



**INSTITUTO POLITÉCNICO NACIONAL**  
**SECRETARÍA DE APOYO ACADÉMICO**  
**DIRECCIÓN DE INFORMÁTICA**

**U M L**

**Lenguaje Unificado de Modelado**  
**(Unified Modeling Language)**

## Tabla de Contenido

<b>1</b>	<b>INTRODUCCIÓN .....</b>	<b>5</b>
<b>1.1</b>	<b>Antecedentes .....</b>	<b>6</b>
1.1.1	Orígenes .....	6
1.1.2	UML es un lenguaje.....	6
1.1.3	¿Por qué modelamos ? .....	7
<b>1.2</b>	<b>UML de un Vistazo .....</b>	<b>8</b>
1.2.1	Cosas en UML .....	8
1.2.2	Relaciones en UML .....	11
1.2.3	Diagramas en UML 1.3.....	13
1.2.4	Diagrama de Casos de Uso .....	14
1.2.5	Diagrama de Clases.....	15
1.2.6	Diagrama de Objetos.....	16
1.2.7	Diagramas de Interacción .....	17
1.2.8	Diagrama de Colaboración .....	18
1.2.9	Diagrama de Secuencia.....	19
1.2.10	Diagrama de Transición de Estados.....	20
1.2.11	Diagrama de actividad .....	21
1.2.12	Diagrama de Distribución.....	22
1.2.13	Diagrama de Componentes .....	23
1.2.14	Diagrama de Paquetes.....	24
<b>2</b>	<b>CLASES, OBJETOS, ATRIBUTOS, OPERACIONES Y RELACIONES: DIAGRAMAS DE CLASES.....</b>	<b>25</b>
<b>2.1</b>	<b>Conceptos Básicos .....</b>	<b>26</b>
<b>2.2</b>	<b>Cadenas (Links) .....</b>	<b>34</b>
<b>2.3</b>	<b>Asociaciones.....</b>	<b>35</b>
<b>2.4</b>	<b>Asociaciones Ternarias.....</b>	<b>37</b>
<b>2.5</b>	<b>Multiplicidad (Cardinalidad).....</b>	<b>38</b>
2.5.1	Ejemplos de cardinalidades.....	38
2.5.2	Ejemplos de cardinalidades.(cont.) .....	39
<b>2.6</b>	<b>Atributos de una asociación. ....</b>	<b>41</b>
<b>2.7</b>	<b>Agregación (Composición) .....</b>	<b>44</b>
<b>2.8</b>	<b>Roles .....</b>	<b>47</b>
<b>2.9</b>	<b>Ordenamiento.....</b>	<b>48</b>
<b>2.10</b>	<b>Calificación .....</b>	<b>49</b>
<b>2.11</b>	<b>Generalización y Herencia. ....</b>	<b>50</b>
<b>2.12</b>	<b>Caso Práctico.....</b>	<b>54</b>

2.13	Tips.....	57
2.14	Ejercicios.....	58
<b>3</b>	<b>ESPECIFICACIÓN DE REQUERIMIENTOS: CASOS DE USO .....</b>	<b>60</b>
3.1	Casos de Uso y diagramas de casos de uso.....	61
3.2	Actor.....	61
3.3	Casos de Uso .....	65
3.3.1	Identificación de los casos de Uso .....	66
3.3.2	Diagramas de Casos de Uso.....	67
3.4	Relaciones en un caso de uso .....	69
3.4.1	Relación de Uso .....	69
3.4.2	Otra relación entre casos de uso: Extensión.....	70
3.5	Casos de uso y escenarios .....	71
3.5.1	Ejemplo de un caso de uso.....	72
3.5.2	Ejemplo de un escenario .....	73
3.6	Sumario de uso de los casos de uso .....	74
<b>4</b>	<b>ESPECIFICACIÓN DE TRANSACCIONES: DIAGRAMAS DE INTERACCIONES.....</b>	<b>75</b>
4.1	Diagramas de Colaboración.....	76
4.2	Diagramas de Secuencia .....	77
4.3	Notación común en diagramas de interacción.....	78
4.4	Creación y Eliminación en Diagramas de colaboración .....	78
4.5	Creación y Eliminación en Diagramas de secuencia.....	79
4.6	Iteración en un diagrama de colaboración .....	80
4.7	Iteración en un diagrama de secuencia .....	80
4.8	Bifurcación en diagramas de colaboración.....	81
4.9	Bifurcación en diagramas de secuencia.....	81
4.10	Tiempos límite en diagramas de secuencia .....	82
4.11	Implementación de ligas en diagramas de colaboración .....	83
<b>5</b>	<b>MODELOS DINÁMICOS: DIAGRAMAS DE TRANSICIÓN DE ESTADOS ....</b>	<b>84</b>
5.1	El Modelamiento Dinámico.....	85

<b>5.2</b>	<b>Estado.....</b>	<b>86</b>
<b>5.3</b>	<b>Evento.....</b>	<b>89</b>
<b>5.4</b>	<b>Escenario: .....</b>	<b>90</b>
<b>5.5</b>	<b>Estados y su naturaleza .....</b>	<b>94</b>
<b>5.6</b>	<b>Diagramas de transición de estados .....</b>	<b>97</b>
5.6.1	Condiciones .....	105
5.6.2	Operaciones en un diagrama de transición de estados.....	107
<b>5.7</b>	<b>Notación (Resumen).....</b>	<b>111</b>
<b>5.8</b>	<b>Ejercicio: .....</b>	<b>111</b>
<b>5.9</b>	<b>Diagramas anidados.....</b>	<b>115</b>
<b>6</b>	<b>OTROS DIAGRAMAS .....</b>	<b>122</b>
<b>6.1</b>	<b>Diagrama de Actividad .....</b>	<b>123</b>
<b>7</b>	<b>BIBLIOGRAFÍA .....</b>	<b>143</b>

# I

## 1 Introducción

## 1.1 Antecedentes

### 1.1.1 Orígenes

OMT	Rumbaugh	1991
-----	----------	------

OOSE	Jacobson	1986
------	----------	------

METODO BOOCH	Booch	1981
--------------	-------	------

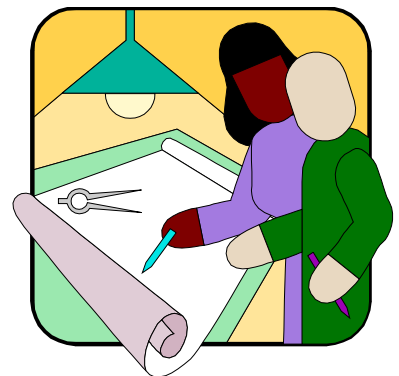
### 1.1.2 UML es un lenguaje

¿Qué implica esto?

No es una metodología

- Sirve para:

1. Modelar
2. Especificar
3. Construir
4. Documentar



### 1.1.3 ¿Por qué modelamos ?

- *¿Cómo haríamos un edificio sin planos?*
- *¿Cómo dividiríamos el trabajo entre los trabajadores?*
- *¿Cómo relacionaríamos la tubería y la instalación eléctrica con lo demás?*
- *¿Cómo manejaríamos un error de requerimientos?*
- *¿Esperaría el cliente hasta la terminación para evaluar si es lo que quiere?*



#### **Se modela:**

- Para manejar la complejidad
- Para enfocar aspectos específicos
- Es más barato
- Para proponer soluciones

## 1.2 UML de un Vistazo

### 1.2.1 Cosas en UML

Sobre estructuras  
(¿De que está formado...?)



Sobre comportamiento  
(¿Qué pasa cuando...?)

Sobre agrupamiento  
(¿Qué se agrupa con qué?)



## Cosas sobre estructura:

Casos de Uso

- Usuario



Clases

-Conceptos  
Negocio



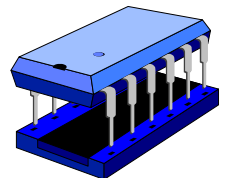
Interfaces

- Servicios ofrecidos



Componentes

- Agrupaciones



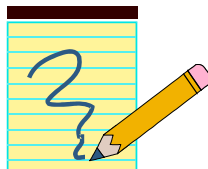
Nodos

comunicación Física



Colaboraciones

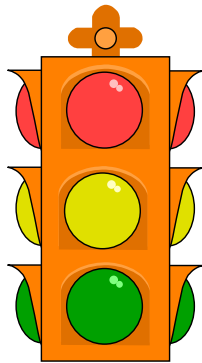
- Especificación



## Cosas sobre comportamiento:

Máquinas de estado

Estados, eventos,  
transiciones



Interacciones

Mensajes,  
operaciones



## Cosas sobre Agrupamiento:

- Paquetes
- Modelos
- Sistemas

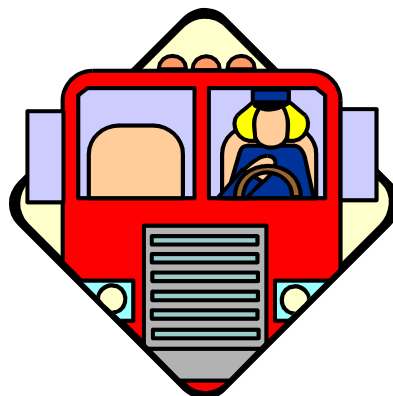
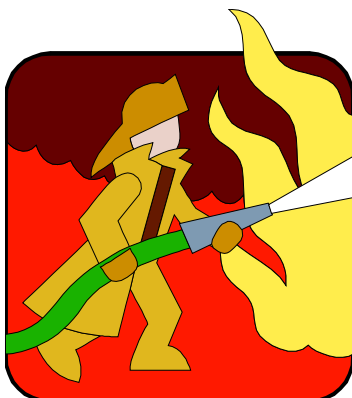
### **1.2.2 Relaciones en UML**

*Las cosas se enlazan conceptualmente vía relaciones*

#### Tipos principales de relaciones:

##### -Dependencia

¿Qué pasaría si se modificara...un caso de uso, un nodo, un concepto de negocio, etc?



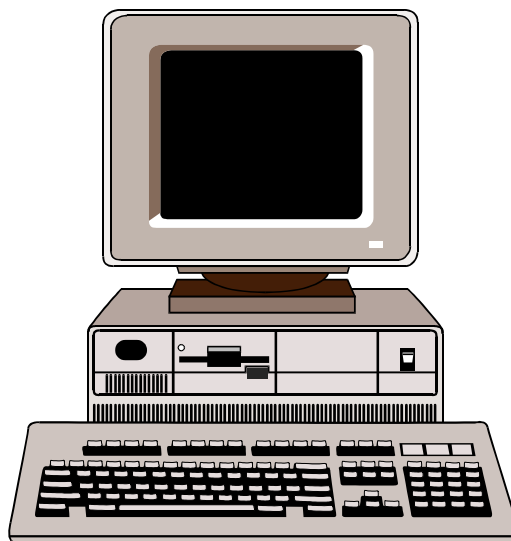
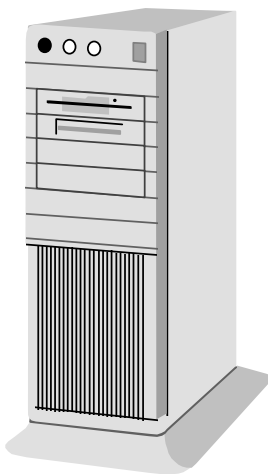
### -Asociación

¿Qué tiene que ver un caso de usos con otro, un concepto de negocio con otro, etc.?



### -Generalización

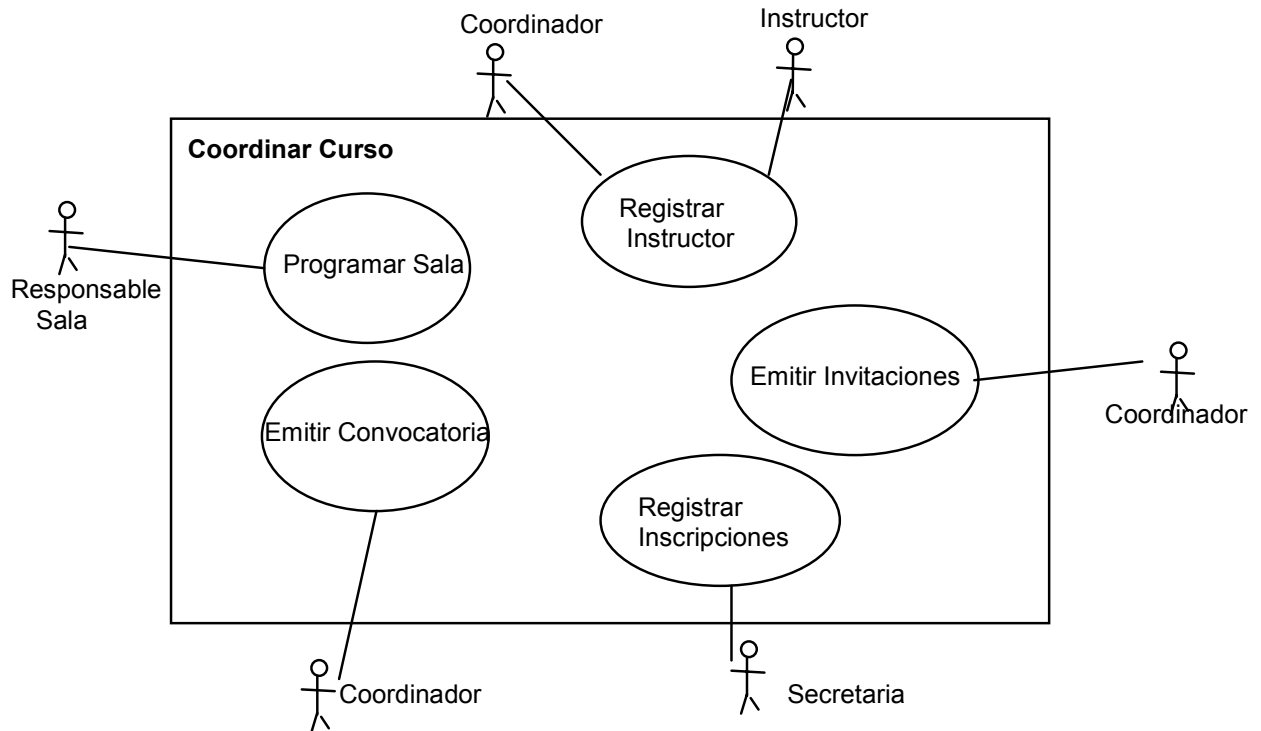
¿Cuántas clases de conceptos de negocios, nodos, casos de uso, etc., hay?



### **1.2.3 Diagramas en UML 1.3**

- Diagrama de Caso de Uso
- Diagrama de clases
- Diagrama de Objetos
- Diagrama de Secuencia
- Diagrama de Colaboración
- Diagrama de Transición de estados
- Diagrama de Actividad
- Diagrama de Componentes
- Diagrama de Distribución
- Diagrama de Paquetes

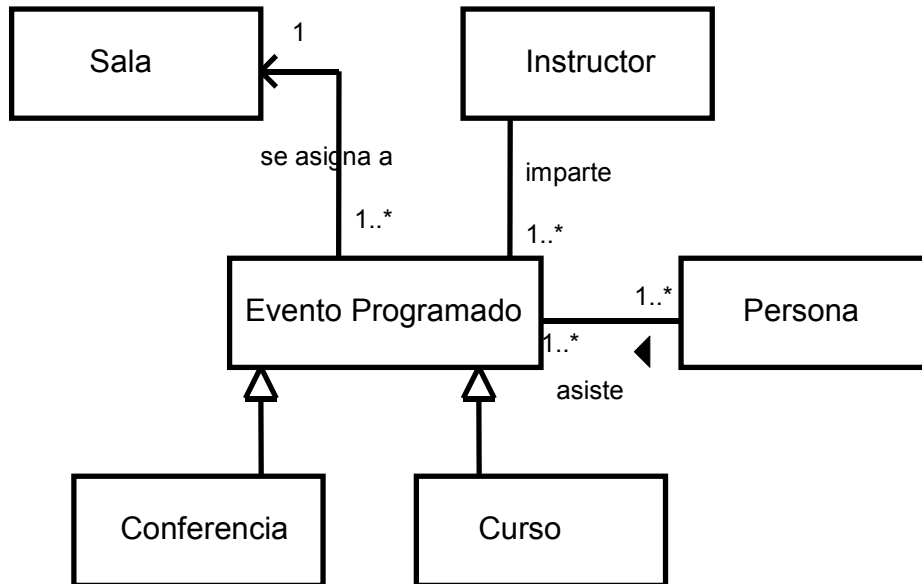
## 1.2.4 Diagrama de Casos de Uso



### Objetivos:

- Ilustrar los roles tomados en una situación por los usuarios (actores)
- Modelar los procesos que un actor requiere de un sistema desde su propia perspectiva

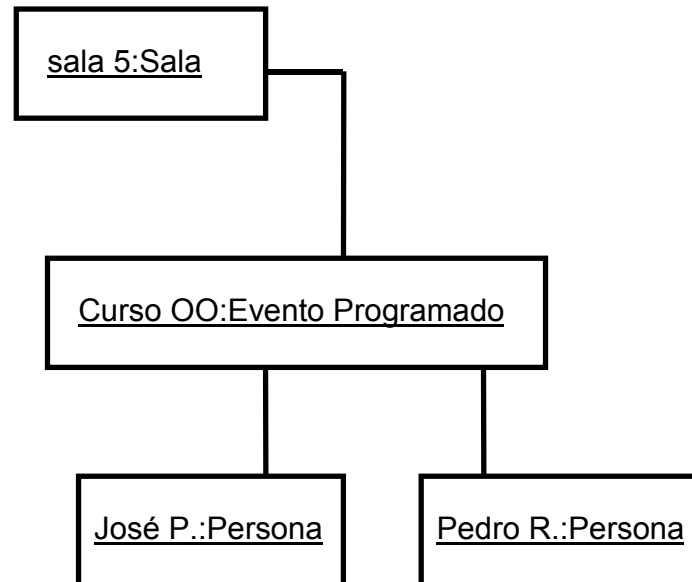
## 1.2.5 Diagrama de Clases



### Objetivos:

- Modelar las clases participantes
- Identificar sus atributos
- Identificar sus operaciones
- Mostrar relaciones estáticas entre clases

## 1.2.6 Diagrama de Objetos



### Objetivos:

- Mostrar una posible configuración del sistema

### Nota:

- *Corresponde con el diagrama de clases*



## 1.2.7 Diagramas de Interacción

### Objetivos:

- Mostrar comportamiento a través de un escenario

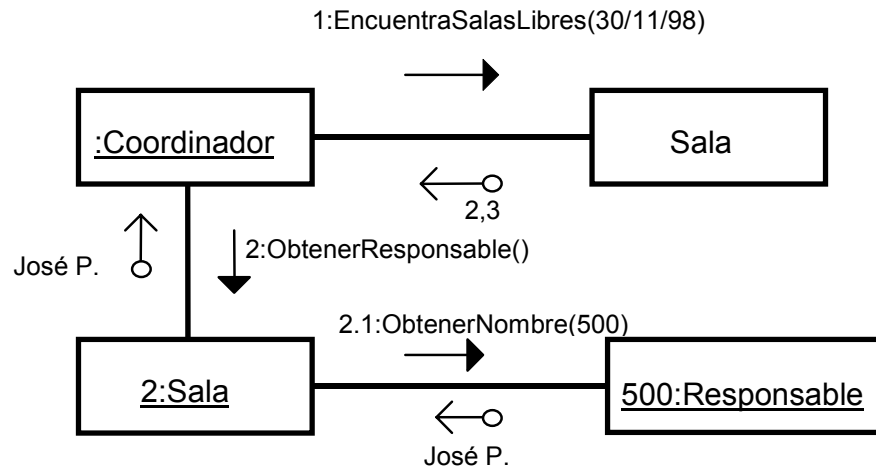
### Notas:

- Un escenario representa una situación típica de un proceso o función del sistema. *En un automóvil, por ejm., quedarse sin gasolina, o acelerar en 3ª velocidad.*
- Un diagrama de interacciones muestra interacciones de objetos en un escenario. Los objetos interactúan enviándose mensajes para la acción: *El chofer oprime el acelerador, que se comunica con sistema de inyección de combustible, que a su vez se comunica con el motor, etc.*
- UML soporta 2 formas de diagramas de interacciones:

*Diagramas de Colaboración*  
*Diagramas de Secuencia*

## 1.2.8 Diagrama de Colaboración

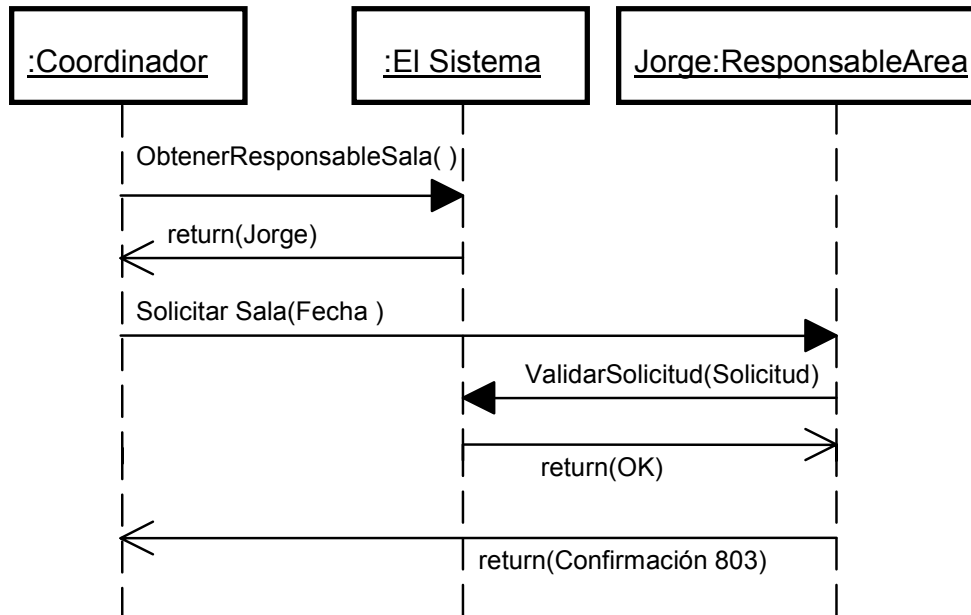
### Obtener Sala Disponible y Responsable



### Objetivos:

- Mostrar un grupo de objetos y ligas describiendo un escenario.
- Mostrar eventos (mensajes) pasando entre los objetos. Puede incluir retorno de resultados.
- Mostrar el orden relativo de los eventos. Los llamados anidados son mostrados por numeración anidada.

