APÉNDICE I

Orientación a Objetos: Conceptos Básicos

La Descomposición Orientada a Objetos

En el proceso de diseño de un sistema complejo, es fundamental su descomposición en partes más pequeñas, donde cada una se pueda refinar de forma independiente. La descomposición orientada a objetos, consiste en ver al mundo como un conjunto de objetos que colaboran entre sí para llevar a cabo algún comportamiento de nivel superior. Cada objeto tiene su propio comportamiento, y modela un entidad del mundo real o abstracta. Desde esta perspectiva, cada objeto no es más que una entidad que muestra un comportamiento bien definido.

Este tipo de descomposición tiene una serie de ventajas altamente significativas sobre la descomposición algorítmica. La descomposición orientada a objetos produce sistemas más pequeños a través de la reutilización de mecanismos comunes, proporcionando así una importante economía de expresión. Los sistemas orientados a objetos son también más resistentes al cambio y están mejor preparados para evolucionar en el tiempo, porque su diseño está basado en formas intermedias estables.

El Paradigma Orientado a Objetos

La orientación a objetos es una nueva tecnología basada en objetos y clases. En la actualidad, representa uno de los mejores marcos metodológicos para la ingeniería de software, proporcionando las bases para una disciplina de ingeniería sistemática. Los preceptos del paradigma orientado a objetos mejoran el modelado y la implementación de sistemas, brindando soporte de primera clase a los objetos y las clases de los objetos de un dominio de aplicación. Los objetos proveen un foco canónico en el análisis, diseño, e implementación, enfatizando el estado, el comportamiento, y la interacción de ellos en los modelos, facilitando la propiedad deseable de "*libre de irregularidades"* entre las distintas actividades del proceso de desarrollo.

Esta metodología comprende una completa y comprensiva aproximación general, repleta de un paradigma de técnicas, métodos, procesos, normas, modelos, notaciones, herramientas, componentes, lenguajes, ambientes, ejemplos, práctica, y habilidades. Su dominio de aplicación es muy amplio, y va desde el modelado de dominios de aplicaciones del mundo real a la arquitectura, modelado e implementación de sistemas que corren debajo de estos dominios y los sistemas distribuidos que los conectan.

Visión General de los Objetos

Las aplicaciones orientadas a objetos están constituidas por un conjunto de **objetos**. Los objetos pueden representar abstracciones del mundo real o entidades abstractas. Generalmente, un objeto posee datos y operaciones que operan sobre esos datos. A dichas operaciones se las conoce comúnmente como **métodos**. La única manera de conocer los datos que conserva un objeto es mediante sus métodos. Un método contiene un código que especifica cual será el comportamiento del objeto que ejecute tal método. Un objeto ejecuta un método cuando recibe un **mensaje**, el cual es enviado por otro objeto. Dicho proceso se conoce como interacción entre objetos, donde el objeto que emite el mensaje cumple el rol de emisor y el objeto que recibe el mensaje cumple el rol de receptor. Por lo general, el código de un método determina ciertas acciones para el objeto que lo ejecuta, tales como cambios en sus datos, envío de mensajes a otros objetos, etc.

Cada mensaje que recibe un objeto provoca la ejecución de un determinado servicio por parte de dicho objeto. Tal servicio es la respuesta que brinda al objeto emisor, la cual está especificada en un método. Existe una relación "uno a uno" entre el conjunto de mensajes que es capaz de responder un objeto y su conjunto de métodos, así la cantidad de servicios que es capaz de brindar un objeto se ve acotado a la cantidad de métodos que es capaz de ejecutar. El conjunto de mensajes a los cuales es capaz de responder un objeto se denomina **protocolo.**

Conocimiento entre Objetos

Los objetos ofrecen, a través de sus mensajes, ciertos servicios que contribuyen a la satisfacción de algún requerimiento[1] bajo la responsabilidad de otro objeto. De este modo, un objeto cuenta con la colaboración de otros para cumplir con un determinado servicio. El objeto que colabora con algún servicio se conoce como **colaborador**, y ofrece tal servicio a través de un determinado mensaje al cual es capaz de responder. El objeto que requiere de una colaboración, debe tener la capacidad de comunicarse con el colaborador, por lo cual, es necesario que se establezca una relación de **conocimiento** entre ambos objetos. Generalmente, un objeto puede recurrir a la colaboración de otro mediante alguno de los siguientes tipos de conocimiento:

- **Conocimiento interno:** Se establece cuando un objeto posee una referencia a otro objeto, la cual puede conservar durante todo su tiempo de vida. En cualquier momento, el objeto que conserva la referencia puede utilizarla dentro del desarrollo de un requerimiento para comunicarse con el objeto referenciado y requerir cierta colaboración de su parte para que realice un servicio. Por ejemplo, en el ambiente de desarrollo *Smalltalk*, las variables de instancia son un medio que brindan soporte a este tipo de conocimiento.

