# Ingeniería de software

Ingeniería en Sistemas ula

- **Modelado de sistemas**

Diagramas de clase
Diagramas de secuencia

Fuentes:
Taller de Ing. De requisitos de Prof. Judith Barrios
Clases de la prof. Isabel Besembel

# Lenguajes de modelado

Ingeniería en Sistemas ula

- **Un lenguaje de modelado consta de**
  - Un vocabulario:
    - Conjunto de símbolos (constructos) empleados para modelar
  - Una sintáxis:
    - Conjunto de reglas que describen como se usan los símbolos
  - Una semántica:
    - Describe el significado de los símbolos
- **Las notaciones son, también, medios para modelar; pero tienen una semántica menos rigurosa que los lenguajes**
- **Lenguajes de modelado más recientes:**
  - UML (*Unified Modeling Language*), UML Business
  - BPML (*Business Process Modeling Language*)
  - WebML (*Web Modeling Language*)

# UML

- **UML (*Unified Modeling Language*):**
  - Es un lenguaje de modelado de sistemas de software que integra y unifica diferentes notaciones y lenguajes formales
  - Facilita la representación del conocimiento acerca de un sistema y la comunicación de dicho conocimiento
- **Es un estándar administrado por el consorcio OMG**
  - *Object Management Group* (www.omg.org)
- **Ha evolucionado agregando mayor poder y capacidad semántica a cada nueva versión**
  - Versiones más recientes:
    - UML 2.0 (Junio 2003)
    - UML 2.5 (Septiembre 2013)
- **Inicio de los ochenta: Métodos orientados por objetos**

# UML

- **Es utilizado en la industria del software para:**
  - especificar,
  - diseñar,
  - visualizar,
  - comunicar y
  - documentar sistemas de software y aplicaciones
- **Consta de un conjunto de notaciones gráficas y un lenguaje formal**
  - Las notaciones gráficas son usadas para:
    - Modelar la **estructura**, **funcionalidad**, **comportamiento** e **implementación** de un sistema
    - Organizar los modelos producidos
  - El lenguaje formal es usado para expresar formalmente restricciones acerca de los elementos modelados de un sistema
    - OCL (*Object Constraint Language*) permite representar:
      - Invariantes
      - Pre y postcondiciones
      - Referencias nulas
      - Otras restricciones

# UML

- **Permite modelar un sistema de software desde diferentes perspectivas**
  - Funcional
    - Usos del sistema
  - Estructural
    - Abstracciones del sistema
      - Clases, objetos, relaciones e interacciones
  - Comportamiento
    - Dinámica del sistema
  - Implementación
    - Componentes del sistema
    - Despliegue (instalación) de los componentes del sistema

# UML 2.0 y posteriores

Ingeniería en Sistemas ula

- **Los 13 tipos de diagramas se clasifican en dos grupos:**
  - **Diagramas para Modelado Estructural**
    - Diagramas de clase
    - Diagramas de objetos
    - Diagramas de componentes
    - Diagramas de estructura compuesta
    - Diagramas de despliegue (*deployment*)
    - Diagramas de paquetes
  - **Diagramas para Modelado del Comportamiento:**
    - Diagramas de casos de uso
    - Diagramas de interacción:
      - Secuencias, Comunicación, Temporización, Vistas de Interacción
    - Diagramas de máquinas de estado
    - Diagramas de actividad

- **Documentos oficiales de UML 2.5 (www.omg.org/uml)**
  - UML 2.5 OMG, Septiembre 2013
  - UML 2.4 Object Constraint Language, Febrero 2014. Define el lenguaje formal OCL

# Diagramas de clases

Ingeniería en Sistemas ula

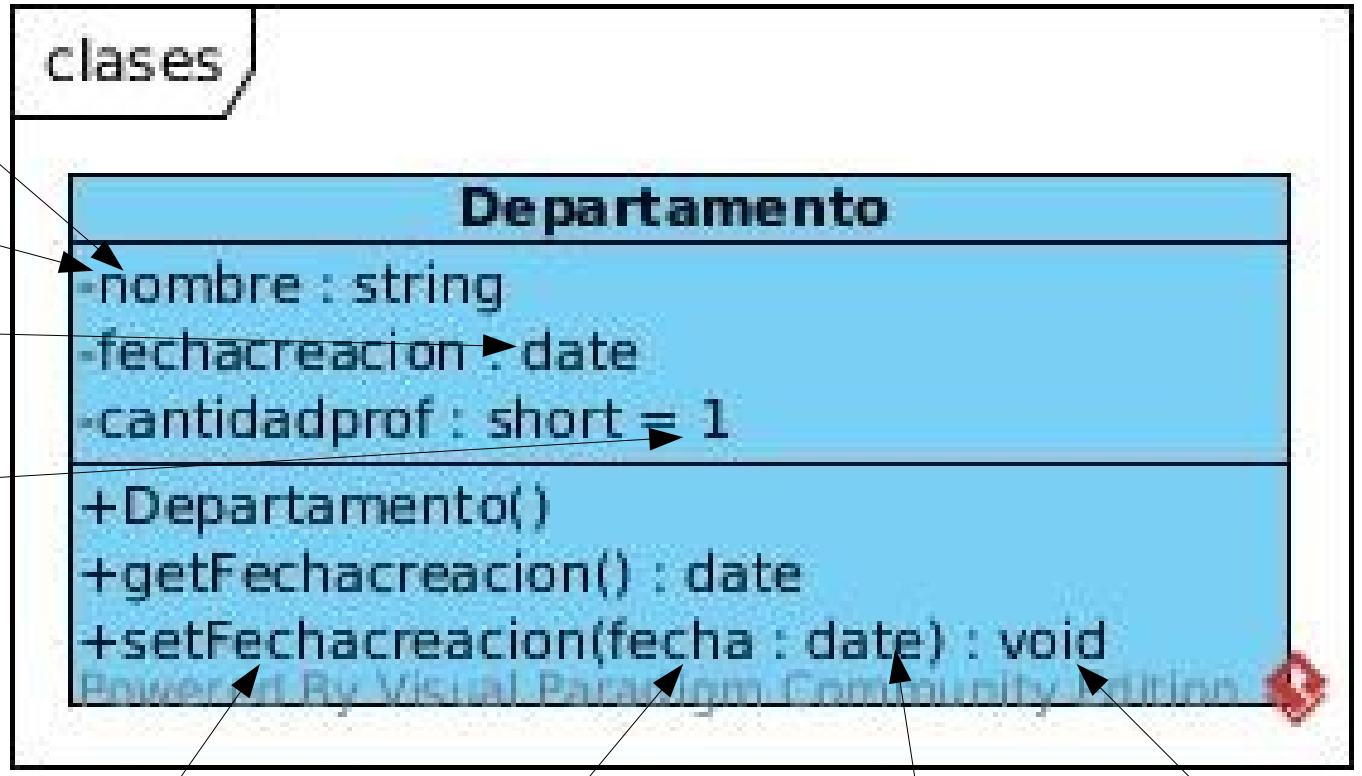
- Representa la estructura estática del sistema
- Muestran las clases y objetos del sistema y su estructura interna, además de las relaciones existentes entre ellos.
- Sirven para
  - definir y documentar modelos de datos,
  - Definir y documentar una solución de diseño (estructura del sistema)

