Caso Práctico

*Una empresa de software, para tener éxito, debe producir de forma consistente software de calidad que satisfaga las necesidades de sus clientes, usando los recursos humanos y materiales de forma efectiva y eficiente. Este principio está más que claro para **José**, que desde la fundación de la empresa SI Andalucía se ha preocupado por incorporar técnicas y tecnologías que permitieran a su empresa mantener los niveles deseados de calidad y competitividad, y para ello es muy importante tener en cuenta la importancia de modelar. No es lo mismo construir la caseta del perro que un rascacielos, pero hay empresas de software que quieren comenzar a construir rascacielos, pero enfocan el problema como si estuvieran haciendo la caseta del perro.*

El modelado es una técnica de ingeniería probada, también aplicable al desarrollo del software, ya que un modelo es una representación simplificada de la realidad, que nos permite comprender mejor el sistema que se está desarrollando.

Una de las metodologías de modelado más extendidas actualmente, más comúnmente usada y que ofrece mejores resultados es UML (Unified Modeling Language, Lenguaje de Modelado Unificado), y también en SI Andalucía es usada con frecuencia, siendo mucho más necesaria y útil mientras mayor es el tamaño del proyecto que se está desarrollando.

*José necesita que **Víctor**, como un miembro más de la empresa que es, aprenda esta técnica de modelado para contribuir a mantener los criterios de calidad y eficiencia antes mencionados. Como ha notado que entre Víctor y **Carmen** hay un entendimiento especial, ha decidido que sea ella la que le introduzca los conceptos principales de esta metodología, que deberá usar frecuentemente en el futuro. Comienza con una introducción en la que explica lo que es UML, sus características, y una breve reseña sobre su evolución histórica. Pero lo que de verdad despierta la curiosidad de Víctor, es la explicación sobre lo que se pretende conseguir con UML:*

- *Visualizar*
- *Especificar*
- *Construir*
- *Documentar*

Y Carmen satisface su curiosidad, dándole los detalles.

Introducción al UML

Esta unidad será la primera de tres en las que se abordarán diferentes aspectos sobre UML. En concreto, los puntos que abordaremos en ella serán:

- En qué consiste **UML (Unified Modeling Language, Lenguaje de Modelado Unificado)**.
- Los diferentes diagramas que utiliza esta notación para modelar sistemas.
- Introduciremos el diagrama de clases de este modelo.

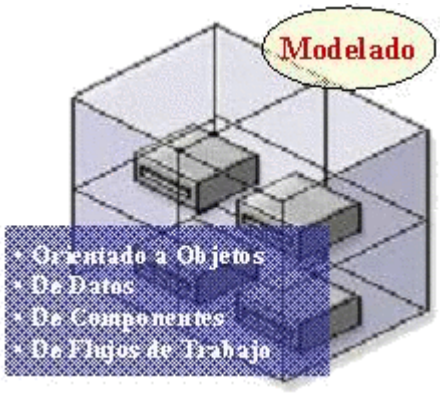
Pero, ¿qué es UML?

UML es un lenguaje para modelar aplicaciones o sistemas.

Para ello introduce una serie de notaciones y diagramas estándar, y describe una semántica esencial de lo que estos diagramas y símbolos significan.

Antes de que apareciese, existían muchas **notaciones y métodos** usados para modelar sistemas, creados por las grandes empresas informáticas. Esto significaba que un mismo problema se podía realizar usando cualquiera de estas notaciones y presentaba el problema de que a los modeladores no les bastaban con conocer una notación solamente. Ahora **los modeladores o analistas sólo tienen que aprender una única notación, UML**.

Imagina las diferentes lenguas que existen en el mundo, una misma cosa se puede nombrar usando cualquiera de ellas y nos estaremos refiriendo a la misma cosa. Bien, ahora imaginemos que elegimos el idioma inglés como idioma oficial para la comunicación entre países de distinta habla. Con esto conseguiríamos que con independencia del país al que pertenezcamos, nos entendiéramos siempre, independientemente de nuestro idioma materno. Con UML pasa exactamente lo mismo, aplicado al mundo informático.



UML no parte de cero sino que es una notación que se basa en otras notaciones ya existentes, provenientes de:

- Modelado Orientado a Objetos,
- Modelado de Datos,
- Modelado de Componentes,
- Modelado de Flujo de Trabajo.

Autoevaluación

Características de UML

Las características principales de este lenguaje se pueden englobar en las siguientes:

- **Aporta una notación estándar (un lenguaje) orientada a objetos,** para el desarrollo de aplicaciones o sistemas informáticos y que es aceptada por todas las grandes empresas del momento.
- **No es un proceso de desarrollo, sino un lenguaje de modelado, es decir no nos indica cómo tenemos que realizar el desarrollo de un sistema.** Este lenguaje puede ser aplicado a cualquier metodología existente.
- **Se basa en especificaciones anteriores como son BOOCH, RUMBAUGH Y COAD-YOURDON**.
- **Permite describir un sistema en diferentes niveles de abstracción,** es decir, plasmar la idea desde diferentes puntos de vista, simplificando la complejidad de la misma y sin que se produzca pérdida de información. Esto permite que todas las personas participantes en el desarrollo de la aplicación (usuarios, desarrolladores, analistas, jefes, etc.) puedan comprender sus características ya que mostrará el sistema desde el punto de vista que les interesa a cada uno de ellos.
- **Divide cada proyecto en un número de diagramas que representan diferentes vistas del proyecto**. Estos diagramas juntos son los que representan la arquitectura del proyecto.
- **UML se puede aplicar tanto a sistemas informáticos como a sistemas que no son informáticos**, como pueden ser los flujos de trabajo en una empresa o diseño de la estructura de una organización.

Tenemos que tener claro que UML no es un proceso o método de desarrollo, es decir, no nos indica los pasos o la forma en la que tenemos que resolver un problema concreto. Para solventar esta carencia posteriormente aparece lo que se denomina RUP (Rational Unified Process), proceso unificado de desarrollo racional. RUP si es un proceso de desarrollo que se aplica a UML.

Autoevaluación

Historia del UML

La situación en la que se encuentra el mundo del desarrollo de software en el momento que aparece UML es la siguiente:

- Existen **diversos métodos y técnicas Orientadas a Objetos** que abarcan muchos aspectos pero que utilizan distintas notaciones, con lo que esto tiene de negativo, como ya hemos visto anteriormente.
- Existen **inconvenientes para el aprendizaje, aplicación, construcción y uso de herramientas**.
- Existe **una lucha constante entre los diferentes enfoques existentes**.

Todo esto desemboca en la **necesidad de establecer una notación estándar que elimine estos problemas existentes**.

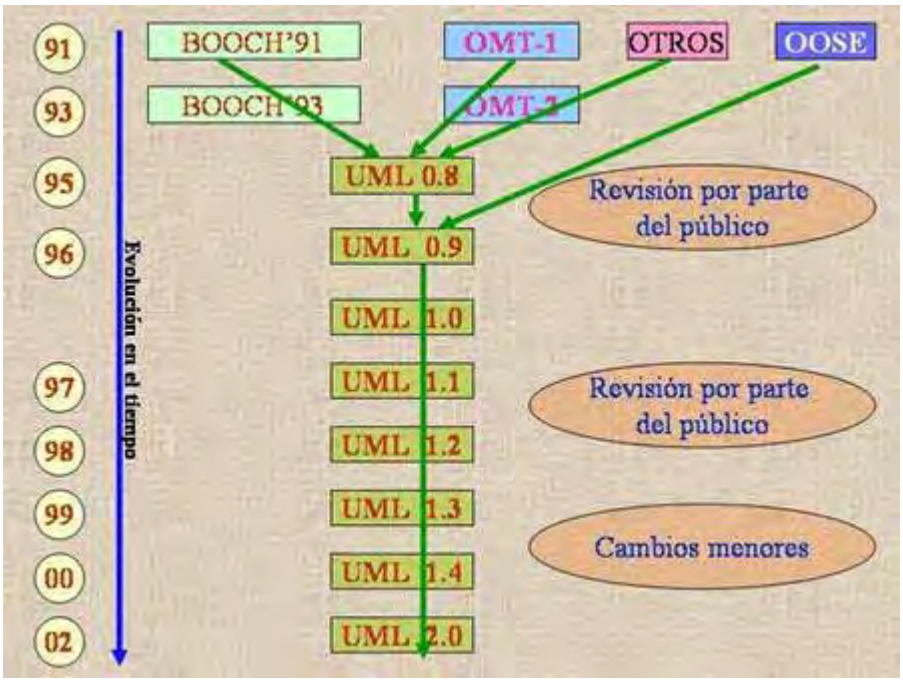
> **El desarrollo de UML comenzó en octubre de 1994 cuando Grady Booch y Jim Rumbaugh de Rational Software Corporation comenzaron a trabajar en la unificación de los lenguajes de modelado Booch y OMT**

Desde este momento fueron reconocidos mundialmente en el desarrollo de metodologías orientadas a objetos. Así, en octubre de 1995, terminaron su trabajo de unificación obteniendo el borrador de la versión 0.8 del denominado **Unified Method**. Hacia fines de este mismo año, Ivar Jacobson (creador de la metodología OOSE - Object Oriented Software Engineer) se unió con Rational Software para obtener finalmente UML 0.9 y 0.91 en junio y octubre de 1996, respectivamente.

