# Temas de Hoy

Modelos
y
Diagramas

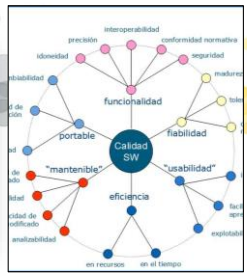



# Retomando…

# Arquitecturas de Software


**Arquitectura de Software**

Historia
Definiciones

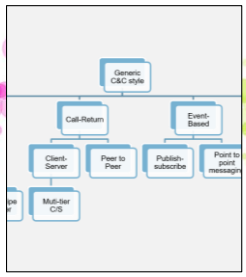



**Atributos de Calidad**

Taxonomía

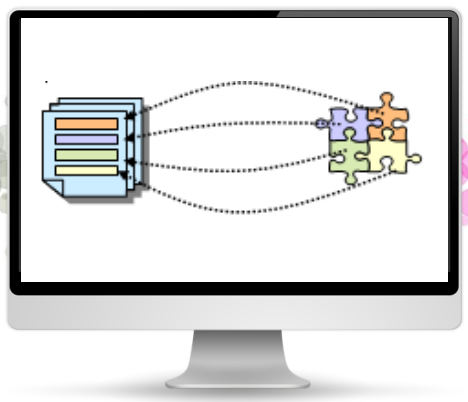

**Vistas y Viewtypes**

4+1
Module Viewtype
C&C Viewtype
Allocation Viewtype




**Estilos Arquitectónicos**

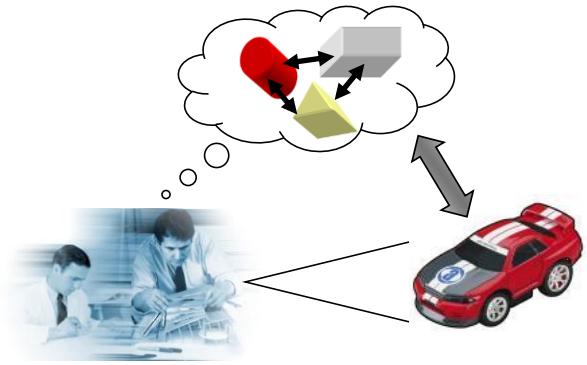
Descomposición
Layers
Call/Return
Pipe&Filter
Publish&Suscribe



# Modelos y Diagramas

# Modelo

El modelo es conocimiento depurado





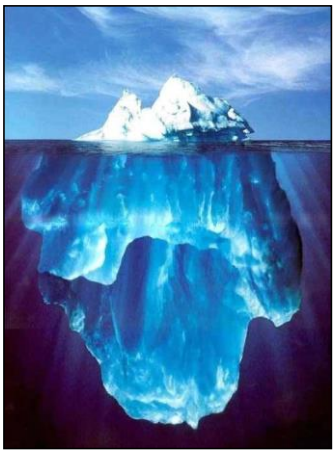
# Conocimiento implícito y explícito

El conocimiento sobre un proyecto está fragmentado, repartido entre muchas personas y documentos, y está mezclado con otra información de tal manera que ni siquiera conocemos cuáles son los fragmentos de información que realmente necesitamos.

- **Conocimiento explícito**: este conocimiento puede expresarse en palabras y números y compartido en la forma de datos, fórmulas científicas, especificaciones, manuales, etc. Puede transmitirse entre las personas formal y sistemáticamente
- **Conocimiento implícito**: es en gran medida personal y muy difícil de formalizar, dificultando mucho la comunicación o el ser compartido por otros. Los pálpitos subjetivos, las intuiciones, los presentimientos caen dentro de esta categoría. Es difícil de verbalizar, dado que está íntimamente enraizado en las acciones y experiencias de una persona, además de los ideales, valores o emociones que esa persona pueda adoptar.

# Conocimiento implícito y explícito

Relación -de acuerdo a varios autores- entre conocimiento tácito y explícito



10% Conocimiento visible, comunicable, formalizable (Explícito)

90% Conocimiento oculto, ligado a la experiencia (Implícito)



# Modelos y Diagramas

- Un **modelo** captura una vista de un sistema del mundo real. Es un a abstracción de dicho sistema, considerando un cierto propósito. Así, el modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.
- **Diagrama**: una representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo con vértices conectados por arcos.

# Modelos y Diagramas

- Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés
- El código fuente del sistema es el modelo más detallado del sistema (y además es ejecutable). Sin embargo, se requieren otros modelos ...
- Cada modelo es completo desde su punto de vista del sistema, si n embargo, existen relaciones de trazabilidad entre los diferentes modelos



# ¿Qué es UML?

# ¿Qué es UML?

- UML = **Unified Modeling Language**
- Un lenguaje de propósito general para el modelado orientado a objetos
- Documento "***OMG Unified Modeling Language Specification***"
- Combina notaciones provenientes de:
  - Modelado Orientado a Objetos
  - Modelado de Datos
  - Modelado de Componentes
  - Modelado de Flujos de Trabajo (Workflow)



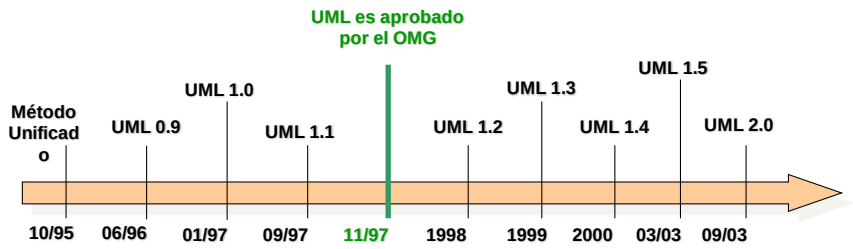
# Punto de Partida

- Diversos métodos y técnicas OO, con muchos aspectos en común pero utilizando distintas notaciones
- Inconvenientes para el aprendizaje, aplicación, construcción, y uso de herramientas, etc.
- Pugna entre distintos enfoques

**Establecer una notación estándar**

# Historia de UML

- Comenzó como el “Método Unificado”, con la participación de Grady Booch y Jim Rumbaugh. Se presentó en el OOPSLA’95 .
- El mismo año se unió Ivar Jacobson. Los tres fueron socios de la compañía Rational Software.



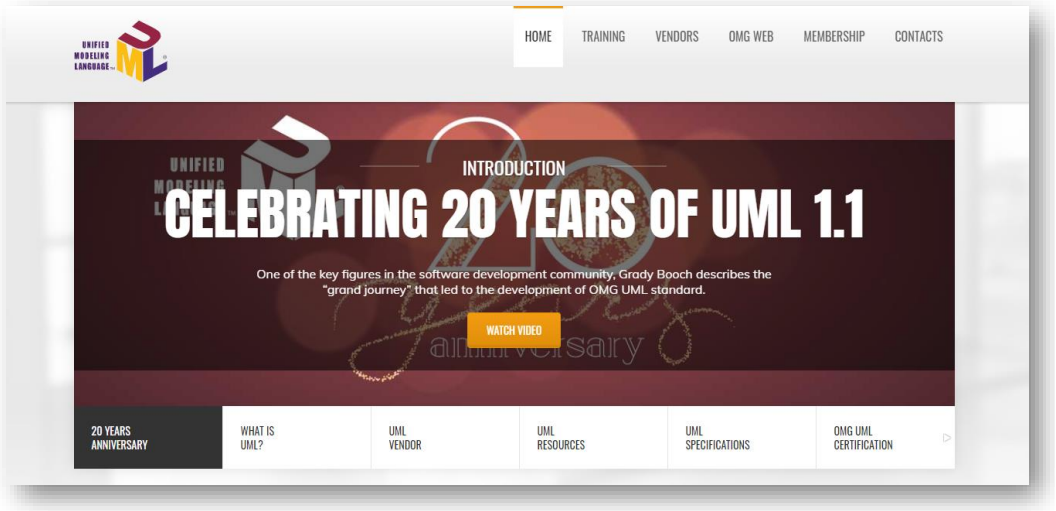



# Aspectos Novedosos UML 2.0

- **Mecanismos de Extensión en UML:**
  - Estereotipo
  - Restricción de Integridad
  - Valores Etiquetados, es un par (nombre propiedad, valor)

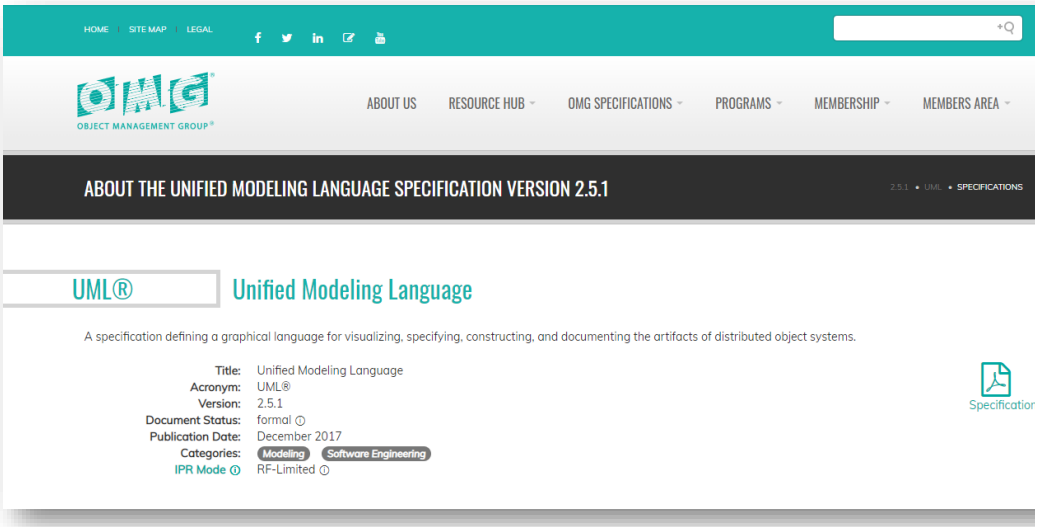
**Permiten adaptar los elementos de modelado, asignándoles una semántica particular.**

# https://www.uml.org/






# Versión vigente: 2017



# Diagramas de UML

Diagramas Estructurales
- Diagrama de Casos de Uso
- Diagrama de Clases
- Diagrama de Objetos

Diagramas de Comportamiento
- Diagrama de Estados
- Diagrama de Actividad

Diagramas de Interacción
- Diagrama de Secuencia
- Diagrama de Colaboración

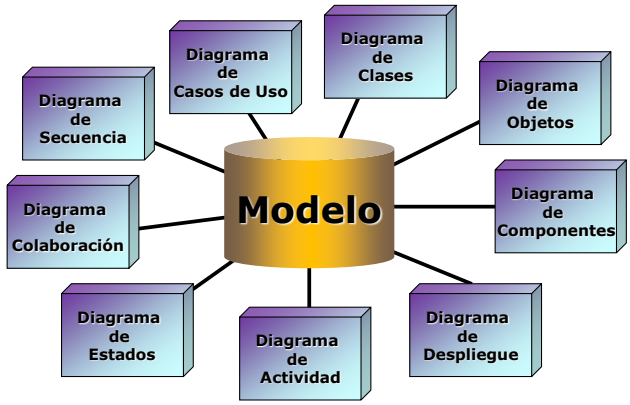
Diagramas de Implementación
- Diagrama de Componentes
- Diagrama de Despliegue



# Inconvenientes

- Definición del proceso de desarrollo usando UML: **UML no es una metodología**.
- Falta integración con respecto a otras técnicas tal es como patrones de diseño, interfaces de usuario, documentación, etc.
- “Monopolio de conceptos, técnicas y métodos en torno a UML”

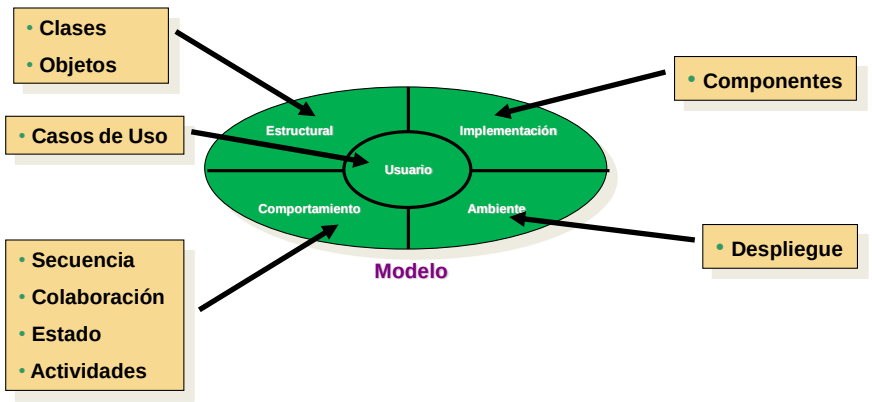
# Diagramas de UML




Los diagramas expresan gráficamente partes de un modelo.



