I102 - Paradigmas de Programación

UML — DIAGRAMA DE CASOS DE USO Y DIAGRAMAS DE CLASES

8

Prof. Mariano Scaramal

Ingeniería en Inteligencia Artificial

Que conceptos de C++?

- Conceptos
  - Introducción a UML.
  - Diagrama de Casos de Uso
    - Casos de Uso, Actores y sus Relaciones
    - Ejemplos
  - Diagrama de Clases
    - Notación
    - Asociaciones de Clases, Agregación y Generalización
    - Clases abstractas e Interfaces
    - Clasificación

UML – Unified Modeling Language

UML es un lenguaje de modelado visual estandarizado para visualizar el diseño, el comportamiento y la estructura de un sistema.

Este lenguaje tuvo su versión 1.0 en 1997 y unificó la gran diversidad de este tipo de lenguajes que se originaron en la década del 80.

UML no está atado a ningún lenguaje o plataforma en particular. Tampoco indica un proceso de desarrollo de software.

UML es un lenguaje flexible con conceptos genéricos que permiten una amplia gamas de aplicaciones.

Para evitar restringir el uso del UML, el standard es intencionalmente vago. Esto hace que se arriben a distintas implementaciones según la herramienta

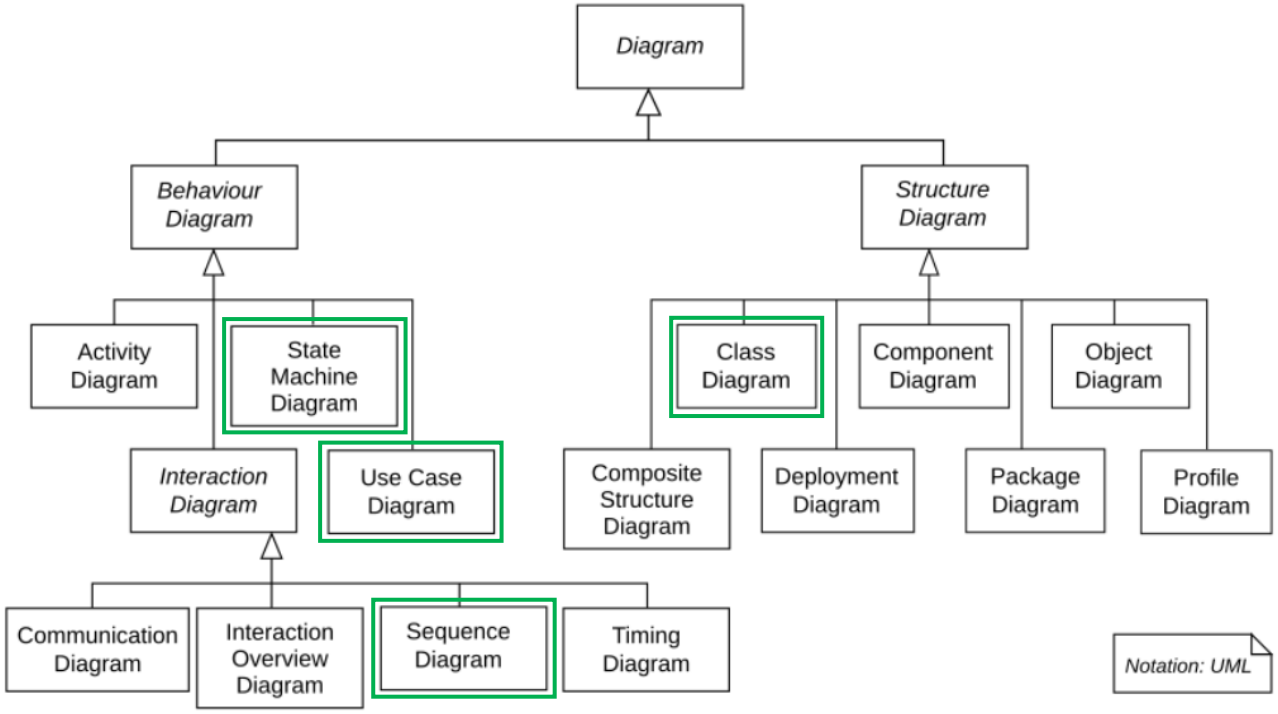
De en.wikipedia.org/wiki/Unified_Modeling_Language

Diagramas en UML

UML se divide en 14 diagramas.

Se observan dos grandes categorías de diagramas: Estructura (structure) y Comportamiento (behavior).

Los diagramas se dividen para poder describir el sistema en sus distintas partes.



De en.wikipedia.org/wiki/Unified_Modeling_Language

Otro elemento utilizado es el bloque de nota (notation) que permite aclarar conceptos como el uso, limitaciones o la intención del diagrama. Este bloque puede ser utilizado con el resto de los bloques.

El Diagrama de Casos de Uso

- Permite describir posibles escenarios de uso del sistema, "*use case*". También se utiliza para indicar que usuario puede utilizar una funcionalidad del programa.
- No indica como se implementa el caso de uso, sólo lo describe.
- Este diagrama responde las siguientes preguntas:
  - Cuál es el caso de uso?
  - Quién interactúa con el sistema?
  - Qué pueden hacer los actores en el sistema?
- Este diagrama expresa que es lo que el sistema logra.
- Si bien se trata de un diagrama simple, un error en esta etapa conlleva grandes problemas: incremento de costos en el desarrollo o mantenimiento, insatisfacción del cliente o usuarios, etc.

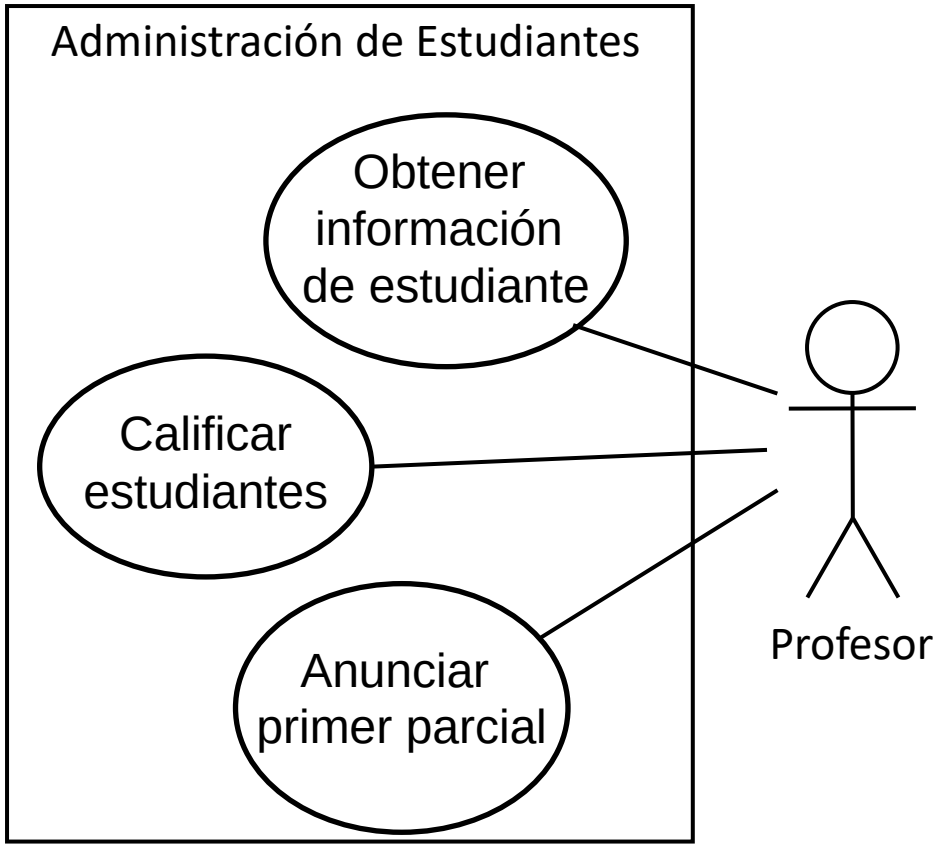
Casos de Usos

Los casos de uso describen la funcionalidad del sistema y suelen ser disparados por invocación de un actor o el inicio de un evento en particular. Por ejemplo: anunciar primer parcial.

Los casos de uso son originados de los requerimientos de los clientes y del análisis de los problemas a resolver mediante el sistema.

Se suelen describir con elipses el contenido del caso de uso.

El conjunto de casos de prueba que describe la funcionalidad que se provee generalmente se agrupan en rectángulos. Estos simbolizan los límites de lo que se quiere describir.



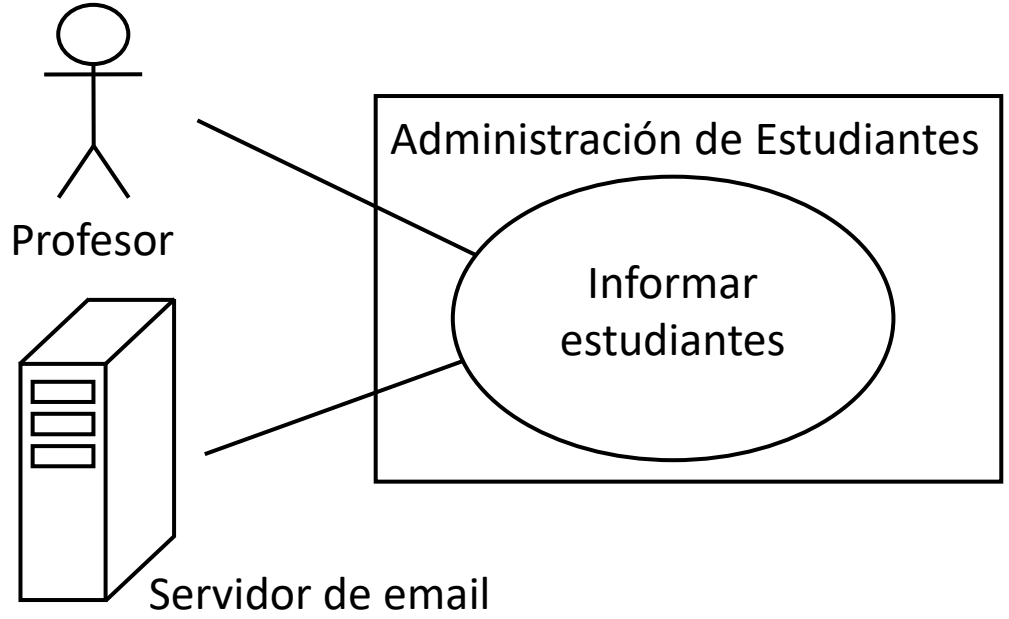
Actores

Los actores son los usuarios con roles específicos que interactúan con el sistema en el caso de uso que tengan asociado.