# Clases

- Abstracción (representación estructural) de algo

- Definición de la estructura (atributos) y el comportamiento (métodos) de un conjunto de objetos que comparten el mismo patrón estructural y de comportamiento

# Clase




Nombre

Visibilidad

Tipo de dato

Valor por omisión

Métodos

Atributos

Nombre del método

Nombre del parámetro

Tipo de dato del parámetro

Valor de retorno

# Sintaxis de los atributos

- **[visibilidad] [/] nombre [:tipo] [multiplicidad] [=valor por omisión] [{propiedad}]**
  - visibilidad: ~ en el paquete, + público, # protegido, - privado
  - /: el valor del atributo puede ser derivado de los valores de 1 o más atributos
  - tipo: es el tipo de dato del atributo
  - multiplicidad:
    - Un valor fijo: 1, 4, 6
    - Muchos: *
    - Rango de valores, inferior..superior: 1..3, 3..*
    - Un conjunto de valores: [1, 3, 5, 7]
  - valor por omisión: cualquier valor inicial para el atributo
  - propiedad:
    - No cambia el valor del atributo: ***readOnly***

# Sintaxis de los métodos

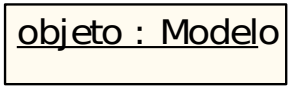
- **[visibilidad] nombre [(lista de parámetros)] [{propiedad}]**
  - **visibilidad:** ~ en el paquete, + público, # protegido, - privado
  - **lista de parámetros** de una operación separados por comas, cada uno con

[dirección] nombre: tipo [multiplicidad] [=valor por omisión]

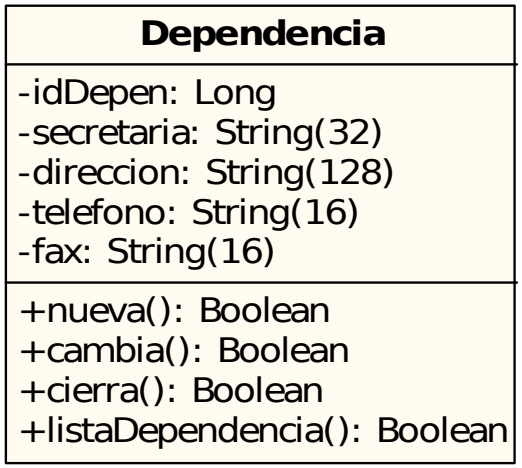
    - **dirección:**
      - Solo entrada: ***in***
      - Solo salida: ***out***
      - Entrada y salida: ***inout***
    - **multiplicidad:** igual que para los atributos
    - **valor por omisión:** igual que para los atributos
  - **propiedad:**
    - No cambia el valor de los atributos: ***isQuery***
    - Cambia el valor de algunos o todos los atributos:
      - Ejecución de las invocaciones en forma concurrente: ***concurrent***
      - Varias invocaciones pero ejecución secuencial: ***guarded***
      - Una invocación a la vez: ***sequential***

# Vista estructural

- **Objeto:** representación de algo que se describe mediante un *identificador*, una *estructura* y un *comportamiento*
- **Atributos:** propiedades relevantes de un objeto que representa su estructura. **Simples o compuestos**
- **Clasificación:** agrupación de objetos con propiedades y comportamiento similares dentro de una clase
- **Clase:** objeto que define la estructura y el comportamiento de un conjunto de objetos que tienen el mismo patrón estructural y de comportamiento
- **Instancia:** Cada objeto que pertenece a una clase

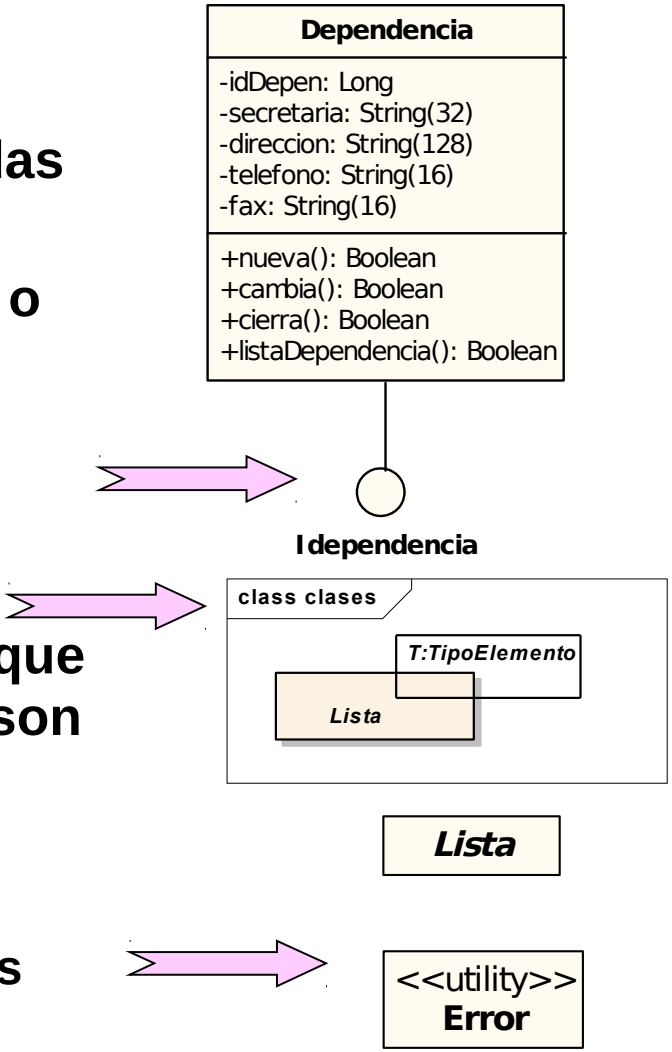






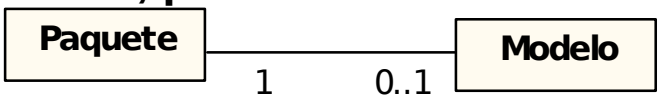
# Vista estructural

- **Extensión de clase:** conjunto de todas las instancias de una clase
- **Instanciación:** proceso de generación o creación de las instancias de una clase
- **Interfaz:** clase asociada que describe su comportamiento visible
- **Clases abstractas o parametrizables:** modelos de clases que se corresponden con clases genéricas (No son instanciables)
- **Clases utilitarias:** librerías de funciones



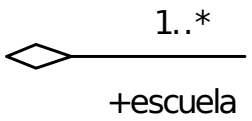
# Vista estructural

- **Jerarquía de clases:** **relación ES-UN(A), abstracciones de generalización/especialización de clases**
  - **Herencia:** **propiedad que tienen las clases de heredar de sus superclases estructura y/o comportamiento. Simple o múltiple.**
- **Asociaciones:** **relaciones estructurales entre las clases, pueden ser: reflexivas, binarias, etc.**



