Tema 6: Modelado estructural

Análisis y Diseño del Software
Grado en Ingeniería Informática

Departamento de Estadística, Informática y Matemáticas
Universidad Pública de Navarra-*Nafarroako Unibertsitate Publikoa*

Curso académico 2022-2023

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Objetivos del tema

- Adquirir la capacidad de estructurar los objetos de una aplicación en un diagrama
- Describir las características y diseño de los diagramas de clase
- Organización de las clases en diagramas de paquetes

Que es un Objeto

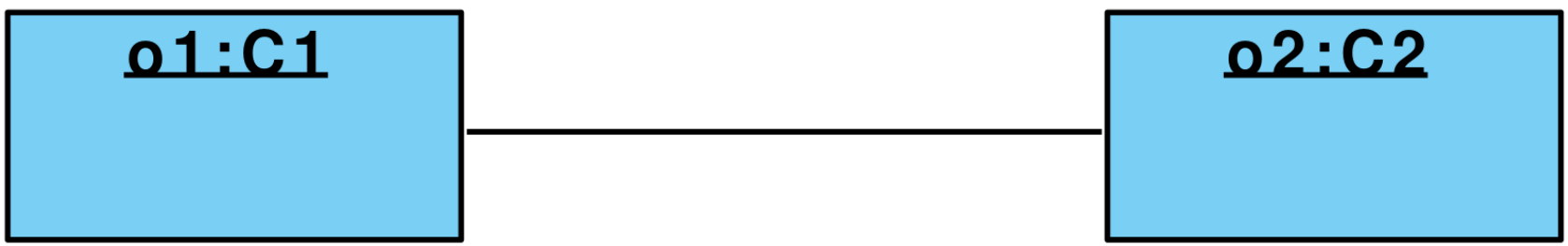
- Un sistema esta compuesto por individuos diferentes
  - No solo personas, cualquier elemento que puede ser identificado unicamente
  - Alice, Bob, la asignatura de Análisis y Diseño del Software, el Grado en Ingeniería Informática
- Son instancias de una *clase*
  - El *objeto* tiene un estado
  - El estado es expresado por los valores de los atributos

Diagrama de Objetos

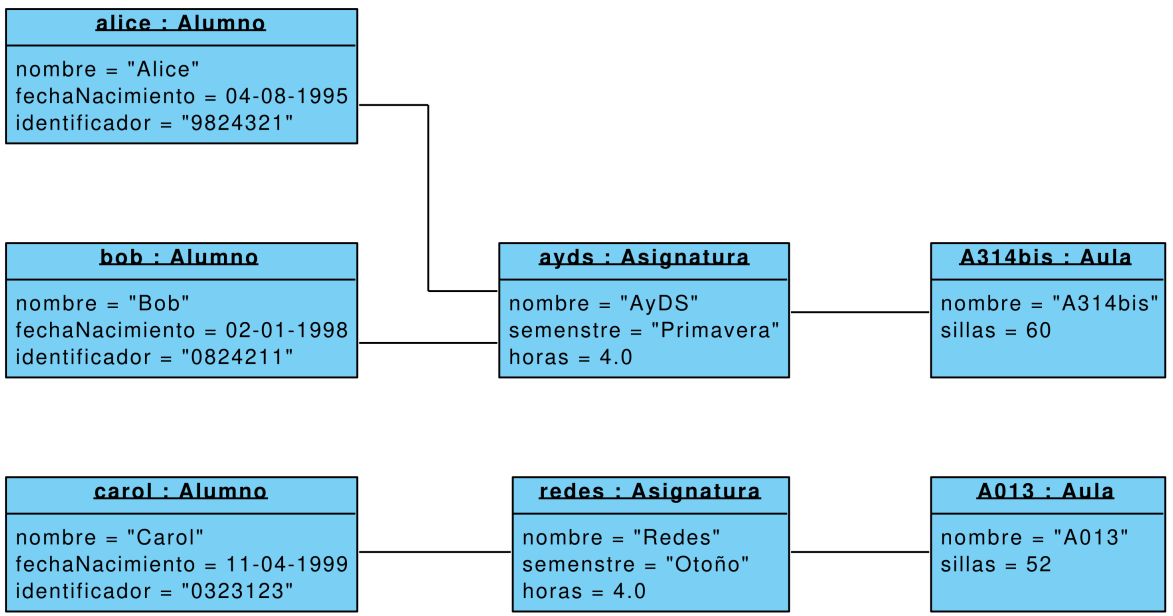
- En UML, para mostrar los objetos y sus relaciones se usa un **Diagrama de Objetos**
  - Permite mostrar los objetos del sistema en un instante concreto
- Los objetos tienen una clase asignada
  - Los atributos están definidos
  - Puede darse el caso de que dos objetos tengan los mismos atributos, pero al ser instancias diferentes son tratados como dos objetos
  - Las clases se organizan en un **Diagrama de Clases**

Elementos Diagrama de Objetos

- Objetos
  - Instancias de clases
  - Se asignan valores concretos a los atributos
  - Las operaciones no se muestran, ya están definidas en la clase
- Enlaces
  - Relacionan los objetos entre si
  - Puede ser entre mas de dos objetos, o un objeto consigo mismo

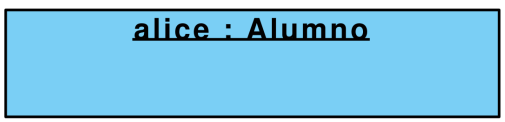


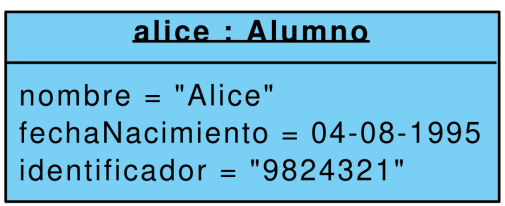
Ejemplo Organización de Objetos

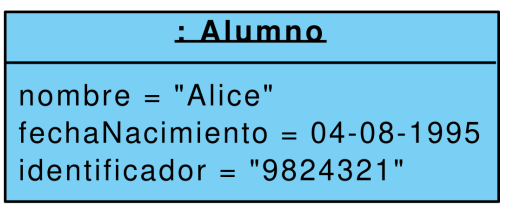


Notación de Objetos

- Los objetos se muestran como un rectángulo que puede dividirse en varios compartimentos
  - El primer compartimento siempre contiene información en formato `nombreObjeto:Clase`, centrado y subrayado
    - Si un objeto no tiene nombre, se considera un objeto anónimo
  - El segundo compartimento es opcional contiene los atributos y el valor







Diferencias entre Objetos

- Que dos objetos tengan los mismos valores de atributos no implica que sean el mismo objeto
  - Dos personas pueden tener el mismo nombre y apellidos
- Los valores de los atributos cambian con el tiempo, pero el objeto no cambia
  - Una persona puede cambiarse el nombre o el apellido
  - El Diagrama de Objetos siempre muestra instantánea del sistema
- Si un objeto no se muestra no indica que no exista, simplemente no es relevante en ese momento

Del Objeto a la Clase

- Muchos de los objetos que aparecen en el sistema tienen características y comportamientos similares
  - Los estudiantes tienen identificadores
  - Los cursos tienen un nombre y un numero de horas
- No se puede modelar cada elemento del sistema de forma individual
  - El modelo seria extremadamente complejo
- Las **clases** nos permiten describir objetos similares sin necesidad de detallar cada uno de manera individual

1. Diagrama de Objetos

2. Diagrama de Clases
   - Clases
   - Asociaciones
   - Clases de Asociación
   - Agregaciones/Composiciones
   - Generalizaciones
   - Clases Abstractas e Interfaces
   - Tipos de Datos

3. Diagrama de Paquetes

4. Ejercicio

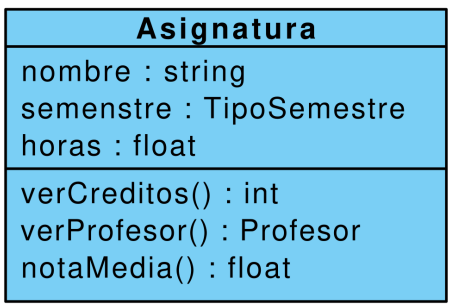
El Diagrama de Clases

- Un Diagrama de Clases modela la estructura estática del sistema
  - Los elementos y sus relaciones
- Es el diagrama mas usado dentro de UML
- El nivel de detalle es diferente en cada fase
  - En las primeras fases permite crear una vista conceptual del sistema
  - Según avanza el proyecto se refina hasta llegar a las clases que se programaran
- Normalmente se usa con fines de documentación, pero se puede generar código automáticamente a partir de un Diagrama de Clases

Que es una Clase

- Una plantilla para un conjunto de objetos similares
  - Una clase puede caracterizar personas, edificios, eventos...
- Los objetos se crean a partir de una clase a base de la **instanciación**
- Describe características estructurales (atributos) y comportamiento (operaciones)
  - Los atributos permiten almacenar información
  - Las operaciones habilitan la comunicación entre objetos

Ejemplo de UML a Java



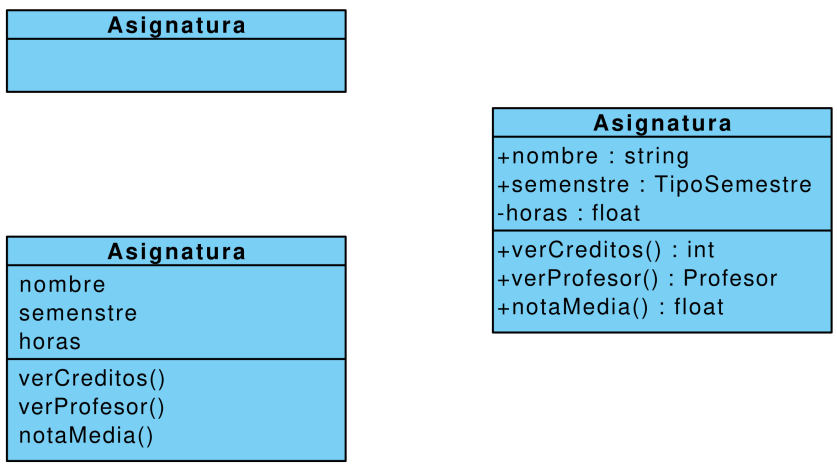

```
class Asignatura {
    String nombre;
    TipoSementre semestre;
    float horas;

    int verCreditos();
    Profesor verProfesor();
    float notaMedia();
}
```