# Diagramas de UML




Cinco vistas y nueve diagramas

# Documentos y Diagramas

- La primera percepción de la utilidad de un diagrama UML -un diagrama de clases u objetos- puede ser cuando **discutimos** sobre un diseño de software.
- Mucha gente es naturalmente visual y los diagramas ayudan a captar cierto tipo de información: los diagramas UML son muy útiles para comunicar **relaciones** entre objetos y para mostrar **interacciones**.
- Pero no transmiten la definición conceptual de estos objetos. En una discusión deberemos rellenar estos significados hablando mientras vamos dibujando estos diagramas, o surgirán en un diálogo con otros participantes.



# Documentos y Diagramas

- Es decir que un diagrama UML **no puede transmitir** dos de los aspectos más importantes de un modelo: el significado de los conceptos que representa y qué se supone que hacen los objetos.
- Esta es la función complementaria del lenguaje hablado.
- Los diagramas son un medio de comunicación y de explicación y facilitan el proceso de creación de ideas (brainstorming).

# Documentos y Diagramas

- Deben ser **mínimos**: diagramas completos -aunque sean comprensibles- del modelo total de objetos fracasan en su intento de comunicar o explicar. Agobian al lector con detalles al mismo tiempo que carecen de los significados más importantes.
- Algunos prefieren utilizar los diagramas UML al revés: en vez de un diagrama con anotaciones en forma de texto que aclaren detalles, prefieren un documento escrito ilustrado con diagramas seleccionados y simplificados.



# Documentos y Diagramas

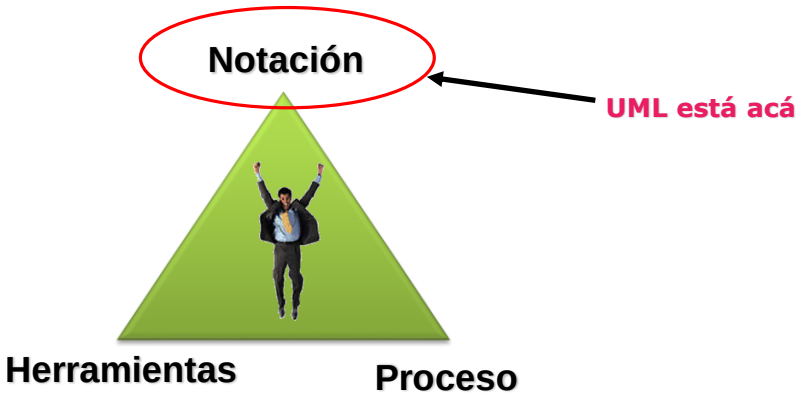
- Siempre debemos recordar que ***el modelo no es el diagrama*** (lo mismo que el mapa no es el territorio), el propósito del mismo es ayudar a comunicar y explicar el modelo.

# Documentos y Diagramas

- La comunicación hablada complementa el rigor de los diagramas y del código, pero aunque sea vital para conectar a todos con el modelo, a partir de una cierta cantidad de integrantes de un equipo se necesita la estabilidad y el poder compartir que ofrecen los documentos escritos.
  - Los documentos deben complementar al modelo y a las conversaciones.
  - Un documento no debería intentar hacer lo que el código hace perfectamente.
  - Los documentos deberían servir para seguir funcionando y estar actualizados.
  - Un documento debe estar involucrado en las actividades del proyecto.



# Modelado de Software




Triángulo de Éxito

# UML

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

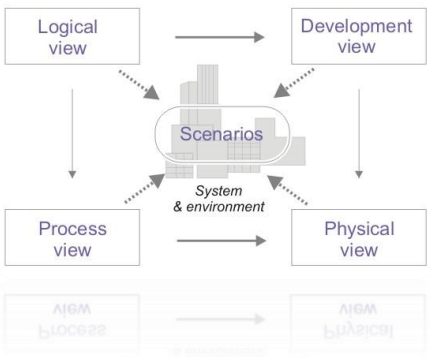
## Objetivos de UML

- Suministrar un lenguaje visual de modelado expresivo con el cual se pudieran crear e intercambiar modelos inteligibles.
- Proveer mecanismos de extensión y especialización para ampliar los conceptos básicos.
- Ser independiente de cualquier lenguaje de programación y de cualquier proceso de desarrollo.

# Objetivos de UML

- Incluir los fundamentos formales para entender el lenguaje.
- Impulsar el desarrollo del mercado de herramientas OO.
- Soportar los conceptos de desarrollo de alto nivel, tales como colaboraciones, marcos (*frameworks*), patrones (*patterns*) y componentes.






# Diagrama de Casos de Uso

# Casos de Uso

Un ***requerimiento funcional*** describe un servicio o función del sistema. Un ***requerimiento no-funcional*** es una restricción sobre el sistema (por ejemplo el tiempo de respuesta) o sobre el proceso de desarrollo (por ejemplo el uso de un lenguaje específico).

- Los Casos de Uso (Ivar Jacobson) describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde el **punto de vista del usuario**.
- Permiten definir los límites del sistema y las relaciones entre el sistema y el entorno.
- Son descripciones de la funcionalidad del sistema independientes de la implementación
- Los Casos de Uso particionan el conjunto de necesidades atendiendo a la categoría de usuarios que participan en el mismo.
- Están basado en el lenguaje natural, es decir, es accesible por los usuarios.

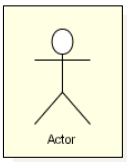


# Casos de Uso: Actores

Un Actor:

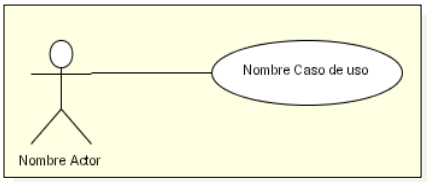
- Representa un tipo de usuario
- Es una agrupación uniforme de personas, sistemas o máquinas que interactúan con el sistema de la misma forma

- ✓ Los actores se representan con dibujos simplificados de personas, llamados en inglés “stick man” (hombres de palo). La notación puede adecuarse al contexto.
- ✓ La misma persona física puede interpretar varios pape les como actores distintos.
- ✓ El nombre del actor describe el papel desempeñado.



# Casos de Uso

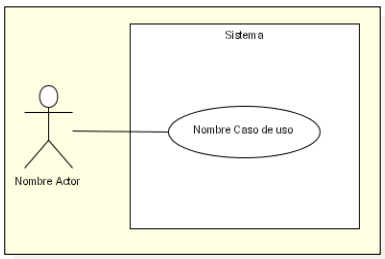
- Un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia.
- Es iniciado por un actor.
- El nombre se expresa con un verbo en gerundio.
- Se expresa desde el punto de vista del actor.





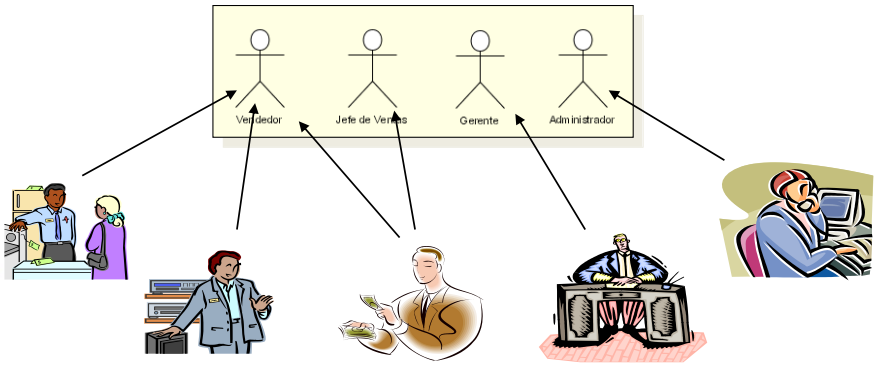

# Casos de Uso

- Otro objetivo de los diagramas de casos de uso es colaborar con la determinación del alcance del sistema.
- El sistema se representa con un rectángulo, dentro del cual se ubican los casos de uso.
- Los actores se encuentran fuera del alcance del sistema.



# Ejemplo

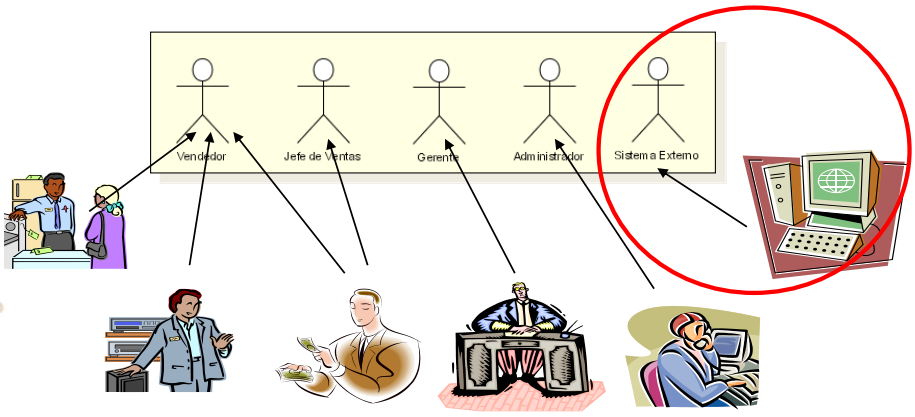
"El sistema de un local de venta de electrodomésticos es utilizado por los vendedores, los jefes de ventas, el gerente y el administrador del sistema…"






# Ejemplo

"…también el sistema deberá ser capaz de recibir órdenes de compra en viadas por el sistema actual de facturación llamado Facturator IV…"



# Casos de Uso: Descripción

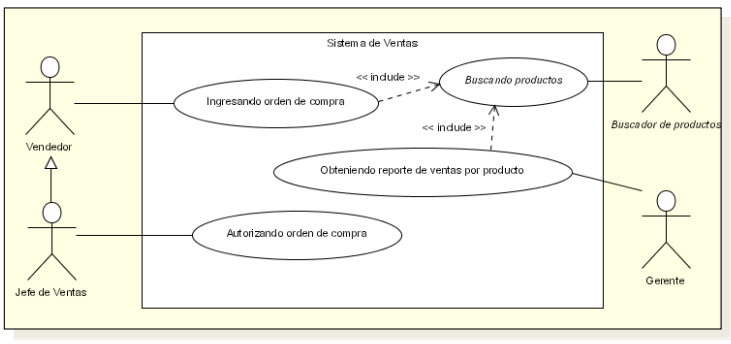
- Los casos de uso se documentan con texto informal.
- En general, se usa una lista numerada de los pasos que sigue el actor para interactuar con el sistema (Curso normal).

| **Caso de Uso**: Ingresando Orden de Compra | |
|---|---|
| **Actor**: Vendedor | |
| ***Curso Normal*** | ***Alternativas*** |
| 1. El vendedor ingresa el número de cliente en el sistema. | |
| 2. El sistema obtiene la información básica sobre el cliente. | 2.1 Si el cliente no está registrado, debe registrarse primero. |
| 3. El vendedor ingresa el código del producto que el cliente quiere comprar, informando su cantidad. | |
| 4. El sistema obtiene información del producto solicitado, y confirma su disponibilidad. | 4.1 Si no hay disponibilidad del producto, el sistema informa la fecha de reposición. |
| 5. Se repite el paso 3 hasta que el cliente no solicita más productos. | |
| 6. El sistema registra la orden de compra | |
| 7. Fin del caso de uso | |



# Ejemplo

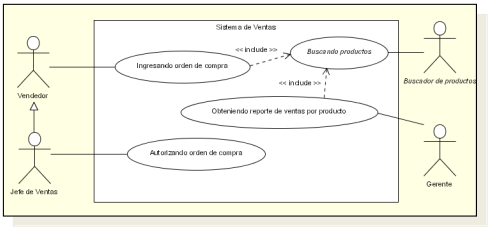
"… el gerente podrá consultar un reporte de ventas por producto: tras b uscar y seleccionar el producto elegido, el sistema le mostrará la inform ación correspondiente…"

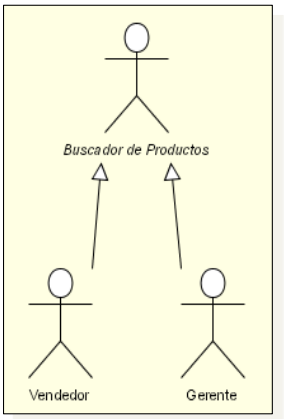



En ambas descripciones debería aparecer una referencia a la necesidad de buscar un producto. Este "buscador de productos" es una funcionalidad común a ambos casos de uso.