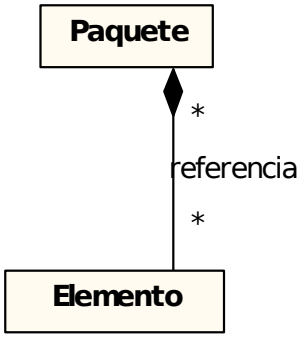



- **Agregaciones:**
  - Una clase forma parte de otra
  - Los valores de los atributos de una clase se propagan en los valores de los atributos de otra
  - Una acción sobre una clase implica una acción sobre otra
  - Los objetos de una clase están subordinados a los objetos de otra

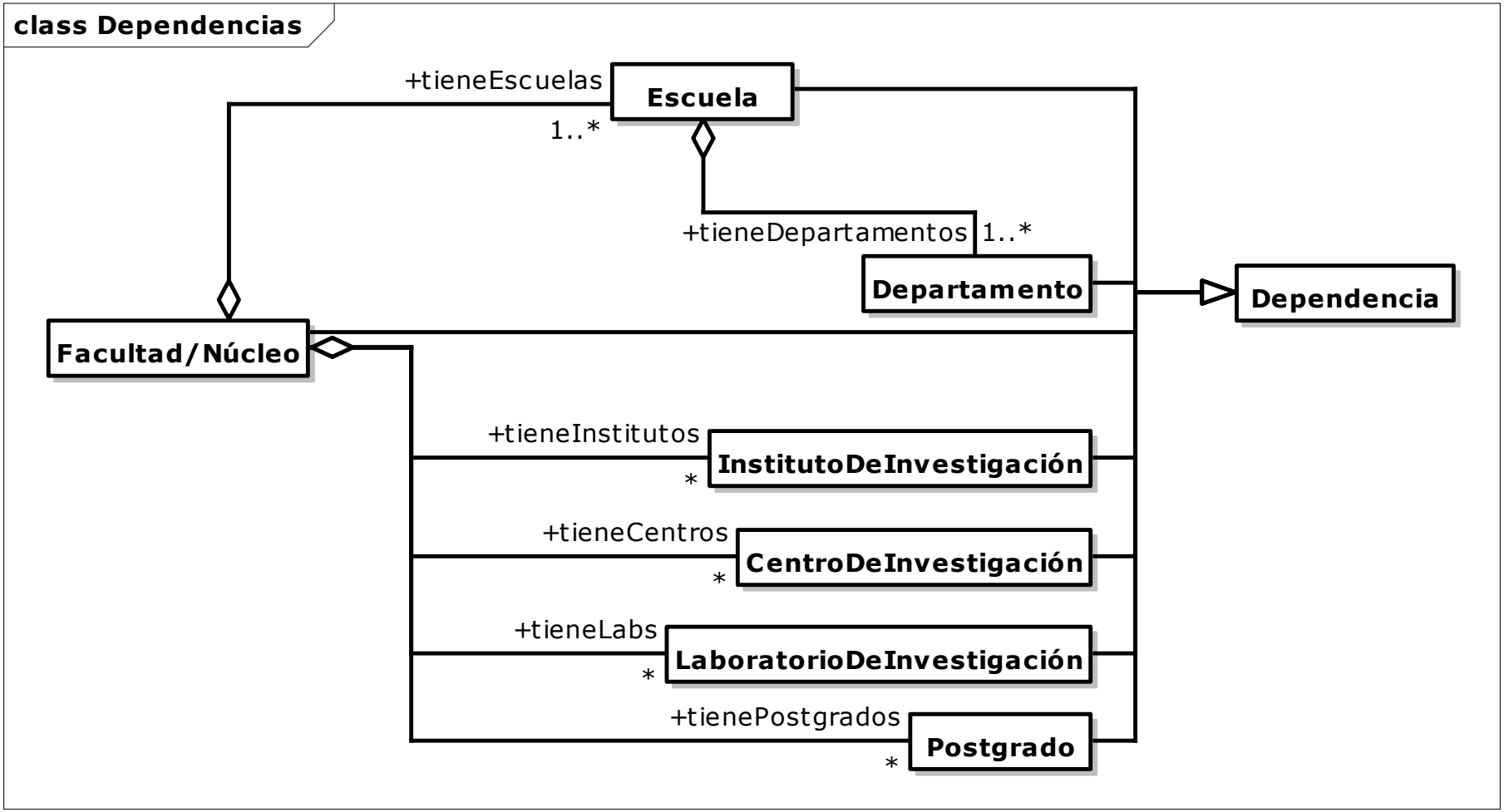


# Vista estructural

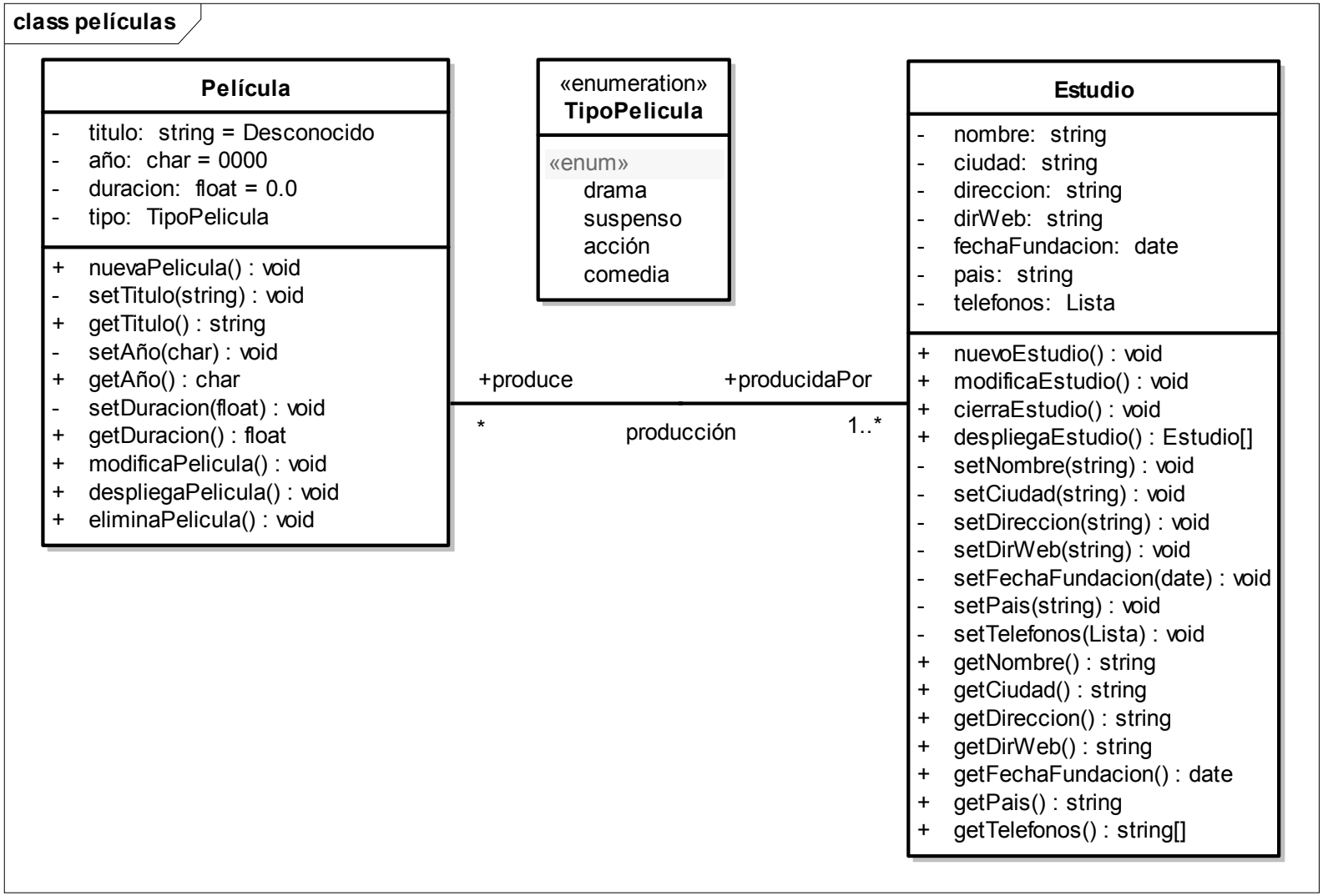
- **Composición:** relación ES-PARTE-DE, asimétrica, una de las extremidades juega un rol predominante en relación a la otra, por lo cual sólo lleva un rol. Indica clases de objetos compuestos
- **Mensajes:** especificación de un objeto junto con la invocación de uno de sus métodos **objetoDestino.nombreDelMétodo(listaDeArgumentos)**
- **Encapsulación:** propiedad que permite que los objetos sean definidos en su estructura y su comportamiento, obligando al uso del pase de mensajes o de la invocación de sus métodos, si se quiere acceder al objeto



