Índice

Operadores....1
Visibilidad....3
Especificación de clases....4
Clases....4
Clases abstractas....8
Clases parametrizadas y estereotipadas....9
Interfaces....11
Definición de relación....13
Relación de dependencia....16
Relación de generalización....17
Relación de asociación....19
Navegación....24
Visibilidad en las navegaciones....27
Agregación....28
Composición....31
Clase asociación....34
Diagramas estructurales en UML....37
Asociación n-aria....37
Realización....40
Instancias....41
Paquetes....44
Dependencia entre paquetes....47
Introducción a Diagrama de Clases....50
Modelo conceptual....53
Modelo de Diseño....54
Diagrama de Objetos....56
Diagrama de Paquetes....57

Operadores

Dos palabras, conceptos, son los importantes en este contexto, la operación vista como la especificación ya sea de una transformación si cambia el estado como de una consulta si estamos no cambiando el estado, sino visualizando valores de atributos de un objeto; y el método, que es un concepto de más bajo nivel porque hace referencia a la implementación de la operación.

La operación constituiría el nivel descriptivo, más centrado en la especificación, el método constituiría la perspectiva más operacional centrado en la implementación, en la materialización de la operación. Las declaraciones de una operación se pueden heredar por los descendientes del clasificador en el caso de que exista una jerarquía de especialización, cuando se da el caso de que otra declaración tiene la misma signatura, entonces podemos concluir que se trata de la misma operación. En ese caso cuando existe un antepasado común final de todas las declaraciones, este antepasado único final se conoce con el nombre de origen, es el origen de la operación y constituiría en última instancia la definición más concreta de la operación en su inicio.

Aquí tenemos la sintaxis que empieza, pues igual que ocurría en el caso de los atributos, con ese estereotipo que en su momento veremos y que como adelantábamos permite introducir nuevos

conceptos dentro de lo que es el el ámbito de modelado, nuevos nuevas primitivas de modelado un estereotipo, estereotipar es convertir un clasificador en otro tipo de concepto. En el caso de la visibilidad tenemos lo mismo que veíamos por atributos, pública, privada, protected, el nombre de la operación con su lista de parámetros, el tipo de retorno y las propiedades adicionales que podemos declarar entre llaves como podemos ver en esta plantilla de sintaxis. Bien en cuanto a lo que hace referencia a lista de parámetros, para la sintaxis de dichos parámetros pues tenemos una dirección, que puede tomar los cuatro valores que vemos aquí

- in hace referencia a que existe un paso de valor dentro de la operación, el objeto emisor no va a poder no va a poder tener acceso a ese, a cualquier cambio que se produzca sobre ese valor pasado por como argumento de entrada.
- out es un argumento de salida el el objeto emisor ahí y sí que va a tener acceso al argumento de salida y no va a haber ningún argumento, ningún valor de entrada.
- Inout tenemos un valor de entrada y un valor de salida que será accesible por el objeto emisor
- return. hace referencia a un valor devuelto por la llamada que va a ser accesible también para el objeto emisor.

Junto a este junto a este, con estas palabras clave para la dirección viene a continuación el nombre del parámetro en cuestión y el tipo del parámetro con lo que podemos ya complementar la declaración de la operación con estas propiedades que, como comentábamos, aparecen entre llaves y que son fundamentalmente estas tres que vemos. Aquí, las dos primeras son de tipo booleano, si es una consulta, isQuery, true o false eso querrá decir si se trata de una operación que cambia el estado, que cambia algún atributo del objeto sobre el que se ejecuta la operación o si si no lo cambia, pues entonces va a ser una operación de consulta y eso, en función de que el valor sea true o false, va a ser una consulta o no va a ser una consulta y va a querer decir
que la operación cambia el estado del objeto. IsPolimorfic, si es polimórfico hace referencia a la posibilidad de modificar la operación en el caso de que estemos, esté implicada en una jerarquía de especialización. Si isPolimorfic is true va a querer decir que se puede, se puede cambiar y se puede modificar la implementación de la operación a lo largo de la jerarquía de herencia o especialización, si es falso no se puede hacer y finalmente la propiedad de concurrencia que tiene tres tres valores posibles:

- secuencial cuando tenemos que obtienen los las operaciones que van acceder a un objeto concreto tienen que de alguna forma estar estar sincronizadas a la hora de ver cuál es la que va a ser ejecuta porque sólo una operación va a poder actuar sobre el objeto en concreto.
- guarded, guardadas, una relajación de esta condición pueden haber muchas operaciones en paralelo pero sólo una se va ejecutar, en el caso secuencial solo puede haber un accediendo y un ejecutándose, en el guarded pueden haber varias que acceden pero una se ejecuta mientras las otras esperan bloqueadas
- concurrent, concurrente, hace referencia al hecho de que tengamos concurrencia masiva, que pueda tener varias operaciones ejecutándose en paralelo sobre el mismo objeto.

Bien finalmente las operaciones importante destacar que igual que ocurría con los atributos de clase, recuerden un atributo subrayado quería decir que un atributo que aplicar, se aplicaba todas las instancias de una clase de la misma forma una operación cuyo alcance es la clase se presenta, se específica como una operación con una línea subrayada, se suelen expresar en minúscula para de alguna forma determinar visualmente de que trata de operaciones, también se suelen también mostrar en cursiva las operaciones abstractas cuando son operaciones cuya implementación todavía no es conocida y puede, podemos ver en los siguientes ejemplos pues casos concretos de operaciones.

Para completar o para terminar la sesión, insistir en que estas operaciones pueden ser operaciones de cambio de estado cuando se trata de operaciones que cambian el valor de algún atributo o pueden

ser operaciones de consulta cuando se trata de operaciones que lo único que van a hacer es visualizar al uno o varios de los atributos una clase para conocer el estado del objeto que está, que esté siendo accedido.

Visibilidad

Hemos estado adelantando en algunos temas previos este concepto de visibilidad que me permite determinar de qué forma van a ser accesibles los clasificadores y las propiedades a ellos asociadas. Cuatro palabras clave son las que nos van a permitir delimitar el tipo de visibilidad.

Estas cuatro opciones determinadas por las cuatro palabras clave que vemos en la figura son:

- acceso público que se especifica con un signo más y que hace referencia a las situaciones en las que el clasificador o sus propiedades correspondientes puede ser accesibles por cualquier objeto externo.
- Protected (#), con el segundo el segundo tipo de visibilidad que vamos a ver cuando esta ese acceso a esa a ese clasificador o esas propiedades de clasificador solo va a ser posible en el ámbito de los objetos derivados del clasificador. El ejemplo típico en el ámbito de un diagrama de clase sería el de una propiedad que declarada como protected solo va a ser accesible, sólo va a ser visible en el ámbito de la jerarquía de especialización en la que ese clasificador este este inmerso, por ejemplo si estamos hablando de una clase y tenemos una jerarquía de especialización pues yo podré en el caso de que lo tenga asi especificado, como protected, sólo podré acceder a las propiedades de esa de ese clasificador en el ámbito del conjunto de objetos derivados de clases derivadas que estarían incluidas en la correspondiente jerarquía de especialización o de herencia.
- Privada (-) El tercero de los casos es la privacidad sin restricciones, un un tipo de visibilidad privado hace referencia a que no hay posibilidad acceder externamente, desde objetos o clasificados externos, a esta a este clasificador o a esta propiedad que esté especificada en el clasificador y por lo tanto la visibilidad se va a restringir al propio clasificador en el que está siendo definido, especificado el tipo de propiedad al que le estamos otorgando esta característica de visibilidad privada.
- Finalmente una cuarta opción, la de package(~), que hace referencia a que la visibilidad para ese recurso va a estar en el ámbito de lo que se conoce como contenedor de del conjunto de recursos, veremos cuando estudiamos diagramas de paquetes como podemos definir en UML. Ese concepto de paquete con el que consigo agrupar, agrupar clasificadores comunes en un entorno en ese paquete que va a permitir que de alguna forma gestionemos la complejidad de grandes sistemas creando subsistemas en los que la visibilidad queda restringida dentro de ese subsistema, dentro de ese paquete, a aquellos clasificadores que están formando parte de dicha visibilidad parcial que va asociada a ese contenedor que el paquete constituye.

En el caso al hablar de tipos de visibilidad en atributos y operaciones de clasificadores, lo habitual es que los atributos sean, tengan una visibilidad privada como consecuencia de ese principio de encapsulación que hace que yo no pueda o que cualquier objeto tenga que estar protegido a la hora de que no se pueda modificar el valor de los atributos de de un objeto de una clase, no se pueda

modificar el estado que a través de el valor de dichos atributos de la clase conforma conforma el el estado actual del del objeto, que sea privado vale es decir que sólo lo pueda modificar usando operaciones de esa clase y no con cualquier otro tipo de recurso de cualquier otra clase. Por eso sus atributos son privados y van a ser modificable sólo a través de operaciones, operaciones que modifican los atributos pero que sí que son habitualmente públicas porque esas operaciones van a tener que poder ser ejecutadas por algún actor, van a tener que poder ser ejecutadas por algún cliente que va a ser el cliente agente encargado de o que tiene la el privilegio de poder ejecutar esas operaciones y de esa forma ser el encargado de activar el la operación que permitirá a cambiar atributos del estado y por lo tanto cambiar el los atributos del estado a través de la ejecución de la operación. Es por ello que habitualmente los atributos son privados, preservamos el principio de encapsulación, las operaciones son públicas, en un caso real esta publicidad de las operaciones también se restringe a aquellos aquellos actores, aquellos usuarios que van a ser los que tengan la posibilidad, el privilegio, el permiso de poder activar la operación correspondiente y que luego se va a plasmar esto que en la aplicación software final pues bueno aquello que podrá hacer un usuario que se conecte va a ser aquello para lo que haya sido autorizado de acuerdo con la visibilidad que en el diagrama de clases correspondiente se haya especificado.

Especificación de clases

Clases

Las clases constituyen un concepto básico, en UML es el clasificador esencial en UML y hace referencia a un conjunto de objetos que comparten tanto su estado a nivel estructural como su comportamiento. La noción de estado en general, la de especificación del estado una clase y de todos y cada uno de los objetos que puedan componer la población de esa clase, se describe a través de atributos cuando estamos hablando de tipos de datos básicos y de asociaciones cuando estamos hablando de tipos de datos objeto valuados, aclaramos esto a continuación.

Los atributos se utilizan para expresar los valores que podríamos llamar puros de los datos, sin identidad, estamos haciendo referencia a lo que serían tipos abstractos de datos, tengo una propiedad de una clase que represento mediante un atributo al que asigno un tipo de datos básico, yo puedo tener perfectamente pues por ejemplo un atributo de tipo nombre en una clase empleado que es de tipo pues cadena de caracteres, ese atributo hace referencia el nombre, hace referencia a un tipo de datos básico que es la cadena o string de caracteres.

Pero tenemos asociaciones también, que vamos a utilizar para hacer conexiones entre objetos que sí que tienen identidad, yo puedo tener por ejemplo un empleado con un nombre que su que tiene un tipo de datos básico como atributo, pero que va asociado donde departamento que es otra clase y la clase departamento tiene objetos también, que referencia a departamentos individuales con su identidad particular. Ven el empleado igual que tiene un nombre, que es un atributo, tendrá un departamento que es una asociación. Es por eso que las asociaciones podemos decir que denotan atributos objeto valuados, porque realmente a través de la asociación estamos navegando de un objeto de la clase en la en la que estoy, al objeto asociado con él que es el que constituye ese atributo objeto evaluado al que nos estábamos refiriendo.

Es importante destacar que las piezas individuales de comportamiento se invocan a través de operaciones y esta parte compar, de comportamiento compartimental es la que permite complementar la especificación estructural de la clase que realizamos con atributos y asociaciones con las operaciones a partir de las cuales yo voy a ser capaz de cambiar el estado del objeto modificando atributos de dicho objeto.

Bien, como decíamos anteriormente, la construcción estructural por excelencia en sistemas orientados a objetos y en UML que tiene una vocación orientada a objetos desde su nacimiento muy grande pues, lo lo asume de manera explícita las clases son el constructor fundamental y de hecho estudiaremos en este curso como diagrama esencial de UML el diagrama de clases en el que representaremos clases y relaciones entre clases como ya estamos empezando a caracterizar.

La clase representa un concepto discreto dentro de la aplicación que estamos modelando, cuando estamos hablando de ejemplos típicos como una clase empleado o una clase departamento pues estamos visualizando un dominio en el que tenemos empleados, tenemos departamentos y por lo tanto los conceptos que percibimos se convierten en clases que son la clase empleado, la clase departamento y posteriormente veremos que las relaciones necesarias para que estas clases pues en el caso de que tengan asociaciones o especializaciones asociadas las podamos especificar.

El concepto distancia es fundamental, realmente el molde para crear instancias es lo que es la clase y la materia la instanciación de una clase en cada uno de los objetos que van a constituir su población, es una es un es un paso pues básico dentro del del ámbito de modelado en el que estamos hablando.

Hablar de istanciación es hablar del proceso de creación de objetos individuales que toman la plantilla genérica de la clase como ejemplo y generan una instancia particular de dicha clase en forma de objeto. La clase por supuesto dicta a través de ese nivel descriptivo, de especificación, la estructura y el comportamiento que tendrán todas y cada una de las instancias de la clase. Estas instancias, estos objetos, van a contener localmente todos los datos que se corresponden con la estructura que viene dictada por la clase y eso conforma una noción muy importante que es la de estado del objeto.

El estado del objeto viene dado por, observado un objeto en un momento determinado, su estado será el conjunto de valores que todos y cada uno de sus atributos tienen el momento en el que está siendo observado.

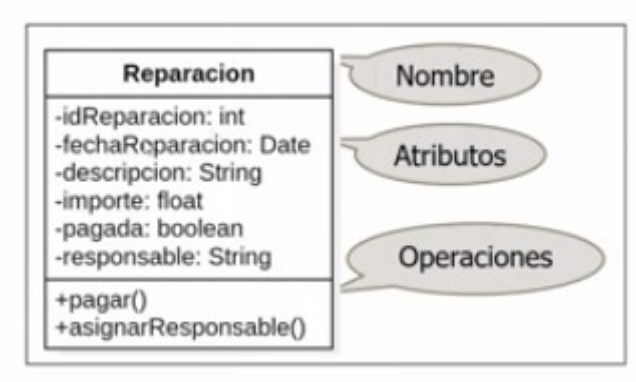
Si hablamos de un objeto empleado, un empleado de una organización, como objeto individual de la clase empleado su estado en el momento en el que es observado va a venir dado por el valor que tienen todos y cada uno de los atributos de ese objeto empleado en el momento en el que realizamos la observación.

Si su nombre es, su nombre, su estado civil, fecha de nacimiento, dirección, etcétera son atributos pues el valor de cada uno de estos atributos va a caracterizar el estado del objeto.

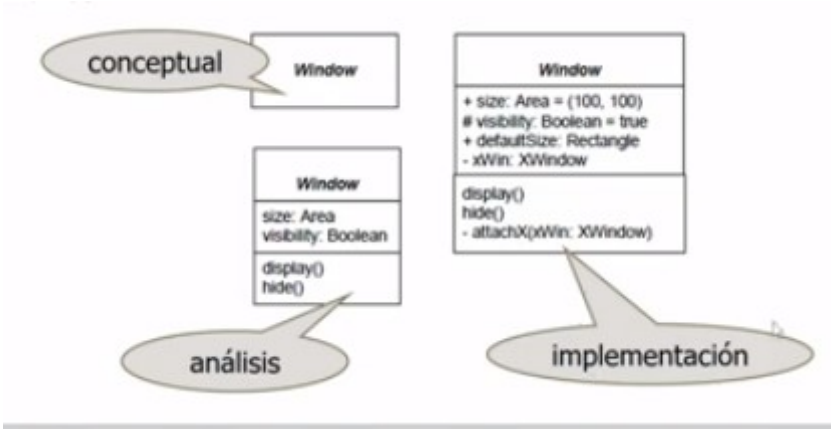
Bien, la representación gráfica que ya estábamos viendo y que poco a poco vamos a ir refinando es, lo veíamos en una lección anterior, con la factura pues aquí tenemos una clase, una clase reparación, una reparación de algún tipo de avería. Esa reparación tiene tres partes, el nombre de la clase que lo vemos en la parte superior en lo que es la reparación, a continuación en la parte inferior tenemos la

declaración de atributos que que determinan la la parte estructural, lo que es la estructura de datos de de la clase que estamos definiendo, pues aquí tenemos un identificador de la reparación, una fecha de tipo fecha como ven cada cada nombre de atributo viene con su tipo de datos de acuerdo con la, con la fórmula de especificación de atributos que vimos cuando veíamos cómo especificar atributos de clasificadores, la descripción de la reparación, su importe, un atributo booleano que me indique si ha sido pagada o no ha sido pagada y un atributo de tipo de tipo string, cadena de caracteres, que me indique quién es el responsable de esa reparación.

Bien, tras el nombre y los atributos viene la declaración de las operaciones, las operaciones van a permitir incluir para, la perspectiva de comportamiento dentro de la especificación de la clase. No sólo tenemos el nombre y los atributos que caracterizan el estado de la clase, sino que tenemos aquí las operaciones que van a permitirme cambiar ese estado, en este caso pues tenemos dos operaciones por ejemplo que son el pagar y asignar responsable, dos operaciones que no tienen en este caso argumentos adicionales que están como ven representadas como operaciones públicas, es decir, que son accesibles por objetos externos y cuya ejecución va a tener como consecuencia pues que cambiará, cambiará el estado de los objetos de la clase a través del cambio de cualquiera de los atributos que hayamos declarado para dichos objetos. La clase, por recordar lo que se comentaba anteriormente, la especificación de la clase constituye la definición de la plantilla con la que voy a poder crear objetos que van a tener como estado característico el valor asignado a cada uno de los atributos que hayamos especificado en la clase. Atributos que van a poder ser modificados de acuerdo con las operaciones que vayan también especificadas en la clase correspondiente.



Bien recuerden también, lo vamos a ver en la transparencia siguiente, como tenemos tenemos tres niveles en todo este discurso que estamos empleando, la parte más conceptual donde inicialmente pues bueno en el primer lugar tenemos que ser capaces de identificar las clases que son relevantes en nuestro universo de discurso, nuestro sistema analizado. Una vez analizamos ese concepto, pues somos capaces de definir atributos, de definir operaciones y dentro de esa fase de análisis estamos concretizando ese nivel conceptual a través de la caracterización y la especificación más precisa de las propiedades que definen el concepto que inicialmente conforma la clase que estamos queriendo especificar.



Finalmente hay un nivel más próximo a la máquina por eso hablamos de bajo nivel, un nivel mas bajo a hablamos de nivel alto cuando nos movemos en el espacio del problema, a nivel conceptual, hablamos de nivel de bajo nivel cuando estamos en el ámbito de la implementación, del espacio de la solución. Bien, en este caso ya estoy, como ven estamos asociándole a cada tributo su visibilidad, algunos valores por defecto y estamos caracterizando operaciones que se van a convertir en métodos, recuerden que la diferencia entre la operación y el método estaba en que la operación es la declaración de la de una de una operación, el método la implementación. Con la operación especifico la funcionalidad de lo que va a ser ese granulo de funcionalidad específica, con a nivel de implementación habitualmente bajamos al concepto de método, donde ya materializo y preciso cuál va a ser la implementación con la que esa operación va a ser ejecutada. Bien la clase, tiene un nombre único dentro de su contenedor, el contenedor es el el ámbito de especificación de una clase, habitualmente en UML hablamos de paquetes cuando yo tengo un diagrama de clases que tiene tres, cuatro, cinco clases, pues no necesito definir estos contenedores porque en un diagrama único puedo visualizar el conjunto de mi diagrama de clases, cuando tenemos diagramas de clases que puedan tener centenares de clases lógicamente es necesario abordar esa complejidad a través de la creación de contenedores, que en el caso de UML se llaman paquetes y que me permiten definir subsistemas en el que agrupo clases comunes dentro de paquetes comunes a través de los cuales ya me puedo referir a una clase dentro de su paquete correspondiente.

Cuando esto ocurre, para hacer referencia una clase que está en un paquete determinado, sea el mismo el que está definida u otro, se usa una notación especial en el que aparece el nombre del paquete identificando en qué contra qué contenedor nos referimos y posteriormente el nombre de la clase que queremos, a la que queremos referirnos dentro de ese paquete. Cuando veamos los diagramas de paquetes veremos en concreto como manipular este este asunto de abordar la complejidad de diagramas de clases muy extensos a través de la especificación de estos contenedores que son los paquetes.

Bien, las clases tienen una visibilidad con respecto a su contenedor, tiene una multiplicidad que me permite, me permite expresar cuantas instancias de la clase pueden existir cuando esa información es relevante y también puede ser activa, una clase activa es aquella clase en la que puedo recibir eventos sin necesidad de que ninguna de las operaciones que tengo especificadas en la asignatura de la clase se invocada, ese el concepto de clase activa, la posibilidad de activar espontáneamente o

recibir eventos que no tienen que ir relacionados con la invocación de operaciones realizadas en la clase.

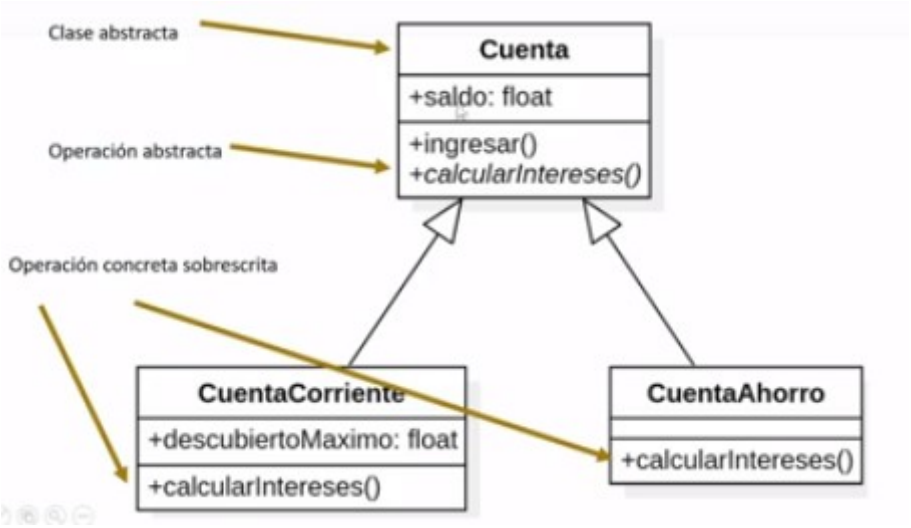
Clases abstractas

Antes de empezar con las clases abstractas me gustaría recordar el concepto que vimos en la sesión anterior de clase como constructor fundamental en el ámbito de UML y del diagrama de clases de UML que veremos con todo detalle y que se definía como un conjunto de objetos que comparten estructura y comportamiento. Estructura a través de atributos y asociaciones y comportamiento a través de las correspondientes operaciones con las que puedo cambiar su valor. Bien, en el caso en el caso del comportamiento decíamos también que una vez especificadas las operaciones esas piezas individuales de comportamiento pueden ser invocadas para ser activadas y por lo tanto ejecutadas y poder originar el cambio de estado de un objeto determinado de la clase a través del cambio del atributo que se ve afectado por la ejecución de una operación. Insistíamos en como las clases me permiten especificar ese ese molde genérico a partir del cual instancio la clase generando objetos individuales y bueno de esa forma también dejabamos claro que la clase dicta de manera natural la estructura y el comportamiento de las instancias que puedo obtener en la clase y que las instancias contenían siempre localmente el conjunto de valores asociados los atributos que se, que iban asociados a ella lo que caracterizaba es la noción de estado del objeto.