## 1.2.9 Diagrama de Secuencia



### Objetivos:

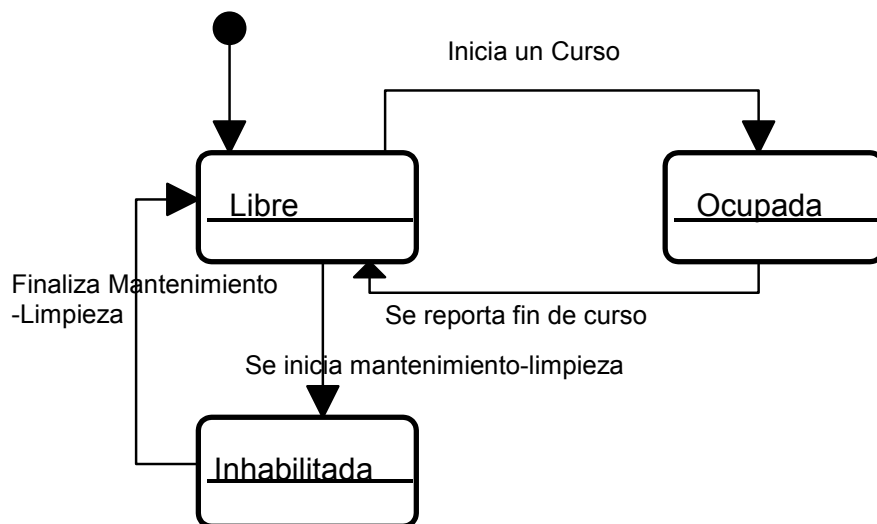
- Mostrar Eventos entre objetos en secuencia
- Mostrar Valores de retorno

### Nota:

*Es un diagrama de interacción “secuencial”*

## 1.2.10 Diagrama de Transición de Estados

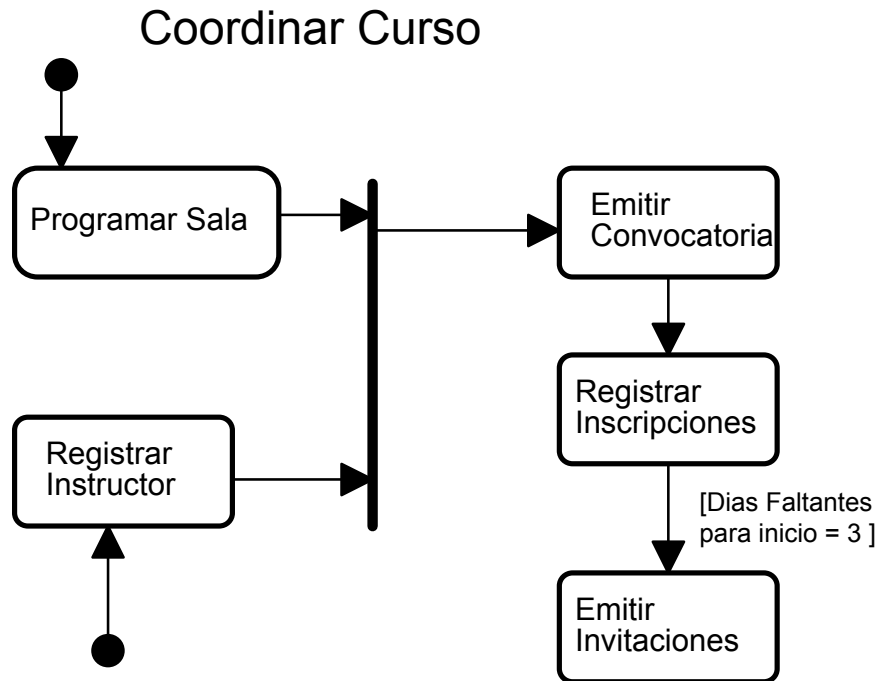
Sala 1 el día de hoy



### Objetivos:

- Mostrar estados. eventos y transiciones.
- Describir ciclos de vida de objetos
- Identificar operaciones relacionadas con estados y eventos

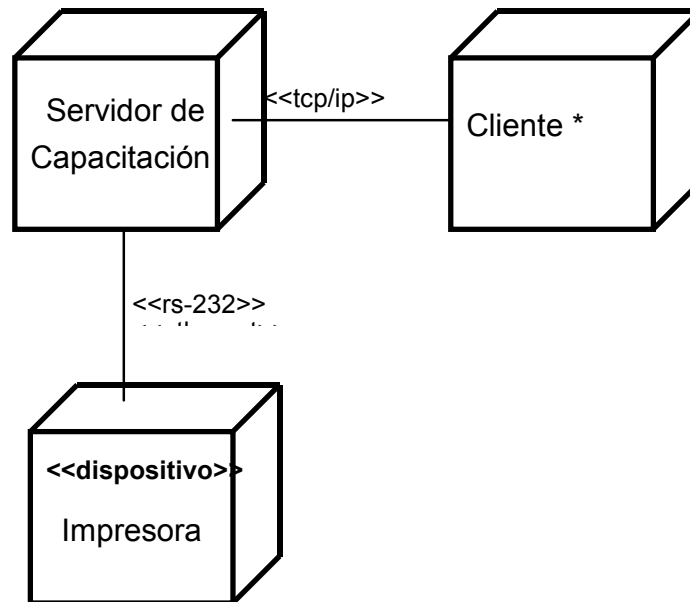
### 1.2.11 Diagrama de actividad



#### Objetivos:

- Mostrar relaciones temporales entre actividades
- Mostrar flujo de trabajo (workflow) entre casos de uso
- Identificar Paralelismo

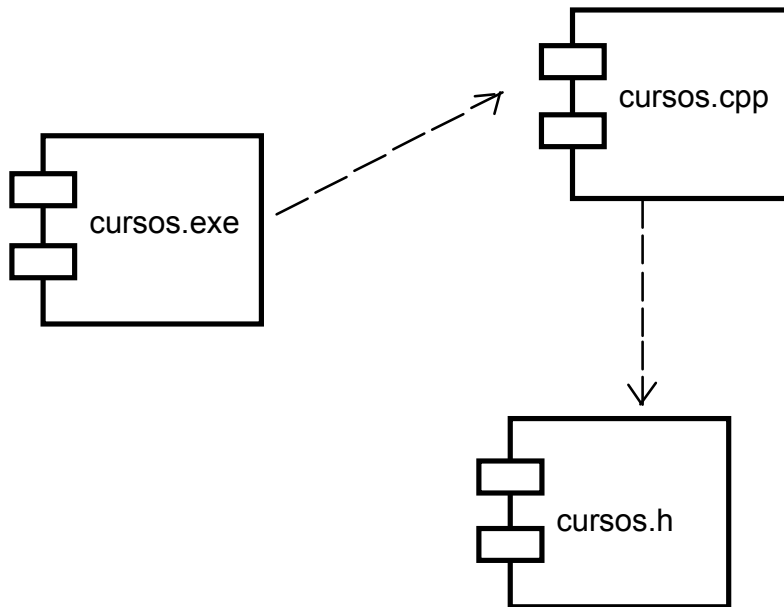
## 1.2.12 Diagrama de Distribución



### Objetivos:

- Mostrar la arquitectura de distribución
- Mostrar los nodos de hardware: procesadores y/o dispositivos
- Mostrar las conexiones físicas entre los nodos

### 1.2.13 Diagrama de Componentes

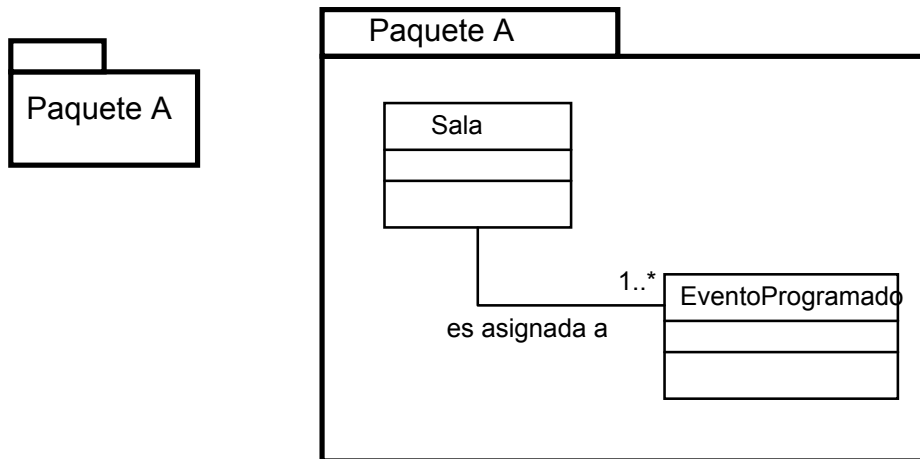


#### Objetivos:

- Representar paquetes físicos

Componentes reusables  
Archivos fuente  
Archivos ejecutables  
Librerías  
etc.

## 1.2.14 Diagrama de Paquetes



Es un grupo de elementos modelo (clases, otros paquetes, etc.)

### Objetivos:

- Agrupar diversos tipos de cosas: Módulos, modelos, subsistemas, grupos físicos, etc.





## **2 Clases, Objetos, Atributos, Operaciones y relaciones: Diagramas de Clases**

## 2.1 Conceptos Básicos

¿Qué es un objeto?

Una instancia de un concepto:abstracción o cosa

...de importancia en un ámbito

En un supermercado:

El cajero Jorge Hernández,. la caja 5, uno de los jabones del anaquel 24, el anaquel 24, la venta 2347, etc.

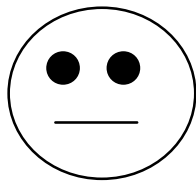
Hablando de programación de T.V.:

Los Simpson, canal 9, Televisa,  
el especial de los Beatles, Brozo, Jacobo, etc.

... y entonces...

¿ Qué es una clase ?

Clase es el nombre de un conjunto que agrupa objetos con las mismas características. Denota un concepto.



¿Un qué?

Por ejemplo, en el zoológico:

Animal

Denota un grupo de objetos (el tigre, el mono, el león, etc.) que están en jaulas para que la gente los conozca.

Comparten características y operaciones comunes:

¿Qué animal no come?

Ahora tú:....

Elabora una lista de objetos, luego identifica las clases a las que pertenecen.

Usa los ámbitos siguientes:

Un juego de Pac-Man

Una Agenda

Un método:

Hacer oraciones con sujeto, verbo y complemento de acciones que sucedan en el ámbito de interés.

Por ejemplo:

Pepe pateó la pelota

José Pérez vendió su datsun rojo al Sr. González

El monstruo persiguió al Pac-Man

...cada sustantivo es un candidato a objeto o clase

Atributo:

Una característica de un objeto que se aplica a todos los miembros de su clase:

Nombre

Dirección

Color

Sabor

etc.

Algunas veces se pueden identificar con la aplicación de los verbos ser y estar.

Por ejemplo:

El león es amarillo

El coche es grande

Ahora tú:

Adicionar atributos a cada clase encontrada en el ejercicio anterior.

Operación:

Función, transformación, acción que se puede aplicar a ó por los objetos de una clase.

Hay 2 niveles:

Nivel Clase (aplica sobre conjuntos)

Nivel Objeto (aplica sobre instancias)

Ejemplos de operaciones:

Clase	Operación	Nivel
Alumno	CalcularTotalAlumnos	Clase
Alumno	CambiarDirección	Objeto
Monstruo	Crear	Clase
Monstruo	Animar	Objeto

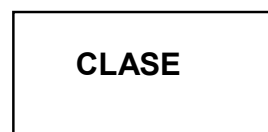
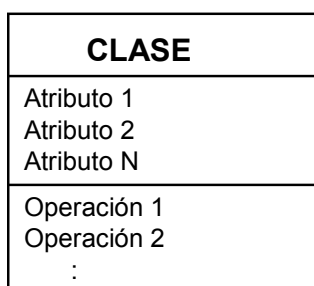
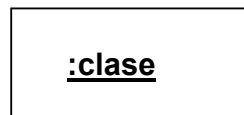
Ejercicio:

Adiciona operaciones a las clases obtenidas en el ejercicio anterior.



## Notación UML para objetos y clases:

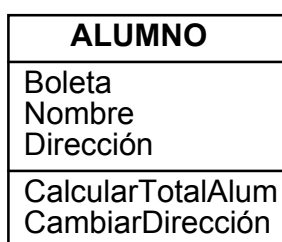
### OBJETO:



*Notación Abreviada*

Usando esta simbología, expresa las clases obtenidas en el ejercicio anterior bajo la Notación UML.

Por ejemplo:



CASO PRACTICO...

## 2.2 Cadenas (Links)

Link: Cohesión física o conceptual entre objetos

Ejemplo:

Bart es hijo de Homero

"Romance" fue grabado por Luis Miguel

Don King promueve a J.C. Chavez

Un link es un tuple, es decir, una lista ordenada de instancias de objetos:

(Bart, Homero)

("Romance", Luis Miguel)

(Don King, J.C. Chavez)

## 2.3 Asociaciones

Una asociación describe un grupo de links.

Por ejemplo:

Niño es hijo de Padre

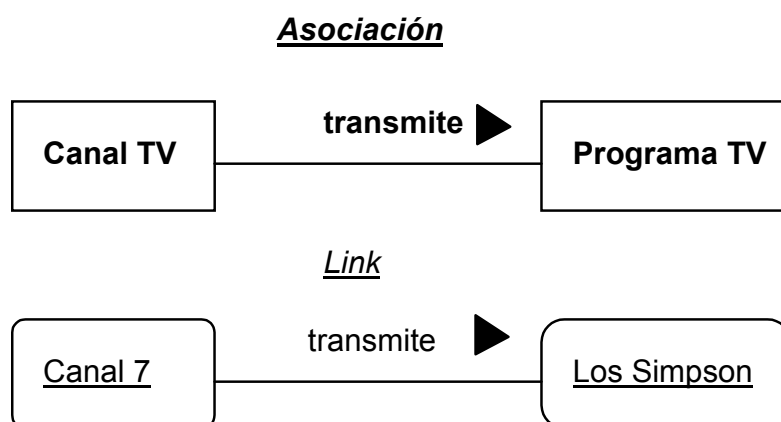
Disco fue grabado por Artista

Promotor promueve a Deportista

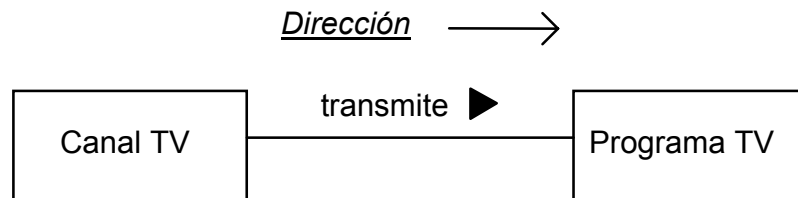
Son todos ejemplos de Asociaciones

### Notación:

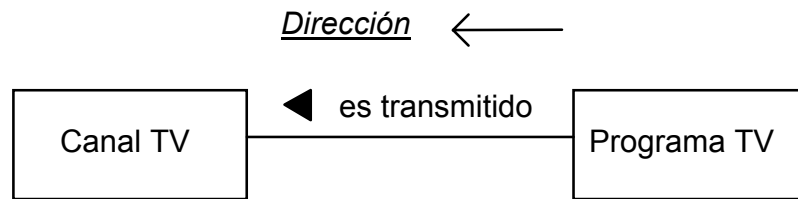
En UML una asociación ó link entre dos objetos o clases se representa por una linea uniendo los objetos ó clases.



Una asociación o link determina su dirección mediante un triángulo donde la flecha indica la dirección :



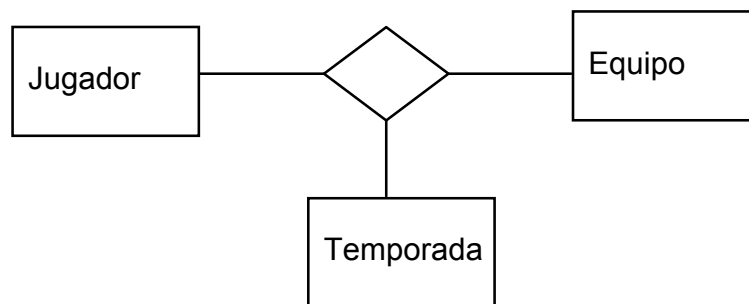
o bien



Usualmente las direcciones se leen de izquierda a derecha y de arriba a abajo.

## 2.4 Asociaciones Ternarias.

Asociación entre 3 clases, que no es posible dividir en asociaciones binarias sin perder información.



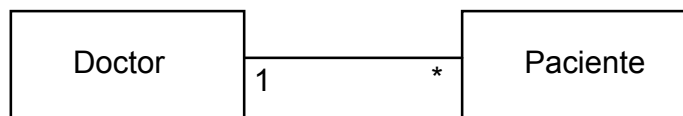
(Pelé, Santos, 69-70)  
(Pelé, Cosmos, 71-72)

## 2.5 Multiplicidad (Cardinalidad)

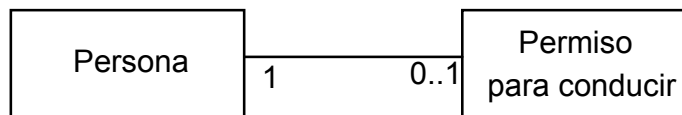
### Cardinalidades Básicas (Notación)

Muchos	*
0 ó 1	0..1
1	1
m a n	m..n

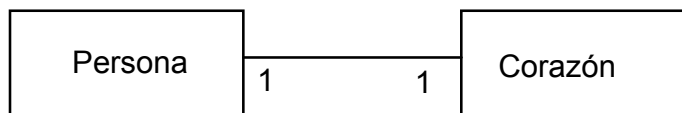
### 2.5.1 Ejemplos de cardinalidades.



Un doctor tiene 1 o muchos pacientes.

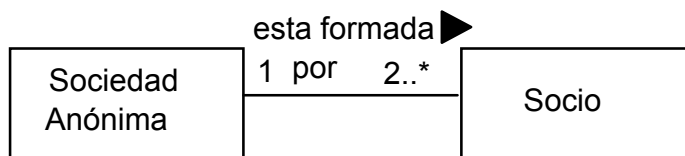


Una persona tiene 0 ó 1 permiso para conducir.

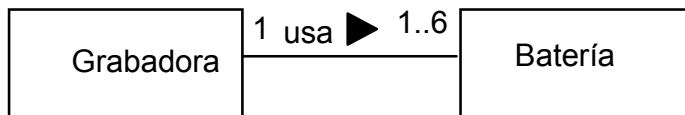


Una persona tiene 1 corazón.

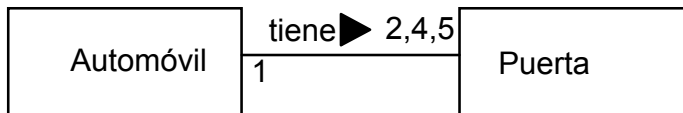
## 2.5.2 Ejemplos de cardinalidades.(cont.)



Una sociedad anónima está formada por 2 o más socios.



Una grabadora usa de 1 a 6 baterías.



Un automóvil tiene 2,4 o 5 puertas.

## Ejercicio 5

Coloque las asociaciones a los modelos desarrollados con anterioridad:

Un juego de Pac-Man

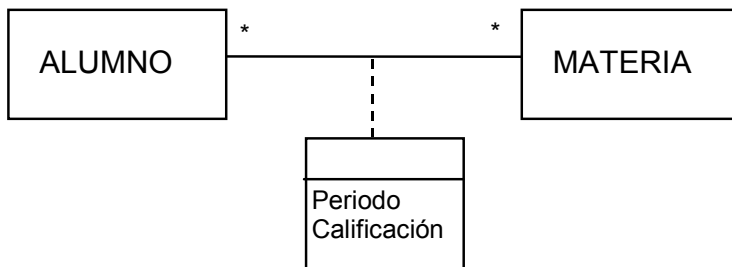
Una agenda



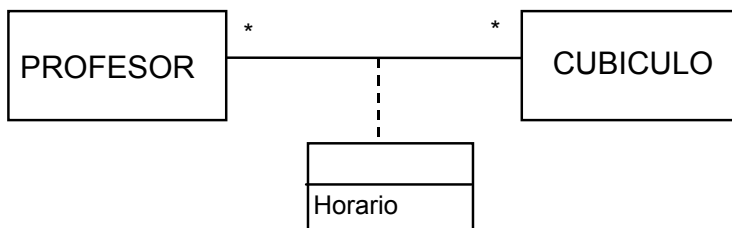
## 2.6 Atributos de una asociación.

Hay atributos que nacen de la relación entre objetos de distintas clases.

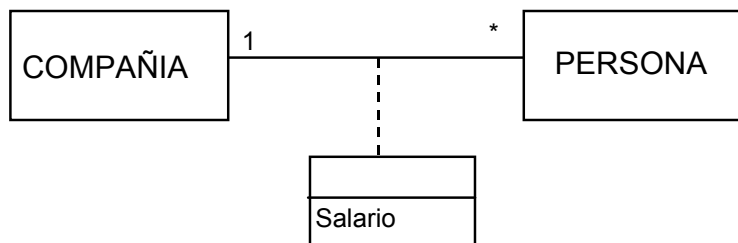
Por ejemplo:



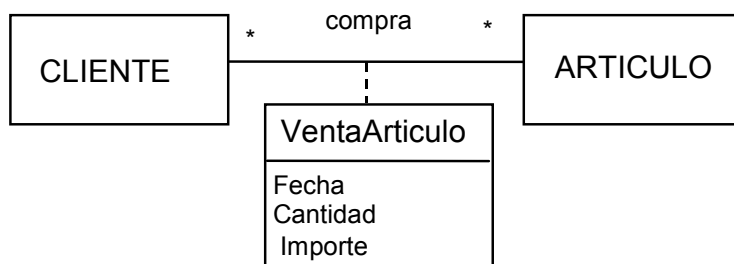
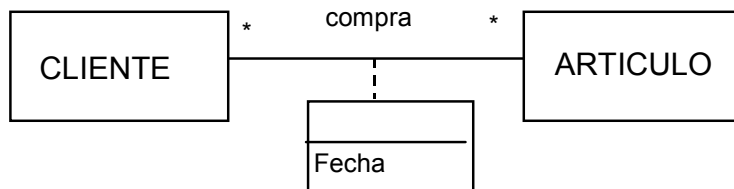
Las relaciones N:M son fuente común de asociaciones con atributos:



Sin embargo no es exclusiva de este tipo de asociaciones la existencia de atributos:

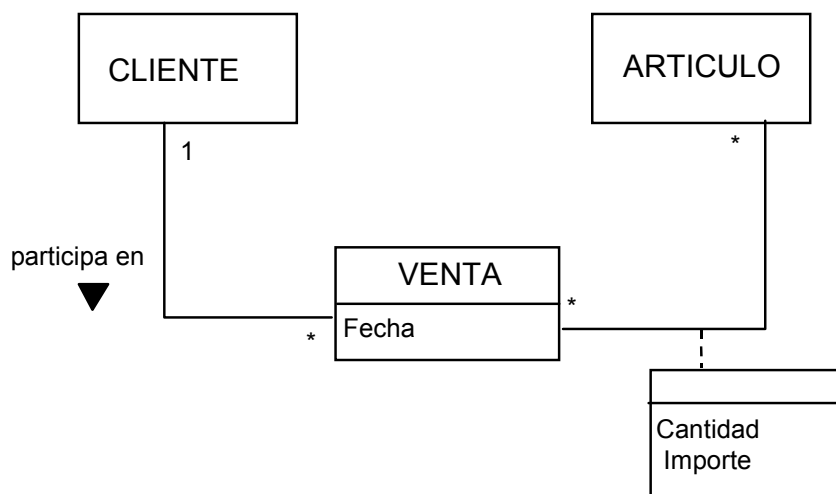


En la mayoría de casos, las asociaciones de N:M denotan eventos. En este caso, generalmente conviene expresar la asociación como una clase.



¿Qué problema tiene?

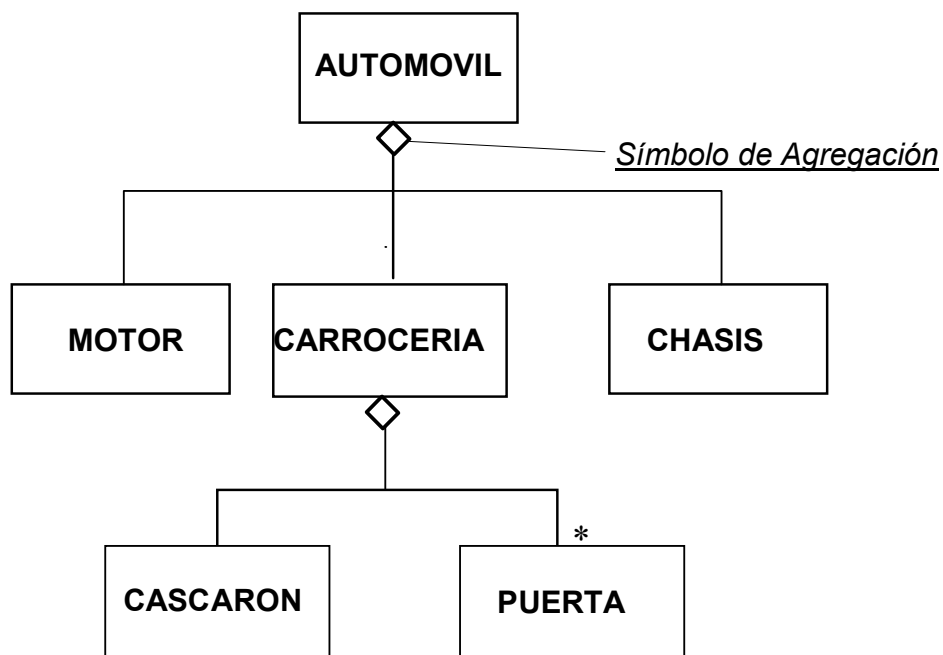
Otra manera de expresarlo es:



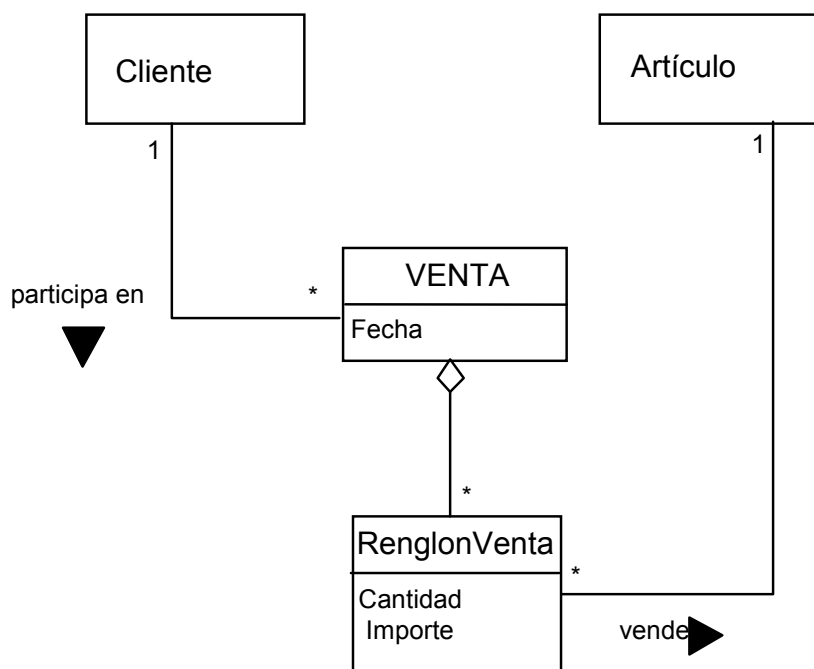
¿Ya está bien ?

## 2.7 Agregación (Composición)

Existe un tipo de asociación llamado Agregación (o composición) y sirve para expresar relaciones de "todo y partes"



Regresando al ejemplo previo:



## Ejercicio 6:

a) Elabore 3 ejemplos de asociaciones N:M y refínelas del mejor modo. Discuta los resultados.

b) Resuelva las relaciones N:M siguientes:

Alumno-Materia

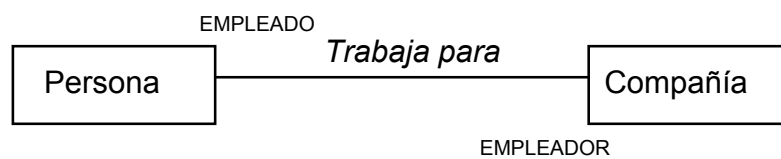
Profesor-Materia

Salón-Grupo

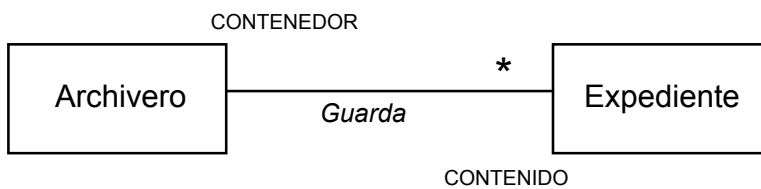
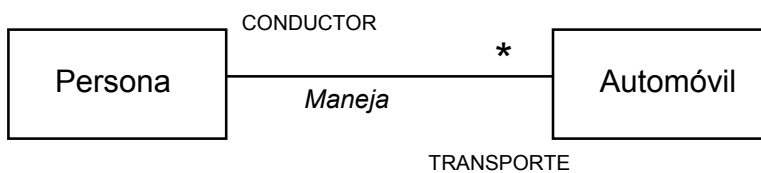
¿Es aplicable el refinamiento a relaciones ternarias, por ejemplo cliente-vendedor-auto ?

