

*Recursos Informáticos*

# UML 2

**Iniciación, ejemplos  
y ejercicios corregidos**

**3ª edición**

**Laurent DEBRAUWER  
Fien VAN DER HEYDE**

 **INFORMÁTICA TÉCNICA**

  
**eni**  
ediciones

# UML 2

## Iniciación, ejemplos y ejercicios corregidos [3ª edición]

Este libro sobre UML 2 está dirigido tanto a **estudiantes** como a **desarrolladores** que se ocupan del **modelado de sistemas, de programas y de procesos**.

Etapa a etapa, el lector descubrirá los elementos de modelado a partir de ejemplos pedagógicos extraídos del mundo de los caballos. Tras una **introducción a la orientación a objetos**, la obra presenta los diferentes diagramas de UML 2, desde la **descripción de los requisitos** a partir de casos de uso, hasta el diagrama de **componentes** pasando por los **diagramas de interacción, de clases, de estructura compuesta, de estados transiciones y de actividades**. El lector aprenderá de qué manera los **diagramas de interacción** pueden utilizarse para descubrir los objetos que componen el sistema.

Esta nueva edición de la obra introduce los **diagramas de perfil**.

### Los capítulos del libro:

Introducción – A propósito de UML – Conceptos de la orientación a objetos – Modelado de los requisitos – Modelado de la dinámica – Modelado de objetos – Estructuración de los elementos de modelado – Modelado del ciclo de vida de los objetos – Modelado de las actividades – Modelado de la arquitectura del sistema – Los perfiles – Anexo 1: Arquitectura MDA: la herramienta DB-MAIN – Anexo 2: Corrección de los ejercicios – Anexo 3: Glosario – Anexo 4: Léxico – Anexo 5: Notación gráfica – Anexo 6: Bibliografía

---

### Fien VAV DER HEYDE - Laurent DEBRAUWER

**Fien Van der Heyde**, con formación superior en finanzas e informática, es responsable de informática en un gran banco en Luxemburgo. El modelado de procesos ocupa un lugar importante dentro de su actividad profesional, pero también siente un enorme interés por... los caballos.

**Laurent Debrauwer** es doctor en informática por la Universidad de Lille 1. Es autor de programas en el campo de la lingüística y de la semántica editados por las empresas META-AGENT Software y Semántica de las que es director. Especialista en orientación a objetos, es profesor de ingeniería de software y de Design Patterns en la Universidad de Luxemburgo.

## Motivaciones de la obra

UML (*Unified Modeling Language* o lenguaje unificado de modelación) es un lenguaje gráfico destinado al modelado de sistemas y procesos. Está basado en la orientación a objetos que condujo, en primer lugar, a la creación de lenguajes de programación como Java, C++ o Smalltalk.

UML está unificado, ya que deriva de varias notaciones precedentes. En la actualidad UML es promovido por el OMG (*Object Management Group*), un consorcio de más de 800 sociedades y universidades activas en el campo de las tecnologías orientadas a objetos.

Nuestra opinión es que UML se convertirá en un lenguaje de modelación muy extendido, sobre todo gracias a su riqueza semántica, que lo abstrae de numerosos aspectos técnicos. El objetivo principal de la presente obra es divulgar el lenguaje UML. El libro, por tanto, va dirigido a un amplio abanico de público: informáticos, jefes de proyecto, directores, o cualquier otra persona que desee disponer de una visión de conjunto de un sistema a partir de varios esquemas.

Para lograr dicho objetivo hemos adoptado la estrategia siguiente:

- Explicar todos los conceptos de la manera más simple y completa posible.
- Introducir un gran número de ejemplos con el fin de conferir la mayor claridad posible a las explicaciones.
- No dar explicaciones basadas en código C++, Java o derivado de otros lenguajes de programación.
- Evitar los ejemplos clásicos y decantarse por un mundo rico y escasas veces abordado: el de los caballos.
- Proponer ejercicios cuyas soluciones podrán encontrarse al final de la obra.

UML es un lenguaje semánticamente rico y, por tanto, resulta bastante difícil retener todos los conceptos en los que se basa. Esperamos que la presente guía constituya una herramienta útil para comprenderlos y memorizarlos, y pueda servir de referencia a la hora de modelar con UML.

El título del libro es *UML 2: Iniciación, ejemplos y ejercicios corregidos*. El segundo capítulo, A propósito de UML, ofrece al lector un paseo por la historia del UML. Descubriremos entonces que la versión actual es precisamente la 2.

## El mundo de los caballos

Todos los ejemplos incluidos en la obra pertenecen al mundo equino. No ocultaremos por más tiempo que Fien, uno de los autores, es una gran apasionada del mundo de los caballos. Fien es propietaria de una yegua de cabeza acarnerada que responde al nombre de Jorgelina, y cuya fotografía puede observarse en la figura 1.1.

Mientras lee la obra imagine que se encuentra al frente de una granja de cría de caballos. Podríamos situar el escenario en Kentucky. En ese caso su rancho estaría compuesto por una manada de quarter horse. No obstante, si se decanta por un ambiente más distinguido, entonces su papel sería el de director de un picadero en Andalucía formado por purasangres ingleses destinados a la cría de yearlings. Sea cual fuere la elección, deberá llevar a cabo un seguimiento riguroso y gestionar una gran cantidad de datos con vínculos específicos entre sí y ciclos propios.

No cabe duda de que se le plantearán a diario preguntas como estas:

- La yegua Jorgelina: ¿Es hoy cuando tiene que parir?
- El caballo Travieso: ¿Qué cantidad de avena debo darle?
- El caballo Quincy: ¿Cuándo debo vacunarle?



*Figura 1.1 - Jorgelina de Gisors*

# Contenido de la obra

La obra se organiza en once capítulos cuyo contenido describimos brevemente a continuación.

## Capítulo 1

La presente introducción.

## Capítulo 2 - A propósito de UML

Capítulo dedicado, por un lado, al origen del UML y, por otro, al Proceso Unificado y a la MDA (*Model Driven Architecture* o arquitectura guiada por modelos).

El Proceso Unificado es un proceso de desarrollo y evolución de software. La arquitectura MDA se destina a la realización de programas, independientemente de la plataforma y del lenguaje de programación.

## Capítulo 3 - Conceptos de la orientación a objetos

El capítulo 3 describe los diferentes conceptos y principios de la orientación a objetos en que se basa el lenguaje UML. Conocerlos resulta indispensable para comprender los elementos utilizados en los diagramas UML.

## Capítulo 4 - Modelado de los requisitos

El objetivo del capítulo 4 es mostrar los casos de uso empleados para describir los requisitos funcionales perseguidos a la hora de redactar el pliego de condiciones de un sistema o las funcionalidades de un sistema existente.

## Capítulo 5 - Modelado de la dinámica

El capítulo 5 explica la manera en que UML representa las interacciones entre objetos. La descripción de las interacciones se utiliza también para ver qué objetos componen un sistema. La vista está basada en las interacciones que intervienen en los casos de uso del sistema.

## Capítulo 6 - Modelado de objetos

El capítulo 6 es primordial. Está dedicado al modelado estático de objetos, es decir, sin descripción de las interacciones o del ciclo de vida de los objetos. Los métodos se introducen desde un punto de vista estático, sin describir su encadenamiento.

El capítulo incluye un estudio del diagrama de clases. Dicho diagrama contiene los atributos, métodos y asociaciones de los objetos. Es básico a la hora de modelar un sistema mediante objetos. De todos los diagramas UML, es el único obligatorio en tales modelados.

## Capítulo 7 - Estructuración de los elementos de modelado

El capítulo 7 está dedicado a los empaquetados. UML 2 describe los empaquetados con ayuda de un diagrama específico. Un empaquetado es un agrupamiento de elementos de modelado: clases, componentes, otros empaquetados, etc.

## Capítulo 8 - Modelado del ciclo de vida de los objetos

El capítulo 8 estudia el ciclo de vida de los objetos. El ciclo de vida de un objeto está constituido por las diferentes etapas o estados por los que éste pasa para participar, dentro del sistema, en la realización de un objetivo. Un estado corresponde a un momento de actividad o de inactividad (espera) del objeto.

## Capítulo 9 - Modelado de las actividades

El capítulo 9 está dedicado al diagrama de actividades. Éste se apoya en el diagrama de estados-transiciones estudiado en el capítulo 8, Modelado del ciclo de vida de los objetos. Se trata de una forma específica del diagrama de estados-transiciones en el cual todos los estados se asocian a una actividad y todas las transiciones son automáticas. En este diagrama las transiciones reciben el nombre de encadenamientos.

## Capítulo 10 - Modelado de la arquitectura del sistema

El capítulo 10 está dedicado al modelado de la arquitectura del sistema. Dicho modelado presenta dos

aspectos:

- El modelado de la arquitectura de programas y su estructuración en componentes.
- El modelado de la arquitectura material y la repartición física de los programas.

## **Capítulo 11 - Los perfiles**

El capítulo 11 introduce la noción de perfil, destinado a enriquecer las capacidades de modelado de UML. Un perfil define clases, tipos de datos, tipos primitivos y estereotipos. Estos últimos se utilizan para extender las metaclases del metamodelo de UML.

## **Anexo 1 - Arquitectura MDA: la herramienta DB-MAIN**

El anexo 1 presenta la herramienta DB-MAIN en el marco de la arquitectura MDA aplicada a la realización de esquemas de bases de datos relacionales.

## **Anexo 2 - Corrección de los ejercicios**

El anexo 2 desglosa una posible corrección de los ejercicios incorporados al final de algunos capítulos.

## **Anexo 3 - Glosario**

El anexo 3 es un glosario de los diferentes términos empleados en la obra.

## **Anexo 4 - Léxico**

El anexo 4 presenta un léxico español-inglés e inglés-español con los diferentes términos empleados en la obra.

## **Anexo 5 - Notación gráfica**

El anexo 5 es un resumen de la notación gráfica de los principales elementos de UML.

## **Anexo 6 - Bibliografía**

El anexo 6 es una bibliografía de las principales obras de referencia de la notación UML.

# Introducción

El presente capítulo está dedicado, por un lado, al origen del UML y, por otro, a dos elementos vinculados a UML:

- El Proceso Unificado, un proceso de desarrollo y evolución de programas.
- La arquitectura MDA (*Model Driven Architecture* o arquitectura guiada por modelos), destinada a la realización de sistemas, independientemente de la plataforma física y de los aspectos tecnológicos.



## El origen del UML: Unified Modeling Language

El UML está basado en la orientación a objetos, sistema que vio la luz mucho antes que el UML en el campo de los lenguajes de programación. Simula, el primer lenguaje orientado a objetos, nació en los años 1960 y conoció numerosos sucesores: Smalltalk, C++, Java o, más recientemente, C#.

En un lenguaje de programación la descripción de los objetos se realiza de manera formal, con una sintaxis rigurosa. Dicha sintaxis resulta ilegible para los no programadores y difícil de descifrar para los programadores. A diferencia de las máquinas, los humanos prefieren utilizar lenguajes gráficos para representar abstracciones, ya que dominan este tipo de lenguaje con mayor facilidad y obtienen una visión de conjunto de los sistemas en mucho menos tiempo.

En los años 80 y principios de los 90, las notaciones gráficas se multiplican y, muy a menudo, cada uno utiliza su propia notación. En 1994, James Rumbaugh y Grady Booch deciden unirse para unificar sus notaciones, procedentes de sus respectivos métodos: OMT para James Rumbaugh y método Booch para Grady Booch. En 1995, Yvar Jacobson decide unirse al equipo de los "tres amigos". El equipo trabaja entonces dentro de Rational Software.

La versión 1.0 de UML se publica en 1997. El trabajo de evolución de la notación empieza a hacerse demasiado voluminoso para sólo tres personas y los tres amigos solicitan la ayuda del Object Management Group (OMG), un consorcio de más de 800 sociedades y universidades que trabajan en el campo de las tecnologías del objeto. La OMG adopta la notación UML en noviembre de 1997 en su versión 1.1 y crea una Task Force encargada de la evolución del UML.

Desde entonces, la Task Force ha actualizado UML en varias ocasiones. En marzo de 2003 ve la luz la versión 1.5, que ofrece la posibilidad de describir acciones gracias a una extensión de UML llamada *Action Semantics* o semántica de acciones.

La versión 2.0 fue publicada en julio de 2005. Constituye la primera evolución importante desde la aparición del UML en 1997. A ella se han ido añadiendo numerosos diagramas y los ya existentes se han enriquecido con nuevas construcciones. Desde julio de 2005 esta versión 2.0 ha sido mejorada. En febrero 2009, la versión 2.2 se publicó incluyendo la taxonomía oficial de los perfiles UML. La última versión es la 2.4, publicada en agosto de 2011.

La figura 2.1 ilustra la evolución de UML y traza su origen y sus principales versiones con ayuda de un diagrama de actividad.

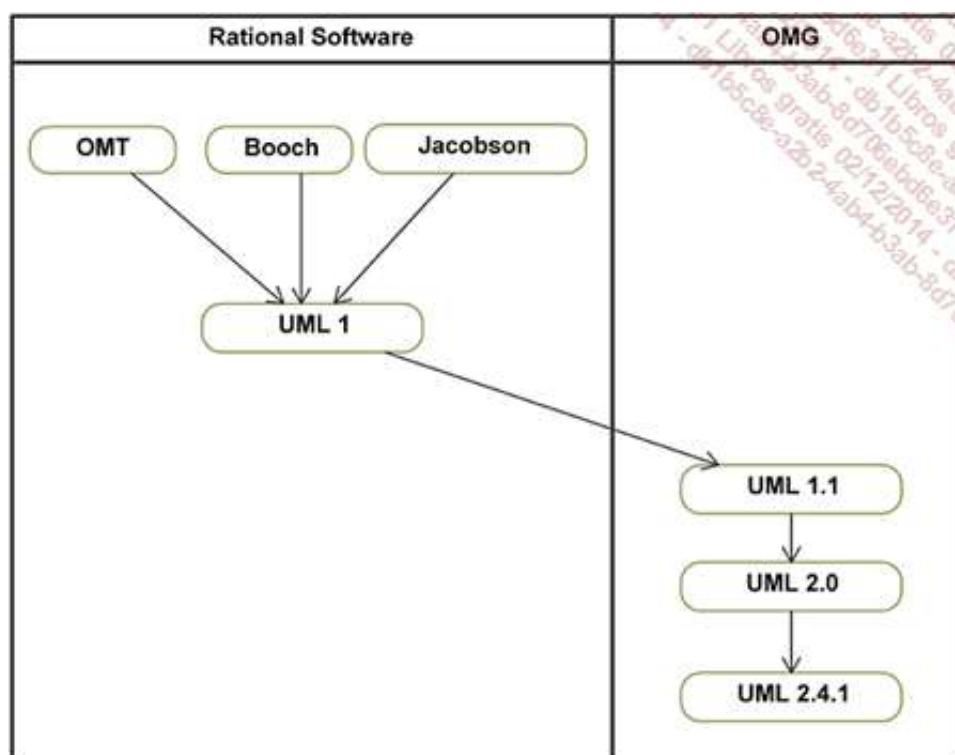


Figura 2.1 - Origen y principales versiones de UML



Recordemos que UML es una notación destinada al modelado de sistemas y de procesos mediante objetos. UML no contiene una guía metodológica, pero constituye un soporte de modelado.

# El Proceso Unificado

El Proceso Unificado es un proceso de realización o de evolución de software enteramente basado en UML, de ahí el interés de presentarlo en esta obra. Está constituido por un conjunto de directivas que permiten producir software a partir del pliego de condiciones (requisitos). Cada directiva define *quién* hace qué y *en qué momento*. Un proceso permite, por tanto, estructurar las diferentes etapas de un proyecto informático.

Los tres autores del Proceso Unificado son los mismos que los del UML. No obstante, el uso de UML no exige la utilización del Proceso Unificado. Un proyecto que emplee UML puede utilizar otro proceso diferente del Proceso Unificado o no emplear ninguno.

El Proceso Unificado es conducido por los casos de uso. Éstos se utilizan para describir los requisitos del proyecto y se describen con ayuda de una representación específica del Proceso Unificado más rica que la contenida en UML.

El Proceso Unificado es incremental. Los proyectos se dividen en una serie de subproyectos. Cada subproyecto es un ladrillo que se añade al subproyecto precedente que, por tanto, debe haberse realizado con antelación. Cuando se ha llevado a cabo el último subproyecto se concluye la totalidad del proyecto.

El Proceso Unificado es iterativo. Todos los subproyectos se efectúan con las mismas actividades. Al concluir cada subproyecto se evalúa una entrega parcial.

Los creadores del Proceso Unificado proponen el desarrollo incremental e iterativo para evitar tener que tratar en su totalidad los proyectos importantes con entregas muy posteriores a la redacción del pliego de condiciones. En efecto, en casos semejantes, es probable que las necesidades del cliente hayan evolucionado desde entonces y que no recuerde con exactitud aquello que había solicitado en el pliego de condiciones. De ser así, podrían llegar a producirse conflictos fácilmente evitables con un desarrollo incremental e iterativo.

El ciclo de desarrollo se divide en cuatro fases:

1. La fase de inicio (*inception*) consiste en evaluar el proyecto. Se decide llevar adelante o no el proyecto en función de los imperativos económicos, se determinan los principales casos de uso y se hace un primer esbozo de arquitectura. También se elaboran una o dos maquetas.
2. El objetivo de la fase de elaboración es construir la arquitectura del sistema. Una vez concluida la elaboración, se conocen definitivamente las exigencias del proyecto y su arquitectura.
3. La fase de construcción corresponde al desarrollo de software de la arquitectura, determinado durante la fase de elaboración.
4. La fase de transición comprende la instalación del software en los equipos del cliente y la formación de los usuarios.

En el Proceso Unificado, cada fase está detallada por un conjunto de actividades. Una actividad es un conjunto de acciones descrito por un diagrama de actividades. Con el Proceso Unificado se suministra también un diccionario muy completo constituido por modelos de actividades y casos de uso adaptados a sectores de actividad específicos.

Las principales actividades del Proceso Unificado son las siguientes:

- Modelado de los procesos de negocio.
- Gestión de los requisitos.
- Análisis y diseño.
- Implantación y test.
- Despliegue.

En la fase de *inception* las actividades utilizadas con mayor frecuencia son el modelado de procesos de negocio y la gestión de requisitos.

Durante la elaboración, las actividades empleadas con mayor frecuencia son la gestión de requisitos y el análisis y diseño.

La fase de construcción comprende principalmente el análisis y el diseño, así como la implantación y el test.

La fase de transición recurre sobre todo a la actividad de despliegue.

A modo de conclusión, diremos que el Proceso Unificado es un método iterativo de desarrollo. Esto lo distingue de los desarrollos clásicos como el ciclo en cascada, que van secuencialmente de la escritura de las necesidades a la entrega.

## MDA: Model Driven Architecture

MDA es una nueva propuesta de la OMG cuyo objetivo es diseñar sistemas basándose únicamente en el modelado del dominio, independientemente de los aspectos tecnológicos. A partir de este modelado, MDA propone obtener por transformación elementos técnicos capaces de funcionar dentro de una plataforma de software como Java o .NET.

En MDA, el modelo de objetos del dominio se llama PIM, es decir, *Platform Independent Model* o modelo independiente de la plataforma. El PIM está constituido por un conjunto de elementos cuyo diseño debe hacerse de forma independiente a cualquier lenguaje de programación o tecnología. Posteriormente, el modelo se transforma, manual o automáticamente, en un modelo específico de una plataforma y de un lenguaje de programación. Dicho modelo específico recibe el nombre de PSM, es decir, *Platform Specific Model* o modelo específico de plataforma.

La relación con UML se establece a nivel del PIM. UML es un excelente candidato a lenguaje a ese nivel. Posee la ventaja de describir con precisión los objetos, manteniéndose al mismo tiempo independiente de las tecnologías. En lo que respecta a los tratamientos, la extensión Action Semantics de UML debería permitirle responder a todas las necesidades de descripción.

# Introducción

El objetivo del presente capítulo es describir los diferentes conceptos y principios de la orientación a *objetos* en los que se basa el lenguaje UML. Conocerlos resulta indispensable para comprender los elementos utilizados en los diagramas UML que abordaremos en los capítulos siguientes.

En primer lugar, trataremos el concepto de objeto y después veremos cómo modelarlo en UML por abstracción.

Introduciremos la noción de clases, representación común de un conjunto de objetos similares.

Hablaremos después del principio de encapsulación, ocultación de informaciones internas y propias del funcionamiento del objeto.

Describiremos las relaciones de especialización y de generalización que introducen las jerarquías de clases, la herencia, las clases concretas y abstractas y, posteriormente, abordaremos el polimorfismo, consecuencia directa de la especialización.

Finalmente trataremos la composición de objetos para concluir con una noción más específica de UML, la especialización de los elementos del diagrama a través de los estereotipos.

# El objeto

Un objeto es una entidad identificable del mundo real. Puede tener una existencia física (un caballo, un libro) o no tenerla (un texto de ley). *Identificable* significa que el objeto se puede designar.

## Ejemplo

Mi yegua Jorgelina  
Mi libro sobre UML  
El artículo 293B del código de impuestos

En UML todo objeto posee un conjunto de atributos (estructura) y un conjunto de métodos (comportamiento). Un atributo es una variable destinada a recibir un valor. Un método es un conjunto de instrucciones que toman unos valores de entrada y modifican los valores de los atributos o producen un resultado.

Incluso los objetos estáticos del mundo real son percibidos siempre como dinámicos. Así, en UML, un libro se percibe como un objeto capaz de abrirse él mismo en una página determinada.

Todo sistema concebido en UML está compuesto por objetos que interactúan entre sí y realizan operaciones propias de su comportamiento.

## Ejemplo

*Una manada de caballos es un sistema de objetos que interactúa entre sí, cada objeto posee su propio comportamiento.*

De esta forma, el comportamiento global de un sistema se reparte entre los diferentes objetos. En nuestro ejemplo bastará con hacer un paralelismo con el mundo real para comprenderlo.

## La abstracción

La abstracción es un principio muy importante en modelado. Consiste en tener en cuenta únicamente las propiedades pertinentes de un objeto para un problema concreto. Los objetos utilizados en UML son abstracciones del mundo real.

### Ejemplo

*Si nos interesamos por los caballos en su actividad de carrera, las propiedades aptitud, velocidad, edad, equilibrio mental y casta de origen son pertinentes para dicha actividad y se tienen en cuenta.*

*Si nos interesamos por los caballos en su actividad de bestia de tiro, las propiedades edad, tamaño, fuerza y corpulencia son pertinentes para dicha actividad y se tienen en cuenta.*



La abstracción es una simplificación indispensable para el proceso de modelado. Un objeto UML es una abstracción del objeto del mundo real de acuerdo con las necesidades del sistema de la que sólo se tienen en cuenta los elementos esenciales.



## Clases de objetos

Un conjunto de objetos similares, es decir, con la misma estructura y comportamiento, y constituidos por los mismos atributos y métodos, forma una clase de objetos. La estructura y el comportamiento pueden entonces definirse en común en el ámbito de la clase.

Todos los objetos de una clase, llamada también instancia de clase, se distinguen por tener una identidad propia y sus atributos les confieren valores específicos.

### Ejemplo

El conjunto de caballos constituye la clase *Caballo*, que posee la estructura y el comportamiento descritos en la figura 3.1.

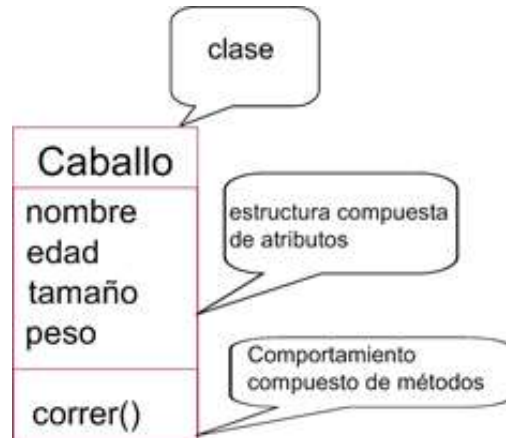


Figura 3.1 - La clase *Caballo*

El caballo *Jorgelina* es una instancia de la clase *Caballo* cuyos atributos y valores se ilustran en la figura 3.2.

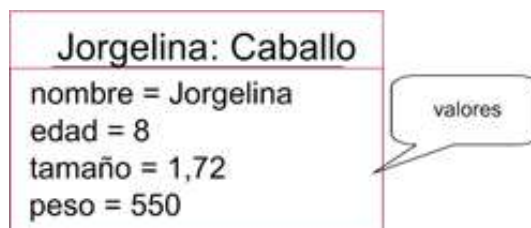


Figura 3.2 - El caballo *Jorgelina*



El nombre de las clases se escribe en singular y está formado siempre por un nombre precedido o seguido por uno o varios adjetivos que lo califican. Dicho nombre es revelador de los objetos que forman la clase.

## Encapsulación

La encapsulación consiste en ocultar los atributos y métodos del objeto a otros objetos. En efecto, algunos atributos y métodos tienen como único objetivo tratamientos internos del objeto y no deben estar expuestos a los objetos exteriores. Una vez encapsulados, pasan a denominarse atributos y métodos privados del objeto.

La encapsulación es una abstracción, ya que se simplifica la representación del objeto con relación a los objetos externos. Esta representación simplificada está formada por atributos y métodos públicos del objeto.

La definición de encapsulación se realiza en el ámbito de la clase. Los objetos externos a un objeto son, por tanto, las instancias de las demás clases.

### Ejemplo

Al correr, los caballos efectúan diferentes movimientos como pueden ser levantar las patas, levantar la cabeza o levantar la cola. Esos movimientos son internos al funcionamiento del animal y no tienen por qué ser conocidos en el exterior. Son métodos privados. Las operaciones acceden a una parte interna del caballo: sus músculos, su cerebro y su vista. La parte interna se representa en forma de atributos privados. El conjunto de atributos y métodos se ilustra en la figura 3.3.

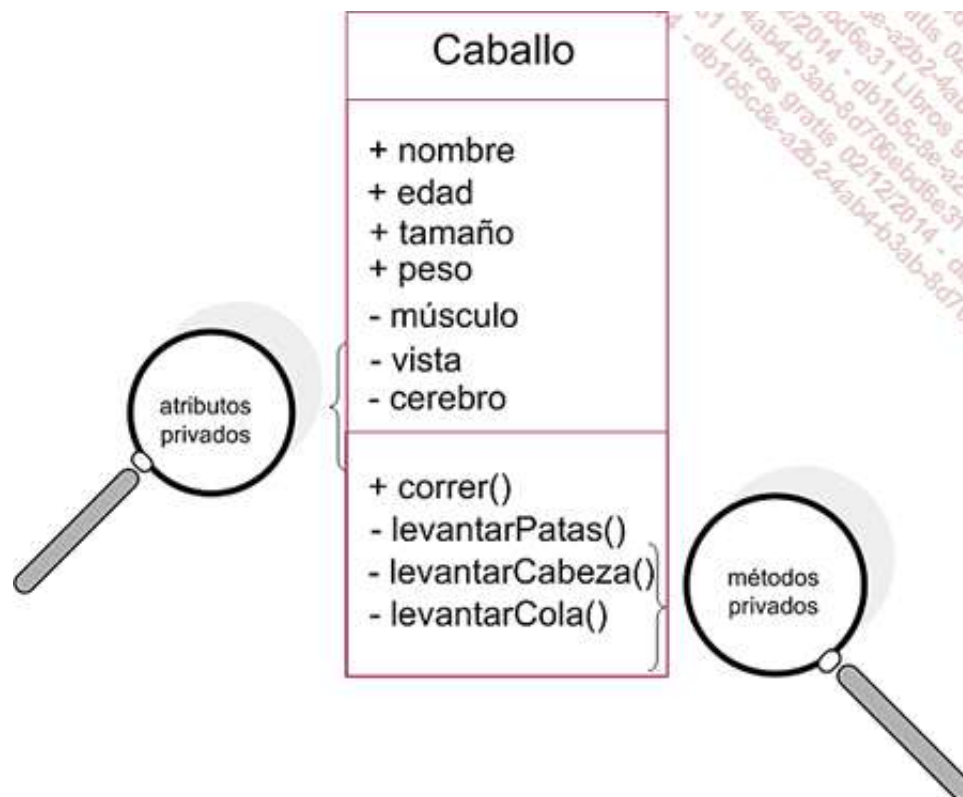


Figura 3.3 - La clase Caballo detallada



En la notación UML, los atributos y métodos públicos aparecen precedidos del signo más mientras que los privados (encapsulados) aparecen precedidos del signo menos.

## Especialización y generalización

Hasta el momento, cada clase de objetos se introduce de forma separada a las demás clases. Las clases pueden definirse también como subconjuntos de otras clases, pero éstos deben constituir siempre conjuntos de objetos similares.

Hablamos entonces de subclases de otras clases que, por tanto, constituyen especializaciones de esas otras clases.

### Ejemplo

*La clase de los caballos es una subclase de la clase de los mamíferos.*

La generalización es la relación inversa a la especialización. Si una clase es una especialización de otra clase, ésta última es una generalización de la primera. Es su superclase.

### Ejemplo

*La clase de los mamíferos es una superclase de la clase de los caballos.*

La relación de especialización puede aplicarse a varios niveles, dando lugar a la jerarquía de clases.

### Ejemplo

*La clase de los caballos es una subclase de la clase de los mamíferos, ella misma subclase de la clase de los animales. La clase de los perros es otra subclase de la clase de los mamíferos. La jerarquía de clases correspondiente aparece representada en la figura 3.4.*

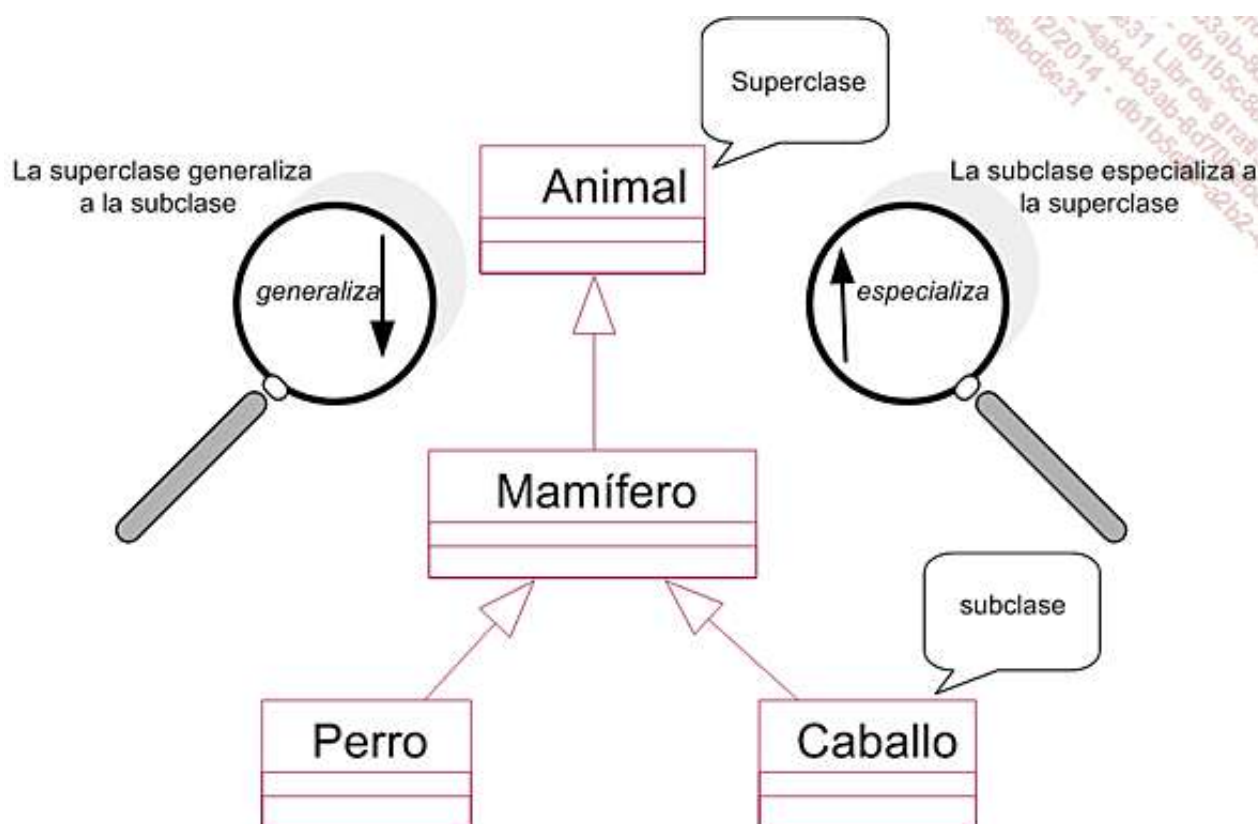


Figura 3.4 - Jerarquía de clases

# Herencia

La herencia es la propiedad que hace que una subclase se beneficie de la estructura y el comportamiento de su superclase. La herencia deriva del hecho de que las subclases son subconjuntos de las superclases. Sus instancias son asimismo instancias de la superclase y, por consiguiente, además de la estructura y el comportamiento introducidos en la subclase, se benefician también de la estructura y comportamiento definidos por la superclase.

## Ejemplo

Tomamos un sistema en el que la clase *Caballo* es una subclase directa de la clase *Animal*. El caballo se describe entonces mediante la combinación de la estructura y del comportamiento derivados de las clases *Caballo* y *Animal*, es decir, mediante los atributos *edad*, *tamaño*, *peso*, *nombre* y *casta* así como los métodos *comer* y *correr*. Esta herencia se ilustra en la figura 3.5.

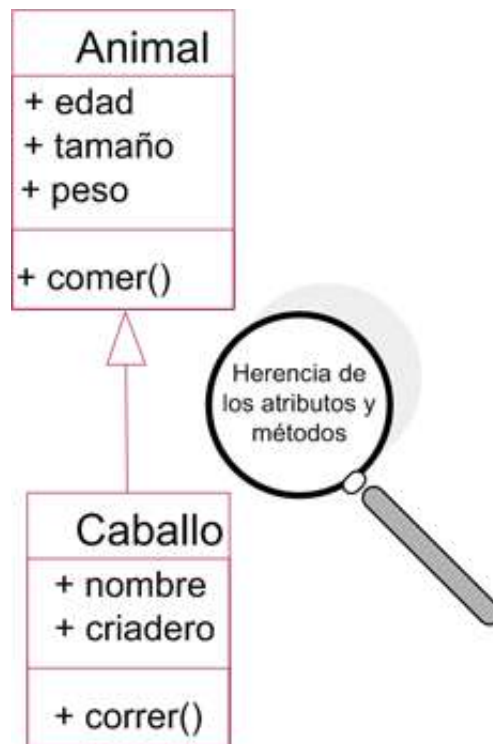


Figura 3.5 - Herencia

La herencia es una consecuencia de la especialización. Sin embargo, los informáticos emplean mucho más a menudo el término *hereda* que *especializa* para designar la relación entre una subclase y su superclase.

## Clases abstractas y concretas

Si examinamos la jerarquía presentada en la figura 3.4 vemos que en ella existen dos tipos de clases:

- Las clases que poseen instancias, es decir, las clases *Caballo* y *Perro*, llamadas clases concretas.
- Las clases que no poseen directamente instancias, como la clase *Animal*. En efecto, si bien en el mundo real existen caballos, perros, etc., el concepto de animal propiamente dicho continuá siendo abstracto. No basta para definir completamente un animal. La clase *Animal* se llama clase abstracta.

La finalidad de las clases abstractas es poseer subclasses concretas y sirven para factorizar atributos y métodos comunes a las subclasses.

### Ejemplo

La figura 3.6 retoma la jerarquía detallando las clases abstractas y las clases concretas. En UML, el nombre de las clases abstractas se escribe en cursiva.

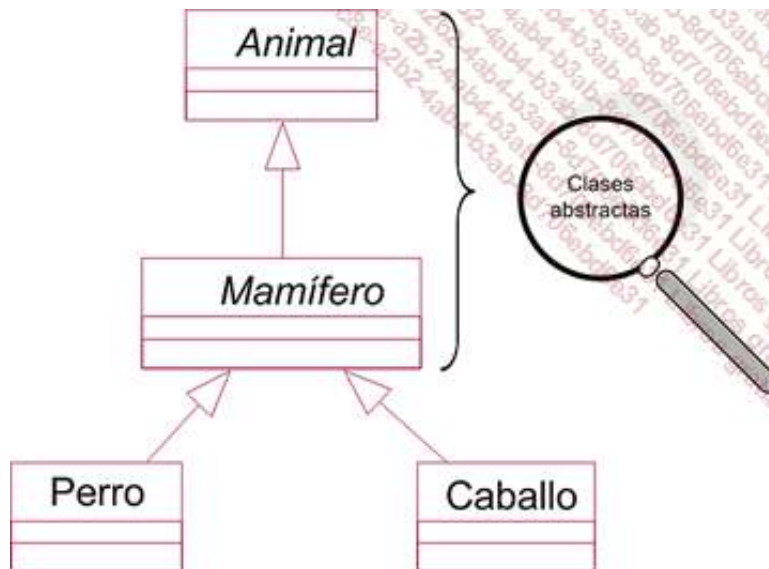


Figura 3.6 - Clases abstractas y clases concretas

## Polimorfismo

El polimorfismo significa que una clase (generalmente abstracta) representa un conjunto formado por objetos diferentes, ya que éstos son instancias de subclases diferentes. Cuando se llama a un método del mismo nombre, esta diferencia se traduce en comportamientos distintos (excepto en los casos en los que el método es común y las subclases lo han heredado de la superclase).

### Ejemplo

Tomemos la jerarquía de clases ilustrada en la figura 3.7. El método *acariciar* tiene un comportamiento diferente según si el caballo es una instancia de *CaballoSalvaje* o de *CaballoDomesticado*. En el primer caso, el comportamiento será un rechazo (que se traducirá en un encabritamiento), mientras que en el segundo el comportamiento será una aceptación.

Si consideramos la clase *Caballo* en su totalidad, tenemos un conjunto de caballos que reaccionan de distinta manera al activarse el método *acariciar*.

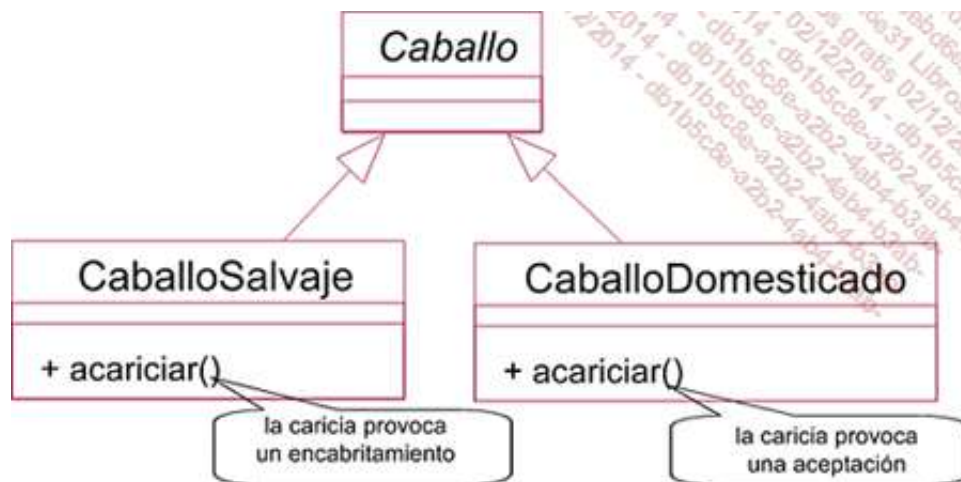


Figura 3.7 - Polimorfismo

## Composición

Un objeto puede ser complejo y estar compuesto por otros objetos. La asociación que une a estos objetos es la composición, que se define a nivel de sus clases, pero cuyos vínculos se establecen entre las instancias de las clases. Los objetos que forman el objeto compuesto se denominan *componentes*.

### Ejemplo

Un caballo es un ejemplo de objeto complejo. Está formado por diferentes órganos (patas, cabeza, etc.). La representación gráfica de esta composición puede verse en la figura 3.8.

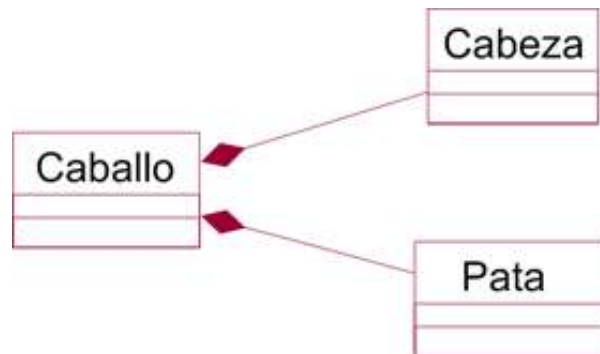


Figura 3.8 - Composición

La composición puede adoptar dos formas:

- composición débil o agregación;
- composición fuerte.

En la composición débil, los componentes pueden ser compartidos por varios objetos complejos. En la composición fuerte, los componentes no pueden compartirse y la destrucción del objeto compuesto conlleva la destrucción de sus componentes.

### Ejemplo

Si retomamos el ejemplo precedente con el supuesto de un caballo de carreras enjaezado y añadimos a sus componentes una silla, obtenemos:

- Una composición fuerte para las patas y la cabeza. Efectivamente, las patas y la cabeza no pueden compartirse y la desaparición del caballo conlleva la desaparición de sus órganos.
- Una agregación o composición débil para la silla.

Todo ello se ilustra en la figura 3.9.



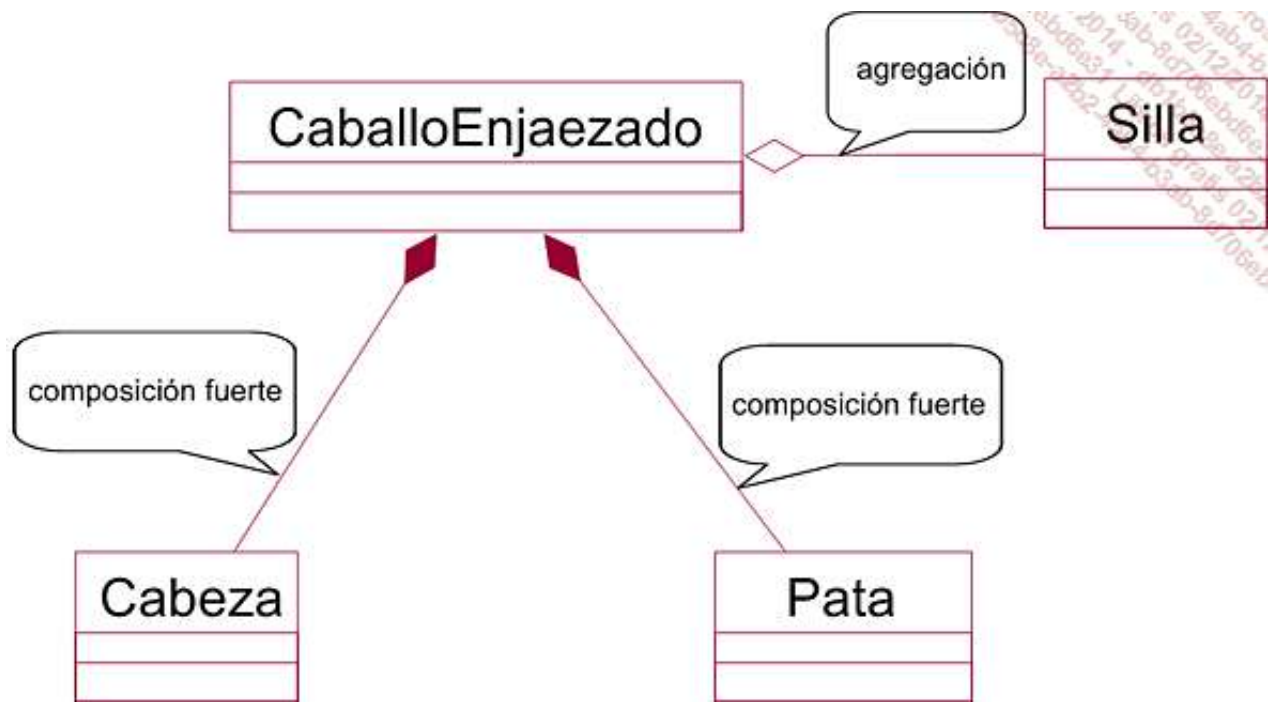


Figura 3.9 - Composición y agregación

## La especialización de los elementos: la noción de estereotipo en UML

En el presente capítulo, hemos abordado los conceptos relativos a la orientación a objetos. Ahora introduciremos los estereotipos de UML cuya finalidad es especializar dichos conceptos.

Los estereotipos están formados por palabras clave que explicitan la especialización. Estas palabras clave aparecen entre comillas.

La especialización se realiza independientemente del sistema que se desee modelar.

### Ejemplo

*El concepto de clase abstracta es un concepto especializado del concepto de clase. Hemos visto que una clase abstracta se representa como una clase con un nombre en cursiva. Esta representación gráfica incluye un estereotipo implícito, pero también podemos prescindir de poner el nombre de la clase en cursiva y precisar de forma explícita el estereotipo «abstract».*

*Presentamos el estereotipo explícito en la figura 3.10. Éste puede emplearse al escribir a mano diagramas UML.*

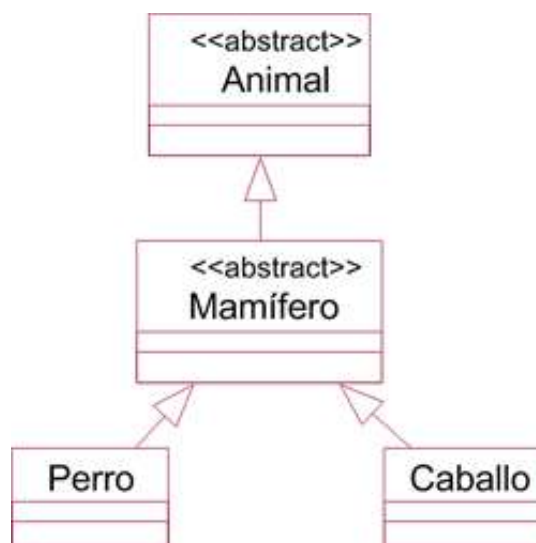


Figura 3.10 - Estereotipo explícito «abstract»

El esquema es equivalente al de la figura 3.11, ya introducido en la figura 3.6.

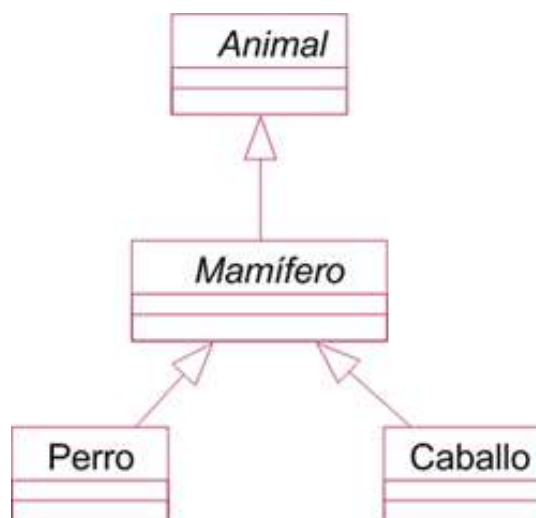


Figura 3.11 - Estereotipo implícito «abstract»

## Conclusión

La orientación a objetos constituye la base del UML. Ésta está formada por conceptos (objetos, clases, especialización, composición) y principios (abstracción, encapsulación). Estos elementos hacen de la orientación a objetos un soporte real para el modelado de sistemas complejos y para la programación de los mismos, más allá de UML.

En los capítulos siguientes, veremos cómo los diferentes diagramas de UML se apoyan en conceptos y principios propios de la orientación a objetos.

# Introducción

El objetivo del presente capítulo es mostrar los casos de uso empleados para describir los requisitos funcionales esperados durante la redacción del pliego de condiciones del sistema o las funcionalidades de un sistema existente.

Los casos de uso de un sistema contienen los requisitos funcionales deseados o existentes, los actores (usuarios del sistema) y las relaciones que unen a actores y funcionalidades. Este conjunto determina asimismo las fronteras del sistema, es decir, las funcionalidades del sistema y aquellas que son externas a él.

Los casos de uso sirven de soporte para las etapas de modelado, desarrollo y validación. Son una referencia del diálogo entre informáticos y clientes y, por consiguiente, constituyen una base para elaborar los aspectos funcionales del pliego de condiciones.

## Casos de uso

Los casos de uso describen en forma de acciones y reacciones el comportamiento del sistema, estudiado desde el punto de vista del usuario. Definen los límites del sistema y sus relaciones con el entorno.

Ahora bien, esta definición debe completarse, ya que no especifica si un caso de uso debe describir la totalidad o sólo una parte del diálogo entre el usuario y el sistema. Podría formularse así:

"Entre un usuario y el sistema, los casos de uso describen las interacciones vinculadas con un objetivo funcional del usuario".

Los casos de uso explicitan los requisitos funcionales del sistema relativos a uno de los objetivos del usuario. Éstos se denominan también, de manera más precisa, casos de uso con objetivo usuario.

### Ejemplo

*Consideremos como sistema un criadero de caballos. La compra de un caballo por parte de un cliente constituye un caso de uso.*

## Actor

Un usuario externo al sistema puede desempeñar diferentes funciones en relación con el sistema. Una pareja (usuario, función) constituye un actor específico, designado en UML únicamente por el nombre de la función.

La definición se extiende a los demás sistemas que interactúan con el sistema. Estos forman tantos actores como funciones desempeñadas.

Debemos distinguir dos categorías de actores:

- Los actores primarios, para los cuales el objetivo del caso de uso es esencial.
- Los actores secundarios, que interactúan con el caso de uso, pero cuyo objetivo no es esencial.

### Ejemplo

*Retomemos el ejemplo del caso de uso de compra de un caballo por parte de un cliente. El comprador del caballo es un actor primario. La parada de sementales del estado que registra el certificado de venta es un actor secundario.*

## Escenario

Un escenario es una instancia de un caso de uso en la cual se fijan todas las condiciones relativas a los diferentes eventos. Por tanto, a la hora del desarrollo, no existen alternativas.

A un caso de uso determinado corresponden varios escenarios.

Al igual que las clases, que albergan los aspectos comunes de las instancias, los casos de uso describen de manera común el conjunto de escenarios utilizando derivaciones condicionales para representar las diferentes alternativas.

### Ejemplo

*La compra de Jorgelina por parte de Fien constituye un ejemplo de escenario del caso de uso de compra de un caballo. Todas las alternativas del desarrollo se conocen, ya que Fien ha comprado a Jorgelina.*



## Relación de comunicación

La relación que vincula a un actor con un caso de uso se denomina relación de comunicación.

Esta relación da soporte a diferentes modelos de comunicación, por ejemplo:

- Los servicios que el sistema debe suministrar a cada uno de los actores del caso de uso.
- Las informaciones del sistema que un actor puede introducir, consultar o modificar.
- Los cambios que intervienen en el entorno, de los cuales un actor informa al sistema.
- Los cambios que intervienen dentro del sistema, de los cuales el sistema informa a un actor.

### Ejemplo

*Cuando Fien compró a Jorgelina recibió de la granja de cría una serie de informaciones (una propuesta de precio, los papeles de la yegua, la libreta de vacunación, etc.) y suministró otras informaciones (una contraoferta de precio, una promesa de compra, etc.).*

## Diagrama de los casos de uso

El diagrama de los casos de uso muestra los casos de uso representados en forma de elipses y a los actores en forma de personajes. También indica las relaciones de comunicación que los vinculan.

### Ejemplo

El caso de uso de compra de un caballo se representa en la figura 4.1:



Figura 4.1 - Caso de uso de compra de un caballo

El sistema que responde al caso de uso puede representarse mediante un rectángulo en cuyo interior aparece el caso.

### Ejemplo

En el ejemplo precedente, el sistema, es decir, la granja de cría de caballos, se ilustra en la figura 4.2.

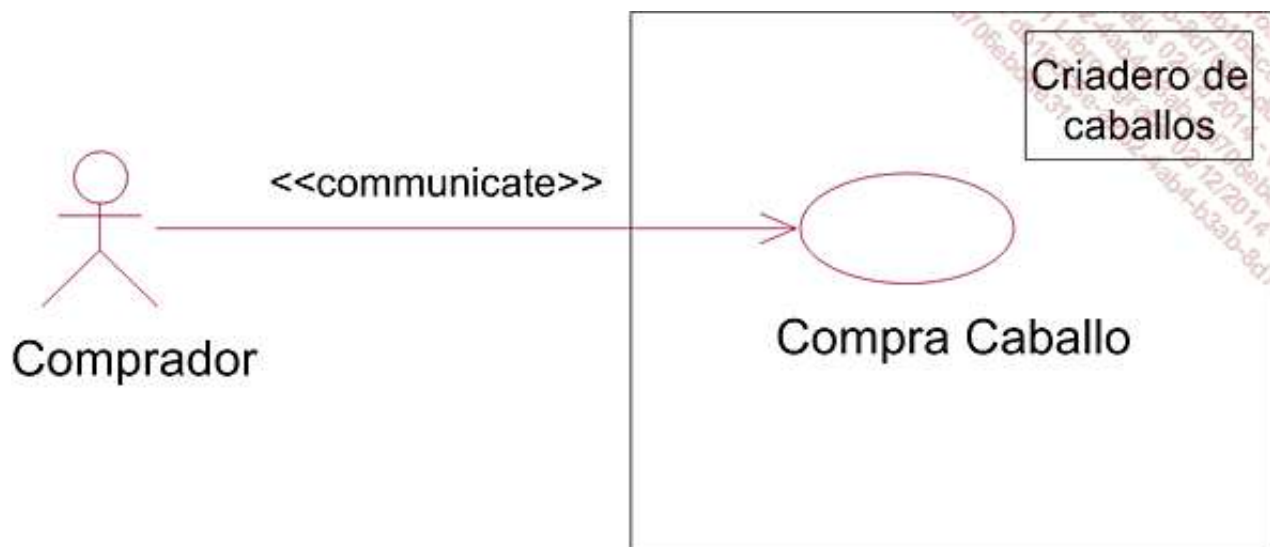


Figura 4.2 - Sistema de un caso de uso

Los actores secundarios se representan del mismo modo que los actores primarios. Muchas veces el sentido de la relación de comunicación entre los actores secundarios y el sistema se invierte con respecto al sentido de la relación entre los actores primarios y el sistema. En efecto, es el sistema y no el actor quien inicia la comunicación.

### Ejemplo

En el ejemplo precedente, las paradas de sementales del estado realizan el cambio de propietario del caballo. Las paradas son un actor secundario (ver figura 4.3).

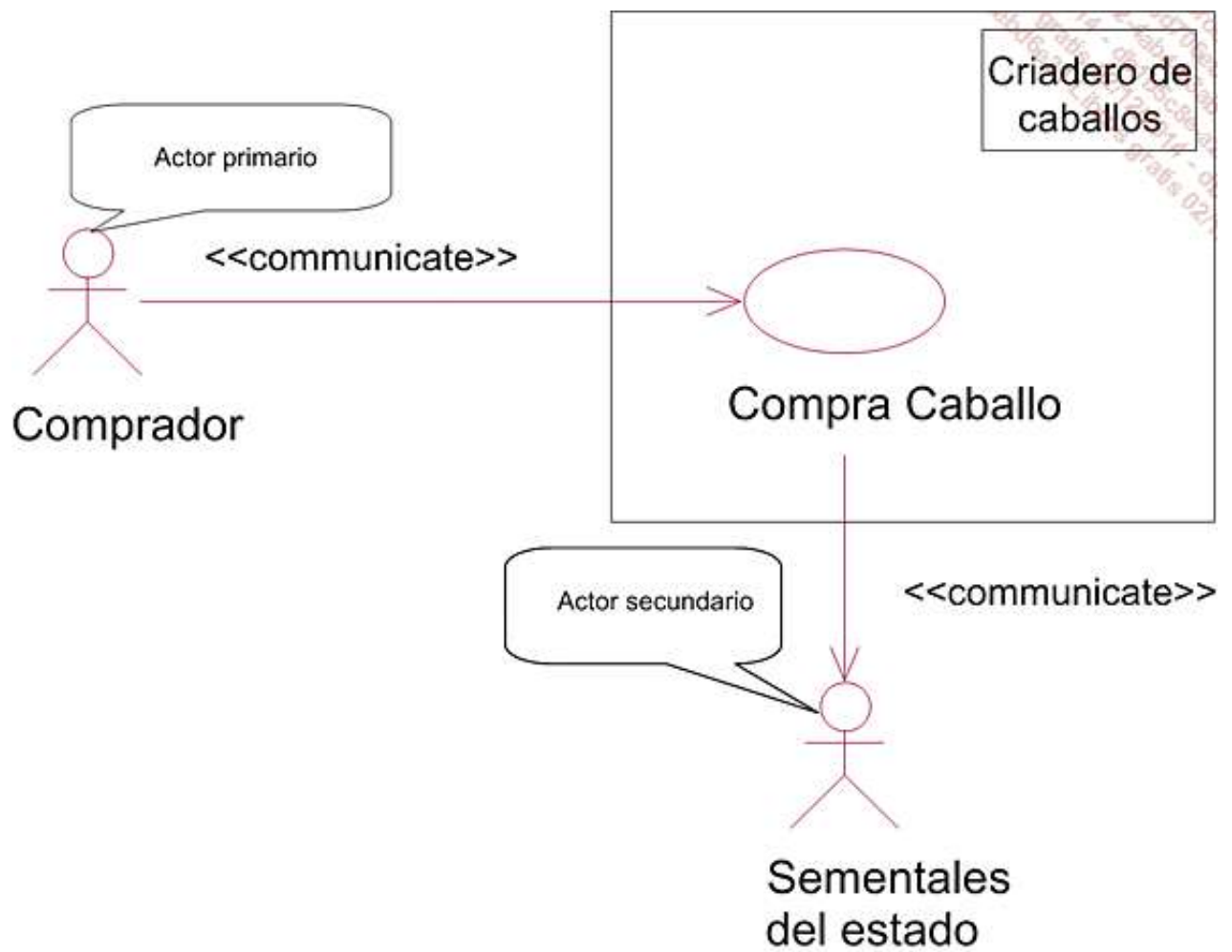


Figura 4.3 - Actores primarios y secundarios de un caso de uso

# Relaciones entre los casos de uso

## 1. Relación de inclusión

La relación de inclusión sirve para enriquecer un caso de uso con otro. Dicho enriquecimiento se lleva a cabo mediante una inclusión imperativa y, por tanto, es sistemático.

El caso de uso incluido existe únicamente con ese propósito, ya que no responde a un objetivo de un actor primario. Estos casos de uso son subfunciones.

La inclusión sirve para compartir una funcionalidad común entre varios casos de uso. También puede emplearse para estructurar un caso de uso describiendo sus subfunciones.

En el diagrama de casos de uso, estas relaciones se representan mediante una flecha discontinua acompañada del estereotipo «include».

### Ejemplo

A la hora de adquirir un semental, el comprador se asegurará de que éste tenga las vacunas en regla. Por consiguiente, el caso de uso de compra de un semental incluye dicha verificación (ver figura 4.4).

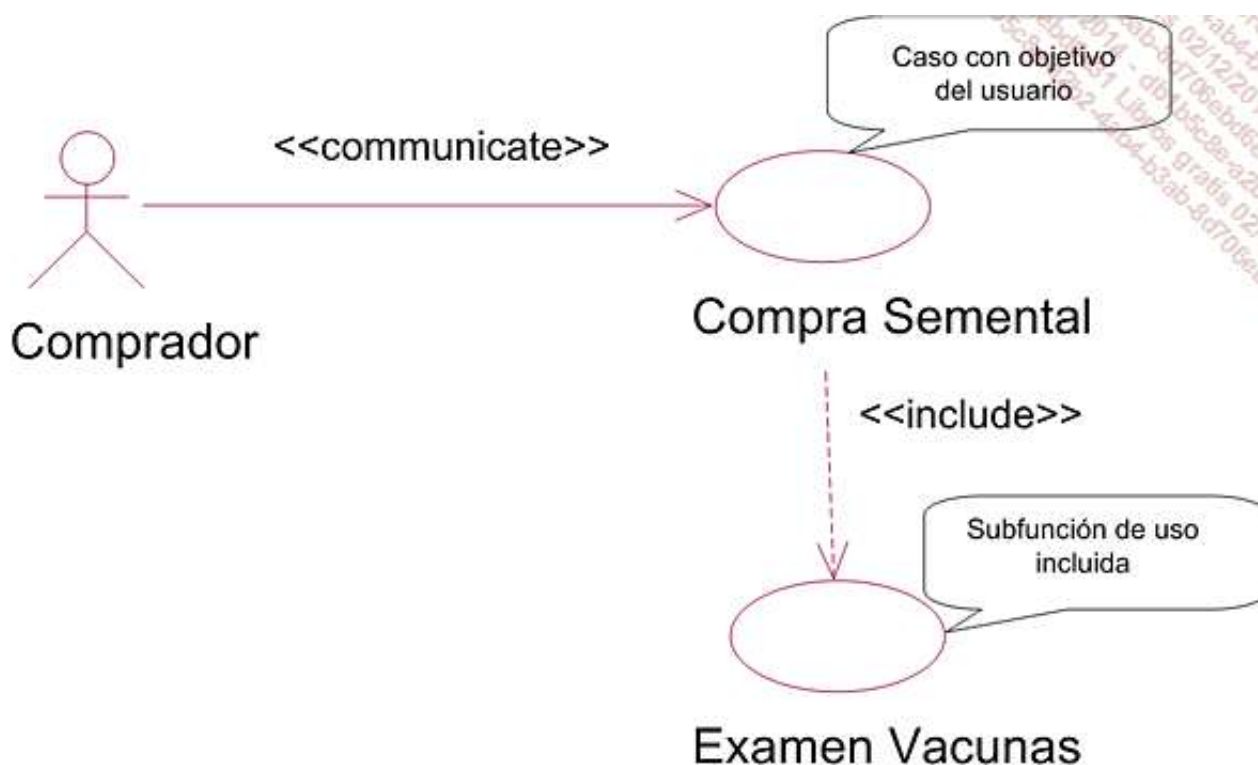


Figura 4.4 - Inclusión de un caso de uso

La puesta en común del caso de uso de comprobación de las vacunas se ilustra en la figura 4.5, ya que este caso de subfunción es igualmente pertinente para la compra de una yegua.

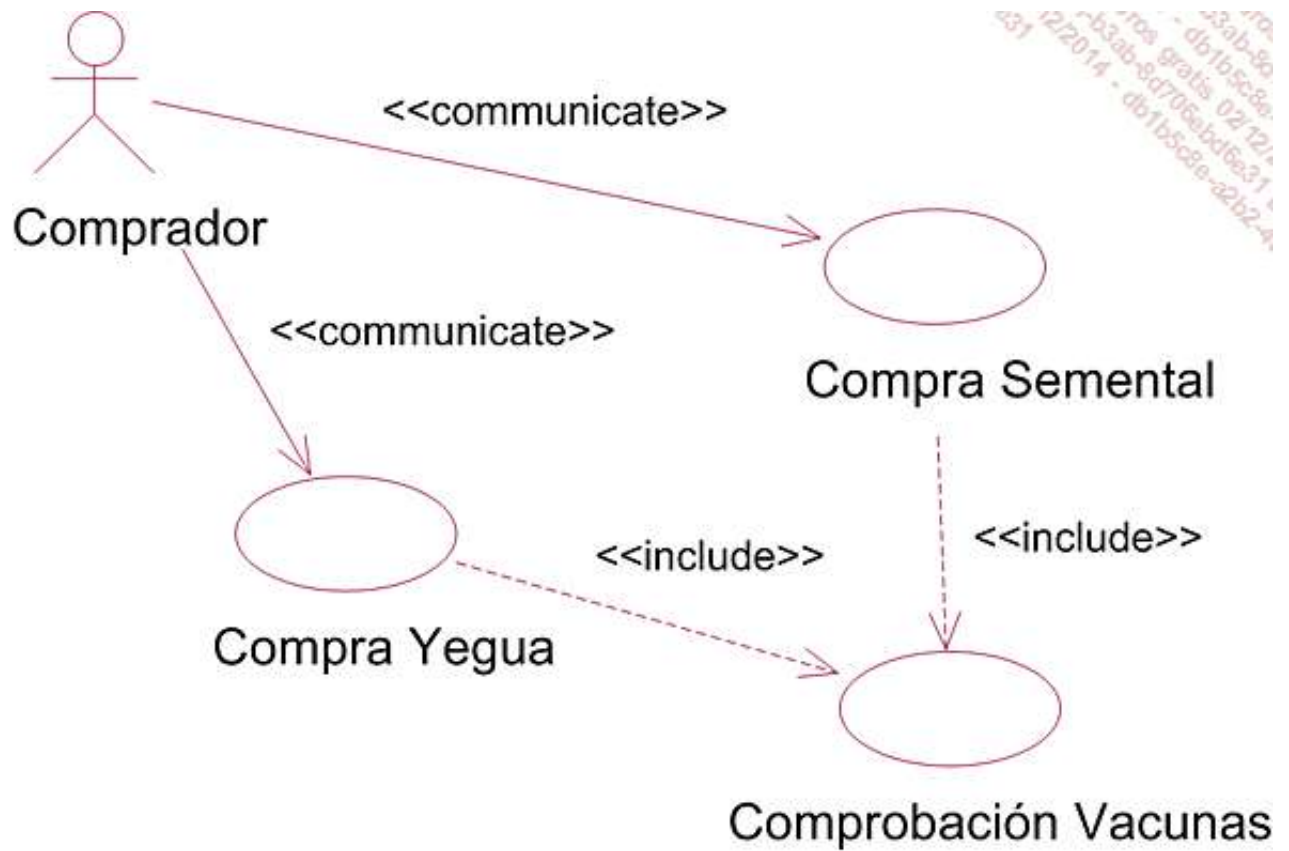


Figura 4.5 - Puesta en común de un caso de uso incluido

La inclusión puede emplearse también para descomponer el interior de un caso de uso sin compartir el caso incluido. En la figura 4.6, la comprobación de los partos de una yegua no se comparte, pero su presencia ilustra bien que dicha comprobación forma parte de los puntos estudiados durante la compra de la yegua.

