

UML

Lenguaje Unificado de Modelado

Introducción. ¿Qué es UML?

- El **Lenguaje Unificado de Modelado** (Unified Modeling Language, UML) es un lenguaje de **modelado visual para sistemas**.
- Está asociado a modelar sistemas de software orientados a objetos pero tiene una aplicación más amplia.
- **No proporciona ningún tipo de metodología de modelado.**
- UML no está unido a ninguna metodología específica o ciclo de vida y se puede utilizar con todas las metodologías existentes.
- UP (Proceso Unificado) y OPEN (Object-oriented Process, Environment and Notation) son metodologías que utilizan UML.

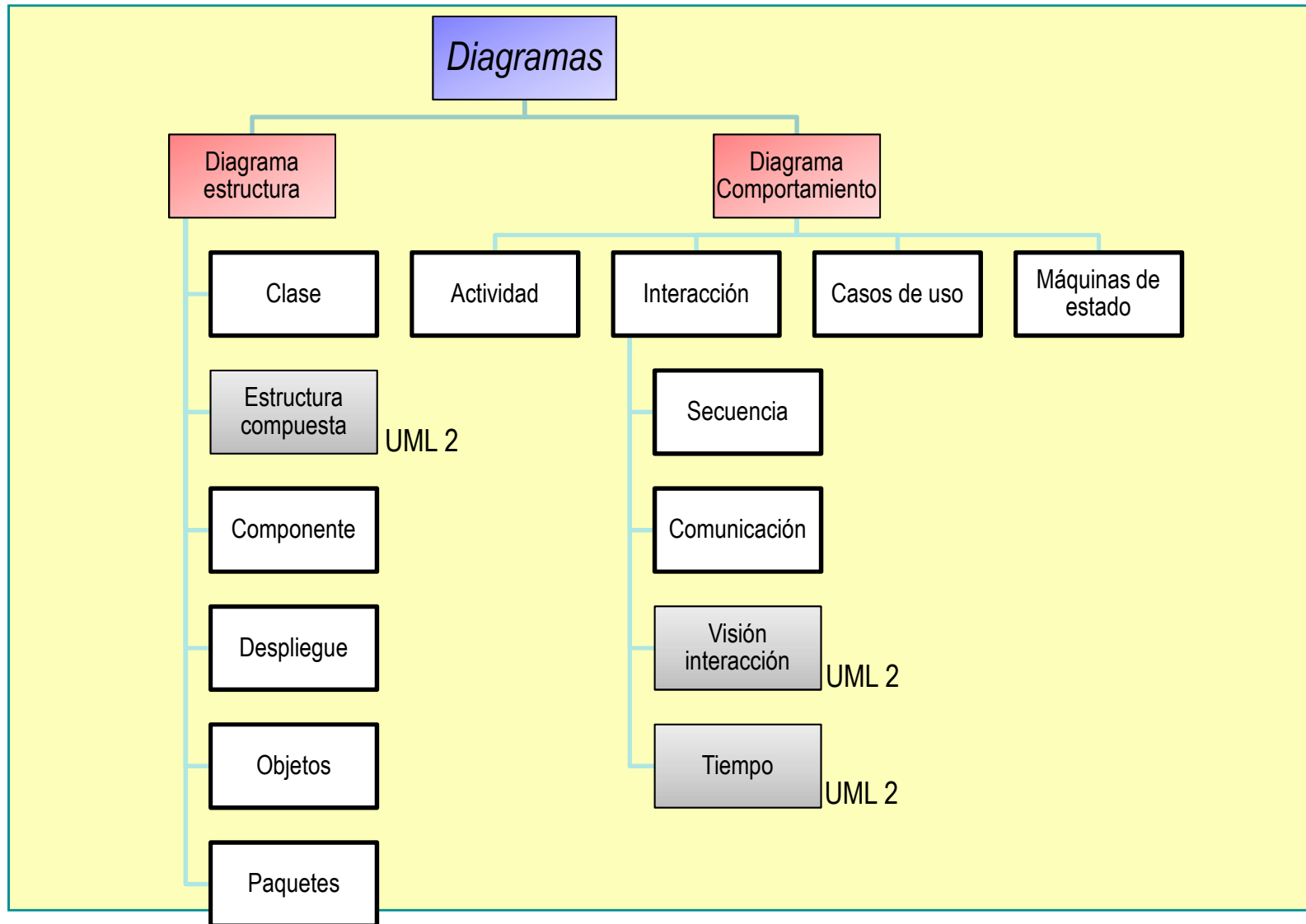
Introducción. ¿Por qué “unificado”?

- La especificación UML trata de estar unificada en diferentes dominios:
 - **Ciclo de vida de desarrollo:** UML proporciona sintaxis visual desde los requisitos hasta la implementación.
 - **Dominios de aplicación:** UML se utiliza para modelar de todo.
 - **Lenguajes y plataformas:** UML es neutro tanto en lenguaje como en plataforma.
 - **Procesos de desarrollo:** aunque UP es el proceso de desarrollo preferido para sistemas orientados a objetos, UML puede soportar otros procesos de ingeniería del software.

Introducción. Diagramas

- Los diagramas son vistas del modelo. Un diagrama no es el modelo en sí.
- El modelo es el repositorio de todos los elementos y relaciones que ha creado para ayudar a describir el comportamiento requerido del sistema de software que está tratando de diseñar.
- UML tiene 13 tipos diferentes de diagramas que podemos dividir en:
 - Modelo estático: captura los elementos y las relaciones estructurales entre elementos.
 - Modelo dinámico: captura cómo los elementos interactúan para generar comportamiento.

Introducción. Diagramas

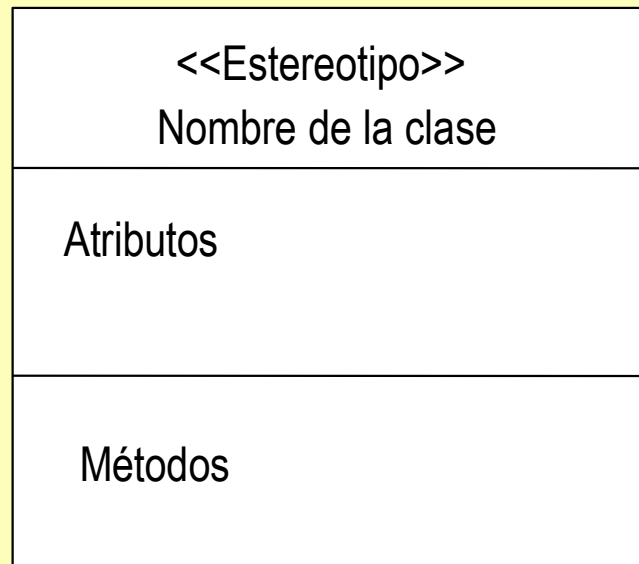


Diagramas de clase.

- Representa la **estructura y comportamiento** de cada uno de los objetos del sistema y sus **relaciones** con los demás objetos.
- Tienen como objetivo representar los **aspectos estáticos** del sistema.
- **No** representa la dinámica de los objetos.

Diagramas de clase. Notación.

■ Notación:



El rectángulo que representa la clase se denomina **Clasificador**.

Diagramas de clase. Notación.

■ Notación:

<<Esterotipo>> Nombre de la clase
Atributos
Métodos

Los estereotipos son **mecanismos de extensibilidad**.

Mediante los estereotipos podemos **diferenciar los elementos de modelado de acuerdo a un uso particular**.

Ejemplos

<<primitivo>> Número imaginario
+ Real: double + Imaginaria: string

<<enumeración>> Tipo Llamada
+ Amóvil + AFijo

<<interface>> IClonable
+ Clone()

Diagramas de clase. Notación.

■ Notación:

<<Estereotipo>> <i>Nombre de la clase</i>
Atributos
Métodos

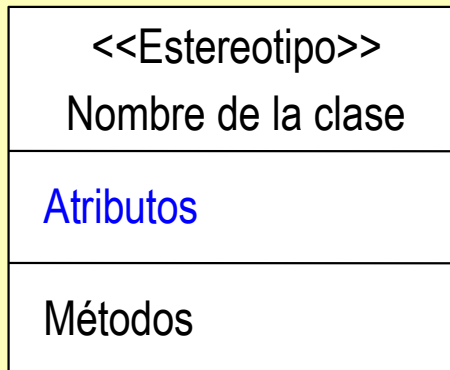
El nombre de clase empezará por una letra mayúscula y el resto irá en minúsculas.

Se evitarán símbolos especiales como signos de puntuación, guiones, subrayados, barras inclinadas, etc.

Cuando el nombre de la clase esté en letra *cursiva* significa que la clase es *abstracta*.

Diagramas de clase. Notación.

■ Notación:



Los atributos representan el estado interno de la clase.

Los atributos se nombran con la primera letra en mayúsculas de cada palabra con la primera letra de todas en minúsculas.

Su modo de representación completo es el siguiente:

visibilidad nombre[multiplicidad]: tipo = valor

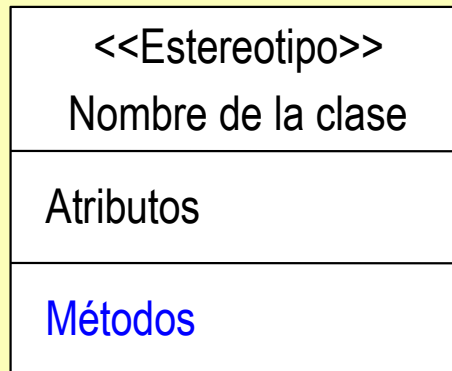
Solo es obligatorio el nombre.

La visibilidad puede ser:

- + : visibilidad pública
- - : visibilidad privada
- # : visibilidad protegida

Diagramas de clase. Notación.

■ Notación:



Los métodos indican el comportamiento de clase.

Se nombran con la primera letra en mayúsculas y el resto en minúsculas.

Los métodos de clase se subrayan.

Su notación es la siguiente:

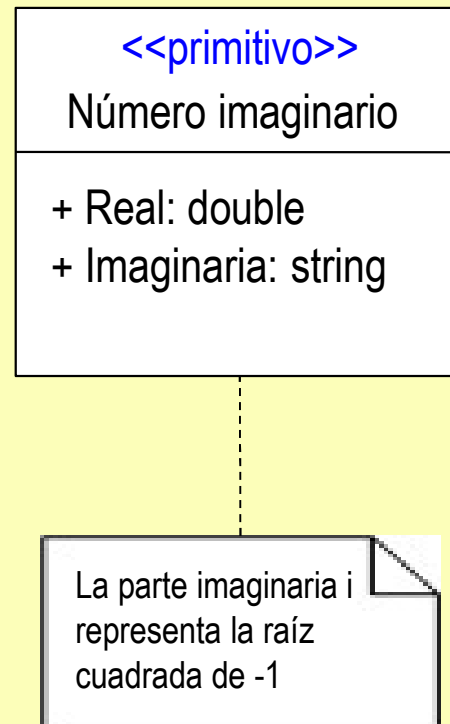
visibilidad nombre (dirección nombreParámetro: TipoParámetro = valorPredeterminado, ...): tipoRetorno



Firma del método

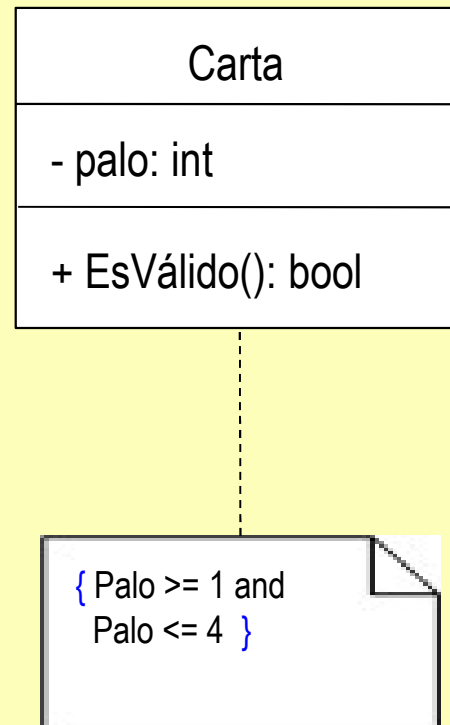
Diagramas de clase. Notación.

■ Comentarios




Diagramas de clase. Notación.

■ Restricciones



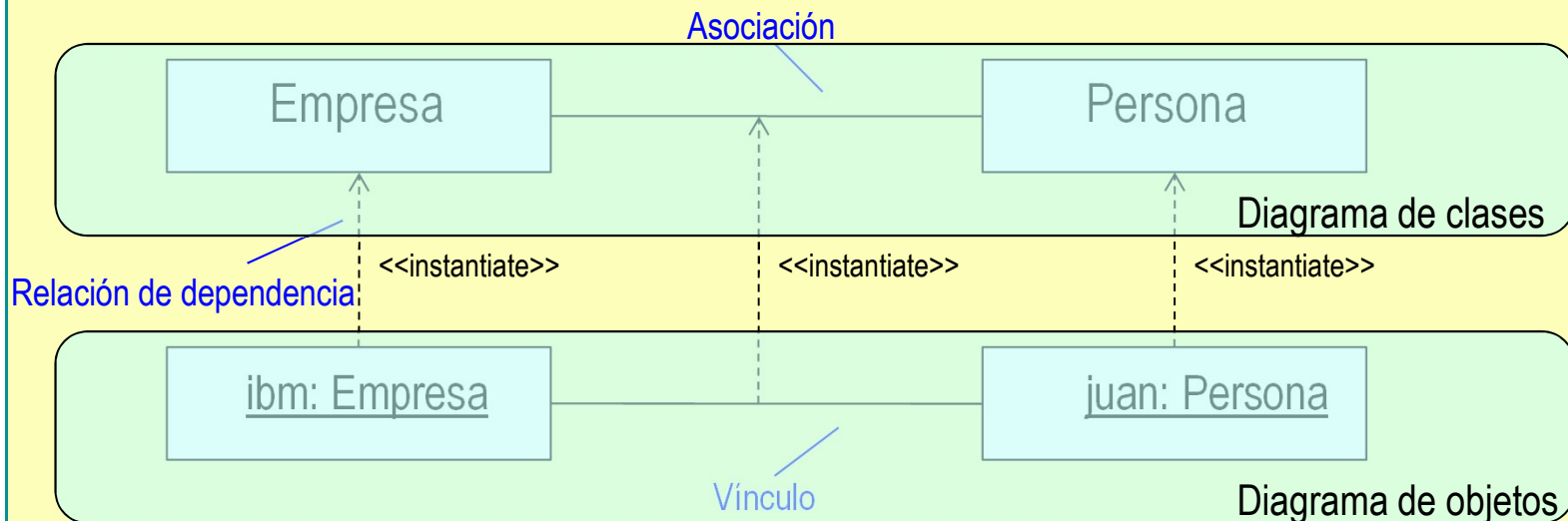
Diagramas de clase. Modelado de relaciones.

