Proceso de abstracción. Encapsulamiento y ocultamiento de información. Modularidad

Sabemos que la programación orientada a objetos tiene cuatro pilares fundamentales que iremos analizando:

1. Abstracción
2. Encapsulamiento
3. Herencia
4. Polimorfismo

Veamos algo más de abstracción y encapsulamiento.

Abstracción y proceso de abstracción

Las abstracciones son los modelos que obtenemos a partir de la realidad que nos rodea. El proceso de abstracción es la serie de acciones que tomamos para modelar la realidad e indicarle al programa cuáles serán las clases (y, por lo tanto, los objetos) con las que va a trabajar. Es por eso que también a este proceso se lo llama simplemente "modelar".

Lo que queremos obtener mediante el proceso de abstracción es "un modelo" de la realidad. El proceso de abstracción incluye no solo pensar los modelos a utilizar, sino también cómo van a interactuar entre sí; debemos considerar, además, el contexto en el cual nuestros modelos van a funcionar. El proceso de abstracción debe responder a preguntas tales como:

- ¿Qué modelos necesito? ¿Qué clases debo definir?
- ¿Cómo interactuarán los objetos entre sí? ¿Qué interfaz pública tendrá cada uno?
- ¿Cuál es el problema a resolver? ¿En qué contexto funcionarán mis objetos?

Si bien hay más preguntas a responder, estas tres son fundamentales. En particular, la del contexto resulta ser una de las más importantes, dado que afectará la forma en la que pensemos los modelos. Supongamos un ejemplo: modelemos, para un sistema, una persona. ¿Qué problema debe resolver el sistema? Pensemos por un segundo que el sistema es una simulación interactiva para alumnos de medicina: la persona, entonces, tendrá muchos atributos: edad, peso, género, tendrá órganos, sistemas, síntomas, niveles de azúcar en sangre, niveles de colesterol, historial de enfermedades, etc. El nivel de detalle

puede ser muy alto. El problema a resolver aquí es que, de acuerdo con las características de la persona y sus síntomas, los alumnos puedan deducir un diagnóstico.

Ahora, supongamos que queremos hacer un listado de personas para liquidación de sueldos. Nuestra abstracción de una persona se puede quedar en algo tan sencillo como el nombre, la edad, antigüedad en la empresa, sueldo, beneficios e identificación. No importa la altura, el género, el peso, etc. Si quiero mostrar un listado de personas, realmente no necesito todos esos datos dentro del modelo de lo que es una persona. Es decir, el modelado depende estrictamente del problema a resolver, del contexto. Esto es lo que se llama comúnmente “dominio” del problema o “negocio”.

Encapsulamiento

Las abstracciones definen los modelos, es decir, las clases que vamos a utilizar. Ya vimos que las clases definen, además de los atributos, los comportamientos. El comportamiento indica cómo interactuarán los objetos entre sí y sabemos que el conjunto de comportamientos se llama “interfaz pública”. Este nombre no es casualidad, dado que hay comportamientos que pueden tener los objetos que no necesariamente estén expuestos hacia otros objetos. El proceso de abstracción debe incluir, entre las preguntas que mencionamos, otra muy importante: **¿cuáles operaciones deben permanecer expuestas y cuáles no?** Es aquí donde entran en juego los conceptos de **encapsulamiento y ocultamiento de información**. Entre los autores y estudiosos del paradigma de orientación a objetos hay cierta controversia al momento de definir uno u otro término y cuál es el límite de la relación entre un concepto u otro.

Particularmente, y por cuestiones de simplicidad, resulta interesante la mirada de Rebecca Wirfs-Brock al respecto. Rebecca Wirfs-Brock es una renombrada ingeniera de software estadounidense que dedicó muchos años al estudio del diseño del software. Rebecca define al encapsulamiento como el simple hecho de poner en una misma entidad, o “cápsula”, los datos y las operaciones posibles sobre esos datos. Encapsular es simplemente “agrupar”, en un mismo contenedor, datos y operaciones relacionadas a esos datos.

Si miramos un lenguaje estructurado, por ejemplo, tendremos datos por un lado y operaciones por otro. Es decir, un lenguaje estructurado opera con funciones o rutinas sobre estructuras de datos. Las rutinas están en un lugar de la memoria y los datos están en otra. Es decir, las rutinas se valen de las estructuras de datos para funcionar.

