Diseño orientado a los objetos

El Diseño Orientado a los Objetos (DOO) crea una representación del problema del mundo real y la hace corresponder con el ámbito de la solución, que es el software.

A diferencia con otros métodos de diseño, el DOO produce un diseño que interconecta objetos de datos y operaciones de procesamiento para esos objetos, de forma que se modulariza la información y el procesamiento, en lugar de aislar el procesamiento.

Todos los métodos de diseño intentan desarrollar software basándose en:

- Abstracción
- Ocultamiento de información
- Modularidad

El DOO proporciona un mecanismo que permite al diseñador consigue estas tres características sin dificultad.

El Análisis Orientado a Objetos, el Diseño Orientado a Objetos y la Programación Orientada a Objetos comprenden un conjunto de actividades de la Ingeniería del Software para la construcción de un sistema basado en objetos.

1. CONCEPTOS DEL DISEÑO ORIENTADO A OBJETOS

Al igual que todos los métodos de diseño utilizan su propia notación y metodología, el DOO introduce un conjunto nuevo de términos, notaciones y procedimientos para la derivación del diseño del software.

A continuación resumimos la terminología orientada a objetos e introducimos algunos conceptos propios de esta forma de diseño-

1.1 Objetos, operaciones y mensajes

El funcionamiento del software se consigue al actuar uno o más procesos sobre una estructura de datos de acuerdo con un procedimiento de invocación.

Para conseguir un DOO, tenemos que establecer un mecanismo para:

- Representar la estructura de datos
- Especificar el proceso
- Realizar el procedimiento de invocación

Objeto: Componente del mundo real que se hace corresponder con el software. En un Sistema de Información basado en Computador, un objeto es un producto o consumidor de información, o un elemento de información.

Cuando se hace corresponder un objeto con su realización software, implementamos una estructura de datos y usa serie de procesos que pueden transformar la estructura de datos.

Operaciones, métodos o servicios: Procesos a los que se le permite transformar estructuras de datos.

Mensajes: Peticiones que se realizan a los objetos para que realicen alguna de sus operaciones. Las operaciones contienen construcciones procedimentales y de control, que se invocan mediante un mensaje.

Al definir un objeto con parte privada y proporcionar mensajes para invocar al procedimiento adecuado conseguimos el **ocultamiento de información**. De esta forma dejamos ocultos al resto de los elementos de programa los detalles de implementación del objeto.

Los objetos con sus operaciones proporcionan una **modularidad inherente**, es decir los elementos del software (datos y procesos) están agrupados con un mecanismo de interfaz bien definido, que son **los mensajes**.

1.2. Aspectos de diseño

Meyer sugiere cinco criterios para evaluar la calidad de un método de diseño a partir de la modularidad.

Descomponibilidad: Facilidad con la que un método de diseño ayuda al diseñador a descomponer un problema en subproblemas más sencillos.

Componibilidad: Grado en el que un método de diseño permite la reutilizabilidad de módulos.

Compresibilidad: Facilidad para comprender un componente del programa sin tener que hacer referencia a otros módulos.

Continuidad: Capacidad de realizar cambios en un programa y que esos cambios afecten a un número mínimo de módulos.

Protección: Característica arquitectónica que reduce la propagación de errores.

A partir de estos criterios, Meyer sugiere la derivación de cinco principios de diseño para arquitecturas modulares:

- Unidades modulares
- Pocas interfaces
- Interfaces pequeñas (acoplamiento débil)
- Interfaces explícitas
- Ocultamiento de información

Para conseguir un acoplamiento débil, se debe minimizar el número de interfaces entre módulos y minimizar la cantidad de información que se mueve a través de una interfaz.

Siempre que los módulos tengan que comunicarse tiene que hacerlo de forma clara, mediante interfaces explícitas, y no mediante una zona global de datos, ya que la comunicación entre módulos no sería fácilmente comprensible para un observador externo.

El principio de ocultamiento de información se consigue cuando toda la información del módulo está oculta para el acceso desde el exterior, a menos que la información se defina explícitamente de forma pública.

Si bien estos principios son aplicables a cualquier tipo de diseño, el método de Diseño Orientado a Objetos consigue alcanzar estos principios de forma más eficiente que el resto de los enfoques, consiguiendo arquitecturas más modulares.

1.3. Clases, instancias y herencia

Muchos objetos del mundo real tienen características prácticamente iguales y realizan operaciones prácticamente similares.

Por ejemplo, si observamos un conjunto de vehículos, vemos que existen motos, automóviles y camiones. Aunque todos estos objetos son diferentes, todos pertenecen a una **clase** superior, llamada *vehículo*.

