Definiciones básicas de POO

El objetivo de este apunte es describir brevemente la terminología básica de la programación orientada a objetos, y no profundizar en los fundamentos. Está pensado más bien como un repaso de POO. También para definir el lenguaje (no de programación, sino la terminología) que vamos a tratar de utilizar de manera consistente.

Programa

Un **modelo computable** de un **dominio** de la **realidad**. Definimos realidad como todo aquello de lo cual podemos hablar o describir de alguna manera, y un dominio es una porción (coherente) de esa realidad. Por ejemplo, el dominio financiero. Una característica de un programa *bueno* es cuando su *gap semántico* es corto (diferencia entre el modelo y la realidad).

Modelo computable

Modelo se refiere a una construcción que realizamos acorde a los lineamientos que nuestro enfoque de programación nos define (en nuestro caso, el enfoque orientado a objetos). Y por último, es fundamental que sea computable, es decir que pueda ser ejecutado por una computadora (por algún lenguaje Turing-completo).

Programa orientado a objetos

Es un programa, que está compuesto (desde un enfoque puro) únicamente por objetos que se envían mensajes entre sí. Cualquier cosa que "suceda" en nuestro programa será modelada como el envío de un mensaje a un objeto.

Programador OO

Es la persona que tiene 2 responsabilidades básicas: (1) observar la realidad, y en particular el dominio del problema para (2) construir un modelo que conste de objetos y mensajes. Estas dos tareas se repiten en secuencia tantas veces como sea necesario para comprender aún más el dominio.

Diseño orientado a objetos

Es la actividad de identificación de responsabilidades, selección de nombres, creación de objetos, mensajes y colaboraciones de la forma más fiel a la realidad que se pueda. También se refiere a la combinación de estos conceptos básicos para generar conceptos más avanzados y utilizarlos en la construcción del programa. Naturalmente, es un proceso iterativo e incremental (en donde vamos aprendiendo a medida que lo vamos realizando) y subjetivo ya que depende de la observación del sujeto, no obstante se apunta a tener una *subjetividad colectiva* que esté definida por una organización o equipo de desarrollo del programa. Pero si dos personas que no se conocen ni se comunican realizan un programa para el mismo dominio, vamos a obtener dos programas distintos (en el "cómo", no en el "qué").

Objeto

La representación de una cosa de la realidad. Su esencia la determinan los mensajes que sabe responder. Colabora con otros objetos para cumplir con su cometido.

Mensaje

Representa una solicitud o pedido que podemos realizar a un objeto. El mensaje tiene un nombre (también llamado firma o signatura) que lo identifica, y puede requerir de objetos adicionales para que tenga significado. Ejemplo de mensaje que no requiere de otro objeto más que el receptor, es solicitarle el factorial a un número (si al objeto 4 le solicitamos su factorial, obtendremos como resultado el objeto 24). Ejemplo de mensaje que requiere de más objetos, es saber si un número se encuentra entre otros dos ("el número 3, ¿está entre el 1 y el 8?" 3 recibe el mensaje pero 1 y 8 también son objetos, colaboradores externos).

Colaboración

La unidad fundamental de todo programa OO, el envío de un mensaje a un objeto. Lo único que ocurre en un programa de un lenguaje OO puro. La colaboración siempre entrega como resultado un objeto (el cual no estamos obligado a utilizar). El objeto que recibe el mensaje se denomina receptor.

Colaborador

Un objeto del que disponemos para enviarle mensajes en el contexto en el que nos encontramos.

Colaborador interno

Un colaborador que está asociado siempre a un objeto, desde que nace hasta que muere, y es sólo accesible por dicho objeto. También conocido (por su nombre más técnico) como *member*, o *field*, o *instance variable* (variable de instancia). Ejemplo: en un objeto que representa un par ordenado, la coordenada x y la coordenada y podrían ser colaboradores internos del par ordenado. El objeto debe nombrar a sus colaboradores. El nombre dependerá del rol que ese objeto referenciado cumple dentro del contexto del objeto que lo conoce.