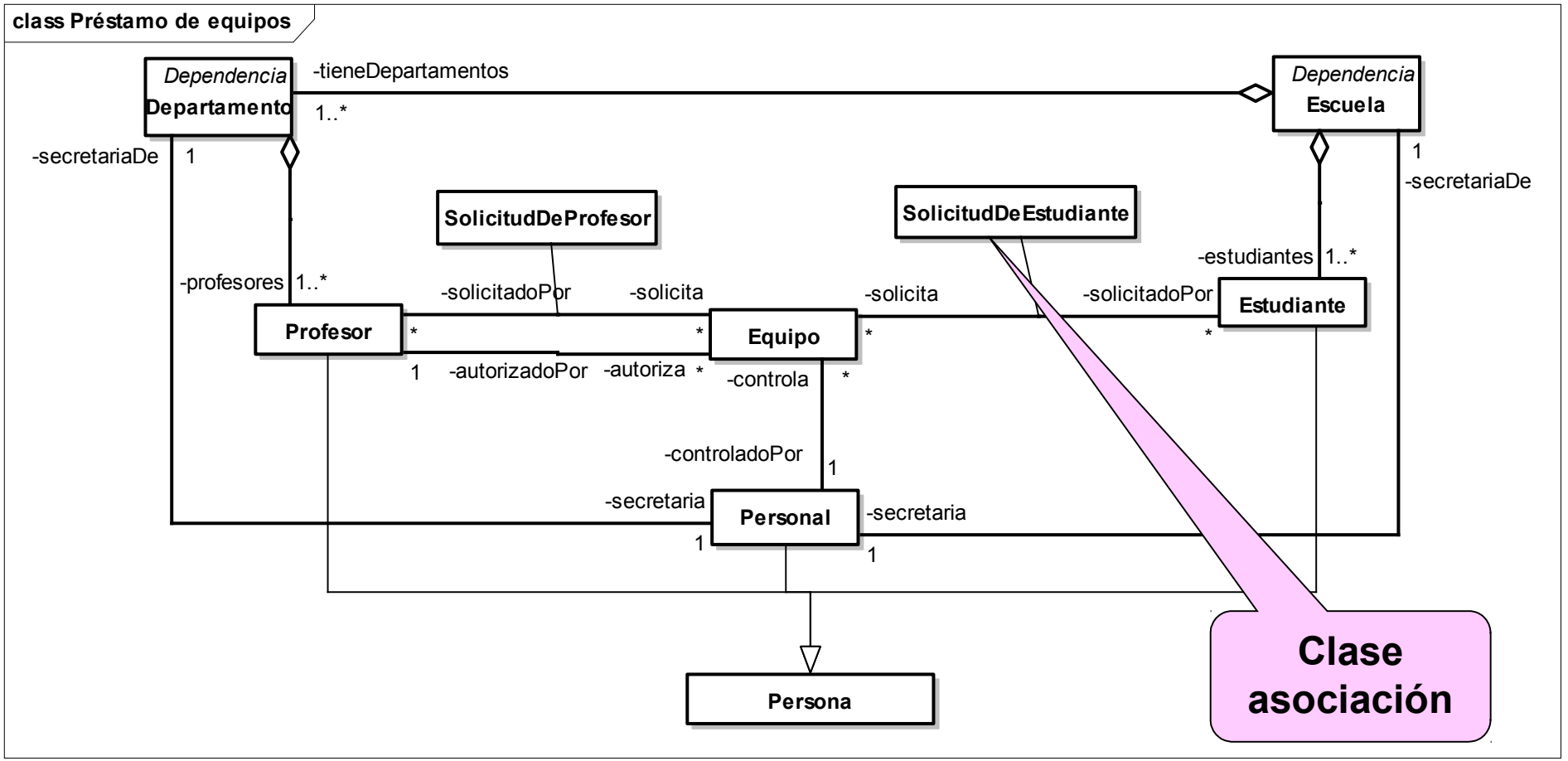
# Modelado estructural



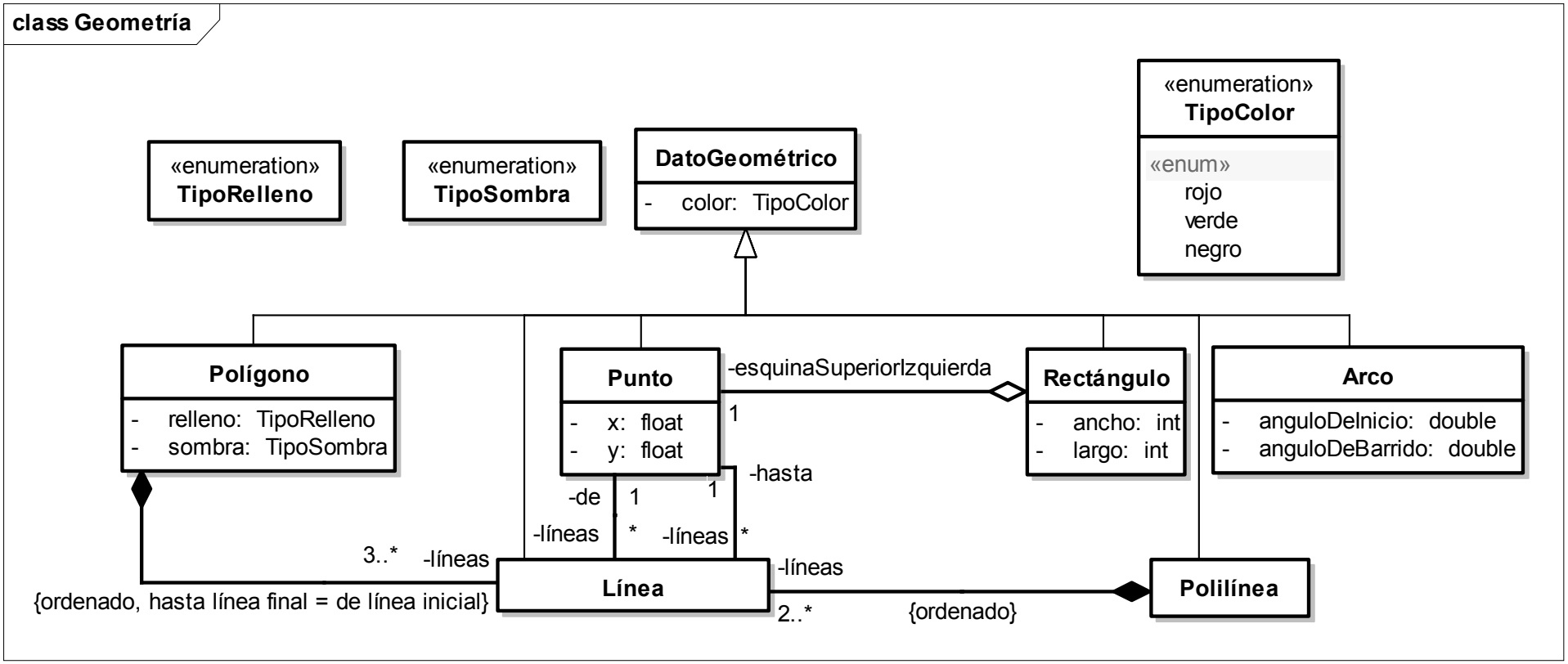
# Diagrama de clases



# Diagrama de clases

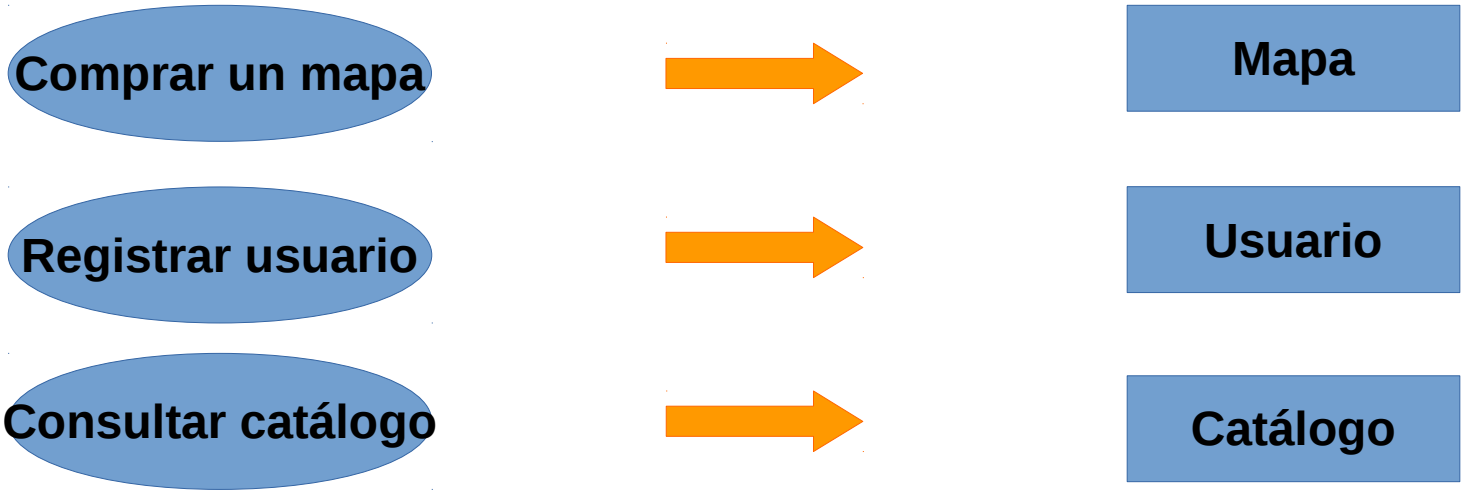


