



# INGENIERÍA DEL SOFTWARE I

## Tema 7

### *Lenguaje Unificado de Modelado - UML*

*Univ. Cantabria – Fac. de Ciencias*

*Francisco Ruiz*



### Objetivos del Tema

- Presentar el estándar UML 2.
- Conocer sus principales características.
- Conocer los principales constructores del lenguaje, así como los diversos tipos de diagramas y vistas de modelado que incluye.



## Contenido

---

- Introducción
- Objetivos
- Modelo Conceptual
- Elementos
  - Estructurales
  - De Comportamiento
  - De Agrupación
  - De Anotación
- Relaciones
- Diagramas
  - Estructurales
  - De Comportamiento
- Reglas
- Mecanismos Comunes
  - Especificaciones
  - Adornos
  - Divisiones comunes
  - Extensibilidad
- Conceptos de Modelado
  - Vistas Arquitecturales
  - Modelos
- OCL



## Bibliografía

---

- Básica
  - Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado. 2ª edición.
    - Caps. 2 y 7.
- Complementaria
  - Booch, Rumbaugh y Jacobson (2006): El Lenguaje Unificado de Modelado. 2ª edición.
    - Caps. 4, 5 y 6.
  - Rumbaugh, Jacobson y Booch (2007): El Lenguaje Unificado de Modelado. Manual de Referencia. 2ª edición.
    - Cap. 3.



## Bibliografía

---

- Estándares UML y OCL
  - [www.uml.org](http://www.uml.org)
- Webs
  - Múltiples enlaces e informaciones sobre UML:
    - [http://www.cetus-links.org/oo\\_uml.html](http://www.cetus-links.org/oo_uml.html)
  - Nuevas características del UML 2
    - [http://www.epidataconsulting.com/tikiwiki/tiki-read\\_article.php?articleId=31](http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=31)
  - Diagramas de UML 2 (con Visual-Paradigm)
    - <http://www.visual-paradigm.com/VPGallery/diagrams/index.html>



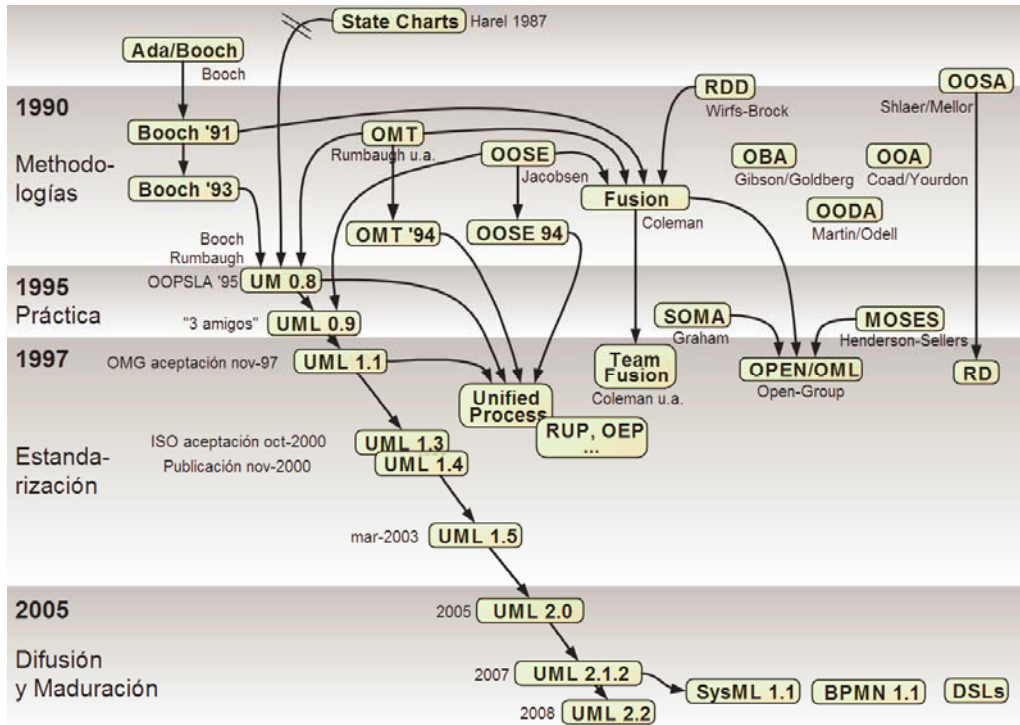
## Introducción

---

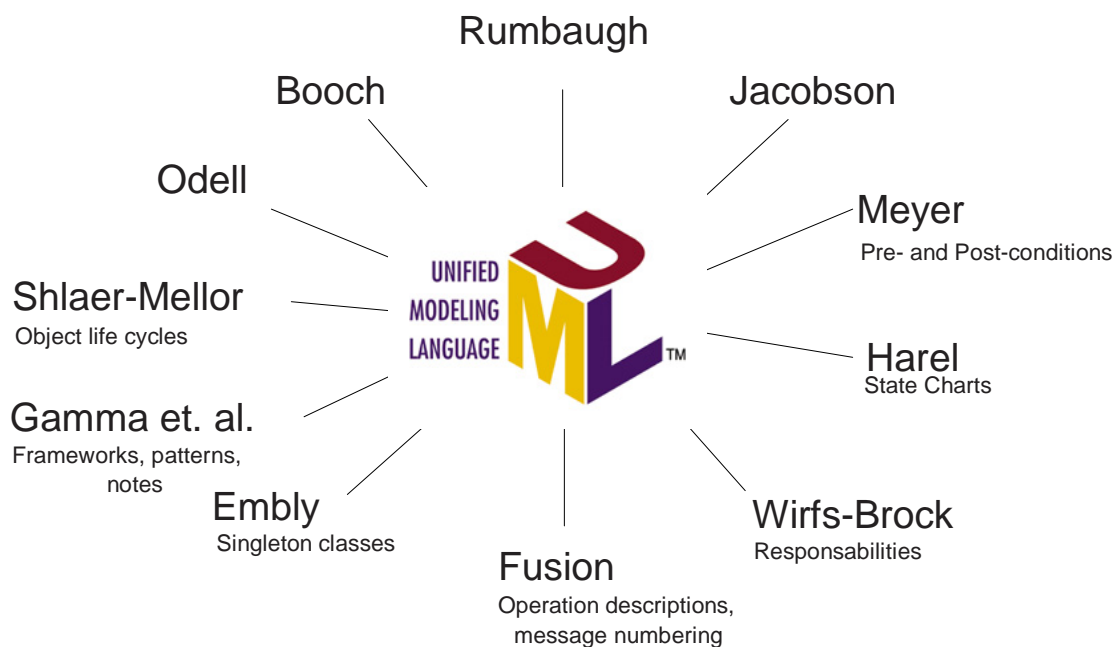
- **UML = Unified Modeling Language**
- Es un lenguaje de propósito general para el modelado orientado a objetos.
  - Impulsado por el **Object Management Group**
  - [www.omg.org](http://www.omg.org)
- Combina notaciones provenientes de varios tipos de modelado:
  - Orientado a Objetos
  - Datos
  - Componentes
  - Flujos de Trabajo (Workflows)



# Introducción



# Introducción





## Introducción

- **Lenguaje de Modelado**
  - “Lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema” (Booch, Jacobson y Rumbaugh).
  - = Notación + Reglas (Sintácticas, Semánticas)
  - estándar para escribir los **planos del software**.
  - **UML** ofrece vocabulario y reglas para crear y leer modelos bien formados.
  - **No ofrece un método.**



## Introducción

- **UML** es sólo un lenguaje
  - Independiente del Proceso
  - Pero su **uso óptimo** aconseja procesos dirigidos por casos de uso, centrados en la arquitectura, iterativos e incrementales
    - → Proceso Unificado de Desarrollo (RUP)
  - Cubre las diferentes **vistas** de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de software
  - **Vistas Software** (estáticas, dinámicas, etc..)
    - Son como los distintos planos de una casa (estructura-cimientos, albañilería, fontanería, electricidad, etc..)





## Introducción

- ¿**Cuando** puede utilizarse UML?
- En **sistemas** intensivos en **software** en diversos dominios:
  - Sistemas de información empresariales
  - Bancos y servicios financieros
  - Telecomunicaciones
  - Transporte
  - Defensa e industria aeroespacial
  - Comercio
  - Electrónica médica
  - Ámbito Científico
  - Servicios basados en Web
  - y también, **sistemas que no son software**



## Introducción

- **Inconvenientes** de UML
  - **No es una metodología.** Además de UML, hace falta una guía metodológica de cómo desarrollar software con OO.
  - **No cubre todas** las necesidades de especificación de un proyecto software.
    - No define los documentos textuales
  - Faltan **ejemplos** elaborados en la documentación.
  - Hay un cierto **monopolio** en torno a UML y OMG (libros, certificaciones, etc.).



## Introducción

---

- **Tendencias de Futuro**
  - **Extensiones** de UML
    - SysML - Systems Modeling Language (SysML, [www.sysml.org](http://www.sysml.org))
  - **Generación automática de código** a partir de modelos
    - **Model-Driven Engineering** (MDE)
  - Extendiendo UML mediante **perfiles**
  - Entornos avanzados basados en **metamodelado**
    - Eclipse Modeling Framework (EMF)
    - Microsoft Tools for Domain Specific Languages (DSLs)
  - Modelado y generación de código en **dominios específicos**
    - **Domain-Specific Modeling** (DSM, [www.dsmforum.org](http://www.dsmforum.org))



## Objetivos

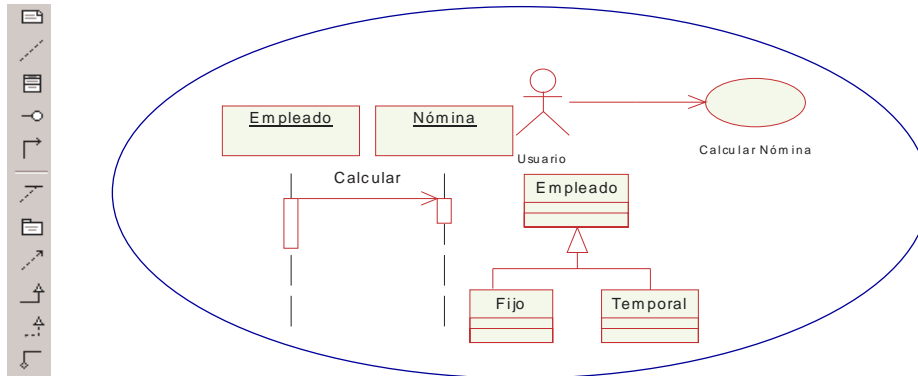
---

- **UML** es un **lenguaje de modelado visual** que sirve para
  - **visualizar**,
  - **especificar**,
  - **construir** y
  - **documentar**
- Sistemas
- Independientemente de la metodología de análisis y diseño



## Objetivos - Visualizar

- Detrás de cada símbolo en **UML** hay una semántica bien definida.
  - Basada en un metamodelo estándar MOF-compliant.
- Trasciende a lo que puede ser representado en un lenguaje de programación.
- Es más que un montón de símbolos gráficos (detrás de cada uno existe una semántica bien definida).
- Modelo explícito, que facilita la comunicación.



Francisco Ruiz - IS1

7.15



## Objetivos - Especificar

- Especificar es equivalente a **construir modelos precisos, no ambiguos y completos**.
- **UML** cubre la especificación del análisis, diseño e implementación de un sistema intensivo en software.
  - Resuelve muchos de los problemas de "Especificación" que había con el desarrollo convencional estructurado.

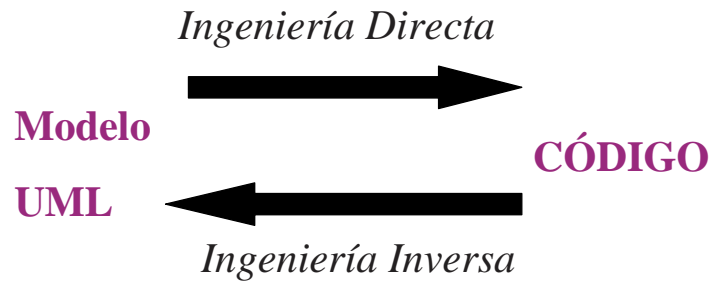
Francisco Ruiz - IS1

7.16



## Objetivos - Construir

- **UML** no es un lenguaje de programación visual, pero es posible establecer **correspondencias** entre un modelo UML y lenguajes de programación (Java, C++) y bases de datos (relacionales, OO).



## Objetivos - Documentar

- **UML** cubre toda la documentación de un sistema:
  - Arquitectura del sistema y sus detalles (diseño)
  - Expresar requisitos y pruebas
  - Modelar las actividades de planificación y gestión de versiones.

**Importancia en el mantenimiento**



## Modelo Conceptual

- Aprender a aplicar UML de modo eficaz comienza por conocer y comprender un **modelo conceptual del lenguaje**.
- El modelo conceptual de UML incluye tres tipos de elementos principales:
  - **Bloques de construcción** de UML
    - Elementos (estructurales, de comportamiento, de agrupación, de anotación), Relaciones, Diagramas
  - **Reglas** que dictan cómo pueden combinarse estos bloques
    - Sintácticas y semánticas
  - **Mecanismos comunes** que se aplican a lo largo del lenguaje
    - Especificaciones, adornos, divisiones comunes, mecanismos de extensibilidad.



## Elementos

- Los **elementos de UML** son abstracciones que constituyen los ciudadanos de primera clase en un modelo.
  - Son los bloques básicos de construcción orientada a objetos.
  - Se utilizan para construir **modelos bien formados**.
- Hay cuatro **tipos de elementos**:
  - Estructurales
  - De Comportamiento
  - De Agrupamiento.
  - De Anotación.



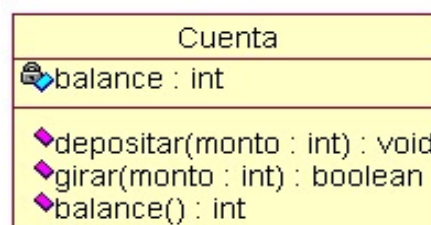
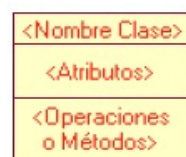
## Elementos - Estructurales

- Son los **nombres** (sujetos) de los modelos UML.
- En su mayoría, son las partes **estáticas** de un modelo.
- Reciben el nombre colectivo de **clasificadores**.
- En UML 2 existen los siguientes tipos:
  - Clase Clase activa
  - Interfaz Componente
  - Colaboración Artefacto
  - Caso de Uso Nodo
- Representan cosas conceptuales o lógicas (seis primeros) o elementos físicos (dos últimos).



