PROGRAMACIÓN ORIENTADA A OBJETOS

Master de Computación

II MODELOS y HERRAMIENTAS UML

II.3 UML: Modelado estructural

Concepto de objeto y de clase

- **Objeto**: Concepto, abstracción o cosa con límites bien definidos y con significado para el problema que se está manejando.
  - Todos los objetos presentan una identidad, que los hace distinguibles, un estado y un comportamiento.
  - El término *identidad* significa que los objetos se distinguen por su existencia inherente y no por propiedades descriptivas que puedan tener.

- **Clase**: Descripción abstracta de un grupo de objetos con propiedades similares (atributos), comportamiento común (operaciones), relaciones comunes con otros objetos y semántica común.

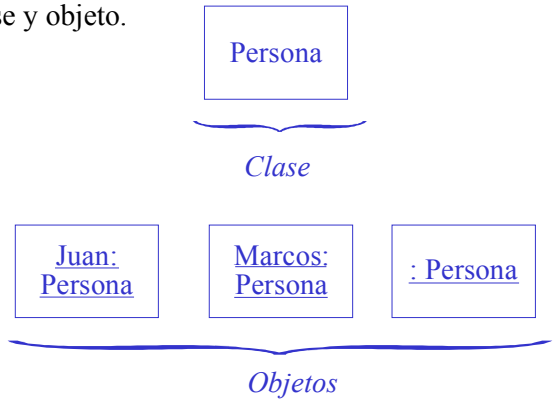
Concepto de objeto y de clase

- Todos los objetos son instancias de una clase y la clase de un objeto es una propiedad implícita del objeto.
  - Cada objeto *conoce* su clase y la mayoría de los lenguajes de programación orientados al objeto pueden determinar la clase de un objeto en tiempo de ejecución.

- La agrupación en clases de los objetos permite la abstracción de un problema:
  - Las definiciones comunes, tales como nombres de clases y de atributos se almacenan una vez por cada clase.
  - Las operaciones se escriben una vez para cada clase ⇒ reutilización de código.

Concepto de objeto y de clase

- En UML, una clase es un tipo de clasificador cuyas características son atributos y operaciones.
- Símbolos UML de clase y objeto.

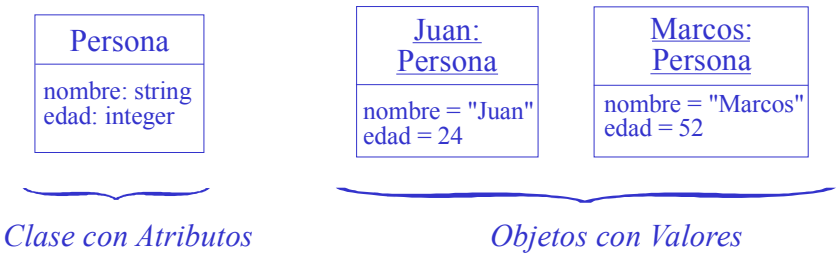


Atributos

- Las clases tienen *atributos* que representan alguna propiedad de la clase que comparten todos los objetos de esa clase.
- Un *atributo* es una propiedad nombrada de una clase, que describe un rango de valores que puede tomar esa propiedad en las instancias.
  - Por ejemplo, *nombre*, *edad* o *peso* son atributos de objetos *Persona*.
- Cada nombre de atributo es único dentro de una clase, pero cada atributo tiene un valor para cada instancia de la clase.
  - Diferentes instancias de objetos pueden tener los mismos o distintos valores para un atributo dado.
  - La identidad implícita del objeto permite distinguir objetos en que todos los valores de los atributos sean idénticos.

Atributos

- Un atributo debería ser un valor de datos puro, no un objeto.
  - Los valores de datos puros, a diferencia de los objetos, no tienen identidad.



Clase con Atributos *Objetos con Valores*

Operaciones

- Una *operación* es una función o transformación que puede ser aplicada por o sobre objetos de una clase.
  - Todos los objetos de una clase comparten las mismas operaciones.
  - Una operación es la implementación de un servicio que puede requerirse de cualquier objeto de la clase.

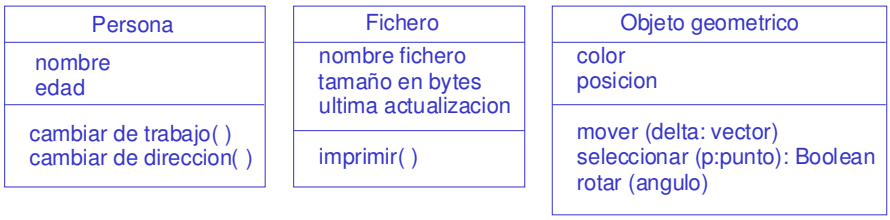
- Cada operación tiene a un objeto determinado como argumento implícito y el comportamiento de la operación depende de la clase de este objeto.
  - Un objeto conoce su clase y, por tanto, la implementación correcta de la operación.

Operaciones

- *Operación polimórfica*: la misma operación toma formas diferentes sobre clases diferentes.
  - Por ejemplo, la operación *mover* para una figura de dos dimensiones y para una figura de tres dimensiones.
- *Método*: implementación de una operación para una clase.
- Cuando una operación tiene métodos para diferentes clases es importante que todos ellos tengan la misma *signatura* (número y tipo de argumentos y tipo de resultado).
  - Los argumentos son parámetros de la operación pero no afectan a la elección del método. El método sólo depende de la clase del objeto sobre el que actúa.

Operaciones

- Las operaciones en una clase definen lo que la clase puede hacer y pueden considerarse como la *interfase* de la clase.
- *Operación pregunta (query)*: es aquélla operación que únicamente calcula un valor funcional sin modificar ningún objeto.
  - Las preguntas sin argumentos, salvo el objeto al que se aplican, pueden considerarse como atributos derivados.



Resumen de notación

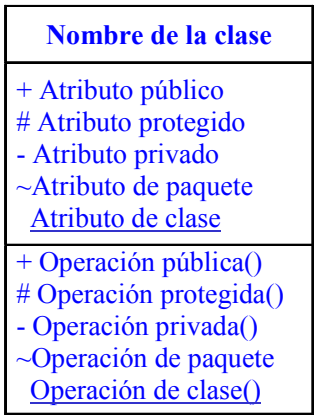
- Una clase se representa con un recuadro dividido en tres regiones, que contienen, de arriba a abajo:

Nombre de la clase
Nombre_de_atributo_1: tipo_de _dato_1 = valor_por_defecto_1 Nombre_de_atributo_2: tipo_de _dato_2 = valor_por_defecto_2 . . .
Nombre_de_operación_1 (lista_de_argumentos_1): tipo_de_resultado_1 Nombre_de_operación_2 (lista_de_argumentos_2): tipo_de_resultado_2 . . .

- Los atributos y operaciones pueden mostrarse o no, dependiendo del nivel de detalle deseado.

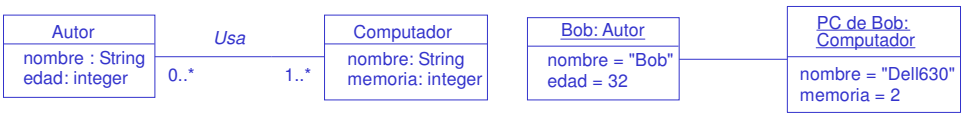
Resumen de notación

- Los atributos y las operaciones pueden tener diferentes niveles de *visibilidad*.
  - *Público*: visible por todos los clientes de la clase.
  - *Protegido*: visible por las subclases de la clase.
  - *Privado*: visible sólo para la clase.
  - *Paquete*: visible para cualquier clase del mismo paquete.
  - *Atributos y operaciones estáticos (static) o de clase*: son propios de la clase, no de la instancia. Son visibles por todos los objetos de la clase.