- **Conocimiento temporal:** Para que un objeto pueda prescindir de la colaboración de otro objeto, no es necesario que conozca al colaborador durante todo su tiempo de vida. En ciertos casos, es suficiente que la relación de conocimiento entre un objeto y su colaborador persista en el instante que dura la satisfacción del requerimiento en el cual tiene lugar la colaboración. Se pueden distinguir dos tipos de conocimiento temporal:

  - **Conocimiento temporal interno:** La relación de conocimiento se establece explícitamente en algún punto del desarrollo del requerimiento. Para esto, se crea y conserva una referencia al colaborador, la cual tiene un tiempo de vida menor o igual al tiempo que dura el requerimiento. Una forma típica de lograr esta clase de conocimiento en *Smalltalk* es mediante la asignación de variables temporales dentro de los métodos.

  - **Conocimiento temporal externo:** En este tipo de conocimiento un objeto conoce a su colaborador dentro del instante que dura el desarrollo de un requerimiento. Sin embargo, la referencia hacia el colaborador está dada por un parámetro del mensaje que inicia el servicio; por lo cual, el conocimiento externo solo se establece cuando la ejecución de un requerimiento cuenta con uno o más parámetros.

- **Conocimiento indirecto:** Esta clase de conocimiento tiene lugar cuando se aplica un tipo especial de relación de dependencia[2]. En tales casos, un objeto independiente no puede comunicarse directamente con un objeto dependiente (carece de una referencia hacia él), pero existe un mecanismo mediante el cual el objeto independiente puede notificar indirectamente de ciertos cambios a los objetos dependientes interesados.

- **Conocimiento global:** Existen ciertos objetos que pueden ser referenciados en cualquier momento, mediante identificadores únicos asociados a los mismos. En *Smalltalk,* los literales (números, caracteres, símbolos, etc.) y algunas instancias particulares (*true*, *false*, *nil*, etc.) son ejemplos de esta clase de objetos. De esta forma, es posible referirse a dichos objetos

[1] Requerimiento o servicio, ambos términos se utilizan de manera intercambiable.
[2] La dependencia es una relación que se establece entre dos objetos, en la cual un cambio en uno de ellos (el objeto independiente) afectará al otro objeto (el objeto dependiente).

utilizando sus respectivos identificadores, sin la necesidad de crear y conservar una referencia a los mismos.

Moldes para los Objetos

La implementación de un objeto está definida por su **clase**. La clase especifica la representación de los datos internos de un objeto, y define los métodos que el mismo puede ejecutar.

Un objeto se crea instanciando una clase, dicho objeto se dice que es una instancia de la clase. El proceso de instanciación de una clase aloca espacio para los datos internos que conservará el objeto creado (en las **variables de instancia**) y asocia los métodos con esos datos. Muchas instancias similares pueden ser creadas instanciando una misma clase.

Estructura Interna de los Objetos

Los mensajes son el único medio para lograr que un objeto ejecute un método. Los métodos son el único medio para cambiar los datos internos de un objeto. El conjunto de datos internos determina el **estado** de un objeto. *Booch* [Booch/94] presenta una definición de bajo nivel de estado, en la cual establece que: «el estado de un objeto abarca todas las propiedades estáticas (las variables de instancia) del mismo, más los valores actuales (las referencias a otros objetos) de cada una de esas propiedades». Se puede desprender que el estado de un objeto queda establecido por el conjunto de colaboradores internos que posee en un determinado instante durante su tiempo de vida. Asimismo, el estado interno de un objeto se dice que está **encapsulado**, ya que no puede ser accedido directamente y su representación es invisible fuera del objeto.

Especificación del Comportamiento de los Objetos

Cada método que puede ejecutar un objeto tiene especificado un nombre o selector, los objetos que toma como parámetros con sus respectivos tipos[3], y el tipo de objeto que retorna como resultado de su ejecución. Esto se conoce como la **firma** del método. En los lenguajes de programación tipados como *Java*, es fácil reconocer en el código la firma de los métodos, debido a que cada uno posee bien definido un selector, la clase o tipo de los parámetros, y la clase o tipo del objeto retornado. En lenguajes no tipados, como *Smalltalk*, es imposible reconocer la firma completa de un método, ya que las clases de los objetos que son argumentos y del objeto retornado por dicho método no está especificado en el código.