Colaborador externo

Un colaborador que está asociado a un objeto durante el envío de un mensaje (o sea, una colaboración). Volviendo al ejemplo del mensaje para preguntar "el 3, ¿está entre el 1 y el 8?", tanto el número 1 como el 8 son colaboradores externos de este mensaje, y "vienen" a colaborar con el objeto 3 para el propósito de esta tarea, luego ya no son más referenciados.

Método

Cómo un objeto responde a un mensaje. Contiene un conjunto de colaboraciones. Entrega siempre un objeto como resultado.

Colaborador local

Un colaborador que se define dentro de un método, y que tiene sólo validez dentro del mismo. En otros enfoques, es lo que llamaríamos una variable, que se asigna una vez, luego se utiliza y se puede reasignar. Como estamos en POO esto no es más que un objeto, que tiene asociado un nombre y a efectos del método en el que estamos es un colaborador más con el que contar.

Asignación y retorno

Vale destacar que generalmente estas dos ideas son las excepciones a la regla de que "todo es una colaboración" o "todo ocurre como envío de mensaje a un objeto". Cuando asociamos un nombre con un objeto, estamos realizando una asignación (en la mayoría de los lenguajes esto NO es un envío de mensaje). Mientras que el retorno es un indicador que se pone en la parte del método en la cual el programador desea, valga la redundancia, retornar el objeto que será el resultado del mensaje cuyo método estamos ejecutando.

Nombre

La palabra o palabras que utilizamos para conocer a un objeto. Generalmente se escriben de manera continuada y siguiendo la convención que establece el lenguaje, como por ejemplo `cuentaADepositar`, o `sueldo`, o `DepositoDeCuentaBancaria`.

Referencia

La asociación de un nombre a un objeto. Indispensable para realizar una colaboración. Las referencias pueden cambiar con el tiempo.

Pseudo-variable

Son nombres especiales cuyas referencias son dinámicas y depende del lugar en el que estemos. Por ejemplo, `this` en Java o `self` en Smalltalk son pseudo-variables, ya que la referencia es hacia el objeto receptor del mensaje.

Clase

Objeto que representa una idea o un concepto de la realidad, por ende es un objeto abstracto. Tiene 2 responsabilidades básicas: (1) saber cómo son concretamente los objetos a los que representa como concepto y (2) crear dichos objetos que se suelen denominar *objetos concretos* o *instancias.* Ejemplo: una cosa es el objeto que representa el concepto de Perro, y otra cosa es el objeto Lupita, que es la perra de Juan. En un lenguaje de clasificación (es decir, donde cada objeto tiene su clase, es decir el concepto que lo representa), Perro sería una clase, que contendría todo aquello que es común a todos los perros del dominio, mientras que Lupita tendría lo que es particular a ella, y la clase Perro debe saber crear perros como Lupita u otros distintos.

Prototipo

Objeto que representa un "esqueleto" o "esquema" de un objeto, y que viene a resolver el mismo problema que la clasificación resuelve con las clases, que es el problema de *sharing*. En este objeto,

que también es abstracto como las clases, va a estar el comportamiento común de una familia de objetos. Siguiendo el ejemplo de perros del concepto de clase, si no tuviéramos clases pero quisiéramos tener varios perros, y no tener que "repetir código", tendríamos que poner ese código en algún lugar. Bueno, Perro entonces sería un prototipo, mientras que Lupita sería un objeto que tiene su propio comportamiento, pero que tiene como objeto padre o parent a Perro.

Clonación

Proceso por el cual a partir de un objeto, se crea un objeto similar, y a partir de allí ambos objetos comienzan una vida independiente (es decir, el objeto creado podría cambiar para diferenciarse de su original).

