# 26 de Enero de 2021

- Calendario
- Repaso de UML
- Repaso de Diagrama de Clases
- Diagramas de Caso de Uso

## Planificación del SEGUNDO Trimestre.

| Fecha | Actividad |
|---|---|
| 12 Enero 2021 | **UT 3.** Programación trimestral<br>▪ Diseño de diagramas de clases.<br>▪ Herramientas para el diseño de diagramas de clases. |
| | |
| 26 Enero 2021 | **UT4**. Seguimiento<br>▪ Diseño de diagramas de casos de uso y de secuencia.<br>▪ Manejo de herramientas para su diseño. |
| | |
| 9 Febrero 2021 | **UT3 y UT4**. Preparación de examen<br>▪ Se resolverán ejercicios y dudas sobre los contenidos del trimestre.<br>▪ Manejo de herramientas.<br>▪ Se especificará el tipo de prueba.<br>▪ |
| ¿? | **EXAMEN TRIMESTRAL (Previsión para la semana del 2 de Marzo)** |

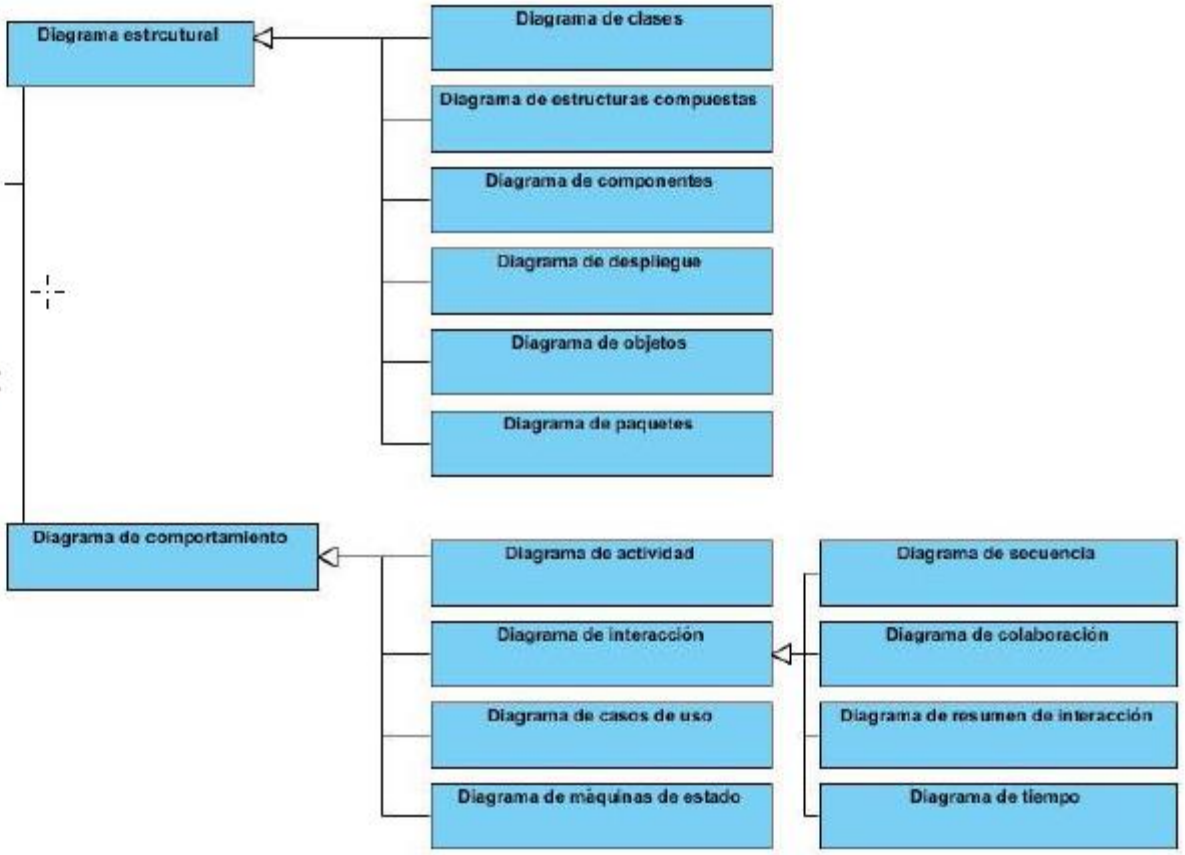
# UML: Lenguaje Unificado de Modelado

Funciones:

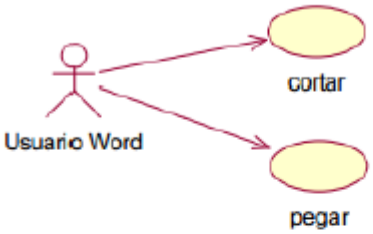
- **Visualizar**: expresa de forma gráfica un sistema.
- **Especificar**: cuáles son las características de un sistema antes de su implementación.
- **Construir**: a partir de los modelos diseñados.
- **Documentar**: los propios diagramas sirven como documentación.

# UML: Tipos de diagramas

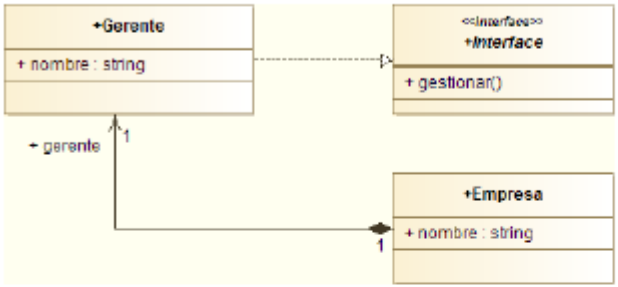
- **Estructurales**: representan la visión *estática* del sistema.
- **De comportamiento**: muestran la *conducta* del sistema *en tiempo de ejecución*.



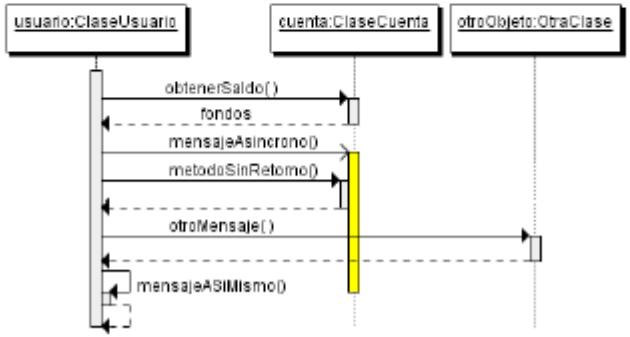
# UML: Tipos de diagramas



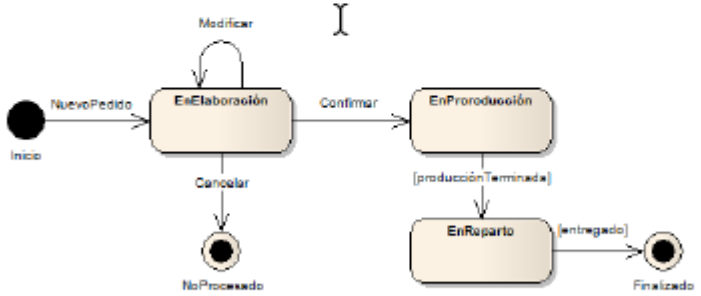

De Casos de uso




De clases




De secuencia



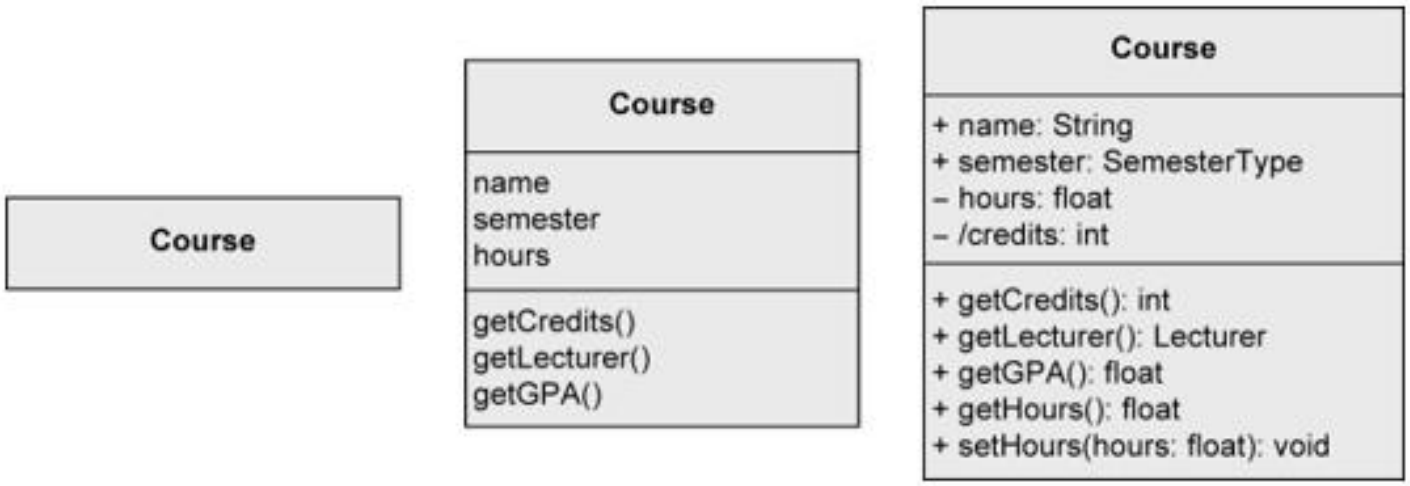

De estado

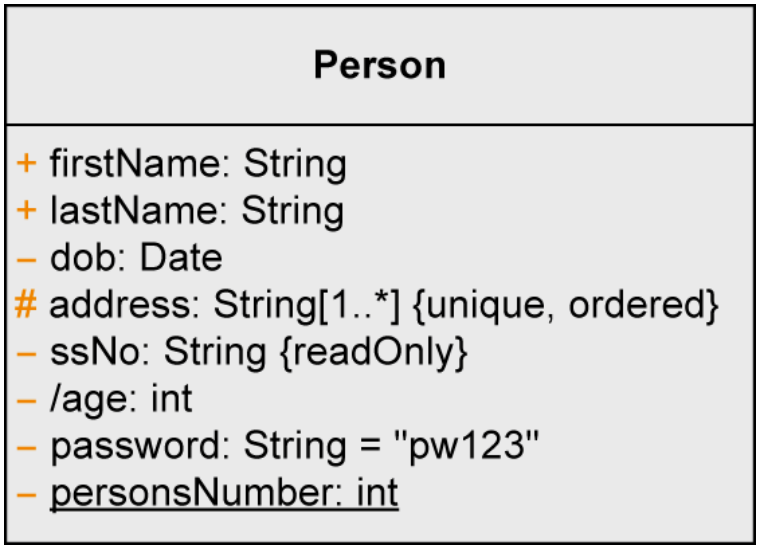
# DIAGRAMA DE CLASES: CONCEPTOS

- CLASES
- MULTIPLICIDAD
- RELACIONES
  - Asociación
    - Agregación
    - Composición
  - Generalización (Herencia)
  - Dependencia
  - Realización