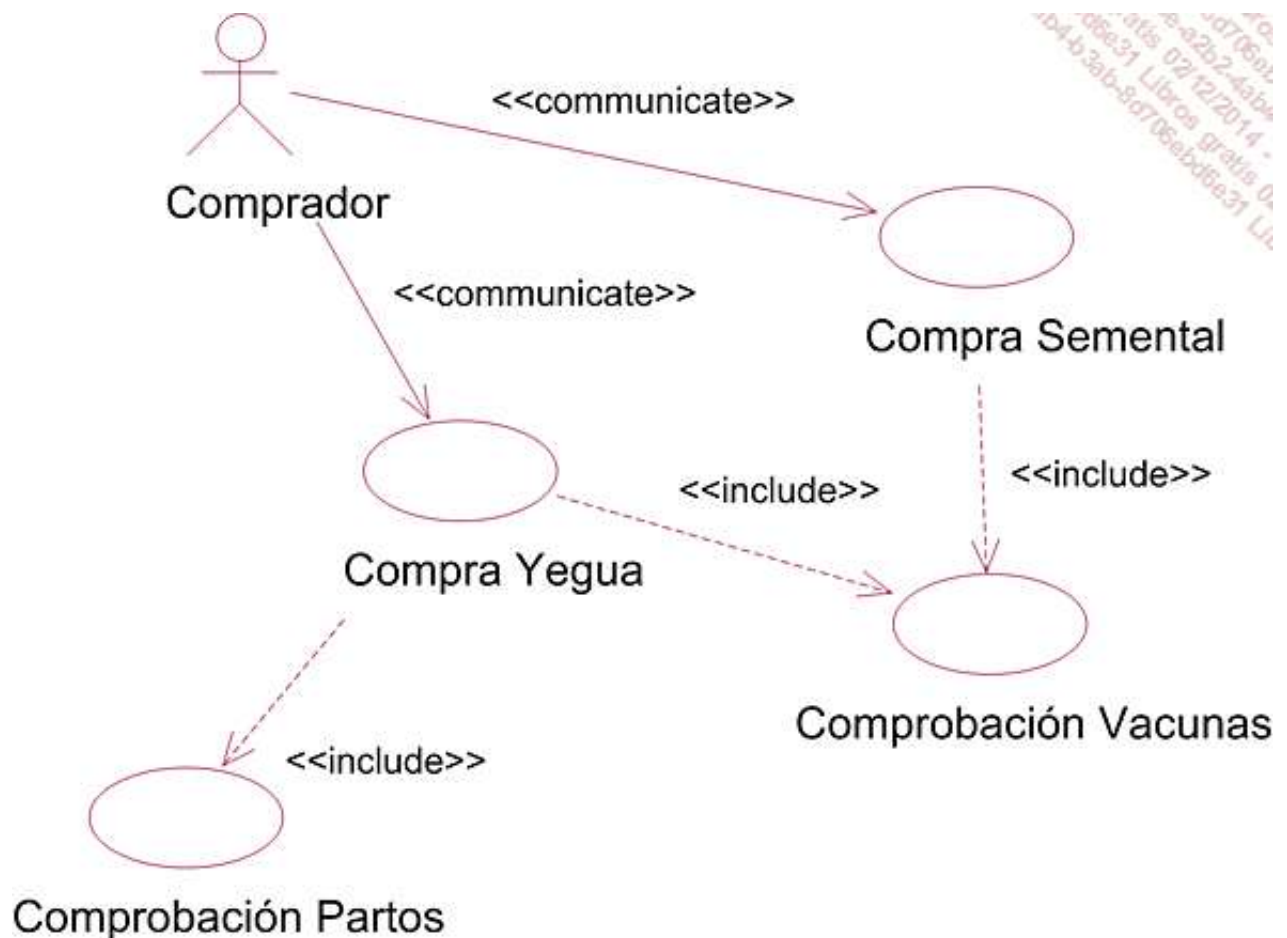


Figura 4.6 - Descomposición de un caso de uso por inclusión

## 2. Relación de extensión

Al igual que la relación de inclusión, la relación de extensión enriquece un caso de uso mediante un caso de uso de subfunción. El enriquecimiento es análogo al de la relación de inclusión, no obstante, es opcional.

En el caso de uso básico, la extensión se hace en una serie de puntos concretos y previstos en el momento del diseño, llamados *puntos de extensión*.

La aplicación de cada extensión se decide durante el desarrollo de un escenario. Por consiguiente, el caso de uso básico puede emplearse sin estar extendido.

Como ocurre con la inclusión, la extensión sirve para estructurar un caso de uso o para compartir un caso de uso de subfunción.

En el diagrama de los casos de uso, esta relación se representa mediante una flecha discontinua acompañada del estereotipo «extend».

### Ejemplo

A la hora de adquirir un caballo, el comprador puede examinar el carácter del animal o su pelaje. Por consiguiente, el caso de uso de compra de un caballo puede extenderse con alguna de esas verificaciones (ver figura 4.7).

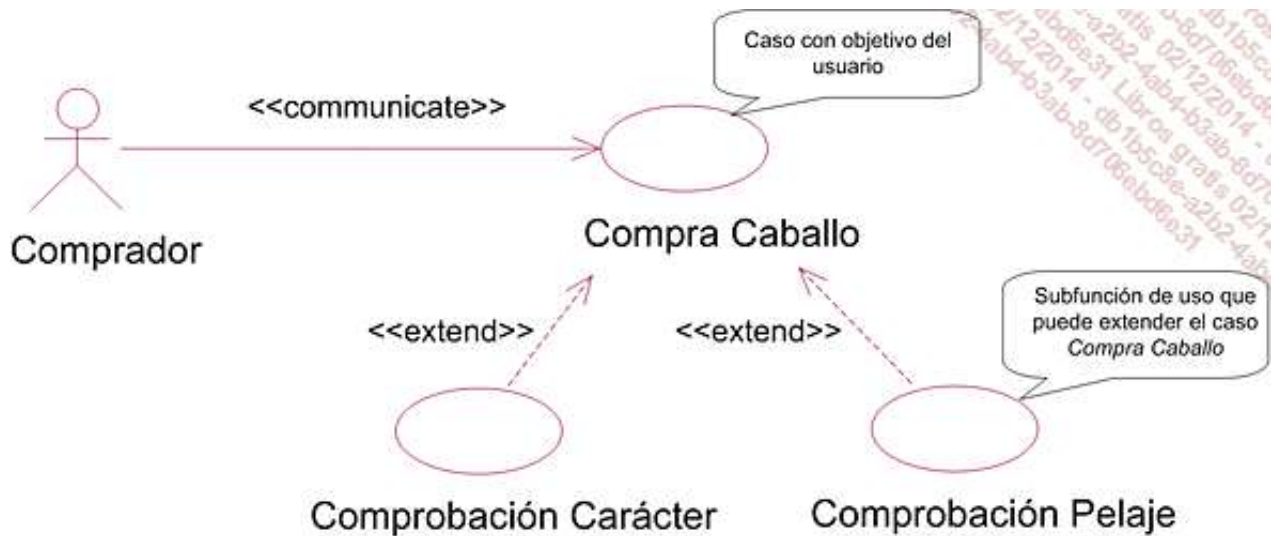


Figura 4.7 - Extensión de un caso de uso

### Ejemplo

Tomemos el caso en el que la compra de un semental se modela separadamente del de una yegua. Opcionalmente, podemos comprobar la capacidad de dar a luz de la yegua (ver figura 4.8). Por otro lado, los casos de uso de verificación del carácter y del pelaje se comparten.

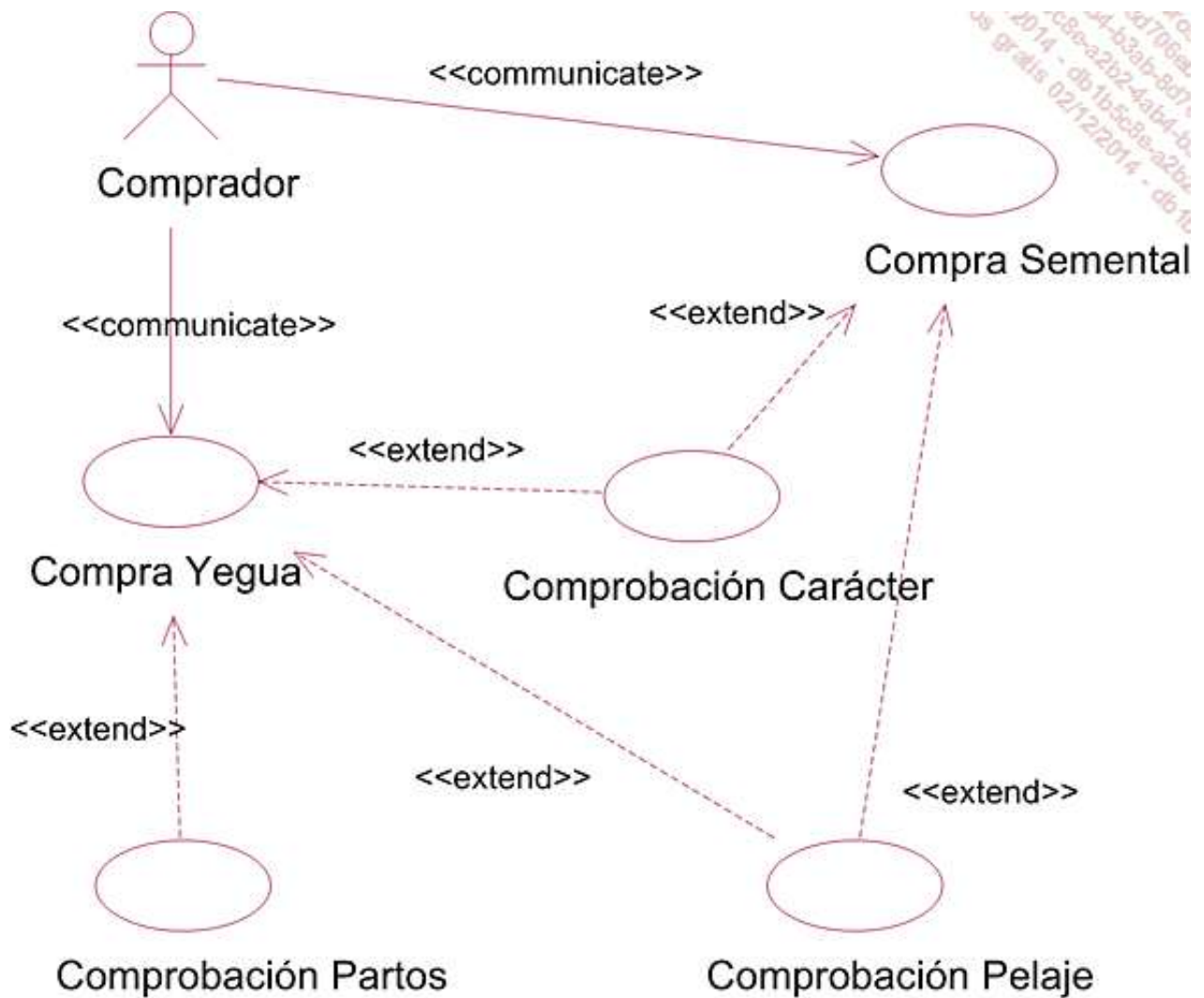


Figura 4.8 - Extensiones compartidas de casos de uso



### 3. Especialización y generalización de los casos de uso

Como ya vimos en el capítulo Conceptos de la orientación a objetos con las clases de objetos, también es posible especializar un caso de uso en otro. Obtenemos así un subcaso de uso.

Al igual que ocurría con las clases, el subcaso hereda el comportamiento y las relaciones de comunicación, inclusión y extensión del supercaso de uso.

Muchas veces el supercaso de uso es abstracto, es decir, corresponde a un comportamiento parcial completado en el subcaso de uso.

Los subcasos de uso tienen el mismo nivel que sus supercasos. Si el supercaso es un caso con objetivo usuario, lo mismo ocurrirá con el subcaso. Si es un caso de subfunción, el subcaso será también una subfunción.

En el diagrama de los casos de uso, la relación de especialización se representa mediante una flecha de especialización idéntica a la que une las subclases con las superclases. El nombre de los casos de uso abstractos se escribe en cursiva (o se acompaña del estereotipo «abstract»).

#### Ejemplo

*El caso de uso de compra de un caballo se especializa en dos subcasos: la compra de una yegua o la compra de un semental. Se trata de un caso abstracto y su nombre aparece en cursiva. La figura 4.9 ilustra esta especialización.*

*Los casos de uso de compra de una yegua y de compra de un semental son casos con objetivo usuario y comunican con el comprador. La relación de comunicación que existe entre el caso de uso de compra del caballo y el Comprador se hereda en los dos subcasos de uso.*

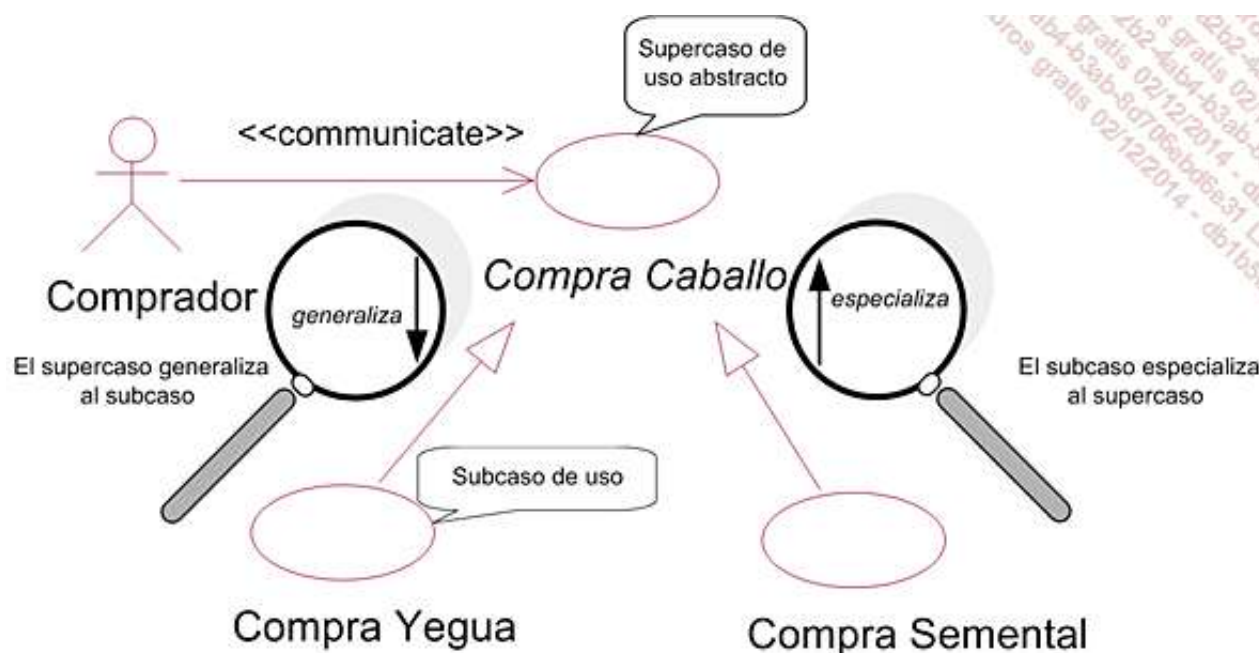


Figura 4.9 - Especialización de un caso de uso

#### Ejemplo

*Las relaciones de extensión relativas a las diferentes inclusiones y extensiones de verificación pueden factorizarse en el caso abstracto. Estas se heredan entonces en los subcasos como ocurre en la relación de comunicación del ejemplo precedente (ver figura 4.10).*

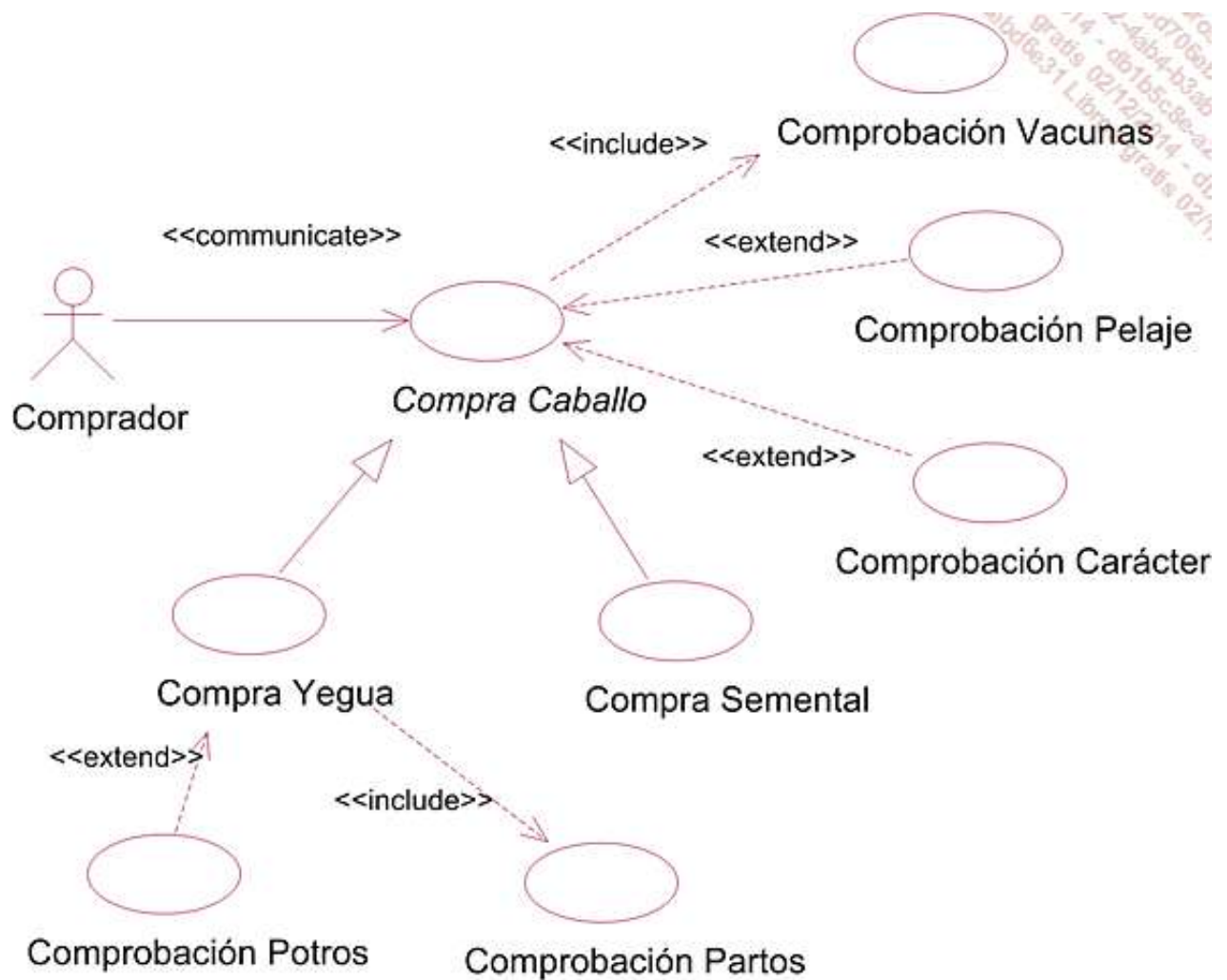


Figura 4.10 - Factorización de inclusión y de extensión

## Representación textual de los casos de uso

La representación textual de los casos de uso no se especifica en UML. No obstante, se utiliza habitualmente y por ello la hemos incorporado en la presente obra.

La representación en forma textual de los casos de uso da una descripción de sus componentes, acciones y reacciones. El contenido de la representación textual es el siguiente:

- Nombre del caso de uso.
- Actor primario.
- Sistema al que pertenece el caso de uso.
- Participantes (conjunto de actores).
- El nivel del caso de uso puede ser:
  - Un objetivo usuario.
  - Una subfunción.
- Condiciones previas que deben cumplirse para que el caso de uso pueda ser ejecutado.
- Operaciones del escenario principal.
- Extensiones.

Caso de uso	Nombre del caso de uso
Actor primario	Nombre del actor primario
Sistema	Nombre del sistema
Participantes	Nombre de los participantes
Nivel	Objetivo usuario o subfunción
Condiciones previas	Condiciones que deben cumplirse para ejecutar el caso de uso
Operaciones	
1	Operación 1
2	Operación 2
3	Operación 3
4	Operación 4
5	Operación 5
Extensiones	
1.A	Condición de aplicación de la extensión A sobre la operación 1
1.A.1	Operación 1 de la extensión A sobre la operación 1
1.A.2	Operación 2 de la extensión A sobre la operación 1
1.B	Condición de aplicación de la extensión B sobre la operación 1
1.B.1	Operación 1 de la extensión B sobre la operación 1
4.A	Condición de aplicación de la extensión A sobre la operación 4
4.A.1	Operación 1 de la extensión A sobre la operación 4

### Ejemplo

A continuación presentamos el caso de uso de compra de una yegua. Las extensiones se numeran por la línea de la operación a la cual se aplican, seguida de una letra que permite distinguir las extensiones de una

misma línea. Luego se numeran las operaciones de una extensión de la misma forma que las operaciones del escenario principal.

Caso de uso	Compra de una yegua
Actor primario	Comprador
Sistema	Criadero de caballos
Participantes	Comprador, Parada de sementales del estado
Nivel	Objetivo usuario
Condición previa	La yegua está a la venta
Operaciones	
1	Elegir la yegua
2	Comprobar las vacunas
3	Examinar los partos
4	Recibir una propuesta de precio
5	Evaluar la propuesta de precio
6	Pagar el precio de la yegua
7	Cumplimentar los papeles de venta
8	Registrar la venta en la parada de sementales del estado
9	Ir a buscar la yegua
10	Transportar la yegua
Extensiones	
2.A	¿Son correctas las vacunas?
2.A.1	Si sí, continuar
2.A.2	Si no, salir
3.A	¿Es correcta la verificación de los partos?
3.A.1	Si sí, continuar
3.A.2	Si no, salir
5.A	¿Es correcto el precio?
5.A.1	Si sí, continuar
5.A.2	Si no, negociar el importe y volver a ejecutar la etapa 5

## Conclusión

Los casos de uso sirven para:

- Expresar los requisitos funcionales que los usuarios comunicaron al sistema durante la redacción del pliego de condiciones.
- Comprobar que el sistema cumple dichos requisitos en el momento de la entrega.
- Determinar las fronteras del sistema.
- Escribir la documentación del sistema.
- Confeccionar los juegos de test.

Los casos de uso ofrecen una técnica de representación adecuada para dialogar con el usuario ya que su formalismo es cercano al lenguaje natural. Se aconseja incorporar un léxico para evitar posibles confusiones.

Más adelante estudiaremos cómo descubrir los objetos utilizando los diagramas de secuencia asociados a los casos de uso.

# Ejercicios

## 1. El hipódromo

Un hipódromo ofrece a sus clientes la posibilidad de asistir a las carreras y de realizar apuestas.

¿Cuáles son los actores que interactúan con estos servicios?

Construya el diagrama de casos de uso.

## 2. El club ecuestre

Un club ecuestre pone a disposición de los clientes establos para guardar los caballos y ofrece cursos de equitación y paseos. Sólo los socios tienen acceso a los cursos y a los servicios de establo. Los demás clientes tienen la posibilidad de participar en los paseos y de convertirse en socios.

¿Cuáles son los actores que interactúan con estos servicios?

Construya el diagrama de casos de uso.

## 3. El tiovivo de caballos de madera

Un tiovivo de caballos de madera ofrece a sus clientes la posibilidad de dar una vuelta previo pago de una cantidad de dinero.

¿Cuáles son los actores vinculados a este servicio?

Construya el diagrama de casos de uso.

Dé la representación textual correspondiente al diagrama.

# Introducción

El objetivo del presente capítulo es explicar de qué manera UML representa las interacciones entre objetos. En el capítulo Conceptos de la orientación a objetos, vimos que los objetos de un sistema poseen su propio comportamiento e interactúan entre sí para dotar al sistema de una dinámica global. En el capítulo Modelado de los requisitos, estudiamos la forma en que los casos de uso representan las acciones y reacciones entre un actor externo y el sistema. Desde el punto de vista del modelado, esos dos tipos de interacciones se distinguen por su diferencia interna/externa, pero no por su naturaleza.

Para responder a la necesidad de representación de las interacciones entre objetos, UML propone dos tipos de diagramas:

- El diagrama de secuencia se centra en aspectos temporales.
- El diagrama de comunicación se centra en la representación espacial.

En el presente capítulo estudiaremos ambos tipos de diagramas. Más tarde examinaremos cómo descubrir progresivamente los objetos que componen un sistema. Dicho descubrimiento se basará en las interacciones entre objetos que intervienen en los casos de uso del sistema. Para representar las interacciones nos decantaremos por el diagrama de secuencia, ya que suele ser la opción preferida por las personas que se encargan de modelar los proyectos.



El diagrama de comunicación se conoce con ese nombre desde UML 2. En UML 1 se denominaba diagrama de colaboración.

# Diagrama de secuencia

## 1. Definición

El diagrama de secuencia describe la dinámica del sistema. A menos que se modele un sistema muy pequeño, resulta difícil representar toda la dinámica de un sistema en un único diagrama. Por tanto, la dinámica completa se representará mediante un conjunto de diagramas de secuencia, cada uno de ellos vinculado generalmente a una subfunción del sistema.

El diagrama de secuencia describe las interacciones entre un grupo de objetos mostrando de forma secuencial los envíos de mensajes entre objetos. El diagrama puede asimismo mostrar los flujos de datos intercambiados durante el envío de mensajes.

- Para interactuar entre sí, los objetos se envían mensajes. Durante la recepción de un mensaje, los objetos se vuelven activos y ejecutan el método del mismo nombre. Un envío de mensaje es, por tanto, una llamada a un método.

## 2. Línea de vida de un objeto

Dado que representa la dinámica del sistema, el diagrama de secuencia hace entrar en acción las instancias de clases que intervienen en la realización de la subfunción a la que está vinculado. A cada instancia se asocia una línea de vida que muestra las acciones y reacciones de la misma, así como los periodos durante los cuales ésta está activa, es decir, durante los que ejecuta uno de sus métodos.

La representación gráfica de la línea de vida se ilustra en la figura 5.1.

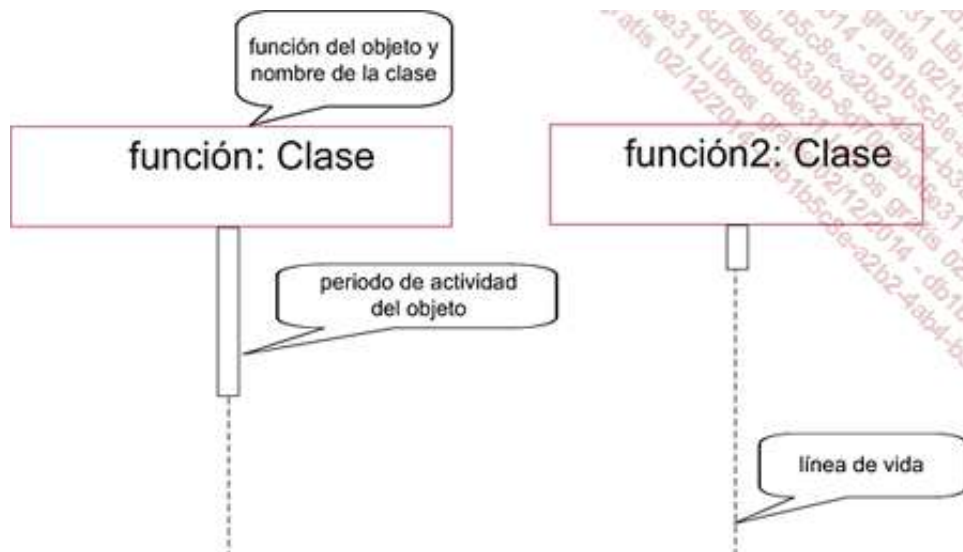


Figura 5.1 - Líneas de vida

- La notación "funcion: Clase" representa la función de una instancia seguida del nombre de su clase. Para simplificar, en esta obra consideraremos que la función de la instancia corresponde a su nombre, al igual que ocurría en UML 1. Si sólo una instancia de la clase participa en el diagrama de secuencia, la función de la instancia es opcional. El nombre de la clase puede también omitirse en las etapas preliminares del modelado, pero debe especificarse lo antes posible.
- Los diagramas de secuencia contienen varias líneas de vida, ya que tratan de las interacciones entre varios objetos.



### 3. Envío de mensajes

Los envíos de mensajes se representan mediante flechas horizontales que unen la línea de vida del objeto emisor con la línea de vida del objeto destinatario (ver figura 5.2).



Figura 5.2 - Envío de un mensaje

En la figura 5.2, el objeto de la izquierda envía un mensaje al objeto de la derecha. El mensaje da lugar a la ejecución del método *mensaje* del objeto de la derecha, lo que provoca su activación.

Los mensajes se numeran secuencialmente a partir de uno. Si un mensaje se envía antes de que concluya el tratamiento del precedente, es posible utilizar una numeración compuesta (ver figura 5.3) en la que el envío del mensaje 2 se produzca durante la ejecución del mensaje 1.

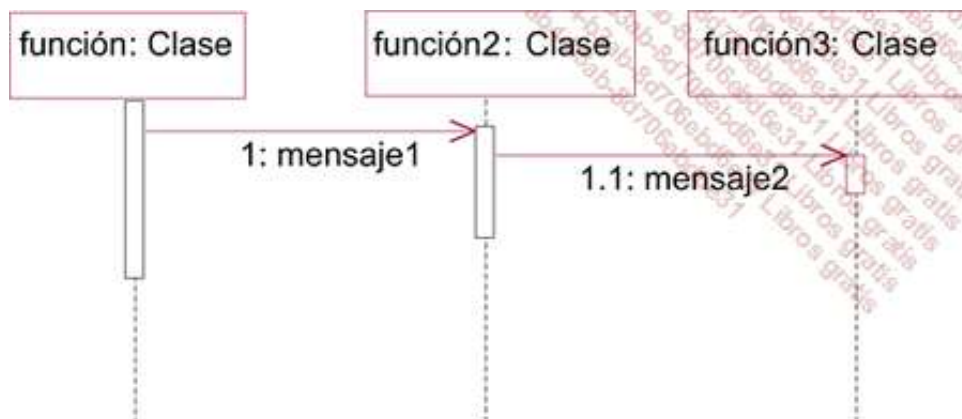


Figura 5.3 - Numeración de los mensajes



La numeración de los mensajes no es obligatoria. No obstante, resulta práctica para mostrar las activaciones anidadas.

También es posible transmitir información; ésta se representa mediante parámetros transmitidos con el mensaje (ver figura 5.4).

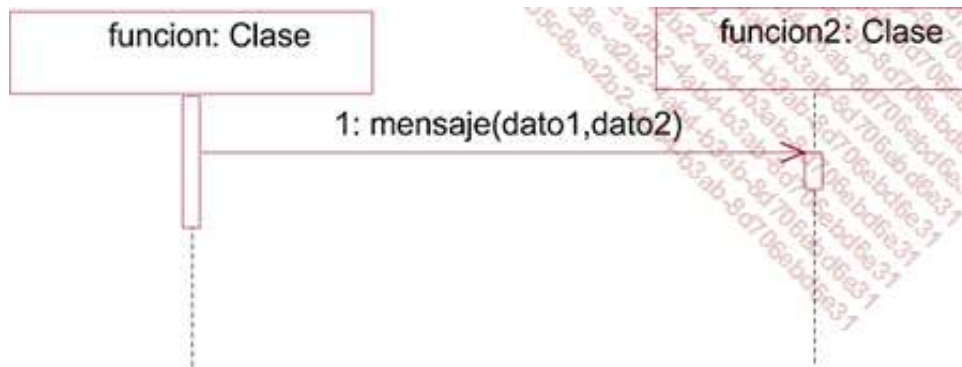


Figura 5.4 - Transmisión de datos durante el envío de un mensaje

Existen diferentes tipos de envíos de mensajes. En la figura 5.5 ofrecemos una explicación gráfica.

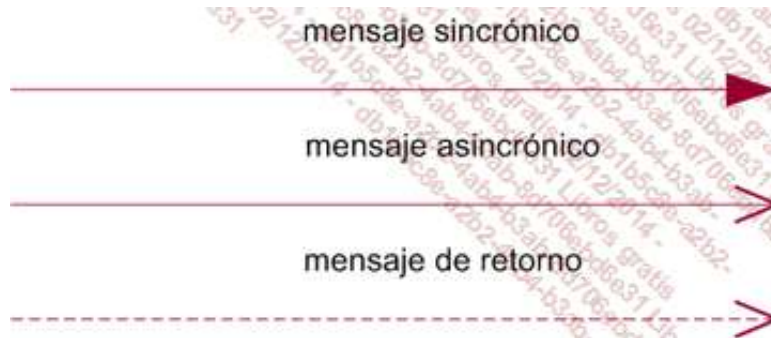


Figura 5.5 - Diferentes tipos de mensajes

El mensaje síncrono es el utilizado con mayor frecuencia. Su uso significa que el expedidor del mensaje espera que la activación del método mencionado por el destinatario finalice antes de continuar su actividad.

En los mensajes asíncronos, el expedidor no espera el término de la activación invocada por el destinatario. Esto se produce al modelar sistemas en los que los objetos pueden funcionar en paralelo (es el caso de los sistemas multi-thread, donde los tratamientos se efectúan en paralelo).

### Ejemplo

*Un jinete da una orden a su caballo, luego le da una segunda orden sin esperar a que concluya la ejecución de la primera. La primera orden constituye un ejemplo de envío de mensaje asíncrono.*



En UML 1 la representación de un mensaje asíncrono se realizaba con media flecha superior. En UML 2 se emplea una flecha completa.

El mensaje de retorno a la llamada a un método no es sistemático, ya que no todos los métodos devuelven un resultado.

Los objetos pueden enviarse mensajes a sí mismos. La representación de tales mensajes se ilustra en la figura 5.6.

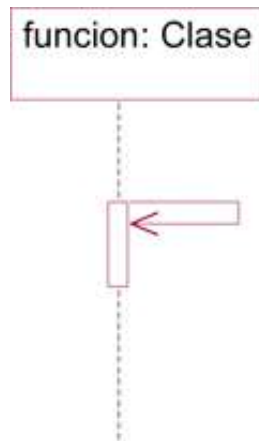


Figura 5.6 - Envío de un mensaje a uno mismo

## 4. Creación y destrucción de objetos

El diagrama de secuencia describe la dinámica de un sistema. Ésta a menudo contiene creaciones y destrucciones de objetos.

La creación de objetos se representa mediante un mensaje específico que da lugar al principio de la línea de vida del nuevo objeto.

La destrucción de objetos es un mensaje enviado a un objeto existente y que da lugar a la finalización de su línea de vida. Se representa mediante una cruz.

Los dos tipos de mensajes se ilustran en la figura 5.7.

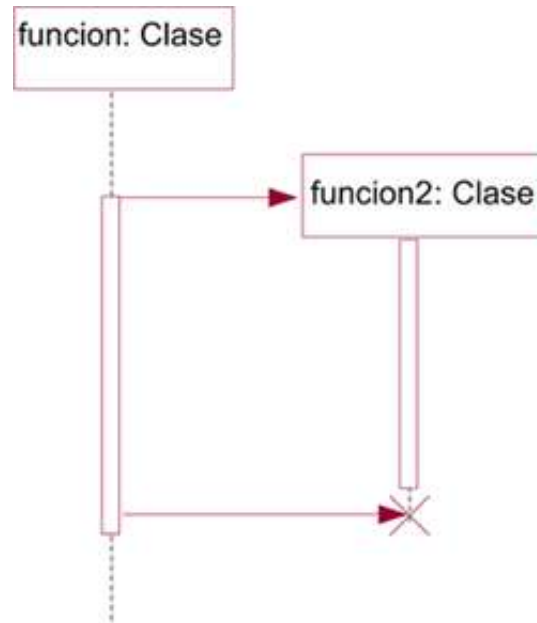


Figura 5.7 - Mensajes de creación y destrucción de un objeto

## 5. Descripción de la dinámica

Con los diferentes elementos anteriormente introducidos, podemos construir un diagrama de secuencia completo y describir la dinámica de un pequeño sistema o una subfunción de un sistema mayor.

Ejemplo

La figura 5.8 representa un escenario de compra de una yegua, ya estudiado en el capítulo Modelado de los requisitos. No existe alternativa posible, por tanto, se trata sin duda de un escenario. Veremos a continuación cómo introducir las alternativas y los bucles.

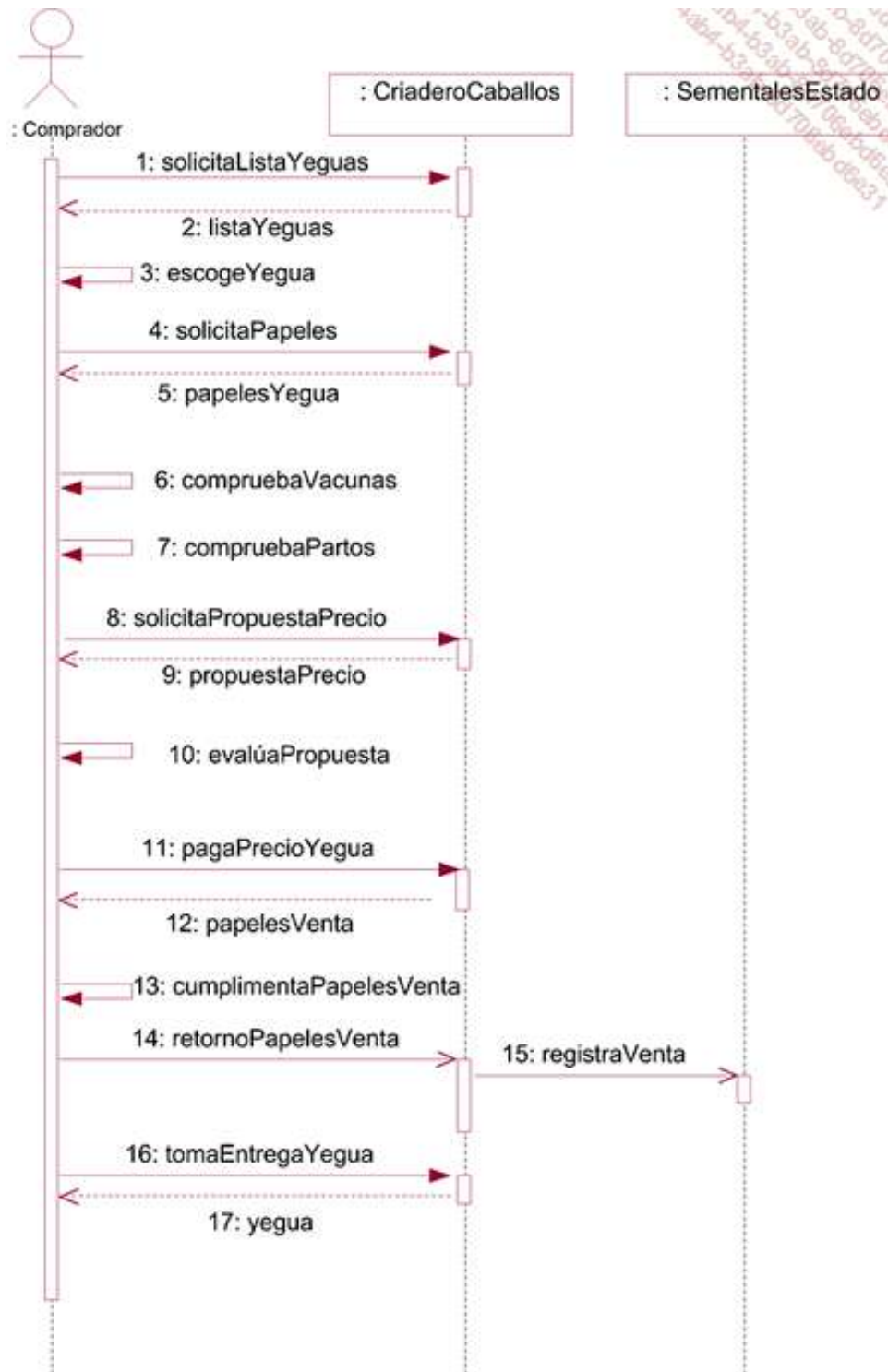


Figura 5.8 - Ejemplo de diagrama de secuencia: la representación de un escenario de compra de una yegua

## Marcos de interacción (UML 2)

Hasta ahora las construcciones introducidas para escribir diagramas de secuencia han sido las de UML 1. Los diagramas así contruidos describen escenarios. Por consiguiente, para representar todos los escenarios de la dinámica de un sistema es conveniente escribir otros tantos diagramas.

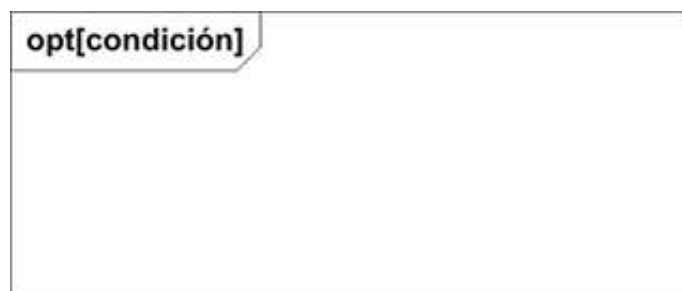
UML 2 generaliza los diagramas de secuencia para introducir los marcos de interacción. Esta importante extensión da soporte a las alternativas y a los bucles y confiere al diagrama de secuencia el estatus de verdadero modelo de interacciones.

### 1. La noción de marco de interacción

Un marco de interacción es una parte del diagrama de secuencia asociado a una etiqueta. Dicha etiqueta contiene un operador que determina la modalidad de ejecución. Las principales modalidades son la derivación condicional y el bucle.

### 2. La alternativa

La alternativa se obtiene utilizando el operador *opt* seguido de una condición de test. Si la condición se verifica, el contenido del marco se ejecuta.



*Figura 5.9 - Marco de interacción de alternativa*

Existe otro operador para la alternativa, denominado *alt*, que va seguido de varias condiciones de test y de la palabra clave *else*. El marco se divide entonces en varias partes cuyo contenido sólo se ejecuta si se cumple la condición asociada. El contenido de la última parte se asocia a la palabra clave *else* (si no) y sólo se ejecuta si no se verifica ninguna de las condiciones precedentes.

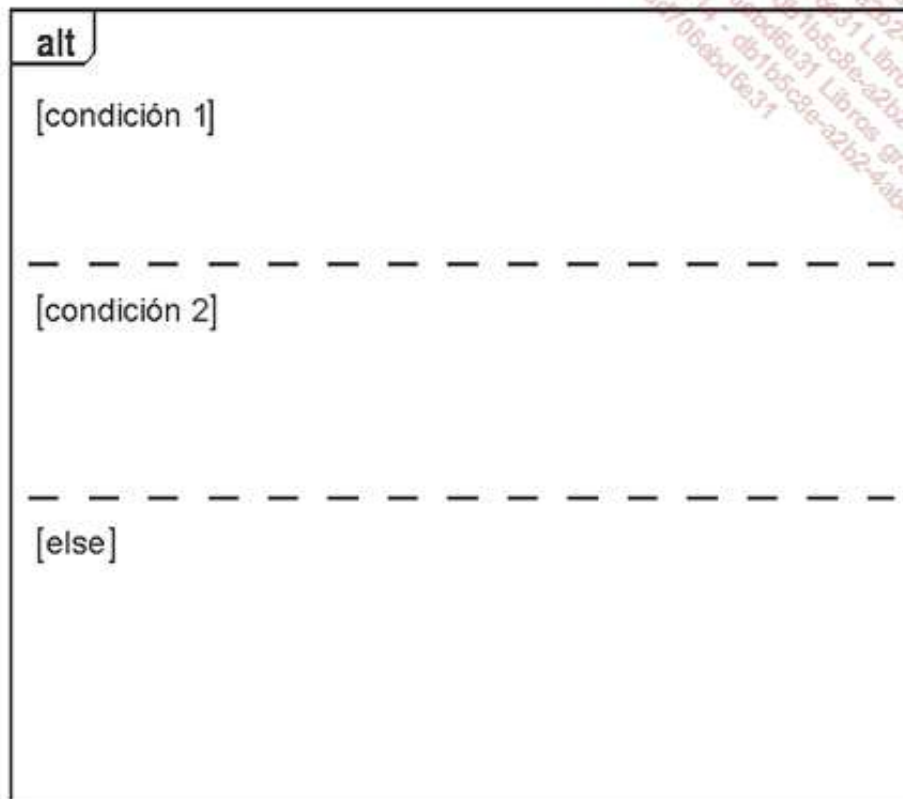


Figura 5.10 - Marco de interacción basado en el operador alt

### 3. El bucle

El bucle se efectúa mediante el operador *loop* seguido de los parámetros *min*, *max* y de una condición de test. El contenido del marco se ejecuta *min* veces. Después sólo lo hace mientras que se verifique la condición de test y el número máximo de ejecuciones del bucle no exceda de *max*. Los parámetros son opcionales.

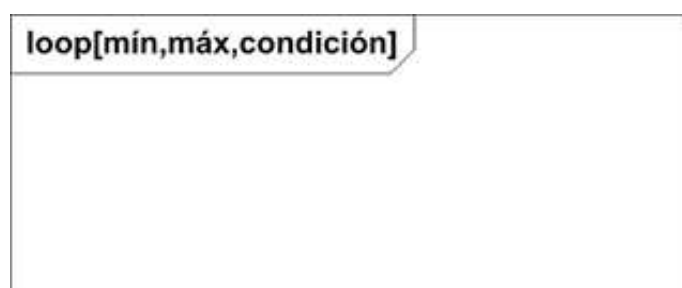


Figura 5.11 - Marco de interacción de bucle

#### Ejemplo

*Un jinete puede intentar superar un obstáculo un determinado número de veces, pero sin sobrepasar dos intentos fallidos.*

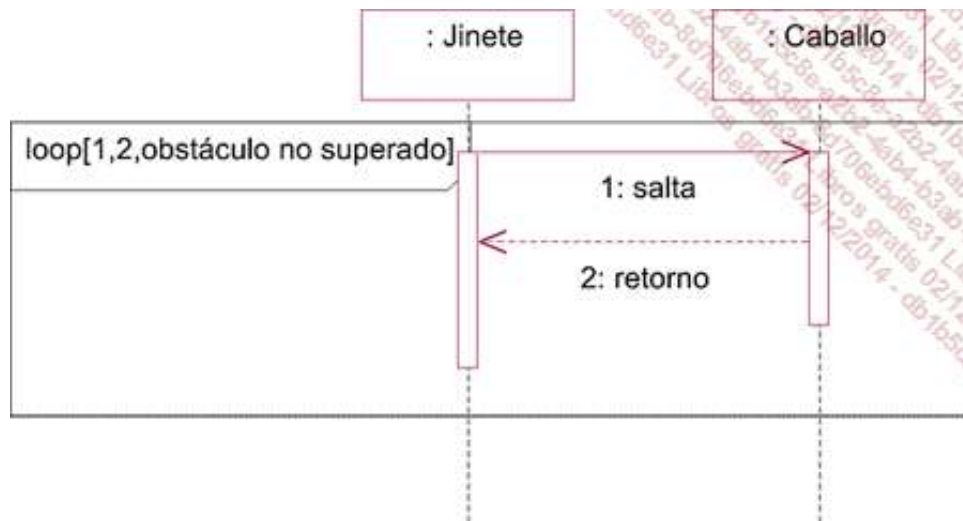


Figura 5.12 - Ejemplo de bucle

#### 4. Utilización de los marcos de interacción

Gracias a los diferentes elementos introducidos hasta el momento, ahora podemos describir la dinámica del sistema de manera más general.

##### Ejemplo

La figura 5.13 representa el caso de uso de compra de una yegua. A diferencia de lo que ocurre en la figura 5.8, se toman en cuenta las alternativas y bucles introducidos en el caso de uso. Concretamente, el diagrama de secuencia sólo se ejecuta hasta el final si el comprador confirma las vacunas y los partos. Se representa además el bucle de negociación del precio de venta.

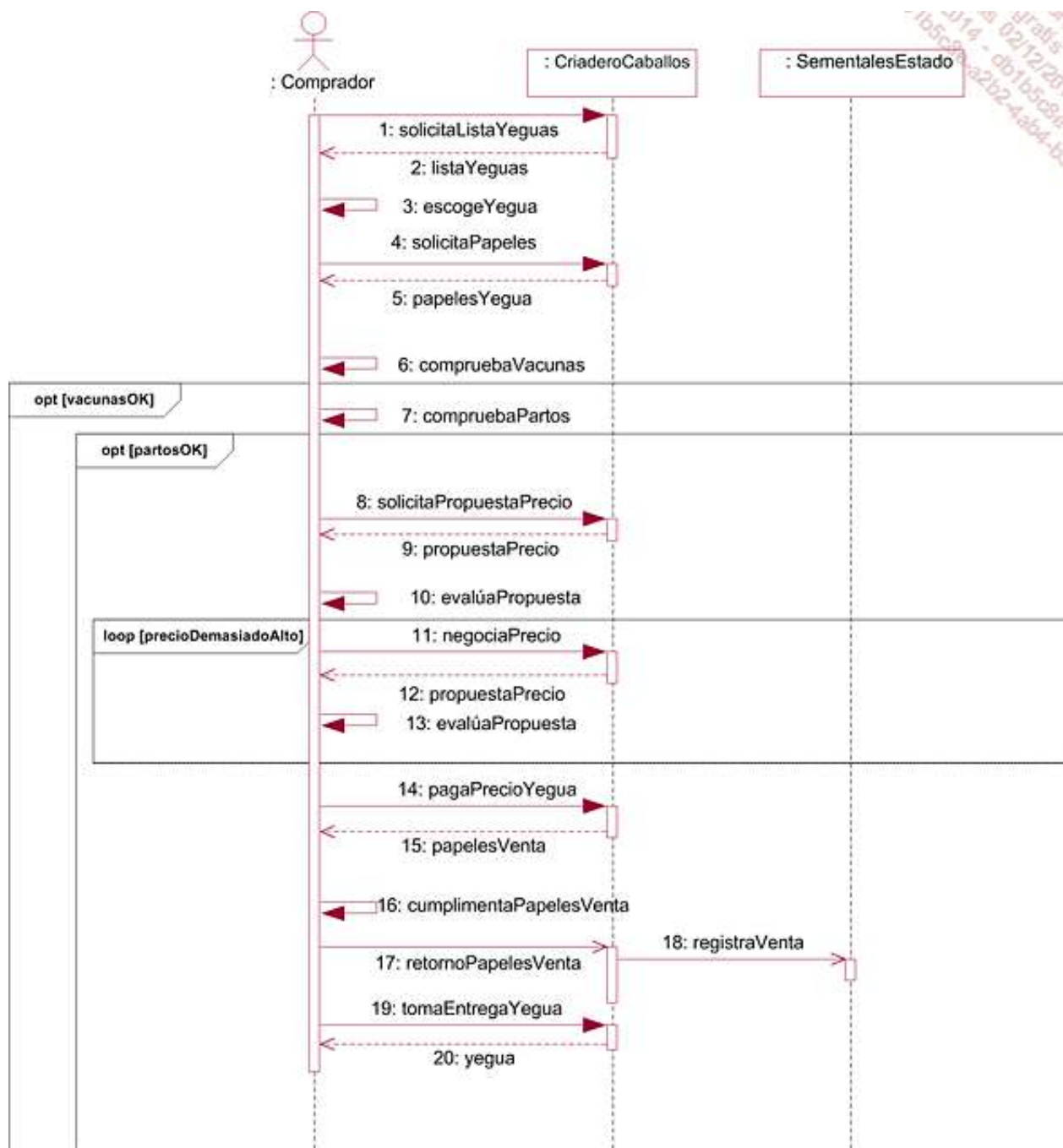


Figura 5.13 - Ejemplo de diagrama de secuencia: representación del caso de uso de compra de una yegua



## Diagrama de comunicación

El diagrama de comunicación es una alternativa al diagrama de secuencia. Éste se centra en una representación espacial de los objetos.

Los objetos intervienen en el diagrama de la misma forma que lo hacían en el diagrama de secuencia. Éste no está asociado a una línea de vida, sino unido gráficamente a los objetos con los que interactúa.

Los envíos de mensajes se sitúan a lo largo de los vínculos entre objetos. Los mensajes deben numerarse obligatoriamente, se puede emplear la numeración compuesta estudiada en el apartado relativo a los diagramas de secuencia.

La figura 5.14 ilustra el diagrama de comunicación correspondiente al diagrama de secuencia de la figura 5.3.

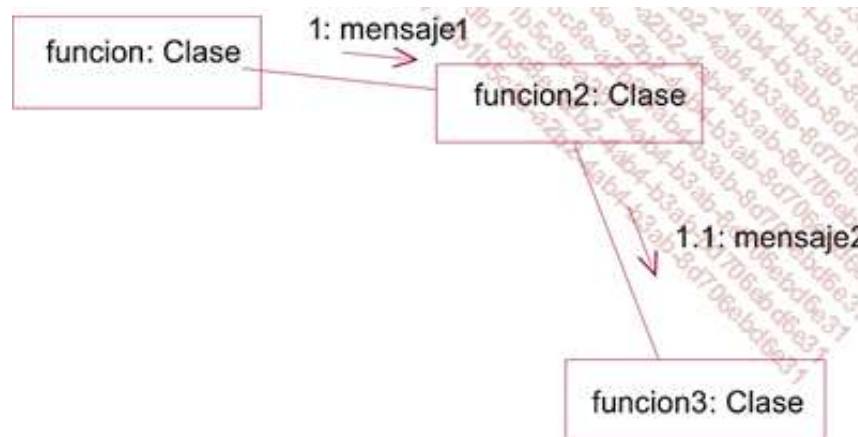


Figura 5.14 - Diagrama de comunicación

También es posible transmitir informaciones durante el envío de la misma manera que en los diagramas de secuencia.

### Ejemplo

En una manada hay una yegua dominante, que es la responsable de la educación de todos los potros. La yegua pasa el relevo a otra para que vigile a un potro en concreto.

En el ejemplo de la figura 5.15, la yegua dominante delega la vigilancia del potro Travieso a otra yegua, que da una orden al potro que éste se niega a obedecer y, a consecuencia de ello, recibe un castigo.

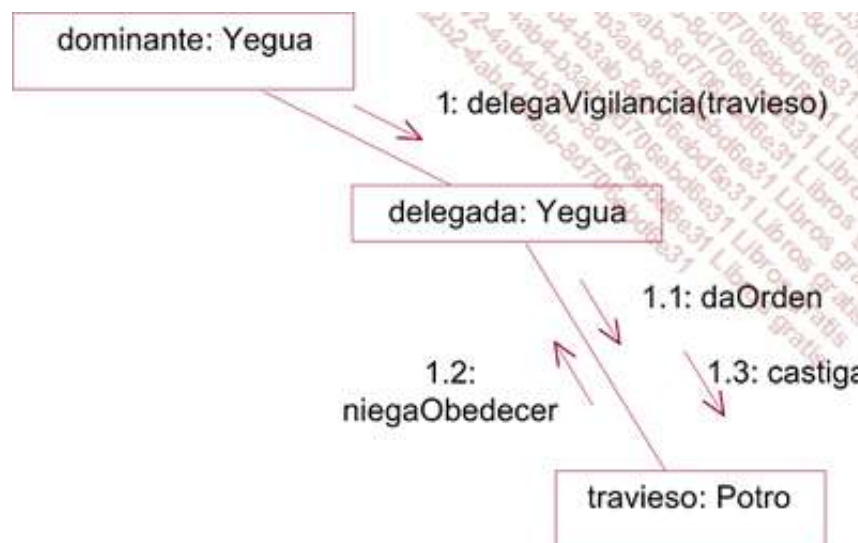


Figura 5.15 - Ejemplo de diagrama de comunicación: la vigilancia del potro Travieso

Dado que no existe un equivalente a los marcos de interacción, UML propone mecanismos de test y de bucle en el envío de los mensajes.

Los mecanismos de test se realizan mediante una condición especificada entre corchetes después del número del mensaje, lo que da la sintaxis siguiente:

Número[condición]: mensaje

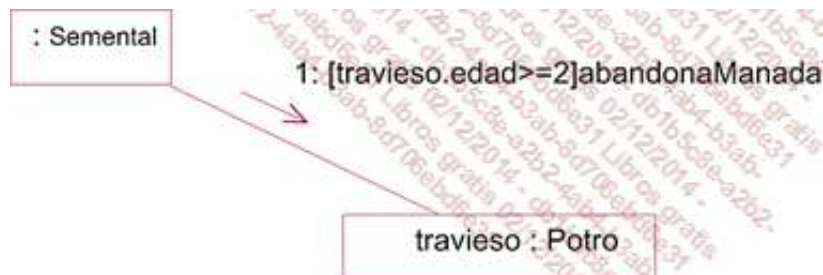
Existen dos mecanismos de bucle:

- El primero se basa en una condición. El bucle se ejecuta siempre que la condición sea verdadera. Su sintaxis es esta: Número\*[condición]: mensaje
- El segundo se basa en una variable de bucle cuyos límites inferiores y superiores están especificados. Su sintaxis es esta: Número\*[variableBucle=limiteInf..limiteSup]: mensaje

### Ejemplo

*Cuando un potro alcanza los dos años, el semental lo expulsa de la manada.*

*La figura 5.16 presenta el mensaje para el potro Travieso.*



*Figura 5.16 - Ejemplo de condición: el potro Travieso debe abandonar la manada a los dos años*



La edad es un atributo del potro Travieso. Se accede a ella a través de la sintaxis: `nombreObjeto.nombreAtributo`. Se recomienda esta sintaxis para acceder a los atributos y a los métodos de un objeto en una condición.

## Descubrir los objetos del sistema

Hemos visto que resulta sencillo representar un caso de uso mediante un diagrama de secuencia, sobre todo después de introducir los cuadros de interacción de UML 2.

En estos diagramas el sistema está representado en forma de objeto y las interacciones con el exterior se producen a lo largo de la línea de vida.

Para descubrir los objetos del sistema a partir de un caso de uso, la primera fase consiste en preguntarse a qué objetos del sistema están destinados los mensajes procedentes del exterior.

Determinar esto constituye una primera etapa de descomposición del sistema en objetos.

### Ejemplo

*En el ejemplo del caso de uso de compra de una yegua, las interacciones entre el comprador y el criadero de caballos pueden descomponerse en interacciones entre el comprador, por un lado, y el director, el contable o el mozo de las caballerizas, por otro (ver figura 5.17).*

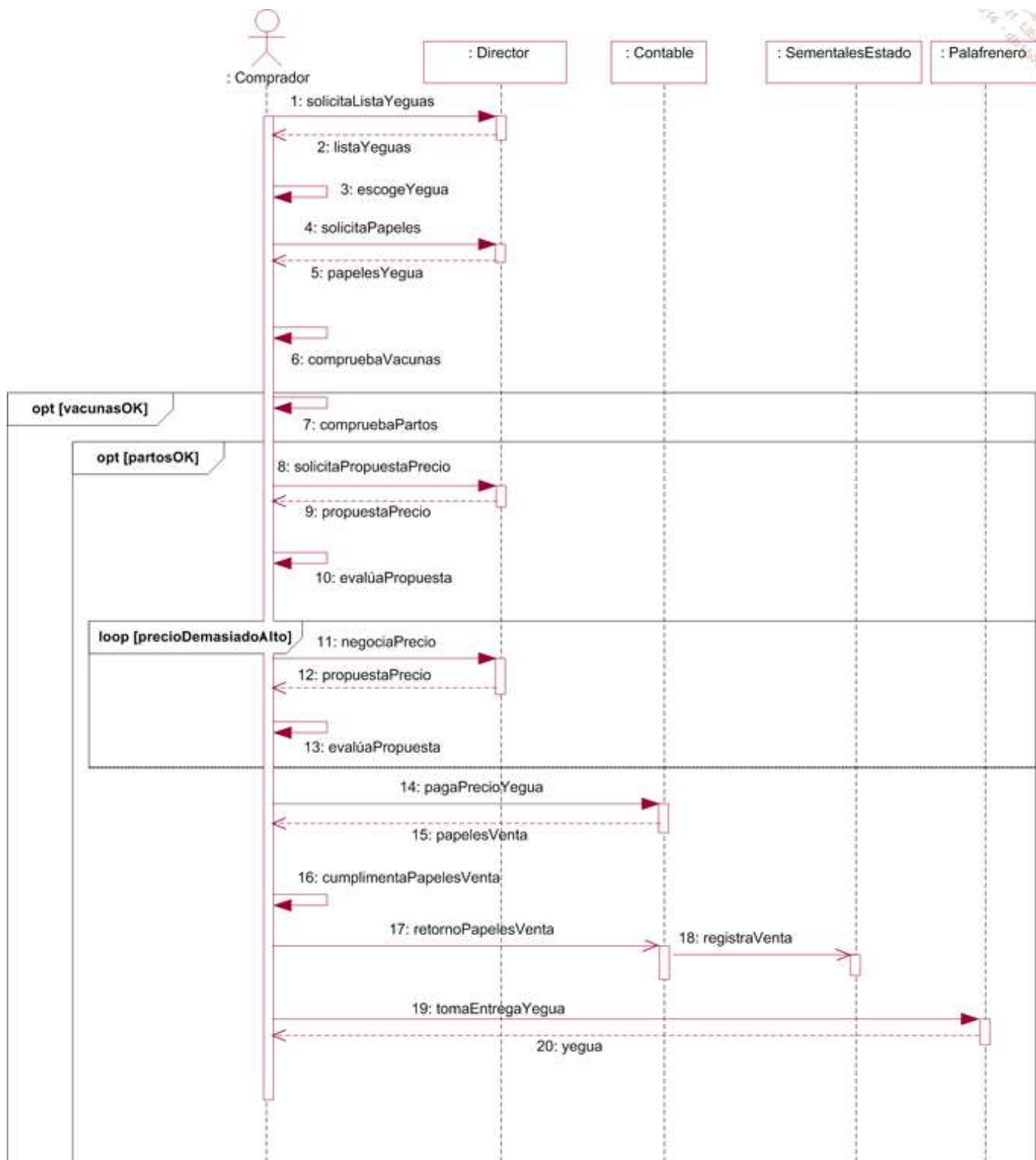


Figura 5.17 - Primer enriquecimiento del diagrama de secuencia de compra de una yegua

En una segunda fase se descomponen progresivamente los mensajes recibidos por los objetos; poco a poco se van descubriendo así los objetos del sistema. La descomposición de los mensajes obliga a utilizar nuevos objetos que responden a las funcionalidades requeridas.

La descomposición se acompaña a menudo de un enriquecimiento de la descripción de los mensajes en la transmisión de informaciones. El tratamiento de dichas informaciones implica con frecuencia la utilización de nuevos objetos.