Bien, ¿cuál es entonces la necesidad de hablar de clases abstractas en este ámbito? Bien ocurre que en algunas ocasiones las clases no son instanciables directamente, sabemos que tenemos ciertas operaciones en una clase pero esa descripción es incompleta, porque sabemos que existe, existe la propiedad pero no podemos ni sabemos instanciarla todavía y ahí es donde está la el quid de la cuestión al hablar de clases abstractas. El objetivo fundamental de una clase abstracta es la especialización, porque yo voy a crear en la clase abstracta la maqueta esencial de lo que van a ser las propiedades de esa clase, voy a declarar sus propiedades genéricas pero no voy a entrar a hablar sobre el cómo esas propiedades se van a plasmar en algo concreto a nivel de implementación.

Es decir declaró el que, que es lo que esa clase tiene que hacer pero no declaró el como esa propiedades pues va a ser resuelta. Bien es por ello que una clase, las clases que hasta ahora habíamos visto que son concretas no pueden tener operaciones abstractas, esto querría decir que una clase concreta tiene una operación cuya implementación desconozco, si es el caso entonces la clase ya no sería concreta sería abstracta. Una clase abstracta si que puede tener operaciones concretas, porque en una clase abstracta yo puedo incluir operaciones cuya implementación desconozco o no he especificado todavía porque no porque no he querido, porque no me interesaba, por la razón que sea junto con otras operaciones que sí que he, cuya implementación sí que he especificado pues porque la conozco o porque era de interés en un contexto de modelado determinado el poder hacerlo. Las operaciones abstractas por tanto son implementadas por subclases de dicha clase y por eso asociado a una clase abstracta siempre hay una jerarquía de especialización como comentabamos. Es habitual también en uml que una una clase abstracta aparezca en cursiva como mecanismo de ayuda su identificación de manera simple.

Bien en este ejemplo podemos materializar las ideas que estabamos comentando, fíjense como tenemos una clase, una clase cuenta:



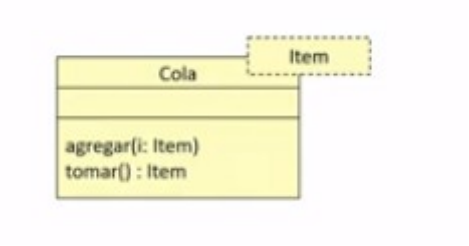
Con un atributo público y dos operaciones públicas, ingresar y calcular intereses pero imaginemos que en este caso yo no, no con, no no tengo especificada la implementación de estas operaciones. Por lo tanto la clase es abstracta porque tengo un conjunto de operaciones cuya implementación todavía no he especificado. A través de esta relación que veremos que son relaciones especialización, la clase abstracta queda concretizada, es decir, esta clase cuenta corriente y esta clase cuenta de ahorro son especializaciones de la clase cuenta, de la clase abstracta cuenta que se convierte en convierten en clases concretas porque en este caso lo que voy a indicar es que tanto calcular intereses en la cuenta corriente como en la cuenta de ahorro van a ser implementadas, es decir, se va a determinar cuál va a ser la forma de materializarlas en ejecución en esta clase y en esta clase. Y es por eso que hablamos de operación concreta sobreescrita y hablamos de clases concretas, como las clases abstractas se convierten en clases concretas en el momento en el que son especializadas y la implementación de las operaciones que aparecía sin acla, sin clarificarse en la clase abstracta es hecha en la clase concreta.

¿Cuándo se utilizan las clases abstractas?, en situaciones en las que conocemos cierta o hemos identificado propiedades cuyo detalle de implementación en un momento dado todavía desconocemos. En ese caso declaramos las propiedades, en este caso las operaciones, como operaciones abstractas indicando que posteriormente serán implementadas cuando se realice ese ejercicio, por llamarlo así, de especialización y de concretización de las operaciones abstractas en operaciones concretas sobreescritas.

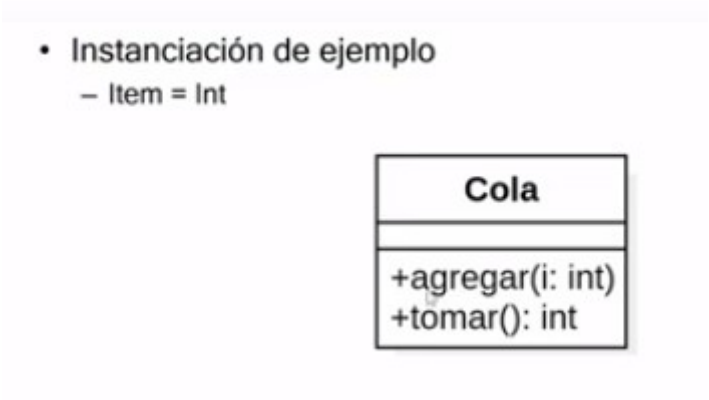
Clases parametrizadas y estereotipadas

Empezamos con las clases parametrizadas que también se conocen como clases plantilla o clases template y que me permiten definir clases con parámetros, es decir, voy a tener un descriptor de una clase con parámetros formales que no especificó inicialmente y que posteriormente voy a poder instanciar para tener familias de clases cada una de ellas asociada a un valor concreto del parámetro que estamos especificando en la clase parametrizada.

Bien, por ejemplo aquí vemos una clase en la que si prescindimos de este parámetro, que aparece como un rectángulo pequeño con líneas de puntos en la parte superior derecha, si prescindieramos de él tendríamos una clase cola en la que yo tendría, pues bueno tengo aquí agregar, puedo añadir elementos a la cola o puedo tomar un elemento de la cola.



Fíjense cuando hablo de colas en general colas que pueden ser de distintos ítems, parametrizado y la forma de poder expresar esta parametrización pues es la adición de este parámetro a través de este rectángulo de líneas punteadas del que estábamos hablando, que me permite usar ese parámetro a la hora de definir argumentos de las operaciones. La agregación agregar se va a realizar sobre items y tomar de la cola se va a realizar sobre item y, bueno, en la medida en la que especifiquemos a que ítem me refiero, la clase parametrizada se convertirá en una clase normal entre comillas cuando este ítem tenga un valor asignado. Bien esto mismo es lo que ocurre aquí, tengo una clase parametrizada y si yo por ejemplo quiero referirme a colas de enteros, asigno el parámetro el valor del parámetro item a un entero y a partir de ahí convierto los items en enteros y tengo colas con dos operaciones, aquí no tengo, ya hemos visto que en la parte superior tenemos atributos, en este caso tenemos el nombre de la clase, no tenemos atributos tenemos operaciones que me permiten agregar enteros a esta cola de enteros que parametrizado o tomar enteros de la cola en cuestión.



Esa expresividad es la que consigo utilizar dentro de UML a través del concepto de clase parametrizada.

Finalmente la noción de clase estereotipada o de estereotipo que si recuerdan cuando estábamos viendo propiedades asociadas a atributos u operaciones ya aparecía como primer componente en la sintaxis de atributos y operaciones ese menor menor estereotipo mayor mayor hablamos de estereotipos, ya adelantábamos que el estereotipo me permite introducir primitivas o elementos de modelado que no existen o no están proporcionados inicialmente por el estándar, es un elemento entonces de extensión del propio UML muy interesante porque permite que yo pueda incorporar nuevos elementos de modelado no existentes cuando necesite expresar conceptos que no estén presentes en el estándar tal y como ha sido, ha sido propuesto. Esta extensión del vocabulario de UML que los estereotipos me permiten realizar, hace posible que yo obtenga nuevos conceptos a partir de la derivación de los existentes con las características asociadas a un problema en particular que me haga necesaria la adición o la incorporación de ese nuevo elemento de modelado.

Ejemplos de estereotipos pueden haber tantos como conceptos yo quiera expresar en un modelo y no tenga ese concepto en el diagrama de UML que yo esté utilizando. Imaginemos que en un diagrama de clases en el que yo tengo clases y relación entre clases yo quisiera distinguir entre tipos o entre clases de numeración o entre clases interfaz, cualquier concepto de estas características que yo quisiera añadir lo tendría que estereotipar. Estereotipar va a querer es decir declarar en la clase el estereotipo a partir del cual yo introduzco esa nueva redefinición del elemento de modelado como estereotipo adicional que a partir de ese momento voy a poder usar como nuevo elemento de modelado de mi diagrama. Un ejemplo bastante clarificador de esta situación puede ir asociado a un patrón no muy conocido arquitecturas de software que es el model, el patrón model view controller, modelo o entidad vista controlador. Imaginemos que yo quisiera en mis diagramas de clases determinar cuando una clase es un modelo o una entidad, cuando es una vista relacionada con la interacción o cuando es un controlador, bien, entonces estas clases del modelo, el modelo o entidad controlador o fachada las podría o las puedo introducir en mi entorno de modelado a través de la noción de estereotipo, estereotipando una clase cuenta que en principio sería una cuenta normal pues como ven aquí con el nombre entidad. Esta, este estereotipo me está indicando que en este caso yo estoy dentro, dentro del model view controller estoy un poco refiriéndome a una clase de tipo modelo o de tipo entidad. Esto lo puedo luego representar también con algún icono adicional, alternativo que yo quiero usar mis diagramas y bueno un poco, es lo mismo. Aquí tengo, como ven tres formas distintas de representar un concepto que inicialmente no estaría en mi diagrama de clases que sería el concepto, la clase, entity que queda de alguna forma caracterizada por este, por este estereotipo. Bien lo mismo ocurre cuando estábamos hablando aquí del del model view controller, yo tengo un controlador pues bueno, esto sería una clase controladora, una clase controladora que quiero que exista de manera específica porque hace referencia a coordinación de diferentes operaciones, transacciones, sincronizaciones. etcétera. A todo lo que va relacionado con el control del flujo de operaciones o acciones en una determinada clase. En este caso estamos creando una clase control y esto es el estereotipo con el que la clase normal, clasificador que proporciona UML de manera estándar, se convierte en un una clase control a través de la introducción de este estereotipo y como ven pues bueno puedo también representar a través de estas representaciones alternativas que yo puedo incluir en mi entorno de modelado. Y bueno lo mismo con lo que sería la fachada o la vista o el boundary como lo llamemos cuando yo quiera referirme a clases para las que quiero una semántica asociada a la especificación del tipo de interacción, del tipo interfaz de usuario, del tipo del tipo de panel que quiero usar, del tipo de visualización que quiero especificar para la realización de una determinada aplicación.

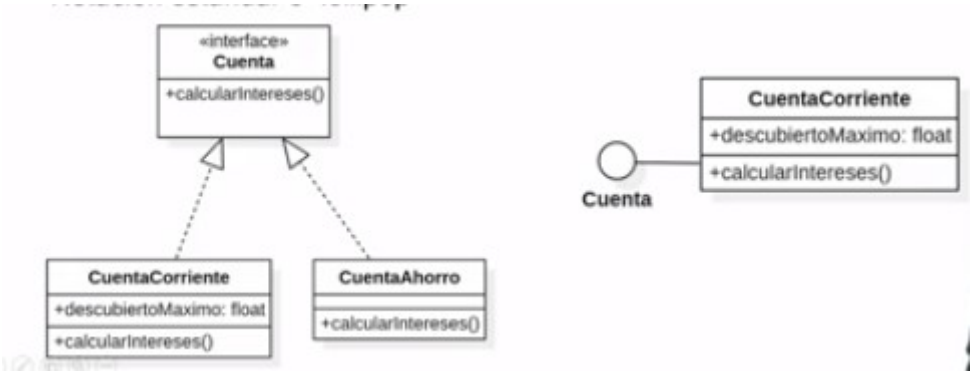
En definitiva, con las clases estereotipadas UML proporciona un mecanismo muy potente de extensión de las capacidades de modelado que el estándar proporciona, con la posibilidad de que en cualquier entorno de modelado, un un modelador pueda estereotipar, es decir, pueda incluir a través de esta noción de clase estereotipada, nuevas primitivas conceptuales, nuevos elementos de modelado, nuevos conceptos que formen parte del arsenal expresivo con el que yo construya los diagramas que quiera construir.

Interfaces

Una interfaz me permite agrupar un conjunto de operaciones, una colección de operaciones con las que voy a especificar los servicios que una clase o componente oferta. Realmente esa agrupación,

esa colección de operaciones asociada cualquier clasificador me permite determinar, como si fuera un contrato, cuáles son las propiedades en este caso ese conjunto de operaciones, que otro clasificador va a tener que cumplir cuando interaccione con el clasificador en el que esa interfaz está siendo definida.

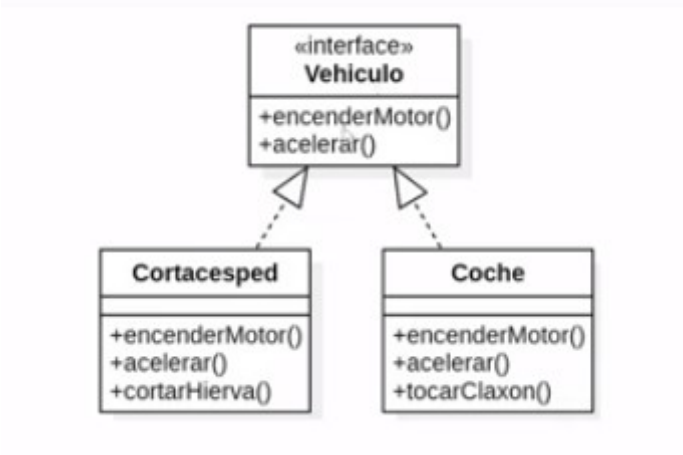
El concepto de interfaz es realmente relevante porque permite que yo agrupe ese conjunto de operaciones, ese contrato que un clasificador va a poder ofertar a otro, a través de dotarle de un nombre y de la caracterización del comportamiento del elemento que el interfaz en concreto, que es el conjunto de operaciones, va a tener que satisfacer. Ese conjunto de operaciones e interfaz no poseen implementación, tienen visibilidad pública, no contienen atributos y van a poder ser generalizables, es decir, van a poder pertenecer a jerarquías de generalización uno más clases o componentes pueden realizar materializar un interfaz.



En este ejemplo que vemos en la figura pues vemos que por ejemplo tengo un interfaz, es una cuenta, esa interfaz determina una operación, en este caso es una única operación, un conjunto de una única operación que es calcular intereses y podemos ver a través de estas de estas relaciones que luego veremos, relaciones de dependencia, como estas dos clases cuenta corriente y cuenta de ahorro materializan esa interfaz, ¿y qué quiere decir que materializan esa interfaz? Pues bueno que cumplen con la restricción de darle, operacionalizar la propiedad de calcular intereses, la vemos aquí en cuenta corriente, la vemos aquí en cuenta el ahorro y bueno la cuenta corriente pues puede a su vez también introducir propiedades adicionales como en este caso el atributo descubierto máximo que es de tipo float. El hecho de que una clase pueda admitir muchas interfaces, pues hace que yo pueda asociar distintos tipos de contratos de conjuntos de operaciones a una a una clase en concreto. En este ejemplo podemos ver también como la notación estándar puede usar o los estereotipos que veíamos en sesiones anteriores, aquí vemos como esta clase, en principio este clasificador de uml, con el estereotipo interface lo estamos declarando como un interfaz o vemos aquí también como a través de esta notación lollipop más gráfica pues bueno tenemos la clase cuenta corriente que materializa o implementa el interfaz cuenta que es un poco el mismo tipo de declaración pero hecha con una notación diferente.

Algunos de ustedes habrán adivinado que existe una singularidad conceptual en el concepto de interfaz y el de clase abstracta. ¿Por qué?, pues porque no tenemos atributos, no tenemos métodos igual que nos ocurría por las clases abstractas y todas las operaciones son abstractas, pero fíjense que hay una diferencia, es una diferencia importante, la diferencia estriba en que mientras que las clases que heredan de una clase abstracta sí están relacionadas semánticametne entre ellas, no existe

ese concepto, esa relación, esa comunalidad entre diferentes clases que puedan implementar una interfaz. De hecho con el interfaz recuerden que estabamos materializando un contrato un contrato que determina que una serie de operaciones tienen que ser satisfechas por la clase que hace uso de la interfaz sea cual sea esa materialización o esa implementación que se va a hacer de dicho conjunto de operaciones que conforman el interfaz. De hecho las implementaciones pueden ser totalmente distintas, las clases que están que están usando ese interfaz puede no tener nada que ver entre sí semánticamente más allá del hecho de estar usando es el mismo conjunto de operaciones que el interfaz proporciona.



Aquí tenemos un ejemplo con el que se puede entender lo que estabamos explicando, fíjense aquí tenemos una una interfaz el vehículo y ese vehículo como clase de operaciones que ofrecen ese contrato, que el interfaz está especificando, incluye estas dos operaciones de encender motor, un vehículo debe poder encender el motor y de poder acelerar independientemente de que luego las clases que materialicen esa interfaz como vemos aquí pues sean dos vehículos que realmente son totalmente diferentes y que no tiene nada que ver entre sí como es la clase corta césped y la clase coche.

¿Por qué ambas clases materializan el interfaz? Bien pues porque como verán tanto encender motor como acelerar están en las dos clases, tanto el corta césped ver como el coche, vistos como una clase, materializan, definen esas estas dos operaciones que toman del interfaz, por lo tanto son compatibles con el interfaz independientemente de cada una de ellas aporte su propia particularidad, en este caso la clase corta césped va a aportar una nueva operación que no estaba en el contrato original, que es la de cortar hierba, mientras que el coche va a aportar la clase coche va a aportar otra operación adicional que no estaba en el contrato inicial que conformaban interfaz vehículo que es el de tocar claxon.

Definición de relación

Hasta ahora hemos visto clasificadores especialmente clases, como ejemplo básico de clasificador que eran independientes y en la realidad un sistema no tiene sólo componentes independientes, cuando veamos los diagramas de clases, pues las clases no van a ser entes independientes sino que van a aparecer relacionadas entre sí, conectadas entre sí para que el funcionamiento del sistema incluya tanto lo que son clasificadores como lo que son relaciones entre clasificadores.

Estos tipos posibles de relaciones son los que vamos a ver en este tema. De hecho en el dominio del problema y también el dominio de la solución nunca vamos a tener clases aisladas más que en casos muy elementales, muy simples que no son reales porque siempre vamos a tener clases que de una forma u otra se relacionan con otras clases para caracterizar la semántica, el significado del sistema que esté bajo estudio.

Por ejemplo una persona aquí tenemos un clasificador que seria una clase, la clase persona, la clase apartamento, pues una persona vive en un apartamento por lo tanto existe una relación, vive en, entre las clases personas y apartamento. Análogamente una persona, una clase, trabaja en una empresa, es decir, entre las clases persona y empresa pues existirá una relación que es la relación trabaja en, o de una forma también en un dominio distinto, pues una rosa es un tipo de flor. Rosa puede ser una clase, flor otra clase y tipo de, un tipo de es una relación que va a haber entre ambos tipos de clase. O una rueda es una parte de un coche y hablamos de una relación también más asociada a la composición, una rueda forma parte de un coche, por lo tanto entre las clases rueda y coche tenemos esta relación.

Hablamos de clases porque los clasificadores en los que nos estamos centrando en este curso básicamente son las clases cuya estructura completa veremos en los diagramas de clases que son el objetivo final de este curso.

Algunos ejemplos de relaciones podemos o podemos ver de manera muy elemental en esta figura tenemos tres clases, personas, organizaciones, asignaturas y relaciones, relaciones entre la persona y la organización y la persona y la asignatura. Conceptos importantes asociados a todo este entramado de clases, en primer lugar cuando hablamos a nivel de clases y aquí fíjense que tenemos clases, persona, organización, asignatura, cuando hablamos de clases instanciar una clase es obtener un objeto de esa clase, es decir, instanciar la clase persona es obtener un objeto de la clase persona.

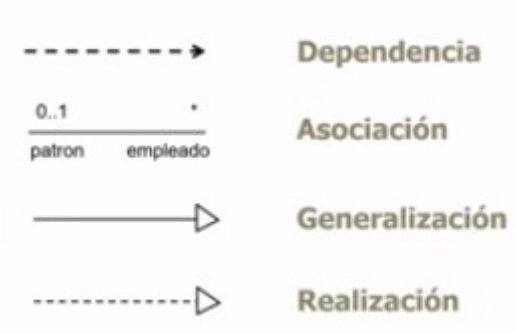
Aquí por ejemplo tendríamos tres objetos de la clase persona que son los objetos de nombre Pepe, Luis y María. Lo mismo ocurriría con la organización, cuando instanciamos la clase organización tenemos objetos de la clase organización. Recuerden, clases que se instancian en objetos. Esos objetos serían USAL, ENUSA, ayuntamiento y bueno, en este caso estos tres, en otro caso pues serían todos aquellos que aparecieran en mi universo de discurso o dominio real del problema que estoy analizando. Lo mismo ocurre con la asignatura, clase asignatura, instanciación de la clase signatura en objetos en este caso tres objetos que referencian tres asignaturas que serían ingeniería del software, redes y POO.

Bien, a nivel de relaciones ocurre lo mismo pero desde o con la utilización de otros conceptos. Antes hablamos de clases cuando hablamos de relaciones entre clases que son estas relaciones que vemos aquí por ejemplo entre persona y organización y persona y asignatura, estamos hablando a nivel conceptual de relaciones entre clases que también se pueden estas relaciones, también se pueden instanciar y al instanciar una relación tenemos un enlace. Recuerden que al instanciar clases teníamos objetos, al instanciar relaciones tenemos enlaces y por lo tanto un enlace es una relación instanciada entre dos objetos concretos.

Por ejemplo, aquí tenemos un enlace en particular entre Pepe y USAL, que me indica que Pepe está participando o pertenece a la organización USAL. Los objetos son Pepe y USAL, el enlace es esta

flechita que me determina el tipo de relación. Y vamos directamente a la parte más importante del del tema que que es el de responder a la pregunta de qué relaciones puedo tener entre clasificadores.

Determinar qué relaciones existen entre clasificadores me determina qué tipo de conexiones semánticas, un modelo que yo construya con la notación que esté seleccionando va a poder, va a poder expresar y en este caso, en el caso de UML, pues tenemos cuatro tipos de relaciones esenciales.



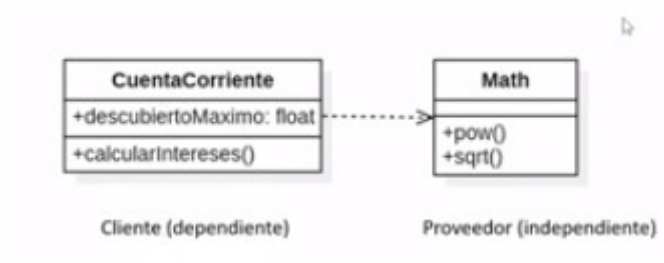
Las cuatro las vemos aquí en la figura

- relación de dependencia. Una relación de dependencia cuya semántica fundamental es la de expresar simplemente que existe una relación de que un objeto de alguna forma depende de otro para poder realizar alguna de las de las operaciones o propiedades que tenga asignado en su definición. Dependencia por lo tanto se refiere como su nombre indica a esa situación en la que una clase depende de otra y esa dependencia la expreso a través de este tipo de notación.
- La asociación es otro tipo de relación más específica en la que yo tengo dos clases que tienen una relación en la que, con la que quedan vinculadas durante sus existencias a través de una semántica en la que cada una de las clases juega un rol específico en la participación de esa asociación. Por ejemplo si yo tuviera una asociación entre una clase patrón o jefe y una clase empleado, aquí estaría determinando pues esta clase, patrón es el responsable de que objetos de la clase empleado mientras que para cada objeto de la clase empleado determinaría cual es la instancia, el objeto de la clase patrón que actúa como responsable de ese empleado, es decir frente a esa dependencia genérica, la asociación establece una relación semántica más fuerte entre dos clases que quedan relacionadas como consecuencia de que sus ciclos de vida permanecen ligados como consecuencia de la existencia de esa asociación.
- La generalización hace referencia al concepto de herencia, básicamente permite especializar especializar una clase padre en una clase hija con lo que voy a poder re-usar propiedades que ya tengo especificadas en la clase padre y con eso agilizar lo que es el proceso de especificación de propiedades en redes de herencia en las que las propiedades se van heredando sin que yo las tenga que repetir en los clases en las clases hijas si ya aparecen especificadas claramente en las clases padre.
- Realización o de materialización la veíamos en la lección anterior cuando hablábamos de interfaces, hace referencia al hecho de que yo cuando tengo una red, una relación de

realización entre dos clases estoy indicando que materializo, realizo, implemento alguna característica o propiedad de la clase que actúa como, como origen de esa relación en la clase que actúa como destinataria. Esa realización por lo tanto me va a permitir expresar situaciones en las que como veíamos en el caso del interfaz, tengo un interfaz que se va a realizar en la clase a partir de la cual esta relación indica que esas propiedades del interfaz van a ser materializadas.