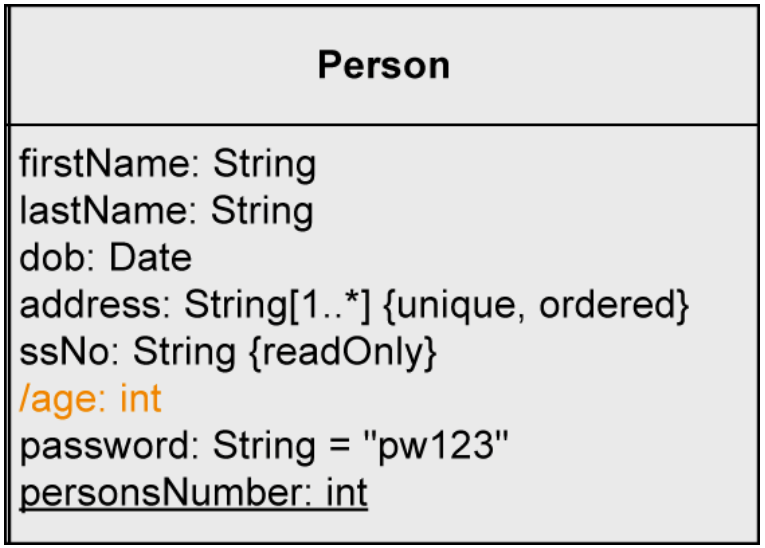
**Representa los elementos estáticos del sistema (clase, interfaz), sus atributos, su comportamiento, y cómo se relacionan entre ellos.**

# Especificación de Clases: Distintos niveles de detalle.

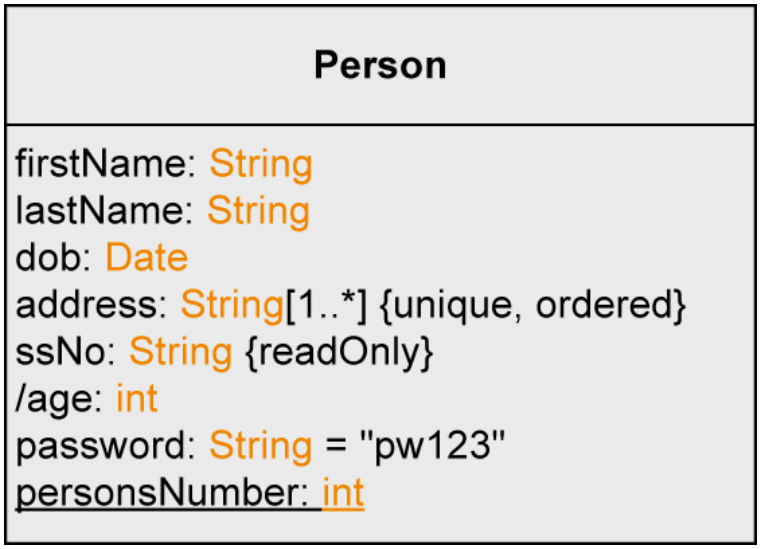



- Visibilidad
  - + public: Acceso de objetos a cualquier clase permitida
  - - private: permitido solo dentro del propio objeto
  - # protected: se permite el acceso de objetos de la misma clase y sus subclases.
  - ~ package: acceso por objetos cuyas clases están en el mismo paquete.

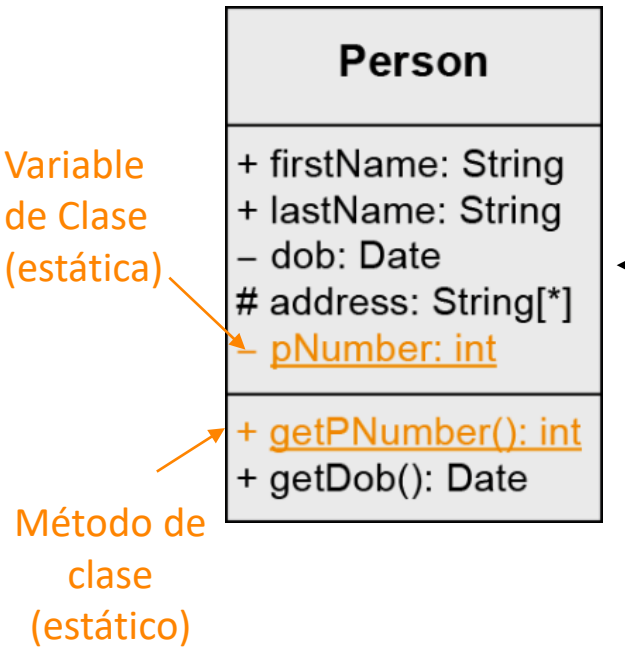



- Atributos calculados

  age: Atributo calculado mediante la fecha de nacimiento.




- Tipos de datos
  - Clases definidas
  - Tipos primitivos.
  - Tipos compuestos ( Date..)
  - Wrappers: Boolean, Integer….
  - Enumerados: «**enumeration**»




```
class Person {

   public String firstName;
   public String lastName;
   private Date dob;
   protected String[] address;
   private static int pNumber;

    public static int getPNumber() {…}
   public Date getDob() {…}
}
```

# RELACIONES

**MULTIPLICIDAD o CARDINALIDAD** de una relación: indica el número de instancias de las clases que participan en ella. Puede ser:

| | |
|---|---|
| **1** | **exactamente 1** |
| **0..1** | **cero o una** |
| **1..*** | **uno o más** |
| **0..*** | **cero o más** |
| * | **varios** (equivalente a "0..*") |
| **n** | **exactamente "n"** (n, entero natural > 0) |
| **n..m** | **de "n" a "m"** (m ≥ n) Ejemplos: 3..n, 1..31 |
| **n..*** | **"n" o más** Ejemplo : 5..* |

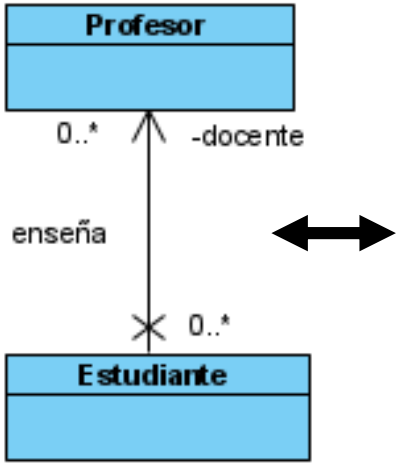
**Si no se especifica la multiplicidad se asume que es 1.**

# Asociación.

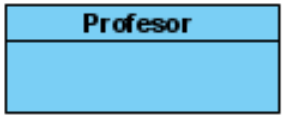

- En el papel de docente, el profesor enseña a 0 o más estudiantes y el estudiante es enseñado por 0 o más profesores como docente. Si fuera tutor podría cambiar.
- **Dirección de lectura**: Los profesores enseñan a los estudiantes y no al revés
- **Dirección de navegabilidad**: Los estudiantes conocen y pueden accede a las caracteristicas visibles de los profesores que les enseñan. (Horario)
- *La navegabilidad desde un objeto a otro asociado indica que ese objeto conoce al asociado por lo que puede accede a sus atributos y operaciones visibles.*

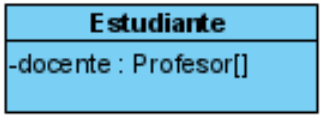
*Ampliación: pdf Navegabilidad en foro*

# Asociación.

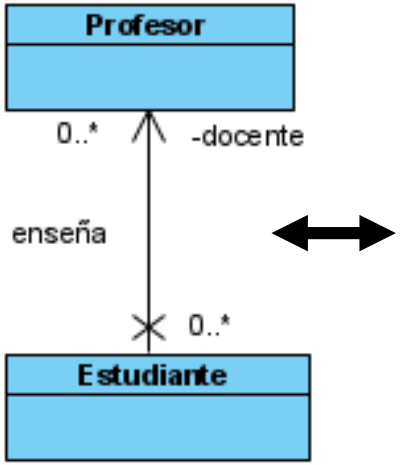



```
class Profesor {..}

class Estudiante{
   private Profesor[] docente;
   …
}
```