## Elementos - Estructurales

- **Clase**
  - Una clase es una descripción de un **conjunto de objetos** que comparten los **mismos atributos, operaciones, relaciones y semántica**.
  - Las clases se pueden utilizar para capturar el **vocabulario del sistema** que se está desarrollando.
  - Una clase implementa una o más interfaces.





## Elementos - Estructurales

---

- **Clase**

- La **clase** es la unidad básica que encapsula toda la información de un Tipo de Objeto (un **objeto** es una instancia de una clase).
- **Responsabilidades** de una clase:
  - Contrato o una obligación de una clase.
  - Se pueden expresar en el extremo inferior de la caja que representa la clase.
- **Especializaciones** de clase: Actores, Señales, Utilidades.



## Elementos - Estructurales

---

- **Interfaz**

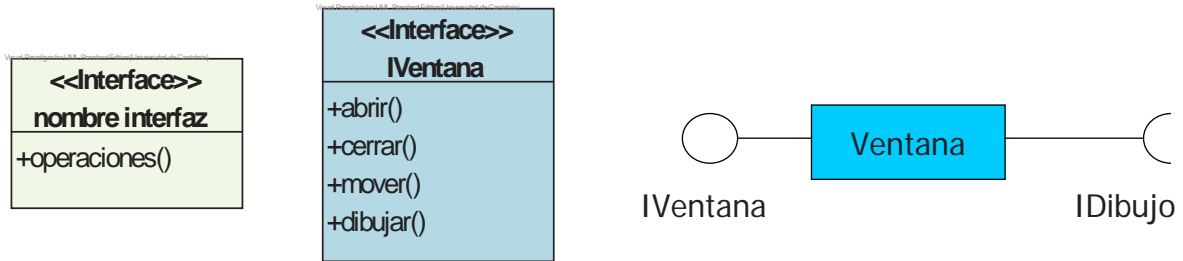
- **Colección de operaciones** que especifican un **servicio** de una clase o componente.
- Describen el comportamiento visible externamente de dichos elementos.
  - Pueden representar el comportamiento completo de la clase/componente o solo una parte.
- Una interfaz define un conjunto de especificaciones de operaciones (**signaturas**), pero no las implementaciones de dichas operaciones.



## Elementos – Estructurales

### • Interfaz

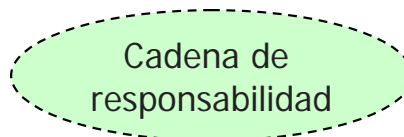
- Se declaran igual que las clases con <<interface>> encima del nombre.
- Una interfaz proporcionada por una clase también se puede representar como una circunferencia unida a la clase.



## Elementos - Estructurales

### • Colaboración

- Define una interacción.
- Es una sociedad de roles y otros elementos que colaboran para proporcionar un comportamiento mayor que la suma de sus comportamientos aislados.
- Tienen dimensión estructural y de comportamiento.
- Se emplean para modelar los patrones arquitecturales y patrones de diseño aplicados.

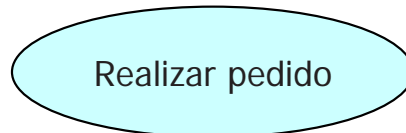




## Elementos - Estructurales

### • Caso de Uso

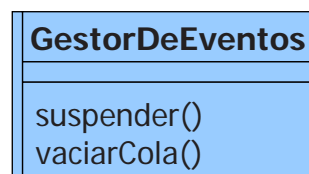
- Descripción de un conjunto de secuencias de acciones que ejecuta un sistema y que produce un resultado observable que es de interés para un actor particular.
- Se emplea para estructurar los aspectos de comportamiento.
- Un caso de uso es realizado por una colaboración.
- 



## Elementos - Estructurales

### • Clase Activa

- Tipo especial de clase cuyos objetos tienen uno o más procesos o hilos de ejecución =>
  - Pueden dar origen a actividades de control.
- Son iguales que las clases salvo que sus **objetos** pueden ser **concurrentes** con otros objetos de la misma clase activa.



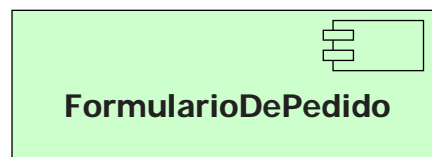
- **Especializaciones** de clase activa: Procesos, Hilos.



## Elementos - Estructurales

### • **Componente**

- Parte modular del diseño de un sistema que oculta su implementación tras un conjunto de interfaces externas.
- La implementación de un componente puede expresarse conectando partes y conectores.
  - Las partes pueden incluir componentes más pequeños.



## Elementos - Estructurales

### • **Artefacto**

- Parte física y reemplazable de un sistema que contiene información física (bits).
- Hay diferentes artefactos de despliegue:
  - código fuente, ejecutables, scripts, etc.



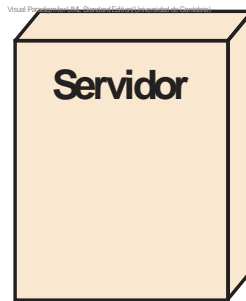
- **Especializaciones** de artefacto: Aplicaciones, Documentos, Archivos, Bibliotecas, Páginas, Tablas.



## Elementos - Estructurales

- **Nodo**

- Elemento físico que existe en tiempo de ejecución y representa un recurso computacional.
- En un nodo pueden residir un conjunto de artefactos.



## Elementos – De Comportamiento

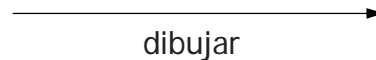
- Son las partes **dinámicas** de los modelos UML.
  - Equivalen a los **verbos** de un modelo.
  - Representan comportamiento en el tiempo y el espacio.
  - Suelen estar conectados semánticamente a elementos estructurales.
  - Hay tres tipos:
    - Interacción
    - Máquina de estados
    - Actividad
- Énfasis en
- conjunto de objetos que interactúan
  - ciclo de vida de un objeto
  - flujo entre pasos, sin mirar qué objeto ejecuta cada paso



## Elementos - De Comportamiento

### • Interacción

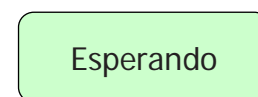
- Comportamiento que comprende un **conjunto de mensajes** intercambiados entre un **conjunto de objetos**, dentro de un contexto particular, para un propósito específico.
- Sirven para modelar el comportamiento de una sociedad de objetos, o una operación individual.
- Involucran a:
  - **Mensajes**,
  - Acciones y
  - Enlaces (conexiones entre objetos).



## Elementos - De Comportamiento

### • Máquina de Estados

- Comportamiento que especifica las **secuencias de estados** por las que pasa un objeto o una interacción durante su vida, en respuesta a **eventos**, junto con sus **reacciones** a dichos eventos.
- Sirven para especificar el comportamiento de una clase individual o una colaboración de clases.
- Involucran a:
  - **Estados**,
  - Transiciones (flujo de un estado a otro),
  - Eventos (que disparan una transición) y
  - Actividades (respuesta a una transición)

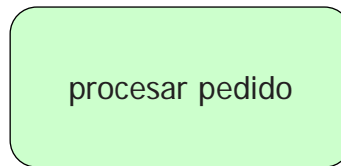
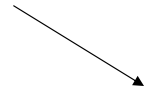




## Elementos - De Comportamiento

- **Actividad**

- Comportamiento que especifica la **secuencia de pasos** que ejecuta un proceso computacional.
- Una **acción** es un paso de una actividad.



- El símbolo es el mismo que para estado. Se distinguen por el contexto.



## Elementos – De Agrupación

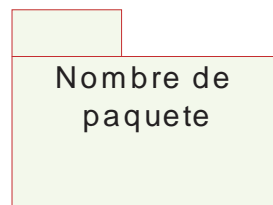
- Son las partes **organizativas** de los modelos UML.
- Son las “cajas” en las que puede dividirse un modelo.
- Hay un tipo principal:
  - Paquete
- Y varias especializaciones (subtipos de paquetes):
  - Frameworks, Modelos, Subsistemas.



## Elementos – De Agrupación

- **Paquete**

- Mecanismo de propósito general para **organizar el propio diseño** (las clases organizan construcciones de implementación).
- Un paquete puede incluir elementos estructurales, de comportamiento y otros paquetes.
- Un paquete es puramente conceptual (sólo existe en tiempo de desarrollo).



## Elementos – De Agrupación

- **Paquete**

- Es recomendable que el contenido sea una colección de elementos UML relacionados de forma lógica.
- Se pueden utilizar en cualquier tipo de diagrama UML.
- Un paquete puede contener otros paquetes, sin límite de anidamiento, pero cada elemento pertenece a (está definido en) sólo un paquete.
- La visibilidad de los elementos incluidos en un paquete puede controlarse para que algunos sean visibles fuera del paquete mientras que otros permanezcan ocultos.



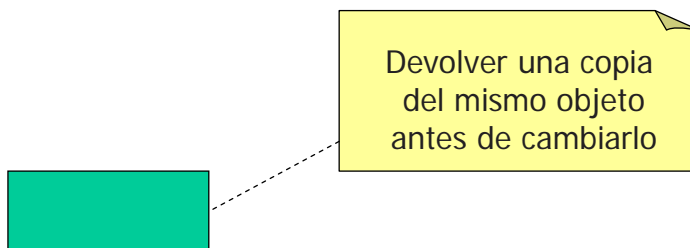
## Elementos – De Anotación

- Son las partes **explicativas** de los modelos UML.
- Son comentarios que se añaden para describir, clarificar y hacer observaciones.
- Hay un tipo principal:
  - Nota
- Y una especialización (subtipo de nota):
  - Requisito.



## Elementos – De Anotación

- **Nota**
  - Símbolo para mostrar restricciones y comentarios asociados a un elemento o colección de elementos.
  - Se usan para aquello que se muestra mejor en forma textual (comentario) o formal (restricción).






## Relaciones

- Una **relación** es una conexión entre elementos estructurales.
- Existen cuatro tipos de **relaciones** en UML2:
  - **Dependencia**
  - **Asociación**
  - **Generalización**
  - **Realización**
- Hay dos tipos especiales de asociación:
  - **Agregación**
  - **Composición**



## Relaciones

- **Dependencia**
    - Relación semántica en la cual un cambio a un elemento (independiente) puede afectar a la semántica del otro elemento (dependiente).
- 
- ```
classDiagram
    class Aplicacion
    class Ventana
    Aplicacion ..> Ventana
```
- Caso frecuente: **uso entre clases**
    - Una clase utiliza a otra como argumento en la signatura de una operación.
  - **Especializaciones**: Refinamiento (refine), Traza (trace), Inclusión (include), Extensión (extend), ...



## Relaciones

### • Asociación

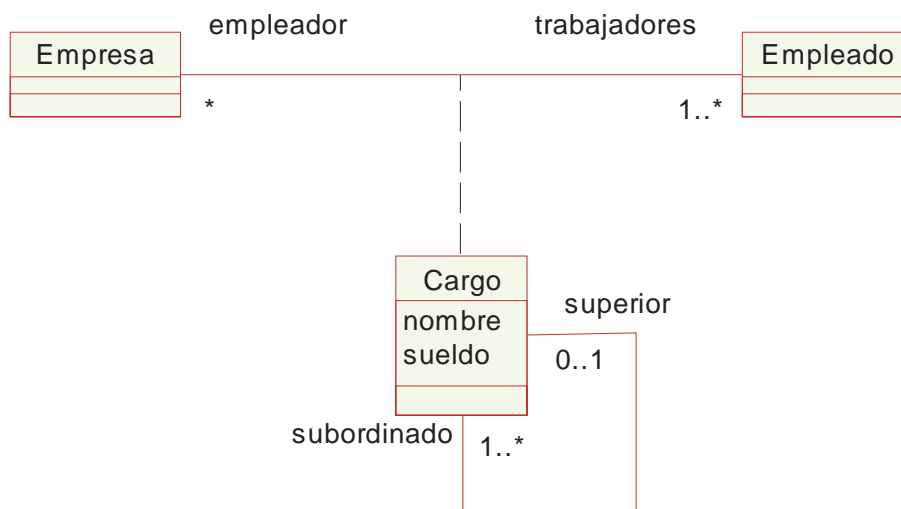
- Relación estructural entre las clases que describe un conjunto de enlaces (conexiones entre objetos que son instancias de las clases).
  - En general es simétrica.
  - Tiene un nombre, que la describe (verbo, con dirección de lectura).
  - Puede tener un rol que describe el papel específico que una clase juega en una asociación.
  - Tiene multiplicidad para clase participante.



## Relaciones

### • Asociación

- Ejemplo de **clase de asociación** y asociación **reflexiva**.





## Relaciones

### • Agregación y Composición

- Modelar objetos complejos en base a **relaciones todo – parte**.
- **Agregación**
  - Relación dinámica: el tiempo de vida del objeto incluido es independiente del objeto agregado.
  - El objeto agregado utiliza al incluido para su funcionamiento.
    - SIMILAR parámetro pasado “por referencia”.
- **Composición**
  - Relación estática: el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del objeto compuesto.
  - El objeto compuesto se construye a partir del objeto incluido.
    - SIMILAR parámetro pasado “por valor”.



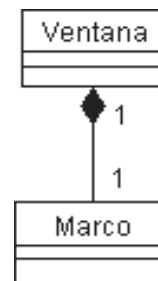
## Relaciones

### • Agregación y Composición

- Ejemplos.



**Agregación**  
(por referencia)



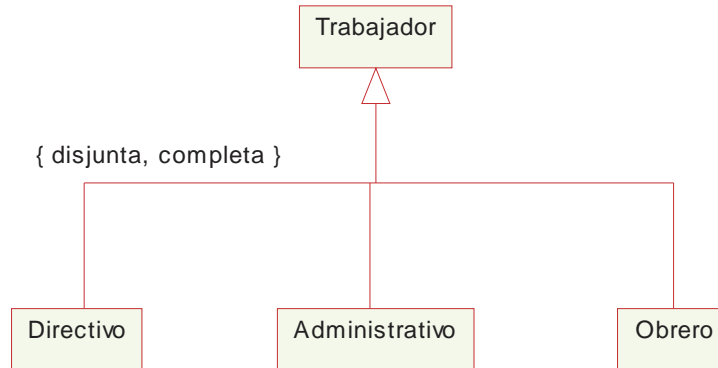
**Composición**  
(por valor)



# Relaciones

## • Generalización

- Relación de especialización/generalización en la que el elemento especializado (hijo) se basa en la especificación del elemento generalizado (padre).
- Las subclases (hijos) comparten la estructura y el comportamiento de la superclase (padre).

