

Introducción al Lenguaje Unificado de Modelado (UML)

En todo proceso de desarrollo de sistemas, es una buena práctica analizar el problema en cuestión y diseñar la solución a modo de mapa o guía para la implementación de la misma. El diseño se considera además una herramienta de comunicación, por lo que debe estar expresado en una manera sencilla, clara y conocida por todos.

El Lenguaje Unificado de Modelado (UML) es una forma estandarizada de plasmar el diseño, brindando una variedad de diagramas para visualizar dicho diseño desde distintas perspectivas y niveles de abstracción. Fue creado por James Rumbaugh, Ivar Jacobson y Grady Booch a mediados de la década del los '90 y actualizado en 2005 con su versión 2.0.

UML cuenta con dos tipos de diagramas:

- *estructurales*, que muestran la estructura estática del sistema
- *de comportamiento*, que muestran los comportamientos dinámicos de los objetos del sistema

Diagrama de Clases

El *diagrama de clases* es uno de los diagramas UML más utilizados para documentar la estructura estática de un sistema. Está integrado por las clases que componen el sistema con sus atributos y operaciones (métodos), y las relaciones que existen entre ellas.

Clase	
Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Gráficamente se representan como rectángulos, con tres secciones: en la primera se especifica el nombre de la clase, en el segundo el listado de sus atributos y en el tercero el listado de sus operaciones.	<div>El nombre de una clase debe ser sencillo, compuesto por una palabra o varias palabras, donde solo la inicial de cada una de ellas se escribe con mayúsculas.</div> <div>Si la clase es abstracta, su nombre se escribe en cursiva, o se agrega el estereotipo <<abstract>> debajo de su nombre.</div> <div>Ejemplo: CuentaBancaria</div>
Atributo → mod_visib nombre_atrib:tipo_atrib = valor_por_defecto	
Un atributo es una propiedad de una clase, que puede tomar un valor determinado por la instancia, es decir, el objeto que se creará a partir de la clase. Por ejemplo, toda cuenta bancaria tiene un titular, un saldo y un interés mensual previamente definido (el mismo para todas las cuentas).	<div>El nombre de un atributo es una o varias palabras, donde cada palabra comienza con mayúscula excepto la primera. De un atributo también se puede especificar su tipo.</div> <div>Ejemplo: titularCuenta: String</div>
Operación → mod_visib nombre_op(nombre_param:tipo_param, ..): tipo_ret	
Una operación es un servicio que se le puede requerir a un objeto de una clase; representa un comportamiento de la misma. Cabe destacar que una operación puede modificar el estado interno	El nombre de las operaciones suele estar compuesto por un verbo o una frase que contiene un verbo, que represente el comportamiento asociado. La inicial de cada palabra se escribe

del objeto, aunque no siempre ocurre. Una operación puede recibir parámetros, en cuyo caso se especificará el nombre de cada uno y su tipo de dato. Si la operación retorna un valor, se deberá también especificar su tipo de dato.	con mayúscula, excepto en la primera, que se escribe toda en minúscula. Si la operación es abstracta, se escribe en cursiva. Ejemplo: <code>mostrarSaldo()</code> <code>realizarExtraccion(monto: double): double</code>
---	--

Visibilidad

La visibilidad define cómo un elemento es visto y, en consecuencia, utilizado por otros. En un diagrama de clases, es aplicable tanto a atributos como a métodos definidos en una clase.

En UML pueden definirse tres niveles de visibilidad:

- pública (+): quien pueda ver a la clase, puede ver también al elemento (atributo/método)
- protegida (#): sólo pueden ver el elemento indicado la propia clase donde está definido y sus descendientes (herencia)
- privada (-): sólo la clase en donde está definido el elemento puede verlo.

En el POO, los atributos suelen estar definidos como privados o protegidos; rara vez como públicos. Por su parte, los métodos que servirán para interactuar con otras clases deberán ser definidos como públicos; del resto no existen restricciones.