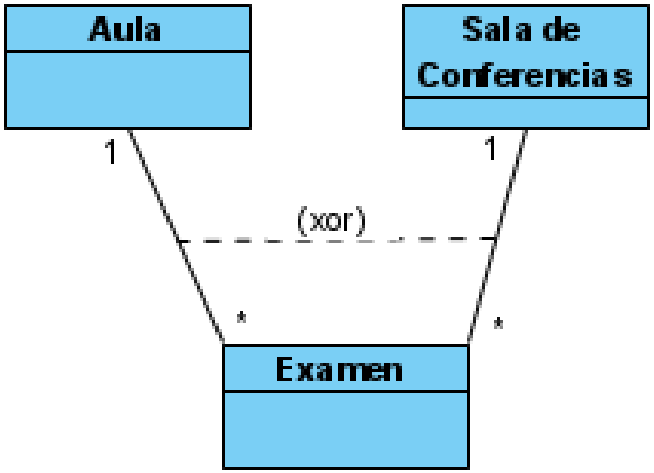



# Asociación.



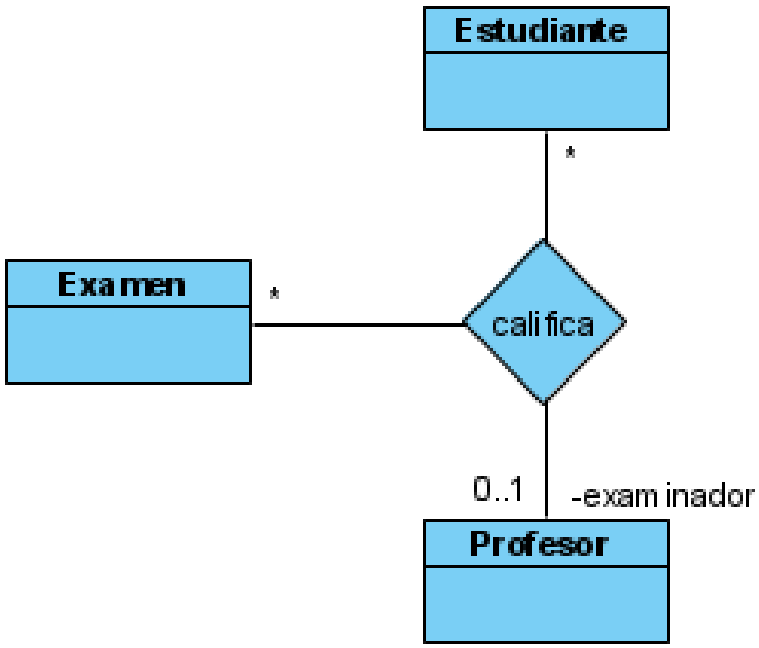

```
class Profesor {..}

class Estudiante{
    private Profesor[] docente;
    …
}
```

# Asociación: restricciones.




- Un examen puede tener lugar en un aula o en una sala de conferencias pero no en los dos.

# Asociación n-ary




***( Estudiante, Examen) -> Profesor***

*Un estudiante realiza un examen y será calificado por uno o ningún profesor.*

***(Examen, Profesor) -> Estudiante***

*Un examen de un profesor puede ser realizado por multiples estudiante. ~ Varios estudiantes serán calificados para un examen por un profesor.*

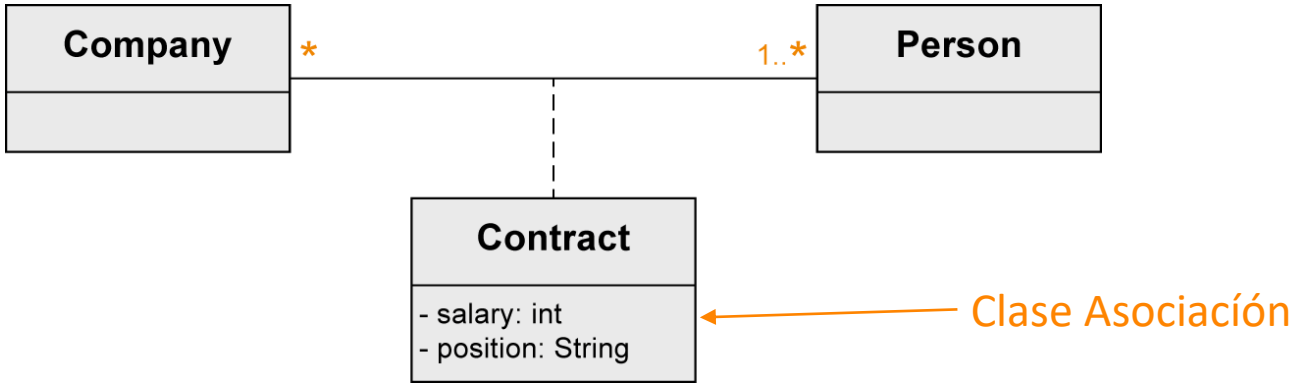
***(Estudiante, Profesor) -> Examen***

*Un estudiante será calificado por un profesor para varios exámenes*

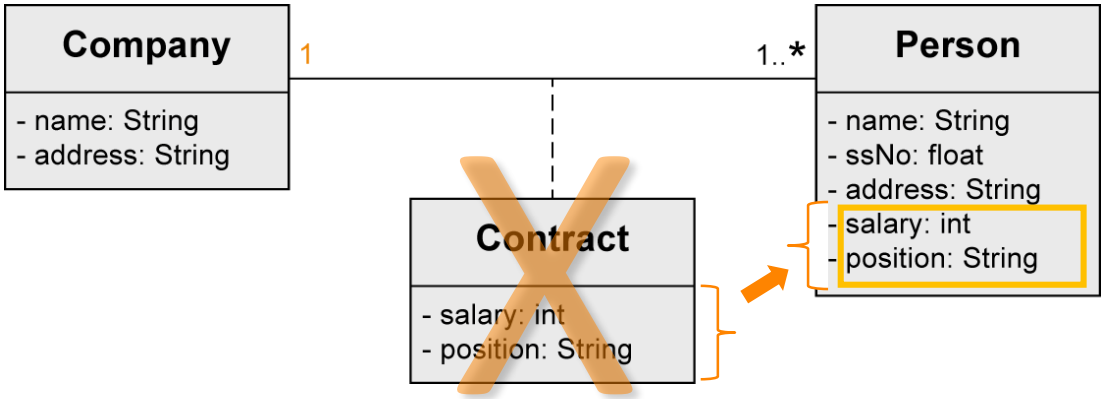
*No es posible que dos o mas profesores califiquen a un estudiante para el mismo examen.*

# Clase Asociación

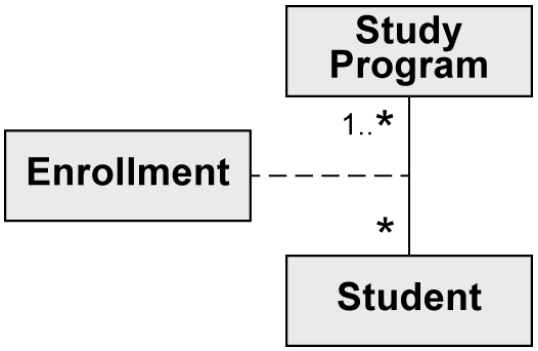
- Suele ser necesaria para modelar asociaciones de muchos a muchos *(n:m)*