- Los diagramas de clases utilizarán principalmente **clasificadores** y conectores que describen las **relaciones** entre las clases.
- Las **relaciones** son **conexiones semánticas** entre **elementos de modelado**.
- Estas relaciones pueden ser:

- Contención 
- Herencia 
- Realización 
- Composición 
- Agregación 
- Dependencia 
- Asociación 

Diagramas de clase. Modelado de relaciones.

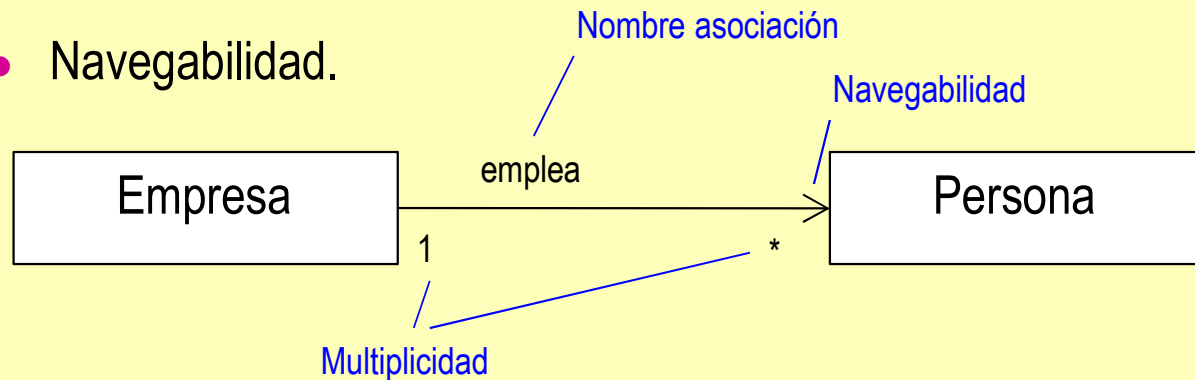
- Para crear un sistema orientado a objetos que funcione, **no puede dejar que los objetos permanezcan solos**, aislados. Necesita **conectarlos** para que puedan realizar un trabajo de utilidad.
- Las **conexiones entre clases** se conocen como **asociaciones**.
- Las **conexiones entre objetos** se conocen como **vínculos**.



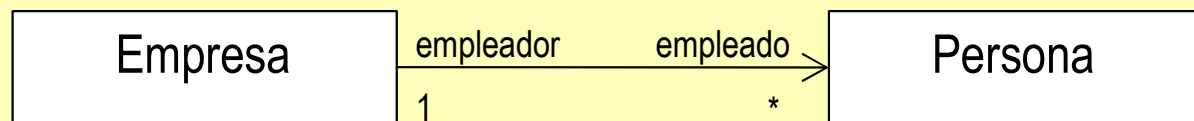
Diagramas de clase. Modelado de relaciones.

- Las asociaciones pueden tener:

- Un nombre de asociación.
- Nombres de roles.
- Multiplicidad.
- Navegabilidad.

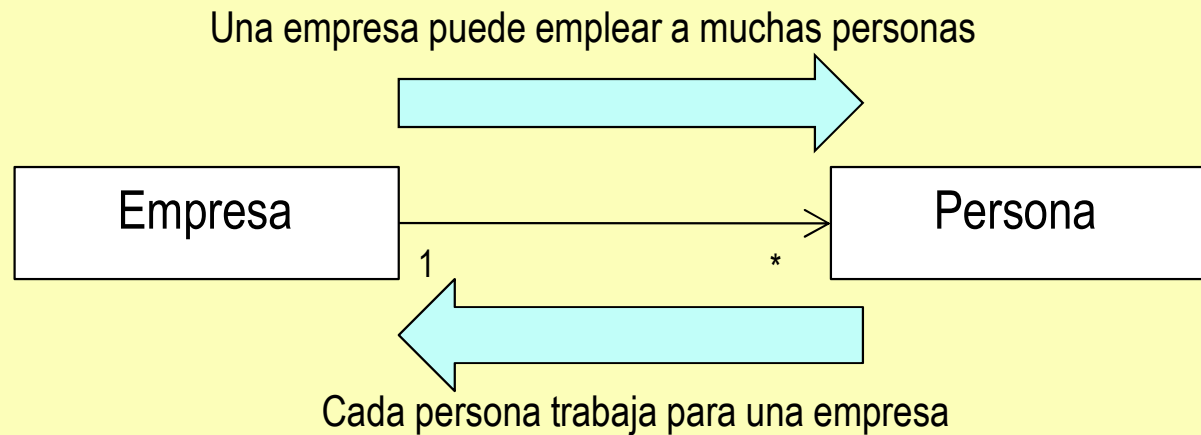


- De forma alternativa, puede asignar nombres de rol a las clases en uno o ambos extremos de la asociación:



Diagramas de clase. Modelado de relaciones. Multiplicidad.

- La **multiplicidad** es el tipo de restricción más común.
- Restringe el número de objetos de una clase que se pueden implicar en una relación determinada.



Diagramas de clase. Modelado de relaciones.

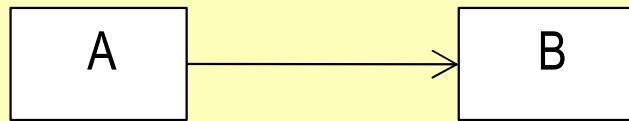
Multiplicidad.

- La multiplicidad se especifica como una lista de intervalos separados por coma, donde cada intervalo es de la forma *mínimo..máximo*
- *mínimo* y *máximo* pueden ser enteros o cualquier expresión que tenga un resultado entero.
- Si la multiplicidad no se indica explícitamente, **no existe multiplicidad predeterminada** en UML.

Adorno	Semántica
0..1	Cero o uno
1	Uno
0..*	Cero o más
*	Cero o más
1..*	Uno o más
1..6	Uno a seis

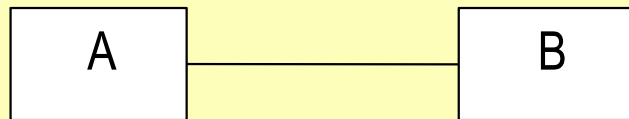
Diagramas de clase. Modelado de relaciones. Navegabilidad.

- La **navegabilidad** nos muestra que es posible pasar desde un objeto de la clase fuente a uno o más objetos de la clase destino, dependiendo de la multiplicidad.



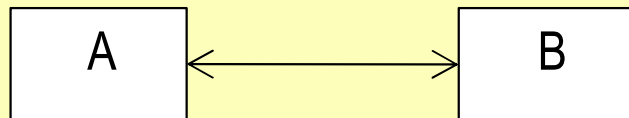
Asociación unidireccional

A a B es navegable
B a A no es navegable



Asociación bidireccional

A a B es navegable
B a A es navegable

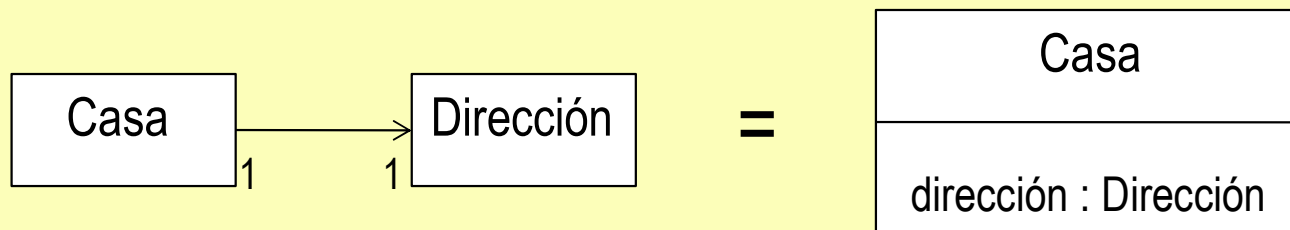


Asociación bidireccional

A a B es navegable
B a A es navegable

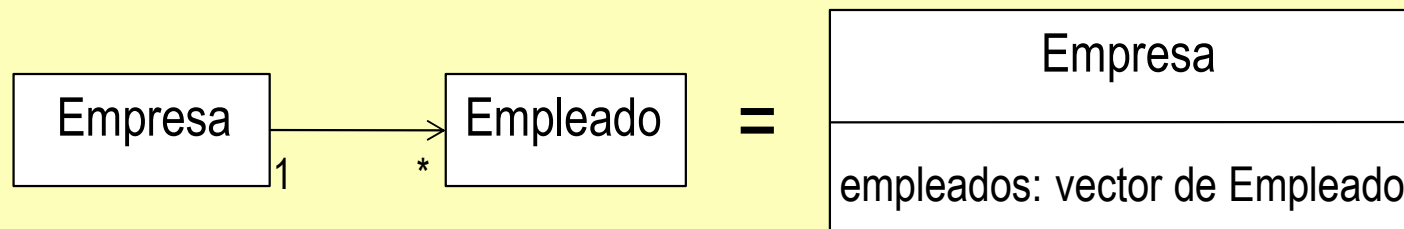
Diagramas de clase. Modelado de relaciones. Navegabilidad.

- Una asociación entre una clase origen y una clase destino significa que los objetos de una clase origen pueden albergar una referencia de objeto a objeto(s) de la clase destino.
- Otra forma de ver esto es que una asociación es equivalente a la clase origen que tiene un atributo de la clase destino.



Diagramas de clase. Modelado de relaciones. Navegabilidad.

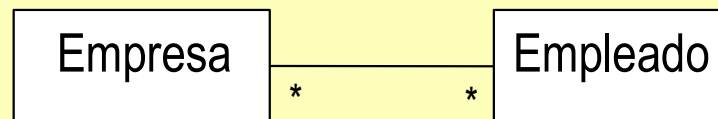
- Las multiplicidades uno a muchos se implementan como un **atributo de tipo vector o colección**.
- Las multiplicidades muchos a muchos no están directamente soportadas en ningún lenguaje orientado a objetos comúnmente utilizado por lo que se debe decidir qué parte de la asociación muchos a muchos es el todo y luego utilizar agregación o composición según sea apropiado.



Diagramas de clase. Modelado de relaciones.

Clases asociación.

- Un problema común en el modelado orientado a objetos es el siguiente: cuando tiene una relación muchos a muchos entre dos clases, a veces existen algunos atributos que no se pueden acomodar fácilmente en ninguna de las clases:

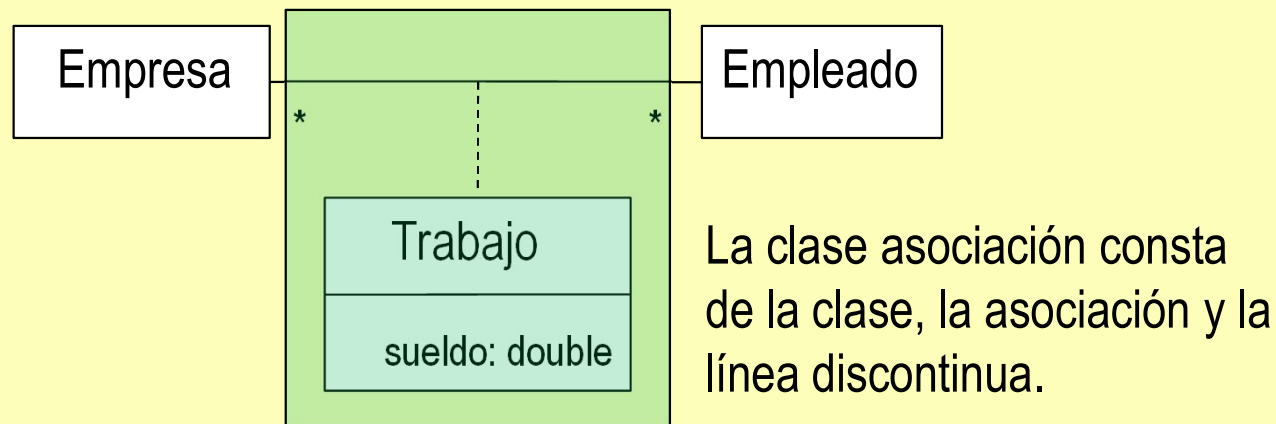


- ¿Dónde situaríamos el **atributo sueldo** del empleado (cada asociación existe un sueldo específico)?
- El sueldo es una propiedad de la propia asociación.

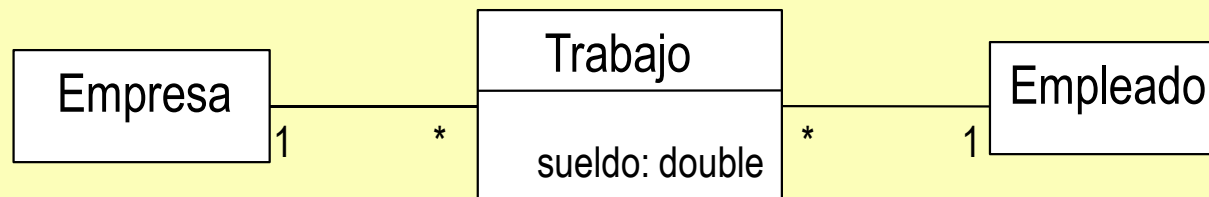
Diagramas de clase. Modelado de relaciones.

Clases asociación.

- UML le permite modelar esta situación con una **clase asociación** como muestra en la siguiente figura:



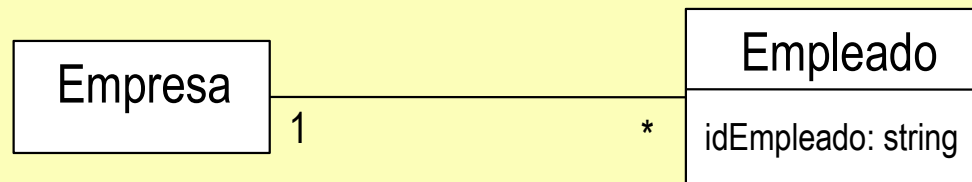
- A la hora de implementarlo en un lenguaje de programación, debe hacerlo como si tuviese el siguiente diagrama de clases:



Diagramas de clase. Modelado de relaciones.