> **UML incorpora ideas de otras metodologías desarrolladas por otros autores. Por tanto en el transcurso del estudio de UML, encontraremos cosas que son heredadas de otras metodologías como pueden ser Orientadas a objetos, Modelos de Entidad y Relación, etc.**

Las organizaciones que intervienen en su creación son: Microsoft, Hewlett-Packard, Oracle, Sterling Software MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjectTime, Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam. Todas ellas se asociaron con Rational Software Corporation para dar como resultado UML 1.0 y UML 1.1. Hoy en día llegamos hasta UML 1.4 y UML 2.0. En la siguiente imagen se muestra de forma gráfica y resumida la evolución de UML. Haz clic sobre ella para ver la animación.

Pulsa en la siguiente imagen para ver la evolución de UML



Para saber más.

En este enlace podrás encontrar más información y enlaces sobre la historia de UML

Breve reseña histórica sobre UML [Versión en caché]

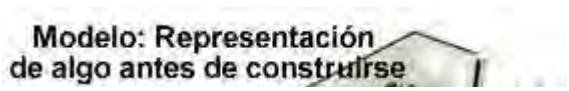
¿Qué se propone conseguir UML?

En definitiva, y a modo de resumen, lo que se pretende a través de **UML** son las siguientes funciones:

- **Visualizar**: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
- **Especificar**: UML permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir**: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- **Documentar**: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Modelos UML

Un modelo es una abstracción (una representación o vista) de "algo", que aplicado a nuestro campo puede ser un sistema informático, una aplicación, etc.



Con la construcción del modelo lo que se pretende es la comprensión de ese "algo" antes de que se construya y para facilitar la comprensión lo que hace el modelo es contemplar y

describir los aspectos importantes del sistema omitiendo los no importantes.

Como veremos más adelante, dependiendo de los aspectos que se tengan en cuenta se irán generando las diferentes vistas que se pueden producir del sistema a entender.

Un **ejemplo** de modelo aplicado a otras profesiones de nuestro entorno puede ser el plano que un arquitecto elabora para construir una casa, una partitura de música, un boceto que utiliza un pintor antes de la creación de un cuadro, etc.

En un modelo podemos interpretar los siguientes elementos de construcción:

- **Elementos**: Los elementos son abstracciones de cosas reales o ficticias (objetos, acciones, etc.)
- **Relaciones**: relacionan los elementos entre sí.
- **Diagramas**: Son colecciones de elementos con sus relaciones.

Autoevaluación

Además, **la construcción de un modelo aporta las siguientes ventajas**:

- Es posible enseñar al cliente una posible **aproximación de lo que será el producto final**.
- Proporcionan una primera aproximación al problema que **permite visualizar cómo quedará el resultado.**
- **Divide un problema complejo en varios problemas de menor complejidad y por tanto más fáciles de resolver**.

Con lo visto en este punto, podemos afirmar que UML persigue obtener un lenguaje que sea capaz de abstraer (representar) cualquier tipo de sistema, independientemente de si es o no informático, mediante el uso de diagramas.

Autoevaluación

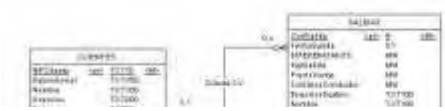
Diagramas UML (Parte I)

Víctor *ha entendido que UML permite mejorar y simplificar el diseño y el análisis de cualquier aplicación, pero todavía se pregunta de qué herramientas se sirve para conseguirlo.*

Carmen *le explica que UML lo consigue mediante una serie de diagramas que representan gráficamente los elementos del modelo y las relaciones que existen entre ellos.*

En total se usan nueve tipos de diagramas, ya que UML permite modelar todo tipo de problemas, pero en la realidad, los más usados quizás sean los diagramas de casos de uso, y los diagramas de clases. Es por eso por lo que ***Carmen*** *se va a centrar por ahora en el estudio de estos últimos. No obstante, también le indica que debe conocer la existencia de los demás (de objetos, de estado, de actividad, de secuencia, de colaboración, de componentes y de despliegue).*

Un diagrama es una representación gráfica de una colección de elementos del modelo y las relaciones que existen entre ellos.



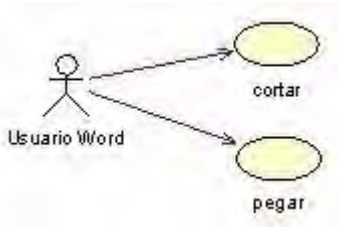
Los distintos puntos de vista de un sistema real se obtienen a partir de estos diagramas. Para ello, cada diagrama resaltará unos u otros detalles del sistema necesarios para

su comprensión.

UML recomienda la utilización de nueve diagramas para representar las distintas vistas de un sistema, aunque para llegar a una solución no tiene por qué ser necesario el empleo de todos ellos. Es más, la mayoría de las veces no obtendremos todos los diagramas, y de hecho en los temas que vamos a dedicar a su estudio no veremos en detalle cada uno de ellos sino sólo los más importantes.

Los **diagramas** de los que se compone UML son los que se mencionan de una forma superficial a continuación:

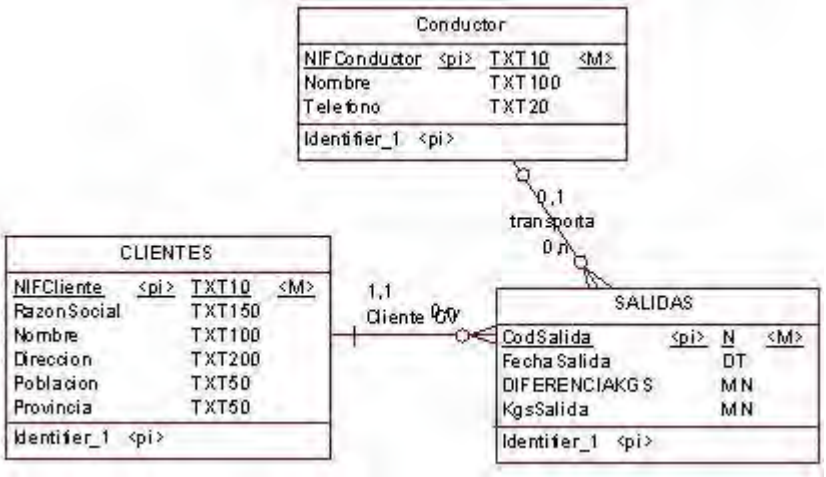
- **Diagrama de casos de Uso**. En ellos se **captura los requisitos del sistema**, **es decir, qué funciones va a realizar el sistema**, **desdeel punto de vista de la interacción con el usuario**. Esto quiere decir, que las funciones o procesos que el sistema va a realizar y que se supone que nosotros como diseñadores tenemos que contemplar, siempre las interpretamos desde el punto de vista del usuario que lo maneja y no como ocurría con los diagramas de flujos de datos o DFDs que los procesos a tener en cuenta ocurren dentro del sistema.



Un ejemplo de casos de uso sería un usuario que maneja un procesador de textos como puede ser Word y en el que pueda realizar (suponemos) dos funciones con la herramienta, pegar y cortar texto. Bien, los casos de uso serían las dos funciones que el usuario va a poder realizar con la herramienta. Esto se representaría gráficamente como se ve en la figura que acompaña al texto de este párrafo.

Autoevaluación

- **Diagramas de clases.** Muestran un conjunto de clases, interfaces y sus relaciones. **Es el diagrama principal de UML** y al que dedicaremos más tiempo. Éste es el diagrama más común a la hora de describir el diseño de los sistemas orientados a objetos.



- **Diagramas de Objetos**. Muestran una serie de objetos (instancias de las clases) y sus relaciones. Es un diagrama de instancias de las clases mostradas en el diagrama de clases. Su representación gráfica sería equivalente a la del diagrama de clases pero más detallado.

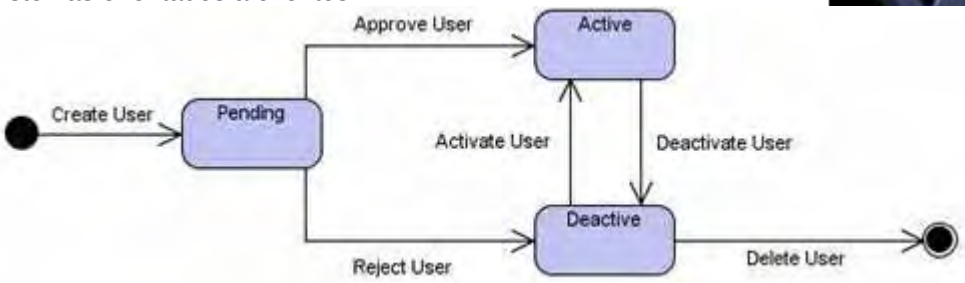
Mediante el modelo de casos de uso obtenemos información acerca de las clases, objetos, atributos y operaciones que debemos establecer en los diagramas de clases y objetos.