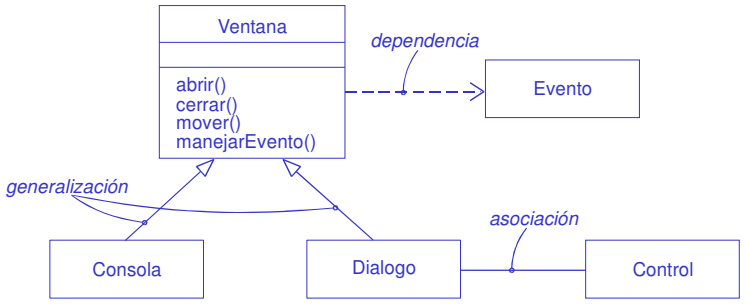
Diagramas de clases y de objetos

- Un *diagrama de clases* describe la estructura estática de un sistema en términos de clases y de relaciones entre estas clases, mostrando los atributos y operaciones que caracterizan cada clase de objetos.
- Un *diagrama de objetos* representa la estructura estática del sistema mostrando los objetos (instancias) en el sistema y las relaciones entre los objetos.
- Un diagrama de clases dado corresponde a un conjunto infinito de diagramas de objetos.



Relaciones entre clases

- *Asociación*: es una conexión entre clases, que implica la existencia de una relación estructural entre objetos de esas clases.
- *Generalización*: es una relación entre una clase más general y una más específica o especializada.
- *Dependencia*: es una relación de uso entre clases.

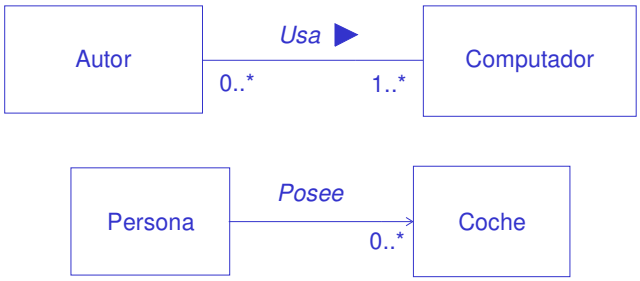


Enlaces y asociaciones

- Los enlaces y las asociaciones son los medios de establecer relaciones entre objetos y clases, respectivamente.
- *Enlace*: es una conexión física o conceptual entre instancias de objetos.
  - Matemáticamente, un enlace se define como una tupla, es decir, una lista ordenada de instancias de objetos.
- *Asociación*: describe un grupo de enlaces con estructura y semántica comunes.
  - Todos los enlaces de una asociación conectan objetos de las mismas clases.
  - Un enlace es una instancia de una asociación.

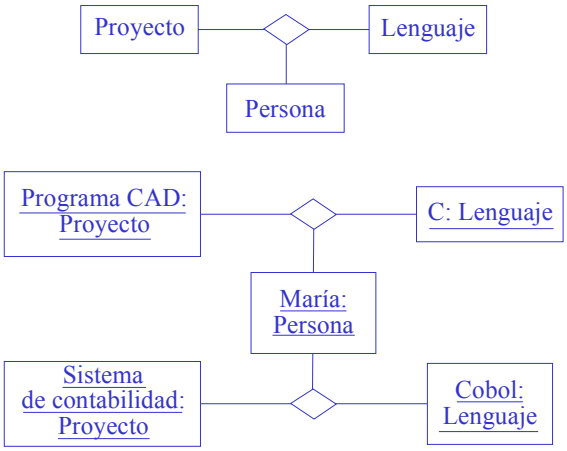
Enlaces y asociaciones

- Una asociación puede tener un nombre que describe la naturaleza de la relación.
- Las asociaciones son bidireccionales.
  - La dirección en que se lee el nombre de una asociación binaria es la dirección *directa* y la dirección opuesta es la dirección *inversa*.
  - La dirección del nombre puede indicarse con un pequeño triángulo sólido.
  - En ciertos casos, sólo es útil una dirección de navegación; esto se representa con una flecha orientada.



Enlaces y asociaciones

- Las asociaciones pueden ser binarias, ternarias o de más alto orden.
  - Una asociación ternaria o de orden mayor es una unidad atómica y no puede subdividirse en asociaciones binarias sin pérdida de información.



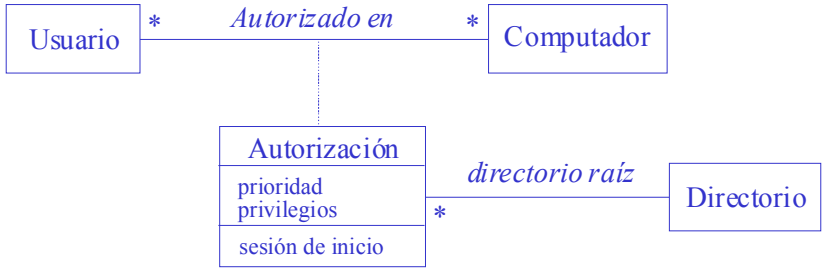
Multiplicidad

- La *multiplicidad* especifica cuántas instancias de una clase pueden relacionarse con una instancia simple de una clase asociada. Por tanto, limita el número de objetos relacionados.
  - En general, la multiplicidad es un subconjunto (posiblemente infinito) de los números enteros no negativos.
- En los diagramas de clases se indica la multiplicidad con símbolos especiales en los extremos de las líneas de las asociaciones.

1	Uno y sólo uno
0.. 1	Cero o uno
M.. N	De M a N (enteros naturales)
*	De cero a varios
0.. *	De cero a varios
1.. *	De uno a varios

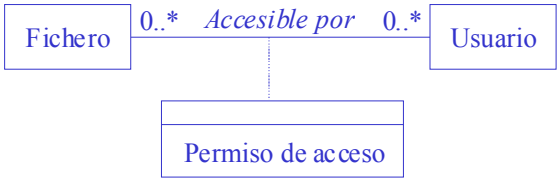
Clase Asociación

- Una asociación puede modelarse como una clase; en ese caso, cada enlace se convierte en una instancia de la clase.
- Es útil modelar una asociación como una clase cuando los enlaces pueden participar en asociaciones con otros objetos o cuando los enlaces están sometidos a operaciones.
  - En la notación UML, una clase asociación se representa con el símbolo de clase conectado a la asociación con una línea punteada.



Clase Asociación

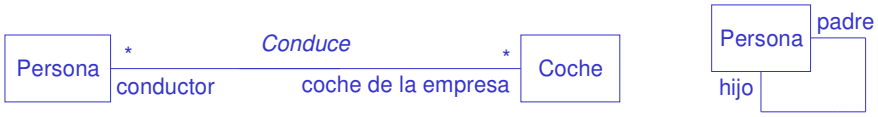
- Una asociación que contiene atributos sin participar en relaciones con otras clases se llama asociación *atribuida*.
- El *atributo de un enlace* es una propiedad de los enlaces de una asociación, que no se puede asociar a los objetos relacionados por el enlace.
  - Cada atributo de un enlace tiene un valor para cada enlace.
  - El símbolo UML de un atributo de enlace es un caso especial de clase asociación en el que aparecen uno o más atributos del enlace.



/etc/termcap	(read)	Juan
/etc/termcap	(read-write)	María
/usr/juan/.login	(read-write)	Juan

Papeles (roles) en una Asociación

- Un *papel (role)* es un extremo de una asociación.
  - Una asociación binaria tiene dos papeles, cada uno de los cuales puede tener un nombre.
- El *nombre de un papel* identifica de manera única un extremo de una asociación y describe el significado de la clase en términos de la asociación.
  - Cada papel en una asociación binaria identifica un objeto o un conjunto de objetos asociados con un objeto en el otro extremo.
  - El nombre del papel es un atributo derivado de la clase fuente, cuyo valor es el conjunto de los objetos relacionados.
  - Los nombres de papeles se necesitan en asociaciones entre dos objetos de la misma clase y para distinguir dos asociaciones entre el mismo par de clases.