Jerarquía (clases o prototipos)

La forma en la que organizamos nuestros objetos que representan conceptos, ya sea usando clases o prototipos. Cada clase va a tener una superclase, que representará un concepto más general, y viceversa, una clase puede tener subclases que representarán conceptos más específicos. Análogamente, en prototipación un objeto podría tener un objeto *parent*, y un parent puede tener muchos *children* o objetos hijos.

Method lookup

Proceso por el cual se busca el método correspondiente para un mensaje. Como ya sabemos, los objetos no tienen definidos todos sus mensajes directamente, sino que algunos son compartidos en una jerarquía. El method lookup es justamente el recorrido que se hace en esa jerarquía hasta encontrar el método adecuado. Pueden pasar dos cosas: (1) se encuentra el método, entonces se ejecuta, o (2) se continúa en la superclase o parent hasta encontrarlo. Obviamente esto tiene un fin, que es el tope de la jerarquía de clases o prototipos, en donde si no se encuentra el mensaje en ese punto, se lanza un error de "mensaje no entendido por el objeto".

Igualdad

Criterio por el cual se establecen que dos objetos deberían ser tratados como similares. Por ejemplo, dos manzanas podrían ser iguales si su sabor, tamaño, forma y color son iguales. Es decir, desde el punto de vista del que usa estos objetos, saber si puede hacer una distinción entre objetos o "le da lo mismo" usar cualquier objeto. Es un criterio definido por el programador, y se corresponde al dominio en el que estamos situados.

Identidad

Característica intrínseca de cada objeto. Un objeto es idéntico sólo a sí mismo. No debería ser idéntico a otro objeto (por más que pueda ser *igual*). Podríamos tener dos referencias con distintos nombres, apuntando al mismo objeto, entonces al comparar por identidad esas referencias debería decirnos que son idénticas.

Rol

La faceta que nos interesa saber de un objeto, que podría tener varios mensajes pero dependiendo de dónde está siendo utilizado algunos mensajes podrían ser necesarios y otros no. Por ejemplo, un empleado puede tomar varios roles en una empresa (podría ser un lider, un mentor, un aprendiz, etc).

Protocolo esencial

Conjunto mínimo de mensajes que dan sentido a un objeto. Si quitamos un mensaje, el objeto pierde la esencia. El protocolo esencial puede ser compartido entre objetos de la misma familia/jerarquía (por ejemplo, todas las cuentas bancarias saben responder a mensajes como "depositar" y "extraer" dinero).

Interfaz/Tipo

Conjunto de mensajes asociados a un nombre, y que está muy relacionado a la idea de rol. Esto cobra relevancia en lenguajes de tipado estático (como Java) donde tenemos que informar siempre con qué tipo de objetos estamos tratando. Es posible tomar de un objeto, un conjunto de mensajes y hacerlo un tipo o interfaz. En ese caso se dice que el objeto implementa esa interfaz. Otros objetos podrían implementarla también. Ejemplo: hay electrodomésticos que producen sonido, entonces `ReproductorDeSonidos` podría ser un tipo, que tanto un `Televisor` como una `Computadora` podrían implementar, pero que una `Batidora` no.

Instanciación

El proceso de creación de un objeto, a partir de una clase o un prototipo. El resultado es un objeto válido y listo para usar.

Inicialización

El proceso de asignar colaboradores internos, y en general, dejar a un objeto en un estado válido. Es parte de la instanciación, pero también puede ejecutarse por separado. Los objetos que no poseen colaboradores internos no necesitan este proceso. Como todo en POO, es un envío de mensaje a un objeto, donde el mensaje tiene el único propósito de servir como inicializador, y en algunos lenguajes hasta lleva siempre el mismo nombre. La inicialización puede darse con objetos especificados al momento de realizarla (como colaboradores externos), o bien con objetos por defecto que se deciden en el método que responde al mensaje de inicialización.