Relación de dependencia

La relación de dependencia se establece entre dos elementos y me me permite determinar o especificar la situación en la que un cambio en un elemento que actúa como el proveedor va a afectar o a proveer la información necesaria para el otro elemento que actúa como cliente de esta relación de dependencia.



Como ven, tengo aquí un rol cliente, un rol proveedor y la relación de dependencia me muestra la relación existente entre dicho cliente y dicho proveedor siempre referida a un servicio que ese proveedor me proporcione que ese cliente va a tener que utilizar.

En este ejemplo podemos ver de una manera muy elemental pues como tengo una clase, una clase cuenta corriente que actúa como cliente, una clase, en este caso un paquete matemático con una serie de funciones, de facilidades a través de paquetes matemáticos ofertados por la clase, que actúa esta clase Math como proveedor. El cliente va a ser dependiente del proveedor, porque aquí lo que estamos indicando es que en la clase cuenta corriente en la que tenemos, por ejemplo, pues una operación que es calcular intereses, para el cálculo de sus intereses voy a utilizar funciones, facilidades que me proporciona mi proveedor que en este caso es la la clase Math, que me estaría proporcionando ese conjunto, esa librería de funciones matemáticas que necesito para poder definir la operación que tiene la cuenta corriente. Relación como ven de dependencia indicando que la cuenta corriente depende de este proveedor que es la la clase que represento como Math para poder ejecutar su servicio correctamente.

¿Qué tipos de relaciones de dependencia contempla UML por defecto? Bien vamos a usar los estereotipos que vimos que vimos en temas anteriores y vamos a ver como dependiendo del estereotipo con el que yo etiquete la relación, aquí la estamos estamos por ejemplo estereotipando esta relación de dependencia a través de esta palabra reservada, este estereotipo, use indicando que la clase cuenta corriente usa la clase Math, es un poco el último de los tipos que vemos aquí y que es el más usado porque es el que se asume por defecto, cuando yo no indico nada es porque estoy utilizando una relación de dependencia de tipo uso que me expresa que la clase origen está usando

la clase destino para realizar algún tipo de o materializar algún tipo de propiedad que está incluida en la clase que actúa como cliente.

Pero hay otras, tenemos la clase bind que normalmente va asociada clases plantilla y que me permite instanciar un tipo de parámetro que una clase parametrizada va a usar o el estereotipo derive que me permite me permite expresar que tenemos una cuenta origen que va a poder ser derivada a partir de una clase destino, cuando tengo esta relación, este estereotipo entre dos clases esta relación de dependencia explica o me permite especificar que la clase origen puede ser derivada a partir de la clase destino.

En el caso de estereotipo permit, lo que me indica es que la clase origen tiene privilegios especiales de visibilidad sobre la clase destino, siempre los estereotipos van en la dirección cliente destino o cliente proveedor.

El estereotipo instanceOf me indica que la clase origen es instancia de objetos de la clase destino, me instancia objetos de la que es destino o me los instancia directamente o instancia de o es creador de instancias de la clase destino si lo que indico o lo que uso es el estereotipo de instantiate. Finalmente el estereotipo del refinamiento, el refine que lo que me indica es que esta clase, la clase origen, esta refinando, tiene más información, información más detallada sobre propiedades de la clase origen que la que está asociada a través de la relación de dependencia la clase destino a la que yo asocio mi clase origen.

Relación de generalización

Esta relación de generalización es un recurso semántico muy importante en el ámbito de UML que me permite representar situaciones en las que tengo esa relación de "es un tipo de" entre dos elementos de, generalizables, del mismo tipo en un diagrama de clases. Esta relación "es un tipo de" puede aplicarse a clases, puede aplicarse a paquetes o puede aplicarse a otras clases de elementos en los que yo quiera determinar que la clase que actúa como clase hija es un tipo de es un caso particular extendido de la especificación contenida en la clase que actúe como padre.

Los propósitos de la generalización son dos básicamente,

- en primer lugar definir las condiciones bajo las cuales se puede usar una instancia de una clase, de hecho yo puedo tener una jerarquía de generalización en la que las propiedades más generales de los objetos de una clase las declaro en la clase padre y conforme voy definiendo a través de relaciones de generalización clases hijas, voy extendiendo la definición de dichas clases con nuevas propiedades emergentes que son consecuencia de esa de esa utilización de la jerarquía de generalización, también referida a veces como relaciones especialización o de herencia que me permiten, como vemos en el segundo propósito
- describir incrementalmente un elemento compartiendo las descripciones que vienen de sus ancestros pero enriquecidas convenientemente a través de las nuevas propiedades, esas propiedades que llamamos emergentes, y que van van extendiendo de una manera semánticamente muy rica el conjunto de propiedades que tiene la clase que actúa como como ancestro de esa jerarquía de generalización.

En el caso de un modelo de diseño de implementación, estas relaciones de generalización son las que se llaman habitualmente relaciones de herencia, es una forma intuitiva, intuitivamente muy clara de ver este tipo de relaciones. Las subclases heredan propiedades de sus clases padre y por lo tanto atributos, operaciones y relaciones de la clase padre van a poder estar disponibles en las clases hijas al mismo tiempo que estas clases hijas van a poder extender este conjunto de propiedades pues añadiendo nuevos atributos, añadiendo nuevas operaciones o añadiendo nuevas relaciones como consecuencia de que la clase padre está viendo, está siendo vista como una clase hija está por lo tanto viendo extendido su comportamiento, su estructura o su especificación en general.

Es importante destacar como se cumple el principio de sustitución cuando hablamos de relaciones de generalización y que queremos decir con esto, pues que una una instancia de un elemento más específico, dentro de una jerarquía de generalización, puede ser utilizada donde se permita el elemento más general o dicho otra forma un objeto de la clase hija siempre va a poder ser visto como un objeto de la clase padre, tiene que mantener el comportamiento del padre y tiene que ser compatible, por lo tanto, con todo aquello que haya sido específico en el en el padre. Dentro de ese concepto de jerarquía en el que no es conveniente que las jerarquías sean muy largas, una recomendación en UML es no pasar de cinco niveles en la jerarquía, fíjense como en este ejemplo en el que tenemos una jerarquía simple la que soy dos niveles, la clase padre cuenta y las clases hijas cuenta corriente, cuenta de ahorro que son la consecuencia de utilizar esa relación de generalización tanto para ver la cuenta corriente como una especialización de la cuenta como para ver la cuenta de ahorro como una especialización de la cuenta. Utilizar los términos generalización, especialización es compatible, esta clase ancestro la clase padre es la generalización de las dos clases, que actúan como hijas, mientras que cada clase hija puede ser vista como la especialización de la clase padre independientemente de que al uso de esta relación le llamemos relación de generalización.

Bien como ven en la clase hija tenemos todas las propiedades implícitamente definidas que están heredadas de la clase padre, por lo tanto si en la clase padre tenemos un atributo que es el saldo, en la clase hija, en la cuenta corriente, tendría el descubierto máximo que es un nuevo atributo introducido en la clase descendiente, pero a la vez tendríamos también como atributo heredado el saldo, por lo tanto, la clase cuenta corriente tendría dos atributos, el atributo que se define en la propia clase hija, el descubierto máximo y el atributo saldo que es un atributo que viene heredado de la clase padre. De la misma forma tenemos una operación calcular intereses que es la operación era del padre y que al repetir el nombre aquí pues podemos estar indicando que es una operación cuya materialización, cuya implementación puede haber sido reescrita en la clase hija.

Es posible, aunque no es recomendable en el ámbito de lo que UML sugiere, la herencia múltiple o generalización múltiple, ¿a qué nos referimos con este término? Bien a esta herencia múltiple o repetida en la que como ven pues una persona puede ser especializada en profesor y estudiante y a la vez un becario puede ser a la vez una clase hija de profesor y de estudiante. Esta herencia múltiple o herencia repetida puede tener un problema que aparece reflejado en la parte izquierda, que es un problema de colisiones de nombres cuando de diferentes clases padres o súperclases heredo una misma propiedad con el mismo nombre pero con dos características diferentes, si tengo por ejemplo un nombre en la clase profesor y un nombre la clase alumno y la clase becario es a la vez una especialización de profesor y alumno, en la clase becario voy a hereda a heredar dos

propiedades con el mismo nombre, el nombre del profesor y el nombre del alumno. El resolver esta colisión de nombres que provienen de clases padres diferentes es un asunto que es importante tratar con cuidado para evitar los errores que puedan derivarse de dicho problema de colisión de nombres.

En este ejemplo podemos ver con un poquito, un ejemplo un poquito más elaborado, pues como una clase una clase padre figura a través de una de tres relaciones de generalización se puede especializar en figuras de dimensión cero, de dimensión uno, de dimensión dos. Las figuras de dimensión cero son se especializan en puntos, las de dimensión uno con una doble relación de generalización en arcos o líneas y las de dimensión dos también con una doble relación de generalización en círculos o polígonos. Bien a la pregunta de cuántos atributos tiene la clase dimensión uno, la respuesta correcta sería cinco, no uno, tiene un atributo propio de esta clase que es la orientación más los cuatro atributos heredados de la clase padre, uno más cuatro cinco atributos, de la misma forma que el número de operaciones de la clase dimensión uno serían cinco, la operación escalado que es una operación definida en la propia clase como propiedad emergente, más las cuatro operaciones que vienen heredadas por la clase padre, que por lo tanto la clase hija toma como propias a través de la semántica de esa relación de generalización y de la herencia que a ella va asociada. Del mismo modo, en la clase dimensión dos, tendríamos pues bueno dos atributos más los cuatro de la clase padre, seis y dos operaciones más los cuatro las cuatro operaciones de la clase padre, seis y bueno la misma, incidiendo en la misma idea si les preguntara cuantas, cuántos atributos tiene la clase dimensión cero si están tentados a responder que ninguno, no es correcta la respuesta, no tiene ninguno emergente pero tiene cuatro que son los cuatro que hereda de la clase figura, por lo tanto la clase dimensión cero tiene cuatro atributos aunque no añada ninguna propiedad emergente adicional en la especificación que en este ejemplo estamos haciendo.

Bien, en el resto, en la segunda aquí tenemos dos, tenemos tres niveles en esta jerarquía todo lo dicho para el nivel primero sería también válido para nivel segundo. En un ejemplo más orientado a sistemas de información organizacionales, pues aquí tenemos una clase persona que se especializa en clase cliente, el cliente es una persona, el cliente es un tipo de persona que va a tener seis atributos, dos atributos que aparecen como consecuencia de que la persona sido especializada en cliente, que es un número de socio y su dirección, más su DNI, su nombre, sus apellidos y su teléfono que como persona sigue teniendo también como atributos cuando lo estamos viendo como una instancia u objeto de la clase cliente.

Relación de asociación

Una de las relaciones más expresivas y más ricas semánticamente hablando que nos proporciona UML. La relación de asociación me permite establecer relaciones estructurales precisas entre objetos con la satisfacción de un requisito básico, es el de necesito conocer, una clase necesita conocer propiedades de otra clase para poder que especificar su comportamiento de forma completa y eso es lo que voy a poder hacer con estas relaciones de asociación a través de la descripción de las conexiones semánticas existentes entre objetos de diferentes clases y a través del establecimiento de las conexiones entre dichos objetos que me permita determinar de una manera precisa el comportamiento y la estructura de cada uno de ellos haciendo uso de aquellas propiedades de las clases asociadas que sean necesarias para que el comportamiento y la estructura sean totalmente descritas y que sean especificadas de forma clara y sin ningún tipo de ambigüedad.

Relacionar una lista de dos o más clasificadores, que es lo que voy a poder hacer con las relaciones de asociación, me va a permitir o le va a permitir a cualquier clase a través del establecimiento de la correspondiente relación de asociación, conocer atributos y operaciones públicas de otra clase que sean necesarias para poder de especificar de una manera correcta y completa el conjunto asocia el comportamiento asociado a la clase que estoy especificando.

Adicionalmente, cada clase va a poder mandar mensajes a la otra y con todo esto vamos a ser capaces de establecer conexiones entre clases con con el con el adición de un concepto nuevo que es el extremo de la asociación, que va a hacer referencia a cada conexión que se establece entre las dos clases que quedan asociadas por la relación de asociación.

Una pregunta esencial cuando hablamos de asociación es la que nos preguntamos en esta en esta imagen ¿cuándo debe existir una asociación entre la clase a y la clase b? Bien la respuesta general a esta pregunta es la de que cuando un objeto de la clase a tenga que saber algo de un objeto de la clase b. Recuerden ese queremos conocer, queremos saber de que hablábamos antes, cuando una clase necesita conocer propiedades de otra clase para completar su propia definición, tenemos que recurrir al establecimiento de estas relaciones de asociación. Esto de que un objeto tenga que saber algo de otro objeto se materializa básicamente en cuatro situaciones o cuatro patrones genéricos.

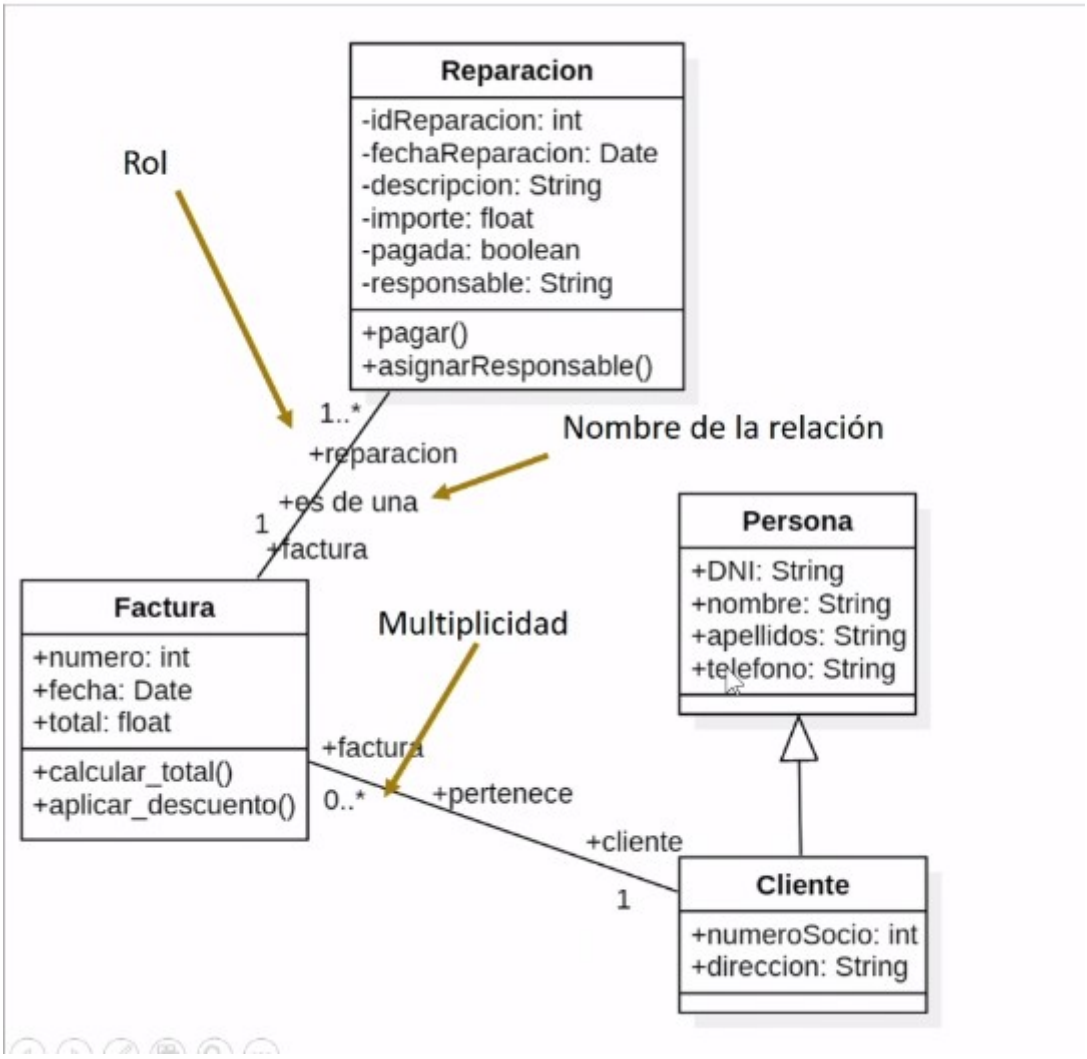
1. Que un objeto de la clase a use un servicio de la clase b,
2. Que un objeto de la clase a pueda ser creador de objetos de la clase b.
3. Que un objeto de la clase a tenga atributos cuyos valores sean objetos de la clase b o colecciones de objetos de la clase b y por lo tanto sea preciso establecer la conexión entre ambas clases para que esos atributos objeto valuados se puedan materializar a través de la correspondiente relación de asociación.
4. Que un objeto de la clase a reciba un mensaje con un objeto de la clase b pasado como argumento y por lo tanto necesite que se establezca esa asociación para poder reconocer a ese objeto de la clase b que está siendo recibido como argumento del mensaje.

Pasemos a los adornos de asociación que me determinan la sintaxis con la que gráficamente vamos a representar las propiedades de la asociación en los diagramas de clases UML. Vamos a ver cómo tenemos seis propiedades esenciales:

1. el nombre de la asociación
2. el símbolo con el que indico si es una agregación, una asociación o una composición
3. el rol o papel que juega la asociación y que va asociado a cada extremo de la asociación, cada extremo va a tener un rol asociado que vamos a ver enseguida como se específica
4. la navegabilidad que me determina si existe direccionalidad en el recorrido de la asociación y que veremos también en un tema específico,
5. la visibilidad que va, me va a permitir determinar las propiedades del tipo de visibilidad que voy a tener sobre el recorrido de la relación de asociación y que veremos también en un tema particular

6. la multiplicidad establecida entre entre los objetos que quedan asociados que recibe el nombre también de cardinalidad en muchas ocasiones y que vamos a ver inmediatamente.

Antes de nada vamos a vamos a usar este ejemplo para recorrer todos sus adornos que hemos introducido en la figura anterior, vamos a empezar a usar este ejemplo en el que tenemos una factura, tres clases, factura, una factura tiene reparaciones, la factura es de un cliente y el cliente es un tipo de persona. Ya ven que tenemos aquí una relación de generalización que vimos en el tema, en el tema previo y nos vamos a centrar acá en las dos relaciones de asociación que hay entre factura y cliente, esta de aquí, y entre factura y reparación.



Empecemos con el nombre de la relación, cada relación puede tener un nombre y ese nombre permite identificarla de una manera precisa en el ámbito de un diagrama de clases completo. En este caso la factura es de una reparación y aquí tenemos algunas reparaciones de una factura, lo podemos leer en las dos direcciones y con este nombre estoy identificando la relación igual que en el otro caso con el nombre pertenece la factura pertenece a un cliente y de esa forma con este nombre pues estoy poniendo en una etiqueta identificativa a la relación de asociación que estamos estableciendo.

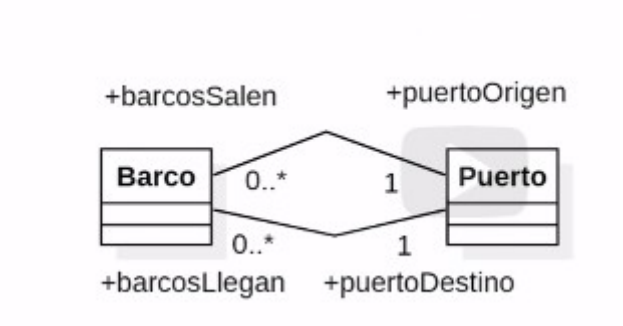
Los roles. Los roles a hemos adelantado que eran las los el papel con el que lo en los extremos de la asociación estamos indicando cómo está actuando cada una de las clases asociadas en el conjunto de la relación de asociación especificada. Bien aquí tenemos que por ejemplo si voy de factura a reparación en la factura tengo el rol reparación que me indica o que me está determinando que por este lado llego a las reparaciones que están asociadas en la factura mientras que si voy en dirección contraria la factura juega el rol de factura sobre la asociación con respecto a la reparación, la reparación es de una factura y aquí está la factura que es el rol que juega esta clase en dicha relación de asociación. En este ejemplo simple pueden ver que los nombres de los roles y los nombres de las clases son los mismos, no tiene porque a ser así siempre, yo puedo poner en el nombre de rol cualquier descripción que me permite identificar de una manera clara qué papel está jugando la clase en el extremo de la asociación al que nos estamos refiriendo.

Si vamos al siguiente caso, fíjense en este caso el rol viene asociado por la facturas de un cliente y le llamo cliente y el cliente tiene facturas y aquí pongo factura un poco pues igual que antes determinando a través del nombre de la clase cuál es el rol que esta clase está jugando dentro de la relación de asociación que estamos definiendo.

Con respecto a la cardinalidad, aunque la vamos a ver enseguida, déjenme adelantar que cuando tenemos tenemos siempre cardinalidad mínima y cardinalidad máxima, este cero asterisco es decir que un cliente puede no tener ninguna factura asociada o tener muchas. Es el significado del asterisco. En el caso del uno punto punto asterisco es lo mismo pero la una factura tiene que tener al menos una reparación.

Si aquí hubiera un cero querría decir una factura podría no incluir ninguna reparación, pero si hablamos de factura tiene sentido asumir que al menos tengo que tener una reparación o muchas, que es lo que indica este asterisco. Cuando sólo tengo un uno, lo que estoy representando es que el valor mínimo es uno y el valor máximo es uno, es decir que una reparación tiene que ser de una y sólo una factura, es lo que implica este uno y que una factura va asociada a uno y solo un cliente de acuerdo con esta cardinalidad que estamos expresando.

De acuerdo con lo dicho podemos ver otro ejemplo



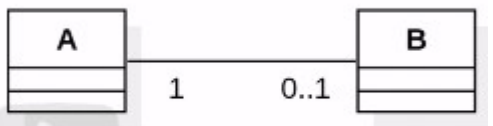
Aquí tenemos dos clases puerto, barco, pero con el mismo uso de las propiedades de multiplicidad o cardinalidad. Un puerto tiene un conjunto de barcos que salen del puerto, ninguno o muchos de un puerto pueden salir cero o muchos barcos, esto quiere decir este cero asterisco mientras que en dirección contraria un barco tiene un puerto origen, este puerto origen es uno y solo uno, esto es lo que indica el uno que vemos aquí de la misma forma que tengo un puerto destino por la otra

relación por la relación de asociación inferior, vemos que un barco puede tener uno y solo un puerto destino mientras que de un puerto pueden o a un puerto pueden llegar cero o muchos barcos. Ya pueden intuir que con este juego de clases y de definición de cardinalidades o multiplicidades tenemos una potencia expresiva muy grande para poder describir sistemas reales cuyo modelo estemos construyendo con un diagrama de clase.