# Diagrama de clases



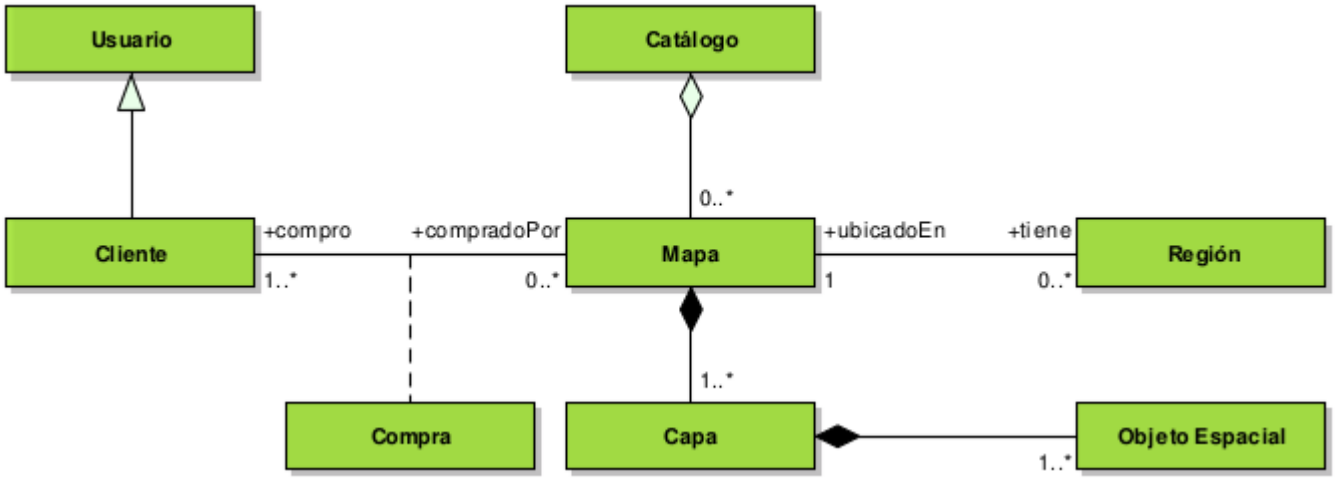
# Modelado estructural de requisitos

- Analizando los casos de uso se pueden identificar las clases del negocio
  - Cada sustantivo del nombre de un caso de uso representa una clase de negocio