El conjunto de todas las firmas definidas por los métodos que un objeto es capaz de ejecutar se conoce como interfaz o **protocolo** del objeto. El protocolo de un objeto caracteriza a la colección completa de requerimientos que pueden solicitarse a dicho objeto. Cualquier mensaje que ajuste con una firma en el protocolo de un objeto puede ser enviado al mismo.

Los protocolos son fundamentales en sistemas orientados a objetos. Los objetos son conocidos sólo a través de sus protocolos. No hay manera de conocer algo acerca de un objeto o solicitarle la ejecución de algún método sin no ser a través de su protocolo. El protocolo de un objeto no dice nada acerca de su implementación, esto implica que diferentes objetos tienen la libertad de implementar sus requerimientos de diferentes maneras. Así, dos objetos que tienen implementaciones completamente diferentes puede tener idénticos protocolos.

Cuando se envía un mensaje a un objeto, el método que se ejecuta depende del mensaje en sí y del objeto receptor. Diferentes objetos que soportan requerimientos idénticos pueden tener diferentes implementaciones de los métodos que satisfacen dichos requerimientos. La asociación en tiempo de ejecución de un requerimiento a un objeto y a uno

[3] Haciendo referencia a los tipos predefinidos del lenguaje.

de sus métodos es conocida como **ligadura dinámica**. La ligadura dinámica significa que el envío de un mensaje no se liga a una implementación en particular hasta el tiempo de ejecución del mismo. Los programas *Smalltalk* no declaran los tipos de las variables; consecuentemente, el compilador no comprueba que el tipo de los objetos asignados a una variable sean subtipos de los tipos de la variable. El envío de un mensaje requiere comprobar que la clase del receptor implementa dicho mensaje, pero no se requiere comprobar que el receptor sea una instancia de una clase particular. Consecuentemente, es posible escribir un programa que usa a un objeto con un protocolo particular, con la seguridad de que cualquier objeto que tiene el protocolo esperado aceptará cualquier requerimiento. Además, la ligadura dinámica permite sustituir un objeto por otro que tiene el mismo protocolo en tiempo de ejecución. Esto se conoce como **polimorfismo**, y es un concepto crucial en los sistemas orientados a objetos. El polimorfismo se manifiesta cuando dos objetos son capaces de responder a mensajes con la misma firma de maneras semánticamente equivalentes. Dos clases son polimórficas si sus instancias también lo son. De este modo, las dos clases definen los mismos mensajes, los cuales resultan implementados por métodos polimórficos: tienen el mismo selector, los mismos tipos de parámetros, los mismos efectos colaterales, el mismo tipo de resultado y el mismo propósito, aunque no tienen los mismos detalles de implementación. Por lo tanto, es posible intercambiar dos objetos polimórficos de manera transparente para el emisor del mensaje, debido a que sus firmas y sus semánticas son las mismas, aunque difieran en sus implementaciones. En forma trivial, un objeto es polimórfico consigo mismo si se considera a la misma implementación de un mensaje.

Tipos de Objetos

Un **tipo** es un nombre utilizado para denotar un protocolo en particular. Por ejemplo, se dice que un objeto posee el tipo "Window" si sabe responder a todo los requerimientos asociados a los métodos que define el protocolo llamado "Window". Un objeto puede tener muchos tipos, y asimismo, distintos objetos pueden compartir un mismo tipo. Parte del protocolo de un objeto puede ser caracterizado por un tipo, y otras partes por otros tipos. Dos objetos del mismo tipo necesitan compartir sólo parte de sus protocolos. Los protocolos pueden contener otros protocolos como subconjuntos. Se dice que un tipo es un **subtipo** de otro si el protocolo asociado al primero contiene el protocolo asociado al segundo, el cual se conoce como **supertipo**. A menudo se dice que un subtipo hereda el protocolo de su supertipo.

Es importante entender la diferencia entre clase y tipo. La clase de un objeto define cómo está implementado el objeto; define la estructura de su estado interno y la implementación de sus métodos. En cambio, el tipo de un objeto sólo se refiere a su protocolo — el conjunto de mensajes a los cuales puede responder. Un objeto puede tener muchos tipos, y los objetos de clases diferentes pueden tener el mismo tipo.