- 0..1. Sin instancia o sólo una (opcional)
- 1. Una y siempre una instancia (obligatorio)
- *. Cero o múltiples instancias (opcional)
- 1..*. Múltiples instancias pero al menos una (obligatorio)

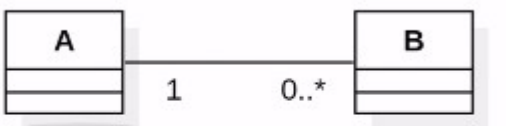
En esta en esta imagen final podemos ver los tipos de multiplicidad que estábamos adelantando. Cero uno, que hace referencia a cardinalidad cero mínima, cardinalidad mínima cero, cardinalidad míxima, máxima uno. Podemos tener ninguna instancia asociada o sólo una como máxima. El uno representaría o sería análogo al uno punto punto uno que sería una y siempre una de forma obligatoria porque es cardinalidad mínima 1, cardinalidad máxima uno también. Este caso representa, es equivalente al cero muchos aunque no aparezca el cero, quiere decir que puede tener ninguna instancia asociada o muchas, el hecho de que no aparezca el cero por facilidad se interpreta como que estamos en un caso de opcionalidad porque si no es opcional si esa multivaluación tiene una cardinalidad mínima de 1, es decir, que al menos uno obligatoriamente, un objeto, tiene que estar participando en la asociación que estamos definiendo, la relación de asociación que estamos definiendo, en este caso redes se representa con el uno punto punto arisco, que quiere decir mínimo, uno máximo muchos.

```
Class A{
Private B b
}

Class B{
Private A a
}
```



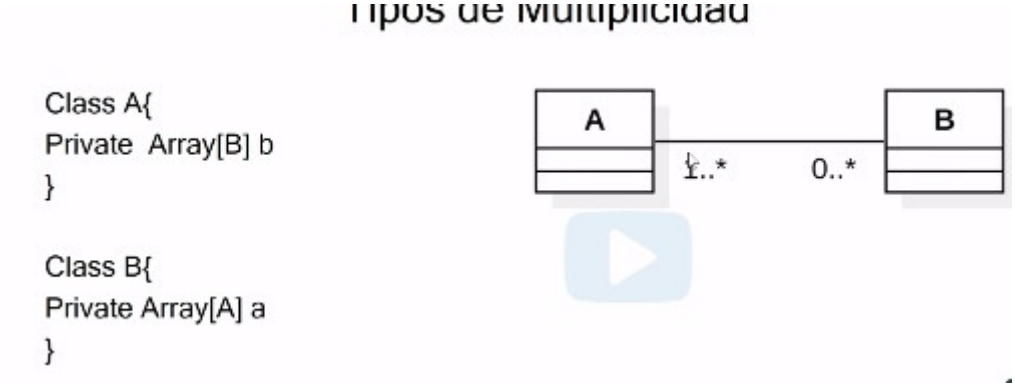
Bien aquí tenemos tipos de multiplicidad de una manera más explícita con una, con un esquema de materialización o implementación, es elemental, una clase a se relaciona con cero o un objeto de la clase b y una clase b se relaciona con algún objeto de la clase a. Esto quiere decir que la clase a va a ser un atributo privado b que es este que podría ser opcional, aquí no lo podemos, no habría que

```
Class A{
Private  Array[B] b
}

Class B{
Private A a
}
```



expresarlo en esta plantilla no queda expresado, pero bueno este puede ser opcional, este no esté de b hacia a es obligatorio, aquí tendríamos un atributo privado de la clase a que sería obligatorio.

De la misma forma si en lugar de ser univaluados son multivaluados pues bueno, aquí tenemos que de a a b hay una multivaluación y por lo tanto la clase a va a tener un atributo b que va a ser pues por ejemplo un array de b para denotar esa colección, ese esa multivaluación que viene representada por este asterisco que me indica que asociado a un objeto a hay muchas b, sin embargo asociado a b hay sólo una a por lo tanto aquí sigo teniendo un atributo a de la clase a que me representa el hecho de que tengo esa univaluación en esa dirección. La clase b tiene un atributo objeto evaluado uno y solo uno que serán un atributo de la clase a que en este caso pues será privado.



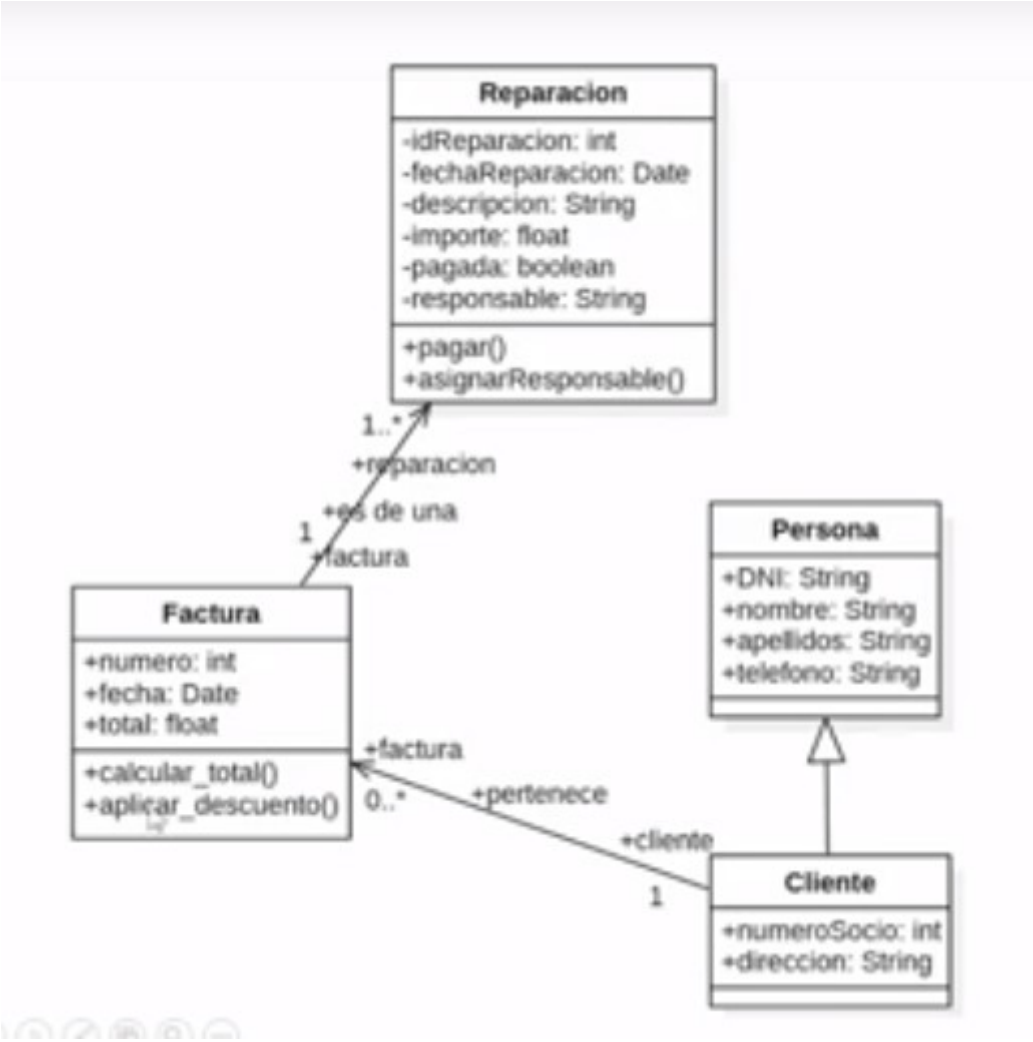
Si tenemos multivaluación en las dos direcciones pueden ver que tengo la colección del atributo b de la clase a, que hace referencia a esta parte de aquí, con una opcionalidad que aquí no se puede representar y el caso contrario, en b tenemos una multivaluación con pertenencia obligatoria por la cardinalidad mínima de 1 a la clase a y bueno, eso lo representaremos también a través un atributo privado en b, que sería este este atributo objeto evaluado a que correspondería un array de objetos de la clase a. Es importante que tengamos presentes como un atributo objeto evaluado una clase se convierte al ser implementado, en el correspondiente atributo de la clase que en el modelo es una relación de asociación. Las relaciones asociación se representan por lo tanto como atributos de la clase cuando esa relación de asociación es implementada utilizando el entorno de desarrollo que seleccionemos.

Navegación

Hasta ahora, hemos visto que la asociación expresa conexiones bidireccionales. Puedo recorrer una asociación de una clase a otra, o viceversa, sin ninguna restricción, pero con la propiedad de navegabilidad vamos a ver cómo existe la posibilidad de limitar la navegación de una asociación a una sola dirección anulando la dirección contraria. De esta forma, vamos a introducir una posibilidad muy relevante desde el punto de vista expresivo, que es la de determinar si una clase de la asociación puede tener conocimiento o no de la otra. Si la navegación está activada, está autorizada, va a poder tener conocimiento de esa otra clase a través del recorrido de la asociación. Si está anulada no va a ser posible.

La flecha con la que vemos, que veremos que se representa esta navegabilidad va a hacer que la dirección, en la que la flecha apunta, pueda ser empleada por la clase de la que emana a la asociación, pero no va a poder ser recorrida en sentido contrario.

Esto lo vamos a indicar a nivel de especificación, tanto de análisis de diseño como a nivel de implementación, con lo cual incluimos esta expresividad en todos los niveles de uso de lo que sería el empleo de las relaciones de asociación en casos reales. Volvamos o recuperemos el ejemplo que usamos en la última sesión, de en la que estudiamos la relación de asociación y fíjense que si buscamos la diferencia ahora con respecto a aquella sesión, la diferencia fundamental está en que las relaciones de asociación aparecen con una flecha.



Esta flecha es la que indica la navegabilidad que estamos estudiando. En el caso de la sesión anterior, una factura y una reparación tenía una asociación que era recorrible en los dos sentidos. En este caso, la factura sólo me va a, la relación de asociación sólo me va a permitir ir de una factura a las reparaciones esa factura, pero no va a estar autorizada la navegación en sentido contrario, es decir, no voy a poder ir de una reparación a la factura en la que o a la que esa reparación está asociada.

Lo mismo ocurre, tal y como está modificado el ejemplo, con la relación de asociación entre cliente factura. Ya no es bidireccional, ya no puedo moverme libremente en una dirección o la contraria, en este caso, desde un cliente voy a poder acceder a las facturas de cliente, pero tal y como lo tengo específicado en este caso, desde la factura no tengo activada la posibilidad de navegar al cliente al que pertenece esa factura por el tipo de modelado que estamos empleando en este ejemplo.

Veamos esto con un poquito más de detalle usando también o retomando el ejemplo de plantilla de implementación elemental que también estudiamos en la última sesión.

Fíjense que el caso en el que yo no tenga ningún tipo de navegabilidad representada en el modelo, se asimila al caso en el que tengo una navegabilidad bidireccional, es decir, en estos dos casos, yo tengo la posibilidad de ir desde a a b por, desde a tengo acceso a b o desde b a a, desde b tengo definido como atributo privado el atributo de la clase a. Repito, tanto esto me representa tanto el hecho tenemos navegabilidad explícitamente o no explícitamente especificada por el hecho de que por defecto, al no decir nada, se asume que estamos permitiendo una navegabilidad bidireccional.

Sin embargo, cuando estamos expresando navegabilidad, la situación cambia.

Es equivalente el decir, como ven, que de a voy a b, a decir, que de b no voy a a través de esta, de esta sintaxis. En cualquiera de los dos casos estamos diciendo que yo tengo desde a la posibilidad de acceder a b, pero desde b, no tengo la posibilidad de acceder a a diferencia de lo que ocurría en el caso anterior.

Y finalmente, cuando estamos en el caso en el que la no navegabilidad está marcada explícitamente, ya no es indefinida como en el primer caso, sino que marcamos explícitamente con esta notación que no está autorizada la navegabilidad en ninguna dirección, esto nos llevaría a la situación en la que desde la, ni desde la clase a puedo acceder a b, ni desde la clase b puedo acceder a a.

Visibilidad en las navegaciones

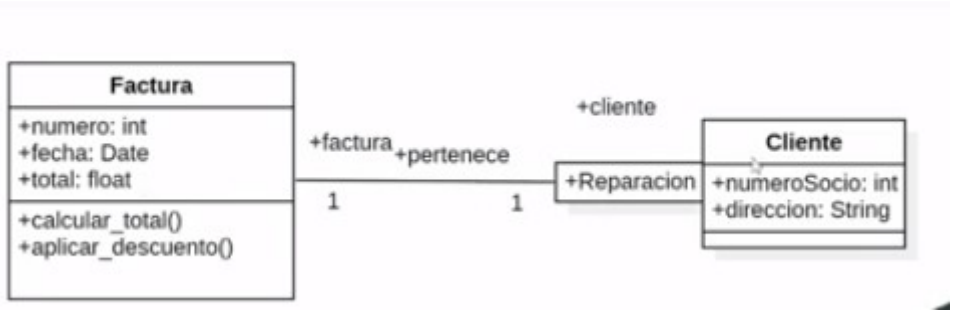
De acuerdo con lo que vimos ya en temas anteriores tenemos tres opciones de visibilidad, que son las proporcionadas por UML por defecto, que son las tres que vemos en la figura, pública, visibilidad pública, accesible a cualquier clase del sistema, se representa con este más, protegida, a través de esta de esta almohadilla (numeral) que hace referencia a situaciones en las que la visibilidad queda restringida a las jerarquías de generalización en la que está implicada la clase a la que nos estamos refiriendo, haciendo uso de las relaciones de generalización que ya hemos estudiado y finalmente este menos que me indica que estamos hablando de una propiedad de visibilidad privada, solo va a ser visible accesible en el ámbito de la clase en la que se efectúa la definición.

Volviendo a nuestro ejemplo, si se fijan estos días pasados cuando veíamos el la definición de cada uno de los componentes de la clase aparecían aparecían signos más o menos en función de que los atributos de la clase por ejemplo, la clase reparación, su atributos son privados lo que va a querer decir que sólo van a ser modificables, accesibles, a través del ámbito de la propia clase, mientras que los atributos de la clase factura son públicos y las operaciones son públicas lo que indica que van a ser atributos y operaciones accesibles por cualquier clase externa del sistema. Bien, también como pueden ver, tanta todas las propiedades, tanto roles como nombre asociación pues tienen esa característica de pues poder ser definidas como públicas, privadas o protecteds en función de cuál sea la característica de visibilidad que queramos introducir o especificar en nuestro modelo correspondiente.

Bien, dicho esto, el siguiente concepto importante en este ámbito es el de asociación calificada, este concepto se utiliza cuando la multiplicidad de una asociación es de uno a muchos pero queremos seleccionar de ese conjunto de muchos, de esa multivaluación queremos seleccionar un único objeto específico que pueda cumplir la asociación y que sea el que queremos recuperar.

Eso implica seleccionar ese objeto de entre el conjunto de objetos y para que esto sea posible necesitamos un identificador que permita diferenciar el objeto que queremos separar de los demás de una manera individualizada y bueno a esto lo a este a este identificador es el que vamos a llamar calificador, i de ahí el nombre de asociación calificada.

El calificador va a ser un atributo o tupla de atributos de la asociación cuyos valores nos van a servir para a partir o para clasificar un conjunto de objetos asociados de manera que podamos discriminar de esa multivaluación cuál es el objeto que nos interesa. Es decir vamos a convertir una multi valoración de la parte correspondiente de la relación de asociación que estemos analizando en una univaluación, vamos a convertir esos atributos multivaluado en un único atributo unievaluado que sea aquel cuyo calificador nos permita determinar. El calificador, y esto es importante resaltarlo, es parte de la asociación, no parte de la clase es una propiedad asociación que me permite distinguir entre el conjunto de objetos en el extremo multivaluado para identificar el objeto que queremos de alguna manera particularizar o seleccionar como elemento único característico y especial de esa multievaluación.



Por ejemplo, tenemos el en el ejemplo nuestra factura con el cliente pues imagínese en este caso cuando yo, cuando un cliente hacer referencia un cliente tiene muchas facturas, por lo tanto esta relación de cliente factura era multivaluada porque un cliente podía tener muchas facturas.

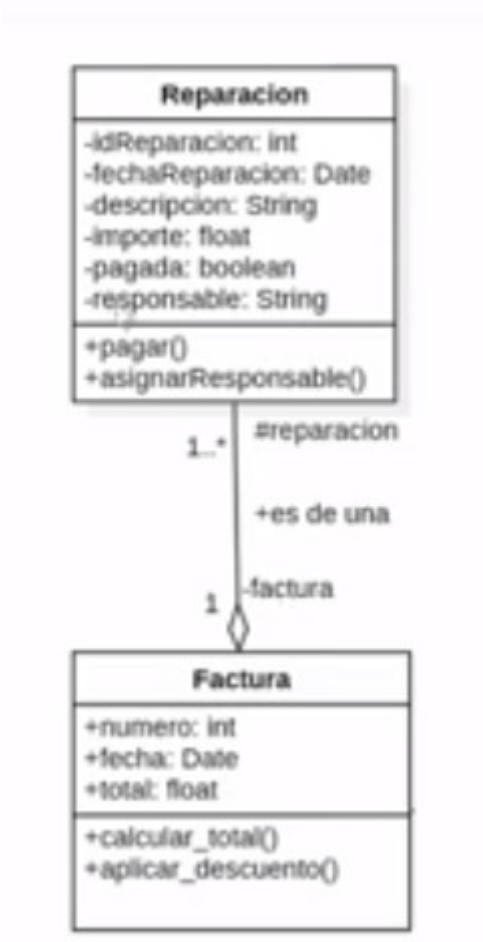
Pero si califico al cliente con la reparación, la un cliente y una reparación de ese cliente convierte esa multievaluación en una única, en una única factura que es la que ha hecho que este atributo multivariado se convierta en un atributo univaluado. Bien, esta asociación calificada, a título de curiosidad ya existía en el método OMT, de James Rumbaugh, Blaha y otros autores que como las primeras elecciones veíamos estaban en la en el origen de lo que fue la propuesta UML y simplemente es un concepto que pasó del modelo de objeto, de OMT al diagrama de clase de UML y aquí está disponible para ser usado en el caso de que un modelador requiera su utilización.

Agregación

Es una propiedad muy importante en el ámbito de las relaciones de asociación. De hecho la agregación es un caso especial asociación con el que puede expresar una situación desde el punto de vista semántico muy significativa que es la de la existencia de una relación estructural todo parte de

estar formado por ser parte de entre un objeto que actúa como objeto agregado el "todo" y aquellos objetos que actúan como partes que componen o que son o están contenidas en dicho objeto agregado.

La agregación entendida de manera general una parte puede pertenecer a más de un agregado y puede existir independientemente del agregado veremos en temas posteriores en el caso de la composición como una agregación se puede definir de manera más estricta no permitiendo que la interpretación de esas propiedades sea tan flexible eso lo veremos cuando estudiemos la composición.



Si vemos en este o nos fijamos en este ejemplo en la que tenemos dos clases una clase reparación, una clase factura, en el tema anterior definíamos una relación de asociación entre ellas aquí ahora estamos definiendo este rombo junto a la factura este rombo de fondo blanco que me introduce la noción de agregación y que me permite especificar que en este caso la factura es vista como un agregado de reparaciones. La factura es una clase agregada cuya clase contenida son las reparaciones y por lo tanto ese conjunto de mínimo uno máximo muchas reparaciones asociadas a una factura en este caso se convierten en una asociación que es una agregación y en la que esa relación unidireccional "parte de" viene dada por este rombo que me permite leer la relación de asociación, de agregación en este caso, como una relación que va del objeto contenedor la factura a los objetos contenidos que son las reparaciones.

Asociación o agregación. Bien, la diferencia entre asociación y agregación es más conceptual que una diferencia semántica real más allá de una característica esencial que es la expresividad asociada esa característica de ser "parte de".

Es más conceptual por el hecho de que las propiedades de la asociación se mantienen también en la agregación tanto la asociación como la agregación tienen nombre, tienen roles, tienen visibilidad, tiene multiplicidad. Realmente el hecho de que tengamos que expresar una relación de asociación como una agregación va a venir condicionado por el hecho de que queramos que esa bidireccionalidad de una clase asociación se convierta en unidireccionalidad como consecuencia de la necesidad de querer expresar la existencia de esa relación "parte de" que establece una dirección de objeto contenedor hacia objetos contenidos. Las prestaciones de la agregación se dan en el paso de análisis al diseño por lo que realmente es la inclusión en el modelo dominio, por lo dicho anteriormente, por el hecho de que se comparten muchas propiedades pues no es especialmente significativa.

Una pregunta interesante es cuando emplear la agregación. Aquí tenemos cuatro indicadores claros de situaciones en las que una asociación tiene que ser expresada como una agregación.

1. Puede ser que yo quiero expresar que existe un ensamblaje obvio evidente del todo con las partes y que quiera que esa que esa perspectiva de especificación de modelado este explícitamente incluida en el modelo, por lo tanto, en ese caso tengo que definir una agregación.
2. También tengo un indicador claro de agregación cuando existen propiedades del objeto que va actuar como compuesto que se propagan a los objetos que son partes de dicho objeto compuesto.
3. Otro indicador habitual de situaciones en las que tengo una agregación es aquella situación en la que existen operaciones que están en la clase compuesto y que se van a propagar a las clases que actúan como partes del objeto compuesto.
4. Finalmente el tiempo de vida de la parte del objeto agregado está habitualmente ligado al tiempo de vida del objeto compuesto esto aquí tenemos cierta dependencia creación-eliminación de la parte del todo. En el caso de la agregación no es estricta no exige que si se elimina el objeto agregado, por ejemplo, haya que eliminar todos los componentes pero que veremos más adelante que en una versión más estricta más restringida de la agregación que es la composición justamente esta este tiempo de vida esta perspectiva de que la vida del objeto contenido está ligada al objeto compuesto pues veremos como en el caso de la composición es más estricta.

¿Qué beneficios tiene utilizar la agregación? En primer lugar, el hecho de que yo haga explícita la existencia de esa relación "parte de" como ya hemos indicado anteriormente en varias ocasiones. También es importante incidir en que me ayuda identificar un creador que es el objeto compuesto recordándonos al patrón GRASP creador, recuerden que los patrones GRASP van asociados a la a la especificación de responsabilidades, a la asignación de responsabilidades en el ámbito de un diseño orientado objetos y que hay un patrón en particular el creador que me determina quién actúa como creador distancias. En el caso de la agregación tengo un caso claro de objeto creador de instancias que va a ser el objeto que actúa como contenedor y al mismo tiempo las operaciones, por ejemplo,

de copia, de eliminación que se aplicará al todo van a ser propagadas a cada una de las partes lo cual desde el punto de vista semántico pues es también un recurso valioso. En cualquier caso es una recomendación habitual en el que en el caso de estas dudando entre sí cuando tenemos una relación entre dos clases de relaciones de asociación o de agregación en el caso de duda es recomendable utilizar la asociación por ser una relación menos restrictiva de perspectiva semántica más amplia y que delimita menos el tipo de expresividad que en cada una de las claves asociadas yo voy a poder especificar.

Composición

La propiedad de composición asociada una agregación como forma de materializar una perspectiva más estricta, menos flexible de lo que es una agregación tal y como la veíamos en el tema anterior.