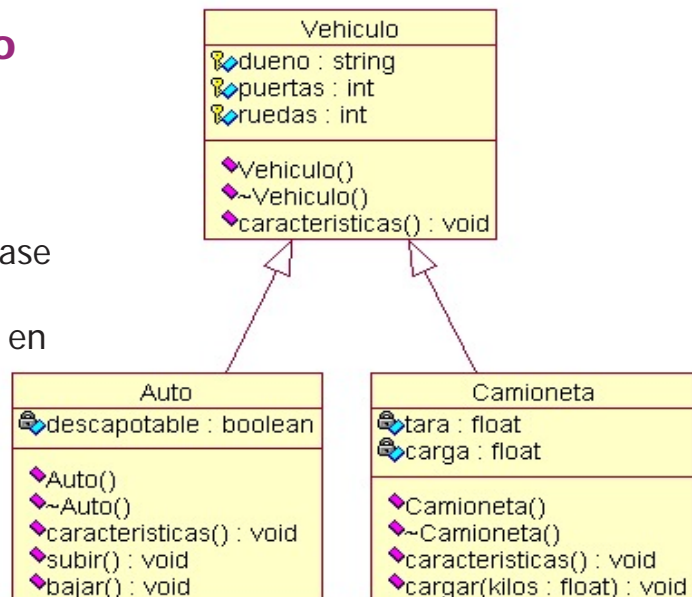


# Relaciones

## • Generalización

- Una generalización da lugar al **polimorfismo** entre clases de una jerarquía de generalizaciones:

- Un objeto de una subclase puede sustituir a un objeto de la superclase en cualquier contexto. Lo inverso no es cierto.

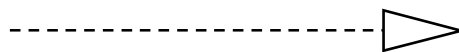




# Relaciones

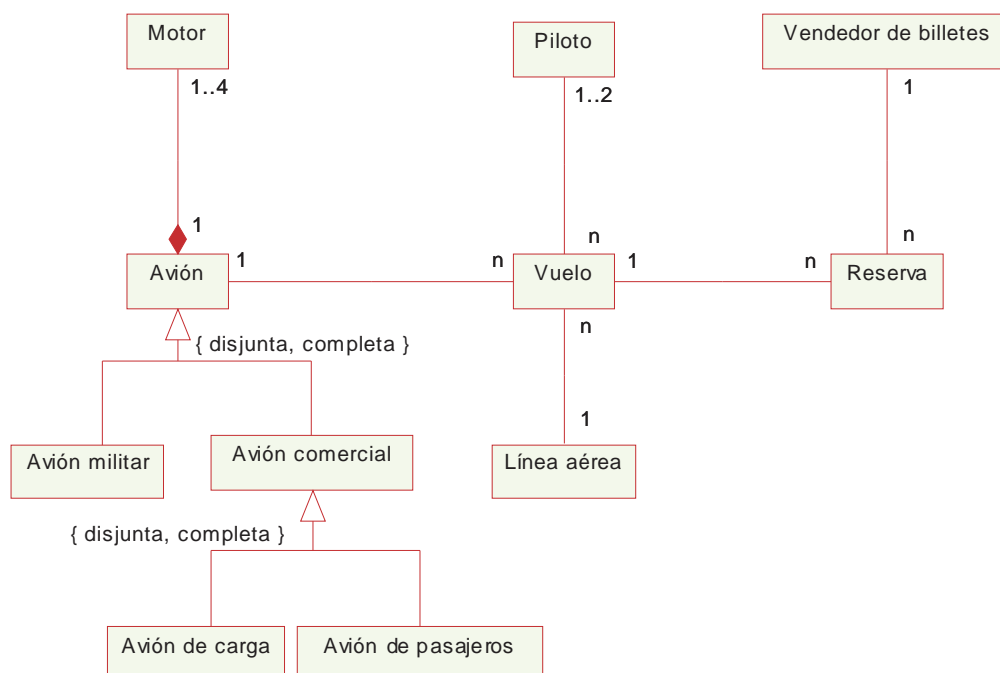
## • Realización

- Relación semántica entre clasificadores, donde un clasificador especifica un **contrato** que otro clasificador garantiza que cumplirá.
- Se pueden encontrar en dos casos:
  - Clases o componentes - realizan - interfaces.
  - Colaboraciones - realizan -> casos de uso.



# Relaciones

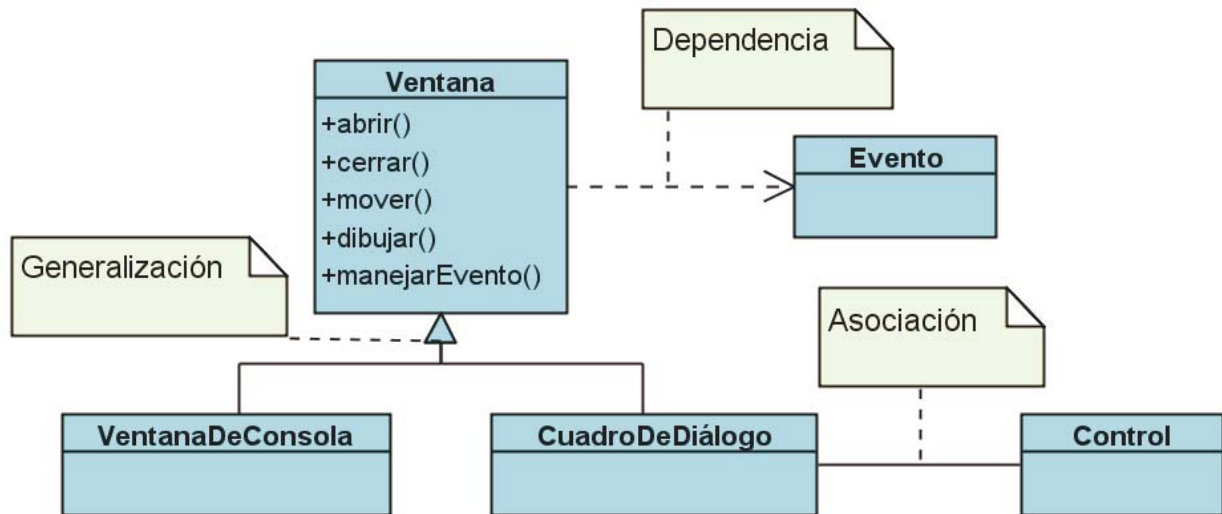
## • Jerarquías de Relaciones





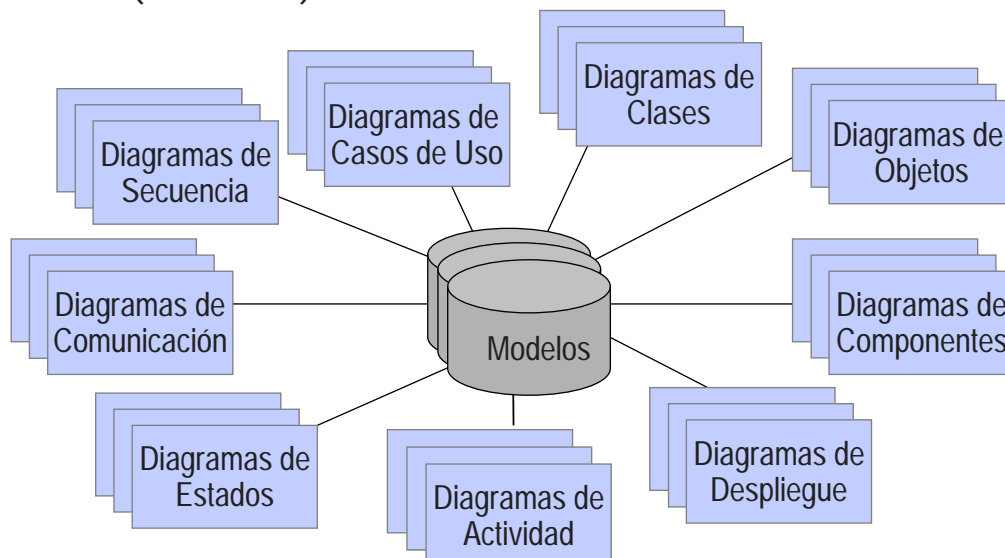
## Relaciones

- **Ejemplo** con generalizaciones, asociación y dependencia.



## Diagramas

- Un **diagrama** es la **representación gráfica** de un conjunto de elementos de modelado (parte de un modelo).
  - A menudo se dibujan como un grafo conexo de nodos (elementos) y arcos (relaciones).









## Diagramas

- Sirven para visualizar un sistema desde **diferentes perspectivas**.
- Un diagrama es una **proyección gráfica de un sistema**.
  - Vista resumida de los elementos que constituyen el sistema.
- El mismo elemento puede aparecer en varios diagramas.
- En teoría, un diagrama puede contener cualquier combinación de elementos y relaciones, sin embargo en la práctica solo un pequeño número de combinaciones tiene sentido:
  - Surgen así los tipos de diagramas de UML 2.



## Diagramas

- **UML 2** incluye **13 Tipos de Diagramas**:
- **Estructurales**  
*(ESTÁTICA)*
  - Clases
  - Objetos
  - Componentes
  - Despliegue
  - Paquetes
  - Estructura Compuesta 
- **De Comportamiento**  
*(DINÁMICA)*
  - Casos de Uso
  - Estados
  - Actividades
  - Interacción
    - Secuencia
    - Comunicación
    - Tiempos 
    - Revisión de Interacciones 

 nuevo en UML 2.0



# Diagramas

## Estructurales

: elementos de especificación, sin factores temporales

- Clases
- Componentes
- Despliegue
- Objetos
- *Estructura Compuesta*
- *Paquetes*

## Comportamiento

: comportamiento de un sistema /proceso de negocio

- Actividades
- Estados
- Casos de Uso
- *Interacción*


## Interacción

: comportamiento con énfasis en la interacción entre objetos

- Comunicación (colaboración)
- Secuencia
- *Revisión de Interacciones*
- *Tiempos*



# Diagramas - Estructurales

- Los diagramas estructurales de UML 2 sirven para visualizar, especificar, construir y documentar los **aspectos estáticos** de un sistema.
- Ser organizan en base a los principales grupos de elementos que aparecen al modelar. 

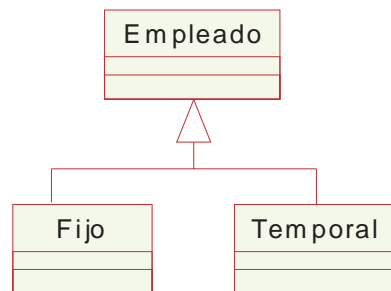
| Tipo de Diagrama     | Elementos Centrales                        |
|----------------------|--------------------------------------------|
| Clases               | Clases, Interfaces, Colaboraciones         |
| Componentes          | Componentes                                |
| Estructura Compuesta | Estructura Interna de Clase o Colaboración |
| Objetos              | Objetos                                    |
| Paquetes             | Paquetes                                   |
| Despliegue           | Nodos, Artefactos                          |



## Diagramas - Estructurales

### • De Clases:

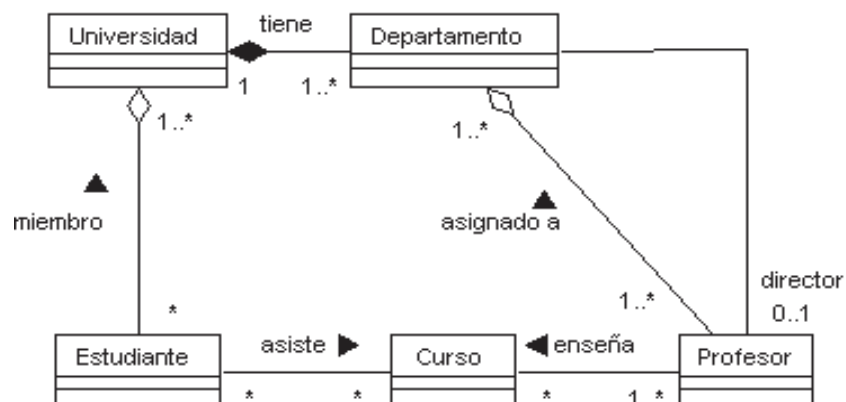
- Muestran un conjunto de clases, interfaces y colaboraciones, así como sus relaciones.
- Son los diagramas más comunes en el análisis y diseño del sistema:
  - Explorar conceptos del dominio (**Modelo de Dominio**).
  - Analizar requisitos (**Modelo de Análisis / Conceptual**).
  - Describir el diseño detallado de un software OO (**Modelo de Diseño**).
- Abarcan la vista de diseño estática de un sistema.
  - Con clases activas cubren la vista de procesos estática.



## Diagramas - Estructurales

### • De Clases:

- Principalmente, un diagrama de clases contiene:
  - **Clases** (con atributos, operaciones y visibilidad).
  - **Relaciones**: Dependencia, Generalización, Asociación, Agregación y Composición.

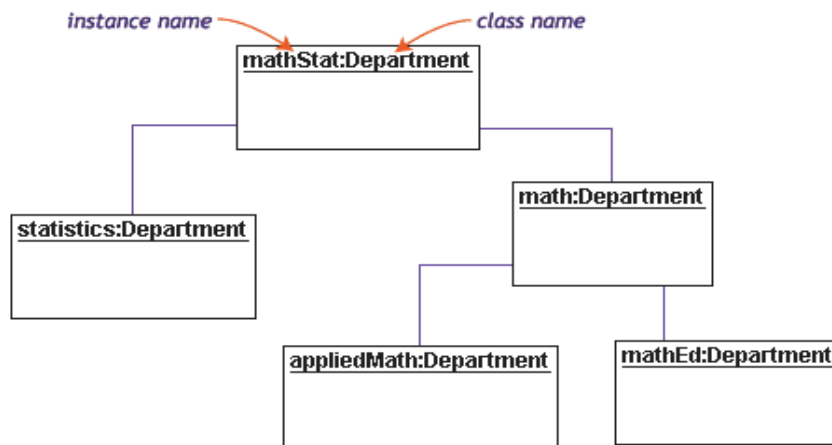




## Diagramas - Estructurales

- **De Objetos:**

- Muestra un conjunto de objetos y sus relaciones.
- Representan instantáneas estáticas de instancias de los elementos existentes en los diagramas de clases.
- Sirven para describir estructuras de datos.