Por tanto los diagramas de Casos de Uso serán una fuente de información para el desarrollo de los diagramas de clases y objetos.

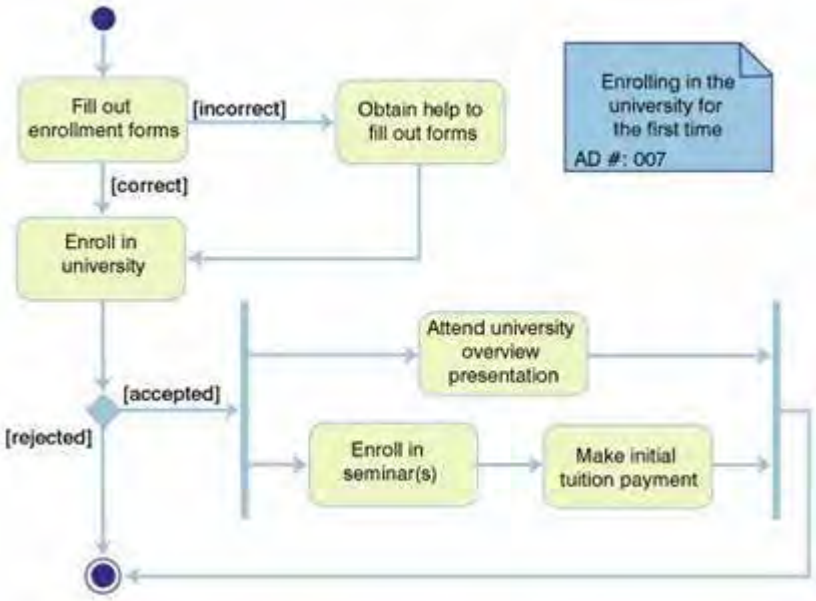
Diagramas UML (Parte II)

Como mencionábamos en el apartado anterior, son **nueve** los diagramas que se recomienda usar con UML. Eso significa que todavía quedan más diagramas por mostrarte, así que vamos a ello.

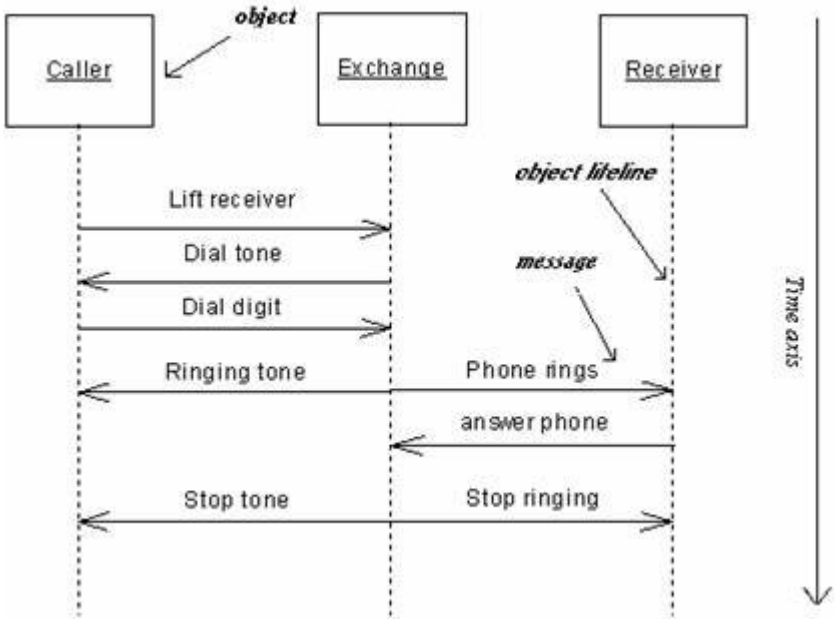
- **Diagramas de estado**. Se utilizan para analizar los cambios de estado de los objetos. Muestran los estados, eventos, transiciones y actividades de los diferentes objetos. Estos diagramas son útiles en sistemas orientados a eventos.



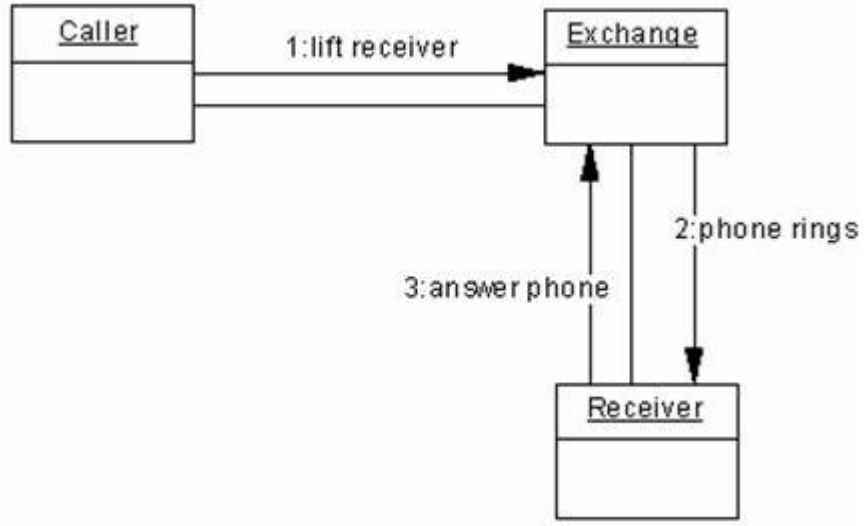
- **Diagramas de Actividad,** son un caso especial del diagrama de estados, simplifica el diagrama de estados modelando el comportamiento mediante flujos de actividades. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.



- **Diagramas de Secuencia,** capturan la interacción entre los objetos y los mensajes que intercambian entre sí atendiendo al orden temporal de los mismos.

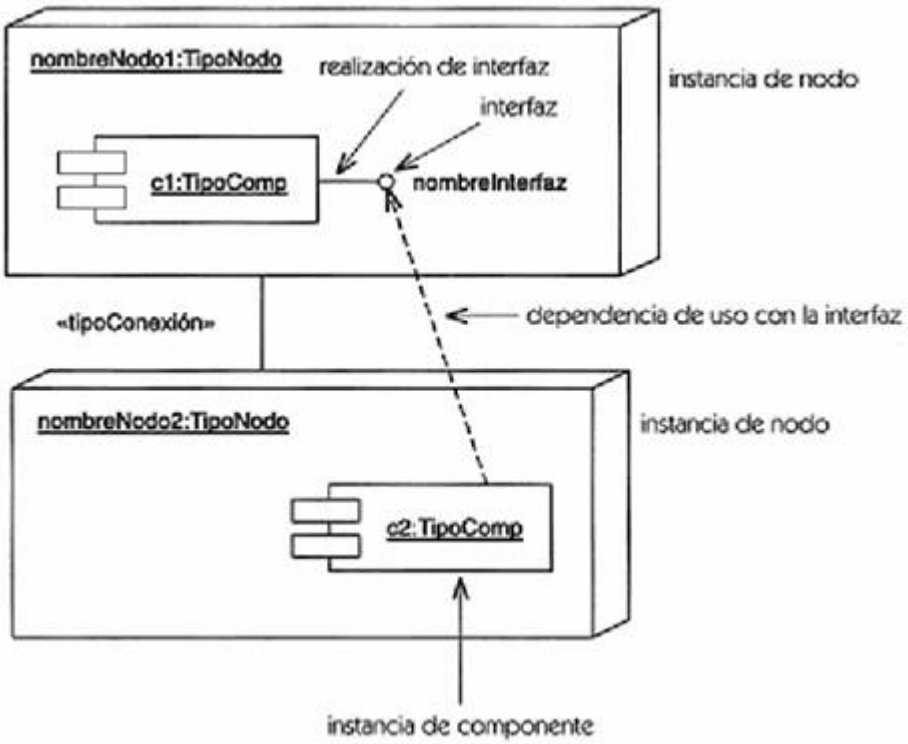


- **Diagramas de colaboración**, igualmente, muestra la interacción entre los objetos resaltando la organización estructural de los objetos en lugar del orden de los mensajes intercambiados.

