UML

Rodrigo García Carmona
Universidad San Pablo-CEU
Escuela Politécnica Superior

MODELADO DE DATOS CON UML

MODELADO DE DATOS

- La forma en la que representamos los datos para trabajar con ellos.
- Depende en gran medida del uso que se les vaya a dar.
- Ejemplos de modelos:
  - **Relacional:** Para implementar DBMS. Muy eficiente.
  - **XML:** En forma de árbol. Legible por un ser humano.
- Son modelos de **bajo nivel**, implementados pensando en el sistema.
- Como alternativa existen los modelos de **alto nivel**:
  - **E-R:** Modelo Entidad-Relación. Ya estudiado.
  - **UML:** *Unified Modelling Language*.
    - El que estudiaremos ahora, más popular.
- Los lenguajes de alto nivel son de carácter gráfico.
  - Se pueden representar mediante “dibujos”.
  - Posteriormente son traducidos a un modelo de bajo nivel.

CARACTERÍSTICAS DE UML

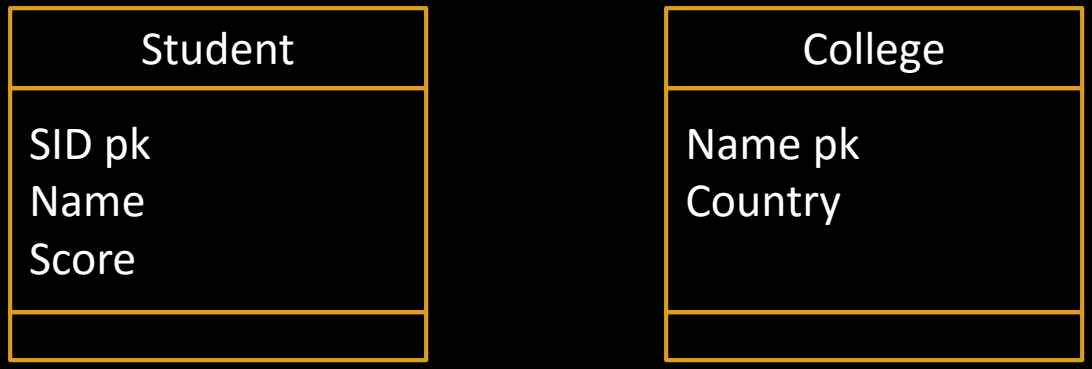
- Pensado para lenguajes orientados a objetos.
- Adecuado para arquitectos software y perfiles de gestión.
- Estándar muy extenso:
  - Abarcamos en esta asignatura la parte de modelado de datos.
- Con múltiples tipos de diagramas.
  - **Estructurales:** Análisis y diseño estáticos.
    - De clases, componentes, paquetes, despliegue…
  - **De comportamiento:** Análisis y diseño dinámicos.
    - De actividad, secuencia, estado, casos de uso…
  - En esta asignatura nos centramos en los **diagramas de clases**.
- **Un diagrama de clases puede traducirse fácilmente a código en un lenguaje de programación orientado a objetos.**
- La forma de entender los datos de la orientación a objetos **es muy distinta** a la del modelo relacional.

MODELADO DE DATOS CON UML

- **7 conceptos fundamentales:**
  - Clases.
  - Asociaciones.
  - Clases de asociación.
  - Composición.
  - Agregación.
  - Herencia (generalización y especialización).
  - Realización.
- Asociación, composición, agregación, generalización, especialización y realización son subtipos de un concepto genérico llamado relación.
- **No confundir "relación" en UML con "relación" en el modelo relacional.**
- Vamos a estudiar cómo usar UML para **modelar datos**.

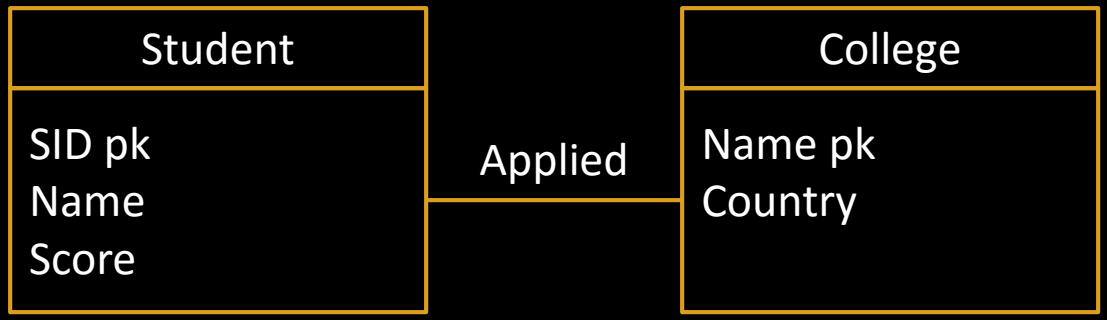
CLASES

- Modelan un tipo de componente.
- Una clase en orientación a objetos. Compuestas de **nombre**, **atributos** y **métodos**.
- Parecidas a entidades en diagramas E-R.
- **Para modelar datos** tenemos en cuenta los siguientes detalles:
  - Añadimos "**pk**" a la clave primaria.
  - Hay quién opina que los datos no deben poseer comportamiento (métodos). Si se sigue esta escuela de pensamiento no debería haber bloque de métodos.



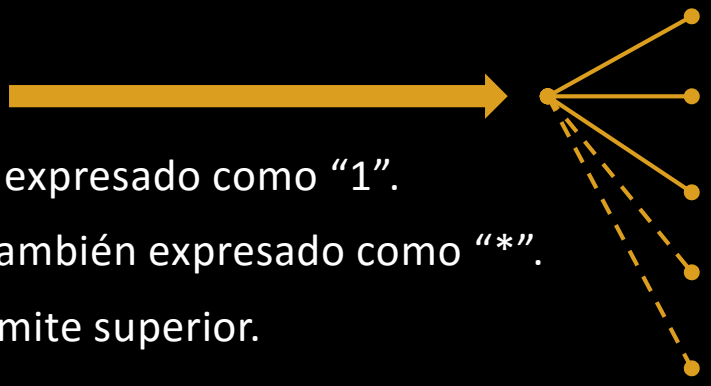
ASOCIACIONES

- Relaciones genéricas entre instancias concretas (objetos) de dos clases.
- Representadas con una línea sólida.
- Opcionalmente pueden tener un nombre que las identifique.
- Parecidas a las relaciones en diagramas E-R.