## 2.8 Roles

Cada extremo de una asociación denota un rol. Dan un punto de vista "desde el objeto".



Ejemplos de Roles:



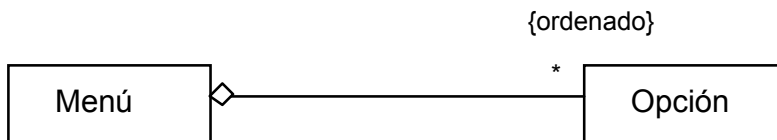
## 2.9 Ordenamiento

***Cuando requerimos expresar que una clase debe guardar un orden, usamos el símbolo:***

{ ordenado }

Esto en la clase de interés.

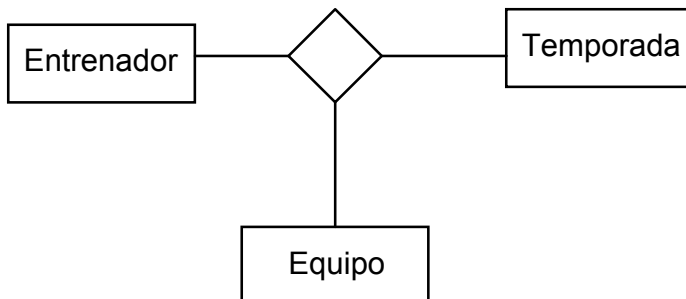
Ejemplo:





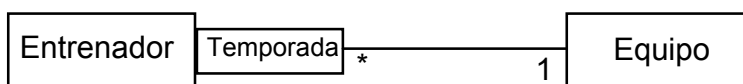
## 2.10 Calificación

Dada una relación ternaria:

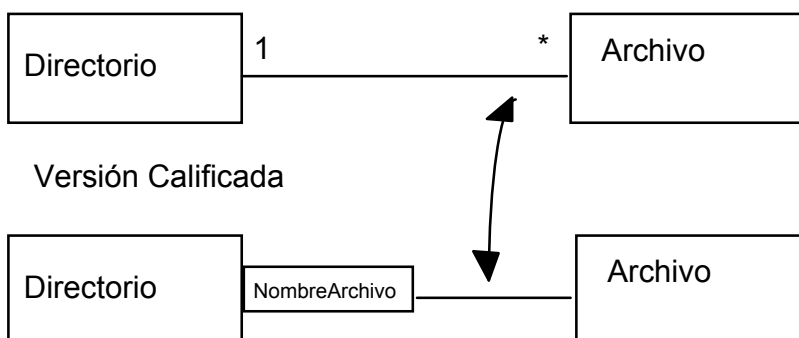


Se puede expresar que una asociación se da a través de un atributo dado. En el ejemplo anterior, el entrenador + la temporada está asociado con un solo equipo.

El atributo "temporada" se denomina "calificador" y a la asociación resultante "relación calificada"



Ejemplo de calificación.



## **2.11 Generalización y Herencia.**

Son abstracciones para compartir similitudes entre clases y al mismo tiempo conservar sus diferencias.

Por ejemplo:

Publicación tiene :

- Título
- Fecha de Impresión
- Índice

Un libro tiene:

- Autores
- Tema principal

Una revista tiene :

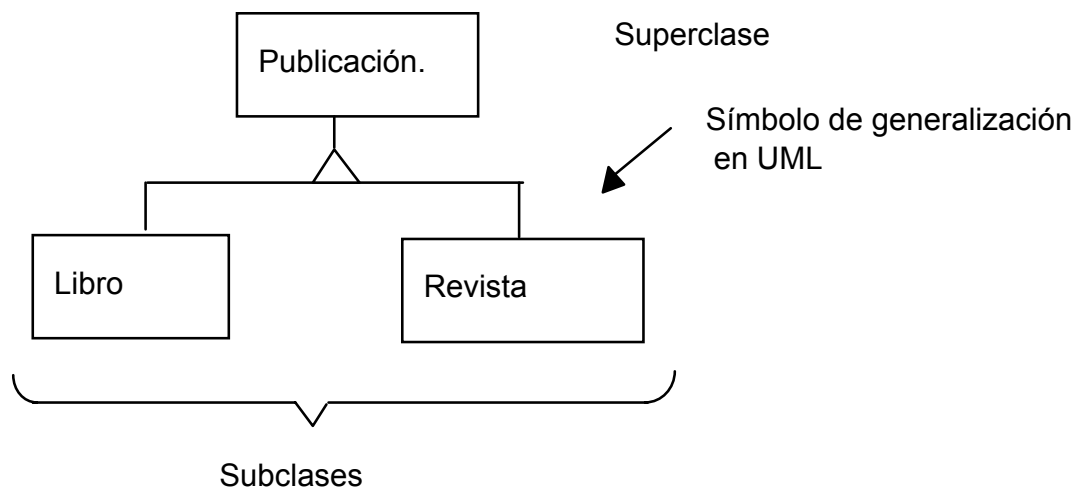
- Artículos
- Periodicidad
- Publicidad

## Generalización.

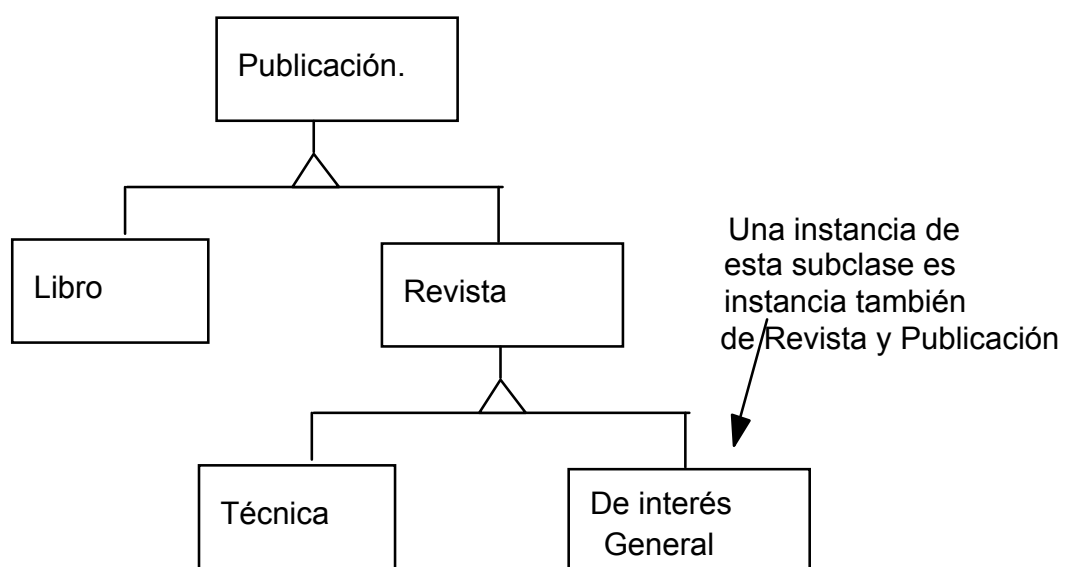
Es la relación entre una clase y una o más versiones refinadas de ella.

La clase que se está refinando es llamada superclase, mientras que la versión refinada es llamada subclase.

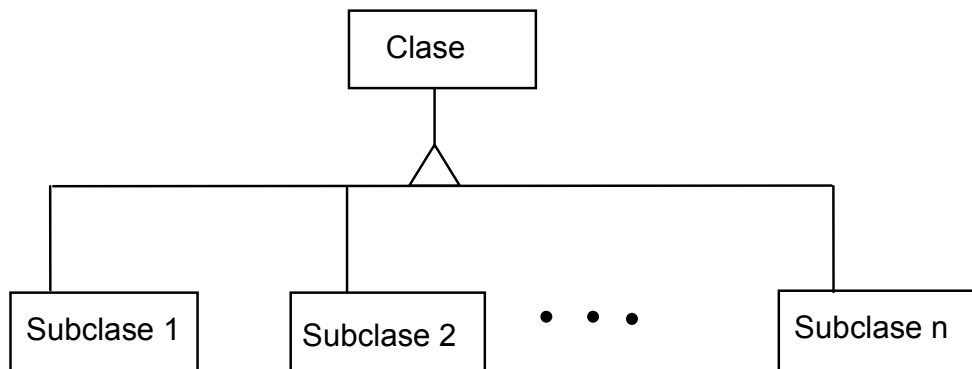
Así:



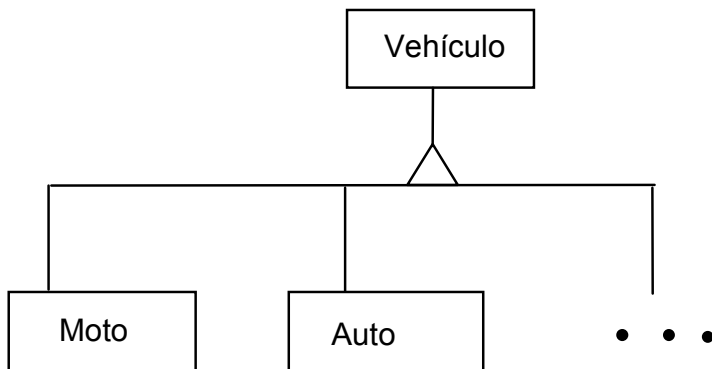
La generalización es una relación transitiva con un número de niveles arbitrarios



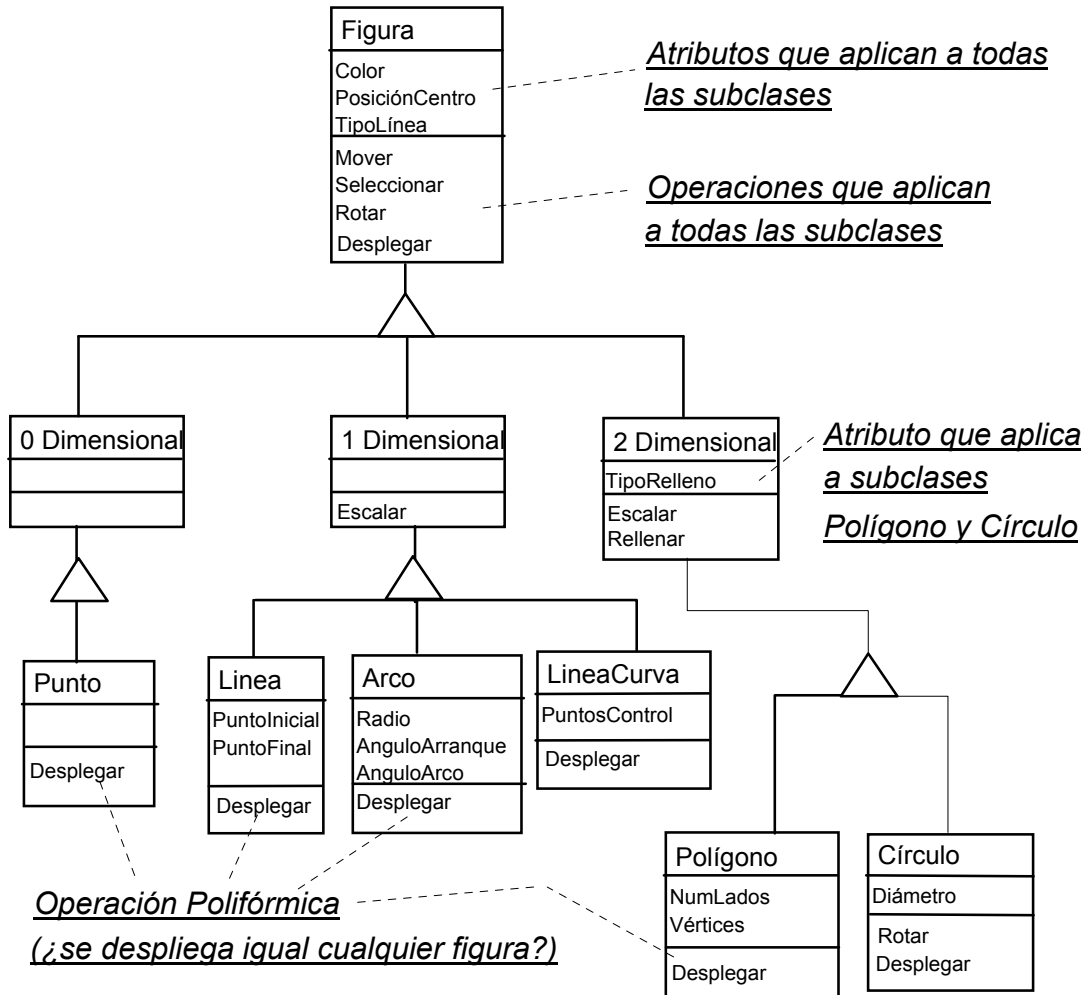
## Notación de Generalización



## Ejemplo



# Herencia , Polimorfismo



Herencia implica compartir atributos y/u operaciones

Polimorfismo implica operaciones que se implementan de modos distintos

## 2.12 Caso Práctico

## **CASO DE ESTUDIO**

Moon Launch Enterprise vende viajes a la Luna, ya sea de placer o de negocios en el año 2010.

Los usuarios quieren un sistema que soporte la calendarización, horarios y manejo en general de tales viajes. Iremos dando mayores detalles de la empresa como sea necesario para soportar el caso de estudio.

Su primer trabajo de modelamiento para este caso es con el jefe de reservaciones a pasajeros de Moon Launch Enterprise. Un agente, es la persona responsable de colocar reservaciones para los pasajeros de negocios (denominados internamente ejecutivos) o de recreo (denominados turistas). Desde luego, un pasajero dado puede ser ejecutivo para varios viajes y turista para otros. Los detalles siguen a continuación.

### **1. Información de pasajeros.**

a. Moon Launch asigna a todos los pasajeros números de identificación dentro de sus respectivos países (por ejemplo, en Estados Unidos, el número de seguro social). Los números usados para cada país varían en el tamaño y pueden incluir caracteres especiales.

b. En un futuro puede necesitarse registrar la edad de los pasajeros.

c. Cada pasajero tiene un apellido, primer nombre, segundo nombre y título (Sr., Sra., Sita., etc.).

d. Cada pasajero tiene una dirección para envío de factura. En el caso de un ejecutivo, la dirección de envío de factura es su oficina. La dirección para envío de boletos puede ser su residencia personal o bien su oficina. En el caso de turistas, tanto los boletos como la factura son enviados a la residencia personal.

e. Todos los pasajeros tienen descuentos asignados por Moon Launch Enterprise. Para ejecutivos, el tipo de descuento es dado de acuerdo a la empresa a la que éste pertenezca.

### **2. Información sobre reservaciones.**

a. El agente de reservaciones registra vuelo, tipo de pago (por ejemplo, Tarjeta World Express), precio de boleto y fecha de entrada de la reservación. El agente también registra su propio número de identificación de empleado de Moon Launch con la reservación.

b. Para calcular los precios de los boletos, el agente checa el tipo de descuento del pasajero y aplica el descuento apropiado al precio base.

Tipo de descuento	Descuento
A	-20%
B	-10%
C	0%
D	10%

Nota: El tipo de descuento D es para pasajeros indeseables.

c. Todos los lugares son para no fumadores y no tienen ventanas.

### **3. Consolidado de entradas.**

Moon Launch programa 4 vuelos por día. Dos a la Luna y dos de regreso.

Hay 100 lugares disponibles en cada vuelo. En promedio 75% son ocupados, otro 10% son ocupados y luego cancelados. También 8 miembros de la tripulación atienden cada vuelo.

Aproximadamente 50% de los pasajeros en cada vuelo son turistas y vuelan por Moon Launch un promedio de 1.05 veces por año (viaje redondo, desde luego). El resto son ejecutivos y vuelan 3 veces por año en promedio.

Moon Launch negocia regularmente con cerca de 1000 empresas. De cada una, un promedio de 6 empleados por año vuelan irregularmente.

Moon Launch usa 4 tipos de descuento A, B, C y D.

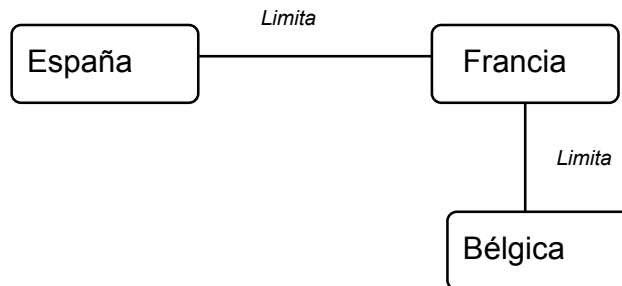


## 2.13 Tips

- Tratar de conservar el modelo simple
- Usar nombres adecuados
- Intentar resolver relaciones complejas.
- No usar niveles de generalización demasiado profundos.
- Intentar obtener revisiones de otros.
- Documentar los modelos.
- No intentar usar todos los constructores de UML (Son una idealización).

## 2.14 Ejercicios

1. Prepare un diagrama de clases para el siguiente diagrama de objetos:



2. Prepare diagramas de objetos mostrando cuando menos 10 relaciones entre las siguientes clases. Incluya asociaciones, agregaciones y generalizaciones. Use relaciones calificadas donde sea posible. No es necesario que exprese atributos u operaciones. Nombre las asociaciones donde sea necesario, adicione las clases que necesite.

- escuela, patio de recreo, director, pizarrón, salón, libro, alumno, profesor, cafetería, baño, computadora, escritorio, pupitre, regla, puerta, cancha.
- castillo, foso, puente levadizo, torre, fantasma, escaleras, calabozo, piso, corredor, habitación, ventana, piedra, señor, señora, cocinero
- expresión, constante, variable, función, lista de argumentos, operador relacional, término, factor, operador aritmético, declaración, programa.
- file system, archivo, directorio, nombre archivo, archivo ASCII, archivo ejecutable, disco, drive, pista, sector.
- automóvil, motor, rueda, freno, calavera, puerta, batería, mofle, tubo de escape.
- pieza de ajedrez, rango, cuadro, fila, tablero, movimiento, posición, secuencia de movimientos
- fregadero, congelador, refrigerador, mesa, luz, interruptor, ventana, alarma humo, alarma robo, armario, pan, queso, hielo, puerta, cocina.

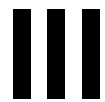
3. Elabore el modelo de objetos para un juego de dominó.

4. Para el caso de Moon Launch, ¿Cómo se alteraría el modelo de objetos si un pasajero fuera siempre un ejecutivo o un vacacionista pero nunca los dos?

5. Para el caso de Moon Launch, ¿Cómo se alteraría el modelo si se considerara la clase "boleto", donde un boleto puede o no ser resultado de una reservación?

6. Los agentes de reservaciones de pasajeros representan solamente uno de los grupos de usuarios de Moon Launch. Otro grupo es el de control de equipaje. Cuando los pasajeros registran su equipaje, un encargado registra la identificación del pasajero (pais y código de pasajero), el número de vuelo, el destino, la fecha de partida, y un número de equipaje único (generado por el sistema). Siempre que una maleta se extravía, otro empleado llena una forma de "equipaje

perdido", la cual contiene la siguiente información: vuelo, fecha de partida, identificación del pasajero, nombre del pasajero, dirección, teléfono, numero de equipaje, descripción de la maleta, lista de contenido de la maleta (numerada secuencialmente) y el valor de la maleta incluyendo contenido. Elabore un modelo de objetos para el control de equipaje.



## **3 Especificación de requerimientos: Casos de Uso**

### 3.1 Casos de Uso y diagramas de casos de uso

Técnica desarrollada por Jacobson en OOSE.  
Se basa en los conceptos de actor y de caso de Uso.

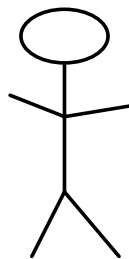
### 3.2 Actor

***Un actor es un agente externo al sistema que interactúa con él.***

***Los actores son entidades en la frontera de nuestro sistema. Interactúan con el sistema pero no son parte de él.***

***Pueden ser personas, otros sistemas, dispositivos, etc.***

***Se representan por la figura:***



***Representa un rol desempeñado por un usuario.***

## **Caso Ejemplo**

Debemos escribir un sistema de procesamiento de pedidos para una compañía que vende artículos de oficina.

Los clientes hacen sus pedidos a los agentes vía telefónica; esos agentes entonces capturarán los pedidos en nuestro sistema. Un pedido puede incluir varios artículos y un cliente puede pedir más de un ejemplar de cada artículo. (Por ejemplo, con un solo pedido un cliente podría pedir un cartucho de toner, una caja de papel y tres cuadernos). Cada artículo es referido por un número de artículo. Además nuestro sistema debe generar un número único para cada pedido recibido.

En algunos casos, se podría requerir ofrecer como un solo artículo una colección de otros artículos. Por ejemplo, podríamos ofrecer un precio especial para un paquete de impresión que consiste de una impresora, un cartucho de toner y una caja de papel. En otras ocasiones se podrían agrupar colecciones dentro de otras colecciones; por ejemplo, podríamos desear ofrecer un paquete de cómputo que incluya el paquete de impresión (arriba mencionado), una computadora y un monitor.

Cuando un cliente hace su pedido, debe indicar una tarjeta de crédito o de débito con la que el pedido será pagado. Nuestro sistema deberá solicitar una autorización de un banco por la cantidad que importe el pedido. Diferentes tipos de tarjetas pueden ser autorizados por diferentes bancos. El sistema deberá registrar la fecha y hora de la autorización y cualquier número de autorización proporcionado por el banco.

Un artículo del pedido se considera satisfecho cuando se ha embarcado. (El pedido se considera satisfecho cuando todos sus artículos a su vez han sido satisfechos). Los artículos de cada pedido pueden ser embarcados individualmente o en grupos. Cuando uno o más artículos están disponibles en inventario, el sistema debe informar a "Embarques" (un sistema externo) para que esos artículos sean enviados al cliente. Embarques enviará un acuse de recibo cuando los artículos sean embarcados. Debemos de conservar un registro de que artículos son embarcados, cuando y a quien.

Un cliente puede cancelar un pedido completo o un artículo en particular de una orden. Un pedido de un artículo en particular puede ser cancelado hasta antes de que sea embarcado. Cancelar un pedido completo implica cancelar todos los artículos del pedido que aún no se han embarcado. Cuando cualquier parte del pedido es cancelado, debemos acreditar a la cuenta del cliente el precio de compra de los artículos cancelados. Además la cancelación de un pedido requiere regresar los artículos al inventario.

El cliente también puede devolver artículos por rechazo (falta de calidad, errores, etc.). En tal caso, los artículos llegan a Embarques. Embarques informará al sistema de la devolución. El sistema deberá tener un registro de que porción del pedido ha sido devuelta. Los artículos devueltos deberán ser regresados al inventario y deberemos acreditar el importe correspondiente a la cuenta del cliente de donde se pagó el pedido.

El sistema deberá conservar información de cada artículo en el inventario. En particular, la existencia actual, la existencia mínima y el punto de reorden de cada artículo. Además cuando un artículo no está disponible, el sistema deberá ser capaz de dar la fecha estimada en que el artículo estará nuevamente disponible.

Cuando la cantidad en inventario de un artículo cae debajo de la existencia mínima, se deberá enviar un mensaje al almacén para que se pida a nuestro proveedor. Este mensaje incluye el número de artículo y la cantidad a pedir (que se obtiene a partir del punto de reorden del artículo).

El almacén acusará de recibido el mensaje con una fecha estimada en la que los artículos llegarán al almacén.

## Los Actores del caso

***Los actores son aquellos entes con quienes el sistema interactúa.***

***¿Quién podemos considerar que está “fuera” del sistema?***

Un actor representa un **rol**

***Pregunta:***

***En una empresa con 3 empleados ¿Cuántos actores hay ?***

**Ejercicio:**

***Para el caso anterior, elabore una lista de “entes” externos al sistema. Discuta sus resultados.***



### 3.3 Casos de Uso

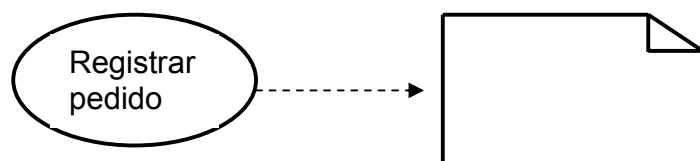
Un caso de uso es análogo a una función del sistema.

- Describe un posible uso del sistema por un actor

“Un caso de uso es un conjunto de secuencias de acción que un sistema ejecuta que dan un resultado tangible y de valor para un actor en particular”

- Un caso de uso en UML es un elemento a modelar

Se describe con una elipse etiquetada. Se le puede adicionar una especificación textual.



### 3.3.1 Identificación de los casos de Uso

¿Cómo identificar los casos de uso de nuestro problema?

Un enfoque es revisar la manera en que cada actor usa el sistema.

- ¿Qué le pide el sistema a un actor que haga?
- ¿Qué consultas al sistema hace el actor?
- ¿ De qué acciones internas el sistema informa al actor ?

Ejercicio:

Identifique y describa los casos de uso del caso práctico.