## Diagramas - Estructurales

- **De Componentes:**

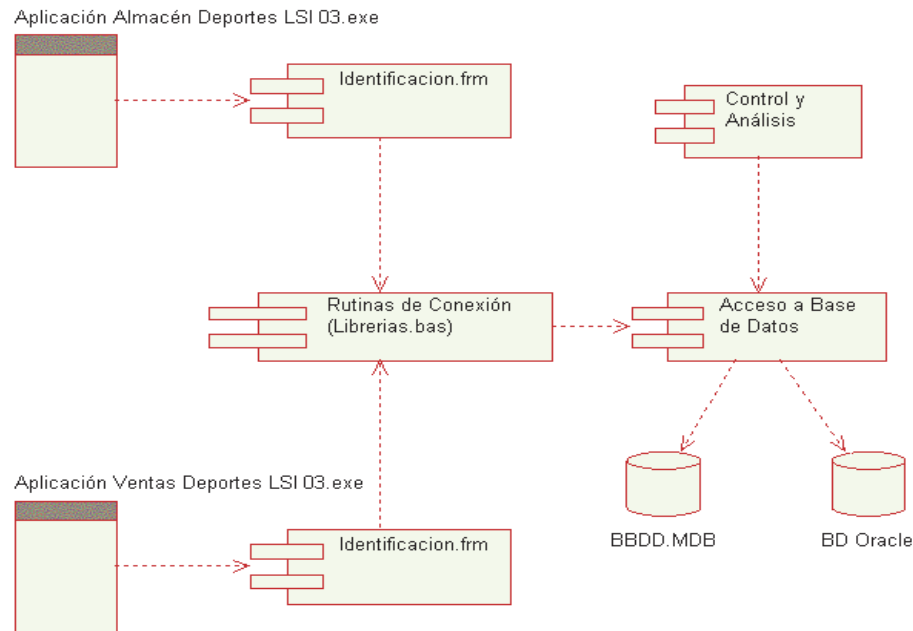
- Representan la encapsulación de un componente con sus interfaces, puertos y estructura interna (formada por otros componentes anidados y conectores).
- Describen los elementos físicos del sistema y sus relaciones.
- Muestran la **organización y las dependencias** entre un conjunto de **componentes**.
- Cubren la vista de implementación estática del diseño de un sistema.
- Muestran las opciones de realización, incluyendo código fuente, binario y ejecutable, scripts, DLLs, tablas de datos, etc.



## Diagramas - Estructurales

- **De Componentes:**

- Ejemplo.



Francisco Ruiz - IS1

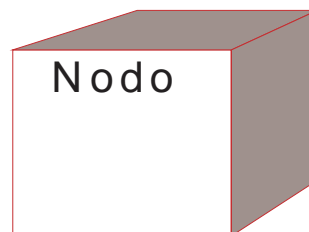
7.61



## Diagramas - Estructurales

- **De Despliegue:** [distribución]

- Muestran un conjunto de **nodos** y sus relaciones.
  - Describen la vista de despliegue estática de una arquitectura.
  - Cada nodo (hardware) suele albergar uno o más componentes.
  - Muestran el hardware, el software y el middleware usado para conectar las máquinas.



- Algunos autores identifican una variante llamada **diagramas de artefactos**.

Francisco Ruiz - IS1

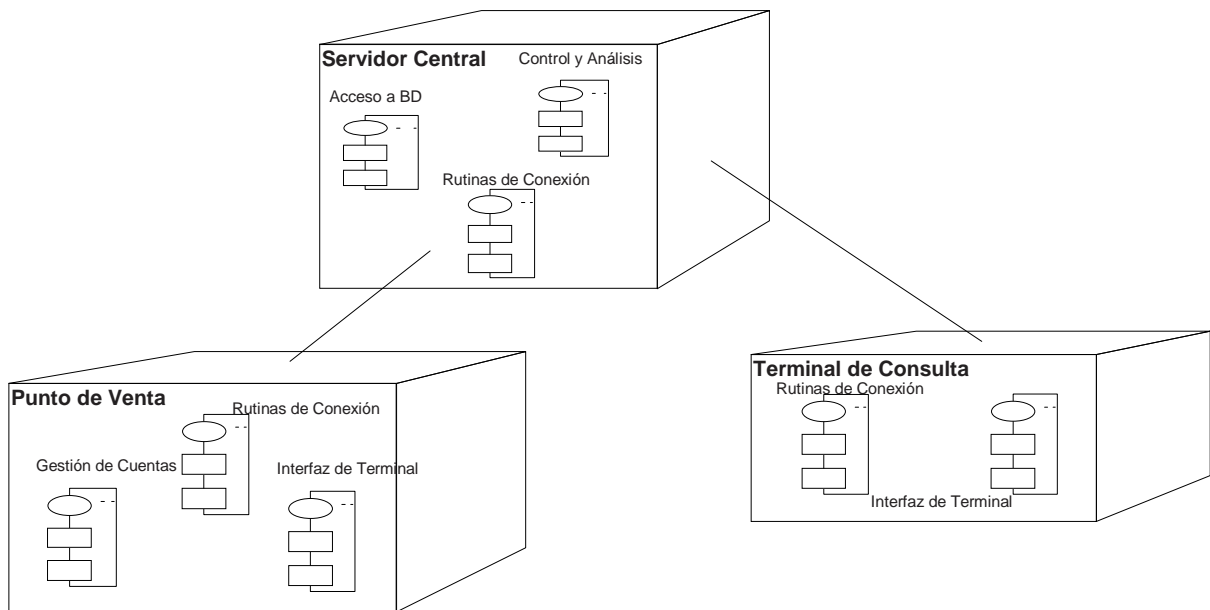
7.62



## Diagramas - Estructurales

- **De Despliegue:**

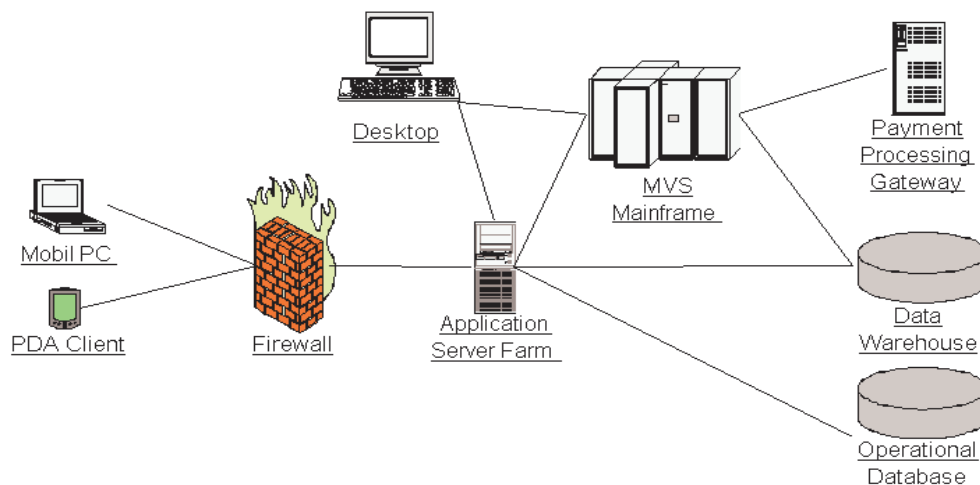
- Ejemplo.



## Diagramas - Estructurales

- **De Despliegue:**

- Mediante **iconos especializados** se puede precisar la naturaleza de los nodos como constituyentes físicos (dispositivos, archivos, bases de datos, etc.) de un sistema.

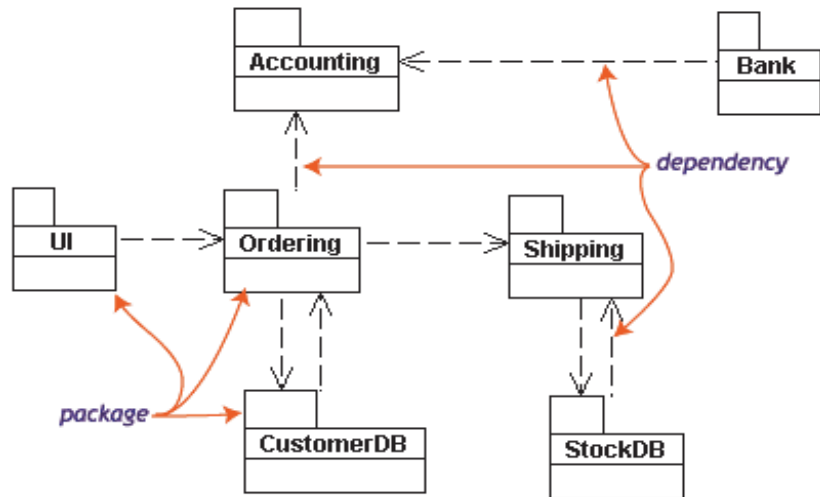




## Diagramas - Estructurales

### • De Paquetes:

- Muestran la descomposición del propio modelo en unidades organizativas (paquetes) y sus dependencias.
- Sirven para simplificar los diagramas de clases complejos, permitiendo el agrupamiento de los clasificadores en paquetes.



Francisco Ruiz - IS1

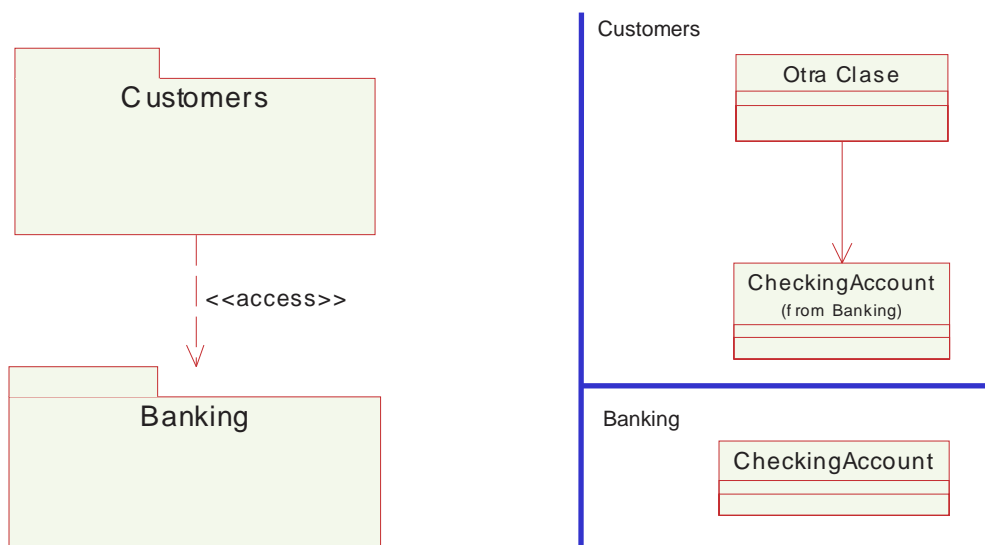
7.65



## Diagramas - Estructurales

### • De Paquetes:

- Una clase de un paquete puede aparecer en otro paquete por la importación a través de una relación de **dependencia** entre paquetes



Francisco Ruiz - IS1

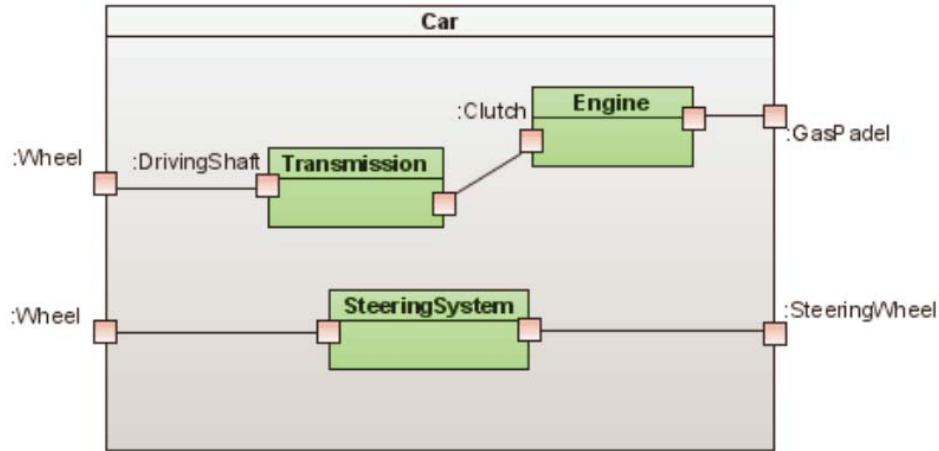
7.66



# Diagramas - Estructurales

## • De Estructura Compuesta:

- Muestran la **estructura interna** (incluyendo partes y conectores) de un clasificador estructurado o una colaboración.
- Muy parecidos a los diagramas de componentes.
- Ejemplo.