Nivel de Detalle y Abstracción

- Para asegurar que el modelo es entendible no se modelan todos los detalles
  - Solo se incluye aquella información que sea relevante para la fase del proyecto en el que estemos, y para que se implemente el sistema
- Debemos abstraernos de la realidad y simplificar el modelo y restringirlo a lo que sea necesario
  - Al modelar alumnos de una universidad es importante gestionar los nombres y las matriculas de los alumnos, pero su talla de zapatillas es irrelevante

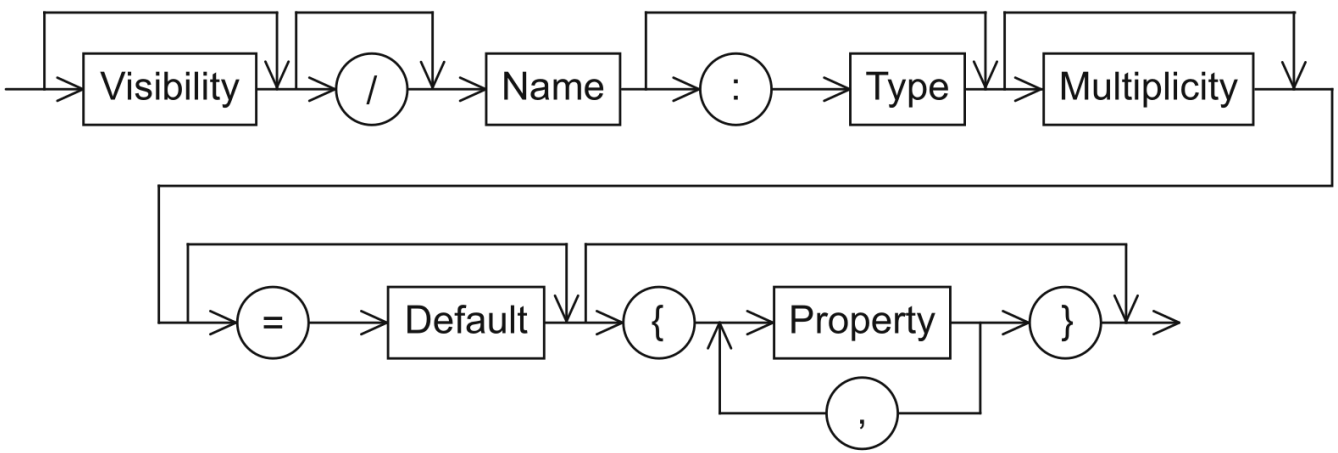
Notación

- Una clase es representada por un recuadro con varios compartimentos
  1. El nombre de la clase
  2. Los atributos
  3. Las operaciones
- El nivel de detalle refleja la fase del desarrollo del proyecto



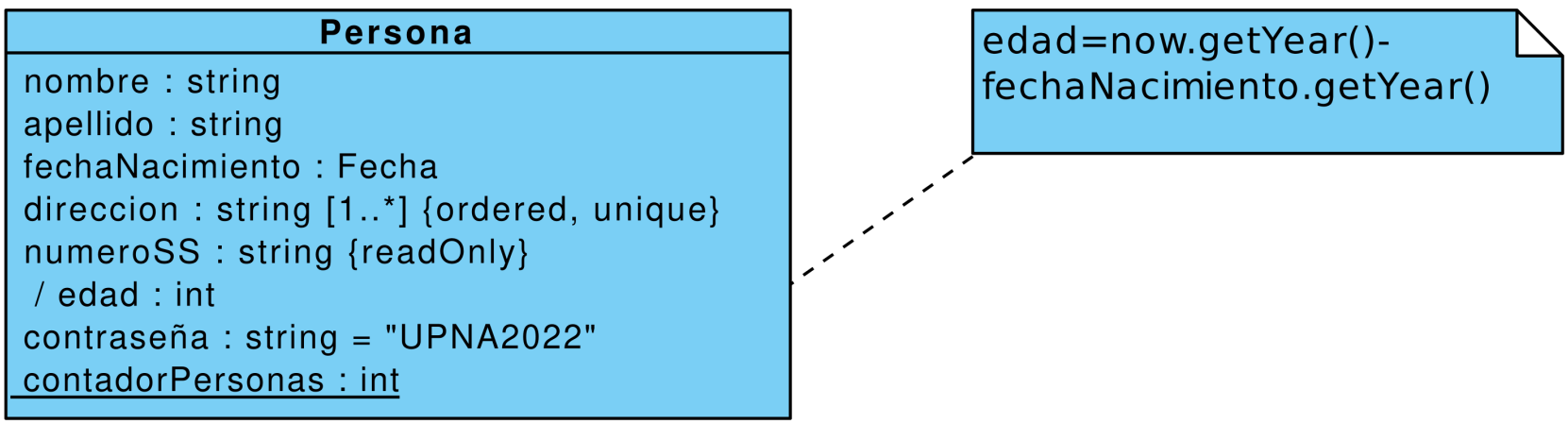
Atributos

- Cada atributo tiene al menos un nombre
- El tipo se especifica después del nombre separándolo con :
  - Puede ser primitivo, compuesto (fecha, enumeración) u otras clases
- Se les puede dar un valor por defecto especificado con =
- Si el atributo empieza con /, significa que se deriva de otros atributos



Detalles Atributos

- Los atributos pueden tener propiedades que dan mas detalles
- Si un atributo esta subrayado significa que es de la clase y no de objeto (variables globales)



Multiplicidad de los Atributos

- La multiplicidad de un atributo indica cuantos valores puede contener
  - Definición de arrays
- Por defecto todos los atributos tienen una multiplicidad de 1
- `[min..max]`
  - `[1..*]` → una lista con al menos un elemento
  - `[5]` → el atributo tiene exactamente 5 valores
  - `[0..*]` o `[*]` → una lista que puede estar vacía

Especificación de Conjunto/Lista

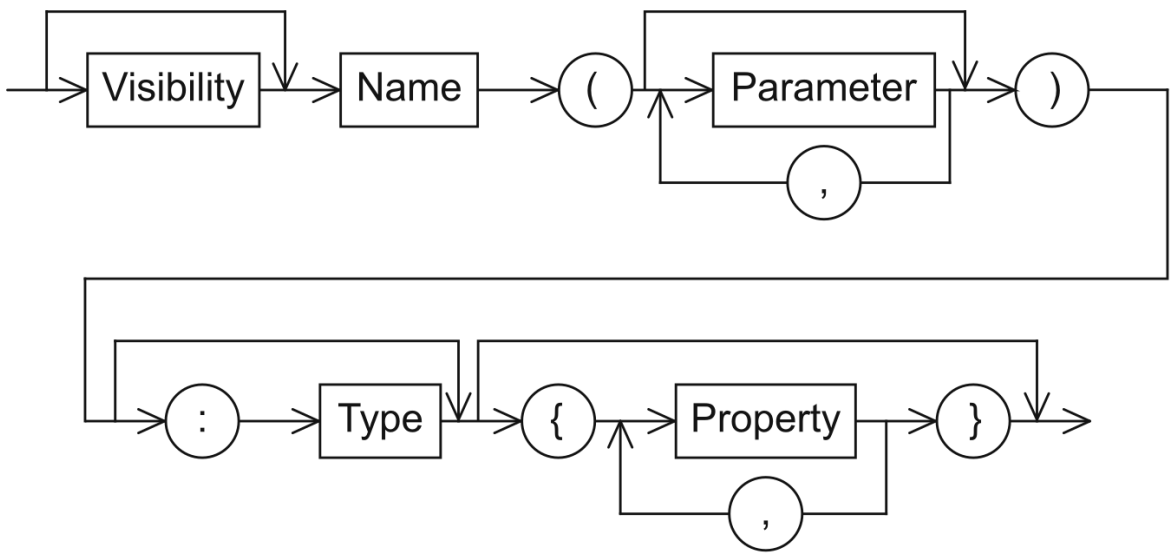
- Si un atributo puede tener mas de un valor se puede definir a que tipo de lista o conjunto pertenece usando propiedades

set:	{unordered, unique}
multi-set:	{unordered, non-unique}
ordered set:	{ordered, unique}
list:	{ordered, non-unique}

- `direccion:string[1..*]{ordered, unique}`
  - Una persona puede tener varias direcciones, pero sin repeticiones
  - Al ordenarlas damos relevancia al orden en el que se guardan
    - La primera podría ser su dirección principal

Operaciones

- Una operación esta caracterizada por su nombre
- Cuando se llama a una operación el comportamiento asignado a esa operación es ejecutado
  - Corresponde a la declaración de un método o función
- En el diagrama de clases solo se muestran, no se detalla su ejecución
  - Diagramas de Secuencia o Actividad



Parámetros y Return

Parámetros: El nombre de la operación es seguido por una lista de parámetros entre paréntesis

- La lista puede estar vacía
- Los parámetros siguen la misma nomenclatura que los atributos
- El único elemento obligatorio es el nombre

Return: La operaciones pueden opcionalmente devolver un valor

- Se especifica el tipo de la variable devuelta

```
actualizarApellido(nuevoApellido:string):boolean
```

Parámetros de Entrada/Salida

- Si fuese necesario se puede especificar la dirección al nombre de los parámetros, por defecto se usa `in`

  in: Parámetros de entrada, deberían tener un valor

  out: Parámetros de salida, al finalizar la operación este parámetro tendrá un nuevo valor. Si una operación necesita devolver varios valores

  inout: Combinación de `in` y `out`

```
getNombre(out nombre:string, out apellido:string):void
```

Marcadores de Visibilidad

- La visibilidad de los atributos u operaciones especifica quien puede acceder a ellos
- En UML damos información sobre la visibilidad añadiendo un símbolo por delante del nombre del atributo u operación
- La implementación de la visibilidad dependerá exclusivamente del soporte que tenga el lenguaje de programación
  - Dependiendo el lenguaje el significado de los nombres de UML no se corresponde
  - Por ejemplo `protected` de Java tiene la misma visibilidad que package en UML