### 3.3.2 Diagramas de Casos de Uso

***Un diagrama de caso de uso en UML muestra los casos de uso de una aplicación.***

- La aplicación puede ser un sistema completo o un subsistema.

Este es descrito como un diagrama de paquete, con nombre. Se empieza con una aplicación, el sistema completo

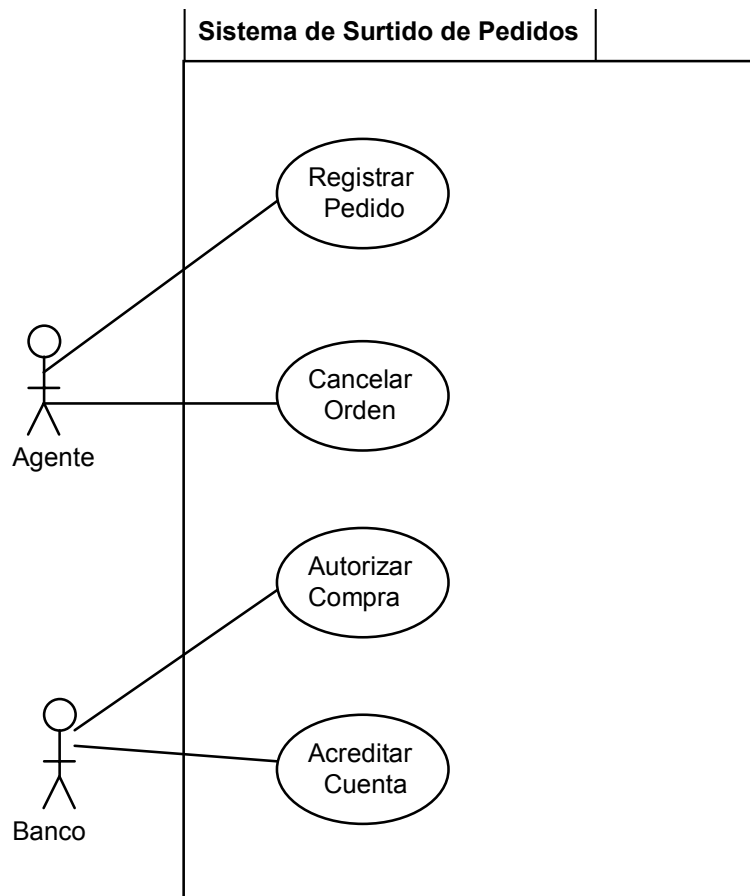
- Cada caso de uso aparece dentro del sistema que lo manipula.

Esto es, aparece dentro del diagrama de paquete apropiado.

- El diagrama incluye también a los actores involucrados.

Los actores estarán asociados con los casos de uso con los que interactúan.

## Ejemplo (fragmento)



## Ejercicio:

Completar el diagrama con el resto de casos de uso encontrados.

### 3.4 Relaciones en un caso de uso

¿Como se relacionan los casos de uso?

#### 3.4.1 Relación de Uso

Considerando al actor “Agente”, tenemos 4 casos de uso:

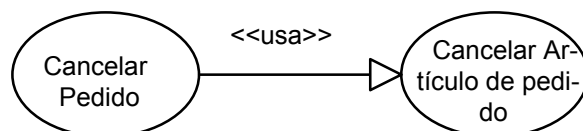
- Hacer un Pedido
- Cancelar Pedido
- Cancelar Artículo del Pedido
- Devolver Artículo del Pedido

Encontramos que dos de los casos de uso están relacionados.

Cancelar Pedido debe Cancelar artículos del Pedido

Esto es, “cancelar pedido” “usa” el comportamiento de “Cancelar Artículo del Pedido”

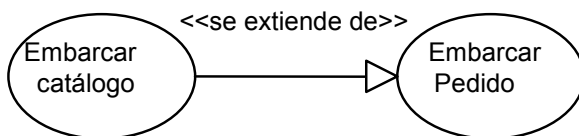
Este es un ejemplo de la relación “usa” entre casos de uso



### 3.4.2 Otra relación entre casos de uso: Extensión

Un caso de uso puede extender a otro

Un caso de uso se extiende de otro cuando se requiere expresar algo que se hace en un escenario atípico: por ejemplo, bajo una política o un error



#### Ejercicio

1. Identifique las relaciones entre los casos de uso del ejercicio
2. ¿Existe alguna manera de expresar secuencia entre nuestros casos de uso?
3. Agrupe los casos de uso en subsistemas (use diagramas de paquete para ello).
4. ¿ Es posible manejar casos de uso por niveles ?

### 3.5 Casos de uso y escenarios

- Un escenario es una instancia de un caso de uso
- Pensamos un caso de uso en términos generales
  - Un caso de uso describe funciones del sistema
  - Puede tener varias rutas de ejecución de interés
  - Este es descrito típicamente en términos de clases, nombres de parámetros, etc.
- Pensamos un escenario en términos muy específicos
  - Un escenario describe un conjunto de acciones e interacciones.
  - Un escenario típicamente describe una ruta de ejecución particular (aunque UML permite bifurcación y concurrencia)
  - Un escenario es descrito en términos de objetos, valores de parámetros, etc.
  - Un diagrama de interacción muestra las interacciones de objetos en un escenario.

### 3.5.1 Ejemplo de un caso de uso

Considere el manejo de fondos de un pedido

***El caso de uso de la autorización de la compra describe una función general***

Un pedido registrado debe tener fondos, el costo total de la orden debe ser obtenido de una tarjeta de crédito o débito. El banco es contactado, se le da el número de tarjeta y el importe de compra y se le solicita la autorización del importe.

***Este caso de uso tiene al menos dos rutas de ejecución de interés***

- El banco autoriza la cantidad
- El banco niega la autorización (en cuyo caso la orden debe ser cancelada)

Este caso puede tener rutas adicionales

- ¿Qué pasa si el banco no puede ser contactado?
- Tal vez tipos de tarjetas diferentes tienen modos de solicitar autorización distintos.



### **3.5.2 Ejemplo de un escenario**

Considere escenarios para el manejo de fondos de un pedido.

#### **Escenario donde la compra es autorizada.**

- 1.Una orden pide al objeto autorización de crédito autorizar una compra para la orden número 312799, contra el número de cuenta 8144-22-7999 por 22.64.
- 2.El objeto autorización crédito crea una instancia de requisición de autorización con el número de orden, cuenta e importe
- 3.El objeto autorización de crédito pide a la instancia de requisición de autorización su ejecución.
- 4.La instancia de requisición de autorización contacta al banco apropiado y hace la requisición. El banco indica que la requisición ha sido autorizada al regresar un número de autorización; el 2345.
- 5.La instancia de requisición de autorización salva el número de autorización y hace su campo “autorizado” Verdadero.
- 6.La instancia de requisición de autorización devuelve “Cierto” al objeto Autorización crédito, que devuelve “cierto” a la instancia de Orden.

## **Ejercicio**

Diseñe escenarios y diseños de interfaz para los casos de uso del ejemplo.

¿Puede generar al mismo tiempo un diagrama de clases?

### **3.6 Sumario de uso de los casos de uso**

- Identificamos clases basados en los casos de uso
- Identificamos las rutas de ejecución de interés de cada caso de uso.
  - Especificamos una instancia de caso de uso para cada uno
  - Dibujamos un diagrama de interacciones para cada instancia del caso de uso (en el nivel apropiado de detalle)
- Identificamos subsistemas basados en los casos de uso
  - Podemos aplicar los diagramas de paquetes para ello

# **IV**

## **4 Especificación de Transacciones: Diagramas de Interacciones**

## 4.1 Diagramas de Colaboración

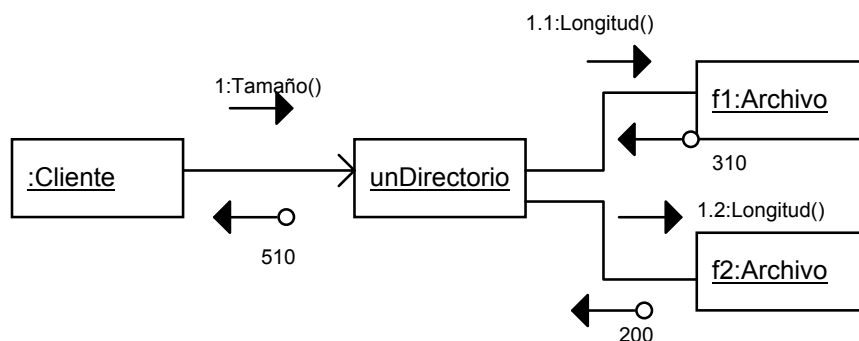
Un diagrama de colaboración es un diagrama de interacción “espacial”

- Muestra un conjunto de objetos y ligas (como en un diagrama de objetos)
- Muestra eventos (mensajes) pasando entre esos objetos

*Puede también incluir resultados siendo retornados*

- Muestra (por numeración) el orden relativo de esos eventos

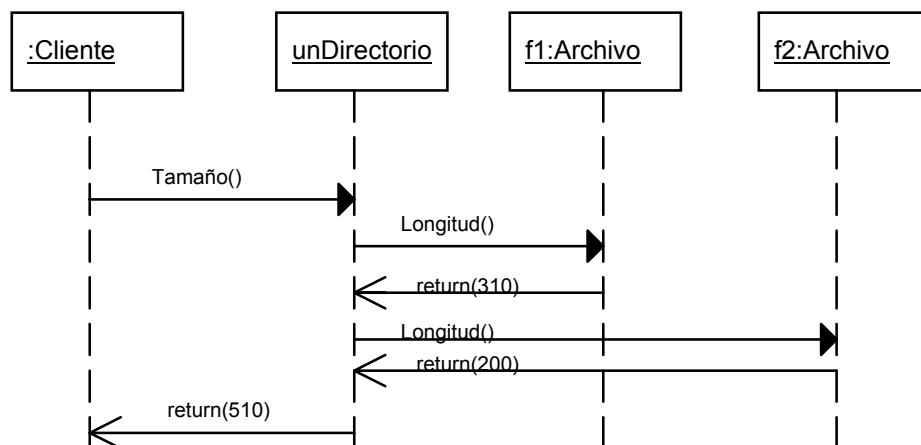
*Llamados anidados se muestran por numeración anidada*



## 4.2 Diagramas de Secuencia

Un diagrama de secuencia es un diagrama de interacción “temporal”.

- Los objetos son descritos usando líneas verticales
- Los eventos enviados entre objetos son descritos como arcos dirigidos
  - *El nombre del evento (y datos) aparecen sobre el arco*
  - *Los llamados tienen cabeza solida, devoluciones de valores (return) son flecha simples sin relleno*
- El tiempo pasa como vamos de arriba hacia abajo



### 4.3 Notación común en diagramas de interacción

UML incluye notación para:

- Creación y eliminación de objetos
- Iteración
- Bifurcación
- Límites de tiempo
- Ilustrar como se establecen ligas

### 4.4 Creación y Eliminación en Diagramas de colaboración

- A un objeto creado durante un escenario se le da la restricción de “nuevo”
- A un objeto destruido durante un escenario se le da la restricción de “destruido”
- A un objeto que es tanto creado como destruido durante un escenario se le puede dar la restricción de “nuevo, destruido” o la restricción de “transitorio”

*(La ligas pueden ser anotadas del mismo modo)*

<u>unObjeto</u> {nuevo}
----------------------------

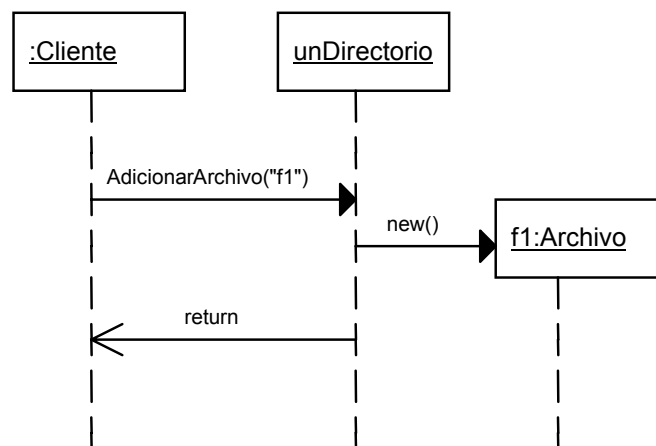
<u>unObjeto</u> {destruido}
--------------------------------

<u>unObjeto</u> {nuevo,destruido}
--------------------------------------

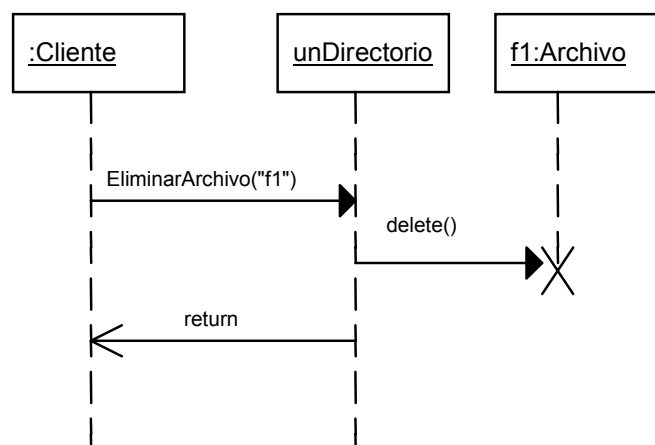
<u>unObjeto</u> {transitorio}
----------------------------------

## 4.5 Creación y Eliminación en Diagramas de secuencia

Cuando un objeto es creado por un mensaje, la flecha del mensaje llega al rectángulo que representa al objeto en sí.

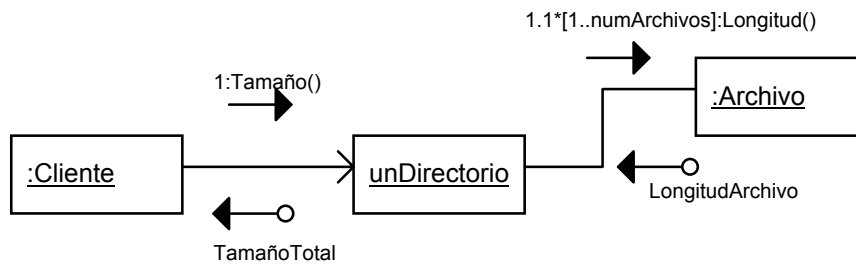


Cuando un objeto es borrado por un mensaje, la flecha del mensaje está dentro de una “X” sobre la línea del objeto (y la línea finaliza ahí)



## 4.6 Iteración en un diagrama de colaboración

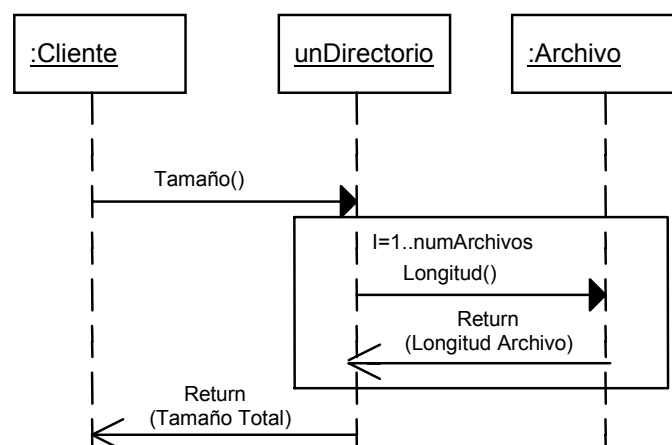
El número de evento incluye un \* con el rango de iteración entre corchetes



## 4.7 Iteración en un diagrama de secuencia

Las porciones repetidas son limitadas de un mismo modo (por ejemplo en una caja)

La iteración se muestra sobre la construcción limitante.

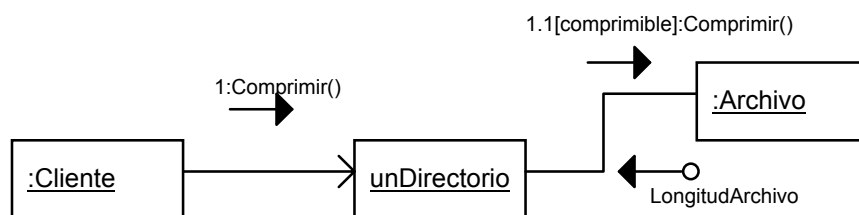




## 4.8 Bifurcación en diagramas de colaboración

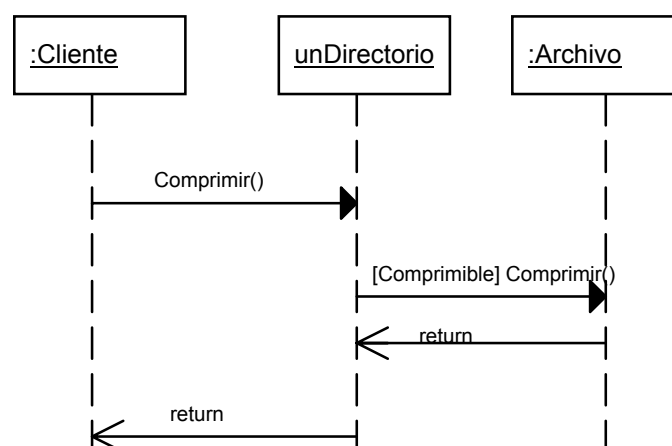
Las condiciones se muestran entre corchetes

(Se supone que un directorio se comprime a sí mismo solo si su campo “comprimible” es verdadero)



## 4.9 Bifurcación en diagramas de secuencia

Un evento incluye una condición entre corchetes



## 4.10 Tiempos límite en diagramas de secuencia

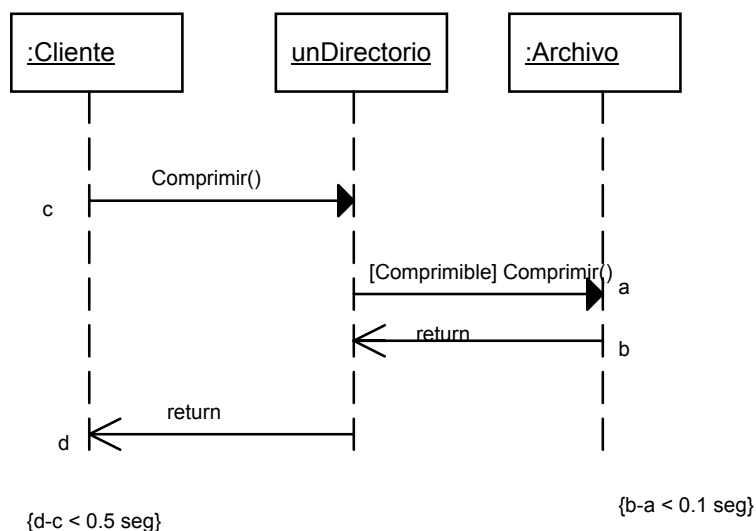
Los Diagramas de secuencia pueden mostrar tiempos límite entre eventos.

- Un límite puede ser especificado para cualquier par de puntos

Un punto es cualquiera: el envío o la recepción de un evento

- Los puntos pueden ser etiquetados con “marcas de tiempo” explícitas

Alternativamente, las marcas pueden ser encerradas entre llaves.

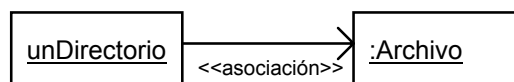


## 4.11 Implementación de ligas en diagramas de colaboración

- Podemos describir como un objeto obtiene una liga a otro.

*Por ejemplo, un objeto directorio tiene una liga de asociación con su objeto archivo.*

- La liga es estereotipada para mostrar implementación
  - **<<asociación>>**: El objeto cliente tiene un campo manejando la referencia al objeto servidor.
  - **<<global>>** El objeto cliente obtiene una referencia al objeto servidor de una variable global
  - **<<local>>** El objeto cliente tiene una referencia al objeto servidor en una variable local de un método.
  - **<<parámetro>>**: El objeto cliente recibe una referencia al objeto servidor como un parámetro para un método del cliente.
  - **<<self>>**: El objeto cliente se referencía a sí mismo vía “this” o “self”
  - (<<asociación>> es el default, pero puede ser especificado por énfasis.



**V**

**5 Modelos dinámicos:  
Diagramas de Transición de  
estados**

## 5.1 El Modelamiento Dinámico

Objetivo:

Analizar relaciones temporales.

Descripción:

Está formado por aquellos aspectos de un sistema que están relacionados con el tiempo.

Estudia estados, eventos y transiciones.

Es una colección de diagramas de transición de estados por cada clase con comportamiento no-trivial. Estos se combinan vía eventos compartidos.

## 5.2 Estado

Valores de los atributos y ligas manejadas por un objeto.

Por ejemplo en un objeto de la clase "alumno", en una aplicación escolar, se podrían presentar los estados siguientes:

Inscrito

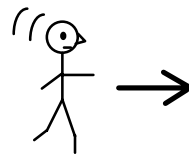
Dado de Baja

Suspendido

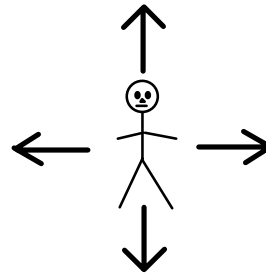
## ..Ejemplo de Estados en un objeto

En un objeto de la clase "Mono", en una aplicación de juego de video podríamos tener por su parte los estados que siguen:

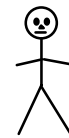
En movimiento



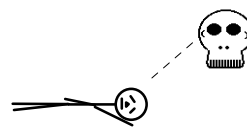
En cambio dirección



Estático



Muerto



Los estados no se deben traslapar:

Estados "académicos" de un objeto de la clase "Alumno":

Irregular (1 ó 2 materias reprobadas)

Regular (Sin materias reprobadas)

Año Sabático (3 materias reprobadas)

Fuera Reglamento (4 ó más materias reprobadas)

Estados "económicos" del alumno:

Al corriente

con adeudo (debe colegiaturas)



## 5.3 Evento

Estímulo individual de un objeto sobre otro que genera un cambio de estado

Algo que pasa en un instante de tiempo. Es algo instantáneo.

Ejemplos de evento:

- Usuario hace clic con el ratón
- Alumno llega al módulo de inscripciones
- Inicia la aplicación de un examen
- Finaliza la calificación de un examen

No son eventos:

- Usuario llena una forma
- Alumno hace examen
- Alumno se inscribe
- El proceso corre

¿Por qué?

...La clave: El verbo

## 5.4 Escenario:

Es el conjunto de eventos que ocurren durante la ejecución de un sistema.

Escenario de un juego de video:

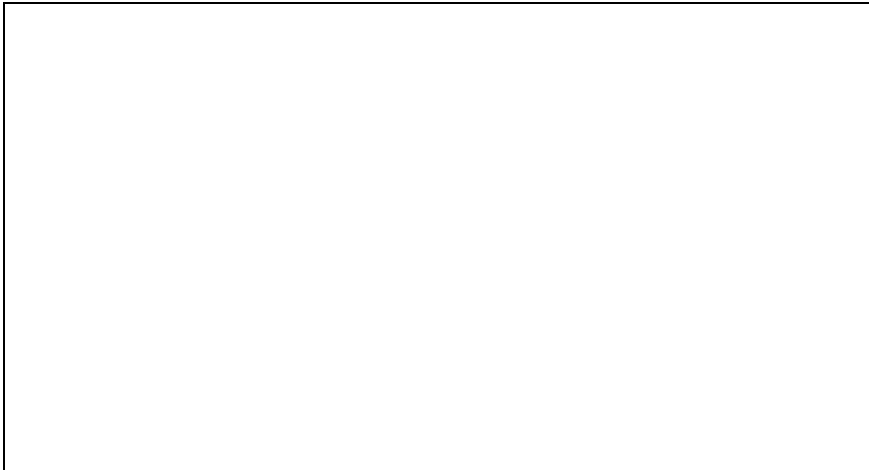
- Monito gira 180 grados
- Monito inicia visualización monstruo
- Monito es herido
- Monito es muerto
- Monito encuentra provisiones
- Monito mata monstruo
- Monito dispara
- Monito falla disparo
- Monito inicia huída

Escenario de un Sistema Escolar:

- Inicia el Curso
- Inician Exámenes extraordinarios
- Termina el Curso
- Terminan exámenes extraordinarios
- Termina un periodo parcial

...Ahora tú:

Escenario de operación de un Walkman reversible:

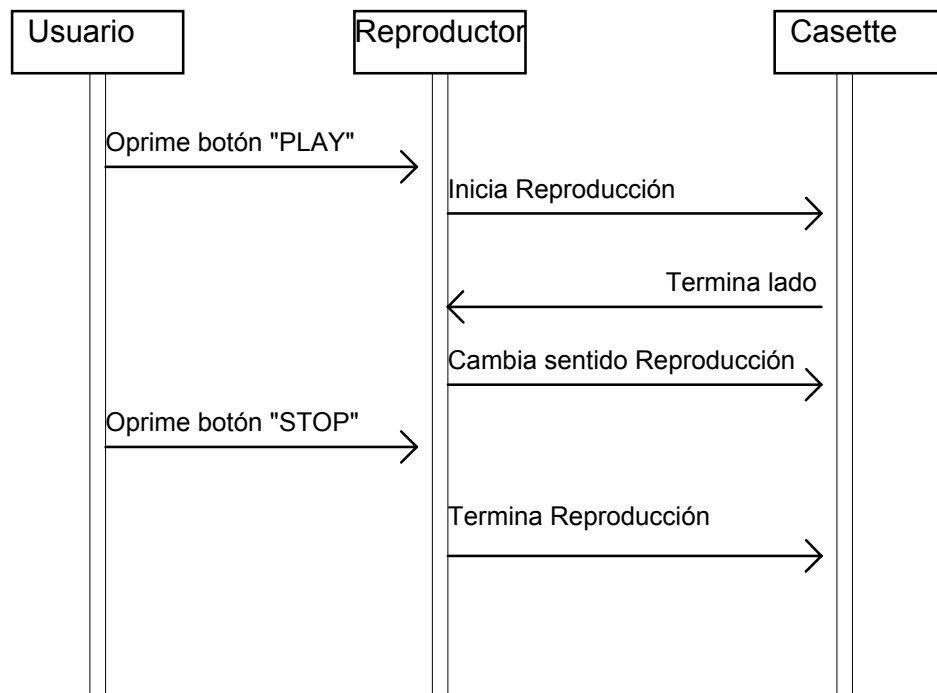


Escenario de operación de un Walkman reversible:

- Usuario oprime boton "PLAY"
- Reproductor cambia sentido reproducción cassette
- Reproductor Inicia reproducción cassette
- Reproductor termina reproducción cassette
- Reproducción inicia distorsión música
- Pilas llegan al nivel mínimo de carga
- Usuario finaliza cambio pilas
- Usuario inicia cambio pilas
- Termina lado cassette
- Usuario oprime botón "STOP"

## Secuencia de eventos "Walkman"

Muestra la secuencia de eventos entre distintos objetos.