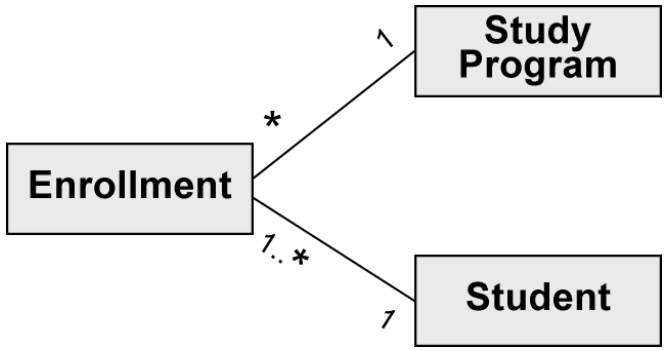

- Con 1:1 o 1:n se puede usar pero no es necesaria



# Clase Asociación vs Clases



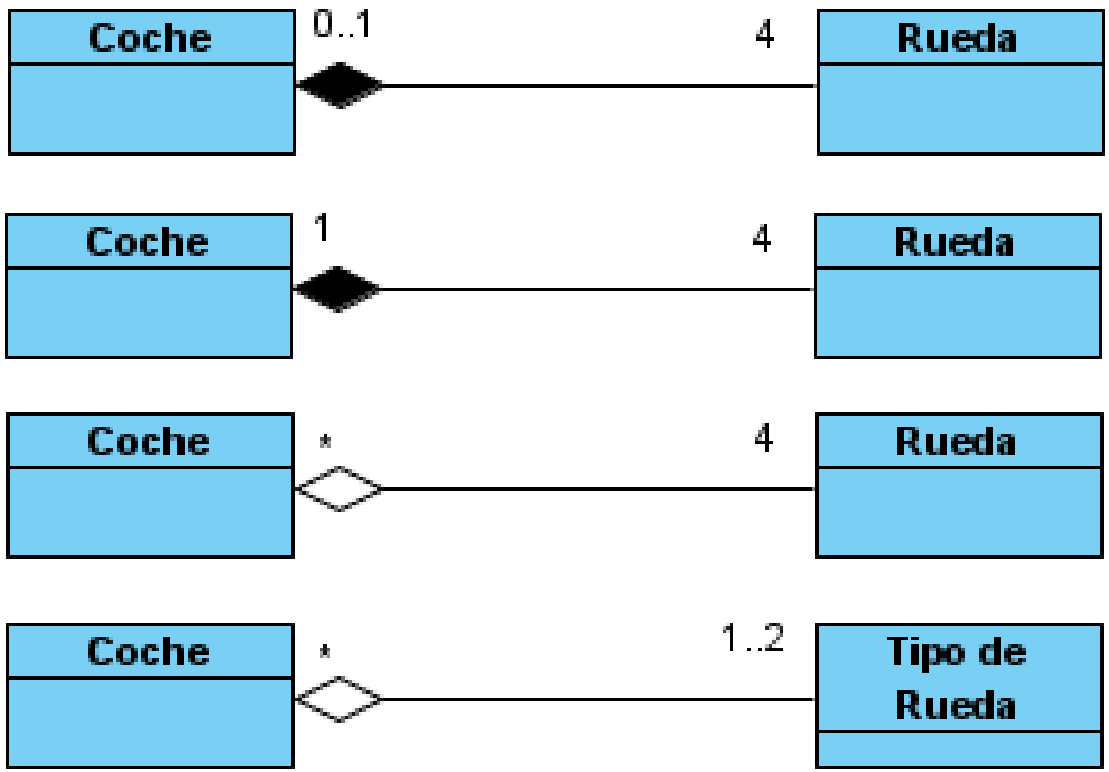

≠



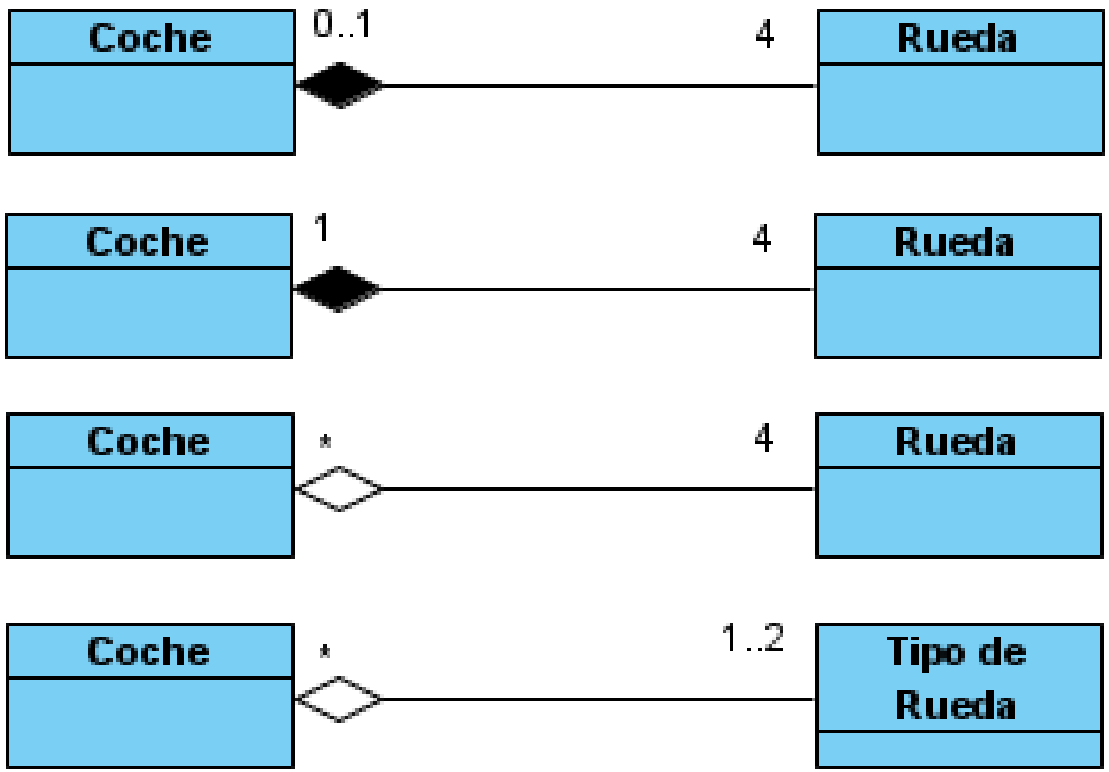

A `Student` can enroll for one particular `StudyProgram` only once

A `Student` can have mutiple `Enrollments` for one and the same `StudyProgram`

# Agregacion

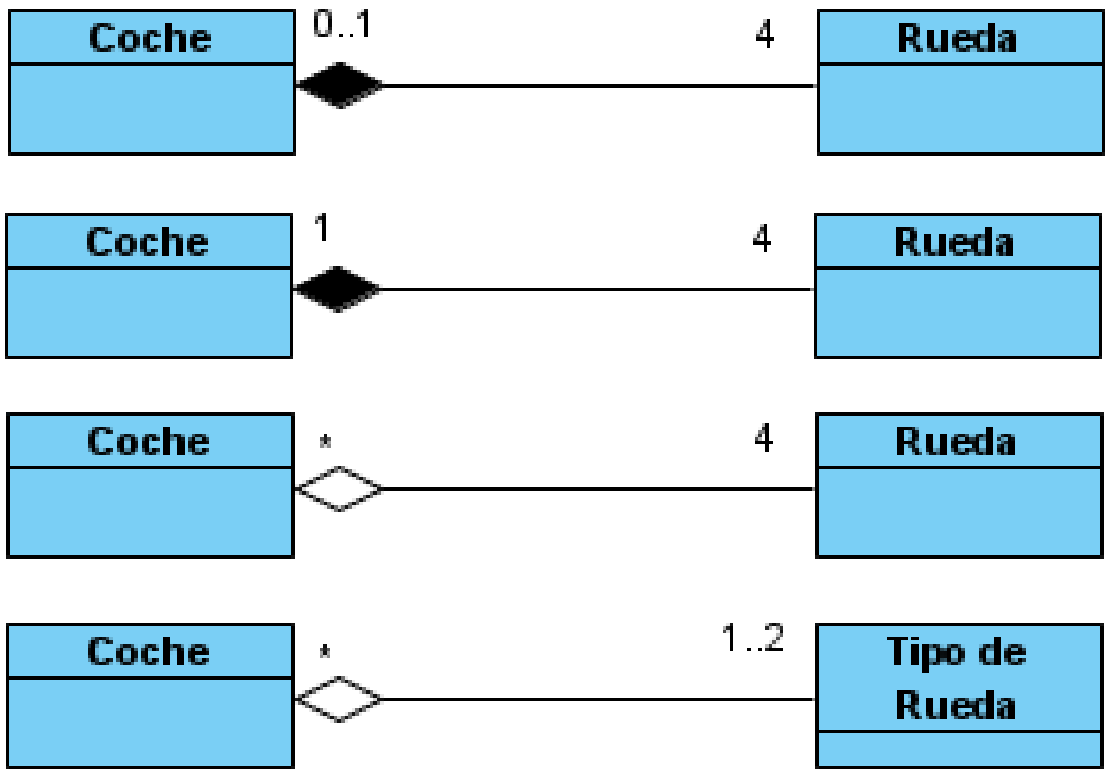

Coche
0..1
4
Rueda
Coche
1
4
Rueda
Coche
*
4
Rueda
Coche
*
1..2
Tipo de Rueda

# Agregacion




*una rueda puede existir sin un coche. Una rueda pertenece a un solo coche. Verdadero*

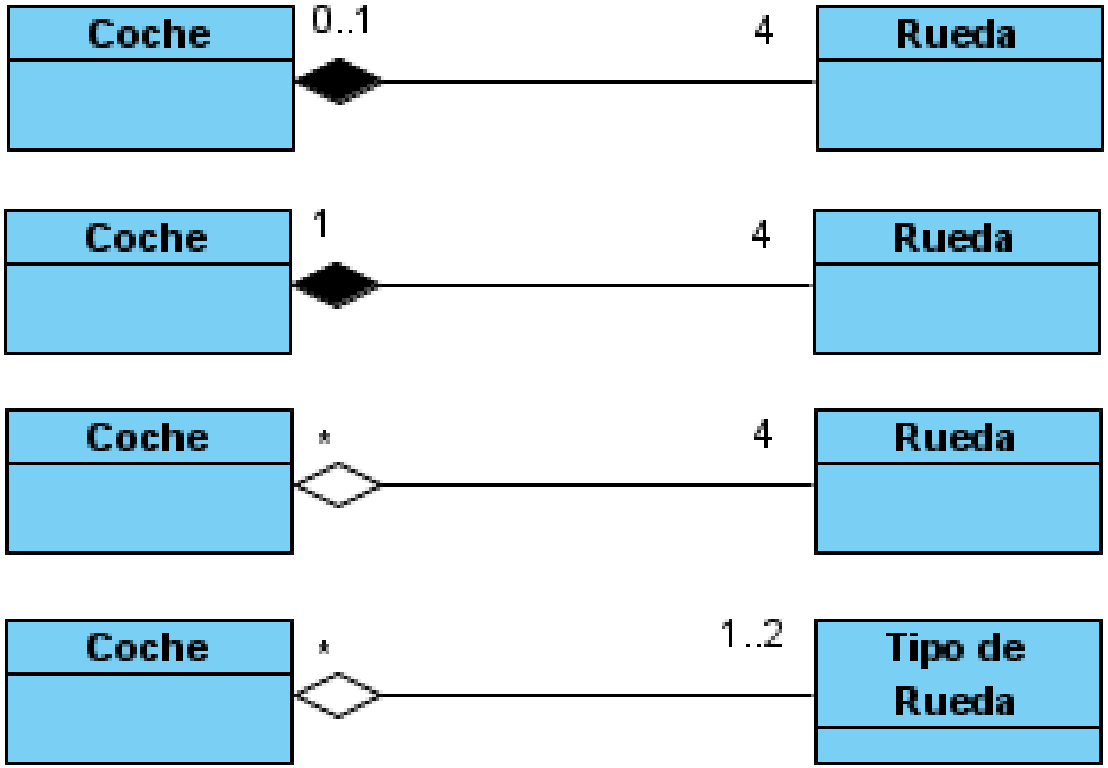
# Agregacion




*una rueda puede existir sin un coche. Una rueda pertenece a un solo coche. Verdadero*

*una rueda no puede existir sin un coche. Falso*

# Agregacion

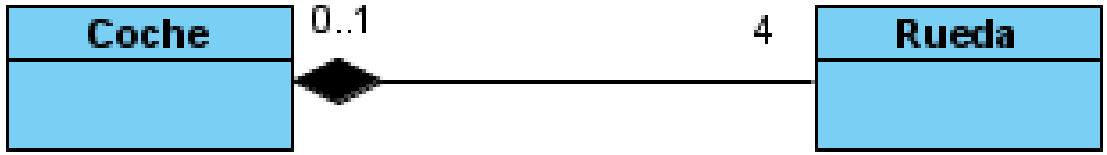



*una rueda puede existir sin un coche. Una rueda pertenece a un solo coche. Verdadero*