Tabla Marcadores de Visibilidad

Nombre	Símbolo	Descripción
public	+	Acceso permitido por objetos de cualquier clase
private	-	Acceso permitido solo desde el propio objeto
protected	#	Acceso permitido por objetos de esa clase y subclases
package	~	Acceso permitido por objetos cuyas clases estén en el mismo paquete

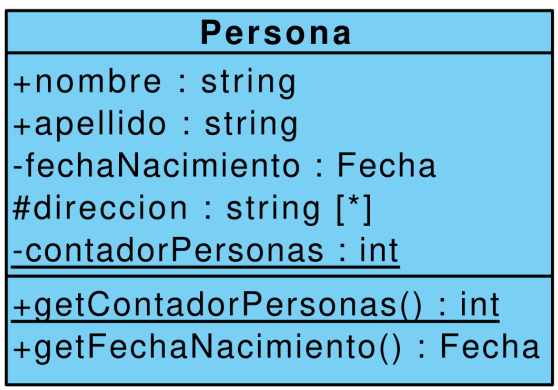
Ocultar Información

- El concepto de limitar el acceso a la información es importante en informática
- Marcar los atributos que definen el estado de un objeto como `private` protege estos a accesos no autorizados
  - Se puede controlar el acceso a través de operaciones publicas
- Ayuda a encapsular funcionalidades en objetos
- En algunos casos los diagramas solo contienen aquellas operaciones o atributos marcados como `public`
  - Simplifica el diagrama
  - Resalta las partes importantes para el uso de la clase

Instancias y Atributos/Operaciones

- Los atributos generalmente se definen a nivel de instancia
  - Si un objeto p1 es una instancia de `Persona` podemos acceder a su nombre con `p1.nombre`
  - Para acceder a los atributos u operaciones necesitamos una instancia de esa clase
- Una variable de clase, o atributo estático, es creado solo una vez por clase, no por instancia
  - Constantes como π
  - Operaciones estáticas o de clase también funcionan de la misma manera
- En UML para indicar que un atributo u operación es estática se subraya

Variables y Operaciones Estáticas



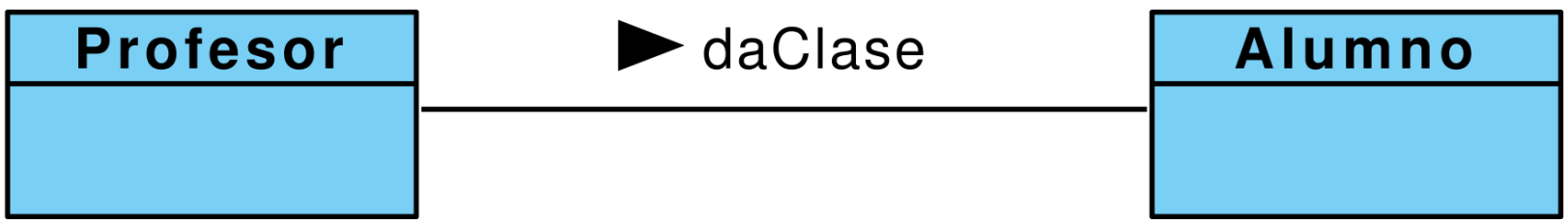

```
class Persona {
    public String nombre;
    public String semestre;
    private Fecha fechaNacimiento;
    protected String[] direccion;
    private static int contadorPersonas;

    public static int getContadorPersonas(){}
    public Fecha getFechaNacimiento(){}
}
```

Que es una Asociación

- Las asociaciones entre clases modelan relaciones, también conocidas como enlaces, entre instancias de las clases
- Podrán acceder al otro elemento dependiendo de la visibilidad de los atributos y operaciones
- Un Diagrama de Clases puede verse como un grafo en el que las clases representan los nodos y las asociaciones las vértices
- Hay dos grupos de asociaciones, binarias y n-arias

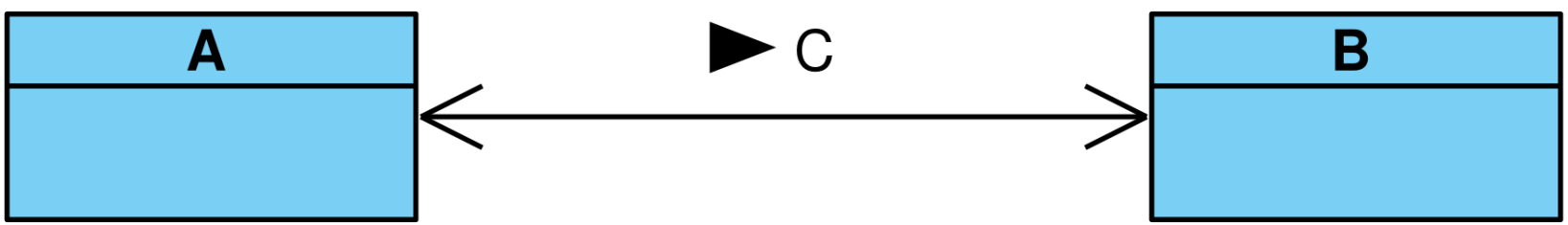
Asociación Binaria

- Permite la asociación de las instancias de dos clases
- La asociación puede tener un nombre
  - En caso de que tenga nombre la dirección de la asociación se puede indicar con un triangulo, indicando en que dirección hay que leerla



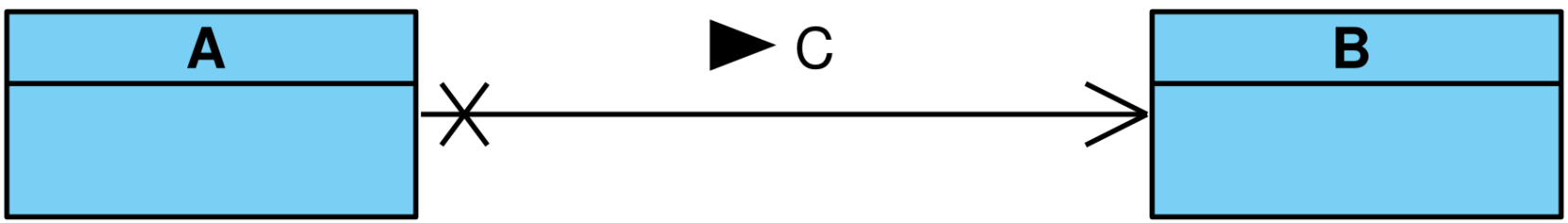
Navegabilidad de una Asociación

- Por defecto las asociaciones de UML son navegables en ambas direcciones
- Se puede especificar explícitamente con una flecha en la direccion en la que es navegable
- Esto implica que un objeto conoce al otro
  - Un atributo del objeto hace referencia al otro

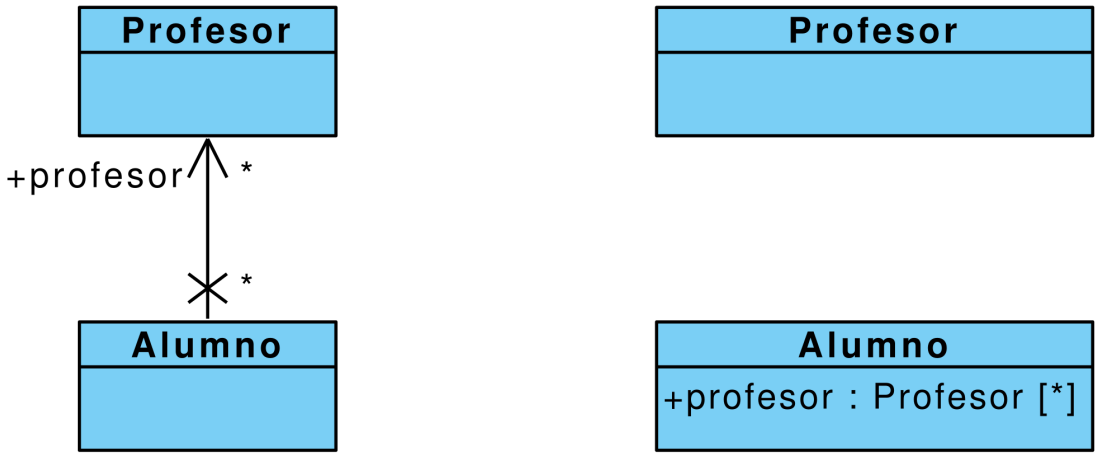


Asociación no Navegable

- En algunos casos queremos indicar que una asociación no es navegable en una de las direcciones
- Una X en uno de los lados de la asociación indica que no se nabegable en esa direccion
  - A puede acceder a los atributos y operaciones de B
  - B no puede acceder a los atributos y operaciones de A
- Da mas información a la hora de implementarla



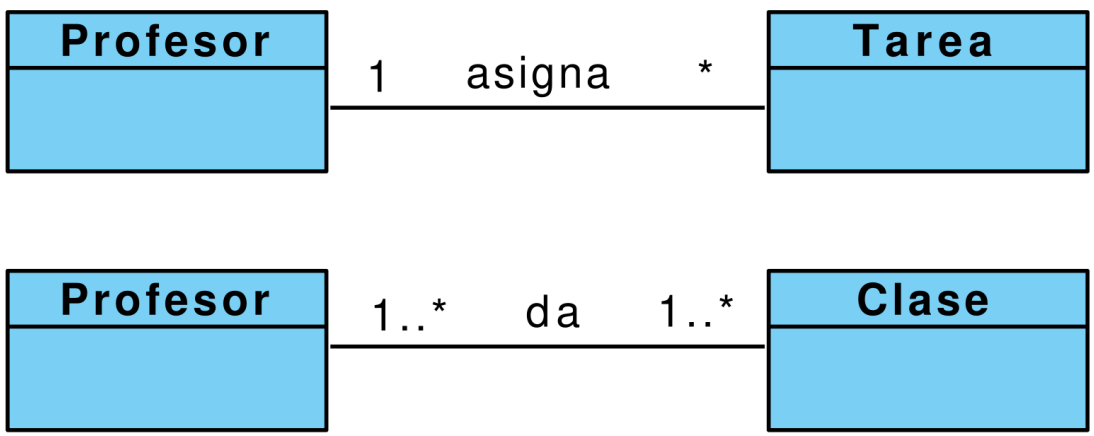
Asociación en Java




```
class Profesor {...}

class Alumno {
    public Profesor[] profesor;
}
```