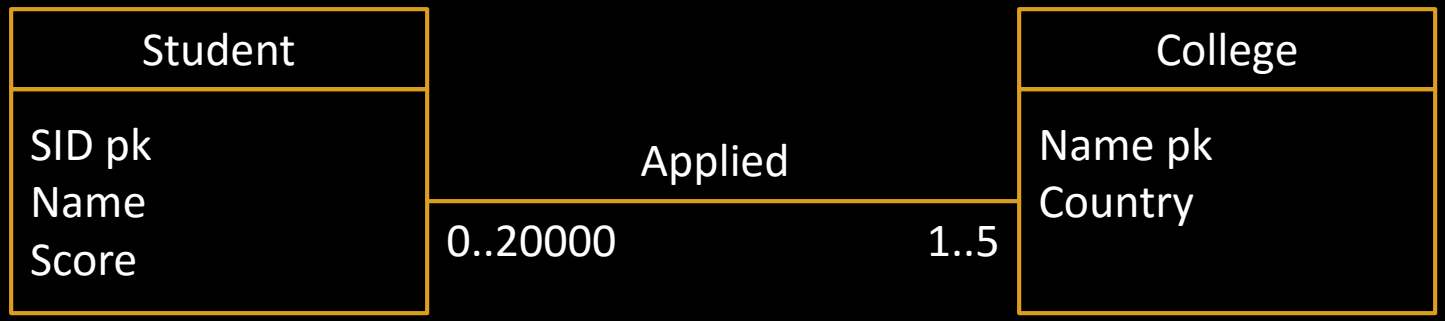
MULTIPLICIDAD DE ASOCIACIONES

- La cantidad de objetos a cada lado de la asociación se indica con un intervalo.
- **Cada lado se lee de forma independiente.**
- La cantidad en un lado indica “cómo ve” a dicho lado el opuesto.
- Se expresa de la forma: *cantidad mínima .. cantidad máxima*.
  - Mínimo y máximo pueden coincidir.
    - En este caso puede eliminarse el “..”
  - El carácter “*” indica cualquier cantidad.
- Ejemplos:
  - 3..5: Entre tres y cinco.
  - 1..1: Sólo uno. También expresado como “1”.
  - 0..*: Sin restricciones. También expresado como “*”.
  - 1..*: Más de uno y sin límite superior.



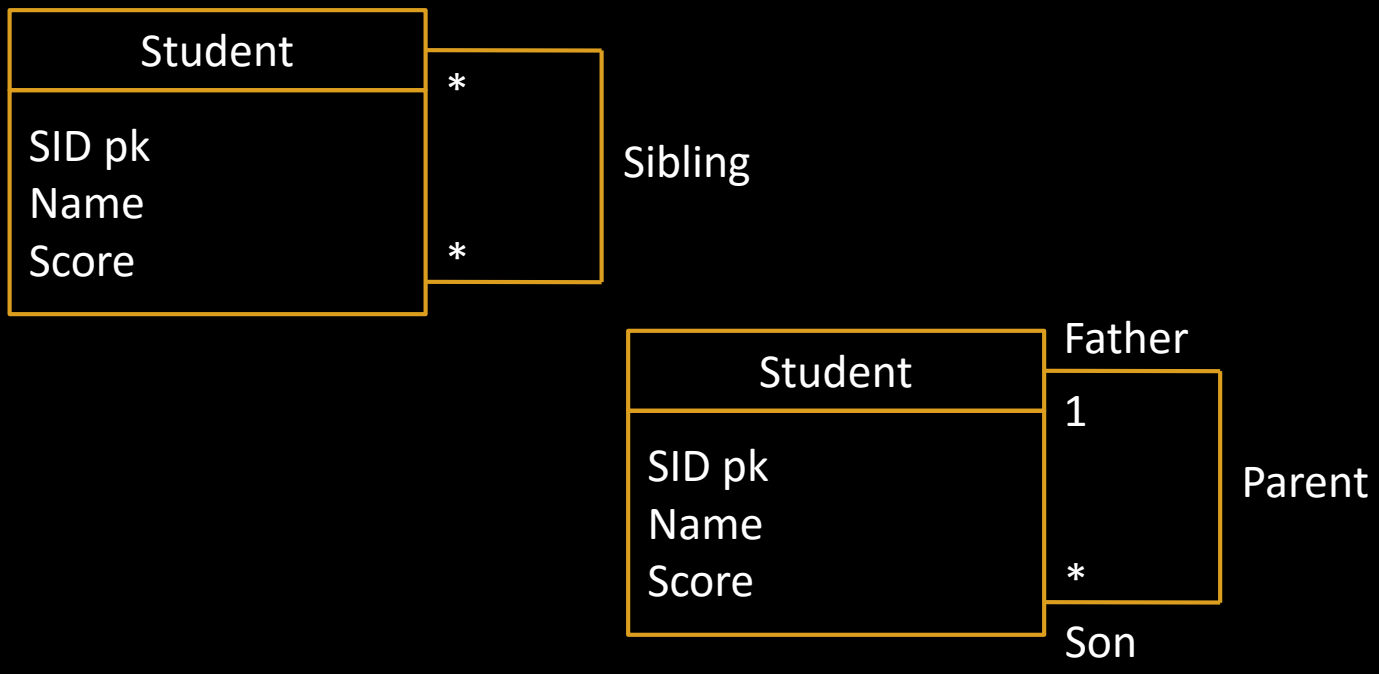
EJEMPLO DE ASOCIACIÓN

- Los estudiantes universitarios pueden solicitar el acceso hasta en 5 universidades, aunque como mínimo tendrán que hacerlo en una.
- Una universidad no puede tener más de 20.000 solicitantes al mismo tiempo.



ASOCIACIÓN SOBRE UNA MISMA CLASE

- Una asociación puede tener como ambos extremos la misma clase.
- Suele ser necesario especificar qué rol cumple cada extremo si la asociación no es simétrica.