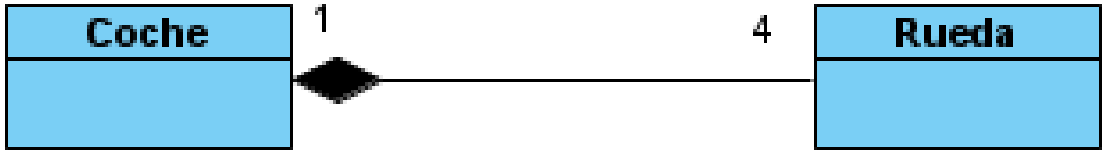
*una rueda no puede existir sin un coche. Falso*

*una rueda puede pertenecer a varios coches. Falso*

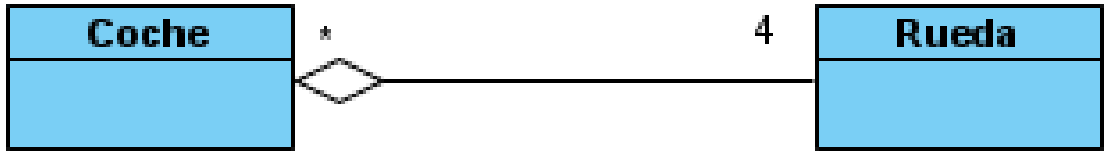
# Agregacion



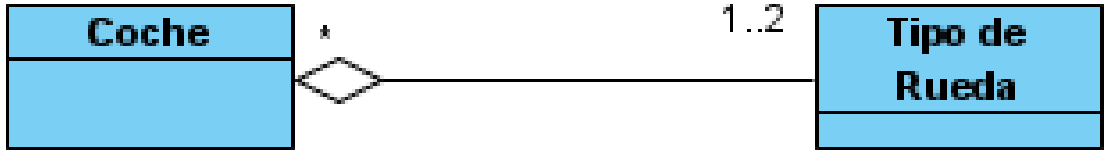

*una rueda puede existir sin un coche. Una rueda pertenece a un solo coche. Verdadero*




*una rueda no puede existir sin un coche. Falso*

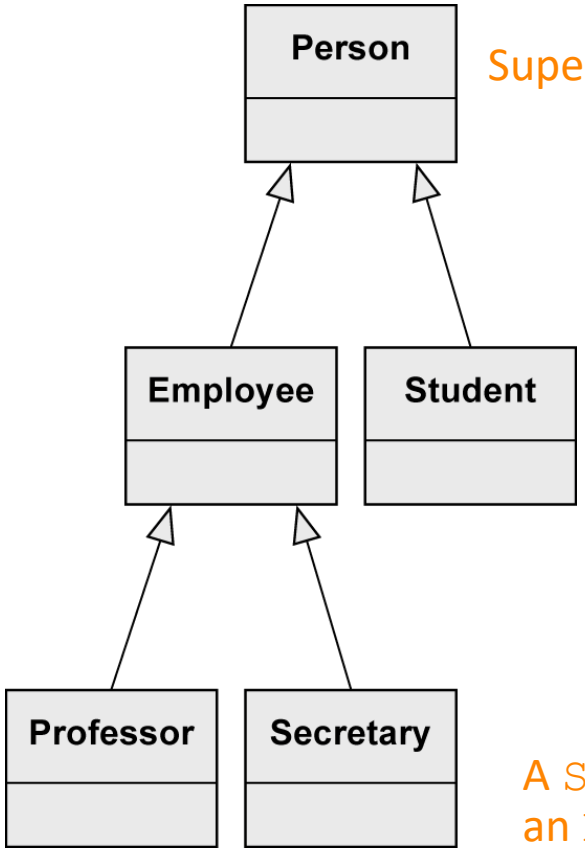



*una rueda puede pertenecer a varios coches. Falso*




*Un coche tiene uno o dos tipos de ruedas. Varios coches pueden tener el mismo tipo de rueda. Verdadero.*

# Generalización

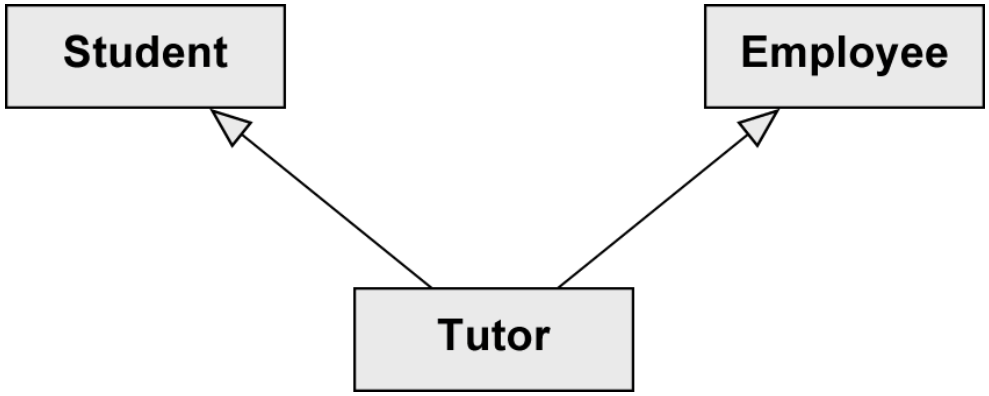



Superclass

Subclasses
…inherit characteristics, associations, and aggregations

A `Secretary` is an `Employee` and a `Person`

# Generalización




A `Tutor` is both an `Employee` and a `Student`

# DIAGRAMAS DE COMPORTAMIENTO

- Un diagrama de clases nos da información estática pero no dice nada acerca del comportamiento **dinámico** de los objetos que lo forman.

- Estudiaremos dos tipos:
  - De Casos de Uso
  - De Secuencia

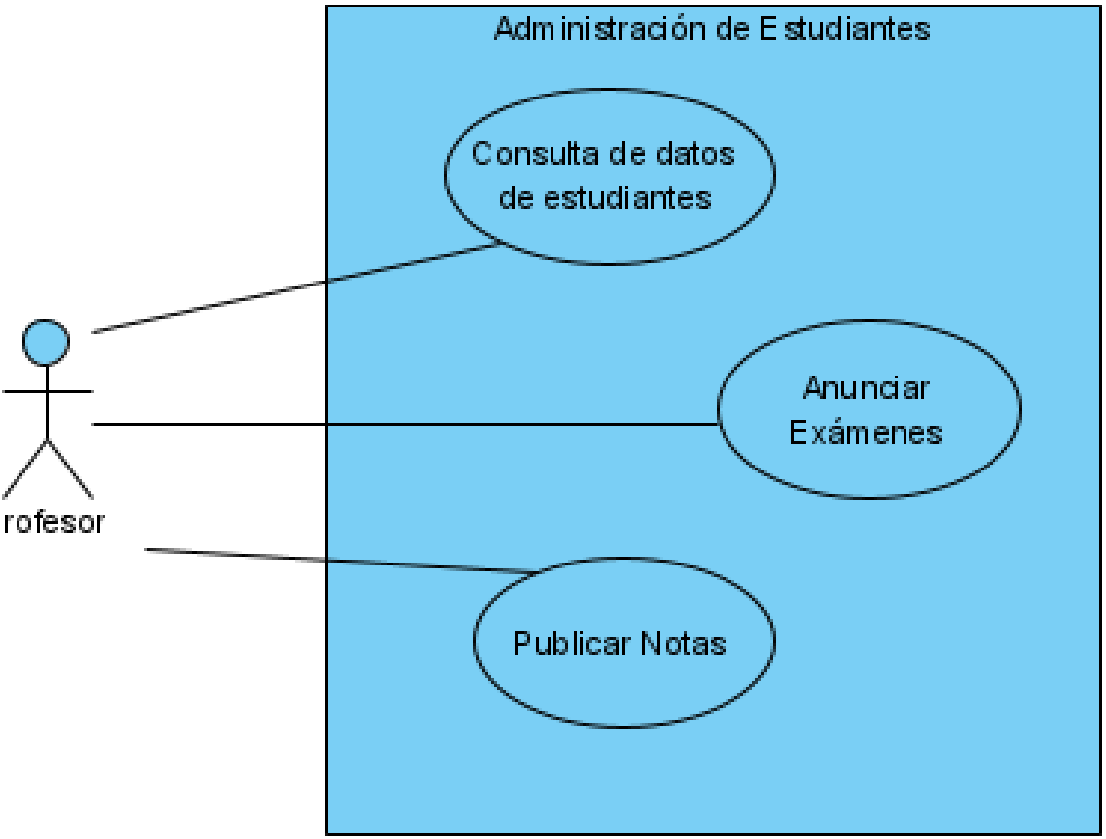
# Diagramas de CASOS DE USO .- ¿Para qué se utilizan?

- Los casos de uso son una técnica para especificar el comportamiento de un sistema,
- Permiten determinar el alcance del sistema,
- Modelan las funcionalidades de un sistema como interacciones entre los usuarios y el sistema,
- Los componentes principales de un diagrama de *Casos de Uso son*:
  1. Actores,
  2. Casos de Uso,
  3. Relaciones entre ellos.

***¿Qué se está describiendo? ( El Sistema)***

***¿Quién interactúa con el Sistema? ( Actores)***

***¿Qué pueden hacer los actors? (Los caso de uso)***

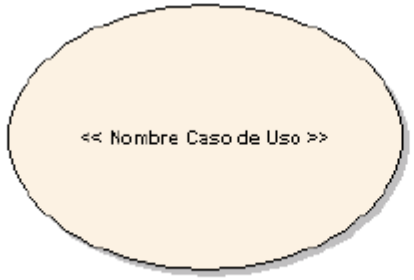



- **Sistema**

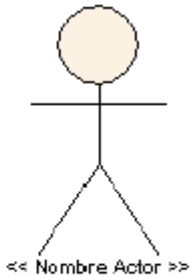
  *(¿Qué se está describiendo?)*
  - Sistema de Administración de Estudiantes.
- **Actores**

  *(¿Quién interactúa con el Sistema?)*
  - Profesor
- **Casos de Uso**
  (¿Qué pueden hacer los actores?)
  - Consultar los datos de estudiantes.
  - Anunciar Exámenes
  - Publicar Notas.