# Ejemplo

Los actores concretos Vendedor y Gerente heredan del actor abstracto Buscador de Productos

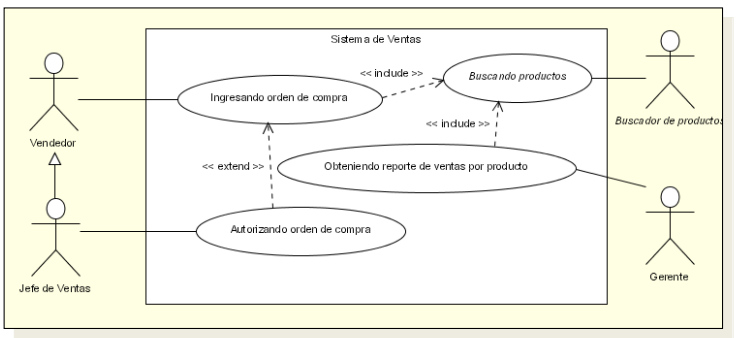





# Ejemplo

"… cuando es el jefe de ventas quien está ingresando una orden de co mpra podrá, opcionalmente, autorizarla inmediatamente …"



# Casos de Uso: Construcción

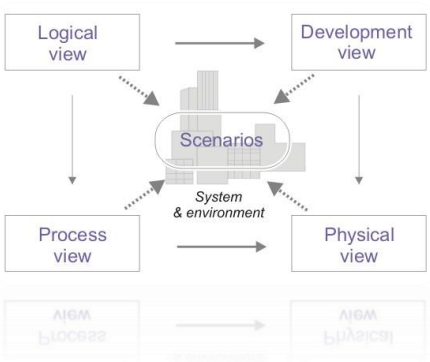
**La descripción del Caso de Uso comprende:**

- Nombre.
- Descripción.
- Actor asociado.
- Requerimientos asociados.
- Pre y post condiciones.
- Secuencia de pasos con sus correspondientes alternativas.
- Pueden distinguirse los manejos de errores.
- Bifurcaciones e iteraciones en el curso normal.
- Puntos de extensión y uso.



# Casos de Uso: Descripción

| | | |
|---|---|---|
| RF- <id del requisito> | <nombre del requisito funcional> | |
| Versión | <numero de versión y fecha> | |
| Autores | <autor> | |
| Fuentes | <fuente de la versión actual> | |
| Objetivos asociados | <nombre del objetivo> | |
| Descripción | El sistema deberá comportarse tal como se describe en el siguiente caso de uso { concreto cuando <evento de activación> , abstracto durante la realización de los casos de uso <lista de casos de uso>} | |
| Precondición | <precondición del caso de uso> | |
| Secuencia Normal | Paso | Acción |
| | 1 | {El <actor> , El sistema} <acción realizada por el actor o sistema>, se realiza el caso de uso < caso de uso RF-x> |
| | 2 | Si <condición>, {el <actor> , el sistema} <acción realizada por el actor o sistema>>, se realiza el caso de uso < caso de uso RF-x> |
| | 3 | |
| | 4 | |
| | 5 | |
| | 6 | |
| | n | |
| Postcondición | <postcondición del caso de uso> | |
| Excepciones | Paso | Acción |
| | 1 | Si <condición de excepción>,{el <actor> , el sistema} }<acción realizada por el actor o sistema>>, se realiza el caso de uso < caso de uso RF-x>, a continuación este caso de uso {continua, aborta} |
| | 2 | |
| | 3 | |
| Rendimiento | Paso | Cota de tiempo |
| | 1 | n segundos |
| | 2 | n segundos |
| Frecuencia esperada | <n° de veces> veces / <unidad de tiempo> | |
| Importancia | {sin importancia, importante, vital} | |
| Urgencia | {puede esperar, hay presión, inmediatamente} | |
| Comentarios | <comentarios adicionales> | |




# Diagrama de Actividad

# Diagrama de Actividad

- El Diagrama de Actividad es una especialización del Diagrama de Estado, organizado respecto de las acciones y usado para especificar:
  - Un método
  - Un caso de uso
  - La interacción entre varios casos de uso
  - Explotar un paso del curso normal de un caso de uso.
  - Un proceso de negocio (Workflow)

# Diagrama de Actividad

- Las actividades se unen por transiciones automáticas.
- Cuando una actividad termina se desencadena el paso a la siguiente actividad.
- Un Diagrama de Actividad está asociado a la implementación de un caso de uso.
- El propósito de este diagrama es enfocarse en los flujos manejados por el procesamiento interno (en contraposición con eventos externos).

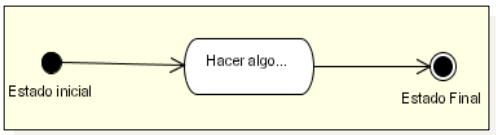


# Diagrama de Actividad

- Este tipo de diagrama no es adecuado en situaciones donde ocurren eventos asincrónicos.
- Se centran en el procesamiento interno del sistema para cumplir una funcionalidad.
- De esta forma, un caso de uso puede estar acompañado por cero, uno o más diagramas de actividad.

# Diagrama de Actividad: Acción

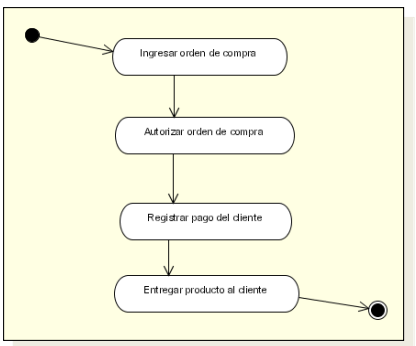
- Una acción es mostrada como una figura con la parte superior e inferior recta y con arcos convexos en los dos lados (estado de acción).
- La *expresión de acción* se ubica dentro del símbolo. La expresión de acción no es necesariamente única dentro del diagrama, y debe comenzar con un verbo en infinitivo.
- El uso normal de una acción es modelar un paso o un conjunto de pasos en la ejecución de un algoritmo (un procedimiento).





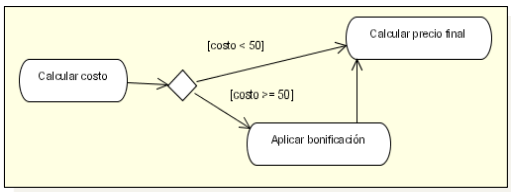

# Ejemplo

"…el proceso de negocio que describe una venta se inicia con la orden de compra, una vez que está autorizada, se debe registrar el pago del cliente para luego realizar la entrega del producto…"



# Diagrama de Actividad: Decisión

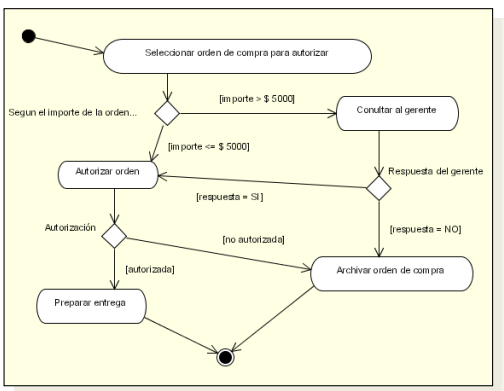
- Un diagrama de actividad expresa una decisión cuando una condición es usada para indicar diferentes transiciones posibles que dependen de un valor booleano.
- El ícono provisto para una decisión es la tradicional figura con forma de diamante, con una o más flechas entrantes y con dos o más flechas salientes, cada una etiquetada por una condición diferente y sin evento que la dispare.
- Todos los posibles valores para la condición deben aparecer en las transiciones salientes.





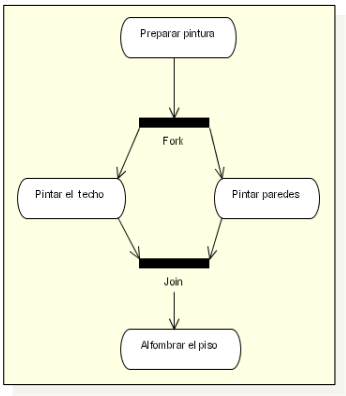

# Ejemplo

"… las órdenes de compra cuyo importe sea menor a $5000 pueden ser autorizadas directamente por el jefe de ventas. En cambio, si la orden supera dicho valor se le deberá consultar al gerente. Las órdenes autorizadas deberán ser preparadas para su entrega. Las órdenes no autorizadas deberán ser archivadas …"



# Diagrama de Actividad: Fork y Join

- Los diagramas de actividad pueden representar concurrencia, mediante los operadores fork y join.

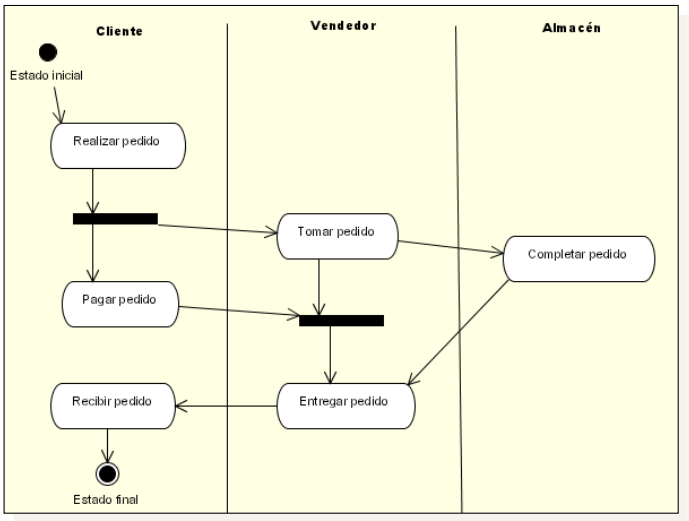





# Diagrama de Actividad: Andariveles

- Las acciones pueden ser organizadas en *andariveles*.
  Los andariveles se usan para organizar las responsabilidades de las actividades.
- Usualmente corresponden a unidades organizacionales dentro de un modelo de negocio (por ejemplo áreas de una empresa).
- También pueden corresponderse con los actores de los casos de uso.
- Contribuyen a la identificación de responsabilidades.
- Cada acción es asignada a un único andarivel.

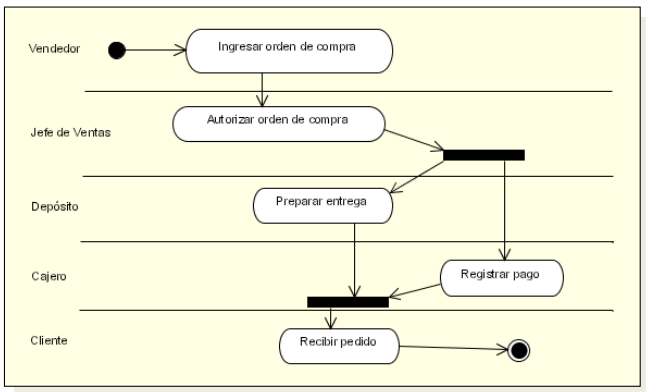
# Diagrama de Actividad: Andariveles

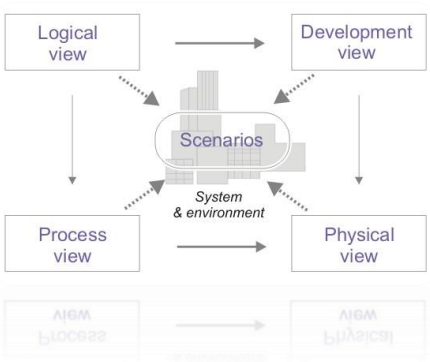





# Ejemplo

"… de esta manera las órdenes ingresadas por los vendedores, son autorizadas por el jefe de ventas. Una vez autorizada, el depósito es el responsable de preparar la entrega. C uando la órden se encuentre paga, el cliente se encontrará en condiciones de recibir s u pedido …"






# Diagrama de Estados

## Diagrama de Estados

- Los diagramas de estados representan autómatas de estados finitos, desde el punto de vista de los estados y las transiciones.
- Son útiles sólo para los objetos con un comportamiento significativo.
- El formalismo utilizado proviene de los Statecharts (Harel).

# Diagrama de Estados

- Cada objeto está en un estado en cierto instante.
- El estado está caracterizado parcialmente por los valores algunos de los atributos del objeto.
- El estado en el que se encuentra un objeto determina su comportamiento.
- Cada objeto sigue el comportamiento descrito en el diagrama de estados asociado a su clase.

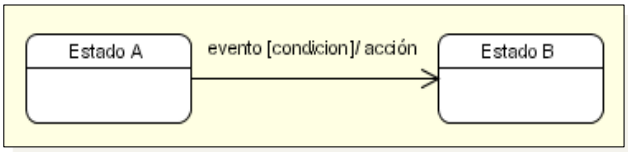


# Diagrama de Estados

- Son autómatas jerárquicos que permiten expresar concurrencia, sincronización y jerarquías de objetos.
- Son grafos dirigidos.
- Los diagramas de estado de UML son determinísticos.
- Los estados inicial y final están diferenciados del resto.
- La transición entre estados es instantánea y se debe a la ocurrencia de un evento.