Asociaciones cualificadas.

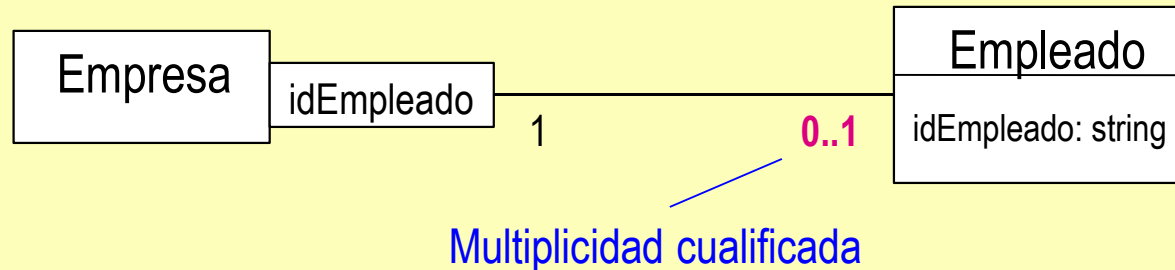
- Podemos utilizar asociaciones cualificadas para reducir una asociación n-a-muchos a una asociación n-a-1 al especificar un objeto único en el destino establecido.



- Podemos plantearnos la siguiente cuestión: si estamos situados en el objeto *Empresa*, ¿cómo podríamos *navegar* hasta un objeto empleado?
- Claramente, necesitamos una clave que identifique unívocamente un empleado. Esto se conoce como *Calificador*.

Diagramas de clase. Modelado de relaciones. Asociaciones cualificadas.

- Es importante fijarse que este calificador pertenece al extremo de la asociación y no a la clase *Empleado*.

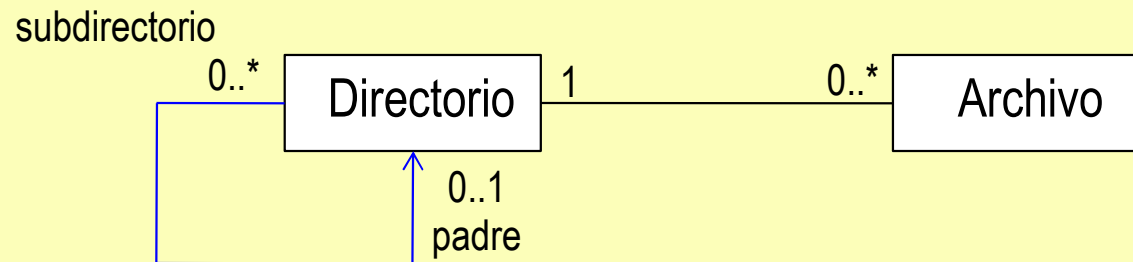


- Las asociaciones cualificadas son una forma estupenda de mostrar cómo se selecciona un objeto específico de un conjunto al utilizar una clave única.
- Los calificadores hacen referencia normalmente a un atributo de la clase destino pero puede haber alguna otra expresión siempre que seleccione un solo objeto del conjunto.

Diagramas de clase. Modelado de relaciones.

Asociaciones reflexivas.

- Es bastante común para una clase tener una asociación consigo misma.
- Esto se denomina **asociación reflexiva** y significa que los objetos de esa clase tienen **vínculos a otros objetos de la misma clase**.



Diagramas de clase. Modelado de relaciones. Dependencias.

- La relación de dependencia se da entre un **cliente** (destino) y un **proveedor** (origen).
- La clase cliente depende de una segunda clase, el proveedor, para proporcionar un servicio.
- Tenemos varios tipos de asociaciones de dependencia:
 - De uso.
 - De abstracción.
 - De permiso.

Diagramas de clase. Modelado de relaciones.

Dependencias de uso.

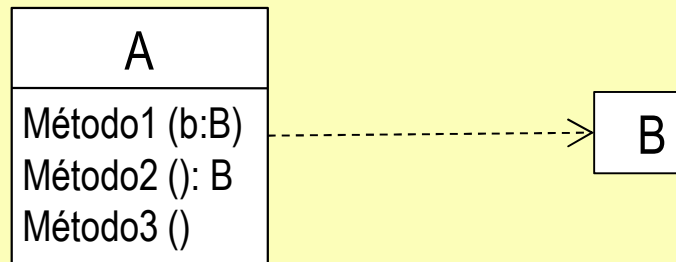
- El cliente utiliza alguno de los servicios puestos a disposición por el proveedor para implementar su propio comportamiento.
- Este es el tipo de dependencia utilizado más comúnmente.
- Hay cinco dependencias de uso:
 - <<use>>
 - <<call>>
 - <<parameter>>
 - <<send>>
 - <<instantiate>>

Diagramas de clase. Modelado de relaciones.

Dependencias de uso.

<<use>>

- Simplemente indica que el cliente hace uso del proveedor de alguna forma.
- Si ve una flecha discontinua de dependencia sin estereotipo, podemos estar seguros de que se trata de una dependencia de uso.



- Esta dependencia está generada por cualquiera de los siguientes casos:
 - Una operación de la clase A necesita un parámetro de la clase B.
 - Una operación de la clase A devuelve un valor de la clase B.
 - Una operación de la clase A utiliza un objeto de la clase B en algún lugar en su implementación, pero no como atributo.

Diagramas de clase. Modelado de relaciones.

Dependencias de uso.

- Un ejemplo del tercer caso (A usa B en algún lugar de la implementación de un método pero no como un atributo) podría ser el siguiente:

```
public class A
{
    void HacerAlgo()
    {
        B miB = new B();
        // Utilizar B de alguna forma
    }
}
```

Diagramas de clase. Modelado de relaciones.

Dependencias de uso.

<<call>>

- La dependencia <<call>> se da entre métodos.
- La operación cliente invoca la operación proveedor.
- Prácticamente no se utiliza y pretende llegar a un nivel más profundo de detalle de lo que pretenden la mayoría de los modeladores.

<<parameter>>

- El proveedor es un parámetro de la operación cliente.

<<send>>

- El cliente es una operación que envía el proveedor (que debe ser una señal) a algún destino sin especificar.

<<instantiate>>

- El cliente es una instancia del proveedor.

Diagramas de clase. Modelado de relaciones.

Dependencias de abstracción.

- Las dependencias de abstracción modelan dependencias entre elementos que se encuentran en diferentes niveles de abstracción.
- Un ejemplo podría ser una clase en un modelo de análisis y la misma clase en el modelo de diseño.
- Existen cuatro tipos:
 - <<trace>>
 - <<substitute>>
 - <<refine>>
 - <<derive>>

Diagramas de clase. Modelado de relaciones. Dependencias de abstracción.

<<trace>>

- Ilustra una relación en la que el proveedor y el cliente representan el mismo concepto pero están en diferentes modelos.

<<substitute>>

- Indica que el cliente se puede sustituir por el proveedor en tiempo de ejecución.
- El proceso de sustitución se basa en que el cliente y el proveedor se ajusten a interfaces comunes.
- Está pensada para aquellos entornos que no soportan generalización/especialización.

Diagramas de clase. Modelado de relaciones. Dependencias de abstracción.

<<refine>>

- Se utiliza entre elementos del mismo modelo
- Por ejemplo, puede tener dos versiones de una clase en un modelo, una de las cuales está optimizada para rendimiento.

<<derive>>

- Se utiliza cuando queremos mostrar que el cliente se puede inferir observando al proveedor.

Diagramas de clase. Modelado de relaciones.

Dependencias de permiso.

- Tratan de expresar la posibilidad de que un elemento acceda a otro elemento.
- Existen tres tipos de dependencia de permiso:
 - <<access>>
 - <<import>>
 - <<permit>>

Diagramas de clase. Modelado de relaciones.

Dependencias de permiso.

<<access>>

- Se da entre paquetes.
- Los paquetes se utilizan en UML para agrupar elementos.
- El punto esencial es que <<access>> permite que un paquete acceda a todos los contenidos públicos de otro paquete.


<<import>>

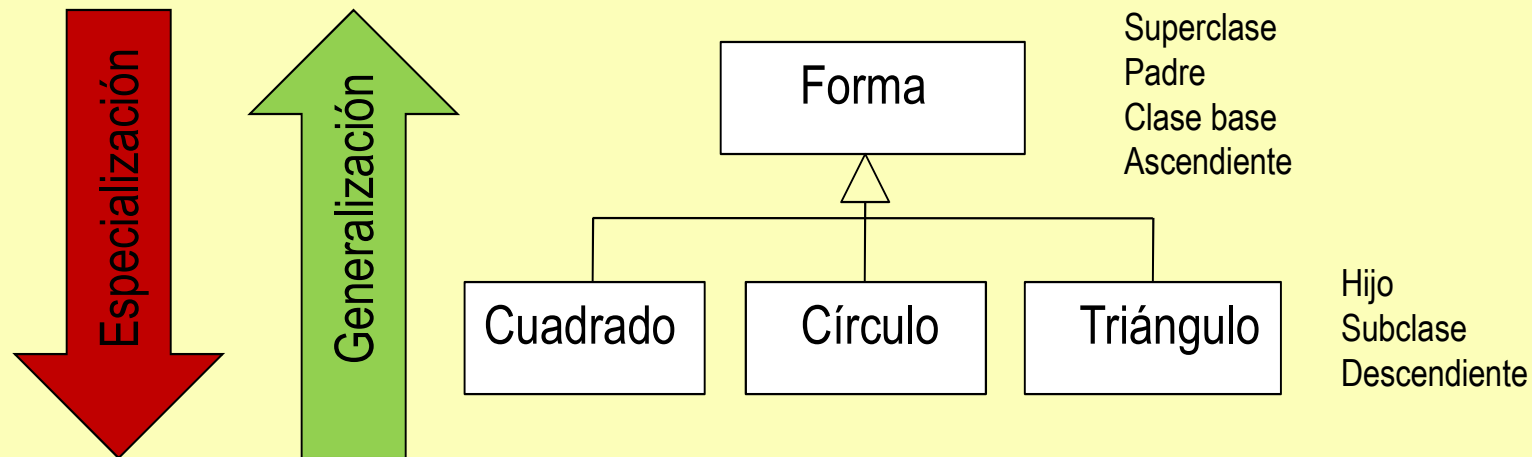
- Es conceptualmente similar a <<access>> excepto que el espacio de nombres del proveedor se fusiona en el espacio de nombres del cliente.

<<permit>>

- Permite una violación de encapsulación controlada, pero se debería evitar.
- C++ permite que una clase declare amigos que tienen permiso para acceder a sus socios privados pero esta característica se ha excluido de java y C#.

Diagramas de clase. Modelado de relaciones. Generalización.

- Las relaciones de herencia se representan con el símbolo 
- Cuando utilice la herencia, las clases hijo heredan las restricciones definidas por todos los antepasados.
- Cada elemento tiene la unión de las restricciones que define y las restricciones definidas por sus antepasados.



Diagramas de clase. Modelado de relaciones.

Generalización.

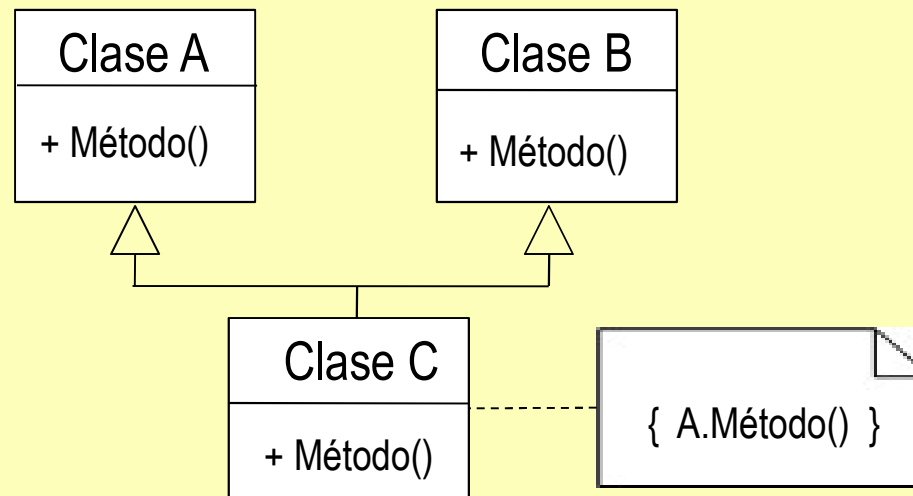
- Cuando examinemos los atributos y operaciones de las clases anteriores, podemos llegar a la jerarquía anterior de dos formas:
 - Por medio de un proceso de **especialización**.
 - Por medio de un proceso de **generalización**.
- En **especialización**, identificaríamos primero el concepto general de *Forma* en análisis y luego lo especializaríamos en sus tipos específicos.
- En **generalización**, identificaríamos los *Cuadrado*, *Círculo* y *Triángulo* más especializados en análisis y luego observaríamos que tienen características comunes que podríamos resolver en una súper clase más general.
- Los analistas orientados a objetos tienden a utilizar la especialización y generalización al mismo tiempo, aunque es mejor acostumbrarse a ver el caso más general lo más pronto posible en el proceso de análisis.

Diagramas de clase. Modelado de relaciones. Generalización.

- Cuando organiza clases en una jerarquía de generalización, implícitamente tiene herencia entre las clases donde las subclases heredan todas las características de sus súper clases.
- Para ser más específico, las clases descendientes heredan:
 - Atributos
 - Operaciones
 - Relaciones
 - Restricciones
- Las subclases también pueden añadir nuevas características y anular operaciones de superclase.

Diagramas de clase. Modelado de relaciones. Generalización.