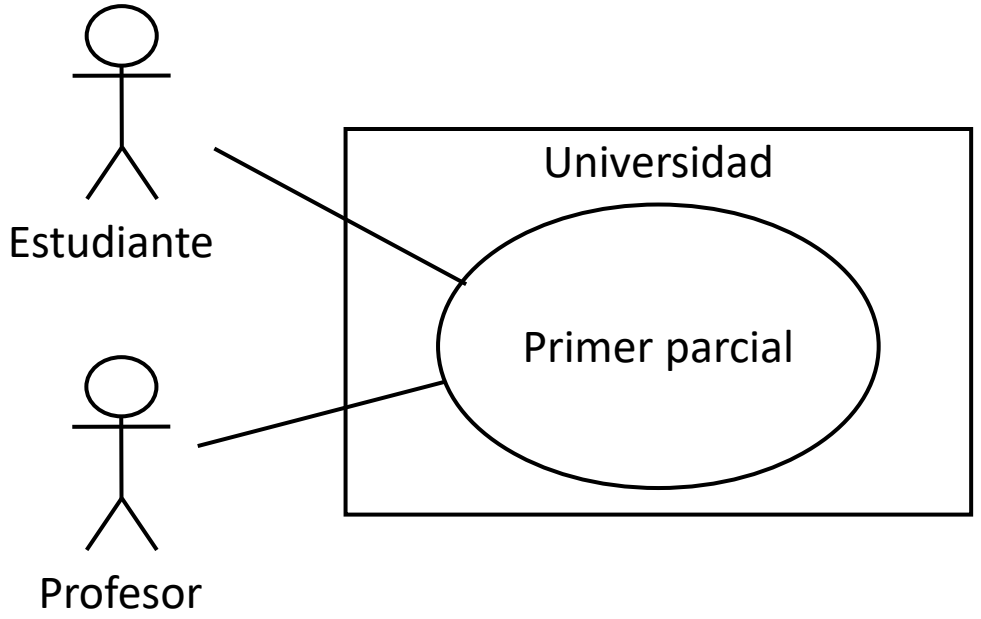
Los actores pueden ser:

- Humanos (figura de palitos) o no humanos, como ser un servidor de mails o una impresora.
- Activos, que el actor inicia la ejecución, o pasivos, que el actor es utilizado por el sistema (provee funcionalidad al sistema).
- Primario, que el actor toma un beneficio de la ejecución del caso de prueba, o secundario, quien no recibe un beneficio. Los usuarios secundarios no tienen que ser necesariamente pasivos.

Actores (cont'd)



El profesor es un actor primario mientras que servidor de email es secundario.



El profesor y el estudiante son ambos actores activos, pero el primario es el estudiante y el secundario el profesor.

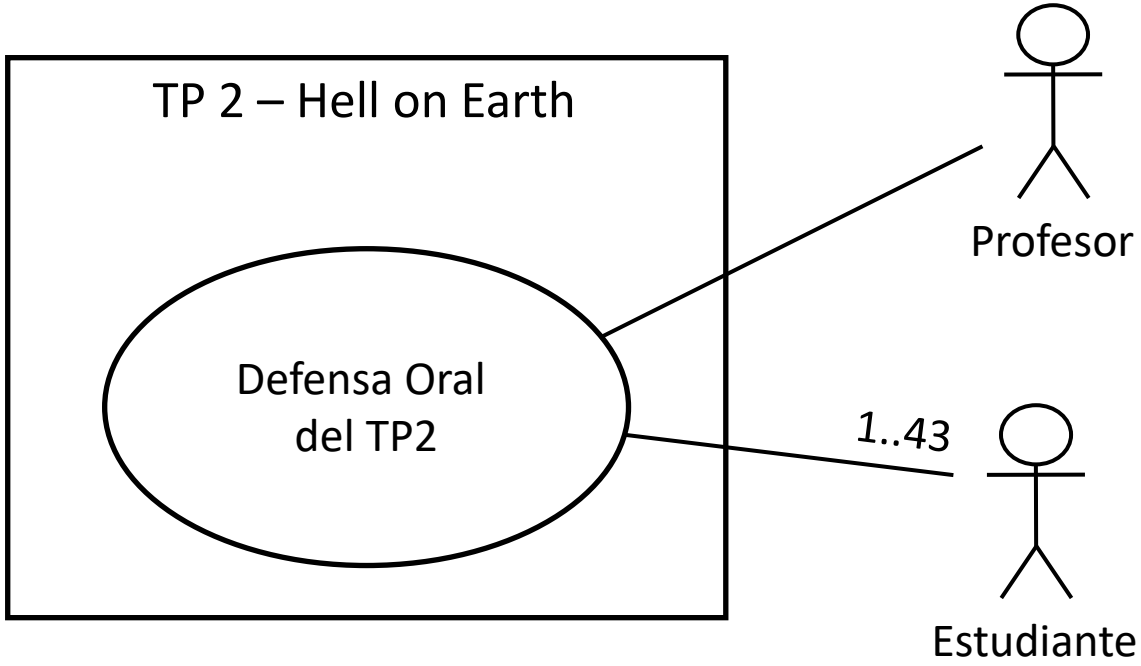
Es importante notar que el actor esta siempre fuera del sistema. En el caso de la izquierda, el servidor de email puede ser eliminado si este forma parte del sistema.

Relaciones entre Actores - Asociación

Una asociación es expresada como una línea recta entre un actor y el sistema.

Cada actor debe estar asociado, al menos, con un caso de uso y viceversa, sino existirán actores y/o casos de uso que no interactúan entre ellos.

La asociación siempre es entre un actor y un caso de uso. Para multiplicidades mayores a uno, se indican en la asociación del lado del actor.



- $\min..\max$: intervalo desde $\min$ a $\max$
- N: cantidad fija
- $\min..*$: intervalo desde $\min$ a infinito
- * = 0.. *: cualquier cantidad
- Sin indicación: se asume 1 por default

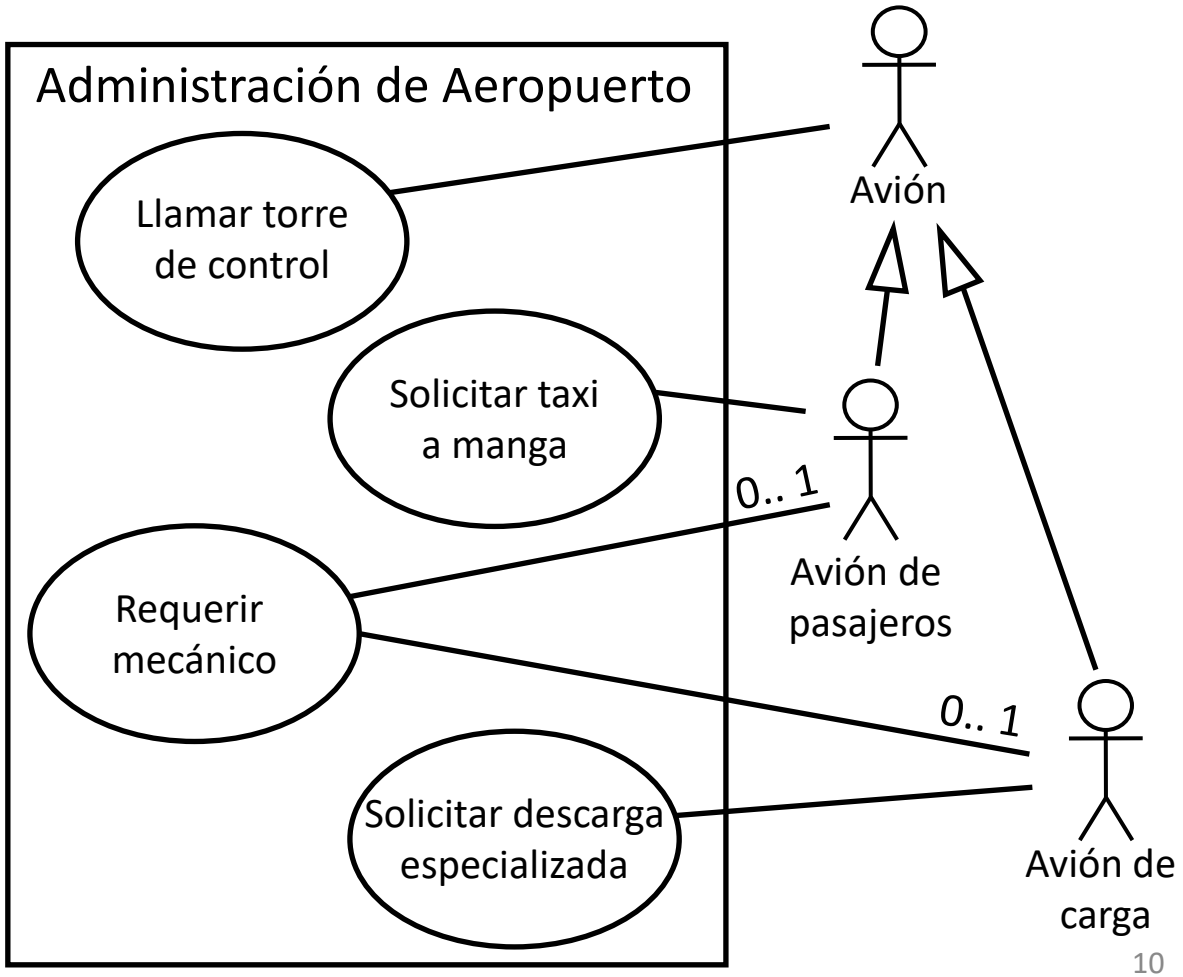
Relaciones entre Actores - Herencia

Los actores pueden estar relacionados mediante herencia como en el caso de objetos. Se representa mediante una línea entre los actores con un triángulo apuntando al superactor.

Tanto un avión de pasajeros como uno de carga pueden comunicarse con la torre de control por herencia.

El avión de pasajeros es el único que puede solicitar taxi a manga, y el de carga es el único que puede solicitar descarga especializada.