# Diagrama de Estados

- Estados y Transiciones

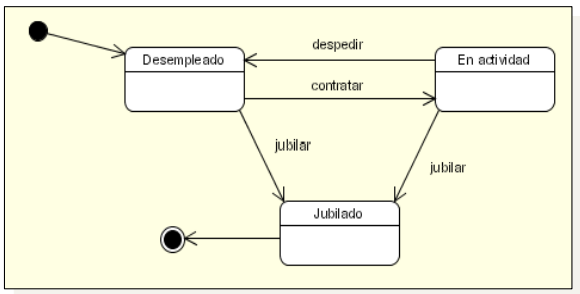



Tanto el evento como la acción se consideran instantáneos



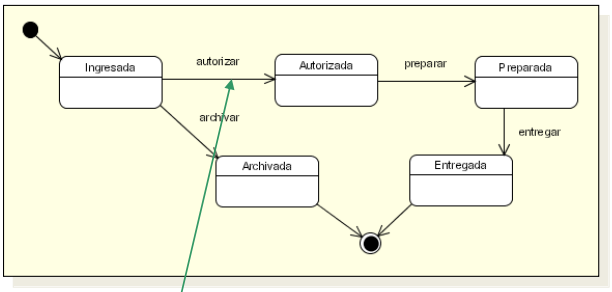
# Ejemplo

Estados de una persona con respecto a su vida laboral.



# Ejemplo

"… una orden de compra, a lo largo del tiempo, podrá estar ingresada, autorizada, preparada, archivada o entregada …"

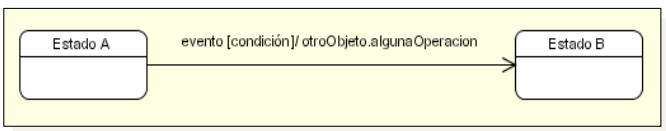



**Estímulos que recibe el objeto Orden Compra**



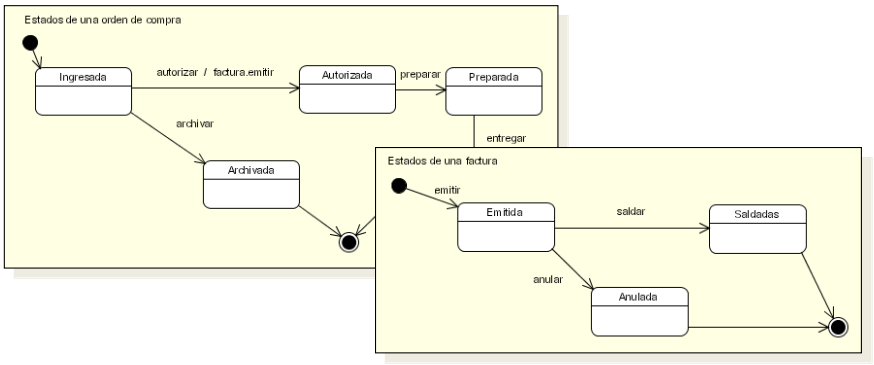
# Diagrama de Estados

- Podemos especificar la solicitud de un servicio a otro objeto como con secuencia de la transición:



# Ejemplo

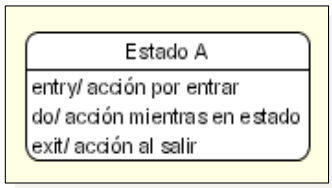
"… al autorizar una orden de compra, se emite una factura que debe ser cancelada por el cliente… "





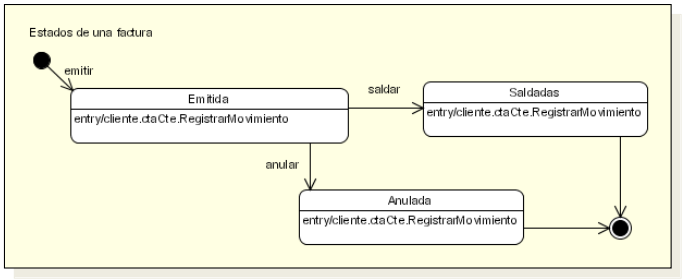

# Diagrama de Estados

- Se puede especificar ejecutar una acción como consecuencia de entrar, salir, estar en un estado, o por la ocurrencia de un evento:



# Ejemplo

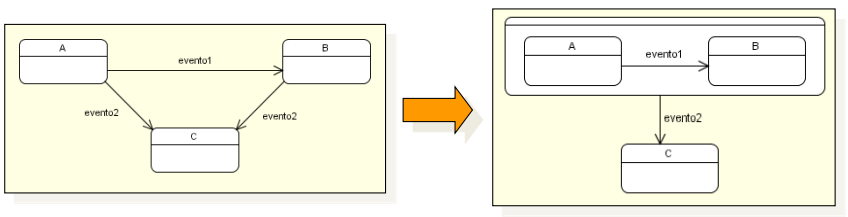
"… al emitir una factura, se debe registrar un movimiento en la cuenta corriente del cliente para que modifique su saldo. Lo mismo debe suceder al saldarla y/o anularla … "





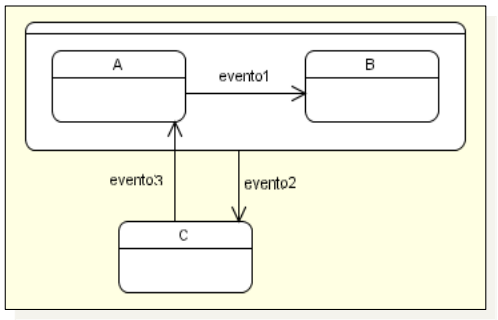

# Diagrama de Estados: Composición

- Podemos reducir la complejidad de estos diagramas usando la generalización de estados.
- Distinguimos así entre superestado y subestados.
- Un estado puede contener varios subestados disjuntos.
- Los subestados heredan las variables de estado y las transiciones externas.



# Diagrama de Estados: Composición

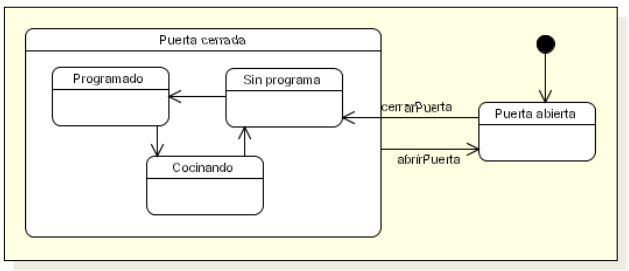
- Las transiciones de entrada deben ir hacia subestados específicos:





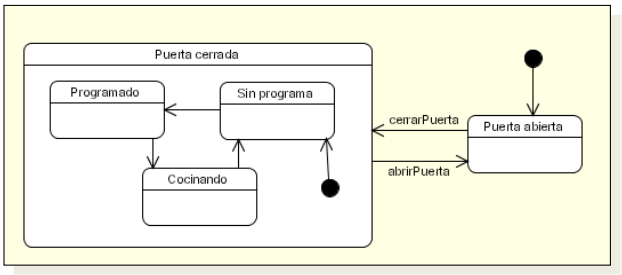

# Ejemplo

Descripción simplificada de un horno microondas.



# Ejemplo

Descripción simplificada de un horno microondas.



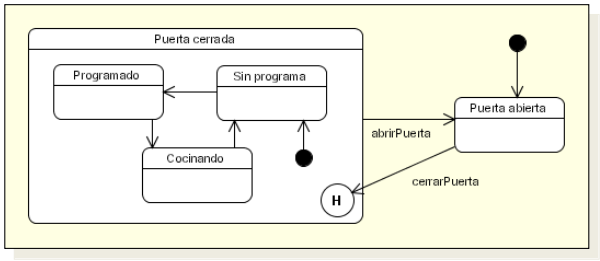



# Diagrama de Estados: Historia

- Por defecto, los autómatas no tienen memoria.
- Es posible memorizar el último subestado visitado para recuperarlo en una transición entrante en el superestado que lo engloba.
- También es posible la memorización para cualquiera de los sub estados anidados (aparece un * junto a la H).

# Ejemplo

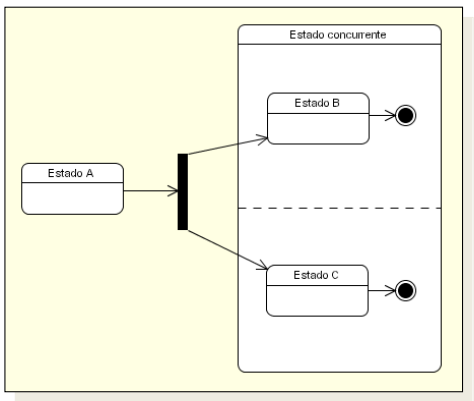
Cuando se cierre la puerta, el microondas regresa al estado en el cual se encontraba antes de abrirla…





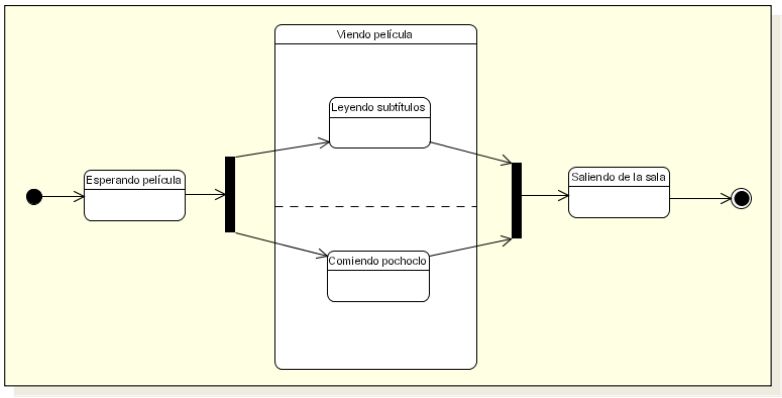

# Diagrama de Estados: Concurrencia

- Al igual que en los diagramas de actividad, se cuenta con los operado res fork y join.

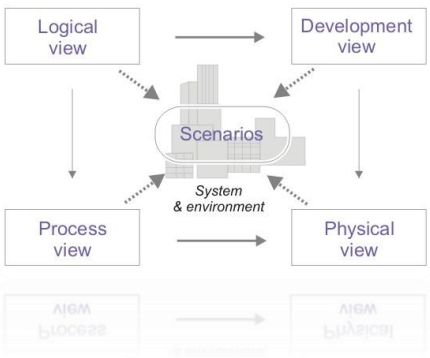


# Ejemplo

Imaginen un espectador de cine…






# Diagrama de Clases

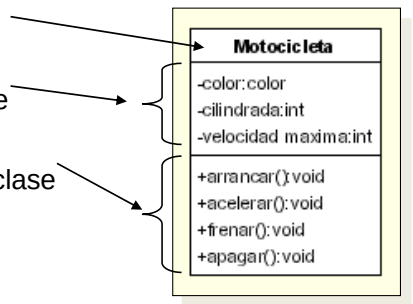
# Diagrama de Clases

- El propósito de este diagrama es el de representar los objetos fundamentales del sistema, es decir los que percibe el usuario y con los que espera tratar para completar su tarea en vez de objetos del sistema o de un modelo de programación.
- La clase define el ámbito de definición de un conjunto de objetos.
- Cada objeto pertenece a una clase.
- Los objetos se crean por instanciación de las clases.



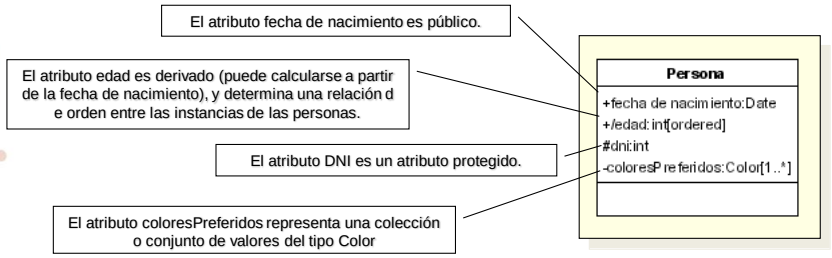
# Diagrama de Clases

- Cada clase se representa en un rectángulo con tres compartimientos:
  - Nombre de la clase
  - Atributos de la clase
  - Operaciones de la clase



# Diagrama de Clases: Atributos

- **Tipo**: puede llegar a depender del lenguaje de programación a utilizar.
- **Valor inicial**: valor que poseerá el atributo al crear un objeto.
- **Visibilidad**: está relacionado con el encapsulamiento.
- **Multiplicidad**: determinar si un atributo debe estar o no, y si posee un único valor o una lista de valores.
- **Ordenamiento**: especifica si el atributo determina alguna relación de orden dentro de la clase.
- **Capacidad de cambio**: permite definir atributos con valores constantes.
- **Modificadores**: un atributo puede ser de clase, derivado, volátil, transitorio.