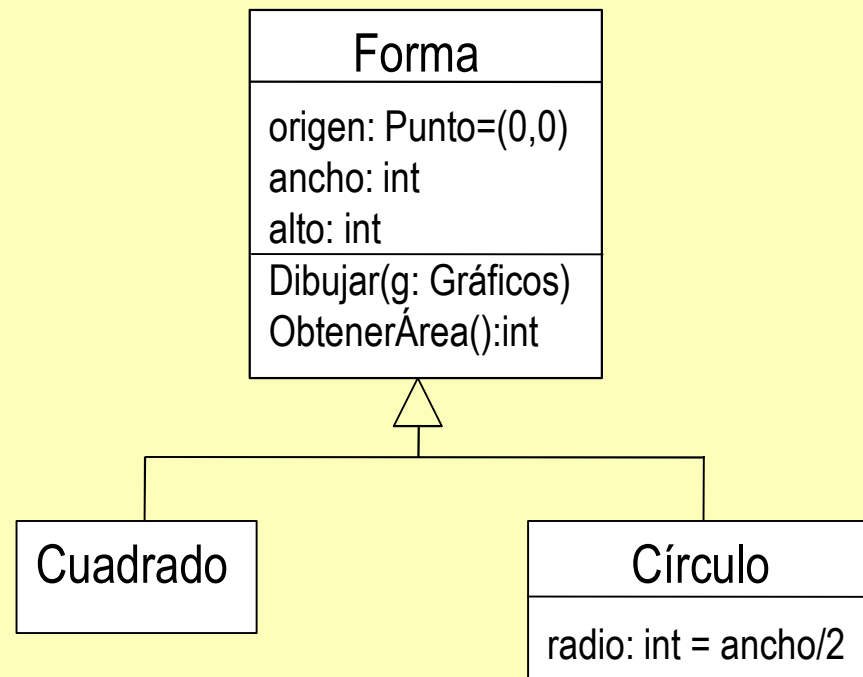
- La **herencia múltiple** se da cuando una clase desciende de dos o más clases simultáneamente.
- Se pueden producir problemas cuando más de una superclase introduce elementos comunes:



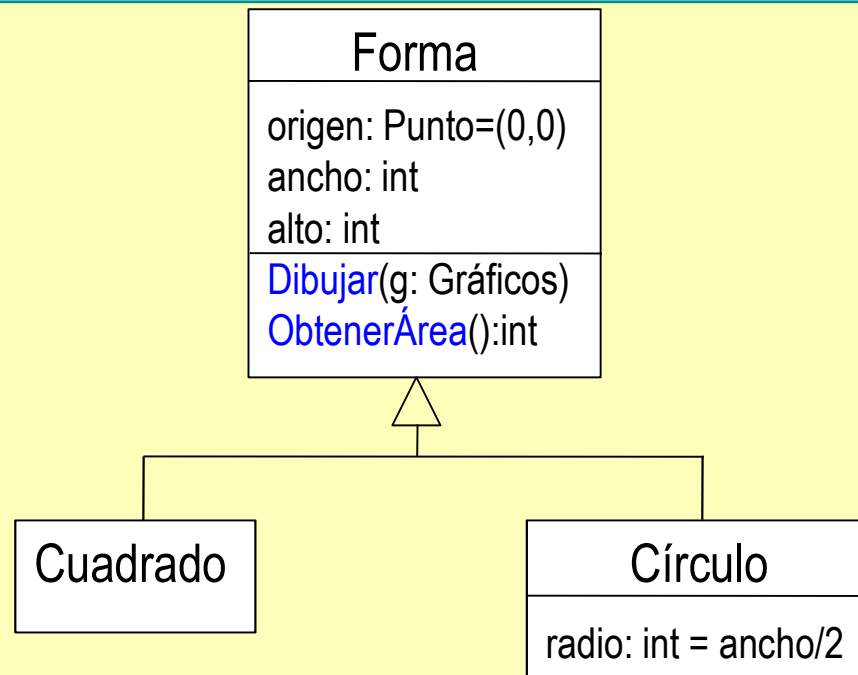
- En estos casos es una buena práctica resolver los conflictos de manera explícita con el uso de una restricción.

Diagramas de clase. Modelado de relaciones. Generalización. Anular.

- En el siguiente ejemplo, las subclases de *Forma*, heredan todos los atributos, operaciones y restricciones de la superclase.
- Esto significa que aunque no ve estas características en las subclases, están allí implícitamente.
- Decimos que Cuadrado y Círculo son tipos de Forma.



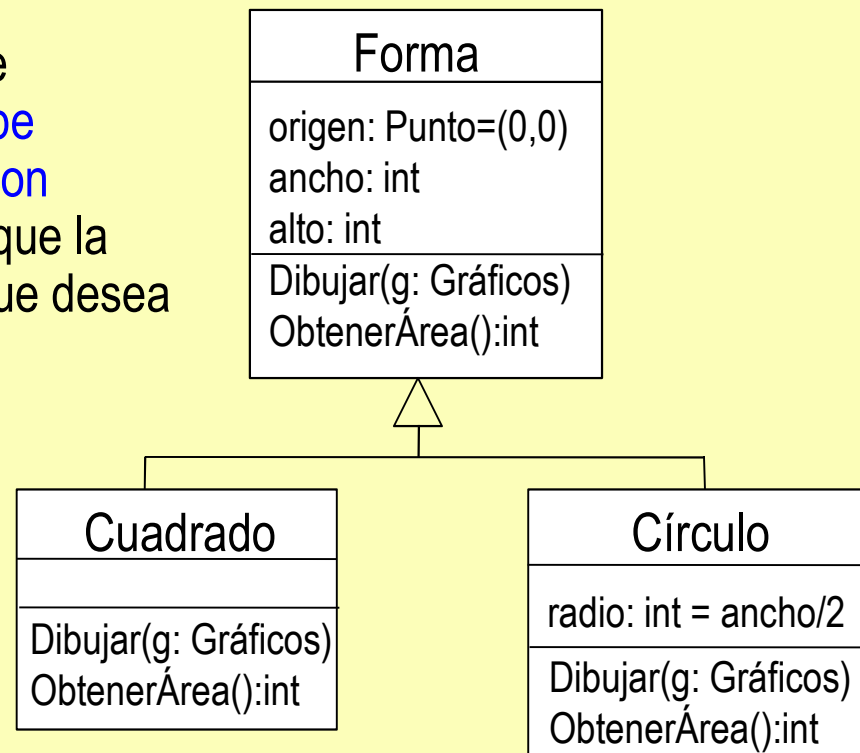
Diagramas de clase. Modelado de relaciones. Generalización. Anular.



- Observe que las operaciones `Dibujar()` y `ObtenerÁrea()` en *Forma* pueden no ser apropiadas para las subclases.
- La operación heredada `Dibujar()` **no sabrá** dibujar un cuadrado ni un Círculo. De hecho, esta operación puede no dibujar nada en absoluto.
¿Cuál debería ser el aspecto de una *Forma*?

Diagramas de clase. Modelado de relaciones. Generalización. Anular.

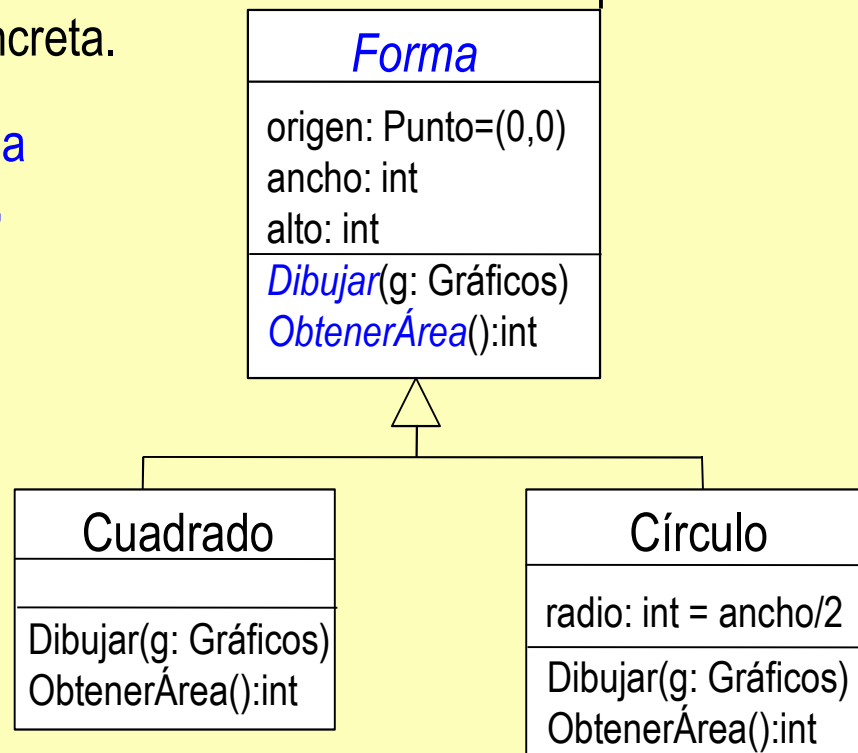
- Estos problemas apuntan a la necesidad de que las subclases puedan cambiar el comportamiento de la superclase.
- Cuadrado y Círculo necesitan implementar sus propias operaciones Dibujar() y ObtenerÁrea() que anulen las operaciones predeterminadas facilitadas por el objeto ancestro.
- Para anular una operación de superclase, una subclase debe proporcionar una operación con exactamente la misma firma que la operación de la superclase que desea anular.



Diagramas de clase. Modelado de relaciones.

Generalización. Operaciones y clases abstractas.

- Fijémonos en la operación `Forma::Dibujar(g: Gráficos)`. No podemos proporcionar ninguna implementación sensata de esta operación en la propia clase `Forma` ya que no sabemos cómo se deberían dibujar las formas.
- El concepto de “Dibujar una forma” es demasiado abstracto para tener cualquier implementación concreta.
- En UML, para indicar que una clase o método es abstracto, se escribe su nombre en cursiva.
- Obviamente, una clase abstracta no se puede instanciar.

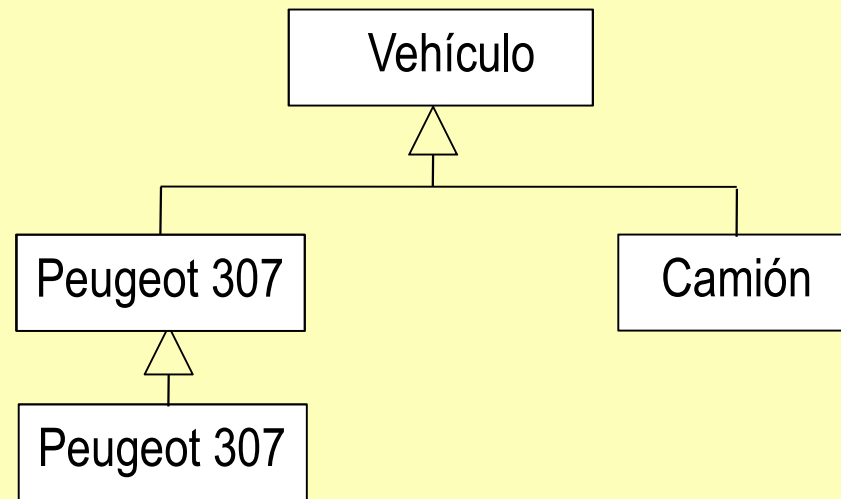


Diagramas de clase. Modelado de relaciones. Generalización. Operaciones y clases abstractas.

- Existen un par de ventajas al utilizar clases y operaciones abstractas:
 - Puede definir un conjunto de operaciones abstractas en la superclase abstracta que se deberían implementar en todas las subclases de Forma. Puede pensar en esto como **definir un “contrato”** que todas las subclases concretas de Forma, deben implementar.
 - Puede escribir código para manipular Formas y luego sustituir Círculo y Cuadrado según sea apropiado. Según el **principio de sustitución**, el código escrito para manipular Formas debería funcionar para todas las subclases de Forma.

Diagramas de clase. Modelado de relaciones. Generalización. Niveles de abstracción.

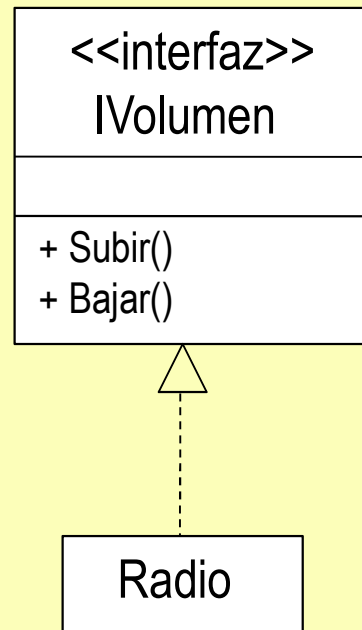
- ¿Qué está mal en el siguiente modelo?:



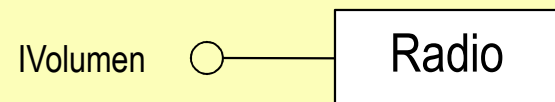
- La respuesta es “los niveles de abstracción”.
- Una jerarquía de generalización define un conjunto de niveles de abstracción desde el más general en la parte superior, al más específico en la parte inferior.
- Siempre se debería **tratar de mantener un nivel uniforme de abstracción en cada nivel de la jerarquía de generalización**.

Diagramas de clase. Interfaces.

- En el diagrama de clases, los interfaces se pueden dibujar de los dos modos siguientes:



Estilo 1

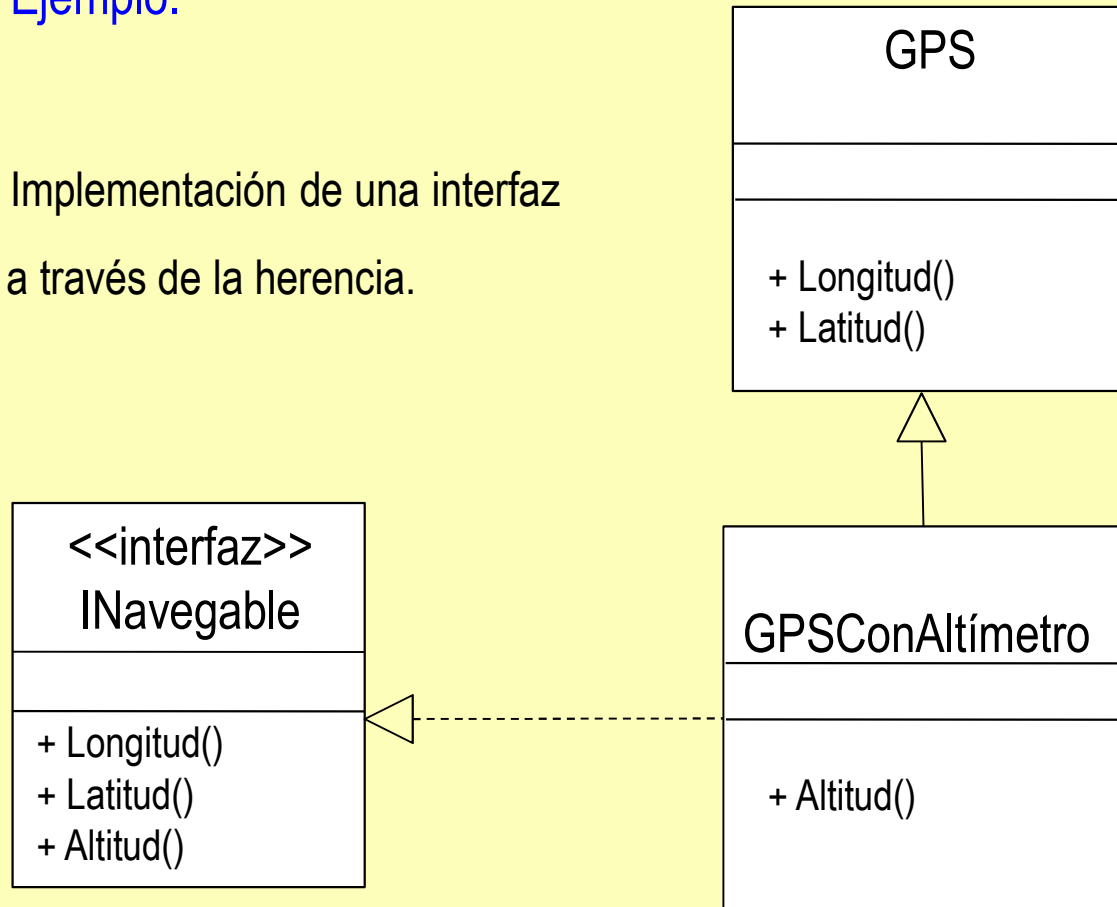


Estilo 2



Diagramas de clase. Interfaces.