Un **caso de uso** es una unidad de funcionalidad, proporcionada por el sistema.

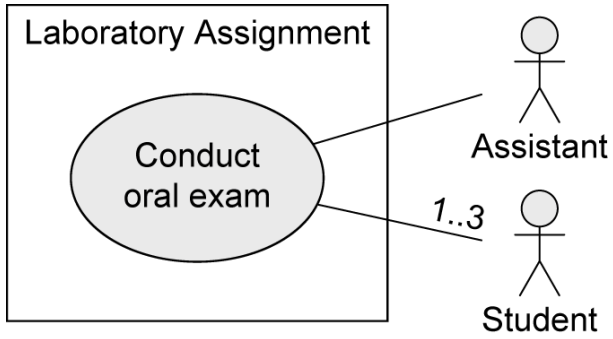


Un ***Actor*** es una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema,

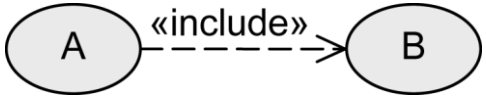


## Relación entre Casos de Uso y Actores

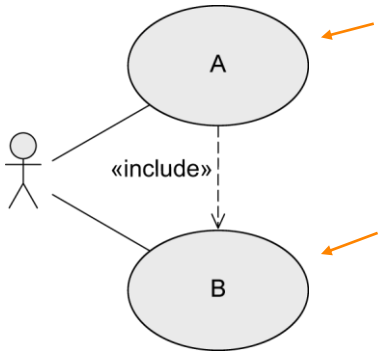
- ✓ Los actores se conectan con los casos de usos mediante lineas continuas. Asociación.
- ✓ Cada actor debe comunicarse con al menos un caso de uso.
- ✓ Siempre es binaria
- ✓ Se pueden especificar multiplicidades.



# Relación entre Casos de Uso: «include»




- ✓ El comportamiento de un caso de uso *(caso de uso incluido)* está integrado en el comportamiento de otro caso de uso *(caso de uso base).*

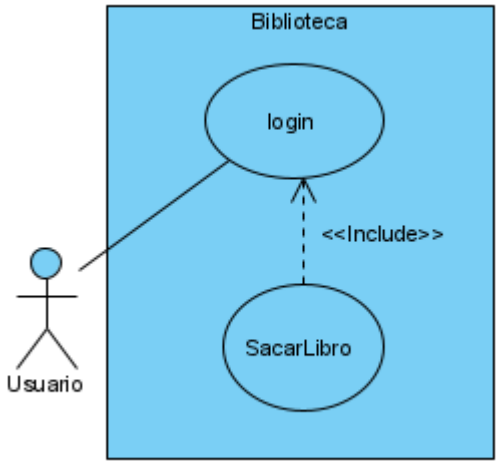



*Caso de uso BASE*

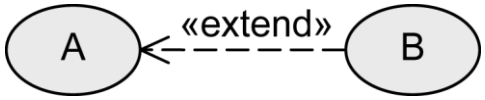
Necesita del comportamiento del caso de uso incluido para poder realizar su función.

*Caso de uso Incluido*

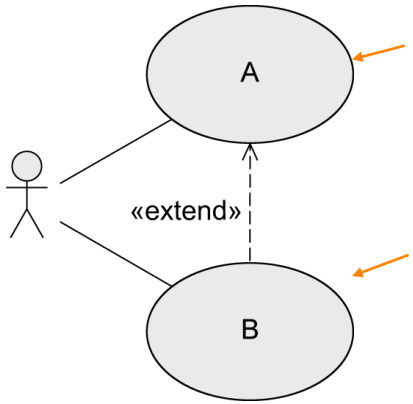
Puede ser ejecutado por si mismo



# Relación entre Casos de Uso: «extends»




- ✓ El comportamiento de un caso de uso *(caso de uso extendido)* puede estar integrado en el comportamiento de otro caso de uso *(caso de uso base).*

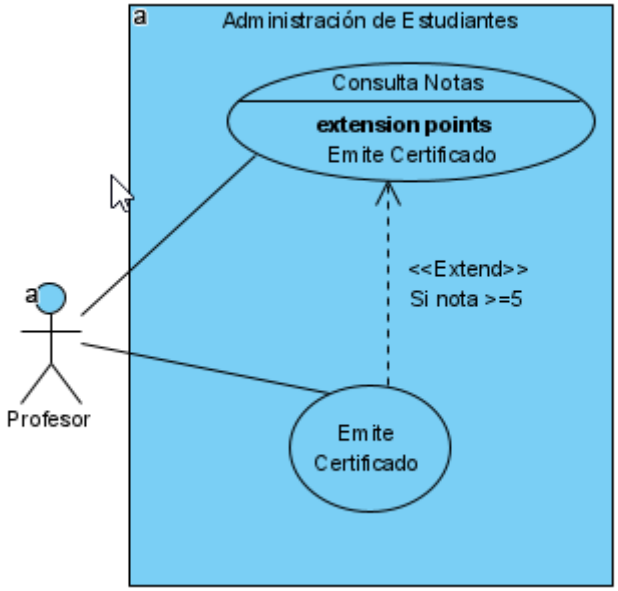



*Caso de uso BASE*

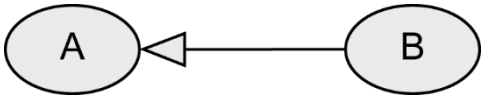
A decide si se ejecuta B evaluando una condición.

*Caso de uso extendido*

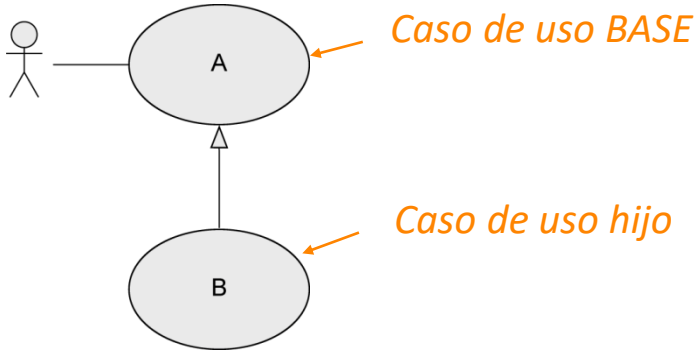
Puede ser ejecutado por si mismo

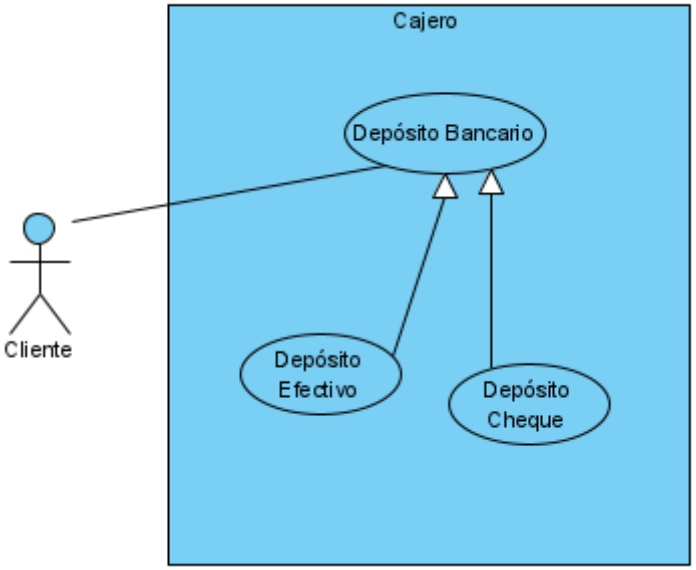


## Relación entre Casos de Uso: Generalización

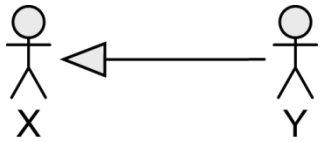


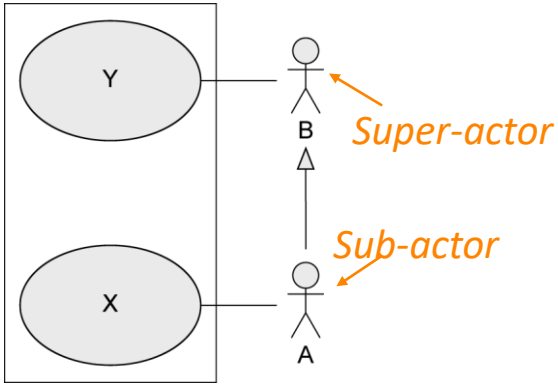

- ✓ El caso de uso A generaliza el caso de uso B, es decir, B hereda el comportamiento de A ademas de extender o sobreescribir su comportamiento.



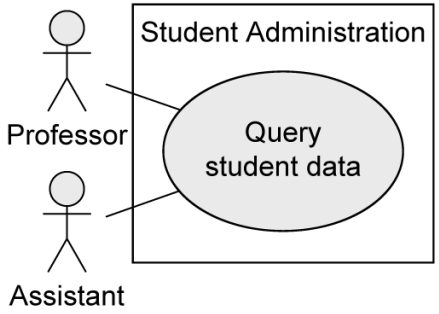



# Relación entre Actores: Generalización



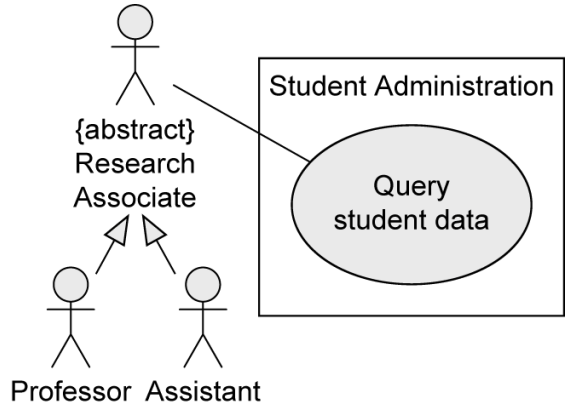




✓ El actor A hereda del actor B, A se puede comunicar con X e Y pero B solo se comunica con Y.