# Diagrama de Clases: Atributos

**Visibilidad**

La encapsulamiento presenta tres ventajas básicas:

- Se protegen los datos de accesos indebidos
- El acoplamiento entre las clases se disminuye
- Favorece la modularidad y el mantenimiento

Los atributos de una clase no deberían ser manipulables directamente por el resto de objetos.

**Niveles de encapsulamiento**:

**(-) Privado** : es el más fuerte. Esta parte es totalmente invisible desde fuera de la clase (excepto para clases friends en terminología C++).
**(~) Package** : Sólo es visible dentro del mismo package.
**(#)** Los atributos/operaciones **protegidos** están visibles para las clases friends y para las clases derivadas de la original.
**(+)** Los atributos/operaciones **públicos** son visibles a otras clases (cuando se trata de atributos se está transgrediendo el principio de encapsulamiento).

# Diagrama de Clases: Atributos

### Multiplicidad

| | |
|---|---|
| 1 | El atributo debe tener un único valor. |
| 0..1 | El atributo puede o no tener un valor. |
| 0..* | El atributo puede tener varios valores o ninguno. |
| 1..* | El atributo puede tener varios valores, pero debe tener al menos uno |
| * | El atributo puede tener varios valores. |
| M..N | El atributo puede tener entre M y N valores. |

### Modificadores

- De **clase** o **estático**: el atributo se aparece subrayado. No es necesario contar con un objeto par a ejecutarlo.
- **Derivado**: es calculable a partir de otros atributos.
- **Transitorio**: tendrá valor sólo durante una porción de la ejecución.
- **Volátil**: no se persiste.



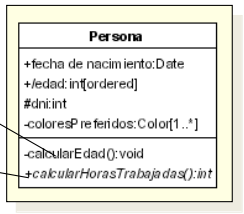
# Diagrama de Clases: Operaciones

Una operación es un servicio que una instancia de la clase puede realizar.

- **Tipo devuelto**: puede llegar a depender del lenguaje de programación a utilizar.
- **Parámetros**: además del tipo, puede especificarse si son In, Out o InOut.
- **Visibilidad**: está relacionado con el encapsulamiento.
- **Modificadores**: una operación puede ser de clase, abstracta, query o constructor.

La operación calcularEdad es privado y no devuelve nada.

El método público calcularHorasTrabajadas es abstracto, las subclases de la clase Persona deberá implementarlo para utilizarlo.



# Diagrama de Clases

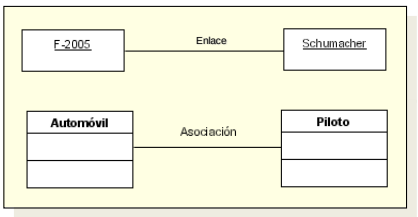
## Relaciones entre Clases

- Una asociación es una conexión estructural simple entre clases. Las instancias de las clases implicadas en una asociación estarán probablemente comunicándose en el momento de ejecución.
- Los enlaces entre de objetos pueden representarse entre las respectivas clases
- Formas de relación entre clases:
  - Asociación y Agregación (vista como un caso particular de asociación)
  - Generalización/Especialización



# Diagrama de Clases: Asociación

- La asociación expresa una conexión bidireccional entre objetos.
- Una asociación es una abstracción de la relación existente en los enlaces entre los objetos.

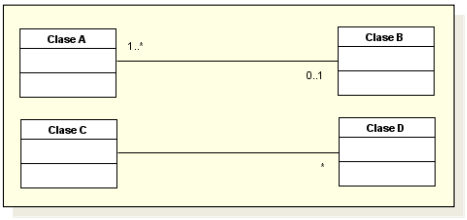


# Diagrama de Clases

## Relaciones entre Clases

**Multiplicidad**

| | |
|---|---|
| 1 | Un elemento relacionado. |
| 0..1 | Uno o ningún elemento relacionado. |
| 0..* | Varios elementos relacionados o ninguno. |
| 1..* | Varios elementos relacionados pero al menos uno. |
| * | Varios elementos relacionados. |
| M..N | Entre M y N elementos relacionados. |

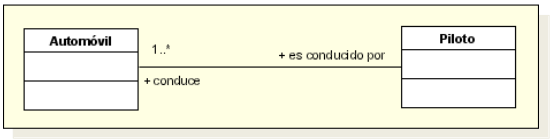





# Diagrama de Clases: Asociación

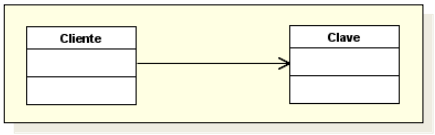
**Rol**

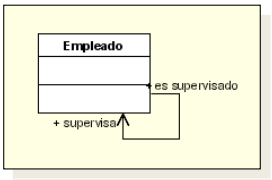
- Identificado como un nombre a los finales de la asociación, describe la semántica de la relación en el sentido indicado.
- Cada asociación tiene dos roles; cada rol es una dirección en la asociación.



# Diagrama de Clases: Asociación

- Se asume que una asociación es bidireccional, es decir que se puede navegar desde cualquiera de clases implicadas a la otra, pero es posible indicar que la navegación ocurrirá en una sola dirección.

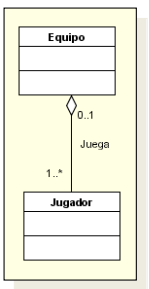


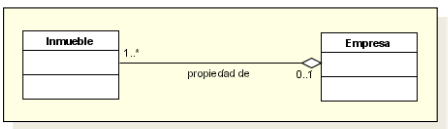






# Diagrama de Clases: Agregación

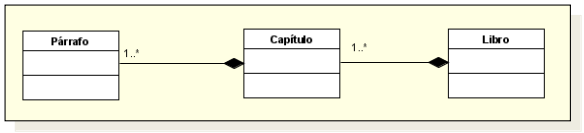
- Es una asociación especial, una relación del tipo “todo/parte” dentro de la cual una o más clases son partes de un conjunto.



# Diagrama de Clases: Composición

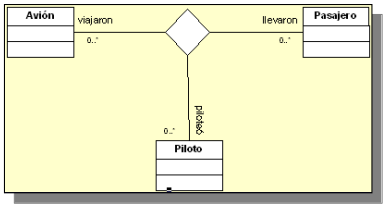
- La composición es una forma ‘fuerte’ de agregación.
  Se diferencian en:
  - En la composición tanto el todo como las partes tienen el mismo ciclo de vida.
  - Un objeto puede pertenecer solamente a una composición.



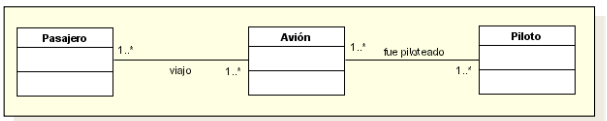



# Diagrama de Clases: Asociación n-arias

- Son asociaciones que se establecen entre más de dos clases
- Una clase puede aparecer varias veces desempeñando distintos roles.
- Las asociaciones n-arias se representan a través de rombo que se une con cada una de las clases.

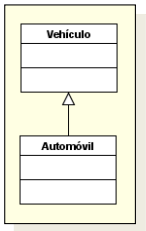


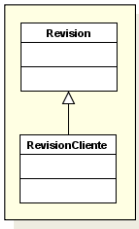

**La relaciones n-arias pueden ser usadas para impedir inconsistencias en el modelo.**

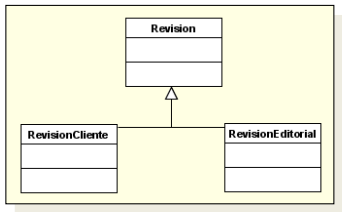


# Diagrama de Clases: Generalización

- Una generalización se refiere a una relación entre una clase general (superclase o padre) y una versión más específica de dicha clase (subclase o hija).



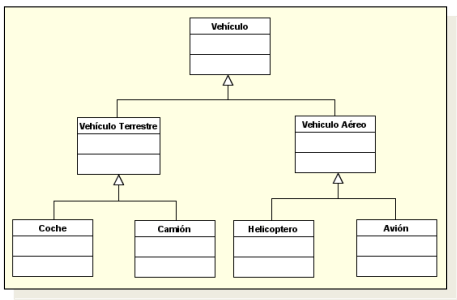



# Diagrama de Clases: Generalización

- Nombres usados: clase padre - clase hija. Otros nombres: superclase - subclase, clase base - clase derivada.
- Las subclases heredan propiedades de sus clases padre, es decir, atributos y operaciones (y asociaciones) de la clase padre están disponibles en sus clases hijas.
- La especialización es una técnica muy eficaz para la extensión y reutilización.

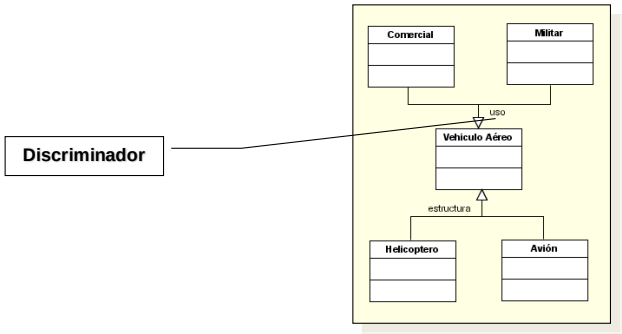
**Restricciones predefinidas en UML:**

- **Overlapping**
- **Disjoint**
- **Complete**
- **Incomplete**



# Diagrama de Clases: Generalización

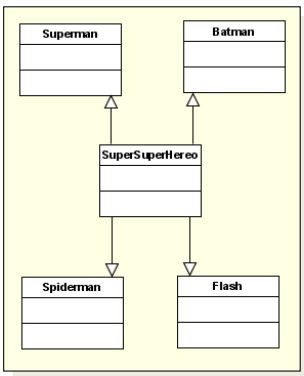
- Particionamiento del espacio de objetos → Clasificación Estática
- Particionamiento del espacio de estados de los objetos → Clasificación Dinámica
- En ambos casos se recomienda considerar generalizaciones/especializaciones disjuntas
- Usando discriminadores se pueden tener varias especializaciones de una misma clase padre





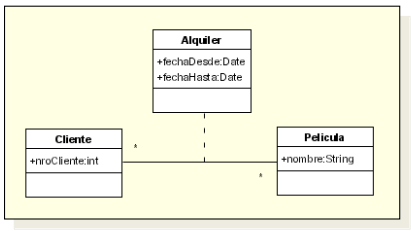

# Diagrama de Clases: Generalización

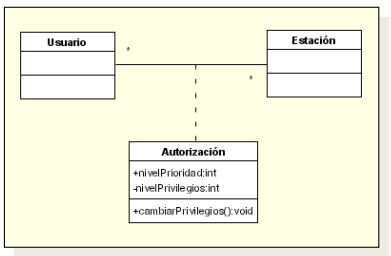
- La herencia múltiple debe manejarse con precaución. Algunos problemas son el conflicto de nombre y el conflicto de precedencia.
- Se recomienda un uso restringido y disciplinado de la herencia.
- Permite modelar jerarquías alternativas.



## Diagrama de Clases: Clase de asociación

- Es una asociación y una clase simultáneamente.
- Hay que tener en cuenta dónde se colocan los atributos.

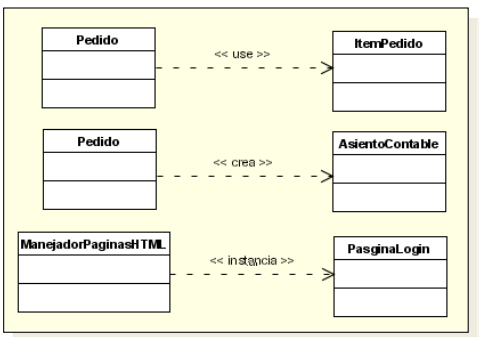





## Diagrama de Clases: Dependencia

- Una dependencia es una relación de “uso” en la que un cambio en uno de los términos -por ejemplo, una clase- puede afectar a otro (otra clase)



# Diagrama de Clases: Dependencia

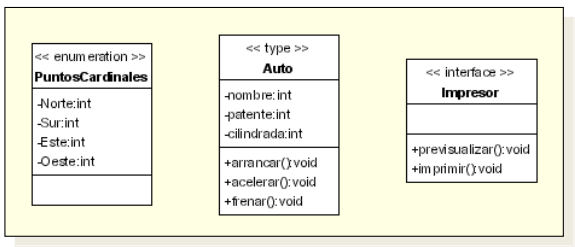
**Posibles dependencias entre clases**

- use: el funcionamiento del origen depende de la presencia del destino
- instantiate: el origen crea instancias del destino
- derive: el origen puede calcularse a partir del destino
- refine: el origen está un grado de abstracción más detallado.
- bind(): derivación genérica de una plantilla
- friend: visibilidad característica de C++



# Diagrama de Clases: Estereotipos

- Un estereotipo representa el principal mecanismo de extensión de UML. Ofrece una forma de extender una metaclase, creando un nuevo elemento de metamodelo.