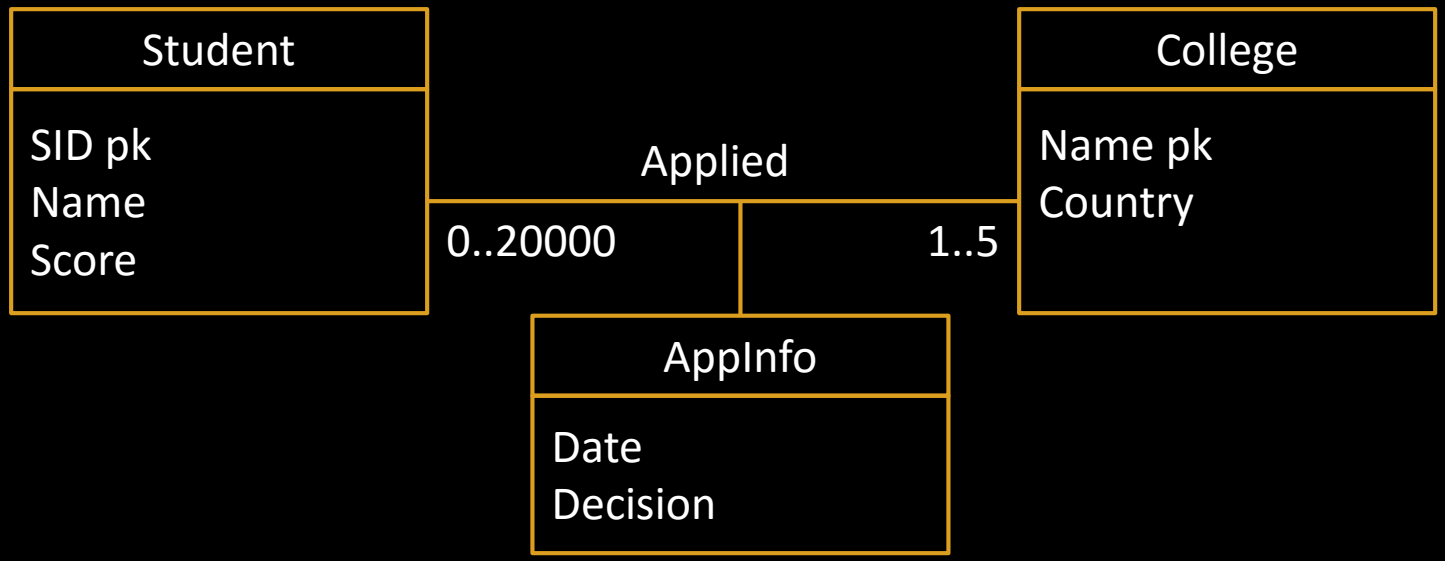


TIPOS DE ASOCIACIÓN EN FUNCIÓN DE LA MULTIPLICIDAD

- Tomando las posibles multiplicidades a ambos lados de una asociación definimos los siguientes **tipos especiales de asociaciones**:
  - *One-to-One* (una con una):
    - 0..1 en ambos lados.
  - *Many-to-One* (una con muchas):
    - 0..1 en un lado y 0..* en otro.
  - *Many-to-many* (muchas con muchas):
    - 0..* en ambos lados.
- También podemos especificar que estos tipos sean **completos**:
  - *One-to-One* completo: 1 en ambos lados.
  - *Many-to-One* completo: 1 en un lado y 1..* en el otro.
  - *Many-to-Many* completo: 1..* en ambos lados.

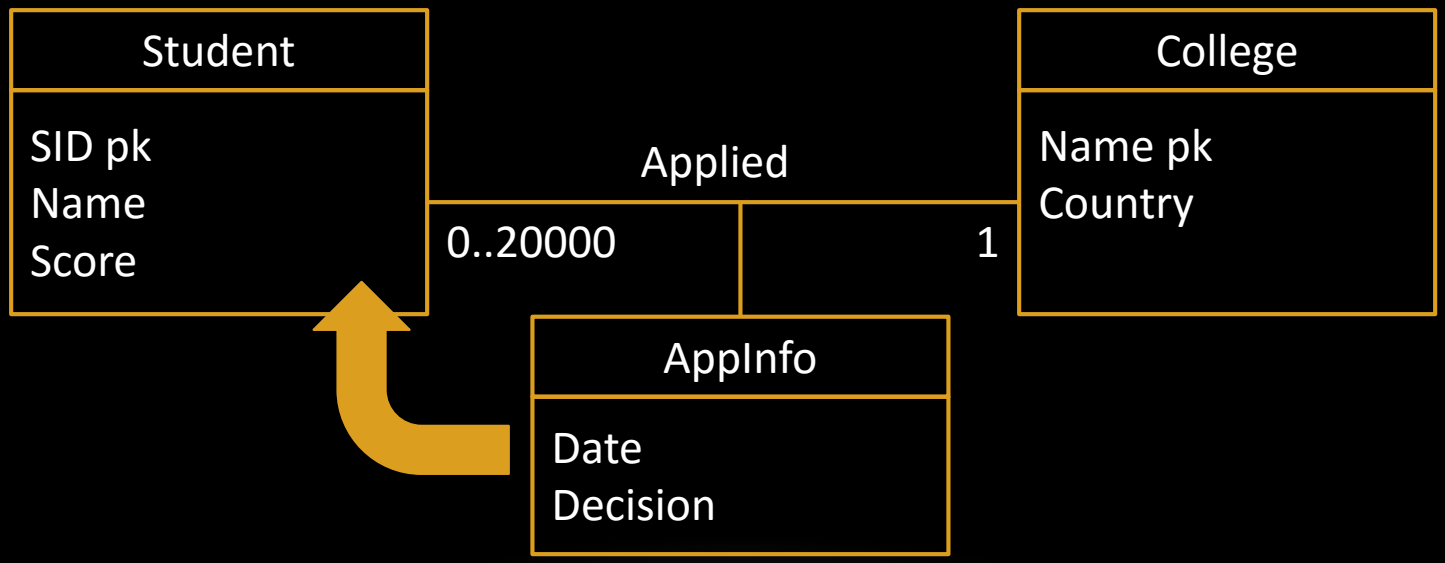
CLASES DE ASOCIACIÓN

- Son clases que no representan objetos, sino características de una relación entre otras dos clases.
- Proporcionan un detalle extra que sólo un nombre no puede dar.
- Tienen atributos pero no clave primaria.