**Professor** AND **Assistant** needed for executing **Query student data**

≠




**Professor** OR **Assistant** needed for executing **Query student data**

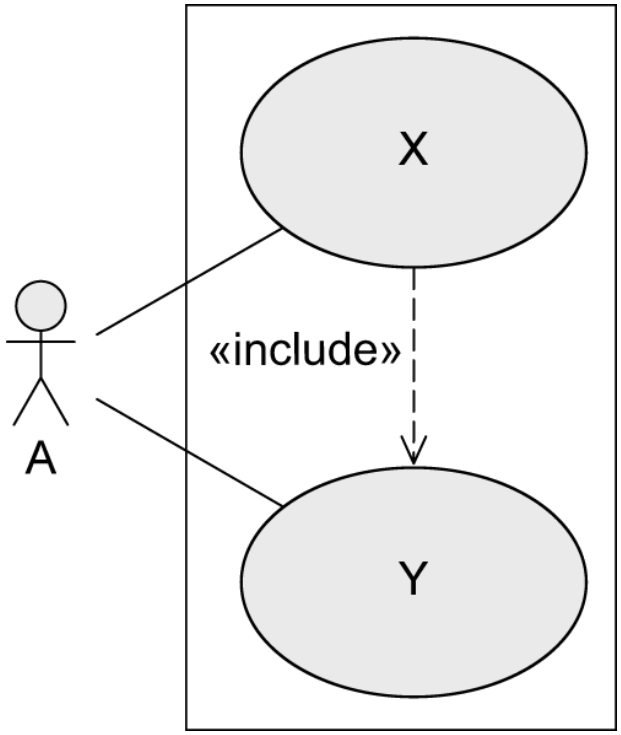
# Descripción de los casos de uso: Documentación

| | |
|---|---|
| **Nombre Caso de Uso:** <Nombre del Caso de Uso> | |
| **Creado por:** <Responsable> | |
| **Fecha:** <Fecha de creación> | |
| **Última modificación:** <Fecha de última modificación> | |
| **Descripción:** <Breve descripción del caso de uso> | |
| **Actor primario:** <Actor que dispara el caso de uso> | |
| **Actores secundarios:** <Actores que intervienen en el caso de uso> | |
| **Precondiciones:** <Precondiciones del caso de uso> | |
| **Poscondiciones:** <Poscondiciones del caso de uso> | |
| **Curso Normal** | **Curso Alternativo** |
| | |

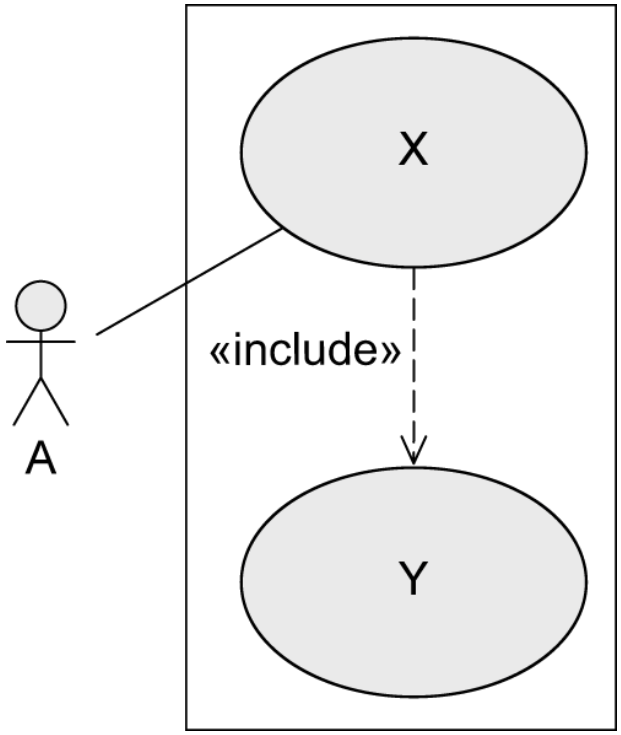
## Esquema

# Buenas Prácticas «include»

*UML standard*

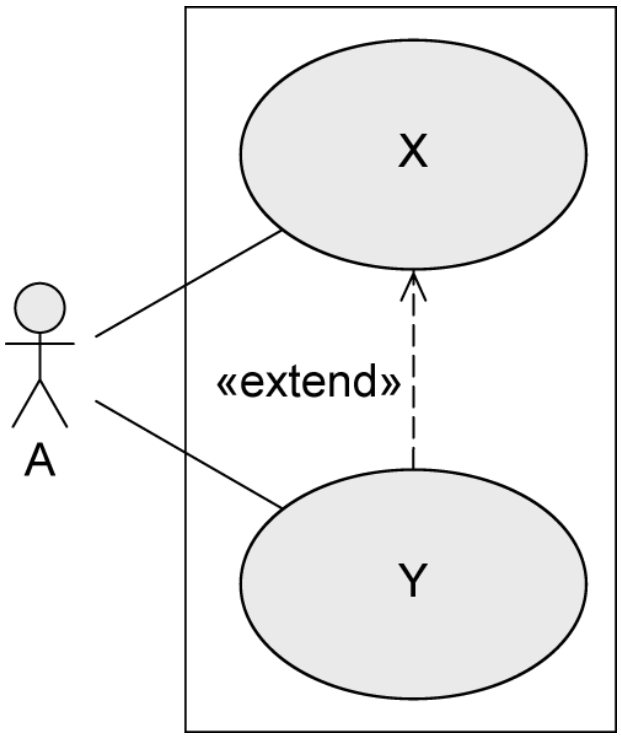



*Best practice*

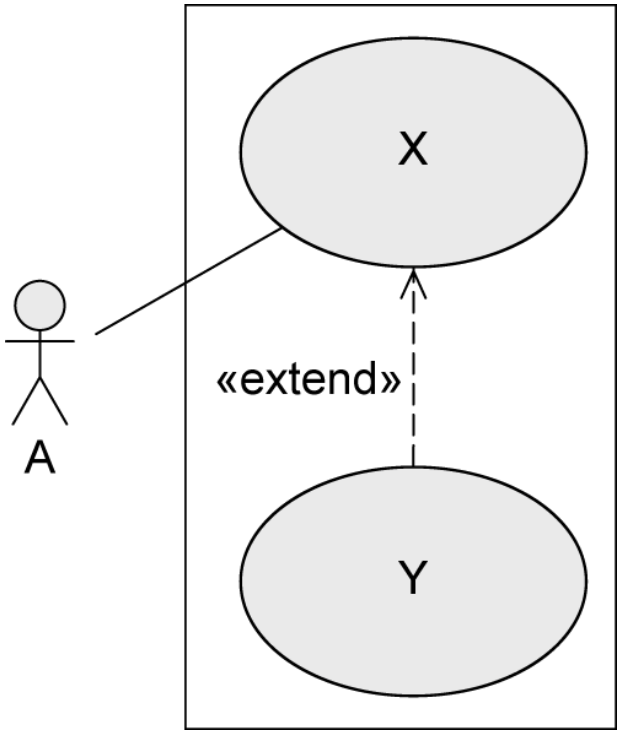


# Buenas Prácticas «extend»

*UML standard*




*Best practice*



# Buenas Prácticas

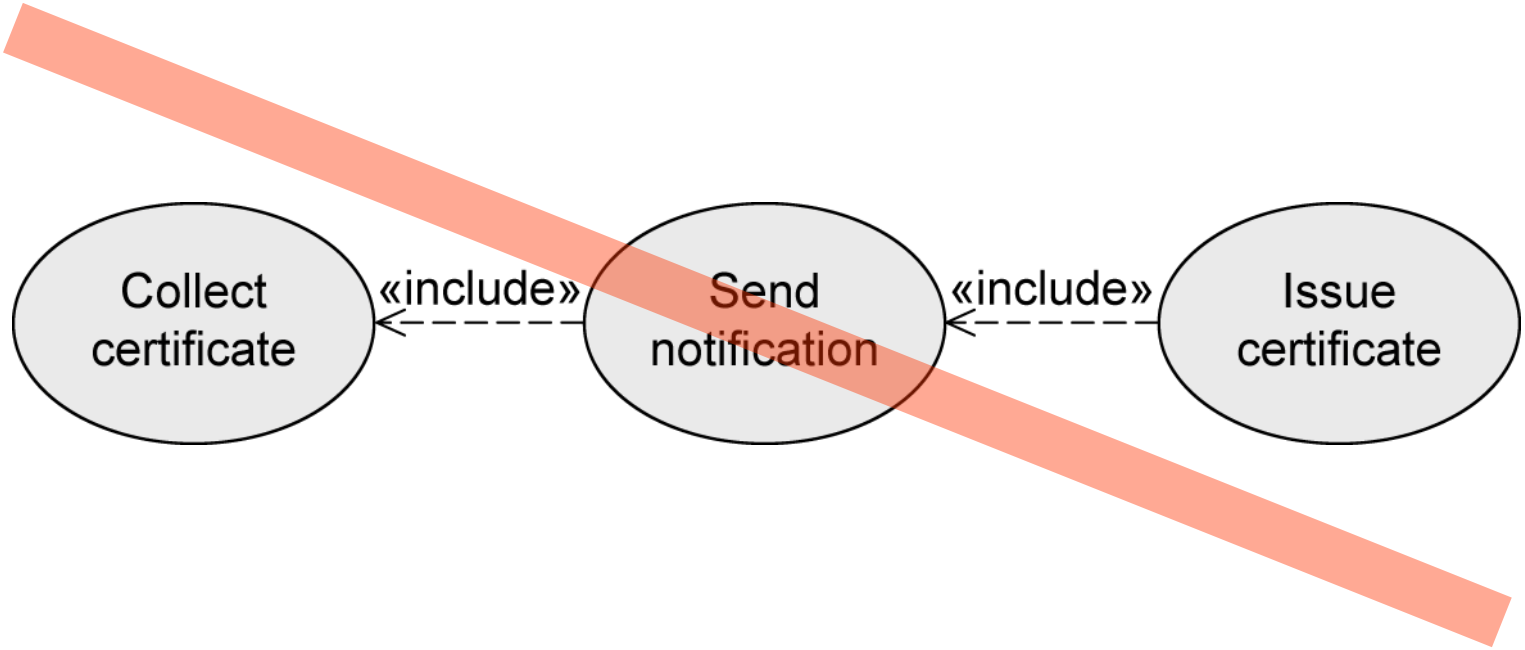
## Identificar Actores

- ¿Quién ejecuta el caso de uso principal?
- ¿Quién necesita soporte para el dia a dia?
- ¿Quién es el responsable de la administración?
- ¿Cuáles son los sistemas externos con los que se debe comunicar el Sistema?
- ¿Quién recibe los resultados del Sistema?