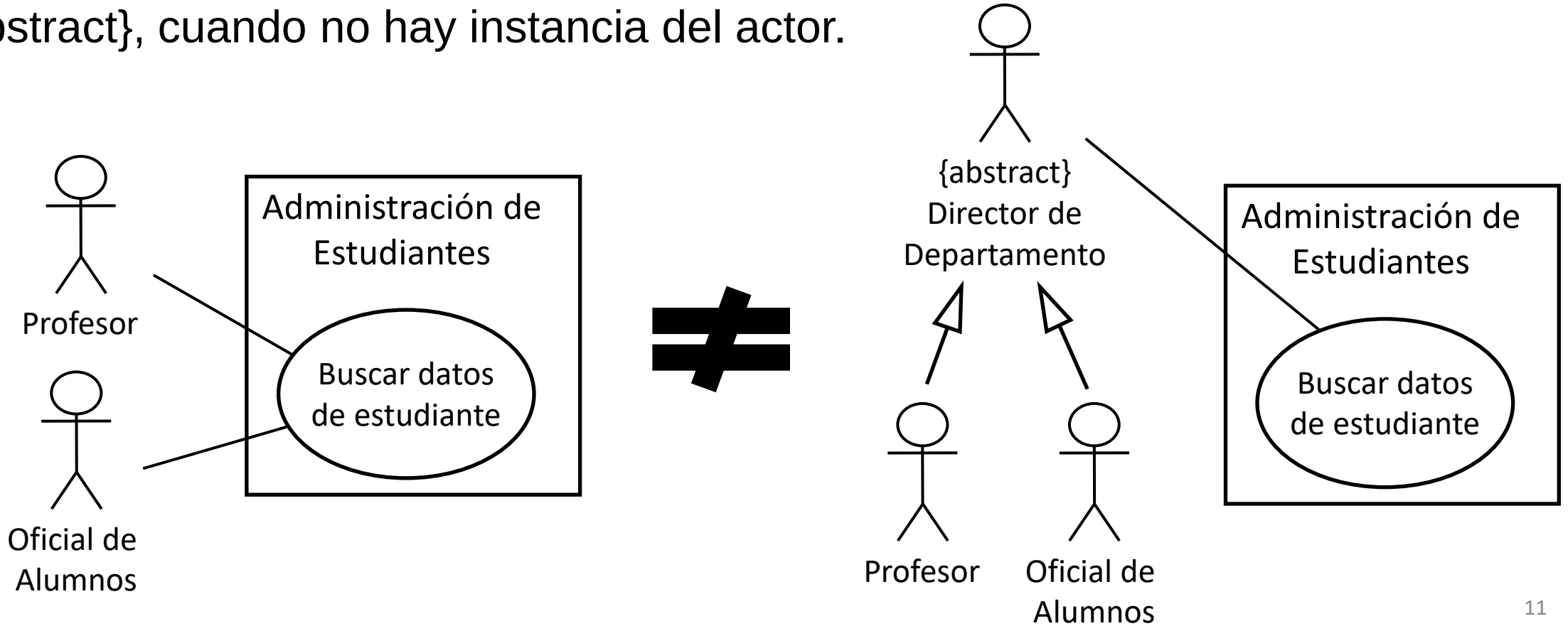
Ambos tipos de aviones pueden pedir un mecánico. Se pueden pedir dos al mismo tiempo, pero no para el mismo tipo de avión.



Relaciones entre Actores – Herencia (cont'd)

Es importante diferenciar cuando dos actores deben participar en un caso de uso, como es el caso de la izquierda. Y cuando dos actores heredan la asociación y cada uno de ellos puede participar en forma individual, caso de la derecha.

{abstract}, cuando no hay instancia del actor.



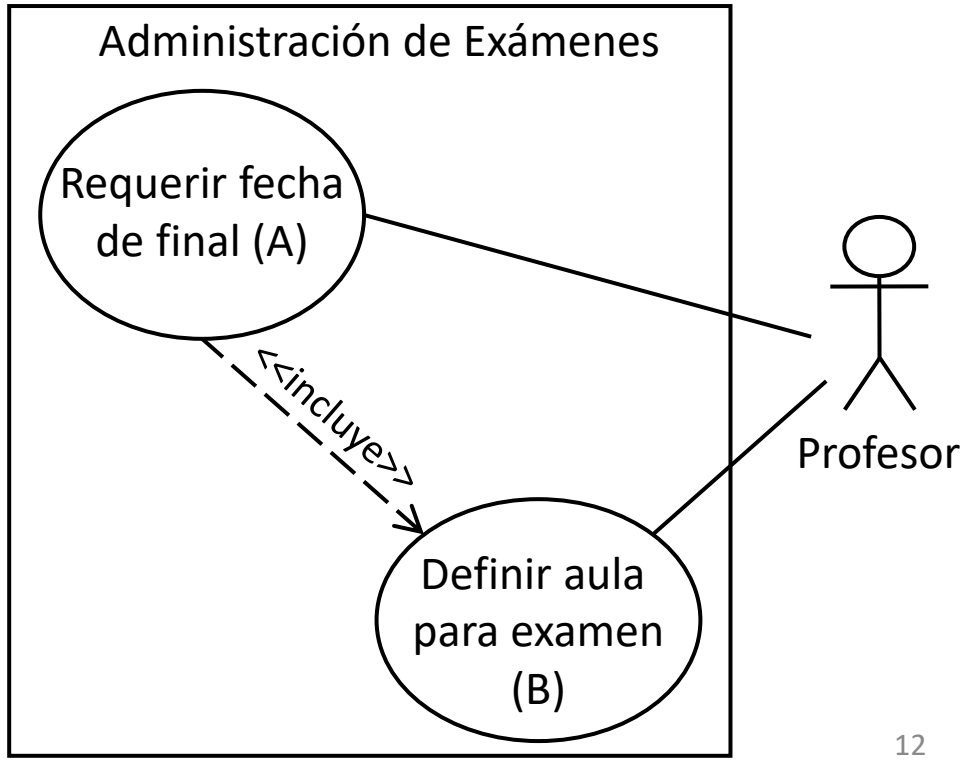
Relaciones entre Casos de Uso - Incluido

El uso de *include* puede verse como el llamar a un procedimiento en un lenguaje.

Si el caso de uso A incluye el B, entonces se dice que A es el caso de uso base y B es el caso de uso incluido. Se representa mediante una flecha discontinua desde A hasta B y la leyenda <<*include*>> *o* <<*incluye*>>.

En el ejemplo, el profesor puede ejecutar:

- Requerir fecha de final: al mismo tiempo, este caso de uso ejecutará "*Definir aula para examen*" debido a que este está incluido.
- Definir aula de examen: aquí, sólo se ejecutará este caso de uso. Por ejemplo, esto se puede hacer si el aula inicialmente asignada no es satisfactoria.



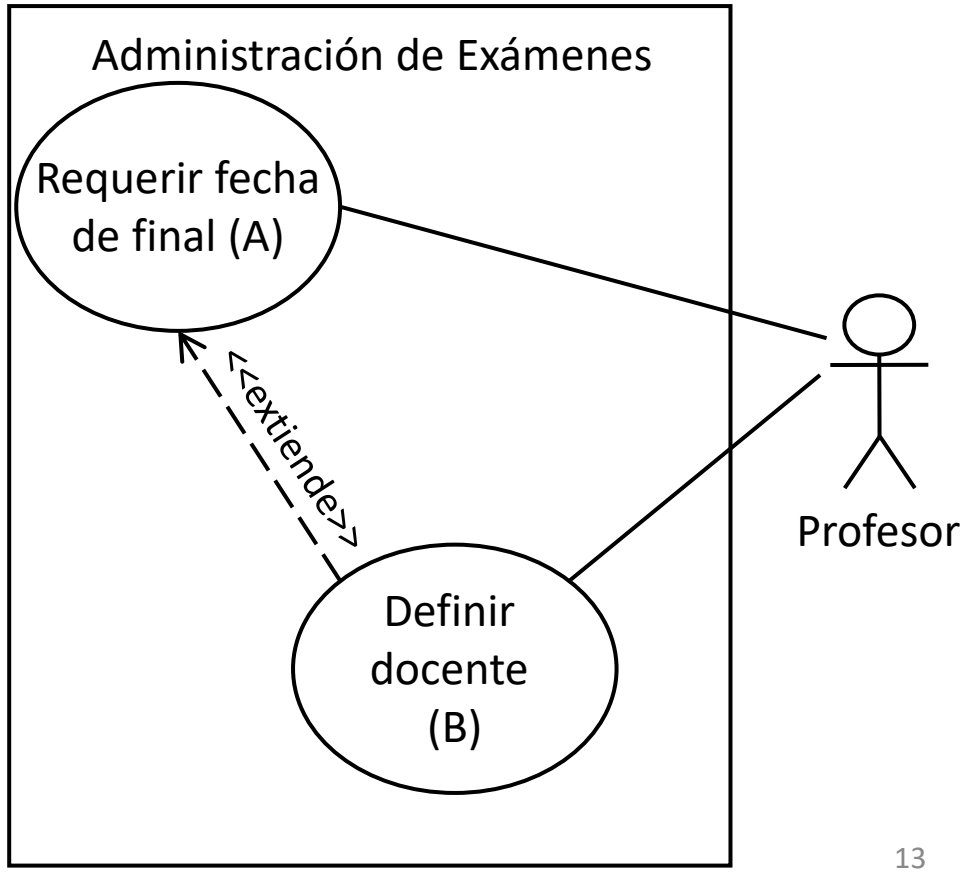
Relaciones entre Casos de Uso - Extensión

Aquí, A es llamado el caso de uso base y B es el caso de uso extendido. La forma de interpretarlo es con una flecha discontinua desde el caso de uso extendido al caso de uso base (al revés de del caso de inclusión).

En una relación de extensión como la de la figura, el caso de uso extendido puede ser ejecutado, pero no es mandatorio.

En el ejemplo, se entiende que el profesor requiere una fecha de final, pero no es necesario definir al profesor dado que se asume que alguien tomará el final.

No obstante, el profesor puede ponerse a él o a algún otro docente que lo reemplace.