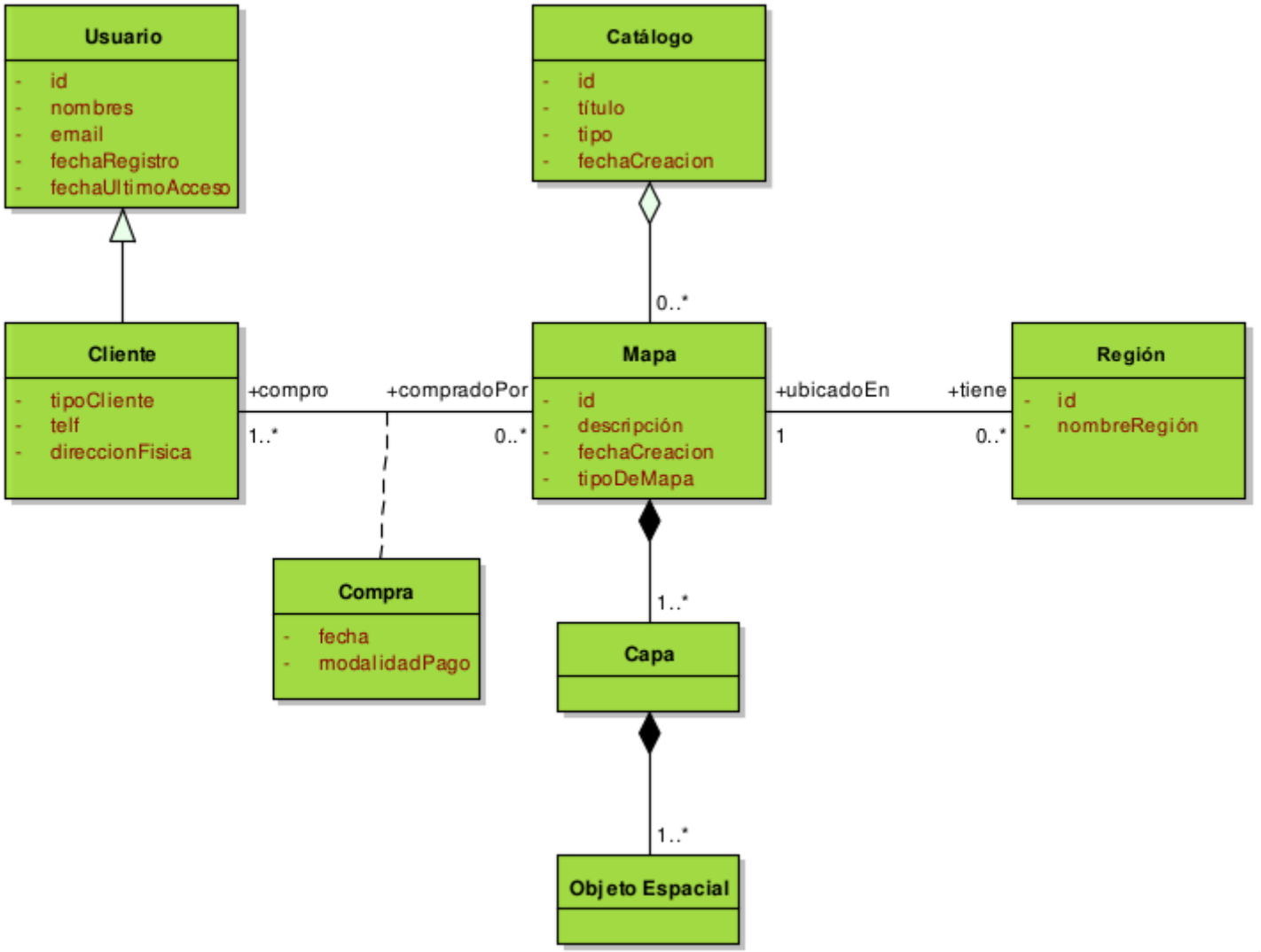


# Modelado estructural de requisitos

- En la especificación de requisitos los diagramas de clases se usan para representar
  - Los objetos de negocio del dominio de aplicación
  - Las relaciones entre estos objetos



# Modelado estructural de requisitos



# Modelado dinámico de requisitos

- Toda aplicación tiene una dinámica asociada
  - Esta dinámica está determinada por el comportamiento de la aplicación ante eventos
  - Los eventos hacen que la aplicación responda o reaccione. Estos eventos pueden ocurrir debido a:
    - La interacción del usuario con la aplicación
    - Una transición de estado en un objeto de la aplicación
    - Una falla de la aplicación

# Vista de interacción

- **Modelan la interacción entre los objetos y relaciones de una aplicación, sistema, subsistema, programa o clase**
  - Muestran el flujo de mensajes entre los objetos de una aplicación ocasionados por una interacción
  - Una interacción es una relación entre los objetos y su entorno (actores)
    - Ejemplo, la activación de una función descrita por un caso de uso
- **La interacción se puede representar con los diagramas**
  - Secuencia
  - Comunicación
  - Vistas de interacción