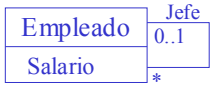


Calificación de asociaciones

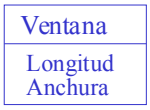
- Una *asociación calificada* relaciona dos clases de objetos y un calificador.
- El *calificador* es un atributo especial que reduce la multiplicidad efectiva de una asociación.
  - Las asociaciones *uno a muchos* y *muchos a muchos* deben ser calificadas.
  - El calificador distingue entre el conjunto de objetos en el extremo *muchos* de una asociación, especificando cómo identificar un objeto determinado.
  - Una asociación calificada puede considerarse también como una forma de asociación ternaria.
  - Un calificador se representa con un recuadro en el extremo de la asociación cerca de la clase que califica.
- La calificación mejora la exactitud semántica e incrementa la visibilidad de los caminos de navegación.

Restricciones (constraints)

- Las *restricciones* son relaciones funcionales entre entidades de un diagrama de clases o de objetos.
  - Una restricción limita los valores que las entidades pueden tomar.
  - El término *entidad* incluye objetos, clases, atributos, enlaces y asociaciones.
- La notación UML para las restricciones consiste en delimitarlas con llaves y colocarlas cerca de la entidad restringida.
  - También se pueden mostrar en una nota conectada a la entidad a la que se aplican.
  - Otras veces, una línea punteada conecta diversas entidades restringidas.
- Los diagramas capturan algunas restricciones a través de su propia estructura.
  - La multiplicidad restringe una asociación limitando el número de objetos relacionados con un objeto dado.



{Salario <= Jefe.Salario}

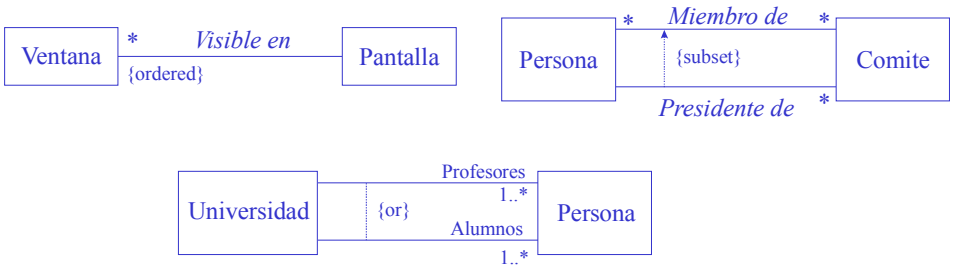


{0.8 <= longitud/anchura <= 1.5}

{La prioridad no incrementa}

Restricciones sobre las asociaciones

- Un conjunto ordenado de objetos en el extremo *muchos* de una asociación se indica escribiendo '{ordered (ordenado)}' cerca del punto que representa la multiplicidad, como si fuera un papel.
- La restricción '{subconjunto (subset)}' indica que una asociación es un subconjunto de otra.
- La restricción '{or}' indica que los objetos de una clase pueden participar en una sola asociación entre un grupo de asociaciones.

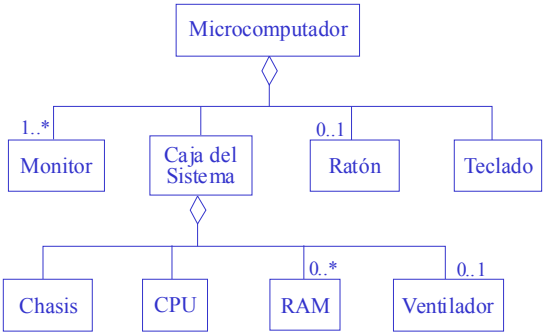


Agregación

- La *Agregación* es un caso especial de asociación. Es una relación **parte de** en la que los objetos que representan los *componentes* de algo se asocian con un objeto que representa el *agregado* completo.
  - La propiedad más significativa de la agregación es la *transitividad*, es decir, si A es parte de B y B es parte de C, entonces A es parte de C. También es *antisimétrica*, si A es parte de B, B no es parte de A.
  - Algunas propiedades del agregado se propagan a los componentes, posiblemente con modificaciones locales.
  - La agregación se dibuja como la asociación, excepto en que el extremo agregado de la relación se indica con un pequeño rombo.

Agregación

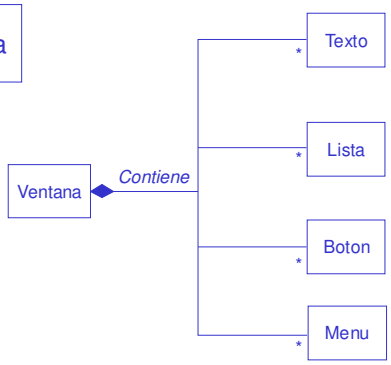
- La relación de agregación se define como una conexión entre la clase agregada y *una* clase componente.
  - Un agregado con muchas clases de componentes corresponde a muchas relaciones de agregación.
  - Cada emparejamiento individual se define como una agregación de modo que se puede especificar la multiplicidad de cada componente dentro del agregado.
  - La agregación puede tener un número de niveles arbitrario.



Agregación

- La multiplicidad en el lado del agregado puede ser superior a 1 ⇒ Agregación *compartida (shared)*.
  - En una agregación compartida los componentes pueden formar parte de diversos agregados.

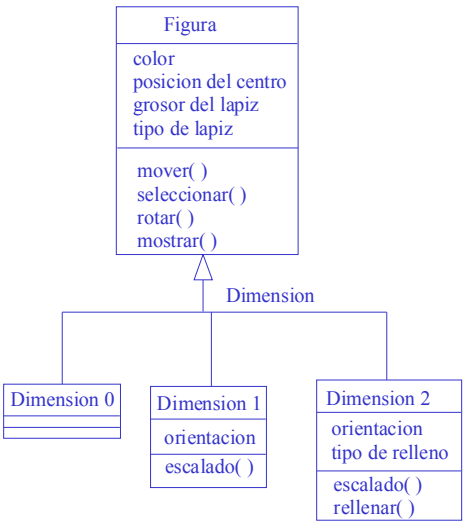
- La contención física es un caso particular de la agregación, llamado *composición*.
  - Los componentes no pueden existir sin el agregado, que es responsable de la creación y destrucción de sus componentes.
  - La composición se representa con un rombo negro en el extremo del agregado, cuya multiplicidad debe ser 0 ó 1.



- Una agregación compuesta forma un árbol de componentes, mientras que una agregación compartida forma una red.

Generalización y herencia

- La *generalización* es la relación entre una clase (*superclase*) y una o más versiones especializadas (*subclases*) de la misma.
  - Los atributos y operaciones comunes a un grupo de subclases se asocian a la superclase y son compartidos por las subclases.
  - Se dice que cada subclase *hereda* las características de su superclase, aunque también añade su propios atributos y operaciones específicos.
  - La notación UML para la generalización es una flecha que conecta la superclase con sus subclases, apuntando a la superclase.
  - Junto a la punta de flecha en el diagrama pueden colocarse un *discriminador*, un atributo de tipo enumeración que indica la propiedad de un objeto que se abstrae en una relación de generalización particular.