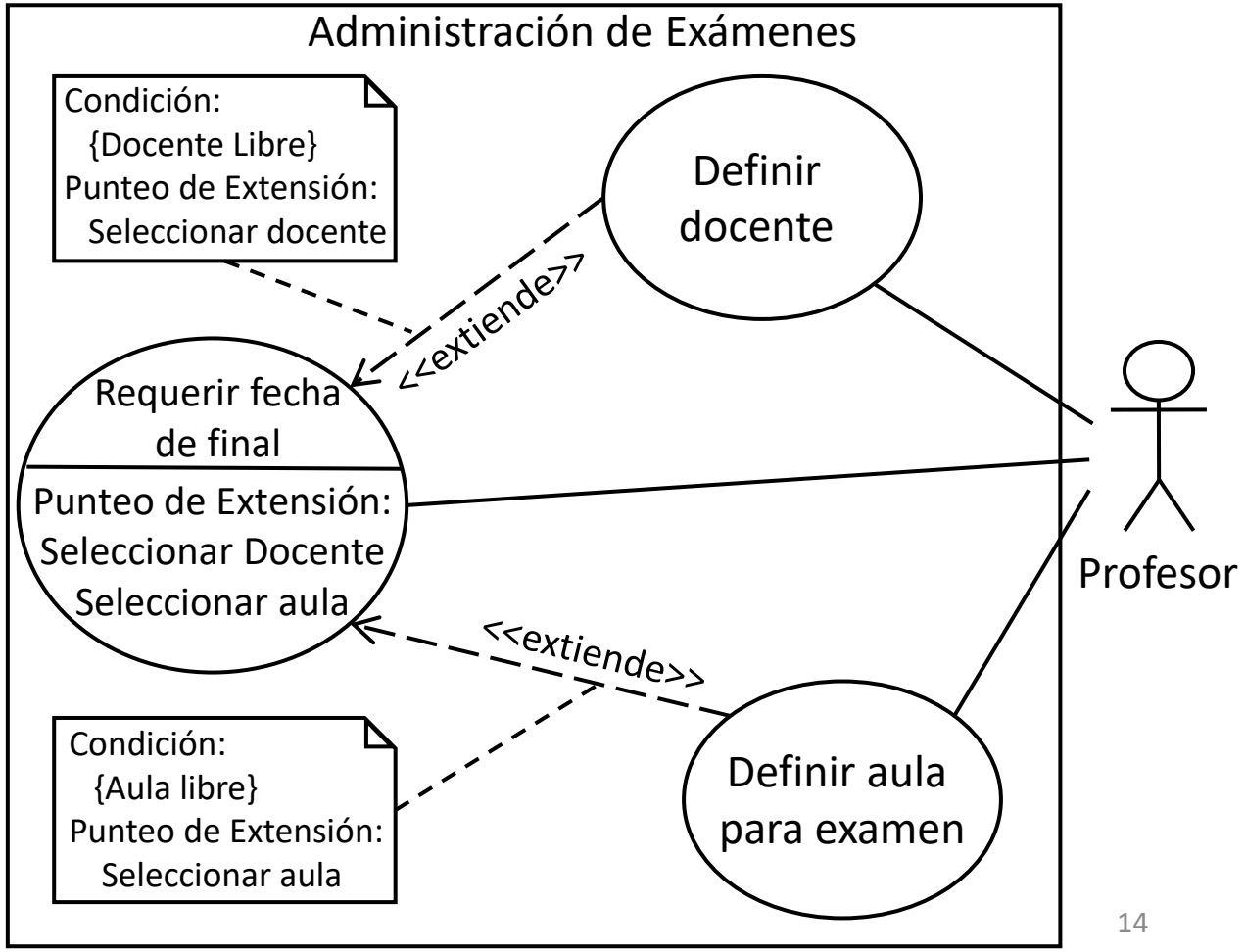


Condición y Punto de Extensión

Las preguntas que surgen para el uso de las extensiones es bajo que condición y en que punto se deben insertar los comportamientos de las extensiones.

La condición y el punto en el cual se utiliza el caso de uso extendido son especificados mediante una nota.

Al mismo tiempo, los puntos de extensión son especificados dentro del caso de uso (Requerir fecha de final).



Generalización entre Casos de Uso

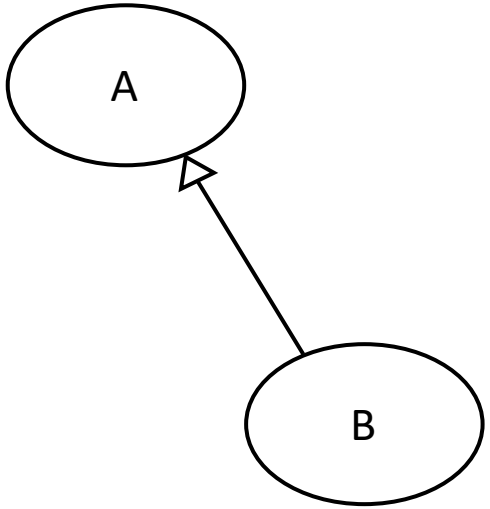
Dado que pueden existir propiedades y comportamientos comunes a diferentes casos de uso, la relación de generalización también existe entre estos.

Si el caso de uso A generaliza el caso de uso B (expresado como en la figura), B hereda el comportamiento de A, donde B puede extenderlo o sobrescribirlo. Al mismo tiempo, B también hereda todas las relaciones que tenga A.

Así, B adopta la funcionalidad de A pero podrá seleccionar que parte de A ejecutar o modificar.

Si un caso de uso es definido como {abstract}, el mismo no podrá ser ejecutado directamente.

Sólo los casos de usos que heredan de un caso de uso abstracto pueden ser ejecutados (en forma similar a lo que ocurre en OOP).



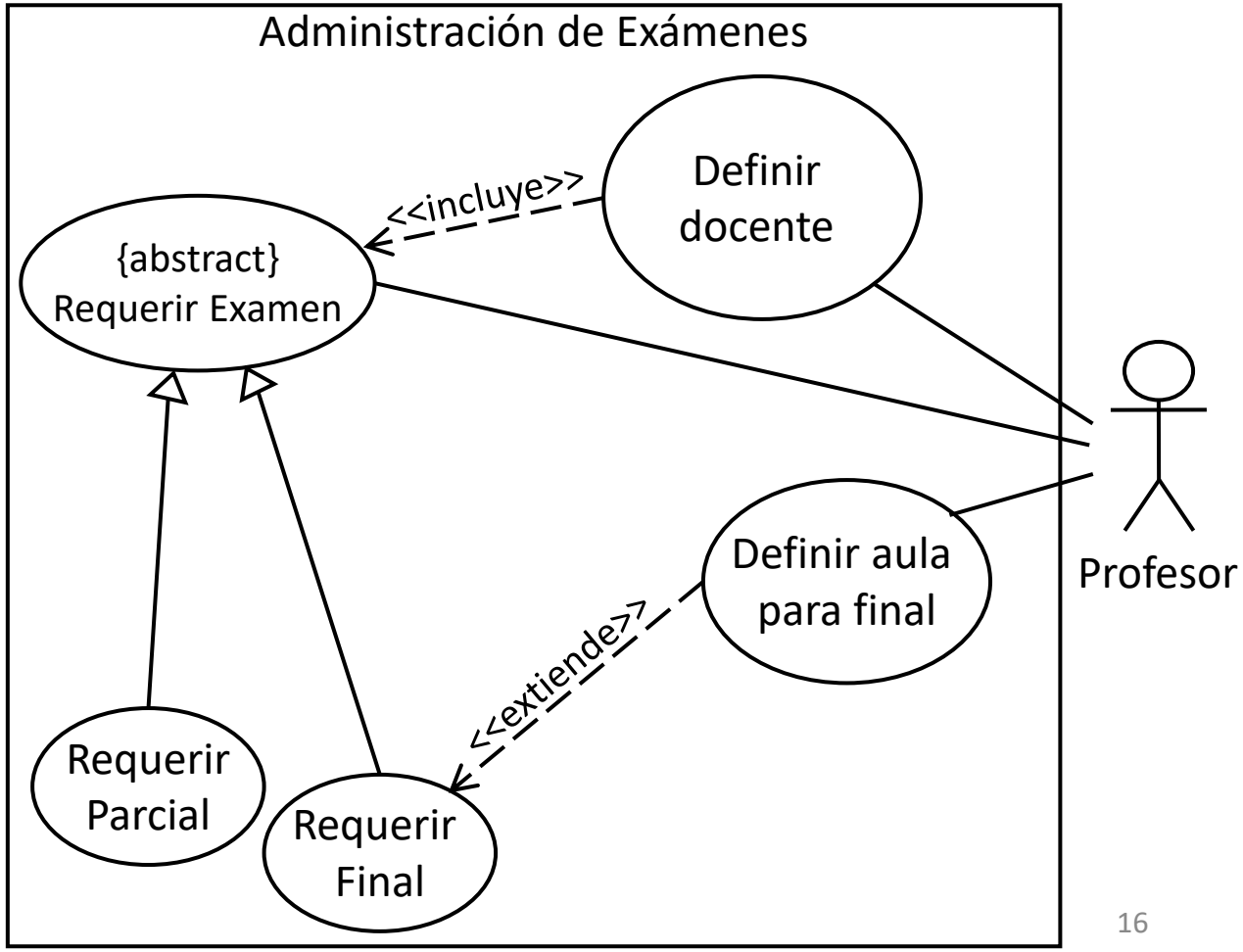
Generalizaciones entre Casos de Uso - Ejemplo

Aquí, el caso de uso abstracto “Requerir Examen” pasa sus propiedades y comportamientos a “Requerir Parcial” y “Requerir Final”.

Como resultado de la inclusión entre “Requerir Examen” y “Definir docente”, ambos casos de uso deberán ejecutar “Definir docente”.

Cuando se requiere un final, se puede potencialmente definir un aula para el examen.

Los casos de uso también heredan la relación con el actor Profesor. Así, todos los casos de uso están asociados a un actor, como requiere el uso correcto de UML.



Ejemplos de Relaciones

En este ejemplo, se observan las siguientes relaciones:

1. El caso de uso B hereda de A. Así, los casos de uso A y B son ejecutados por el actor J. Los actores K y L heredan de J, por lo que A y B también podrán ser ejecutados por estos actores (K y L). Esto es A o B, puede elegir cual ejecutar.

2. El caso de uso C hereda de G. Por esta herencia entre casos de uso, un actor K estará involucrado en la ejecución del caso C. Debido a la herencia entre actores, un actor K y un actor J, L u otro actor K (sería un segundo objeto de K) deben ejecutar el caso de uso C. Si son dos K, se puede usar sólo uno.