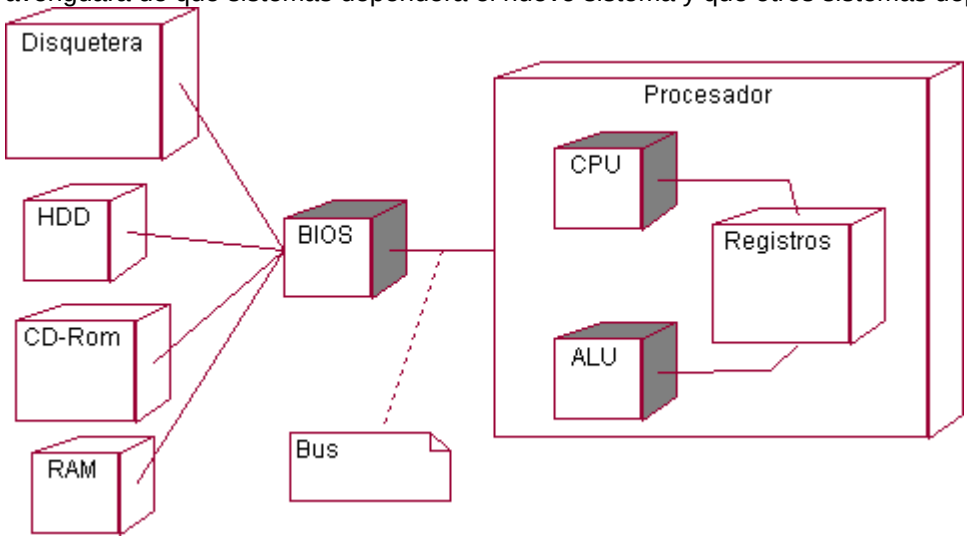


El diagrama de secuencia y el diagrama de colaboración, muestran los mensajes que se envían a los diferentes objetos y las relaciones que pueden tener entre ellos.

- **Diagramas de componentes**, muestra la organización y las dependencias entre un conjunto de componentes. Se usan para agrupar clases en componentes o módulos.



- **Diagramas de Despliegue**, muestra los dispositivos que se encuentran en un sistema y su distribución en el mismo. Se utiliza para identificar Sistemas de Cooperación: Durante el proceso de desarrollo el equipo averiguará de qué sistemas dependerá el nuevo sistema y qué otros sistemas dependerán de él.



Autoevaluación

Para saber más

En este enlace podrás encontrar mas información sobre los diagramas que componen UML así de cómo otros enlaces donde podrás seguir profundizando sobre el tema.

Información sobre diagramas UML **[Versión en caché]**

Clasificación de los Diagramas y Vistas que generan

*A **Víctor** le llama la atención la clasificación que ha hecho **Carmen** de los diagramas usados en UML, sobre todo por el hecho de que según la **agrupación** que hagamos de diagramas construimos distintos modelos o*

vistas del sistema, cada una de ellas más apropiada según el puesto que desempeñe una persona en el proyecto le serán más útiles unas vistas que otras.

Ahora que conocemos superficialmente cuáles son y qué hacen los diagramas de los que se compone UML, vamos a ver cómo se clasifican. Posteriormente entraremos en profundidad en cada uno de ellos.

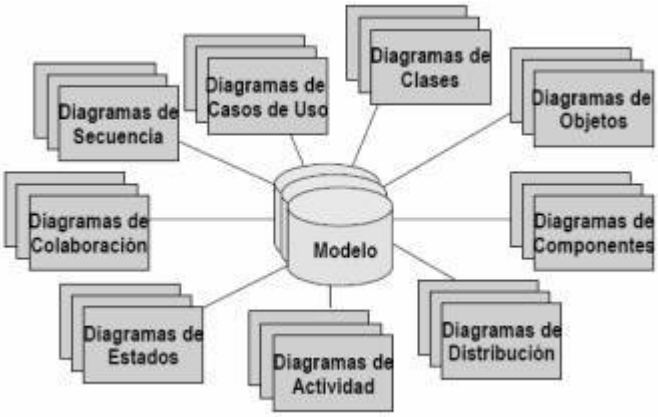
Los diagramas los vamos a clasificar según el estudio que hacen del comportamiento del sistema. De esta forma podemos agrupar los diagramas en:

- **Estáticos,** si representan el sistema de forma estática.
- **Dinámicos,** que por el contrario representan el comportamiento (dinámico) del mismo.

De esta forma tenemos que:

- **Los diagramas que representan al sistema en su forma estática son:**
  - De clases,
  - De objetos,
  - De componentes e
  - De implementación.
- **Los diagramas que representan cómo se comporta el sistema son:**
  - De colaboración,
  - De estado,
  - De actividad y
  - De casos de uso.

Por otro lado según la **agrupación** que realicemos de los diagramas podemos hablar de que se genera una vista u otra del sistema, es decir, obtenemos diferentes modelos del sistema. De esta forma mediante distintos modelos representamos el producto desde las diferentes perspectivas de interés.



Dependiendo del **puesto** que desempeñe una persona en el proyecto le serán más útiles unos modelos que otros. Así, la persona encargada de determinar cuáles son las funciones que el sistema desempeña (analista del sistema), utilizará los diagramas de casos de uso, mientras que la persona encargada de implementar dichas funciones (programador del sistema), utilizará directamente el diagrama de clases.

Según la agrupación que hagamos de diagramas encontramos las siguientes vistas del sistema:

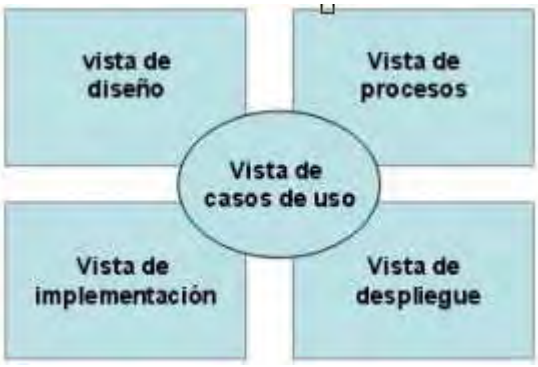
- **Vista de casos de uso**, compuesta por los siguientes diagramas:
  - de casos de uso,
  - de colaboración,
  - de estados y
  - de actividades.
- **Vista de diseño**, compuesta por los diagramas:
  - de clases,
  - de objetos,
  - de colaboración,
  - de estados y
  - de actividades.
- **Vista de procesos**, formada por los diagramas que componen la vista de diseño pero ahora indicando con detalle en la vista de clases y objetos cuáles son los procesos.
- **Vista de implementación**, compuesta por los diagramas:

- de componentes,
- de colaboración,
- de estados y
- de actividades.

- **Vista de despliegue**, compuesta por los diagramas:
  - de despliegue,
  - de interacción,
  - de estados y
  - de actividades.

Autoevaluación

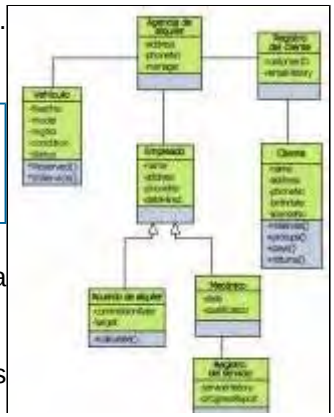
Indicar que la forma de agrupar estos diagramas coincide con la forma como lo hace la herramienta **CASE Rational Rose**.



Los diagramas más importantes y los más usados son los de casos de uso, clases y secuencia.

Diagrama de clases

María *le ha comentado a* ***Carmen*** *que una de las cosas que más va a necesitar* ***Víctor*** *es hacer los diagramas de clases UML. Hay numerosas herramientas informáticas que permiten construirlos, pero antes de hacer ninguno es necesario conocer los elementos que componen esos* ***diagramas****, y la forma de construirlos. Mientras más le explica* ***Carmen*** *estos diagramas, más familiares le resultan a* ***Víctor****, y es que recuerda haber visto que* ***José*** *los manejaba cuando hacía el análisis de alguna de las aplicaciones que posteriormente* ***Víctor*** *se ha encargado de codificar. Por eso sabe que son los planos sobre los cuales* ***José*** *le iba indicando cómo construir el esqueleto de la aplicación que querían construir. Estos diagramas representan de una forma muy gráfica las distintas clases que forma parte del problema, los atributos que tienen cada una, y las relaciones que existen entre ellas, de forma que eran muy útiles a la hora de definir las clases y subclases de las que constaba un programa Java, por ejemplo. Entonces él no sabía entenderlos, pero ahora ve que son una excelente herramienta de documentación, al mismo tiempo que de representación del sistema y de planos a seguir en la construcción de la solución informática.*



El diagrama de clases **es el más importante dentro de los que forman UML**. Además, éste va a ser el diagrama al que más esfuerzo vamos a dedicar.

Para el desarrollo de este diagrama debemos conocer a fondo conceptos sobre la orientación a objetos ya que se emplean para la construcción del mismo.

Durante el desarrollo de este punto haremos referencia a los que vamos a utilizar para su construcción.