Podemos ver la composición como una forma fuerte de agregación. ¿Qué quiere decir esto? Cuando hablamos de agregación en el tema anterior estamos refiriéndonos a una relación parte de, genérica, en el que no exigíamos dos principios en particularidad que en el caso de la composición van a ser de obligado cumplimiento.

Estos dos principios son la exclusividad y la dependencia fuerte.

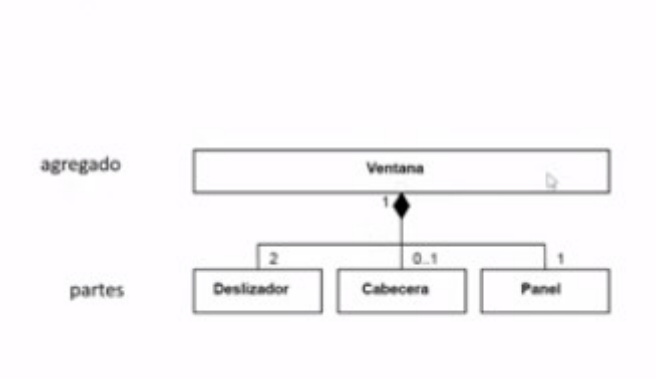
1. La exclusividad va a querer decir que una parte debe de pertenecer a un único agregado, veíamos en el caso de la agregación o de un agregado podía ser parte de diferentes objetos contenedores. En el caso de la composición existe esta característica exclusiva de que la parte sólo pueda pertenecer a un único agregado.
2. Al mismo tiempo esa dependencia estricta, esa dependencia fuerte que hace que cuando elimine un agregado, elimino obligatoriamente todas sus partes. Esto es una agregación estricta, es una composición, en el caso general de la agregación, que no es una composición no es necesario exigir esta dependencia fuerte.

La responsabilidad de la disponibilidad de sus partes es también una característica asociada a la noción de composición. El elemento compuesto es por lo tanto responsable de la creación y destrucción de todas sus partes de una forma explícita.

Las partes tienen un tiempo de vida que va a ser coincidente con el conjunto por definición, las partes con multiplicidad no fija van a poder ser creadas después del elemento compuesto pero una vez hayan sido creadas tendrán que compartir su ciclo de vida con dicho elemento compuesto, es decir tendrán que vivir y morir al mismo tiempo que el elemento compuesto existe.

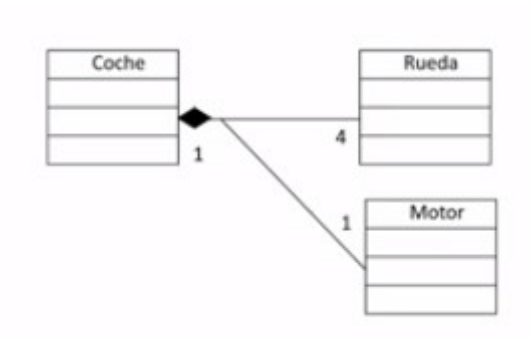
Finalmente durante la vida de este objeto agregado ningún otro objeto puede tener la responsabilidad de crear o destruir las partes que lo componen, por lo tanto es es responsabilidad exclusiva del objeto que actúa como contenedor esa propiedad de crear o destruir cualquiera de sus clases componentes.

Bien tanto a nivel notacional como a nivel semántico tenemos aquí un ejemplo de cómo representar la composición.



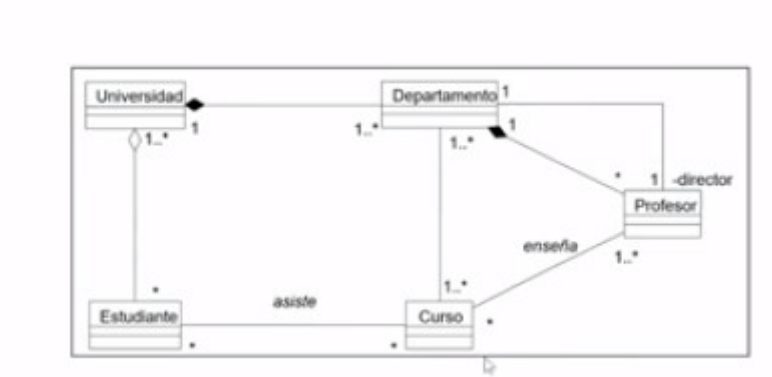
Aquí tenemos una clase, un objeto agregado, la ventana, que se representa a través de este rombo de fondo oscuro como una composición de tres objetos que actúan como partes que es un desliz, dos deslizadores de acuerdo con la cardinalidad que estamos expresando, mínimo cero, máximo una cabecera, y uno y sólo uno, panel que son los que van a van a estar asociados como componentes a lo que es el objeto, el objeto ventana visto como un objeto agregado por composición. En este, en esta representación utilizando un un rectángulo externo con el objeto agregado compuesto y rectángulos interiores con cada uno de sus componentes estamos representando el mismo tipo de situación. Una ventana con sus dos en este caso al no poner mínimo va a querer decir que son dos y sólo dos deslizadores, cero mínimo, máximo una cabecera y un panel lo tablero y sólo uno como consecuencia de la cardinalidad que aquí estamos expresando.

Otro ejemplo que nos permite entender quizá de una manera pues pues muy muy descriptiva el la semántica de la composición es el que vemos en esta figura.



Tenemos un coche y la clase coche, esta clase coche es una composición, una agregación composición como consecuencia de utilizar este rombo de fondo oscuro de cuatro ruedas y un motor. Cuatro ruedas y un motor que van a ser partes del coche que por lo tanto no voy a poder tener motores o ruedas que estén sueltos, no voy a poder tener ni objetos de la clase rueda, ni objetos de la clase motor que estén viviendo de forma independiente a su contenedor que tiene que ser un coche. Bien como ver aquí nos serviría para modelar un desguace, porque no puedo tener nada que no esté incluido en el coche que actúa como clase contenedora y que como consecuencia de esa propiedad de de exclusividad que comentamos anteriormente al borrar el coche, se borra todo al eliminar el coche, elimino sus cuatro ruedas y elimino igualmente su motor. Otro ejemplo en el

que también podemos ver pues la volatilidad que puede tener la decisión de cuando utilizo agregación, cuando utilizo composición aparece en este diagrama de clase



en el que modelamos una universidad como una composición de departamentos. Visto así, esta composición esta relación esta relación de asociación que es una composición quiere decir que estoy visualizando la, una instancia de la clase universidad es decir, una universidad, un objeto universidad como un conjunto de departamentos que son componentes intrínsecos de esa universidad.

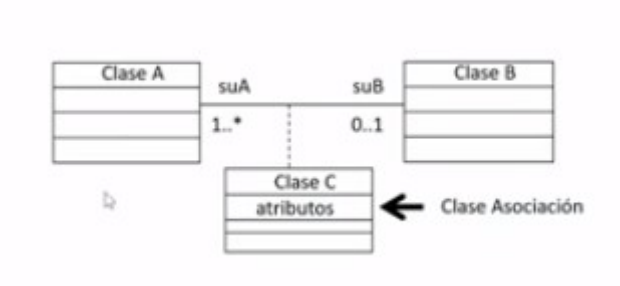
Al mismo tiempo, un departamento es una composición de profesores. Entre profesor y curso sin embargo no defino esa relación de contenido en o de parte de, sino que defino simplemente una asociación en la que pues puedo ir tanto del profesor de los cursos como del curso a los profesores sin tener que estar restringido por esa semántica de parte de o de objeto contenido en que la relación de composición me implica. ¿Qué diferencia tendríamos si por ejemplo en lugar de tener aquí esta composición representada con este rombo de fondo oscuro, tuviéramos una agregación que vendría representada por un mismo rombo pero de fondo blanco, no de fondo oscuro? Bien en este caso tenemos una agregación estaríamos indicando que el departamento es también una agregación de profesores pero no una composición y por lo tanto estaríamos indicando en ese caso que el profesor no tiene que ser únicamente miembro de un departamento y no tiene que darse de forma obligatoria el caso de que al eliminar el departamento tenga que borrar también todos los profesores que componen ese departamento. Esa sería la diferencia que existiría entre la semántica de la relación de asociación en primer lugar y las relaciones de agregación y composición que son las últimas que hemos estudiado.

Volviendo a nuestro ejemplo inicial podríamos ver también una factura con una composición de reparaciones, en este caso estaríamos indicando nuevamente que cuando borro una factura voy a borrar todas sus reparaciones, recuerden que eso no pasa necesariamente si tengo una agregación, si tengo un rombo de fondo blanco, porque la restricción de borrado total, de pertenencia exclusiva solo ocurre cuando tengo la composición, en este caso es una definición asociada a la propia noción de composición que estamos especificando. Igualmente cuando tenemos una composición una reparación, sólo puede ser miembro de una y solo una factura que es a la que pertenece, es de la que forma parte. Recuerden que esas son las dos características que son esenciales en el caso de estar utilizando la composición como elemento de modelado de la relación de asociación.

Clase asociación

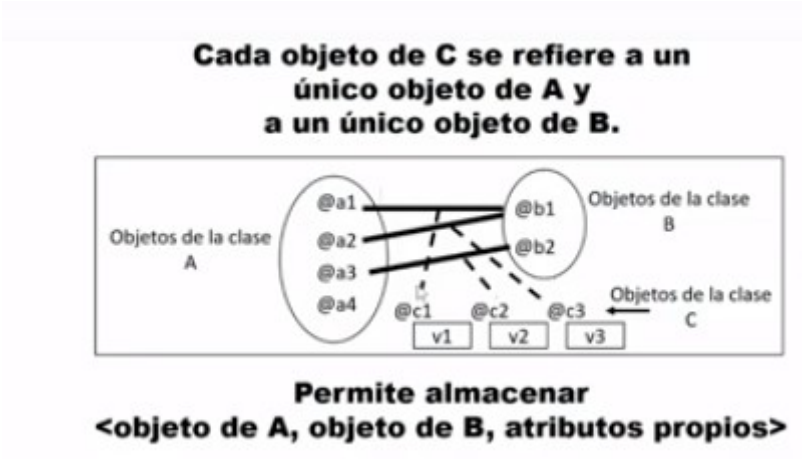
Una clase asociación es un concepto muy importante en UML que me permite tratar una relación de asociación como una clase ,es decir, estamos hablando de una asociación que también va a ser una clase que va a poseer por tanto sus propias características en su propio conjunto de atributos y que me va a permitir añadir una restricción a mi entorno de modelado que va a ser la restricción con la que yo pueda asegurar que sólo existirá una y solo una instancia de la asociación entre cualquier par de objetos participantes en la clase asociación.

Bien, veamos en este ejemplo a que nos estamos refiriendo aquí tenemos un diagrama de clase elemental en el que aparece una clase A, una clase B, una relación de asociación, las cardinalidades correspondientes que hemos estudiado en temas anteriores

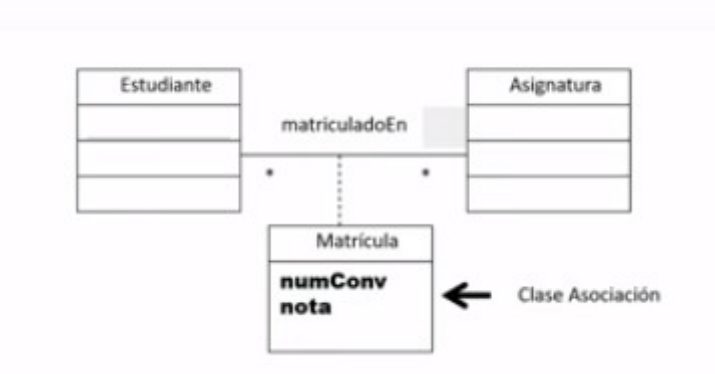


pero fíjense como aquí aparece esta línea de puntos y esta clase C colgando de esa asociación que es justamente la clase asociación. Esta clase C es la que la asociación y esta clase C va a poder tener sus propios atributos que van a ir asociados a un par formado por una asociación entre un objeto de la clase A y un objeto de la clase B que están asociados.

Vamos a ver esto vamos a verlo con este diagrama en el que en el que representamos el hecho de que cada objeto de la clase asociación cada objeto de esa de esa clase asociación C se va a referir a un único objeto de A un único objeto de B. Aquí tendríamos cuatro objetos de la clase A, dos objetos de la clase B, una serie de enlaces entre objetos como ven aquí y fíjense como a cada uno de estos enlaces ,por ejemplo, al enlace definido entre el objeto a1 y el objeto b1 le asociamos ese objeto c1. El objeto c1 va a tener un valor de atributo y esas propiedades van a ir asociadas al hecho de que el objeto a1 y el objeto b1 esten asociados.

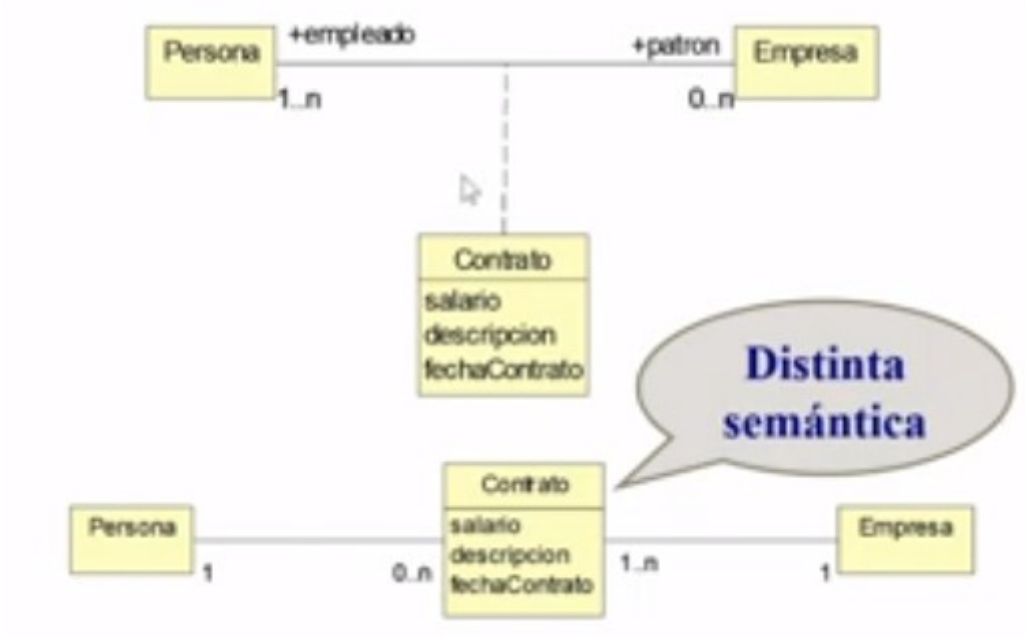


De la misma forma, si el objeto a2 y el objeto b1 están asociados definimos otra otros atributos para esa instancia de la clase asociación de manera que yo voy a poder almacenar en cada uno de estos casos un objeto de a un objeto de b y un conjunto de atributos propios por el hecho de haber asociado ese objeto de la clase a con ese objeto de la clase b. Instanciando en un caso real esto que estamos comentando pues aquí podemos ver como estas clases a y b se convierte una clase estudiante, una que la asignatura, los estudiante están matriculados en asignaturas con la cardinalidad de cero a muchos, un estudiante puede estar matriculado en cero o muchas asignaturas, en una asignatura pueden estar matriculados cero muchos estudiantes



pero fíjense que cuando yo tengo un estudiante y una asignatura asociados porque el estudiante está matriculado en esta asignatura en particular aparece una instancia de la clase asociación en la que yo puedo obtener o puedo adicionar como atributos de esa instancia de la clase asociación el número de convocatoria y la nota que ese estudiante obtengan es asignatura. Como ven estas propiedades tanto el número de convocatoria, la nota obtenida, el curso académico… Son propiedades que aparecen como consecuencia del hecho de que un estudiante en particular está matriculado en una asignatura en particular que es lo que representa justamente una instancia de esta asociación y cuyos atributos son los que yo represento en esa clase asociación que cuelga de la relación de asociación.

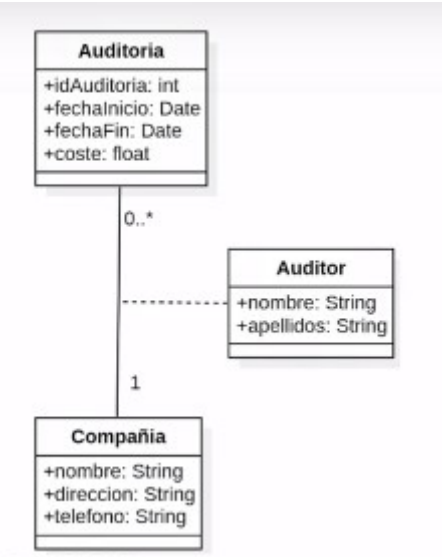
Bien, es importante destacar que cuando yo quiero expresar aquí que la clase esta clase de la que estamos hablando como clase intermediaria, por así decirlo, entre la clase a y la clase b si yo quiero repito que ésta que un objeto de la clase c puede relacionarse con muchos de la clase a ya no puedo usar la clase asociación porque la clase asociación me fuerza a considerar pares únicos formados por un objeto de la clase a y un objeto de la clase b, un objeto único de cada una de las clases que están asociadas por la relación de asociación. En este caso esta multiplicación impide que yo pueda usar la clase asociación porque eso no va asociado a la semántica de la clase asociación como hemos indicado anteriormente. Es por eso que estos dos casos tienen una semántica diferente.



Cuando utilizamos la clase asociación, estamos asociando una instancia de contrato a una persona que está asociada a una empresa, el contrato es de una persona en una empresa.

En este caso inferior en el diagrama de clase simplificado de la parte inferior por el contrario, un contrato se refiere a una persona pero una persona puede tener cero muchos contratos y si yo quiero especificar esta multivaluación eso no lo puedo hacer usando la clase asociación y por lo tanto tengo que utilizar un recurso válido también que es el del lugar de definir la clave asociación, definir directamente ese concepto que va asociado a la clase asociación como una clase independiente de mi diagrama de clases.

Bien, volviendo o ampliando los ejemplo que estamos viendo, un ejemplo claro en un entorno en el que imaginemos que yo tengo una clase compañía que se asocia a auditorías, la compañía se ve cómo se realizan auditorías sobre ella cada una de estas clases auditoría y compañía tiene su conjunto atributos



fíjense que dentro de asociación una compañía puede tener cero mucha auditorías. Una auditorías de una y solo una compañía pero asociado a un par compañía auditoría, es decir, cuando estamos viendo una auditoría de una compañía cuando nos estamos refiriendo a una auditoría de una compañía tengo un auditor colgando de esa asociación y eso está representado por esta clase asociación auditor con dos atributos que me indica cuál es el nombre y los apellidos del auditor con el que ha quedado asociada esa auditoría a esa compañía.

Nota: Una clase que define una familia de clases, cada una de las cuales es especificada asociando los parámetros a una lista de valores actuales, recibe el nombre de Clase Parametrizada.

Diagramas estructurales en UML

Asociación n-aria

En cursos anteriores habéis visto lo que es una relación entre dos clases y hoy lo que vamos a ver es cómo es una asociación entre más de dos clases. Puede ser de tres, sería ternaria o de n cantidad sería n-aria. Ya les digo que tampoco se asusten porque tipo de asociaciones se ven rara vez. De hecho es aconsejable que las evitéis en la medida de lo posible ¿vale? Tienen unas restricciones importantes.

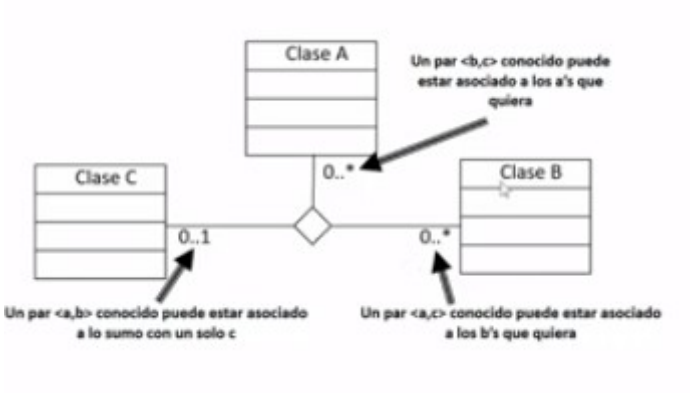
La primera de todas es que es bastante complicado de implementar ¿vale? se pueden expresar a nivel conceptual, pero luego al nivel de implementación, pues no se pueden llevar a cabo de forma sencilla. Y luego tienen también otra restricción y es que podemos especificar la cardinalidad desde la relación hacia la clase, pero no desde la clase hacia la relación ¿vale?

Si se acuerdan, en el tema de vimos la asociación entre dos clases, la asociación tenía una cardinalidad en ambos sentidos, de una clase a hacia una clase b y de una clase b hace una clase a. En este caso, una sección n-aria, únicamente podemos tener la relación con cardinalidad desde sentido de la relación hacia la clase. Ahora lo veremos un ejemplo ¿vale? Pero es una restricción importante.

Luego también, otra restricción de este de tipo de asociación es que no podemos definirla ni de tipo agregación ni de tipo composición ¿vale? Con lo cual también quiere decir que nos restringe mucho las posibilidades.

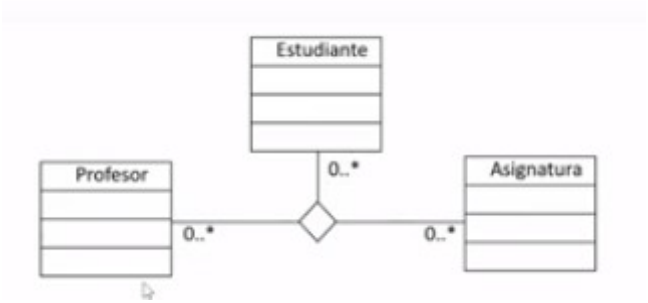
También hay que tener en cuenta que las relaciones ternarias y cuaternarias únicamente están pensadas para aquellas relaciones entre clases que se van a dar siempre ¿vale? ¿Qué quiere decir? que si tengo tres clases implicadas en la relación, la relación se dará si se da entre las tres clases ¿vale? y no puede tener una relación entre dos clases y dejar la tercera parte. O se da entre las tres clases o no se da, esto también restringía bastante.

Este es un ejemplo de relación. Tendremos tres clases la clase a, b y c y aquí vemos el significado de cada tipo de cardinalidad ¿vale? Por ejemplo, la clase c, aquí tenemos la cardinalidad, que decir que la relación entre a y b está relacionada entre cero y una instancias de la clase c, o sea, que el par a b estará relacionado entre cero y una instancias de la clase c.



En este caso de aquí, tenemos otro ejemplo, que decir que entre la clase c y b habrá una relación de cero a muchas con la clase a. Y por último, esta cadena de aquí me indica que entre el par c y a habrá una relación de cero a muchas con la clase b ¿vale?

Resaltar que hablamos siempre de relaciones c y a, para esta relación de aquí, c y b para esta de aquí, y a y b para esta de aquí ¿vale? siempre hablamos de un par junto con la clase que estamos contrastando. Paso ya con un ejemplo ¿vale? ¿Qué significa este diagrama de aquí que veis a modo de ejemplo?



Quiere decir que un profesor está relacionado con un par de estudiante-asignatura ¿vale? o por ejemplo, yo que soy profesor, instancia profesor ignacio, está relacionado con estudiantes en asignaturas ¿vale? tendríamos asignatura de UML y estudiante puede ser Juan. Entonces para Juan y la asignatura de UML, puede haber de cero a n profesores ¿vale? entre el estudiante Juan y la asignatura UML hay en este caso dos profesores está ignacio y está Óscar podría ¿vale? Pero podría haber de cero a varios profesores. Eso en cuanto a la relación de profesor con el par estudiante asignatura. Ahora vamos a ver el ejemplo de estudiante con el par profesor asignatura ¿vale? El estudiante Juan tiene un profesor, que sería Ignacio, en la asignatura UML ¿vale? y Ignacio con UML tiene un conjunto de estudiantes ¿vale? de cero a muchos.