3. El caso de uso I extiende G. Como el caso de uso C hereda de G y como I extiende de G, esta relación se pasa a C (I extiende de C). Si G e I fueran una relación incluir, también se pasaría a C.

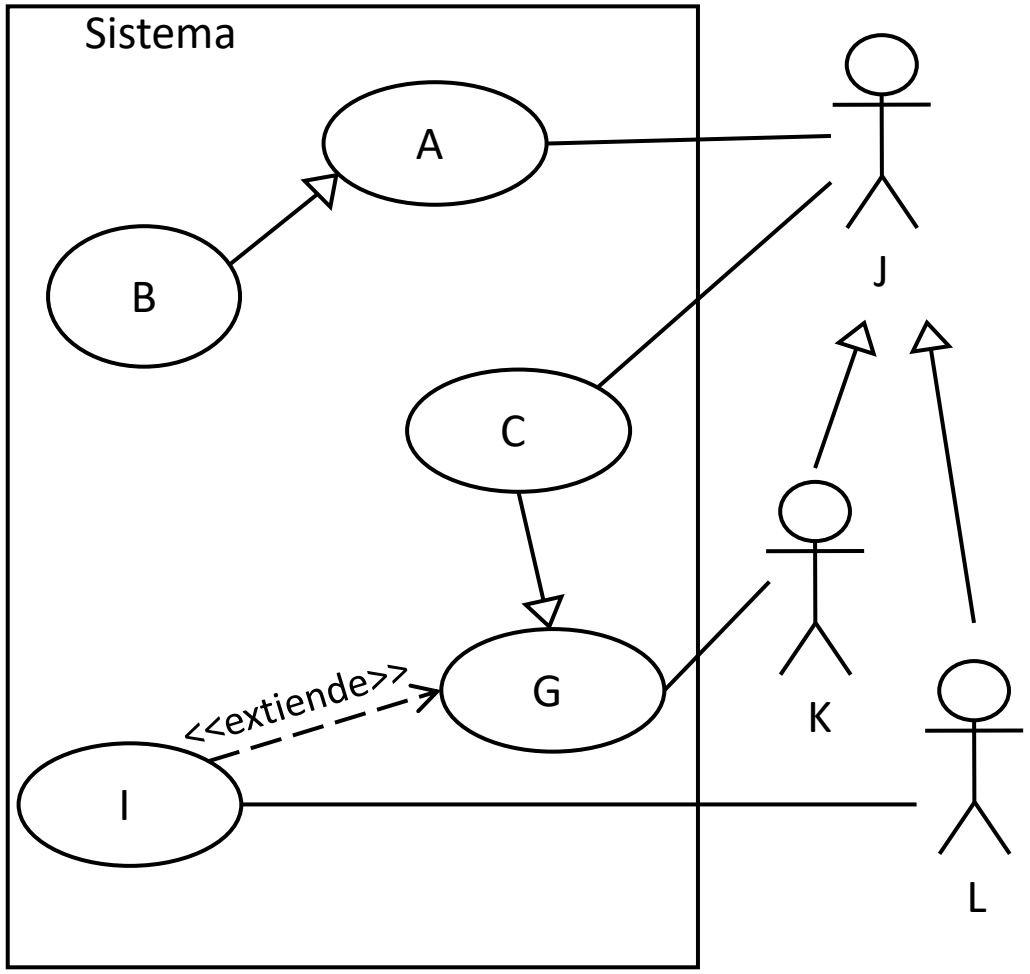


Diagrama de Clase y Objetos

Es importante discernir entre diagrama de clase y de objeto:

- Diagrama de clase: es utilizado para modelar la estructura estática del sistema, describiendo los elementos del sistema y las relaciones entre ellos.
- Diagrama de objeto: describe los objetos que aparecen en el sistema en determinado momento.

Las clases pueden poseer las descripciones básicas de las características estructurales (atributos) y comportamientos (métodos). Y los objetos presentan valores en dichos atributos para un determinado momento.

No obstante, el nivel de detalles se limita a lo relevante en el momento. Así, se abstrae la realidad para hacer el modelo menos complejo para su interpretación.

Recuerde que los objetos son instancias de las clases, poseen valores para un determinado momento.

Diagrama de Objetos - Notación

En los diagramas de objeto, las relaciones se establecen mediante links o enlaces, líneas continuas. Su representación es mediante rectángulos con la información de la clase. Es obligatorio tener: objectName:Class, sólo el nombre del objeto o la clase (objectName debe ser único). Centrado y subrayado.

En el ejemplo, el diagrama muestra al estudiante Paul Lewis inscripto en los cursos IPC y Física.

Se puede ver que estos cursos se dictan en el semestre del verano y también la cantidad de horas de clase.

Finalmente, Física tiene asignada el aula LH1 mientras que IPC aún no tiene un aula asignada.

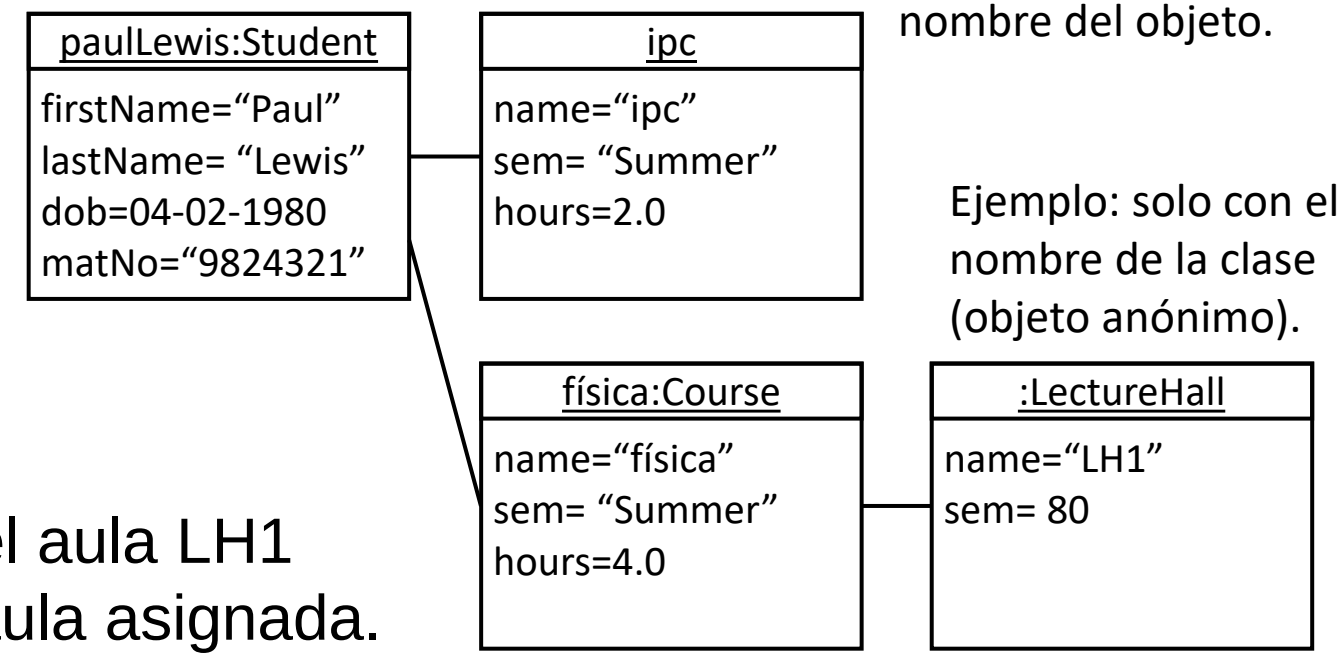
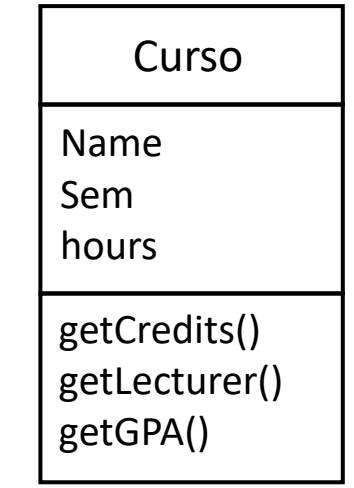


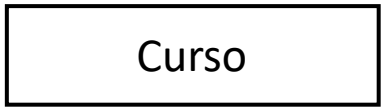
Diagrama de Clase - Notación

Como en los objetos, la representación se realiza con rectángulos. La cantidad de detalles es opcional, como se ve en el ejemplo de los 3 diagramas válidos.

Los atributos y métodos tienen su propia nomenclatura. Para atributos, debe ir el nombre del mismo con el tipo de dato, por ejemplo, name:String. Estos son acompañados por marcadores de visibilidad (ver tabla) que también son utilizados para los métodos.

Visibilidad		
Nombre	Símbolo	Descripción
public	+	Acceso público.
private	-	Acceso sólo desde dentro del objeto.
protected	#	Acceso sólo desde objetos de la misma clase o sus subclases.
package	~	Acceso sólo por objetos que están dentro del mismo package.





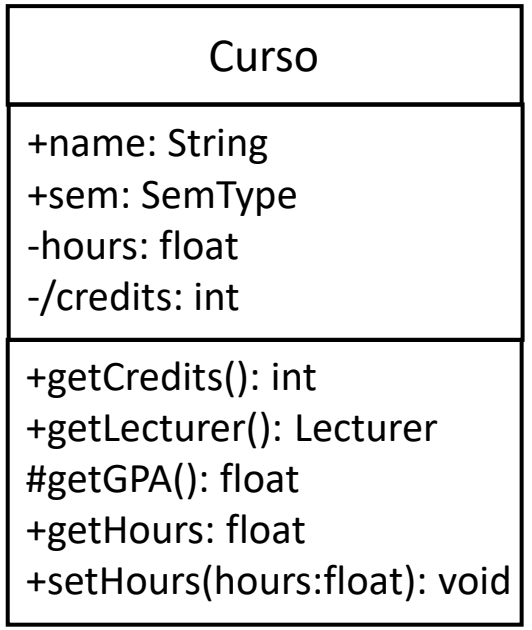


Diagrama de Clase – Notación de Atributos

Para especificar un atributo se puede usar el proceso de la imagen inferior.

La multiplicidad, se utiliza para crear arrays. La notación es [min .. max], donde min puede ser cero y max puede no existir si se indica un *, ([0..*]=[*]). Con un número exacto se usa [N].

La barra / indica que el valor de la variable es obtenido en función de otros valores (“dob” se usa para obtener “age”).