El **diagrama de clases** empieza a gestarse en el mismo momento en el que nos

entrevistamos con la persona responsable que nos encarga el desarrollo de nuestro producto informático. Si suponemos que se va a informatizar la actividad de un negocio, por ejemplo un videoclub, la persona encargada de entrevistarse con el cliente, será el analista. Éste debe tomar apuntes siguiendo una serie de reglas que le ayudarán a determinar las clases, atributos, métodos, etc. En las que se descompondrá el sistema.

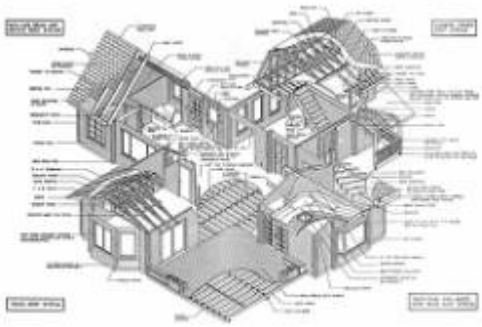
El analista para ir identificando todos estos elementos, clases, atributos, métodos, etc. **desarrollará el diagrama de casos de uso.** Por esto, podemos decir que a partir del diagrama de casos de uso iremos determinado el diagrama de clases.

Los elementos que encontraremos en un diagrama de clases son:

- **Clases**, compuestas por atributos, métodos y su visibilidad.
- Las **relaciones** que existen entre ellos, donde se indica su visibilidad, asociación, ensamblado y uso.

Clases

Una clase es una plantilla que define la forma de un tipo de objetos.

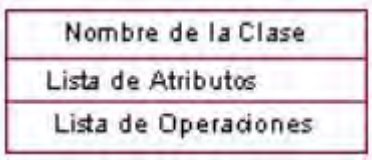


Para entender mejor esto, imagina un plano de una casa. Éste define cómo va a ser la casa, pero la casa todavía no existe en la realidad. Bien, cuando la casa se construye a partir del plano obtenemos un objeto de esa clase.

En esta plantilla se especifican **los atributos y los métodos** con los que van a contar los objetos que se construyan a partir de dicha plantilla, respectivamente, determinan la estructura y su comportamiento. Por tanto, podemos afirmar que **una clase describe objetos que van a tener la misma estructura y el mismo comportamiento**. Además, **el hecho de crear un objeto a partir de una clase**, que equivaldría a la acción de crear la casa partir de su plano, **recibe el nombre de instanciar un objeto**. Así, podemos afirmar que **los objetos son las instancias de las clases**.

Una cosa que debemos tener en cuenta para entender la verdadera naturaleza de los objetos es que **los objetos no existen hasta el momento de su creación, es decir, en un programa los objetos no existen hasta que el programa empieza a ejecutarse.**

En UML, las clases se representan gráficamente mediante un rectángulo en el que encontramos tres divisiones:



- En la primera de ellas encontramos el **nombre de la clase**,
- en la segunda división encontramos los **atributos** y
- por ultimo, están los **métodos de la clase**.

Visualiza un ejemplo de cómo se representarían los datos de una clase

Autoevaluación

Atributos

Un atributo representa alguna propiedad o característica de la clase.

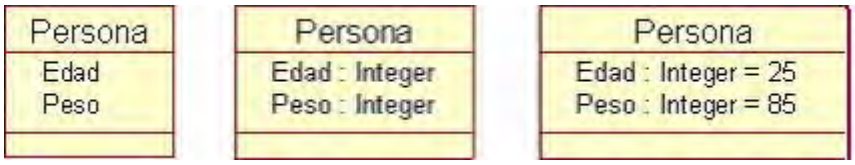
Éste se va a encontrar en todas las instancias que hagamos de la clase. Los atributos **pueden representarse mostrando su nombre,**

mostrando su nombre y su tipo, e incluso su valor por defecto.
Aunque lo normal siempre que diseñemos una clase es que junto al atributo vaya asociado su tipo.

> **El tipo del atributo nos indicará la naturaleza de los datos que va a poder tomar.**

De esta forma, si indicamos que el atributo va a ser de tipo entero indicaremos que podrá tomar los varores: ...-4,-3,-1,0, 1,2,3,4... etc.

Otro factor a tener en cuenta a la hora de indicar el **tipo de atributo**, es que cuando se implemente sobre una herramienta, ésta guardará una u otra cantidad de memoria del sistema para su representación. Por ejemplo se necesitará la misma cantidad de memoria para guardar un valor entero que un valor real.



Los atributos o características de una Clase pueden clasificarse en tres grupos haciendo referencia a su nivel de encapsulación, es decir, el grado de comunicación y visibilidad de ellos con el entorno.

- **Públicos (public)**: un atributo público es aquél que es visible tanto dentro como fuera de la clase. Esto quiere decir que el atributo será accesible tanto desde dentro como desde fuera de la clase.
- **Privados (private):** Este tipo de atributo sólo será visible desde dentro de la clase, lo que implica que sólo los métodos de la clase a la que pertenecen podrán utilizarlos.
- **Protegidos (protected):** Indica que el atributo no será visible desde fuera de la clase, pero sí desde dentro de la clase a través de sus métodos y además podrá ser accedido por las subclases que hereden (subclases) de la clase que lo contiene.

Autoevaluación

En la figura que acompaña a este párrafo podemos observar un **ejemplo** de la clase Persona. En ésta encontramos los siguientes atributos:

- Nombre (público)
- Edad (privado)
- Teléfono (privado)
- Peso (privado)
- Color del pelo (protegido)

> **Los símbolos que indican su visibilidad, es decir, si son privados, protegidos o públicos dependerán de la herramienta que utilicemos para crear el modelo.**

Los que se encuentran en la figura anterior corresponden a la herramienta Rational Rose y son:

- El símbolo  indica que el atributo es **público**, también representado en otras herramientas mediante el símbolo suma (+).
- El símbolo  indica que el atributo es **privado**, también representado en otras herramientas mediante el símbolo resta (-).
- El símbolo  indica que el atributo es **protegido**, también representado en otras herramientas mediante el símbolo almohadilla (#).

Los niveles de encapsulación son heredados del lenguaje de programación C++ donde se utilizan los símbolos +.- y # para indicar su visibilidad, de aquí que algunas herramientas para construir el diagrama usen estos símbolos.

Mira cómo indicar los atributos de forma correcta

El tipo de símbolos que vamos a utilizar para representar los niveles de encapsulación gráficamente, no tiene mucha importancia en este momento ya que va a depender de la herramienta que utilicemos en el futuro para realizar el modelo y por tanto como sucede con los compiladores o intérpretes que utilicemos para implementar el código de un programa, deberemos estudiar las peculiaridades de cada una de ellas.

Autoevaluación

Retomando el tema de los **atributos**, tenemos que tener claro que **cuando establecemos qué atributos van a pertenecer a una clase, lo que estamos indicando es la estructura que esa clase va a tener y la de sus instancias**.

Algunas **reglas que los atributos deben cumplir dentro de una clase** son las siguientes:

- **El nombre de un atributo debe ser único dentro de la clase**, aunque puede aparecer un mismo nombre de atributo en diferentes clases.
- **El nombre de un atributo no debe tener espacios entre sí.** Por ejemplo "**nombreCliente"** sería un nombre correcto para el atributo que hace referencia al nombre del cliente, mientras que "**nombre Cliente"** no lo es. Lo que sucede es que las herramientas UML permiten escribir "**nombre cliente"** e internamente ellas lo convierten a "**nombreCliente"**, es decir, permiten mostrar una cosa pero manejan otra de forma automática.
- **Los atributos no tienen niguna identidad, al contrario que los objetos**, es decir, un mismo atributo puede tener un mismo valor para dos objetos diferentes y además poseer el mismo valor, como por ejemplo la edad (atributo) que dos personas (objeto) tienen puede ser la misma pero sin embargo son dos personas diferentes.
- **Los atributos toman valores dentro de un determiando rango.** A esto se conoce como dominio. Así por ejemplo supongamos que la edad de una persona puede ir desde 0 a 120 años. Pues bien, al conjunto de todos estos valores posibles que puede tomar el atributo (0-120) se le conoce con el nombre de domino.
- **Los atributos de una clase no deberían ser manipulados directamente por el resto de objetos.** Normalmente los atributos de una clase serán manipulados por los métodos de dicha clase.

Autoevaluación

Identificadores

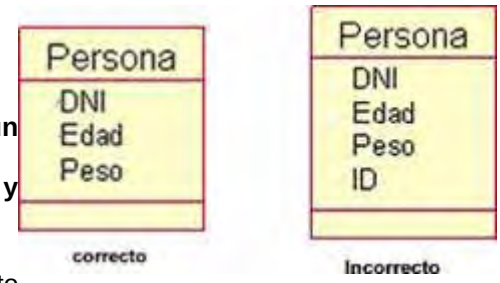
Los identificadores son un tipo de atributo cuya función no va a ser la de reflejar características de la clase en la que se encuentran sino la de identificar unívocamente a la clase una vez que se instancia, es decir, va a identificar a los objetos, diferenciando unos de otros.