# Diagramas - De Comportamiento

- Los diagramas de comportamiento de UML 2 sirven para visualizar, especificar, construir y documentar los **aspectos dinámicos** de un sistema.
- Ser organizan en base a las formas en que se puede modelar la dinámica de un sistema. →

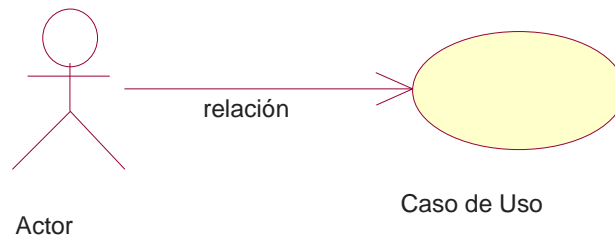
| Tipo de Diagrama          | Forma de Modelar la Dinámica                                        |
|---------------------------|---------------------------------------------------------------------|
| Casos de Uso              | Organizar los comportamientos                                       |
| Secuencia                 | Ordenación temporal de los mensajes                                 |
| Comunicación              | Organización estructural de los objetos y envían y reciben mensajes |
| Estados                   | Estado cambiante de objetos dirigidos por eventos                   |
| Actividades               | Flujo de control de actividades                                     |
| Tiempos                   | Tiempo real de los mensajes y estados                               |
| Revisión de Interacciones | Vista general de las interacciones                                  |



## Diagramas – De Comportamiento

### • De Casos de Uso:

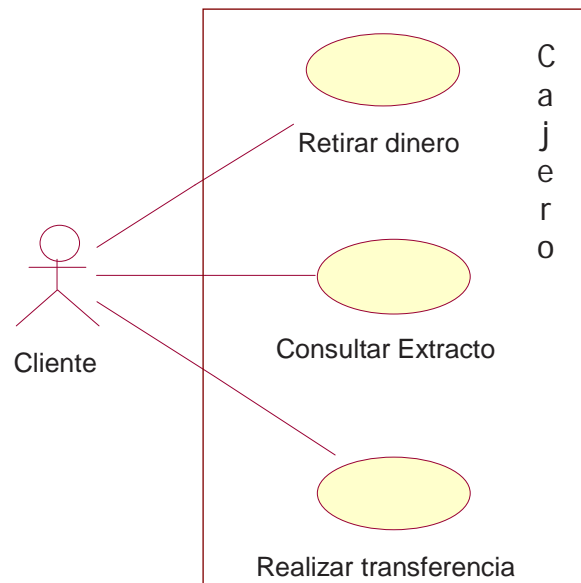
- Muestran un conjunto de **casos de uso** y **actores** (tipo especial de clase) y sus relaciones.
- Cubren la vista de casos de uso estática de un sistema.
- Son importantes en el modelado y organización del comportamiento de un sistema.
- Juegan un papel clave en metodologías muy usadas (desarrollo dirigido por casos de uso).



## Diagramas – De Comportamiento

### • De Casos de Uso:

- **Casos de Uso** es una técnica para capturar información respecto de los servicios que un sistema proporciona a su entorno (captura y especificación de requisitos).

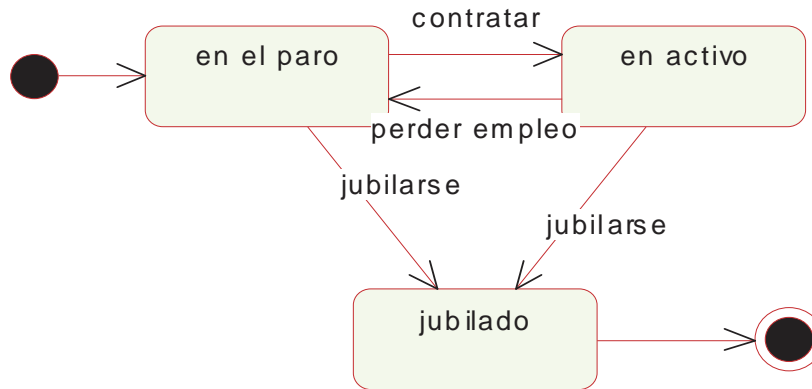




## Diagramas – De Comportamiento

- **De Estados:**

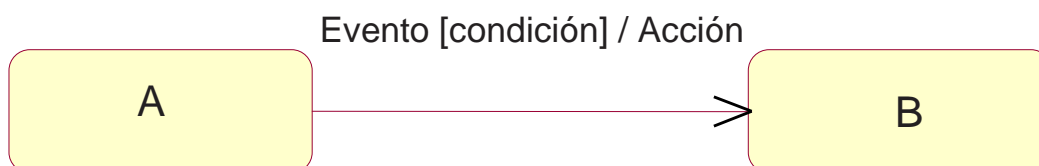
- Muestran **máquinas de estados**, que constan de:
  - estados, transiciones, eventos y actividades.
- Cubren la vista dinámica de un objeto.
- Son especialmente importantes en el modelado de una interfaz, clase o colaboración con comportamiento significativo.
- Resaltan el comportamiento dirigido por eventos de un objeto.



## Diagramas – De Comportamiento

- **De Estados:**

- **Estados y Transiciones:**



Tanto el evento como la acción se consideran instantáneos

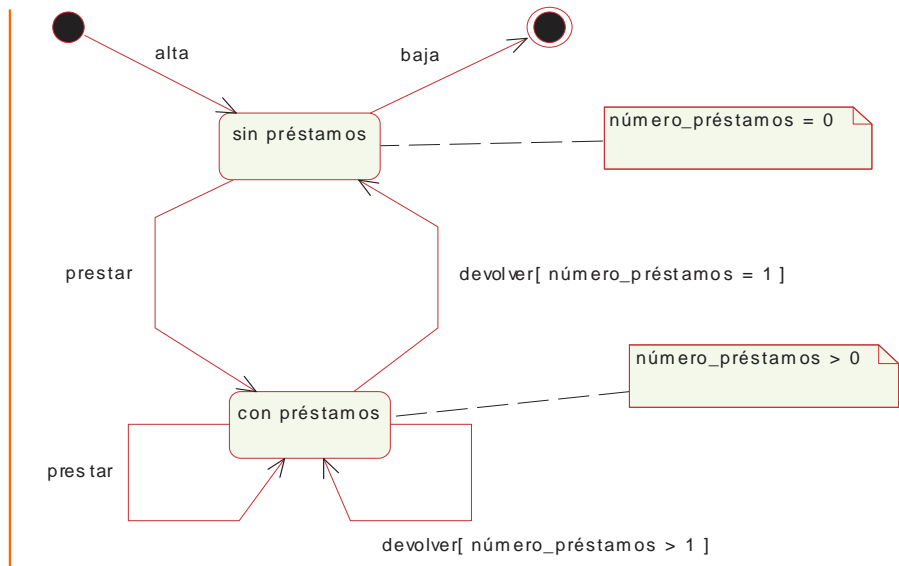


# Diagramas – De Comportamiento

## De Estados:

### Ejemplo.

| Socio                                      |
|--------------------------------------------|
| número : int                               |
| nombre : char[50]                          |
| número_prestamos : int = 0                 |
| alta()                                     |
| baja()                                     |
| prestar(código_libro : int, fecha : date)  |
| devolver(código_libro : int, fecha : date) |

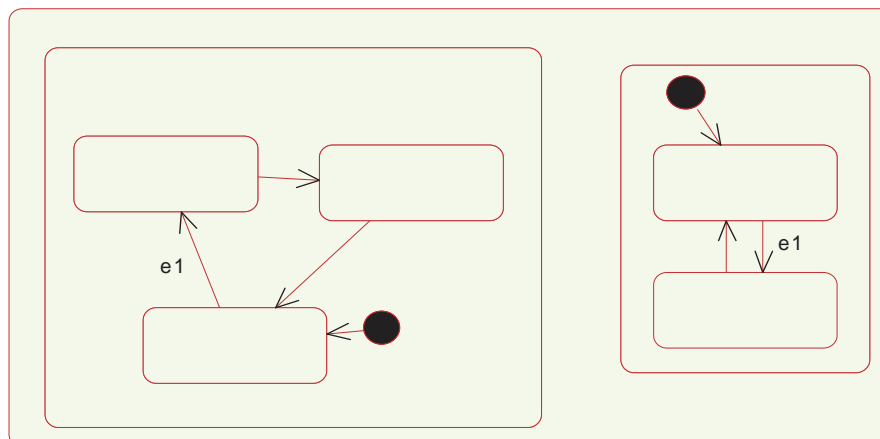


# Diagramas – De Comportamiento

## De Estados:

### Generalización de estados:

- Para reducir la complejidad, es posible formar estados compuestos (superestado) formados por agregación de estados más simples (subestados).
- En cada estado compuesto el objeto estará en alguno de los estados de cada uno de los subestados componentes (conurrencia de estados).





## Diagramas – De Comportamiento

### • De Actividad:

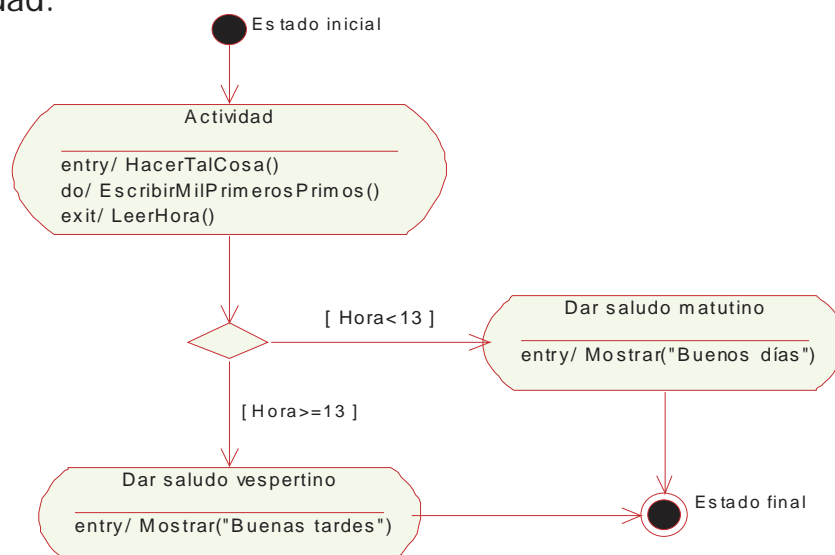
- Muestran el flujo paso a paso de una computación (proceso, flujo de control o flujo de datos).
- Una **actividad** muestra un conjunto de **acciones**, el **flujo** entre ellas y los **valores** producidos o consumidos.
- Cubren la vista dinámica de un sistema.
- Resaltan el flujo de control entre objetos.
- Son el equivalente en OO a los diagramas de flujo y DFDs.
- Se emplean para especificar:
  - Una operación compleja.
  - Un proceso de negocio (business process) o flujo de trabajo (workflow).
  - El proceso de negocio asociado a un caso de uso.



## Diagramas – De Comportamiento

### • De Actividad:

- Las actividades se enlazan por transiciones automáticas.
- Cuando una actividad termina se desencadena el paso a la siguiente actividad.

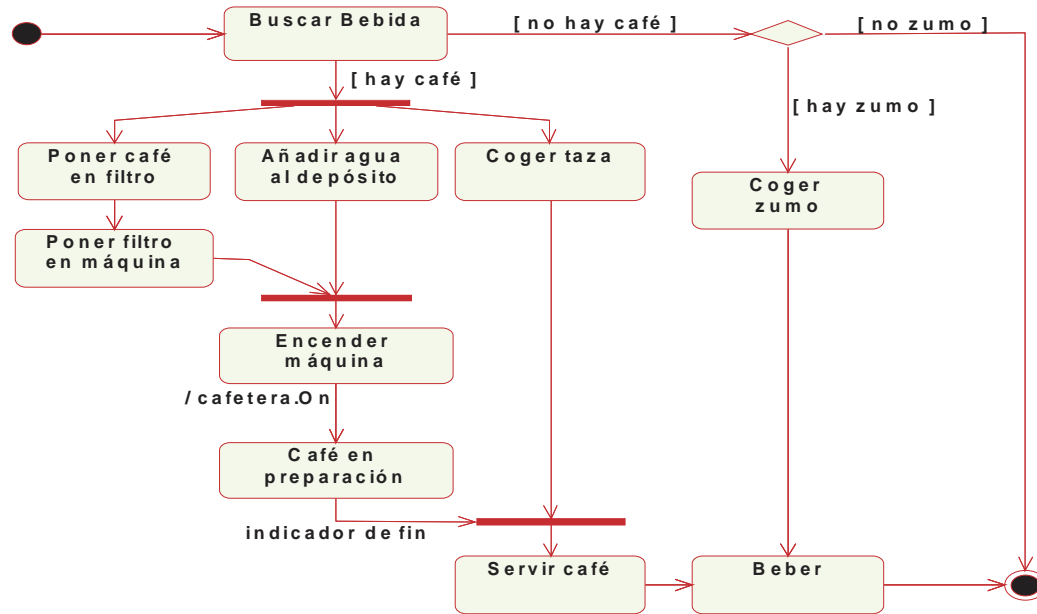




# Diagramas – De Comportamiento

## De Actividad:

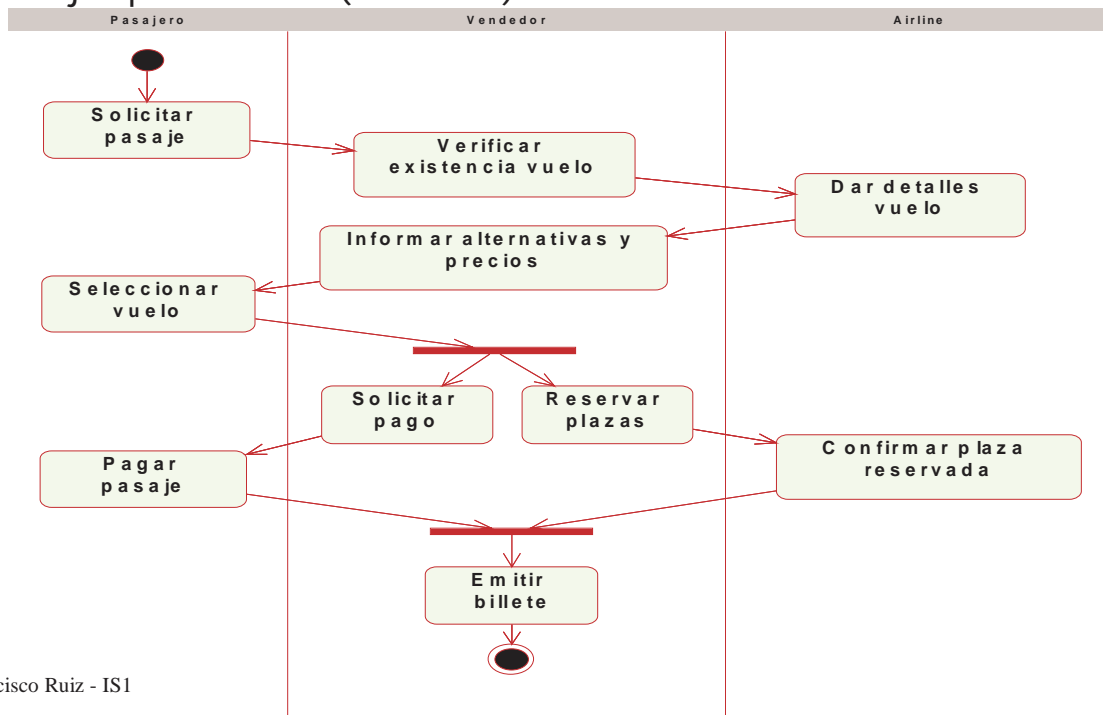
- Ejemplo.