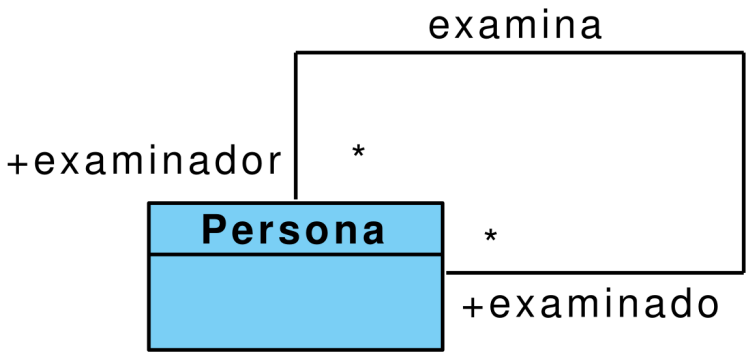
Multiplicidad

- De la misma manera que los atributos y operaciones, las asociaciones también tienen multiplicidad
- Se especifica el numero de objetos que puede estar *relacionado con exactamente un objeto del lado contrario*
- Un profesor puede asignar 0, 1 o muchas tareas, y cada tarea es asignada por un profesor
- Un profesor da al menos una clase, y cada clase tiene al menos un profesor



Roles de Asociaciones

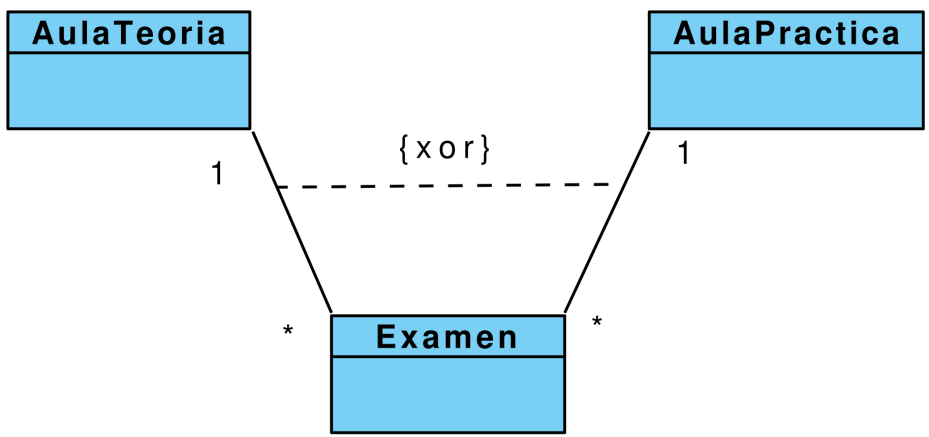
- Las asociaciones pueden tener un rol asignado
  - Este rol se traducirá a la variable que se crea para la asociación
  - Se puede indicar la visibilidad
  - Hereda la multiplicidad de la asociación



Puede una persona examinarse a si misma?

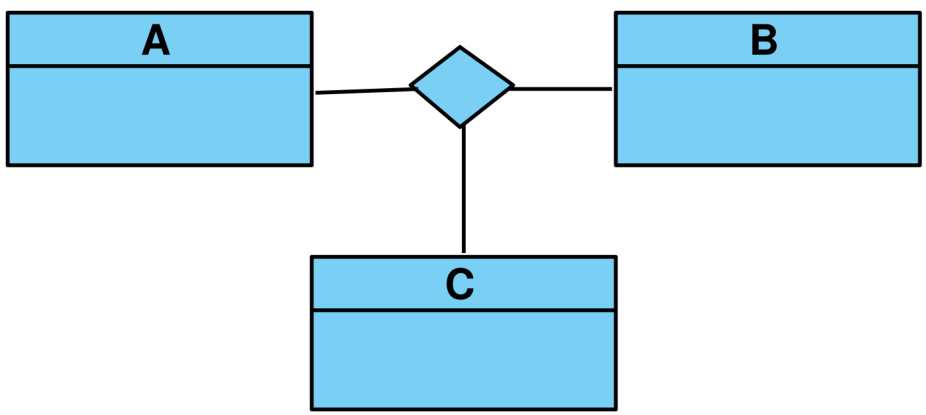
Asociación xor

- Si queremos expresar que un objeto puede estar relacionado con una instancia de una clase u otra, pero no ambas, podemos usar una restricción xor
- Un examen puede tener lugar en un aula de teoría o de practicas, pero no en ambos



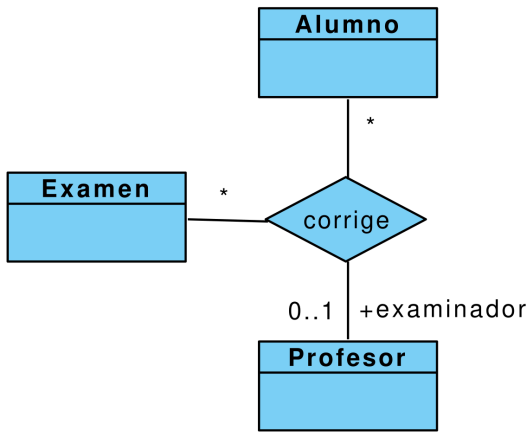
Asociaciones n-arias

- Si hay mas de dos objetos en una relación lo modelamos usando asociaciones n-arias
- Se puede especificar multiplicidad y roles
  - La multiplicidad en este caso implica cuantos objetos se pueden asignar a una tupla que contenga los otros objetos

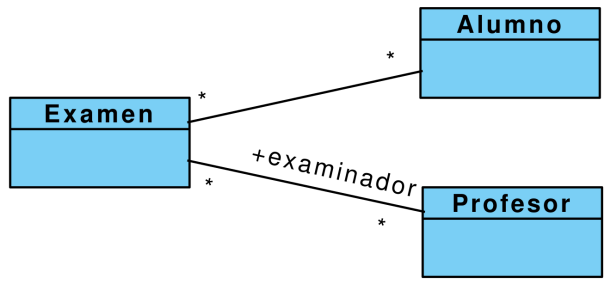


Ejemplo Asociación Ternaria

Es posible especificar que el alumno a1 hizo el examen e1 con el examinador p1 y el alumno a2 hizo el mismo examen e1 con otro examinador p2

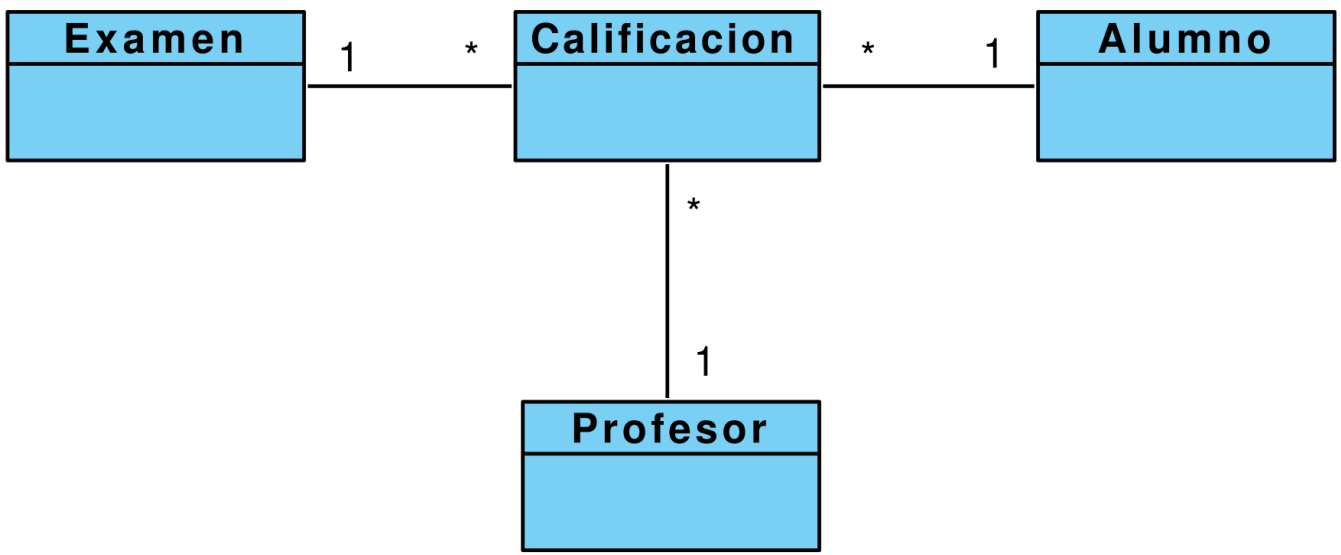


Solo se puede mostrar que los alumnos a1 y a2 hicieron el examen e1 con los examinadores p1 y p2. No se puede saber quien hizo el examen con que profesor



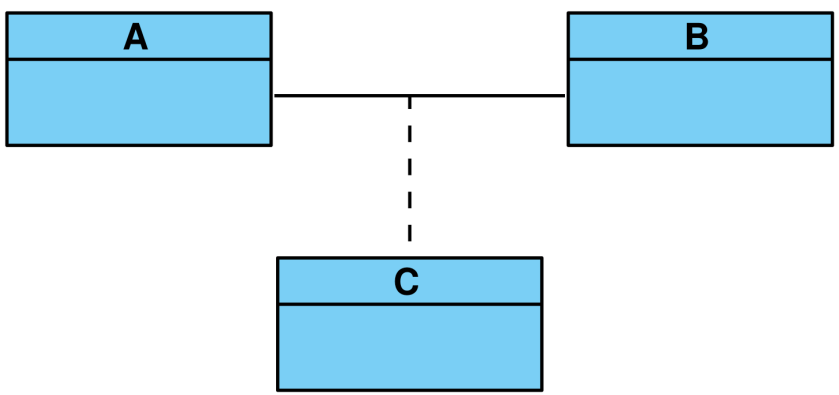
Simplificación con un Clase

- Como alternativa se puede modelar la asociación con una clase intermedia
- Un inconveniente de este modelo es que un alumno puede ser calificado mas de una vez para el mismo examen