En un lenguaje orientado a objetos, los datos van de la mano con las operaciones. Es como si las estructuras de datos y las rutinas estuvieran juntas. Esto es algo muy conceptual, dado que en el hardware todo estará separado, en la memoria siempre habrá operaciones por un lado y datos por otro, pero el paradigma orientado a objetos lo concibe de esa manera, dado que intenta siempre modelar la realidad. En cierto punto, esto es muy conveniente. Nosotros vemos a las cosas como entidades que tienen atributos (con valores cuantitativos) y realizan operaciones (hacen cosas). Para nosotros, las personas, el perro simpático del vecino es un "objeto perro" que se llama "Bobby", de color café que ladra y mueve la cola. No veremos nunca al perro del vecino como una estructura de datos consistente en una cadena de caracteres para su nombre, un número representando un color, que lleva colgado las instrucciones para que mueva la cola o ladre. No. Para nosotros, un perro es toda una entidad, que tiene y hace lo que hacen los perros y lleva sus datos y operaciones dentro de esa entidad.

En Java, encapsulamos mediante la Clase o "class". Otros lenguajes pueden tener otros medios de encapsulamiento. La clase en Java es nuestra mínima expresión de encapsulamiento. Pero podemos encapsular en entidades más grandes: podemos tener encapsulada funcionalidad detrás de un subsistema, conformado por una multitud de clases. O podríamos tener encapsulada funcionalidad detrás de un sistema, conformado por subsistemas, conformado por clases…

Ocultamiento de información

Ocultar información no se refiere a simplemente ocultarla de los demás objetos. La idea detrás del ocultamiento de la información es proveer un mecanismo controlado para acceder a los valores de los atributos de los objetos. Nuevamente, controlar el acceso no es una cuestión pura y exclusivamente de seguridad. Controlada se refiere a que antes de obtener un valor o antes de cambiar un valor podamos ejecutar una acción, realizar un cálculo, un chequeo adicional, etc. Durante el proceso de abstracción deberemos decidir cómo se va a acceder a los datos y/o cómo se los va a modificar: ¿debemos permitir acceder a un dato libremente o que sea accesible solo a través de una operación? Tomemos un ejemplo: así, de realizar algún cambio sobre el dato (por ejemplo, una fecha, pasa de ser un número entero a ser un dato de tipo fecha) aquellos que accedan al dato mediante la operación que devuelve un texto con la fecha, nunca se verán afectados.

Podemos decir que el ocultamiento de la información es el mecanismo mediante el cual obtenemos encapsulamiento.

Muchas veces se confunden los conceptos, pero no todo lo que está encapsulado estará oculto. Por ejemplo, supongamos un objeto “Representante de ventas” que interactúa con un objeto “Persona”. El vendedor interactuará con la persona e indagará su edad, por ejemplo, ejecutando su método “decirEdad()”. Al objeto vendedor no le incumbe si el objeto persona guarda un contador de años o si guarda la fecha de nacimiento y se la resta la fecha actual. Cómo llega al valor de la edad no es relevante, lo que importa es el valor, el resultado de la operación “decirEdad()”. Es así como la edad en el objeto persona no está oculta, pero sí encapsulada. Podemos saber cuál es la edad de una persona, solo que no sabremos cómo la calcula.

David Parnas, un reconocidísimo pionero de la ingeniería de software, doctorado en la Universidad de Carnegie Mellon, fue quien desarrolló el concepto de ocultamiento de la información. Según David Parnas, aquello que quede expuesto debe revelar lo menos posible acerca del funcionamiento interno de un objeto.

Modularidad

El mismo Parnas fue un acérrimo defensor de los módulos en el software. Como ingeniero, fue uno de los primeros en aplicar conceptos de ingeniería al diseño de software. La idea detrás de la modularidad en el software es que gracias al correcto uso del encapsulamiento y ocultamiento de la información podemos lograr dividir al software en módulos intercambiables entre sí. Parnas encontraba una gran ventaja en lograr software cuyas partes estuvieran lo suficientemente autocontenidas como para que el recambio de una no afectara (o afectara lo menos posible) a otras partes. Como ingeniero, pretendía que las partes del software fueran reemplazables como los repuestos de un motor.

La modularidad pretende justamente eso, lograr dividir al software de forma tal que podamos intercambiar partes sin afectar a otras. La modularidad no solo debe entenderse como el armado de un sistema en módulos: nuevamente, un módulo puede ser un sistema entero en sí mismo, un módulo puede ser un subsistema o más aún, un módulo puede ser una simple clase. El diseño modular implica tener una baja interrelación entre los módulos, lo que implica que los módulos estén lo más autocontenidos que se pueda. Tener en cuenta esto nos introducirá a los conceptos de cohesión y acoplamiento.

¿Crees que hay beneficios en ocultar la información? ¿Crees que facilitaría en algo tener el estado y el comportamiento de un objeto público? ¿Cuáles serían las desventajas?