Objeto válido

Técnicamente, un objeto que está referenciado a todos los colaboradores que necesita (que también deben ser objetos válidos). Conceptualmente, es cuando un objeto está siendo fiel a lo que representa en la realidad (en la realidad, todas las cosas son "válidas"). Es importante notar esto ya que muchas bibliotecas y/o herramientas de programación permiten tener objetos inválidos lo cual hace que éstos no se correspondan con la realidad (al menos en ese momento en que están en ese estado inválido).

Cohesión

Característica que tienen los objetos *bien* diseñados (es decir, que su gap semántico con la realidad es chico) de realizar sólo sus responsabilidades y no más. Si un objeto toma más responsabilidades de las que debe, pueden estar pasando dos cosas: (1) le estamos "sacando" responsabilidad al objeto correcto en donde debería ir esa responsabilidad o (2) no identificamos al objeto que debería encargarse de la responsabilidad, entonces ante la ausencia, "alguien más lo debe hacer".

Acoplamiento

Característica que tienen los objetos *mal* diseñados de depender demasiado de otro objeto. Depender demasiado sería tener varias interacciones donde se podría tener solo una, o asumir ciertos detalles de cómo el otro objeto tiene implementado un método. La forma de darnos cuenta si tenemos acoplamiento entre objetos, es si al realizar un cambio en la implementación de un objeto, y no en su interfaz, igualmente los objetos relacionados a él dejan de funcionar.

Objeto literal

Son aquellos objetos que no se construyen como el envío de mensaje a un objeto, sino que tienen una sintaxis particular. Todos los lenguajes tienen objetos literales, si no sería muy difícil y engorrosa la lectura del programa. Ejemplo: en Smalltalk, para escribir el número `42`, simplemente escribimos el numero `42`, no enviamos un mensaje para obtener dicho objeto (`Numero cuarentaYDos` podría ser un ejemplo de colaboración, que podemos apreciar que claramente, "no escala" para representar todos los números). Otro ejemplo son los strings, que se declaran entre comillas simples: 'hola'. Este objeto es complejo, está compuesto de 4 caracteres, es evidente que construirlo de manera "paradigmática" (envío de mensajes) sería algo del estilo: `String con: $h y: $o y: $l y: $a`. Los caracteres se denotan con `$` y también son literales! Si intentamos evitar los literales de los caracteres, la expresión final nos quedaría algo como: `String con: Caracter h y: Caracter o y: Caracter l y: Caracter a`.

Encapsulamiento

La cualidad más importante que tienen los objetos es que ocultan su implementación (es decir, cómo realizan sus tareas, responden a sus mensajes), y sólo dejan ver su protocolo (mensajes que sabe responder). Esto es fundamental porque nos abstrae de detalles, y divide quién es responsable de cada cosa, de manera que un objeto no pueda conocer o tomar responsabilidades de otro. Este ocultamiento es denominado encapsulamiento. El objetivo final es que si realizamos cambios únicamente en la implementación y no en el protocolo de un objeto, todos aquellos objetos que interactúan con él no se ven afectados por ese cambio, ni siquiera saben que ese objeto cambió. Es el objeto "puertas hacia adentro".

Polimorfismo

El polimorfismo (quizás una palabra confusa por su significado literal, y porque es ambigua ya que se usa para varios conceptos dentro de la programación) en POO se refiere a tener un mensaje o conjunto de mensajes que se comparte entre un grupo de objetos. Compartir quiere decir que todos los objetos saben responder a ese conjunto de mensajes, pero cada uno decide cómo

implementarlo, es decir que los métodos pueden ser distintos. Para que haya verdaderamente polimorfismo, no alcanza con que los objetos entiendan los mismos mensajes, sino que cuando reciben colaboradores externos, estos son del mismo tipo, y el objeto resultado también es del mismo tipo en las diferentes implementaciones.

Delegación

Es el acto de confiar una responsabilidad, parcial o total en otro objeto. La delegación puede darse de dos maneras: explícitamente a un objeto colaborador, o a una superclase en una jerarquía/prototipo en una cadena de prototipos.