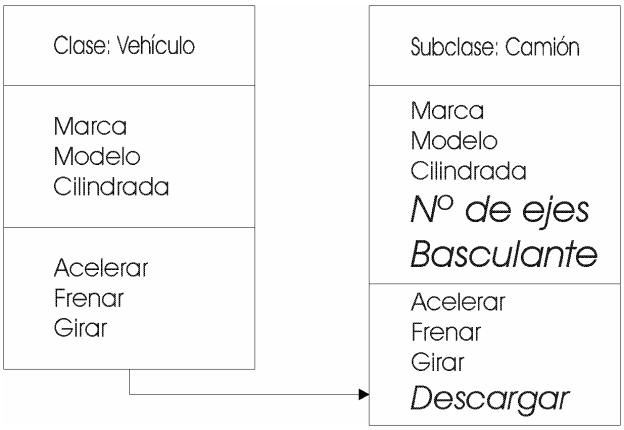
Todos los objetos de la clase *vehículo* tienen atributos comunes (marca, modelo, ...) y realizan una serie operaciones comunes (acelerar, frenar, ...)

Las implementaciones en software de objetos del mundo real se hacen de forma muy similar. Todos los objetos son miembros de una clase mayor y **heredan** la estructura de datos privada y las operaciones que se hayan definido para esa clase.

Es decir una **clase** es un conjunto de objetos que tienen las mismas características. Así, un objeto es una **instancia** de una clase mayor.

Pero, ¿qué ocurre cuando una instancia de una clase mayor contiene una serie de atributos y/o operaciones que son propias de la instancia?

La solución a este problema es la creación de **subclases**, que heredaría los atributos y las operaciones de la clase superior, y permite modificarlos para ajustarlos a sus necesidades.



El uso de clases, subclases y herencia tiene mucha importancia en la Ingeniería del Software moderna. La reutilización de componentes del software (que nos permite conseguir la componibilidad) se lleva a cabo creando objetos (instancias) que se forman sobre los atributos y las operaciones de una clase o una subclase. Sólo hay que especificar cuáles son las diferencias entre el objeto nuevo y la clase, en vez de tener que volver a definir todas las características del objeto.

1.4. Descripciones de los objetos

La descripción del diseño de un objeto (una instancia de una clase o de una subclase) está compuesta de dos partes:

Descripción de la interfaz: Establece la interfaz del objeto, definiendo los mensajes que puede recibir el objeto y las operaciones que puede realizar cuando el objeto recibe el mensaje.

Descripción de la implementación: Muestra los detalles de cada una de las operaciones implicadas como consecuencia de la recepción de un mensaje. Los detalles de implementación incluyen información sobre la parte privada del objeto, es decir, los detalles de las estructuras de datos y los detalles procedimentales que describen las operaciones.

La descripción de la interfaz son un conjunto de mensajes con sus comentarios correspondientes

La descripción de la implementación debe contener la información suficiente para que se puedan manejar de forma adecuada todos los mensajes recibidos en la descripción de la interfaz.

A la diferencia entre la forma de especificar qué es lo que se desea y cómo se proporciona ese servicio es lo que se conoce como **encapsulación**.

Por tanto, el paradigma orientado a objetos persigue la encapsulación, es decir, mostrar sólo las operaciones visibles, y ocultar los detalles de implementación, creando una sensación de cajas negras a las que se envían mensajes, pero no se conocen los detalles de implementación de la gestión de esos mensajes.

2. METODOS DE DISEÑO ORIENTADOS A OBJETOS

Antes de continuar debemos comprender la diferencia entre el Análisis Orientado a Objetos, que es una actividad de clasificación, y el Diseño Orientado a Objetos, que define los objetos que se derivan de cada clase, se tiene que indicar las relaciones que existen entre ellos mediante una notación.

El DOO tiene que comenzar con una descripción en lenguaje natural de la estrategia de solución, y a partir de esta descripción, el diseñador identifica los objetos y operaciones.

2.1. Pasos del Diseño Orientado a Objetos

1) Definición del problema
2) Desarrollo informal de la forma de procesamiento para la realización del software
3) Formalización de la forma de procesamiento
   a) Identificar los objetos y sus atributos
   b) Identificar las operaciones de los objetos
   c) Establecer las interfaces que muestren las relaciones entre los objetos y las operaciones
   d) Crear un diseño detallado que proporcione una descripción de la implementación de los objetos

3. DEFINICIÓN DE CLASES Y DE OBJETOS

La aplicación de los principios y métodos de análisis de requisitos permite al analista y la diseñador llevar a cabo dos subpasos necesarios.

a) Descripción del problema
b) Análisis y aclaración de las limitaciones conocidas

La realización del software del problema del mundo real debe describirse de forma sencilla y correcta para que **permita a los ingenieros del software que trabajan en el proyecto comprender el problema de forma sencilla y uniforme**.

El cometido del AOO es el de aislar todos los nombres y frases del texto explicativo del procesamiento que describe que es lo que ha de realizar el sistema.