Las propiedades permiten también definir tipos de datos abstractos con: {único}, {ordenado}, {orden fijo}, {sin duplicados}, {con duplicados}, etc.

Los atributos estáticos son indicados subrayándolos como en el caso de *personsCounter.*

Curso
firntName: String lastName: String dob: Date address: String [1..*] {unique, ordered} dni: String {readOnly} /age: int password: String = “pw123” personsCounter: int
getName(out fn: String, out In String): void updateLastName(newName: String): boolean getPersonCounter(): int

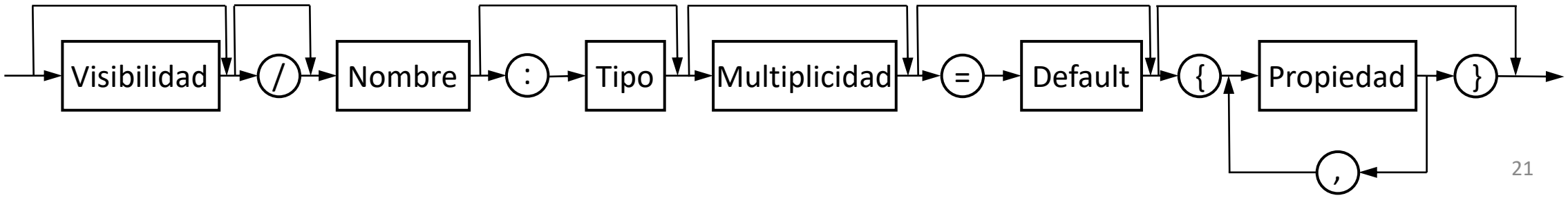


Diagrama de Clase – Notación de Operaciones

Las operaciones (métodos) son indican con un nombre, sus parámetros y el tipo de dato a retornar, como se muestra en el proceso de la imagen inferior.

Se inicia indicando la visibilidad (opcional), luego el nombre (obligatorio). A continuación se definen los parámetros entre paréntesis (los parámetros son opcionales, los paréntesis no), luego el tipo de dato que se retorna y finalmente las propriedades del método.

El caso de los métodos estáticos se indica al subrayar el método, como en *getPersonCounter().*

Estudiante
firntName: String lastName: String dob: Date address: String [1..*] {unique, ordered} dni: String {readOnly} /age: int password: String = “pw123” personsCounter: int
getName(out fn: String, out ln String): void updateLastName(newName: String): boolean getPersonCounter(): int

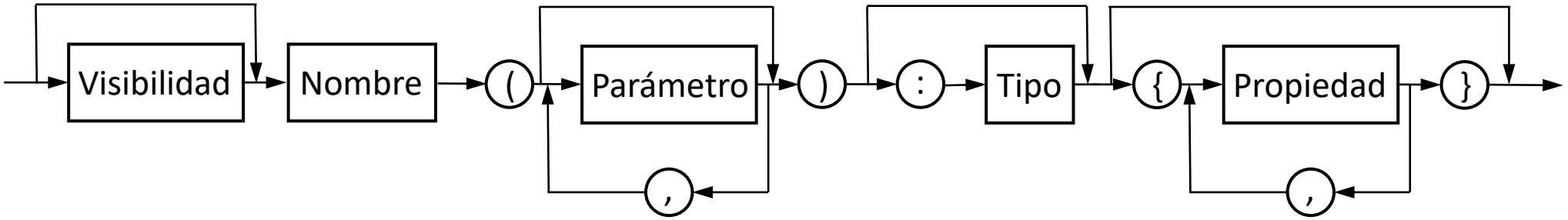


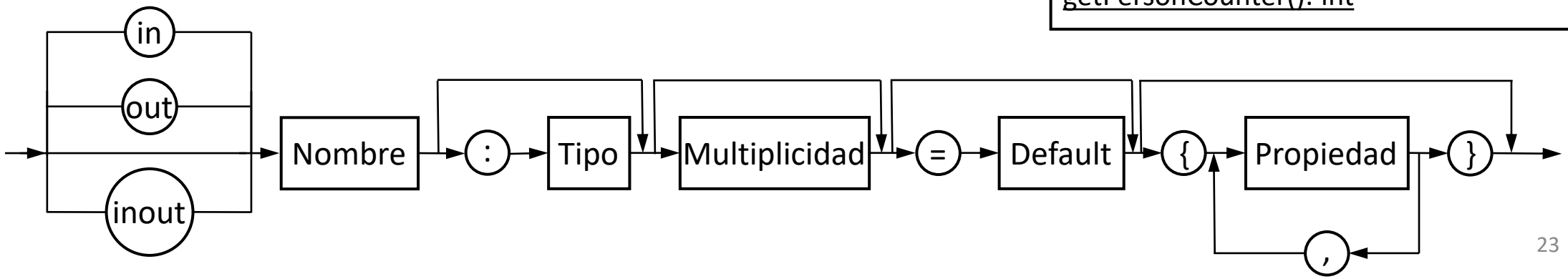
Diagrama de Clase – Notación de Parámetros

La notación de los parámetros se observa en el proceso de la figura inferior.

La única diferencia con los atributos es la dirección del parámetro, la cual puede ser *in*, *out* o *inout*.

Con un parámetro con *in*, desde la función, se espera recibir un valor. En el caso de usar *out*, indica que se tendrá un valor nuevo al finalizar la ejecución de la operación. Cuando se usa *inout*, se tiene una combinación de ambos.
Si no se indica nada, se asume que es *in*.

Curso
firntName: String lastName: String dob: Date address: String [1..*] {unique, ordered} dni: String {readOnly} /age: int password: String = “pw123” personsCounter: int
getName(out fn: String, out ln: String): void updateLastName(newName: String): boolean getPersonCounter(): int



Asociación Binaria

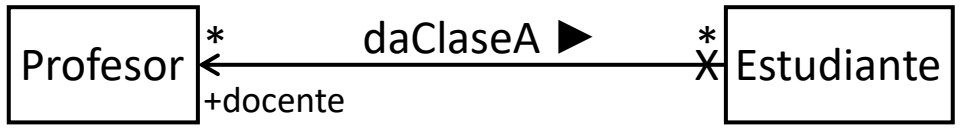
Una asociación entre clases es una relación simple entre dos o más clases en la que ninguna de ellas tiene ownership sobre la otra. Usualmente indica que una clase hace referencia a otra.

Una asociación binaria es una relación entre dos clases. Su interpretación es marcada con el **nombre de la asociación** seguida por una flecha (opcional) con la dirección de lectura.

Navegabilidad es posible cuando el enlace posee al menos una flecha. Esto indica el sentido en que una clase puede acceder a los miembros de la clase asociada. La no navegabilidad se marca con una X o se deja sin marcar (por default se asume que es no navegable). En el ejemplo, un estudiante puede acceder las características visibles de un profesor.

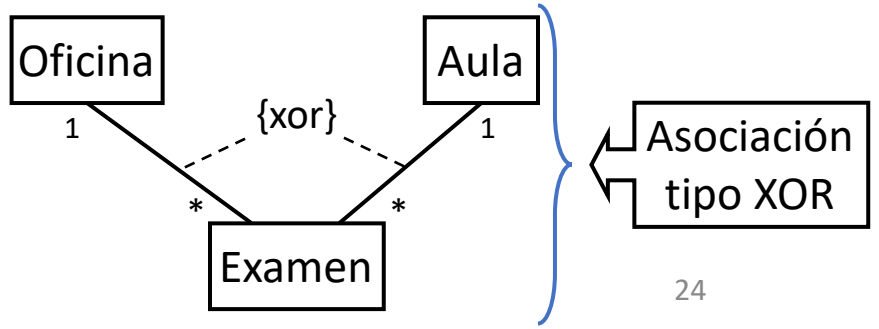
La multiplicidad sigue las reglas flexibles de multiplicidad que se vieron previamente.

La asociación puede llevar el nombre de un rol utilizado. El rol describe la forma en que el objeto está vinculado en la asociación, el rol en la relación descripta.



+docente: indica el rol que tendrá Profesor en Estudiante

```
class Profesor {…}
class Estudiante {
public:
   Profesor docente;
   …
}
```



Asociaciones N-arias

Una asociación N-aria ocurre cuando el vínculo es entre más de dos clases. Se representa con un diamante conectado con todos los miembros de la relación.

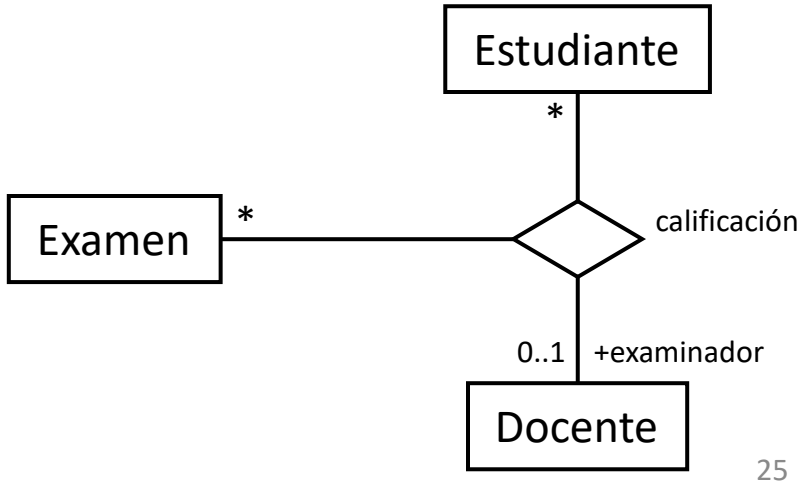
No hay direcciones de navegación, pero si multiplicidades y nombres de roles.

La figura modela la relación calificación con las instancias de las clases Docente, Estudiante y Examen.

Las multiplicidades son definidas como:

- Un estudiante específico toma un examen específico sin o con un docente,
- Un examen específico con un docente específico puede ser tomado por cualquier cantidad de estudiantes,
- Un estudiante específico puede ser calificado por un docente específico en cualquier cantidad de exámenes.

En este modelo no es posible que dos o más docente califiquen un estudiante en el mismo examen.