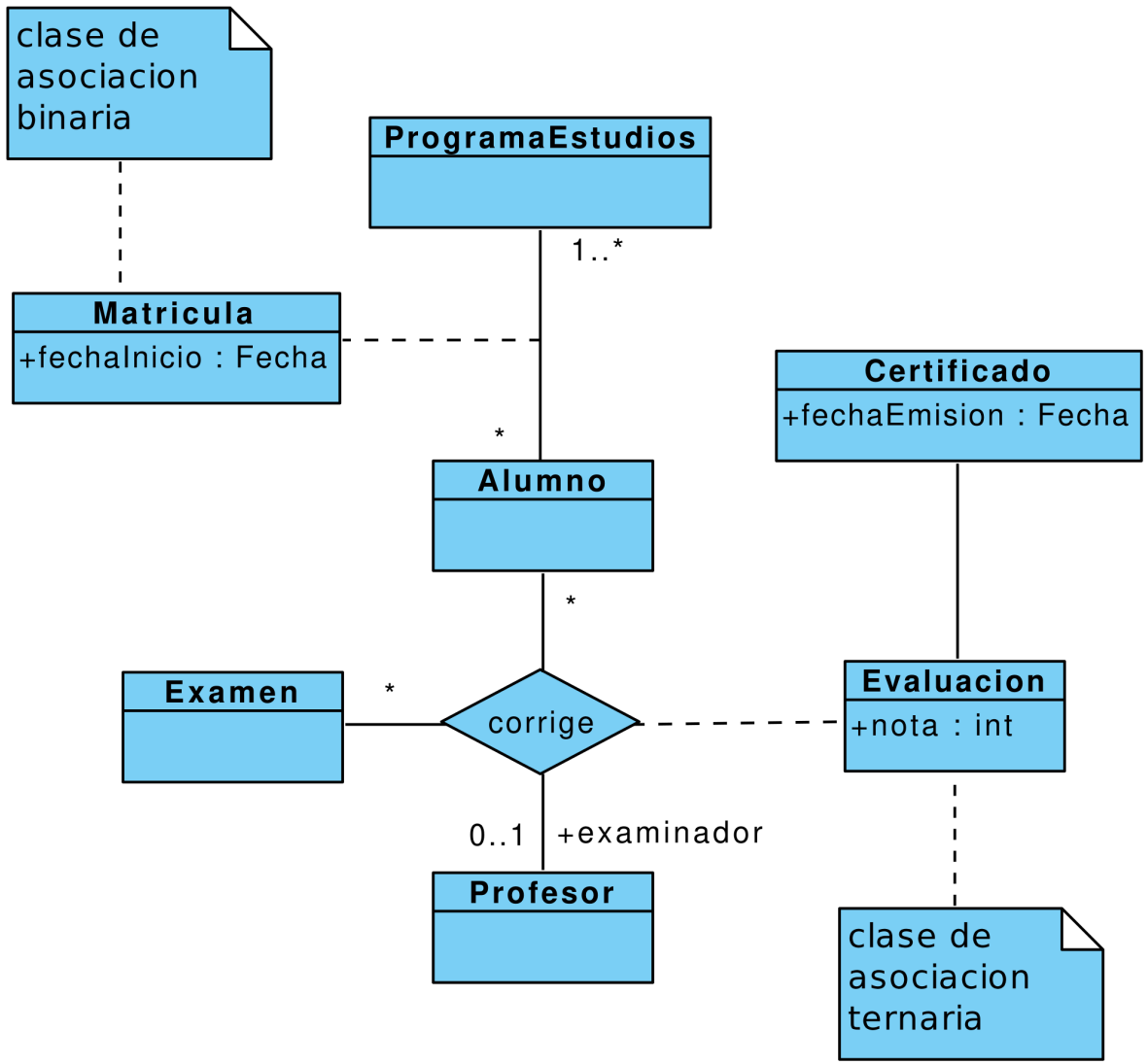


Clases de Asociación

- A veces debemos asignar atributos u operaciones a una relación entre dos clases
- Podemos relacionar una clase a la asociación entre otras dos clases
  - Puede ser binaria o n-aria
- La clase de la asociación comparte el nombre con la propia asociación
- Puede tener mas asociaciones con otras clases

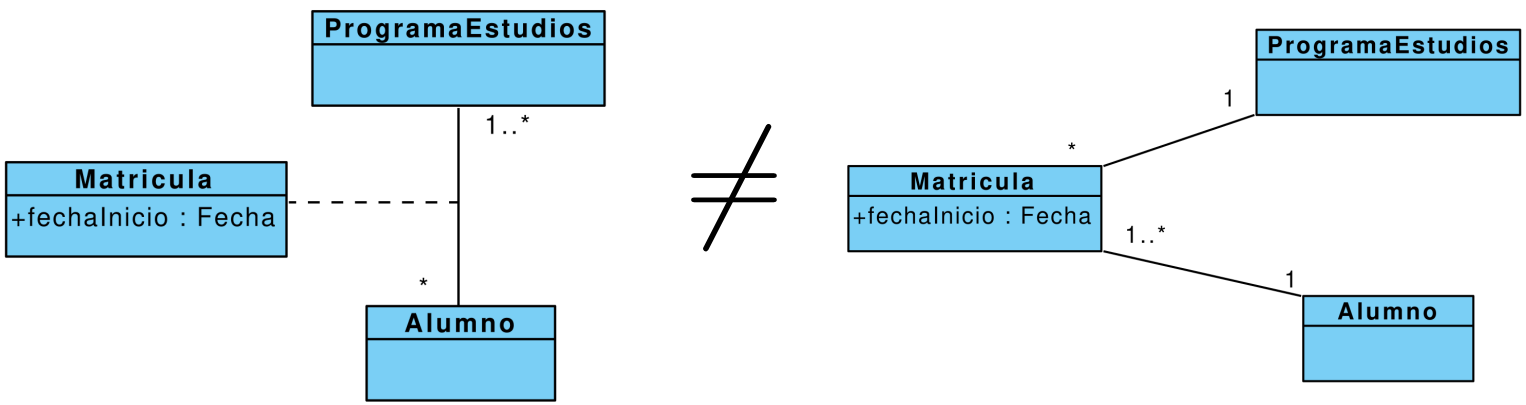


Ejemplo de Clases de Asociación



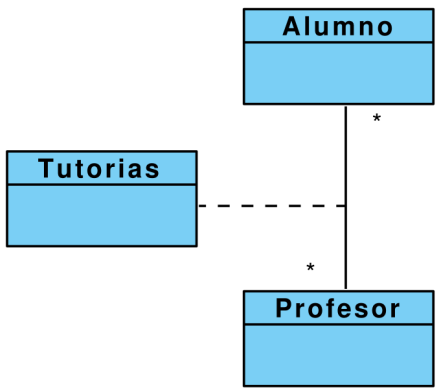
Diferencias con Clases "Normales"

Por norma general no se puede sustituir una clase de asociación por una clase "normal". En el primer ejemplo un alumno solo puede matricularse **una vez** en un programa de estudios, en el segundo no se cumple esa restricción

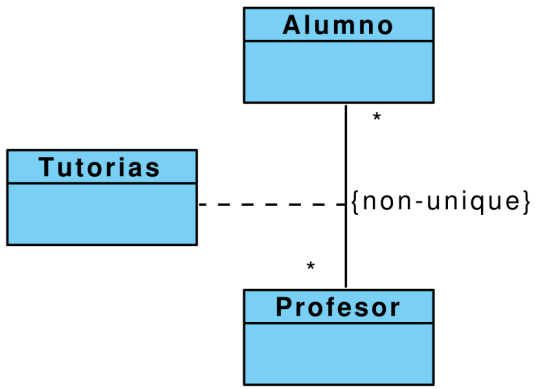


Por Defecto unique

- Por defecto las clases de asociación tienen la restricción de {unique}
- Debemos marcar la asociación como {non-unique} para permitir duplicados



El alumno solo puede tener una tutoría con cada profesor



El alumno puede tener varias tutorías con cada profesor

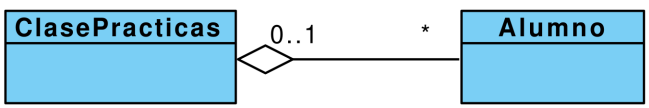
Relación Parte-Todo

Agregaciones

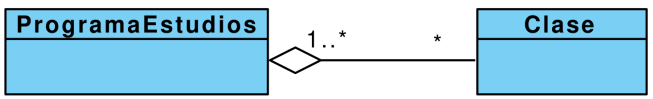
- Una agregación es una asociación especial en la que las instancias de una clase son parte de la instancia de otra
- Las relaciones de agregación son transitivas y asimétricas
  - Si B es parte de A, y C es parte de B, entonces C es parte de A
  - Si B es parte de A, A no puede ser parte de B
- UML diferencia entre dos tipos de agregaciones
  - Agregación compartida
  - Composición

Agregación Compartida

- Describe una composición débil entre las partes y el todo
  - Las partes pueden existir de manera independiente del todo
  - Las partes pueden pertenecer a mas de un elemento a la vez



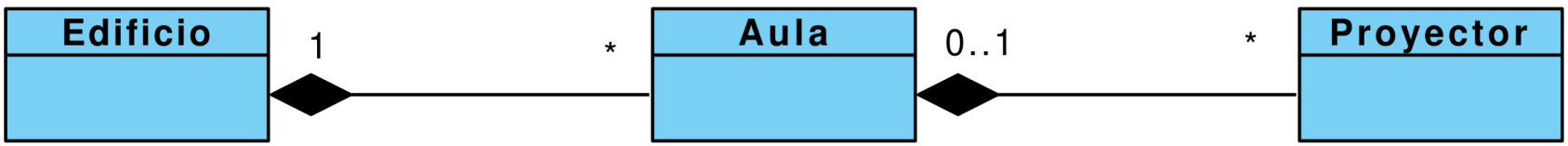
Una clase practica tiene varios estudiantes, pero cada estudiante solo puede participar en una



Cada programa de estudios esta compuesto por cualquier numero de clases y cada clase esta asignada al menos a un programa de estudios

Composición

- Las partes solo existen como dependencia del todo
  - Cuando el elemento que define el todo es eliminado las partes que lo componen también son eliminadas
- La multiplicidad en el lado compuesto es como máximo 1
  - Un edificio puede tener varias aulas, pero un aula solo puede pertenecer a un edificio
  - Al indicar una multiplicidad de `0..1` indicamos que el proyector puede existir sin aula, se puede retirar de ella, pero si el aula deja de existir, el proyector también dejara de existir