Esta primera selección nos puede ayudar a definir las clases, subclases y objetos del sistema.

a) Un nombre común suele representar una clase de objetos (una abstracción de datos), como puede ser *automóvil*.
b) Un nombre propio suele representar una instancia de una clase, como puede ser *Seat Ibiza*.

4. PROCESO DEL ANÁLISIS Y DISEÑO ORIENTADO A LOS OBJETOS

El resultado del diseño orientado a objetos es una jerarquía de clases.

Los elementos iniciales de un DOO son los propios objetos, y posteriormente, a medida que se van identificando aspectos comunes, los objetos se van agrupando en clases, que a su vez serán subclases de clases más abstractas.

Los métodos estructurados y sus correspondientes notaciones definen un sistema como un conjunto secuencial de módulos interdependientes que comparten datos. En cambio, los métodos orientados a objetos definen un conjunto de módulos independientes relacionados y con visibilidad limitada entre ellos.

4.1. Pasos del DOO

Esencialmente, el DOO consta de cuatro pasos fundamentales:

a) Identificación y definición de objetos y clases
b) Organización de relaciones entre clases
c) Extracción de estructuras en una jerarquía de clases
d) Construcción de bibliotecas de clases reutilizables

Como en todas las fases de Ingeniería del Software, el DOO es cíclico, es decir, los diseñadores comienzan definiendo un conjunto de clases, que se amplían, modifican, ..., y vuelta a empezar.

4.2. Identificación y definición de objetos

El principal problema del desarrollo de un sistema orientado a objetos es encontrar los objetos en la fase de AOO y DOO.

El método que utilizaremos para la identificación de objetos es el propuesto por Booch en 1983, que dio origen al **método gramatical**.

Esta metodología sugiere que se comience con una descripción textual del sistema deseado y que el diseñador vea:

- A los nombres como posibles identificadores de las clases de los objetos
- A los verbos como posibles métodos

La lista resultante de clases (nombres) y métodos (verbos) se utilizará para comenzar el proceso de diseño.

4.3. Ejemplo para un procesador de textos sencillo

La metodología de Booch comienza por una definición del problema y una descripción de la solución, como se indica a continuación:

4.3.1. Definición del problema

Desarrollo de un sistema sencillo de procesamiento de texto

4.3.2. Descripción de la solución

El sistema de procesamiento de texto permite a los usuarios crear documentos. Los documentos creados se pueden archivar en un directorio. Los usuarios pueden imprimir o mostrar sus documentos. Se pueden modificar los documentos. También se pueden borrar del directorio.

4.3.3. Identificación de los posibles objetos

El paso siguiente consiste en identificar los posibles objetos subrayando los sustantivos (y frases sustantivadas), tal y como se muestra a continuación.

El sistema de procesamiento de texto permite a los usuarios crear documentos. Los documentos creados se pueden archivar en un directorio. Los usuarios pueden imprimir o mostrar sus documentos. Se pueden modificar los documentos. También se pueden borrar del directorio.

Documento y **directorio** parecen ser conceptos importantes, y por tanto, objetos.

Hay que tener en cuenta que no todos los nombres que aparecen en la descripción de la solución terminan siendo objetos, por lo que este método de Booch, puede generar una serie de conceptos que no pertenecen al sistema a modelar, y que por tanto no tienen que incorporarse al software.

En la descripción anterior, el sustantivo **usuario**, es un claro ejemplo de un sustantivo que no se convierte en objeto, ya que no pertenece al sistema que queremos desarrollar.

4.3.4. Asociación de atributos a los objetos

Una vez identificados los objetos, se detallan los atributos de cada uno de los objetos, como se indica a continuación:

Objeto	Atributos
Documento	Se puede crear Se pueden archivar Se pueden imprimir Se pueden mostrar Se pueden modificar Se pueden borrar
Directorio	Contiene uno o más documentos Se pueden archivar documentos Se pueden borrar documentos

4.3.5. Identificación de los posibles métodos

A continuación se trata de identificar los posibles métodos, subrayando los verbos de la forma siguiente:

El sistema de procesamiento de texto permite a los usuarios crear documentos. Los documentos creados se pueden archivar en un directorio. Los usuarios pueden imprimir o mostrar sus documentos. Se pueden modificar los documentos. También se pueden borrar del directorio.

Hay que tener en cuenta que no todos los verbos que aparecen en la descripción de la solución terminan siendo métodos.

En la descripción anterior, la acción verbal **permite**, es un claro ejemplo de un verbo que no se convierte en método, ya que no se refiere a acciones que se realice o sufran los objetos del sistema, sino que es un *adorno* lingüístico.

4.3.6. Asociación de los métodos a los objetos

A continuación se recogen estos métodos con sus objetos correspondientes.