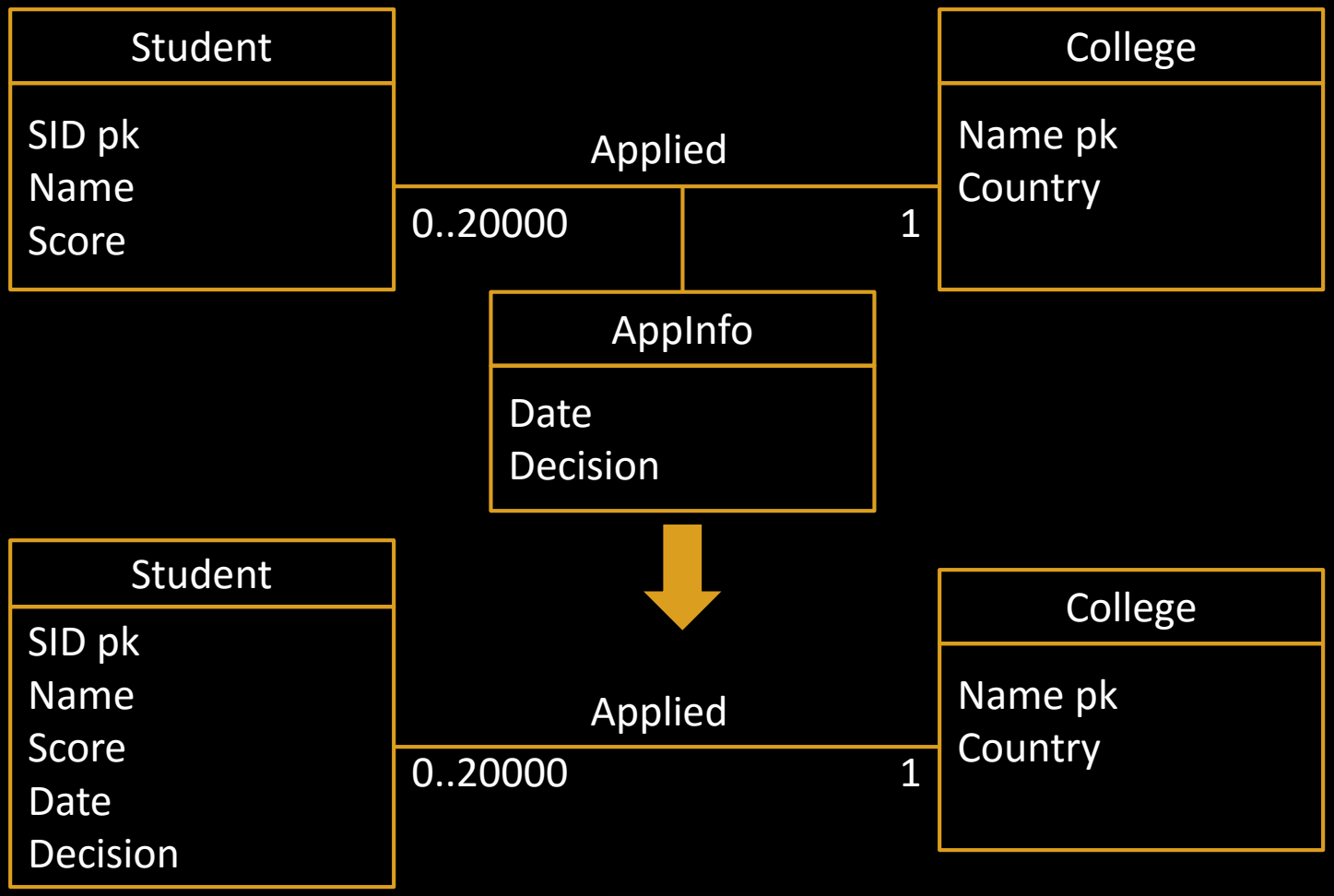


ELIMINAR CLASES DE ASOCIACIÓN

- Muchas veces las clases de asociación no son realmente necesarias:
  - Si una de las multiplicidades es 0..1 o 1.
  - Especialmente recomendable en caso de 1.
- En tal caso podemos incorporar los atributos de la clases de asociación a una de las dos clases.



EJEMPLO DE ELIMINACIÓN DE CLASES DE ASOCIACIÓN



COMPOSICIÓN Y AGREGACIÓN

- Dos casos particulares de la asociación. Ambas representan la relación entre partes y un todo.
- **Agregación:**
  - Caso especial de una asociación *Many-to-One*.
  - Las partes y el todo no se necesitan mutuamente para existir.
    - Tienen sentido independientemente. El todo "**usa**" a las partes.
  - En lugar del 0..1 se utiliza un rombo hueco.
  - Si no se indica explícitamente el número de partes se considera *.
  - Es necesario que las partes tengan "pk".
- **Composición:**
  - Caso especial de una asociación *Many-to-One*.
  - Las partes y el todo se necesitan mutuamente para existir.
    - **No** tienen sentido independientemente. El todo "**posee**" a las partes.
  - En lugar del 1 se utiliza un rombo relleno.
  - Si no se indica explícitamente el número de partes se considera *.
  - En principio, no es necesario que las partes tengan "pk" (pero es **muy recomendable**).

EJEMPLOS DE COMPOSICIÓN Y AGREGACIÓN

- Agregación:

- Composición:

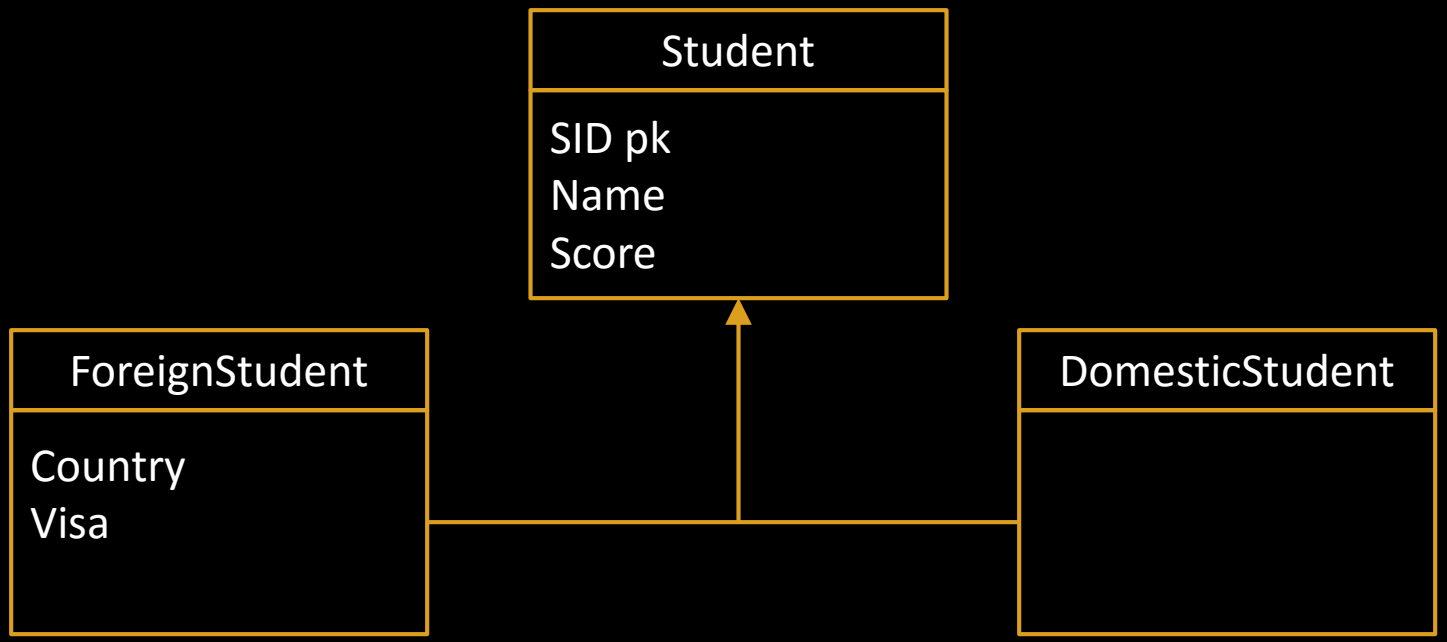
HERENCIA

- Una flecha con “cabeza sólida” representa la herencia entre dos clases.
- Esta herencia se compone de dos relaciones:
  - **Generalización:** La **superclase** (o clase padre) es la versión más general.
    - La apuntada por la flecha.
  - **Especialización:** La **subclase** (o case hija) es la versión más específica.
- En este caso no se especifica la multiplicidad.
  - No se trabaja con objetos, sino con clases.
- La subclase posee todos los atributos, asociaciones, composiciones y agregaciones de la superclase.
- Se suelen combinar varias flechas en una.
- Las subclases no necesitan “pk”.