Representa la secuencia en modo cronológico.

¿Cómo quedaría esta secuencia si se considerara la descarga de las pilas?

## 5.5 Estados y su naturaleza

Un estado es una abstracción de los valores de los atributos y ligas de un objeto.

Por ejemplo:

Dado un objeto de la clase "artículo", identificamos los siguientes estados:

Insuficiente

Suficiente

Excedente

¿Qué atributo(s) están involucrados?

...Estados y su naturaleza

Clase: "ciudadano"

Estados:

Empleado

Desempleado

Clase: CuentaBanco

Estados :

Deudor

Acreedor

¿Qué lo determina?

Un estado corresponde al intervalo entre 2 eventos recibidos por un objeto

Un evento representa un punto en el tiempo

Un estado tiene duración

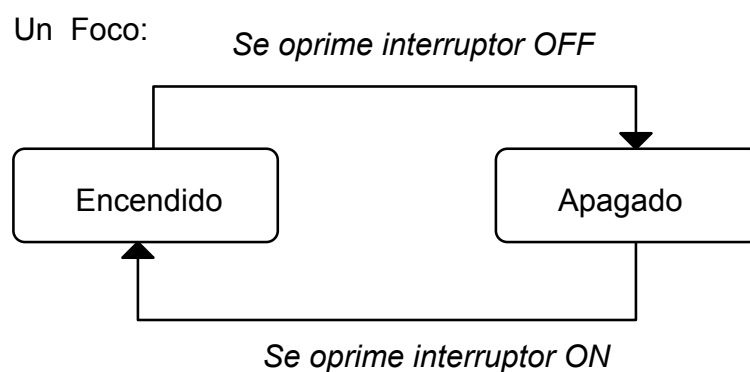
Evento	<u>Estado avión</u>	Evento
Avión despegue	<u>En vuelo (1 hr)</u>	Avión toca tierra



## 5.6 Diagramas de transición de estados

El paso de un estado a otro se denomina transición.

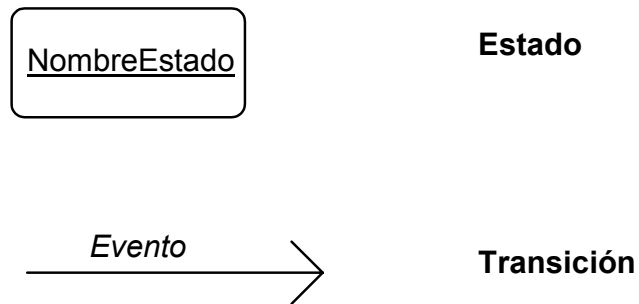
Un diagrama de transición de estados es una red de estados y eventos representando transiciones



Un diagrama de estados relaciona eventos y estados

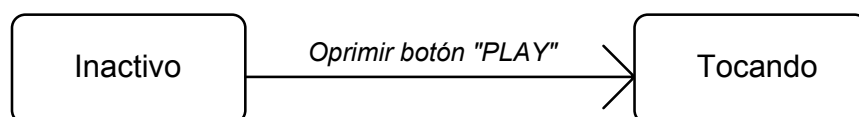
Expresa transiciones

Su simbología es:



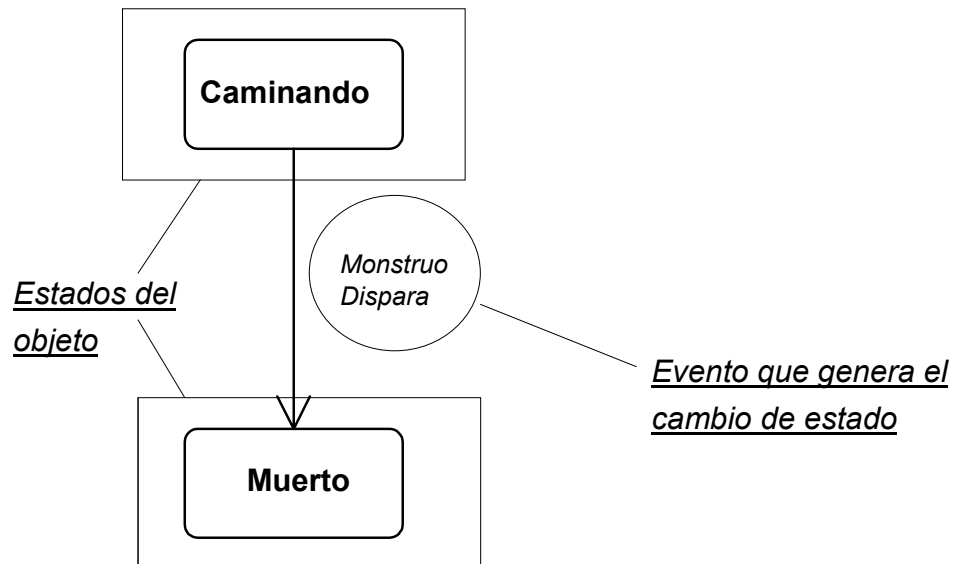
Ejemplos:

Transición de estados en un objeto de la clase "Reproductor Cassettes"



Transición de "Inactivo" a "Tocando"

Transición de un objeto de la clase "monito" del estado "caminando" a "muerto"



## ...Diagramas de Transición de Estados

Describen el comportamiento de una clase de objetos en particular.

Pueden ser loops o secuencias (o combinaciones)

Una secuencia representa objetos con vida finita

Por ejemplo:

Un ser humano pasa por los estados siguientes:

Niñez - Adolescencia - Adultez - Vejez

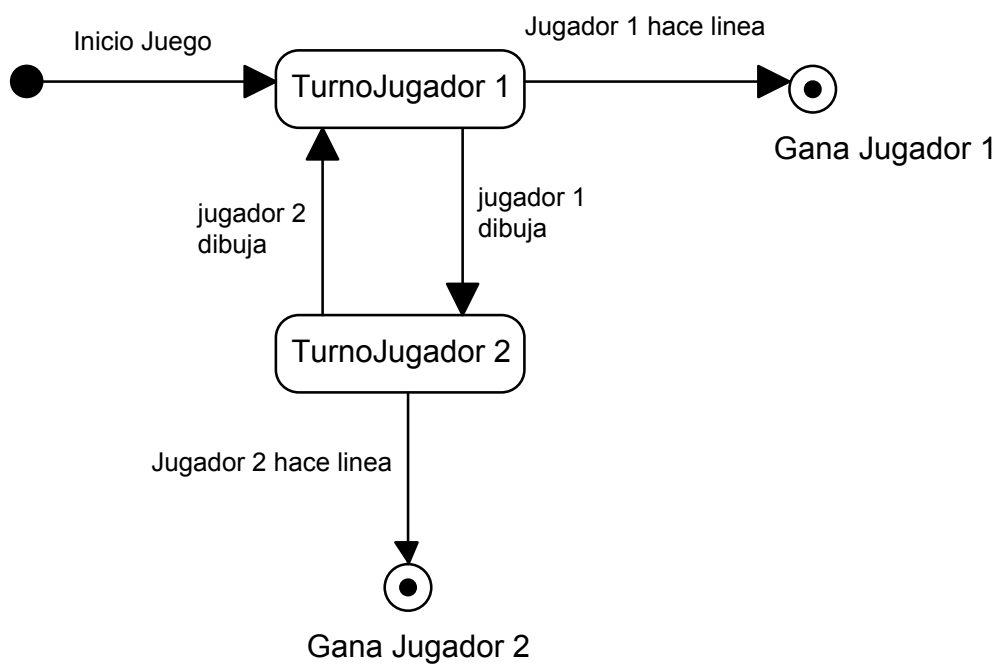
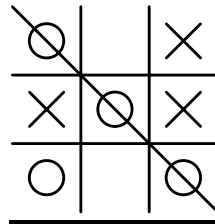
Cuando se expresan secuencias se requiere un estado inicial y uno final.

En ese caso:

- Significa Estado Inicial
- ⊙ Significa Estado Final

## Ejemplo de Diagrama de Transición de Estados con secuencia:

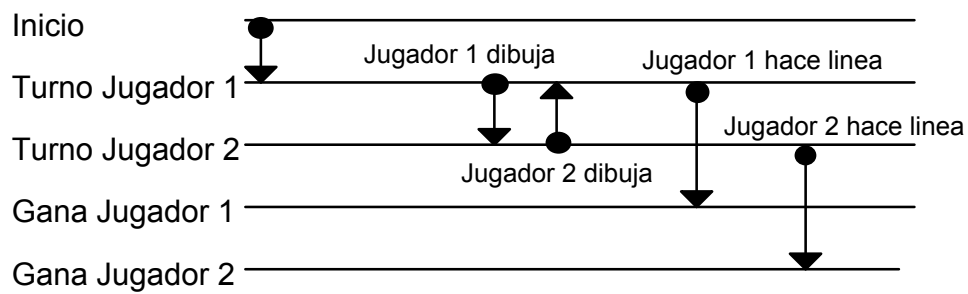
Juego de Gato:



## Diagrama de Transición de estados

### Notación alternativa (no UML)

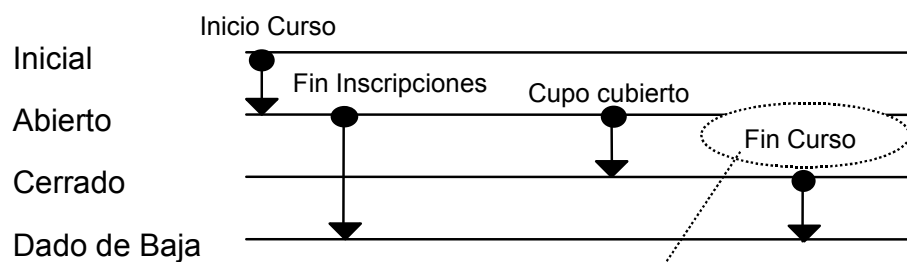
#### Juego de Gato



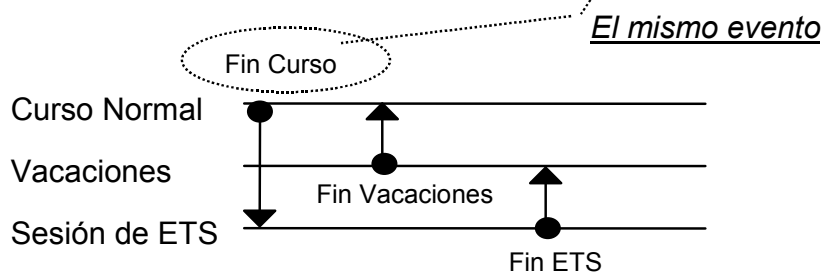
El modelo dinámico es una colección de diagramas de transición de estados que interactúan entre ellos vía eventos compartidos.

Por ejemplo:

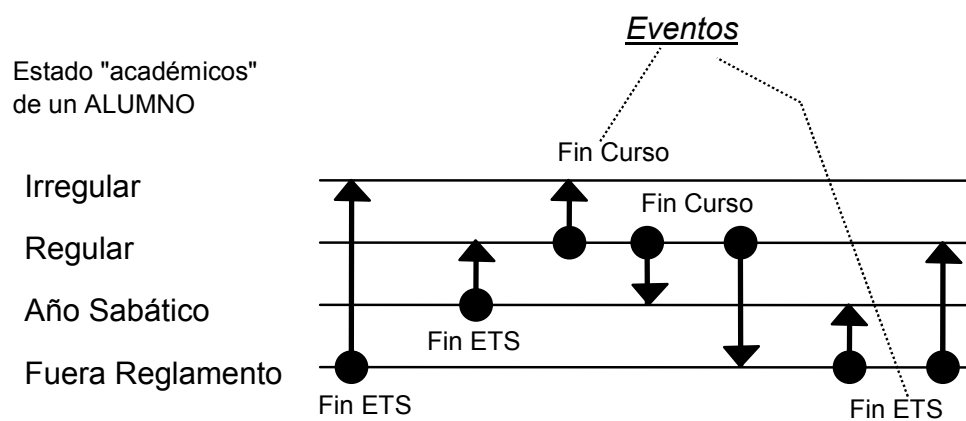
**CLASE: GRUPO**



**CLASE: ESCUELA**



## Un pequeño problema:



## ¿ Qué diferencia cada evento ?



## 5.6.1 Condiciones

Se pueden usar condiciones como "guardianes de transiciones".

Ejemplo:

Automóvil (En cruce)

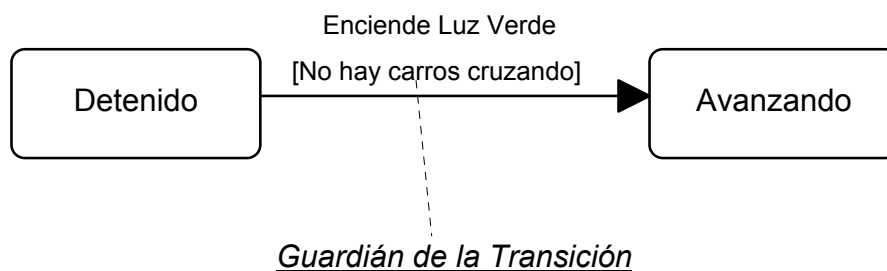
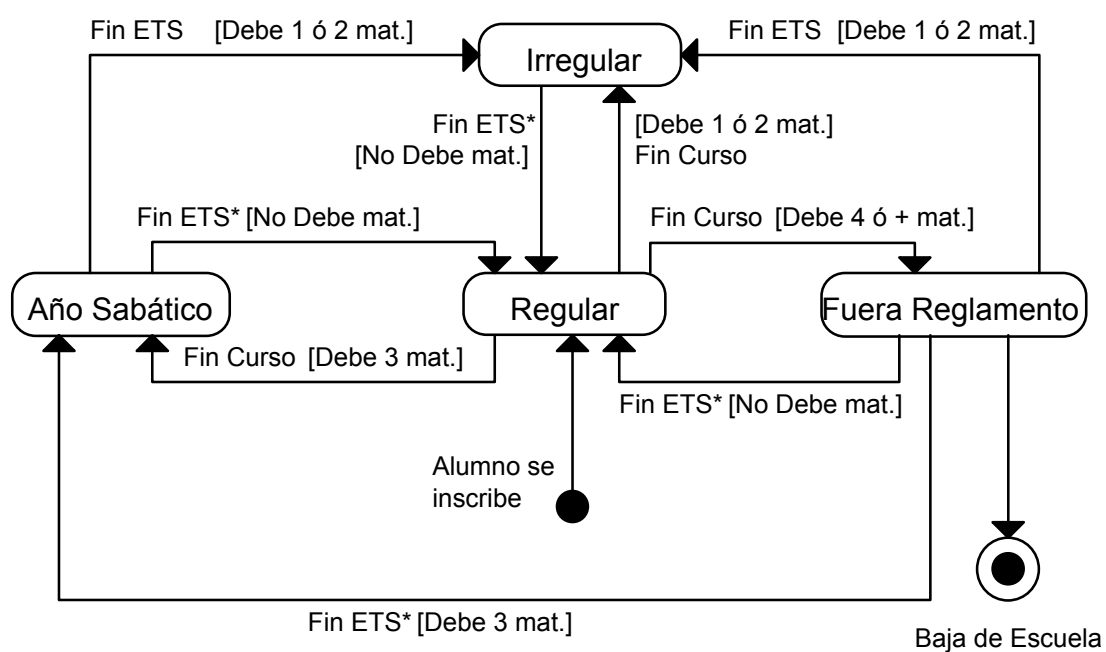


Diagrama de Transición de estados para un objeto de la clase alumno (con guardianes):



\* ETS = Exámenes a Título de Suficiencia (extraordinarios)

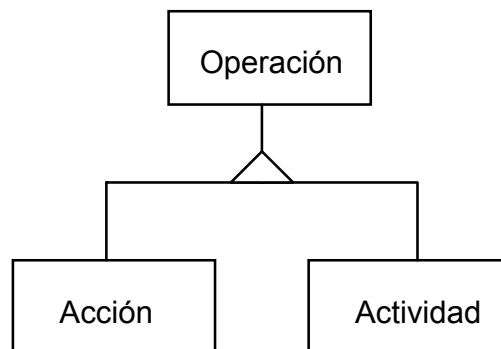
## 5.6.2 Operaciones en un diagrama de transición de estados

Preguntas:

¿ Qué hace un objeto en respuesta a un evento ?

¿ Cómo se expresan estados "dinámicos" (por ejemplo, el estado "reproduciendo" en un reproductor de cassetes) ?

Tipos de operaciones:



Actividad: Operación que se lleva tiempo completar.

Acción: Operación instantánea (de tiempo insignificante comparado con la resolución del diagrama de transición de estados).

### Ejemplos de Actividades:

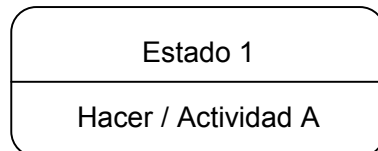
- Clasificar un Archivo
- Emitir Estados de cuenta de ctes de Bancomer
- Reproducir un cassette
- Animar un monito en un juego de video
- Efectuar una actualización masiva
- Desplegar una fotografía (pixel por pixel)

### Ejemplos de Acciones

- Insertar una ocurrencia de alumno en la B.D.
- Calcular ISR de un empleado
- Cambiar dirección de animación de un monito en un juego de video
- Calcular situación académica en un alumno
- Detener la reproducción de un cassette

## Actividades

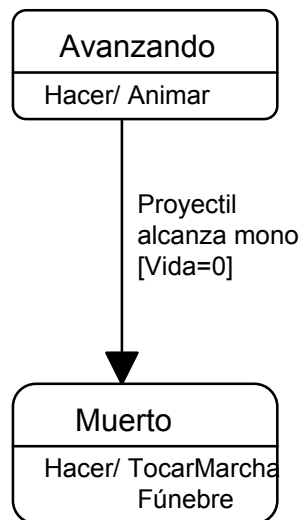
Una actividad está asociada con un estado



La actividad A arranca al entrar en el Estado 1 y termina a la salida de tal estado.

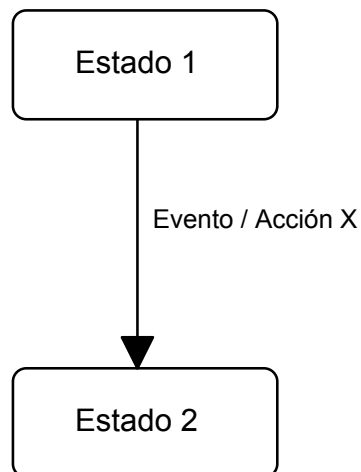
Ejemplo:

Objeto: Un monito (de un juego de video)



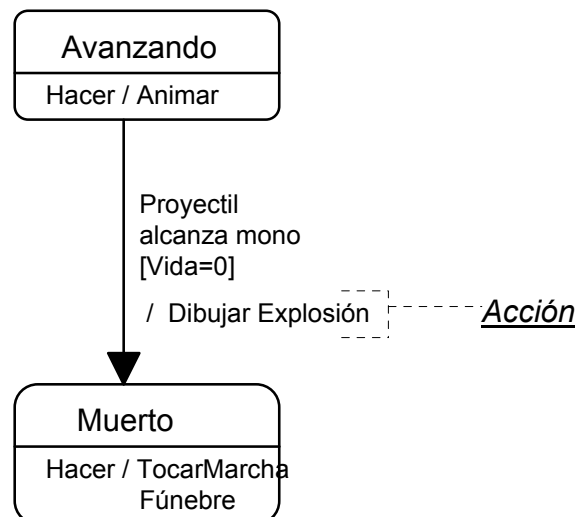
## Acciones

Una acción es asociada con un evento

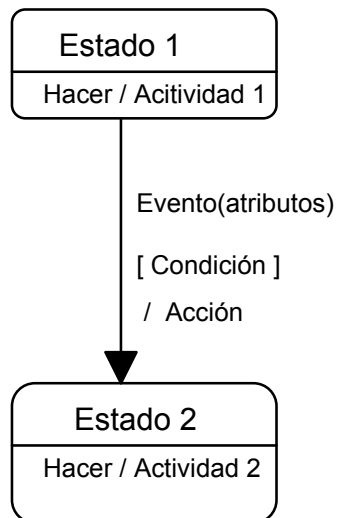


Ejemplo:

Objeto: Un monito (de un juego de video)

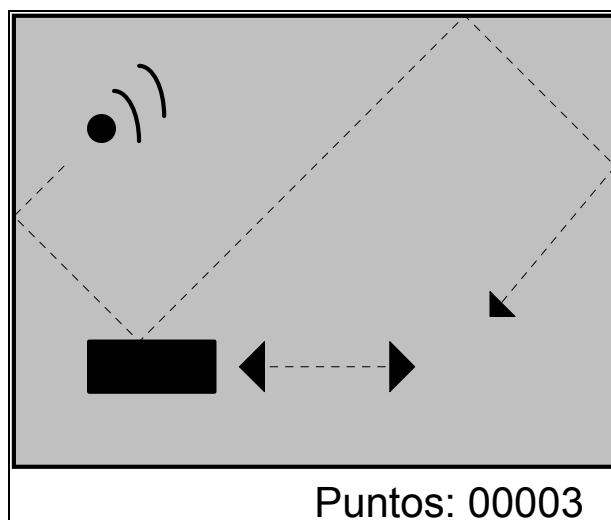


## 5.7 Notación (Resumen)



## 5.8 Ejercicio:

Elabore el modelo dinámico de un videojuego de frontón de un jugador.



Las reglas son:

El jugador obtiene 1 punto por cada golpe de su raqueta a la pelota.

El juego termina si la pelota sale de la pantalla por el lado inferior.

## Requisitos:

Considere objetos de las clases: Raqueta, pelota, juego y marcador.

Considere el manejo de un record máximo, cuando éste sea superado, debe parpadear el marcador y hacer algún efecto.

Si la raqueta está en el límite de la pantalla y se oprime una tecla de dirección hacia ese lado, se deberá emitir un "beep".

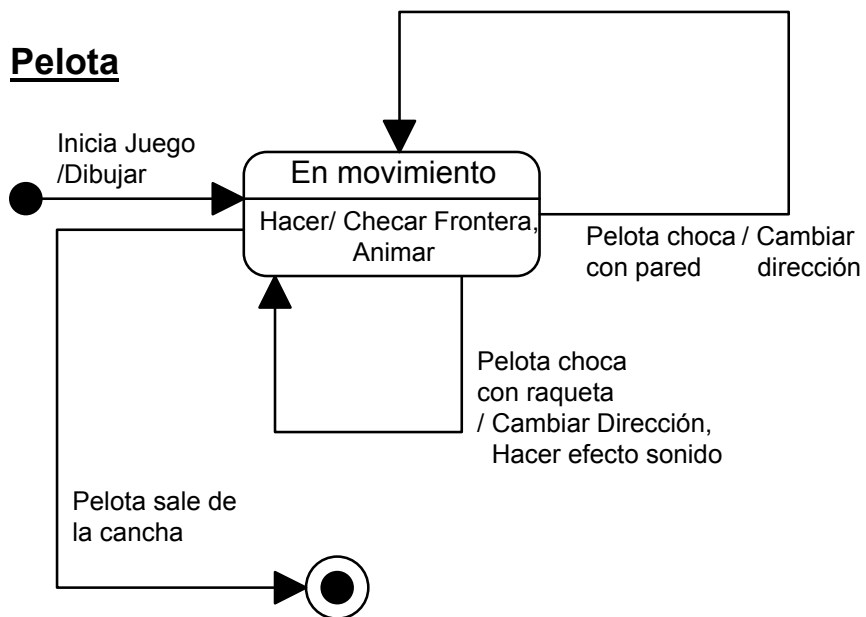
Cuando la raqueta toque la pelota, debe emitirse algún efecto de sonido.

Al inicio del juego se toca una marcha deportiva (la de los santos, por ejemplo).