Objeto	Métodos
Documento	Crear Archivar Imprimir Mostrar Modificar Borrar
Directorio	Archivar Borrar

4.3.7. Definición de las interfaces entre objetos

El último paso del proceso de identificación y definición de los objetos define las interfaces entre los objetos definidos, mediante una descripción escrita como la siguiente.

El sistema se realiza con dos clases: *documento* y *directorio*.

La clase *documento* contiene una variable instancia llamada *documentId* y los métodos siguientes: *crear*, *archivar*, *imprimir*, *mostrar*, *modificar* y *borrar*.

La clase *directorio* contiene *directorioId* y los métodos *archivar* y *borrar*.

4.3.8. Conclusiones sobre el método de Booch

Con el método de Booch para encontrar clases es difícil conseguir un resultado de alta calidad, pues la abstracción de clases que consigamos, depende de una estructuración inteligente de la descripción del problema en elementos independientes e intuitivamente correctos.

4.3.9. Directrices para la identificación y definición de clases y métodos

Las metodologías para apoyar el AOO y el DOO se encuentran en sus primeras fases de desarrollo, pero se han definido una serie de directrices que facilitan la identificación y definición de clases y métodos.

- Modelar con clases las entidades que ocurren de forma natural en el problema
- Diseñar métodos con una sola finalidad
- Diseñar un método nuevo antes de ampliar uno existente
- Evitar métodos extensos
- Guardar como instancia los datos necesitados por más de un método o por una subclase
- El diseñador debe trabajar para obtener una biblioteca de clases, y no para él mismo, ni para desarrollar el sistema actual.

Además, para evitar la creación de clases innecesarias o declaración de clases que no lo sean, una clase debe ofrecer una serie de servicios a objetos de un tipo determinado.

Una clase se debería crear cuando:

- La nueva clase represente una abstracción significativa del problema
- Es probable que los servicios que proporciona sean utilizados por otras clases.
- Su comportamiento sea complejo.
- Si se representara como un método de otra clase, pocos usuarios de ésta lo invocarían.

4.4. Definición y organización de clases

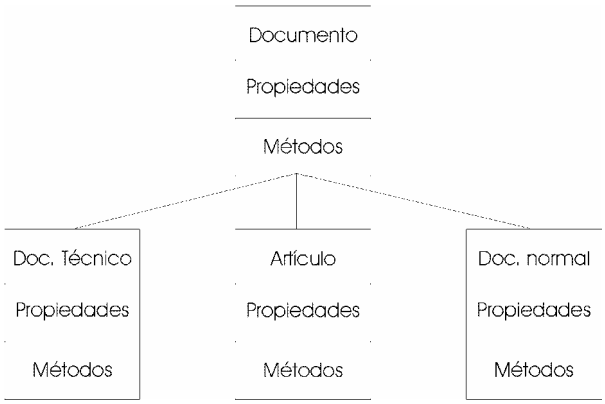
La identificación y definición de objetos es sólo el primer paso en el diseño de un Sistema OO.

La abstracción es la tarea continua de un diseñador OO.

Una vez definidos los objetos, el paso siguiente consiste en observar características comunes para crear abstracciones a nivel de clase, pero no existe ninguna metodología formal para la realización de estas abstracciones.

La definición de una clase para generar múltiples instancias de un objeto ofrece la primera visión del poder de la abstracción.

Si por ejemplo, el sistema de procesamiento de textos se ampliase para convertirlo en un sistema ofimático que utilice documentos de diferentes tipos como pueden ser *documentos técnicos*, *artículos* o *documentos de texto normales*, la naturaleza de *documento* cambiará.

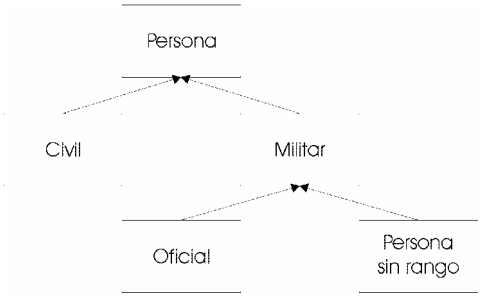


Cada una de estas posibilidades de tratamiento de texto posee documentos con características propias y métodos propios (para poder dar formato a características propias de cada documento), por lo que *documento* se convierte en una superclase, incluyendo propiedades y métodos comunes a todos los tipos de documentos. Las subclases especiales de *documento* incluyen métodos adicionales y propiedades para atender a las necesidades locales.

Por tanto, el objetivo final de un DOO es guardar propiedades y métodos de clases en el nivel de abstracción más alto posible, de forma que los compartan la mayoría de clases y se fomente la reutilización.

4.4.1. Diagramas de generalización o de herencia

Expresan la organización de las clases de un sistema OO.



Las flechas indican generalización o herencia.