### Ejemplo

Cuando el director recibe la solicitud de los papeles de la yegua, recurre a la base de datos del criadero para encontrarlos. En consecuencia, el diagrama de secuencia correspondiente se enriquece con la figura 5.18 (vista parcial correspondiente a la búsqueda de los papeles). A partir de entonces, la yegua elegida se transmite dentro del diagrama como parámetro. De esta forma, es posible encontrar sus papeles en la base de datos del criadero.

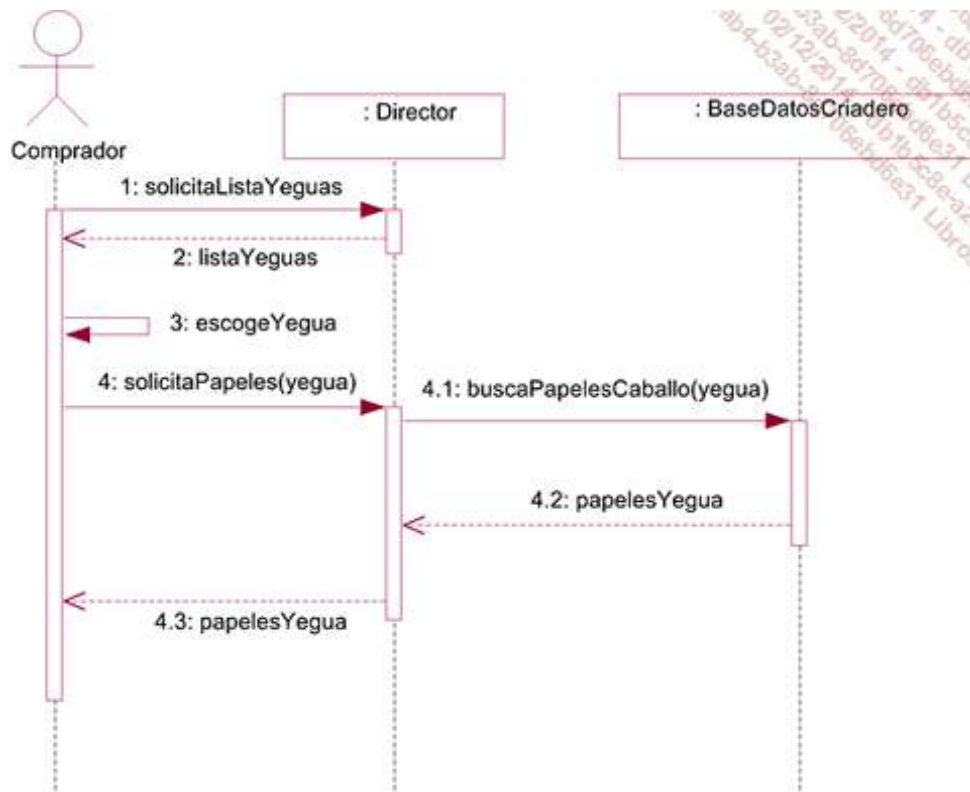


Figura 5.18 - Segundo enriquecimiento del diagrama de secuencia de compra de una yegua (vista parcial)

#### Ejemplo

Cuando el contable recibe el pago, redacta los papeles de venta antes de enviarlos al comprador. Esta descomposición introduce un nuevo objeto del sistema: los papeles.

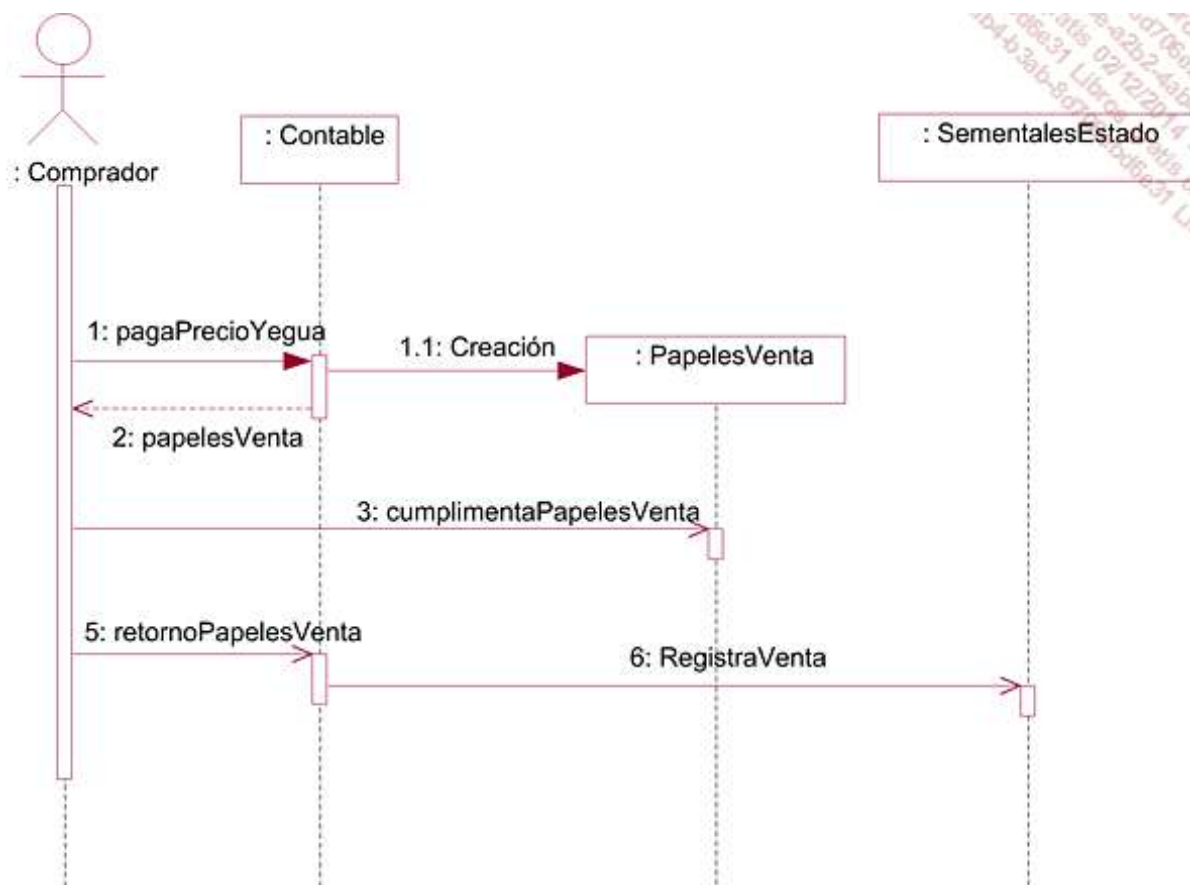


Figura 5.19 - Tercer enriquecimiento del diagrama de secuencia de compra de una yegua (vista parcial)

El proceso de descomposición de los mensajes es iterativo y debe continuarse hasta obtener objetos de tamaño

suficientemente preciso.



A menudo se emplea la palabra grano para designar el tamaño de un objeto. Existe todo un abanico de granos, desde el más fino hasta el más grueso. El sistema, tomado como un único objeto, es un objeto de grano grueso o de granulado significativo. Por el contrario, la base de datos de la granja de cría es un objeto de granulado más fino. Dentro de ella, las tablas o los datos serán objetos de granulado aún más fino. Esta noción es relativa. La persona encargada del modelado será quien determine el nivel de grano de los objetos que desea obtener.

## Conclusión

Los diagramas de secuencia y de comunicación son importantes para lo siguiente:

- Ilustrar y comprobar el comportamiento de un conjunto de objetos (sistema o subsistema).
- Ayudar a descubrir los objetos del sistema.
- Ayudar a descubrir los métodos de los objetos.

Desde la introducción de los marcos de interacción, los diagramas de secuencia pueden utilizarse para describir casos de uso.

Los diagramas de secuencia ponen en primer plano los aspectos temporales, mientras que los diagramas de comunicación muestran los vínculos entre clases.

# Ejercicios

## 1. El hipódromo

Construya el diagrama de secuencia de compra de una entrada para una carrera de caballos.

¿Cuáles son los objetos del sistema descubiertos así?

## 2. La central de compra de caballos

Construya el diagrama de secuencia de un pedido de productos en la página Web de la central de compra de caballos.

¿Cuáles son los objetos del sistema descubiertos así?



# Introducción

El objetivo del presente capítulo es dar a conocer las técnicas UML de modelado estático de objetos.

Dicho modelado se denomina estático porque no describe las interacciones o el ciclo de vida de los objetos, sino que los métodos se introducen desde un punto de vista estático, sin describir su encadenamiento.

Estudiaremos el diagrama de clases. Este diagrama contiene los atributos, métodos y asociaciones de los objetos. Como ya vimos en el capítulo Conceptos de la orientación a objetos, son las clases las que realizan la descripción.

Este diagrama es fundamental para el modelado de un sistema mediante objetos. De todos los diagramas UML, éste es el único obligatorio para ese tipo de modelado.

Veremos de qué manera el lenguaje OCL (*Object Constraint Language* o lenguaje de especificación orientado a objetos) puede extender el diagrama de clases para expresar con mayor riqueza las especificaciones.

El uso del OCL o del diagrama de objetos es opcional, depende de las especificaciones del proyecto de modelado.

## Conocer los objetos del sistema por descomposición

En el capítulo Modelado de la dinámica, estudiamos cómo descubrir los objetos desde un punto de vista dinámico. Primero presentamos los casos de uso en forma de diagrama de secuencias y luego enriquecimos dichos diagramas mediante el envío de mensajes para descubrir los objetos del sistema.

La descomposición de los mensajes hace aparecer los objetos del sistema, ya que conduce a mensajes más finos cuyo destinatario conviene buscar.

Otro posible planteamiento es la descomposición de la información contenida en un objeto. Con frecuencia, esta información es demasiado compleja para ser representada sólo por la estructura de un único objeto. A veces, también debe repartirse entre varios objetos.

### Ejemplo

En el ejemplo del capítulo Modelado de la dinámica, el director busca los papeles (la información) de la yegua que desea vender en la base de datos de la granja de cría. La base constituye un objeto de granulado grueso compuesto a su vez por otros objetos, como los papeles de los caballos, las informaciones económicas o contables y los documentos de compraventa de los caballos. Los papeles de una yegua están compuestos, entre otras cosas, por la cartilla de vacunación y los papeles de sus crías. Los papeles de las crías se comparten con otros objetos como, por ejemplo, los papeles del padre semental. Esta descomposición se guía por datos y no por aspectos dinámicos. La figura 6.1 ilustra la composición de PapelesYegua en el diagrama de clases.

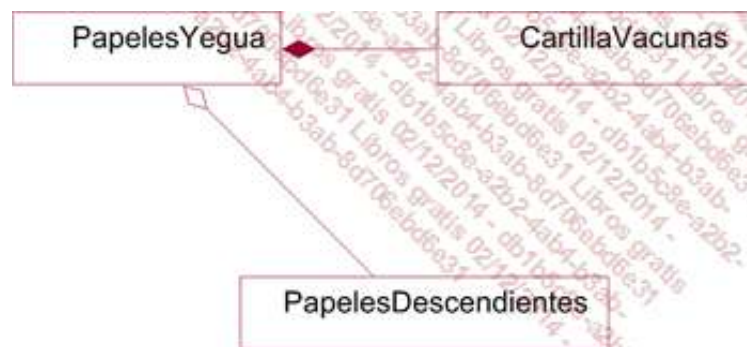


Figura 6.1 - Composición de PapelesYegua

Recordemos que el granulado de un objeto define su tamaño. El sistema, tomado como un objeto, es de grano grueso o granulado importante. Por el contrario, la cartilla de vacunación de un caballo es un objeto de grano mucho más fino que el sistema.

### Ejemplo

La descomposición de un caballo con el fin de mostrar sus diferentes órganos puede hacerse, bien mediante la descomposición de un diagrama de secuencia o bien mediante la descomposición guiada por datos.

La descomposición con el diagrama de secuencia consiste en analizar diferentes envíos de mensajes: dar miedo, correr, comer, dormir. Los mensajes harán aparecer progresivamente los diferentes órganos del caballo. En la figura 6.2, presentamos la descomposición del mensaje darMiedo. Los caballos dilatan las aletas de la nariz cuando están alerta, sorprendidos o tienen miedo. Aprietan la boca cuando están tensos o enfadados. Las coces, finalmente, constituyen un movimiento defensivo.

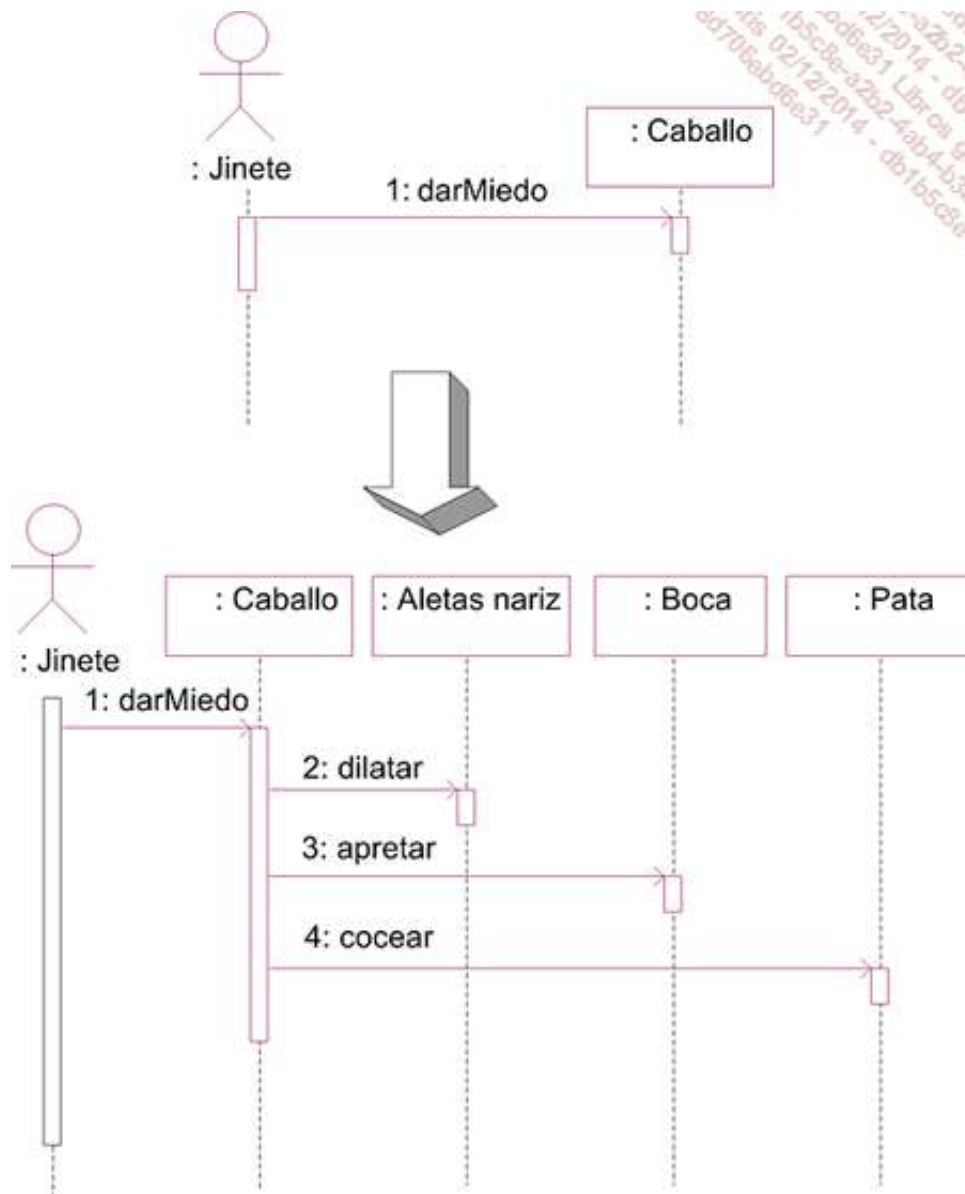


Figura 6.2 - Conocimiento de los objetos mediante enriquecimiento del diagrama de secuencia

La descomposición guiada por datos consiste en estudiar directamente los diferentes órganos de un caballo y tenerlos en cuenta en el diagrama de clases. En la figura 6.3 se representa un caballo compuesto por sus diferentes órganos.

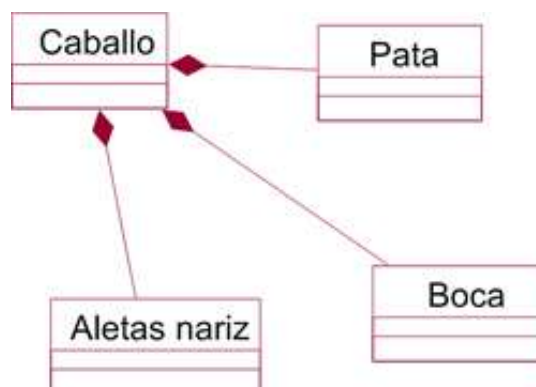


Figura 6.3 - Composición de Caballo

➤ La descomposición mediante diagramas de secuencia, al igual que la descomposición por datos, son formas naturales de conocer los objetos. El resultado es normal, ya que un objeto es la unión de una estructura y un comportamiento. Por último, conviene destacar que ambos planteamientos no son

incompatibles.

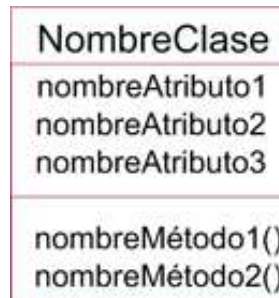


La descomposición guiada por datos es más eficaz cuando la persona encargada del modelado conoce bien el tema. La descomposición mediante objetos se realiza entonces de manera inmediata.

# Representación de clases

## 1. La forma simplificada de representación de clases

Los objetos del sistema se describen mediante clases. Presentamos una forma simplificada de representación de las clases en UML en la figura 6.4. La representación consta de tres partes.



*Figura 6.4 - Representación simplificada de una clase en UML*

La primera parte contiene el nombre de la clase.

- Recordemos que el nombre de las clases se escribe en singular y está siempre formado por un nombre común precedido o seguido de uno o varios adjetivos que lo califican. Dicho nombre es representativo del conjunto de objetos que forman la clase, representa la naturaleza de las instancias de una clase.

La segunda parte contiene los atributos. Éstos contienen a su vez la información de los objetos. El conjunto de atributos forma la estructura del objeto.

La tercera parte contiene los métodos, que corresponden a los servicios ofrecidos por el objeto y pueden modificar el valor de los atributos. El conjunto de métodos forma el comportamiento del objeto.

- El número de atributos y métodos varía de acuerdo con la clase. No obstante, se desaconseja emplear un número elevado de atributos y métodos ya que, en general, éste refleja una mala concepción de la clase.

### Ejemplo

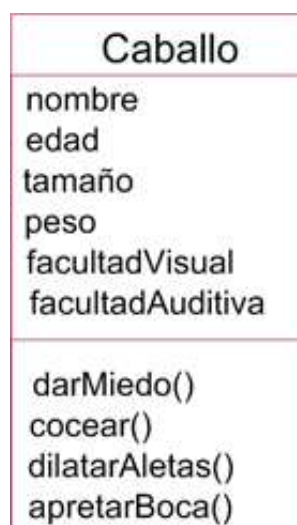


Figura 6.5 - La clase Caballo

Esta es la forma más simple de representación de las clases porque no hace aparecer las características de los atributos y de los métodos, a excepción de su nombre. Se utiliza a menudo en las primeras fases del modelado.

Antes de examinar las representaciones más completas, debemos abordar las nociones esenciales de encapsulación, tipo y firma de los métodos.

## 2. La encapsulación


Introducimos el concepto de encapsulación en el capítulo Conceptos de la orientación a objetos. Algunos atributos y métodos no se exponen en el exterior del objeto, sino que son encapsulados y reciben el nombre de atributos y métodos privados del objeto.

UML, al igual que la mayoría de lenguajes modernos orientados a objetos, introduce tres posibilidades de encapsulación:

- El atributo o el método privado: la propiedad no se expone fuera de la clase, ni tampoco dentro de sus subclases.
- El atributo o el método protegido: la propiedad sólo se expone en las instancias de la clase y de sus subclases.
- La encapsulación de empaquetado: la propiedad sólo se expone en las instancias de clases del mismo empaquetado. Abordaremos la noción de empaquetado en el capítulo Estructuración de los elementos de modelado.

La noción de propiedad privada se utiliza raramente, ya que conduce a establecer una diferencia entre las instancias de una clase y las de sus subclases. Esta diferencia está vinculada a aspectos bastante sutiles de la programación con objetos. La encapsulación de empaquetado, por su parte, procede del lenguaje Java y se reserva a la escritura de diagramas destinados a los desarrolladores.

Nosotros sugerimos el uso de la encapsulación protegida.

 Curiosamente, la mayoría de modelados emplean la encapsulación privada, pero ello se debe, en nuestra opinión, a que sus autores no han prestado atención a la diferencia existente entre encapsulación privada, protegida o de empaquetado. Eso fue justamente lo que nosotros hicimos, para simplificar las cosas, en el capítulo Conceptos de la orientación a objetos.

La encapsulación se representa con un signo más, un signo menos, una almohadilla o una tilde colocados antes del nombre del atributo. En el siguiente cuadro se detalla el significado de los signos.

público	+	elemento no encapsulado visible para todos
protegido	#	elemento encapsulado visible en las subclases de la clase
privado	-	elemento encapsulado visible sólo en la clase
empaquetado	~	elemento encapsulado visible sólo en las clases del mismo empaquetado

### Ejemplo

La figura 6.6 muestra la clase Caballo con las características de encapsulación.



Figura 6.6 - La clase Caballo con las características de encapsulación

### 3. La noción de tipo

En este caso, llamamos variable a cualquier atributo, parámetro o valor de retorno de un método. De manera general, llamamos variable a cualquier elemento que pueda tomar un valor.

El tipo es una especificación aplicada a una variable. Consiste en fijar el conjunto de valores posibles que la variable puede tomar. Dicho conjunto puede ser una clase, en cuyo caso la variable debe contener una referencia a una instancia de la misma y puede ser estándar, como el conjunto de enteros, cadenas de caracteres, booleanos o reales. En estos últimos casos, el valor de la variable debe ser respectivamente un entero, una cadena de caracteres, un valor booleano y un real.

Los tipos estándar se designan del siguiente modo:

- Integer para el tipo de los enteros.
- String para el tipo de las cadenas de caracteres.
- Boolean para el tipo de los booleanos.
- Real para el tipo de los reales.

#### Ejemplo

*1 ó 3 ó 10 son ejemplos de valores de entero. "Caballo" es un ejemplo de cadena de caracteres en la cual se ha optado por las comillas como separadores. False y True son los dos únicos valores posibles del tipo Boolean.*

*3.1415, donde el punto hace la función de separador de decimales, es un ejemplo bien conocido de número real.*

Veremos que, en general, el tipo de un atributo sólo recurre a una clase si ésta es una clase de una biblioteca externa al sistema modelado o una interfaz como las que estudiaremos al final del capítulo. No es aconsejable utilizar clases del sistema para dar un tipo a un atributo. En esos casos, como veremos a continuación, vale más recurrir a las asociaciones interobjetos.

Por el contrario, el tipo de un parámetro o del retorno de un método puede ser un tipo estándar o una clase, pertenezca o no al sistema.

El tipo de un atributo, de un parámetro y del valor de retorno de un método se especifica en la representación de clase.

#### Ejemplo

*La figura 6.7 muestra la clase Caballo, en la cual se ha establecido el tipo de todos los atributos. Los*

atributos de esta clase utilizan tipos estándar.

Caballo
+ nombre: String + edad: Integer + tamaño: Integer + peso: Integer # facultadVisual: Integer # facultadAuditiva: Integer
+ darMiedo() # cocear() # dilatarAletas() # apretarBoca()

Figura 6.7 - La clase Caballo con el tipo de los atributos

## 4. Firma de los métodos

Un método de una clase puede tomar parámetros y devolver un resultado. Los parámetros son valores transmitidos:

- En la ida, al enviar un mensaje que llama a un método.
- O en el retorno de llamada del método.

El resultado es un valor transmitido al objeto que efectúa la llamada cuando ésta se devuelve.

Como hemos visto anteriormente, tanto los parámetros como el resultado pueden tener tipos. El conjunto constituido por el nombre del método, los parámetros con su nombre y su tipo, así como el tipo de resultado se conoce como firma del método.

Una firma adopta la siguiente forma:

```
<nombreMétodo> (<dirección><nombreParámetro>: <tipo>, ...) :  
<tipoResultado>
```

Recordemos que el nombre de los parámetros puede ser nulo y que el tipo de resultado es opcional.

Es posible indicar la dirección en la cual el parámetro se transmite colocando delante del nombre del parámetro una palabra clave. Las tres palabras clave posibles son:

- in: el valor del parámetro sólo se transmite al efectuar la llamada.
- out: el valor del parámetro sólo se transmite en el retorno a la llamada del método.
- inout: el valor del parámetro se transmite en la llamada y en el retorno.

Si no se especifica ninguna palabra clave, el valor del parámetro sólo se transmite en la llamada.



Las direcciones out y inout no son compatibles con llamadas en modo asíncrono en las que aquél que efectúa la llamada no espera el retorno de llamada del método.

### Ejemplo

La figura 6.8 muestra la clase Caballo, cuyo método darMiedo se ha provisto de un parámetro (la intensidad con la que el jinete provoca miedo) y de un retorno (la intensidad del miedo que siente el caballo). Ambos valores son enteros. El resto de métodos no toma parámetros ni devuelve resultados.



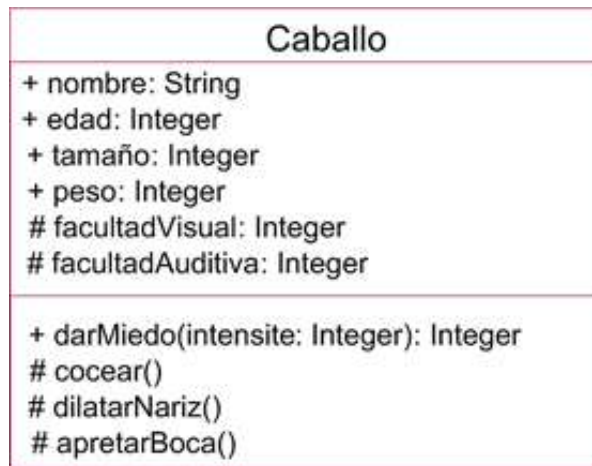


Figura 6.8 - La clase Caballo con la firma de los métodos

## 5. La forma completa de representación de las clases

La representación completa de las clases muestra los atributos con las características de encapsulación, el tipo y los métodos con la firma completa.

También es posible dar valores predeterminados a los atributos y a los parámetros de un método. El valor predeterminado de un atributo es el que se le atribuye al crear un nuevo objeto. El valor predeterminado de un parámetro se utiliza cuando aquél que llama a un método no proporciona explícitamente el valor del parámetro en el momento de la llamada.

La figura 6.9 ilustra la representación completa de una clase. Por supuesto, es posible escoger una representación intermedia entre la representación simplificada y la representación completa.

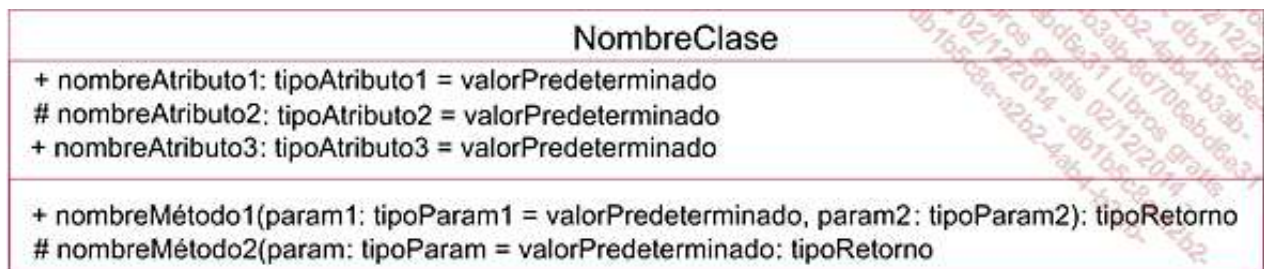


Figura 6.9 - Representación completa de una clase en UML

- La representación completa interesa sobre todo a los desarrolladores encargados de realizar el programa, previamente al modelado. Disponen así de una descripción de la clase muy similar a la que deben utilizar en su lenguaje de programación.

## 6. Los atributos y los métodos de clase

Las instancias de una clase contienen un valor específico para cada uno de sus atributos. Este valor, por tanto, no se comparte con el conjunto de instancias. En algunos casos, es preciso utilizar atributos cuyo valor es común a todos los objetos de una clase. Tales atributos comparten su valor al mismo título que su nombre, tipo y valor predeterminado y se conocen como *atributos de clase* porque están vinculados a la clase.

Los atributos de clase se representan mediante un nombre subrayado. Pueden estar encapsulados y poseer un tipo. Se recomienda vivamente asignarles un valor predeterminado.

### Ejemplo

Estudiamos el sistema de una carnicería exclusivamente caballar. La figura 6.10 introduce una nueva clase que describe una porción de carne de caballo. La venta de este producto está sujeta a una tasa de IVA cuyo montante es similar para todas las porciones. El atributo se destaca y protege, ya que sirve para calcular el precio con el IVA incluido y se expresa en porcentajes, de ahí que su tipo sea Integer. El valor predeterminado es 10, es decir 10%, la tasa de IVA que se aplica en España a los productos alimenticios.

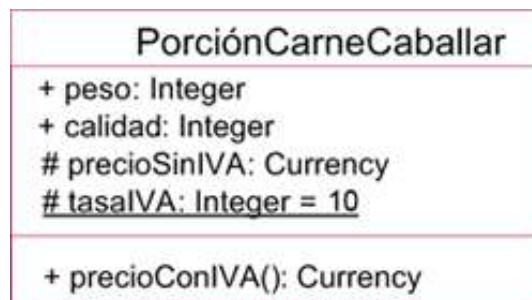


Figura 6.10 - Atributo de clase

- El tipo utilizado para el atributo precioSinIVA y el resultado del método precioConIVA es Currency, que indica que los valores son cantidades monetarias.

Dentro de una clase también pueden existir uno o varios métodos de clase vinculados a la misma. Para llamar a un método de clase, hay que enviar un mensaje a la propia clase y no a una de sus instancias. Estos métodos sólo manipulan los atributos de clase.

### Ejemplo

La figura 6.11 agrega un método de clase a la clase PorciónCarneCaballar que sirve para fijar la tasa de IVA. En efecto, la ley es la que fija dicha tasa y la ley puede modificarse.

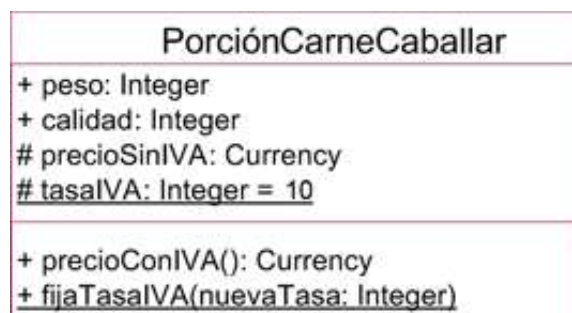


Figura 6.11 - Método de clase

- En muchas herramientas UML, no se utilizan los términos "atributo y método de clase". Estas herramientas dan preferencia a la denominación "atributos o métodos estáticos", denominación que se emplea en lenguajes de programación modernos como C++ o Java.
- Los atributos o los métodos de clase no se heredan. La herencia se aplica a la descripción de las instancias, calculada a través de la unión de la estructura y del comportamiento de la clase y de sus superclases. Una subclase puede acceder a un atributo o a un método de clase de una de sus superclases, pero no hereda de ellas. De haber herencia, tendríamos tantos ejemplares de atributos o métodos como subclases poseyera la clase que los introdujo.
- Recordemos también que una subclase puede acceder a un atributo o a un método de clase de una

de sus superclases, a condición de que no se haya utilizado la encapsulación privada.

## 7. Los atributos calculados

UML introduce la noción de atributo calculado, cuyo valor viene determinado por una función basada en el valor de otros atributos. Estos atributos poseen un nombre precedido del signo / y van seguidos de una especificación que determina el modo de calcular su valor.

### Ejemplo

Retomamos el ejemplo de la figura 6.10. El método `precioConIVA` es reemplazado por un atributo calculado `/precioConIVA`, como ilustra la figura 6.12.

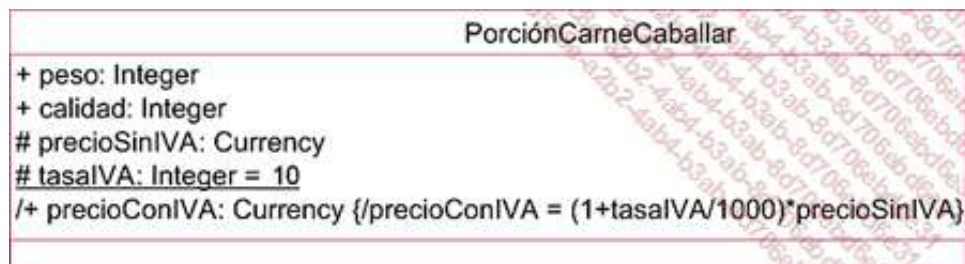


Figura 6.12 - Atributo calculado



Las especificaciones expresadas en diagramas UML se escriben entre llaves.

# Las asociaciones entre objetos

## 1. Los vínculos entre objetos

En el mundo real, muchos objetos están vinculados entre sí. Dichos vínculos corresponden a una asociación existente entre los objetos.

### Ejemplos

- El vínculo existente entre el potro Travieso y su padre.
- El vínculo existente entre el potro Travieso y su madre.
- El vínculo existente entre la yegua Jorgelina y el criadero de caballos al que pertenece.
- El vínculo existente entre el criadero de caballos y su propietario.

En UML, estos vínculos se describen mediante asociaciones, de igual modo que los objetos se describen mediante clases. Un vínculo es un elemento de una asociación. Por consiguiente, una asociación vincula a las clases. Los elementos de la asociación vinculan entre sí las instancias de las clases.

Las asociaciones tienen un nombre y, como ocurre con las clases, éste es un reflejo de los elementos de la asociación.

### Ejemplos

- La asociación padre entre la clase Descendiente y la clase Semental.
- La asociación madre entre la clase Descendiente y la clase Yegua.
- La asociación pertenece entre la clase Caballo y la clase CriaderoCaballos.
- La asociación propietario entre la clase CriaderoCaballos y la clase Persona.

Las asociaciones que hemos estudiado hasta el momento a título de ejemplo establecen un vínculo entre dos clases. Estas asociaciones reciben el nombre de asociaciones binarias. Las asociaciones que vinculan tres clases se denominan asociaciones ternarias y aquellas que vinculan  $n$  clases reciben el nombre de asociaciones  $n$ -arias. En la práctica, la gran mayoría de asociaciones son binarias y las asociaciones cuaternarias y superiores prácticamente no se usan.

## 2. Representación de las asociaciones entre clases

La representación gráfica de una asociación binaria consiste en una línea continua que une las dos clases cuyas instancias se vinculan. Las clases se sitúan en los extremos de la asociación.

La figura 6.13 muestra la representación de una asociación binaria. El nombre de la asociación se indica encima de la línea.

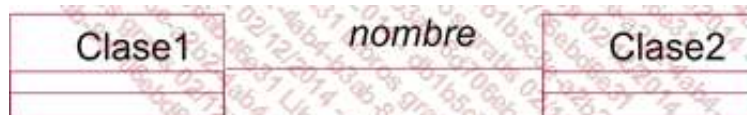


Figura 6.13 - Asociación binaria entre clases

Para señalar el sentido de lectura del nombre de la asociación con respecto al nombre de las clases, éste puede precederse del signo  $<$  o seguirse del signo  $>$ . Si la asociación se sitúa en un eje vertical, el nombre puede ir precedido de  $\wedge$  o de  $\vee$ .

Los extremos de una asociación también pueden recibir un nombre. Dicho nombre es representativo de la función que desempeñan en la asociación las instancias de la clase correspondiente. Una función tiene la misma naturaleza que un atributo cuyo tipo sería la clase situada en el otro extremo. Por consiguiente, puede ser pública o estar encapsulada de manera privada, protegida o para empaquetado. Cuando se especifican las funciones, muchas veces no es preciso indicar el nombre de la asociación, ya que éste suele ser el mismo que el de una de las funciones.

La figura 6.14 ilustra la representación de una asociación binaria mostrando las funciones.

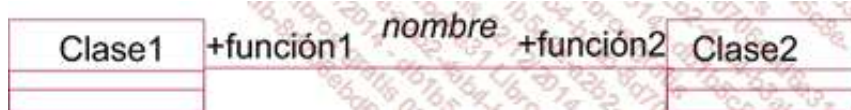


Figura 6.14 - Funciones de una asociación binaria

### Ejemplo

La figura 6.15 muestra la representación gráfica de las asociaciones introducidas en el ejemplo precedente.

En estas asociaciones se ha indicado el nombre de la asociación o bien sus funciones.

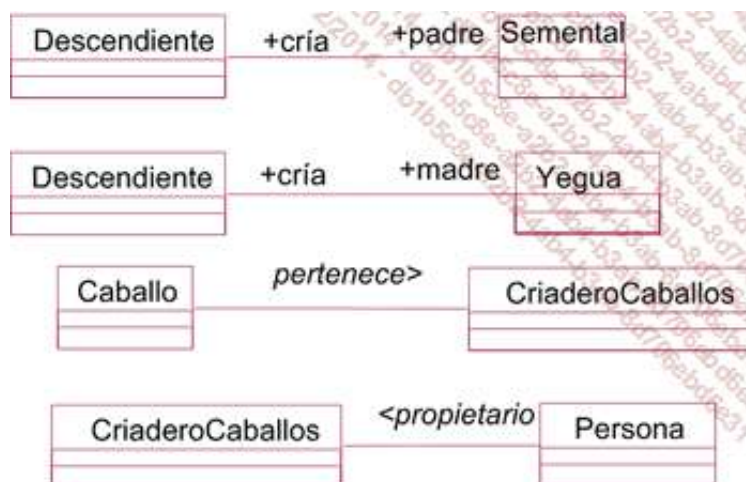


Figura 6.15 - Ejemplos de asociaciones binarias

La representación gráfica de una asociación ternaria y superiores consiste en un rombo que une las diferentes clases. La figura 6.16 ilustra la representación de una asociación ternaria.

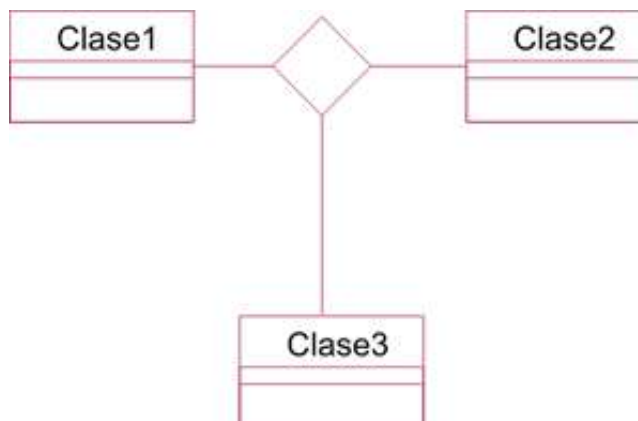


Figura 6.16 - Asociación ternaria entre clases

### Ejemplo

La asociación familia que vincula las clases Semental, Yegua y Descendiente se presenta en la figura 6.17.

Cada uno de los elementos constituye un triplete (padre, madre, potro).



Figura 6.17 - Ejemplos de asociación ternaria

### 3. La cardinalidad de las asociaciones

La cardinalidad situada en un extremo de una asociación indica a cuántas instancias de la clase situada en ese mismo extremo está vinculada una instancia de la clase situada en el extremo opuesto.

En uno de los extremos de la asociación, es posible especificar la cardinalidad mínima y la máxima con el fin de indicar el intervalo de valores al que deberá pertenecer siempre la cardinalidad.

La sintaxis de especificación de las cardinalidades mínimas y máximas se describe en el siguiente cuadro.

Especificación	Cardinalidades
0..1	cero o una vez
1	únicamente una vez
*	de cero a varias veces
1..*	de una a varias veces
M..N	entre M y N veces
N	N veces

➤ De no existir una especificación explícita de las cardinalidades mínimas y máximas, éstas valen 1.

#### Ejemplo

En la figura 6.18, retomamos los ejemplos y les agregamos las cardinalidades mínimas y máximas de cada asociación. Un criadero de caballos puede tener varios propietarios y una persona puede ser propietario de varios criaderos.

Para ilustrar la lectura, indicamos cómo debe leerse la primera asociación: un descendiente posee un solo padre. Un semental puede tener de cero a varios potros o crías.





Figura 6.18 - Ejemplos de asociaciones descritas con sus cardinalidades

## 4. Navegación

Por defecto, las asociaciones tienen una navegación bidireccional, es decir, es posible determinar los vínculos de la asociación desde una instancia de cada clase de origen. Las navegaciones bidireccionales resultan más complejas de realizar para los desarrolladores y, en la medida de lo posible, conviene evitarlas.

Para especificar el único sentido útil de navegación durante las fases de modelado cercanas al paso al desarrollo se dibuja la asociación en forma de flecha.

### Ejemplo

En el contexto concreto de un criadero, resulta útil conocer los caballos que posee dicho criadero, pero lo contrario no es necesario.

La figura 6.19 muestra la asociación resultante con el sentido de navegación adecuado.

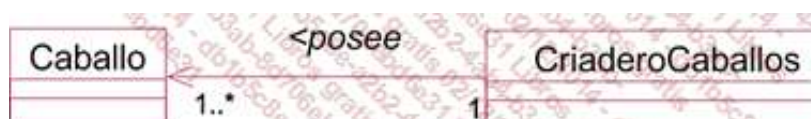


Figura 6.19 - Ejemplo de navegación

## 5. Asociar una clase a sí misma


Cuando encontramos una misma clase en los dos extremos de una asociación, hablamos de asociaciones reflexivas que unen entre sí las instancias de una misma clase.

En estos casos, resulta preferible asignar un nombre a la función desempeñada por la clase en cada extremo de la asociación.

Como veremos en los ejemplos, las asociaciones reflexivas sirven principalmente para describir dentro del conjunto de instancias de una clase:

- Grupos de instancias.
- Una jerarquía dentro de las instancias.

Para los lectores expertos diremos que, en el primer caso, se trata de una asociación que representa una

 relación de equivalencia, y en el segundo, una asociación que representa una relación de orden.

### Ejemplo

Para poder superar las pruebas de selección de un concurso hípico internacional, es preciso que los caballos hayan ganado otros concursos previos. Podemos crear, por consiguiente, una asociación entre el concurso internacional y los concursos celebrados previamente a él. La figura 6.20 muestra dicha asociación, que crea una jerarquía dentro de los concursos.

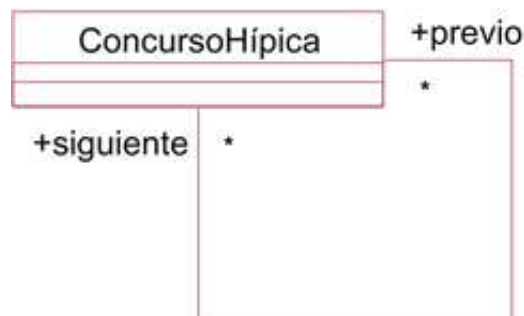


Figura 6.20 - Asociación reflexiva entre concursos hípicos

### Ejemplo

La figura 6.21 muestra la asociación "ascendente/descendiente directo" entre los caballos. Esta asociación crea una jerarquía dentro de los caballos.

La cardinalidad para los ascendentes es 2, ya que cualquier caballo tiene un padre y una madre.

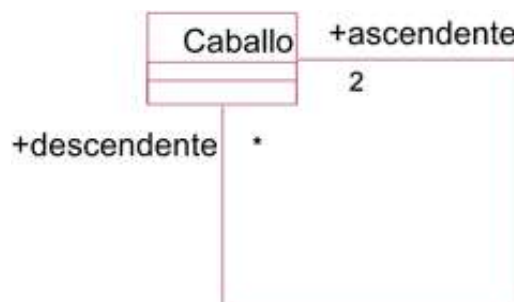


Figura 6.21 - Asociación "ascendente/descendiente directo" entre caballos

### Ejemplo

La figura 6.22 muestra la asociación entre los caballos que se encuentran en el mismo criadero. Esta asociación crea grupos dentro del conjunto de instancias de la clase **Caballo**, ya que cada grupo corresponde a un criadero.

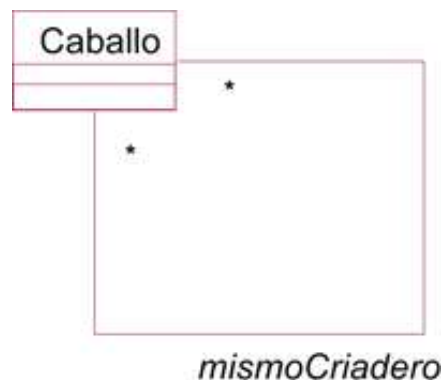


Figura 6.22 - Asociación reflexiva que vincula los caballos de un mismo criadero



## 6. Las clases-asociaciones

Los vínculos entre las instancias de las clases pueden llevar informaciones. Éstas son específicas a cada vínculo.

En esos casos, la asociación que describe los vínculos recibe el estatus de clase y sus instancias son elementos de la asociación.

Al igual que el resto, estas clases pueden estar dotadas de atributos y operaciones y estar vinculadas a otras clases a través de asociaciones.

La figura 6.23 representa gráficamente una clase-asociación que se une a la asociación mediante una línea discontinua.



Figura 6.23 - Representación gráfica de una clase-asociación

### Ejemplo

Cuando un cliente compra productos para caballos (productos de mantenimiento, etc.) conviene especificar la cantidad de productos adquiridos mediante una clase-asociación, aquí denominada clase Adquisición (ver figura 6.24).

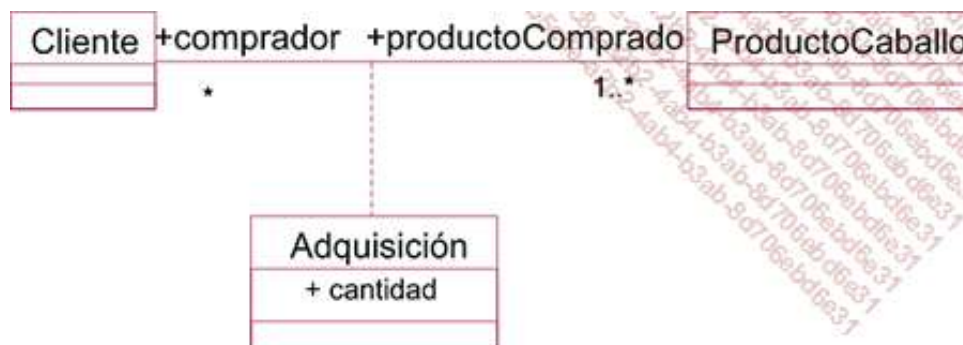


Figura 6.24 - Clase-asociación Adquisición

## 7. La calificación de las asociaciones

En caso de cardinalidad máxima no finita en un extremo de la asociación, si las instancias situadas en dicho extremo son calificables, la calificación puede usarse para pasar de la cardinalidad máxima no finita a una cardinalidad máxima finita.

La calificación de una instancia es un valor o un conjunto de valores que permiten encontrar dicha instancia. Muchas veces, las calificaciones son índices para, por ejemplo, encontrar un elemento en una tabla, o llaves para, por ejemplo, localizar una línea en una base de datos relacional.

La calificación se inserta en el extremo opuesto en forma de uno o varios atributos (ver figura 6.25), donde las instancias de la clase 2 se califican en el ámbito de la clase 1.



Figura 6.25 - Representación gráfica de una calificación

### Ejemplo

Una tabla está formada por elementos. La figura 6.26 describe dos posibilidades de modelado en UML. La primera no recurre a la calificación, mientras que en la segunda el calificador índice permite encontrar un único elemento de la tabla. Por consiguiente, la cardinalidad máxima pasa a uno.

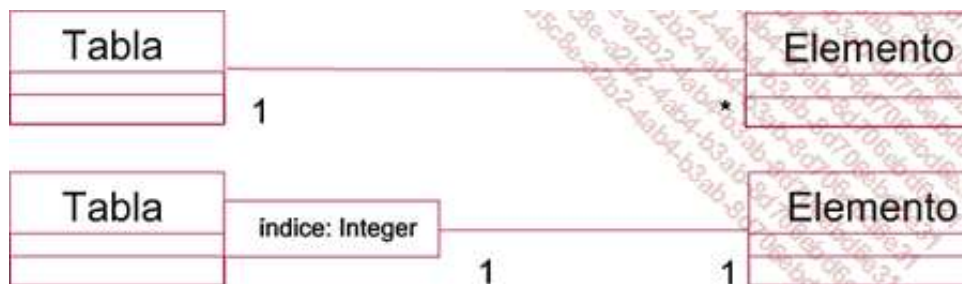


Figura 6.26 - Calificación de los elementos de una tabla

### Ejemplo

En las carreras de caballos, todos los caballos poseen un número. La figura 6.27 muestra a los participantes en una carrera calificándolos por su número.

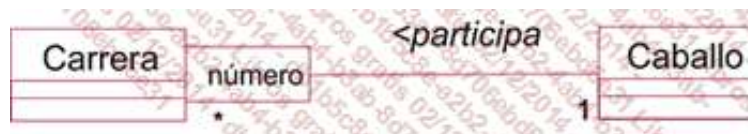


Figura 6.27 - Calificación de los participantes en una carrera hípica

## 8. La expresión de las especificaciones en las asociaciones

UML ofrece la posibilidad de expresar las especificaciones gracias a ciertas construcciones del modelado *objeto* que ya hemos estudiado: las cardinalidades, el tipo de un atributo, etc.

No obstante, estos tipos de especificaciones pueden resultar insuficientes. UML propone expresar otras especificaciones en lenguaje natural.

También existe el OCL, lenguaje de especificación *objeto* en forma de condiciones lógicas. El lenguaje OCL forma parte del conjunto de notaciones UML.

Ya se escriban en lenguaje natural o bien en OCL, ambos tipos de especificaciones se representan en las notas incluidas dentro del diagrama de clases.

Las especificaciones escritas en OCL se expresan sobre el valor de los atributos y de las funciones (extremos de las asociaciones). Las especificaciones deben tener un valor lógico (verdadero o falso).

Para construir una especificación se utilizan todos los operadores aplicables al valor de los atributos en función de su tipo (comparación de enteros, suma de enteros, comparación de cadenas, etc.). En las condiciones pueden emplearse los métodos de los objetos. Para los valores de conjunto (colecciones de objetos), OCL propone un juego de operadores: union, intersection, "-" (diferencia).

Para las colecciones de objetos, OCL propone sobre todo los operadores siguientes: collect, includes, includesAll, asSet, exists, forAll.

Para designar un atributo o un método en una expresión OCL, hay que elaborar una expresión de ruta que empiece por self. A continuación, se designa directamente el atributo o el método por su nombre o se recorre una asociación utilizando el nombre de la función. Es conveniente entonces designar un atributo o un método de la clase situada en el otro extremo de la asociación, o bien designar de nuevo una función para recorrer otra asociación hasta elegir un atributo o un método.

La sintaxis de una expresión de ruta es la siguiente:

```
self.atributo  
  
o  
  
self.método  
  
o  
  
self.función.función. ... .función.atributo  
  
o  
  
self.función.función. ... .función.método
```

Si una especificación OCL no está incluida directamente en el diagrama de clases, entonces hay que precederla de su contexto, que se inscribe así:

**Contexto** Clase inv Especificación:

➤ En el presente manual, ofrecemos únicamente una breve introducción al lenguaje OCL. Los lectores interesados en obtener más información pueden consultar la obra "The Object Constraint Language: Getting Your Models ready for MDA, Second Edition" de Jos Warmer y Anneke Kleppe, Addison-Wesley, 2003.

### Ejemplo

*La comida para caballos contiene los elementos siguientes:*

- paja;
- minerales;
- heno o gránulos.

La figura 6.28 ilustra la comida y sus diferentes componentes. Que contenga heno o bien gránulos es una especificación enunciada por el operador o. Este operador expresa la exclusión entre ambas asociaciones.

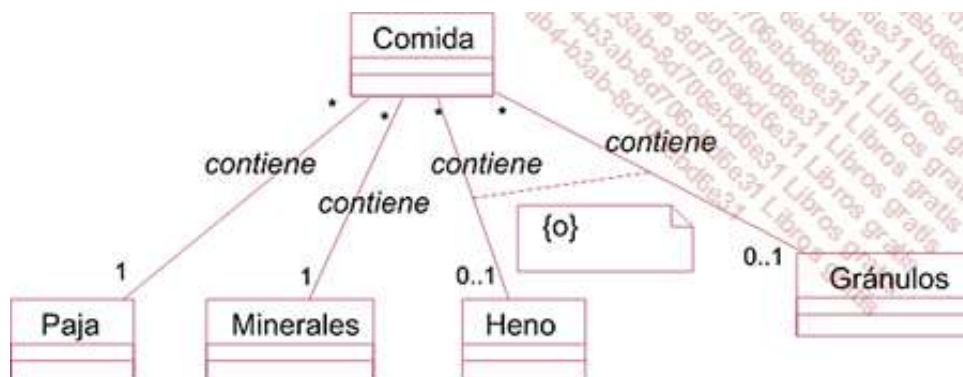


Figura 6.28 - Expresión de la exclusión entre dos asociaciones

### Ejemplo

Un hipódromo hace correr a varios caballos. Los jockeys que ha contratado deben participar en las carreras montando una parte de los caballos. Esto puede reformularse así: el conjunto de jockeys empleados por el hipódromo debe estar incluido en el conjunto de jockeys que montan los caballos participantes en las carreras del hipódromo.

En OCL, esta especificación se escribe así:

```
Contexto Hipódromo inv jockeysEmpleados:
self.participaCarrera->collect(c | c.monta)->includesAll
(self.empleado)
```

La especificación se apoya en la utilización de las expresiones de camino en OCL. La figura 6.29 muestra el diagrama de clases donde la especificación escrita en OCL se incluye directamente.

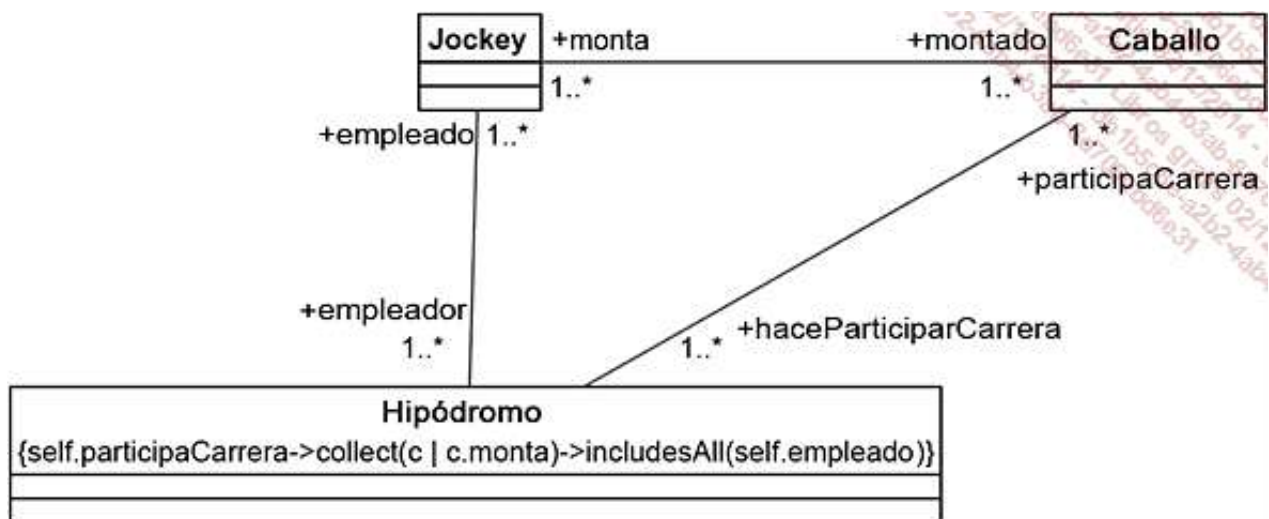


Figura 6.29 - Especificación con expresiones de camino en OCL

## 9. Los objetos compuestos

En el capítulo Conceptos de la orientación a objetos, vimos que un objeto puede estar compuesto por otros objetos. En tales casos, nos encontramos ante una asociación entre objetos de un caso particular llamada *asociación de composición*. Ésta asocia un objeto complejo con los objetos que lo constituyen, es decir, sus componentes.

Existen dos formas de composición, fuerte o débil, que vamos a examinar a continuación.

### a. La composición fuerte o composición

La composición fuerte es una forma de composición en la que los componentes constituyen una parte del objeto compuesto. De esta forma, los componentes no pueden ser compartidos por varios objetos compuestos. Por tanto, la cardinalidad máxima, a nivel del objeto compuesto, es obligatoriamente uno.

La supresión del objeto compuesto comporta la supresión de los componentes.

La figura 6.30 muestra la representación gráfica de la asociación de composición fuerte. A nivel del objeto compuesto, la cardinalidad mínima se ha establecido en cero, pero también podría ser uno.



Figura 6.30 - Asociación de composición fuerte

➤ En adelante, la asociación de composición fuerte será denominada simplemente composición.

### Ejemplo

Un caballo está compuesto, entre otras cosas, por un cerebro. El cerebro no se comparte. La muerte del caballo comporta la muerte del cerebro. Se trata, por tanto, de una asociación de composición, ilustrada en la figura 6.31.



Figura 6.31 - Asociación de composición entre un caballo y su cerebro

### Ejemplo

Una carrera hípica está constituida por premios. Los premios no se comparten con otras carreras (un premio es específico de una carrera). Si la carrera no se organiza, los premios no se atribuyen y desaparecen. Se trata de una relación de composición, ilustrada en la figura 6.32.



Figura 6.32 - Asociación de composición entre una carrera hípica y los premios que forman parte de ella

## **b. La composición débil o agregación**

La composición débil, llamada habitualmente agregación, impone muchas menos especificaciones a los componentes que la composición fuerte estudiada hasta ahora. En el caso de la agregación, los componentes pueden ser compartidos por varios compuestos (de la misma asociación de agregación o de varias asociaciones de agregación distintas) y la destrucción del compuesto no conduce a la destrucción de los componentes.

La agregación se da con mayor frecuencia que la composición. En las primeras fases de modelado, es posible utilizar sólo la agregación y determinar más adelante qué asociaciones de agregación son asociaciones de composición.

➤ Determinar sobre un modelo que una asociación de agregación es una asociación de composición, representa agregar especificaciones, asignar un tipo o precisar las cardinalidades. Hemos estudiado también las especificaciones explícitas en lenguaje natural o en OCL. Agregar especificaciones significa agregar sentido, agregar semántica a un modelo, es decir, enriquecerlo. Es por tanto normal que ese proceso de enriquecimiento requiera de fases sucesivas.

### Ejemplo

Un caballo enjaezado está compuesto, entre otras cosas, por una silla. Una silla está compuesta por una cincha, estribos y una manta de montura. Esta composición es una muestra de agregación (ver figura 6.33). En efecto, la pérdida del caballo no acarrea la pérdida de los objetos y la pérdida de la silla no acarrea la pérdida de sus componentes.



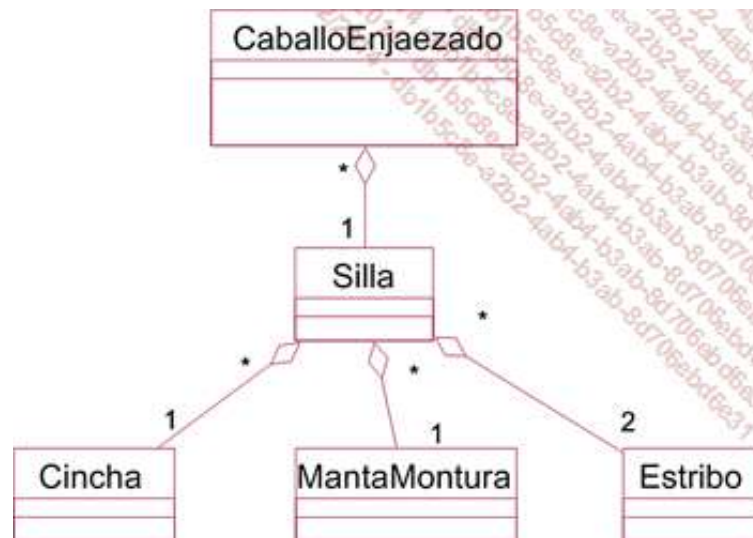


Figura 6.33 - Asociación de agregación entre un caballo enjaezado y su equipamiento y entre una silla y sus componentes

### Ejemplo

Un propietario ecuestre posee una colección de caballos. Un caballo domesticado pertenece a una sola colección y puede simultáneamente ser confiado a un criadero o no serlo. Por tanto, puede ser componente de dos agregaciones. La figura 6.34 muestra las dos asociaciones.

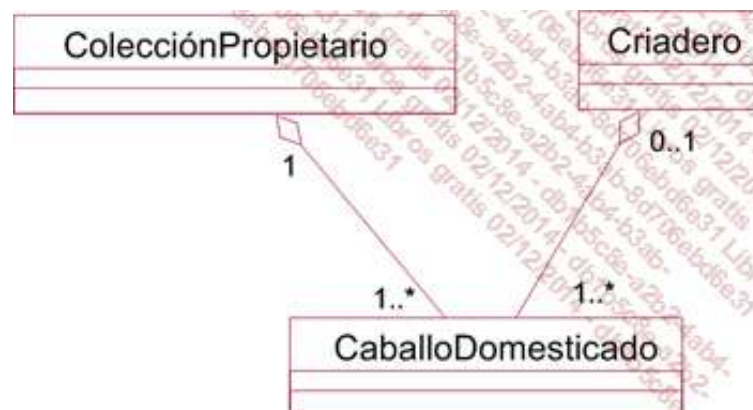


Figura 6.34 - Componente compartido por varias agregaciones distintas

### Ejemplo

Hilando más fino, un caballo puede pertenecer a varios propietarios. En ese caso, la cardinalidad a nivel de la colección del propietario ya no es 1, sino 1..\* para expresar la multiplicidad. El caballo puede entonces ser compartido varias veces en una misma agregación (ver figura 6.35).



Figura 6.35 - Componente compartido varias veces en una misma agregación

## c. Diferencias entre composición y agregación

La siguiente tabla resume las diferencias entre agregación y composición.

	Agregación	Composición

<b>Representación</b>	rombo transparente	rombo negro
<b>Varias asociaciones comparten los componentes</b>	sí	no
<b>Destrucción de los componentes al destruir el compuesto</b>	no	sí
<b>Cardinalidad a nivel del compuesto</b>	cualquiera	0..1 ó 1

# Relación de generalización/especialización entre clases

## 1. Clases más específicas y clases más generales

Una clase es más específica que otra si todas las instancias que la componen son a su vez instancias de la otra clase. La clase más específica es una subclase de la otra clase. Esta última, más general, recibe el nombre de superclase.

La figura 6.36 muestra ambas clases así como la relación de generalización/especialización que las une.

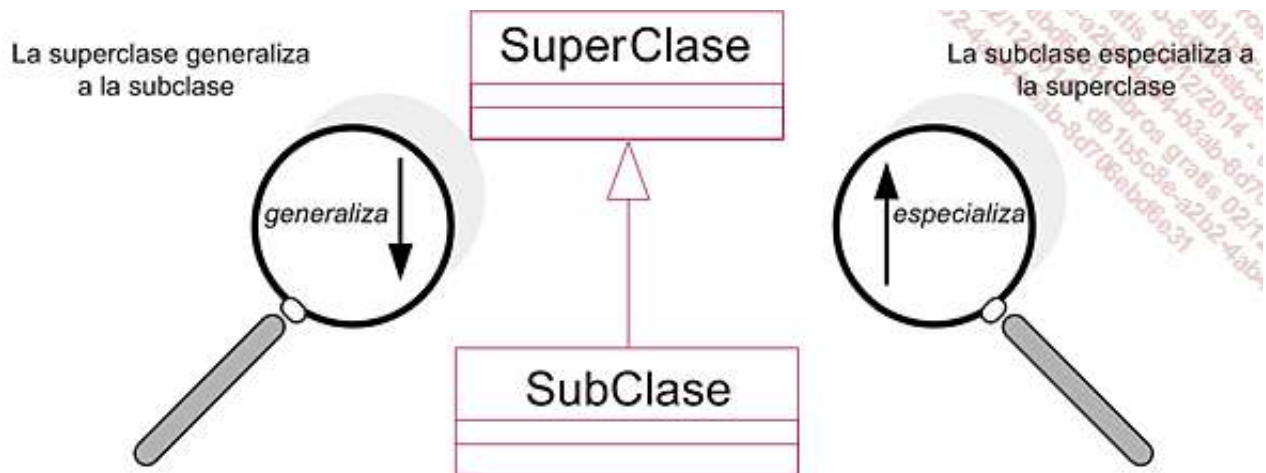


Figura 6.36 - Subclase y superclase

### Ejemplo

El caballo es una especialización del équido, que a su vez es una especialización del animal. La zebra es otra especialización del équido. El resultado es la minijerarquía presentada en la figura 6.37.