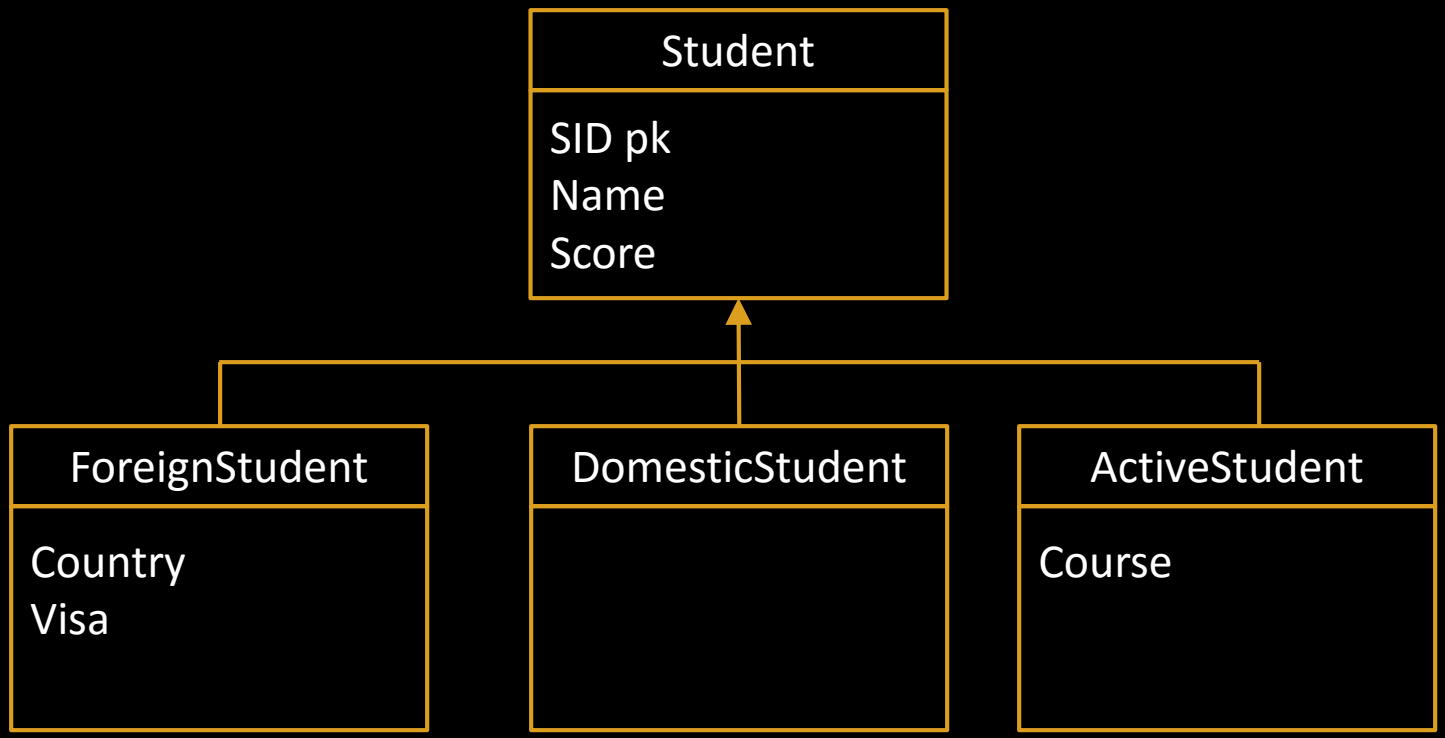
TIPOS DE HERENCIA

- Podemos clasificar una relación de herencia atendiendo a dos dimensiones.
- **Completitud**:
  - **Complete:** *Completa*. Todo objeto instancia de la superclase es también instancia de al menos una de las subclases.
  - **Incomplete (partial):** *Incompleta*. Un objeto puede ser instancia de la superclase sin ser instancia de ninguna de las subclases.
- **Exclusividad:**
  - **Disjoint (exclusive):** *Disjunta*. Si un objeto es instancia de una subclase entonces no puede serlo de ninguna otra subclase de la misma superclase.
  - **Overlapping:** *Superpuesta.* Un objeto puede ser instancia de varias subclases de la misma superclase al mismo tiempo.
- La completitud y la exclusividad se indican en el diagrama UML entre llaves: "{}"