Esto quiere decir que este tipo de atributo no se va a indicar cuando se define la clase y sí cuando se crea el objeto.

Cuando construimos las clases del sistema, los identificadores no deben ser incluidos como atributos de la clase, ahora bien, hemos mencionado que un identificador va a ser un tipo de atributo que distinguirá a los objetos unos de otros, pero ¿que pasará con los atributos que son únicos para un tipo de clase, como por ejemplo el DNI de una persona o el Número de Matrícula de un coche? Pues que nos encontramos ante un atributo del mundo real, que aporta información de la clase y que además nos va a servir como

identificador de los objetos que se creen a partir de ella.

Por tanto podemos afirmar lo siguiente:



- **Una clase estará determinada por un estado y un comportamiento**.
- **Un Objeto estará determinado por un identificador, un estado y un comportamiento**.

El estado de una clase viene dado por el valor que en un momento determinado tienen sus atributos, mientras que su comportamiento lo determinan los métodos de la clase o más exactamente las funciones que estos desempeñan dentro de la clase.

Las **características de un identificador** son las siguientes:

- **Es único y global para cada objeto dentro del sistema**.
- **Es determinado en el momento de la creación del objeto**.
- **Es independiente de las propiedades del objeto**.
- **No cambia durante la vida del objeto ni tampoco se reutiliza cuando éste deje de existir**.

Autoevaluación

Aunque para identificar a un objeto se utilizan los identificadores, es necesario poder acceder o referenciar a los objetos mediante sus atributos. ¿Por qué? Pues porque será más fácil para las personas recordar un atributo más ligado a la realidad como el número de hijos de una persona, número de puertas de un coche, etc. que un identificador que lo único que hace es eso, identificar a los objetos, pero no aporta ninguna información a priori sobre ellos.

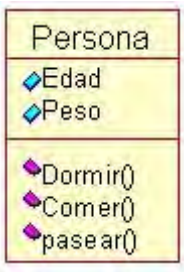
Métodos

> **Son el conjunto de funcionalidades que describen la naturaleza y el comportamiento que tendrán los objetos de la clase; es decir, las operaciones que van a poder realizar los objetos de esa clase.**

Un método bien construido debería ejecutar una única tarea. Además, los métodos deberían ser el único modo de acceder a los atributos de los objetos.

Las operaciones que se definen mediante los métodos de una clase pueden ser ejecutadas por el objeto o sobre el objeto. Para entender un poco mejor qué podrían ser métodos, vamos a ver unos cuantos **ejemplos**.

- Para la clase Puerta podemos tener los siguientes: abrir, cerrar, pintar, etc. Cuando se instancia esta clase y se crea el objeto Puerta, éste puede ejecutar su método abrir y la puerta se abrirá.
- Para la clase Persona: comer, dormir, andar, etc. De la misma forma que el ejemplo anterior un objeto Persona comerá cuando ejecute su método comer.

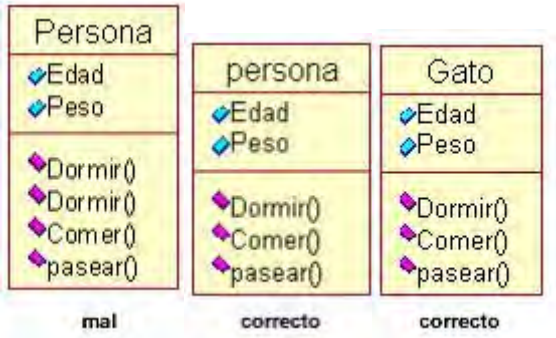


Los ejemplos vistos anteriormente nos facilitan la comprensión de que los métodos son las funcionalidades de los objetos.

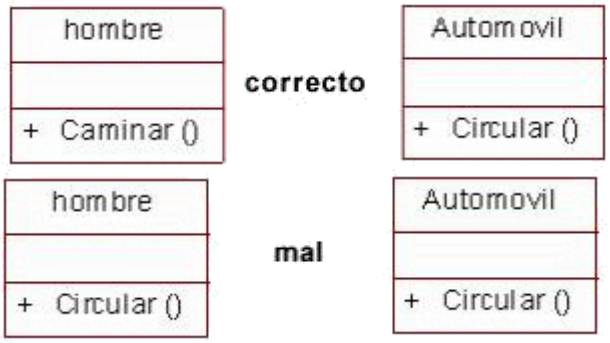
Autoevaluación

A la hora de establecer los métodos de una clase debemos seguir una serie de **reglas**:

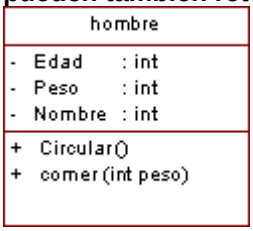
- **Los métodos deben ser únicos dentro de una misma clase**, aunque no necesariamente para diferentes clases. Así por ejemplo, la clase Persona y Gato pueden tener un método que sea dormir pero no pueden existir dos métodos dormir dentro de una misma clase.



- **No se debe utilizar el mismo nombre para métodos que tengan un significado totalmente diferente**. Por ejemplo, supongamos un método que se llame desplazarse para la clase Persona y para la clase Coche. Este método no debe llamarse igual en ambas clases ya que la manera en que los dos objetos se desplazarán no es la misma y por tanto el método no debe llamarse de la misma forma, porque podría generar confusión. Para corregir esto, la manera adecuada sería llamarlos por ejemplo **Caminar** para la clase Persona y **Circular** para la clase Coche.



- **Los métodos pueden tener argumentos, es decir, una lista de parámetros, cada uno con un tipo y pueden también retornar resultados, cada uno con un tipo**.



Al igual que ocurría con los atributos, **encontramos también tres formas de encapsular los métodos de una clase**, es decir, de determinar la visibilidad de los métodos en una clase.

- **público (public):** Representados gráficamente mediante el símbolo o símbolo **+** , Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible sin restricciones.
- **Privado (private):** Representado gráficamente mediante el símbolo o símbolo **-**, indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la misma clase lo pueden usar).
- **Protegido (protected):** Representado gráficamente mediante el símbolo o símbolo #, indica que el método no será accesible desde fuera de la

clase, pero sí podrá ser accedido por métodos de la clase además de métodos de las subclases que se deriven de ella.

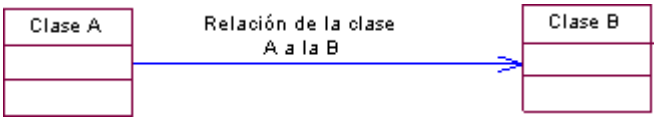
Relaciones entre clases

Vamos a ver ahora otro elemento que integran las clases: **Las relaciones**.

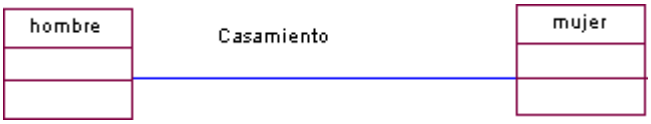
En los puntos sucesivos vamos a ver los **tipos de relaciones** que se pueden dar entre clases.

En las relaciones podemos encontrar con que dicha relación será **unidireccional** (en un solo sentido) o bien **bidireccional** (en ambos sentidos). Lo normal es que cuando creemos relaciones entre clases éstas sean bidireccionales.

La siguiente figura muestra una relación unidireccional. En este caso al ser de un solo sentido la flecha partirá de la clase origen para llegar hasta la clase destino. En este caso la relación parte de la clase origen A que se relaciona con la clase destino B.



Cuando las relaciones son bidireccionales, es decir, en ambos sentidos se omite la flecha de dirección.



Para saber más.

En este enlace podrás encontrar mas información sobre los relaciones en los diagramas de clases de UML.

Información sobre relaciones en diagramas de clases de UML [Versión en caché]

Autoevaluación

Relación de Asociación

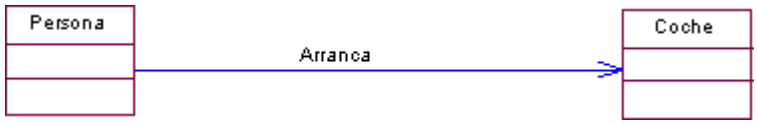
Cuando se establece una relación entre objetos, en términos de programación orientada a objetos se dice que se produce una asociación.

Por ejemplo, las clases Persona y Coche se relacionan entre sí, ya que la persona arranca el coche, lo conduce, etc. Decimos entonces que la persona y el coche se asocian.

Las asociaciones indican cómo se conectan las clases entre sí.

En el caso anterior la conexión que se produce entre las clases es unidireccional ya que la persona arranca el coche y la clase Coche en este caso no realiza nada sobre la clase Persona. Como dijimos anteriormente, cuando se establecen

relaciones unidireccionales, Éstas son indicadas mediante una flecha de dirección que indica en que sentido se establece la relación.