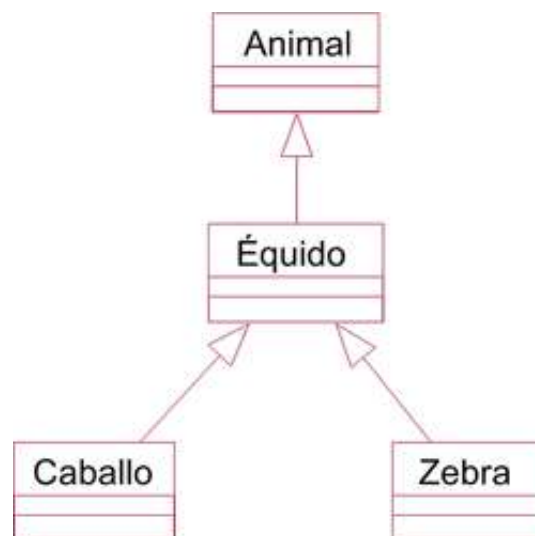


Figura 6.37 - Jerarquía de clases

## 2. La herencia

Las instancias de una clase son también instancias de su superclase o superclases. Por consiguiente, aprovechan los atributos y métodos definidos en la superclase, además de los atributos y métodos introducidos en la clase.



Esta facultad se conoce como herencia, es decir, una clase hereda los atributos y métodos de las superclases para que sus instancias se beneficien de ellos.

- Recordemos que los atributos y métodos privados de una superclase se heredan en sus subclases, pero no son visibles en ellas.
- En la herencia, los métodos pueden redefinirse en la subclase. A continuación, veremos que esta redefinición se aplica principalmente a la herencia de las clases abstractas.

### Ejemplo

Tal y como se muestra en la figura 6.38, los atributos y métodos de la clase *Caballo* se heredan en sus dos subclases.

La herencia significa que, al igual que un caballo de tiro, un caballo de carreras posee un nombre, un peso, una edad y puede caminar y comer.

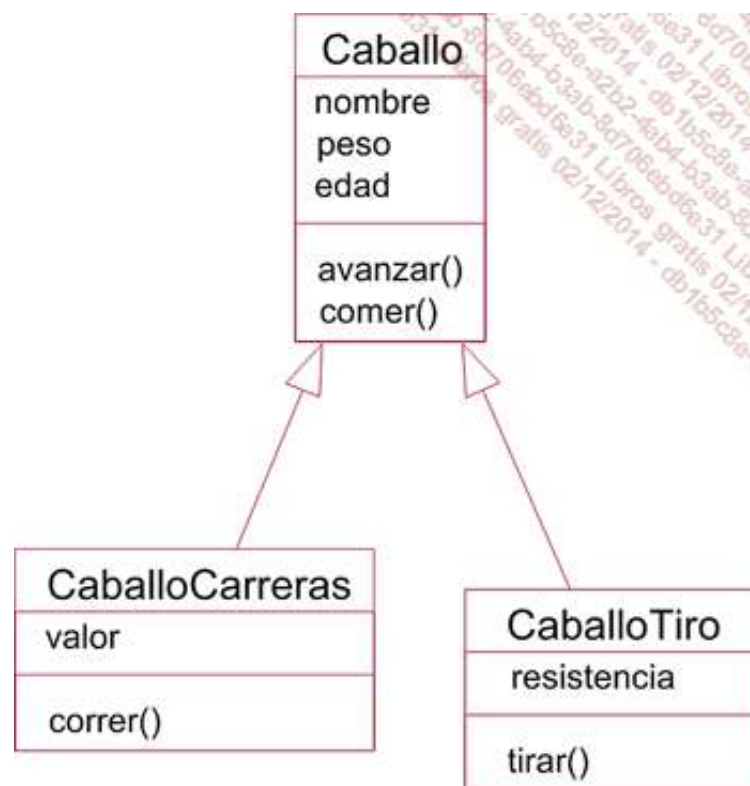


Figura 6.38 - Herencia de atributos y métodos

## 3. Clases concretas y abstractas

La figura 6.39 muestra la existencia de dos tipos de clases en la herencia: las clases concretas *Caballo* y *Lobo*, que aparecen en la parte más baja de la jerarquía, y la clase abstracta *Animal*.

Una clase concreta posee instancias y constituye un modelo completo de objeto (todos los atributos y métodos se describen completamente).

Por el contrario, una clase abstracta no puede poseer una instancia directa ya que no proporciona una descripción completa. Su finalidad es poseer subclases concretas y sirve para factorizar los atributos y métodos comunes a las subclases.

Con frecuencia la factorización de métodos comunes a las subclases se traduce en la factorización únicamente de la firma. Los métodos introducidos en una clase sólo con la firma y sin código se

denominan métodos abstractos.

En UML, las clase o métodos abstractos se representan mediante el estereotipo «abstract». Gráficamente se representan explícita, o bien implícitamente, poniendo en cursiva el nombre de la clase o del método.

### Ejemplo

*Los animales pueden dormir o comer, pero lo hacen de manera distinta de acuerdo con la naturaleza del animal. Estos métodos poseen su firma como única descripción a nivel de la clase Animal. Se trata de métodos abstractos.*

*La figura 6.39 muestra estos métodos abstractos dentro de la clase abstracta Animal y presenta la redefinición para hacerlos concretos (es decir, la atribución de código) en las clases Caballo y Lobo. En efecto, los caballos duermen de pie mientras que los lobos duermen tumbados. Tampoco comen de la misma manera: los lobos son carnívoros y los caballos herbívoros.*

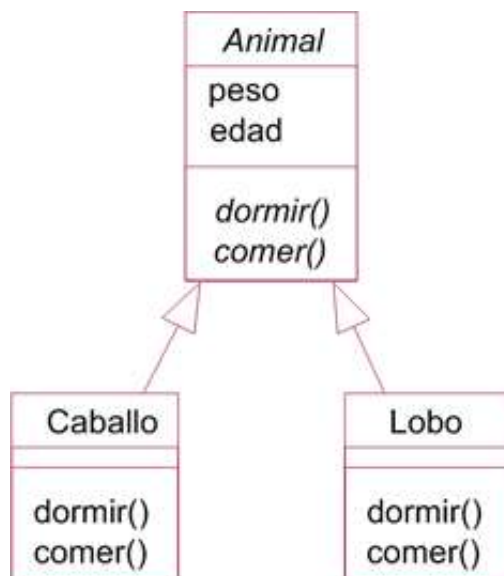


Figure 6.39 - Clases y métodos abstractos

- Recordemos que la firma se compone de un conjunto formado por el nombre del método, los parámetros con el nombre y el tipo, y el tipo del resultado, a excepción del código del método.
- Cualquier clase que posea al menos un método abstracto se considera una clase abstracta. La presencia de un solo método incompleto (el código no está) implica que la clase no es una descripción completa de objetos.

## 4. Expresión de especificaciones sobre la relación de herencia

UML ofrece cuatro especificaciones sobre la relación de herencia entre una superclase y sus subclases:

- La especificación {incomplete} significa que el conjunto de subclases está incompleto y que no cubre la superclase o incluso que el conjunto de instancias de las subclases es un subconjunto del conjunto de instancias de la superclase.
- Por el contrario, la especificación {complete} significa que el conjunto de subclases está completo y cubre la superclase.
- La especificación {disjoint} significa que las subclases no tienen ninguna instancia en común.
- La especificación {overlapping} significa que las subclases pueden tener una o varias instancias en común.

### Ejemplo

La figura 6.40 ilustra una relación de herencia entre la superclase *Équido* y dos subclases: los caballos y los burros. Estas dos subclases no cubren la clase de los équidos (existen otras subclases, como las cebras). Además, también están las mulas, que derivan de un cruce y son a la vez caballos y burros. Por ese motivo, la figura presenta las especificaciones {incomplete} y {overlapping}.

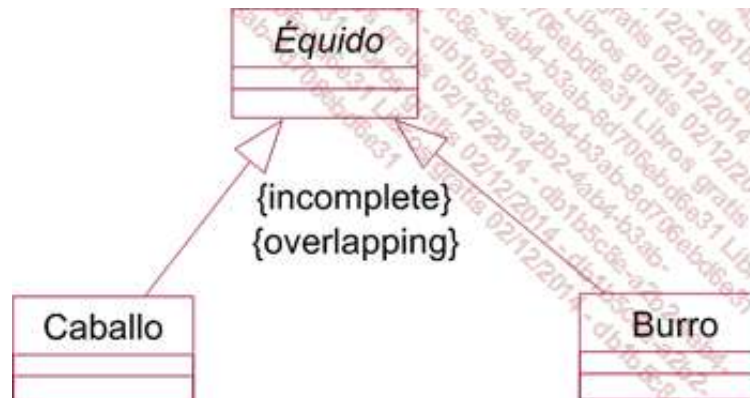


Figura 6.40 - Subclases sin cobertura y con instancias comunes

### Ejemplo

La figura 6.41 muestra otra relación de herencia entre la superclase *Caballo* y dos subclases: los caballos macho y los caballos hembra. Estas dos subclases cubren la clase de los caballos. No existe ningún caballo que sea a la vez macho y hembra. Por ello, la figura presenta las especificaciones {complete} y {disjoint}.

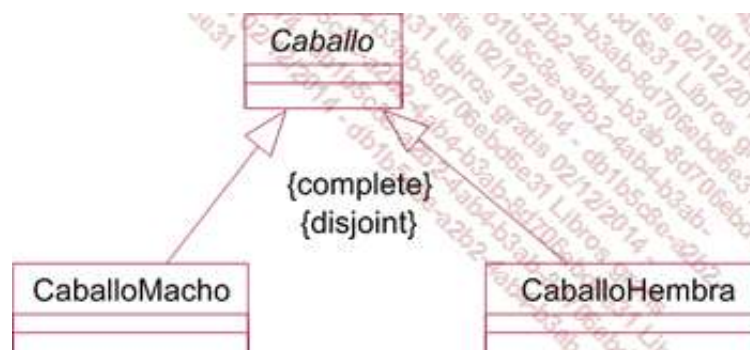


Figura 6.41 - Subclases sin cobertura y sin instancia común

## 5. La herencia múltiple

La herencia múltiple en UML consiste en que una subclase hereda de varias superclases. Esto plantea un solo problema: cuando un mismo método, es decir, un método con la misma firma, se hereda varias veces en la subclase se crea un conflicto. Al recibir un mensaje de llamada al método, es preciso definir un criterio para elegir un método entre todos los heredados.

En tales casos, una solución consiste en redefinir el método en la subclase para suprimir el conflicto.



Si bien la utilización de la herencia múltiple es posible en modelado, a menudo se revela necesario transformar los diagramas de clases para eliminarla al pasar al desarrollo. Pocos son los lenguajes de programación que soportan esta forma de herencia. Por ello, existen diferentes técnicas, entre las cuales se encuentra la transformación de la herencia múltiple en una agregación.

### Ejemplo

La figura 6.42 retoma las dos subclases *Caballo* y *Burro* e introduce una subclase común, las mulas.

La figura 6.42 muestra también el método trotar, abstracto en la clase Équido y concreto en las subclases inmediatas y redefinido en la subclase fruto de su herencia múltiple. Cuando el jinete pide a la mula que trote, ésta responde con reacciones de caballo y de burro a la vez. Puede ser peligrosa como un caballo e irritante como un burro.

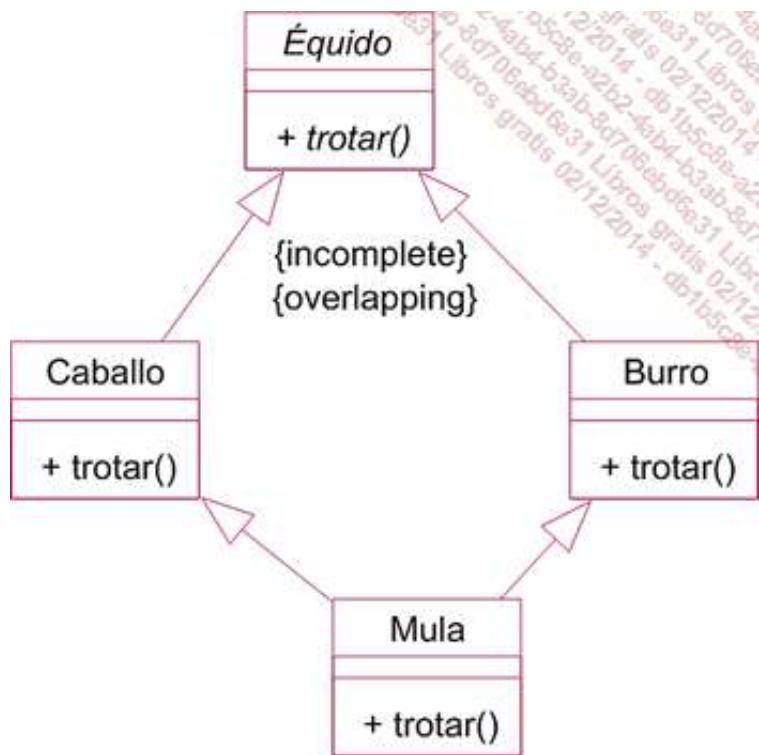


Figura 6.42 - Herencia múltiple con conflicto

## 6. Factorización de las relaciones entre objetos

Hemos visto que las clases abstractas sirven para factorizar los atributos y métodos de varias subclases. A veces también, es posible factorizar el extremo de una asociación en una superclase con el fin de hacer más simple el diagrama.

### Ejemplo

Al igual que los lobos, los caballos poseen dos ojos y una nariz (ver figura 6.43).

La figura 6.44 muestra que, una vez creada, la clase abstracta Mamífero es una superclase de las clases Caballo y Lobo, las dos asociaciones de composición se factorizan.

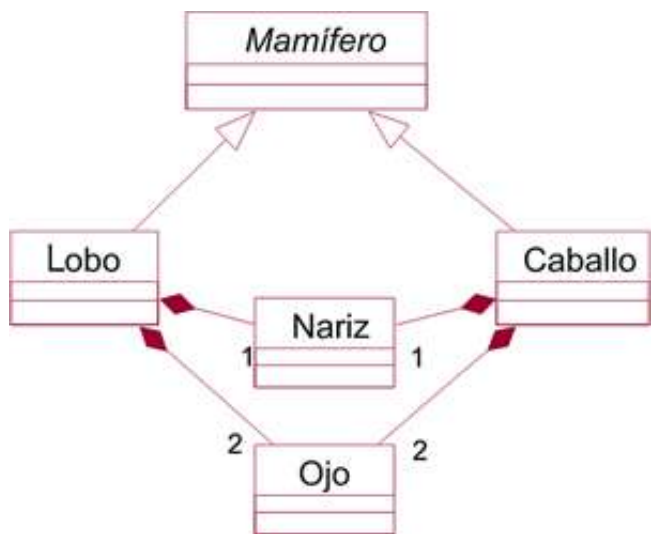


Figura 6.43 - Asociaciones entre objetos no factorizados

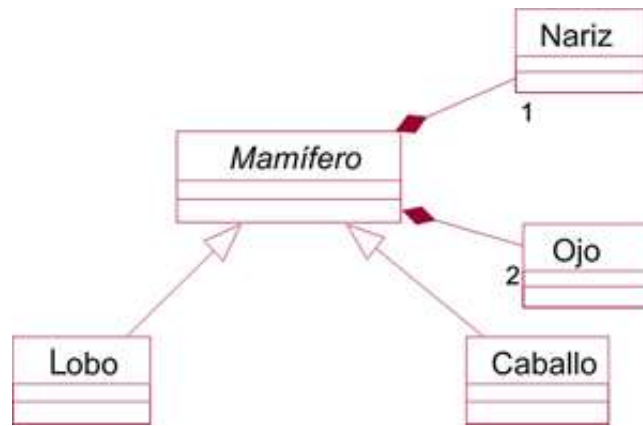


Figura 6.44 - Asociaciones entre objetos factorizados

## 7. Interfaz

Una interfaz es una clase totalmente abstracta, es decir, no tiene atributo y todos sus métodos son abstractos y públicos. Estas clases no contienen ningún elemento de implantación de los métodos. Gráficamente se representan como una clase con el estereotipo «interface».

La implantación de los métodos se realiza mediante una o varias clases concretas, subclases de la interfaz. En ese caso, la relación de herencia existente entre la interfaz y la subclase de implantación se conoce como *relación de realización*. En las relaciones de herencia entre dos clases, se representa gráficamente mediante una línea discontinua en lugar de mediante una línea completa.

La figura 6.45 muestra esta representación.

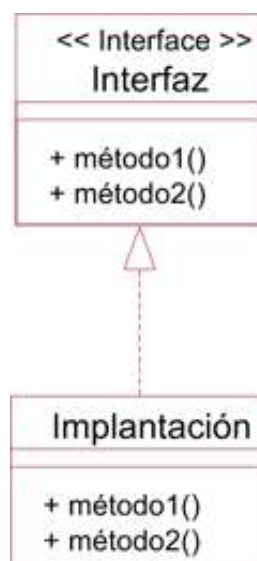


Figura 6.45 - Interfaz y relación de realización

### Ejemplo

*Un caballo de carreras puede considerarse como una interfaz. La interfaz se compone de varios métodos: correr, detenerse, etc.*

A continuación la implantación puede diferir. Las carreras al galope o al trote se realizan sólo con caballos entrenados específicamente para un tipo u otro de carrera. Para los caballos de galope, correr significa galopar. Para los caballos de trote (trotadores), correr significa trotar. Ambos responden a la interfaz CaballoCarrera, pero con una implantación diferente resultado de su entrenamiento.

La figura 6.46 muestra el ejemplo.

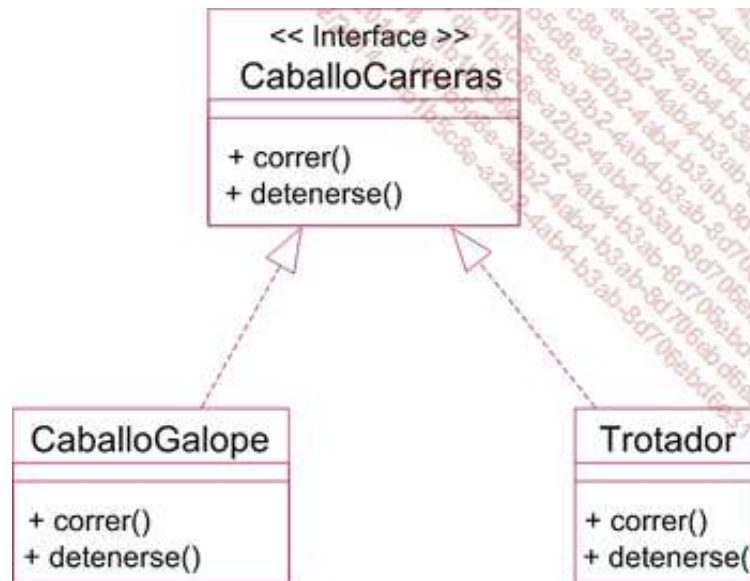


Figura 6.46 - Interfaz y subclases distintas de realización

La figura 6.45 puede también representarse con un *lollipop* (un *lollipop* es un símbolo en forma de piruleta) (ver figura 6.47).



Figura 6.47 - Representación lollipop de una interfaz y de la relación de realización

- Una misma clase puede realizar varias interfaces. Se trata de un caso particular de herencia múltiple. No puede darse ningún conflicto, ya que en la clase de realización sólo se heredan las firmas de los métodos. Si hay varias interfaces con la misma firma, ésta se implanta en la clase común de realización mediante un solo método.

Las clases también pueden depender de una interfaz para realizar sus operaciones. La interfaz se emplea entonces como tipo dentro de la clase (atributo, parámetro o variable local de uno de los métodos).

Decimos que una clase que depende de una interfaz es cliente de ella.

La figura 6.48 muestra la asociación de dependencia entre una clase y una interfaz retomando la representación de la figura 6.45 y la representación *lollipop* de la interfaz.

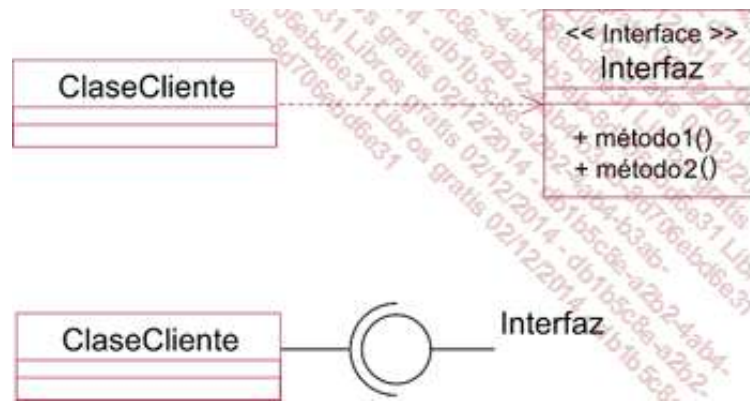


Figura 6.48 - Relación de dependencia entre una clase y una interfaz

- La relación de dependencia también existe entre dos clases. En efecto, para realizar estas operaciones, una clase puede depender de otra y no sólo de una interfaz.

### Ejemplo

Para poder organizar una carrera se necesitan caballos. La clase Carrera depende por tanto de la interfaz CaballoCarrera (ver figura 6.49).

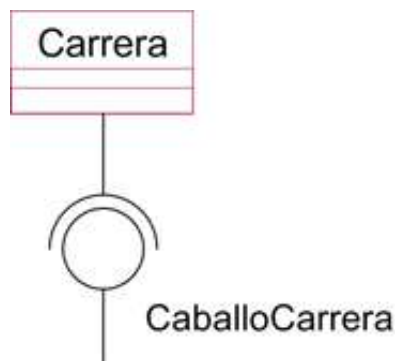


Figura 6.49 - Ejemplo de dependencia entre una clase y una interfaz



## Diagrama de objetos o instancias

El diagrama de clases es una representación estática del sistema. El diagrama de objetos muestra, en un momento determinado, las instancias creadas y sus vínculos cuando el sistema está activo.

Las instancias se representan dentro de un rectángulo con su nombre subrayado y, eventualmente, el valor de uno o varios atributos.

El nombre de una instancia presenta la forma siguiente:

`nombreInstancia : nombreClase`

El nombre de la instancia es opcional.

El valor del atributo presenta esta forma:

`nombreAtributo = valorAtributo`

Por último, los vínculos entre instancias se representan mediante simples líneas continuas. Recordemos que dichos vínculos son los elementos de las relaciones interobjetos.

### Ejemplo

La figura 6.50 muestra un ejemplo de diagramas de objetos. El diagrama de clases del que éste deriva se representa en la parte superior.

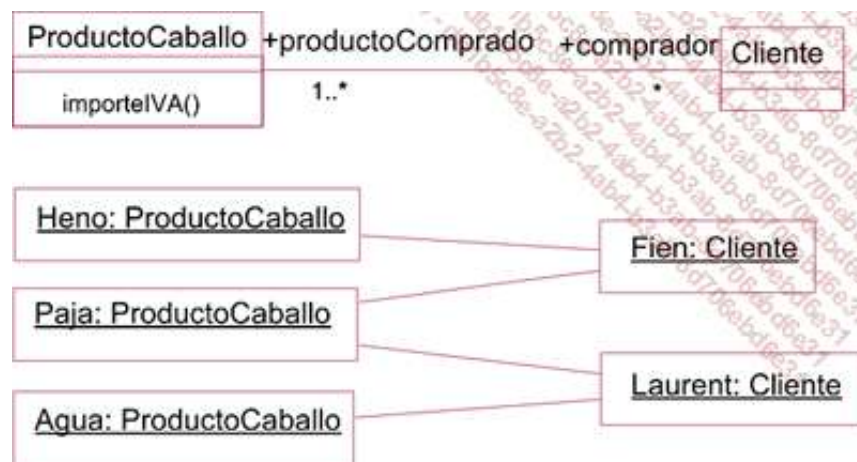


Figura 6.50 - Ejemplo de diagrama de objetos



# Diagrama de estructura compuesta

## 1. Descripción de un objeto compuesto

La finalidad principal del diagrama de estructura compuesta es describir con precisión objetos compuestos. Estos diagramas no sustituyen a los diagramas de clases, sino que los completan.

En los diagramas de estructura compuesta, el objeto compuesto se describe mediante un clasificador, mientras que sus componentes se describen mediante las partes. Un clasificador y una parte están asociados a una clase, cuya descripción completa se realiza en un diagrama de clases.

Observemos ahora el objeto compuesto descrito en el diagrama de clases de la figura 6.51. Posee un componente derivado de una composición fuerte y otro derivado de una agregación.

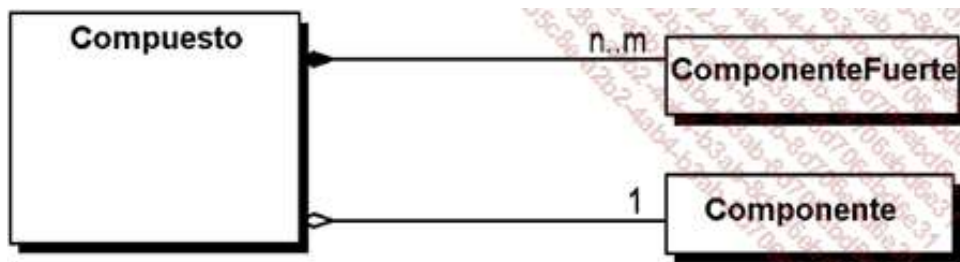


Figura 6.51 - Objeto compuesto

La figura 6.52 muestra el diagrama de estructura compuesta correspondiente a ese objeto. Los componentes están integrados dentro del clasificador que describe el objeto compuesto. El tipo de las partes es la clase del componente. La cardinalidad se indica entre corchetes. Su valor por defecto es uno. Los componentes derivados de agregaciones aparecen representados mediante una línea de puntos, los componentes derivados de composiciones fuertes aparecen representados mediante una línea continua.

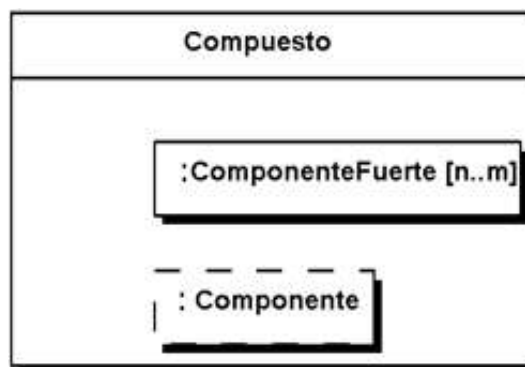


Figura 6.52 - Diagrama de estructura compuesta

### Ejemplo

La figura 6.53 muestra un ejemplo de diagrama de estructura compuesta en el que se describe un automóvil en tanto que objeto compuesto. El diagrama de clases correspondiente aparece representado abajo.

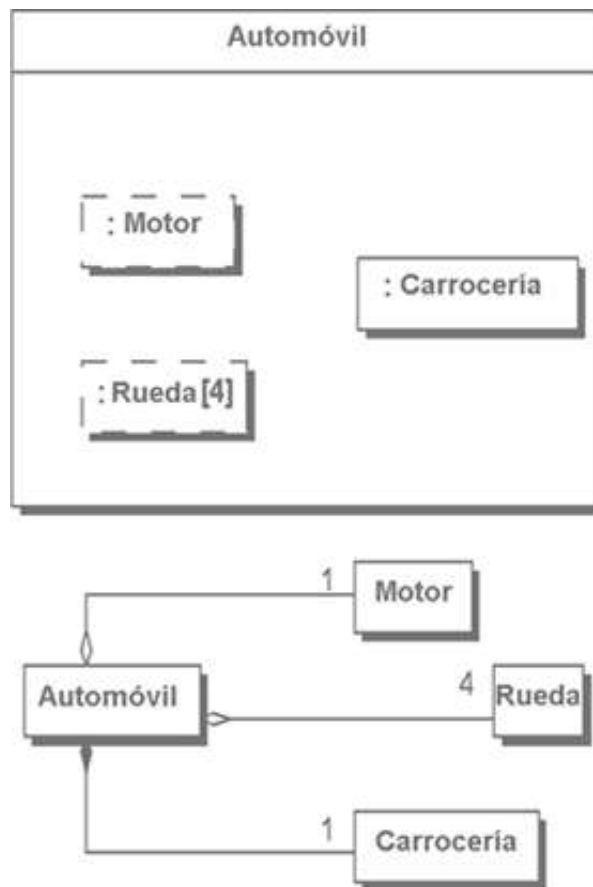


Figura 6.53 - Ejemplo de diagrama de estructura compuesta

➤ En el diagrama de estructura compuesta, Motor, Carrocería y Rueda no son clases, sino partes. Las partes se consideran siempre dentro de clasificadores.

El diagrama de clases de la figura 6.54 muestra nuevamente un automóvil en tanto que objeto compuesto. Introduce la asociación *vinculada* entre las ruedas y los semiárboles que se ocupan de la transmisión entre el motor y las ruedas delanteras, que son las ruedas motrices.

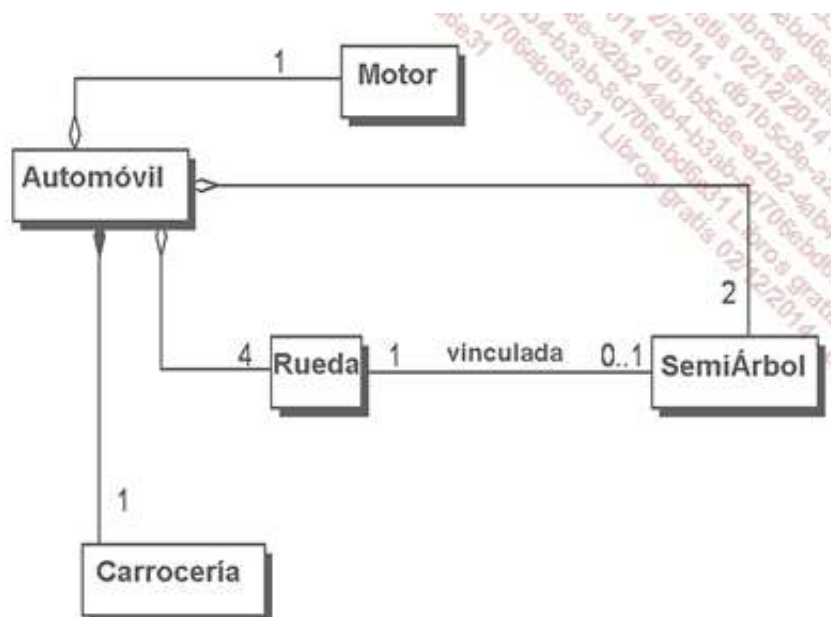


Figura 6.54 - Diagrama de clases del objeto compuesto Automóvil con los semiárboles

La cardinalidad de la asociación *vinculada* es 0..1 en el extremo del semiárbol. En efecto, la cardinalidad de

las ruedas delanteras es uno y la de las traseras es cero. Esta última información no puede ser descrita en el diagrama de clases, a menos que se introduzcan dos subclases de *Rueda*: *RuedaDelantera* y *RuedaTrasera*. No obstante, el inconveniente de incorporar dos subclases para precisar una cardinalidad es que el diagrama de clases se hace más enrevesado. La opción no es, por tanto, demasiado deseable.

El diagrama de estructura compuesta permite especificar la función de una parte. La función describe el uso de la parte dentro del objeto compuesto. La figura 6.55 introduce tres partes para las ruedas correspondientes a la rueda delantera izquierda, la rueda delantera derecha y las dos ruedas traseras, respectivamente. La cardinalidad de las partes se adapta en consecuencia. El nombre de la función se indica en la parte antes del tipo.

- Esta notación no es la misma que la usada en el diagrama de objetos. Efectivamente, la notación para especificar una instancia utiliza el estilo subrayado.

En la figura 6.55 se han introducido también dos partes correspondientes a cada semiárbol.

Entre cada parte correspondiente a una rueda delantera y cada parte correspondiente a un semiárbol, un conector representa la asociación *vinculada*. Los conectores sirven para unir dos partes. Están tipificados por una asociación interobjetos, del mismo modo que una parte está tipificada por una clase.

- Si existen varios conectores entre dos partes y éstos están tipificados por la misma asociación, es posible distinguirlos atribuyéndoles nombres de función al modo de los nombres de función de las partes.

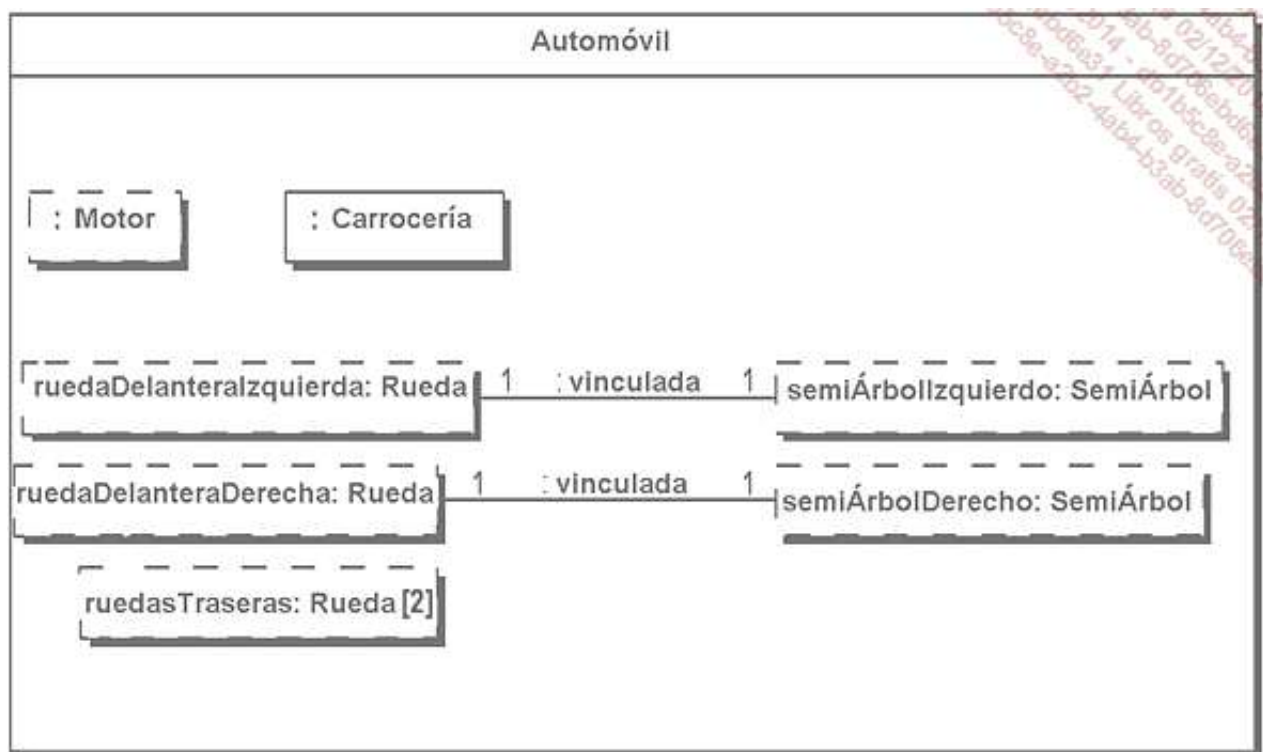




Figura 6.55 - Diagrama de estructura compuesta con funciones y conectores

Los conectores pueden también vincular las partes entre ellas a través de puertos. Un puerto es un punto de interacción. Posee una interfaz que constituye su tipo y define el conjunto de interacciones posibles. Las interacciones definidas por un puerto se hacen con los otros puertos vinculados a él mediante un conector.

Los puertos también pueden introducirse en los clasificadores. En ese caso, el objetivo de los puertos es servir de pasarela entre las partes internas del clasificador y los objetos externos a éste (su entorno).


Desde el punto de vista de la encapsulación, se trata de puertos generalmente públicos. Y se conocen, por

tanto, fuera del clasificador.

-  Un puerto definido en el clasificador puede también definirse como privado. Se reserva entonces a una comunicación interna entre el clasificador y sus partes. No realiza funciones de pasarela entre el interior y el exterior del clasificador.
-  Una parte puede poseer varios puertos, cada uno de ellos dotado de su propia interfaz. Varios puertos de una misma parte pueden tipificarse con la misma interfaz. En esos casos es posible distinguirlos asignándoles nombres de función diferentes, a la manera de lo que se hace con las partes y los conectores.

La figura 6.56 muestra la misma descomposición en partes del objeto *Automóvil* que en el último ejemplo. Entre el motor y los semiárboles de transmisión se han agregado algunos conectores. Los conectores entre las partes están unidos a través de un puerto representado en forma de cuadrado blanco. También se ha añadido un puerto en el clasificador, que está tipificado por la interfaz *Orden* y conectado a un puerto del motor igualmente tipificado por esa interfaz.

En lo que se refiere a las interacciones, la figura muestra:

- Que la clase *Automóvil* puede interactuar con el exterior para recibir órdenes destinadas al motor y que le son transmitidas.
  - Que el motor se comunica con los semiárboles (transmisión de movimiento).
  - Que cada semiárbol se comunica con las ruedas (transmisión de movimiento).
-  Los conectores no están tipificados por razones de simplificación. De igual modo, la interfaz de los puertos de los semiárboles, de las ruedas y del puerto del motor conectado a los semiárboles no se ha indicado. Podría tratarse de la interfaz *Transmisión*.

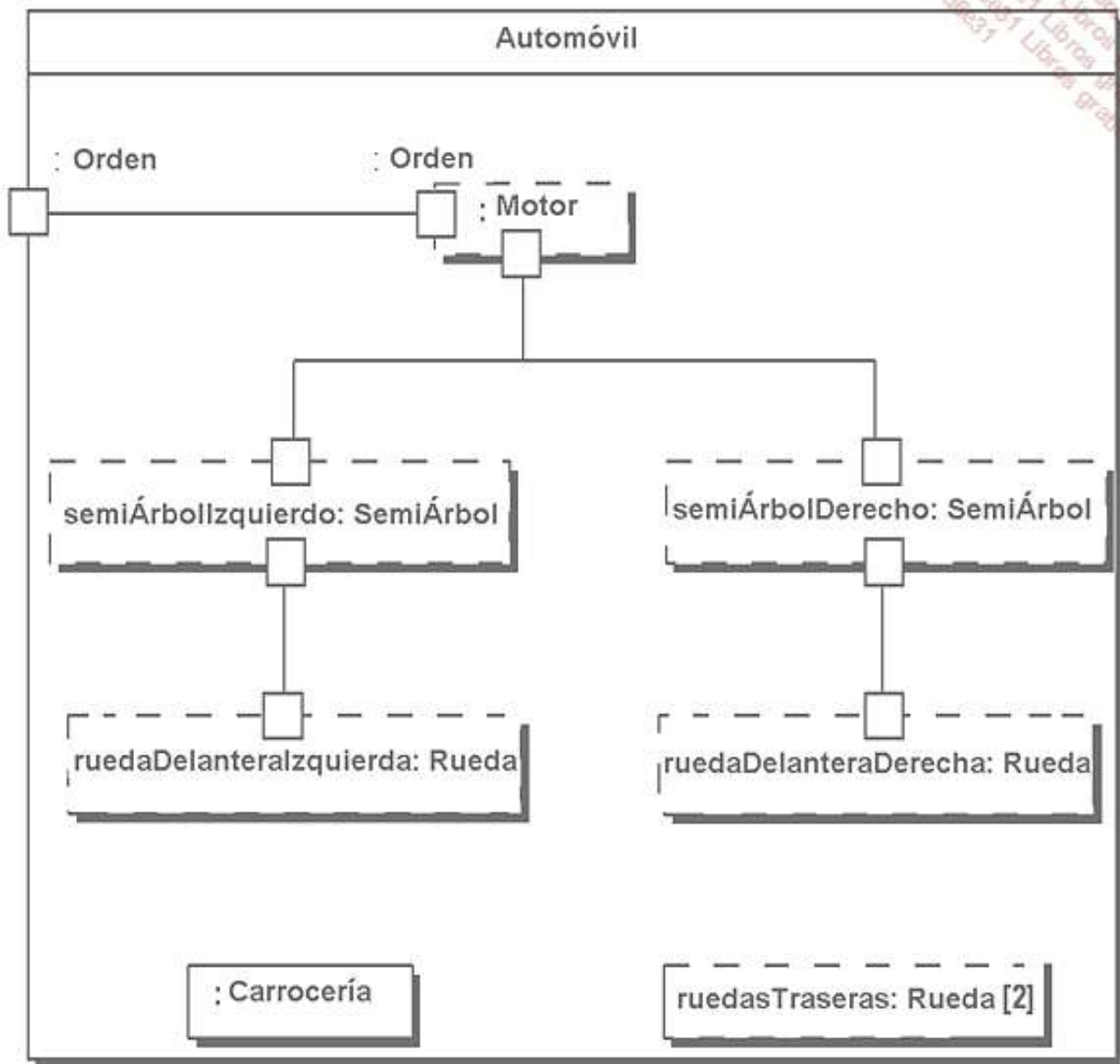


Figura 6.56 - Diagrama de estructura compuesta en el que se han introducido una serie de puertos

## 2. Colaboración

Las colaboraciones describen un conjunto de objetos que interactúan entre sí con el fin de ilustrar la funcionalidad de un sistema. Cada objeto participa en esa funcionalidad efectuando un papel concreto.

En las colaboraciones, los objetos se describen como en un clasificador, con los mismos elementos: partes, conectores, puertos, etc.

Las colaboraciones pueden usarse para describir los patrones de diseño (Design Patterns) utilizados en la programación con objetos.

La figura 6.57 muestra el diagrama de clases de uno de esos patrones; a saber, el patrón *Composite*.

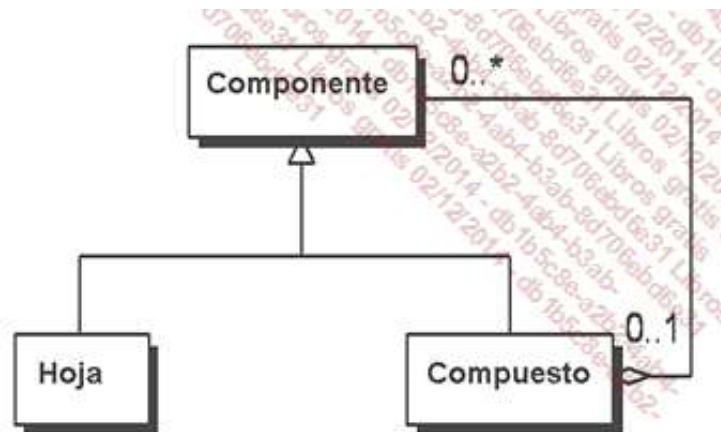


Figura 6.57 - Pattern Composite

La finalidad del patrón *Composite* es ofrecer un marco de diseño de una composición de objetos cuya profundidad puede variar. Un componente puede ser una hoja o un objeto compuesto, a su vez, por hojas y otros compuestos. Uno de los ejemplos que mejor ilustran este tipo de patrón es el sistema de archivos del disco duro de un ordenador personal, compuesto de archivos y carpetas. Cada carpeta puede, a su vez, contener archivos u otras carpetas.

En la figura 6.58 se muestra una colaboración en la que se describe el patrón *Composite*. En ella están representadas las dos partes de la colaboración, a saber, la de la clase *Hoja* y la de la clase *Compuesto*. Toda la hoja está necesariamente contenida en un solo y único compuesto, es decir, que existe al menos un compuesto. Los compuestos pueden estar contenidos en otros compuestos o no (caso del compuesto raíz).

- Las colaboraciones no sustituyen el diagrama de clases, sino que lo completan. Como mencionamos al principio: el diagrama de estructura compuesta no está llamado a reemplazar el diagrama de clases, sino a completarlo.

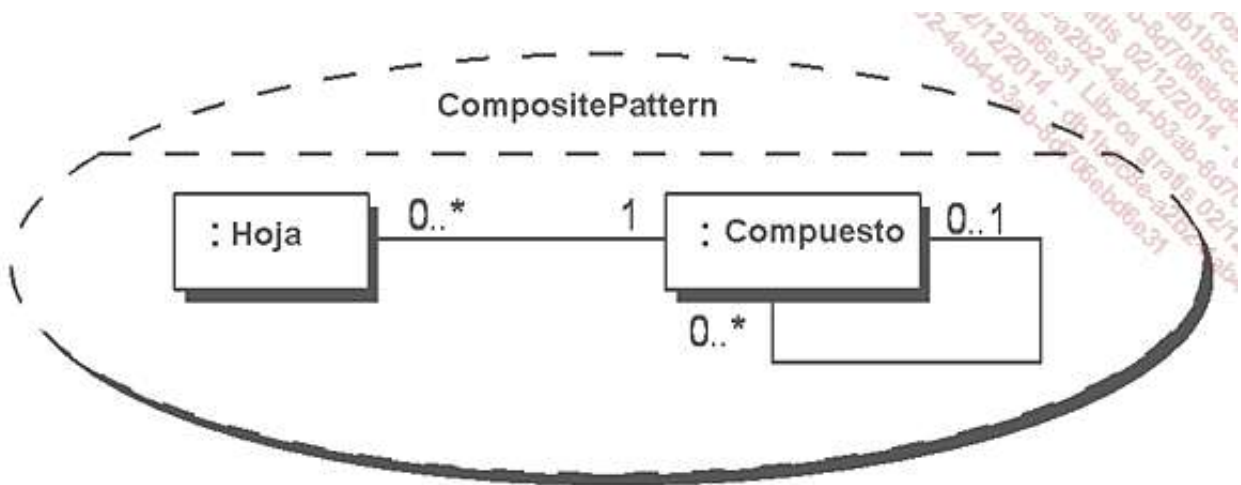


Figura 6.58 - Colaboración en la que se describe el patrón Composite

El diagrama de estructura compuesta ofrece la posibilidad de describir una aplicación de una colaboración fijando las funciones de las partes. Es lo que hemos hecho en la figura 6.59 tomando nuestro ejemplo de archivos como aplicación de la colaboración del patrón *Composite*.

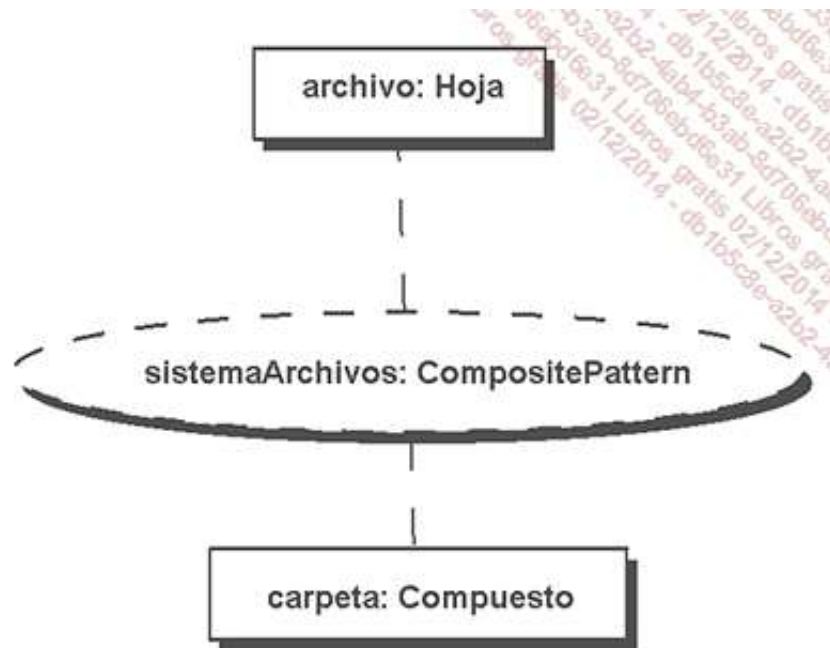


Figura 6.59 - Aplicación de la colaboración Composite al sistema de archivos

## Conclusión

En el presente capítulo hemos estudiado los diagramas de clases. Las clases describen los atributos y los métodos de sus instancias, así como los atributos y métodos de clase.

Las asociaciones entre objetos son obligatorias al elaborar un diagrama de clases. Dichas asociaciones constituyen la base de la interacción de las instancias a través del envío de mensajes cuando el sistema está activo.

Las relaciones de herencia entre clases son, asimismo, indispensables, ya que favorecen la factorización de elementos comunes, permitiendo así reducir el tamaño del diagrama.

Por último, la expresión de las especificaciones en lenguaje natural o en OCL conduce a enriquecer aún más la semántica expresada en el diagrama.



# Ejercicios

## 1. La jerarquía de los caballos

Tenemos las clases *Yegua*, *Semental*, *Potro*, *Potranca*, *Caballo*, *Caballo macho* y *Caballo hembra*, así como las asociaciones padre y madre. Establezca la jerarquía de las clases haciendo figurar en ella ambas asociaciones.

Utilice las especificaciones {incomplete}, {complete}, {disjoint} y {overlapping}.

Introduzca la clase *Manada*. Establezca la asociación de composición entre esta clase y las clases ya introducidas.

## 2. Los productos para caballos

Modele los aspectos estáticos del texto siguiente en forma de diagrama de clases.

Una central de caballos vende diferentes tipos de productos para caballos: *productos de mantenimiento*, *alimentación*, *equipamiento* (para montar el caballo), *herraje*.

Un pedido contiene una serie de productos y especifica la cantidad de cada uno de ellos. En caso necesario, se puede elaborar un presupuesto antes de pasar el pedido. Si alguno de los productos no está en stock, a petición del cliente el pedido puede dividirse en varias entregas. Cada entrega da lugar a una factura.

# Introducción

UML 2 describe los empaquetados gracias a un diagrama específico. Un empaquetado es una agrupación de elementos de modelado: clases, componentes, casos de uso, otros empaquetados, etc.

Los empaquetados de UML resultan útiles durante el modelado de sistemas importantes para reagrupar los diferentes elementos. La agrupación estructura de ese modo el modelado.



En UML 1, los empaquetados formaban parte del diagrama de clases y reagrupaban exclusivamente conjuntos de clases.

## Empaquetado y diagrama de empaquetado

Los empaquetados se representan con una carpeta (ver figura 7.1) y constituyen un conjunto de elementos de modelado UML.

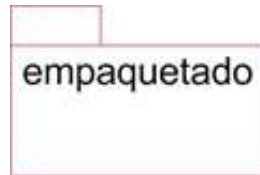


Figura 7.1 - Representación gráfica de un empaquetado

### Ejemplo

La figura 7.2 muestra el empaquetado que representa los diferentes elementos de modelado de un criadero de caballos.

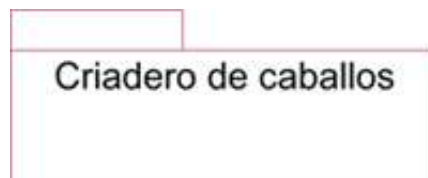


Figura 7.2 - Ejemplo de empaquetado

El contenido de un empaquetado se describe mediante un diagrama de empaquetado que representa los diferentes elementos del mismo con su propia representación gráfica. Estos elementos pueden ser clases, componentes, casos de uso, otros empaquetados, etc.

Los elementos de un empaquetado pueden incluirse directamente dentro de la carpeta que lo representa.

### Ejemplo

El contenido del empaquetado "Criadero de caballos" se ilustra a través de su diagrama. Éste consta de tres empaquetados que contienen casos de uso y clases (ver figura 7.3).

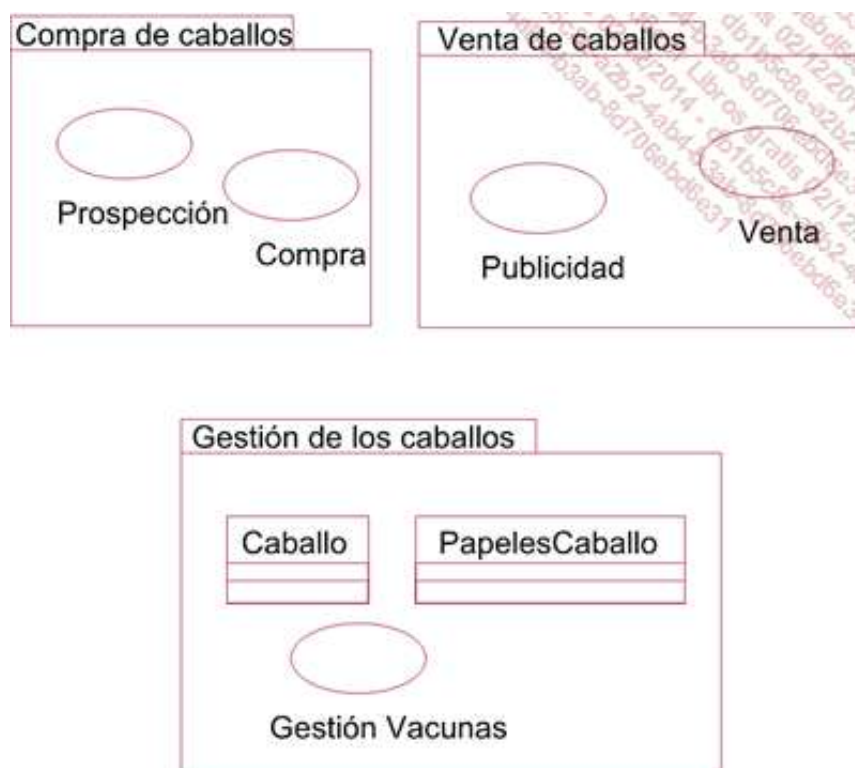


Figura 7.3 - Ejemplo del diagrama de empaquetado que representa un criadero de caballos

- Los elementos del modelado de un sistema pertenecen a un solo empaquetado. Más adelante veremos cómo pueden ser compartidos por varios empaquetados.

Los elementos incluidos en el empaquetado pueden ser accesibles en el exterior o estar encapsulados dentro del mismo. Por defecto, los elementos son accesibles en el exterior.

La encapsulación se representa mediante un signo más o un signo menos delante del nombre del elemento. El signo más significa que no hay encapsulación y que el elemento es visible fuera del empaquetado. El signo menos significa que el elemento está encapsulado y no es visible en el exterior.

#### Ejemplo

La figura 7.4 muestra el empaquetado "Criadero de caballos" para el cual ha sido precisa la encapsulación.

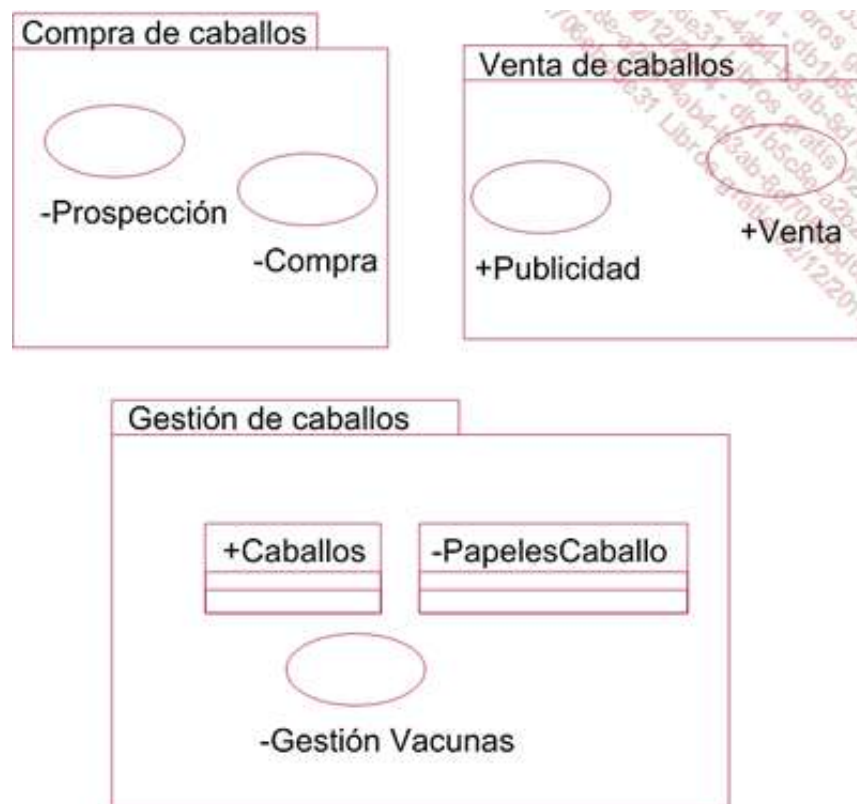


Figura 7.4 - Ejemplo de diagrama de empaquetado con elementos encapsulados y visibles

## Asociaciones entre empaquetados

Los elementos del modelado sólo pueden estar presentes en un empaquetado.

Para que un empaquetado pueda utilizar los elementos de otro, existen dos tipos de asociación:

- La asociación de importación consiste en llevar un elemento del empaquetado de origen al empaquetado de destino. El elemento forma parte entonces de los elementos visibles del empaquetado de destino.
- La asociación de acceso consiste en acceder a un elemento del empaquetado de origen desde el empaquetado de destino. El elemento no forma parte entonces de los elementos visibles del empaquetado de destino.

Sólo es posible importar o acceder a un elemento si éste está definido como visible en el empaquetado de origen.

Las dos asociaciones pueden aplicarse también a los empaquetados completos: importan o acceden a todos los elementos del empaquetado de origen definidos como visibles.

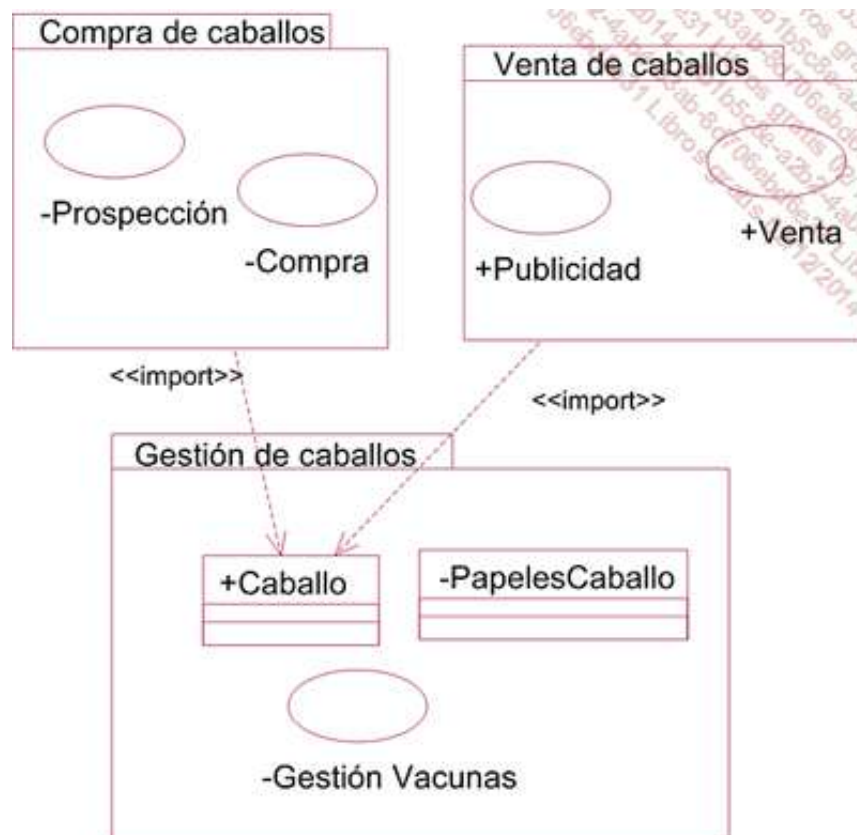
Ambas son asociaciones de dependencia, especializadas con ayuda de los estereotipos «import» o «access».

➤ En UML 1, sólo existía la dependencia entre empaquetados. En UML 2, ésta se ha especializado y ha dado lugar a las dos asociaciones que acabamos de ver. A pesar de ello, sigue siendo posible utilizar la dependencia simple de UML 1.

### Ejemplo

En el empaquetado "Criadero de caballos", los empaquetados "Compra de caballos" y "Venta de caballos" importan la clase Caballo. El resultado habría sido el mismo si ambos empaquetados hubieran importado el empaquetado "Gestión de los caballos", ya que la clase Caballo es la única pública.

La figura 7.5 muestra las dos asociaciones de importación.



*Figura 7.5 - Asociaciones de importación entre empaquetados*

## Conclusión

Los empaquetados son útiles para estructurar el modelado de sistemas importantes. Los empaquetados *sistema* contienen empaquetados de mayor nivel que, a su vez, albergan otros empaquetados y así sucesivamente hasta llegar a los elementos básicos del modelado, como las clases o los casos de uso.

Una de las ventajas de esta estructuración del modelado es que forma parte de la notación UML y, por consiguiente, está estandarizada.

# Introducción

Una vez estudiadas las interacciones y los aspectos estáticos de los objetos del sistema, procederemos a abordar su ciclo de vida. El ciclo de vida de un objeto representa las diferentes etapas o estados por los que pasa para llegar, dentro del sistema, a realizar un objetivo. Un estado corresponde a un momento de actividad o de inactividad del objeto. La inactividad se produce cuando el objeto espera a que otros objetos concluyan una actividad.

Estudiaremos la noción de evento, señal que hace que el objeto cambie de estado. El cambio de estado consiste en traspasar una transición.

El diagrama de estados-transiciones ilustra el conjunto de estados del ciclo de vida de un objeto separados por transiciones. Cada transición se asocia a un evento.

Examinaremos detalladamente la noción de señal y las posibilidades de enviarla en diferentes momentos. Veremos también que es posible asociar condiciones para traspasar transiciones y administrar así las alternativas.

Analizaremos la noción de estado compuesto con el fin de simplificar la escritura del diagrama de estados-transiciones. Veremos que es posible disponer de subestados que evolucionen en paralelo.

Por último, presentaremos el diagrama de *timing*, diagrama que describe los cambios de estado de un objeto cuando éstos se producen exclusivamente en función del tiempo.



## La noción de estado

El estado de un objeto corresponde a un momento de su ciclo de vida. Mientras se encuentran en un estado, los objetos se limitan a esperar una señal procedente de otros objetos. Cuando ocurre esto decimos que están inactivos. Sin embargo, también pueden estar activos y realizar actividades. Una actividad es la ejecución de una serie de métodos y de interacciones con otros objetos. Está vinculada a un objetivo. En el capítulo siguiente, estudiaremos detalladamente su descripción con ayuda de los diagramas de actividades.

### Ejemplo

*En un concurso de salto de obstáculos, los caballos se encuentran en estado de reposo antes de empezar la competición. Se trata de un estado inactivo y a la espera de la orden de salida.*

*Cuando saltan un obstáculo, los caballos están en un estado activo que concluye al acabar de saltar el obstáculo.*



Esta nota esta dirigida, sobre todo, a los desarrolladores que utilizan lenguajes de objetos como Java o C++: con dichos lenguajes la mayoría de programas funcionan en monothread, es decir, en monotarea. En ese caso, decimos que un objeto está inactivo cuando no tiene ningún método activado y que se vuelve activo al activar uno de sus métodos. La activación corresponde a la recepción de una señal en UML.

Los estados del ciclo de vida de un objeto contienen un estado inicial -que corresponde al estado del objeto justo después de su creación- y uno o varios estados finales, que corresponden a la fase de destrucción del objeto. También puede ocurrir que no exista estado final, ya que es posible que un objeto no sea nunca destruido.

# El cambio de estado

## 1. Noción de evento y de señal

Un evento es un hecho que activa el cambio de estado. Está condicionado a que el objeto reciba una señal. La recepción de dicha señal equivale a la recepción de un mensaje. Tratamos el tema de los envíos y recepciones de mensajes en el capítulo Modelado de la dinámica, dedicado a las interacciones.