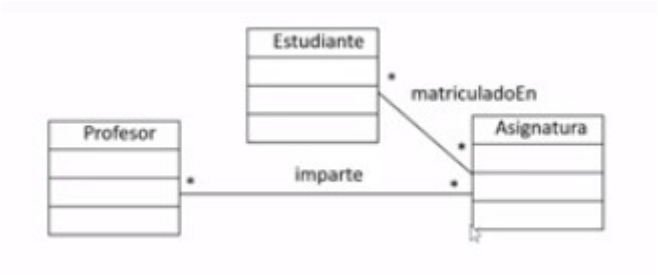
Pues Ignacio con UML tiene de estudiantes a Juan, a María, a Antonio, a n ¿vale? el par ignacio UML ¿vale? Dos instancias concretas de estas clases.

Y por último, vamos a ver la relación que habría entre el par profesor estudiante con asignatura.

Tenemos que el profesor Ignacio con el estudiante Juan pueden estar en más de una asignatura. ¿Qué quier decir esto? pues que puede estar estar conmigo en UML y conmigo pues en otra materia

que puede ser, no sé, diagramas de estados ¿vale? El par profesor estudiante tiene de cero a n asignaturas.

¿Cómo evitar las de un ternaria? Hemos dicho antes que es recomendable evitarla en la medida posible. Pues se puede evitar poniendo relaciones más simples. Aquí tenemos un ejemplo ¿vale? Tenemos profesor, relacionado con asignaturas y asignatura con estudiante.



¿En qué se diferencia esta relación de la anterior? En que no sabemos el profesor qué estudiantes tiene asignados. Sabemos que asignatura tiene y qué estudiantes están en cada asignatura, pero no sabemos qué estudiante está con qué profesor.

Si quisiéramos saberlo, ¿cómo lo haríamos? Pondríamos una relación entre estudiante y profesor ¿vale? Tendríamos aquí un círculo, no pasaría nada ¿vale? pero tendríamos un círculo ¿vale? Así si cambiamos este modelo por el círculo, tenemos una equivalencia perfecta que reemplazaría totalmente a la relación ternaria.

¿Vale? Este es el ejemplo anterior, aquí se soluciona el problema de que sabemos el profesor con qué estudiante está relacionado ¿vale? porque de hecho profesor se relacionar tanto con estudiantes como con asignaturas y la relación va siempre unida. Por ejemplo, no puedo tener un profesor con un estudiante sin saber la asignatura ¿vale? Si tengo relación profesor estudiante, es porque también el estudiante está en una asignatura, si no, la relación no existe.

Imaginaos que el el par estudiante asignatura tiene sentido únicamente para un profesor, pues en ese caso cambiamos la cardinalidad. Tengo el estudiante Juan, que está en la asignatura UML y tiene solamente un profesor, que sería Ignacio. Vale, pues en ese caso cambiamos la cardinalidad. Sería el par estudiante asignatura tiene únicamente entre cero y un profesor ¿vale? sería un ejemplo distinto.

En este caso no cambiaría, tendríamos profesor y estudiante relacionado con más de una asignatura, de cero a muchas ¿vale? pues en este caso sería profesor Ignacio con el estudiante Juan puede estar en más de una asignatura, puede estar en UML o en diagramas de estado.

Y aquí tampoco cambiaría, sería el profesor con la asignatura, entre cero y muchos estudiantes. El profesor sería Ignacio, la asignatura sería UML y pues habría más de un estudiante. pues Ignacio en UML le da al estudiante Juan, a María y a Antonio.

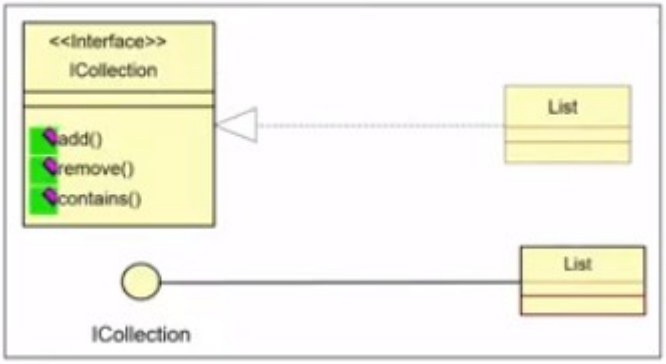
Nota: En una relación ternaria se pueden representar las cardinalidades de la relación a la clase pero no al revés

Realización

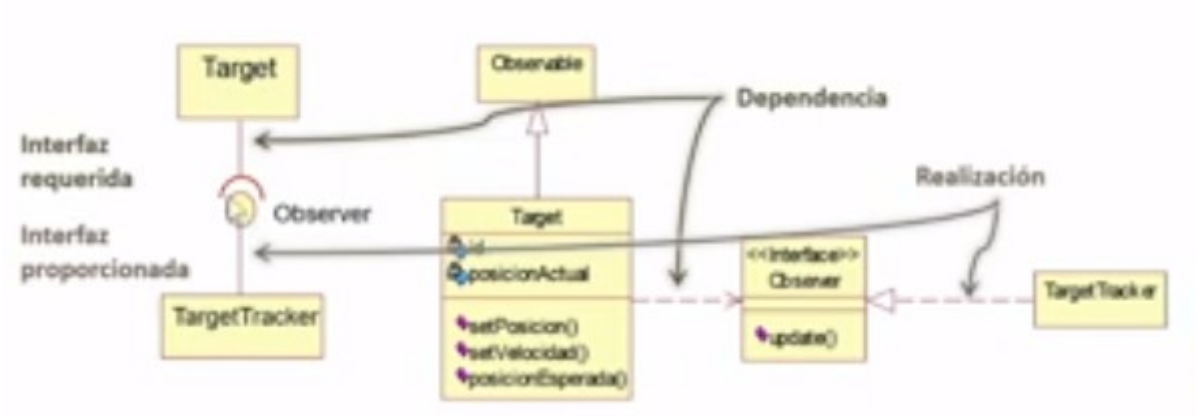
La realización es una relación entre dos clases de UML de forma que una clase que especifica un contrato se relaciona con una clase que lo implementa. ¿Qué quiere decir definir un contrato?,

si os acordáis en temas anteriores hemos visto el concepto de interfaz, vale, una interfaz no es más que un contrato que especifica una serie de operaciones con su signatura o con los parámetros que va a recibir pero no se implementa, únicamente dice que operaciones va a tener esa clase pero no las implementa, no sabemos realmente lo que hacen, vale, entonces esa interfaz se tiene que relacionar con una clase que realmente implemente la interfaz, que realmente de contenido a esas operaciones.

La relación que hay entre la interfaz o contrato y la clase que la implementa es lo que se conoce como realización, entonces tendremos una relación de realización entre cada interfaz que tengamos en nuestro diagrama y cada clase que la implemente.



En UML dos punto cero hay dos notaciones para representar este tipo de, de relación de realización. La primera que tenéis aquí, vale, es una línea con puntos, punteada, y en la de abajo es lo que se conoce como lollipop, vale, como si fuera esto de aquí pues un chupachups, una piruleta, vale. ¿En qué se diferencian?, que la primera es más tradicional, la que más se utiliza, es más detallista porque aparte de proporcionar el nombre de la interfaz también nos da la lista de operaciones, vale, no las implementa pero sí que nos dice qué operaciones tenemos que implementar en la clase que implemente la interfaz, por tanto si yo tomo la clase lista que implementa esta interfaz se que tengo implementar a todas todas, a la fuerza, esas tres operaciones de aquí, por tanto esta notación a parte indicarme que esta clase implementa esta interfaz, me está indicando qué operaciones tengo que implementar. En cambio en esta anotación aquí, es más reducida visualmente pero da menos detalle pues aquí tengo la clase lista que implementa esta interfaz pero no tengo forma de ver qué operaciones incluye la interfaz, por tanto pues es una notación más ligera visualmente pero es verdad que menos detallista. Vamos a verlo con un ejemplo, de hecho en este ejemplo tenemos las dos notaciones, la lollipop y la tradicional.



Aquí tenemos una interfaz que es observar, esta bolita aquí, como he dicho antes se que es una interfaz porque es una bolita pero no sé qué operaciones hay dentro, no tengo forma de saberlo.

Tengo aquí el target tracker que lo que hace es implementar esta interfaz, aquí tengo la interfaz y aquí tengo la clase que implementa la interfaz, es la clase que toma las operaciones de interfaz y le da un cuerpo, les da un contenido. Y luego tengo aquí una relación de dependencia, la clase target tiene una dependencia con el target tracker, esta relación de dependencia lo que me indica es, que la clase target necesita la interfaz observer pero no la implementa, la implementa el target tracker, vale, por tanto tengo como tres entidades o tres tipos de clases aquí metidas, la interfaz que es el observer, que me indica que interfaz necesito mi, en mi sistema, la clase que implementa la interfaz que es target tracker y la clase que hace uso de esa interfaz que es target. La que hace uso es la relación de dependencia y la que implementa la interfaz es la realización, esto es dependencia, esto es realización en notación lollipop.

La otra notación la tenéis aquí en la parte derecha, pues tengo la interfaz igual que antes, solo que en este caso pues veo el detalle, veo la operación que realmente tiene la interfaz.

Tengo el target tracker que me implementa la interfaz, esta clase aquí tendrá el método update con cuerpo, pues tendrá ciertas operaciones, lo que tenga que hacer, y luego tengo la dependencia que es target, pues dependencia quiere decir que target hace uso de esta interfaz.

¿Qué tipo de uso hace?, pues aquel que le proporcione el que la implementa, esta clase implementa la interfaz y este de aquí al fin y al cabo tendrá que depender de alguna forma de la implementación que le hayamos dado a este target tracker.

En este caso como curiosidad pues target hereda observable pero bueno, para el caso de la realización es una anécdota, no aporta información relevante, simplemente que vean en este ejemplo los dos tipos de acciones que tenemos en la realización.

Instancias

En temas anteriores hemos visto lo que es una clase lo que es un diagrama de clases y hoy lo que vamos a ver es lo que es una instancia. Si las clases son como moldes que representan cualidades, las instancias, lo que son, son elementos que se quedan con esos moldes, elementos que se quedan con esas clases. Yo puedo tener una clase coche, por ejemplo, pues que tengo atributos matrícula y

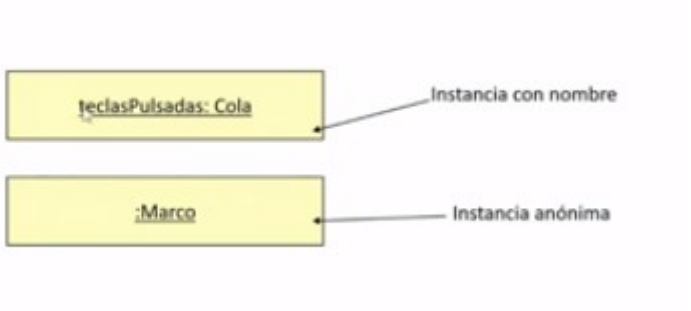
color y luego, cada instancia a esa clase coche, será un coche particular; pues será un coche con una matrícula concreta y un color concreto y puedo tener N istancias de la clase coche.

La clase me indica qué cualidades o características van a compartir todas las instancias de esa clase. Una instancia nada más que es un objeto de una clase concreta. Esa entidad o esa instancia tendrá los atributos que hemos definido en el diagrama de clases y los servicios que hemos diseñado en el diagrama de clases.

Además, estos objetos tienen un estado ¿Qué es el estado? El estado lo que me indica es por qué fases va a pasar cada combinación de atributos. Pues en base al atributos que tenga un objeto, extraer un estado o extraer en otro estado distinto. Digamos que el estado lo definen los atributos del objeto. Vamos a ver un ejemplo.

Esta es un ejemplo instancia, que podría ser un objeto que llamaremos cuenta bancaria y tenemos atributos. Por ejemplo, el balance pues ¿cuál es la cantidad de dinero que hay cuenta o el tipo de interés que se pagan esa cuenta bancaria? o el interés que hay que dar alrededor son atributos que pueden tomar uno u otro valor. Proyecto atributo, que es saldo actual, que me marca el estado. Pues por ejemplo, si el saldo actual está por encima de cero, el estado de la cuenta bancaria es un estado normal, pero si baja de cero, el estado de la cuenta bancaria serán números rojos o negativo por tanto, el valor de este atributo, me está marcando el estado. El objeto cuenta bancaria tendría dos estados: estado normal y cuenta al descubierto o números rojos, si el saldo baja de cero. Pues en base a este valor, que tenemos el atributo estado, tendríamos dos estados en la instancia cuenta bancaria. Además de atributos balance, tipo de interés, interés acreedor y saldo actual, tendríamos operaciones, tendríamos comportamiento. Estas operaciones estarán definidas como operaciones de la clase que representa este objeto. La función haber, debe, informe, abrir y cerrar y luego este objeto tendrá una finalidad, que es la responsabilidad. En este caso sería almacenar dinero en base a los cheques que le vamos proporcionando al interesado.

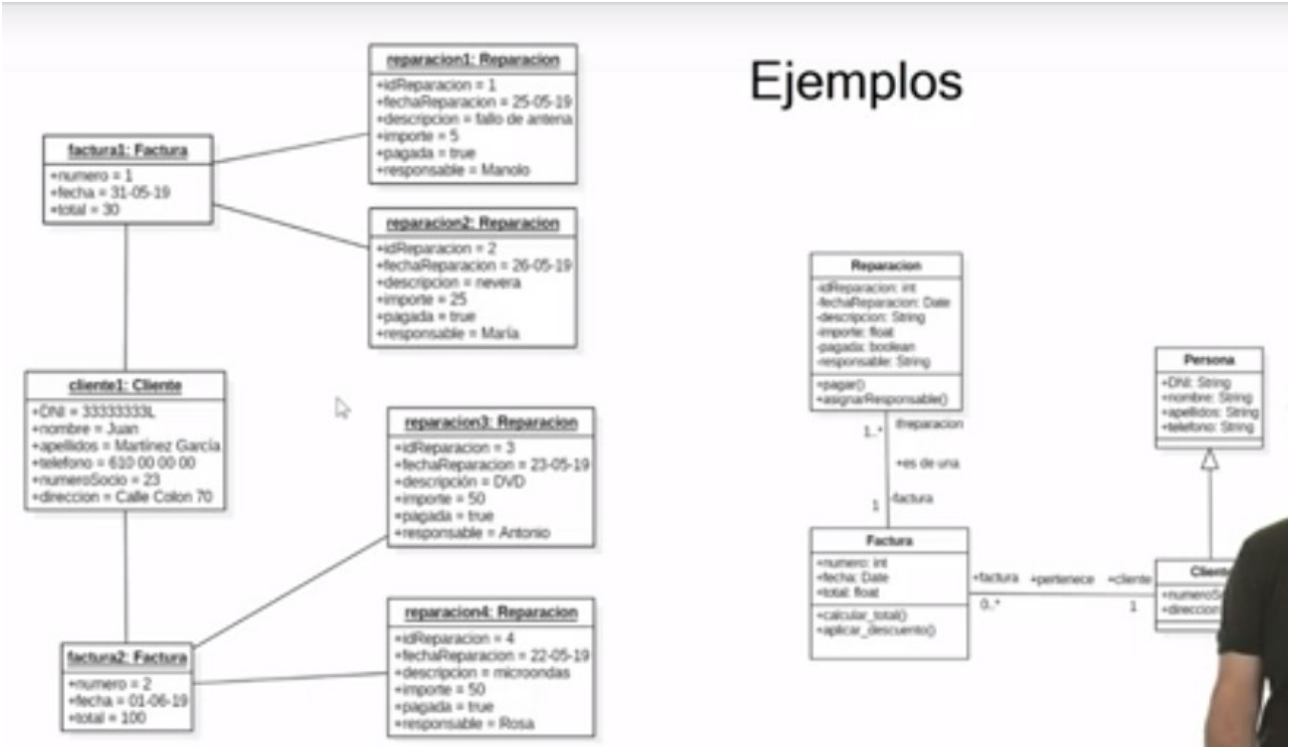
¿Cómo representamos los objetos en UML? Pues lo tenemos aquí a modo de ejemplo



Lo que está a la parte izquierda de dos puntos es el nombre de la instancia, que suelen empezar con letras minúsculas y a la derecha, tenemos el nombre de la clase, que suele aparecer con nombre mayúscula la primera letra. Todo esto es un objeto de una clase de hecho esta clase cola puede tener n objetos, n instancias en este caso tenemos únicamente una.

Si no queremos poner nombres a las instancias, dejamos como está aquí en el segundo ejemplo, como anónimo. Pues sé que esto aquí es una instancia de marco, de la clase marco, pero no le he

puesto ningún nombre por tanto pues, se que esta instancia se ha instanciado de la clase marco, pero no tiene nombre. Vamos a ver un ejemplo.



Es el ejemplo que hemos usado anteriormente a lo largo del curso. Aquí en la parte derecha tenemos el diagrama de clases, que ya hemos visto, serían como los moldes de los objetos que vamos a instanciar y aquí la parte izquierda, tenemos las instancias de ese modelo de clases cosa que igual les llama atención así de antemano.

La primera, aquí no tenemos cardinalidades. Aquí sí que había cardinalidades, aquí no las hay. Por qué no hay cardinalidades en un diagrama de objetos? Porque realmente no hacen falta, aquí en el diagrama de clases como no sabemos cuántas instancias vamos a tener de cada clase, tenemos que poner la cardinalidad.

¿Qué quiere decir esto? que reparación puede estar solamente con una instancia factura y factura puede estar con una o muchas reparaciones. Pero si vamos al diagrama de objetos, esas relaciones son ya instancias particulares. Ya no específico que se puede relacionar con cero a muchas, o no, si se relaciona con cero a muchas tengo que especificar cuántas relaciones hay, si hay dos, si hay tres o si no hay ninguna.

Vamos a ver el caso particular. Aquí tengo una factura que he llamado factura uno. Este es el nombre del objeto y este me de la clase y esa factura la relaciono con dos reparaciones. En este caso, la factura puede tener una o muchas reparaciones, pero en mi caso particular de mi instancia, lo he puesto con dos relaciones, reparación uno y reparación dos.

Además, tengo que la factura pertenece a un único cliente.

Si lo instancio en mi ejemplo particular, tengo la factura relacionada con el cliente. Además, ese cliente puede estar relacionado con más de una factura, de cero a muchas facturas, como es el caso

de ejemplo. Este cliente uno está con la factura uno, pero también con la factura dos tiene dos relaciones en este caso particular.

Y esta factura dos, por supuesto, como indica aquí este diagrama de clases, esta con reparaciones distintas a las anteriores. Pues en este caso de ejemplo, he puesto dos reparaciones más. Por tanto, en el diagrama de clases y cardinalidades lo que tengo que especificar relaciones genéricas, no sé cuántas se van a instanciar, pero en el caso del diagrama de objetos, como ya tengo objetos concretos, instancias concretas, ya sé cuántas relaciones tengo que poner, entonces no es de cero a muchas, es dos, tres, ninguna, pero sé exactamente cuántas tengo. Por tanto, para cada una, dibujo la línea de relación entre ellas y ya está.

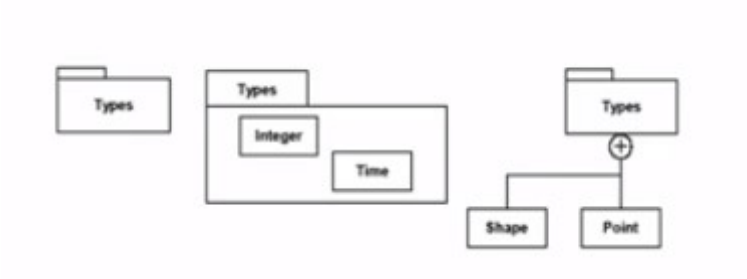
Luego también una cosa que nos puede llamar la atención es que aquí tenemos atributos, pero tenemos algo más, tenemos valores. Aquí en el diagrama de clases, indicamos el tipo de cada atributo, pero no damos ningún valor. Aquí en este caso ya estamos dando valores, estamos dando ya información particular de cada objeto. Pues por ejemplo, la reparación pues tenemos un identificador, la fecha en la que se hizo, en qué consiste la la reparación, lo que ha costado, si está pagado no y luego responsable técnico que la atendido. Si se fijan, esto de aquí es un objeto concreto de esta clase que es el molde, tengo un molde que es reparación y instancio objetos particulares de ese molde. Cada instancia, si se fijan, comparte las misma cualidades, que quiere decir, que comparten los atributos. Cada uno tiene sus valores por ejemplo, está aquí reparación uno, esta es dos, esta es tres y esta es cuatro. Cada una tiene sus valores particulares, pero todas ellas comparten las mismas características, los mismos atributos y además también compartirían las mismas operaciones.

El diagrama de objetos es un diagrama un poco especial ¿Cuando se hace un diagrama de objetos? Se hace cuando yo quiero probar si mi diagrama de clases tiene sentido no. Por lo general, el diagrama de objetos no se hace, se hace básicamente para probar que aquello que he diseñado en el diagrama de clases puede funcionar pero no tiene sentido hacer un diagrama de objetos para hacer sistema, básicamente porque no cabe en una pantalla normal.

Paquetes

El paquete no es más que un elemento organizativo que lo que permite es distribuir las clases en un diagrama bastante grande, pensar que en un sistema real podemos hablar de fácilmente de treinta y sesenta clases en un diagrama de clases y si no se divide en pequeños contenedores, pues aquello es inmanejable, ¿no? Entonces, estos contenedores que agrupan clases se llaman paquetes, simplemente pues agrupan clases en base a la semántica que puedan tener estas clases. Cada paquete tendrá un nombre que lo distingue, los paquetes se pueden anidad, que quiere decir que yo puedo tener un paquete muy grande y dentro de este paquete grande puedo tener más subpaquetes, lo que quieran, no hay un máximo de anidación se puede anidar todos los paquetes que quieran.

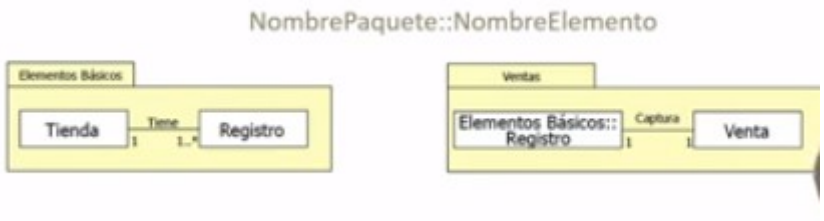
Otra cualidad es que también permite definir la visibilidad de las clases o paquetes que hay dentro de un paquete, acuerdense que vimos tres tipos de visibilidades en temas anteriores, que es el público, privado, protegido, lo mismo que aplicabamos para las clases aplicaría para el paquete. Pues la clase pública es visible para todos, la protegida solo herencia y la privada sólo para elementos propios del mismo del paquete al igual que si fuera una clase.



Aquí vemos tres tipos de anotaciones para los paquetes, la primera es la más simple, indica que tiene un paquete types pero no sé lo que tengo dentro. Aquí tenemos un paquete types pero indicando las clases que están dentro, pues tengo la del paquete types con dos clases, integer y time, son dos clases dentro de este paquete. No quiere decir que aquí no haya clases, simplemente quiere decir que están ocultas, son dos vistas distintas de un mismo paquete, y en el último caso tengo otra tercera vista que indica que tengo dentro del paquete types dos clases, shape y point. La notación entre este tipo y este tipo distinto, vale, la sintaxis distinta pero la semántica es la misma, representan lo mismo, vale, simplemente cambia como se dibuja en UML, pero las dos anotaciones son UML se puede tomar la que prefieran.