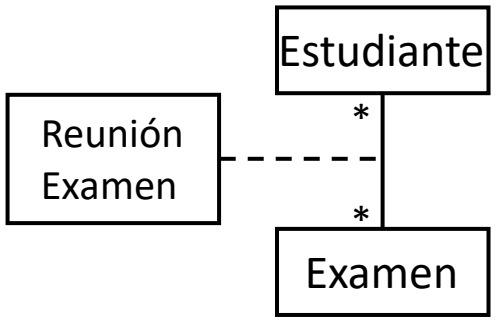
Clases de Asociación

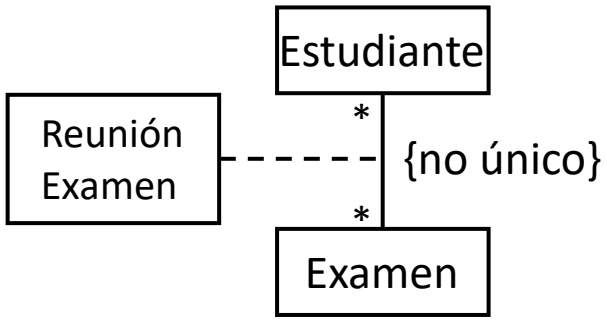
Si se desean asignar atributos u operaciones a la relación entre una o más clases en lugar de a una clase, se puede hacer usando una clase de asociación.

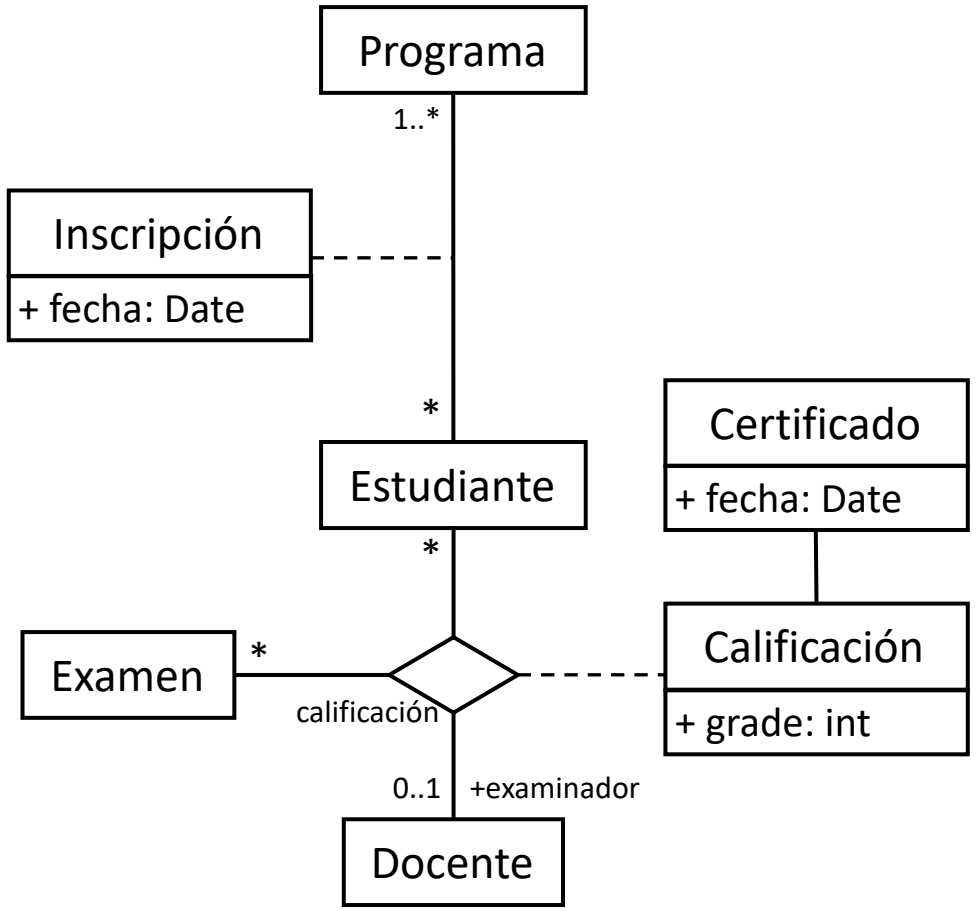
Las clases de asociación también pueden estar enlazadas a otras clases. En un diagrama de clases, la clase asociada debe tener el nombre de la asociación de clase, aunque no es necesario indicarlo.

Si se requieren duplicados para una clase de asociación, uno de los extremos debe ser identificado como {no único}, si no se indica, el valor por defecto es {único}.

Izquierda, sólo hay una reunión para ver la calificación por estudiante, en la derecha, hay más de una.







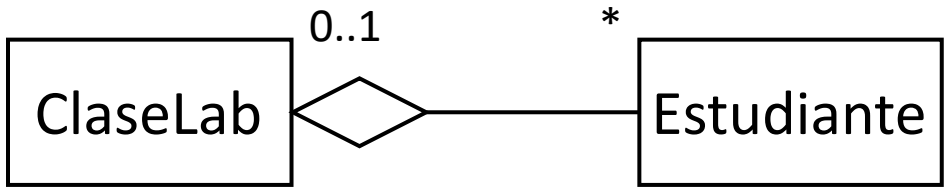
Agregación

Una agregación es un caso especial de asociación que es usado para expresar que instancias de una clase son parte de una instancia de otra clase. Es decir, existe una relación todo-partes, pero estas existen en forma independiente.

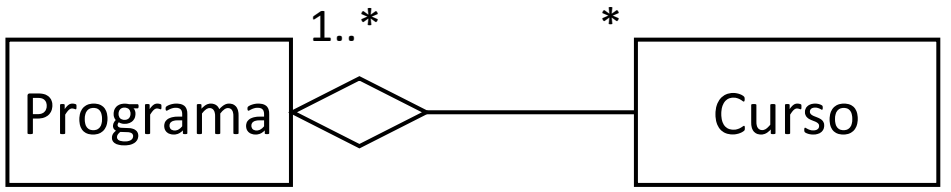
Su representación utiliza un diamante vacío.

La multiplicidad en la parte de agregación puede ser mayor a 1, indicando que un elemento puede ser parte de otros elementos simultáneamente.

El ejemplo 1 muestra que una clase de laboratorio puede tener cualquier cantidad de estudiantes, pero que un estudiante sólo puede ir a una clase. El ejemplo 2, un programa está hecho de cero o más cursos, pero un curso se asigna al menos a un programa.



Ejemplo 1



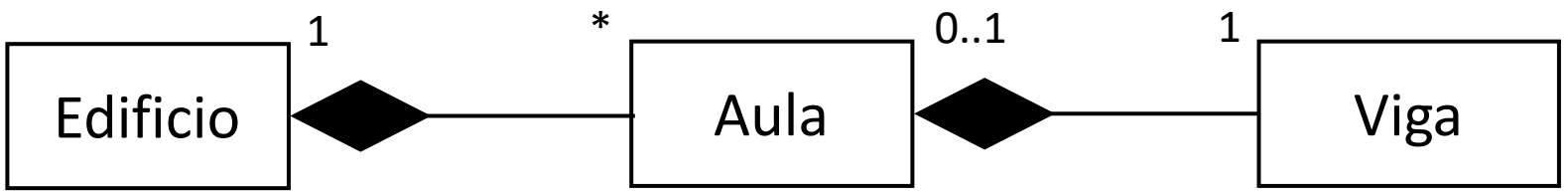
Ejemplo 2

Composición

Una composición expresa que una parte específica, a los sumo puede ser contenida en un objeto específico en un dado tiempo. Esto resulta que el lado de composición tenga una multiplicidad máxima de 1.

En el ejemplo, se observa la dependencia de existencia. Un aula es parte de un edificio, se observa que si el edificio no existe, el aula deja de existir.

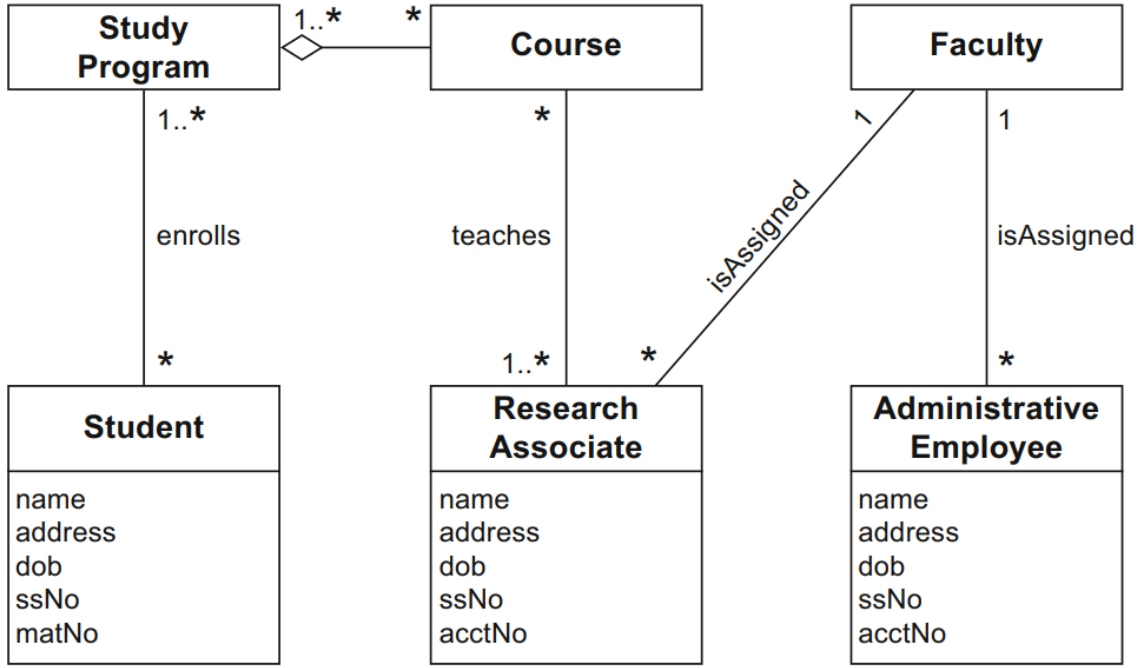
Esto es diferente para la viga. La viga está asociada con el aula por una multiplicidad de 0 o 1. Esto quiere decir que la viga puede existir sin el aula, es decir, si se remueve la viga. No obstante, si el aula deja de existir, la viga también dejará de existir.



Generalización

Generalización se utiliza para derivar clases más específicas a partir de clases existentes y así evitar definiciones redundantes.