- La señal puede ser emitida por cualquier objeto, incluso por el propio objeto que está a la espera de la señal para cambiar de estado.

UML propone describir las señales mediante clases. Las señales emitidas y recibidas son entonces instancias de una clase de señales. Para distinguirlas del resto de clases, las clases de señales se describen con el estereotipo «señal». Los atributos de los objetos de esas clases son los parámetros de mensaje. La descripción mediante clases conduce a la organización de las señales jerárquicas.

- UML no obliga a describir de forma precisa los eventos. En una primera fase de modelado, los eventos pueden especificarse sólo con su nombre.

### Ejemplo

La figura 8.1 muestra la jerarquía de señales que pueden recibir el jinete y su caballo. Son señales emitidas por el juez o bien por uno de los obstáculos.

- Consideramos que un obstáculo puede emitir señales al mismo título que el juez. No hay que olvidar que en modelado mediante objetos, todos los objetos deben interactuar unos con otros, aunque en el mundo real sean pasivos.

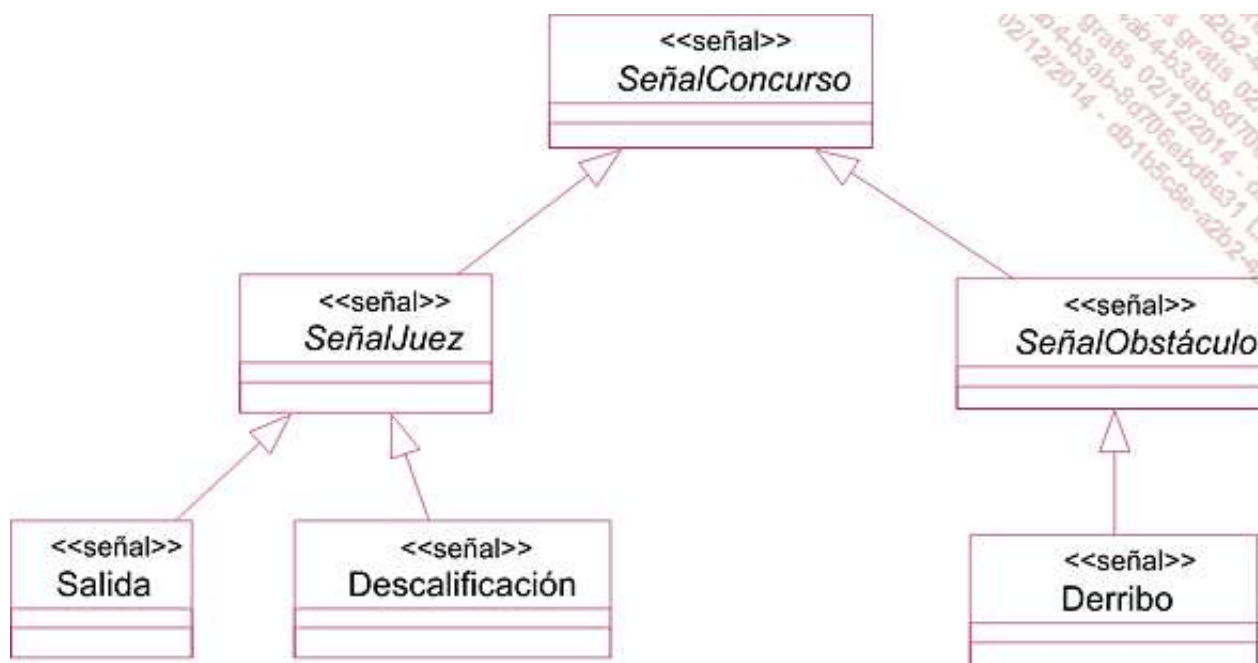


Figura 8.1 - Ejemplo de jerarquía de clases que representan señales

- La operación consistente en describir una estructura, en este caso en describir un mensaje mediante una clase, se conoce como reificación. Reificar es transformar en cosa. En la orientación a objetos, por tanto, es transformar en objeto.

## 2. La transición

Una transición es un vínculo orientado entre dos estados que expresa que el objeto puede pasar del estado inicial de la transición al estado de destino. Cuando el objeto realiza este paso del estado inicial al estado de destino, decimos que se ha traspasado la transición.

Las transiciones se asocian generalmente a eventos. En esos casos, la transición se traspasa si el objeto se encuentra en el estado inicial de la transición y recibe el evento. Este traspaso tiene lugar tanto si el objeto está activo como si no. Si está inactivo, el traspaso se produce inmediatamente, mientras que si está activo, éste tiene lugar cuando finaliza la actividad asociada al estado.

### Ejemplo

*Cuando el jinete y el caballo reciben la orden de salida del juez pasan del estado de espera al estado de carrera.*

Las transiciones automáticas no se asocian a eventos, sino que se traspasan cuando el objeto termina la actividad vinculada al estado inicial. En esos casos es preciso asociar una actividad al estado inicial.

### Ejemplo

*Cuando el jinete y el caballo terminan el recorrido pasan automáticamente al estado final.*

Una transición reflexiva posee el mismo estado inicial y final. Si la transición está asociada a un evento, la recepción de éste no hace cambiar el estado. Si la transición es reflexiva y automática resulta útil para realizar una actividad en bucle.



Si un evento asociado a una transición reflexiva no hace cambiar el estado del objeto, su recepción puede provocar reacciones como la llamada a un método del objeto o el envío de una señal a otros objetos.

### Ejemplo

*Mientras el caballo se niegue a saltar un obstáculo permanecerá en el estado de carrera que precede a dicho obstáculo. El número de intentos aumenta de uno en uno.*

# Elaboración del diagrama de estados-transiciones

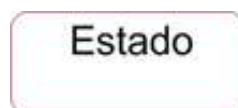
El diagrama de estados-transiciones representa el ciclo de vida de las instancias de una clase.

Describe los estados, las transiciones que los vinculan y los eventos que provocan el traspaso de las transiciones.

Este tipo de diagramas sólo resulta útil para los objetos que tienen un ciclo de vida. Otros objetos, simplemente portadores de información, no cambian de estado a lo largo de su vida y, por tanto, resulta inútil crear un diagrama de estados-transiciones para ellos.

## 1. Representación gráfica de los elementos básicos

Los estados se representan mediante un rectángulo de esquinas redondeadas con su nombre en el interior. La figura 8.2 muestra la representación gráfica de un estado.



*Figura 8.2 - Representación gráfica de un estado*

En un diagrama de estados-transiciones, el primer estado corresponde al estado inicial del objeto a la salida de su fase de creación. Este estado es único en un diagrama de estados-transiciones.

El estado inicial se representa mediante un punto negro (ver figura 8.3).



*Figura 8.3 - Representación gráfica del estado inicial*

Un estado final corresponde a una etapa en la que el objeto ya no resulta necesario en el sistema y se destruye. No todos los objetos tienen estado final. Ese es el caso, sobre todo, de los objetos permanentes del sistema.

Un estado final se representa con un punto negro rodeado de un círculo (ver figura 8.4).



*Figura 8.4 - Representación gráfica de un estado final*

Una transición entre dos estados se representa mediante una línea negra con una flecha en un extremo que une ambos estados (ver figura 8.5). El evento que determina el traspaso de la transición se indica al lado de la transición. Si la transición es automática no se indica ningún evento.



*Figura 8.5 - Representación gráfica de una transición*

Una transición reflexiva posee el mismo estado inicial y final (ver figura 8.6).



*Figura 8.6 - Representación gráfica de una transición reflexiva*

### Ejemplo

*En un concurso de obstáculos, la prueba consiste en que cada participante salte dos o tres obstáculos diferentes. A veces, ocurre que el caballo se niega a saltar el obstáculo, entonces el jinete vuelve a intentar el salto. La figura 8.7 representa el diagrama de estados-transiciones que describe la prueba del objeto "participante en la prueba". Los obstáculos son el muro y la barrera respectivamente. El diagrama contiene transiciones reflexivas y automáticas.*

- La posibilidad de saltar nuevamente un obstáculo se limita a dos intentos, sin contar el intento inicial. Si tras estos intentos el participante no supera el obstáculo, queda eliminado. Más adelante veremos cómo tener en cuenta esta especificación.

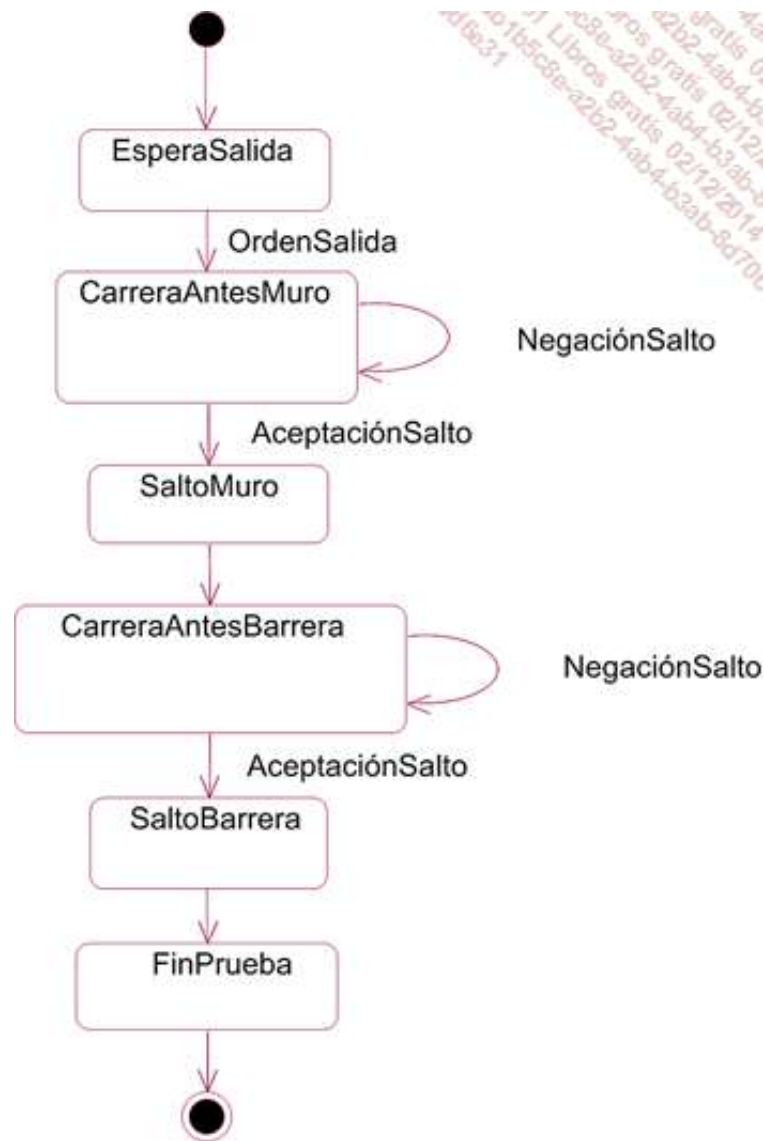


Figura 8.7 - Ejemplo de diagrama de estados-transiciones

## 2. Condiciones de guarda

Es posible asociar condiciones a una transición, éstas se conocen como *condiciones de guarda*. Para traspasar la transición, además de recibir el evento asociado a ella, en caso de que exista, es preciso que la condición se cumpla.

Las condiciones de guarda se escriben entre corchetes. Si hay un evento asociado a la transición, la condición se expresa a la derecha del nombre del evento (ver figura 8.8).

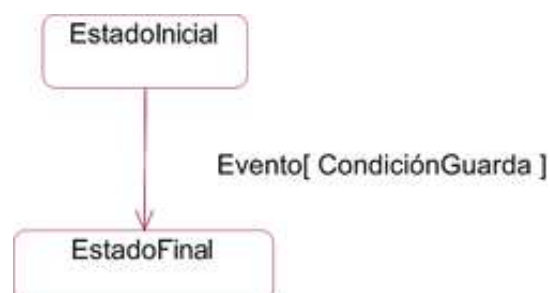


Figura 8.8 - Condición de guarda

Ejemplo

En caso de que el caballo se niegue a saltar un obstáculo, el participante tiene derecho a intentarlo de nuevo dos veces. Queda descalificado, por tanto, tras la tercera tentativa fallida.

La figura 8.9 muestra cómo tener en cuenta el número máximo de intentos fallidos retomando el ejemplo de la figura 8.6 (sólo se muestra el primer obstáculo).

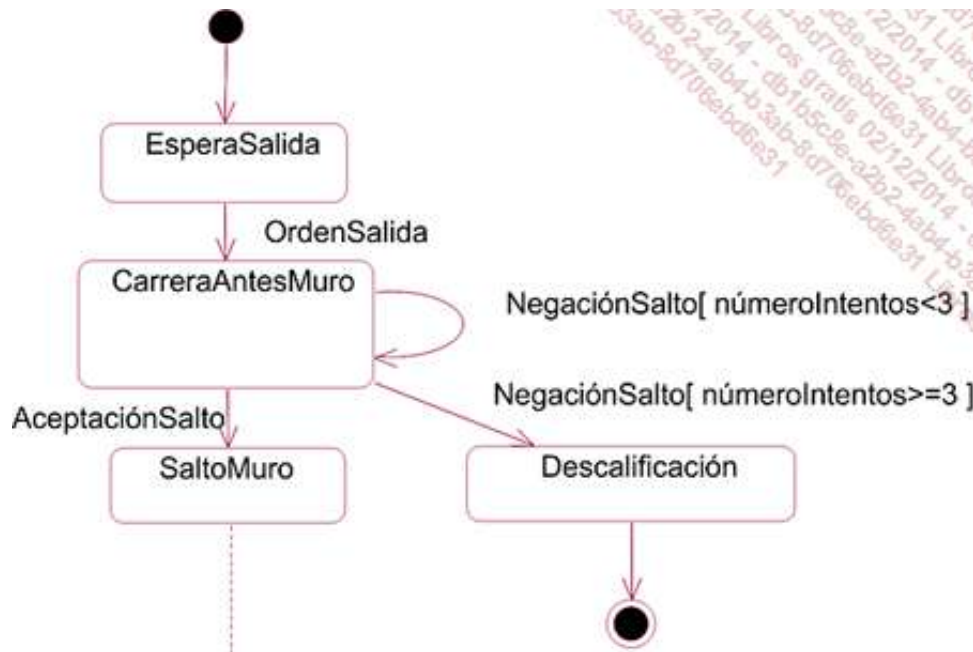


Figura 8.9 - Ejemplo de uso de las condiciones de guarda

### 3. Actividades vinculadas a un estado o al traspaso de una transición

Podemos especificar diferentes actividades:

- durante el estado;
- al traspasar la transición;
- en la entrada y en la salida del estado;
- dentro del estado, al recibir un evento.

Una actividad es una serie de acciones. Una acción consiste en asignar un valor a un atributo, crear o destruir un objeto, efectuar una operación, enviar una señal a otro objeto o a sí mismo, etc. El otro objeto se designará con su nombre, al igual que que en los diagramas de interacción estudiados en el capítulo Modelado de la dinámica.

La figura 8.10 muestra la representación gráfica de las diferentes posibilidades. Las actividades precedidas de la palabra clave *entry/* se ejecutan a la entrada de un estado. Las actividades precedidas del nombre de un evento se ejecutan si se ha recibido el evento. La palabra clave *do/* introduce la actividad realizada durante el estado. Las actividades precedidas de la palabra clave *exit/* se ejecutan en la salida del estado. El envío de una señal se precede de un signo *^* seguido del nombre de la señal.

También es posible especificar la actividad durante el traspaso de la transición (que deberá precederse del signo */*) y después del evento y de la condición de guarda, en caso de que existan.

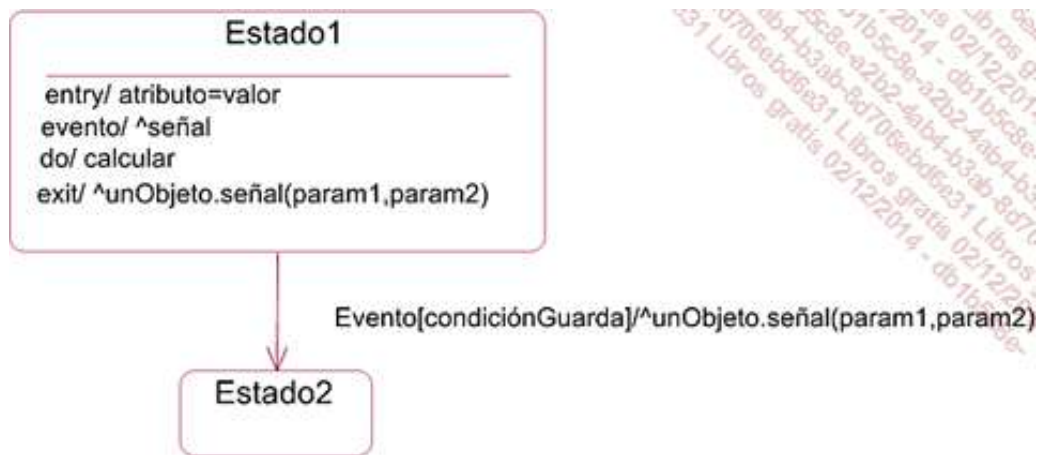


Figura 8.10 - Actividades ejecutadas durante un estado o al traspasar una transición

#### Ejemplo

La figura 8.11 muestra la utilización de las actividades dentro de un estado o bien al traspasar una transición. Esto permite administrar el valor de los atributos númeroPuntosPenalización y númeroIntentos de la clase Participante. El número de puntos de penalización aumenta si se derriba el muro, hecho que se traduce en la recepción del evento derribo durante el estado SaltoMuro. El número de intentos se inicia en uno y va aumentando con cada negativa a saltar durante la transición correspondiente.

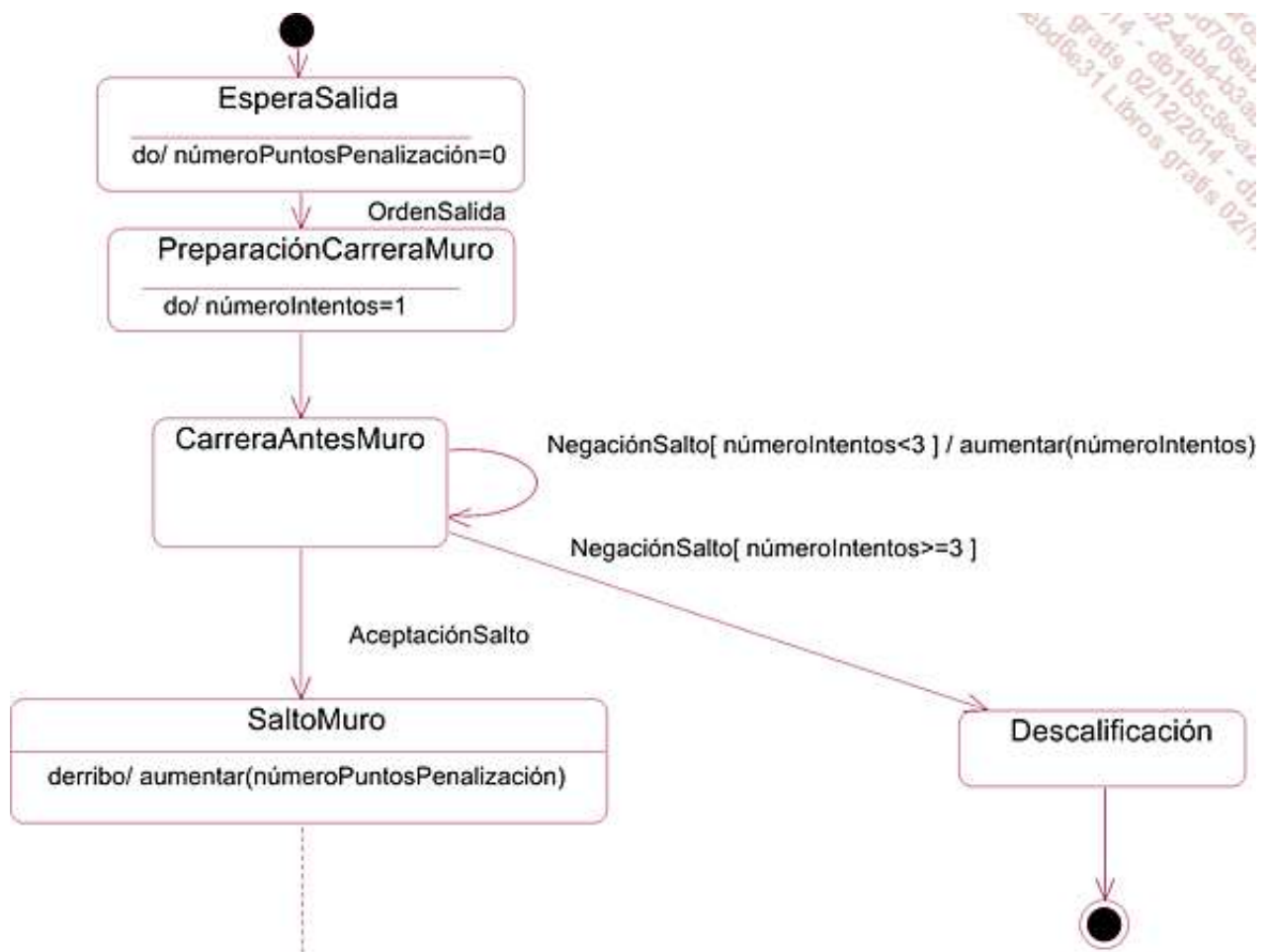


Figura 8.11 - Ejemplo de actividades dentro de un estado o al traspasar una transición

## 4. Estados compuestos

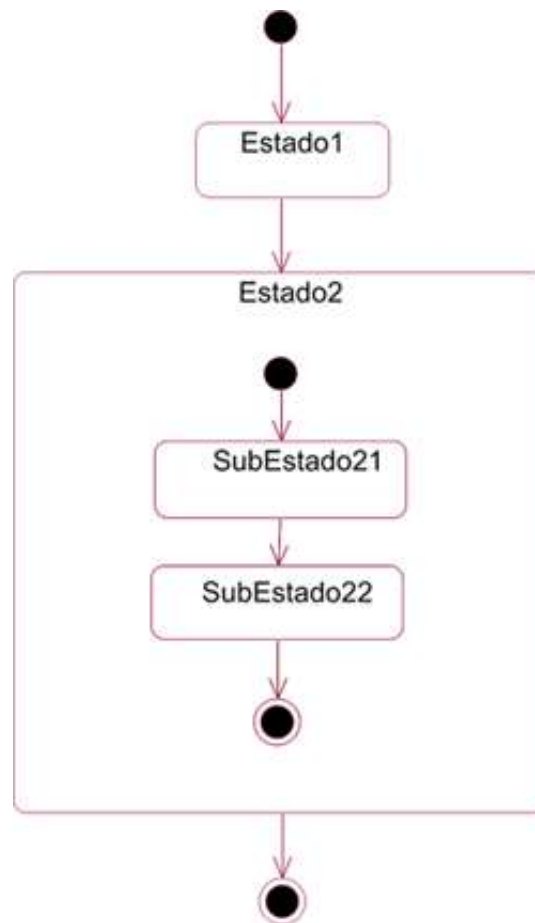
El propio estado puede ser descrito con un diagrama de estados-transiciones. Este tipo de estados se



conoce como estados compuestos. Los estados que lo componen reciben el nombre de subestados.

El principio es simple: cuando el objeto pasa al estado compuesto, pasa también al subestado inicial del diagrama interno de estados-transiciones. Si el objeto traspasa una transición que le hace salir del estado compuesto, sale a su vez de los subestados.

La figura 8.12 muestra un estado compuesto. Un diagrama interno de estados-transiciones puede tener uno o varios estados finales o no tener ninguno.



*Figura 8.12 - Estado compuesto*

Cuando un objeto abandona un estado compuesto, es posible memorizar el subestado activo con el fin de volver a él. Para ello, es preciso utilizar el subestado especial de memoria H que representa el último subestado activo memorizado en el estado compuesto.

La figura 8.13 muestra el subestado especial de memoria.

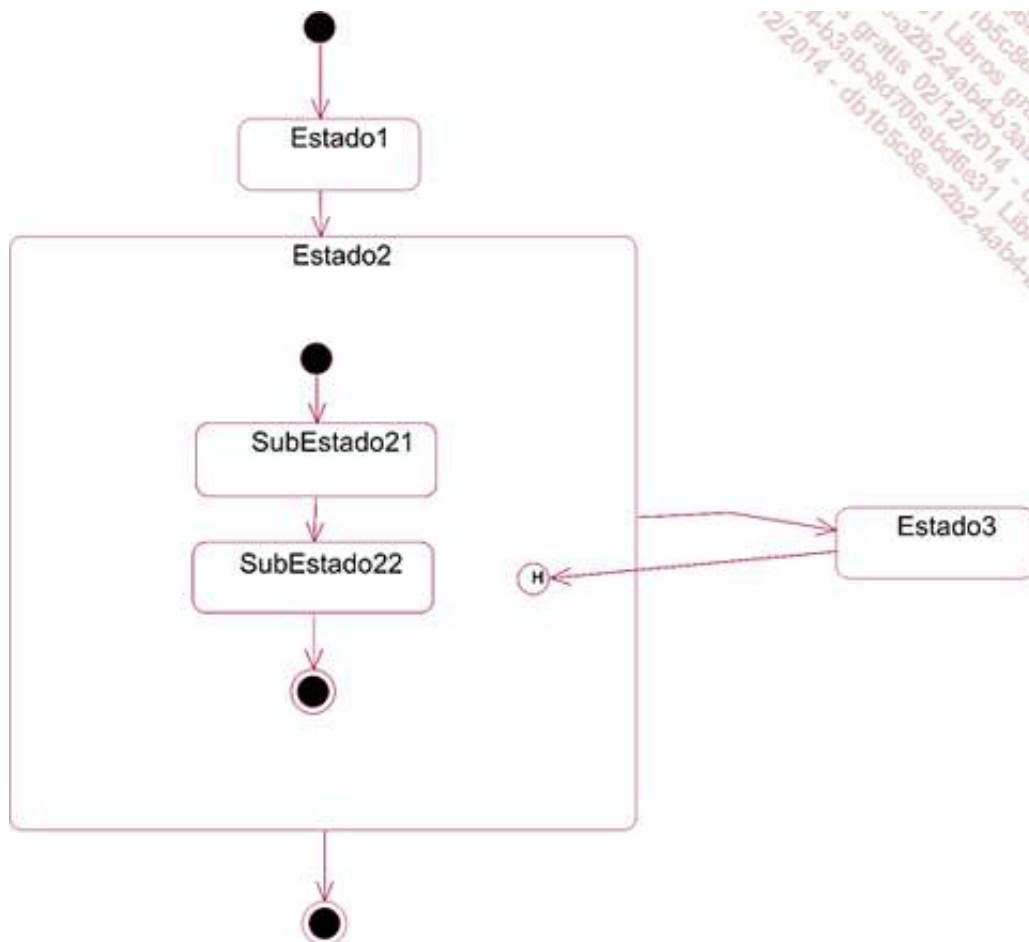


Figura 8.13 - Subestado de memoria

- Un subestado puede estar compuesto a su vez de otros subestados. En ese caso, existen dos subestados de memoria H y H\*. El primero permite volver al subestado que se encuentra en el nivel más elevado mientras que el segundo facilita el regreso al subestado anidado.

### Ejemplo

Una vez dada la orden de salida, y hasta el último salto, los participantes se encuentran en el estado Concurso. Pueden ser descalificados en cualquier momento, pero la descalificación deberá ser confirmada (en caso de polémica, por ejemplo). Si se anula, la prueba vuelve a iniciarse en el estado en el que quedó en el momento de su interrupción.

La figura 8.14 muestra dicho funcionamiento utilizando un subestado de memoria para volver al último subestado del concurso en caso de anularse una descalificación.

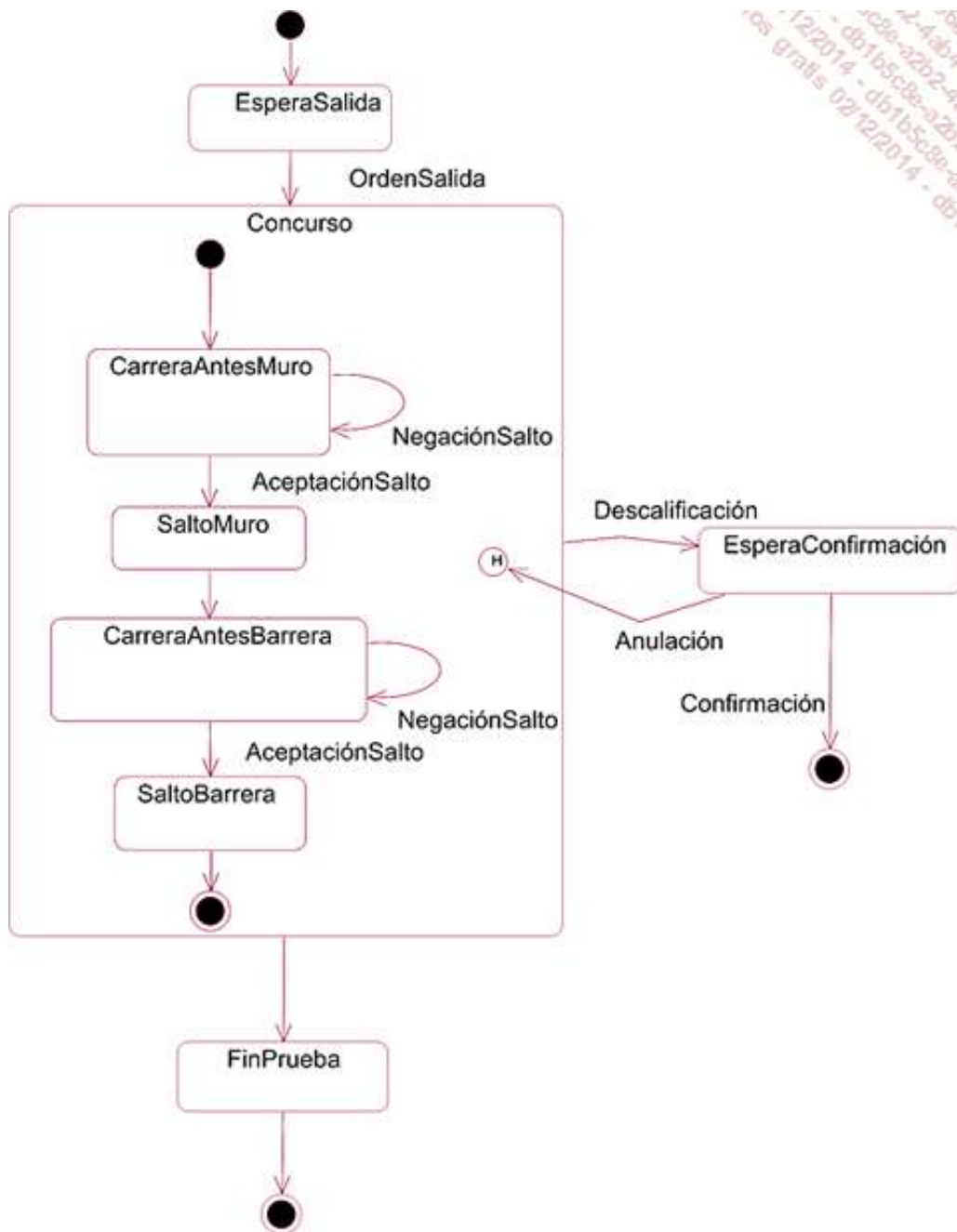


Figura 8.14 - Ejemplo de subestados de memoria

Dentro de un objeto compuesto, es posible encontrar subestados que evolucionen en paralelo. Para ello, existe una transición de tipo *horquilla* con varios subestados finales. Una vez franqueada, el objeto se encuentra en todos los subestados iniciales.

La transición de tipo *sincronización* posee varios subestados iniciales y un único estado final. Es preciso que el objeto se encuentre en todos los subestados iniciales para que se traspase la transición.

La figura 8.15 muestra la representación de ambos tipos de transición.

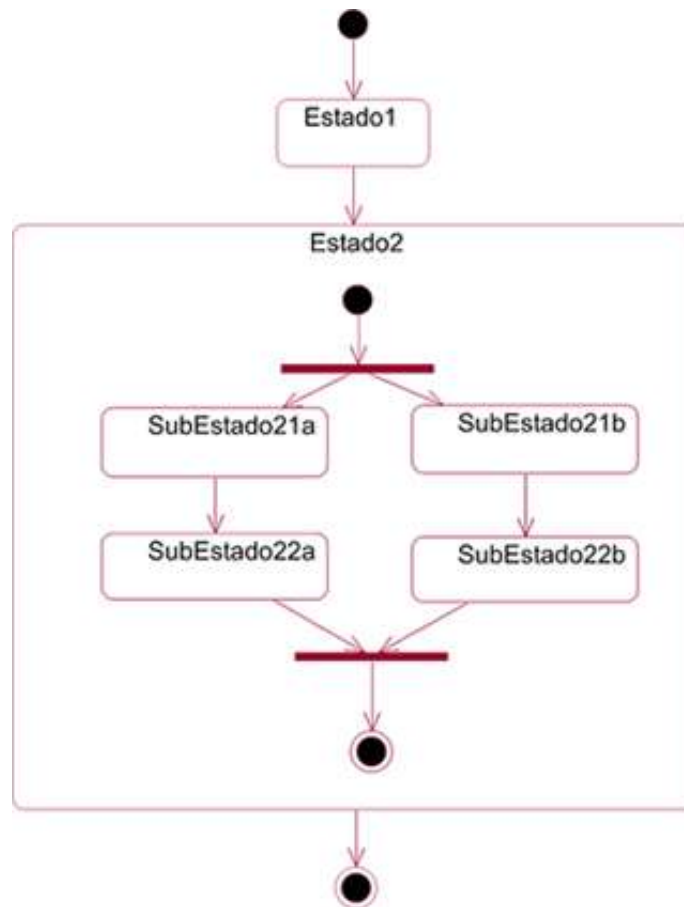


Figura 8.15 - Subestados paralelos y transiciones de horquilla y de sincronización

### Ejemplo

Un salto se puede descomponer en varios subestados, diferentes para el caballo y para el jinete, pero que tienen lugar simultáneamente. Presentamos esta situación en la figura 8.16. En el estado SaltoMuro, una transición de horquilla permite distinguir los subestados del caballo y del jinete. Al principio, el caballo se encuentra en el subestado Aproximación, después se levanta y se sitúa en el subestado Elevación y por último pasa al subestado Planeo. El jinete permanece en el subestado PosiciónSalto durante el salto. Al final del salto, se traspasa una transición de sincronización.

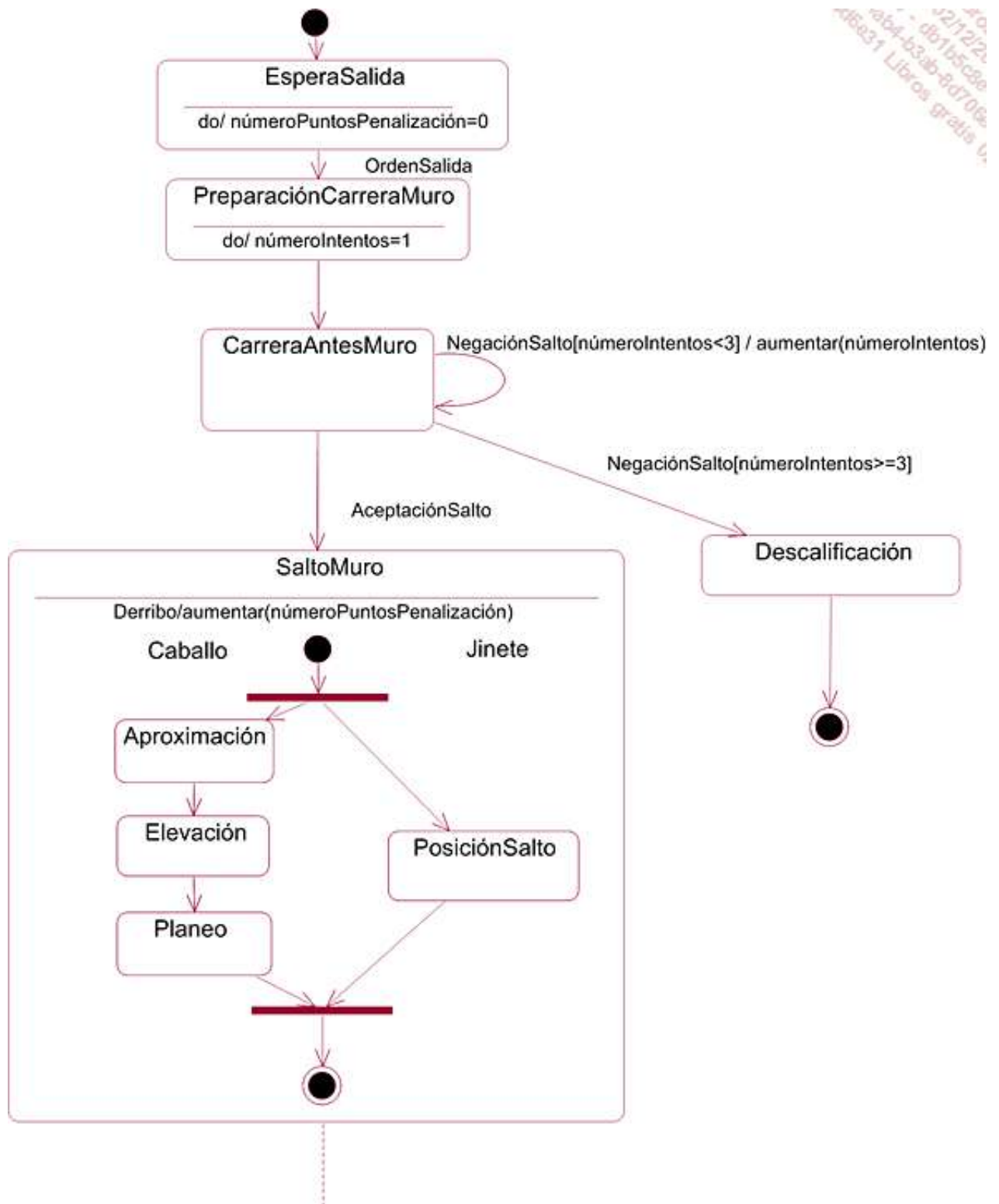


Figura 8.16 - Ejemplo de subestados paralelos

## El diagrama de timing

El diagrama de timing se introdujo en UML 2 para mostrar los cambios de estado de un objeto cuando éstos dependen exclusivamente del tiempo. El diagrama indica entonces la duración mínima y máxima de cada estado con ayuda de especificaciones temporales.

La figura 8.17 muestra la representación gráfica del diagrama de timing.

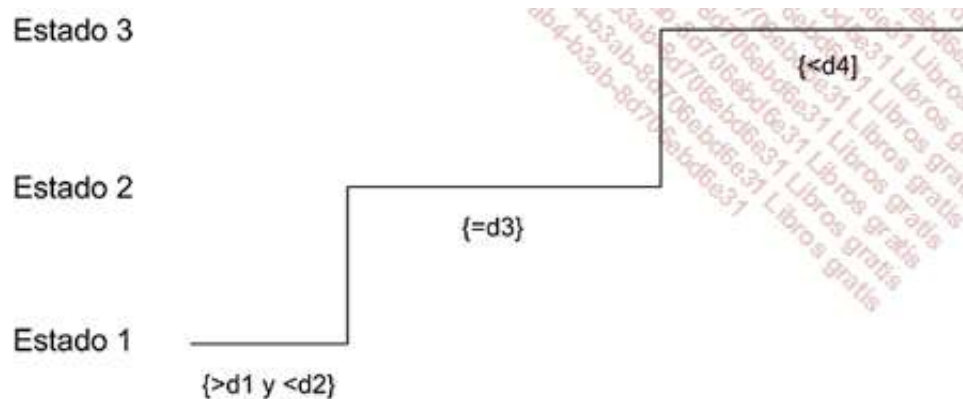


Figura 8.17 - Diagrama de timing

### Ejemplo

En un concurso de obstáculos, el jinete debe establecer un tiempo máximo para realizar la totalidad de la prueba, de lo contrario quedará eliminado. Él mismo descompone el tiempo de cada prueba para estar seguro de superarla con éxito.

La figura 8.18 muestra el correspondiente diagrama de timing.

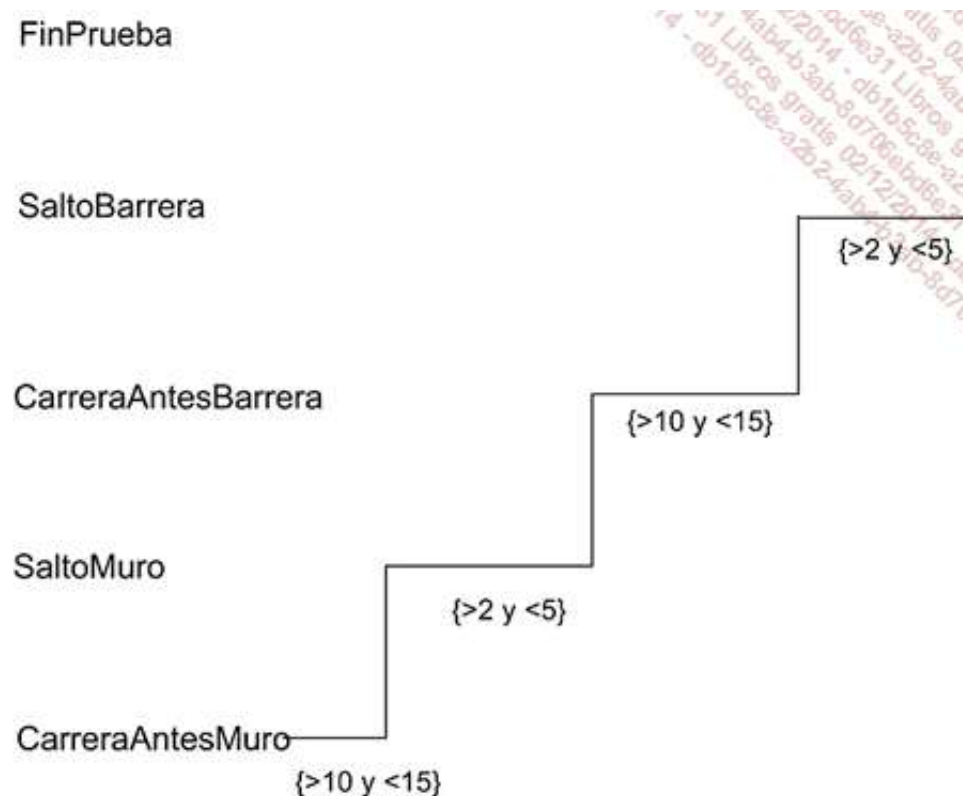


Figura 8.18 - Ejemplo de diagrama de timing

## Conclusión

El diagrama de estados-transiciones describe el ciclo de vida de los objetos encargados de asegurar la dinámica del sistema. La descripción del ciclo de vida se realiza de forma separada para cada uno de los objetos.

Este modelado es muy importante para asegurar que los objetos respondan a las interacciones descritas en los diagramas de secuencia y comunicación estudiados en el capítulo Modelado de la dinámica.

# Ejercicios

## 1. El ticket de apuesta trifecta

¿Por qué estados puede pasar un ticket de apuesta trifecta?

Construya el diagrama de estados-transiciones de una instancia de la clase *Ticket*.

## 2. La carrera de caballos

¿Por qué estados puede pasar una carrera de caballos?

Construya el diagrama de estados-transiciones de una instancia de la clase *Carrera*.

## 3. El tiovivo de madera

Describa los diferentes estados posibles de un tiovivo de caballos de madera y construya su correspondiente diagrama de estados-transiciones.



# Introducción

El diagrama de actividades está basado en el diagrama de estados-transiciones, estudiado en el capítulo precedente. Se trata de una forma específica del diagrama de estados-transiciones en la que cada estado se asocia a una actividad y todas las transiciones son automáticas. En este tipo de diagramas, las transiciones reciben el nombre de encadenamientos.

Posteriormente, el diagrama de actividades se amplió para describir las actividades de varios objetos. De esta forma, se pueden representar los encadenamientos entre las actividades de diferentes objetos, cosa que no es posible con el diagrama de estados-transiciones. Veremos cómo designar el objeto responsable de cada actividad con ayuda de la noción de tramo.

El diagrama de actividades ofrece alternativas gracias a las condiciones de guarda. Puede asimismo contener encadenamientos de tipo horquilla y sincronización para administrar actividades paralelas.

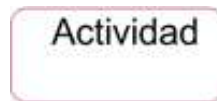
Presentaremos el diagrama de vista de conjunto de las interacciones propio de UML 2.

# Las actividades y los encadenamientos de actividades

## 1. Las actividades

Una actividad es una serie de acciones. Una acción consiste en asignar un valor a un atributo, crear o destruir un objeto, efectuar una operación, enviar una señal a otro objeto o a uno mismo, etc.

La figura 9.1 muestra la representación gráfica de una actividad.



*Figura 9.1 - Representación gráfica de una actividad*

### Ejemplo

Retomamos el ejemplo del capítulo Modelado de los requisitos referente a la compra de una yegua. Tanto la elección de la yegua como la comprobación de las vacunas son ejemplos de actividad.

La actividad inicial es la primera en ejecutarse y se representa con un punto negro (ver figura 9.2).



*Figura 9.2 - Representación gráfica de la actividad inicial*

La actividad final representa el término de la ejecución de las actividades de un diagrama. No tiene porqué ser única y tampoco es obligatoria.

La actividad final se representa con un punto negro rodeado de un círculo (ver figura 9.3).



*Figura 9.3 - Representación gráfica de una actividad final*

## 2. Los encadenamientos de actividades

Un encadenamiento de actividades es un vínculo orientado entre dos actividades. Puede traspasarse cuando concluye la actividad inicial, lo que conduce a la activación de la actividad final.

Puede ser simple, es decir, vincular sólo dos actividades. La figura 9.4 muestra su representación gráfica.



*Figura 9.4 - Representación gráfica de un encadenamiento de actividades*

Un encadenamiento de actividades puede ser también una alternativa. Cada rama de la alternativa está dotada de una condición de guarda que excluye el resto de condiciones. Le recordamos que estudiamos el tema de las condiciones de guarda en el anterior capítulo.

La representación gráfica de la alternativa se muestra en la figura 9.5.

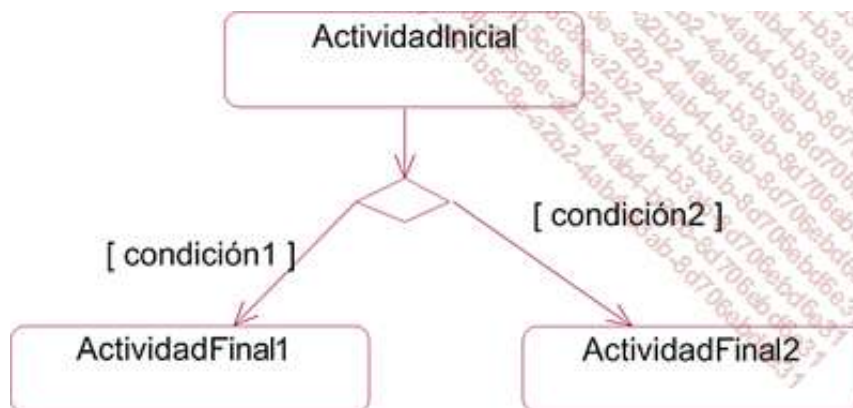


Figura 9.5 - Representación gráfica de la alternativa

Un encadenamiento de actividades de tipo *horquilla* posee también varias actividades de destino. Al traspasarlo, todas las actividades de destino se activan en paralelo.

La representación gráfica del encadenamiento de tipo *horquilla* se presenta en la figura 9.6.

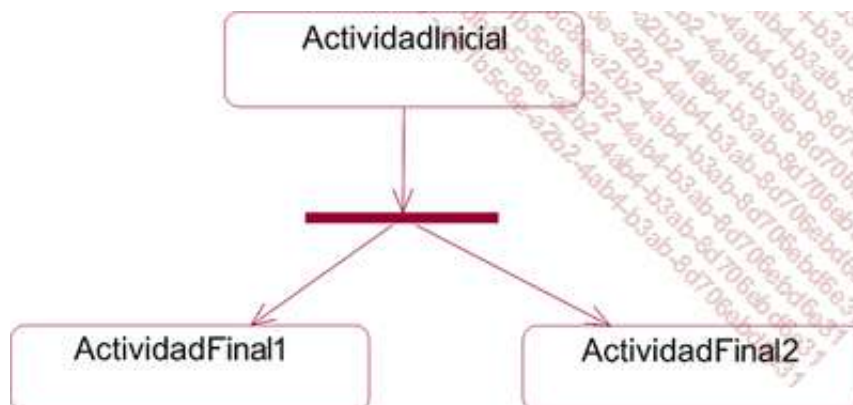


Figura 9.6 - Representación gráfica del encadenamiento de tipo horquilla

Un encadenamiento de actividades de tipo *sincronización* posee varias actividades iniciales y una sola actividad final. Es preciso que todas las actividades iniciales estén terminadas para que el encadenamiento se traspase y se inicie la actividad final.

El encadenamiento de tipo *sincronización* se muestra en la figura 9.7.

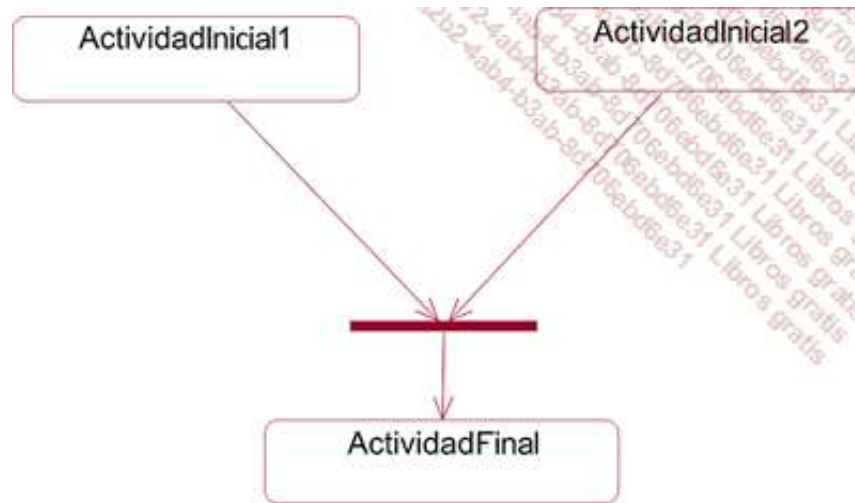


Figura 9.7 - Representación gráfica del encadenamiento de tipo sincronización

#### Ejemplo

Retomamos el ejemplo de compra de una yegua. El diagrama de actividades correspondiente se describe en la figura 9.8.

La gestión de los papeles y el traslado de la yegua se tratan en paralelo, ya que el comprador puede muy bien realizar ambas actividades al mismo tiempo.

Las condiciones de guarda expresan las diferentes alternativas. Es conveniente anotar dos actividades finales, una correspondiente a la renuncia a la compra y otra correspondiente a una compra exitosa.

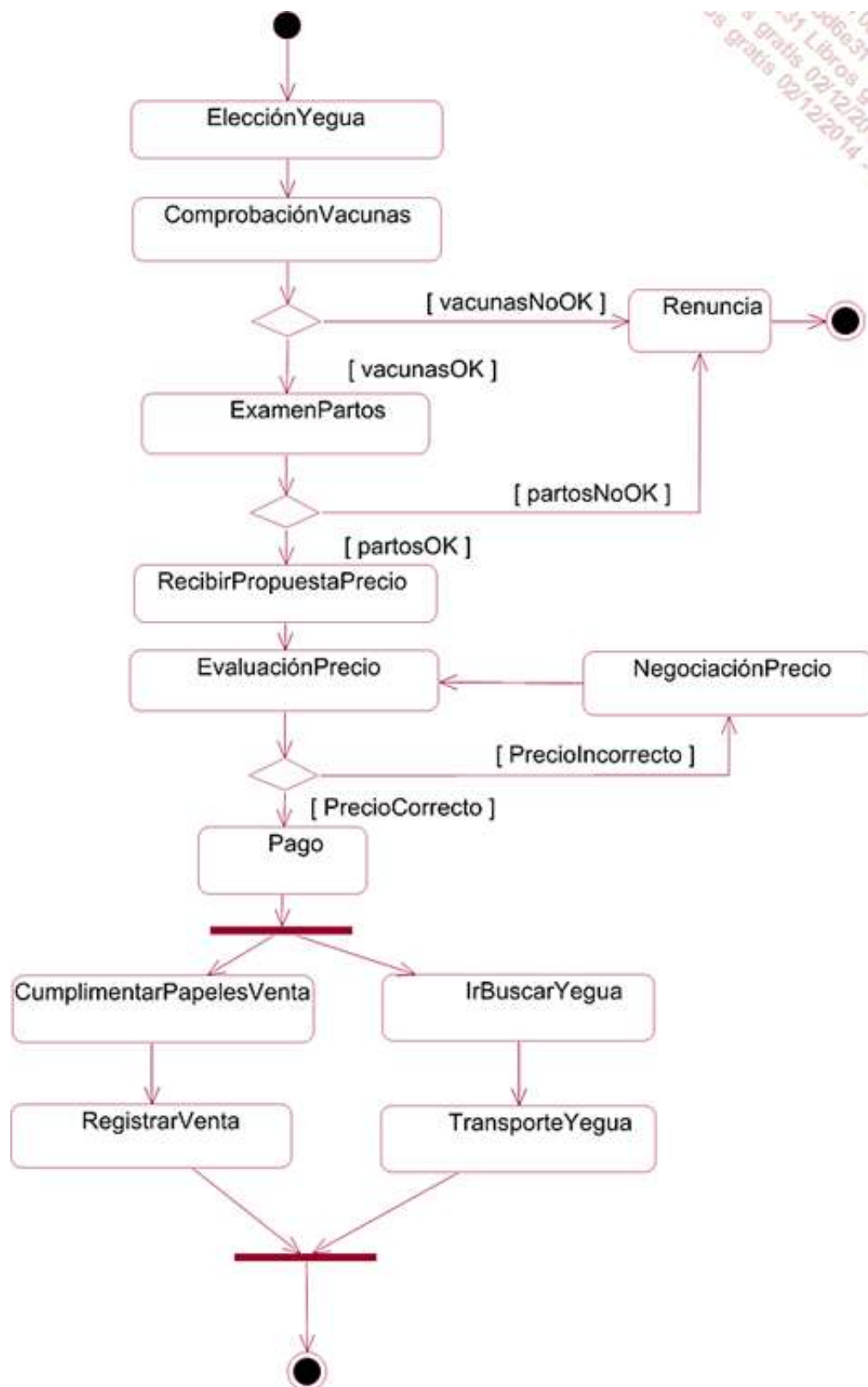


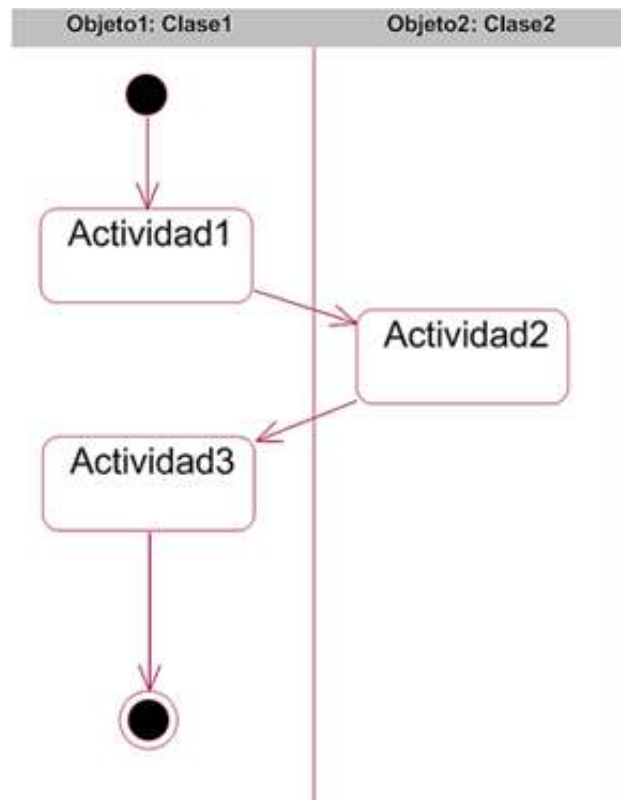
Figura 9.8 - Ejemplo de diagrama de actividades

## Las particiones o calles

A diferencia del diagrama de estados-transiciones, el diagrama de actividades puede representar las actividades realizadas por varios objetos con sus encadenamientos.

Para ello el diagrama se divide en particiones o calles. A cada calle corresponde el objeto responsable de la realización de todas las actividades contenidas en esa calle o partición.

La figura 9.9 muestra la representación gráfica de las calles. Un encadenamiento puede cortar la línea de separación de dos calles para mostrar un cambio de objeto entre la actividad inicial y la final.



*Figura 9.9 - Las calles de un diagrama de actividades*

### Ejemplo

La figura 9.10 representa el ejemplo de compra de una yegua en el que se describen las actividades relativas al comprador y a la granja de cría.

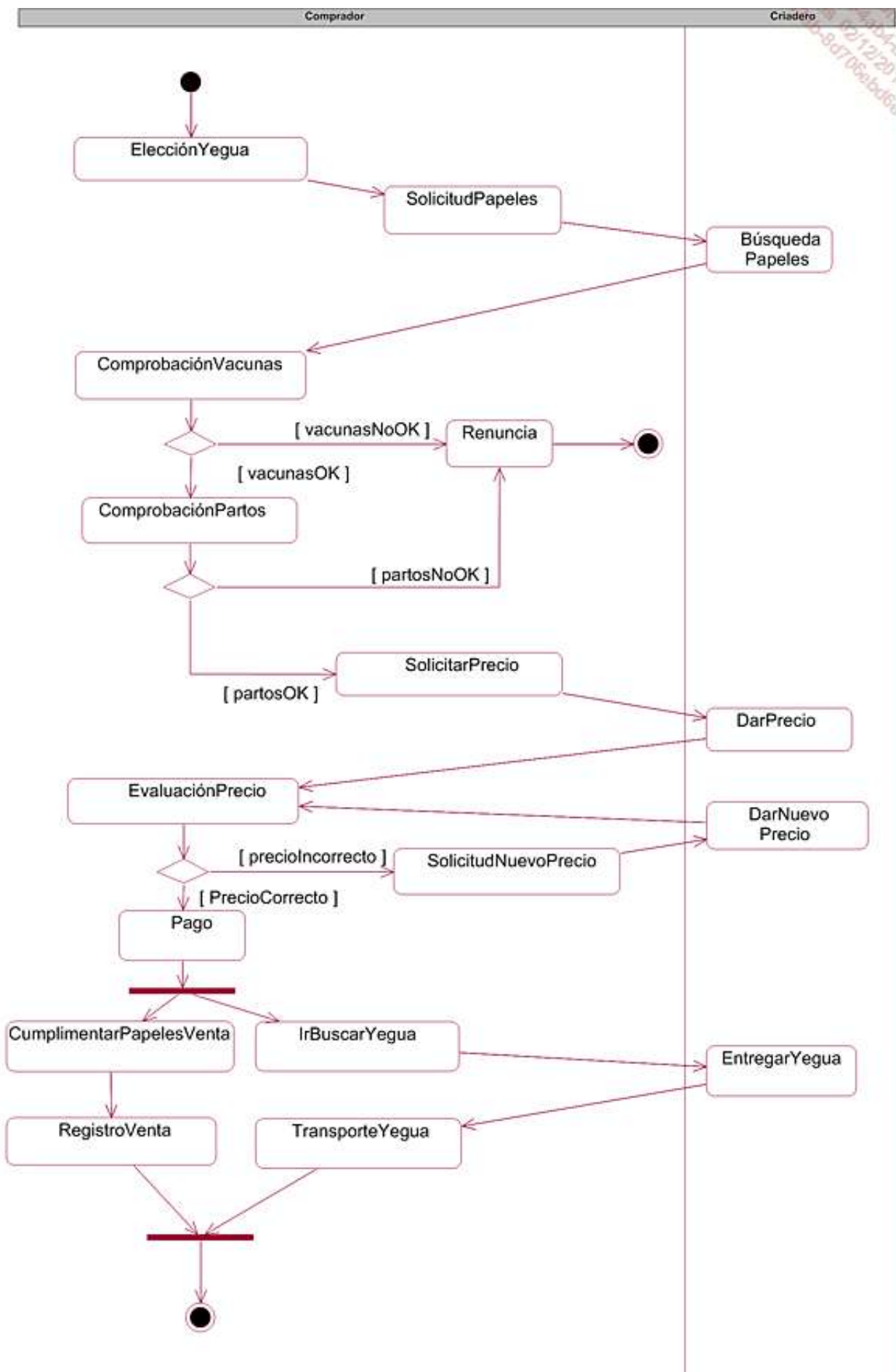


Figura 9.10 - Ejemplo de diagrama de actividades dividido en calles

## Las actividades compuestas

Una actividad puede estar compuesta de otras actividades. Cuando eso ocurre, un diagrama de actividades específico describe su composición en subactividades. Las actividades compuestas se representan en los diagramas en los que están presentes mediante un símbolo de horquilla.

La figura 9.11 muestra la composición de una actividad en subactividades. La figura 9.12 presenta la actividad sin la composición, tal y como puede utilizarse dentro de un diagrama de actividades.

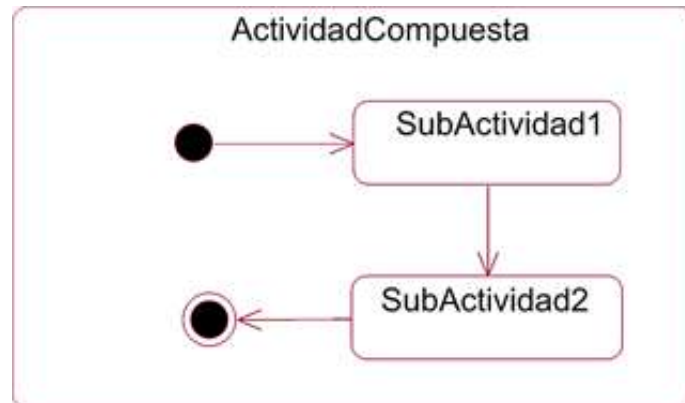


Figura 9.11 - Composición de una actividad en subactividades

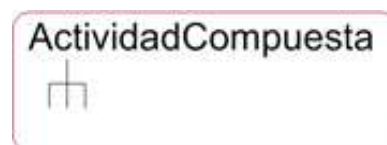


Figura 9.12 - Representación de una actividad compuesta

### Ejemplo

La figura 9.13 representa la gestión del pago de una yegua integrando la negociación del precio. La gestión se incluye en el diagrama general de compra de la figura 9.14 en tanto que actividad compuesta y se representa con una horquilla.