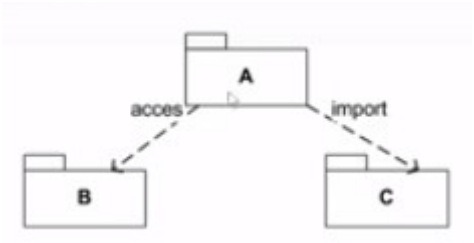
El paquete como he dicho principio lo que hace es definir un contenedor donde dentro tengo otros paquetes u otras clases, las clases que tengo dentro paquete crean relaciones, por ejemplo tengo una tienda que tiene de uno a muchos registros, pues aquí tengo un paquete. ¿Qué ocurre si yo quiero hacer uso de una clase que está en otro paquete?, bueno para eso esta espacio de nombres, el espacio de nombres me indica el nombre del paquete que yo quiero incluir en el paquete actual, por ejemplo yo tengo aquí el paquete ventas y quiero hacer uso de una clase que no están ventas, está en otro paquete que se llama elementos básicos, en concreto este va a hacer uso de la clase registro.

¿Qué es lo que hago?, pongo el nombre del paquete y que quiero referenciar el espacio de nombres serían elementos básicos, este nombre es el nombre de este paquete, dos puntos dos puntos nombre de la clase a la que quiero referenciar, por tanto en el paquete ventas puedo hacer uso de la clase registro que está en el paquete elementos básicos, de esa forma puedo separar las clases en varios paquetes y referenciarlas, pensar que al fin y al cabo un diagrama de clases no son clases aisladas sino que son clase que están relacionadas de una u otra forma.

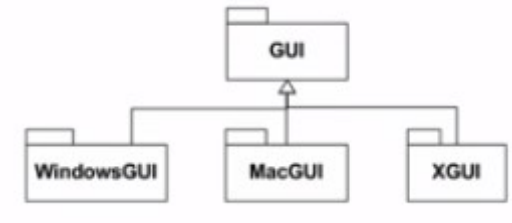


Entonces si divido las clases en paquetes las tengo que relacionar de alguna forma, cómo la relaciono, con el espacio de nombres. Aquí vemos un poco la finalidad de este diagrama de paquetes, básicamente es para facilitar el trato diario con el diagrama, son diagramas muy grandes que se particionar en pequeños paquetes. ¿En base a qué?, bueno pues en base un poco a la actividad que veas entre aquellas clases que tenga más sentido agrupar porque están muy

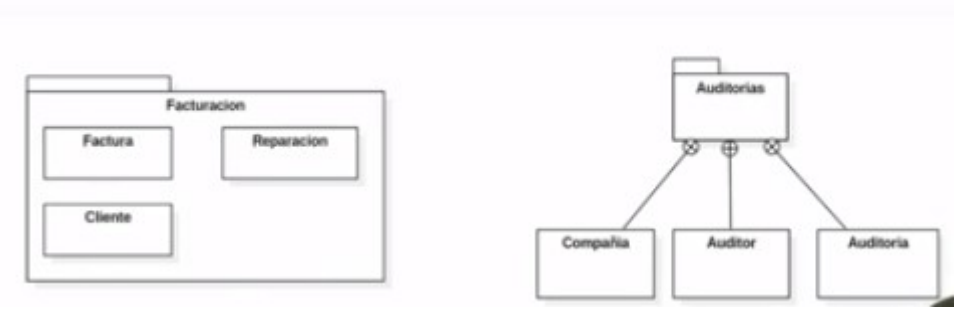
relacionadas o porque semánticamente tienen sentido las pondré el mismo paquete, es más comprensible, más fácil de manejar además aporta también el nivel de ocultación más allá del nivel de clase, y luego entre paquetes hay varios tipos de relación está el importación/exportación, acceso y herencia o generalización, los tipos de dependencia lo veremos en otro tema.



Aquí vemos un ejemplo de paquetes, pues tengo el paquete a que accede a las clases del paquete b, vale, quiere decir que a puede hacer uso de ese paquete, de esas clases, y además a importa también clases del paquete c, la diferencia que hay entre el acceso la importación la veremos otro tema, pero digamos que de alguna forma a puede acceder tanto las clases están en b como la clases que están en c, puede acceder a ambas clases. Luego aquí vemos otro ejemplo de referencia de paquetes pero en este caso son de herencia, vale, pues tengo un paquete que es el GUI del queda hereda WindowsGUIaquí, MacGUI y XGUI.

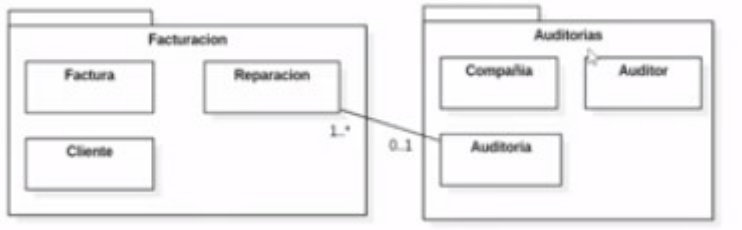


¿Qué quiere decir esta anotación?, pues que las clases que tengan estos paquetes hijo redefinirán o aportarán más información de la que tengo en el padre, pues igual que si fuera una relación de herencia entre clases, recuerden que la relación de herencia lo que permitía a las clases hijas era redefinir operaciones o añadir atributos y operaciones, en este caso es igual, puede redefinir las clases que tengo en el padre o puedo añadir nuevos elementos.



Este es ejemplo particular de nuestro curso, el del ejemplo facturación, pues tenemos un paquete de facturación con tres clases, factura, reparación y cliente y luego tengo otro paquete con otra anotación distinta para qué vean el tipo de anotación, es el paquete auditorías con tres clases,

compañía, auditor y auditoría, pueden quedar con la anotación que prefieran, no tienes por que tomar distintas anotaciones, puedes tomar la anotación más tradicional o esta anotación de aquí, aquí he puesto las dos para que vean dos ejemplos distintos pero vamos, no tienes por que mezclar anotaciones.



De aquí la relación que hay entre las dos, entre los dos paquetes, pensar que además de poder usar los nombres de paquetes, si la herramienta gráfica lo permite pueden también relacionar clases directamente entre paquetes, si no es muy grande el diagrama se puede directamente relacionar las clases entre paquetes, si no fuera posible porque la herramienta no lo permite o porque visualmente pues no está en el mismo fichero habría que hacer uso del espacio de nombres que hemos comentado antes.

Aquí vemos lo que tenemos dentro de cada paquete, pues facturación teníamos las clases relacionadas con la facturación, estaría reparación, factura y cliente, que son clases que están dentro del paquete facturación, y en el paquete auditorías pues tendríamos las clases relacionadas con la funcionalidad de la auditoría que es auditoría, compañía y auditor, vale, estas tres clases irían dentro del paquete auditoría.

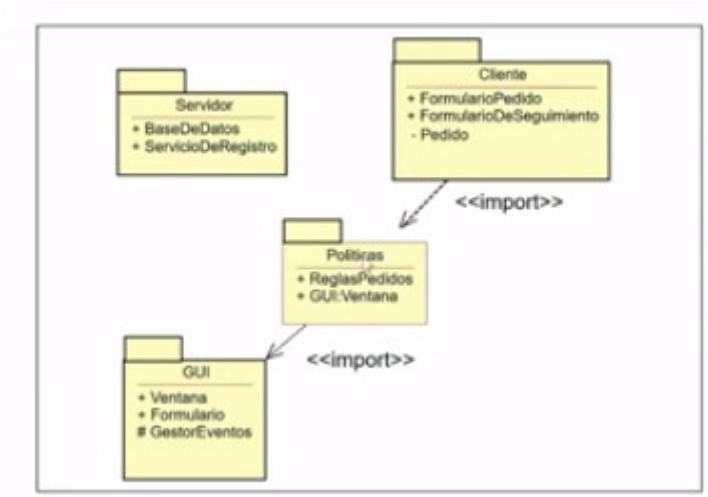
Dependencia entre paquetes

Hay dos tipos de dependencia:

- importación/exportación, quiere decir que si yo tengo un paquete a e importo un paquete b, el paquete b es como si estuviera totalmente dentro del paquete a, vale, además lo introduzco de forma pública, ¿qué quiere decir? Que el paquete que importa otro paquete a su vez puede delegar es importación a otro paquete que lo importe, además esta importación tiene una, una particularidad respecto a otro tipo de relaciones y es que no requiere que se especifique el espacio de nombres, si yo tengo una clase a que importa una clase b puedo hacer uso de la clase b sin referenciar el espacio de nombres de la clase b, vale, no tengo invocar espacio nombre directamente la importa y es como si estuvieran las clases definidas dentro de la clase a.
- acceso, ¿en qué se diferencia el acceso de la importación? El acceso es más restrictivo, así por la importación es transitiva, el el acceso no, que quiere decir esto, la transitividad quiere decir que si yo tengo una clase a que importa una clase b, esa importación la puedo delegar a otra tercera clase, si es de acceso no. Si yo accedo una clase b, no la puedo acceder o exportar a una tercera clase, el acceso es más restrictivo. ¿Por qué?, porque el acceso hace que lo que yo incorpore a mi paquete sea privado, así como la importación era público, el

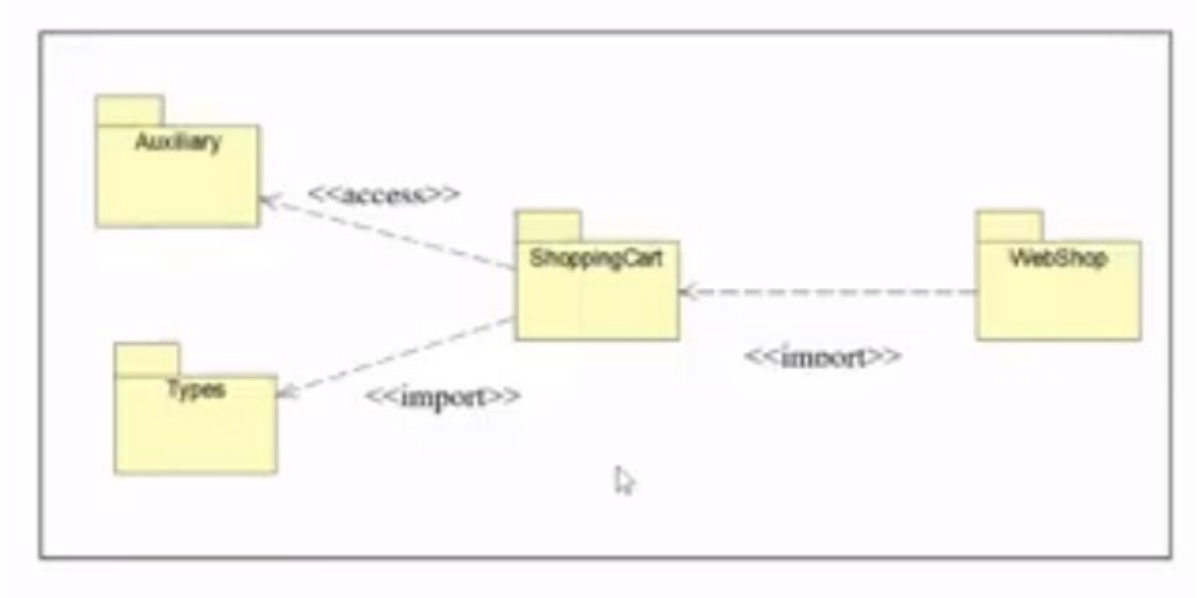
acceso es privado, por tanto cuando yo accedo a un paquete accedo a él pero de forma privada, no la pudo exportar, además en el caso de acceso si que tengo que especificar el espacio de nombres, en importación no hacía falta, pero en acceso sí, si yo quiero hacer uso de un paquete al cual he accedido tengo que poner delante el nombre del paquete al que voy acceder.

Vamos a ver ejemplos. Aquí por ejemplo tengo el paquete cliente que importa el paquete políticas que a su vez importa el paquete GUI, vale, ¿esto qué quiere decir? Que el paquete políticas ha incorporado a su conjunto de clases todas las clases del paquete GUI y los ha incorporado de forma pública, de forma que cliente al importar las políticas, por transitividad, está importando también las clases del GUI, por tanto cliente puede hacer uso tanto de las clases de políticas como las clases de GUI porque aquí tengo relaciones tipo import, que son transitivas.



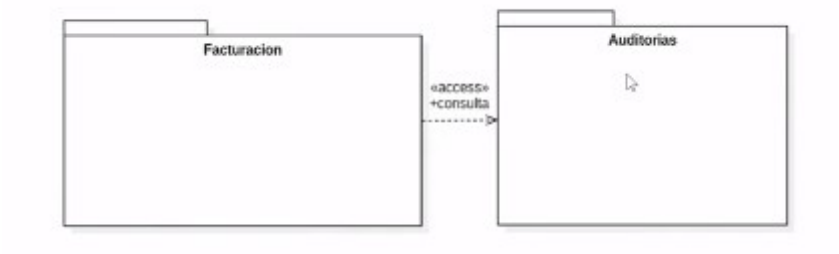
Además, cliente puede hacer uso tanto de las clases de políticas como de GUI sin especificar el nombre del paquete al cual quiere acceder, únicamente con el nombre de la clase es suficiente, porque la dependencia es de tipo import.

Vamos a ver otro ejemplo, en este caso combinamos import con access.



Tengo la, el paquete WebShop que importa el paquete ShoppingCart y este por una parte accede a Auxiliary y por otra parte importa el paquete Types. ¿Qué quiere decir esto?, que ShopingCart puede hacer uso de las clase que tengo en Auxiliary, puede hacer uso pero no las puede por transitividad delegar a WebShop, vale, WebShop no puede hacer uso de las clases de Auxiliary, ShoppingCart sí, pero WebShop no, ShoppingCart accede a Auxiliary y puede hacer uso de esas clases, pero no las puede pasar a WebShop. En el cambio de types sí que transitividad, WebShop puede hacer uso de ShoppingCart y también puede hacer uso de Types, vale, a través de import, por transitividad, puede acceder a los dos paquetes.

Además, WebShop puede hacer uso de types, de ShoppingCart sin especificar el espacio de nombres, pero si quiero hacer uso de clases de Auxiliary, tiene que indicar el espacio de nombres de este paquete, se pondría Auxiliary dos puntos dos puntos nombre de la operación que queda invocar de la clase.



Aquí vemos un ejemplo de acceso entre dos paquetes, pues tenemos una relación entre paquete facturación y auditorías, pues imaginaros que desde facturación queremos hacer una consulta de cuantas auditorías hay, pues habría que poner un acceso antre, entre los paquetes que hemos especificada antes

Por último aparte de import y access tenemos la generalización, la herencia. Este tipo de relación está pensada como tengo un paquete más generalista, más general, y varios paquetes más específicos que hereda de esos paquetes más generales. En este caso de ejemplo, tengo el paquete padre que es GUI y dos paquetes hijos WidnowsGUI y MacGUI, vale, en este caso puedo redefinir clases que yo tenga definida mi paquete padre, por ejemplo formular ya está redefinido, vale, meintras que ventana pues la ha heredado tal cual, no la ha querido modificar, el formulario si.

Y aquí pues no hemos especificado ningún cambio por tanto se mantiene la mismas clases que hemos definido la clase padre. Vale la herencia está pensada para o bien heredar la especificación de la clase padre o bien heredar y aplicar cambios a esa clase, a esa, a ese paquete padre.

¿Cómo divido las clases en paquetes?, pues esto depende cada clase particular por algunas ideas podemos ver a continuación, pero depende de cada caso en particular. Por ejemplo, en base al área de interés, en base a la jerarquía por si hay una relación de herencia entre clases conviene ponerlas juntas en un paquete, si las clases vienen derivadas de un mismo caso de uso, pues si tienen una funcionalidad bastante relacionada entre si, conviene ponerlas un paquete junto o si hay muchas relaciones de asociación, pues si son clases muy relacionadas entre sí, conviene ponerlas juntas.

La idea es que los paquetes sean lo más autónomos posibles, pues que no haya mucha relación entre paquetes, pensar que también cuanto más relación entre paquetes también se complica bastante el mantenimiento. Y luego evitar algo, relaciones circulares, pues que quiere decir esto, pues que yo tengo un paquete a con un paquete b y un paquete c que no es circular la relación sino que haya únicamente un sentido.

Nota: Una relación de tipo > entre un Paquete A y un Paquete B implica que el Paquete A puede acceder a todos los elementos públicos del Paquete B

Introducción a Diagrama de Clases

Un diagrama de clases no es más que un conjunto de clases relacionadas entre si, es un grafo bidimensional con clases que se relacionan en un sentido y en otro.

Tenemos varios tipos de vistas de diagrama, puede ser un diagrama muy detallista a nivel de implementación con mucho detalle o puede ser un diagrama más abstracto que es el modelo conceptual, ahora vamos a ver los tipos diagramas de clases que hay, en base a la fase que estén de trabajo pues será un diagrama más generalista o un diagrama más específico. Estos diagramas incluyen sobre todo las clases de nuestro sistema, las interfaces que puedan tener, relaciones que haya entre estas clases, pues de herencia, de asociación, de composición, y luego también si lo ven adecuado pueden incluir paquetes para agrupar estas clases de alguna forma.

Qué uso damos a un diagrama de clases, pues el uso principal es definir la solución a un problema, pues yo defino conceptos donde cada concepto es una clase y relaciono entre esos conceptos, pues yo defino cuál es la solución a un dominio concreto. Además podemos ver el tipo de relaciones que hay, cardinalidades, atributos de cada clase, que son sus características, operaciones que puedo lanzar sobre cada clase, además viene indicado también qué vocabulario va a tener mi sistema, el nombre de las clases me va a indicar también el vocabulario junto con atributos, pues esos atributos, lo normal es que en esos atributos tengan nombres significativos de la semántica de esa clase.

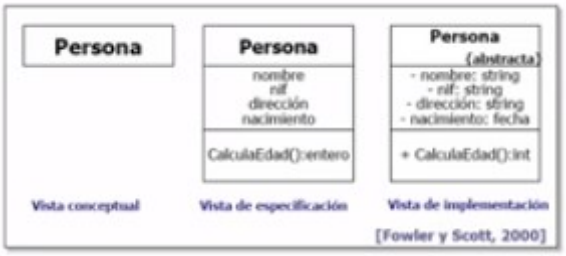
También pensar que la base de datos se puede derivar de un diagrama de clases, de hecho hay herramientas que automáticamente a partir de un diagrama de clases te generan las tablas de una base de datos, por tanto el tiempo que inviertan en definir un diagrama de clases es tiempo que pueden ahorrar luego en el diseño de bases de datos. Y luego hay como dos caminos para definir un diagrama de clases, es ingeniería directa e inversa.

Ingeniería directa quiere decir que yo primero hago el diagrama de clases y una vez está hecho, paso a implementación, o inversa que es que tengo el código y a través de ciertas herramientas, a partir de ese código, automáticamente se me dibuja el diagrama de clases. Lo lógico es hacer ingeniería directa, pero muchas veces llega a nuestras manos código que han hecho terceras personas y queremos generar el diagrama de clases, también se puede hacer mediante ingeniería inversa. La ingeniería inversa a veces funciona, a veces no, hay veces que se pierde información pero bueno, está bastante automatizada.

Ideas o trucos que les puedo aconsejar cuando hagan un diagrama de clases, la primera es que no se preocupen excesivamente en intentar meter todas las clases que tengan que estar. Por ejemplo, en las primeras fases cuando estamos trabajando con un modelo conceptual, estamos diciendo conceptos, estamos pensando en que es lo que vamos a hacer, no tenemos mucho detalle. En esos casos si nos faltan clases tampoco es grave, por supuesto si estamos ya una fase muy final, estamos implementando, pues si ahí se nos olvida poner alguna clase es más importante, pero fases iniciales a nivel de modelo conceptual, si se nos olvida poner alguna clase tampoco son una cosa grave. Luego también es muy importante que pongan un nombre representativo de lo que representa esa clase, pensar que el que lea el diagrama puede que sea una persona que no participe en su diseño, por tanto los nombres tienen que ser representativos de la clase y luego también evitar los cruces, las relaciones entre clases, cuando hay muchas clases hay muchas relaciones, entonces si hay cruces y pasa una línea por encima de una clase se emborrona todo y luego es difícil entender, por tanto yo aconsejo que eviten los cruces o líneas que pasen por encima de otras líneas o por encima de las clases porque luego se complica bastante a la hora de entender el diagrama.

Niveles de representación de los diagramas de clase:

- el conceptual, el detalle es más generalista y se hace en primeras fases desarrollo es es un diagrama donde tengo conceptos y relaciones pero no tengo ni tipos de datos ni operaciones, es una fase muy temprana donde aún no me he puesto a pensar en esos detalles.
- especificación, que sería un avance más en detalle, en este caso si que tengo operaciones aunque sean operaciones sin especificar a nivel, es como si fuera una interfaz. Pues tengo claro qué operaciones me hacen falta, pero no como las voy a hacer
- implementación, en este nivel ya sí que te poner tipos y tengo que darle contenido o cuerpo a esas operaciones que voy a implementar.



Aquí vemos un ejemplo de los tres tipos de clases que puedo dibujar en mis diagramas, pues una primera fase sería una clase a nivel de modelo conceptual, pues tengo el nombre de la clase y si quieren pueden poner también atributos, aunque sin tipo, por si queréis poner atributos sin tipo.

A nivel de vista de especificación, tenemos la clase con el atributo sin tipo también, que si se fijan ya tenemos operaciones, aunque únicamente es la signatura de la operación, que es el nombre operación, los parámetros y el tipo de retorno.

Y luego tenemos aquí el nivel de implementación, donde ya sí que pongo tipos y además los tipos de la operación son tipo particulares de lenguaje implementación, pues aquí es entero, que esto no es lenguaje en java, pero si voy a implementar para java, a nivel de vista implementación tengo que poner el tipo con el que trabaja java, que sería int, digamos que esto de aquí es una nomenclatura más abstracta y esto de aquí es una nomenclatura específica del lenguaje de implantación que voy a utilizar.