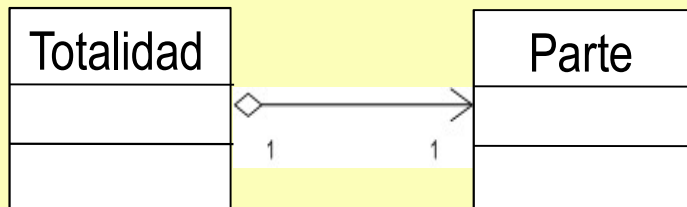
■ Ejemplo.

Implementación de una interfaz
a través de la herencia.

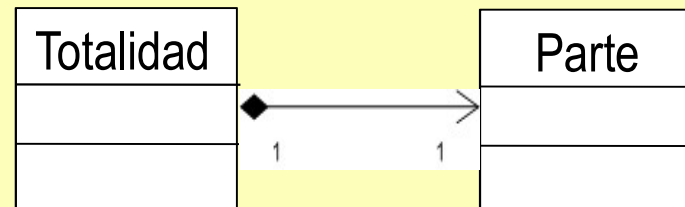


Diagramas de clase. Agregación y composición.

- Las relaciones de agregación y composición tienen que ver con las relaciones de totalidad y parte.
- El conector para la agregación es 
- El conector para la composición es 
- El diamante se agrega al clasificador de totalidad y la flecha al de parte.



Agregación



Composición

Diagramas de clase. Agregación y composición.

- Imaginarse cómo usar la agregación y la composición se puede decidir de manera muy sencilla.
- La composición es agregación, excepto que la clase totalidad es responsable de la creación y de la destrucción de la clase parte, y esta última no puede existir en alguna otra relación al mismo tiempo.
- En una relación de composición hay una regla de “no compartir”, pero los objetos parte se pueden compartir en las relaciones de asociación y agregación. (Fowler)

Diagramas de interacción. Introducción.

- Los **diagramas de interacción** nos muestran cómo se comunican entre sí los objetos.
- Las **interacciones** son sencillas **unidades de comportamiento** de un clasificador.
- Una interacción **puede utilizar cualquiera de las características de su clasificador** (cualquier característica a la que tenga acceso).
- A medida que trabajamos con los diagramas de interacción, vamos descubriendo más métodos y atributos de las clases de análisis.
- Los diagramas de clases **se deberían actualizar** con esta información.

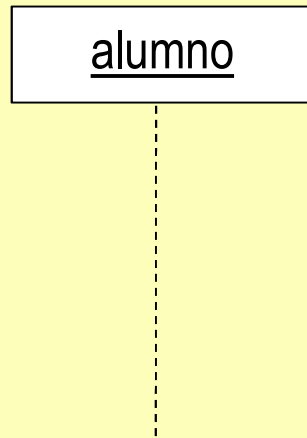
Diagramas de interacción. Introducción.

- Tenemos cuatro tipos de diagramas de interacción:
 - **Diagramas de secuencia:** Enfatizan la secuencia de envíos de mensajes ordenados en el tiempo.
 - **Diagramas de comunicación:** Enfatizan las relaciones estructurales entre objetos.
 - **Diagramas de visión de interacción:** Muestran lo complejo que se realiza el comportamiento por un conjunto de interacciones sencillas.
 - **Diagramas de tiempo:** Enfatizan los aspectos en tiempo real de una interacción.

Diagramas de interacción. Diagramas de secuencia.

Línea de vida.

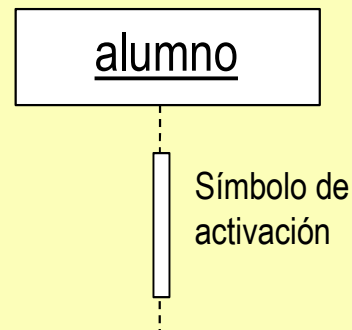
- Una **línea de vida** es un rectángulo con una recta vertical que desciende de ese rectángulo.



- Las líneas de vida **pueden representar actores u objetos.**

Diagramas de interacción. Diagramas de secuencia. Línea de vida.

- Los objetos tienen una duración. Por ejemplo, en un lenguaje determinístico como C++, un objeto vive hasta que se llama al destructor. En C# vive hasta que el recolector de basura lo elimina.
- Desde nuestra perspectiva, sólo nos preocuparemos de cuándo empezamos a usar un objeto y cuándo terminamos de usarlo.
- Es importante saber que un objeto se puede representar como creado y destruido con el uso de una sola línea de vida.
- El símbolo de activación es un rectángulo vertical que reemplaza la línea de vida en el transcurso de la duración de la existencia de ese caso de uso.

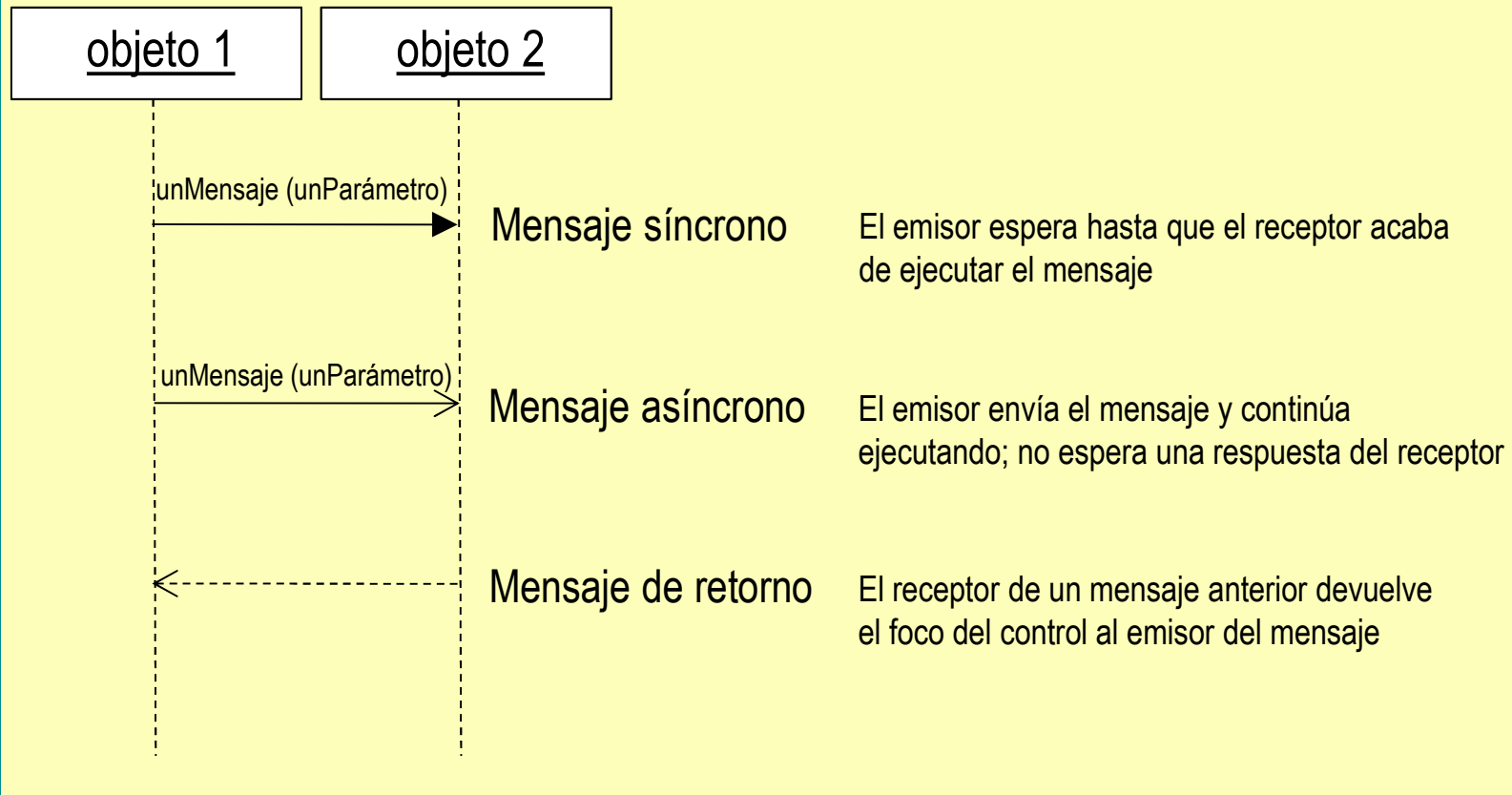


El símbolo de activación
representa la amplitud de la
duración de un objeto

Diagramas de interacción. Diagramas de secuencia.

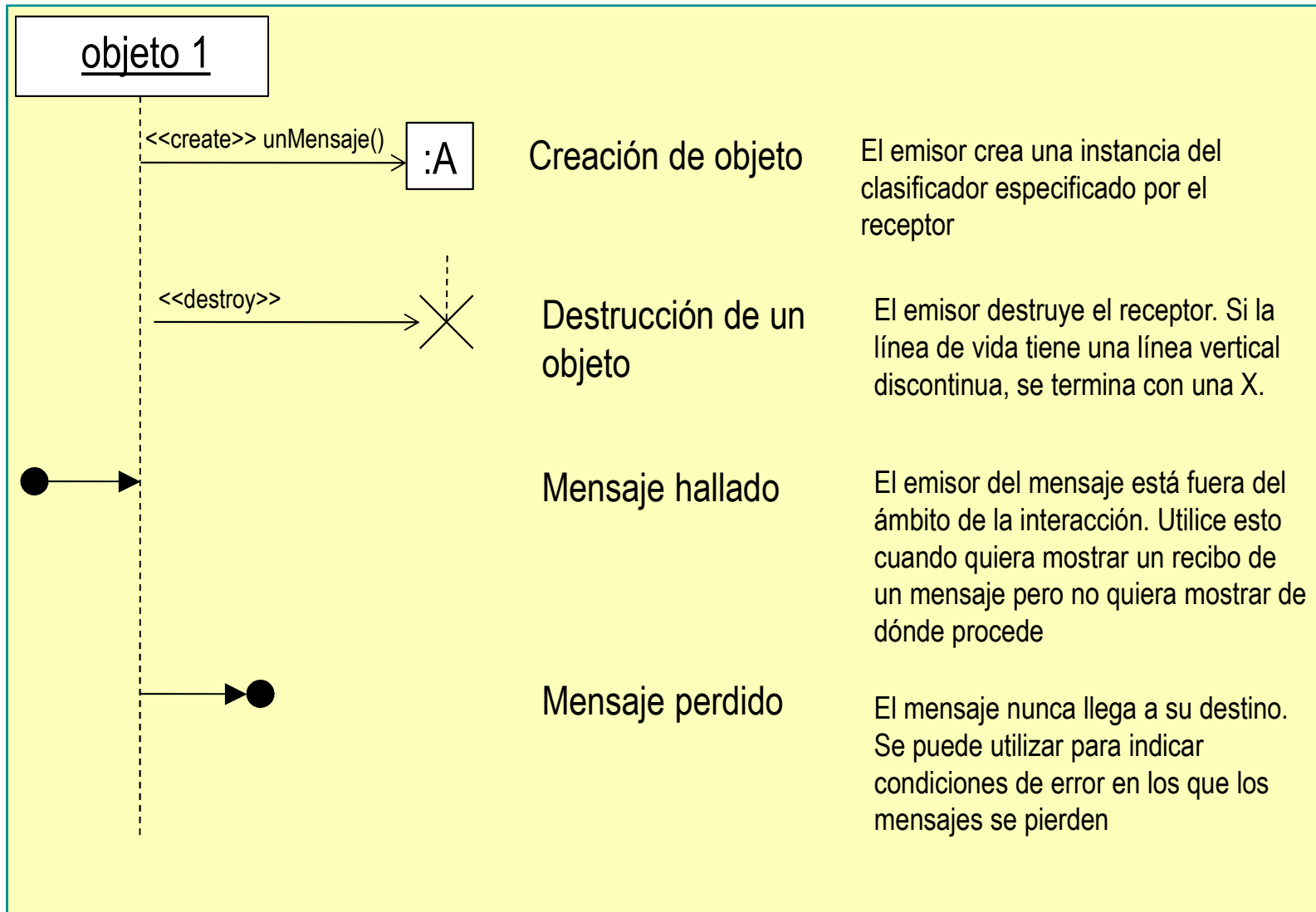
Envío de mensajes.

- Los **mensajes** son **líneas dirigidas que conectan líneas de vida**.
- La **línea se inicia en una línea de vida y la flecha apunta hacia aquella línea de vida que contenga el mensaje invocado**.