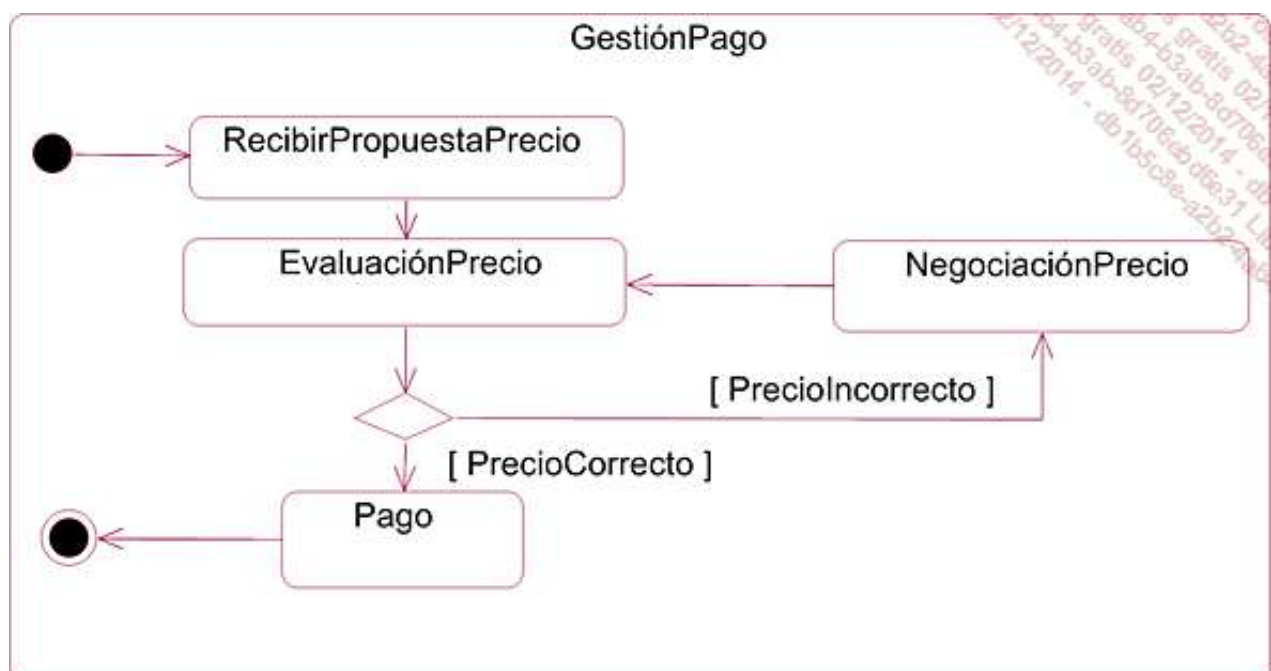


Figura 9.13 - Ejemplo de actividad compuesta



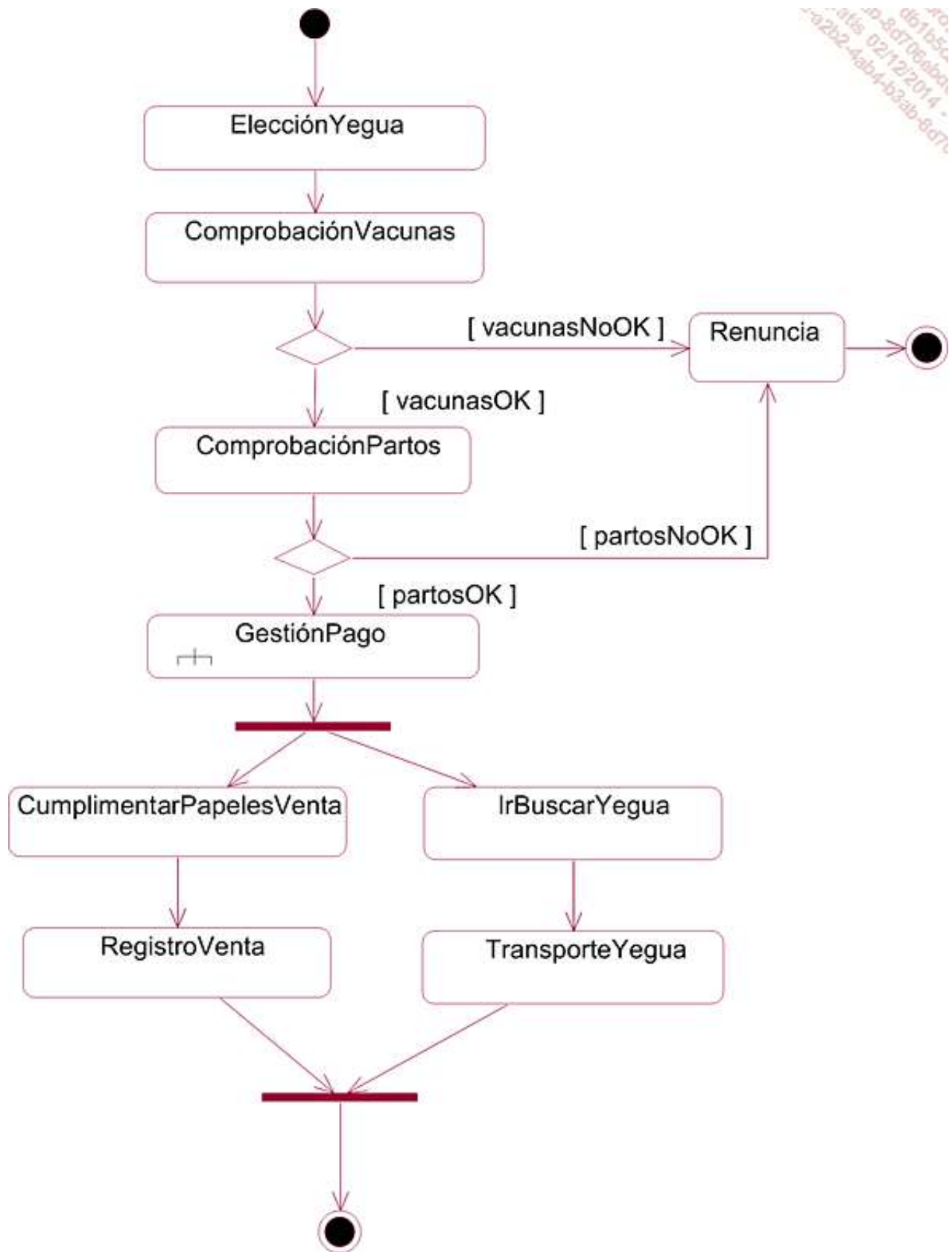


Figura 9.14 - Ejemplo de inclusión de una actividad compuesta

## El diagrama de vista de conjunto de las interacciones

El diagrama de vista de conjunto de las interacciones es específico de UML 2. Se trata de un diagrama de actividades en el que éstas pueden describirse mediante diagramas de secuencia. La figura 9.15 muestra la representación gráfica de ese tipo de diagramas.

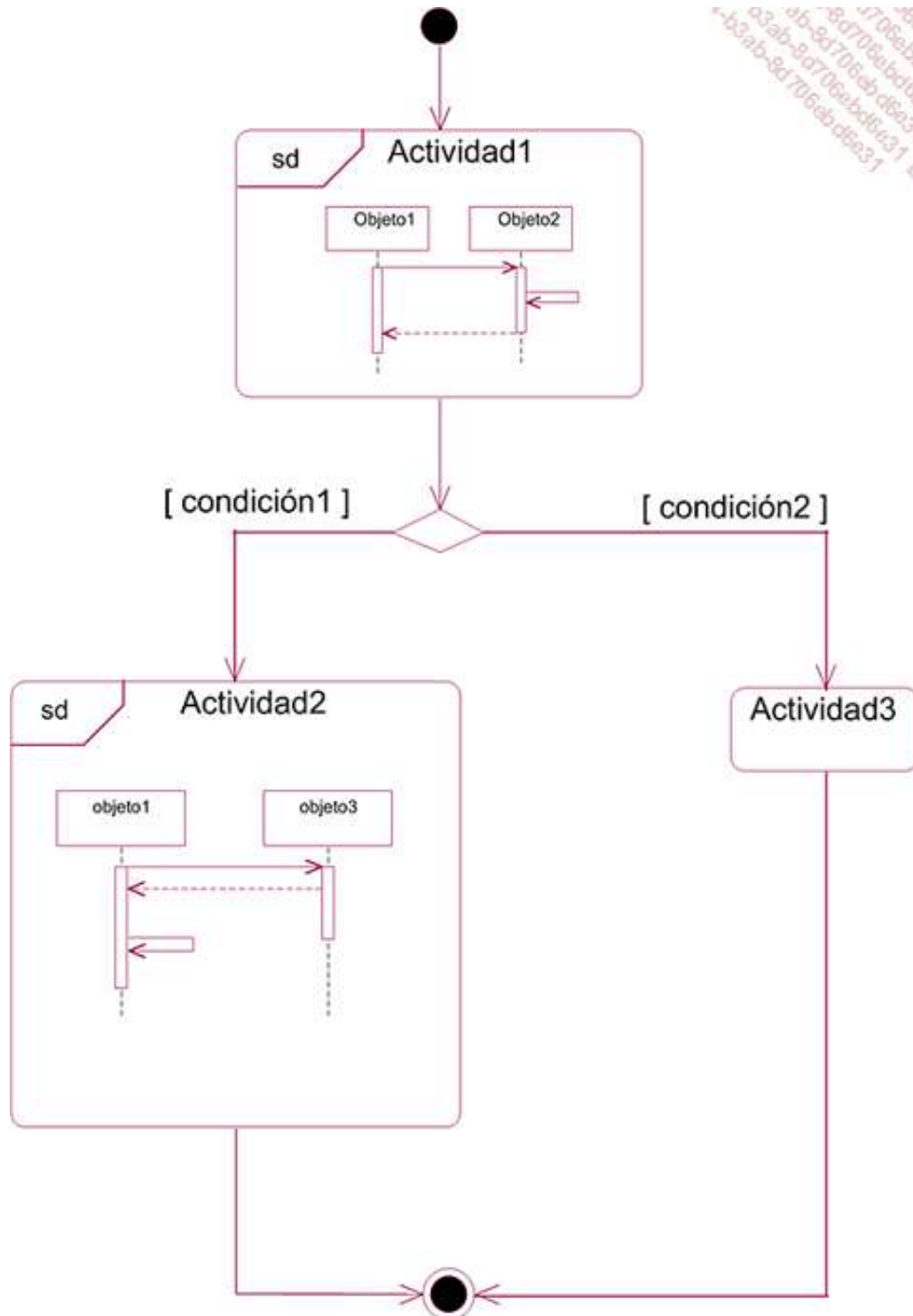


Figura 9.15 - Diagrama de vista de conjunto de las interacciones

## Conclusión

El diagrama de actividades representa las actividades realizadas por uno o varios objetos. Puede corresponder a la descripción detallada de una actividad del diagrama de estados-transiciones o a la descripción de un método. También puede describir la actividad de un sistema o subsistema asignando las responsabilidades a los actores. El diagrama de actividades constituye también una buena opción para describir casos de uso.

# Ejercicios

## 1. El espectáculo ecuestre

Construya el diagrama de actividades de compra de una entrada para un espectáculo ecuestre.

## 2. La apuesta trifecta

Construya el diagrama de actividades de comprobación de la caja de una taquilla de apuestas de trifecta (sólo la parte referente a la venta de boletos, sin tener en cuenta el pago de ganancias).

# Introducción

En el presente capítulo, abordaremos las posibilidades de UML para modelar la arquitectura del sistema. Dicho modelado presenta dos aspectos:

- El modelado de la arquitectura del software y su estructuración en componentes.
- El modelado de la arquitectura material y la repartición física de los programas.

Estudiaremos la noción de componente de software. Un componente es una caja negra que ofrece servicios de software. Los servicios son descritos por una o varias interfaces del componente.

En el capítulo Modelado de objetos estudiamos la noción de interfaz que, como veremos, se aplica también a los componentes.

Recordemos que una interfaz es una clase abstracta que sólo contiene las firmas de método. La firma de un método se compone de su nombre y sus parámetros.

Los componentes también pueden depender de otros componentes para llevar a cabo los servicios que ofrecen. Esta dependencia se expresa en forma de una interfaz necesaria que describe los servicios deseados.

El modelado de los componentes y sus relaciones se describe mediante el diagrama de componentes.

El modelado de la arquitectura material describe los nodos y sus vínculos e incluye la localización de los elementos de software dentro de los nodos en su forma física, llamada *artefact*. La descripción se efectúa mediante el diagrama de despliegue.

# El diagrama de componentes

## 1. Los componentes

Un componente es una unidad de software que ofrece una serie de servicios a través de una o varias interfaces. Se trata de una caja negra cuyo contenido queda fuera del interés de los clientes. Está completamente encapsulado. La definición de los componentes recuerda a la definición de clases que implantan una o varias interfaces, como vimos en el capítulo Modelado de objetos. Una clase que implanta una o varias interfaces es un componente. Por el contrario, un componente no es necesariamente una clase. Las interfaces pueden implantarse dentro de los componentes con lenguajes de programación no orientados a objetos, como puede ser el lenguaje C.

La tecnología es otro de los aspectos de los componentes. Hoy en día existen muchas tecnologías de componentes. Una tecnología de componentes define, entre otras cosas, el lenguaje de programación de los clientes, el entorno de ejecución y la integración en la plataforma de software subyacente (Windows, Java, etc.). Los componentes que usan una tecnología se benefician de un estándar y, por tanto, se convierten en comercializables: podemos encontrarlos en los estantes de las tiendas.

Existen varias tecnologías de componentes:

- Los componentes COM y .NET.
- Los componentes Java: JavaBeans y Enterprise JavaBeans.

➤ En UML 1, la noción de componente era muy amplia: los archivos ejecutables, las bibliotecas compartidas o los scripts se consideraban componentes. En UML 2, la definición se ha reducido a los componentes que ofrecen servicios a través de una o varias interfaces.

Los componentes pueden depender de otros componentes para realizar operaciones internas. En ese caso se convierten en clientes de esos otros componentes. Como cualquier cliente de un componente, no conocen su estructura interna. Por tanto, sólo dependen de las interfaces de los componentes de los cuales son clientes. Estas interfaces reciben el nombre de **interfaces necesarias** del componente *cliente*. Las interfaces que describen los servicios ofrecidos por un componente se denominan **interfaces suministradas**.

La notación gráfica de una interfaz suministrada es idéntica a la que representamos en el capítulo Modelado de objetos. Las interfaces necesarias se representan mediante un semicírculo y los componentes dentro de un rectángulo con el estereotipo «component». Este estereotipo puede ser reemplazado por el logo del componente. La figura 10.1 muestra la representación gráfica de un componente en dos formas, una con el estereotipo y otra con el logo.

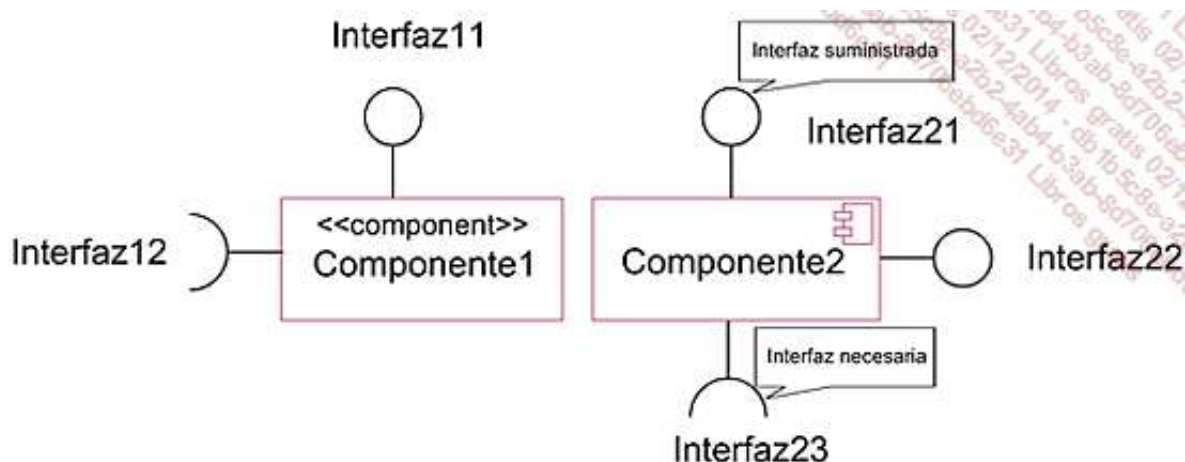


Figura 10.1 - Representación gráfica de un componente y sus interfaces

También es posible usar la relación de realización para las interfaces suministradas y la relación de dependencia para las interfaces necesarias. En la figura 10.2, vemos esta representación alternativa, que presenta la ventaja de detallar las firmas de método contenidas en las interfaces.

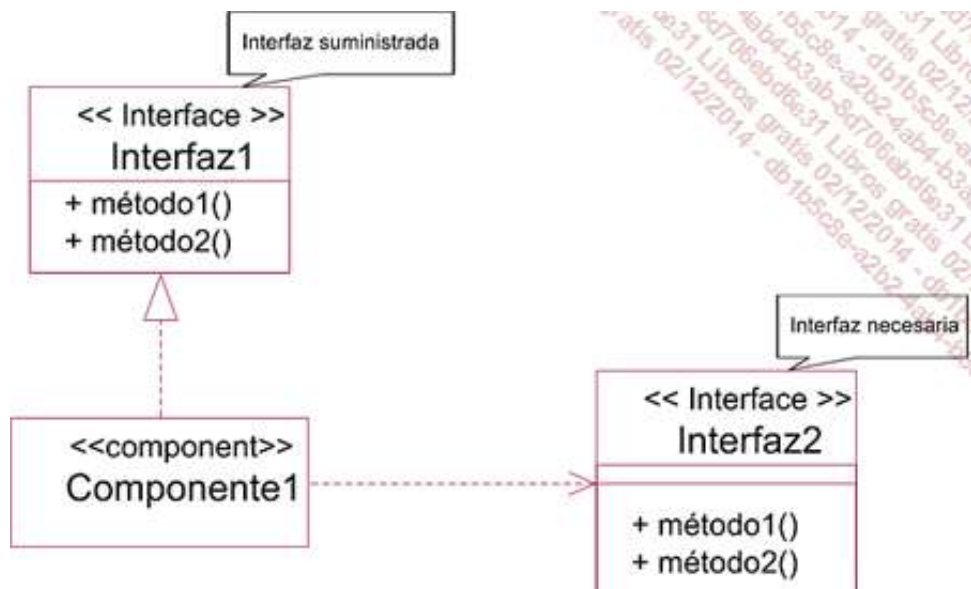


Figura 10.2 - Representación alternativa de las interfaces

Ejemplo

Un componente de gestión de un criadero de caballos suministra una interfaz para gestionar los caballos y una interfaz para gestionar las ventas. Además, requiere un componente de la base de datos. El componente se muestra en la figura 10.3.

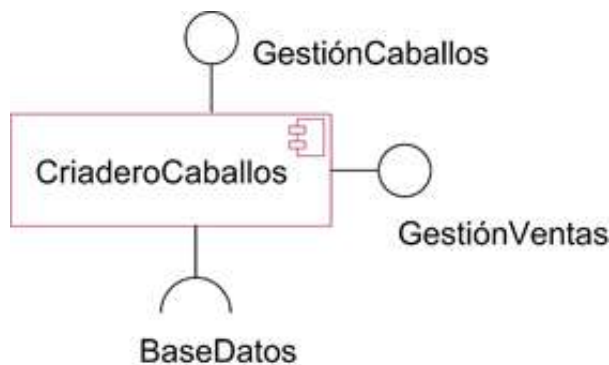


Figura 10.3 - Ejemplo de componente

2. La arquitectura del software por componentes

En la orientación a objetos, la arquitectura del software de un sistema está construida por un compendio de componentes vinculados por interfaces suministradas e interfaces necesarias. El diagrama de componentes describe esta arquitectura.

Ejemplo

El sistema de información de un criadero de caballos está formado por un compendio de componentes. La figura 10.4 muestra el diagrama de componentes de dicho sistema. Los componentes *VentanaGestiónCaballos* y *VentanaGestiónVentas* administran en la pantalla una ventana dedicada a la gestión de los caballos y otra dedicada a la gestión de las ventas de caballos, respectivamente. El componente *SistemaBaseDatos*, como su nombre indica, es un sistema de base de datos.

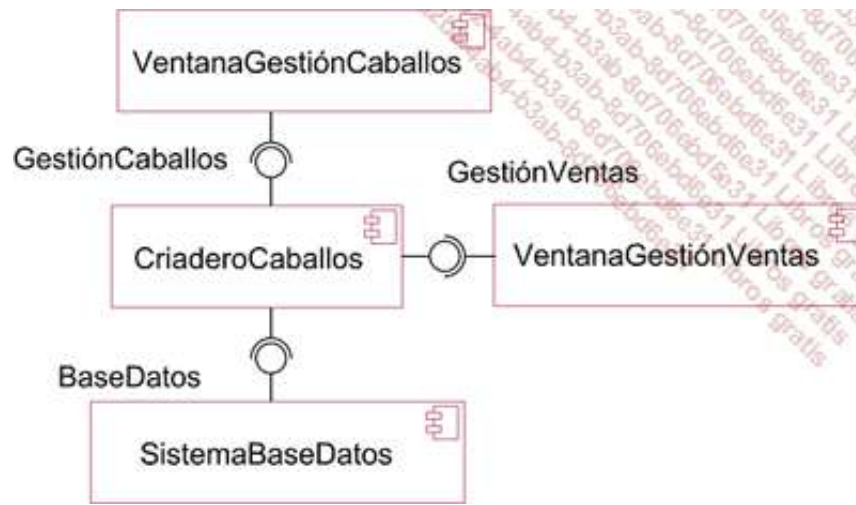


Figura 10.4 - Ejemplo de diagrama de componentes



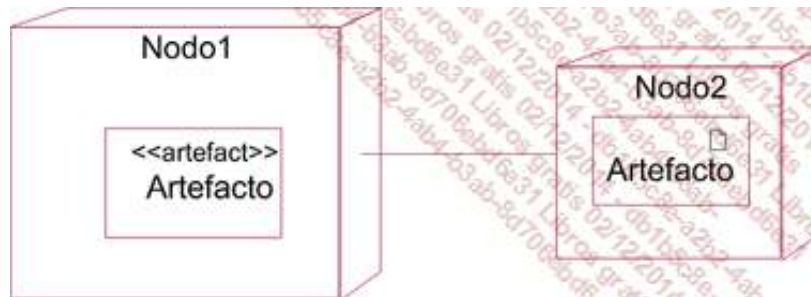
## El diagrama de despliegue

El diagrama de despliegue describe la arquitectura física del sistema. Está compuesto de nodos. Un nodo es una unidad material capaz de recibir y de ejecutar software. La mayoría de nodos son ordenadores. Los vínculos físicos entre nodos también pueden describirse en el diagrama de despliegue, corresponden a las ramas de la red.

Los nodos contienen software en su forma física, conocida como artefacto. Los archivos ejecutables, las bibliotecas compartidas y los scripts son ejemplos de formas físicas de software.

Los componentes que constituyen la arquitectura del software del sistema se representan en el diagrama de despliegue mediante un artefacto que, con frecuencia, es un ejecutable o una biblioteca compartida.

La representación gráfica de los nodos, sus vínculos y los artefactos que contienen se muestra en la figura 10.5.

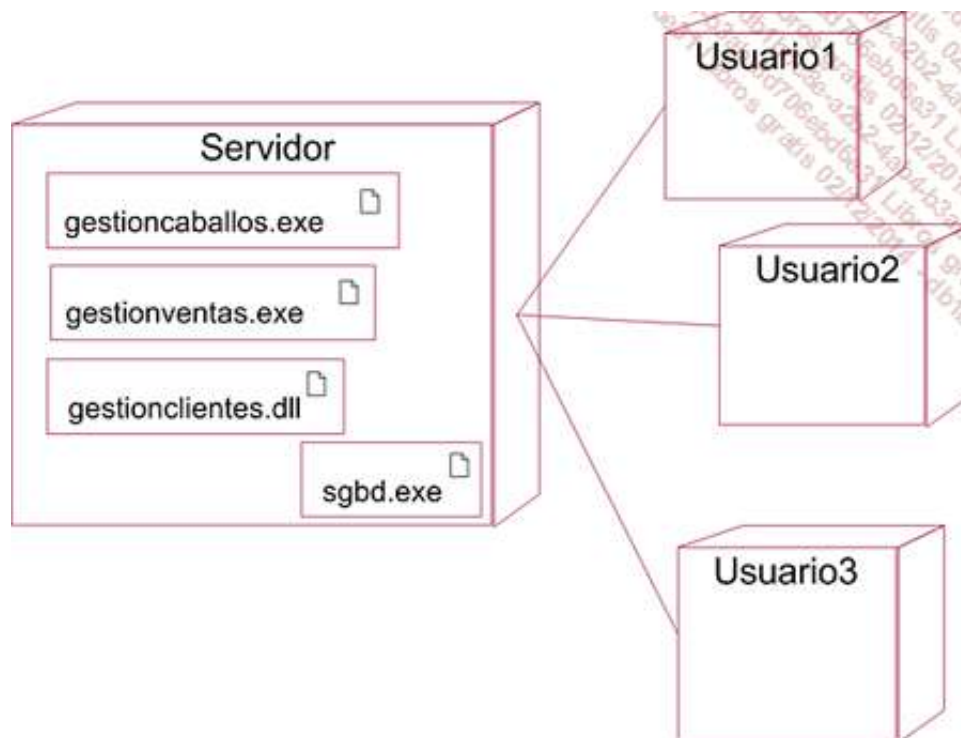


*Figura 10.5 - Representación gráfica de los nodos, sus vínculos y artefactos*

### Ejemplo

La figura 10.6 muestra la arquitectura material del sistema de información de un criadero de caballos. Esta arquitectura está basada en un servidor y tres puestos clientes conectados al servidor mediante enlaces directos. El servidor contiene varios artefactos:

- Un ejecutable (.exe), forma física del componente de gestión de la base de datos.
- Un segundo ejecutable encargado de la gestión de los caballos.
- Un tercer ejecutable encargado de la gestión de las ventas.
- Una biblioteca compartida (.dll) de gestión de las máquinas de los diferentes usuarios.



*Figura 10.6 - Ejemplo de diagrama de despliegue*

## Conclusión

El diagrama de componentes y el diagrama de despliegue poseen menos elementos que los diagramas estudiados en los capítulos precedentes. No obstante, resultan útiles para el ensamblaje y el despliegue del sistema.

# Introducción

En este capítulo estudiaremos la noción de perfil, que es un soporte para enriquecer las capacidades de modelado de UML, especialmente en lo que respecta a la semántica, con el objetivo de adaptar UML:

- A plataformas específicas como Java, .NET, EJB.
- A dominios específicos del usuario como, por ejemplo, el dominio de los équidos.

Los perfiles son soportes ligeros de extensión: introducen más construcciones en metamodelo de UML, pero no permiten modificar las ya existentes. Estas nuevas construcciones son principalmente los estereotipos, las tagged values (valores etiquetados) y las especificaciones. Las especificaciones se describen con lenguaje natural o con la ayuda del lenguaje OCL, abordado en el capítulo consagrado al modelado de objetos.

Los metamodelos contienen todas las construcciones que describen los elementos UML y, en particular, aquellos que hemos estudiado en los capítulos precedentes. Un ejemplo de estas construcciones es la clase `Class` que describe las clases o incluso la clase `Interface` que describe las interfaces. Las clases son instancias de la clase `Class`, mientras que las interfaces son instancias de la clase `Interface`. Las clases del metamodelo se conocen como metaclases.

Los perfiles se describen mediante diagramas de perfiles, que forman parte de los diagramas de estructura de UML. Introducen un tipo específico de empaquetado. Hemos tratado los empaquetados en el capítulo relativo a la estructuración de los elementos de modelado.



Para ser preciso, los perfiles no se reservan a extender únicamente el metamodelo de UML, sino cualquier metamodelo. El MOF (meta-Object Facility) es un meta-meta-modelo definido por la OMG para describir metamodelos. No obstante, ese tema sobrepasa el marco de la presente obra y nosotros nos contentaremos escribiendo perfiles que extienden el metamodelo de UML.

# Los estereotipos

## 1. Las metaclases

Los estereotipos se definen como extensiones de una metaclase. Es conveniente, por tanto, que en un primer momento estudiemos la noción de metaclase. A pesar de que las especificaciones oficiales de UML no definan explícitamente esta noción, podemos decir que una metaclase es una clase cuyas instancias son elementos de UML que a su vez poseen instancias. Citemos, a título de ejemplo, la metaclase `Class` cuyas instancias son clases, la metaclase `Interface`, cuyas instancias son interfaces y la metaclase `Association`, cuyas instancias son asociaciones entre dos clases.

Una metaclase está representada en el metamodelo como una clase dotada del estereotipo «Metaclass». La figura 11.1 muestra la representación simplificada de la metaclase `Class` en UML, es decir, sin sus atributos, métodos y asociaciones.

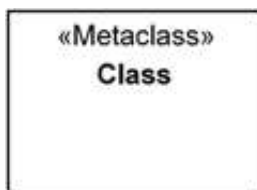


Figura 11.1 - Representación simplificada en UML de la metaclase `Class`

Las principales metaclases del metamodelo de UML son las siguientes:

Metaclass	Descripción
Association	Metaclass concreta que describe las asociaciones entre dos instancias de la metaclass <code>Class</code> .
Class	Metaclass concreta que describe las clases. <code>Class</code> es una subclase de la metaclass <code>Classifier</code> . Introduce la descripción de las operaciones y de las propiedades (llamadas atributos o roles).
Classifier	Metaclass abstracta que describe elementos que pueden ser especializados (introducción de la relación de especialización/generalización).
Element	Metaclass abstracta situada en la parte superior de la jerarquía de las metaclases del metamodelo UML.
Interface	Metaclass concreta que describe las interfaces que sólo poseen operaciones. <code>Interface</code> es una subclase de la metaclass <code>Classifier</code> .
Operation	Metaclass concreta que describe la firma de un método.
Package	Metaclass concreta que describe un conjunto de instancias de subclases concretas de la metaclass <code>Element</code> (empaquetado).
Property	Metaclass concreta que describe un atributo o un extremo de asociación (rol) con uno o varios valores caracterizados.

## 2. Las nociones de estereotipo y de asociación de extensión

### a. Las nociones de base

Los estereotipos son clases que extienden metaclases del metamodelo de UML dentro de un perfil. Introducen la terminología específica de una plataforma o de un dominio específico. Un estereotipo no puede utilizarse solo: existe únicamente si está asociado al menos a una metaclass. Esta asociación

específica se conoce como asociación de extensión y se representa mediante una flecha que parte del estereotipo y señala hacia la metaclase extendida. La punta de la flecha es un triángulo negro completo, como se ilustra en el diagrama de perfil de la figura 11.2.



Figura 11.2 - Representación en UML de un estereotipo que extiende la metaclase *Class*

En esta figura el estereotipo se representa en forma de clase llamada `UnEstereotipo` que está provista del estereotipo «estereotipo». La metaclase extendida que se encuentra en el otro extremo de la asociación de extensión es `Class`. Las cardinalidades de esta asociación son:

- 1 en el extremo situado del lado de la metaclase.
- 0..1 en el extremo situado del lado del estereotipo.

De esta forma, cada instancia del estereotipo está vinculada a una instancia de la metaclase. Cada instancia de la metaclase está vinculada o no a una instancia del estereotipo.

La figura 11.3 muestra la creación y puesta en ejecución del estereotipo «`CaballoDeTiro`».

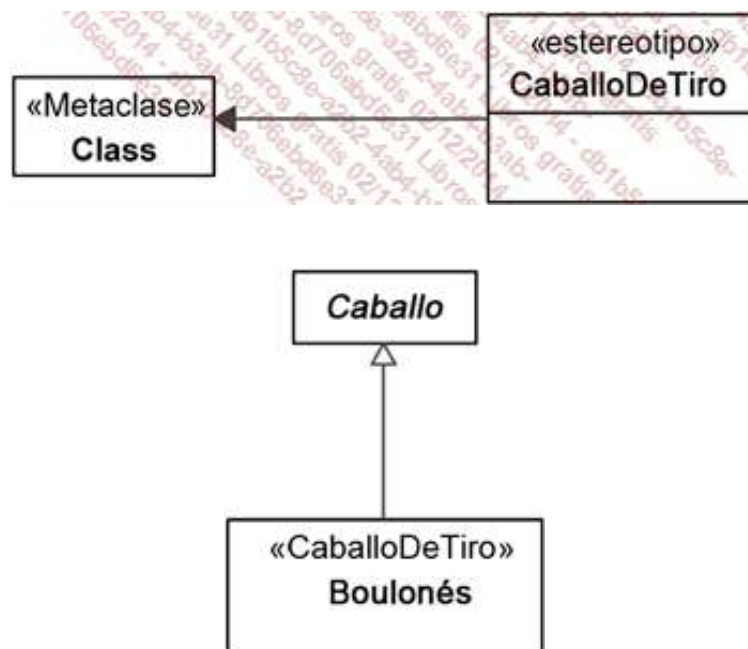


Figura 11.3 - Definición y aplicación del estereotipo «`CaballoDeTiro`»

La parte superior de la figura muestra el diagrama de perfil que introduce este estereotipo como extensión de la metaclase `Class`. La parte inferior muestra la aplicación de ese estereotipo en un diagrama de clases. La clase concreta `Boulonés` está provista del estereotipo «`CaballoDeTiro`» para especificar que el caballo de la raza `Boulonés` es un caballo de tiro. Esta clase es una instancia de la metaclase `Class` vinculada a una instancia del estereotipo «`CaballoDeTiro`».

➤ El perfil corresponde al nivel del metamodelo. En este ejemplo, incluye el estereotipo «`CaballoDeTiro`» que extiende la metaclase `Clase` del metamodelo de UML. El diagrama de clases corresponde al nivel del modelo.

## b. La noción de estereotipo requerido

Los estereotipos pueden ser requeridos por las metaclases. En ese caso, cada instancia de la metaclass debe estar asociada a una instancia del estereotipo. Para establecer esta modalidad, es preciso agregar la obligación {required} en el extremo de la asociación de extensión situada del lado del estereotipo, tal y como se muestra en la figura 11.4.

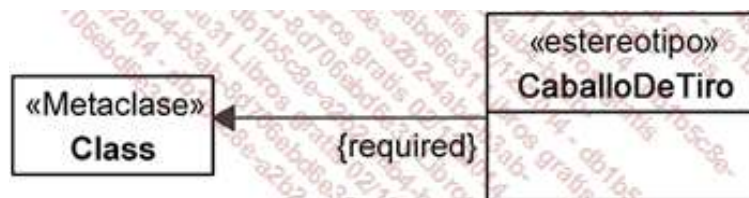


Figura 11.4 - Definición del estereotipo requerido «CaballoDeTiro»

Una vez aplicado el diagrama de perfil de la figura 11.4, todas las nuevas clases aparecerán provistas sistemáticamente del estereotipo «CaballoDeTiro». Siempre es posible crear otros estereotipos, como se ilustra en la figura 11.5.

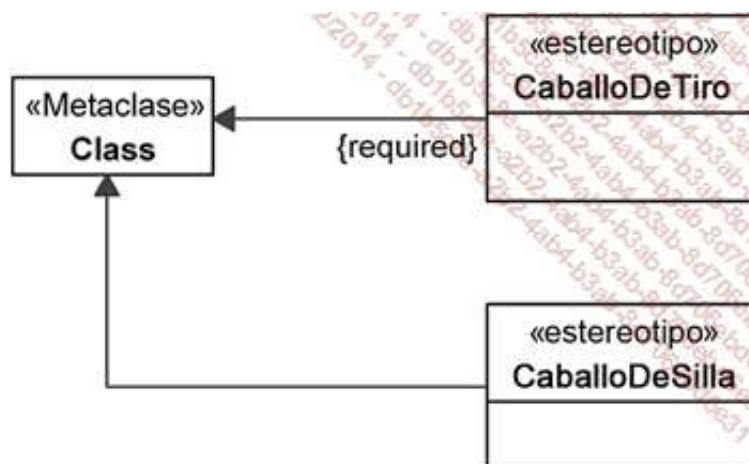


Figura 11.5 - Definición del estereotipo requerido «CaballoDeTiro» y del estereotipo no requerido «CaballoDeSilla»

Al proceder a la aplicación, el estereotipo «CaballoDeTiro» será requerido por todas las clases: La aplicación del perfil autoriza la creación de clases con los dos estereotipos o sólo con el estereotipo «CaballoDeTiro», como puede verse en la figura 11.6. El caballo normando de raza Cob se utiliza como caballo de tiro y como caballo de silla.

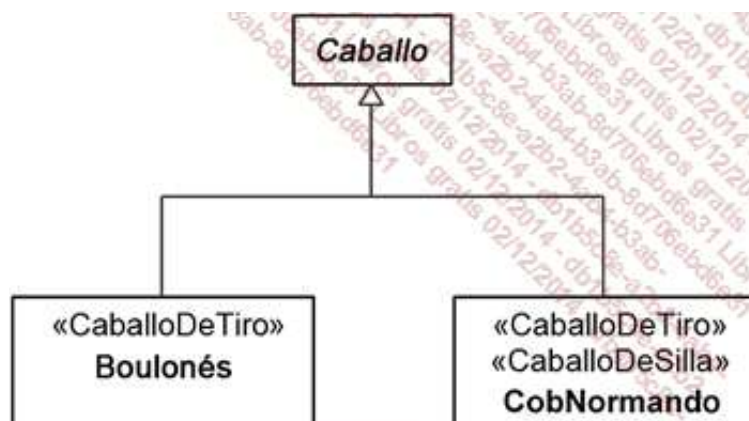


Figura 11.6 - Aplicación del estereotipo requerido «CaballoDeTiro» y del estereotipo no requerido «CaballoDeSilla»

### c. La extensión de varias metaclases mediante un mismo estereotipo

Un estereotipo puede extender varias metaclases. Puede aplicarse entonces a las instancias de cada una de esas metaclases. La figura 11.7 ilustra esa extensión.

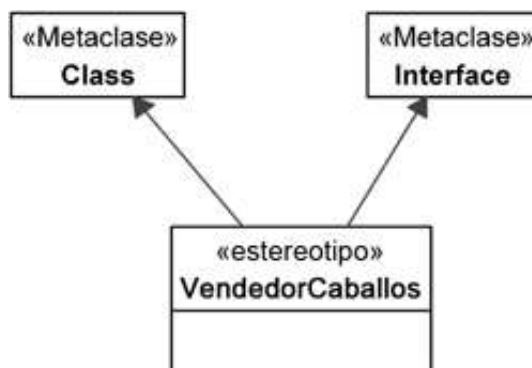


Figura 11.7 - Estereotipo «VendedorCaballos» que extiende las metaclases *Class* e *Interface*

El estereotipo «VendedorCaballos» extiende dos metaclases: *Class* e *Interface*. La aplicación de un perfil que contenga ese estereotipo conduce a la posibilidad de proveer de él a todas las clases o interfaces. Las clases o interfaces describen a vendedores especializados en la venta de caballos.

### d. La generalización y la especialización de los estereotipos

Los estereotipos pueden especializarse mediante la relación de generalización/especialización. Esta última se aplica a los estereotipos de la misma forma que a las clases.

Los estereotipos pueden ser abstractos o concretos. Un estereotipo concreto puede ser instanciado y, por tanto, puede aplicarse a un elemento. Un estereotipo abstracto no se puede instanciar: su finalidad es ser especializado para dar lugar a uno o varios estereotipos concretos.

La figura 11.8 muestra un ejemplo de estereotipo abstracto y especializado: «DeporteEquino» que extiende la metaclass *Class*.

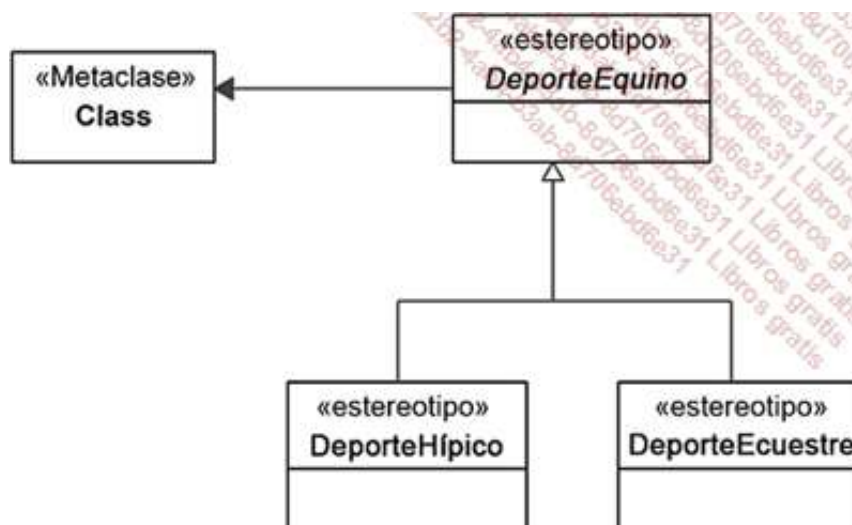


Figura 11.8 - Estereotipo abstracto y especializado «DeporteEquino»

El deporte equino se declina de dos formas: el deporte consagrado a las carreras de trote y galope y el deporte ecuestre consagrado a las demás formas de deporte equino cuya finalidad no es ganar una carrera. El estereotipo «DeporteEquino» se introduce como un estereotipo abstracto especializado por dos estereotipos concretos que corresponden a las dos formas descritas anteriormente. Sólo esos dos estereotipos pueden aplicarse a una clase que describa un deporte equino para cualificarla como deporte hípico o como deporte ecuestre.





Conviene subrayar que la asociación de extensión con la metaclass `Class` se realiza únicamente a nivel del estereotipo abstracto «DeporteEquino». El extremo de esa asociación por el lado de ese estereotipo se hereda en los estereotipos especializados «DeporteHípico» y «DeporteEcuestre».

# Las tagged values

## 1. La noción de tagged value (valor etiquetado)

Un estereotipo puede introducir atributos como cualquier clase. En la terminología UML, ese tipo de atributos se llama tag (etiquetas) y las parejas (atributo, valor) se conocen como tagged value (valor etiquetado por el nombre del atributo). Cada tag de un estereotipo da lugar a una tagged value dentro del elemento UML provisto de ese estereotipo.

Como los atributos, los tag poseen un tipo. Sólo los tipos introducidos en el metamodelo (y, como más adelante veremos, en el perfil) pueden utilizarse para caracterizar un tag.

La figura 11.9 muestra un ejemplo de introducción del tag `precioMedio` en el estereotipo «CaballoDeDeporte» y de la tagged value correspondiente a ese tag en la clase `TrotadorFrancés` provista de ese estereotipo. Las tagged values de un elemento se indican como listas entre llaves.

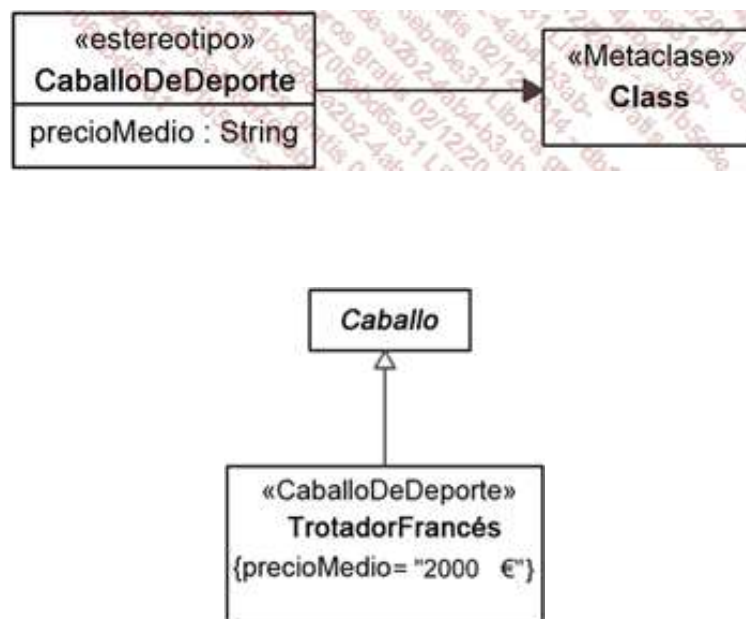


Figura 11.9 - Tag `precioMedio` del estereotipo «CaballoDeDeporte»

## 2. Las asociaciones entre estereotipos

Entre los estereotipos de un perfil es posible introducir asociaciones. Dichas asociaciones son en todo idénticas a las existentes entre las clases. Una asociación binaria entre dos estereotipos relaciona los elementos provistos del estereotipo situado en un extremo de la asociación con los elementos provistos del estereotipo situado en el otro extremo. La figura 11.10 ilustra una asociación de ese tipo entre los estereotipos «CaballoDeDeporte» y «DeporteEquino».

- Dado que el estereotipo «DeporteEquino» es abstracto y especializado, la asociación relaciona cualquier clase provista del estereotipo «CaballoDeDeporte» con clases provistas, bien del estereotipo «DeporteHípico», o bien del estereotipo «DeporteEcuestre».

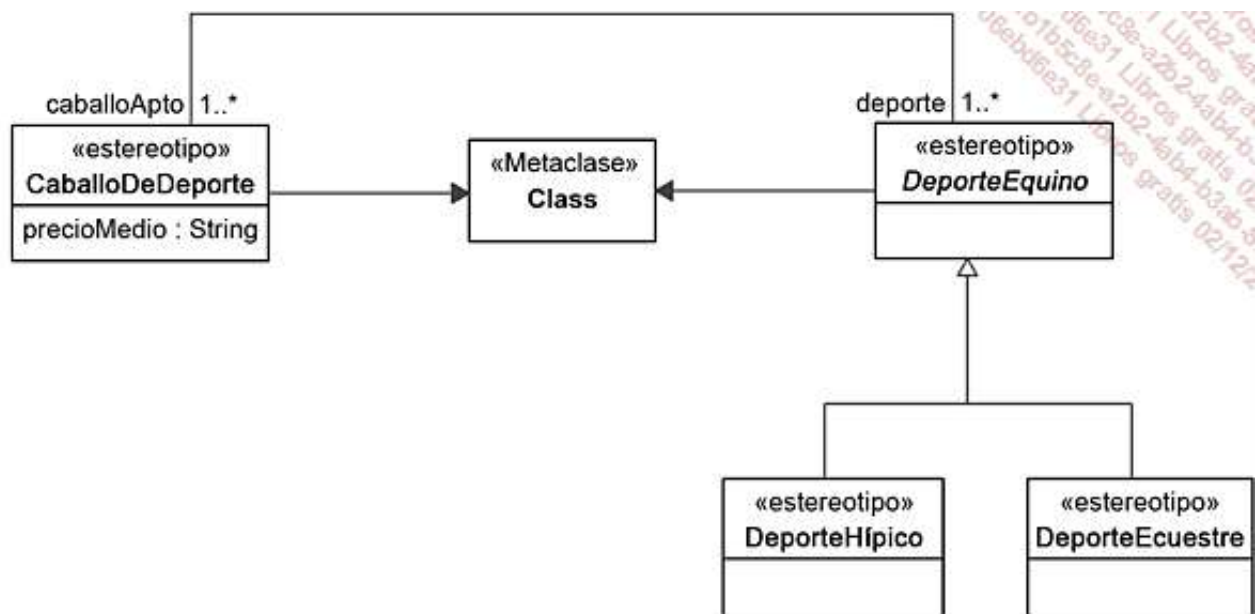


Figura 11.10 - Asociación entre dos estereotipos

La figura 11.11 muestra un diagrama de clases al que se ha aplicado el perfil de la figura 11.10. La clase `TrotadorFrancés` aparece provista del estereotipo «CaballoDeDeporte» y, por consiguiente, detenta dos tagged values: `precioMedio` de valor "2000 €" y `deporte`, cuyo valor es la clase `Trote`. La tagged value `sport` corresponde al rol con el mismo nombre que la asociación entre los estereotipos «CaballoDeDeporte» y «DeporteEquino», introducida en la figura 11.10. De manera similar, la clase `Trote` introduce la tagged value `caballoApto` cuyo valor es la clase `TrotadorFrancés`.

- Conviene bien señalar que la asociación vincula dos clases: `Trote` y `TrotadorFrancés`, y no las instancias de esas clases. Efectivamente, la asociación no es específica de estas clases, sino de los estereotipos que pone en conexión.

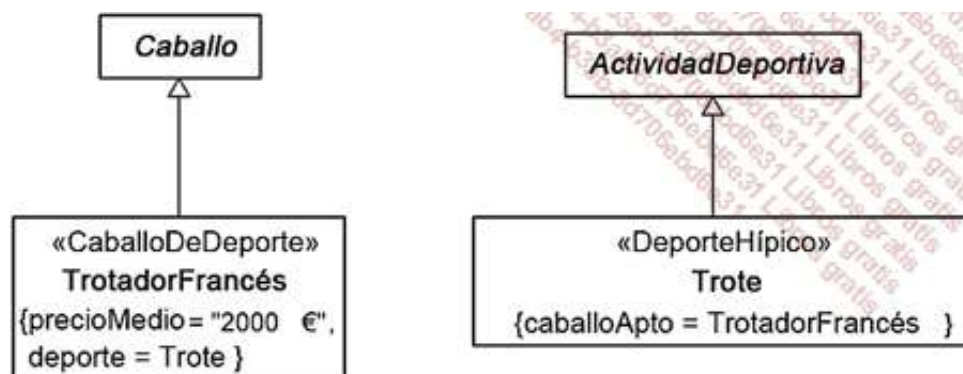


Figura 11.11 - Aplicación de una asociación entre dos estereotipos

# Los demás elementos de un perfil

## 1. Las especificaciones

Un estereotipo puede también introducir especificaciones que se aplican a los elementos provistos de ese estereotipo. La figura 11.12 ilustra ese caso en el marco de un perfil específico del lenguaje Java. En efecto, en UML, es posible emplear la herencia múltiple de las clases mientras que en Java está prohibido. El estereotipo «JavaClass» introduce una especificación escrita en OCL que prohíbe la herencia múltiple.

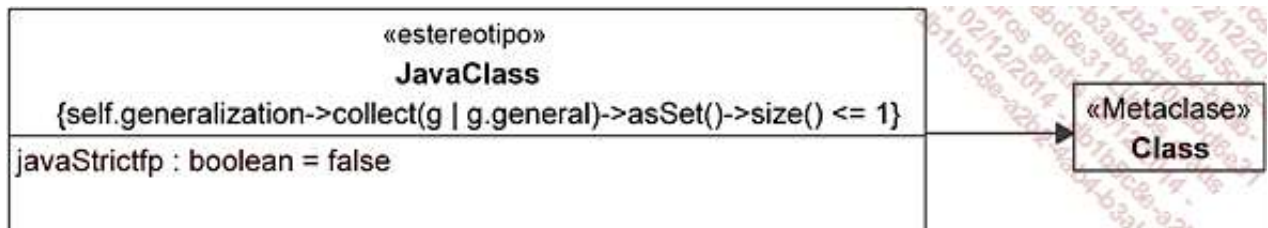


Figura 11.12 - El estereotipo «JavaClass»

Esta especificación escrita en OCL se apoya en la descripción de la relación de generalización/especialización en el metamodelo de UML introducida a nivel de la metaclass Classifier, sobreclase de la claseClass. Esta descripción en forma simplificada se ilustra en la figura 11.13, donde puede verse que los elementos generales de una instancia de Classifier son proporcionados por la expresión de ruta self.generalization.general. Dado que el rol generalization tiene como cardinalidad la cardinalidad múltiple \*, es conveniente utilizar el operador collect para obtener todos los elementos generales bajo la forma de una colección, transformada después en un conjunto para eliminar los posibles duplicados. La especificación comprueba a continuación si el conjunto tiene un número de elementos inferior o igual a 1.

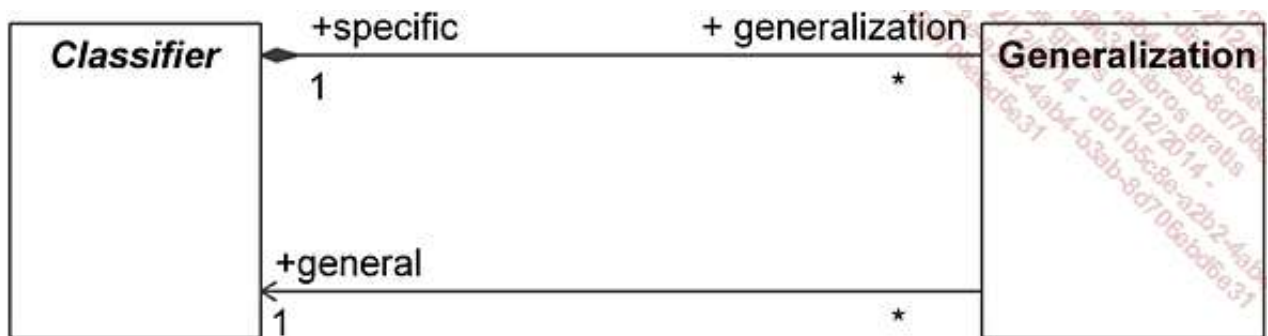


Figura 11.13 - Descripción de la relación de generalización en el metamodelo

Por último, el tag javaStrictfp sirve para indicar si la palabra clave strictfp ha sido especificada o no al definir la clase.

- La presencia de la palabra clave strictfp provoca la utilización de tipos reales IEEE con los métodos de cálculo asociados. Si la palabra clave no está, Java utiliza las operaciones de cálculo real presentes en algunos procesadores.

## 2. Las clases, tipos y enumeraciones

Los perfiles también pueden introducir clases, tipos de datos (DataType), tipos primitivos y enumeraciones. Esos tipos sólo pueden utilizarse dentro del perfil ya que se introducen a nivel del metamodelo y no del modelo. Para usarlos a la vez en el perfil y en un diagrama de clases a nivel del modelo, es conveniente

introducir los tipos en un empaquetado distinto que se puede importar al perfil y al diagrama de clases a la vez.



Un tipo de datos es un tipo en el que la igualdad entre dos instancias reposa en la igualdad entre cada atributo de ambas instancias. Para las instancias de ese tipo no existe noción de identidad como la propia de los objetos. Un tipo primitivo es un tipo de dato cuyo valor es atómico, como los enteros, los reales y los booleanos.

# Los perfiles

## 1. La representación de un perfil

Los perfiles se representan con empaquetados de perfil, es decir, empaquetados provistos del estereotipo «profile» en los que se detallan todos los elementos que contienen así como las metaclases que extienden. La figura 11.14 ilustra un ejemplo de perfil llamado CaballosDeporte.

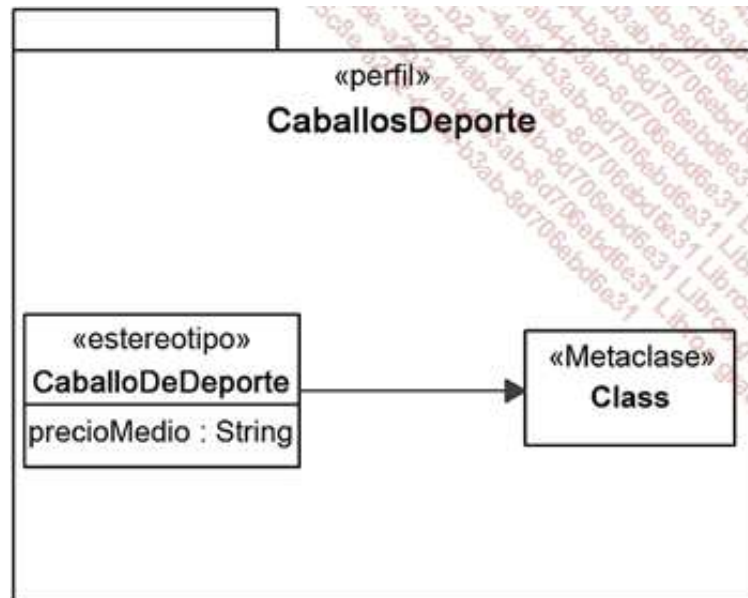


Figura 11.14 - El perfil CaballosDeporte

## 2. La relación de referencia

La relación de referencia permite importar una o varias metaclases desde un metamodelo hasta un perfil, donde se hacen visibles para cualquier modelo que aplique ese perfil. Es posible implícitamente importar en el perfil todas las clases del metamodelo. Otra posibilidad es seleccionar explícitamente las metaclases que deben importarse al perfil. Por último, es posible extender una metaclase sin importarla. En ese caso sólo son visibles las instancias extendidas por el estereotipo.

El primer caso, donde todas las metaclases del metamodelo UML aparecen implícitamente importadas por el perfil Caballos, puede verse en la figura 11.15.

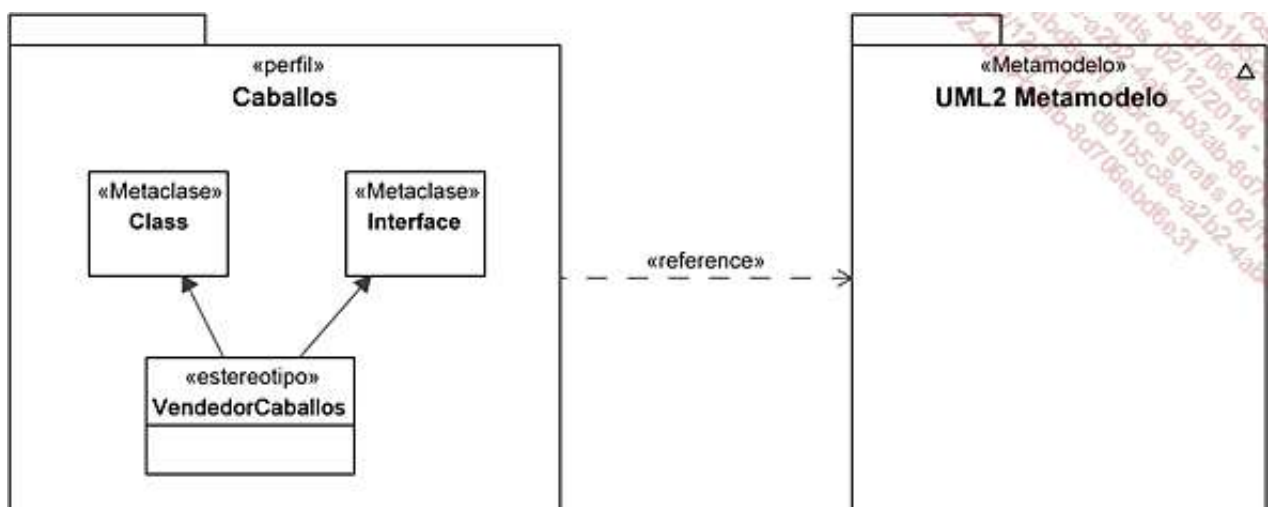


Figura 11.15 - Relación de referencia con importación implícita

El segundo caso, donde sólo se importan explícitamente en el perfil Caballos las dos metaclases `Clase` `Interface` del metamodelo UML, se ilustra en la figura 11.16. Las demás metaclases del metamodelo UML no son visibles. Las dos importaciones explícitas sobrecargan la importación explícita.

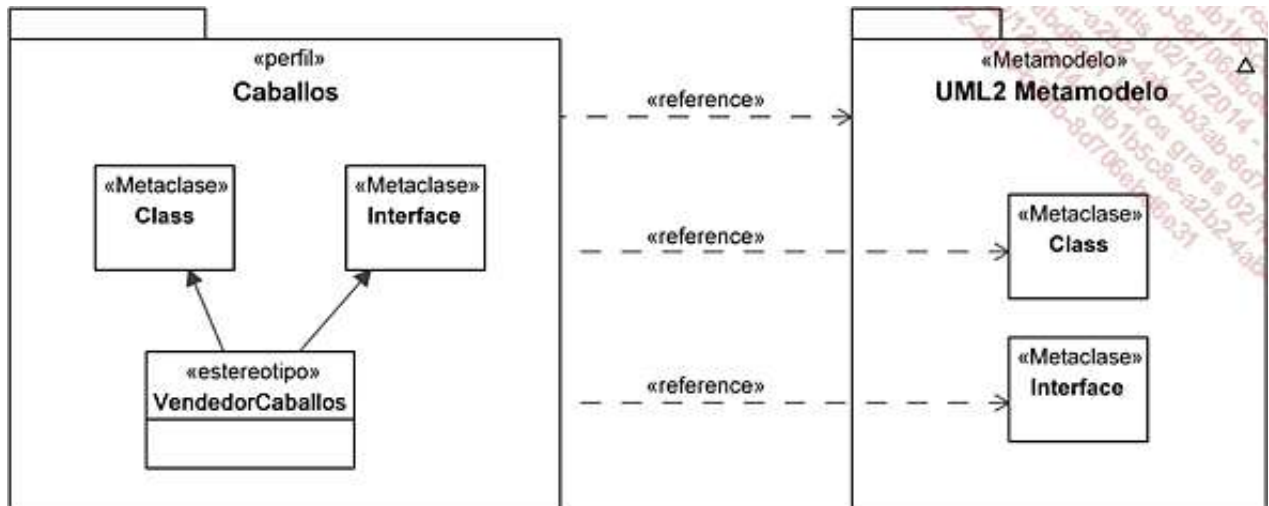


Figura 11.16 - Relación de referencia con importación explícita

La figura 11.17 ilustra un caso de importación explícita de la metaclass `Class` y una extensión explícita de la metaclass `Interface` por el estereotipo «VendedorCaballos». En lo referente a esta última metaclass, sólo las instancias extendidas mediante el estereotipo «VendedorCaballos» son visibles (las que no están extendidas o lo están por otro estereotipo no son visibles).

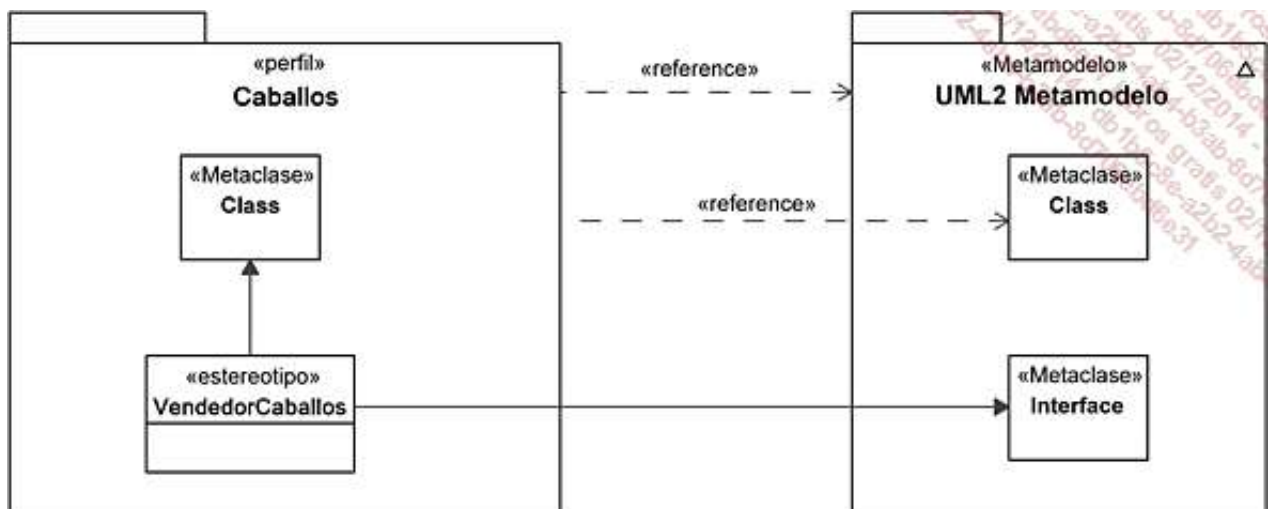


Figura 11.17 - Relación de referencia con importación y extensión explícitas

### 3. La aplicación de un perfil a un empaquetado

Un perfil se aplica a un empaquetado mediante la asociación de aplicación de perfil. De esa forma, todos los elementos del empaquetado pueden estar provistos de los estereotipos del perfil. Si la metaclass de los elementos ha sido extendida de forma obligada por un estereotipo, entonces los elementos deben estar provistos de ese estereotipo.

Es posible aplicar varios perfiles a un mismo empaquetado. En caso de conflicto entre los nombres, es conveniente usar el nombre del empaquetado como prefijo del nombre de los estereotipos.

La figura 11.8 ilustra la aplicación del perfil CaballosDeporte al

empaquetadoCriaderoCaballos. La clase TrotadorFrancés de ese empaquetado está provista del estereotipo«CaballoDeporte» introducido en el perfil CaballosDeporte.

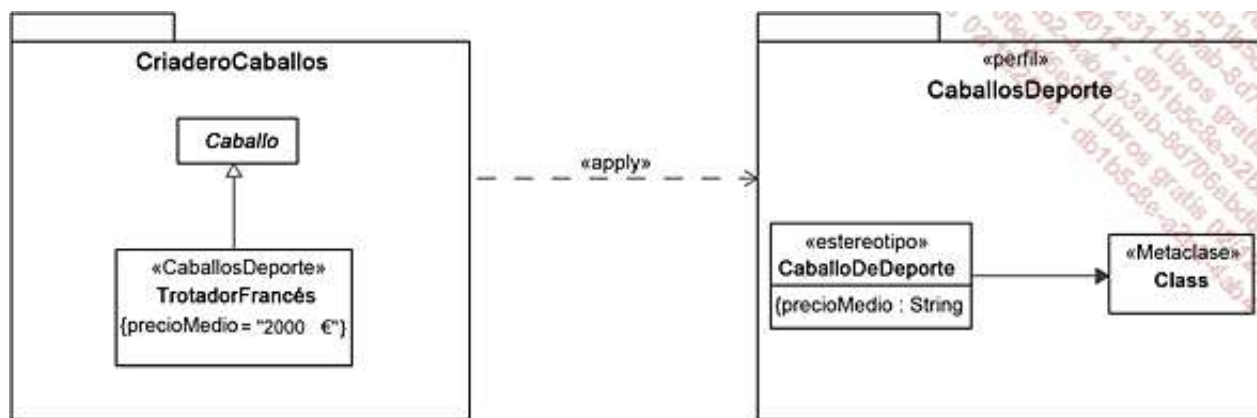


Figura 11.18 - Aplicación del perfil CaballosDeporte al empaquetado CriaderoCaballos



## Un ejemplo de dominio: los équidos

Presentamos, a título de ejemplo, un modelo basado en el dominio de los équidos.