Alcance

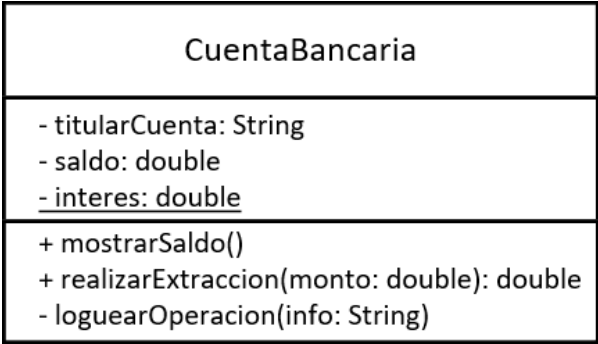
El alcance de un elemento indica si éste aparece en cada instancia de una clase, o es único para todas las instancias de dicha clase. Es aplicable tanto a atributos como a métodos.

Existen dos tipos de alcance:

- a nivel de instancia: cada instancia de la clase contiene su propio valor para el elemento.
 - para un atributo: cada instancia le puede asignar su propio valor
 - para un método: su invocación puede cambiar el estado interno de la instancia
- a nivel de clase: existe un único valor del elemento para todas las instancias de la clase.
 - para un atributo: su valor será el mismo para todas las instancias de la clase
 - para un método: su invocación no afecta a la instancia


Para indicar en el diagrama que un elemento es de clase, su definición aparece subrayada.


En resumen, la clase `CuentaBancaria` ser representaría de la siguiente manera:

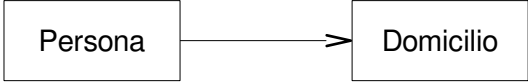


Relaciones

Una relación es una conexión lógica que se establece entre dos clases. Existen distintos tipos a fin de representar las distintas formas en las que las clases colaboran entre sí.

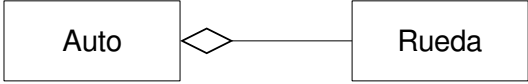
Dependencia	"usa un"	
<p>Una dependencia entre dos clases establece que una clase necesita conocer a la otra para usar objetos de esa clase. Una dependencia solo sugiere que objetos de dos clases "pueden" colaborar; por esto se considera como la relación más débil que puede existir entre dos clases. En este tipo de relación un cambio en un elemento puede afectar a otro que lo usa, pero no necesariamente al revés.</p> <p>La dependencia se representa con una línea punteada conectando las dos clases que la integran.</p>		


Generalización	"es un"	
<p>Una generalización conecta clases generalizadas a otras más específicas, en lo que se conoce como relaciones superclase/subclase o padre/hijo. En otras palabras, es una relación que se establece entre dos conceptos: uno más general (superclase o padre) y uno más específico (subclase o hijo).</p> <p>En este tipo de relación, la subclase hereda tanto atributos como operaciones definidos en su superclase, y agrega los propios. Si una operación en la subclase tiene la misma firma que la definida en la superclase (mismo nombre y parámetros) se dice que la subclase "sobreescribe" la operación de la superclase.</p> <p>La herencia múltiple se da cuando una subclase hereda de dos o más superclases. Si bien el POO lo soporta, no todos los lenguajes de programación que lo implementan lo hacen.</p> <p>La generalización se representa con una línea llena terminada en una flecha con un triángulo vacío; la flecha siempre apunta a la superclase.</p>		

Asociación	"conoce a" "trabaja con"	
<p>Las asociaciones son relaciones estructurales entre instancias, donde una instancia de una clase está conectada con una instancia de la otra; en otras palabras, una instancia mantiene una</p>		

referencia a la otra.

La asociación se representa con una línea llena conectando las dos clases que la integran.