Por supuesto, existe una relación cercana entre el concepto de clase y tipo. Una clase define los métodos que un objeto puede ejecutar, también define el tipo del objeto. Al decir que un objeto es una instancia de una clase, se insinúa que el objeto soporta el protocolo definido por dicha clase. Los lenguajes como *C++* y *Eiffel* usan clases para especificar el tipo y la estructura interna de un objeto.

Mecanismos de Sharing de Información

Los ambientes de programación orientados a objetos pueden ser de dos clases: basados en clases y basados en prototipos. Los ambientes basados en prototipos resuelven la creación de objetos mediante la clonación de prototipos y objetos “no-clases” o *classless*, que son depositarios del comportamiento común. Ejemplos de éstos son: Agora [Agora], Brain [Brain], Cecil [Chambers, 1993] y Self [Ungar et al., 1987]. Por su parte, en los ambientes basados en clases, las clases son una estructura muy relevante, cuyas responsabilidades básicas son la creación de sus instancias y el ser depositaria del comportamiento común de las mismas. Como ejemplos pueden citarse a Smalltalk, CLOS [Keene, 1989] y Python [Python].

Un concepto importante en la orientación a objetos es el **sharing de información**. Dicho concepto establece una manera de definir un nuevo objeto en términos de uno ya

existente, compartiendo tanto la estructura interna como el comportamiento del objeto definido previamente. Se reconocen dos mecanismos fundamentales para implementar el “sharing” en los lenguajes orientados a objetos: *delegación* y *herencia*. La **delegación** es el mecanismo utilizado para implementar sharing de información en ambientes basados en prototipos; en cambio, en ambientes basados en clases, el mecanismo utilizado para este fin es la **herencia**.

Las clases pueden estar definidas en término de clases existentes usando herencia de clase. Cuando una clase hereda de una clase padre o **superclase**, se la conoce como **subclase** e incluye todas las definiciones de los datos y los métodos que la superclase define. Los objetos que son instancias de la subclase contendrán todos los datos definidos por dicha subclase y los definidos por sus superclases, y podrán ejecutar todos los métodos definidos por la subclase y sus superclases.

Las subclases pueden refinar y redefinir el comportamiento de sus superclases, es decir, una clase puede redefinir un método definido en una de sus superclases. La redefinición de métodos da a las subclases la posibilidad de manejar requerimientos en lugar de que los maneje la superclase que también los define. La herencia de clase permite definir clases simplemente extendiendo otras clases, facilitando definir familias de objetos que tienen funcionalidad relacionada.

Es importante resaltar que la herencia de clase combina herencia de protocolo y herencia de estructura interna. La herencia de protocolo define un protocolo nuevo en términos de uno o más protocolos existentes. La herencia de estructura interna define una nueva estructura interna en términos de una o más estructuras existentes. También, es importante entender la diferencia entre herencia de clase y herencia de protocolo. La herencia de clase define la implementación de un objeto en términos de la implementación de otro objeto. Concretamente, es un mecanismo importante para el reuso de código y de representación. En cambio, la herencia de protocolo describe cuando un objeto puede ser usado en lugar de otro.

Una **clase abstracta** es una clase cuyo propósito principal es definir un protocolo común para sus subclases. Una clase abstracta delega parte o toda la implementación para los métodos que define a las subclases; de esta manera, una clase abstracta no debería ser instanciada. Los métodos que una clase abstracta declara pero no implementa se llaman **métodos abstractos**. Las clases que no son abstractas son llamadas **clases concretas**.

La delegación es una forma de hacer tan poderosa a la composición para el reuso como la herencia. En la delegación, dos objetos están involucrados en manipular un requerimiento: un objeto receptor delega requerimientos a su delegado. Esto es análogo para las subclases que difieren requerimientos a sus superclases.

Herencia versus Composición

Las dos técnicas más comunes para reusar funcionabilidad en sistemas orientados a objetos son la herencia de clase y la **composición de objetos**. Como se explicó anteriormente, la herencia de clase permite definir la implementación de una clase en términos de otra. El reuso mediante la subclasificación se conoce como **reuso de caja blanca**. El término "de caja blanca" se refiere a la visibilidad: con herencia, la implementación de las superclases son visibles a las subclases.

La composición de objetos es una alternativa a la herencia de clase. La funcionabilidad nueva es obtenida ensamblando o componiendo objetos para obtener funcionabilidad más compleja. La composición de objetos requiere que los objetos que forman la composición tengan interfaces bien definidas. Este estilo de reuso es llamado **reuso de caja negra**, porque ninguno de los detalles internos de objetos son visibles. Los objetos aparecen sólo como "cajas negras".