### 1. El perfil

El perfil se presenta en la figura 11.19 bajo la forma de un diagrama de perfil. Se introducen tres estereotipos concretos para extender la metaclass `Class` en el marco del modelado de los équidos: «ÉquidoDeTiro», «ÉquidoDeSilla» y «ÉquidoDeDeporte». La jerarquía de los estereotipos relativa al deporte equino está constituida por el estereotipo abstracto «DeporteEquino» especializado por los dos estereotipos concretos: «DeporteHípico» y «DeporteEcuestre». El estereotipo abstracto permite introducir la asociación deporte/équidoApto, cuyos vínculos conectan las instancias de sus dos subestereotipos concretos, con las instancias del estereotipo «ÉquidoDeDeporte». Por último, el estereotipo «Alimentación» extiende la metaclass Asociación introduciendo el tag `alimentaciónPrincipal`. Este estereotipo puede ser aplicado a cualquier asociación del modelo.

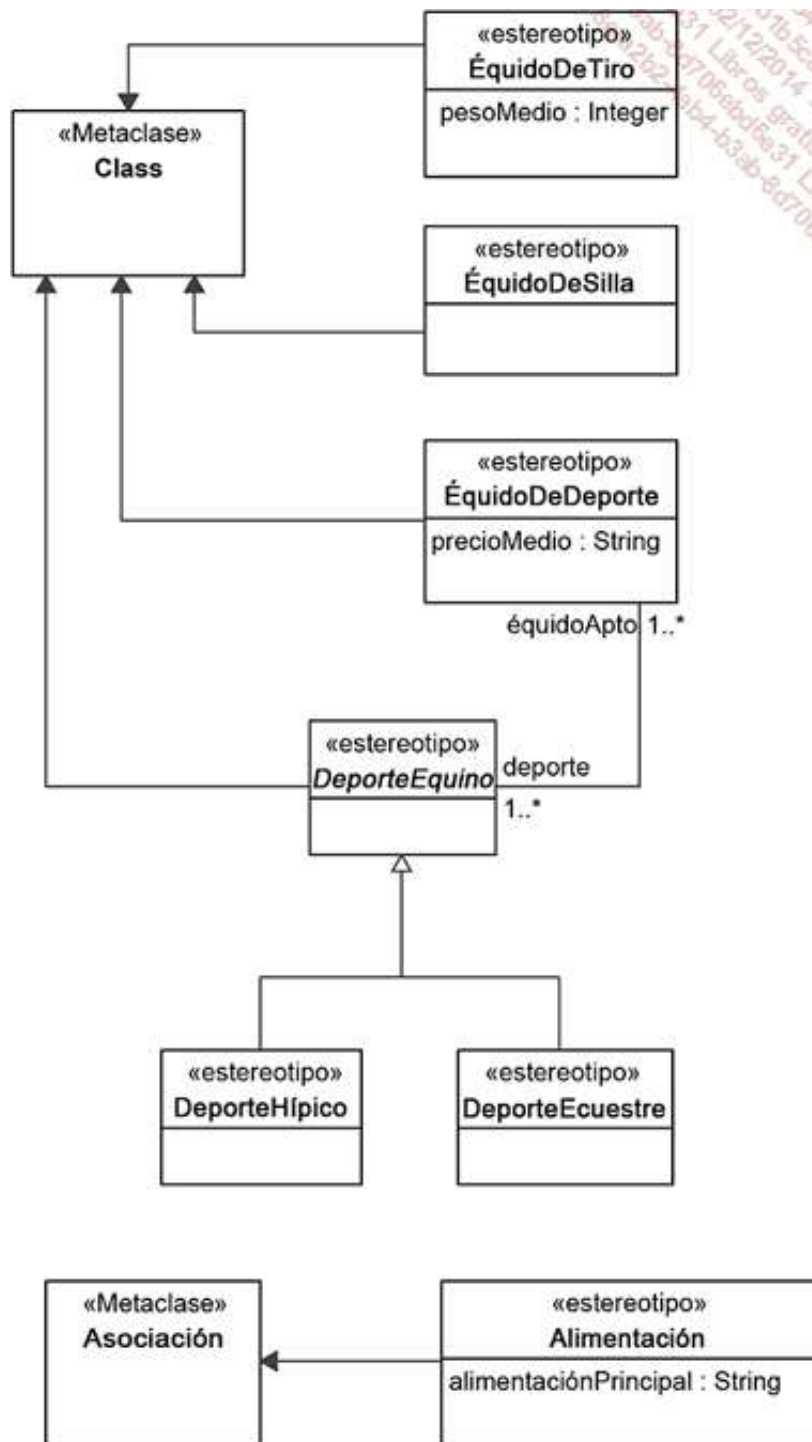


Figura 11.19 - Perfil Équidos

## 2. El modelo

El modelo se presenta en las figuras 11.20 y 11.21 bajo la forma de dos diagramas de clases. El diagrama de la figura 11.20 introduce varias clases entre las que se encuentran las clases *ComidaVegetal* y *Équido*. Estas dos clases están vinculadas por una asociación inter-objetos provista del estereotipo «Alimentación».

El diagrama introduce también una clasificación de varios équidos de raza en función de su especie: caballo, asno o poney. Las clases que corresponden a un équido de raza están dotadas de uno de los tres estereotipos del perfil para determinar la categoría del équido. Las tagged values correspondientes a los estereotipos están provistas de uno o varios valores. Por ejemplo, en el caso de poney francés de silla, hay que observar que la tagged value *deporte* posee tres valores, a saber, un vínculo hacia la clase *Adiestramiento*, un vínculo hacia la clase *ConcursoCompletoEcuestre* y un vínculo

hacia la clase `SaltoObstáculos`, los tres vínculos correspondientes a los tres deportes para los cuales ese poney es apto.

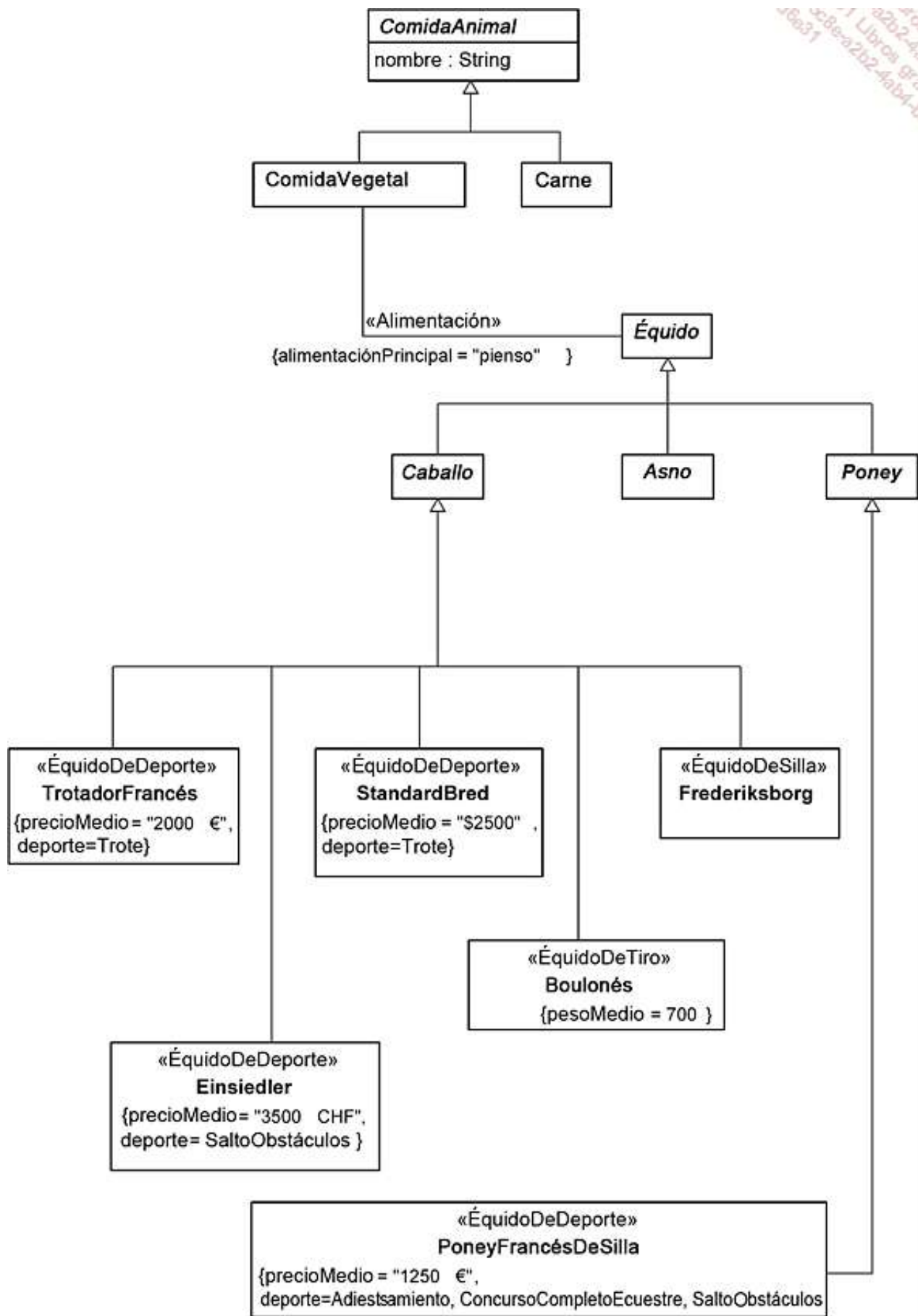


Figura 11.20 - Primera parte del modelo *Équidos*

El diagrama de la figura 11.21 muestra una clasificación de las diferentes actividades deportivas que un

équido puede ejercer. Cada actividad concreta está provista del estereotipo «DeporteHípico», o del estereotipo «DeporteEcuestre». A nivel del tag équidoApto, es conveniente señalar los vínculos inversos en relación con los aparecidos en las clases de équidos del tag deporte. Por ejemplo, la claseSaltoObstáculos aparece vinculada a las clases PonyFrancésDeSilla y Einsiedler. En efecto, si observamos la figura 11.20 podemos comprobar que esos dos équidos son aptos para el salto de obstáculos y que son los dos únicos de la clasificación de los équidos.

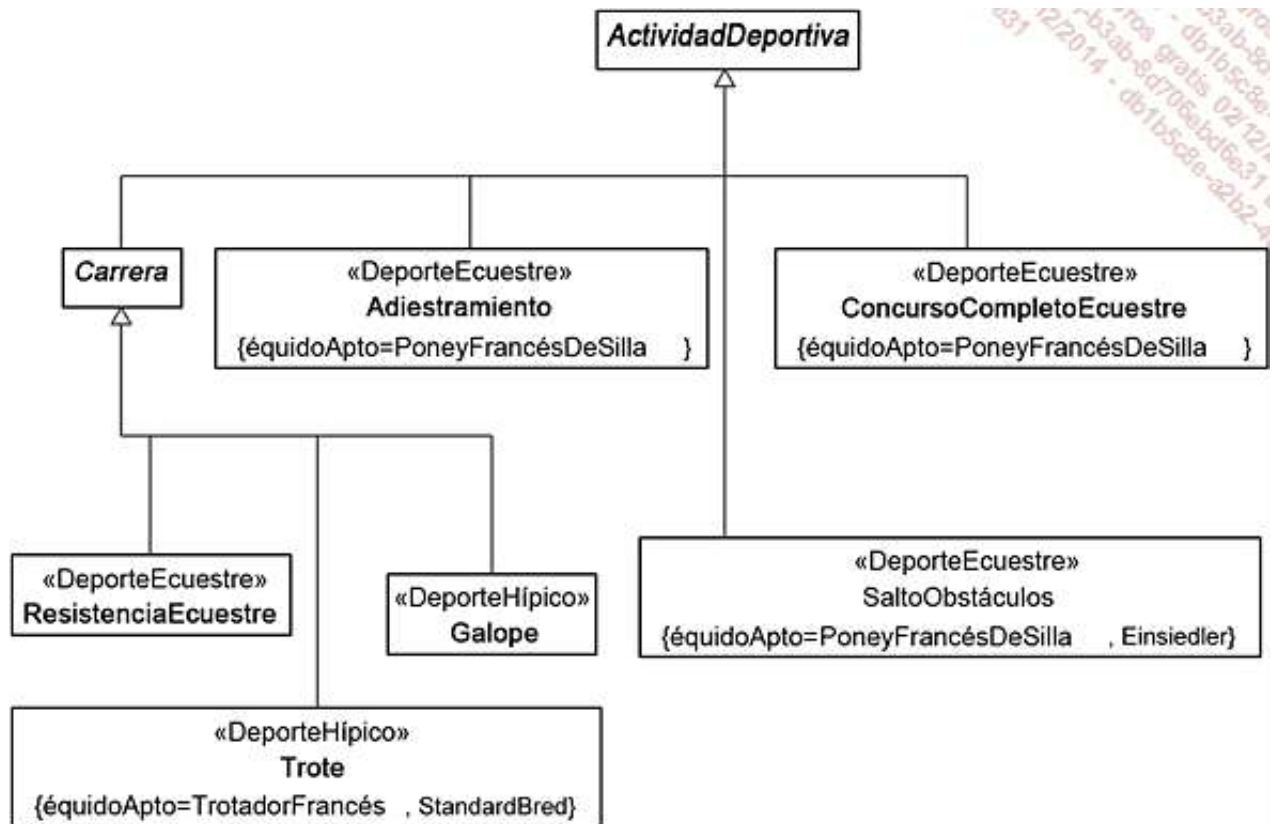


Figura 11.21 - Segunda parte del modelo Équidos

## Ejemplo de perfil de plataforma: un perfil para EJB (Enterprise JavaBeans)

La figura 11.22 ilustra un perfil de plataforma, a saber, la plataforma EJB (*Enterprise JavaBeans*). Este ejemplo de perfil extraído del documento de la OMG que describe la superestructura de UML introduce varios estereotipos:

- El estereotipo «Bean», que extiende la metaclasses `Component`. Es abstracto y especializado en los dos subestereotipos «Entity» y «Session». Como «Bean» es un estereotipo requerido, cada instancia de la metaclasses `Component` debe estar provista de uno de los dos subestereotipos. Este estereotipo introduce una especificación complementaria que prohíbe la generalización y, por consiguiente, la especialización de los beans. Esta especialización es similar a la presentada anteriormente para imponer en Java la herencia simple de las clases.
- El estereotipo «JAR», que extiende la metaclasses `Artifact`.
- Los dos estereotipos «Remote» y «Home», que extienden la metaclasses `Interface`.

➤ La metaclasses `Component` tiene como instancias los componentes abordados en el capítulo dedicado al modelado de la arquitectura del sistema, capítulo en el que se estudian igualmente los artefactos cuya metaclasses es `Artifact`.

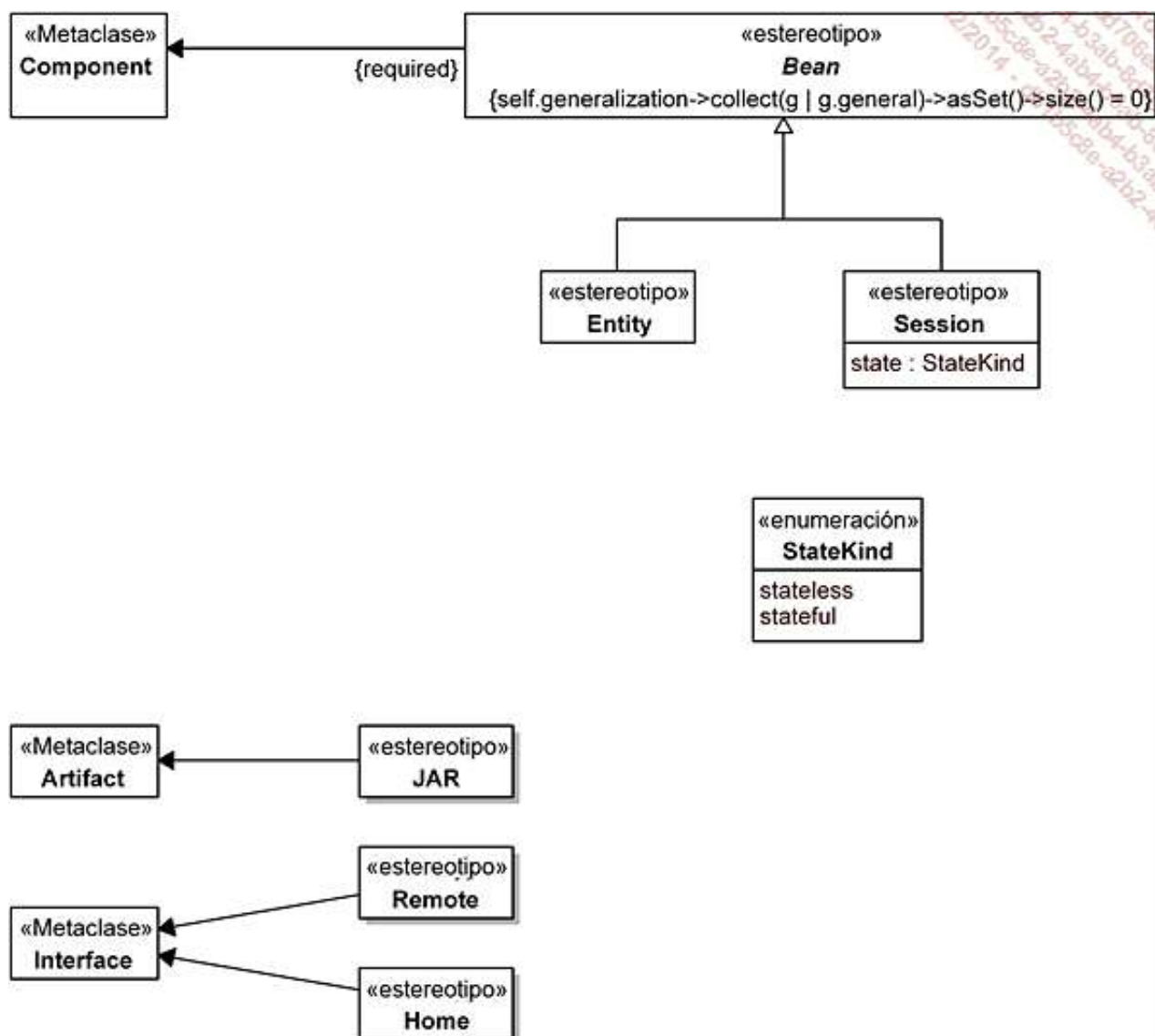


Figura 11.22 - Un ejemplo de perfil EJB

# Introducción

En este anexo, y dentro del contexto del MDA, procederemos a presentar DB-MAIN, una herramienta CASE (*Computer Aided Software Engineering* o ingeniería de software asistida por ordenador) orientada a objetos y destinada a concebir sistemas de información y, más concretamente, bases de datos relacionales. DB-MAIN se desarrolló en la Universidad de Namur. Actualmente está comercializado por la sociedad REVER de Charleroi.

DB-MAIN ofrece un proceso de diseño descendente: análisis de los requisitos, con la posibilidad de describir casos de uso y diagramas de actividad UML; diseño del esquema en un ámbito conceptual, lógico o físico, e integración de esquemas. El ámbito conceptual de DB-MAIN corresponde al modelo *objeto* de UML o al modelo *Entidad-Asociación* ampliado a la herencia. El nivel lógico corresponde al modelo relacional de los datos. El nivel físico es el del lenguaje SQL, propio del SGBDR (Sistema de Gestión de Bases de Datos Relacionales). DB-MAIN incluye también otros modelos lógicos y físicos como pueden ser IDS/2, IMS, archivos estándar o XML.

Si el modelador opta por trabajar en el ámbito conceptual, DB-MAIN realiza automáticamente la transformación al ámbito lógico (relacional) y después al físico (SQL). Esta transformación automática se inscribe en el contexto del planteamiento MDA presentado en el capítulo A propósito de UML.

Recordemos que MDA recomienda la realización de sistemas independientemente de la plataforma física y sus aspectos tecnológicos. MDA introduce el PIM, *Platform Independent Model*, modelo independiente de la plataforma, y el PSM, *Platform Specific Model*, modelo específico de la plataforma. El paso del PIM al PSM debe hacerse de manera automatizada o semiautomatizada.

En la presente obra abordaremos, sobre todo, el aspecto MDA de DB-MAIN, escogiendo como PIM el modelo *objeto* de UML y como PSM el modelo relacional. Estudiaremos de qué manera DB-MAIN transforma las clases, las asociaciones y las relaciones de herencia.



DB-MAIN presenta otras características como pueden ser la técnica retroactiva o reverse engineering (análisis de un esquema en el ámbito físico), la migración, la integración de bases de datos o la tolerancia de XML. Le invitamos a que consulte la Web de DB-MAIN, que podrá encontrar en la siguiente dirección: [www.db-main.be](http://www.db-main.be)

# Transformación del modelo objeto en modelo relacional

## 1. Transformación de las clases

En el esquema relacional las clases se convierten en tablas (también llamadas relaciones).

En DB-MAIN es posible especificar que uno o varios atributos de una clase constituyan una clave primaria, es decir, un identificador único de las instancias. Dos instancias distintas de una clase no pueden presentar los mismos valores para ese atributo o conjunto de atributos. Al producirse la transformación, los atributos constituyen la clave primaria de la tabla generada.

- Los métodos no se tienen en cuenta ya que se trata de una transformación a un PSM que únicamente gestiona datos.

La figura A.1 muestra un diagrama de clases en DB-MAIN. Los atributos que forman la clave primaria aparecen subrayados. La figura A.2 muestra el diagrama transformado en esquema relacional, donde las claves aparecen con el prefijo *id*. El prefijo *acc*, por su parte, significa que las claves primarias sirven para acceder a las líneas de la tabla.

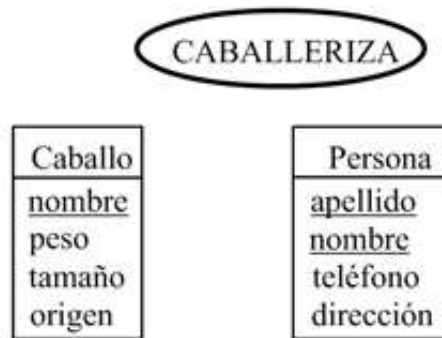


Figura A.1 - Diagrama de clases UML en DB-MAIN



Figura A.2 - Transformación en esquema relacional

- El término instancia se reserva a las clases. El término utilizado para las tablas es fila (row en inglés), o n-uplet (tuple en inglés) en las exposiciones teóricas.

- DB-MAIN ofrece también la posibilidad de introducir claves secundarias en el ámbito de las clases.

Estas claves secundarias se convierten en claves secundarias de la tabla generada.

## 2. Transformación de las asociaciones

### a. Noción de clave extranjera

En el modelo relacional, las claves primarias se usan como soporte para construir las asociaciones. Para que una línea de una tabla A pueda hacer referencia a una línea de una tabla B, se introduce una clave extranjera en la tabla A. Esta clave extranjera está formada por atributos que toman los mismos valores que los atributos de la clave primaria de la tabla B. El valor de la clave extranjera debe corresponder con uno de los valores de la clave primaria para una de las líneas de la tabla B.

### b. Asociaciones con cardinalidad 0..1 ó 1..1 en uno de sus extremos

Las asociaciones con cardinalidad 0..1 ó 1..1 en un extremo se traducen en la introducción de una clave extranjera en la tabla situada en el extremo opuesto.

La figura A.3 muestra esta transformación. El prefijo *ref* sirve para especificar las claves extranjeras.

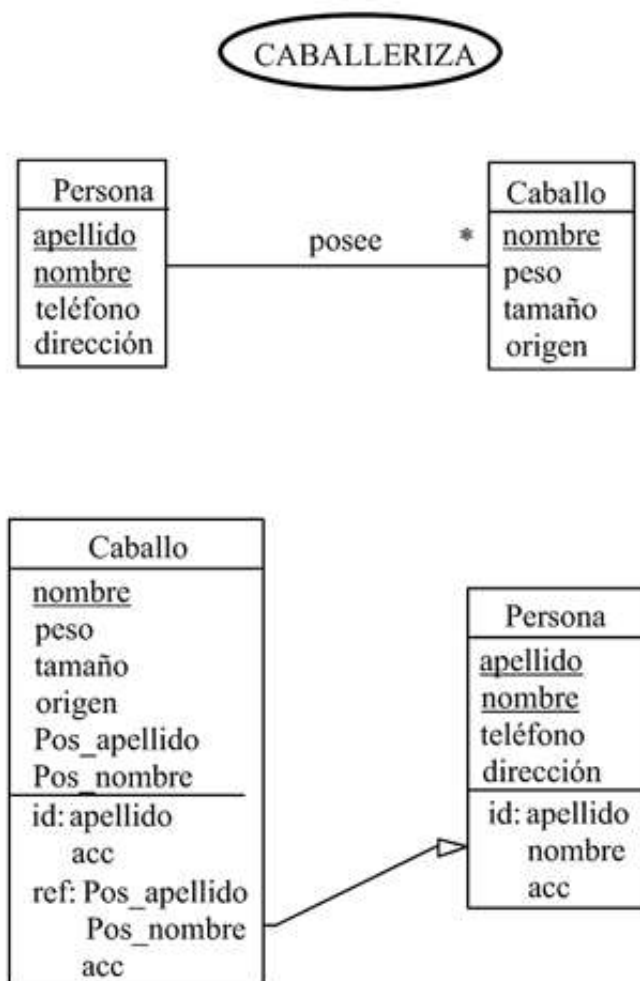


Figura A.3 - Transformación de una asociación con cardinalidad 1

### c. Otras asociaciones

Las demás asociaciones (con cardinalidad máxima en los dos extremos superior a uno) precisan de la creación de una tabla suplementaria para ser transformadas en el esquema relacional.



Ésta está formada por dos claves extranjeras, cada una de ellas correspondiente a la clave primaria de las tablas situadas en los extremos.

La figura A.4 muestra la transformación mediante un ejemplo. La tabla suplementaria *monta* contiene los mismos atributos que las claves primarias de las dos tablas situadas en los extremos de la asociación. Estos atributos sirven para constituir las claves extranjeras de la tabla suplementaria. Las claves extranjeras forman también el identificador de la tabla suplementaria.

Dado que hay dos atributos que se designan con la palabra *nombre*, DB-MAIN ha agregado automáticamente un prefijo al atributo introducido en la clase *Caballo*.

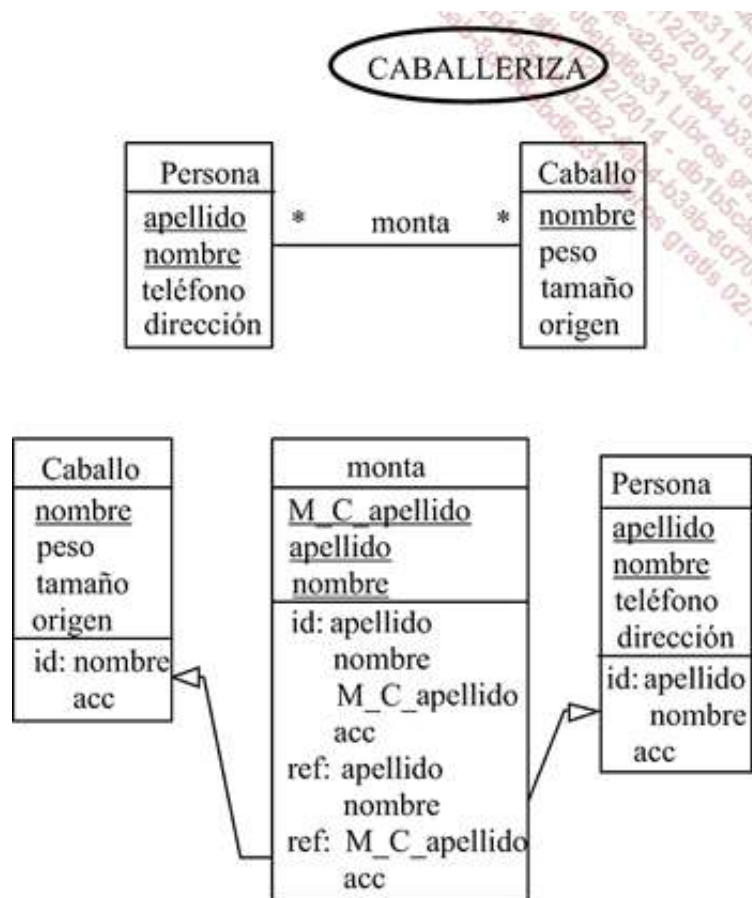


Figura A.4 - Transformación de una asociación con cardinalidades múltiples

### 3. Transformación de la herencia

#### a. Mecanismo de transformación

La relación de herencia se transforma en una asociación cuya clave primaria se sitúa en la tabla correspondiente a la superclase, y cuyas claves extranjeras se colocan en las tablas correspondientes a las subclases.

La figura A.5 muestra la transformación de la relación de herencia mediante un ejemplo.

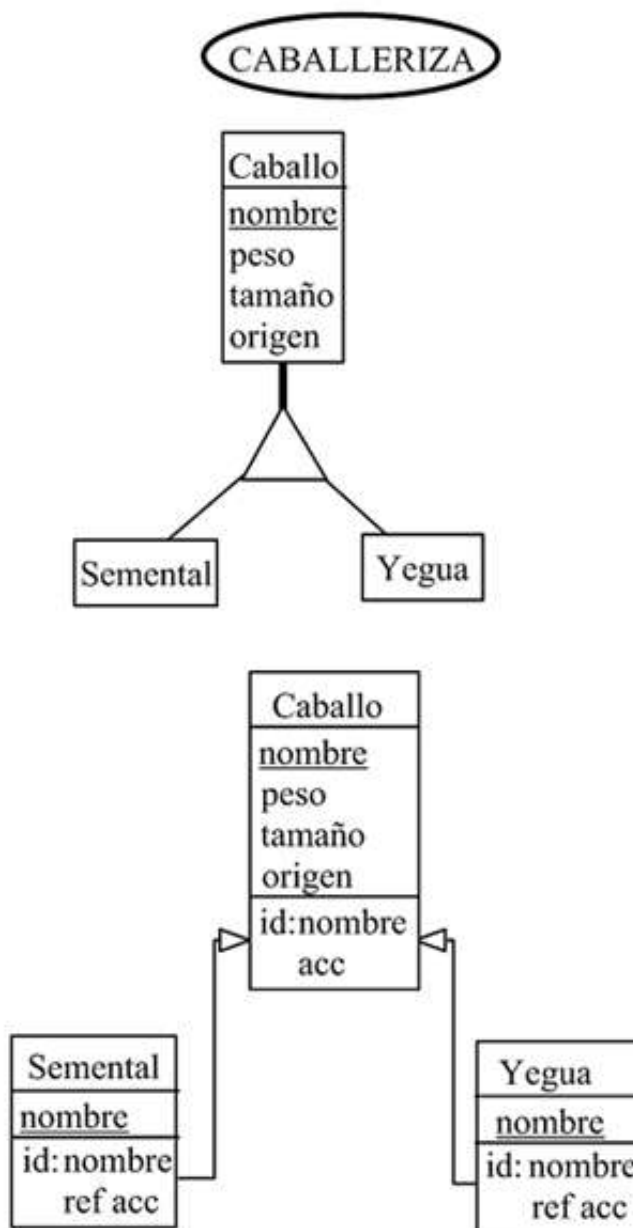


Figura A.5 - Transformación de la relación de herencia

Una instancia de una subclase da lugar a una línea de la tabla derivada de dicha subclase. La línea hace referencia a la línea que le corresponde en la tabla de su superclase. Las tablas proponen tantas claves extranjeras de ese tipo como superclases posean. Para recuperar el conjunto de los valores de la instancia, es preciso recurrir a una operación de unión. Por ese motivo, la transformación no resulta demasiado práctica en las jerarquías complejas.

## b. Especificaciones vinculadas a la relación de herencia

En el capítulo Modelado de objetos vimos que dentro de las subclases pueden expresarse cuatro especificaciones:

- La especificación {incomplete} significa que el conjunto de subclases está incompleto y que no cubre la superclase.
- La especificación {complete}, por el contrario, significa que el conjunto de subclases está completo y cubre la superclase.
- La especificación {disjoint} significa que las subclases no tienen ninguna instancia en común.
- La especificación {overlapping} significa que las subclases pueden tener una o varias instancias en común.

Estas cuatro especificaciones ofrecen cuatro posibilidades diferentes, detalladas en el cuadro de la página siguiente. En dicho cuadro también podemos encontrar el nombre y el símbolo de la especificación en DB-MAIN. El símbolo aparece dentro del triángulo que representa la relación de herencia.

{incomplete} {overlapping}	ninguna especificación en DB-MAIN	ausencia de símbolo
{incomplete} {disjoint}	disyunción	símbolo: D
{complete} {overlapping}	cobertura	símbolo: C
{complete} {disjoint}	partición	símbolo: P

Para gestionar las especificaciones, DB-MAIN introduce una serie de atributos que sirven de vínculo entre la superclase y las subclases. Luego agrega una especificación en el vínculo para expresar la especificación entre subclases:

- La disyunción entre subclases se expresa mediante la especificación relacional *excl*: como máximo uno de los atributos que sirven de vínculo toma un valor.
- La cobertura de la superclase por sus subclases se expresa mediante la especificación relacional *at-1*: como mínimo uno de los atributos que sirven de vínculo toma un valor.
- La partición de la superclase por sus subclases se expresa mediante la especificación relacional *exact-1*: exactamente uno de los atributos que sirven de vínculo toma un valor.

La figura A.6 representa un ejemplo de partición. Las subclases *Semental* y *Yegua* constituyen una partición de la superclase *Caballo*. Sólo uno de los atributos *Semental* y *Yegua* presentes en la clase *Caballo* toma un valor. De esta forma, toda instancia de *Caballo* es obligatoriamente una instancia de *Yegua* o una instancia de *Semental*.

En SQL, el generador estándar produce código que exige al programador de aplicación una gestión explícita de los atributos en función de la configuración de las subclases. El generador paramétrico (hasta ahora para Oracle) produce los componentes (views, triggers, check) que asumen plenamente las especificaciones y las operaciones de mutación (es decir, de cambio de clase de una instancia).

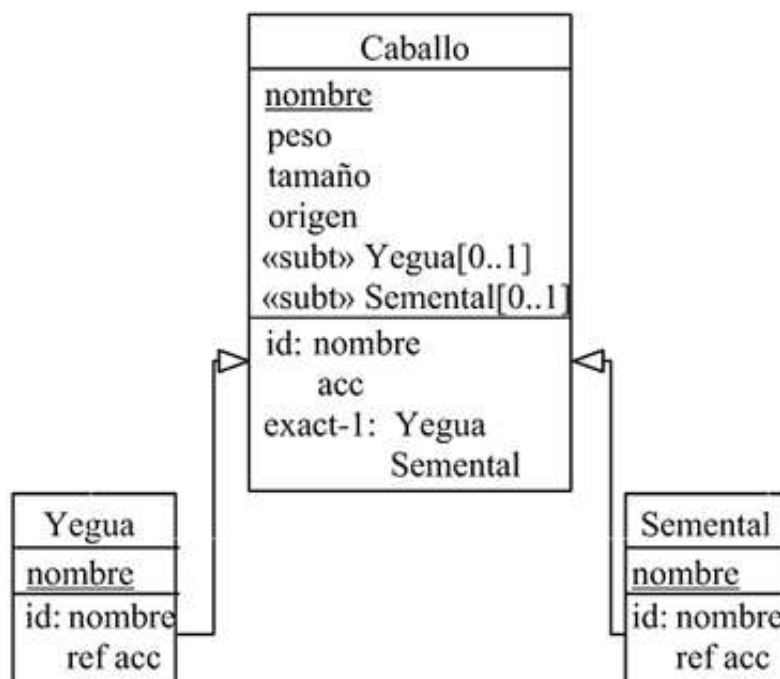
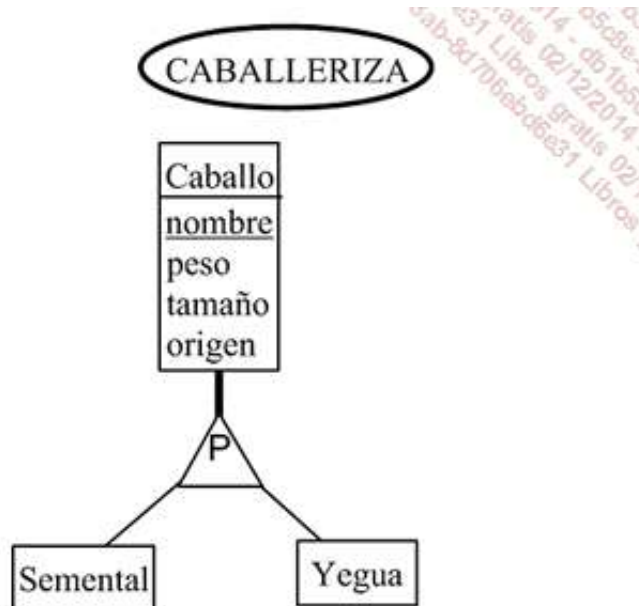


Figura A.6 - Transformación de una partición

## 4. Conclusión

DB-MAIN es una herramienta CASE que se inscribe en el contexto del planteamiento *MDA*. El PIM puede ser el modelo *objeto* de UML o el modelo *Entidad-Asociación* ampliado a la herencia. El PSM es el modelo relacional, que puede transformarse automáticamente en SQL. Existen otros PSM tales como los archivos estándar o los esquemas XML. La ventaja de DB-MAIN es que realiza la transformación de manera exhaustiva, sobre todo tomando en cuenta las especificaciones vinculadas a la herencia. Señalemos también que DB-MAIN ofrece el paso inmediato del modelo *objeto* de UML al modelo *Entidad-Asociación* y viceversa.

Como ya indicamos en el capítulo A propósito de UML, la finalidad de MDA es ofrecer una transformación automática del PIM en PSM. Recordemos las principales ventajas de MDA:

- El diseño se realiza a un nivel más abstracto, cosa que permite centrarse en exclusiva en la especificación sin tener que preocuparse de las imposiciones del código.
- El hecho de centrarse sólo en la realización del PIM supone un verdadero incremento de la productividad.
- Los aspectos semánticos se especifican de manera más explícita que si estuvieran anegados de código, hecho que garantiza una mejor legibilidad de la propia semántica.
- Toda semántica debe especificarse en el PIM para que el PSM sea válido. Por consiguiente, el PIM debe ser exacto y riguroso imperativamente.
- La generación automática del PSM desde el PIM reduce sobremanera la necesidad de recurrir al rediseño y aporta transportabilidad con respecto al objetivo.
- El problema que suponía tener que actualizar la documentación del modelo al efectuar modificaciones en el código queda resuelto. La documentación funcional está compuesta en gran parte por el PIM.
- La obligación de diseñar el PIM impone la realización de la fase de modelado. En general, esta fase no siempre se toma en cuenta en el sector industrial, más interesado en la producción, es decir, en la producción del código. MDA aporta al modelado un incremento muy significativo de la productividad.

# Capítulo Modelado de los requisitos

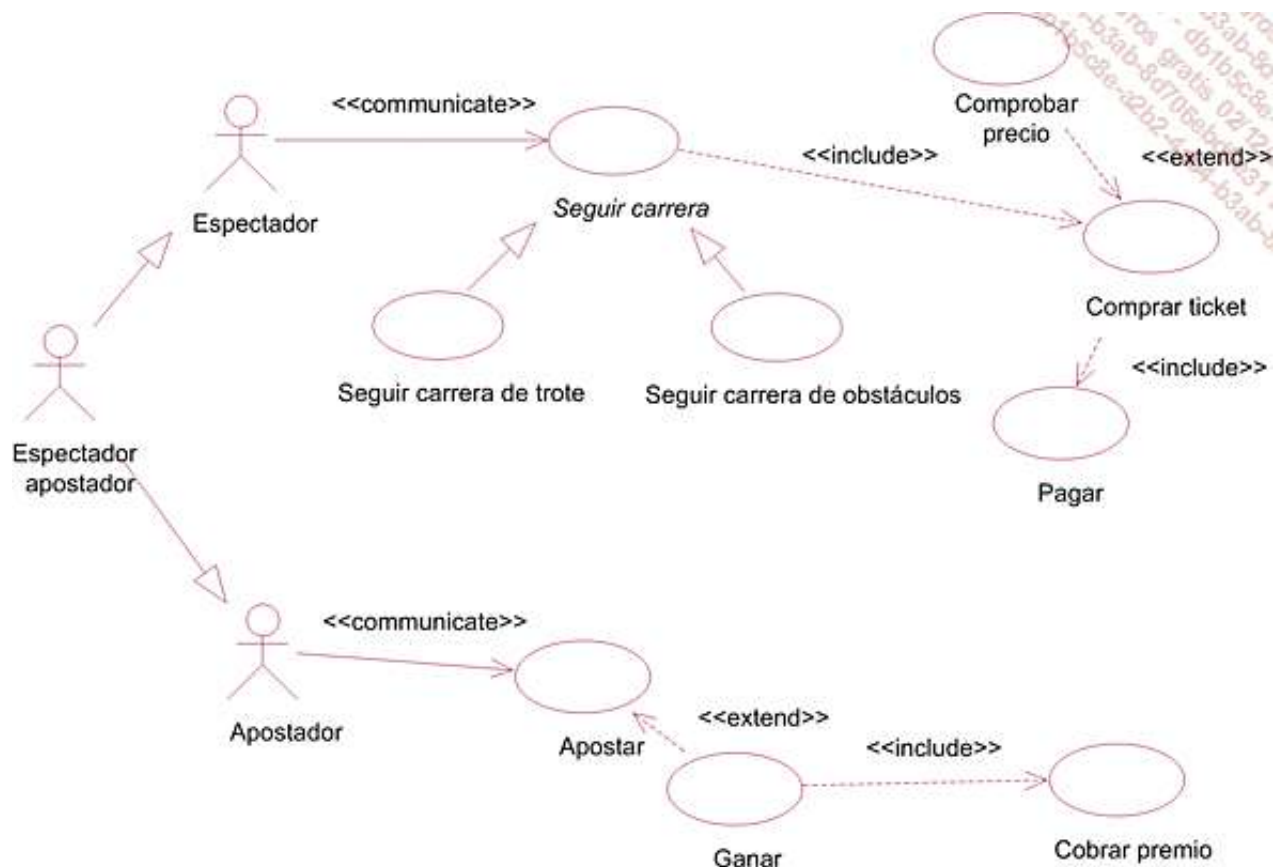
## 1. El hipódromo

Un hipódromo ofrece a sus clientes la posibilidad de asistir a las carreras y de realizar apuestas.

¿Cuáles son los actores que interactúan con estos servicios?

*El espectador, el apostador y el cliente, que es a la vez espectador y apostador.*

Construya el diagrama de casos de uso.



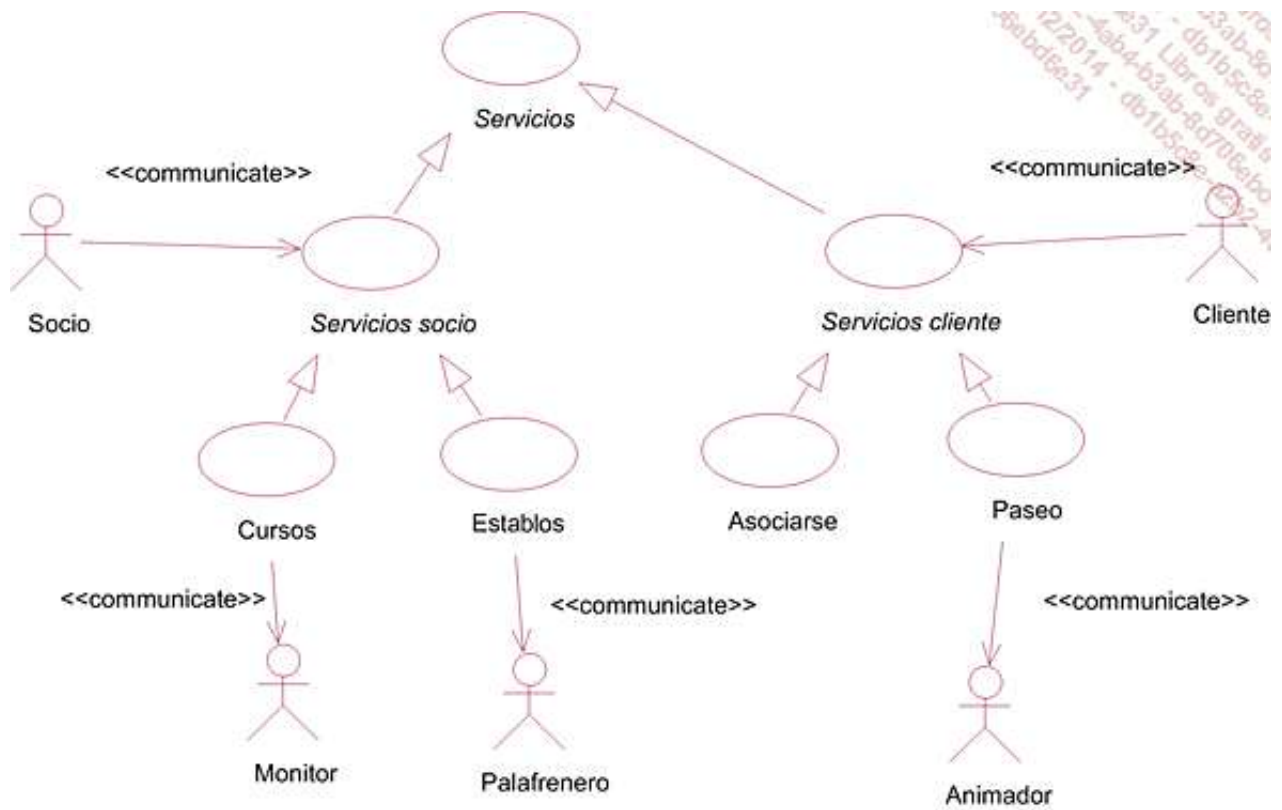
## 2. El club ecuestre

Un club ecuestre pone a disposición de los clientes establos para guardar los caballos y ofrece cursos de equitación y paseos. Sólo los socios tienen acceso a los cursos y a los servicios de establo. Los demás clientes tienen la posibilidad de participar en los paseos y de convertirse en socios.

¿Cuáles son los actores que interactúan con estos servicios?

*El monitor, el palafrenero, el animador, el socio y el cliente. El socio y el cliente son actores primarios. El monitor, el palafrenero y el animador son actores secundarios.*

Construya el diagrama de casos de uso.



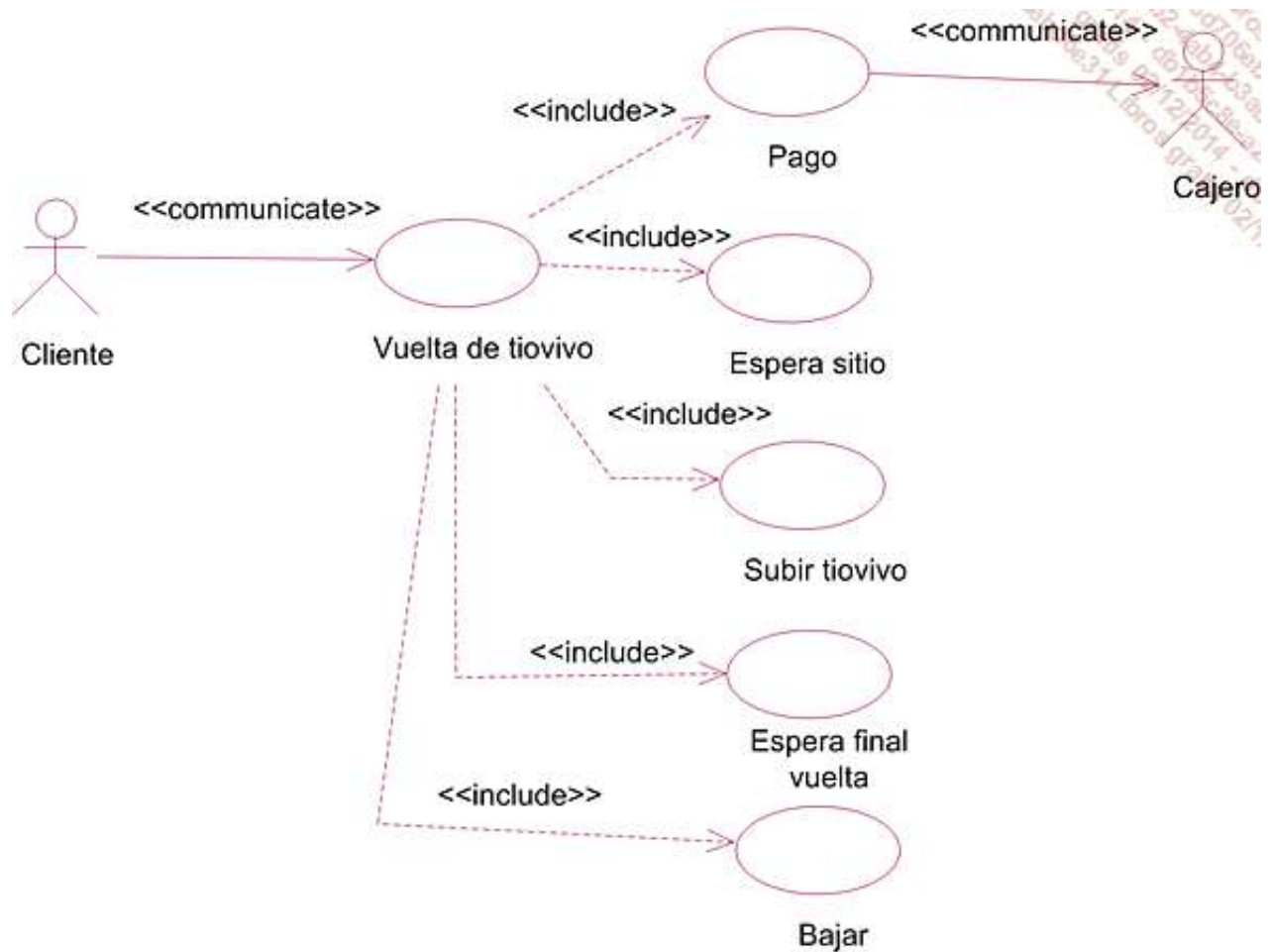
### 3. El tiovivo de caballos de madera

Un tiovivo de caballos de madera ofrece a sus clientes la posibilidad de dar una vuelta en él previo pago de una cantidad de dinero.

¿Cuáles son los actores vinculados a este servicio?

*El cliente y el cajero. El cliente es un actor primario y el cajero un actor secundario.*

Construya el diagrama de casos de uso.



Dé la representación textual correspondiente al diagrama.

Caso de uso	Vuelta de tiovivo
Actor primario	Cliente
Sistema	Tiovivo de caballos de madera
Participantes	Cliente, Cajero
Nivel	Objetivo usuario
Condición previa	El tiovivo funciona
Operaciones	
1	Pago
2	Esperar un sitio
3	Subir al tiovivo
4	Esperar a que acabe la vuelta
5	Bajar
Extensiones	
1.A	¿El cliente tiene bastante dinero?
1.A.1	Si sí, continuar
1.A.2	Si no, salir
2.A	¿La espera es demasiado larga?
2.A.1	Si sí, continuar

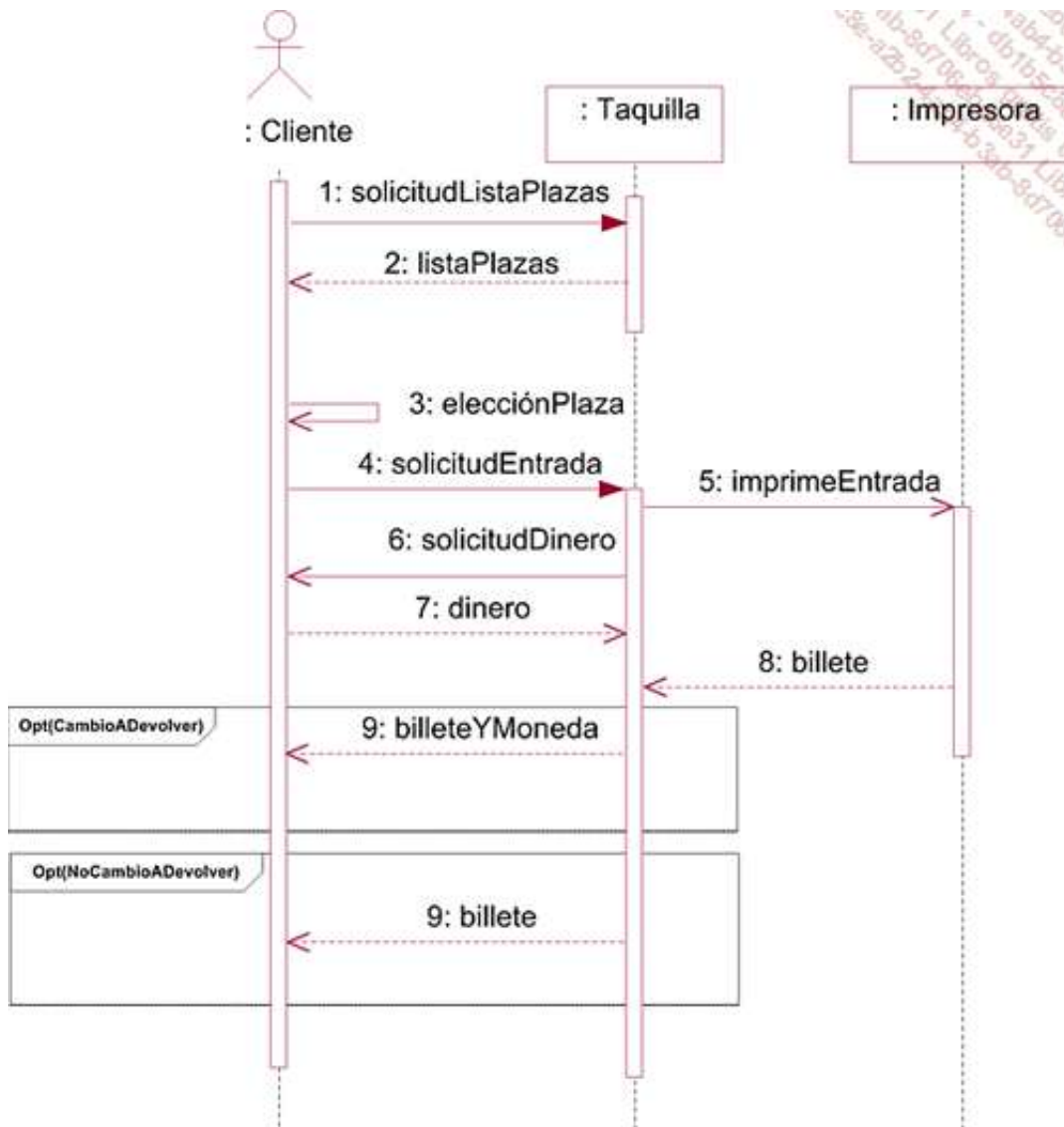


2.A.2	Si no, salir
-------	--------------

# Capítulo Modelado de la dinámica

## 1. El hipódromo

Construya el diagrama de secuencia de compra de una entrada para una carrera de caballos.

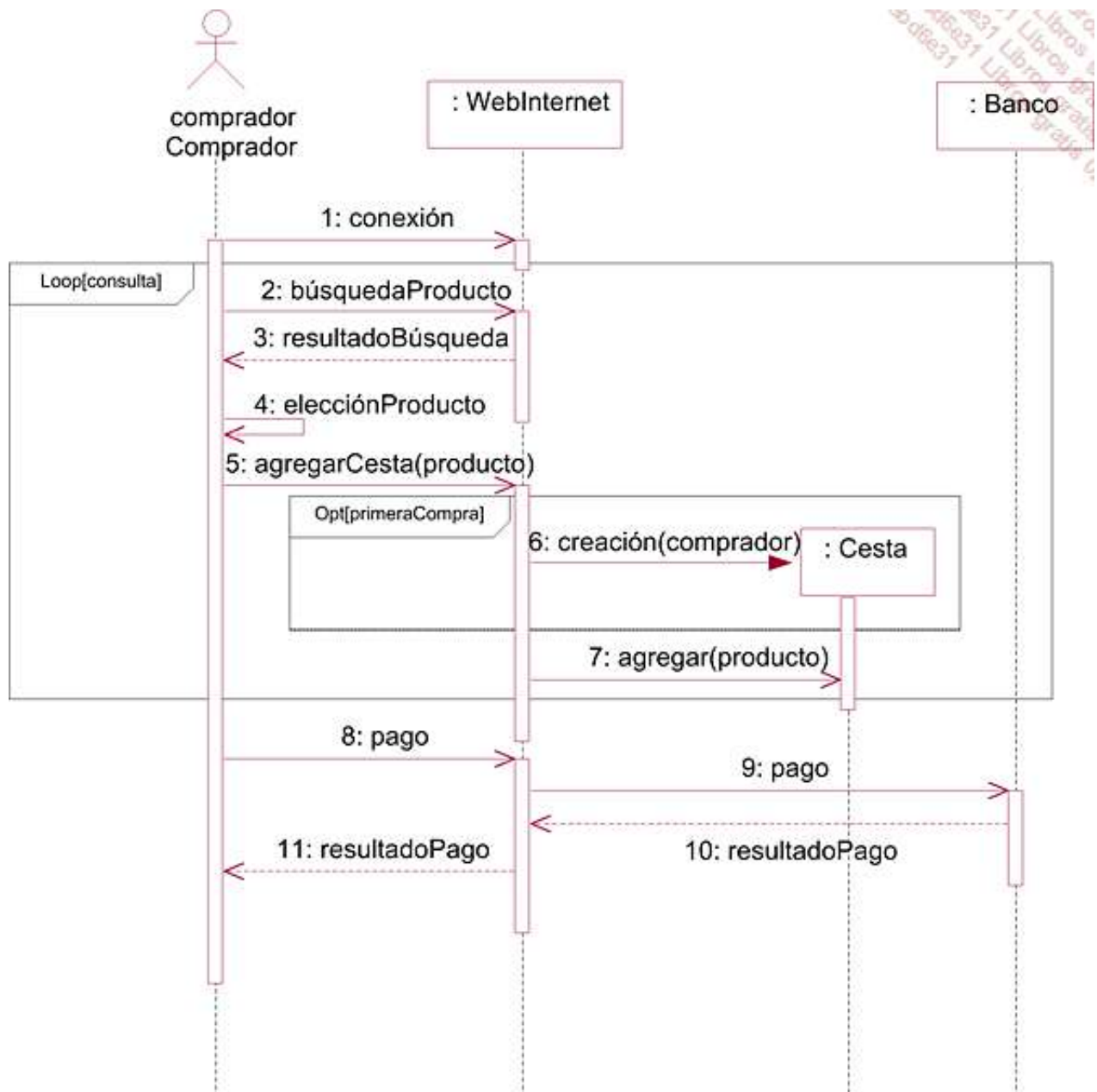


¿Cuáles son los objetos del sistema descubiertos así?

*La taquilla y la impresora.*

## 2. La central de compra de caballos

Construya el diagrama de secuencia de un pedido de productos en la página Web de la central de compra de caballos.



➤ El diagrama de secuencia no incluye la entrega. Ésta puede ser objeto de otro diagrama de secuencia.

¿Cuáles son los objetos del sistema descubiertos así?

*La página Web y la cesta. El banco es un actor secundario.*

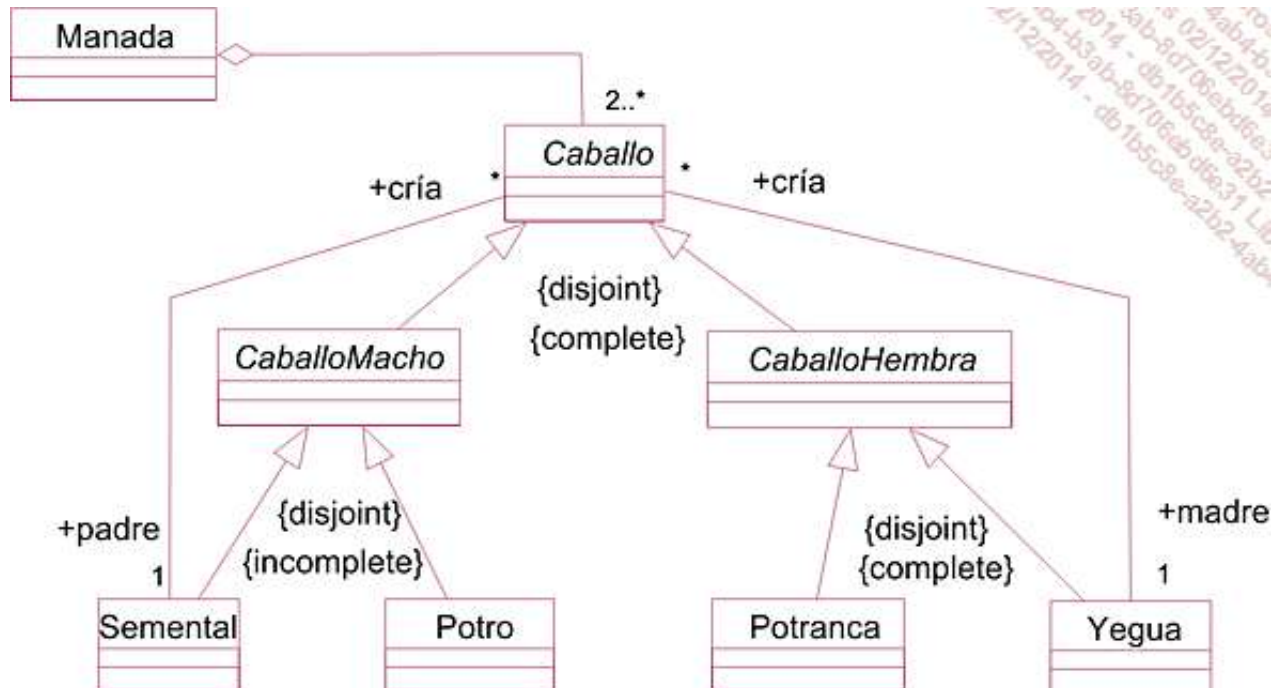
# Capítulo Modelado de objetos

## 1. La jerarquía de los caballos

Tenemos las clases *Yegua*, *Semental*, *Potro*, *Potranca*, *Caballo*, *Cabalo macho* y *Caballo hembra*, así como las asociaciones padre y madre. Establezca la jerarquía de clases haciendo figurar en ella ambas asociaciones.

Utilice las especificaciones {incomplete}, {complete}, {disjoint} y {overlapping}.

Introduzca la clase *Manada*. Establezca la asociación de composición entre esta clase y las clases ya introducidas.