Agregación	“tiene un”	
<p>La relación de agregación es un tipo específico de asociación, que sirve para modelar conceptos del estilo "todo/parte", donde una clase representa el todo y otra las partes. En este caso, objetos del "todo" tendrán objetos de la "parte".</p> <p>Una agregación se representa con una línea llena, agregándose un rombo vacío en el extremo del "todo".</p>		

Composición	“tiene un”	
<p>La composición es una forma de agregación que establece un vínculo más fuerte entre el todo y las partes, dado que el todo es responsable de sus partes, existiendo además una coincidencia en los tiempos de vida de ambos. Si bien las partes pueden ser creadas luego del todo, una vez creadas vivirán y morirán con él. Como consecuencia, las instancias parte pueden estar relacionadas con sólo un objeto todo.</p> <p>La composición se representa de manera similar a la agregación, con la diferencia de que el rombo en el extremo del todo se rellena.</p>		

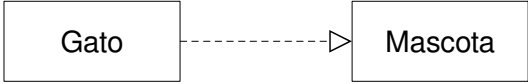
Realización	“se comporta como”	
<p>Una realización se da cuando una clase implementa o realiza una interfaz.</p> <p>En este tipo de relación, la clase se ve obligada a dar implementación a todas las operaciones abstractas definidas en la interfaz, sin excepción. En caso de no hacerlo, la clase deberá ser definida como abstracta y no podrá ser instanciada.</p> <p>La realización se representa con una línea punteada terminada en una flecha con un triángulo vacío; la flecha siempre apunta a la interfaz.</p>		

Diagrama de secuencia

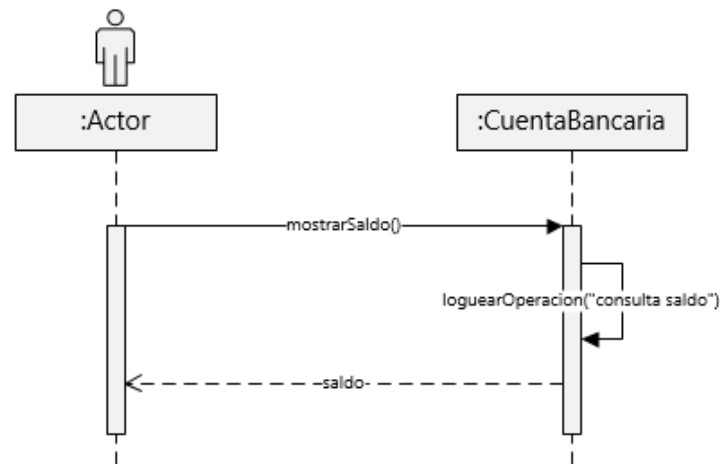
Como complemento al diagrama de clases, podemos utilizar el *diagrama de secuencia*. Este diagrama de comportamiento representa una interacción o colaboración entre objetos en un gráfico bidimensional, donde la dimensión vertical es la temporal, mientras que la horizontal representa los objetos participantes en dicha interacción.

Características

- Cada participante se muestra como una columna, que indica su línea de vida

- Durante el tiempo que existe el objeto, se lo representa con una línea discontinua
- Durante el tiempo que dura un comportamiento, la línea de vida se dibuja como una línea doble
- Los mensajes que se intercambian se representan con una flecha desde la línea de vida de un objeto al otro
- Los retornos se representan con una flecha con línea punteada
- Las flechas se organizan en orden cronológico hacia abajo

Para verlo en un ejemplo, consideremos la operación `mostrarSaldo()` de nuestra `CuentaBancaria`:



Aquí, cuando un actor invoca el método `mostrarSaldo()` sobre la `CuentaBancaria`, ésta ejecuta a su vez la operación `loguearOperacion` con la descripción de la misma como parámetro; una vez terminado el logueo, `mostrarSaldo()` retorna al actor el `saldo` solicitado.

Bibliografía

- *El Lenguaje Unificado de Modelado. Manual de Referencia*
James Rumbaugh, Ivar Jacobson y Grady Booch
Pearson - Addison Wesley
- *The Unified Modeling Language User Guide*
Grady Booch, James Rumbaugh y Ivar Jacobson
Addison Wesley
- *Learning UML 2.0*
Russ Miles y Kim Hamilton
O'Reilly