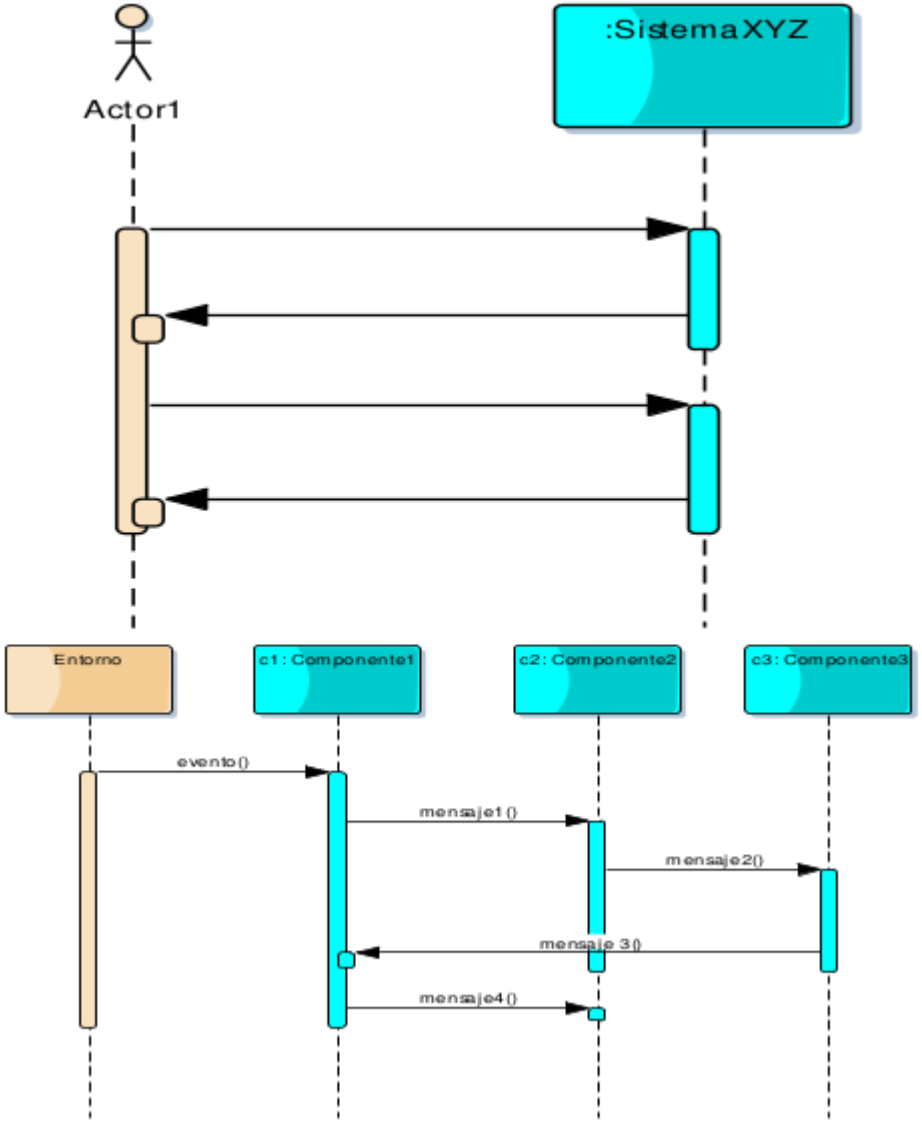
**Modelado del comportamiento**

# Diagramas de secuencia

- Es uno de los tipos de diagramas que proporciona UML para modelar la dinámica o el comportamiento de una aplicación
- Son útiles para:
  - Elaborar las Realizaciones de Casos de Uso (RCU)
    - Recordar que: Una RCU es una descripción de la implementación de un caso de uso y consta de:
      - Un diagrama de colaboración que identifica las clases participantes
      - Un diagrama de secuencia que describe el flujo de mensajes entre las clases participantes

# Diagramas de secuencia

- En el desarrollo de software se usan en:
  - La especificación de requisitos, para modelar la interacción entre el usuario y el sistema
  - En el diseño arquitectónico, para modelar la interacción (ocasionada por un evento) entre los componentes o subsistemas de la solución propuesta .



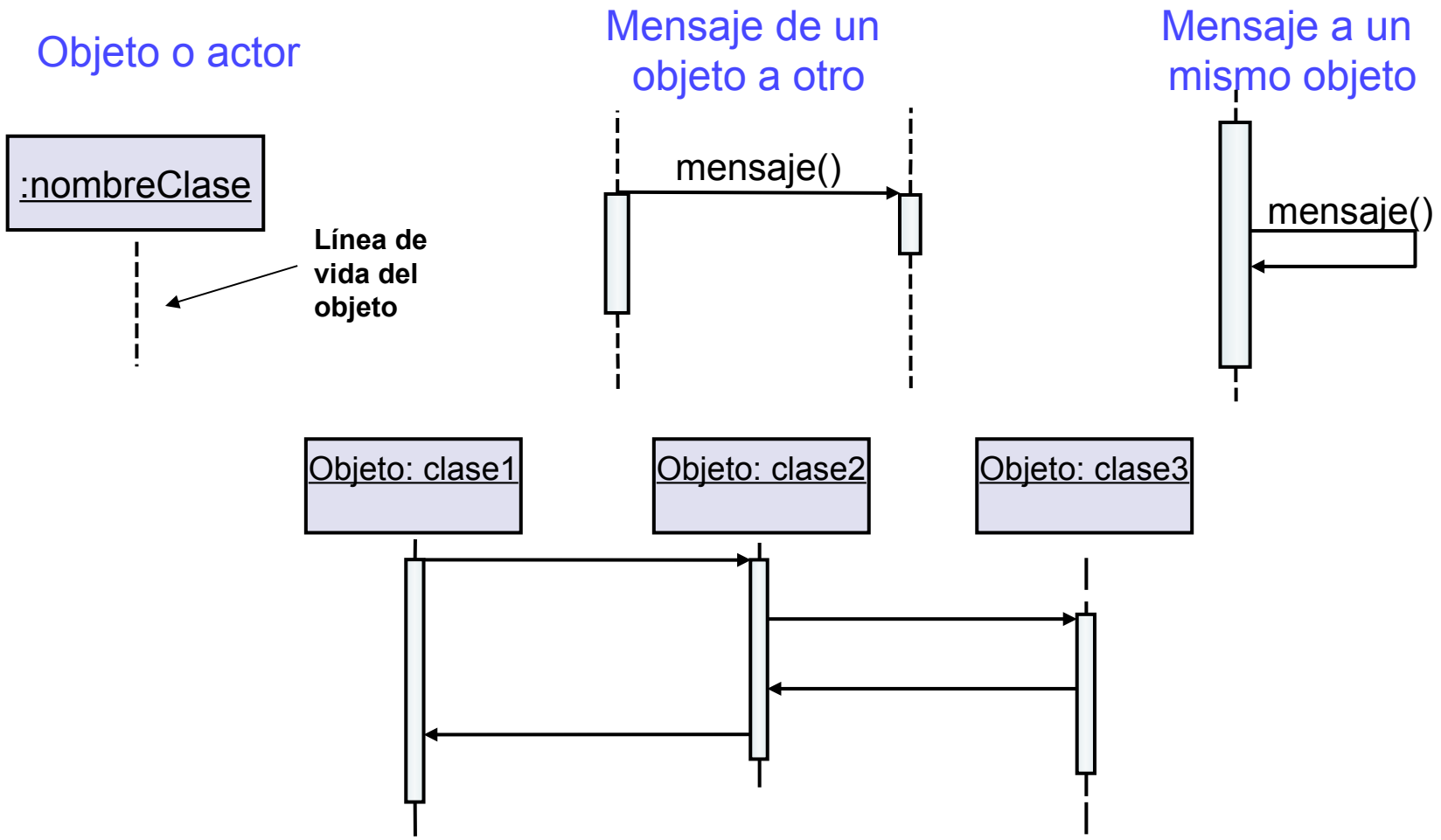
# Diagrama de secuencia

- **Muestran las interacciones entre los objetos desde el punto de vista temporal insistiendo en la cronología del envío de mensajes**
  - Permiten mostrar las relaciones entre los objetos de la aplicación o programa y sus usuarios (actores)
  - Los objetos y los enlaces creados o destruidos en el curso de una interacción pueden llevar las restricciones de: {nuevo}, {destruido}, {transitorio}, iteración sobre todos los objetos de un tipo: *{todos}