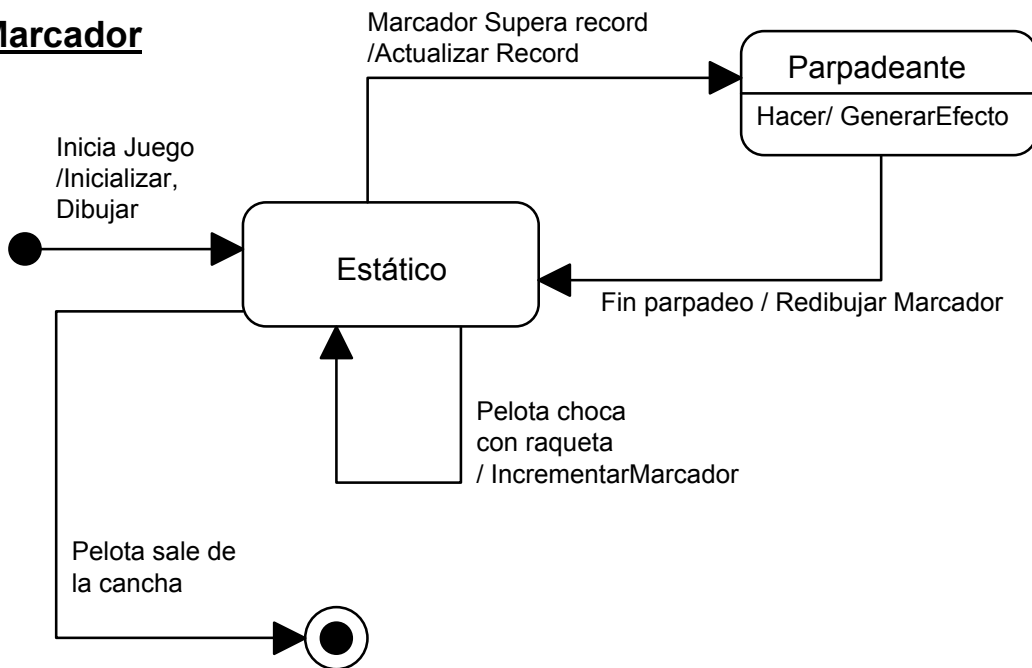
Cuando termina el juego deben desplegarse los créditos del juego (autor, animadores, artistas, etc.).



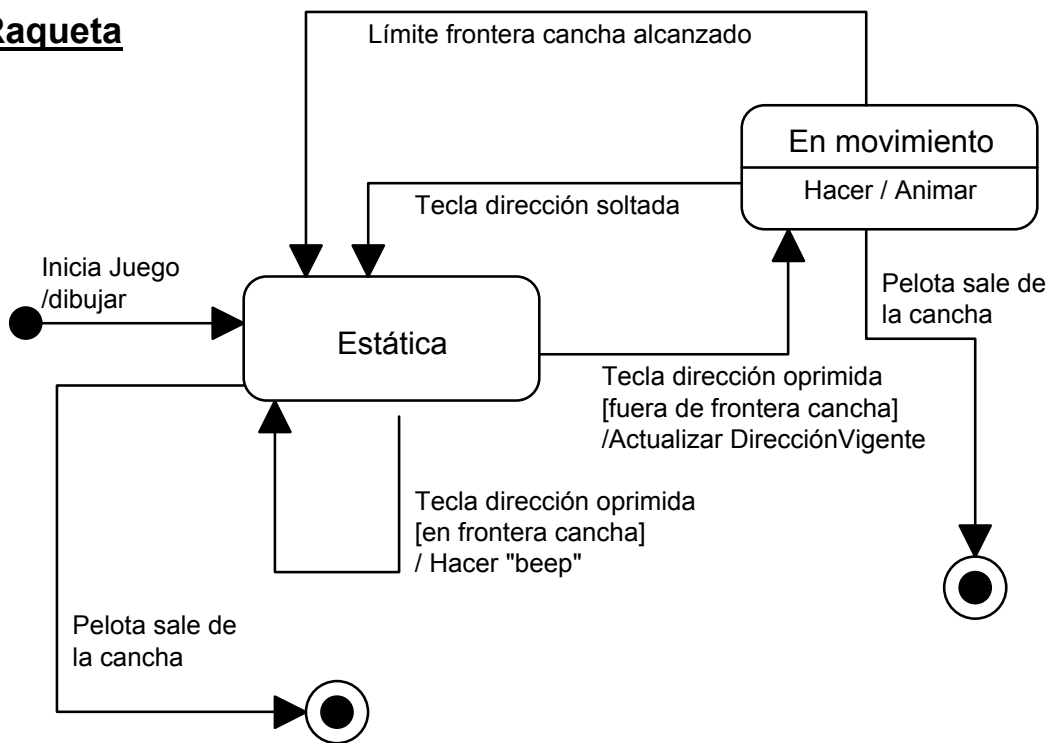
## Pelota



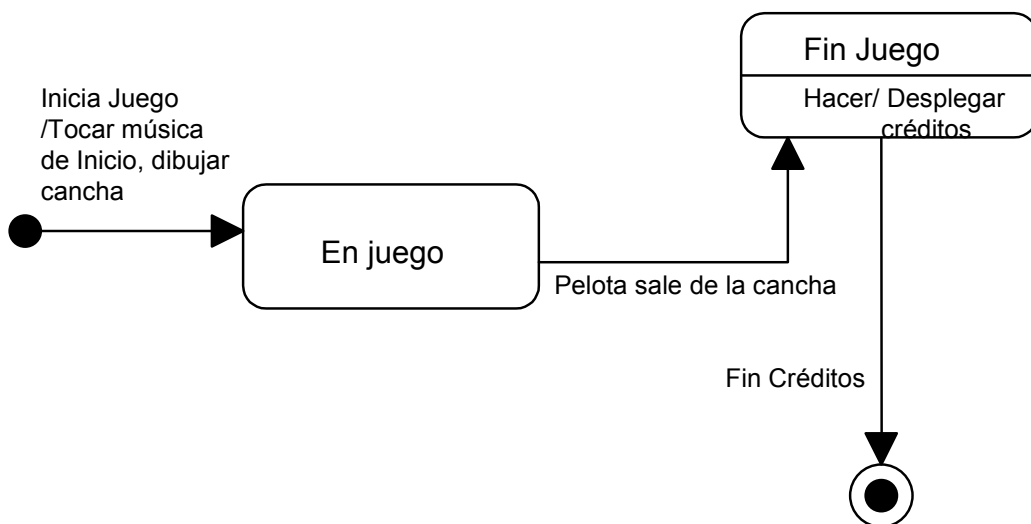
## Marcador



## Raqueta



## Juego



## 5.9 Diagramas anidados

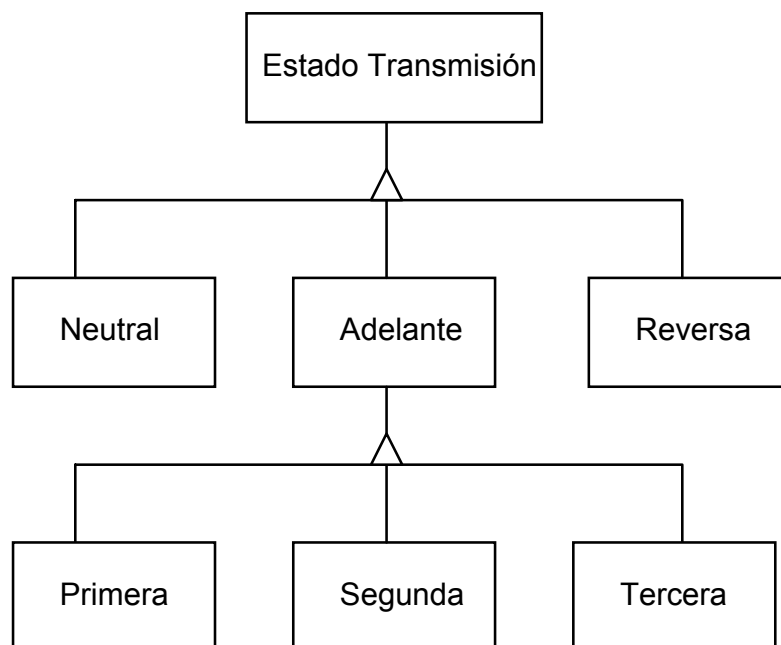
Mecanismo para describir sistemas complejos

Filosofías: Generalización y Agregación

La generalización permite que eventos y estados sean clasificados en jerarquías con herencia de estructura y comportamiento comunes.

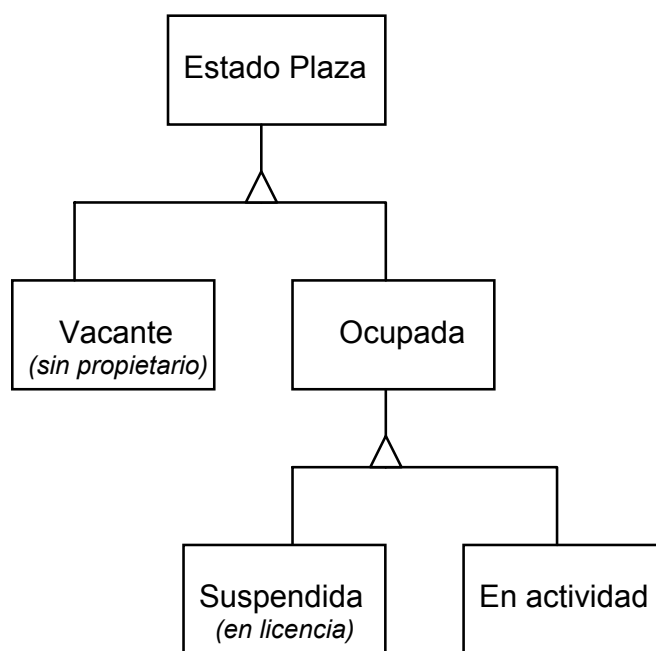
Ejemplo de Generalización de Estados:

Transmisión de un coche:

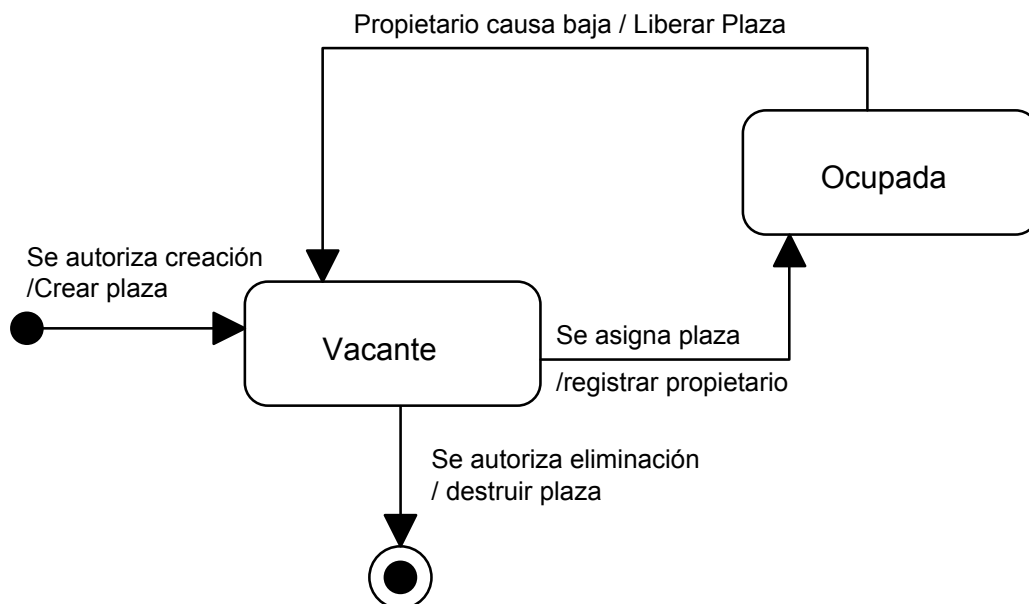


## Ejemplo de Generalización de Estados:

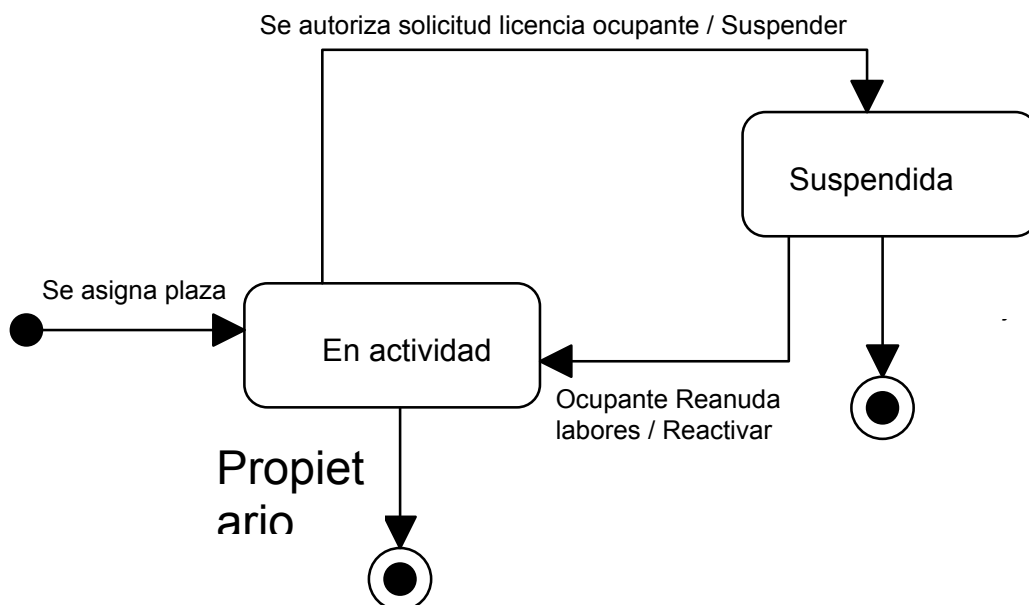
Plazas en una dependencia gubernamental:



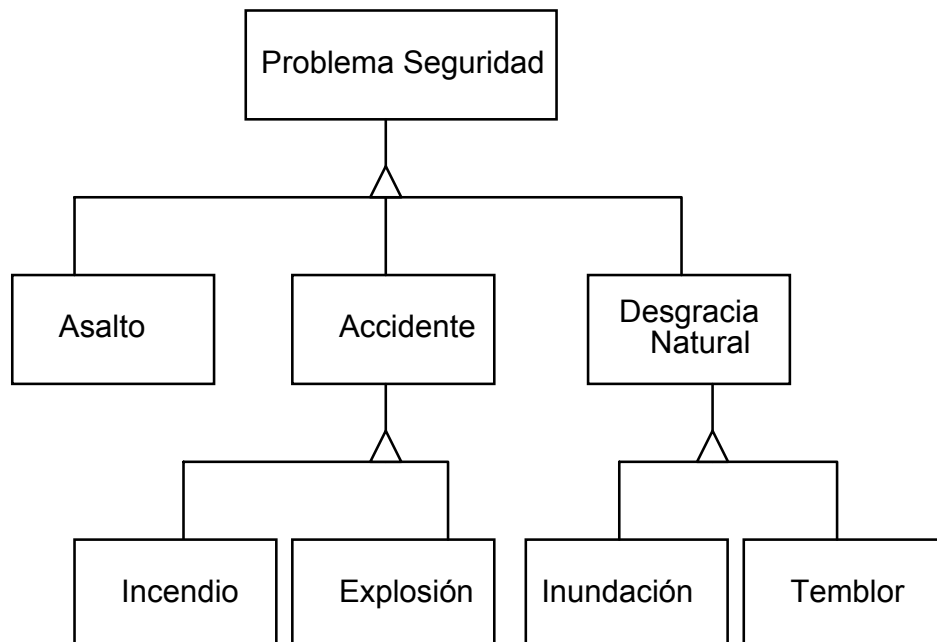
## Diagrama de Transición de estados Plaza (Padre )



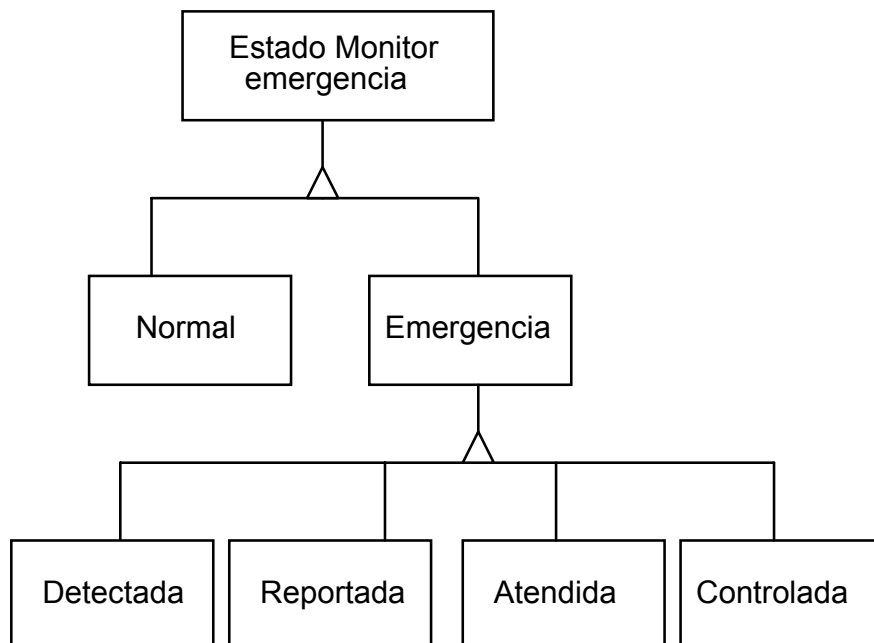
## Diagrama de Transición de estados Plaza (hijo )



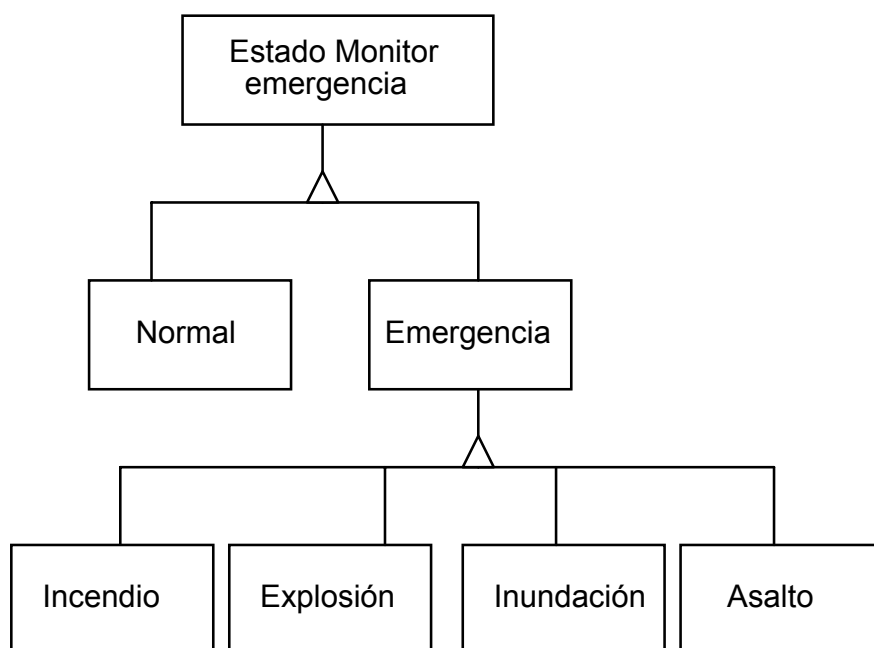
## Generalización de eventos:



## Generalización de Estados:



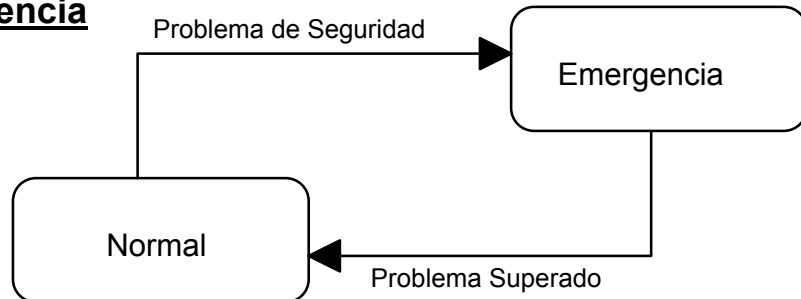
La siguiente Generalización es incorrecta ¿Por qué ?



¿ Expresan cambios en el tiempo ?

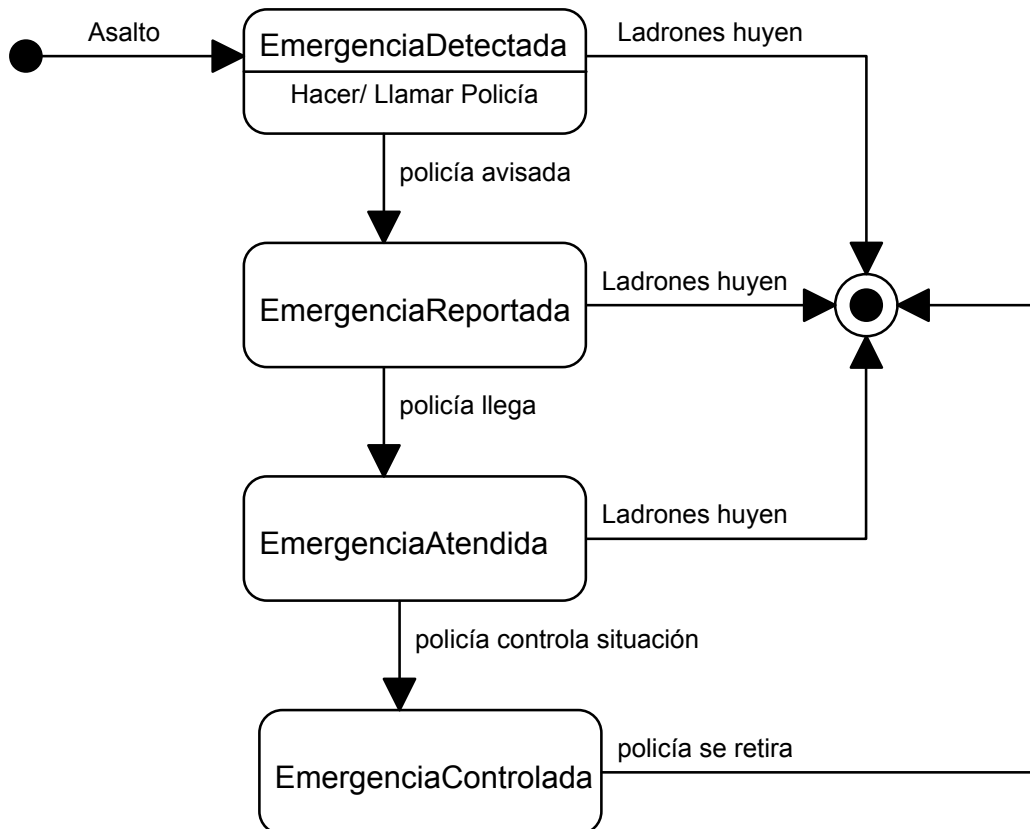
## D. de T. de Estados (padre) Problema de Seguridad

### MonitorEmergencia



## D. de T. de estados (hijo 1: Asalto)

### MonitorEmergencia (Asalto)



¿Sería el mismo para otro sub-evento?



Ejercicio: Editor de Pantalla

Clase: Desktop

Estados:

- Modo Texto
- Modo Comando
- Modo Inserción
- Modo de Marcado
- :

Eventos :

- Click del Ratón
- Tecla Oprimida
- Tecla de Control Oprimida
- Tecla caracter Oprimida

# **VI**

## **6 Otros Diagramas**

## 6.1 Diagrama de Actividad

Un diagrama de actividad muestra las relaciones temporales entre actividades.

Una actividad es algo que debe hacerse.

Una función del sistema

la invocación de una operación.

Una actividad puede describirse como una “cápsula” etiquetada.



## Flujo de Trabajo (Workflow)

Las transiciones muestran el flujo entre actividades.

Una actividad puede seguir a otra incondicionalmente.

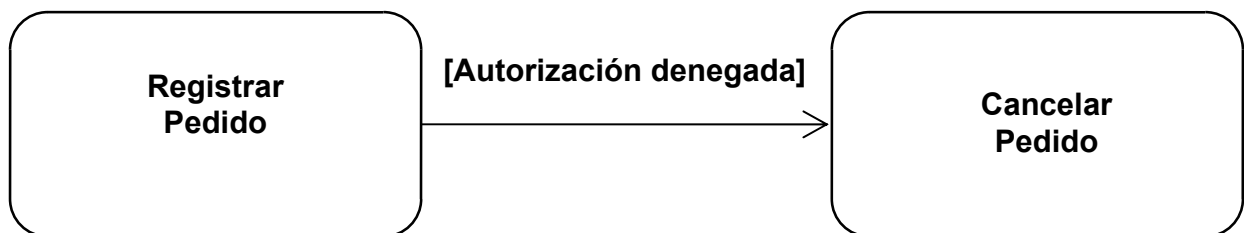


## Guardianes

Una actividad puede seguir condicionalmente a otra.

La transición toma lugar, sólo si la condición se evalúa como verdadera.

Por ejemplo, el pedido se cancela, si no se autoriza la compra.