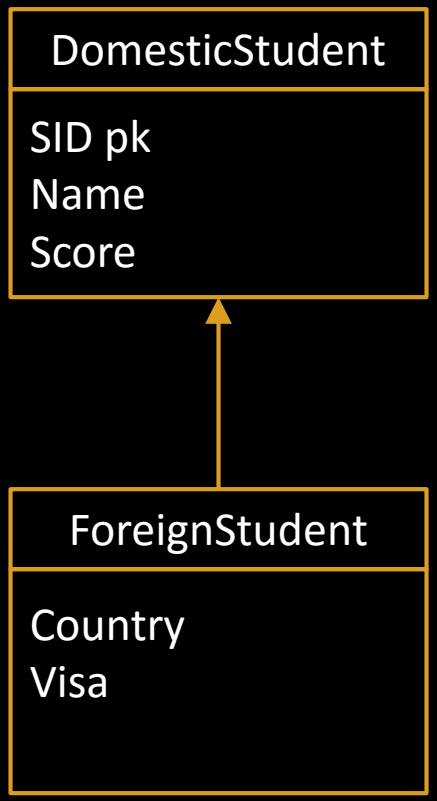
EJEMPLO DE HERENCIA COMPLETA Y DISJUNTA



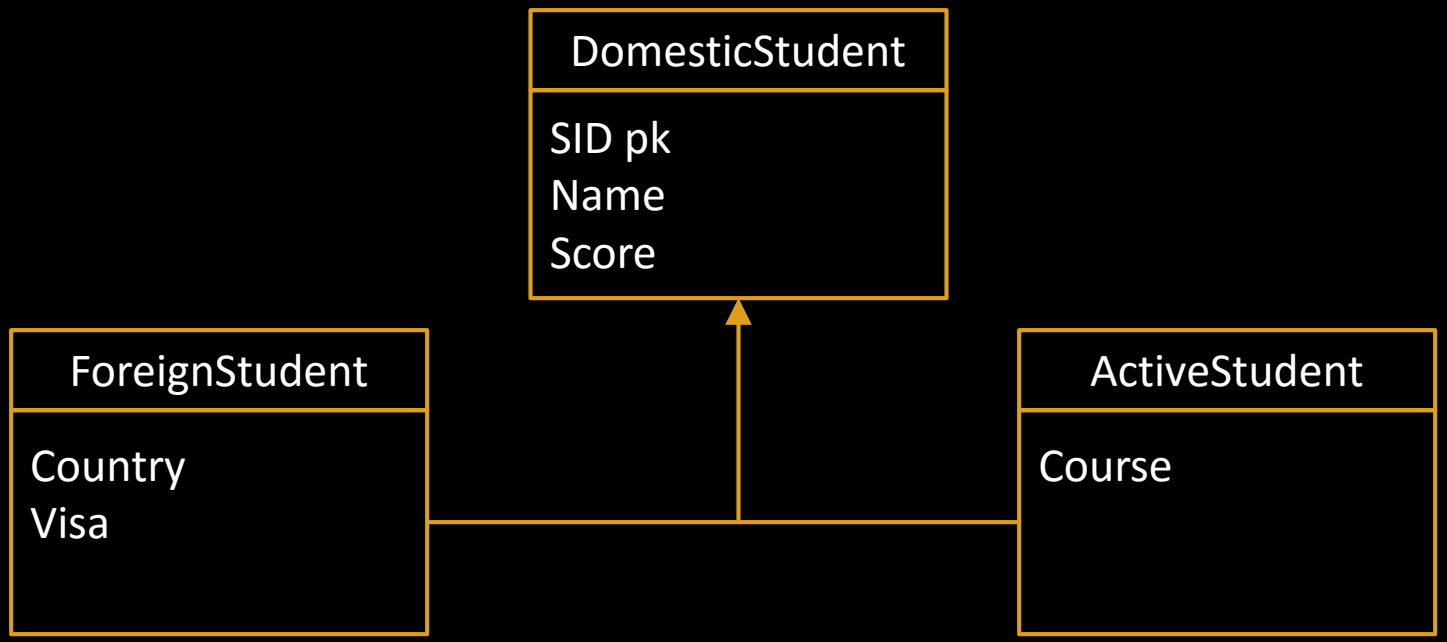
EJEMPLO DE HERENCIA COMPLETA Y SUPERPUESTA



EJEMPLO DE HERENCIA INCOMPLETA Y DISJUNTA



EJEMPLO DE HERENCIA INCOMPLETA Y SUPERPUESTA

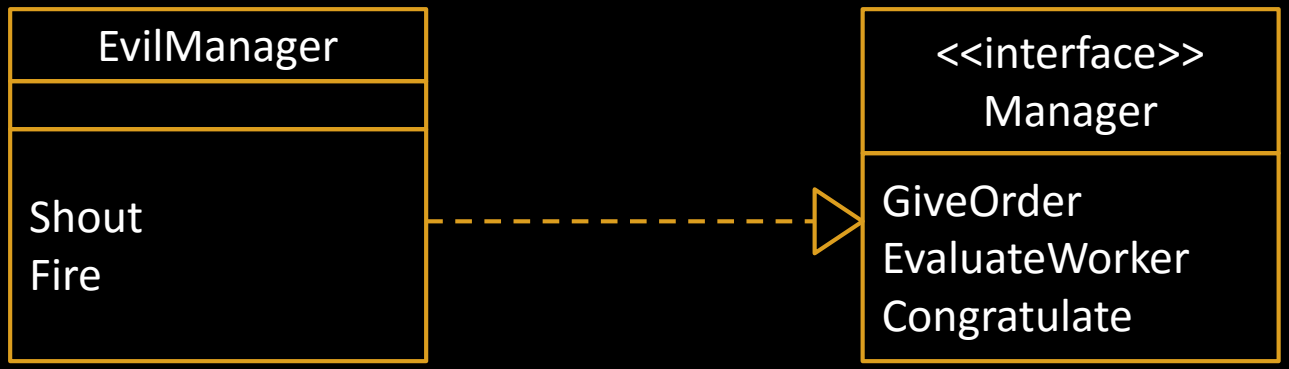


REALIZACIÓN: INTERFACES

- Para entender la realización es necesario entender el concepto de **interfaz**.
- En UML un interfaz es un tipo especial de clase que:
  - Define una funcionalidad que otra clase tiene que implementar.
  - **No proporciona ninguna funcionalidad por sí misma.**
  - Es un "contrato" que la clase que lo implementa debe seguir.
  - No posee atributos.
- Podemos entender un interfaz como una "plantilla" para construir clases.
- Se representan como una clase con la palabra *<<interface>>* en el nombre.
- En la escuela de "los datos no deben tener comportamiento" **no se usan interfaces:**
  - Están pensadas para definir un comportamiento, no información.
  - No poseen atributos, sólo métodos.
  - No pueden ser instanciados.

REALIZACIÓN

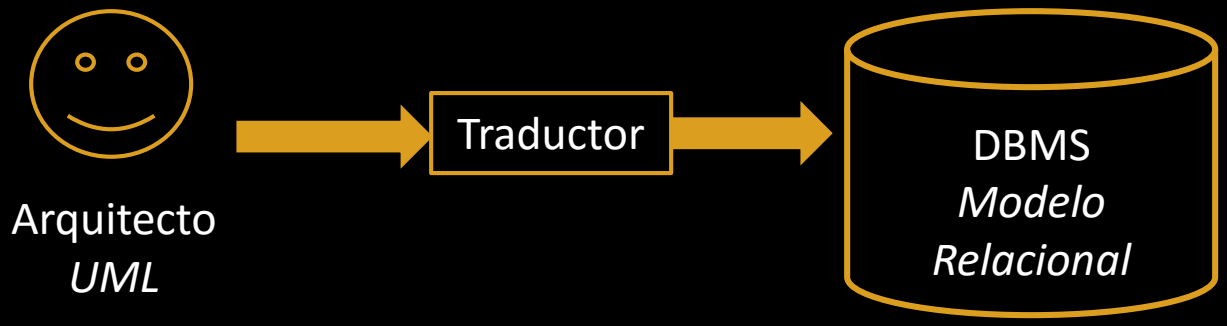
- La realización se establece entre un interfaz y la clase que lo implementa.
- Se representa mediante una flecha de línea punteada con una “cabeza” hueca.
- Conceptualmente es similar a una herencia completa disjunta.



TRADUCCIÓN DE UML A RELACIONES

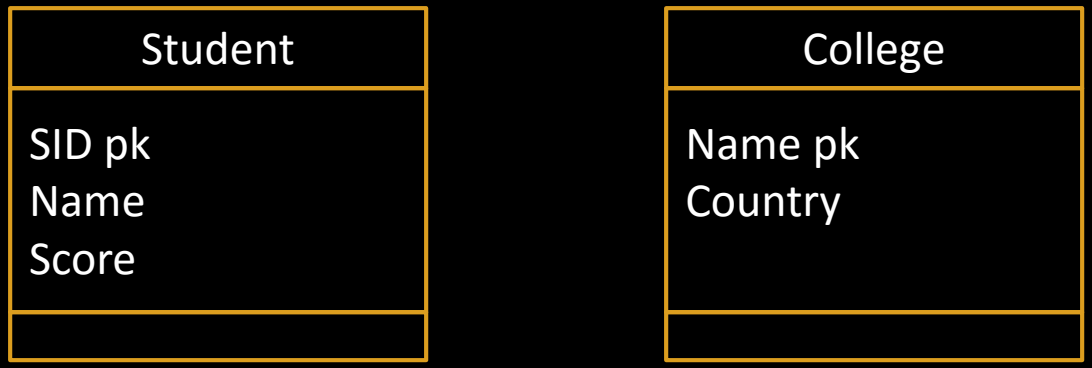
TRADUCCIÓN DE UML A RELACIONES

- UML es apto para ser procesado por un ser humano.
- Pero muchas veces es necesario traducir al modelo relacional.
  - Es el que utilizan los DBMS.
  - Es más eficiente.
- Puede hacerse de forma semiautomática.
  - Siempre que todas las clases “normales” tengan un “pk”.