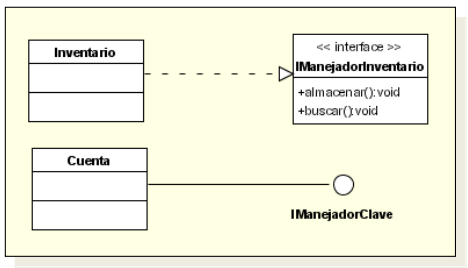
# Diagrama de Clases: Interfaces

- Una interfaz es una colección de operaciones que representan servicios o frecidos por una clase o componente.
- Por definición, todas estas operaciones tendrán una visibilidad pública.
- La interfaz especifica algo similar a un contrato que la clase se compromete a respetar.
- La clase *realiza* (o suministra una *realización* de) una o varias interfaces.
- UML define dos tipos de interfaces: interfaz suministrada e interfaz requerida.



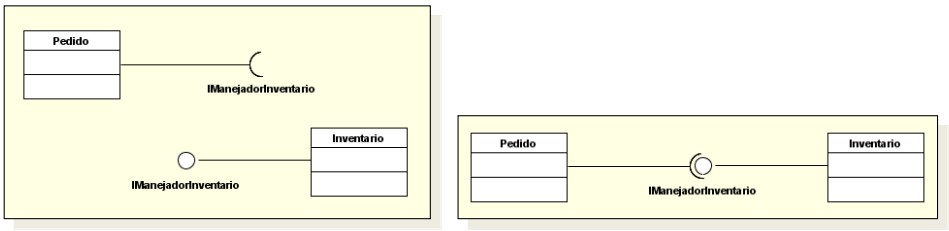
# Diagrama de Clases: Interfaces

- La interfaz suministrada es aquella que una clase efectivamente implementa.



# Diagrama de Clases: Interfaces

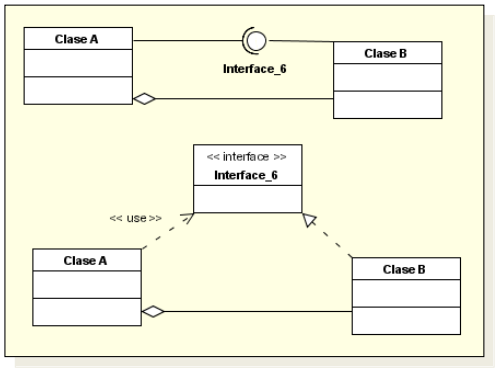
- Las interfaces requeridas son aquellas que necesita una clase para realizar su cometido. El símbolo utilizado para representarla es un semicírculo.






# Ejemplo

¿Cómo interpretaría lo siguiente?



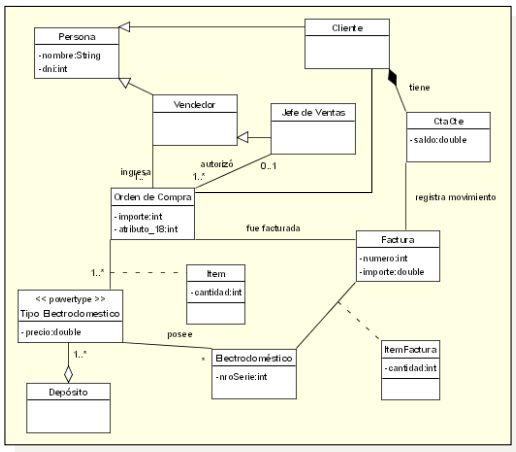
## Modelo de Dominio vs. Modelo de Diseño

- El diagrama de clases puede utilizarse con distintos fines en distintas etapas del proceso de desarrollo.
- Durante la etapa de análisis, el **modelo de dominio** es encargado de mostrar el conjunto de clases conceptuales del problema y las relaciones presentes entre sí.
- Durante la etapa de diseño, el **modelo de diseño** determina las futuras componentes de software (clases) y sus relaciones entre sí.



# Ejemplo

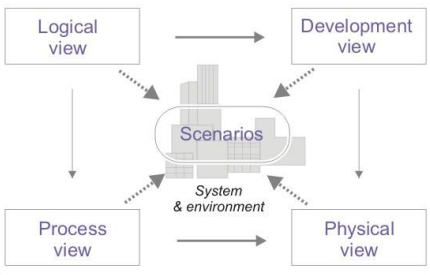
Un posible modelo de dominio para el caso del local de venta de electrodomésticos…



# Diagrama Entidad Relación

- No pertenece a UML
- Nacido para describir bases de datos relacionales (Chen).
- 2 conceptos: entidades y relaciones.
  - Entidades: conjuntos de individuos que poseen atributos.
  - Relaciones entre individuos especificando cardinalidad y opcionalidad.

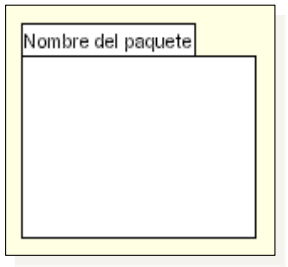



# Paquetes

# Paquetes

- Los paquetes ofrecen un mecanismo general para la organización de los modelos/subsistemas agrupando elementos de modelado.
- Se representan gráficamente como:



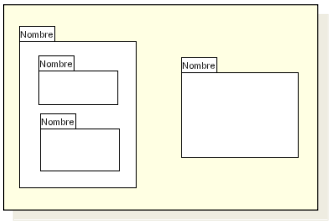



# Paquetes

- Cada paquete corresponde a un submodelo (subsistema) del modelo (sistema).
- Un paquete puede contener otros paquetes, sin límite de anidamiento pero cada elemento pertenece a (está definido en) sólo un paquete.
- Entre paquetes puede aparecer una relación de dependencia.
- Un paquete encapsula a la vez que agrupa.

# Paquetes

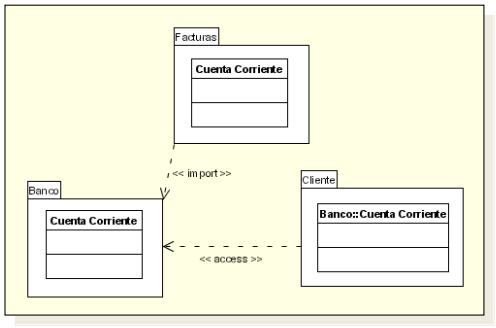
- Un paquete es un grupo de elementos del modelo que pueden a su vez anidarse.
- Son elementos auxiliares de organización y pueden contener cualquier elemento de modelado.
- Su utilidad final es ganar claridad a costa de perder detalles que luego se podrán obtener al abrir el paquete.
- Pueden ser utilizados con los distintos diagramas.





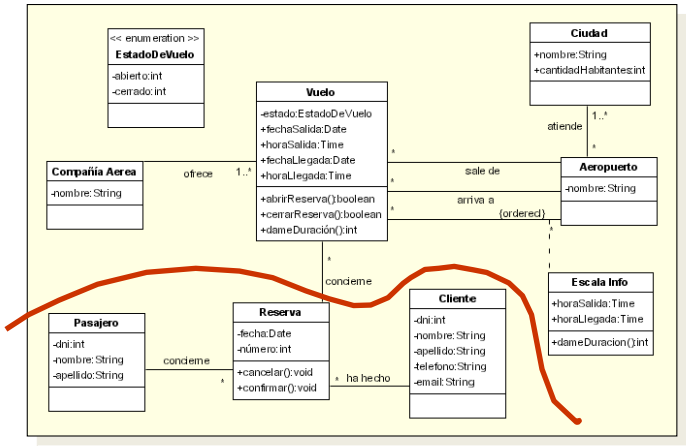

# Paquetes

- Se puede hacer referencia a una clase de otro paquete.
- Cada clase se puede definir con visibilidad pública o privada.
- Hay dos formas de referenciar elementos de otro paquete: **access** e **import**.



# Paquetes

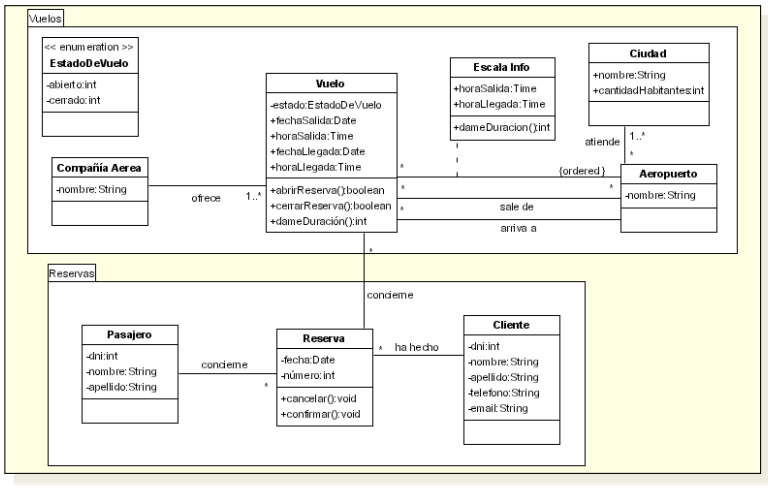
- Se debe buscar que los elementos que se encuentren dentro de un mismo paquete posean alta cohesión entre sí y que haya bajo acoplamiento entre paquetes.

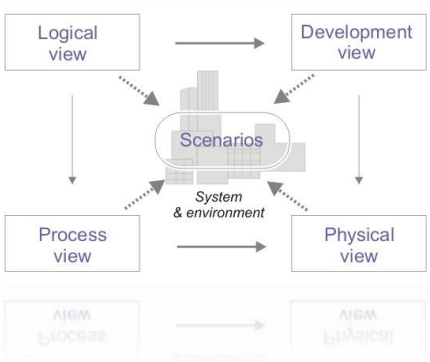





# Ejemplo

- Posible estructuración



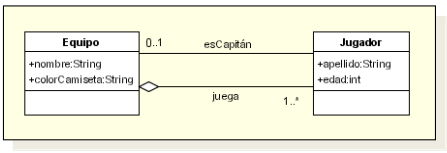


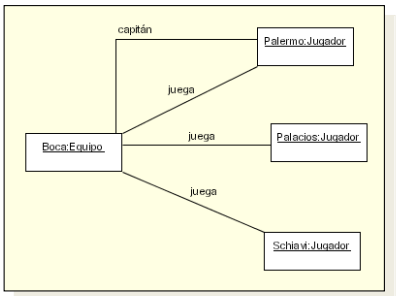

# Diagrama de Objetos

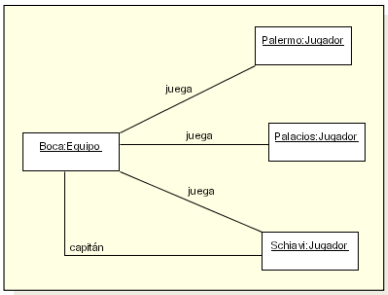
## Diagrama de Objetos

- Muestra la interacción directa entre los objetos.
- Es una representación del diagrama de clases en tiempo de ejecución, por tanto es una instancia posible del diagrama de clases.
- Son útiles para representar escenarios.