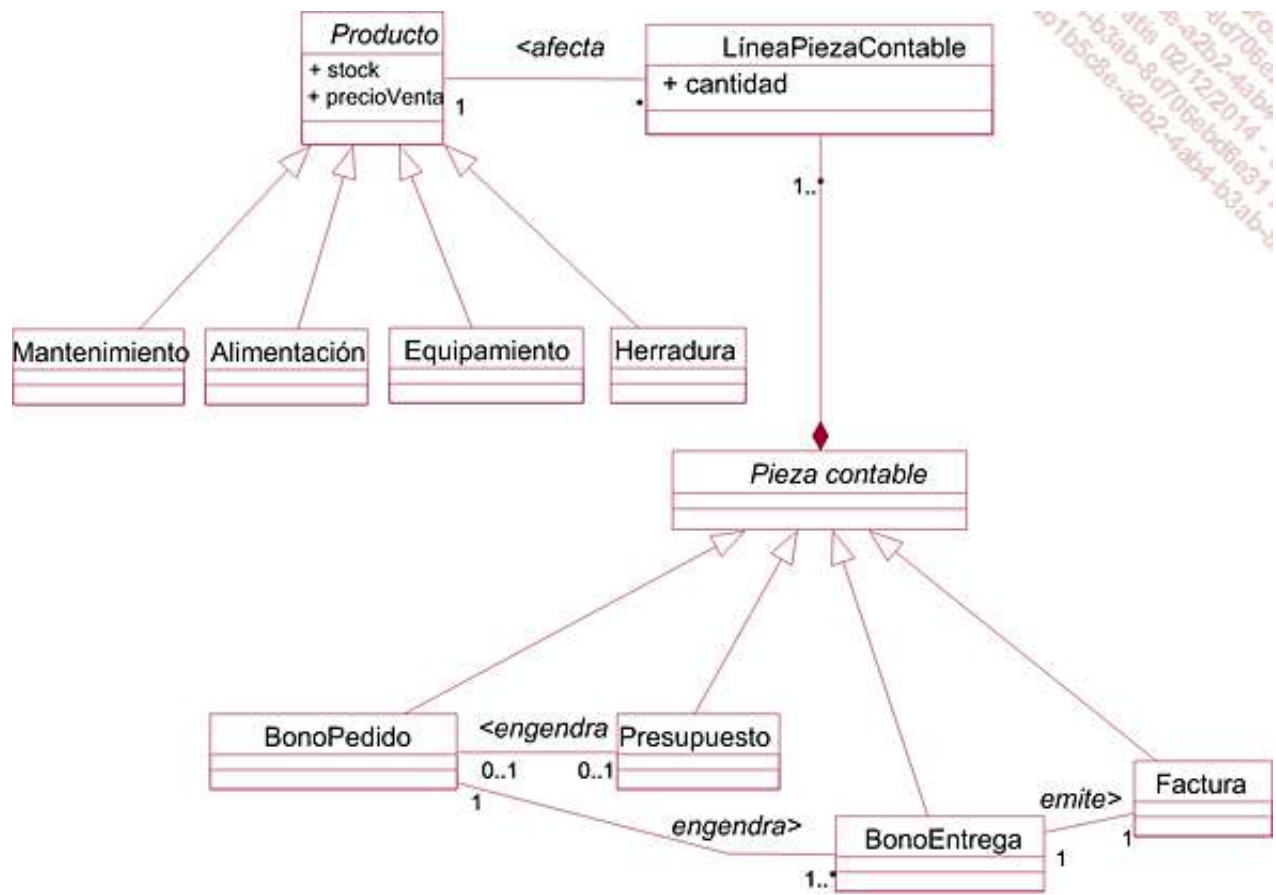
Las clases *Semental* y *Potro* no cubren la clase *CaballoMacho*, ya que existen los caballos castrados.

## 2. Los productos para caballos

Modele los aspectos estáticos del texto siguiente en forma de diagrama de clases.

Una central de caballos vende diferentes tipos de productos para caballos: productos de mantenimiento, alimentación, equipamiento (para montar el caballo), herraje.

Un pedido contiene una serie de productos y especifica la cantidad de cada uno de ellos. En caso necesario se puede elaborar un presupuesto antes de pasar el pedido. Si alguno de los productos no está en stock, a petición del cliente, el pedido puede dividirse en varias entregas. Cada entrega da lugar a una factura.



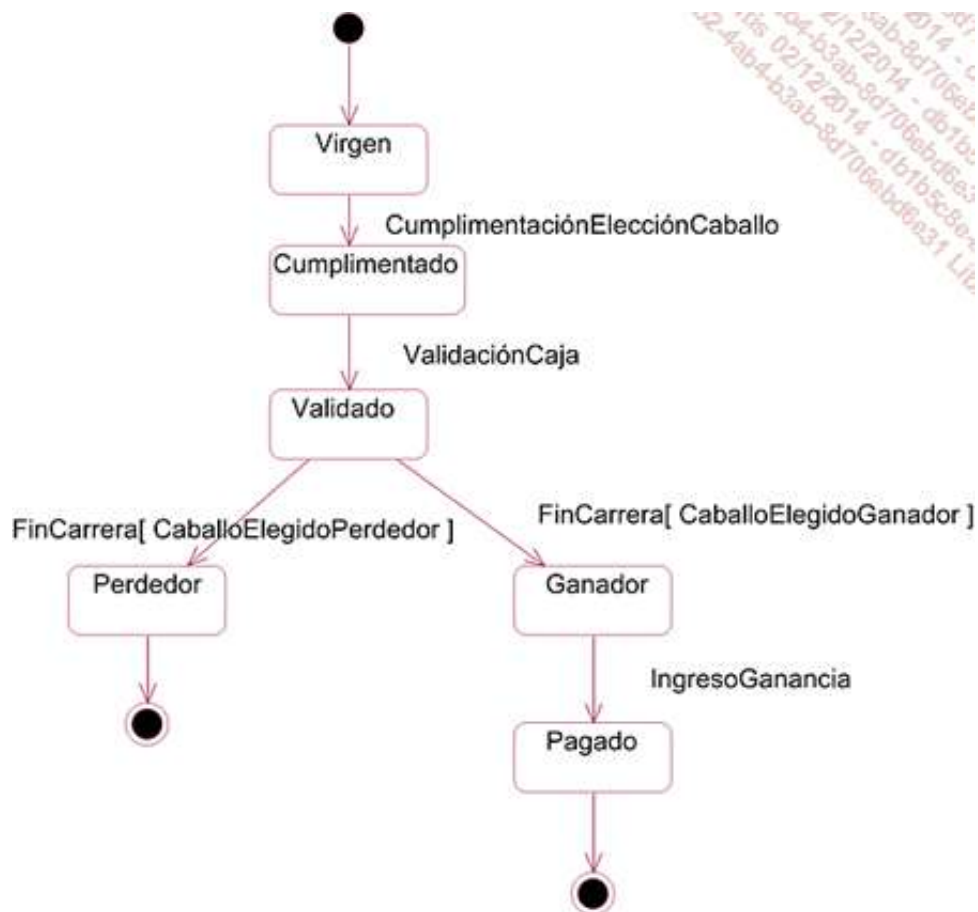
# Capítulo Modelado del ciclo de vida de los objetos

## 1. El ticket de apuesta trifecta

¿Por qué estados puede pasar un ticket de apuesta trifecta?

Los estados de un ticket de apuesta pueden ser: Sin cumplimentar, Cumplimentado, Validado, Perdedor, Ganador, Pagado.

Construya el diagrama de estados-transiciones de una instancia de la clase *Ticket*.

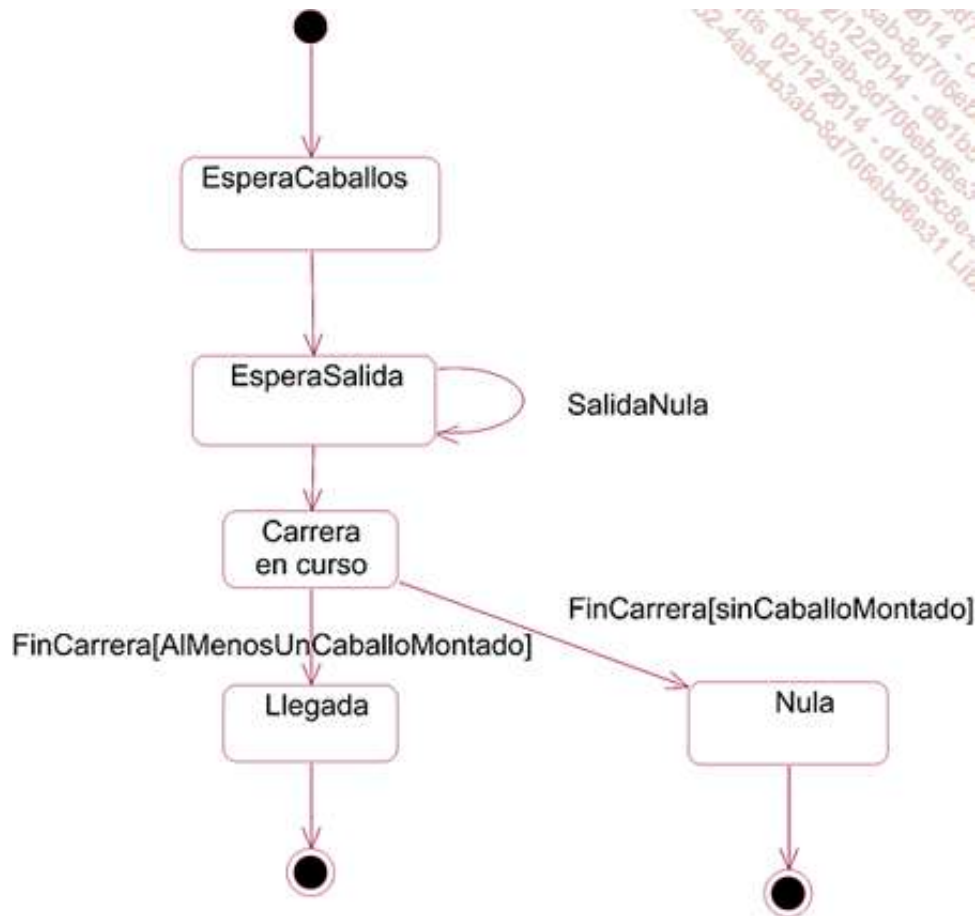


## 2. La carrera de caballos

¿Por qué estados puede pasar una carrera de caballos?

Los estados de una carrera de caballos pueden ser: A la espera de los caballos, A la espera de la salida, Carrera en curso, Llegada, Anulada.

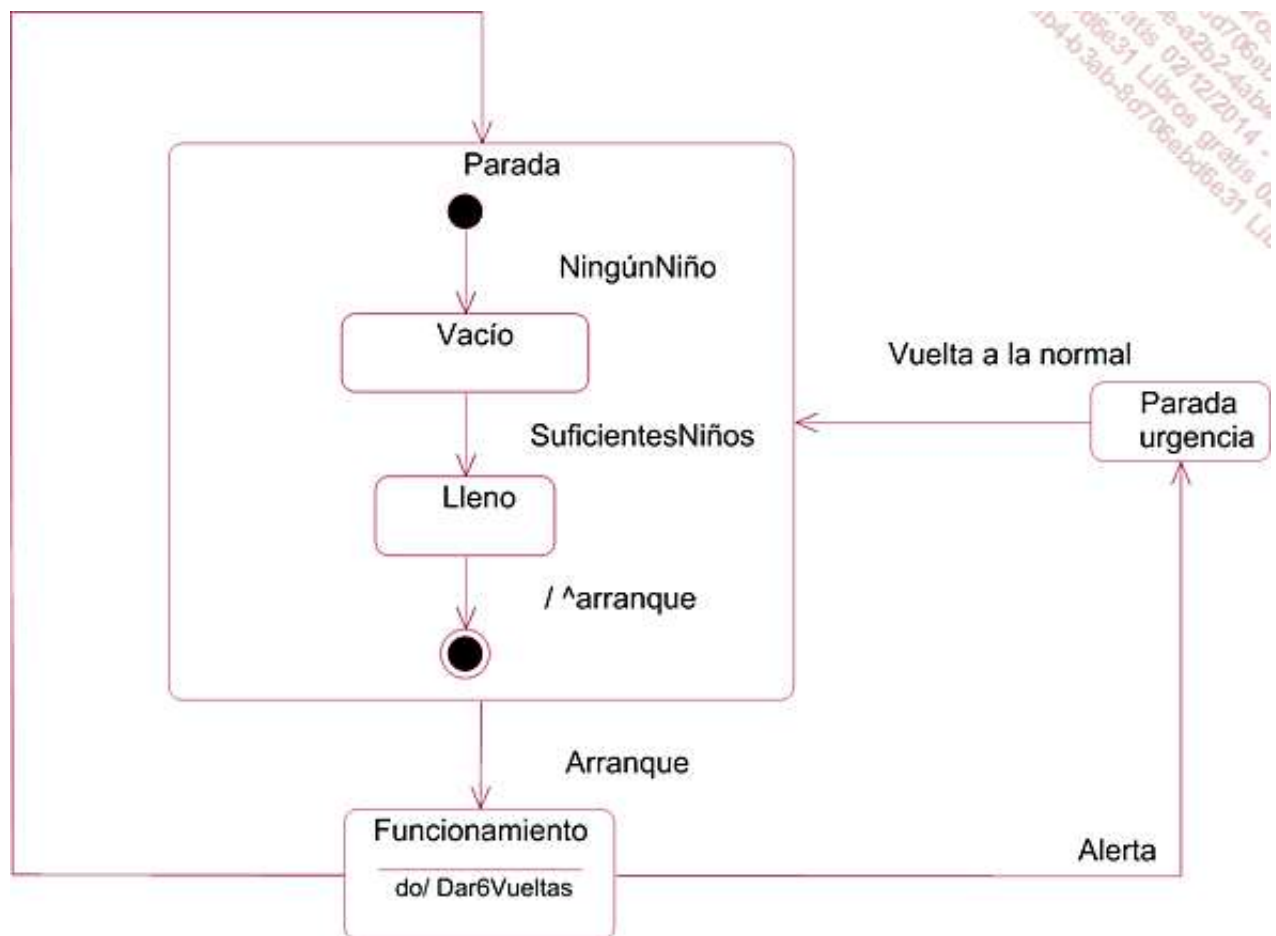
Construya el diagrama de estados-transiciones de una instancia de la clase *Carrera*.



### 3. El tiiovivo de madera

Describa los diferentes estados posibles de un tiiovivo de caballos de madera y construya su correspondiente diagrama de estados-transiciones.

*Los diferentes estados posibles de un tiiovivo de madera son: En parada, en funcionamiento o en parada de emergencia (tras una alerta). Mientras está en el estado de parada, puede estar lleno (antes de una vuelta) o vacío (después de una vuelta, cuando ya se han bajado todos los niños).*

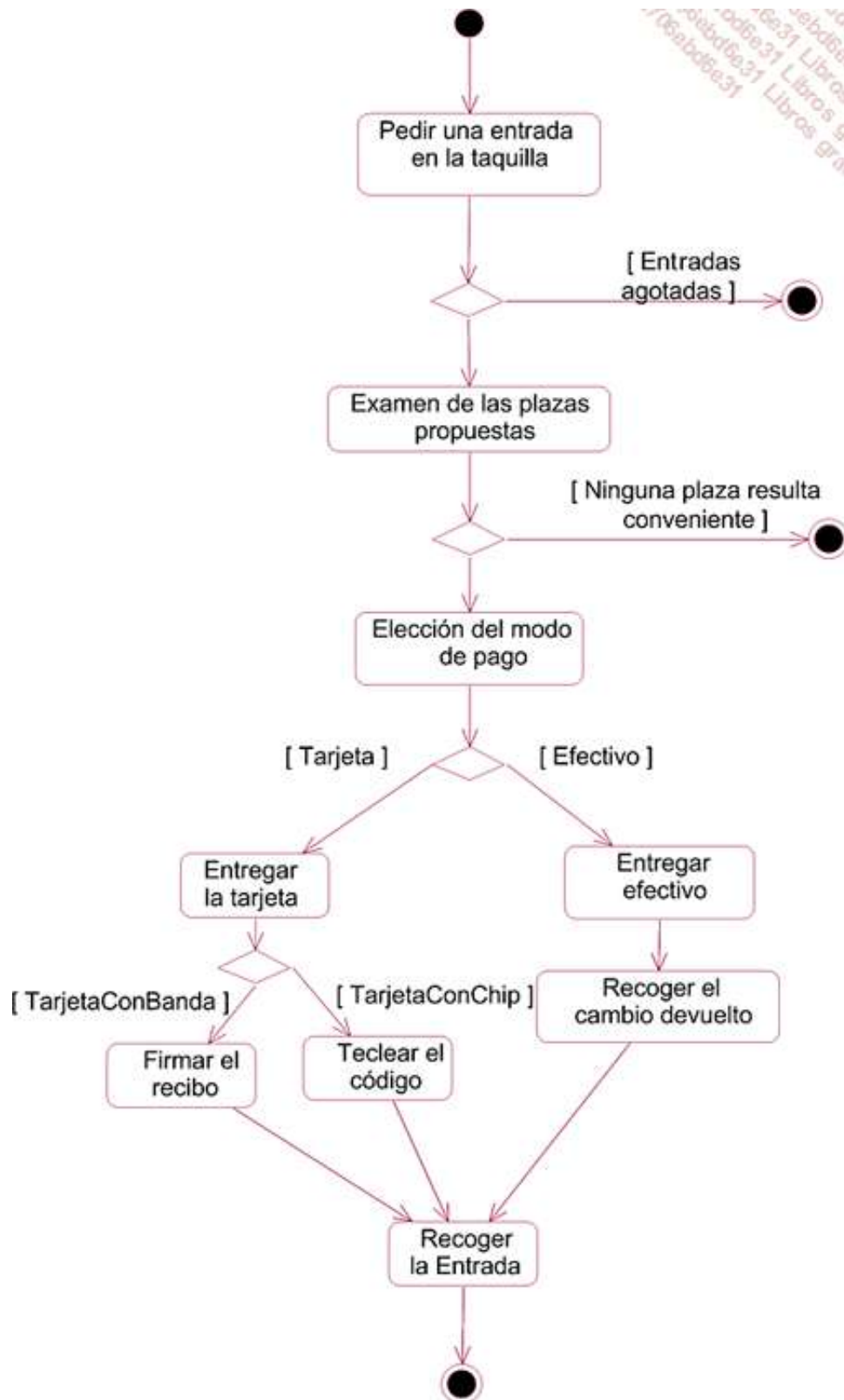




# Capítulo Modelado de las actividades

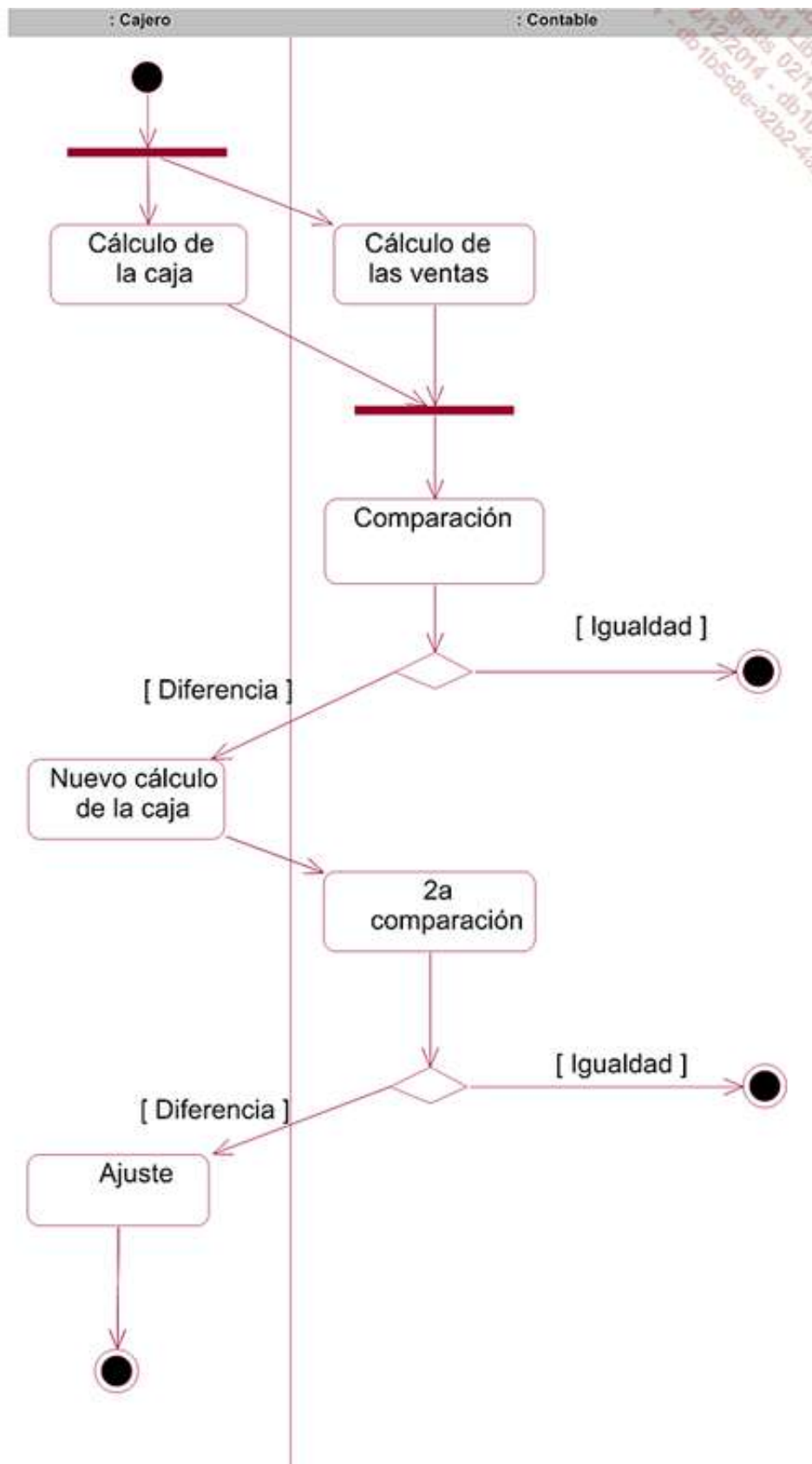
## 1. El espectáculo ecuestre

Construya el diagrama de actividades de compra de una entrada para un espectáculo ecuestre.



## 2. La apuesta trifecta

Construya el diagrama de actividades de comprobación de la caja de una taquilla de apuestas de trifecta (sólo la parte referente a la venta de boletos, sin tener en cuenta el pago de las ganancias).



# Glosario

## Actividad

Una actividad es una serie de acciones. Una acción consiste en asignar un valor a un atributo, crear o destruir un objeto, efectuar una operación, enviar una señal a otro objeto o a sí mismo, etc.

## Actividad compuesta

El contenido de una actividad compuesta está formado por otras actividades.

## Actor

Un actor representa a un usuario de un caso de uso en su papel dentro del sistema. El nombre del actor es el nombre del papel.

Debemos distinguir dos categorías de actores:

- Los actores primarios, para los cuales el objetivo del caso de uso es esencial.
- Los actores secundarios, que interactúan con el caso de uso, pero cuyo objetivo no es esencial.

## Agregación o composición débil

La agregación es la asociación que une un objeto compuesto a sus componentes. Se denomina débil por dos motivos: los componentes pueden pertenecer a otros objetos compuestos y la destrucción del objeto compuesto no comporta la destrucción de sus componentes.

## Alternativa

En un diagrama de secuencia, la alternativa es uno de los operadores de un marco de interacción. Está asociada a una condición. Si la condición se cumple, el contenido del marco se ejecuta.

En un diagrama de actividades, la alternativa sirve para seleccionar la actividad siguiente. Cada rama de la alternativa está dotada de una condición de guarda que excluye las demás condiciones.

## Artefacto

Un artefacto está constituido por la forma física de un software. Un archivo ejecutable, una biblioteca compartida o un script son ejemplos de formas físicas de software.

## Asociación entre objetos

Una asociación entre objetos es un conjunto de vínculos entre las instancias de dos o más clases. Se describe en el diagrama de clases.

## Asociaciones entre empaquetados

Existen dos asociaciones entre empaquetados:

- La asociación de importación consiste en llevar hasta el empaquetado de destino un elemento del empaquetado de origen. El elemento forma parte entonces de los elementos visibles del empaquetado de destino.
- La asociación de acceso consiste en acceder desde el empaquetado de destino a un elemento del empaquetado de origen. El elemento no forma parte entonces de los elementos visibles del empaquetado de destino.

## Asociación reflexiva

Una asociación reflexiva une entre sí las instancias de una clase.

## Atributo calculado

El valor de un atributo calculado viene dado por una función basada en el valor de otros atributos.

## Atributo de clase

Un atributo de clase está vinculado a la clase misma y no a cada una de las instancias. Estos atributos son compartidos por todas las instancias de la clase.

## **Bucle**

En un diagrama de secuencia, el bucle es uno de los operadores de un marco de interacción. Consiste en una ejecución repetida del contenido del marco hasta que la condición final se cumpla o hasta que se alcance un número máximo de repeticiones.

## **Calificación**

Una asociación puede calificarse en un extremo para reducir la cardinalidad máxima en el extremo opuesto. El valor del calificador se toma en cuenta para determinar el número de vínculos.

## **Calle (o partición)**

Una calle agrupa todas las actividades bajo responsabilidad de un mismo objeto.

## **Cardinalidad mínima o máxima**

La cardinalidad se fija en uno de los extremos de una asociación. La cardinalidad mínima (o máxima) permite establecer el número mínimo (o máximo) de instancias a las que está vinculada una instancia de la clase situada en el extremo opuesto de la asociación.

## **Caso de uso**

Un caso de uso describe las interacciones entre un usuario y el sistema.

En los casos de uso con objetivo usuario, esta serie de interacciones está vinculada a un objetivo funcional del usuario.

En los casos de uso de subfunción, la serie de interacciones está destinada a ser incluida en otro caso de uso.

## **Ciclo de vida**

El ciclo de vida de un objeto es el conjunto de sus estados y de las transiciones que los unen.

## **Clase**

Las clases están formadas por conjuntos de objetos similares con los mismos atributos y métodos. Esta representación común se define en común en el ámbito de la clase.

Las clases concretas definen modelos completos y poseen instancias directas.

Las clases abstractas definen modelos abstractos y no poseen instancias directas. Estas clases, en tanto que superclases, sirven para factorizar atributos y métodos comunes de varias clases concretas.

Las clases-asociaciones son a la vez asociaciones y clases cuyas instancias son los elementos de la asociación. De esta forma, los elementos pueden estar dotados de atributos o de operaciones.

## **Componente**

Un componente es una unidad de software que ofrece servicios a través de una o varias interfaces. Es una caja negra cuyo contenido no interesa a sus clientes.

## **Composición**

La composición (o composición fuerte) es la asociación que vincula a un objeto con sus componentes. Decimos que es fuerte por dos motivos: los componentes no pueden pertenecer a otros objetos compuestos y la destrucción del objeto compuesto comporta la destrucción de sus componentes.

## **Condición de guarda**

Las condiciones de guarda se utilizan en los diagramas de comunicación, de estados-transiciones y de actividades. Son condiciones para enviar un mensaje, traspasar la transición o encadenar las actividades respectivamente.

## **Diagrama de actividades**

Este tipo de diagrama describe las actividades de uno o de varios objetos así como sus encadenamientos.

## **Diagrama de casos de uso**

Describe el conjunto (o subconjunto) de casos de uso y de actores de un sistema así como las asociaciones que los unen.

## **Diagrama de clases**

Describe el conjunto (o subconjunto) de clases e interfaces de un sistema así como las asociaciones que los unen.

## **Diagrama de componentes**

Muestra la estructuración en componentes de software de un sistema.

## **Diagrama de comunicación**

Describe las interacciones entre un conjunto de objetos mostrando, de manera espacial, los envíos de mensajes que se producen entre ellos.

## **Diagrama de despliegue**

Describe la arquitectura material del sistema.

## **Diagrama de empaquetado**

Este tipo de diagrama es una agrupación de elementos de modelado.

## **Diagrama de estados-transiciones**

Muestra el conjunto de estados del ciclo de vida de un objeto separados por transiciones.

## **Diagrama de objetos**

Muestra, en un momento determinado, las instancias creadas y sus vínculos cuando el sistema está activo.

## **Diagrama de secuencia**

Describe las interacciones entre un conjunto de objetos mostrando los envíos de mensajes que se producen entre ellos de manera secuencial.

## **Diagrama de timing**

El diagrama de timing muestra los cambios de estado de un objeto en función del tiempo.

## **Diagrama de vista de conjunto de las interacciones**

El diagrama de vista de conjunto de las interacciones es un diagrama de actividades en el que cada actividad puede ser descrita por un diagrama de secuencia.

## **Elemento de una asociación**

Un elemento de una asociación es un vínculo entre las instancias de las clases situadas en los extremos de la misma.

## **Empaquetado**

Un empaquetado es una agrupación de elementos de modelado: clases, componentes, casos de uso, otros empaquetados, etc.

## **Encadenamiento de actividades**

Un encadenamiento de actividades es un vínculo desde una actividad de origen a una actividad de destino. Se traspasa al concluir la actividad de origen.

## **Encapsulación**

Consiste en ocultar la estructura y el comportamiento internos y propios del funcionamiento del objeto. Esta ocultación puede ser completa (encapsulación privada), no aplicarse a las subclases (encapsulación protegida) o no aplicarse a las clases del mismo empaquetado (encapsulación de empaquetado).

## **Envío de mensajes**

*Véase Mensaje.*

## **Escenario**

Un escenario es una instancia de un caso de uso con todas las alternativas establecidas.

## **Especialización**

La especialización es la relación que vincula a una superclase con una de sus subclases.

La especialización se aplica también a los casos de uso.

## **Especificaciones sobre la relación de herencia**

Existen cuatro especificaciones sobre la relación de herencia entre una superclase y sus subclases:

- {incomplete}: las subclases son incompletas y no cubren la totalidad de la superclase, es decir, las instancias de las subclases son un subconjunto de las instancias de la superclase.
- {complete}: las subclases son completas y cubren la totalidad de la superclase.
- {disjoint}: las subclases no tienen ninguna instancia en común.
- {overlapping}: las subclases pueden tener una o varias instancias en común.

## **Estado**

Los estados de un objeto corresponden a momentos de su ciclo de vida. Mientras permanecen en un estado, los objetos pueden realizar una actividad o bien esperar una señal procedente de otros objetos.

## **Estereotipo**

Un estereotipo es una palabra clave usada para explicitar la especialización de un elemento. Los estereotipos se escriben entre comillas.

## **Generalización**

La generalización es la relación que une a una subclase con su superclase (o con una de las superclases, en caso de herencia múltiple).

La generalización también se aplica a los casos de uso.

## **Granulado**

El granulado de un objeto representa el tamaño del mismo. Un objeto de tamaño pequeño es un objeto de granulado fino o grano pequeño. Un objeto voluminoso es un objeto de granulado considerable o de grano grueso.

## **Herencia**

La herencia es la propiedad que hace que una subclase se beneficie de la estructura y del comportamiento de su superclase.

Cuando una subclase posee varias superclases hablamos de herencia múltiple.

## **Instancia**

Una instancia de una clase es un elemento del conjunto de objetos de la clase.

## **Interfaz**

Una interfaz es una clase abstracta que sólo contiene las firmas de métodos. La firma de un método está formada por su nombre y sus parámetros.

Una interfaz suministrada describe los servicios ofrecidos por un componente. Una interfaz necesaria describe los servicios que un componente espera de otro componente del cual es cliente.

### **Línea de vida**

Dentro de un diagrama de secuencia, la línea de vida muestra las acciones y reacciones de una instancia así como los periodos durante los cuales está activa.

### **Marco de interacción**

Un marco de interacción es una parte del diagrama de secuencia asociada a una etiqueta con un operador que determina la modalidad de ejecución. Las principales modalidades son la bifurcación condicional y el bucle.

### **MDA**

MDA (*Model Driven Architecture* o arquitectura dirigida por modelos) es una propuesta de la OMG cuyo objetivo es diseñar sistemas basados sólo en el modelado del dominio, independientemente de la plataforma.

### **Mensaje**

Los mensajes se envían a los objetos para activarlos y provocar la ejecución del método del mismo nombre. Los envíos de mensajes son llamadas al método.

Los mensajes pueden enviarse de manera asincrónica. En ese caso, el que realiza la llamada espera a que concluya la ejecución del método del objeto receptor para continuar su ejecución.

También pueden enviarse de manera sincrónica. En ese caso el que realiza la llamada continúa la ejecución inmediatamente después del envío del mensaje.

### **Método**

Los métodos son conjuntos de instrucciones que toman valores en la entrada y modifican los valores de los atributos o producen un resultado.

El conjunto de métodos de una clase describe el comportamiento de las instancias de la misma.

### **Método de clase**

Los métodos de clase están vinculados a la propia clase y no a una instancia. Se invocan a través de la clase y no a través de una de sus instancias.

### **Navegación**

La navegación de una asociación determina el sentido del recorrido de la misma.

### **Nodo**

Un nodo es una unidad material capaz de recibir y ejecutar software.

### **Objeto**

Los objetos son entidades identificables del mundo real. En el modelo UML, los objetos son instancias de una clase.

### **OCL**

OCL (*Object Constraint Language* o lenguaje de especificaciones orientadas a objetos) es un lenguaje destinado a expresar las especificaciones en un diagrama de clases en forma de condiciones lógicas.

### **OMG**

El OMG o *Object Management Group* es un consorcio formado por más de 800 sociedades y universidades cuyo objetivo es promover las tecnologías orientadas a objetos.

## **Partición**

Véase *Calle*.

## **PIM**

El PIM (*Platform Independent Model* o modelo independiente de la plataforma) es el modelo de diseño de la arquitectura MDA.

## **Polimorfismo**

El polimorfismo es la diferencia de comportamiento existente entre las subclases de una misma superclase para métodos del mismo nombre.

## **Proceso**

Un proceso es un conjunto de operaciones que toman datos en entrada y producen nuevos datos.

## **Proceso unificado**

El Proceso Unificado es un proceso de diseño y evolución de software basado en UML.

## **PSM**

El PSM (*Platform Specific Model* o modelo específico de la plataforma) es el modelo destino de la arquitectura MDA.

## **Relación de comunicación**

La relación de comunicación vincula a los actores con los casos de uso.

## **Relación de extensión**

La relación de extensión permite enriquecer un caso de uso mediante un caso de uso de subfunción. Dicho enriquecimiento es opcional.

## **Relación de inclusión**

La relación de inclusión permite enriquecer un caso de uso mediante un caso de uso de subfunción. Dicho enriquecimiento es obligatorio.

## **Relación de realización**

La realización de una interfaz, es decir, la implantación de sus métodos se confía a una o a varias clases concretas, subclases de la interfaz. La relación de herencia existente entre una interfaz y una subclase de implantación se conoce como relación de realización.

Esta relación existe asimismo entre una interfaz y un componente que implanta sus métodos.

## **Tipo**

El tipo puede ser una clase o un tipo estándar. Los tipos estándar se designan como sigue:

- Integer para el tipo de los enteros.
- String para el tipo de las cadenas de caracteres.
- Boolean para el tipo de los booleanos.
- Real para el tipo de los reales.

## **Transición**

Una transición es un vínculo orientado entre dos estados que expresa el hecho de que el objeto puede pasar del estado inicial de la transición al estado final.

## **UML**

UML (*Unified Modeling Language* o lenguaje unificado de modelado) es un lenguaje gráfico destinado a



modelar sistemas y procesos.

## Léxico Español-inglés

Abstracto	<i>abstract</i>
Actividad	<i>activity</i>
Actividad compuesta	<i>composite activity</i>
Actor	<i>actor</i>
Agregación (o composición débil)	<i>aggregation (or weak composition)</i>
Alternativa	<i>choice</i>
Artefacto	<i>artifact</i>
Asociación reflexiva	<i>reflexive association</i>
Atributo calculado	<i>derived attribute</i>
Atributo de clase	<i>class attribute</i>
Booleano	<i>boolean</i>
Bucle	<i>loop</i>
Cadena de caracteres	<i>string</i>
Calificación	<i>qualification</i>
Calificador	<i>qualifier</i>
Calle o partición	<i>swimlane</i>
Cardinalidad mínima, máxima	<i>minimal, maximal multiplicity</i>
Caso de uso	<i>use case</i>
Ciclo de vida	<i>lifecycle</i>
Clase	<i>class</i>
Columna	<i>column</i>
Componente	<i>component</i>
Composición (fuerte)	<i>(strong) composition</i>
Condición de guarda	<i>guard condition</i>
Dependencia	<i>dependency</i>
Diagrama de actividades	<i>activity diagram</i>
Diagrama de caso de uso	<i>use case diagram</i>
Diagrama de clases	<i>class diagram</i>
Diagrama de componentes	<i>component diagram</i>
Diagrama de comunicación	<i>communication diagram</i>
Diagrama de despliegue	<i>deployment diagram</i>
Diagrama de empaquetado	<i>package diagram</i>
Diagrama de estados-transiciones	<i>statechart diagram or state diagram</i>
Diagrama de estructura compuesta	<i>composite structure diagram</i>
Diagrama de interacción	<i>interaction diagram</i>
Diagrama de objetos	<i>object diagram</i>

Diagrama de perfil	<i>profile diagram</i>
Diagrama de secuencia	<i>sequence diagram</i>
Diagrama de timing	<i>timing diagram</i>
Diagrama de vista de conjunto de las interacciones	<i>interaction overview diagram</i>
Empaquetado	<i>package</i>
Encadenamiento de actividades	<i>activity edge</i>
Encapsulación	<i>encapsulation</i>
Entero	<i>encapsulation</i>
Envío de mensaje	<i>message sending</i>
Escenario	<i>scenario</i>
Especialización	<i>specialisation</i>
Especificación de la relación de herencia	<i>inheritance relationship constraint</i>
Estado	<i>state</i>
Estereotipo	<i>stereotype</i>
Falso	<i>false</i>
Fila	<i>row</i>
Generalización	<i>generalisation</i>
Granulado	<i>granularity</i>
Herencia	<i>inheritance</i>
Instancia	<i>instance</i>
Interfaz	<i>interface</i>
Lenguaje de especificaciones orientadas a objetos (OCL)	<i>Object Constraint Language (OCL)</i>
Lenguaje unificado de modelado (UML)	<i>Unified Modeling Language (UML)</i>
Línea de vida	<i>lifeline</i>
Marco de interacción	<i>interaction fragment</i>
MDA (Arquitectura guiada por modelos)	<i>MDA (Model Driven Architecture)</i>
Mensaje	<i>message</i>
Método	<i>method</i>
Método de clase	<i>class method</i>
Modelo específico de la plataforma (PSM)	<i>Platform Specific Model (PSM)</i>
Modelo independiente de la plataforma (PIM)	<i>Platform Independent Model (PIM)</i>
Navegación	<i>navigation</i>
Nodo	<i>node</i>
Objeto	<i>object</i>
Perfil	<i>profile</i>
Polimorfismo	<i>polymorphism</i>

Proceso	<i>process</i>
Proceso Unificado	<i>Unified Process (UP)</i>
Relación de comunicación	<i>communication relationship</i>
Relación de extensión	<i>extension relationship</i>
Relación de inclusión	<i>inclusion relationship</i>
Relación de realización	<i>realisation relationship</i>
Si	<i>if</i>
Si no	<i>else</i>
Sistema	<i>system</i>
Tipo	<i>type</i>
Transición	<i>transition</i>
Verdadero	<i>true</i>
Vínculo (entre objetos)	<i>link (between objects)</i>

## Léxico Inglés-español

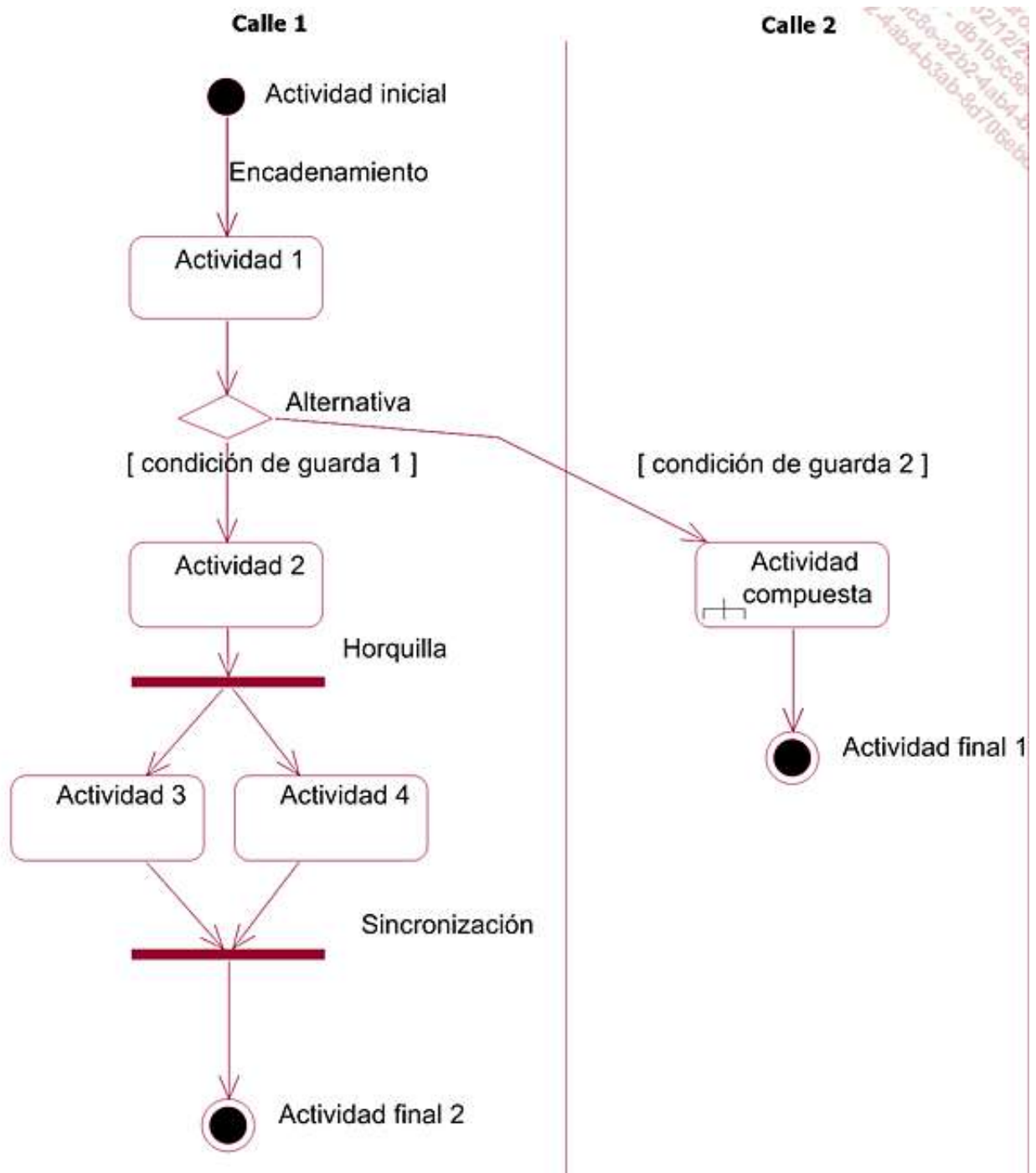
<i>Abstract</i>	abstracto
<i>Activity</i>	actividad
<i>Activity diagram</i>	diagrama de actividades
<i>Activity edge</i>	encadenamiento de actividades
<i>Actor</i>	actor
<i>Aggregation (or weak composition)</i>	agregación (o composición débil)
<i>Artifact</i>	artefacto
<i>Boolean</i>	booleano
<i>Choice</i>	alternativa
<i>Class</i>	clase
<i>Class attribute</i>	atributo de clase
<i>Class diagram</i>	diagrama de clases
<i>Class method</i>	método de clase
<i>Column</i>	columna
<i>Communication diagram</i>	diagrama de comunicación
<i>Communication relationship</i>	relación de comunicación
<i>Component</i>	componente
<i>Component diagram</i>	diagrama de componentes
<i>Composite activity</i>	actividad compuesta
<i>Composite Structure Diagram</i>	diagrama de estructura compuesta
<i>Composition (strong composition)</i>	composición (fuerte)
<i>Dependency</i>	dependencia
<i>Deployment diagram</i>	diagrama de despliegue
<i>Derived attribute</i>	atributo calculado
<i>Else</i>	si no
<i>Encapsulation</i>	encapsulación
<i>Extension relationship</i>	relación de extensión
<i>False</i>	falso
<i>Generalisation</i>	generalización
<i>Granularity</i>	granulado
<i>Guard condition</i>	condición de guarda
<i>If</i>	si
<i>Inclusion relationship</i>	relación de inclusión
<i>Inheritance</i>	herencia
<i>Inheritance relationship constraint</i>	especificación de la relación de herencia
<i>Instance</i>	instancia

<i>Integer</i>	entero
<i>Interaction diagram</i>	diagrama de interacción
<i>Interaction fragment</i>	marco de interacción
<i>Interaction overview diagram</i>	diagrama de vista de conjunto de las interacciones
<i>Interface</i>	interfaz
<i>Lifecycle</i>	ciclo de vida
<i>Lifeline</i>	línea de vida
<i>Link (between objects)</i>	vínculo (entre objetos)
<i>Loop</i>	bucle
<i>MDA (Model Driven Architecture)</i>	MDA (Arquitectura guiada por modelos)
<i>Message</i>	mensaje
<i>Message sending</i>	envío de mensaje
<i>Method</i>	método
<i>Multiplicity (minimal, maximal multiplicity)</i>	cardinalidad mínima, máxima
<i>Navigation</i>	navegación
<i>Node</i>	nodo
<i>Object</i>	objeto
<i>Object Constraint Language (OCL)</i>	lenguaje de especificaciones orientadas a objetos (OCL)
<i>Object diagram</i>	diagrama de objetos
<i>Package</i>	empaquetado
<i>Package diagram</i>	diagrama de empaquetado
<i>Platform Independent Model (PIM)</i>	modelo independiente de la plataforma (PIM)
<i>Platform Specific Model (PSM)</i>	modelo específico de la plataforma (PSM)
<i>Polymorphism</i>	polimorfismo
<i>Process</i>	proceso
<b>Profile</b>	perfil
<i>Profile diagram</i>	diagrama de perfil
<i>Qualification</i>	calificación
<i>Qualifier</i>	calificador
<i>Realisation relationship</i>	relación de realización
<i>Reflexive association</i>	asociación reflexiva
<i>Row</i>	Fila
<i>Scenario</i>	escenario
<i>Sequence diagram</i>	diagrama de secuencia
<i>Specialisation</i>	especialización
<i>State</i>	estado

<i>Statechart diagram (or state diagram)</i>	diagrama de estados-transiciones
<i>Stereotype</i>	estereotipo
<i>String</i>	cadena de caracteres
<i>Swimlane</i>	calle o partición
<i>System</i>	sistema
<i>Timing diagram</i>	diagrama de timing
<i>Transition</i>	transición
<i>True</i>	verdadero
<i>Type</i>	tipo
<i>Unified Modeling Language (UML)</i>	lenguaje unificado de modelado (UML)
<i>Unified Process (UP)</i>	Proceso Unificado
<i>Use case</i>	caso de uso
<i>Use case diagram</i>	diagrama de caso de uso

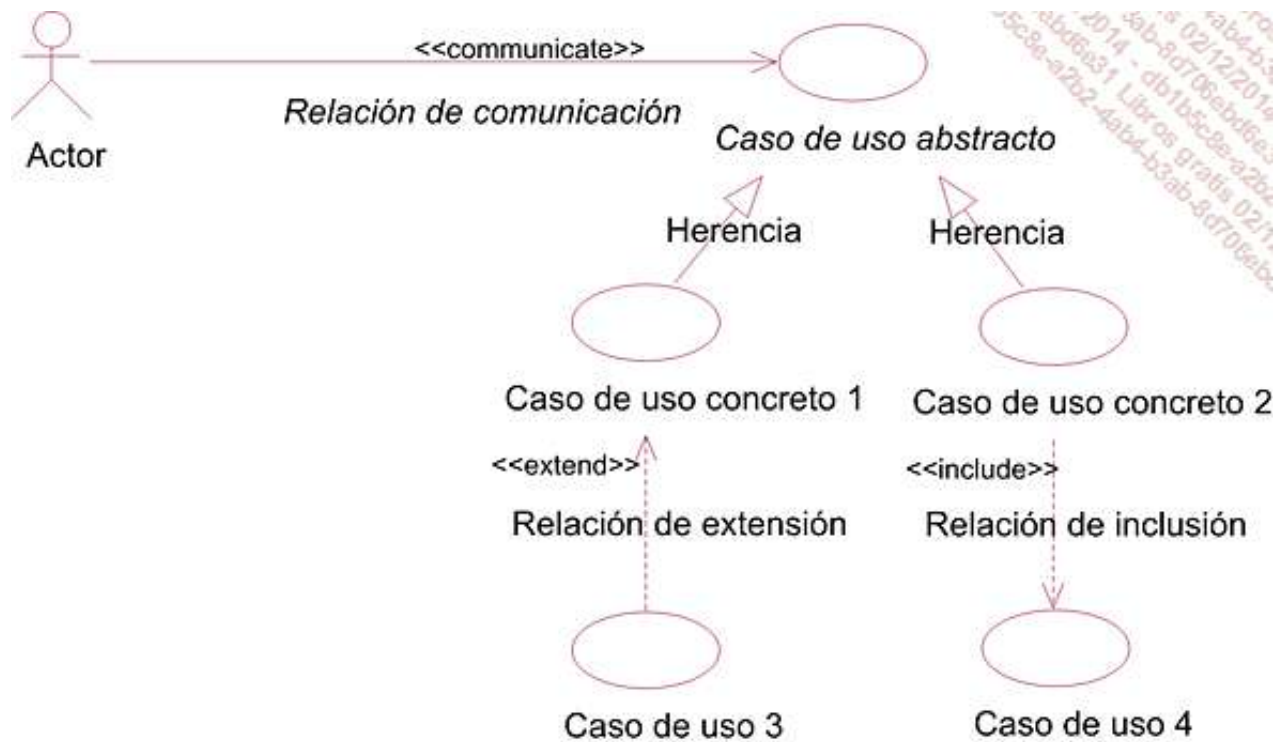
# Notación gráfica

## Diagrama de actividades

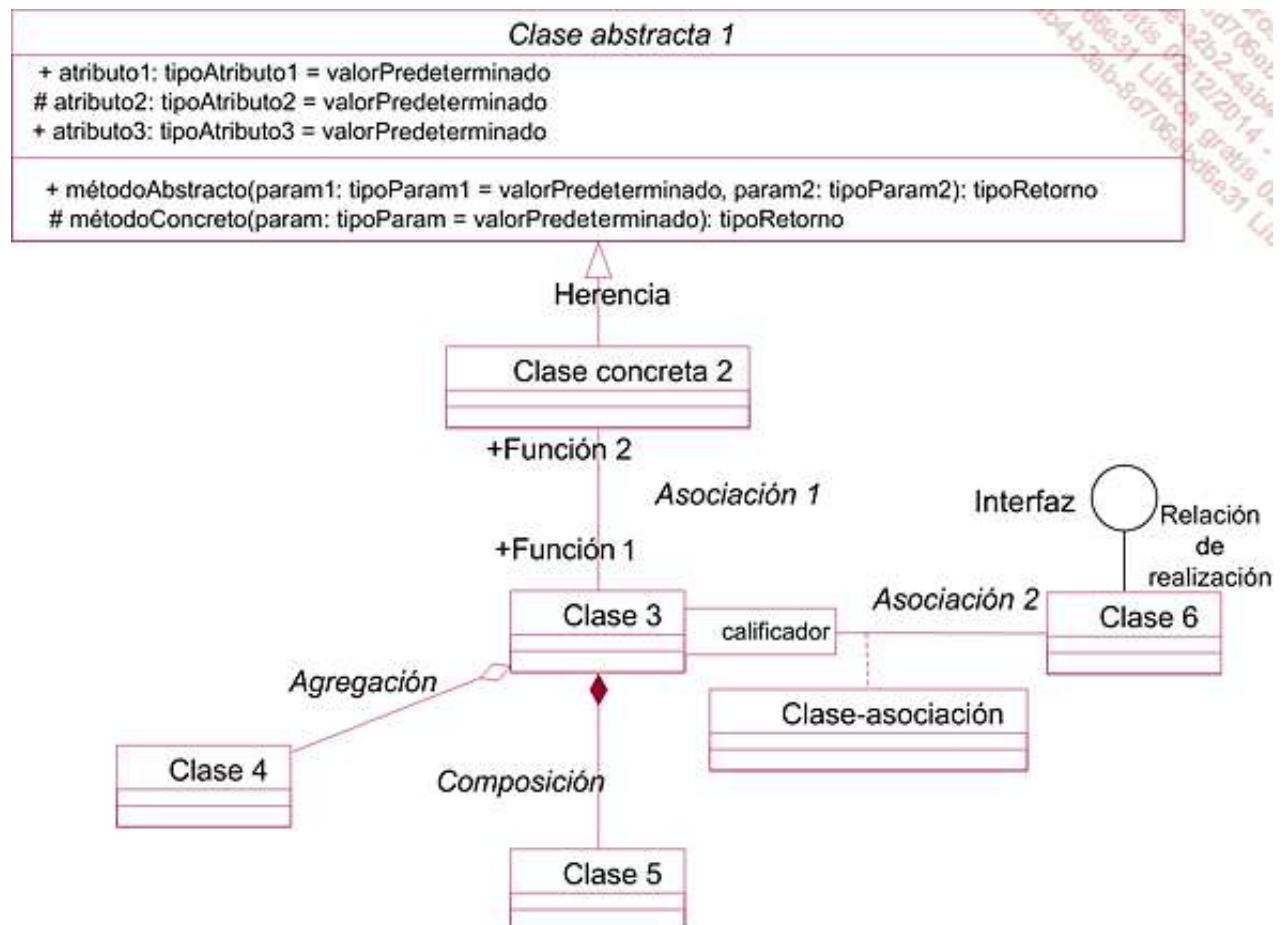


## Diagrama de casos de uso

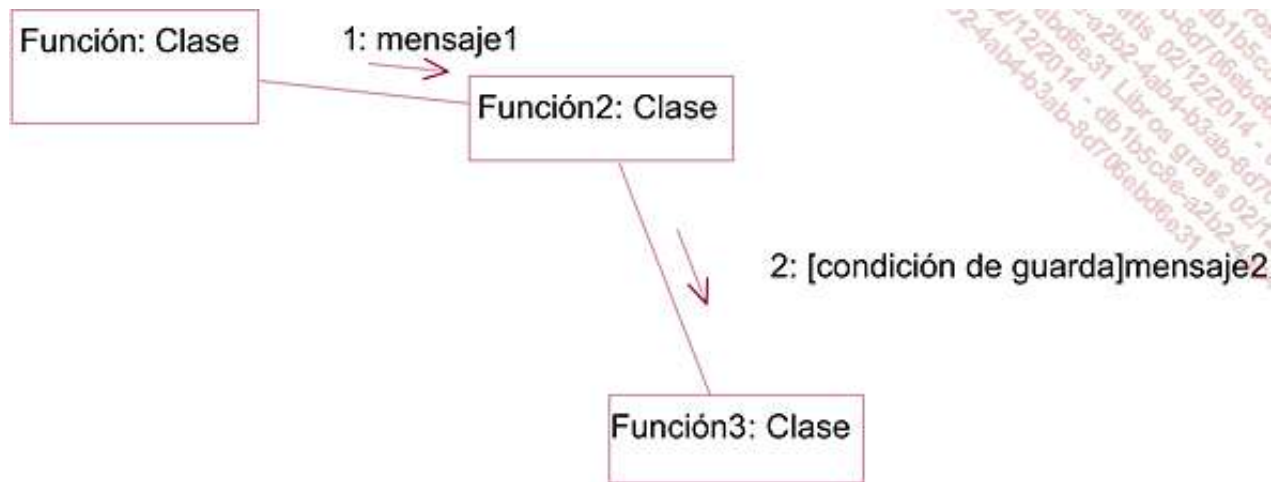




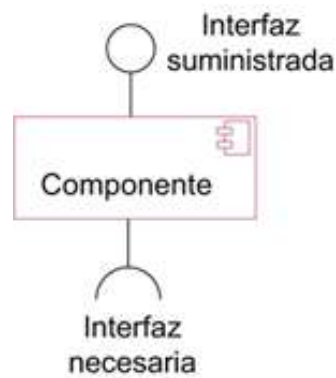
## Diagrama de clases



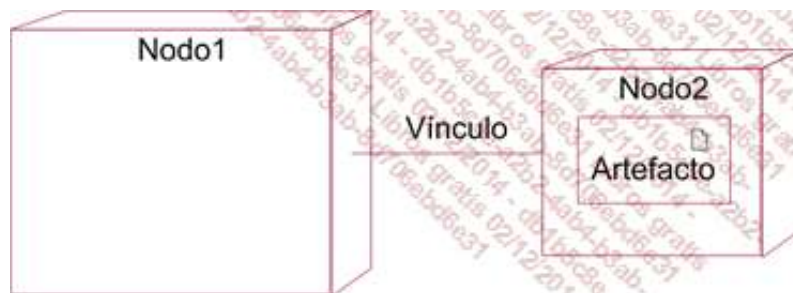
## Diagrama de comunicación



### Diagrama de componentes



### Diagrama de despliegue



### Diagrama de estados-transiciones

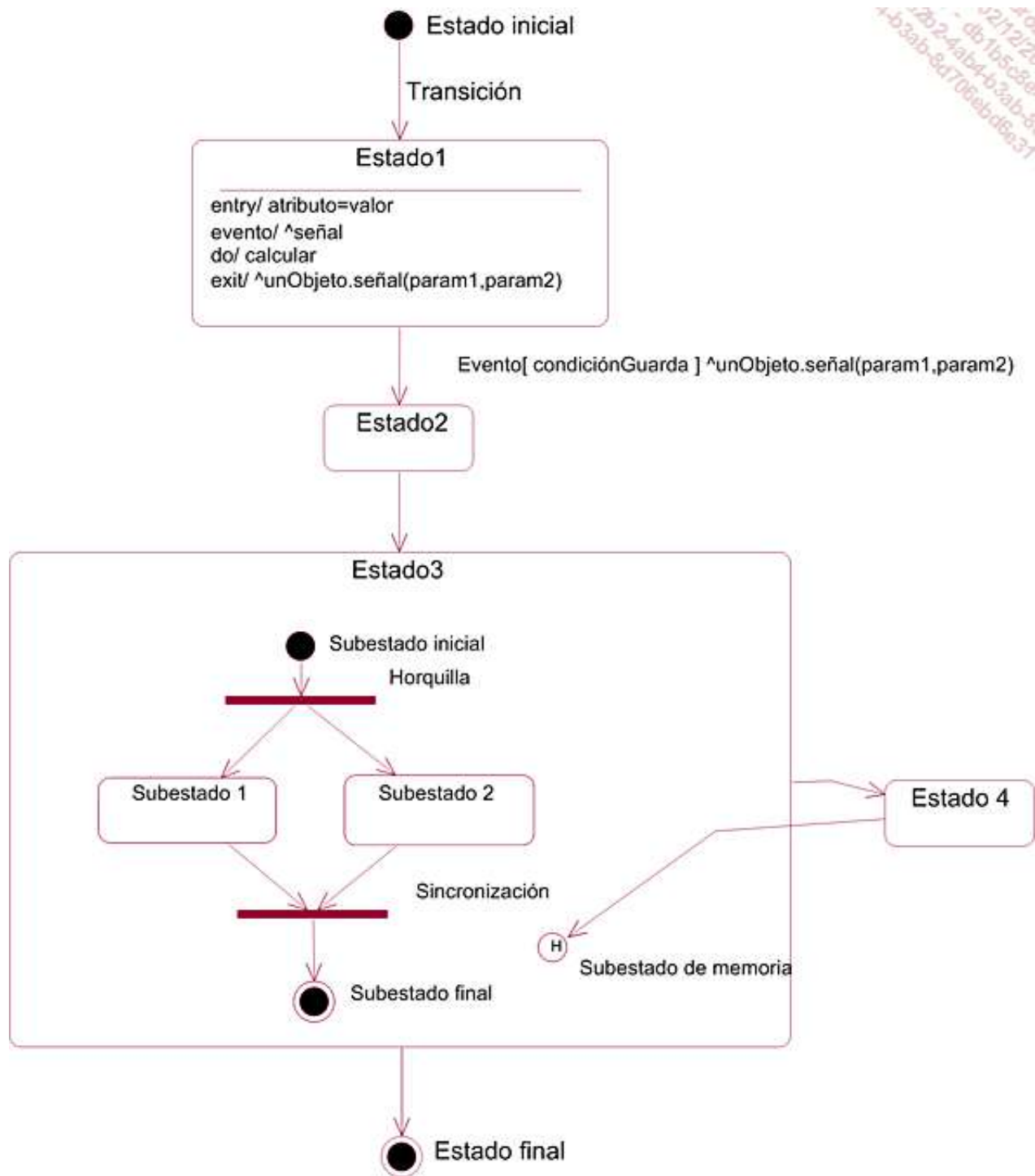
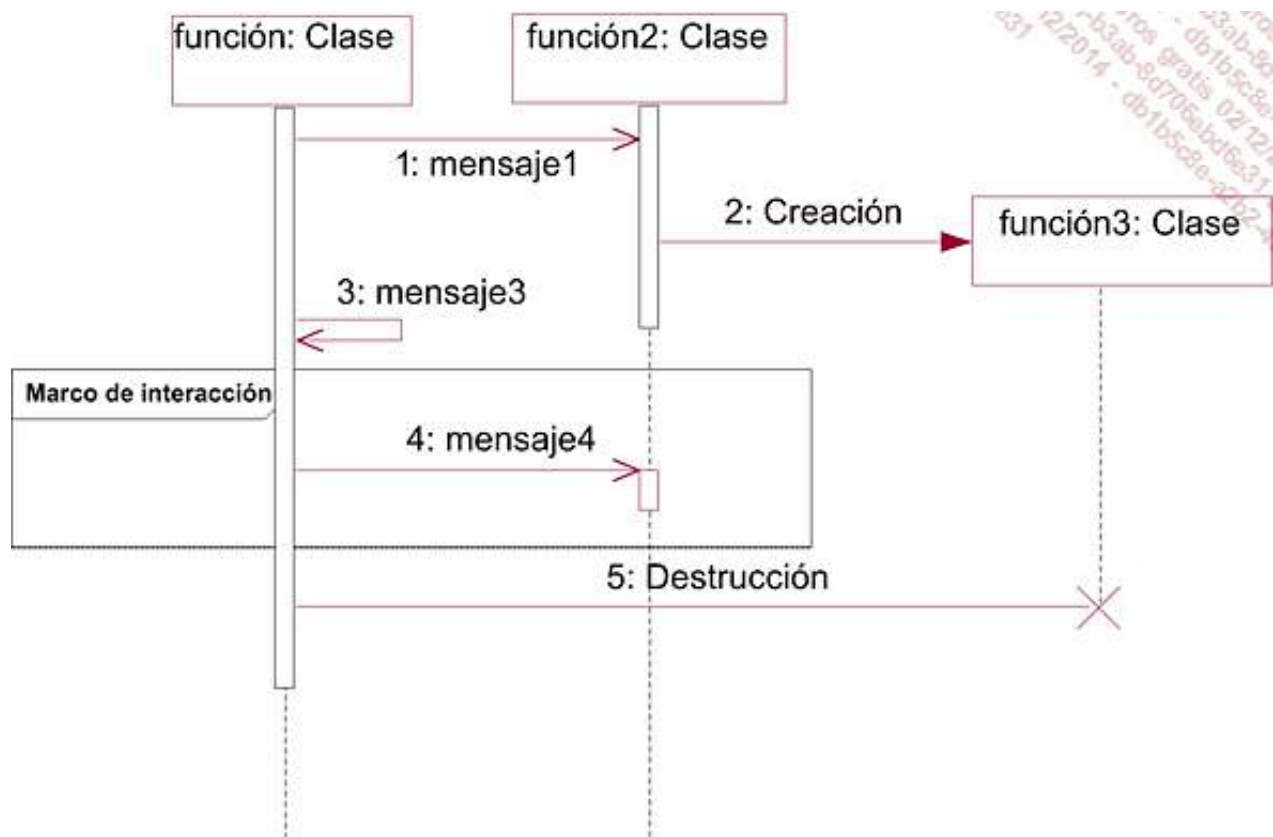


Diagrama de secuencia



## Bibliografía

Alistair Cockburn, *Rédiger des cas d'utilisation efficaces*, Eyrolles, 1999

Anneke Kleppe, Jos Warmer, Wim Bast, *MDA Explained: The Model Driven Architecture - Practice and Promise*, Addison-Wesley, 2003

Ivar Jacobson, Grady Booch, James Rumbaugh, *Le Processus unifié de développement logiciel*, Eyrolles, 2000

James Rumbaugh, Ivar Jacobson, Grady Booch, *Unified Modeling Language Reference Manual, Second Edition*, Addison-Wesley, 2004

Jos Warmer, Anneke Kleppe, *The Object Constraint Language: Getting Your Models ready for MDA, Second Edition*, Addison-Wesley, 2003

Kendal Scott, *Fast Track UML 2.0*, Apress, 2004

Object Management Group, *OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4*, ptc/2010-11-16

Object Management Group, *OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4*, ptc/2010-11-114

Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise, *MDA Distilled*, Addison-Wesley, 2004

Stephen J. Mellor, Marc J. Balcer, Stephen Mellor, Marc Balcer, *Executable UML: A Foundation for Model Driven Architecture*, Addison-Wesley, 2002

Tim Weilkiens and Bernd Oestereich, *UML 2 Certification Guide: Fundamental & Intermediate Exams*, The MK/OMG Press, 2006