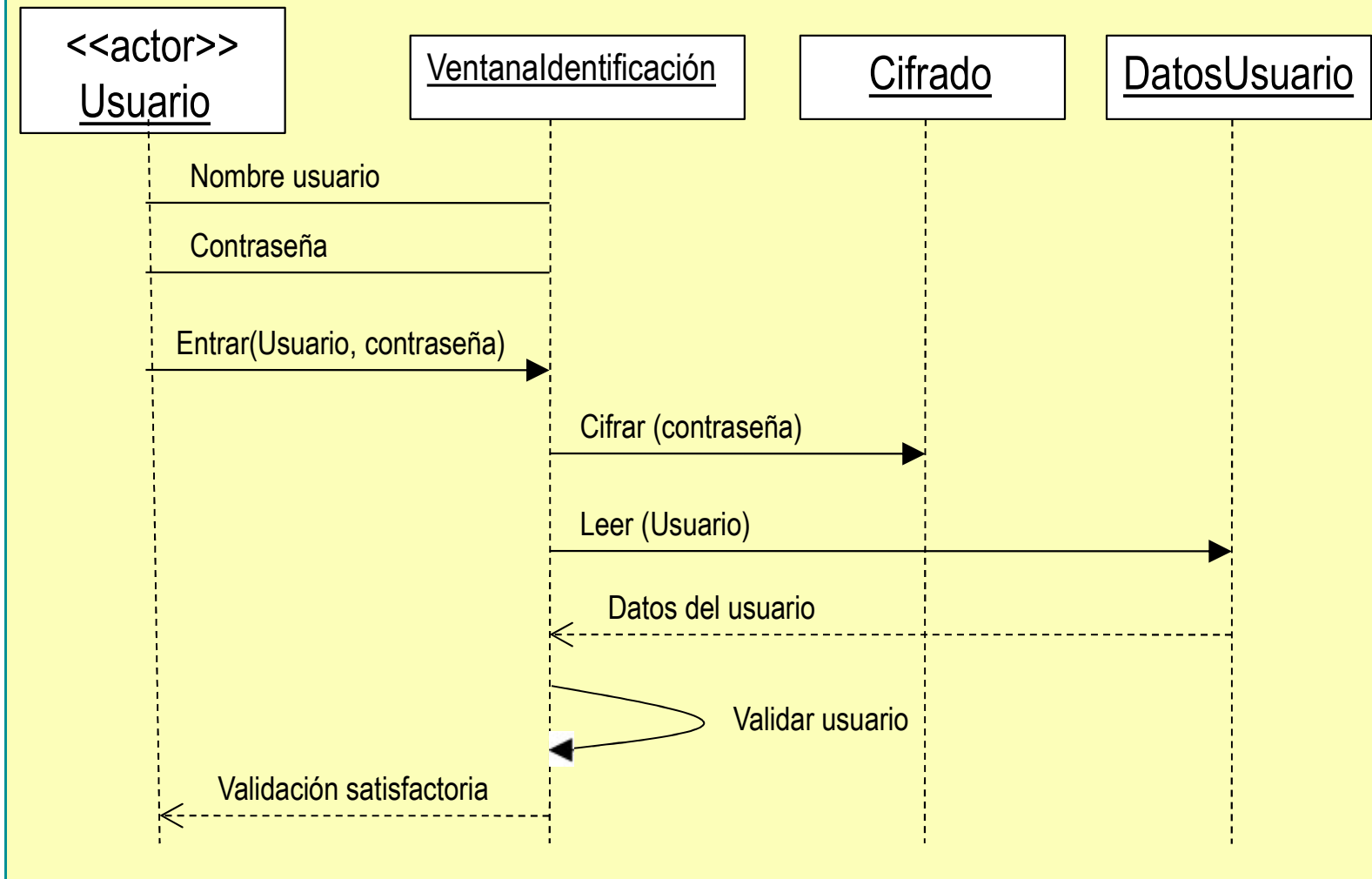
Diagramas de interacción. Diagramas de secuencia.

Envío de mensajes.



Diagramas de interacción. Diagramas de secuencia. Envío de mensajes.

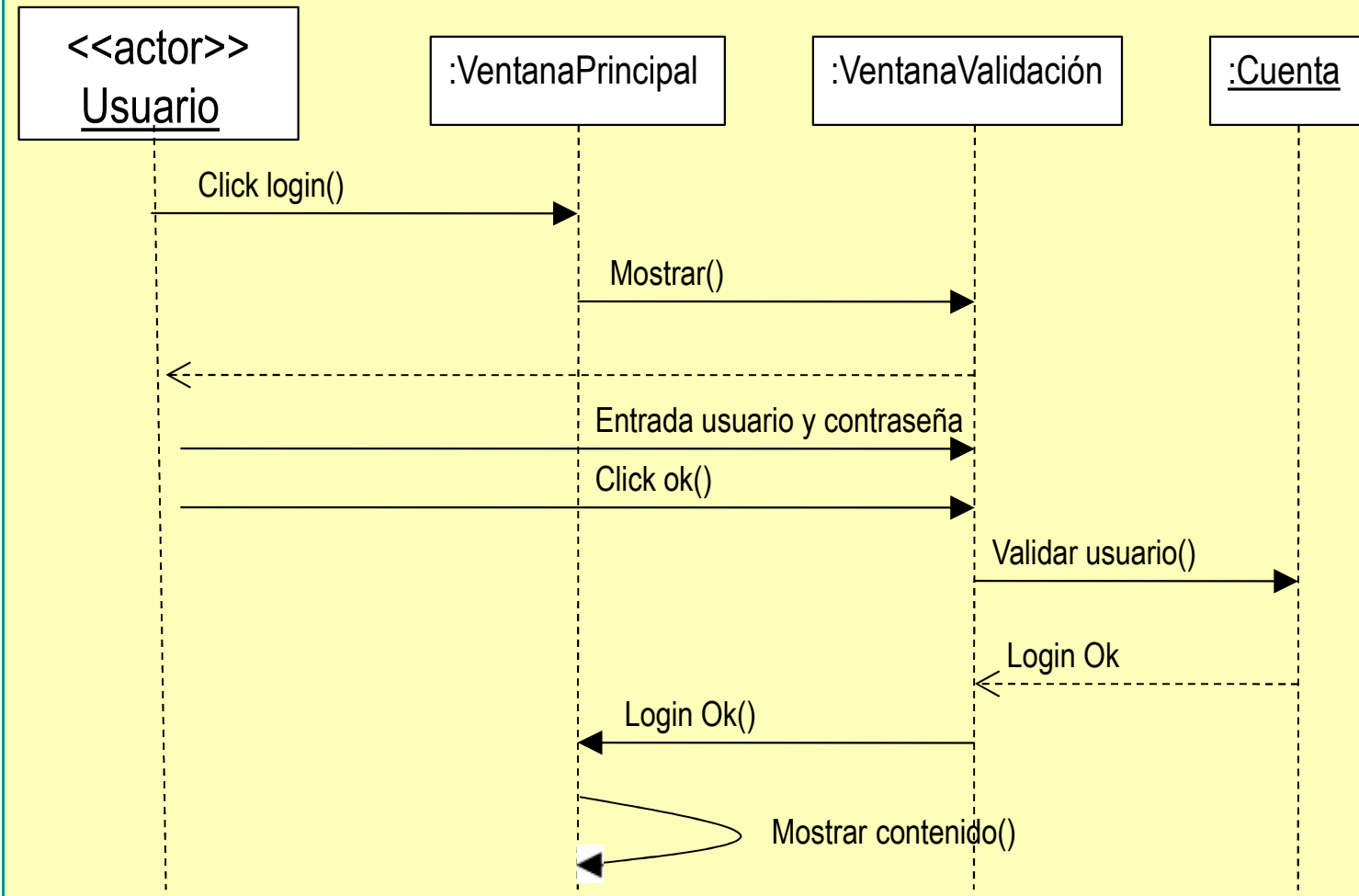
■ Ejemplo:



Diagramas de interacción. Diagramas de secuencia.

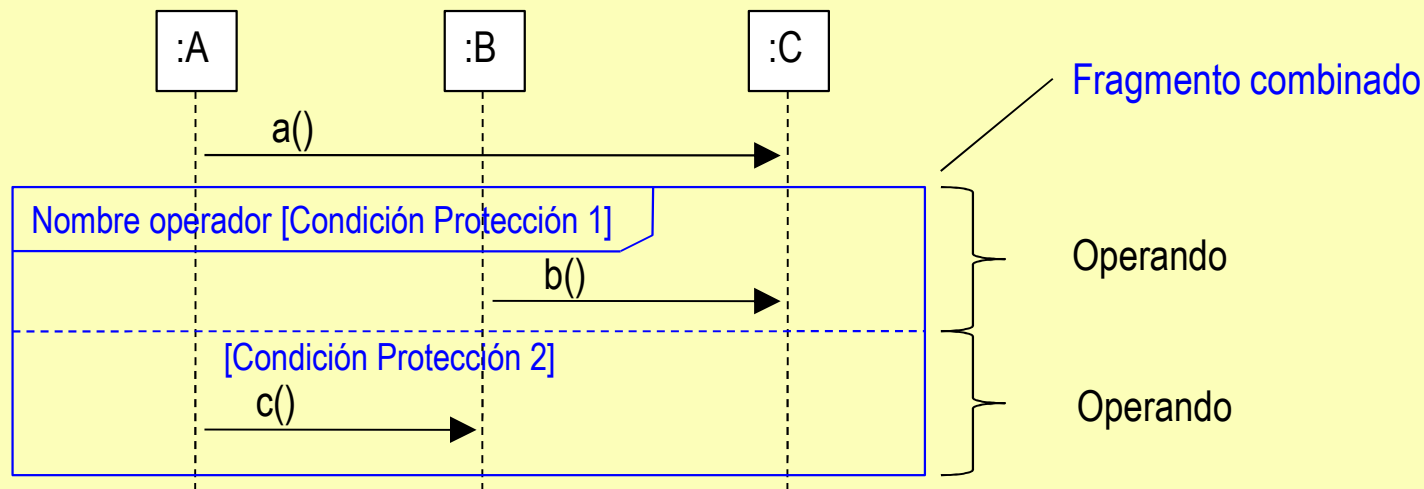
Envío de mensajes.

- **Ejercicio:** implementar el siguiente escenario de un diagrama de secuencia.



Diagramas de interacción. Diagramas de secuencia. Fragmentos combinados y operadores.

- Los diagramas de secuencia se pueden dividir en áreas denominadas fragmentos combinados.
- Todo fragmento combinado tiene:
 - Un operador.
 - Uno o más operandos.
 - Cero o más condiciones de protección.



Diagramas de interacción. Diagramas de secuencia. Fragmentos combinados y operadores.

- La lista completa de operadores es la siguiente:

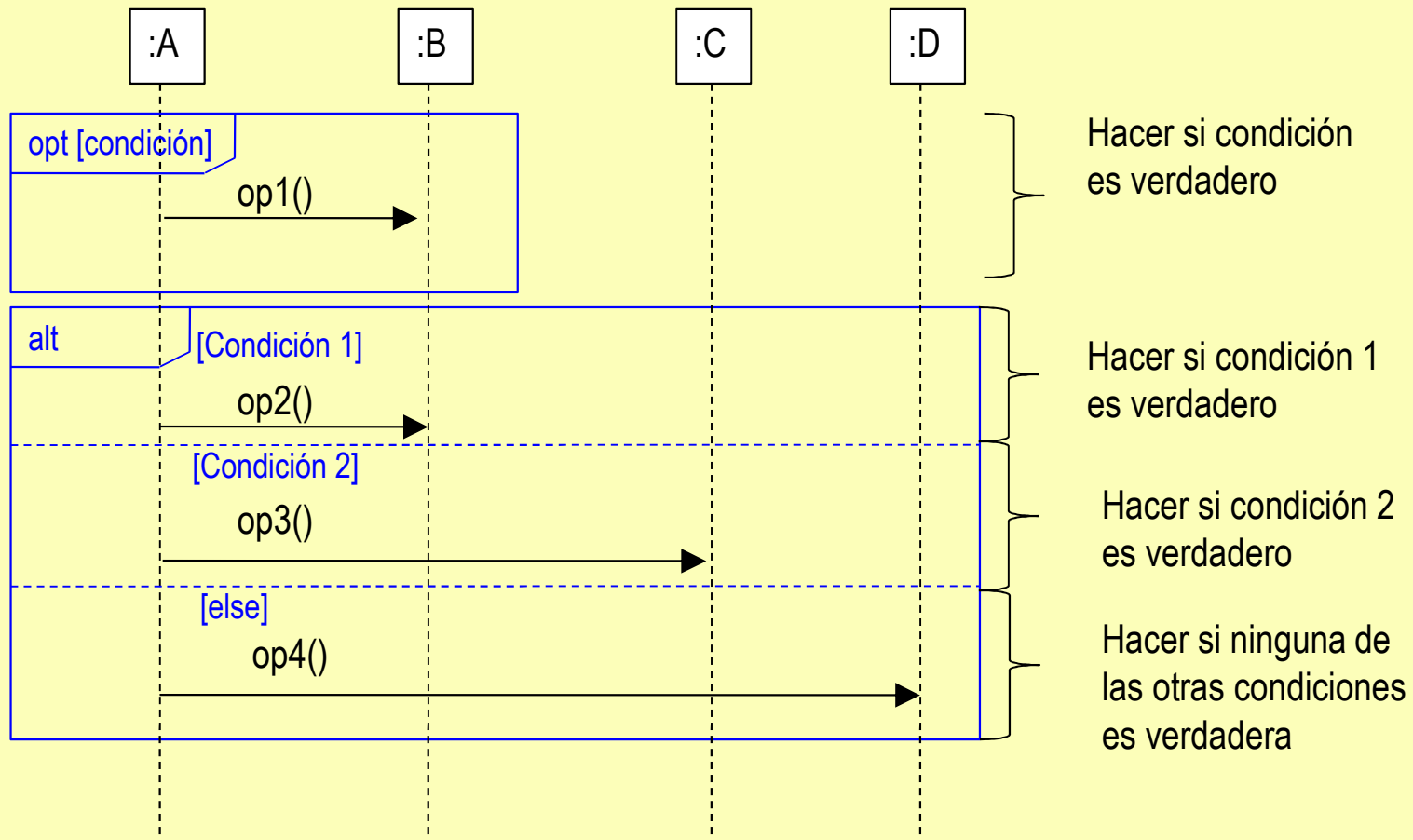
Operador	Nombre	Semántica
opt	Opción	Existe un solo operando que se ejecuta si la condición es verdadera (como if...then)
alt	Alternativas	Se ejecuta al operando cuya condición es verdadera. La palabra clave <i>e/se</i> se puede utilizar en lugar de una expresión booleana (como switch)
loop	Bucle	Esto tiene una sintaxis especial: loop min, max [condición]
break	Saltar	Si la condición de protección es verdadera, el operando se ejecuta, no el resto de la interacción que engloba.
ref	Referencia	El fragmento combinado hace referencia a otra interacción.
par	Paralelo	Todos los operandos se ejecutan en paralelo.
critical	crítico	El operando se ejecuta sin interrupción.
seq	Secuenciación débil	Todos los operandos se ejecutan en paralelo sujetos a la siguiente restricción: los eventos que lleguen en la misma línea de vida de diferentes operandos, ocurren en la misma secuencia que los operandos. Esto da lugar a una forma de secuenciación débil, de ahí el nombre

Diagramas de interacción. Diagramas de secuencia. Fragmentos combinados y operadores.