TRADUCCIÓN DE CLASES

- Cada clase se convierte en una tabla.
- El “pk” se convierte en la clave primaria *(primary key)*.
- Los atributos se convierten en columnas.

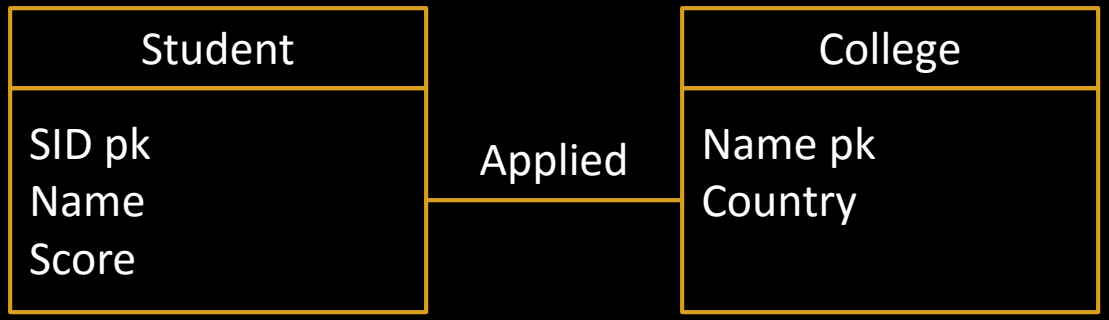


Student(<u>SID</u>, Name, Score)

College(<u>Name</u>, Country)

TRADUCCIÓN DE ASOCIACIONES

- Cada asociación se convierte en una tabla.
- Las columnas son los “pk” de cada clase relacionada. Las *foreign keys*.



Student(<u>SID</u>, Name, Score)

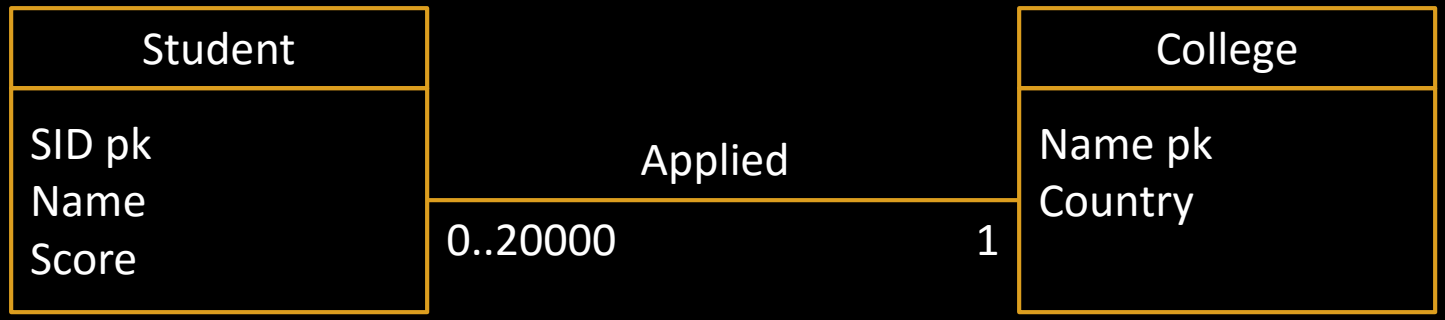
College(<u>Name</u>, Country)

Applied(SID, Name)

TRADUCCIÓN DE ASOCIACIONES: CLAVES PRIMARIAS

- La clave primaria de la tabla que representa la asociación depende de la multiplicidad de ésta.
  - **Ambos lados son 0..1 o 1:**
    - La clave primaria es el “pk” de cualquiera de los dos lados.
    - No es necesario crear la tabla (lo veremos luego).
  - **Sólo un lado es 0..1 o 1:**
    - La clave primaria es el “pk” del lado que no es 0..1 o 1.
    - No es necesario crear la tabla (lo veremos luego).
  - **Ninguno de los lados es 0..1 o 1:**
    - La clave primaria es la combinación de ambos “pk”.

EJEMPLOS DE TRADUCCIÓN DE ASOCIACIONES: CLAVES PRIMARIAS



Applied(<u>SID</u>, Name)

Applied(<u>SID, Name</u>)

TABLAS INNECESARIAS

- No siempre es necesario crear una nueva tabla para representar una asociación.
- A veces la información de la asociación puede incorporarse a una de las dos tablas.
- Esto es posible cuando una de ellas tiene multiplicidad 0..1 o 1.
  - Si la multiplicidad es 0..1 la tabla debe soportar atributos de valor NULL.

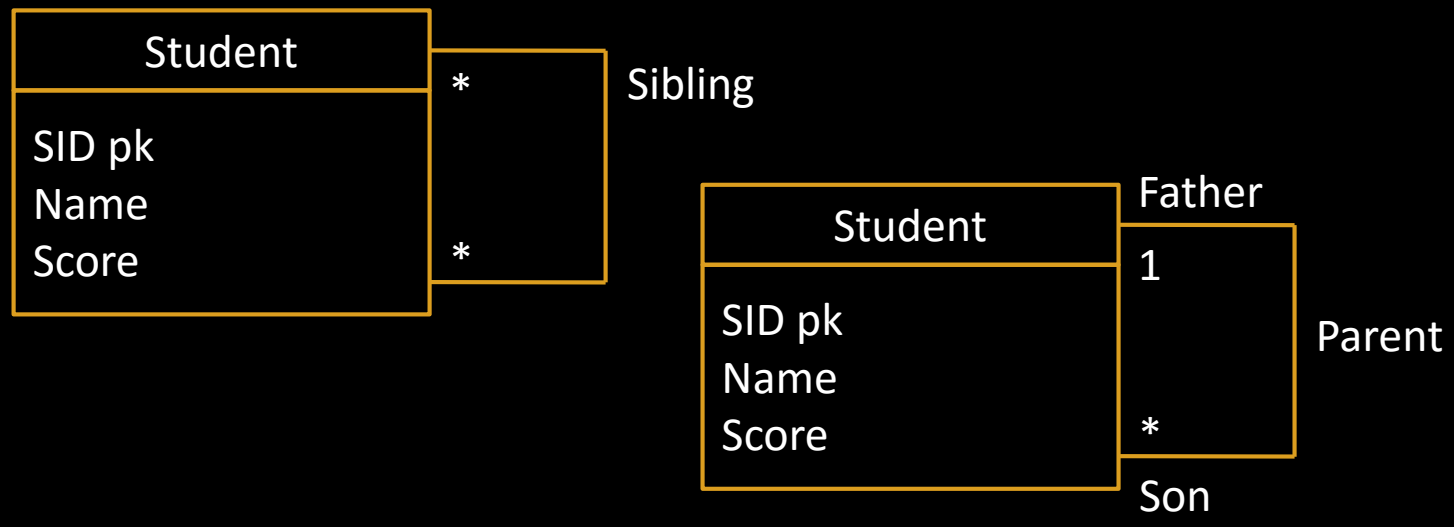
Student(<u>SID</u>, Name, Score, College.Name)