Generalización y herencia

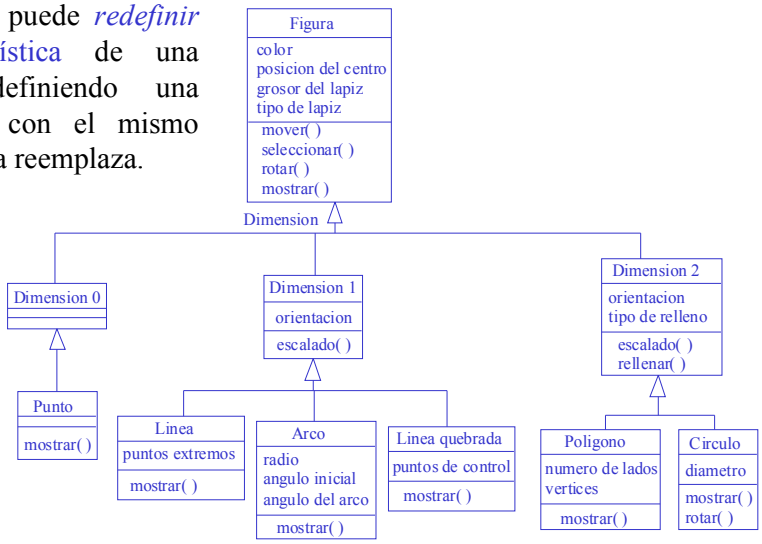
- La generalización es una relación de tipo **es un**, ya que cada instancia de una subclase es además una instancia de la superclase.
- La generalización facilita el modelado estructurando clases y capturando lo que es similar y lo que es diferente entre las clases.
  - La herencia de operaciones resulta de ayuda durante la implementación como un medio de reutilización de código.
- Los términos herencia, generalización y especialización se refieren a aspectos de la misma idea y a menudo se usan indistintamente.
  - Usamos *generalización* para hacer referencia a la relación entre las clases.
  - *Herencia* se refiere al mecanismo por el que se comparten atributos y operaciones usando la relación de generalización.
  - Generalización y especialización son dos perspectivas diferentes de la misma relación, observada desde la superclase o desde la subclase.
  - La palabra *generalización* deriva de que la superclase generaliza a las subclases y *especialización* se refiere a que las subclases perfeccionan o especializan a la superclase.

Generalización y herencia. Redefinición de características

- Una clase descendiente no puede omitir ni suprimir un atributo del ancestro, ya que entonces no sería una instancia del ancestro. Igualmente, las operaciones sobre la clase ancestro deben aplicarse a todas las clases descendientes.
- Una subclase puede añadir nuevas características (*extensión*).
- Una subclase puede implementar de nuevo una operación por razones de eficiencia pero no puede cambiar el protocolo externo.
- Hay varias razones para desear redefinir un aspecto de una clase:
  - Especificar un comportamiento que depende de la subclase.
  - Ajustar la especificación de ese aspecto.
  - Conseguir mejor rendimiento.

Redefinición de características

- Una subclase puede *redefinir* una característica de una superclase definiendo una característica con el mismo nombre, que la reemplaza.

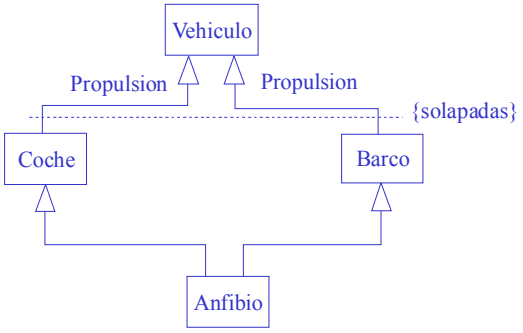


Redefinición de operaciones

- *Redefinición por extensión*. La nueva operación es la misma que la heredada, salvo en que añade algún nuevo comportamiento.
  - Por ejemplo, *Ventana* tiene una operación *dibujar* que dibuja los límites y el contenido de la ventana. *Ventana* podría tener una subclase llamada *Ventana_etiquetada* que tenga una operación *dibujar* cuyo método debería implementarse invocando al método para dibujar una ventana y además añadir código para dibujar la etiqueta.
- *Redefinición por restricción*. La nueva operación restringe el protocolo.
  - Por ejemplo, una superclase *Conjunto* puede tener la operación *añadir(elemento)*. La subclase *Conjunto_de_enteros* debería tener la operación más restrictiva *añadir(entero)*.
- *Redefinición por optimización*. Una implementación puede tomar ventaja de las limitaciones impuestas por una restricción para mejorar el código de una operación.
  - Por ejemplo, la superclase *Conjunto_de_enteros* podría tener una operación para obtener el máximo entero que puede implementarse mediante búsqueda secuencial. La subclase *Conjunto_ordenado_de_enteros* debería proporcionar una implementación más eficiente de la operación *máximo*, al estar ordenado el contenido del conjunto.
- *Redefinición por conveniencia*.

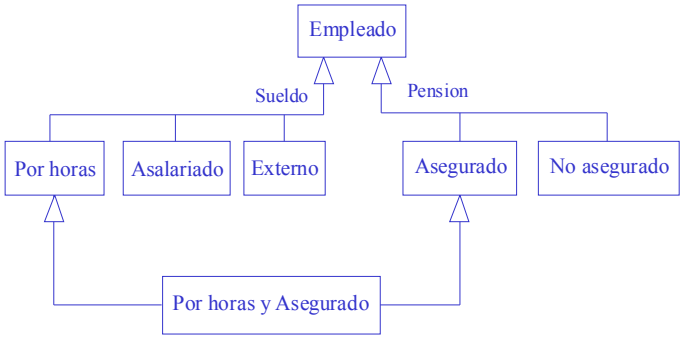
Herencia múltiple

- La *herencia múltiple* o solapada (*overlapping generalization*) permite que una clase tenga más de una superclase y herede características de todos sus padres.
- Una clase puede heredar características de más de una superclase y en este caso se denomina *clase unión*.
  - Una característica de la misma clase ancestro encontrada a través de más de un camino se hereda sólo una vez; es la misma característica.



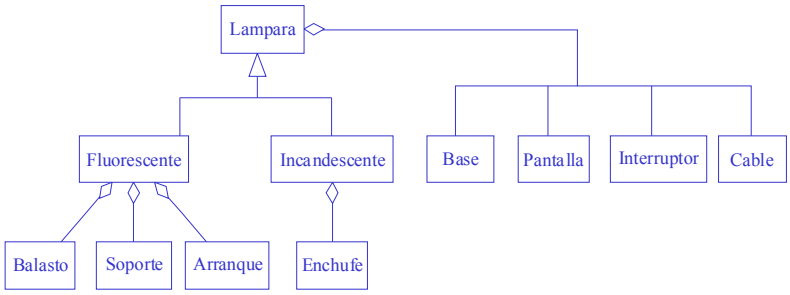
Herencia múltiple

- Si una clase se puede refinar en varias dimensiones distintas e independientes, se usan las generalizaciones múltiples.
  - Las subclases de la generalización pueden ser o no disjuntas.
  - Una clase puede heredar de forma múltiple desde distintas generalizaciones o desde distintas clases dentro de una relación de generalización solapada, pero nunca desde dos clases en la misma generalización disjunta.
  - El término *herencia múltiple* se puede referir tanto a la relación conceptual entre clases (*generalización*) como al mecanismo del lenguaje que implementa esa relación (*herencia*).



Generalización frente a Agregación

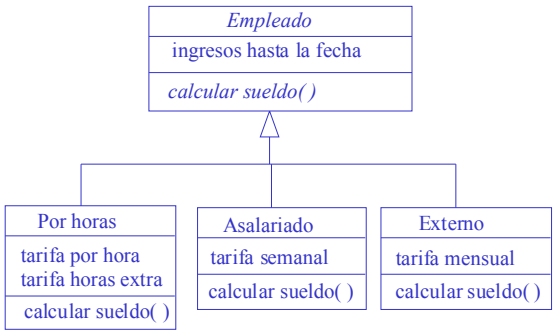
- La *agregación* relaciona instancias y la *generalización* relaciona clases.
  - En la agregación se relacionan dos objetos distintos, uno de ellos es parte del otro.
  - Con la generalización, un objeto es simultáneamente una instancia de la superclase y de la subclase.
  - Un árbol de agregación se compone de instancias de objetos que son parte de un objeto compuesto. Un árbol de generalización se compone de clases que describen un objeto.
  - La agregación suele denominarse relación **and** o **parte de** y la generalización, relación **or**, **tipo de** o **es un**.