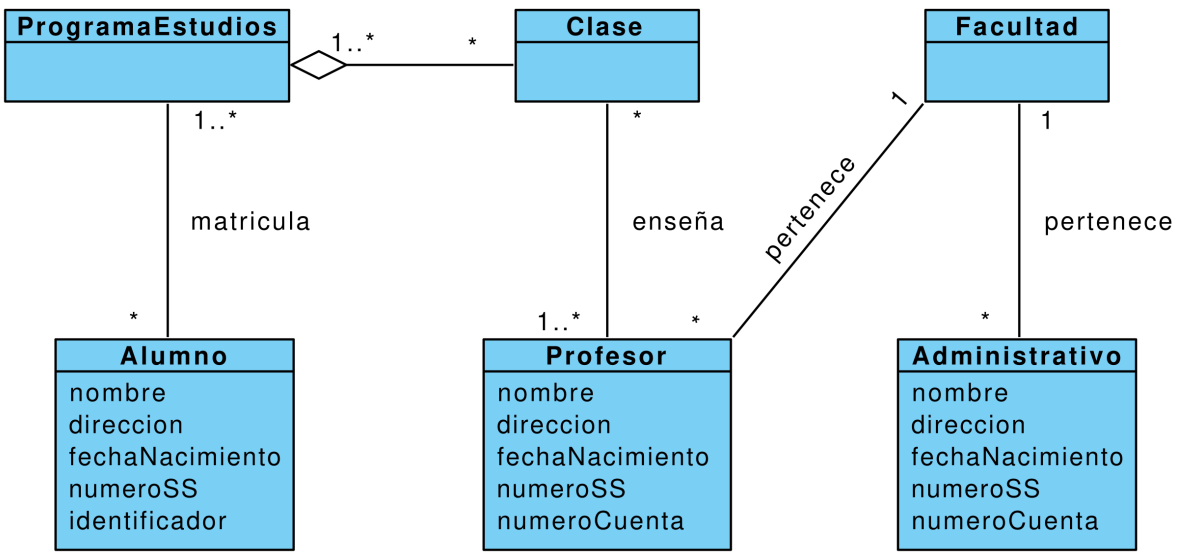


Como Seleccionar la Correcta

- La diferencia entre una agregación compartida y una asociación es que explicita la relación de ser "parte de"
  - Las partes de la agregación pueden seguir existiendo sin el elemento agregado
- En una composición la dependencia es mucho mayor
  - Si el elemento compuesto es eliminado o copiado, los componentes también son eliminados o copiados
  - Cuando los elementos están físicamente incluidos en el elemento compuesto, o solo son visibles para el
  - Si las partes pueden ser referenciadas externamente puede ser una indicación de que una agregación compartida es suficiente

Para que sirve la Generalización

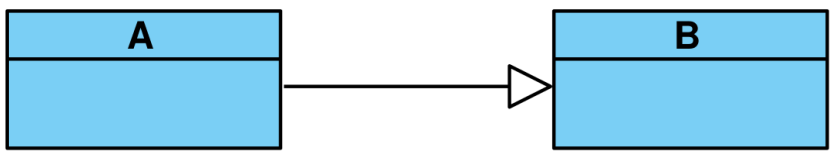
- A menudo clases diferentes tienen unas características comunes
  - La generalización sirve para resaltar lo que las clases tienen en común



Hay elementos comunes entre alumnos, profesores y administrativos

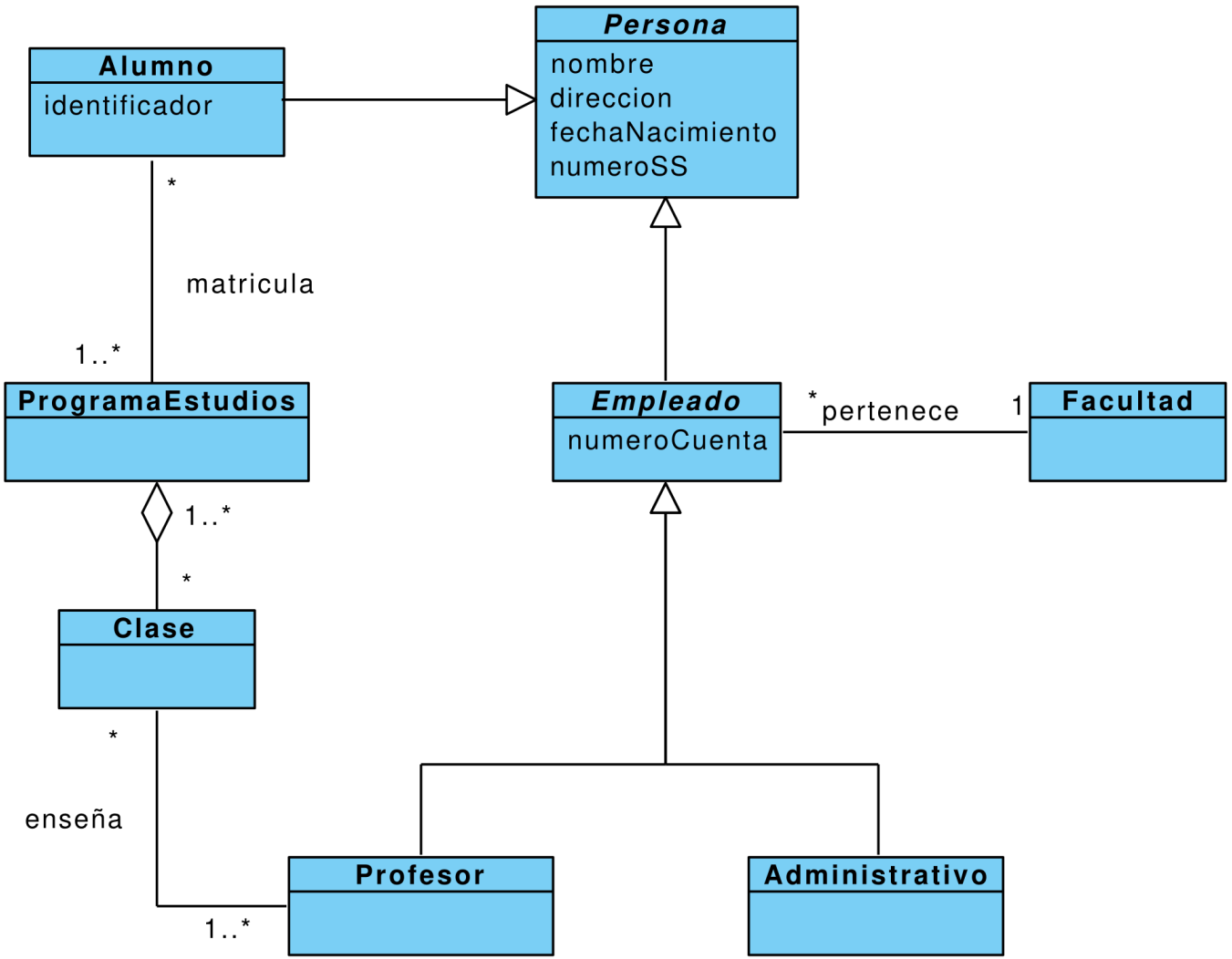
Relación de Herencia

- La relación de generalización expresa que las características y asociaciones de una clase genérica son heredadas por sus subclases
  - Una instancia de una subclase también es una instancia indirecta de la superclase
  - La subclase tiene acceso a los atributos y operaciones de la superclase dependiendo de la visibilidad



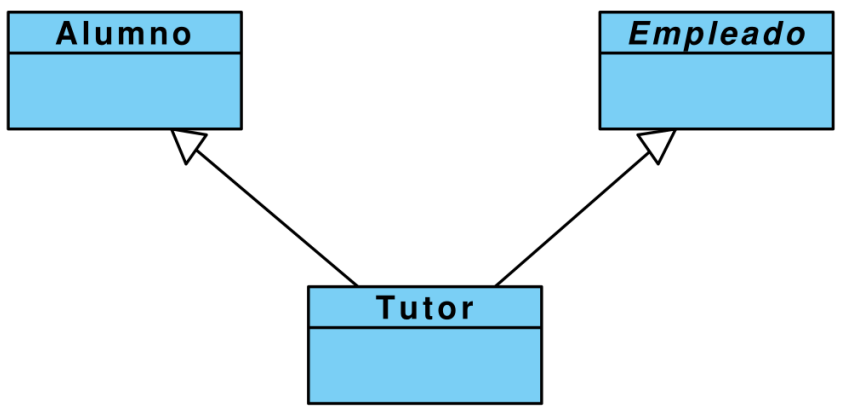
A hereda de B
B generaliza a A

Ejemplo con Herencia



Propiedad Transitiva y Herencia Múltiple

- La relación de generalización también puede llamarse una relación “es un”
  - En el ejemplo anterior un administrativo es un empleado, y gracias la propiedad transitiva también es una persona
- UML permite la herencia múltiple
  - Una subclase puede tener varias superclases
  - La implementación dependerá de cada lenguaje de programación