Operador	Nombre	Semántica
strict	Secuenciación estricta	Los operandos se ejecutan en secuencia estricta.
neg	Negativa	El operando muestra interacciones inválidas. Utilice esto cuando quiera mostrar interacciones que no deben suceder.
ignore	Ignorar	Lista mensajes que se omiten intencionadamente de la interacción; los nombres de los mensajes ignorados están situados entre llaves en una lista delimitada por comas después del nombre del operador, como {m1, m2, m3}. Por ejemplo, una interacción podría representar un caso de prueba en el que elige ignorar alguno de los mensajes.
consider	Considerar	Lista mensajes que se han incluido intencionadamente en la interacción. Los nombres de los mensajes están situados entre llaves, en una lista separados por comas después del nombre del operador. Por ejemplo, una interacción podría representar un caso de prueba en el que elija incluir un subconjunto del conjunto de mensajes posibles.
assert	Aserción	El operando es el único comportamiento válido en ese punto en la interacción y cualquier otro comportamiento tendría un error. Utilice esto como una forma de indicar que cierto comportamiento debe ocurrir en cierto momento en la interacción.

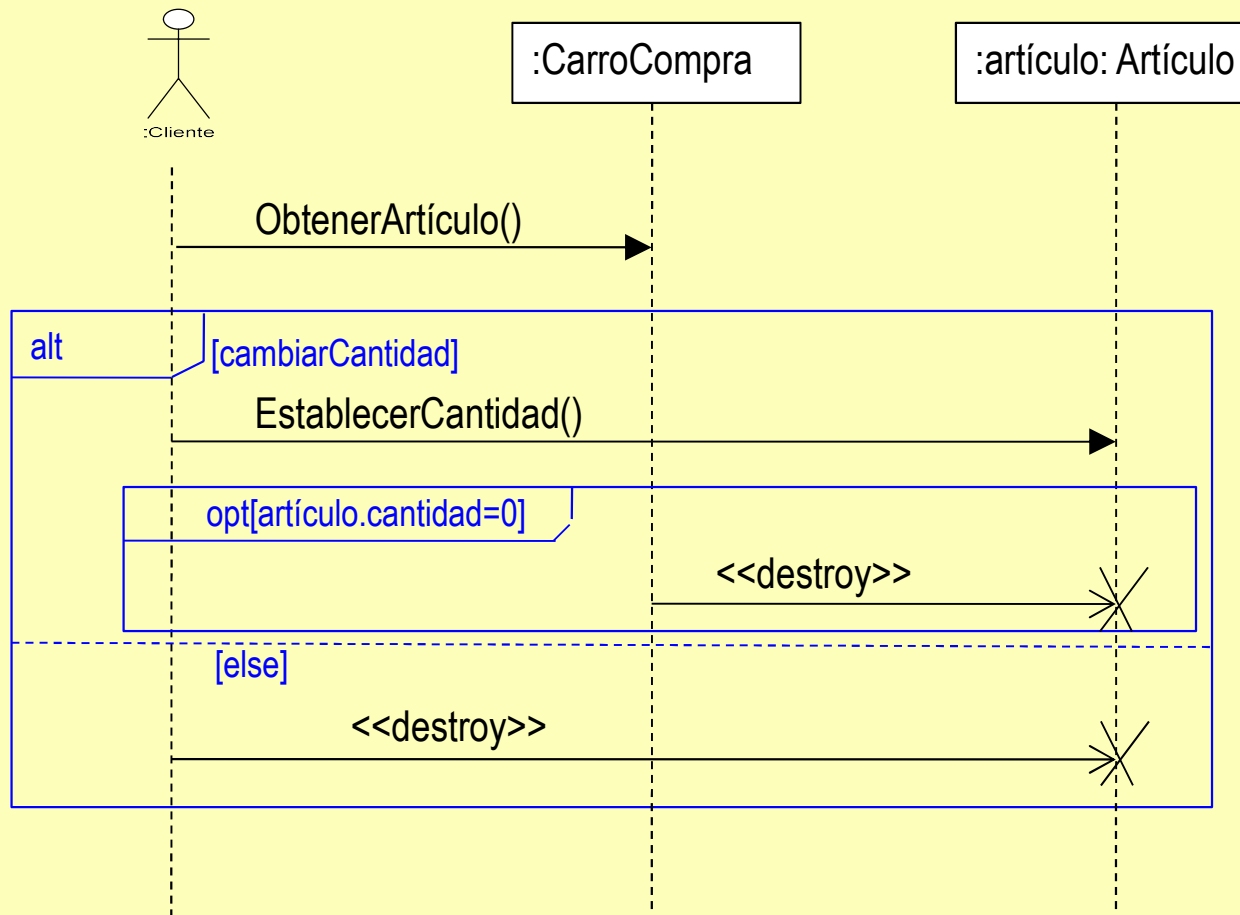
Diagramas de interacción. Diagramas de secuencia. Fragmentos combinados y operadores.

■ Ejemplo:



Diagramas de interacción. Diagramas de secuencia. Fragmentos combinados y operadores.

■ Ejemplo:

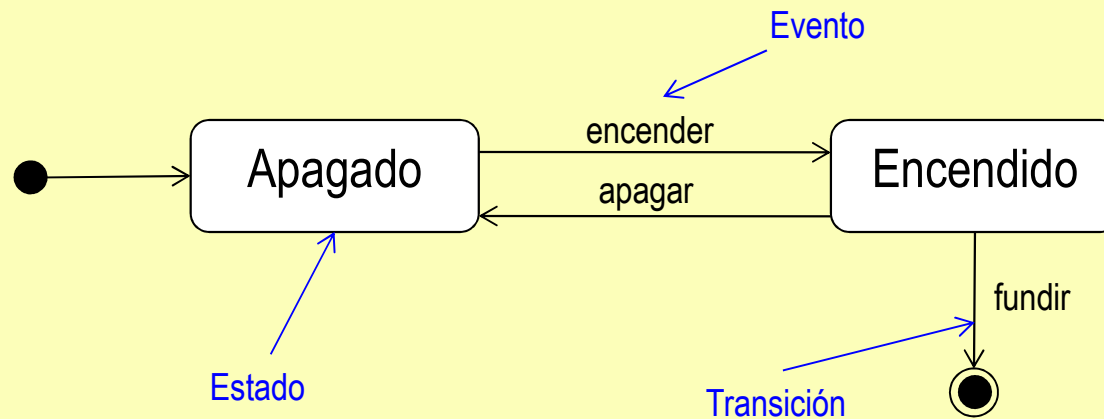


Diagramas de interacción. Diagramas de estado. Introducción.

- Los diagramas, esquemas o máquinas de estado se emplean para **modelar el comportamiento dinámico de los clasificadores**.
- Las máquinas de estado UML están basadas en el trabajo de Harel [Harel] y tienden a utilizarse para **modelar la historia del ciclo de vida de un solo objeto como una máquina de estado finita**, una máquina que puede existir en un número finito de estados.
- Los **tres elementos clave** en las máquinas de estado son:
 - **Estados**: “Una condición o situación durante la vida de un objeto durante el cual se cumple alguna condición, realiza alguna actividad o espera algún evento.” [Rumbaugh]
 - **Eventos**: “La especificación de una ocurrencia que tiene ubicación en tiempo y espacio.” [Rumbaugh]
 - **Transiciones**: El movimiento de un estado a otro en respuesta a un evento.

Diagramas de interacción. Diagramas de estado. Introducción.

- **Ejemplo.** Máquina de estado de una bombilla:



Diagramas de interacción. Diagramas de estado. Estados.

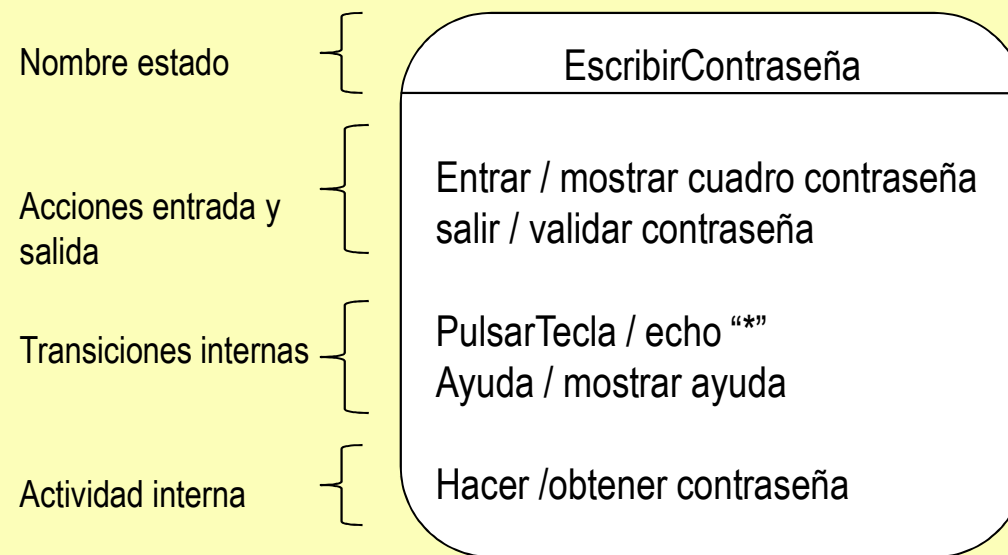
- Debe existir **diferencia semántica que marque una diferencia** para que un estado sea considerado como tal.
- Por ejemplo, si consideramos la clase Color que se muestra a continuación

Color
rojo: byte verde: byte azul: byte

- Si asumimos que *rojo*, *verde* y *azul* pueden adoptar valores entre 0 y 255, los objetos de esta clase pueden tener $256 \times 256 \times 256 = 16.777.216$ estados posibles.
- Pero ¿cuál es la diferencia semántica clave entre cada uno de esos estados? La respuesta es ninguna.
- Cada uno de los 16.777.216 posibles estados representa un color y eso es todo. La máquina de estado se puede modelar con un solo estado.

Diagramas de interacción. Diagramas de estado. Estados.

- La sintaxis de estado de UML se resume en la siguiente figura:

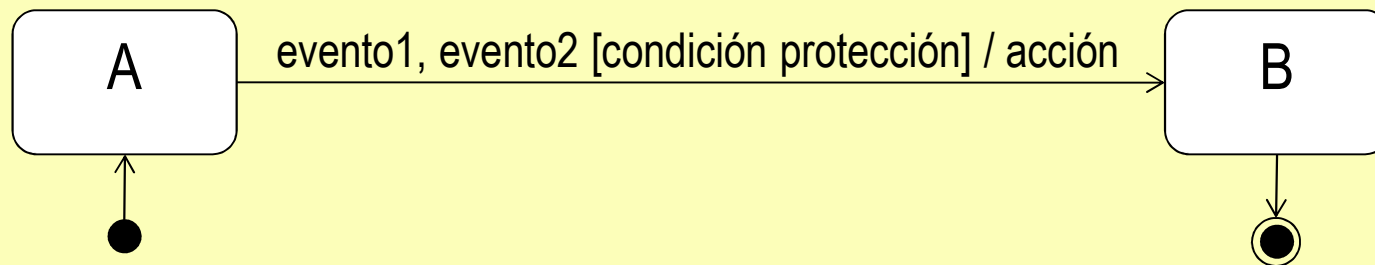


Diagramas de interacción. Diagramas de estado. Estados.

- Las acciones se consideran instantáneas y sin interrupción.
- Las actividades adoptan una cantidad de tiempo finito y se pueden interrumpir.
- Toda acción en un estado está asociada con una transición interna que está activada por un evento.
- Una transición interna le permite capturar el hecho de que algo que merece modelarse ha ocurrido pero no provoca una transición a un nuevo estado.
- Por ejemplo, en el ejemplo de la pantalla anterior, pulsar una tecla es un evento destacable pero no provoca una transición fuera del estado *EscribirContraseña*. Lo modelamos como un elemento interno, *pulsarTecla*, que causa una transición interna que activa la acción echo
“*”
.

Diagramas de interacción. Diagramas de estado. Transiciones.

- La sintaxis de transición de UML para máquinas de estado de comportamiento, se resume en la siguiente figura:



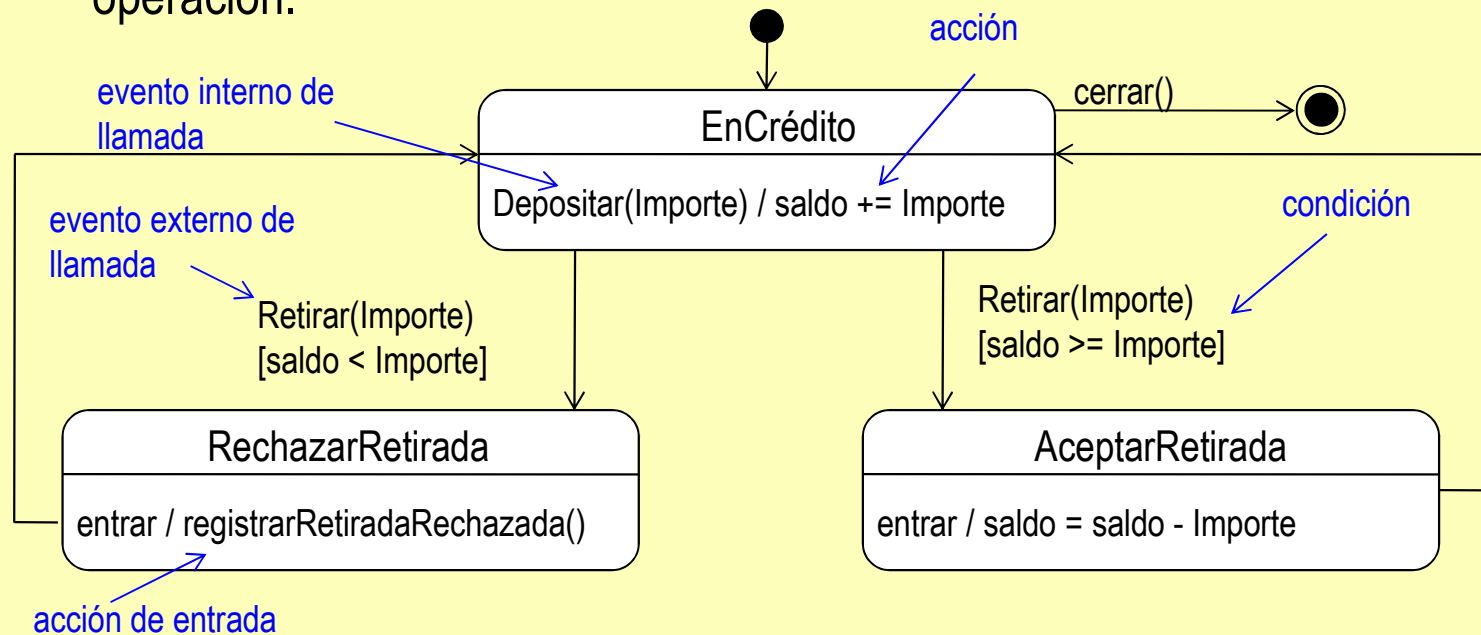
- Toda **transición** tiene **tres elementos opcionales**:
 - **Cero o más eventos**: especifican ocurrencias externas o internas que pueden activar la transición.
 - **Cero o una condición de protección**: expresión booleana que debe ser verdadera antes de que pueda ocurrir la transición. Se sitúa detrás de los eventos.
 - **Cero o más acciones**: parte de un trabajo asociado con la transición y ocurre cuando se activa la transición.