La generalización se logra mediante la herencia entre clases. Vale recordar que una subclase poseerá todos los atributos y métodos de la super que no se marcaron como *private.*



Ejemplo de diagrama de clase sin generalización (de *UML @ Classroom*)

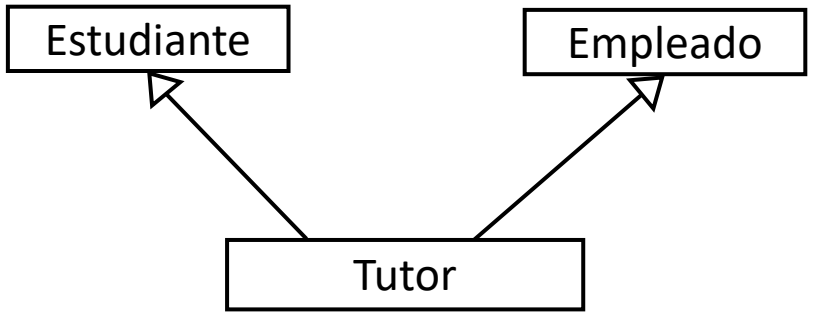
Generalización – Herencia y Polimorfismo

Como se comentó previamente, la cantidad de detalles es opcional.

La generalización se representa con una flecha vacía desde la subclase a la superclase.

Para asegurar que no haya instancias de la Clase Persona, se le agrega la leyenda {abstract}.

UML permite herencia múltiple, es decir que la clase pude tener múltiples superclases.



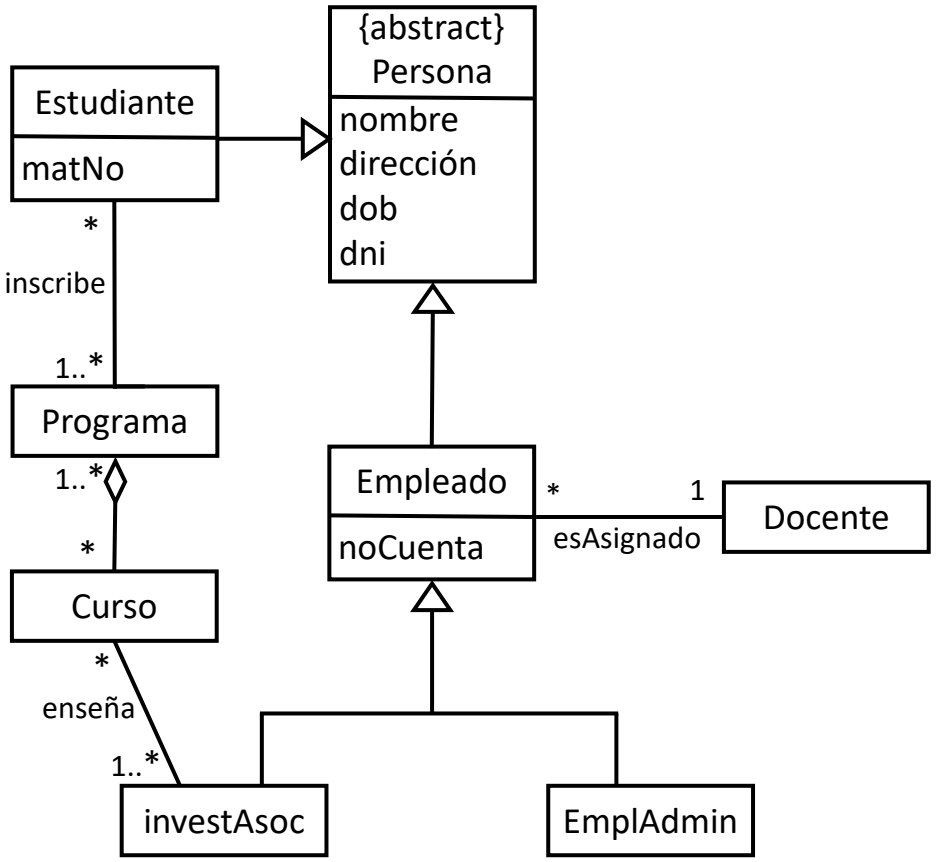


Diagrama de clase con generalización

Clases Abstractas Vs. Interfaces

Las clases abstractas son clases que no se pueden instanciar, sólo sus subclases pueden ser instanciadas. Se las utiliza para agrupar características comunes de sus subclases.

Una operación/método abstracto no ofrece una implementación, de hecho, requiere a sus subclases una implementación concreta.

Se identifican con su nombre en *italic* o con la leyenda {abstract} sobre su nombre.

Las interfaces tampoco tienen una implementación o alguna instancia directa. Una interfaz es un contrato, las clases que lo aceptan, implementan las interfaces al escribir el comportamiento especificado en esta.

Las operaciones/métodos en una interfaz nunca tienen una implementación en la misma, en esta sólo se declaran los métodos.

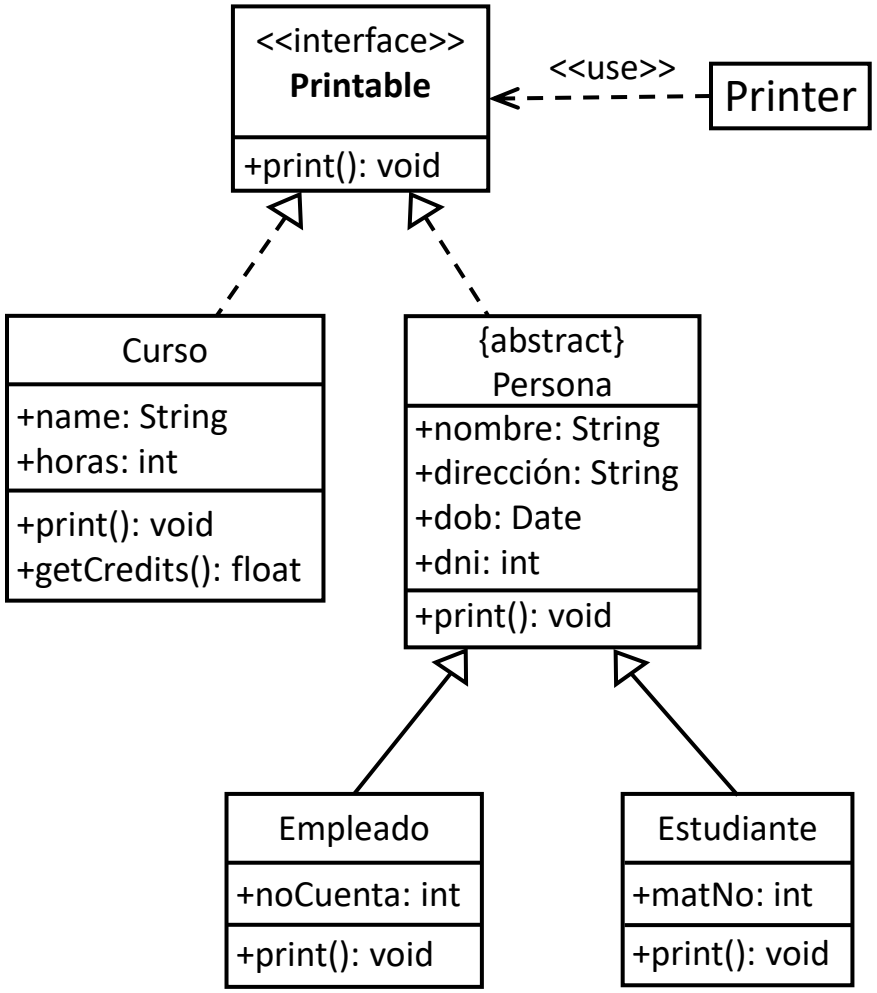
Clases Abstractas Vs. Interfaces (cont'd)

La interfaz se implementa como una clase con la leyenda <<interface>> sobre el nombre de la clase.

Para indicar quien implementa la interfaz se utiliza una línea discontinua con una flecha vacía desde la clase a la interfaz. Esta flecha se utiliza para identificar la relación de dependencia.

Se utiliza una flecha discontinua con la leyenda <<use>> indicando la clase que utiliza la interfaz.

En este ejemplo, la clase *Curso* implementa *print()*, mientras que la clase abstracta *Persona* debe proveer una implementación de *print()*. Pero *Empleado* y *Estudiante* sobreescriben el método *print()* para imprimir el número de cuenta y el número de matrícula, respectivamente.



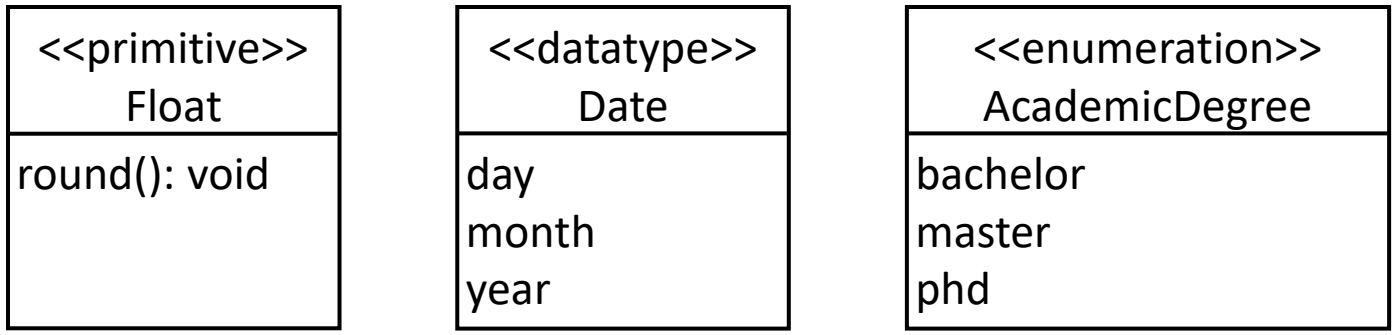
Tipos de Datos en UML

En UML un tipo de dato se visualiza de la misma manera que una clase, pero con la leyenda <<datatype>>. Estos tipos de datos pueden poseer una estructura interna.

Existen dos tipos especiales de TDA, llamados *primitivos* y *enumeraciones*.

Los tipos de datos primitivos, <<primitive>>, no tienen una estructura interna, pero pueden tener operaciones que se realizan con sus valores.

Enumeraciones son tipos de datos definidos en una lista que se identifican con la leyenda <<enumeration>>. Sus valores son denominados *literales*.



Universidad de SanAndrés

Preguntas?