Clasificación

Clasificación

Relación de "`instanceOf`" entre un objeto y su clase

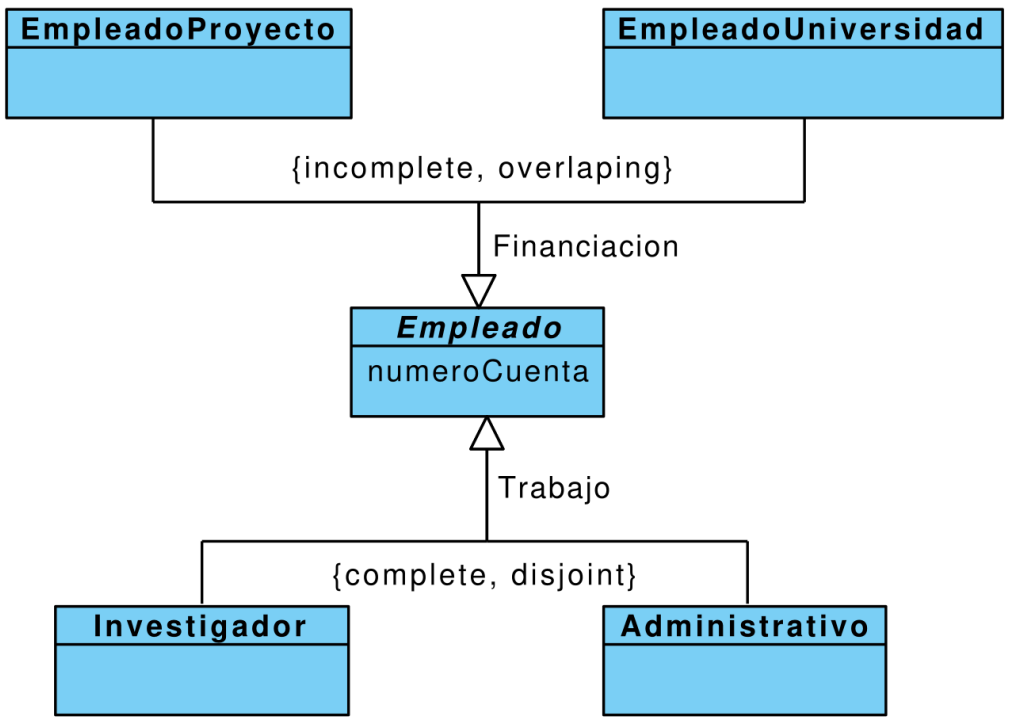
- Muchos lenguajes de programación orientados a objeto solo permiten a un objeto ser la instancia directa de una clase
- UML permite clasificación múltiple
  - Un objeto puede ser la instancia de varias clases sin que estas estén relacionadas entre si
  - No es herencia múltiple. No se introduce una nueva clase que herede de las superclases

Conjuntos de Generalización

- Cuando se agrupan las relaciones de generalización en las subclases creamos **conjuntos de generalización**
- Se pueden describir de manera mas precisa con restricciones
  - Coincidentes o disjuntos
    - Overlaping: Un objeto puede ser instancia de múltiples subclases a la vez
    - Disjoint: Un objeto solo puede ser la instancia de una subclase
  - Completos o incompletos
    - Complete: Cada instancia de la superclase debe ser una instancia de al menos una subclase
    - Incomplete: Cada instancia de la superclase no tiene por que ser una instancia de las subclases
  - Por defecto `incomplete, disjoint`

Ejemplo de Clasificación Múltiple

- Un empleado de la universidad debe ser un investigador o un administrativo
- Un empleado puede ser financiado directamente por la universidad, a través de un proyecto, los dos a la vez, o a través de otro método (una beca)



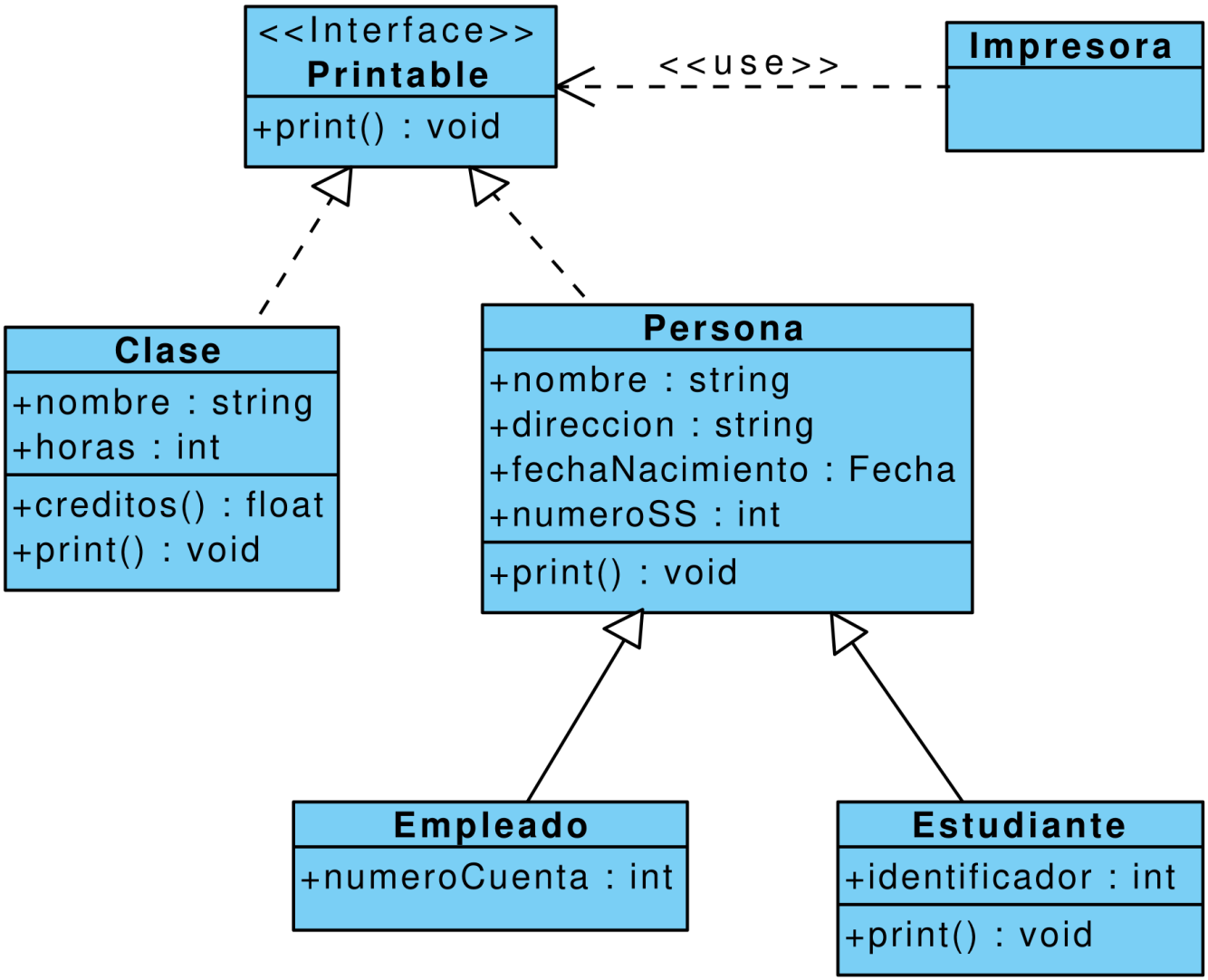
Clase Abstracta

- Si hay una clase que no se puede instanciar directamente la modelamos como **abstracta**
  - No existen objetos de esta clase, solo las subclases pueden instanciarse
  - Se usan para resaltar las características comunes de sus subclases
- Si una clase es abstracta se marca con {abstract} por delante del nombre o el nombre se escribe en cursiva
- Las operaciones de una clase también pueden ser abstractas
- La clase Persona en la diapositiva 51 es una clase abstracta

Interfaz

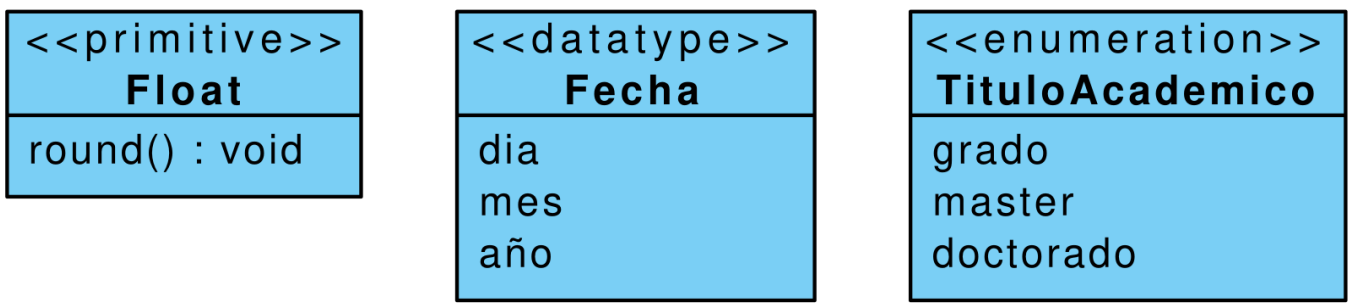
- Similar a la clase abstracta una interfaz no tiene implementación o instancias directas
- La interfaz representa un contrato
  - Las clases que implementan la interfaz se comprometen a cumplir el comportamiento especificado por la interfaz
- Las operaciones de una interfaz nunca tienen una implementación
- La clase se marca con «interface»
- Otras clases pueden hacer uso de la interfaz, sin implementarla

Ejemplo Uso de la Interfaz



Tipos de datos en UML

- Los atributos, parámetros y valores devueltos tienen un tipo
  - Un nombre tendrá el tipo `string`
  - El tipo puede ser una clase o un `data type`
- UML predefine cuatro tipos de datos primitivos
  - Boolean, Integer, UnlimitedNatural y String
- Es posible definir nuevas estructuras de datos u operaciones sobre ellas



1. Diagrama de Objetos
2. Diagrama de Clases
3. Diagrama de Paquetes
   - Pertenecía a un Paquete
   - Relaciones entre Elementos
4. Ejercicio

Que es un Paquete

- Un paquete es un elemento de UML que nos permite agrupar clases, tipos de datos, actividades, estados...
- Ayuda a organizar el proyecto asignando cada elemento al paquete correspondiente
  - Cada elemento del modelo solo puede ser incluido en un paquete
  - La inclusión de un elemento en un paquete define el namespace y la visibilidad
    - Si el paquete `P1` contiene la clase `C`, esta no puede confundirse por la clase `C` en el paquete `P2`
    - Podemos referirnos a ellos como `P1::C` y `P2::C`

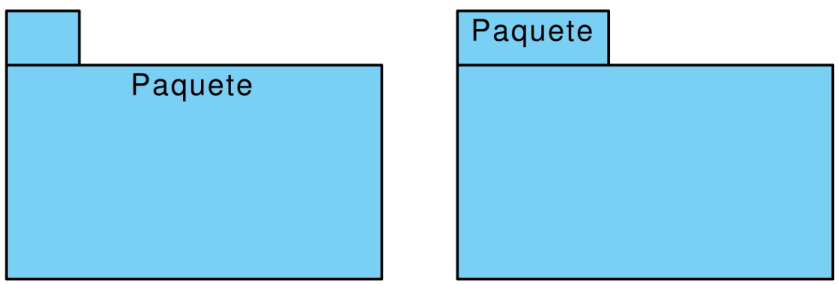
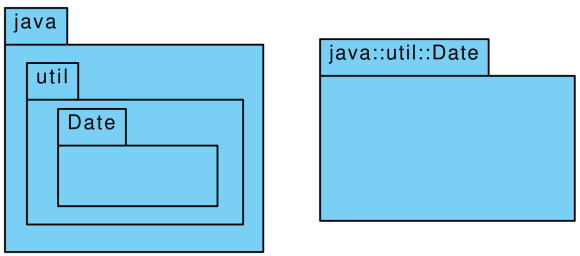