Contexto de los Objetos

Un objeto por sí solo es bastante poco interesante. Los objetos contribuyen al comportamiento de un sistema colaborando entre sí. Como sugiere *Ingalls* [Ingalls], «en lugar de un procesador triturador de *bit* que golpea y saquea estructuras de datos, tenemos un universo de objetos bien educados que cortésmente solicitan a los demás que lleven a cabo sus diversos deseos». La mayor parte de los servicios que brinda un objeto se resuelven gracias a la colaboración de otros objetos. A su vez, cada servicio que ofrece forma parte del desarrollo de algún requerimiento bajo la responsabilidad de otros objetos.

Dentro de una aplicación, los objetos necesitan enviar y recibir mensajes para poder colaborar unos con otros. Para que esto suceda, cada objeto debe tener la capacidad de conocer a sus colaboradores, lo cual logra obteniendo o creando referencias hacia los mismos. Todas las relaciones de colaboración que se producen dentro de una aplicación forman una estructura similar a un grafo, donde los nodos son objetos y las aristas las relaciones entre ellos. Dicho grafo será referenciado como **red de colaboración**. Cada objeto está inmerso en una red de colaboración, y todos los objetos que participan en ella forma su contexto. Por lo tanto, el contexto de un objeto se define como el conjunto de objetos que forman parte de la red de colaboración donde está inmerso dicho objeto.

Una red de colaboración no es estática, ya que sus participantes pueden variar en cada momento. Esto implica que existe la posibilidad de que un objeto sea sustituido por otro objeto, o que aparezcan y desaparezcan objetos en la red. Por lo tanto, es importante destacar que una red de este tipo puede extenderse y atravesar los límites de la aplicación original[4], incluyendo objetos que se encuentran participando en otra red de colaboración al mismo tiempo.

[4] Por aplicación "original" se entiende al conjunto inicial de objetos que dieron origen a la aplicación.

APÉNDICE II

Glosario

Clase: Una clase define el protocolo de un objeto y su implementación. Especifica la representación interna del objeto y define los métodos que el objeto puede ejecutar.

Clase abstracta: Una clase cuyo propósito principal es definir un protocolo. Una clase abstracta delega parte o toda la implementación en sus subclases. Una clase abstracta no debería ser posible instanciarla.

Clase concreta: Una clase que no tiene métodos abstractos y puede ser instanciada.

Colaborador: Un objeto que colabora con algún servicio en la satisfacción de otro servicio, bajo la responsabilidad de otro objeto, mediante un determinado mensaje al cual es capaz de responder.

Composición de objeto: Ensamblaje o composición de objetos para obtener comportamiento más complejo.

Conocimiento: Relación entre dos objetos, donde uno de ellos es capaz de obtener una referencia al otro objeto para solicitarle cierta colaboración, enviándole cierto mensaje.

Conocimiento global: Es el tipo de conocimiento entre objeto mediante el cual, un objeto puede ser referenciado en cualquier momento por otro objeto a través de un identificador único asociado al mismo.

Conocimiento indirecto: Este conocimiento se establece cuando existe una relación de dependencia entre dos objetos, donde el objeto independiente no conoce al objeto dependiente, pero existe un mecanismo mediante el cual puede notificarle de ciertos cambios al mismo.

Conocimiento interno: Se establece cuando un objeto posee una referencia a otro objeto, la cual puede conservar durante todo su tiempo de vida.

Conocimiento temporal: La relación de conocimiento entre dos objetos se mantiene en el instante que dura la satisfacción de un determinado requerimiento.

Conocimiento temporal externo: Es el tipo de conocimiento temporal, donde la referencia del objeto que es conocido, está dada por un parámetro del mensaje que inicia el requerimiento.

Conocimiento temporal interno: Es el tipo de conocimiento temporal, donde la relación de conocimiento se establece explícitamente en algún punto del desarrollo del requerimiento.

Delegación: Un mecanismo de implementación en el cual un objeto reenvía o delega un requerimiento a otro objeto. El delegado realiza la petición en nombre del objeto original.

Encapsulamiento: También conocido como *ocultamiento de la información*, es la propiedad que permite asegurar que la implementación de un objeto le es desconocida a los demás objetos en una aplicación.

Estado: El estado de un objeto está determinado por el conjunto de objetos colaboradores que conoce internamente dicho objeto, mediante sus variables de instancia.

Firma: La firma de un método define su nombre, sus parámetros, y su valor de retorno.