# Diagramas – De Comportamiento

## De Actividad:

- Ejemplo con roles (*swimlines*).





## Diagramas – De Comportamiento

### • Diagramas de Interacción

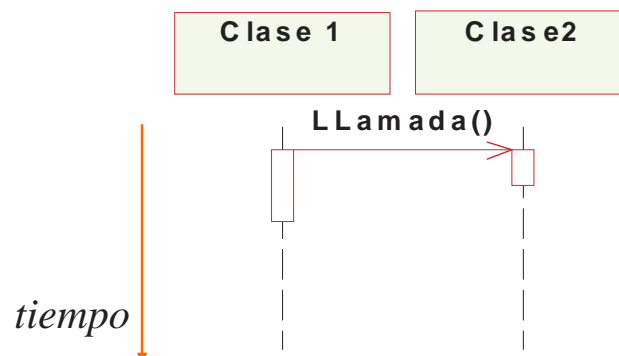
- Son un grupo especial de diagramas de comportamiento que muestran una **interacción**:
  - Conjunto de objetos o roles y mensajes que pueden ser enviados entre ellos.
- Cubren la vista dinámica de un sistema.
- Los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones.
- UML 2 incluye los siguientes
  - **Secuencia**
  - **Comunicación** (antiguo de Colaboración en UML 1.x)
  - **Tiempos**
  - **Revisión de las Interacciones**



## Diagramas – De Comportamiento

### • De Secuencia:

- Es un diagrama de interacción que resalta la ordenación temporal de los mensajes.
- Presentan un conjunto de **roles** y los **mensajes** enviados y recibidos por las instancias que interpretan dichos roles.
- Habitualmente, sirven para mostrar como interaccionan unos objetos con otros en un caso de uso.

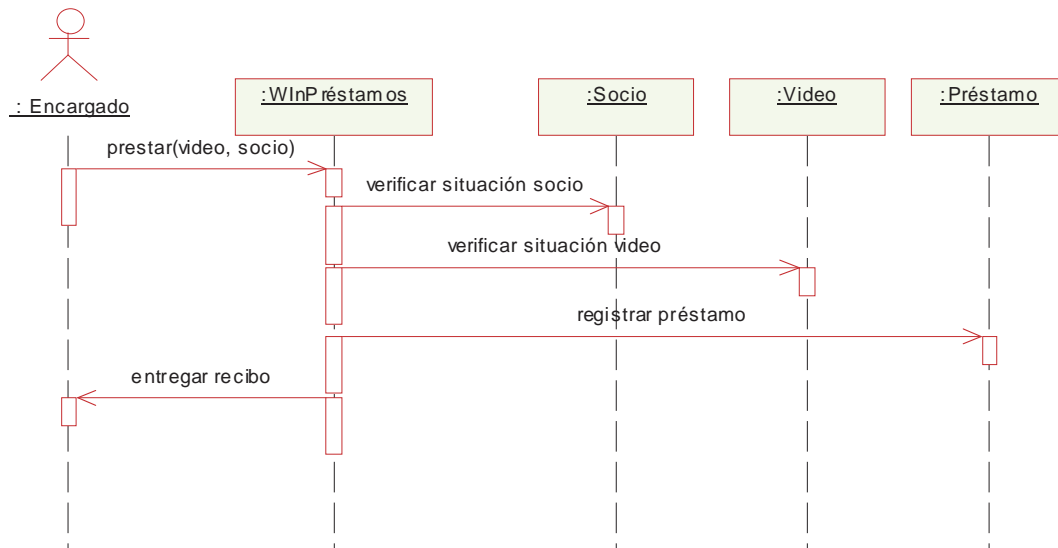




## Diagramas – De Comportamiento

- **De Secuencia:**

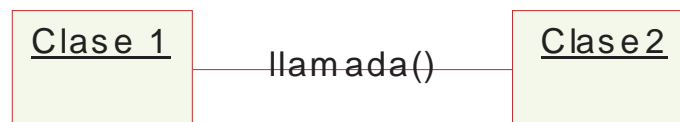
- Ejemplo.



## Diagramas – De Comportamiento

- **De Comunicación:**

- Es un diagrama de interacción que resalta la organización estructural de los objetos o roles que envían y reciben mensajes.
- Muestran un conjunto de **roles**, enlaces entre ellos y los **mensajes** enviados y recibidos por las instancias que interpretan dichos roles.
- Se usan para modelar el comportamiento dinámico de un caso de uso.



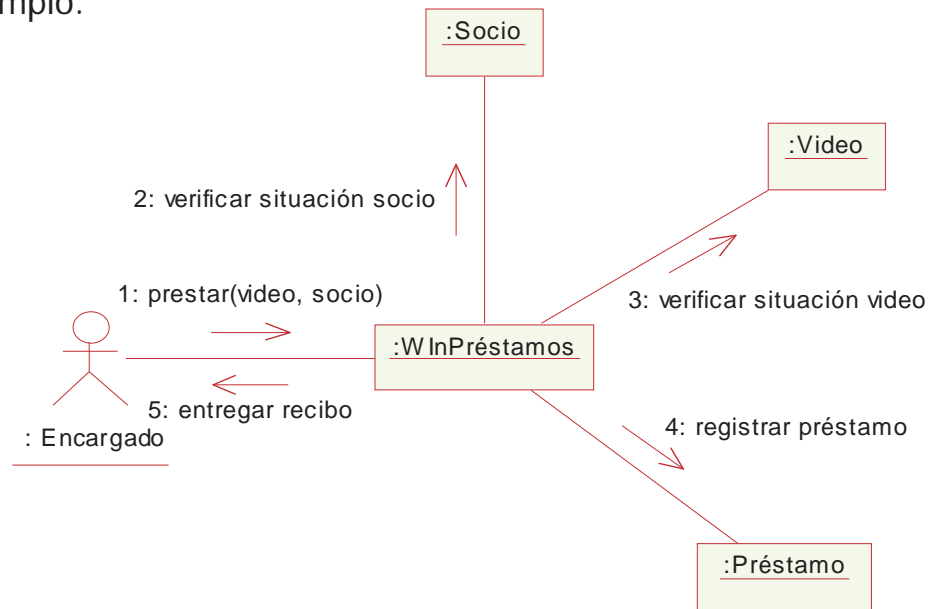
- En versiones anteriores a UML 2 se llamaban de colaboración.



# Diagramas – De Comportamiento

## • De Comunicación:

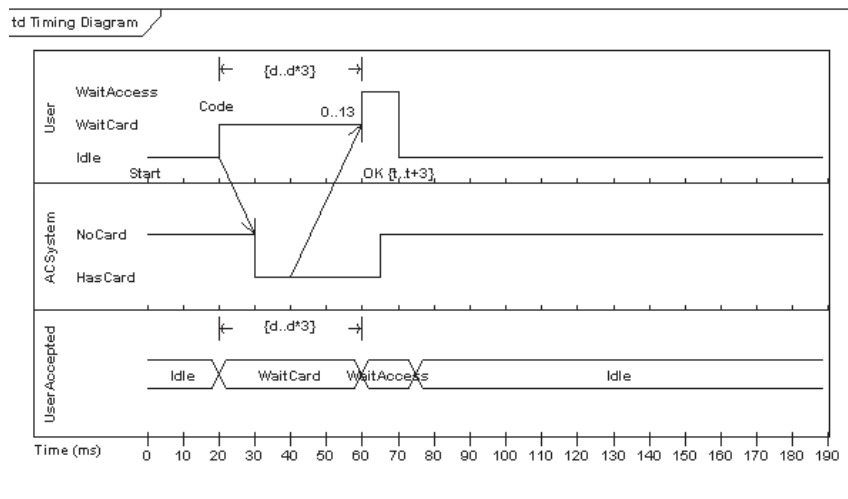
- Ejemplo.



# Diagramas – De Comportamiento

## • De Tiempos:

- Muestran los tiempos reales entre diferentes objetos o roles.
  - Comportamiento de los objetos en un periodo determinado de tiempo.
- Son una forma especial de diagramas de secuencia (los ejes están girados).

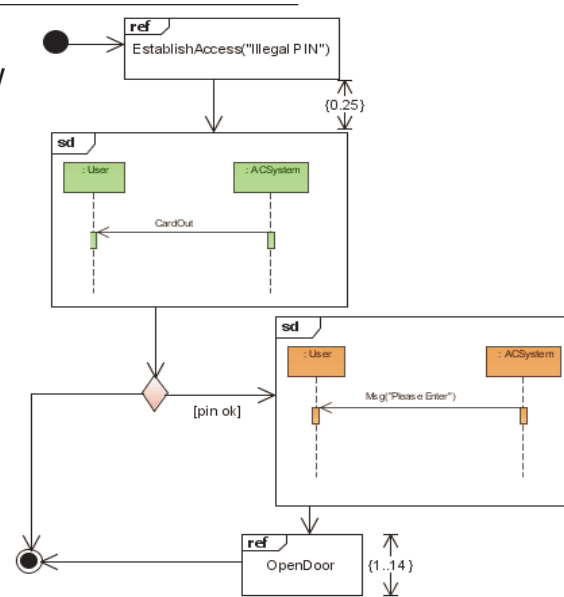




# Diagramas – De Comportamiento

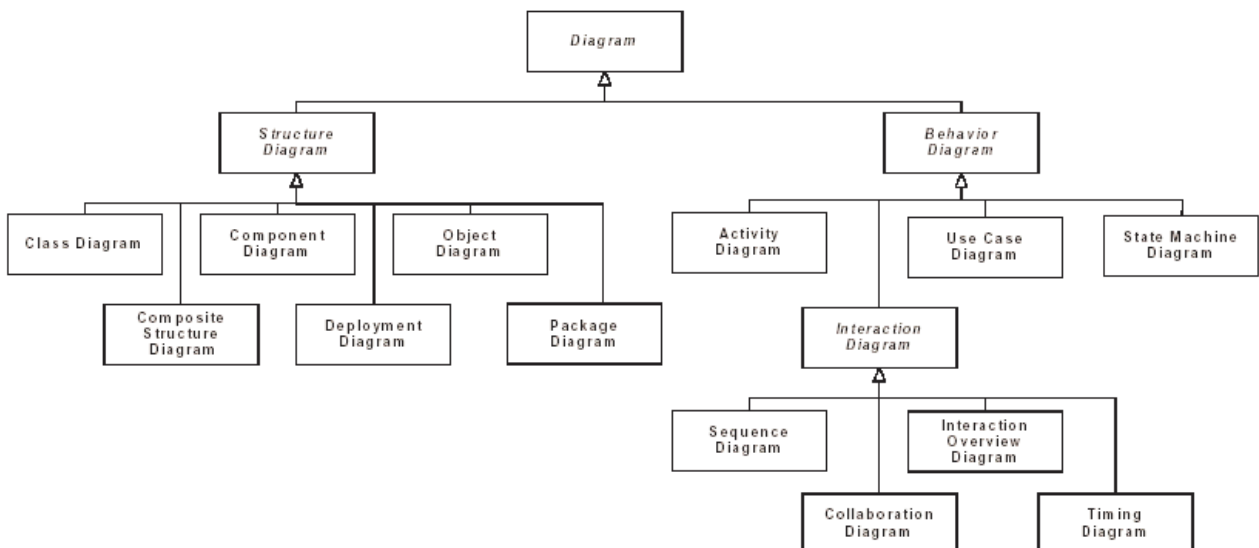
- **De Revisión de Interacciones:**

- Aportan una **visión general** del flujo de control de las interacciones.
- Híbrido entre diagrama de actividad y diagrama de secuencia.
- También llamados  
Visión Global de Interacciones



# Diagramas

- Jerarquía de diagramas en **UML 2**





## Reglas

- Los bloques de construcción de UML no pueden combinarse de cualquier manera.
- Como cualquier lenguaje, UML tiene unas reglas que especifican a qué debe parecerse un **modelo bien formado**.
  - *Semánticamente autoconsistente y que está en armonía con todos sus modelos relacionados.*



## Reglas

- UML tiene **reglas sintácticas y semánticas** para:
  - **Nombres**: Cómo llamar a los elementos, relaciones y diagramas.
  - **Alcance**: El contexto que da significado específico a un nombre.
  - **Visibilidad**: Cómo se pueden ver y utilizar unos nombres por otros.
  - **Integridad**: Cómo se relacionan apropiada y consistentemente unos elementos con otros.
  - **Ejecución**: Qué significa ejecutar o simular un modelo dinámico.



## Reglas

- Los modelos construidos durante el desarrollo de un sistema
  - tienden a evolucionar, y
  - pueden ser vistos por diferentes usuarios de formas diferentes y en momentos diferentes.
- Por estas razones, en la **práctica** es común construir modelos que no están bien formados:
  - **Abreviados**: Ciertos elementos se ocultan para simplificar la vista.
  - **Incompletos**: Pueden estar ausentes ciertos elementos.
  - **Inconsistentes**: No se garantiza la integridad del modelo.
- Las reglas de UML estimulan (pero no obligan) a considerar buenas prácticas que llevan a tales **modelos "incorrectos"** a convertirse en bien formados con el paso del tiempo.



## Mecanismos Comunes

- Un edificio es más simple y armonioso si todo el se ajusta a un patrón de características comunes (estilo colonial).
- Igual ocurre con UML, que tiene cuatro mecanismos comunes que se aplican de forma forma consistente a través de todo el lenguaje:
  - **Especificaciones**
  - **Adornos**
  - **Divisiones comunes**
  - **Extensibilidad**



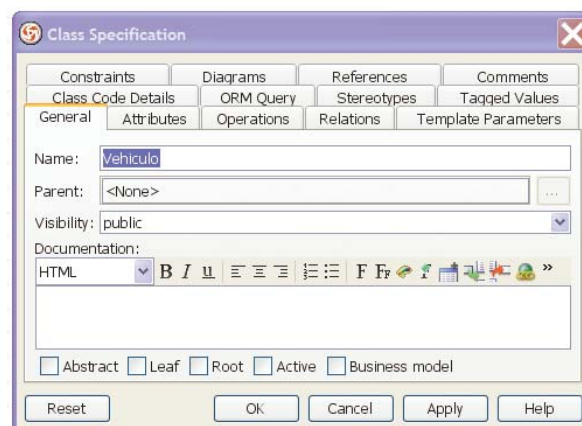
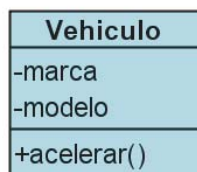
## Mecanismos Comunes - Especificaciones

- UML no es sólo un lenguaje gráfico.
- Detrás de cada elemento de su notación gráfica hay una **especificación** que proporciona una explicación textual de la sintaxis y semántica de ese bloque de construcción.
- La **notación gráfica** de UML se utiliza para **visualizar** el sistema.
- La **especificación** se utiliza para **expresar los detalles** de dicho sistema.
- Los diagramas UML son proyecciones visuales de la base semántica provisto por las especificaciones UML.