Clases abstractas

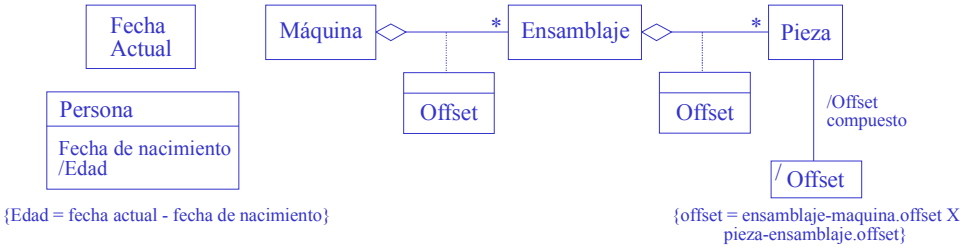
- Una *clase abstracta* es una clase sin instancias directas pero cuyas clases descendientes tienen instancias directas.
  - El nombre de la clase abstracta se escribe en cursiva.
- Una *clase concreta* es una clase instanciable (puede tener instancias directas).
  - Una clase concreta puede tener subclases abstractas (pero estas deben tener a su vez descendientes concretas).
  - Sólo las clases concretas pueden ser clases terminales en el árbol de herencia.

- *Operación abstracta*: Una clase abstracta puede definir el protocolo para una operación sin dar el método correspondiente.
  - Su nombre se escribe en cursiva.
  - Cada subclase concreta debe proporcionar su propia implementación.



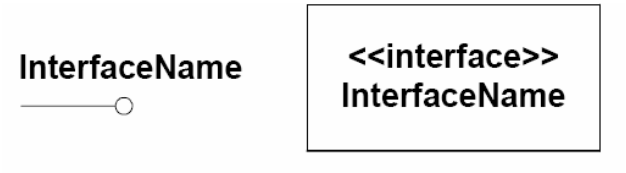
Objetos, enlaces y atributos derivados

- Las entidades derivadas están condicionadas por sus entidades base y la regla de derivación.
  - Un *objeto derivado* se define como una función de dos o más objetos, que a su vez pueden ser derivados. El árbol de derivación termina con objetos básicos.
  - Igualmente, hay *enlaces derivados* y *atributos derivados*.
  - La notación para una entidad derivada es una línea diagonal sobre la esquina del recuadro de una clase, sobre una línea de asociación o enfrente de un atributo y es una notación opcional. Debería mostrarse también la restricción que determina el valor derivado.



Interfases

- Una interfaz (*interface*) es un clasificador que contiene la declaración de un conjunto de operaciones sin su implementación correspondiente y se usa para especificar un servicio proporcionado por una clase o un componente.
  - Las interfases definen una separación entre la especificación (vista exterior) de lo que hace una abstracción y la implementación (vista interior) de cómo lo hace.
  - Se representa con el símbolo de un clasificador (rectángulo), precediendo el nombre con el estereotipo <<interface>>, o con una línea con un círculo en el extremo, etiquetado con el nombre de la interfaz.
  - Una interfaz puede participar en relaciones de generalización, asociación y dependencia.



Interfases

- Una interfaz no tiene instancias, se *realiza* en una instancia de una clase o un componente.
  - La clase o el componente proporcionan un conjunto de métodos que implementan apropiadamente las operaciones definidas en la interfaz.
  - Si una clase realiza una interfaz pero no implementa todas las operaciones especificadas, la clase debe ser abstracta.
- Una clase o un componente pueden realizar muchas interfases (*provided interface*) y depender de varias interfases (*required interface*).

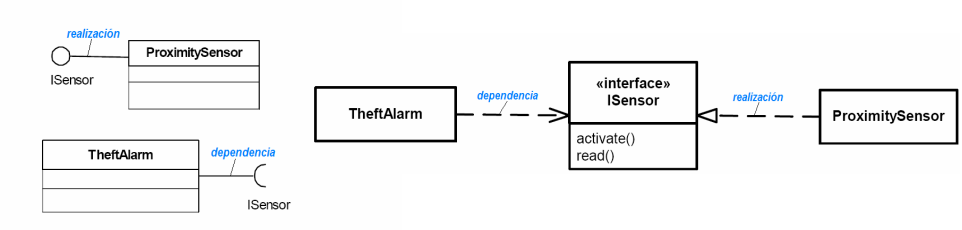


Diagrama de paquetes

- Los *paquetes* ofrecen un mecanismo general para la partición de los modelos y la organización de los elementos de modelado en grupos relacionados semánticamente.
  - Los paquetes ayudan a organizar los elementos en los modelos de modo que sean más fáciles de comprender, pero no tienen identidad (no pueden tener instancias).
  - Cada paquete corresponde a un subconjunto del modelo y contiene clases, objetos, relaciones, componentes, nodos o casos de uso, así como los diagramas asociados y otros paquetes.
  - Cada paquete debe tener un nombre que lo distingue de otros paquetes.

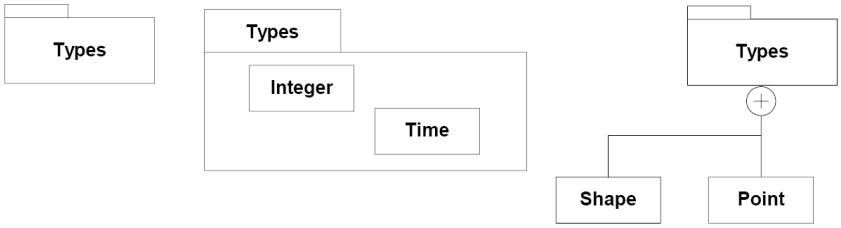


Diagrama de paquetes

- Para controlar el acceso a los elementos del paquete, se puede indicar la visibilidad de un elemento del paquete precediendo el nombre del elemento por un símbolo de visibilidad (´+´ para público y ´-´ para privado).
  - Los elementos con visibilidad *pública* son accesibles fuera del paquete y los elementos con visibilidad *privada* sólo están disponibles para elementos internos al paquete.

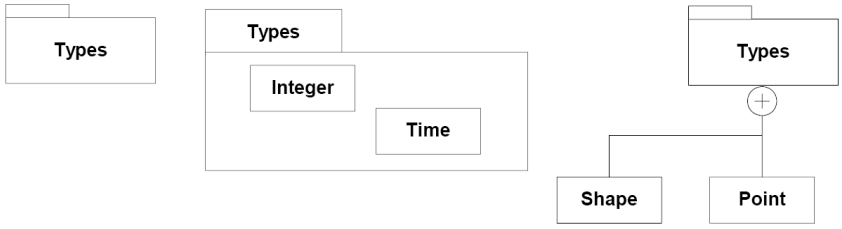


Diagrama de paquetes

- Los paquetes pueden importar elementos de otros paquetes, a través de una relación de dependencia entre paquetes con el estereotipo <<import>>.
  - Una relación de dependencia entre paquetes significa que al menos un elemento del paquete cliente utiliza los servicios ofrecidos por al menos un elemento del paquete proveedor.
  - Cuando se importa un paquete, sólo son accesibles sus elementos públicos.