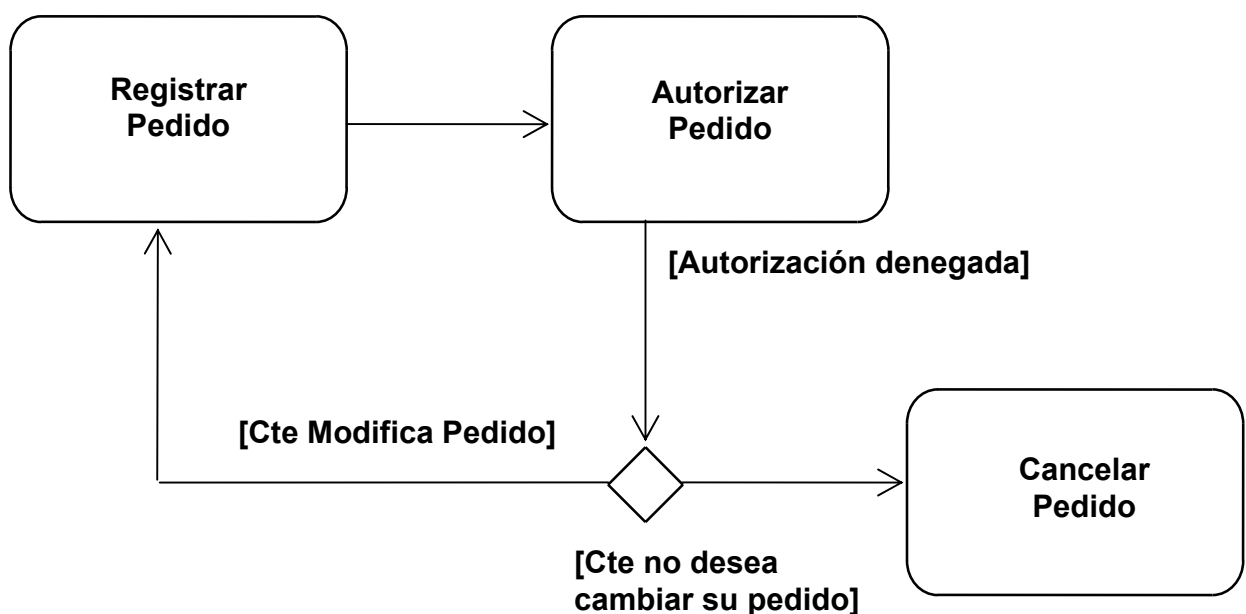
## Decisiones Anidadas

Un símbolo de decisión permite describir decisiones anidadas.

Se describe como un diamante.

Se permite anidar a cualquier nivel.

Por ejemplo, cuando se niega la autorización del pedido, podemos hacer una segunda decisión basada en que el cliente quiera modificar el pedido: reducir el número de artículos pedidos o presentar otra tarjeta de crédito.



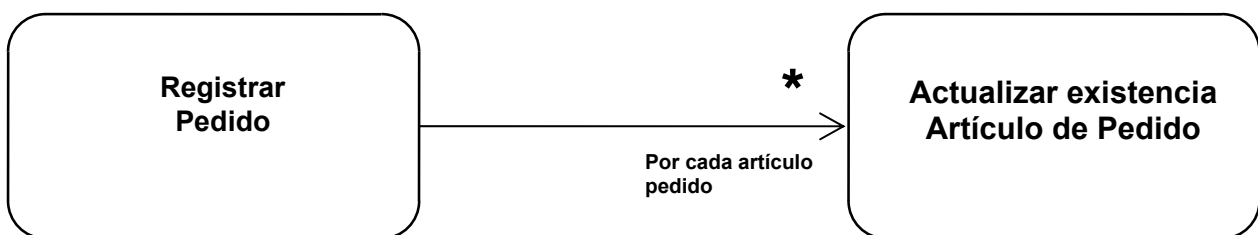
## Ejecución Múltiple

La ejecución múltiple de transiciones similares se muestra como una sola.

Se considera que las transiciones posiblemente sean en paralelo.

La multiplicidad se describe con un asterisco \* en la transición.

Por ejemplo, cuando se registra un pedido, debemos actualizar la existencia de cada artículo del pedido.



## Paralelismo

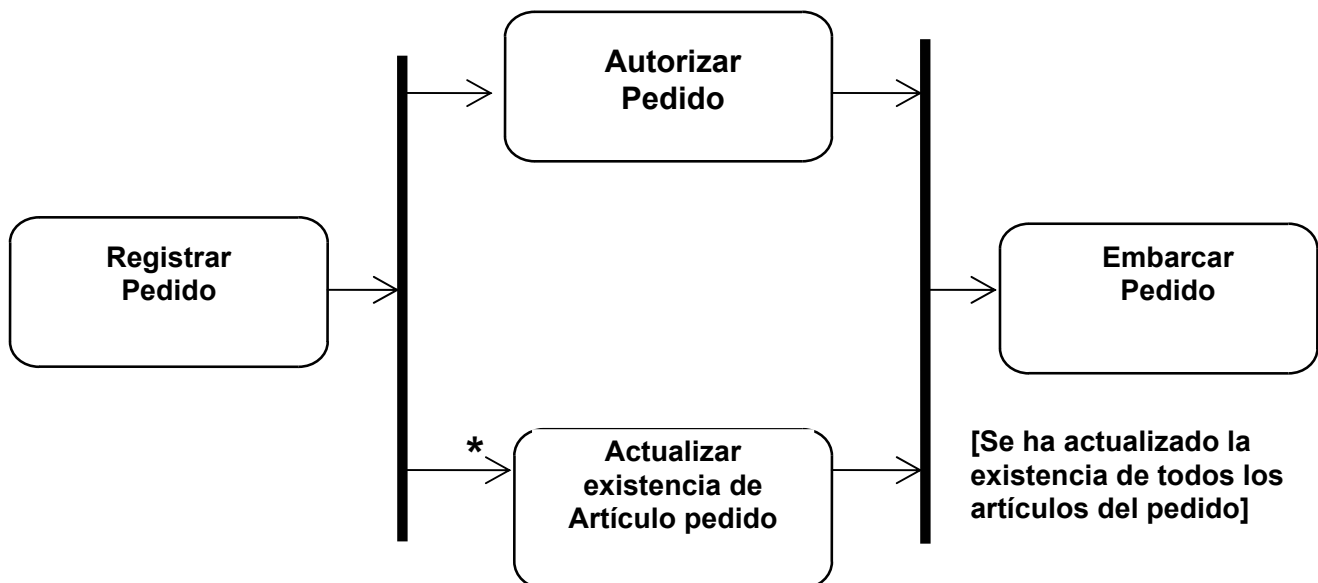
Una barra de sincronización permite expresar paralelismo.

De la barra pueden emitirse varias transiciones, indicando que posiblemente, sigan actividades paralelas.

Pueden llegar varias transiciones a una barra, indicando que las actividades precedentes, deben completarse.

Una barra de sincronización, debe incluir una condición de sincronización, la cual aparece entre corchetes.

La condición debe leerse antes de proceder con las siguientes actividades.





## Diagramas de Actividad para relacionar Casos de Usos

Un propósito de los diagramas de actividad es mostrar la secuencia de los casos de uso.

Esto es, las actividades en un diagrama de actividad pueden ser Casos de Uso.

Entonces, el *diagrama de actividades* muestra la secuencia de esos Casos de Uso.

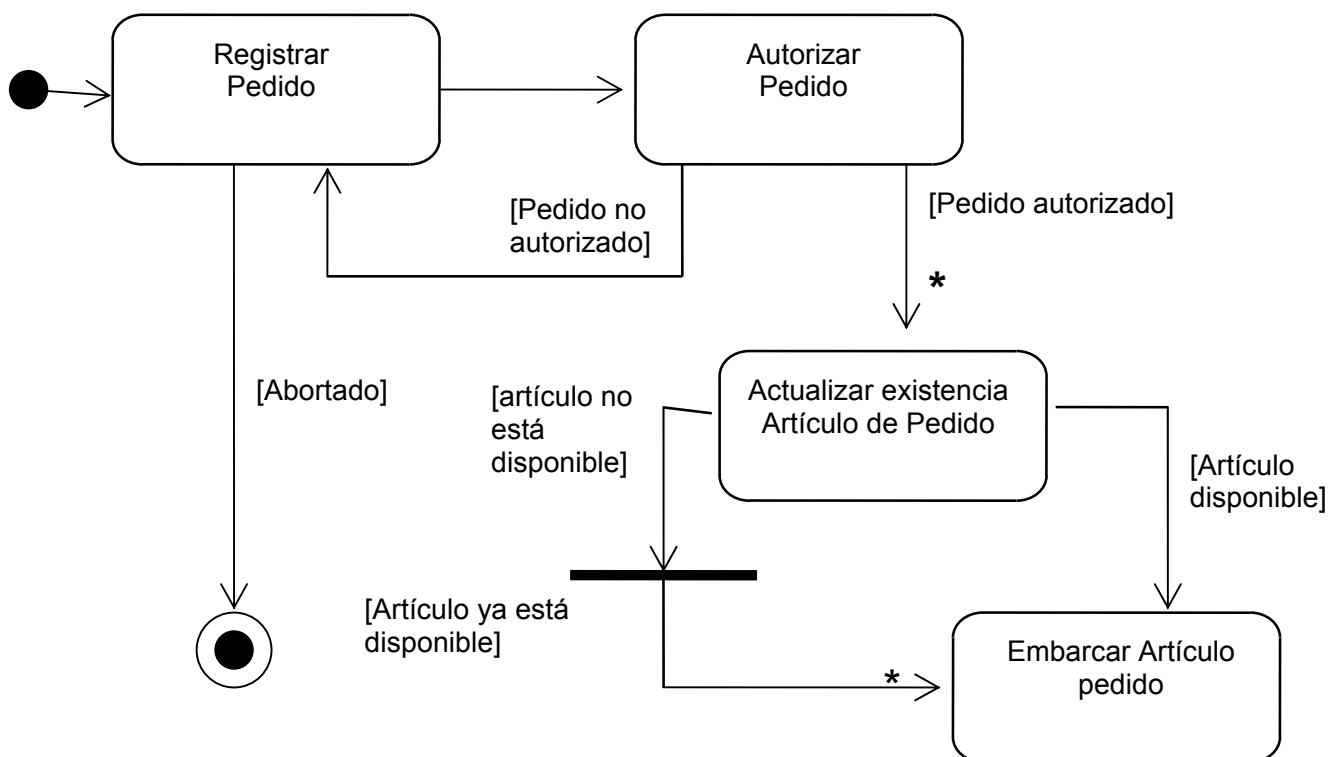
Podemos mostrar la secuencia de actividades del Caso de Uso de pedidos.

Las actividades en el diagrama son casos de uso.

## Diagramas de actividad para el sistema de pedidos

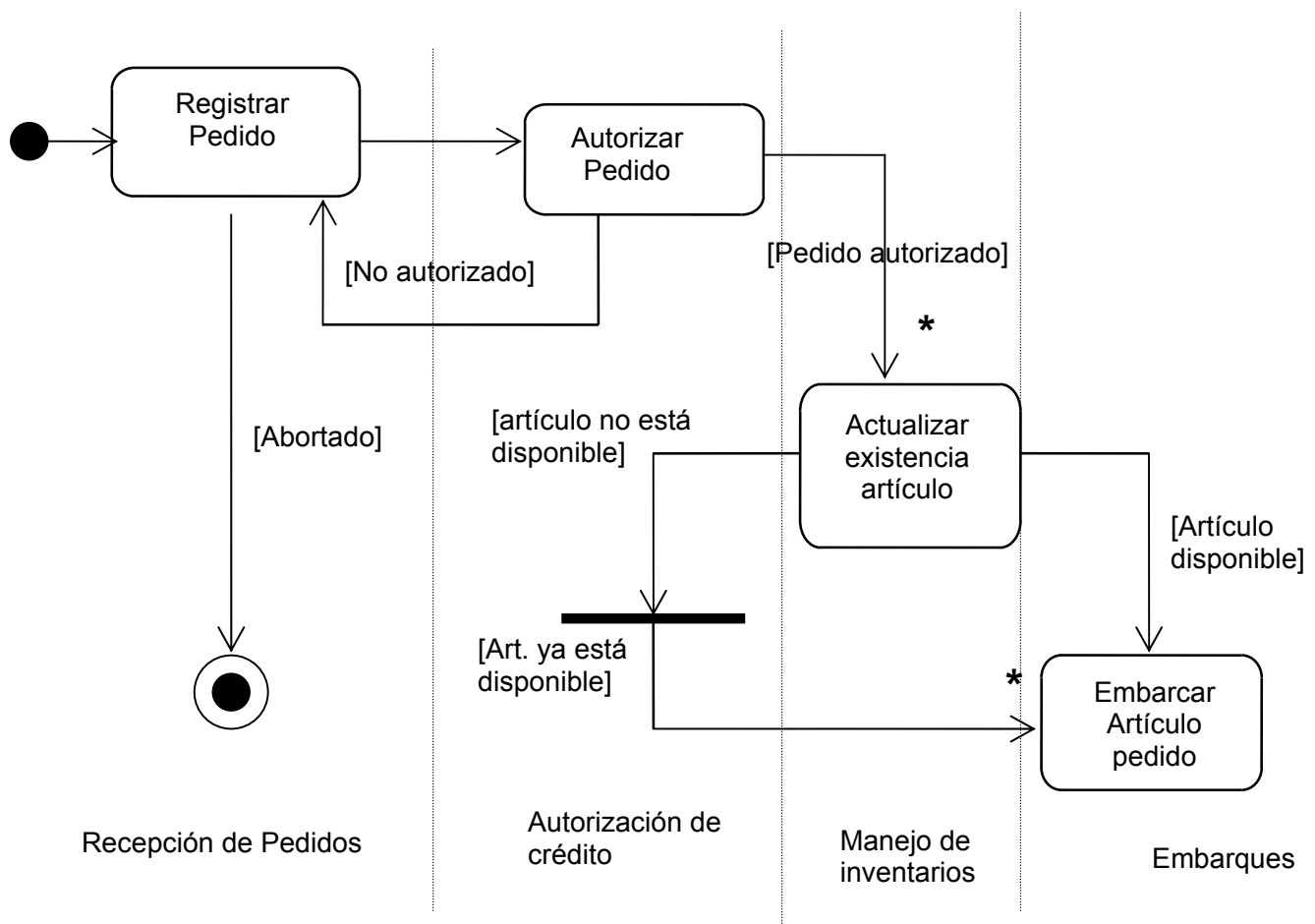
Un diagrama de actividad para procesar pedidos.

Tiene que autorizarse el pedido, antes de actualizar la existencia de los artículos.



Se pueden añadir “carriles” para mostrar quién hace cada actividad.

En este sistema, ese “quién” es un subsistema.



Aplicaciones de diagramas de Actividad

Relacionar requerimientos y diseño.

Mostrar la secuencia de un Caso de Uso.

Pueden mostrar también los “pasos” de un caso de uso.

## Componentes UML

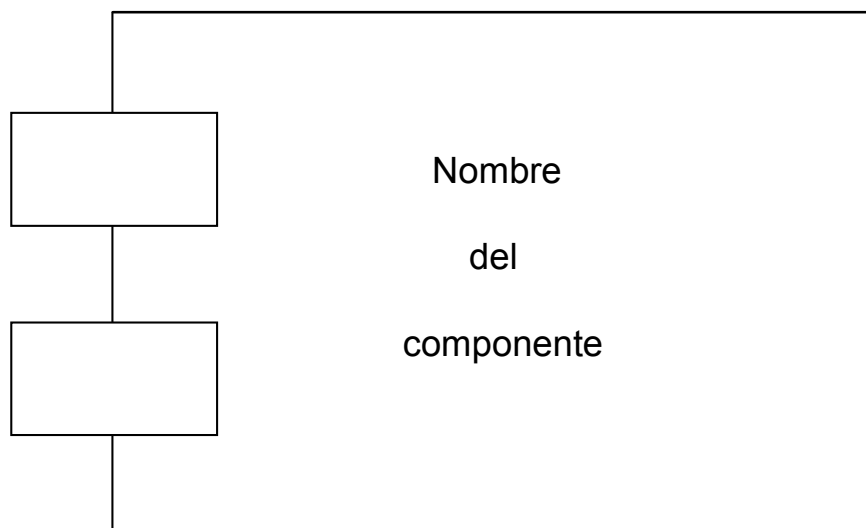
Un componente es una unidad al tiempo de compilación, encadenamiento (*link*) o ejecución.

Esto representa un empaquetamiento físico.

Por ejemplo, un archivo fuente de compilación en la mayoría de los lenguajes de programación, o una librería precompilada.

Una unidad ejecutable en la mayoría de los sistemas de programación, por ejm., COM, DCOM, etc.

La notación para un componente es un rectángulo con “puertos”



## Notación de componentes en UML

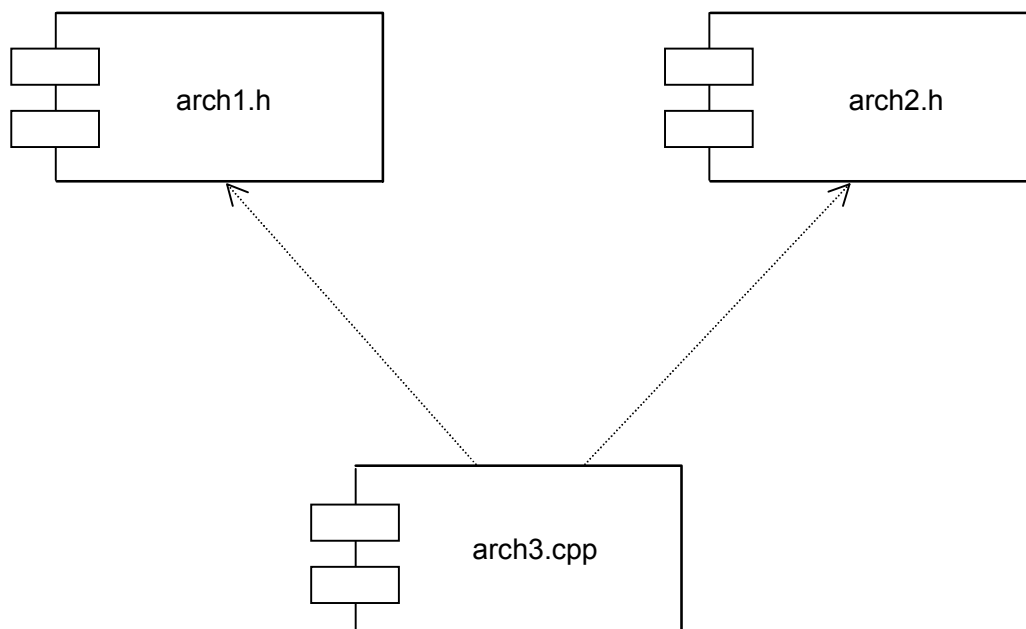
Los componentes pueden tener dependencias.

Por ejemplo, podemos dividir el sistema en componentes a tiempo de compilación.

En C++ tendríamos componentes para archivos de especificación, programas principales y los cuerpos de los programas.

Las dependencias serían Dependencias de Compilación.

UML dice que este es un Diagrama de Componentes.

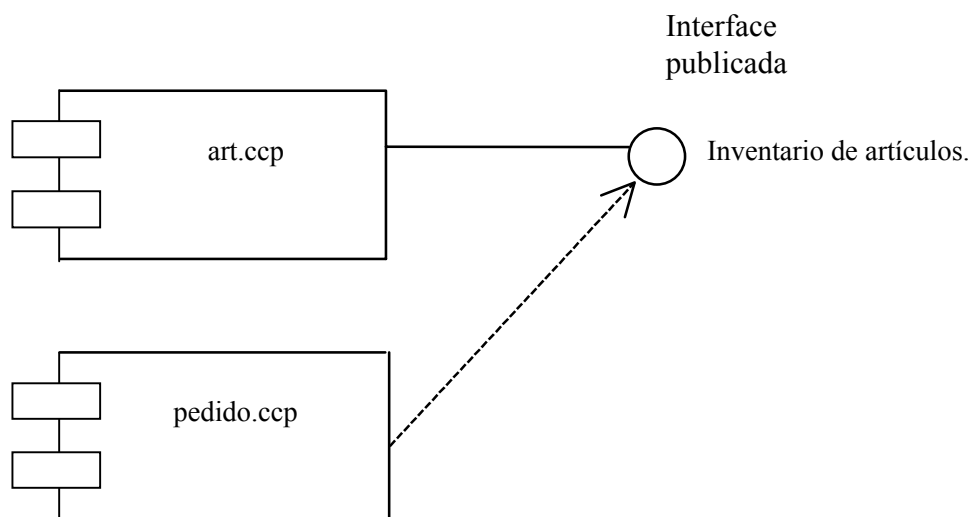


También pueden mostrarse dependencias de llamadas específicas.

Suponga que el catálogo de artículos esta en Art.cpp y que el manejador de pedidos está en Pedido.cpp.

Suponga que Art.cpp implementa una interface de Inventario de Artículos el cual es usado por Pedido.cpp

En tal caso, UML permite el uso de la notación de “paleta”.



## Diagramas de Distribución

Un sistema puede describirse desde el punto de vista de su arquitectura de distribución.

Esto es, se ve el sistema en términos de la arquitectura de su hardware.

Interesan los nodos de hardware.

Un nodo de hardware puede ser un procesador físico (un CPU) o un dispositivo.

Interesan las conexiones entre nodos.

Éstas son conexiones físicas entre nodos.

Las conexiones permiten que los nodos se comuniquen.



## Notación de distribución en UML

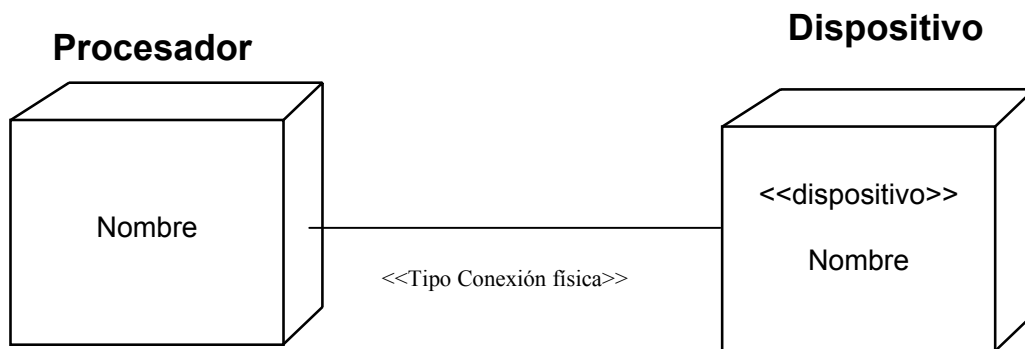
Una caja tridimensional representa un nodo de hardware.

Un nodo es un procesador (como un CPU).

Un dispositivo es un estereotipo de nodo (donde no hay software corriendo).

Las asociaciones entre nodos representan conexiones físicas.

Estas relaciones pueden estereotiparse para mostrar el tipo de conexión.



Un diagrama de Distribución muestra la arquitectura de distribución.

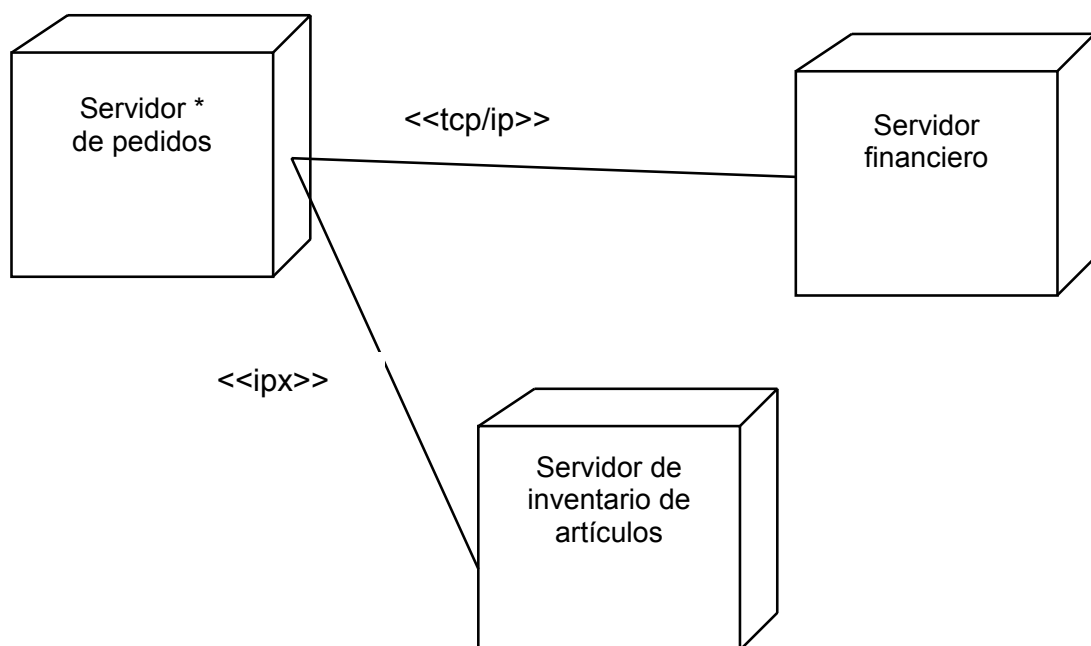
Ejm.: Considérese el sistema de pedido por catálogo.

Se tienen varios servidores de pedidos.

Se tiene un servidor para inventario de artículos.

Se tiene un servidor financiero.