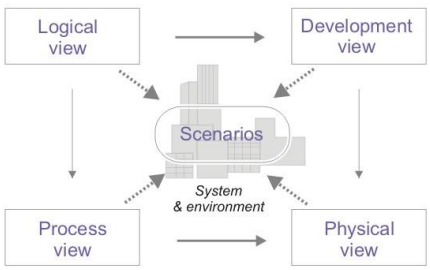
# Diagrama de Objetos



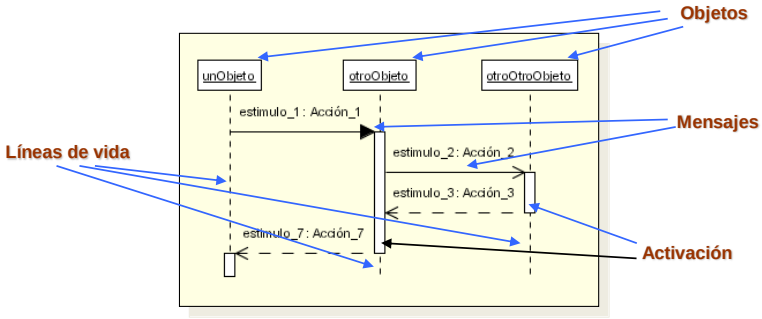



# Diagrama de Secuencia

# Diagrama de Secuencia

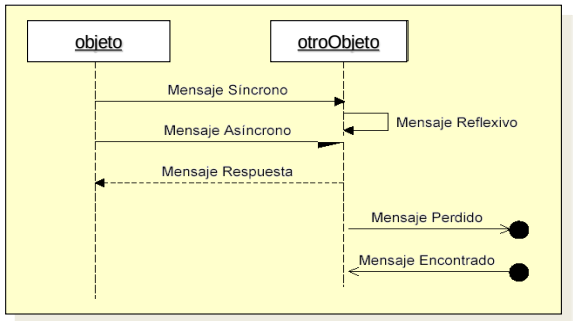
- Una *interacción* es un comportamiento que se centra en los intercambios observables de información entre los objetos.
- Una *línea de vida* representa la participación de un objeto dado en una determinada interacción.





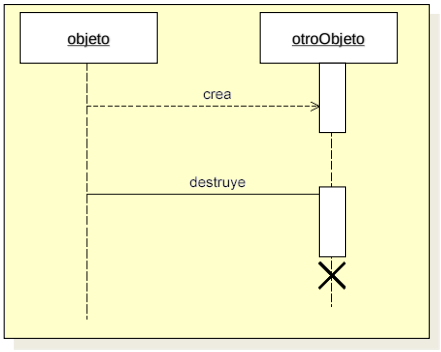

# Diagrama de Secuencia

- Los objetos comunican a través de mensajes entre líneas de vida.
- La notación para el mensaje siempre es una flecha, pero el tipo de flecha y la punta varía en función del tipo de mensaje:



# Diagrama de Secuencia

- También puede incluirse información referente a la creación y destrucción de objetos.

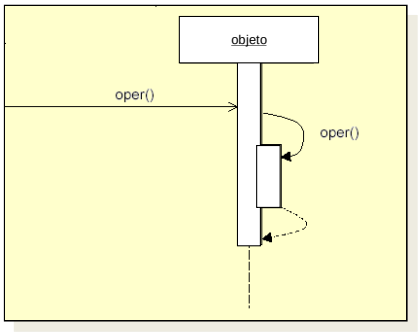





# Diagrama de Secuencia

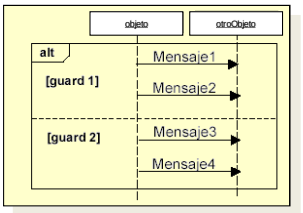
**Recursión**

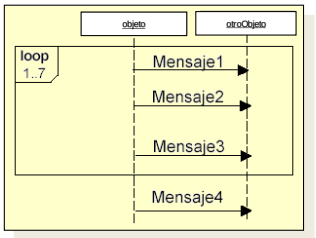
- La operación oper() se llama a sí misma.
  Habrá una condición en la operación que parará la recursión.
- Se muestra explícitamente la respuesta.



# Diagrama de Secuencia

**Ciclos y alternativas**



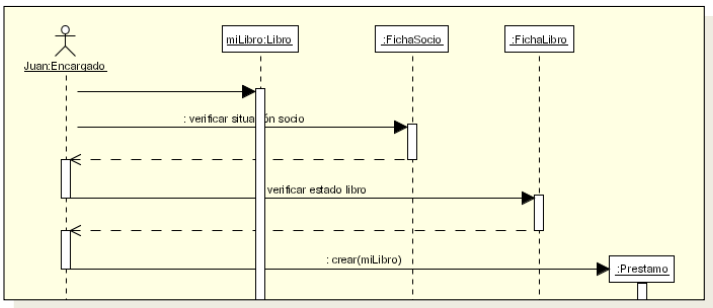




Muchas veces distintas alternativas pueden implicar distintos escenarios, que se deberían especificar con diagramas diferentes.



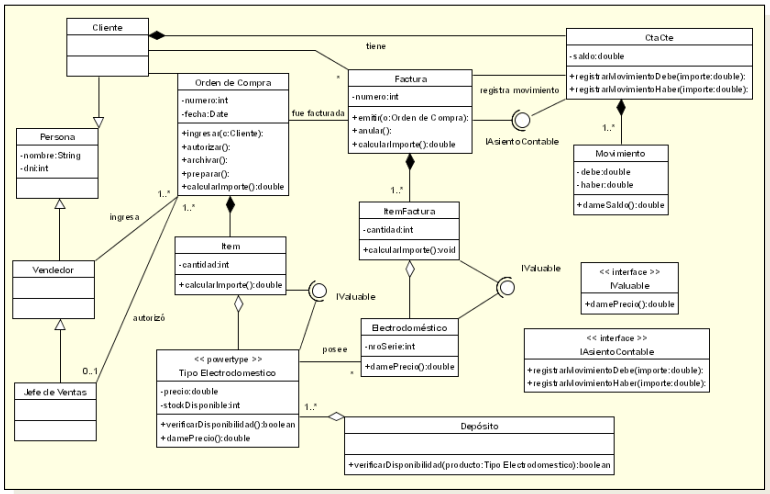
# Ejemplo

"… Supongamos que en una librería, un encargado debe registrar el préstamo de un libro… "



# Otro ejemplo

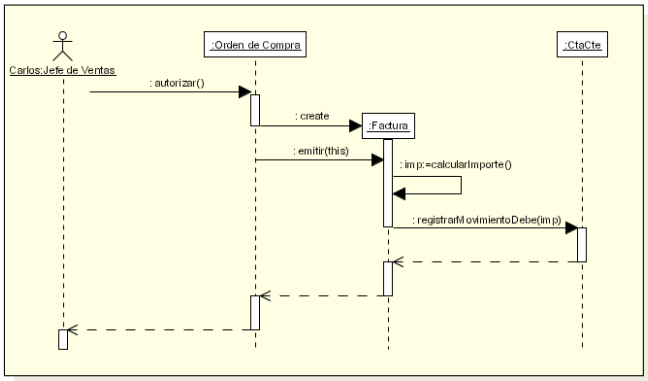
Un posible diseño para el caso del local de venta de electrodomésticos…





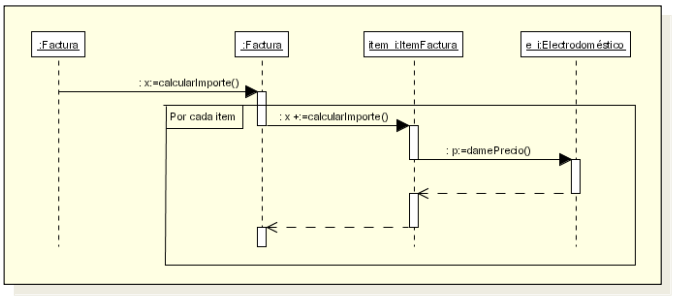

# Ejemplo

"… tras autorizar la orden de compra, se debe emitir la factura correspondiente, la cual debe asentarse en la cuenta corriente …"

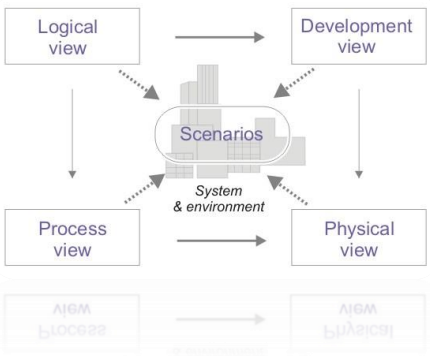


# Ejemplo

Veamos cómo se calcula el importe de la factura…






# Diagrama de Colaboración

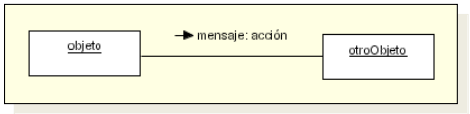
# Diagrama de Colaboración

- Son útiles en la fase exploratoria para identificar objetos.
- La distribución de los objetos en el diagrama permite obser var adecuadamente la interacción de un objeto con respect o de los demás.
- La estructura estática viene dada por los enlaces; la dinámi ca por el envío de mensajes por los enlaces.

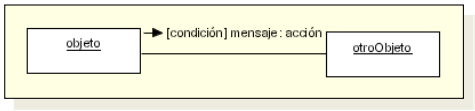


# Diagrama de Colaboración: Mensajes

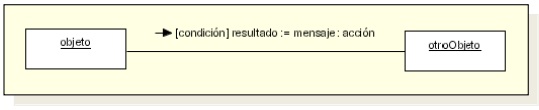
- Un mensaje desencadena una acción en el objeto destinatario.




- Un mensaje puede ser enviado de manera condicionada.

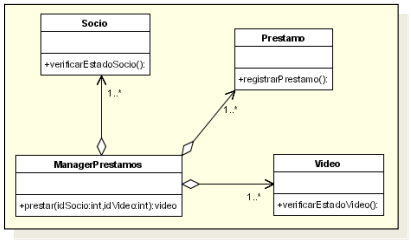


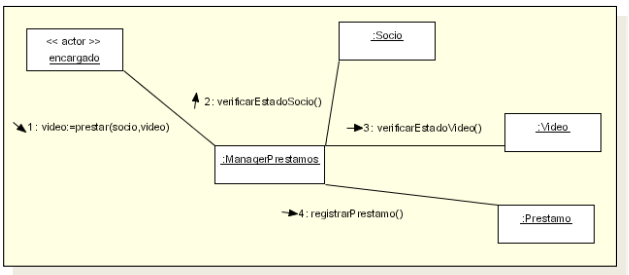

- Un mensaje puede devolver un resultado.



# Diagrama de Colaboración

- La cronología, está dada mediante la numeración de los mensajes.

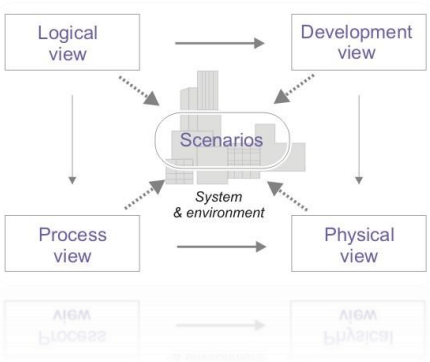





# Colaboración vs. Secuencia

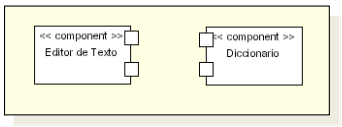
- Ambos diagramas poseen un poder expresivo similar.
- El diagrama de secuencia, otorga una mejor visión desde el punto de vista temporal o cronológico.
- El diagrama de colaboración, otorga una mejor visión desde el punto de vista espacial.
- En la etapa de análisis pueden ser utilizados para especificar escenarios.
- En la etapa de diseño, contribuyen con la definición de los métodos de las clases.




# Diagrama de Componentes

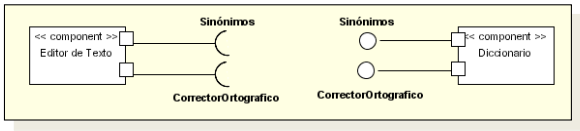
# Diagrama de Componentes

- Muestra las organizaciones y dependencias lógicas entre componente s software, sean éstos componentes de código fuente, binarios o ejecutables.
- Dado que los diagramas de componentes muestran los componentes *software* que constituyen una parte reusable, sus interfaces, y su interrelaciones, en muchos aspectos se puede considerar que un diagrama de componentes es un diagrama de clases a gran escala.



# Diagrama de Componentes

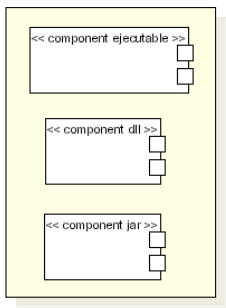
- Componentes bien diseñados tienen las siguientes cualidades:
  - Son independientes de manera directa de otros componentes.
  - Su dependencia es a través de interfaces bien definidas.
  - Pueden ser reemplazados por otro componentes que soportan las mismas interfaces.
- El diagrama de componentes muestra la dependencia entre componen tes a través de sus interfaces.
  - Cada componente “realiza” o soporta algunas interfaces y “usa” otras provistas por otr os componentes





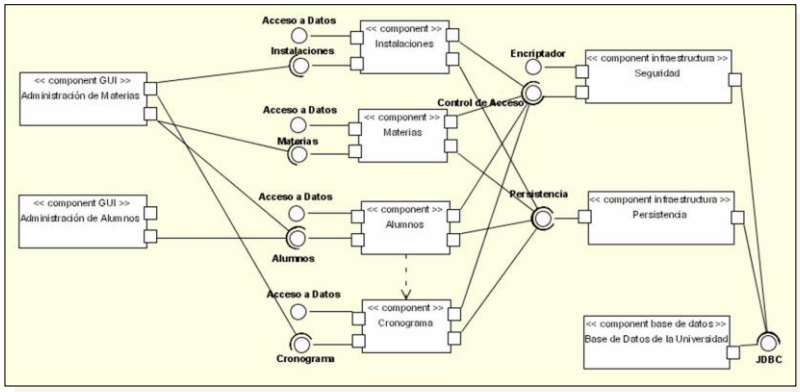

# Diagrama de Componentes

- Utilizando estereotipos es posible distinguir los distintos tipos de componentes de un sistema.