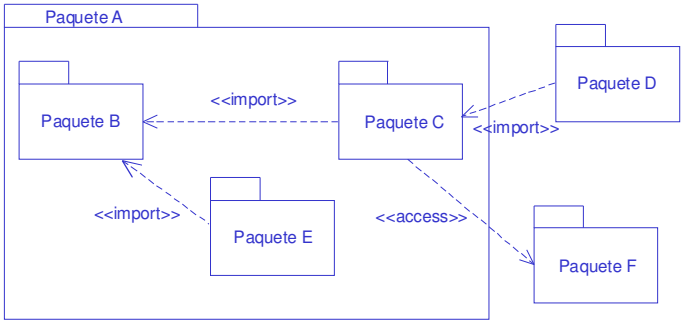


Diagrama de paquetes

- La relación de importación puede ser pública (por defecto, <<import>>) o privada (<<access>>).
  - Una importación pública significa que los elementos importados tienen visibilidad pública en el paquete cliente.
  - Una importación privada significa que los elementos importados tienen visibilidad privada en el paquete cliente y no pueden usarse fuera de él, incluyendo cualquier otro paquete que lo importara.

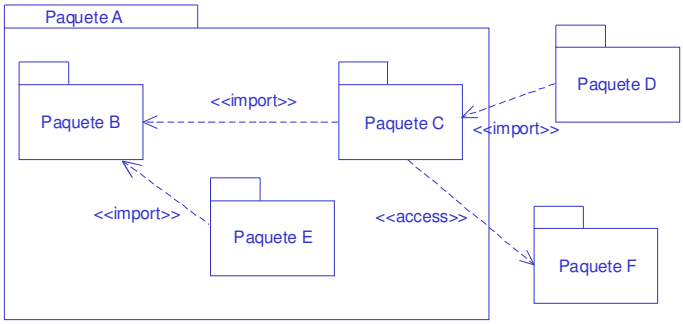


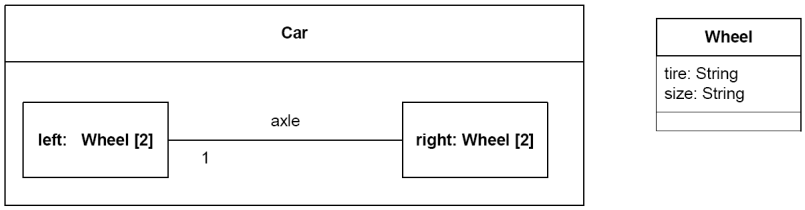
Diagrama de estructuras compuestas

- Una estructura es un conjunto de elementos interconectados que representan instancias en tiempo de ejecución, colaborando mediante enlaces de comunicación para conseguir objetivos comunes.

- Las estructuras compuestas (*composite structures*) permiten especificar situaciones de modelado como:
  - *Estructuras internas (internal structrures)*: muestran los componentes de una clase y las relaciones que mantienen en el contexto de la clase.
  - *Puertos (ports)*: son puntos de interacción entre una clase y el exterior.
  - *Colaboraciones (collaborations)*: muestran objetos que cooperan para conseguir un objetivo.

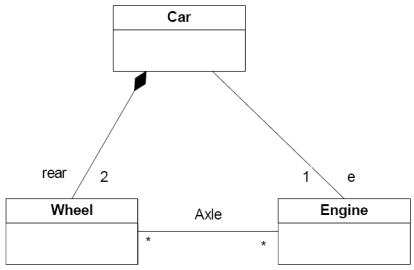
Estructura interna

- Para mostrar la estructura interna de una clase, los componentes (*parts*) que la clase contiene por composición se dibujan dentro del símbolo de la clase contenedora.
  - Los componentes se especifican mediante el papel (*rol*) que representan en la clase, con el formato <*roleName*>: <*type*>.
  - La multiplicidad se escribe en la parte superior derecha del componente o se indica entre corchetes después del nombre y del tipo.
- Las relaciones que existen entre los componentes se muestran dibujando un conector (*connector*) entre ellos, especificando la multiplicidad en cada extremo.
  - Un conector significa que puede existir comunicación entre las instancias en tiempo de ejecución de los componentes. Se puede dar nombre y tipo al conector.

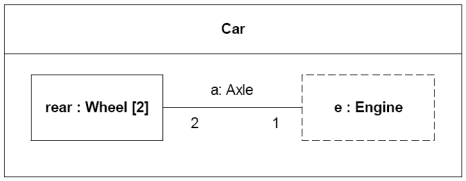


Estructura interna

- Además de componentes, también se pueden mostrar propiedades (*properties*), cuyo rectángulo se dibuja con una línea punteada.
  - Con una propiedad se especifica una instancia que la instancia de la clase contenedora no posee y puede estar compartida con otras estructuras.



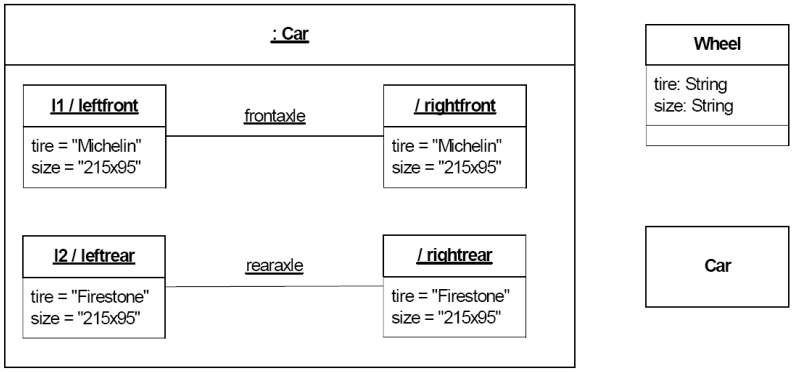
(i)



(ii)

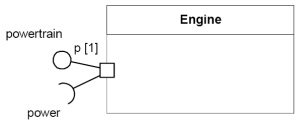
Estructura interna

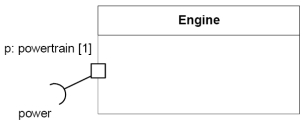
- Si se representa una instancia de una clase con estructura interna, se muestran también las instancias de sus componentes y propiedades.
  - Las instancias de los componentes y propiedades se especifican con el formato *{<name>} / <roleName>: <type>* subrayado.



Puertos

- Un puerto es un punto de interacción entre una clase y su entorno.
  - Cada uso de una clase se representa con un puerto, que se dibuja como un pequeño rectángulo en el borde de la clase y se le da un nombre para indicar su propósito.
  - También se puede especificar multiplicidad y tipo para un puerto.
- Las clases tienen interfases asociadas a los puertos para mostrar los servicios disponibles en el puerto (*provided interfaces*) o los servicios requeridos del entorno (*required interfaces*).
- Los puertos se conectan a la implementación interna usando conectores.
  - Es posible tener múltiples conectores desde un puerto hasta diferentes elementos internos.
- En la práctica, cuando se instancia un puerto, se representa con un clasificador que realiza las interfases ofrecidas (*provided interfaces*).
  - Cualquier comunicación con este punto de interacción consiste en pasar la información a los clasificadores internos que realizan ese comportamiento.



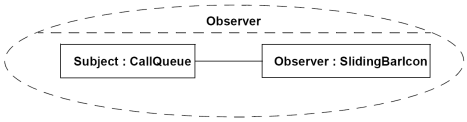


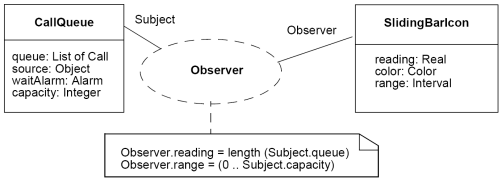
Colaboraciones

- Una colaboración describe una estructura de objetos que cooperan para conseguir una determinada funcionalidad.
  - Los objetos se definen por el papel que representan en esa situación concreta y la comunicación que existe entre ellos se muestra mediante conectores que los enlazan.
- En una colaboración se muestran solamente los aspectos que son necesarios para explicar un comportamiento particular, suprimiendo el resto.
  - Un objeto no está limitado al papel que representa en una colaboración determinada, puede representar distintos papeles en distintas colaboraciones.
  - Los objetos de una colaboración no le pertenecen, pueden existir antes y después de la colaboración.
  - Aunque los objetos de una colaboración estén enlazados, no se comunican necesariamente fuera de la colaboración.