Diagramas de interacción. Diagramas de estado. Eventos.

- UML define un evento como “la especificación de una ocurrencia destacable que tiene ubicación en tiempo y espacio”.
- Los eventos activan las transiciones en máquinas de estado.
- Los eventos se pueden mostrar externamente en transiciones o internamente dentro de estados.
- Existen cuatro tipos de evento, cada uno de los cuales con semántica diferente:
 - Evento de llamada.
 - Evento de señal.
 - Evento de cambio.
 - Evento de tiempo.

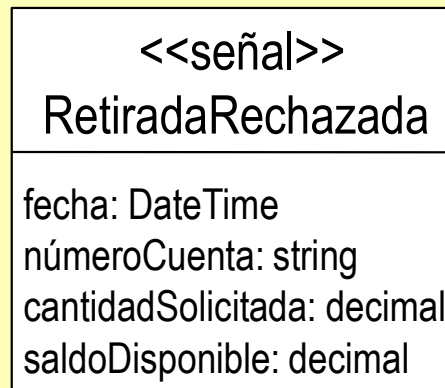
Diagramas de interacción. Diagramas de estado. Eventos. Eventos de llamada.

- Un evento de llamada es una **petición de una operación específica a invocarse en una instancia de clase de contexto**.
- Un evento de llamada debería tener la misma firma que una operación en la clase de contexto de la máquina de estado.
- Recibir un evento de llamada es un activador para que se ejecute la operación.



Diagramas de interacción. Diagramas de estado. Eventos. Eventos de señal.

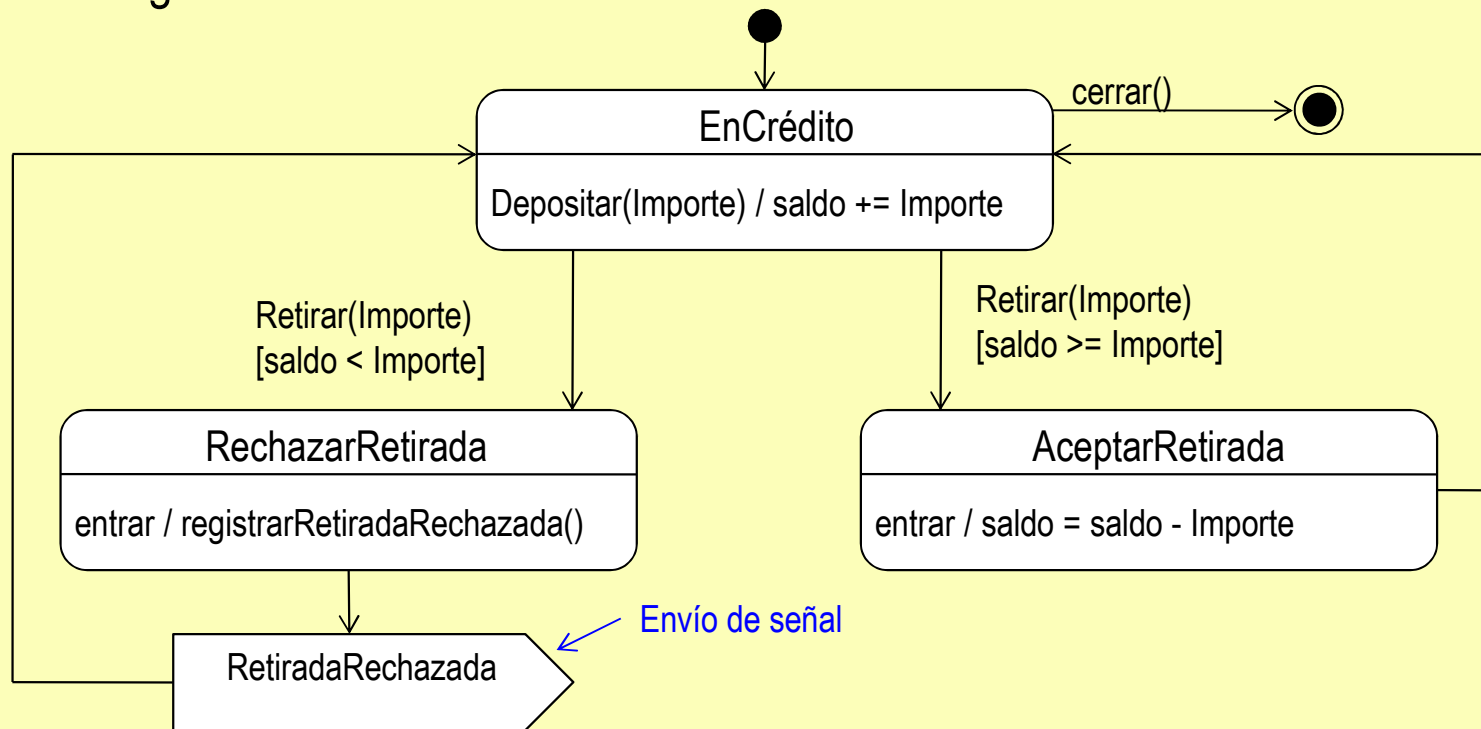
- Una señal es un paquete de información que se envía asincrónicamente entre objetos.
- Modele una señal como una clase estereotipada que alberga la información a comunicarse en sus atributos:



- Una señal no tiene normalmente ninguna operación porque su única finalidad es transportar información.

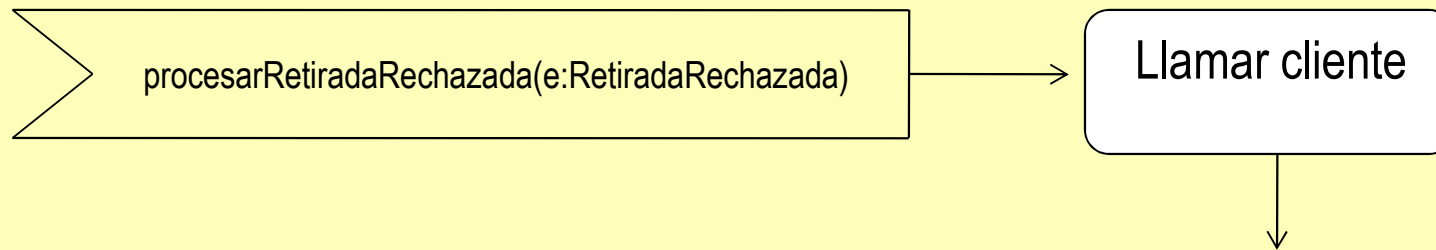
Diagramas de interacción. Diagramas de estado. Eventos. Eventos de señal.

- Se puede ver aquí la máquina de estado anterior actualizada para enviar una señal cuando se rechaza una retirada de efectivo.
- Un envío de señal se indica por medio de un pentágono convexo con el nombre de la señal dentro. Es la misma sintaxis que se utiliza en los diagramas de actividad:



Diagramas de interacción. Diagramas de estado. Eventos. Eventos de señal.

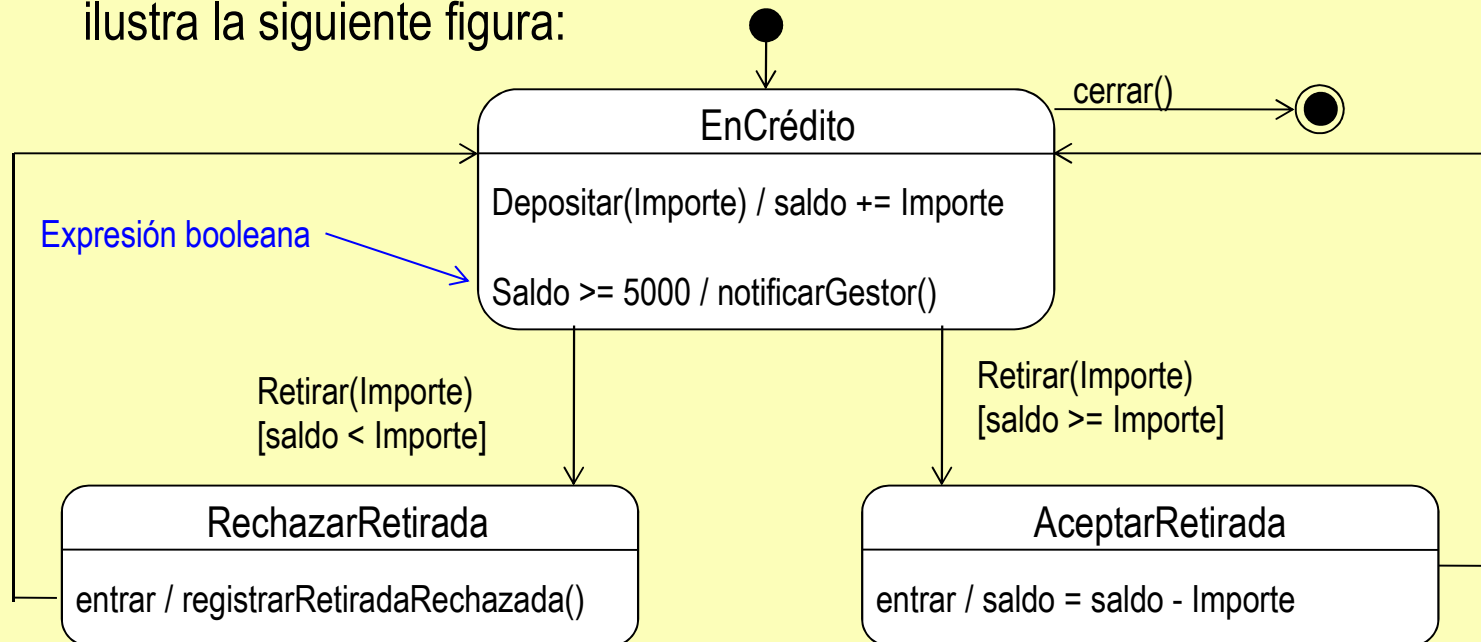
- Recibir una señal se indica por medio de un pentágono cóncavo, como se muestra en el siguiente fragmento de máquina de estado:



- También puede mostrar la recepción de la señal en transiciones internas o externas al utilizar la notación estándar *nombreEvento / acción* que se ha tratado anteriormente.

Diagramas de interacción. Diagramas de estado. Eventos. Eventos de cambio.

- Un evento de cambio se especifica como una expresión booleana según ilustra la siguiente figura:

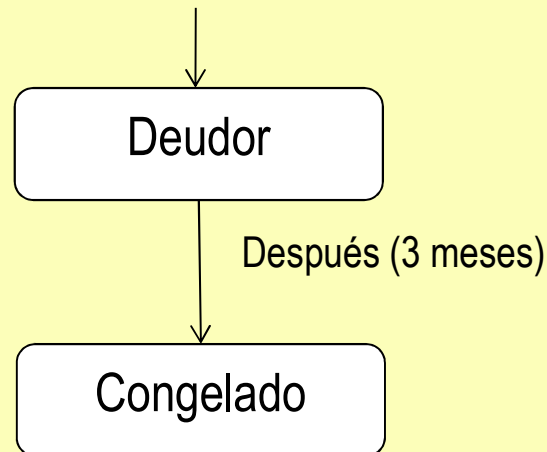


- La acción asociada con el evento se realiza cuando el valor de la expresión booleana es verdadero.
- Todos los valores de la expresión booleana deben ser elementos de la clase.

Diagramas de interacción. Diagramas de estado.

Eventos. Eventos de tiempo.

- Los eventos de tiempo se indican normalmente por medio de las palabras clave *cuando* y *después*.
- La palabra clave *cuando* especifica un momento particular en el que se activa el evento.
- *Después* especifica un momento después del que se activa el evento.
- Cualquier símbolo en la expresión debe ser un elemento de la clase contexto.



Bibliografía.

- [Arlow] **UML 2**. Jim Arlow, Ila Neustadt. Anaya Multimedia, 2005. ISBN 84-415-2033-X.
- [Harel] ***Modeling Reactive Systems with Statecharts: The Statemate Approach***. David Harel, Michal Politi. McGraw Hill, 1998. ISBN 0070262055.
- [Kimmel] **Manual de UML**. Paul Kimmel. McGraw Hill, 2007. ISBN 970-10-5899-2.
- [Kendall] **Fast Track UML 2**. Kendall Scott. Apress, 2004. ISBN 1-59059-320-0.
- [Rumbaugh] **El Lenguaje Unificado de Modelado**. James Rumbaugh, Ivar Jacobson, Grady Booch. Addison-Wesley Iberoamericana, 1999. ISBN 84-7829-028-1.
- <http://uml.sourceforge.net/index.php>: Web de descarga del programa libre "Umbrello". Para hacer diseños UML.
- <http://office.microsoft.com/es-es/visio/default.aspx?ofcresset=1>: Página del programa Visio de Microsoft.