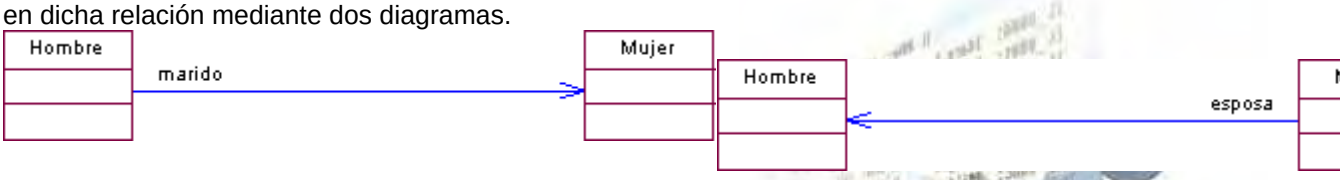


Pero la mayoría de las veces, las relaciones que establezcamos van a ser bidireccionales. En este caso las dos clases intervienen, jugando papeles distintos en la relación.

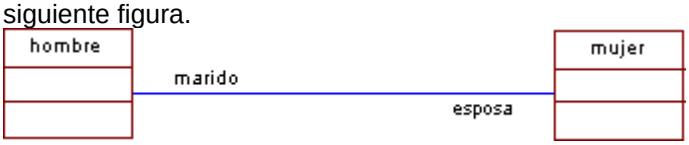
Un ejemplo puede ser cuando dos personas se casan (casamiento), en este caso la clase hombre se relaciona con la clase mujer, siendo su marido y la clase mujer se relaciona con la clase hombre siendo su mujer.

Estos casos se pueden representar de dos formas:

- De manera **unidireccional**, donde tendremos que representar la relación y el papel o Rol que juega cada clase en dicha relación mediante dos diagramas.



- De forma **bidireccional**, indicando también el rol, pero sobre el mismo diagrama, como se puede apreciar en la siguiente figura.



Cuando hablamos de asociaciones tenemos que tener en cuenta que este término indica bidireccionalidad en las relaciones y por tanto casi siempre representaremos de esta última forma una relación de asociación, indicando el Rol o papel que juegan cada una de las clases en la relación.

En este caso la relación se representa sin flecha de dirección.

En este ejemplo podemos observar una relación entre la clase Profesor y Estudiante "Asignatura de análisis" en la que la clase profesor está asociada con Alumnos en la manera en que "Es el profesor de la asignatura", es decir, éste sería el Rol que la clase Profesor juega en la relación, y la clase Estudiante está asociada con la clase Profesor en la manera en que "es alumno de la asignatura" .

El papel o rol que juega cada clase en la relación siempre va a indicarse gráficamente en los extremos de la relación.

Autoevaluación

Multiplicidad de una asociación

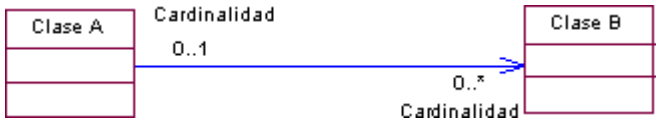
Cuando establecemos una relación entre dos clases, **otro dato importante a indicar es la cardinalidad o multiplicidad de la relación, es decir indicar el grado con el que una clase se asocia con otra** o lo que es lo mismo, cuántas conexiones van a existir entre un objeto de la clase A con un objeto de la clase B y viceversa.

> **La cardinalidad entre dos clases, al igual que el Rol que la clase juega en la relación, la vamos a representar gráficamente en los extremos de la relación.**

Los diferentes **tipos de cardinalidad que podemos indicar en una relación son**:

- **Uno a uno (1,1)**: un objeto de tipo A se relaciona uno a uno con objetos de tipo B.
- **Uno a muchos (1,*)**: indica que un objeto A puede relacionarse con uno o varios objetos de tipo B.
- **Muchos a muchos (*):** indica un objeto de tipo A puede relacionarse con varios objetos de tipo B.

> **Cuando el grado de multiplicidad empieza por 1 obligamos a la existencia de conexión, es decir, obligamos a que por lo menos exista una.**



- **Opcional (0,1):** Indica que el objeto de tipo A puede estar o no relacionado con un objeto de tipo B.
- **Opcional múltiple (0,*):** Otro tipo de cardinalidad opcional pero en este caso se da que un objeto de tipo A puede estar ninguna o muchas veces relacionado con objetos de tipo B.
- **Número fijo (n):** En este caso indicamos un número de veces concreto que se relaciona un objeto de tipo A con objetos de tipo B.

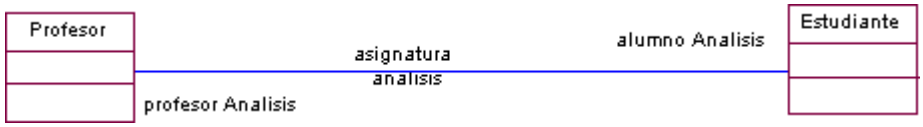
Visualiza un ejemplo de multiplicidad

Autoevaluación

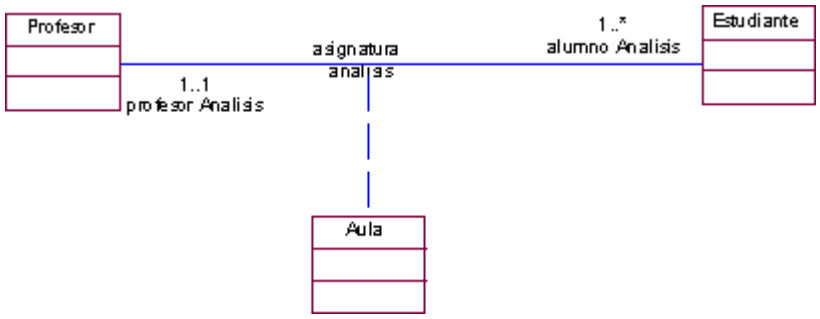
Grado de una Asociación

El grado de una asociación se determina por el número de clases conectadas por la misma asociación. Las asociaciones pueden ser binarias, ternarias, o de mayor grado. Así encontramos que:

- **Asociaciones Binarias**: Las asociaciones binarias son aquéllas que sólo relacionan dos clases.

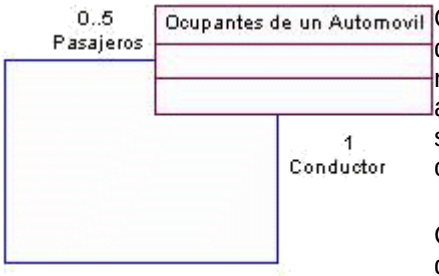


- **Asociaciones Ternarias**: Aquéllas que relacionan tres clases.



- **Asociaciones n-Anarias**: Donde se relacionan n clases.

Asociaciones Reflexivas



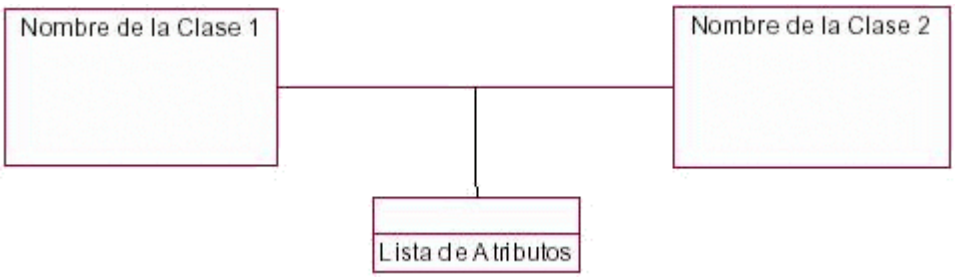
Otro tipo de Relación de asociación son las que relacionan distintos objetos de una misma clase. A este tipo se le conoce con el nombre de Asociaciones reflexivas. Supongamos el ejemplo de los ocupantes de un automóvil. Bien, aquí podemos establecer relaciones entre objetos de la misma clase de la siguiente forma: un ocupante puede ser conductor del automóvil y habrá otros ocupantes que pueden ser pasajeros del automóvil.

Como se observa tanto unos como otros son objetos de la misma clase, es decir todos son ocupantes, lo único que cambia sería el papel que juegan dentro de la relación.

Atributos de liga o Asociación

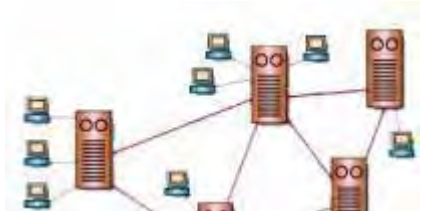
Este tipo de atributos son los que no pertenecen a ninguna de las clases que se asocian sino que son atributos de la asociación que hay entre ellas, es decir son propiedad de la asociación.