Colaboraciones

- Una colaboración se representa con una elipse punteada con el nombre en su interior.
  - La estructura interna formada por los papeles de los objetos y los conectores se puede mostrar en un compartimento dentro de la elipse, nombrando a los objetos con el formato *<role>: <type>*.
- Una notación alternativa dibuja las clases de los objetos participantes fuera de la elipse, y las conecta a ella mediante una línea etiquetada con el nombre del papel correspondiente.





Colaboraciones

- Es posible asociar una colaboración a una operación o a un clasificador para mostrar cómo se aplica en un contexto determinado.
- En este caso, se crea un uso de la colaboración (*collaboration occurrence*) que puede considerarse como una instancia de la colaboración.
  - Puede haber múltiples usos de una colaboración en un clasificador, cada uno con un conjunto diferente de elementos internos que representan los papeles de la colaboración.
  - La notación para un uso de colaboración es la elipse punteada, indicando el nombre del uso seguido de ´:´ y del tipo de la colaboración.
  - Para cada papel de la colaboración original, se dibuja una línea punteada entre el uso de la colaboración y el elemento que representa ese papel, etiquetando la línea con el nombre del papel.

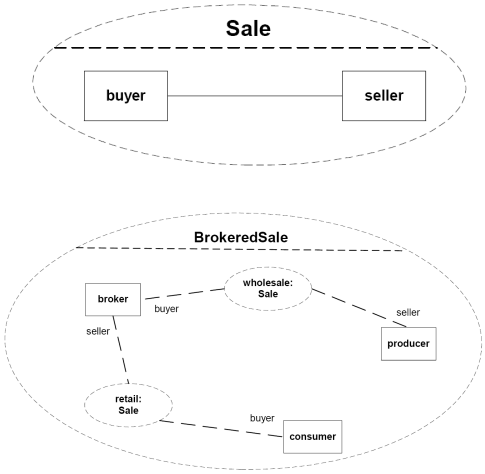
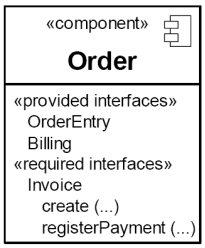


Diagrama de componentes

- El diagrama de componentes describe componentes software y sus dependencias.
  - En este diagrama, el sistema se organiza en módulos reemplazables dentro de su entorno, con interfases bien definidas.
- Un componente es una parte del software encapsulada, reutilizable y reemplazable.
  - Los componentes se pueden combinar entre sí para componer el software completo.
  - En UML, un componente puede tener relaciones de generalización y de asociación con otras clases o componentes, implementar interfases, tener operaciones, etc. Además, como en las estructuras compuestas, pueden tener puertos y mostrar estructura interna.
- Un componente suele tener mayor responsabilidad que una clase.
  - Es común que un componente contenga y use otras clases y componentes para hacer su trabajo.
- Entre los componentes debe existir un acoplamiento débil, para que los cambios en uno de ellos no afecten al resto del sistema. Para ello, se accede a los componentes por medio de interfases.

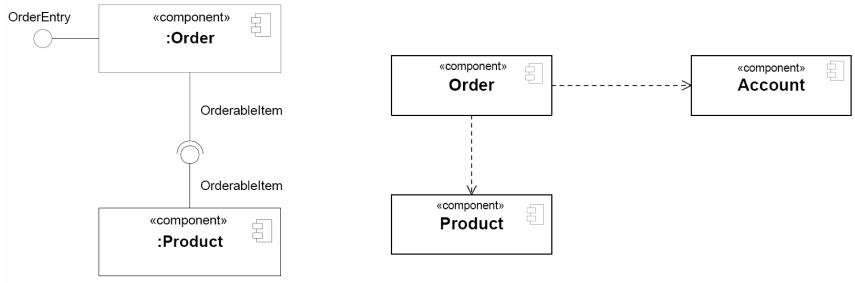
Componentes

- Un componente se muestra en UML como un rectángulo con el estereotipo <<component>> y un icono opcional en la esquina superior derecha.
  - El nombre del componente se escribe dentro del rectángulo.
  - Se puede mostrar que el componente es un susbsistema de un sistema mayor reemplazando <<component>> con <<subsystem>>.
- Los componentes interactúan a través de las interfases ofrecidas y requeridas, usando la misma notación que con las clases.
  - Las interfases también se pueden mostrar como una lista dentro del símbolo de componente, separando las interfases ofrecidas y las requeridas.



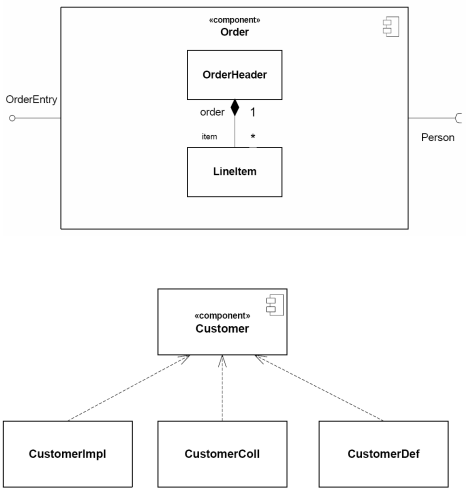
Dependencia entre componentes

- Los componentes pueden necesitar a otros componentes para implementar su funcionamiento.
- Si un componente requiere una interfaz, necesita que otra clase o componente se la ofrezca.
  - Se representa uniendo los símbolos de interfaz ofrecida y requerida (*assembly connector*).
  - Se puede omitir la representación de las interfases y mostrar las dependencias entre componentes con flechas de dependencia entre ellos.



Realización de un componente

- Un componente normalmente contiene y usa otras clases para implementar su funcionamiento. Se dice que estas clases realizan al componente.
  - Se representa dibujando las clases y sus relaciones dentro del componente.
  - Se pueden dibujar las clases fuera del componente, con flechas de dependencia conectando las clases y el componente.
  - Otra posible representación consiste en listar las clases en un compartimento <<realization>> dentro del componente.



Puertos en un componente

- Un componente puede tener puertos para modelar las distintas formas de uso mediante las interfases conectadas al puerto.
- La interfaz ofrecida por un componente puede ser realizada por uno de sus elementos internos. Del mismo modo, una interfaz requerida por el componente puede ser usada por uno de sus elementos internos.
  - Para mostrar que los elementos internos del componente realizan o usan las interfases del componente, se usan conectores de delegación (*delegation connectors*). Estos conectores muestran la dirección de la comunicación y se representan con una flecha entre el puerto y el elemento interno.
    - Si el elemento realiza la interfaz, la flecha apunta desde el puerto al elemento. Si el elemento usa la interfaz, la flecha apunta desde el elemento hasta el puerto.
    - Si se representan las interfases de los elementos internos, los conectores de delegación se unen a la interfaz. Suele hacerse cuando un componente contiene a otros componentes.

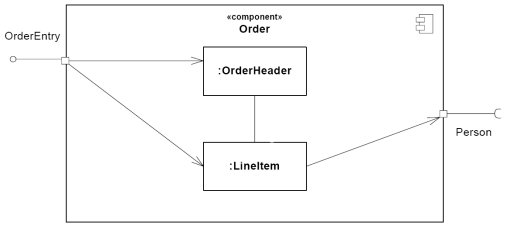


Diagrama de despliegue (deployment)

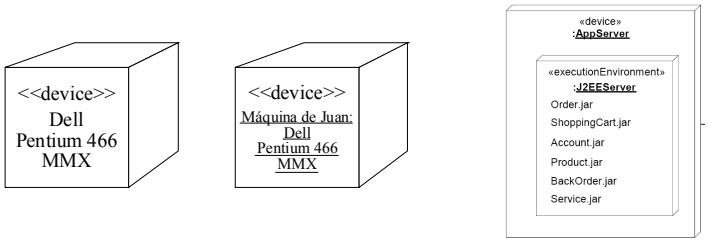
- El diagrama de despliegue representa la arquitectura en tiempo de ejecución de procesadores y dispositivos (*nodes*), y los artefactos (*artifacts*) software que se ejecutan en esa arquitectura.
  - Es la descripción física final de la topología del sistema, que describe la estructura de las unidades hardware y el reparto de los programas ejecutables sobre esas unidades.

