ejemplos:

`frame width + 2`	Tenemos los mensajes #with y #+. Primero se ejecuta el mensaje unario `#width` y el resultado se utiliza como objeto receptor del mensaje `#+` para realizar la suma utilizando el número `2` como argumento.
`2 * tita sin`	En este caso tenemos los mensajes `#*` y `#sin`. El mensaje binario aparece a la izquierda el mensaje unario, pero como los mensajes unarios tienen precedencia sobre los binarios, se ejecuta primero el mensaje `#sin` y luego ese resultado es utilizado como parámetro del mensaje `#*`

En cuanto a los mensajes de palabra clave, cuando aparecen sin paréntesis ellos componen un único selector. Si es necesario separarlos, hay que hacer uso de los paréntesis:

`frame scale: factor max: 5`	Esta expresión se toma como un único selector `#scale:max:` y `frame` es el objeto receptor, siendo `factor` y el número `5` los argumentos

En el caso de que sean separados se debe escribir de esta manera:

`frame scale: (factor max: 5)`	En este caso tenemos dos mensajes de palabra clave, y por lo tanto dos selectores. El selector `#max:` tiene a la variable `factor` como objeto receptor y al número `5` como argumento. El selector `#scale:` tiene a la variable `frame` como objeto receptor y como argumento al resultado de la operación del máximo entre `factor` y `5`.

Los mensajes binarios tienen precedencia sobre los mensajes de palabra clave:

`bigFrame width: smallFrame width * 2`	En este caso, nombrando de izquierda a derecha, tenemos el mensaje de palabra clave `#width:`, el mensaje unario `#width` y el mensaje binario `#*`. El primer mensaje que se ejecutará es el mensaje

	unario `#width` enviado al objeto receptor `smallFrame`. Luego ese resultado se convertirá en el objeto receptor del mensaje `#*` utilizando al número `2` como argumento. Por último, el resultado de la expresión `smallFrame width * 2`, será utilizada como argumento del mensaje `#width:` enviado al objeto receptor `bigFrame`.

Resumen precedencia de mensajes

```
(mensaje) > Unario > Binario > Palabra clave
```

1. Primero se ejecutan las expresiones entre paréntesis.
2. Luego se ejecutan los métodos unarios, luego los binarios y por último los de palabra clave.
3. Si hay dos mensajes del mismo nivel, se ejecuta primero el que esté a la izquierda (de izquierda a derecha)

Estas reglas sirven para minimizar el uso de paréntesis en el código.

Mensajes en cascada

Existe una forma sintáctica especial llamada cascada que especifica múltiples mensajes al mismo objeto. Cualquier secuencia de mensajes puede ser expresada sin usar cascada. Sin embargo, la cascada a menudo reduce la necesidad de usar variables. Una expresión de mensaje en cascada consiste en una descripción del receptor seguida de varios mensajes separados por punto y coma.

```
col := OrderedCollection new.
col add: 1; add: 2; add: 3.
```

Variables

Dando una descripción general, los datos se almacenan de dos maneras: por valor, forma utilizada para almacenar tipos de datos primitivos (enteros, flotantes, caracteres, strings, booleanos); o por referencia, donde en lugar de almacenar directamente la información, se almacena la dirección de memoria donde está alojada la información.

En los lenguajes orientados a objetos se utiliza la forma por referencia para tipos de datos complejos como pueden ser los objetos. Entonces, cuando decimos que asignamos un objeto a

una variable, lo que en realidad se está asignando a esa variable es la dirección de memoria donde se encuentra el objeto, es decir se asigna una referencia al objeto.

En Smalltalk específicamente, todo es un objeto, por lo tanto cuando se asigna cualquier valor a una variable lo que estamos asignando es una referencia al objeto almacenado. Para el desarrollador, se tratan de manera exactamente igual un tipo de dato primitivo o un objeto instancia de una clase. La máquina virtual de objetos es la que implementa este mecanismo y tiene formas de optimización para el acceso y manipulación tanto de datos primitivos como de los objetos.

Variables de instancia

Las variables de instancia existen mientras el objeto exista. Cuando se crea un nuevo objeto (enviando el mensaje `#new` a la clase), el nuevo objeto contendrá el conjunto de variables de instancias como su memoria interna. El valor por defecto que se les asigna a las variables de instancia es `nil`.

Variables temporales

Se definen como apoyo para realizar una determinada actividad y existen sólo mientras la actividad exista (entiéndase actividad como el código de un método o un trozo de código para ejecutar en un workspace). Las variables de instancia representan el estado actual de un objeto. Las variables temporales representan el estado transitorio necesario para llevar a cabo alguna actividad. Por lo general, las variables temporales están asociadas con una única ejecución de un método: se crean cuando un mensaje hace que se ejecute el método y se descartan cuando el método completa al devolver un valor.

Variables de clase, pool y globales

Las variables de clase, globales y pool pueden ser accesibles por más de un objeto. Se escriben comenzando en mayúscula, y lo que diferencia a este tipo de variables entre sí es el ámbito de visibilidad o accesibilidad que poseen.

Variables de clase

Son variables accesibles por todas las instancias de la clase. Por ejemplo, si tenemos una variable de clase definida en la clase Persona, esa variable será accedida únicamente por objetos creados a partir de la clase Persona.

Variables Pool

Son variables accesibles por objetos de un subconjunto de clases del sistema. Supongamos que tenemos dos clases Empresa y Organismo, ambas definen una variable pool, dicha variable será accedida únicamente por objetos creados a partir de la clase Empresa u Organismo.

Variables Globales

Son variables accesibles por todas las instancias de todas las clases, esto significa que son accesibles por cualquier objeto existente en el sistema.

Asignación

La asignación se produce como una referencia a un objeto. La expresión de asignación se escribe por medio del operador `:=`, donde a la izquierda de dicho operador estará el nombre de una variable y a la derecha la expresión lo que se quiere asignar.

Una constante literal siempre se referirá al mismo objeto, pero un nombre de variable puede referirse a diferentes objetos en diferentes momentos. El objeto referido por una variable cambia cuando se evalúa una expresión de asignación.

```
cantidad := 19.
persona := Persona new.
```