# Diagrama de Componentes

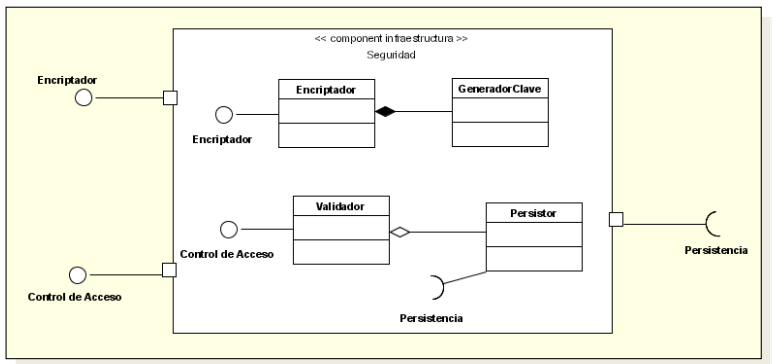
- Pueden ser utilizados para dar una visión arquitectónica del sistema

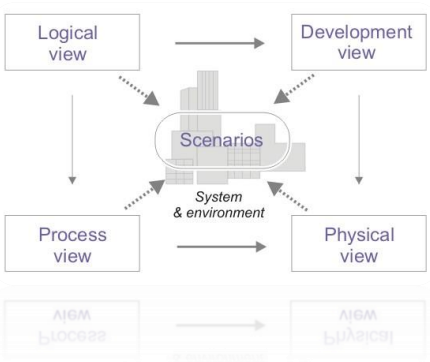





# Diagrama de Componentes

- Pueden combinarse con diagramas de clases para explotar su contenido…

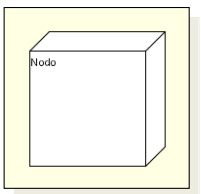





# Diagrama de Despliegue

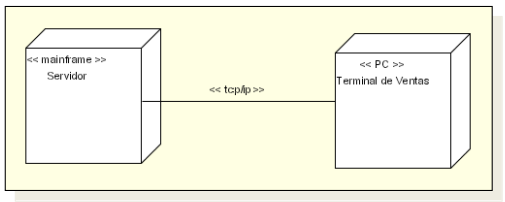
# Diagrama de Despliegue

- Los Diagramas de Despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos
- Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento.



# Diagrama de Despliegue

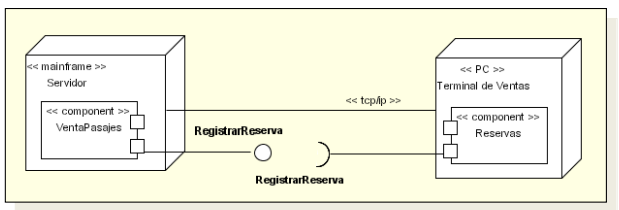
- Los estereotipos permiten precisar la naturaleza del equipo:
  - Dispositivos
  - Procesadores
  - Memoria
- Los nodos se interconectan mediante soportes bidireccionales que pue den a su vez estereotiparse.





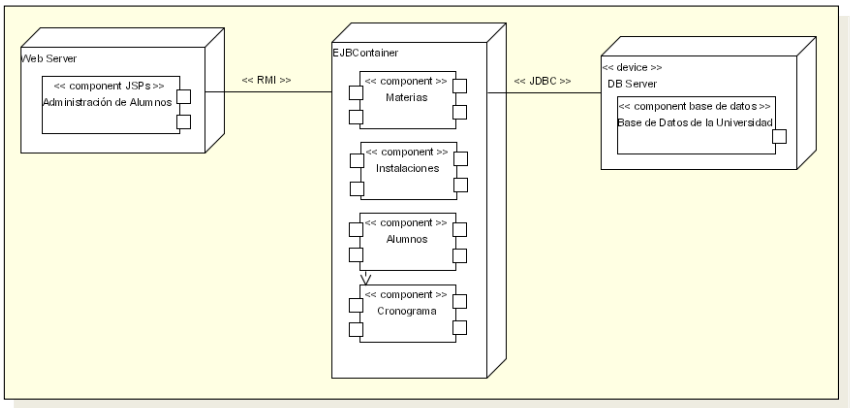

# Diagrama de Despliegue

- Mediante este diagrama es posible describir cómo se distribuirán espacialmente los componentes de software.



# Diagrama de Despliegue

- Los nodos pueden no sólo ser dispositivos, sino también otros componentes de software.






**Extensiones a UML**

# ¿Por qué extender UML?

- Aunque UML esté bien definido, hay situaciones en las cuales tiene que ser adaptado a dominios de problema específicos.
- Los mecanismos de extensión son utilizados para ampliar UML en relación a:
  - Adición de nuevos elementos modelos
  - Creación de nuevas propiedades
  - Especificar una nueva semántica
- Mecanismos de Extensión en UML:
  - Estereotipo
  - Restricción de Integridad
  - Valores Etiquetados, es un par (nombre propiedad, valor)



# Estereotipos

- Los estereotipos son utilizados para crear nuevos elementos del modelo que pueden ser usados en dominios específicos.
- Por ejemplo: modelando un sistema de control de elevador, podemos tener que representar algunas clases, estados, etc. como "hardware“ o "software“.
- Los estereotipos deberían ser siempre aplicados de un modo constante.

# Estereotipos

**Maneras de representar estereotipos**

- Ubicando su nombre sobre un elemento UML existente (si lo hubiera) encerrado entre << >>. Por ejemplo <<*formulario*>>.
- En forma de ícono.



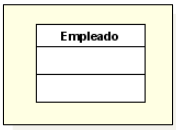

# Valores Etiquetados

- Son utilizados para definir propiedades adicionales para cualquier clase de elemento del modelo.
- También pueden aplicarse para estereotipos.
- Son mostrados como un par de etiqueta/valor donde la etiqueta representa la característica (o propiedad) y el valor representa el valor de la característica.

# Valores Etiquetados

- Pueden ser útiles para añadir información sobre
  - Generación de código
  - Control de versiones
  - Autoría y auditoría
  - etc.
- Se muestran como un texto encerrado entre llaves

{ *propiedad* = *valor*}




{ autor = 'Juan Perez' ,
versión = 1.0 }

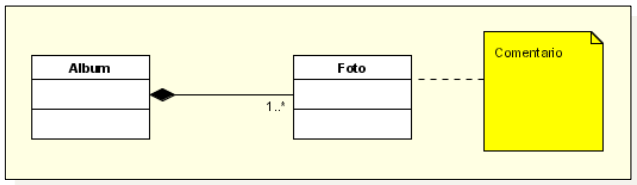


# Restricciones

- Las restricciones son usadas para ampliar la semántica de UML añadiendo nuevas reglas o modificando existentes.
- También pueden ser usadas para especificar condiciones que deben cumplirse en cualquier momento por todos o algún elemento del modelo (invariantes).
- Pueden ser representadas usando lenguaje natural o utilizando OCL (Object Contraint Language).

# Comentarios

- Ayudan a clarificar los diagramas
  - Por ejemplo, los comentarios pueden ser usados para explicar alg unas decisiones de diseño.
- Un comentario es mostrado dentro de un cuadro de nota, unido media nte un link al objeto asociado al comentario.
- Un cuadro de nota también puede contener una expresión OCL.



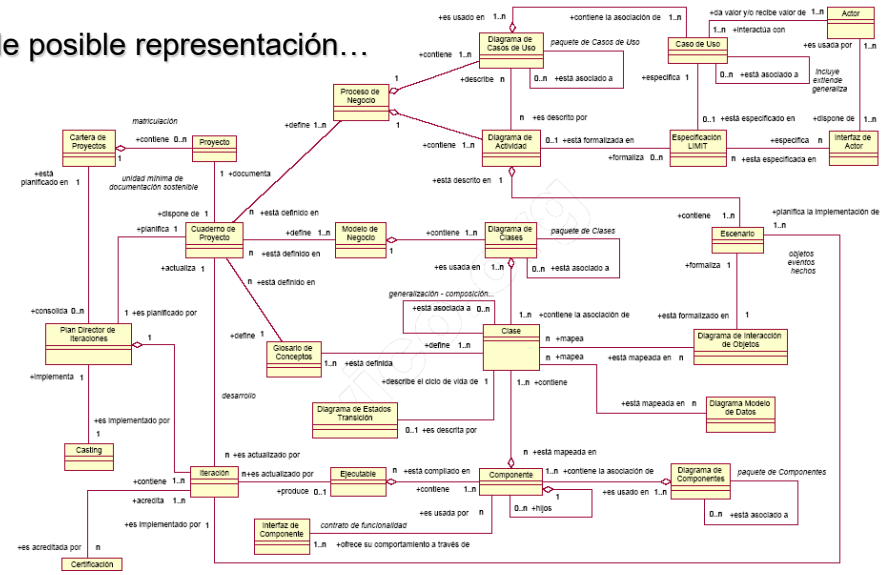



# El metamodelo UML

- Un metamodelo es un modelo que representa la estructura y la semántica de un conjunto particular de modelos.
- Un modelo UML es un caso instanciado del metamodelo UML.
- El metamodelo de UML
  - Describe los elementos de modelo de UML
  - Es definido usando un subconjunto de UML
  - Es organizado en forma de paquetes

# El metamodelo UML

Una posible posible representación…






# El metamodelo UML

- El metamodelo de UML está definido según los siguientes conceptos:
  - ***Sintaxis Abstracta***: El metamodelo de UML puede describirse utilizando diagramas de clase UML.
  - ***Restricciones***: Son usadas para expresar restricciones sobre los elementos de los modelos. Por ejemplo: una clase no puede tener dos nombres.
  - ***Semántica***: describe la utilización lenguaje natural, la semántica de los elementos del modelo.

# UML Profiles

- Proporcionan un mecanismo de extensión para construir modelos UML para dominios particulares. Por ejemplo sistemas de tiempo real, desarrollo de Web, etc. …
- Un perfil consiste en un paquete que contiene uno o varios mecanismos de extensión relacionados (como estereotipos, valores etiquetados y restricciones) aplicados a los elementos del modelo de UML.
- Los perfiles no amplían el metamodelo UML.



# UML Profiles

**Un perfil UML es una especificación que realiza las siguientes tareas:**

- Identifica un subconjunto del metamodelo UML (que puede ser todo el metamodelo UML).
- Especifica estereotipos y/o valores etiquetados.
- Especifica reglas o restricciones más allá de aquellos que ya existen.
- Especifica la semántica expresada en la lengua natural.

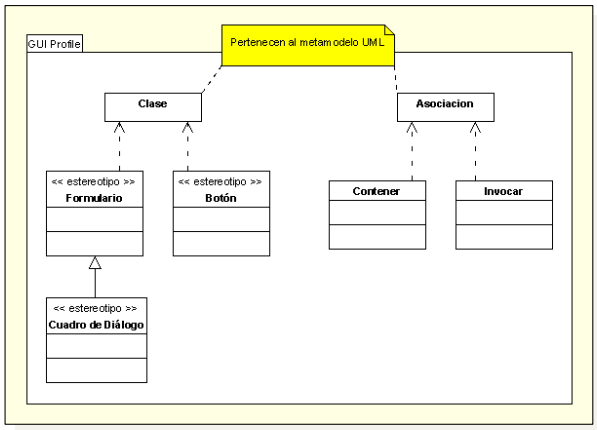
# Ejemplo de Perfil

- Se desea crear un perfil para representar los componentes básicos para construir GUIs (Graphical User Interfaces).
- Supongamos que nuestras GUI tendrán los siguientes componentes:
  - Formularios (que pueden ser cuadros de diálogo)
  - Botones
- Restricciones: (en la práctica deberían ser más precisas)
  - Un formulario puede invocar un cuadro de diálogo
  - Tanto un formulario como una cuadro de diálogo pueden contener botones



# Ejemplo de Perfil

- Paquete GUI Profile



# Ejemplo de Perfil

- Diagrama de objetos utilizando el perfil