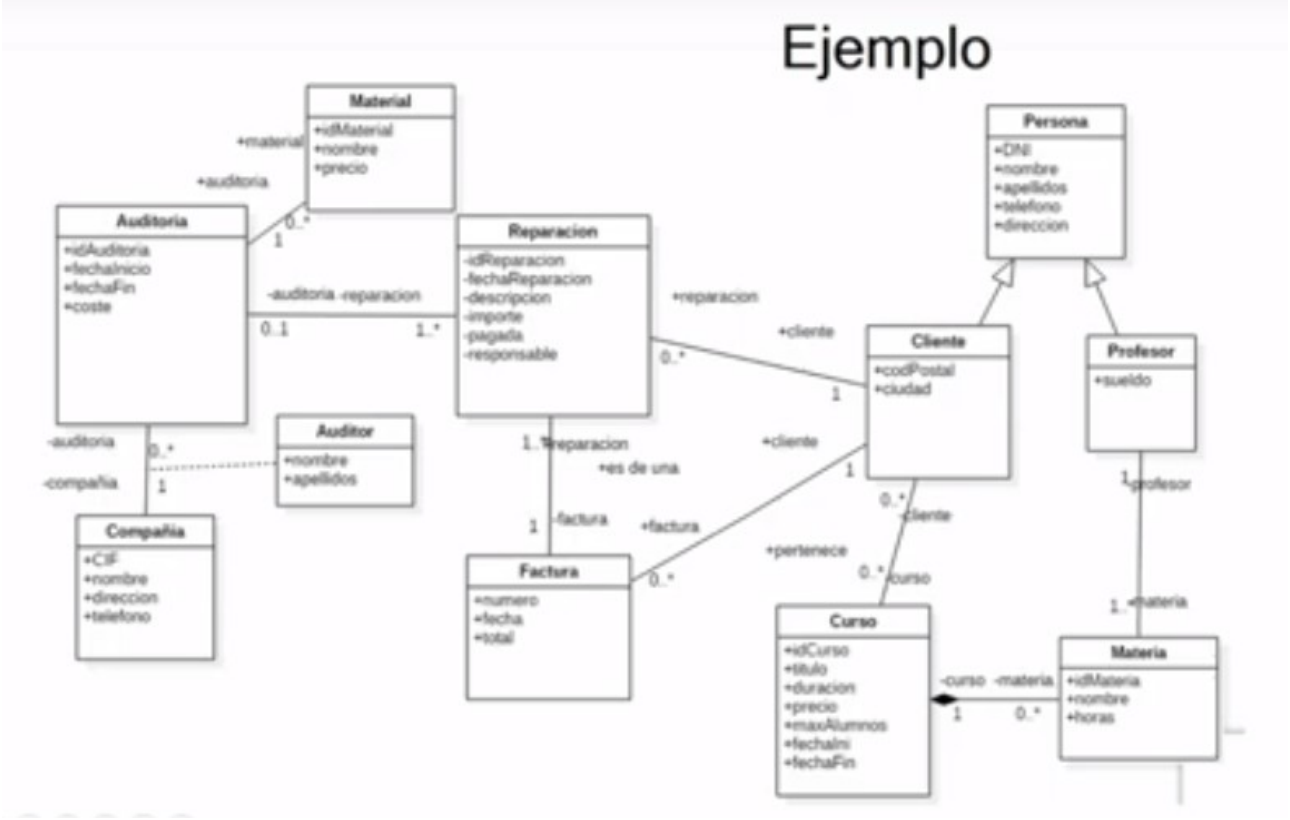
Usos en las diferentes fases del proceso del software:

- el modelo conceptual es como he dicho antes útil sobre todo al nivel de primera fases, para ver los tipos de conceptos que hay y sus relaciones
- luego pasamos a un modelo de análisis, donde ya tengo clases con atributos y puedo añadir métodos aunque no estén implementados
- modelo de diseño ya tengo que introducir más elementos, por ejemplo que patrones diseño utilizar, si tengo que definir estructuras de datos, si tengo que tener persistencia, como me conecto con bases de datos, si tengo elementos distribuidos, pues eso ya es, a nivel diseño que empezar a pensar.
- modelo de implementación tengo que pensar en cómo distribuir las clases acorde al lenguaje de implantación con el que voy a trabajar, por ejemplo, si voy a trabajar con java y necesito usar la librería timer de java, la librería timer la tengo que incorporar a mi diagrama de clases, ¿por qué? Porque aunque no la implemente yo, tengo que indicar que voy a hacer uso de esa clase, por tanto aquí a nivel de implementación tengo que poner todas las clases que voy a utilizar, tanto si son mías como son particulares de un lenguaje en particular, por ejemplo java.

Modelo conceptual

Lo vimos un poco por encima en sesiones anteriores y hoy lo vamos a resaltar todavía más. Un modelo conceptual no es más que un diagrama de clases muy abstracto, tenemos atributos pero no tenemos tipos ni tampoco tenemos operaciones.

Está pensado para trabajar con él en las fases iniciales de desarrollo, donde todavía el requisitos no están muy cerrados. En estas fases iniciales, el analista se debe centrar en lo que es el problema a diseñar, no la solución concreta, por tanto, este diagrama está pensado únicamente para pensar en el problema a un nivel muy abstracto. Vamos a ver a continuación el ejemplo que hemos usado en todo el curso, es el ejemplo la facturación.



En este caso tenemos aquí la materia que pertenece a una auditoría, pues tenemos material está en una auditoría, del material tengo su identificador que se supone que es único, no se puede repetir, el nombre y el precio, y la auditoría tiene un conjunto de materiales. En este caso tienen el mismo identificador, el cual es el ID, la fecha de inicio, la fecha de fin y el coste. Además aquí tenemos un ejemplo de una clase asociación, que es el auditor. La auditoría se desarrolla en una compañía, y se desarrolla si y sólo si hay un auditor, si se fijan si hay una relación entre auditoría y compañía, es porque hay un auditor, no puedo tener, según el diagrama, una relación entre auditoría y compañía si no tengo un auditor. Además, el auditor únicamente puede existir en mi sistema si tengo una auditoría en una compañía, el auditor como no está relacionado con ninguna otra clase, únicamente tiene sentido mi sistema si tengo un auditoría con una compañía en mi información. Además, auditoría se hace sobre reparaciones, tengo el conjunto de reparaciones que están relacionadas con auditoría. La reparación puede que tenga auditoría o puede que no la tenga, en reparación tenemos

aquí sus atributos, para saltar por ejemplo la fecha, si está pagada o no, el importe y una cadena de texto que es responsable.

La reparación pues se tendrá que recibir una factura, si decimos que la factura de reparación, tenemos aquí una clase de tipo factura y la factura se define con un conjunto de reparaciones, además tanto reparación como la factura tiene un cliente. Luego aquí tenemos una herencia, que podemos indicar que aprovechamos tanto el DNI como el nombre como apellido como teléfono como dirección para las clases tipo cliente y las clases tipo profesor.

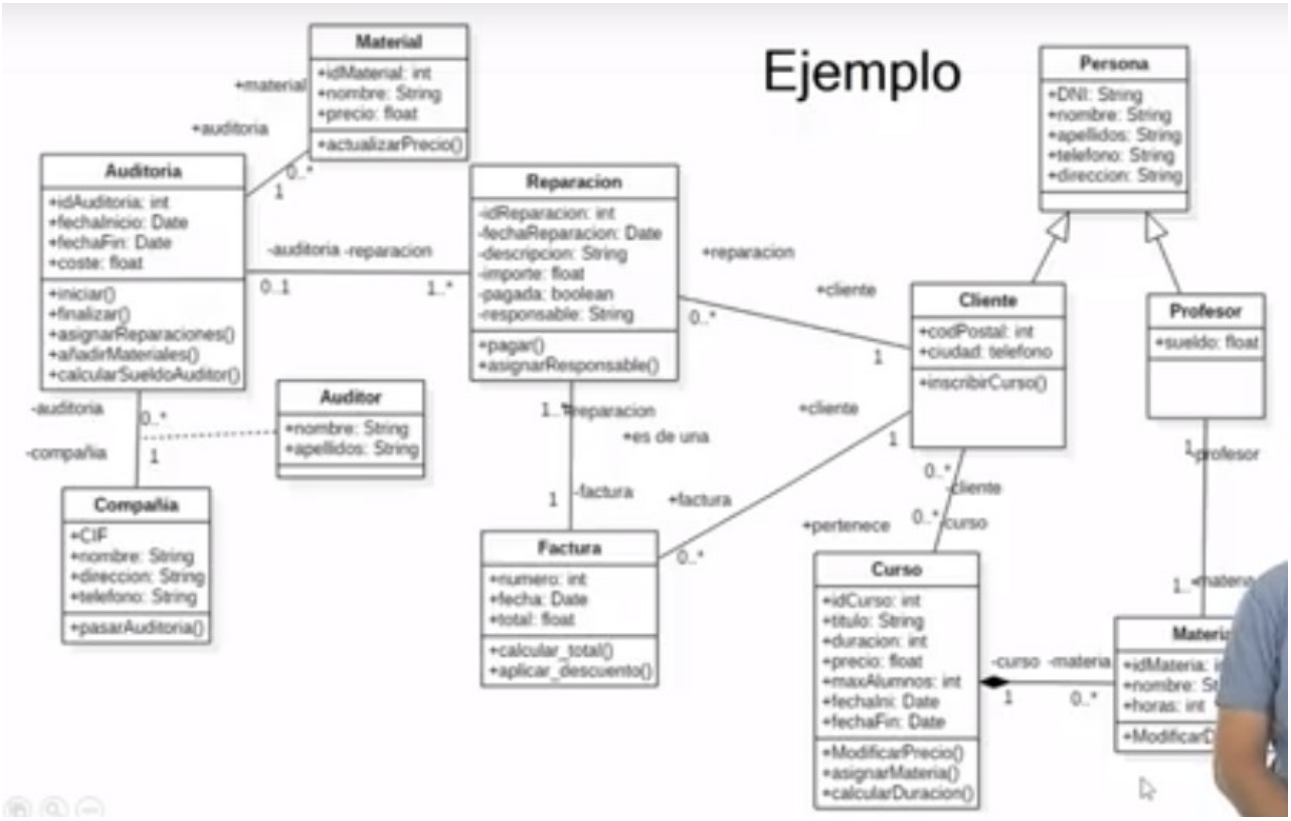
Además tenemos que cliente más allá de lo que comparte con profesor, tiene como atributo particulares el código postal y la ciudad, estos atributos son específicos de cliente, no están en profesor. Además el profesor tiene un atributo particular que es sueldo, y por último tenemos aquí que el curso tiene matriculados un conjunto de clientes y el cliente puede estar en más de un curso. En el curso tenemos duración, el precio, la cantidad máxima de alumnos que se pueden matricular, la fecha de inicio y la fecha de fin. Y además el curso está compuesto de materias, vale, la materia la imparte un profesor. Pues a groso modo este es un ejemplo puesto en práctica de lo que sería un modelo conceptual aplicado a nuestro ejemplo del curso.

Modelo de Diseño

El modo de diseño es un modelo de clases más enfocado a la solución concreta que vamos a implantar implementar.

Así como el modelo conceptual, era un modelo más abstracto, independiente de la solución concreta, el modo de diseño va mucho más enfocado a lo que es la solución que estamos planteando. Por tanto, es un modelo que está muy asociado al lenguaje de programación que al final vamos a utilizar. Por ejemplo, si usamos java para nuestro diseño, en un modelo de diseño debemos de poner aquellas clases de java que soporta nuestra solución. Por ejemplo, si tenemos usar un timer para medir el tiempo de una funcionalidad, deberíamos incorporar la clase timer, aunque no fuera una clase que nosotros vayamos a diseñar, pero sí que es una clase que es necesaria para el buen funcionamiento del sistema y por tanto, debería aparecer en este diagrama. Además, en este tipo de diagramas, ponemos ya también, tipos a los atributos.

Aquí ya ponemos tipos a, si es un entero, un string, un boleano o un float. Y además, también debemos de poner en este tipo de diagramas, los métodos, las operaciones aunque los parámetros sin retorno son más opcionales, por lo menos el nombre de la operación sí que es ya bastante obligatorio. Pensar que en estos niveles de diseño tenemos que pensar ya de que métodos vamos a implementar en mi código para que la funcionalidad del sistema satisfaga los requisitos.



Bien, pues a modo de ejemplo, vamos a hacer un modelo de diseño de nuestro caso particular de la facturación. Lo que son las clases y atributos son lo que ya vimos en el modelo conceptual, pero en este caso concreto, hemos añadido tipos de atributos y hemos añadido operaciones. Estas operaciones lo que van a hacer es modificar el valor de los atributos y también, en algunas ocasiones, modificar la relación que tengo entre clases. Por ejemplo, aquí en material, tengo un método, una operación, que es actualizar precio. Este método, esta operación tiene como fin último modificar el atributo precio.

En auditoria tengo varias operaciones. Iniciar, quiero que ahora será dar un valor la fecha de inicio, finalizar, que dará un valor a la fecha de fin. Asignar reparaciones, que lo que hará será crear relaciones entre auditoría y reparaciones. Añadir materiales, que lo que hará será añadir elementos que relaciona auditoría con material. Y por último, tengo calcular sueldo auditor, que lo que hará será hacer un cálculo de el coste de la auditoría. Si se fijan, como he dicho antes, cada método tiene como fin último dar un valor a los atributos de la clase o bien modificar las relaciones de esa clase con otras relaciones. En compañía, tengo una operación, que es pass auditoría, que lo que hará será relacionarme la compañía con la auditoría y el auditor.

En cuanto a reparación, tengo el método pagar, que lo que hace es convertir este este tributo pagado a verdadero cambia el estado de falso verdadero. Y luego asignar responsable, que lo que hará será asignarme un string al atributo responsable modificar el string en base a lo que indiquemos en la operación.

En factura tengo una operación calcular total, que lo que hace es dar un valor al atributo total y aplicar descuento que lo que haría sería aplicar un porcentaje de descuento a ese total de la factura, pues en base en base a cierto algoritmo que tuviéramos implementado en nuestro código.

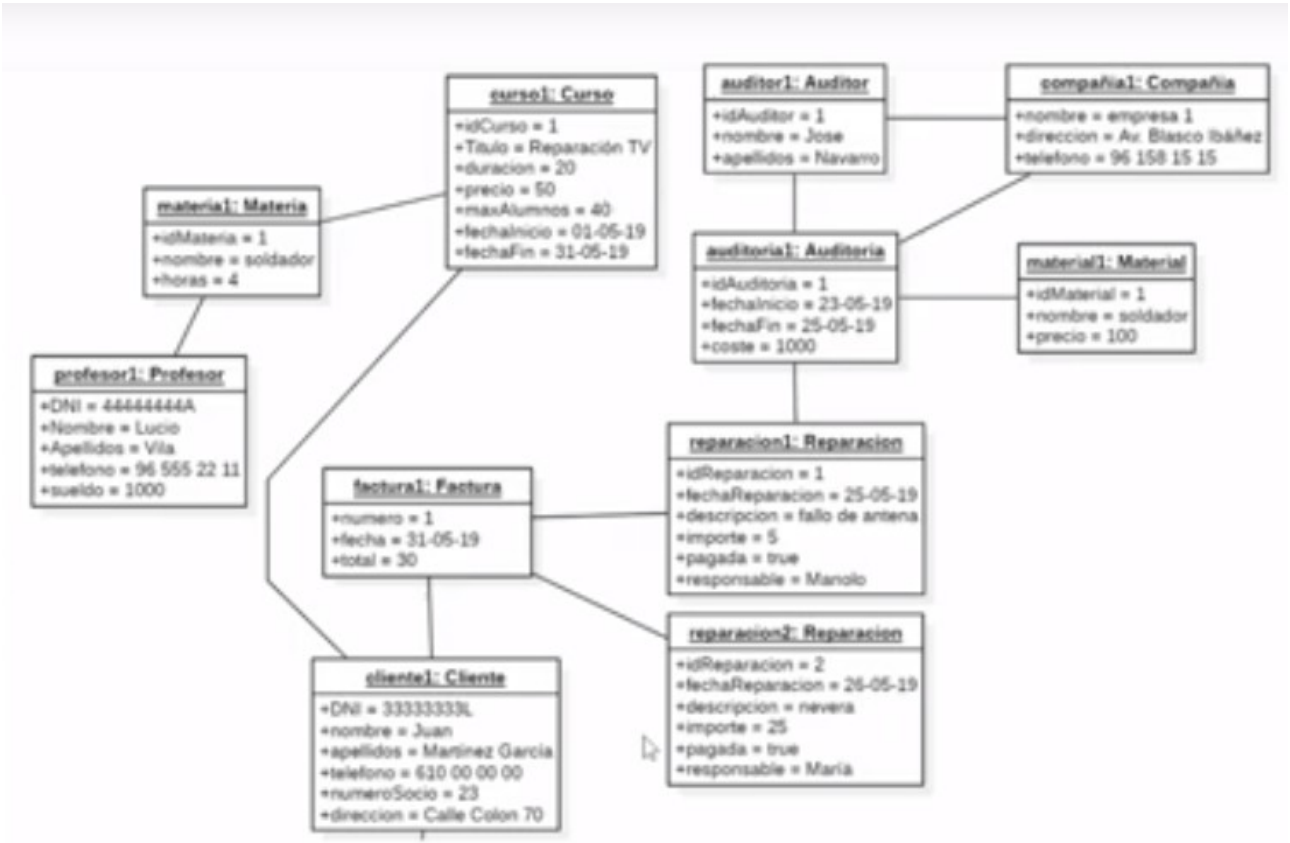
En cuanto al cliente tenemos inscribir curso, que lo que hace es relacionarme cliente con el curso. En curso tenemos modificar precio, que permite modificar el atributo precio que tenemos aquí, un float. Asignar material que lo que hace es modificar relación que tengo perdón, sería materia, que lo hace relacionar el curso con la materia y tengo aquí calcular duración que lo que hace también es calcular cuál es la duración de este curso, que no es más que la suma de las horas de la materia. La duración sería la suma de todas las horas de las materias que componen el curso.

Como ven ese mismo diagrama que hemos definido en el modelo conceptual, pero mucho más enfocado a lo que va a ser mi implementación concreta con un lenguaje de programación.

Diagrama de Objetos

Un diagrama de objetos lo que representa son las instancias de un diagrama de clases. Instancias en el sentido de objetos concretos que se crean en un instante concreto. Así como el diagrama de clases vale para cualquier para representar cualquier tipo de instancia, el modelo de objetos representa unas instancias concretas.

Se usa básicamente, a modo de ejemplo, para ver si el modelo de clases que hemos definido funciona correctamente o no. Más allá de una utilidad de verificar que lo que hemos representado en el diagrama de clases tiene sentido, la verdad es que no tiene mucha utilidad este tipo de diagramas.



Aquí vemos el ejemplo de un diagrama de objetos sobre el ejemplo del curso de facturación. Destacar como diferencia entre el diagrama de objetos y el diagrama de clases, que aquí en este

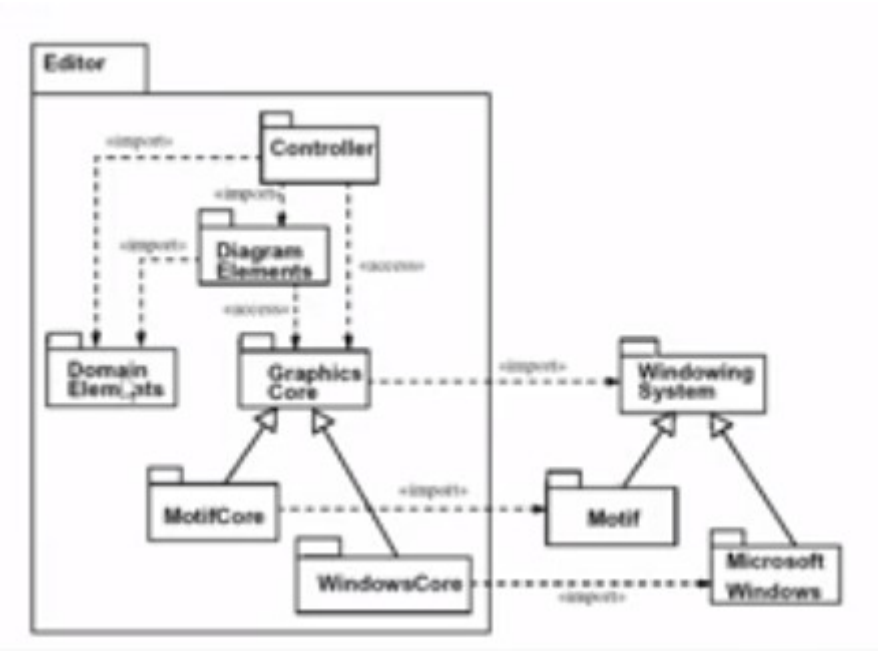
caso no hay cardinalidades porque las relaciones que hay son las que se ven gráficamente, no podemos tener más de las que ya hay especificadas. Y también como dato diferenciador del diagrama de clases, podemos ver que los atributos tienen valor. Por ejemplo, tenemos un profesor con este dni de aquí, cuatro cuatro cuatro cuatro, que da la materia de soldador en el curso de reparaciones de revisión.

A este curso se ha apuntado un cliente con este dni de aquí, el tres tres tres. Este cliente tiene una factura con dos fichas de reparación, la ficha una y la ficha dos y la reparación uno tiene una auditoría de la compañía uno con el material uno y en esta auditoría ha participado el auditor José. Pues esto es un ejemplo concreto de una instancia de nuestro diagrama de clases que hemos visto en sesiones anteriores pero podría ser este ejemplo o podría ser cualquier otro.

El mismo diagrama de clases que hemos definido en el modelo conceptual, nos puede valer para cualquier ejemplo que piensen de instancias concretas.

Diagrama de Paquetes

Un diagrama de paquetes lo que nos permite es organizar los elementos que componen un diagrama de clases, ha de pensar que un modelo de clases de un sistema real puede estar tratando fácilmente entre treinta y sesenta clases y de alguna forma pues hay que agruparlas para que sea sencillo el mantenimiento y entender lo que queremos representar con ese diagrama. Es una forma de empaquetar esas clases a través del diagrama paquetes.

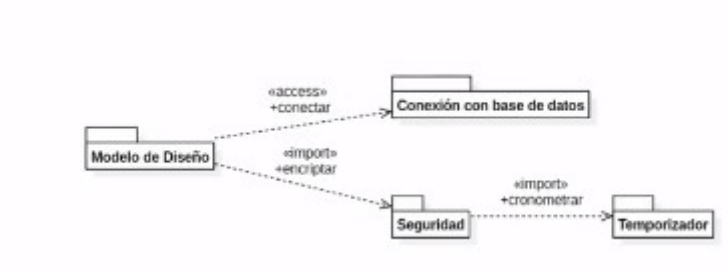


Aquí en el ejemplo vemos un caso práctico. Tenemos un paquete que es editor que a su vez está compuesto de otros paquetes, varios paquetes se pueden anidar tantas veces como se desee.

En este caso tenemos un paquete editor donde dentro tenemos un paquete Controller que importa un paquete que se llama Domain Elements. Recordar como vimos en caso anteriores, la importación lo que hacía era adquirir todas las propiedades del paquete que importamos y además era transitiva, por tanto, este paquete Controller a su vez podía delegar elementos importados a un tercer paquete.

Aquí vemos otro ejemplo de Controller hacia Grafics Core, en este caso es de access, recordar como ya vimos en temas anteriores que el acceso permitía usar las propiedades de Controller pero lo que no permitía era delegar esa importación a terceras clases.

Y como último tipo de ejemplo, tenemos aquí una herencia de Graphics Core y MotiveCore y WindowsCore que lo que permite es redefinir las propiedades que tenemos de la clase padre en las clases hijas añadiendo unas propiedades si fuera necesario.



Este es un ejemplo particular de nuestro curso de la facturación pues imagina que tenemos todas las clases del diagrama que sea de diseño en un paquete que llamado "Modelo de Diseño" y aquí estarían todas las juntas de nuestro diseño y tenemos que acceder a clases, por ejemplo, clases java que nos permitan conectarnos con base de datos con una persistencia. Pues en este caso haríamos uso de la etiqueta access para acceder a las clases que me facilitan o me me permiten la conexión con la base de datos además de tener conexión con bases de datos, mi sistema pues creo que es un sistema seguro y que por ejemplo, pues encriptemos la información con protocolos seguros como pueda ser https ,por ejemplo, en este caso hará falta también utilizar paquetes que estén dentro de la suit de seguridad. En este caso importamos el paquete seguridad que tendrá clases y operaciones particulares para garantizar la seguridad y si luego además queremos también incorporar clases que nos permitan temporizar acciones que ,por ejemplo, cerrar conexiones con base de datos que sobrepase en cinco minutos de inacción o borrar cuentas cada x días, por ejemplo, pues la falta un temporizador y por tanto mientras esta importación de aquí podemos hacer uso de estas clases tanto de seguridad como desde modelo de diseño porque la clase la relación de import es transitiva entonces este paquete se puede usar tanto en seguridad como en el modelo de diseño.