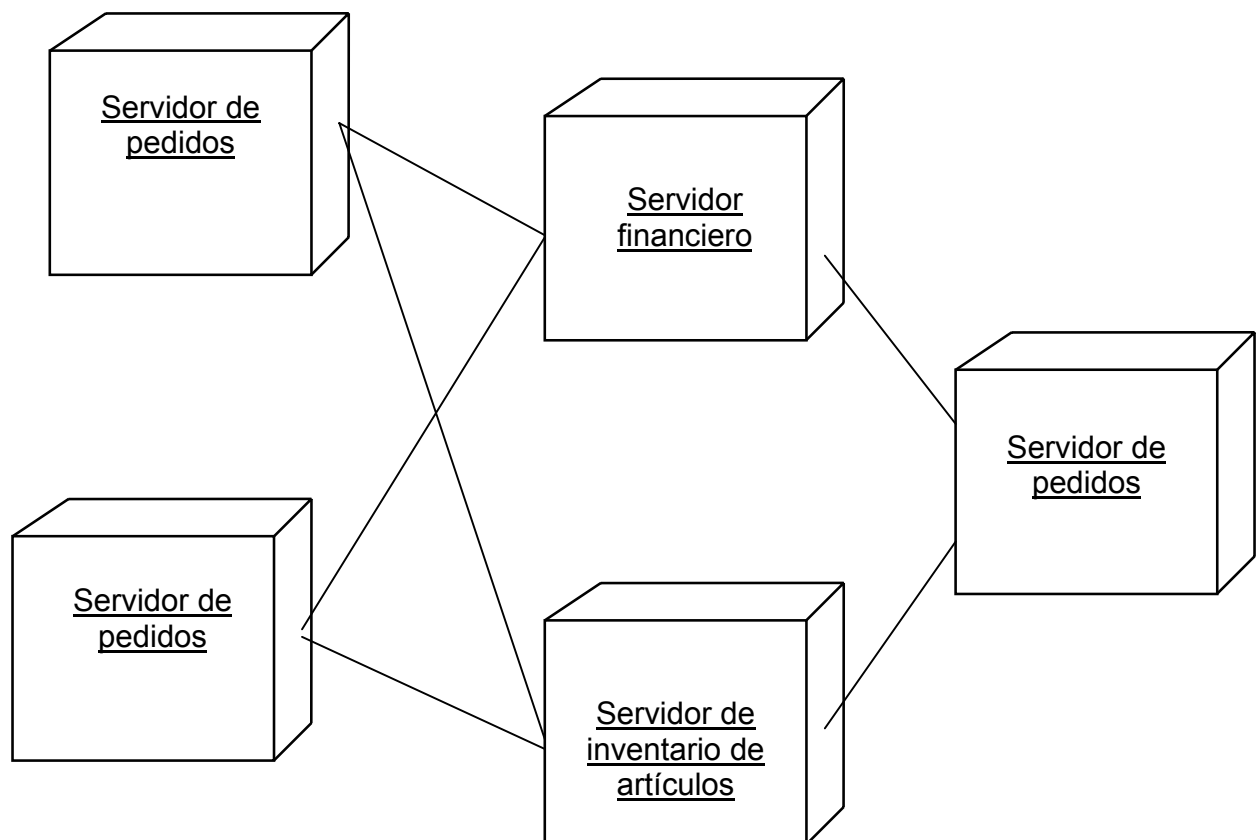
(Pueden incluirse máquinas externas como estereotipos)



También pueden dibujarse “diagramas de objetos” de Diagramas de Distribución:

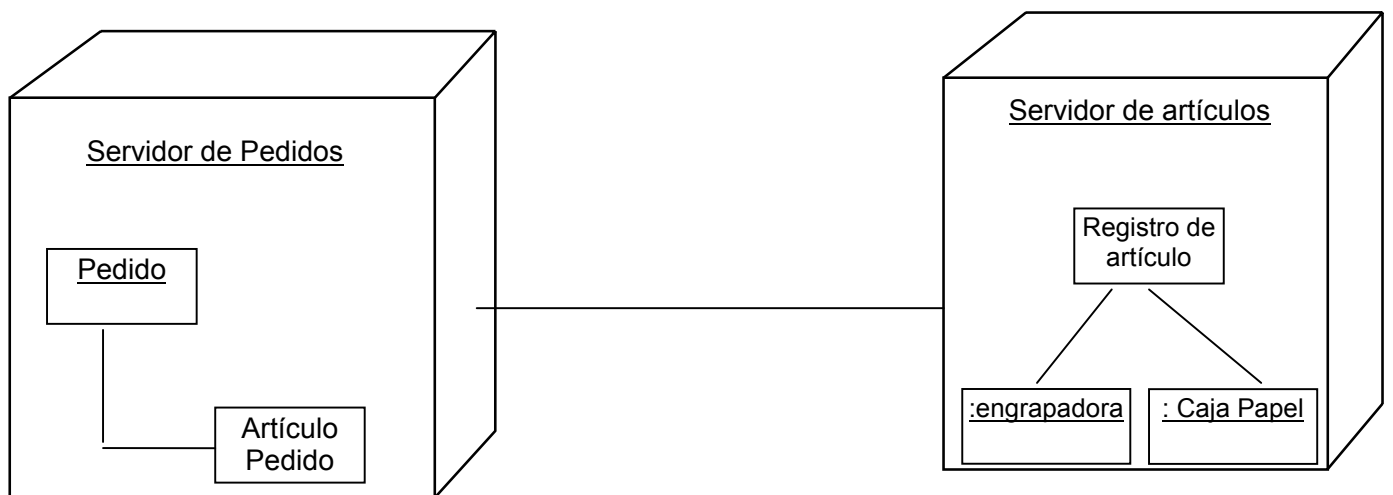
Ej.: Puede representarse un sistema de distribución como tal.

Igual que con las clases, se subrayan los nombres de los procesadores para indicar instancias (instancias de hardware).



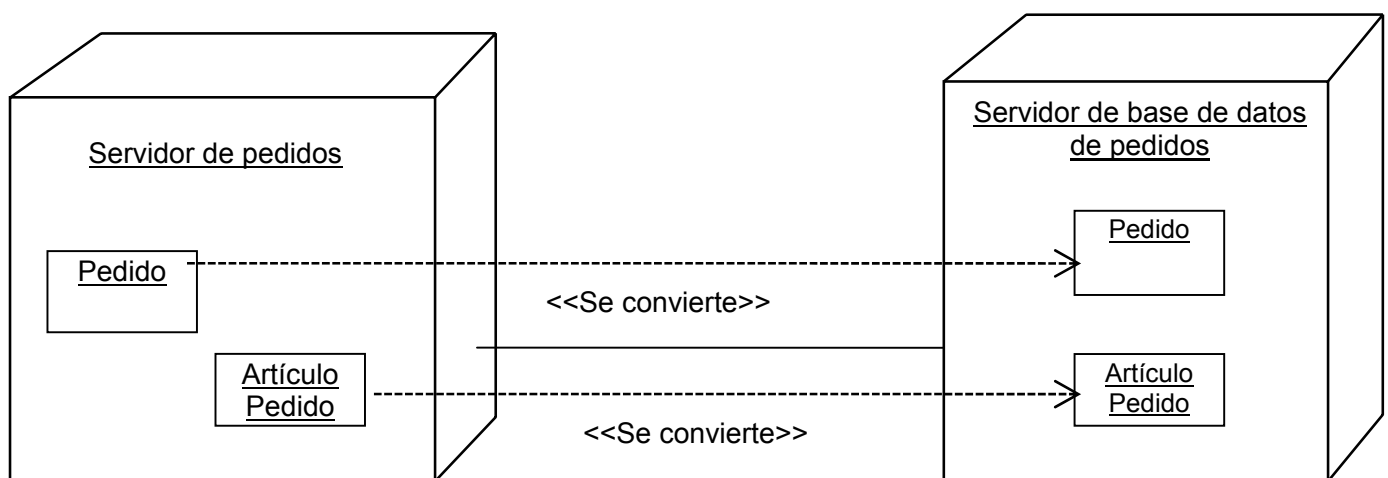
Se puede ilustrar que objetos corren en qué máquinas.

Ej.: Se puede mostrar el contenido de los procesadores del sistema que hemos ejemplificado.



Podemos mostrar un objeto que pasó de un nodo a otro.

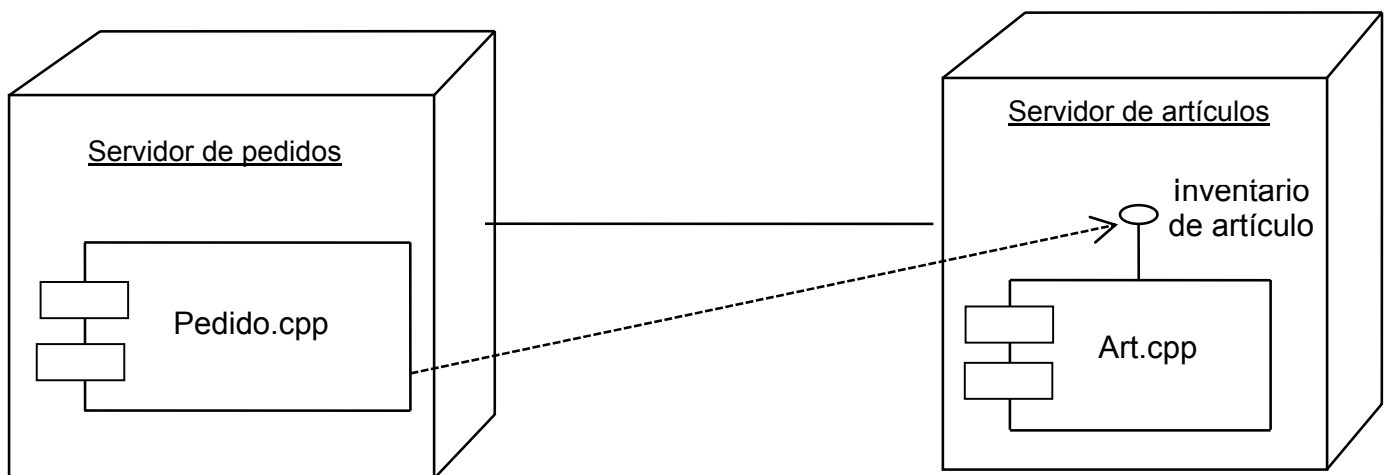
Ejm.: Suponer que pedidos y artículos pedidos se almacenan en una base de datos. Pueden pasar del servidor de pedidos a la base de datos del servidor.



Puede mostrarse qué componentes residen en qué nodos.

Ejem.: implementa una interface de Inventario de artículos que Pedido.cpp usa .

Supongamos que los dos archivos están en diferentes procesadores.



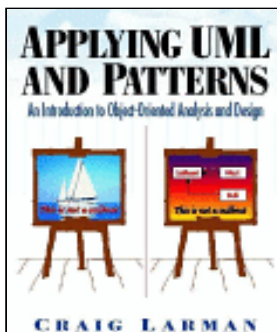
# **VII**

## **7 Bibliografía**

# Applying Uml and Patterns : An Introduction to Object-Oriented Analysis and Design

Por Craig Larman

## Reseñas Amazon.com



Escrito para el desarrollador con experiencia previa en programación y diseño, *Applying UML and Patterns* combina UML, patrones de software y Java para ilustrar la estrategia de diseño del autor. Aunque el autor Craig Larman en ocasiones utiliza mucho la jerga de ingeniería de software, no hay duda que este libro contiene algunas ideas útiles al instante, usando la última y más grande investigación en ingeniería de software.

Este libro comienza delineando un proceso básico de diseño de software, usando técnicas object-oriented iterativas. El caso de estudio utilizado para este texto es un sistema de Punto de Venta, un útil sistema del mundo real. El libro construye diagramas de caso de uso y (básicos) modelos conceptuales y de clases para este sistema. Entonces, el autor agrega diagramas de secuencia para mostrar como el sistema de Punto de Venta (PV) realizará su proceso y diagramas de colaboración para mostrar como interactuarán los



objetos unos con otros. El autor utiliza diagramas UML estándar para documentar el diseño.

Cuando llega el momento de refinar el diseño de clases, la experiencia del autor con patrones realmente brilla. Su patrón GRASP (General Responsibility Assignment Software Patterns) sugiere lineamientos para diseñar clases que trabajen juntos de manera efectiva. Larman cree que la habilidad para asignar responsabilidades a las clases de manera efectiva es uno de los aspectos más importantes de un buen diseño orientado a objetos. Sus patrones permiten que esto suceda y proveen una interesante contribución al diseño de procesos. (El autor también introduce patrones de software más ampliamente usados para mejorar el proceso de diseño.)

Cuando pasa a codificar el diseño, Java es el lenguaje de programación elegido para este texto. Capítulos posteriores discuten como refinar un diseño inicial usando un proceso iterativo de ingeniería de software. Mientras es poco probable adopten por completo el enfoque de Larman para diseñar software, sus lineamientos — y la aplicación de patrones para diseñar clases, todo documentado usando UML — hacen de éste libro un texto que vale la pena para el lector más experimentado.

**Comentario del autor, Craig Larman.  
Clarman@acm.org, 6 de marzo de 1998**

Gracias por tomar en cuenta éste libro. Se enfoca en los "como hacer" del Análisis y Diseño Orientado a

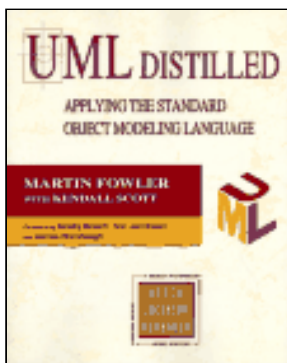
Objetos (A&DOO) en lugar de enfocarse únicamente en la notación UML.. En 500 páginas, se toma un sencillo caso de estudio a través de 2 ciclos iterativos de desarrollo, e ilustra los pasos comunes principales en A&DOO, y la heurística relacionada a varias actividades. También cubre la aplicación de los patrones GoF y GRASP durante la fase de diseño del caso de estudio. El libro termina con una discusión framework de diseño, usando un framework persistente para mapeado hacia el modelo objeto-relacional, como un contexto. Este libro es básicamente de proceso neutral, pero ilustra un proceso ejemplo de actividades comunes y populares, tales como casos de uso, modelado conceptual de objetos, contratos, diagramas de interacción, y demás. Por supuesto, todos los modelos son ilustrados usando UML, y la notación de UML es explicada, pero el énfasis está puesto en contestar preguntas como "¿cómo asigno responsabilidades correctamente?" y "¿cómo creo un modelo conceptual útil?". La audiencia a la que va dirigido son desarrolladores de software (y estudiantes universitarios) que comienzan en A&DOO, mas que a los expertos. También tengo recursos instructivos (tales como exámenes y guías de enseñanza) bajo desarrollo, y estaré instalando un web site instructor en PH pronto.

## Uml Distilled : Applying the Standard Object Modeling Language

(Addison-Wesley Object Technology Series)  
por Martin Fowler, Kendall Scott (colaborador),  
Grady Booch

### Reseñas

Libro recomendado por el editor de [Computer Programming](#)



Muchos programadores tienen poco tiempo para mantenerse al tanto de los últimos cambios del mundo de la ingeniería de software. *UML Distilled: Applying the Standard Object Modeling Language* provee una rápida y útil visión del más importante desarrollo naciente: el surgimiento de UML (Unified Modeling Language).

*UML Distilled* ofrece una útil perspectiva de lo que UML es y para qué es bueno. El autor, un experimentado ingeniero de software, da sus propias opiniones sobre los mejores usos. El libro no es doctrinario y siempre está a favor de utilizar el sentido común en el diseño, en lugar de ajustarse estrictamente a modelos y documentos. Corre a través de la notación básica utilizada en UML para diseño de documentos tales como casos de uso, diagramas de

clases, secuencia, estados, actividad, y de distribución. Aunado a esto, incluye ejemplos concisos de los detalles inherentes a trabajar con objetos, con un excelente desglose paso a paso de los detalles involucrados en UML. El autor incluye algún código para que usted pueda ver hacia donde van todos estos documentos de diseño.

Necesitará alguna idea de los que es ingeniería de software para poder beneficiarse de este libro. De cualquier manera, si tiene los conocimientos apropiados, encontrara este libro invaluable para la comprensión de este naciente nuevo estándar, el cual tiene el potencial de traer ingeniería de software sólida a muchos programadores que nunca han utilizado antes técnicas disciplinadas de diseño de software.

## **Descripción del libro**

El día de hoy, un arquitecto o diseñador de software que busca representar el diseño de un sistema de software puede escoger entre una amplia variedad de lenguajes de notación, cada uno alineado con una metodología de análisis y diseño particular. Irónicamente, esta amplia variedad de elecciones es uno de los impedimentos para los significativos beneficios prometidos por el reuso de software. El surgimiento de UML (Unified Modeling Language) — creado por los esfuerzos conjuntos de los líderes en tecnología de objetos Grady Booch, Ivar Jacobson y James Rumbaugh con colaboraciones de muchos otros en la comunidad de los objetos— representa uno

de los más significativos desarrollos en tecnología de objetos. Apoyado por una amplia base de compañías líderes en la industria, el UML fusiona lo mejor de las notaciones usadas por las tres metodologías de análisis y diseño más populares, Booch, OOSE (casos de uso), y OMT, para producir un sencillo y universal lenguaje de modelado que puede ser usado con cualquier método.

Escrito para aquellos con conocimientos básicos de análisis y diseño orientado a objetos, esta concisa visión general lo introduce a UML, resaltando los elementos claves de su notación, semántica y procesos. Se incluye una breve explicación de la historia de UML, su desarrollo y fundamento, así como discusiones de cómo UML puede ser integrado en el proceso de desarrollo orientado a objetos. Además, este libro perfila varias técnicas de modelado asociadas con los casos de uso de UML, tarjetas CRC, diseño por contrato, clasificación dinámica, interfaces, y clases abstractas junto con descripciones concisas de notación y semántica y numerosos consejos perspicaces para su uso efectivo basados en la experiencia del autor. Además, el autor ofrece un vistazo la naciente Proceso de Desarrollo de Software Objectory derivado de las metodologías de Grady Booch, Ivar Jacobson y James Rumbaugh. Para darle una sensación de UML en acción, el libro incluye un ejemplo de programación en Java que perfila la implementación de un diseño basado en UML.

Usted vendrá con una excelente comprensión de lo esencial de UML, percepción de cómo funciona UML

dentro del proceso de desarrollo de software, y fundamentos firmes sobre los cuales podrá expandirse y construir su conocimiento de UML.

## **Acerca del Autor**

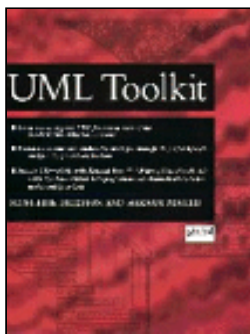
Martin Fowler es un pionero en la aplicación de la tecnología de objetos a sistemas de información. Los últimos diez años, ha realizado consultoría en el área de tecnología de objetos para compañías tales como Citibank, Chrysler Corporation, Xerox, AT&T y UK National Health Service. Es el autor del aclamado libro *Analysis Patterns: Reusable Object Models*. Kendall Scott es un escritor técnico en Oracle Corporation quien ha publicado un número de manuales técnicos en una amplia variedad de tópicos.

## UML Toolkit

By Hans-Erik Eriksson, Magnus Penker

### Reseñas

Libro recomendado por el editor de [Computer Programming](#)



UML (Unified Modeling Language) promete hacer el modelado orientado a objetos mucho más accesible para los desarrolladores de software. UML une los más populares lenguajes de modelado desarrollados por los así llamados "tres amigos" (Grady Booch, James Rumbaugh e Ivar Jacobson) bajo un estándar abierto y *UML Toolkit* puede ayudarlo a usarlo en su próximo proyecto de desarrollo de software.

Los autores detallan la (aproximadamente) docena de diagramas disponibles en UML, lo cual incluye modelado de casos de uso, diagramas de clase, modelos dinámicos (incluyendo diagramas de estado), y modelos físicos (lo cual perfila a los componentes dentro de su sistema y como serán distribuidos). También discuten extensiones de UML (a través de "estereotipos") y documenta que tan correctamente está UML implementado en las herramientas CASE actuales. El libro incluye ejemplos de implementaciones de diagramas de modelado, escritos

en Java, para la mayoría de los capítulos, y el CD-ROM incluido contiene una versión de evaluación de la herramienta CASE Rational Rose.

## **Synopsis**

Los lenguajes de modelado son usados para diseñar y trazar futuros sistemas de computo. UML (Unified Modeling Language) es un estándar abierto para un lenguaje de modelado gráfico creado por Rational Software Corporation. Este libro muestra a los programadores como usar UML para diseñar sistemas orientados a objetos. El CD-ROM contiene una versión de demostración de Rational Rose 4.0, todo el código en Java y los modelos UML que se encuentran en el libro, y la especificación 1.0 de UML junto con Adobe Acrobat Reader.

Con la liberación de UML, los desarrolladores que utilizan el paradigma de orientación a objetos tienen al menos un lenguaje común para modelar y desarrollar sistemas de software. Eso significa menos tiempo desperdiciado sorteando términos conflictivos y pasar más tiempo modelando mejores sistemas. Ahora, este poderoso paquete libro/CD lo arma con todo lo necesario para obtener el máximo provecho de UML y la gran cantidad de productos basados en UML que crece rápidamente. Los autores los guiarán a través de todo el lenguaje, proveyendolo con lineamientos fáciles de seguir y montones de ejemplos de sistemas reales. También le darán explicaciones detalladas de todos los diagramas UML, un caso de estudio de tamaño



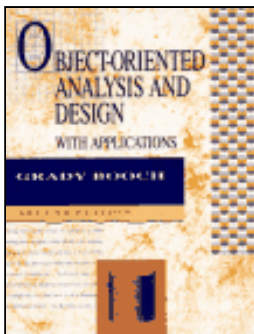
completo mostrando como es usado UML para desarrollar una aplicación, un glosario visual de toda la notación UML, e instrucciones paso por paso de cómo mover Booch, OMT y Objectory hacia UML; modelar Java hacia UML — incluyendo muchos totalmente explotados; define patrones de diseño y como usar patrones en UML; describe sistemas de tiempo real en UML; y emplear casos de uso para capturar los requerimientos funcionales de un sistema.

## Object-Oriented Analysis and Design With Applications

(Addison-Wesley Object Technology Series) by Grady Booch

### Reviews

#### Amazon.com



Object-Oriented Analysis and Design strikes a fine balance between technical details and big-picture information, informing the neophyte of the concepts he or she needs in order to engage in elementary object-oriented software engineering, as well as providing more advanced topics and real-world situations. Booch opens with a discussion of the need for a way of modeling complex systems. He explains all the key concepts--inheritance, encapsulation, and more--in language that is clear and easy to follow. He then moves on to describe his way of analyzing problems and project goals, devoting a fair amount of this section to management issues such as release management and quality assurance. The final part of the book will prove most useful for programmers who already understand object-oriented technologies. The author explores several programming problems, including a cryptography system and a traffic-management system. These examples take the form of architectural discussions, with some key source code--mostly in C++--thrown in. Appendices discuss the

relative merits of all the important object-oriented languages, with the exception of the newcomer Java.

## **Synopsis**

This revised new edition of the bestselling Object-Oriented Design with Applications answers the call for an industry standard in the notation and process for developing object-oriented systems. Gary Booch has now codified an approach to stand as the standard for the industry.

## **Booknews, Inc. , January 1, 1994**

Intended for anyone who implements or manages object technologies, distinguishes between good and bad object-oriented analysis and design and shows how to evaluate architectural tradeoffs to manage complexity. Presents a new, unified notation that incorporates the best ideas from Booch's notation (of the first edition) and other widely-used methods. Includes numerous examples of real-world projects, all of which are now implemented in C++. Annotation copyright Book News, Inc. Portland, Or.

## Object-Oriented Modeling and Design

By James Rumbaugh, Michael Blaha  
(Contributor), William Premerlani, Frederick  
Eddy, Bill Lorensen, William Lorenson  
(Contributor)

### Reviews

#### Amazon.com



Notable mainly for its clear and thorough exploration of the Object Modeling Technique (OMT)--a generic way of representing objects and their relationships--this book is good as a primer and great as a knowledge booster for those already familiar with object-oriented concepts. Object-Oriented Modeling and Design teaches you how to approach problems by breaking them down into interrelated pieces, then implementing the pieces. In addition to its documentation of the Object Modeling Technique (OMT), a graphical notation for depicting object-oriented systems, Object-Oriented Modeling and Design does a first-rate job of explaining basic and advanced object-orientation concepts. The book then moves on to explain the authors' techniques for breaking down problems into components and figuring out systems of interrelated objects that can be used as

designs for programs. Interestingly, the authors devote part of their book to implementing object-oriented solutions in non-object-oriented languages--mainly C, Ada, and Fortran. There's also a great discussion of implementing object-oriented designs in relational database environments.

The authors conclude their book with a sort of recipe section, detailing architectures for various types of programs in OMT.

**The publisher, Prentice-Hall  
Engineering/Science/Mathematics**

This text applies object-oriented techniques to the entire software development cycle.

## **Object-Oriented Software Engineering : A Use Case Driven Approach**

(Addison-Wesley Object Technology Series) by Ivar Jacobson



### **Reviews** **Amazon.com**

A text on industrial system development using object- oriented techniques, rather than a book on object-oriented programming. Will be useful to systems developers and those seeking a deeper understanding of object orientation as it relates to the development process.

## **Journal of Object-Oriented Programming**

Perhaps the most profound and deeply revealing volume on object technology to date...It is simply a must-own book. -- Steve Bilow

## **Object Technology International**

Jacobson is in my opinion one of the foremost methodologists in the field of Software Engineering...I strongly recommend...this book...not only for software managers and designers but for anyone who wishes to understand how the next generation of Software

Systems should be built. -- Dave Thomas, Object Technology International

**Booknews, Inc. , December 1, 1992**

Describes industrial systems development using object-oriented techniques, with emphasis on analysis, design, and testing. Assumes no previous knowledge of systems development and is written in a modular style for selective study. Annotation copyright Book News, Inc. Portland, Or.

Nombre de archivo: IntroduccionUML

Directorio:

C:\alejandra\MANUALES\MANUAL  
ES DEL IX DIPLOMADO VERSIÓN 2

Plantilla: C:\Documents and  
Settings\Adm\Datos de  
programa\Microsoft\Plantillas\Normal.dot

Título: U M L

Asunto:

Autor: Rin Cruz

Palabras clave:

Comentarios:

Fecha de creación: 22/11/2003 11:55

Cambio número: 17

Guardado el: 24/11/2003 12:42

Guardado por: Adm

Tiempo de edición: 20 minutos

Impreso el: 24/11/2003 1:40

Última impresión completa

Número de páginas: 159

Número de palabras: 10,693 (aprox.)

Número de caracteres: 58,813 (aprox.)