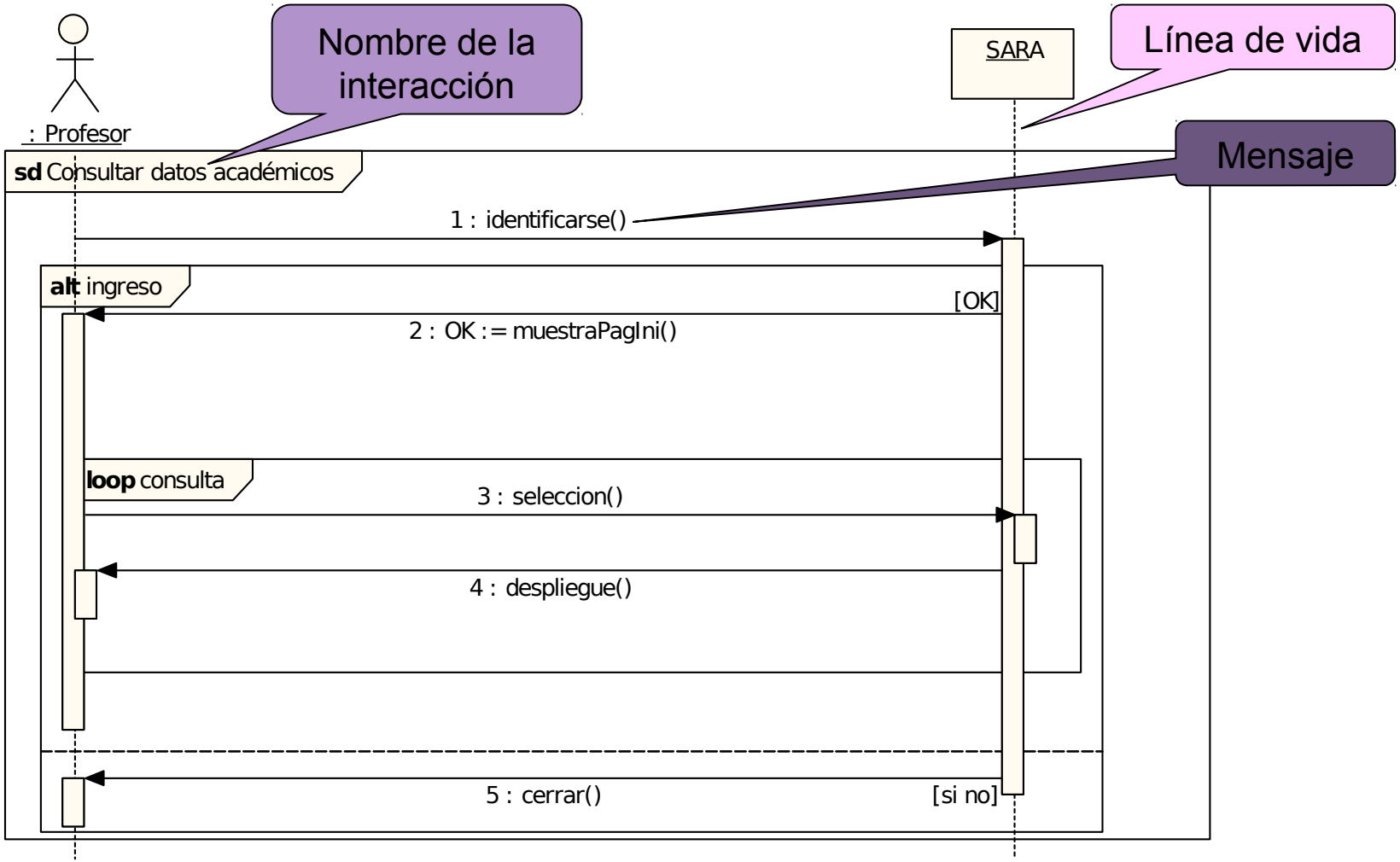
# Diagrama de secuencia

- **Describen la dinámica de una aplicación (o de una parte de ella)**
  - Muestran como los actores (humanos u otros sistemas) se relacionan con los objetos de la aplicación y cómo los objetos del programa se comunican entre sí mediante el pase de mensajes
  - La interacción entre objetos se muestra en forma secuencial (temporal) usando las líneas de vida (de arriba hacia abajo)
  - En sistemas interactivos, estos diagramas permiten mostrar el flujo de mensajes entre objetos que origina la activación de una función
- **Forma del mensaje:**
  ***secuencia : resultado := mensaje(argumentos)***

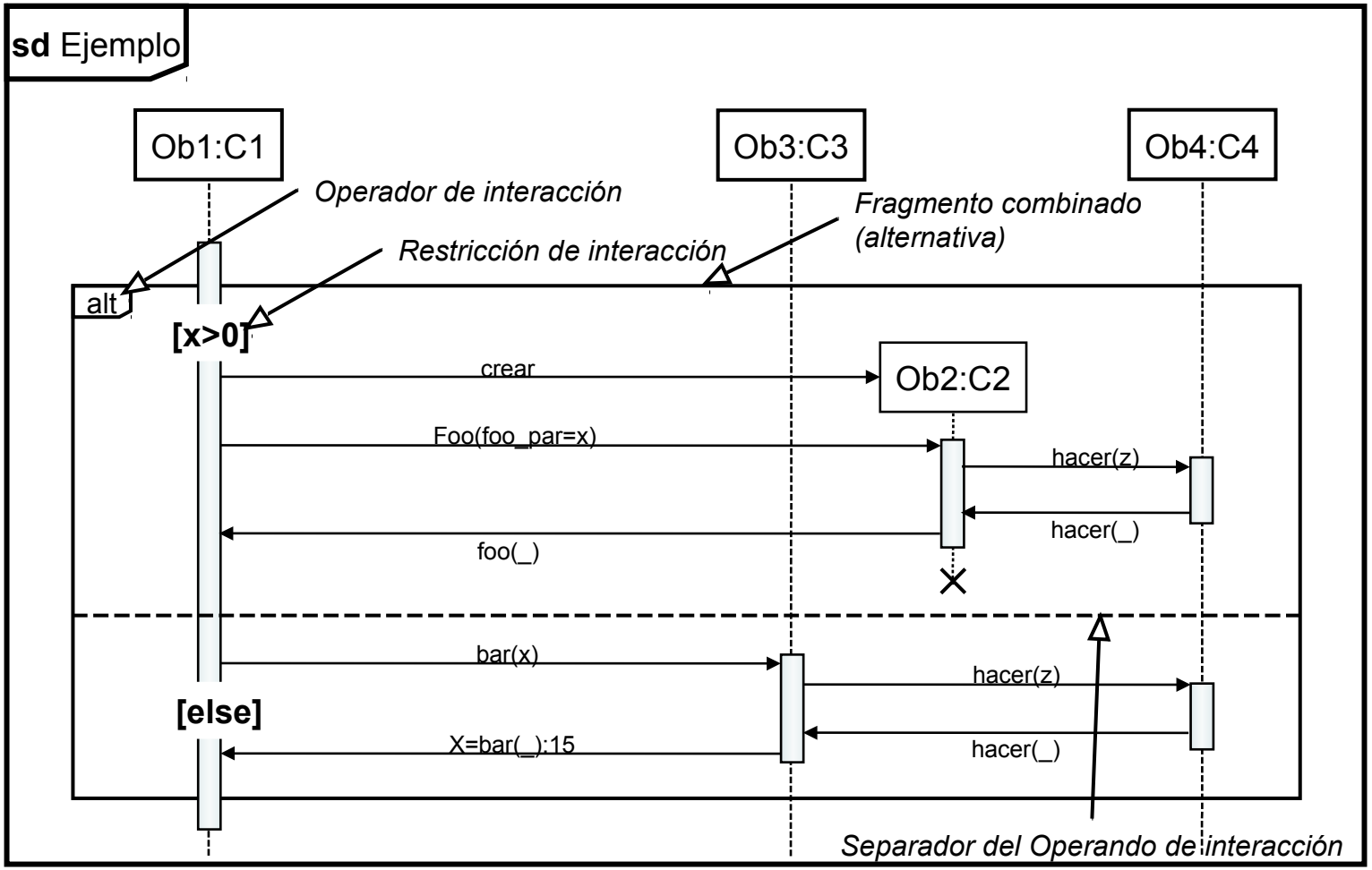
# Diagrama de secuencia



# Diagrama de secuencia



# Diagrama de secuencia




[OMG, 2003a]

**Fragmento combinado**