Herencia: La herencia de clase es una relación existente en los ambientes basados en clases que define una nueva clase en términos de otra clase (*herencia simple*) o más clases padre (*herencia múltiple*). La nueva clase hereda el protocolo y la estructura interna de sus superclases.

Ligadura dinámica: La asociación en tiempo de ejecución de un requerimiento a un objeto y a uno de sus métodos.

Mensaje: Un mensaje es una solicitud en un objeto para que lleve a cabo un servicio. Cuando un objeto recibe un mensaje se ejecuta un método, ya que cada mensaje tiene asociado un método y su invocación dispara la ejecución de dicho método.

Método: Un objeto ejecuta un método cuando recibe un mensaje. La especificación de un método determina cómo debe comportarse un objeto cuando dicho método se ejecute.

Método abstracto: Una método que declara una firma pero no la implementa.

Objeto: Un objeto es una entidad que encapsula datos y operaciones que operan sobre esos datos.

Polimorfismo: La habilidad de sustituir objetos que concuerdan sus protocolos en tiempo de ejecución.

Protocolo: El protocolo de un objeto define la parte externa de su comportamiento, el conjunto de todas las firmas que se le permite acceder a otros objetos, es decir, el conjunto de mensajes a los cuales puede responder.

Red de colaboración: Todos los objetos y las relaciones de colaboración entre ellos que se producen dentro de una aplicación.

Reuso de caja blanca: Un estilo de reuso basado en la herencia de clases. Una subclase reusa el protocolo e implementación de su superclase, y puede tener acceso a los aspectos privados de sus clases padres.

Reuso de caja negra: Un estilo de reuso basado en la composición de objetos. Los objetos que forman la composición no revelan detalles internos al resto y son así análogos a "cajas negras".

Sharing de información: Parte fundamental de la naturalidad de expresión provista por la programación orientada a objetos que permite compartir datos, código y definición.

Subclase: Una clase que hereda de otra clase.

Subtipo: Un tipo es un subtipo de otro si su protocolo asociado contiene el protocolo del otro tipo.

Superclase: La clase desde la cual otra clase hereda.

Supertipo: El tipo padre desde el cual un tipo hereda.

Tipo: Nombre de un protocolo en particular.

Variable de instancia: Una porción de información que define parte de la representación de un objeto.

APÉNDICE III

Notaciones UML utilizadas

El lenguaje para modelado unificado (*UML*), es un lenguaje para la especificación, visualización, construcción y documentación de los artefactos de un proceso de sistema intensivo. Fue originalmente concebido por la corporación *Rational Software* y tres de los más prominentes métodologistas en la industria de la tecnología y sistemas de información: *Grady Booch*, *James Rumbaugh*, y *Ivar Jacobson*. El lenguaje ha ganado un significante soporte de la industria de varias organizaciones vía el consorcio de socios de *UML,* y ha sido presentado al *Object Management Group (OMG)* y aprobado por éste como un estándar en noviembre 17 de 1997.

UML es un lenguaje para modelado de propósito general evolutivo, ampliamente aplicable, factible de ser soportado por herramientas e industrialmente estandarizado. Se aplica a una multitud de diferentes tipos de sistemas, dominios, y métodos o procesos.

Diagramas de Clases

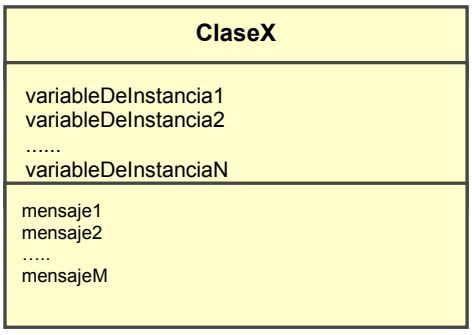
Un diagrama de clases es una vista gráfica de todo o parte del modelo estático estructural de un sistema; es un grafo de elementos conectados por las diversas relaciones estáticas. Puede contener clases, interfaces, paquetes, relaciones, y aún instancias, tales como objetos y enlaces, aunque su uso más común es ilustrar modelos de clases.

Un diagrama de clases que muestra las clases del sistema y sus relaciones refleja:

- Las clases con su estructura y comportamiento.
- Las relaciones entre dichas clases: asociaciones, agregaciones, dependencias y/o herencia.
- Indicadores de multiplicidad y navegación.