## Mecanismos Comunes - Especificaciones

- **Ejemplo de Especificación:**
  - Detrás del icono de una clase hay una especificación con información de los atributos, operaciones, firmas y comportamiento.
  - Visualmente el icono de la clase puede mostrar sólo parte de la especificación.





## Mecanismos Comunes - Adornos

- Todos los elementos en la notación gráfica de UML parten de un símbolo básico, al cual pueden añadirse una variedad de **adornos** específicos de ese símbolo.
  - La notación básica proporciona una representación visual de los aspectos más importantes del elemento.
  - La especificación incluye otros detalles, muchos de los cuales se pueden incluir como adornos gráficos o textuales.



## Mecanismos Comunes - Adornos

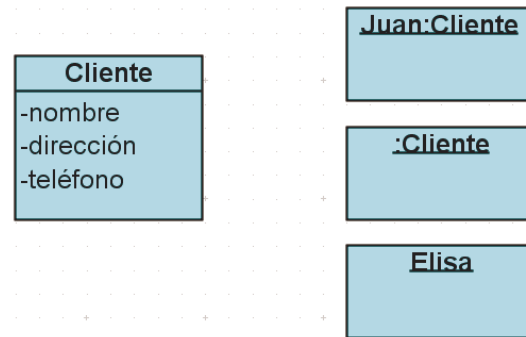
- **Ejemplo de Adornos:**
  - Notación básica de una clase:
    - Nombre, atributos, operaciones.
  - Adornos:
    - Es abstracta o no, visibilidad de los atributos y operaciones, tipos de los atributos, ...





## Mecanismos Comunes - Divisiones

- En el modelado orientado a objetos, existen varias **divisiones comunes** del mundo:
- **Clase vs Objeto.**
  - Una clase es una **abstracción**.
  - Un objeto es una **manifestación concreta** de dicha abstracción.

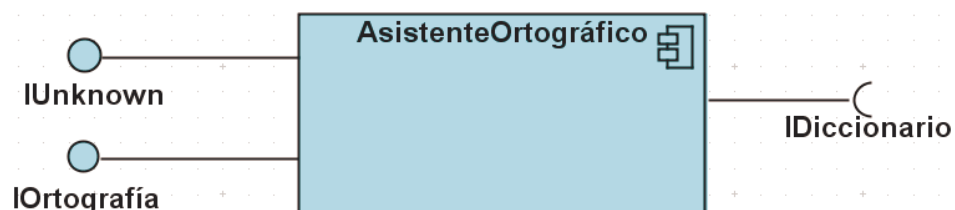


- Se da en muchos otros bloques de construcción de UML: caso de uso vs ejecución de caso de uso, componente vs instancia de componente; nodo vs instancia de nodo, etc.



## Mecanismos Comunes - Divisiones

- **Interfaz vs Implementación.**
  - Una interfaz declara un **contrato**.
  - Una implementación representa una **realización concreta** de ese contrato es una abstracción (hace efectiva la semántica completa de la interfaz).



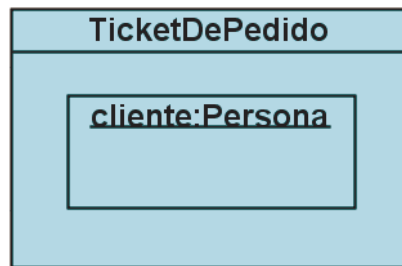
- Similar aparece en casi todos los bloques de construcción de UML: caso de uso vs colaboraciones que los realizan; operaciones vs métodos que las implementan.



## Mecanismos Comunes - Divisiones

- **Tipo vs Rol.**

- Un tipo declara la **clase de una entidad** (objeto, atributo, parámetro, ...).
- Un rol describe el **significado** de una entidad en un contexto (una clase, componente o colaboración).
- Cualquier entidad que forma parte de otra tiene ambas características y su significado depende de las dos.



## Mecanismos Comunes - Extensibilidad

- **Mecanismos de Extensión** en UML:

- **Estereotipos** (Stereotypes)
- **Valores Etiquetados** (Tagged Values)
- **Restricciones** (Constraints)
  - OCL (*Object Constraint Language*)
- Permiten extender UML de manera controlada para poder **expresar** todos los **matices** posibles de todos los modelos en todos los dominios en cualquier momento.
- Es posible añadir nuevos bloques de construcción (estereotipo), modificar la especificación de los existentes (valores etiquetados), e incluso cambiar su semántica (restricciones).



## Mecanismos Comunes - Extensibilidad

### • Estereotipo

- **Extiende el vocabulario** de UML permitiendo crear **nuevos tipos de bloques de construcción** que derivan de los existentes pero que son específicos a un problema.
- **Ejemplo**
  - En Java las excepciones son clases, aunque se tratan de formas especiales.

```
<<exception>>  
Overflow
```

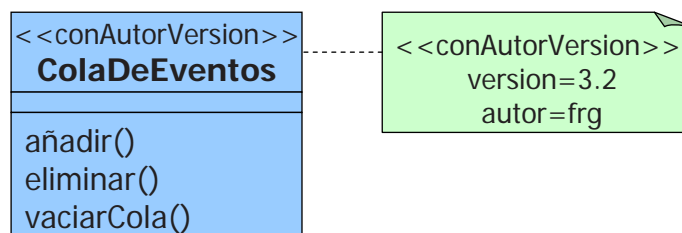
- Se pueden asociar símbolos específicos a los bloques de construcción estereotipados.



## Mecanismos Comunes - Extensibilidad

### • Valor Etiquetado

- **Extiende las propiedades** de un **estereotipo** de UML, permitiendo añadir nueva información en la especificación del estereotipo.
- **Ejemplo**
  - Añadir versión y autor mediante la creación del estereotipo `<<conAutorVersion>>` y asociándole una nota con los dos valores etiquetados.

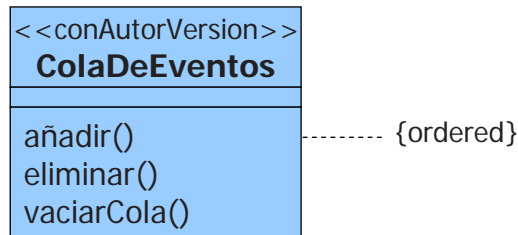




## Mecanismos Comunes - Extensibilidad

### • Restricción

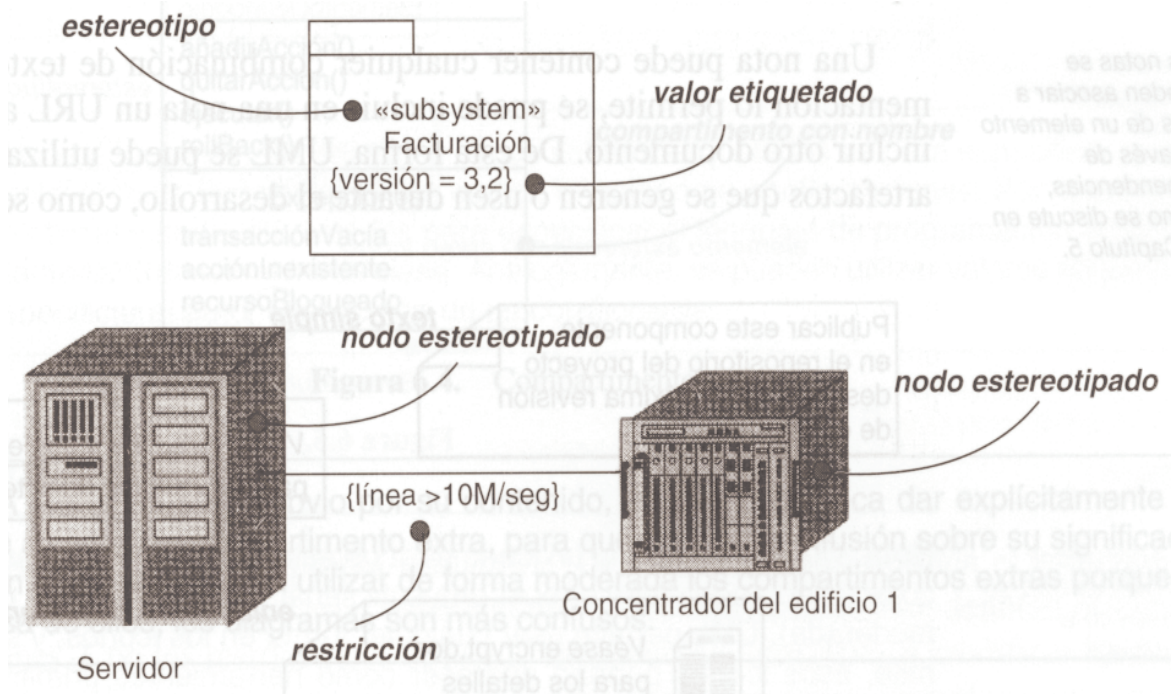
- **Extiende** la **semántica** de un bloque de construcción de UML, permitiendo añadir nuevas reglas o modificar las existentes.
- **Ejemplo:** Restringir la clase ColaDeEventos para que todas las adiciones se hagan en orden.



- Para especificar restricciones de forma precisa se usa OCL (Object Constraint Language). [se presenta al final del tema]



## Mecanismos Comunes - Extensibilidad



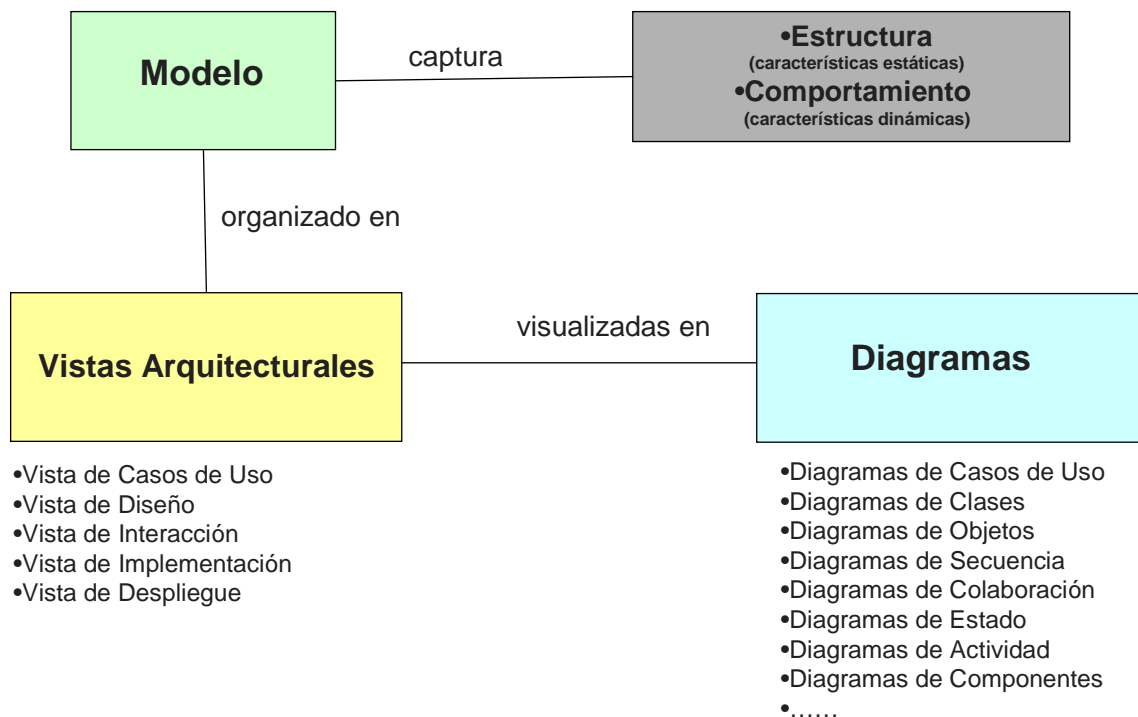


# Conceptos de Modelado

- **Sistema**
  - Colección de subsistemas organizados para lograr un propósito. Está descrito por un conjunto de modelos.
- **Subsistema**
  - Grupo de elementos, algunos de los cuales son una especificación del comportamiento ofrecido por los otros.
- **Modelo**
  - Simplificación completa y autoconsistente de la realidad, creado para comprender mejor un sistema.
- **Vista (Arquitectural)**
  - Proyección de la organización y estructura de un modelo de un sistema, centrada en un aspecto.
- **Diagrama**
  - Representación gráfica de un conjunto de elementos (normalmente mostrado como grafo conexo de nodos (elementos) y arcos relaciones).



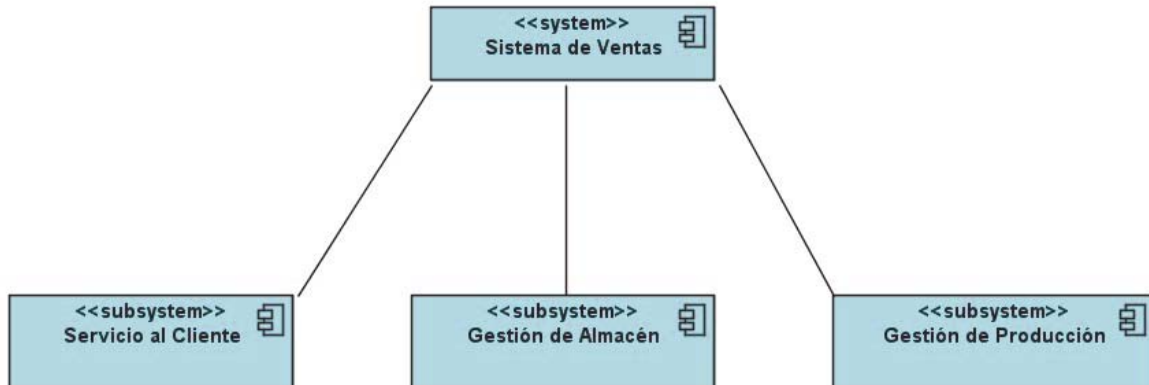
# Conceptos de Modelado





## Conceptos de Modelado

- UML 2 proporciona estereotipos de componente para modelar **sistemas** y **subsistemas**.