Diagrama de Paquetes

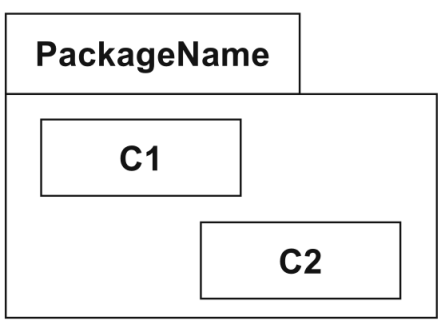
- Para la organización de paquetes podemos usar los **Diagramas de Paquetes**
  - También se pueden definir paquetes dentro del Diagrama de Clases
- En general un Diagrama de Paquetes nos va a ayudar a organizar las clases del sistema que estemos modelando
- Para mostrar las relaciones entre paquetes podemos usar los tipos de asociaciones que hemos introducido para el diagrama de clases
  - Hay otras relaciones especificas para paquetes

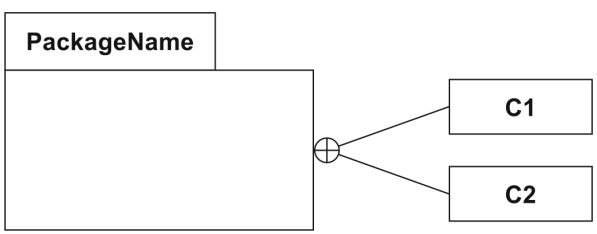
Elementos Pertenecientes a un Paquete

- Podemos introducir un paquete dentro de otro o usar nombres cualificados



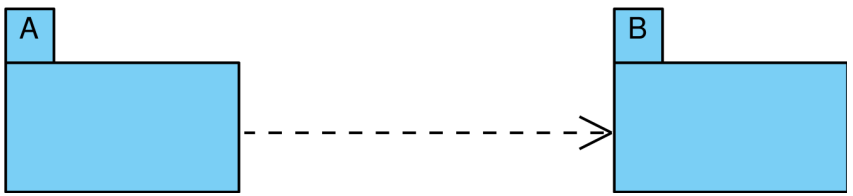
- Las clases pueden ir dentro de un paquete o relacionarlos con una relación de contención
  - De la misma manera que los atributos o las operaciones tienen una visibilidad, los elementos del paquete también pueden tenerla



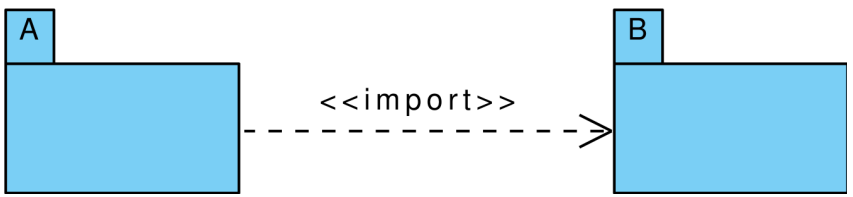


Relaciones entre Paquetes

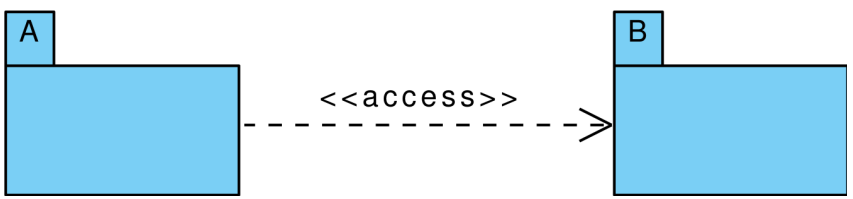
- Relación de dependencia básica



- Un paquete importa la funcionalidad del otro, puede acceder a los elementos visibles



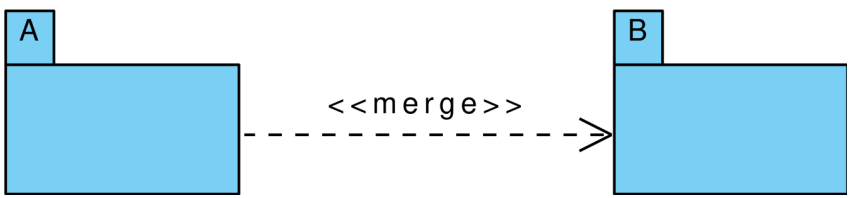
- Un paquete requiere ayuda de las funciones del otro



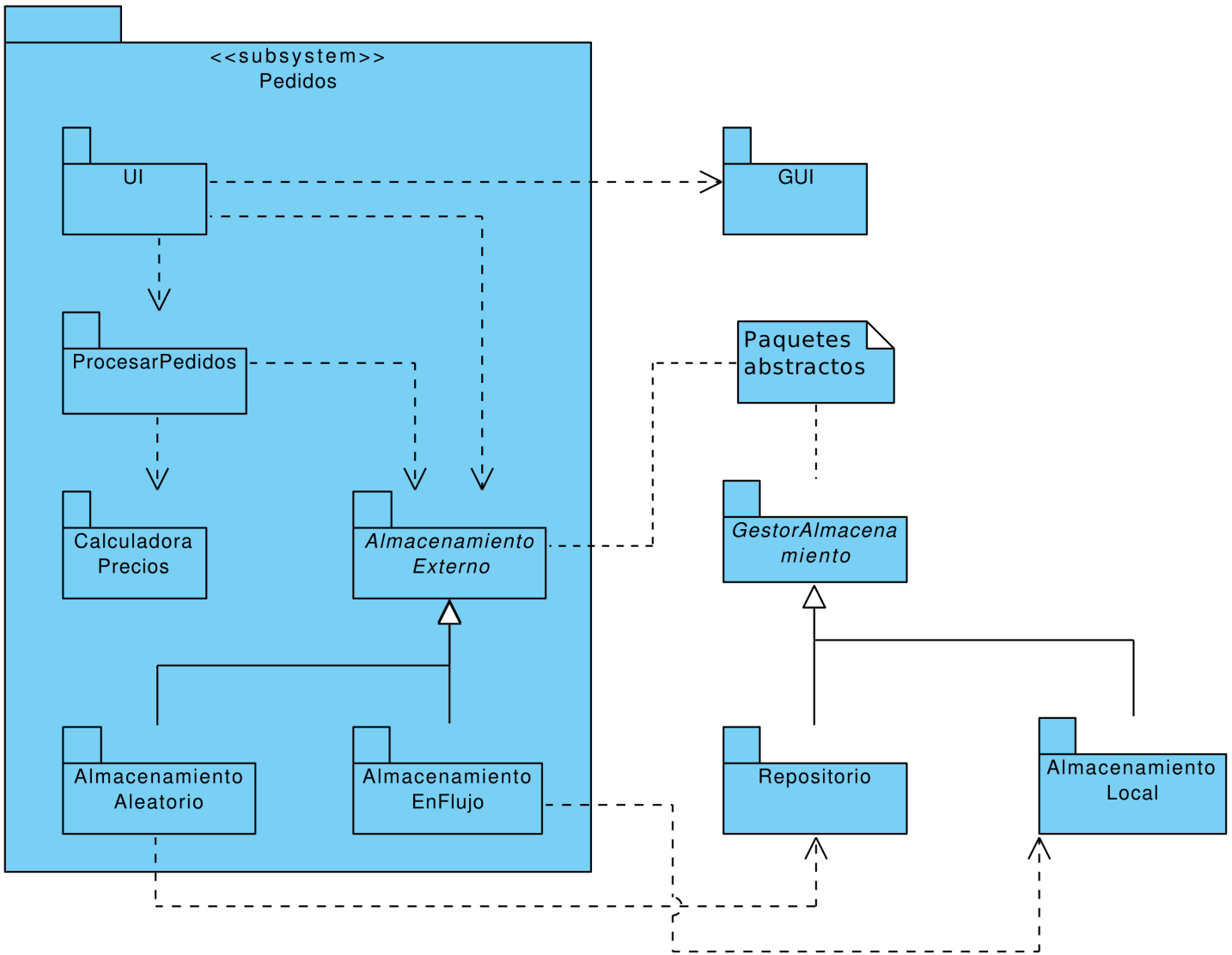
Relaciones Especializadas

- Relación de generalizacion/herencia
  - Un paquete hereda y extiende las funcionalidades de otro

- Relacion «merge»
  - Indica que los elementos que están en un paquete deben incluirse en la otra



Ejemplo Diagrama de Paquetes



Diseño de una Universidad

- Una universidad esta compuesta de múltiples facultades, que a su vez están compuestas por múltiples institutos. Cada facultad e instituto tiene un nombre y los institutos tienen una dirección
- Cada facultad esta dirigida por un decano, que es un empleado de la universidad
- Tenemos conocimiento del numero total de empleados. Los empleados tienen un numero de la seguridad social, un nombre y una dirección de e-mail. Hay una diferenciación entre administrativos e investigadores
- Cada investigador tiene asignado al menos un instituto. El área de investigación de cada uno es conocido. Cada investigador esta implicado en un proyecto un numero de horas. De cada proyecto se sabe el nombre, fecha de inicio y fin
- Algunos investigador dan clases, les llamamos profesores
- Cada clase tiene un identificador único, un nombre y un numero de horas semanales

Pasos a Seguir

1. Identificar los elementos que crearan las calases
   - La universidad no es una clase separada, es el sistema que estamos modelando
2. Identificar los atributos
   - A cada clase encontrada en el paso anterior se le asignan los atributos requeridos
   - Se les puede dar algún tipo y una visibilidad aun que no están especificadas
3. Identificar relaciones entre las clases
   - *"Hay una diferenciación entre administrativos e investigadores"* nos puede dar pistas de relaciones de generalización
   - Identificar asociaciones, *"un profesor da clases"*
   - Relaciones de agregación como *"Cada investigador tiene asignado al menos un instituto"*
   - Relaciones de composición entre las facultades y los institutos