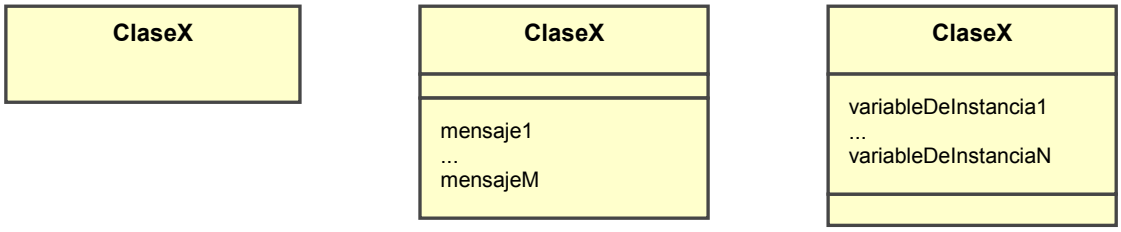
Clases

En un diagrama de clases, un icono de clase se representa como un rectángulo con tres divisiones. En la división superior se escribe el nombre de la clase, con cursiva si la clase es abstracta y normal si la clase es concreta. En la división del medio se escriben las variables de instancias, y en la división inferior los mensajes que define la clase.



Las divisiones que muestran las variables de instancias y los métodos pueden eliminarse para reducir la cantidad de detalles a ser expuestos. Si la división del medio o la

inferior están vacías significa que la clase no tiene variables de instancias o métodos respectivamente.

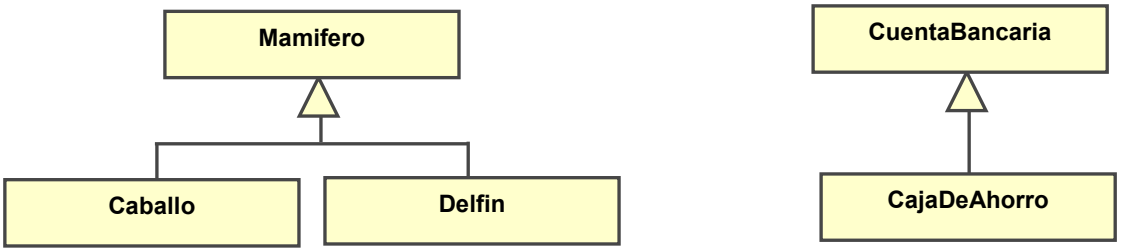


Relación de Herencia

La herencia es el mecanismo por el cual elementos más específicos incorporan estructura y comportamiento de elementos más generales.

Una relación de herencia entre dos clases muestra que una clase (la subclase) comparte la estructura y el comportamiento definido en la otra clase (superclase). Representa la relación "Es-Un" entre dos clases.

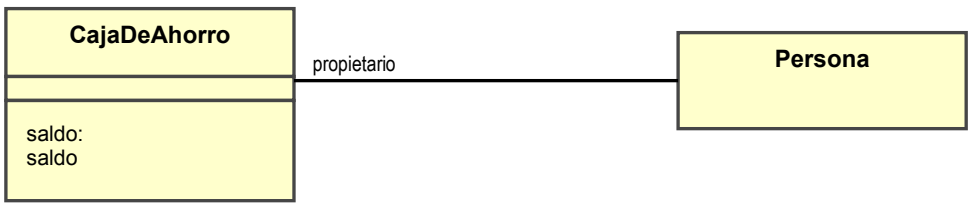
Gráficamente, la herencia entre clases se muestra de la siguiente manera:



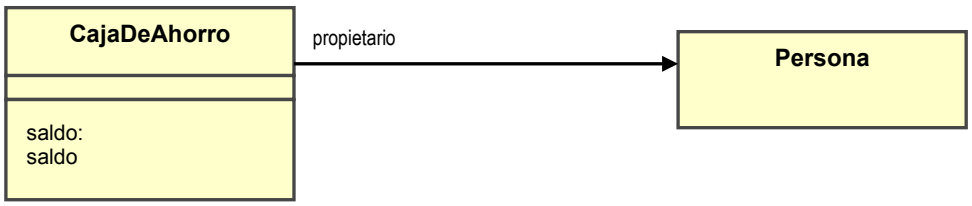
Relación de Asociación

Una asociación denota una dependencia semántica. Es una relación entre dos clases que especifica conexiones entre sus interfaces. La dirección de esta dependencia, y la manera exacta en la cuál una clase se relaciona con otra se deja implícitamente.