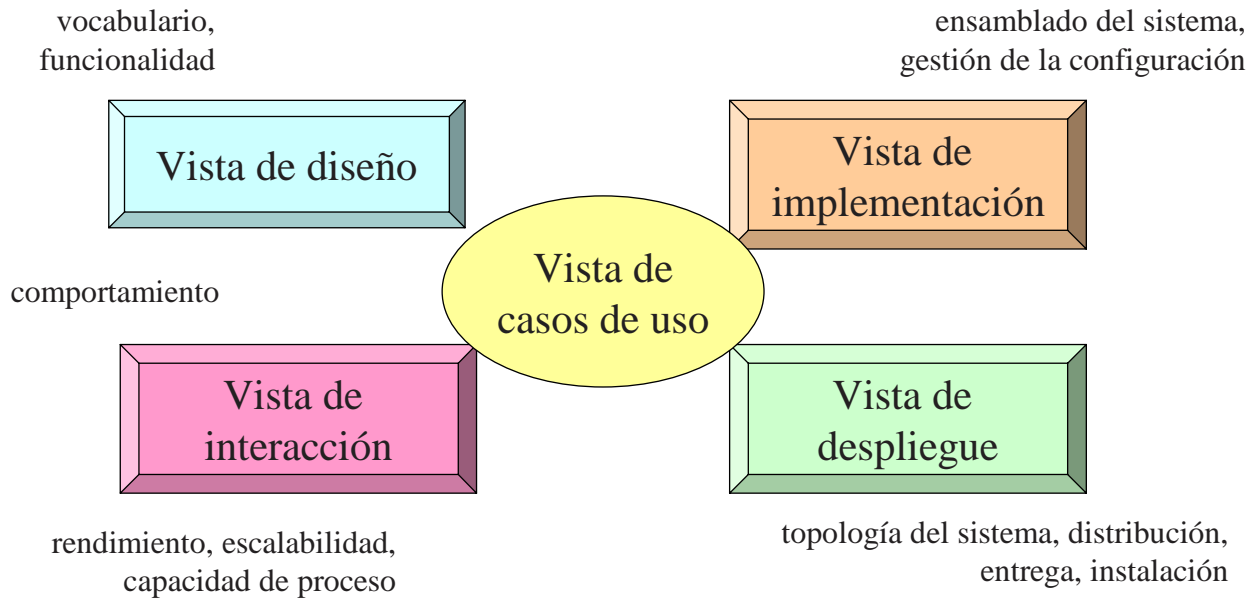


## Conceptos de Modelado - Vistas Arquitecturales

- Durante el desarrollo de un sistema intensivo en software se requiere que el sistema sea visto desde **varias perspectivas**.
- Diferentes usuarios** miran el sistema de formas diferentes en momentos diferentes.
- La arquitectura del sistema, clave para poder manejar estos puntos de vista diferentes, puede describirse mejor a través de **vistas arquitecturales** interrelacionadas.
  - Proyecciones de la organización y estructura del sistema, centradas en un aspecto particular.



### Las 5 Vistas Arquitecturales de UML 2



### • Vista de Casos de Uso

- Comprende los **casos de uso** que describen el comportamiento del sistema tal y como es percibido por los usuarios finales, analistas y encargados de pruebas.
- En esta vista no se especifica la organización real del sistema software.
- Los **diagramas** que le corresponden son:
  - Aspectos **estáticos**: diagramas de casos de uso.
  - Aspectos **dinámicos**: diagramas de interacción, de estados y de actividades.



## Conceptos de Modelado - Vistas Arquitecturales

### • Vista de Diseño

- Comprende las clases, interfaces y colaboraciones que forman el **vocabulario del problema y su solución**.
- Soporta, principalmente, los **requisitos funcionales** del sistema (servicios que el sistema debe proporcionar a sus usuarios finales).
- Los **diagramas** que le corresponden son:
  - Aspectos **estáticos**: diagramas de **clases** y de **objetos**. También son útiles los diagramas de **estructura compuesta** de clases.
  - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.



## Conceptos de Modelado - Vistas Arquitecturales

### • Vista de Interacción

- Muestra el **flujo de control** entre las diversas **partes del sistema**, incluyendo los posibles mecanismos de concurrencia y sincronización.
- Abarca en especial **requisitos no funcionales** como el rendimiento, escalabilidad y capacidad de procesamiento.
- Los **diagramas** que le corresponden son los mismos que la vista de diseño:
  - Aspectos **estáticos**: diagramas de **clases** y de **objetos**.
  - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.
- Pero atendiendo más las **clases activas** que controlan el sistema y los **mensajes** entre ellas.



### • Vista de Implementación

- Comprende los **artefactos** que se utilizan para ensamblar y poner en producción el sistema **software real**.
- Se centra en:
  - La configuración de las distintas versiones de los archivos físicos,
  - Correspondencia entre clases, y
  - Correspondencia entre componentes lógicos y artefactos físicos.
- Los **diagramas** que le corresponden son:
  - Aspectos **estáticos**: diagramas de **despliegue** (especialmente la versión de **artefactos**).
  - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.



### • Vista de Despliegue

- Contiene los **nodos** que forman la topología **hardware** sobre la que se ejecuta el sistema software.
- Se ocupa principalmente de la distribución, entrega e instalación de las partes que forman el sistema software real.
- Los **diagramas** que le corresponden son:
  - Aspectos **estáticos**: diagramas de **despliegue**.
  - Aspectos **dinámicos**: diagramas de **interacción**, de **estados** y de **actividades**.



## Conceptos de Modelado - Vistas Arquitecturales

- Cada vista puede existir de forma independiente.
- Pero a la vez interactúan entre sí:
  - Los nodos (vista de despliegue) contienen componentes (vista de implementación).
  - Dichos componentes representan la realización (software real) de las clases, interfaces, colaboraciones y clases activas (vistas de diseño y de interacción).
  - Dichos elementos de las vistas de diseño e interacción representan el sistema solución a los casos de uso (vista de casos de uso) que expresan los requisitos.



## Conceptos de Modelado - Vistas Arquitecturales

- Resumen del papel de los principales tipos de diagramas y sus conceptos.

| Area principal     | Vista                | Diagrama                                                             | Conceptos principales                                                                                                                         |
|--------------------|----------------------|----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Estructural        | Estática             | Clases                                                               | asociación, clase, dependencia, generalización, interfaz, realización                                                                         |
|                    |                      | Estructura Interna                                                   | conector, interfaz, parte, puerto                                                                                                             |
|                    | Diseño               | Comunicación                                                         | colaboración, conector, rol, uso de la colaboración                                                                                           |
|                    |                      | Componentes                                                          | componente, dependencia, interfaz, puerto, realización, sistema, subsistema                                                                   |
| Casos de Uso       | Casos de Uso         | actor, asociación, caso de uso, extensión, generalización, inclusión |                                                                                                                                               |
| Dinámica           | Máquinas de Estados  | Estados                                                              | disparador, efecto, estado, evento, región, transición                                                                                        |
|                    | Actividad            | Actividad                                                            | acción, actividad, bifurcación, control de flujo, excepción, flujo de datos, fusión (join), nodo de control, nodo objeto, región de expansión |
|                    |                      | Interacción                                                          | Secuencia                                                                                                                                     |
|                    | Comunicación         |                                                                      | colaboración, condición de guarda, interacción, mensaje, rol, número de secuencia                                                             |
| Física             | Despliegue           | Despliegue                                                           | artefacto, componente, dependencia, localización, manifestación, nodo                                                                         |
| Gestión del Modelo | Gestión del Proyecto | Paquetes                                                             | modelo, paquete, subsistema                                                                                                                   |
|                    | Perfil UML           | Paquetes                                                             | estereotipo, perfil, restricción, valor etiquetado                                                                                            |



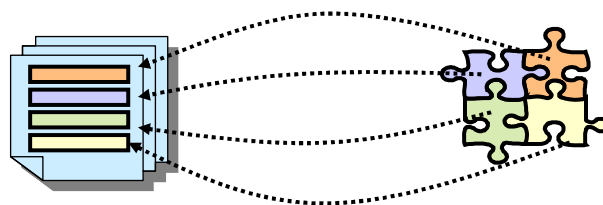
## Conceptos de Modelado - Modelos

- Un **modelo** captura las propiedades estructurales (estática) y de comportamiento (dinámicas) de un sistema.
  - Es una abstracción de dicho sistema, considerando un cierto propósito.
  - El modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle.



## Conceptos de Modelado - Modelos

- 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 un punto de vista del sistema. Sin embargo, existen relaciones de trazabilidad entre los diferentes modelos.





## Conceptos de Modelado - Modelos

- Los modelos a usar vienen determinados por la metodología empleada.
- En **Rational Unified Process (RUP)**
  - De Casos de Uso del Negocio (Business Use-Case Model)
  - De Objetos del Negocio (Business Object Model)
  - De Casos de Uso (Use-Case Model)
  - De Análisis (Analysis Model)
  - De Diseño (Design Model)
  - De Despliegue (Deployment Model)
  - De Datos (Data Model)
  - De Implementación (Implementation Model)
  - De Pruebas (Test Model)



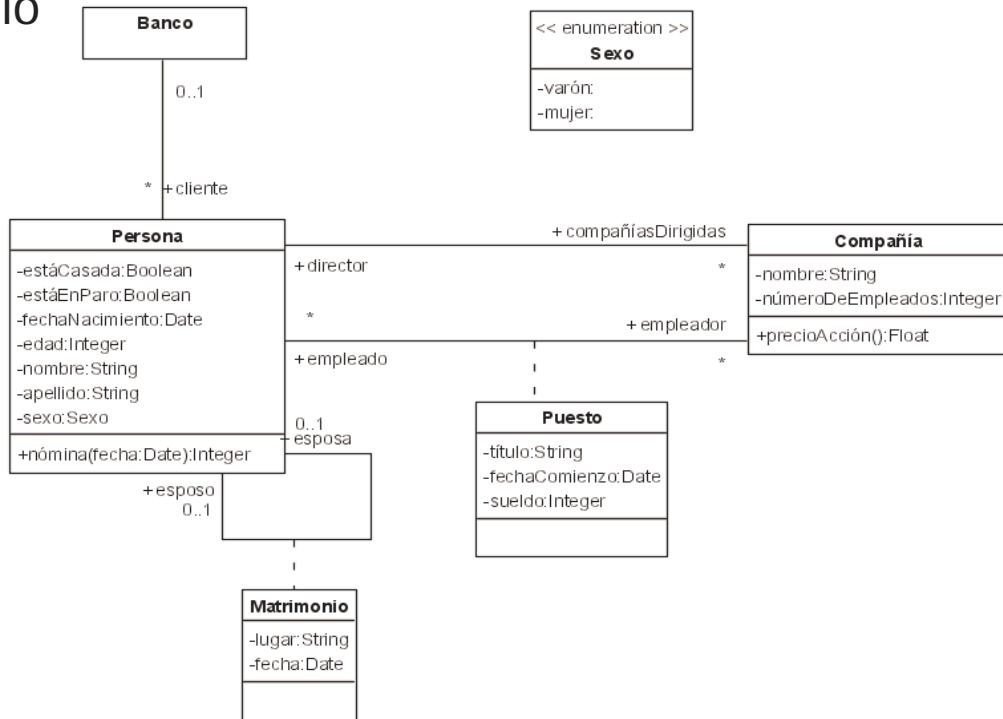
## OCL

- **Object Constraint Language 2.0**
- Complemento a UML, parte del estándar.
  - **Lenguaje** para escribir **expresiones formales** acerca de modelos UML.
  - Usos:
    - Lenguaje de **consulta**.
    - Especificación de **invariantes** en clases y tipos.
    - Especificación de invariantes de tipo para Estereotipos.
    - Describir pre- y post-**condiciones** en Operaciones y Métodos
    - Describir Guardas.
    - Especificar **destinatarios** para mensajes y acciones.
    - Especificar **restricciones** en Operaciones.
    - Especificar **reglas de derivación** para atributos.



# OCL

- Ejemplo



Francisco Ruiz - IS



# OCL

- **Invariantes:**

- Condiciones o restricciones que deben cumplirse siempre.

- Ejemplo

*“El número de empleados debe ser mayor que 50”*

context Compañía

inv: self. númeroDeEmpleados > 50

context c:Compañía

inv suficientesEmpleados: c.númeroDeEmpleados > 50



## OCL

- **Condiciones**

- Pre-condiciones o post-condiciones.

- Sintaxis

```
context NombreTipo::NombreOperación(Param1 : Tipo1, ...
):TipoRetorno
```

```
pre parametroOk: param1 < ...
```

```
post resultadoOk : result > ...
```

- Ejemplo

```
context Persona::nómina(fecha : Date) : Integer
```

```
post: result > 5000
```



## OCL

- **Valores iniciales y derivados**

- Sintaxis

```
context NombreTipo::NombreAtributo: Tipo
```

```
init: -- alguna expresión representando el valor inicial
```

```
context NombreTipo::NombreRolAsociación: Tipo
```

```
init: -- alguna expresión representando la regla de derivación
```

- Ejemplos

```
context Persona::nómina : Integer
```

```
init: padres.nómina->sum() * 1%
```

```
derive: if menorDeEdad then
```

```
        padres.nómina->sum() * 1%
```

```
    else
```

```
        puesto.sueldo
```

```
    endif
```



- **Definiciones**

- Ejemplo

context Persona

def: ingresos : Integer = self.puesto->sum()

def: apodo : String = 'Gallito rojo'

def: tieneElTítulo(t : String) : Boolean = self.puesto->exists(título = t)



- **Navegación**

- Ejemplos

a) "Los casados tienen al menos 18 años de edad"

context Persona inv:

self.esposa->notEmpty() implies self.esposa.edad >= 18 and

self.esposo->notEmpty() implies self.esposo.edad >= 18

b) "Una compañía tiene a lo más 50 empleados"

context Compañía inv:

self.empleado->size() <= 50