La notación es similar a la usada para los atributos de clases, excepto que se añade a la asociación, y no se incorpora un nombre de clase, como se muestra en la siguiente figura:



Autoevaluación

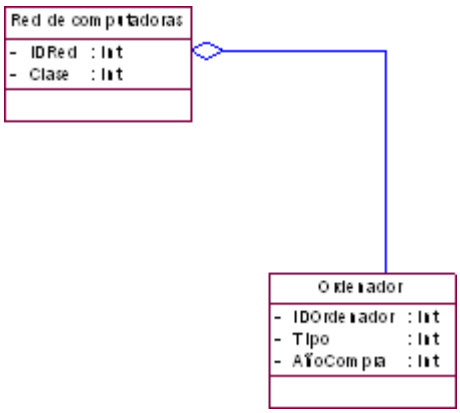
Relación de Agregación



En este tipo de relaciones el objeto base (el que agrega a otros) utiliza a otros objetos para su funcionamiento. Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye ya que estos pueden existir sin que el objeto base exista.

La relación de Agregación se indica mediante un rombo sin rellenar, donde el rombo se sitúa en el extremo de la clase que contiene a otra.

Un **ejemplo de agregación** puede ser el siguiente: Una Red de Computadoras se puede considerar un ensamblado, donde las Computadoras son sus componentes.



Este **ejemplo** presenta las siguientes características:

- Las mismas computadoras que forman parte de la red pueden ser a su vez partes de otras redes por lo que se concluye que pueden aparecer en otros ensamblados.
- Las Computadoras pueden existir independientemente de la existencia de la Red de Computadoras.

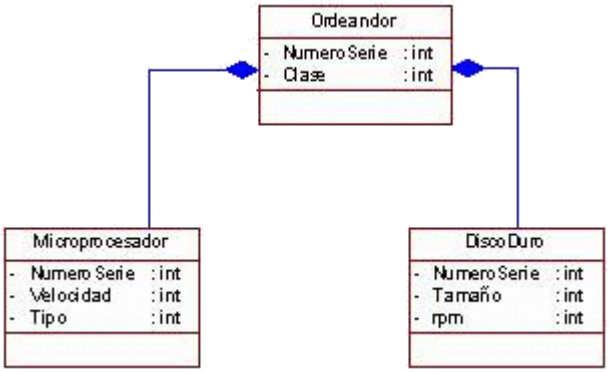
Autoevaluación

Relación de Composición

En este tipo de relaciones se da a entender que el objeto base, se construye a partir de otros objetos incluidos. Es un tipo de relación estática, en donde el tiempo de vida de los objetos incluidos está condicionado por el tiempo de vida del objeto que los incluye, ya que éstos forman parte de él.

La relación de composición se indica mediante un rombo relleno, donde el rombo se sitúa en el extremo de la clase que contiene a otra.

Un ejemplo de composición sería: Un Ordenador como ensamblado, donde el microprocesador y el disco duro son sus componentes. Las características que encontramos en este ejemplo son:



- El microprocesador y el disco duro son partes del Ordenador, y a diferencia de la agregación, no pueden ser compartidos entre distintos Ordenadores a la vez.
- No tiene mucho sentido que el microprocesador y el disco duro existan de manera independiente al ordenador, por lo cual la composición refleja de manera importante, el concepto de propiedad.

En el caso de agregación, las partes del ensamblado pueden aparecer en múltiples ensamblados. Esto no sucede en el caso de composición, donde las partes del ensamblado no pueden ser compartidas entre ensamblados.

La decisión de usar composiciones, agregaciones o asociaciones de una manera imprecisa no causa grandes problemas, aunque siempre que podamos o tengamos claro qué usar deberemos ser precisos.

Autoevaluación

Herencia: Generalización y Especialización

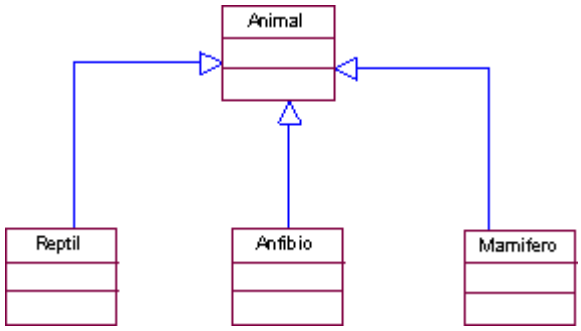
Las clases con atributos y operaciones comunes se pueden organizar de forma jerárquica, mediante la herencia.

La herencia es una abstracción importante para compartir similitudes entre clases, donde todos los atributos y operaciones comunes a varias clases se pueden compartir por medio de la superclase, también llamada clase base, que es una clase más general.

Las clases más refinadas se conocen como subclases, también llamadas clases derivadas.

Los ancestros de una clase son las superclases de una clase en cualquier nivel superior de la jerarquía, y los descendientes de una clase son las subclases de una clase en cualquier nivel inferior de la jerarquía.

Supongamos las clases **Reptil, Anfibio y Mamífero**. Todas estas son subclases de la superclase Animal, de la que derivan. La superclase tendrá una serie de atributos y operaciones comunes a todas las subclases (Reptil, Anfibio y Mamífero), mientras que en las subclases encontraremos atributos y operaciones específicos de cada tipo de animal. Por tanto se puede establecer que las subclases hereden de la superclase, ya que los atributos y operaciones de ésta son comunes a todas ellas.



Con la herencia se conseguirá a nivel conceptual una estructuración de clases mientras que a nivel de implementación evitará la duplicación de código.

Una relación de Herencia la podemos dividir en dos relaciones, una de Generalización y otra de Especialización.

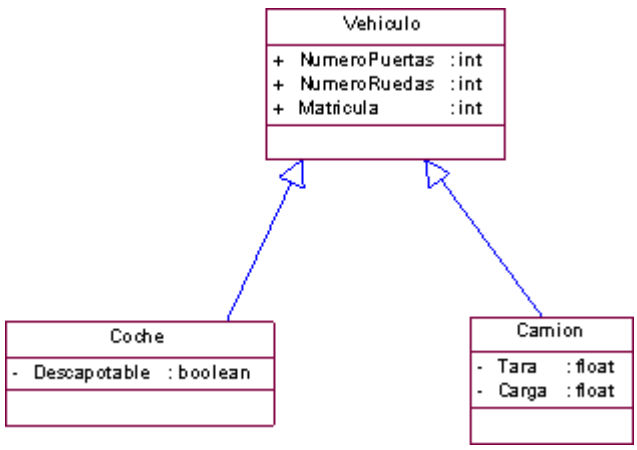
- **La generalización define una relación entre una clase más generalizada, y una o más versiones refinadas de ella.** Éste sería el caso de la clase Animal, que sería una Generalización de las subclases Reptil, Anfibio y Mamífero.
- **La Especialización define una relación entre una clase más general y una o más versiones especializadas de ella**. Éste es el caso de las clases Reptil, Anfibio y Mamífero. Todas ellas son especializaciones de la superclase Animal.

De todo esto se deduce:

- **La superclase generaliza a sus subclases.**
- **Las subclases especializan a la superclase.**
- **El proceso de Especialización es inverso al de Generalización.**
- **Una instancia de una subclase, es también una instancia de su superclase.** Esto quiere decir que cuando creamos un objeto de tipo Anfibio, éste contiene la información definida en él y también la heredada de la superclase Animal. Por eso decimos que es una instancia de ambas clases.

Autoevaluación

Por medio de la herencia las subclases heredan las operaciones y atributos especificados en la superclase, es decir, las subclases además de sus operaciones y atributos contarán con los de la superclase. **Pero para que los atributos y operaciones de la superclase** puedan ser heredados deben ser visibles, esto es públicos o privados.



En la figura anterior, Coche y Camión heredan los atributos de la superclase Automóvil (fíjate que estos atributos están declarados de forma pública para que puedan ser

heredados). Además cada subclase tiene definidos atributos particulares.

Cuando utilizamos la herencia tenemos que tener en cuenta que **los nombres de atributos y operaciones deben ser únicos en la jerarquía de herencia.**

Relaciones de Dependencia

Representa un tipo de relación muy particular, en la que una clase es instanciada y en la que su instanciación es dependiente de otro objeto/clase.

Es una relación de uso, es decir **una clase usa a otra, que la necesita para su cometido. Se representa con una flecha discontinua que va desde la clase utilizadora a la clase utilizada**.

Con la dependencia mostramos que un cambio en la clase utilizada puede afectar al funcionamiento de la clase utilizadora, pero no al contrario.

El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación gráfica que instancia una ventana (la creación del Objeto Ventana está condicionado a la instanciación proveniente desde el objeto Aplicación):

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

Autoevaluación

Finalmente, te recomendamos que leas el documento que te proporcionamos en el siguiente enlace. Es un buen resumen de muchos de los conceptos presentados aquí sobre Diagramas de clases UML

Descarga el documento