Una asociación se representa gráficamente con una línea entre dos clases, y el nombre de la misma se denota sobre dicha línea:

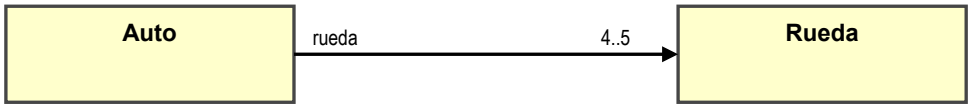


En ocasiones, para facilitar la lectura del gráfico, se puede reemplazar la línea por una flecha para indicar la navegabilidad de la relación:

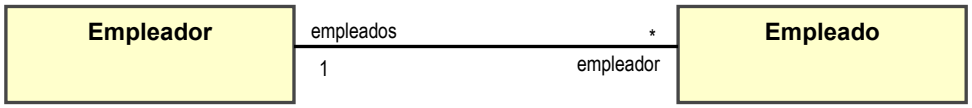


Significa: “Una caja de ahorro conoce a una persona”.

También, es posible indicar la cardinalidad de la asociación, la cual se especifica como un intervalo o rango (posiblemente infinito) de enteros, de la siguiente forma: *valor-bajo.... valor-alto*. El carácter “*” puede ser usado en lugar del *valor-alto*, denotando un valor ilimitado. Si se denota el valor de un entero simple, entonces el rango de enteros contiene sólo un simple valor. Si la especificación de la cardinalidad comprende solamente el símbolo “**”, denota un rango de enteros no negativos infinito, equivalente a “0..*” (cero o más). En el caso de que no se especifique la cardinalidad en la asociación, se asume que contiene el rango “1..1” (en el ejemplo anterior, la asociación se lee “una caja de ahorro tiene un propietario”).



Significa: “Un auto conoce a 4 o 5 ruedas”.

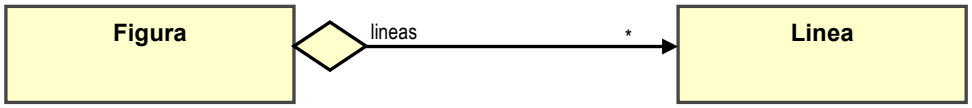


Significa: “Un empleador conoce a 0 ó más empleados, y un empleado conoce a un empleador”.

Relación de Agregación

La relación de agregación entre clases es una forma especial de asociación que especifica una relación de “Todo-Parte” entre el agregado (una clase que representa el “Todo”) y una parte componente (otra clase que representa la “Parte”).

En un diagrama de clases, una agregación se muestra de la siguiente manera:

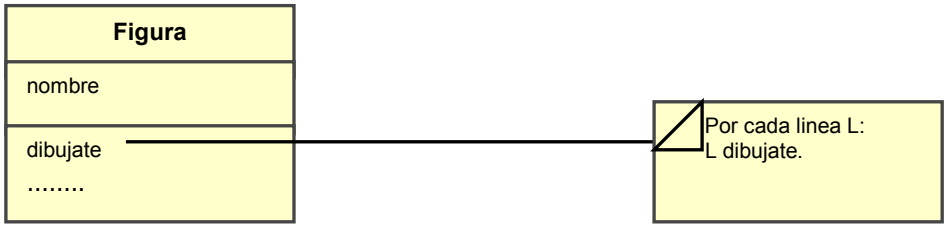


En el ejemplo, la clase *Figura* representa el todo y la clase *Linea* representa la parte.

Notas Aclaratorias

Una nota es un ítem notacional, muestra información textual en algún elemento semántico. Capturan todos los asuntos y decisiones aplicadas durante el análisis y diseño. Las notas pueden contener cualquier información, incluyendo texto, fragmentos de código, o referencias hacia otros documentos.

Una nota se representa en un diagrama de clase de la siguiente manera:



Diagramas de Secuencia

Un diagrama de secuencia muestra interacciones entre objetos ordenadas en el tiempo. Tiene dos dimensiones: la dimensión vertical que representa el tiempo y la dimensión horizontal que representa diferentes instancias. Son diagramas que especifican el orden de llamadas y procesamiento de la información a través de ciertos objetos que pertenecen al sistema.

Desde los objetos que posee un diagrama de secuencia emergen líneas verticales que representan el paso del tiempo. Un rectángulo sobre la línea que emerge de un determinado objeto representa el tiempo relativo en el que el foco de control de la ejecución está en ese objeto. Las flechas entre las líneas verticales representan los mensajes que los objetos se envían y las respuestas devueltas.

Estos diagrama puede especificar valores de retorno, condicionales, iteraciones, autodelegación, creación y destrucción de objetos.