# Buenas Prácticas

## Errores típicos

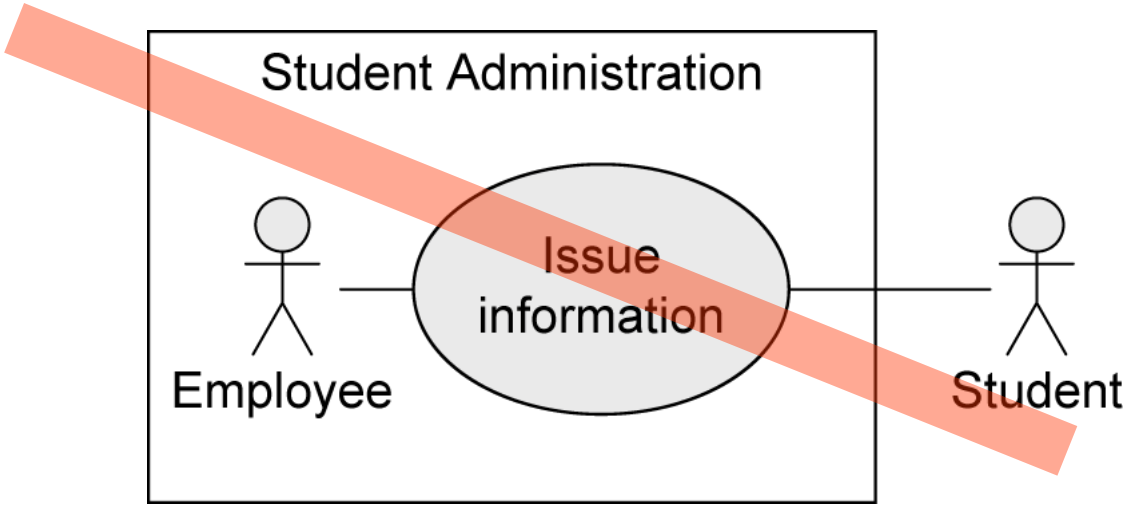
- Usa diagramas,  no se modelan procesos ni flujos



# Buenas Prácticas

## Errores típicos

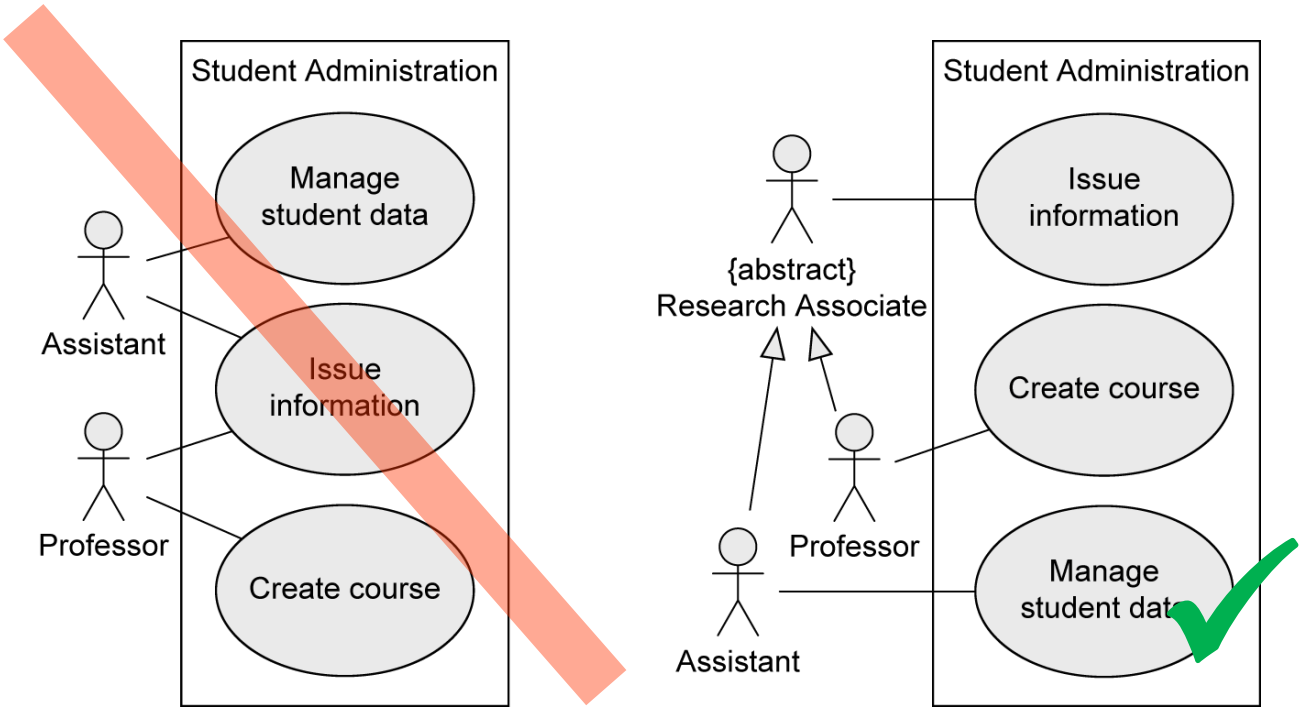
- Los actores no son parte del sistema



# Buenas Prácticas

## Errores típicos

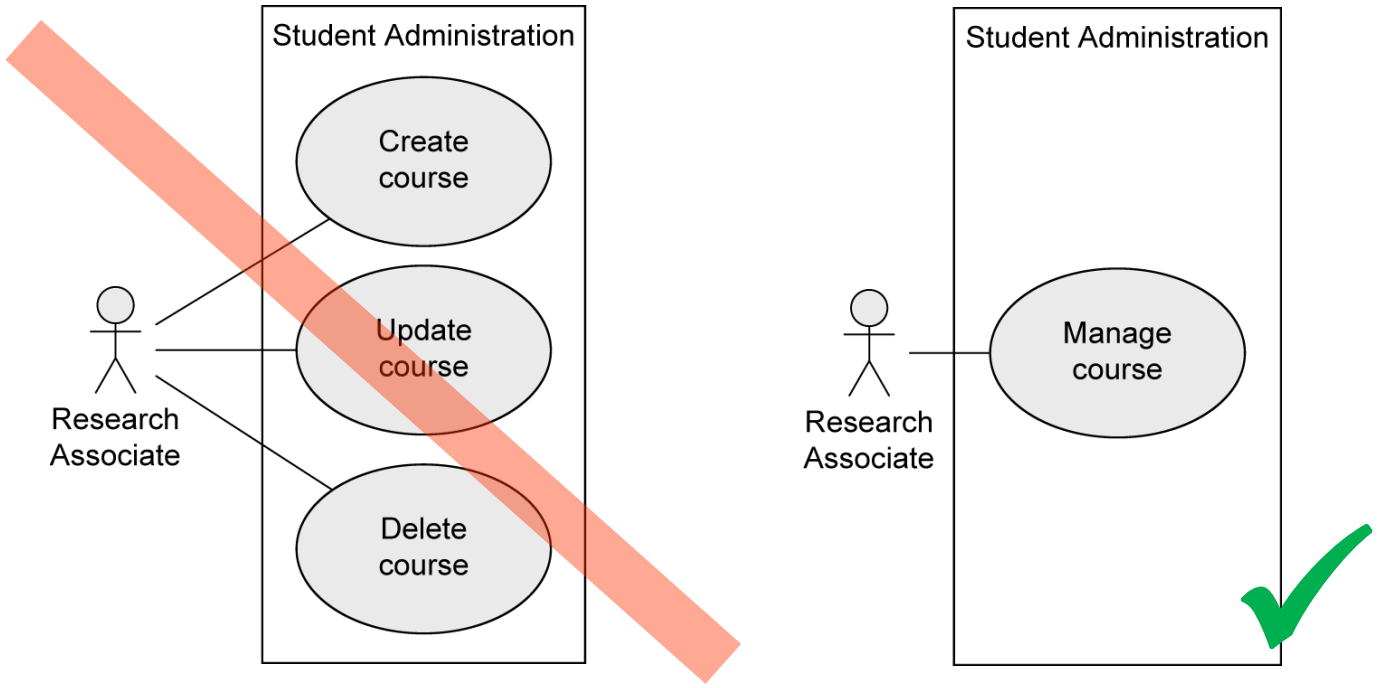
- Use case **`Issue information`** needs **EITHER** one actor **`Assistant`** **OR** one actor **`Professor`** for execution



# Buenas Prácticas

## Errores típicos

- Many small use cases that have the same objective may be grouped to form one use case



# Buenas Prácticas

## Errores típicos

- The various steps are part of the use cases, not separate use cases themselves! -> NO functional decomposition