- Los nodos (*nodes*) son objetos físicos (dispositivos) que tiene alguna clase de recurso de computación (computadores con procesadores, impresoras, etc.).

- Los artefactos (*artifacts*) son ficheros físicos que el software usa o ejecuta (ficheros ejecutables, librerías, ficheros fuente, de configuración, etc.).

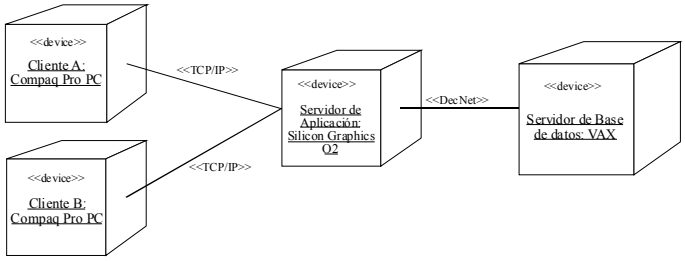
Nodos

- Un nodo se usa para representar un recurso hardware o software, en el que puede residir el software ejecutable o los ficheros relacionados.
  - Un nodo se dibuja como un cubo con el nombre dentro. Para recalcar que se trata de un nodo hardware, se usa el estereotipo <<device>>, y para indicar que se trata de un entorno de ejecución, el estereotipo <<executionEnvironment>>.
  - Un entorno de ejecución debe funcionar en un dispositivo concreto, lo que se muestra dibujando un nodo dentro de otro. Los servicios que ofrece el entorno de ejecución se pueden listar en el interior de su símbolo.
  - Un nodo puede mostrarse como una instancia usando la notación name: type subrayada. El tipo describe las características de un procesador o dispositivo y la instancia representa existencias (máquinas) de ese tipo.



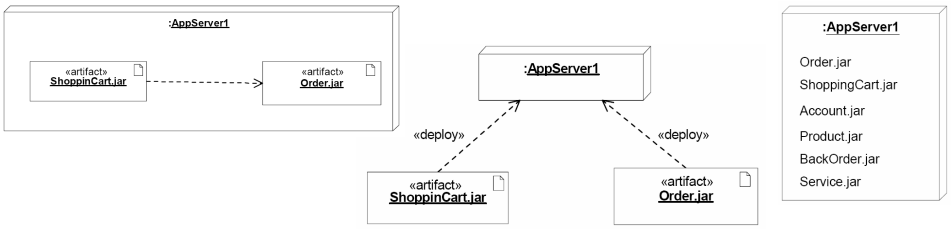
Comunicación entre nodos

- Los nodos se conectan entre sí mediante caminos de comunicación (*communication paths*), que se dibujan con una línea que une los nodos.
- Las conexiones indican que hay alguna clase de comunicación entre los nodos y que los nodos intercambian mensajes a través de ese camino de comunicación.
  - El tipo de comunicación se representa con un estereotipo que identifica el protocolo de comunicación o la red utilizados.



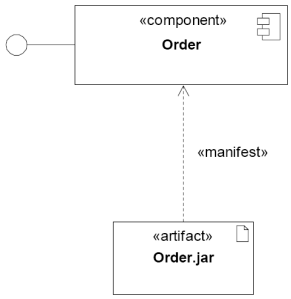
Artefactos

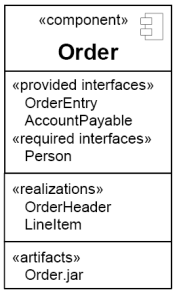
- El software que se ejecuta en los nodos se modela con artefactos (*artifacts*), que son ficheros físicos que el software usa o ejecuta (librerías, ficheros fuente, ficheros ejecutables o ficheros de configuración).
  - Un artefacto se representa mediante un rectángulo con el estereotipo <<artifact>> y un icono de documento en la esquina superior derecha.
- Un artefacto se despliega en un nodo, indicando que está instalado en el nodo.
  - Se puede representar de tres formas: dibujando el símbolo del artefacto dentro del nodo; con una flecha de dependencia desde el artefacto hasta el nodo, con el estereotipo <<deploy>>; o listando el nombre del artefacto dentro del nodo.
  - Si un artefacto usa a otro, se conectan sus símbolos con una flecha de dependencia.



Artefactos

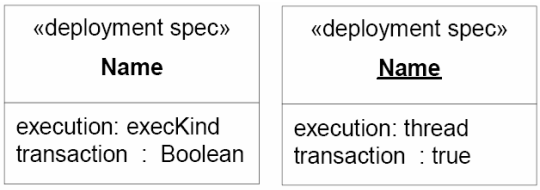
- Si un artefacto representa la versión compilada de un componente o de un paquete, en UML se dice que el artefacto manifiesta (*manifest*) a ese componente, paquete o clase.
  - Esta relación se muestra con una flecha de dependencia desde el artefacto hasta el componente, con el estereotipo <<manifest>>.
- La relación de manifestación proporciona el medio de modelar cómo se organizan los componentes software sobre el hardware del sistema.
- En el diagrama de componentes, se listan los artefactos que manifiestan a un componente dentro del símbolo del componente.





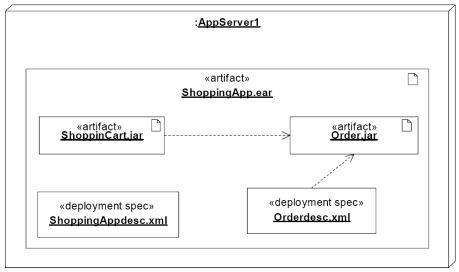
Especificaciones de despliegue

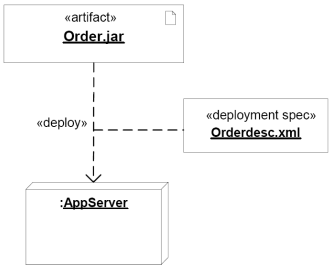
- Una especificación de despliegue (*deployment specification*) es una colección de propiedades que describen cómo se despliega un artefacto en un nodo, para que funcione adecuadamente en su entorno.
  - Es un artefacto especial que se representa mediante un rectángulo con el estereotipo <<deployment spec>>. Las propiedades se listan dentro del símbolo como atributos, usando la notación name: type.
  - Tanto los artefactos como las especificaciones de despliegue se pueden modelar como instancias, subrayando sus nombres. En la instancia de una especificación de despliegue, se muestran los valores de sus propiedades en lugar de los tipos.



Especificaciones de despliegue

- La relación entre un artefacto y su especificación de despliegue se muestra dibujando a ambos dentro del nodo y conectados con una flecha de dependencia desde la especificación hasta el artefacto.
- Otra posible representación consiste en conectar la especificación a la flecha de despliegue (<<deploy>>) con una línea punteada.





Uso de los diagramas de despliegue

- Los diagramas de despliegue son útiles en todas la etapas del proceso de modelado. Se requiere un procedimiento iterativo para obtener el diagrama de despliegue final.
  - Los diagramas de despliegue se revisan añadiendo detalles cuando se deciden las tecnologías, protocolos de comunicación y artefactos software que se van a utilizar.
- Permiten visualizar el estado actual de la estructura física del sistema y discutir diversas soluciones con el resto de los participantes en su desarrollo.
- Idealmente, el sistema será flexible de modo que un artefacto específico pueda moverse entre diferentes nodos.