College(<u>Name</u>, Country)

~~Applied(<u>SID</u>, Name)~~

TRADUCCIÓN DE ASOCIACIONES SOBRE UNA MISMA CLASE

- En la tabla que representa la asociación se utilizan dos “pk” de la clase que toma parte en ella.
- Igual que antes, algunas tablas pueden ser innecesarias y agregarse a la clase.

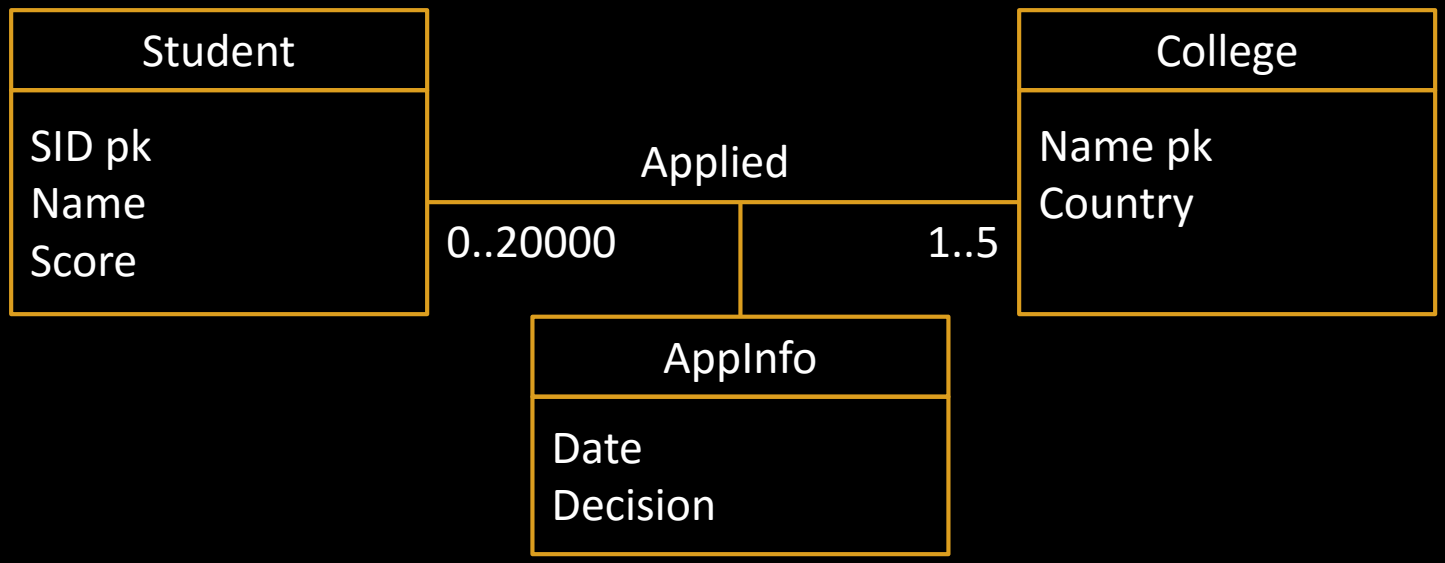


Sibling(<u>SID1</u>, <u>SID2</u>)

Parent(SIDFather, <u>SIDSon</u>)

TRADUCCIÓN DE CLASES DE ASOCIACIÓN

- Añadimos a la tabla que representa la asociación los atributos de la clase de asociación.



Student(<u>SID</u>, Name, Score)

College(<u>Name</u>, Country)

Applied(<u>SID, Name</u>, Date, Decision)

TRADUCCIÓN DE AGREGACIÓN Y COMPOSICIÓN

- Las bases de datos relacionales no soportan la semántica de agregación y composición que ofrece UML.
  - **Se tratan simplemente como asociaciones.**
- La agregación se traduce igual que una asociación con multiplicidad 0..1.
  - La table debe poder soportar atributos de valor NULL.
- La composición se traduce igual que una asociación con multiplicidad 1.

EJEMPLOS DE TRADUCCIÓN DE AGREGACIÓN Y COMPOSICIÓN

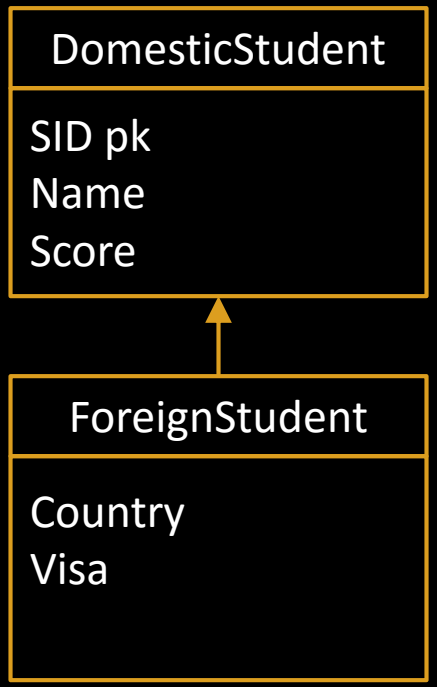
- Agregación: Professor(PID, Name, Course, College.Name)

- Composición: Department(Name, Building, College.Name)

TRADUCCIÓN DE HERENCIA

- La herencia no existe en el modelo relacional. Hay que hacer “apaños”.
- Existen tres acercamientos posibles:
  - **Una tabla para la clase padre y una para cada clase hija:** La tabla de la clase hija contiene el “pk” de la clase padre y los atributos nuevos.
    - Adecuado para herencia disjunta e incompleta.
  - **Una tabla para cada clase hija:** La tabla de la clase hija contiene todos los atributos de la clase padre y de la clase hija.
    - Adecuado para herencia disjunta y completa.
  - **Una única tabla para todas las clases:** La tabla contiene los atributos de la clase padre y de todas las clases hijas.
    - Adecuado para herencia superpuesta.
- La decisión final siempre depende de las circunstancias concretas.

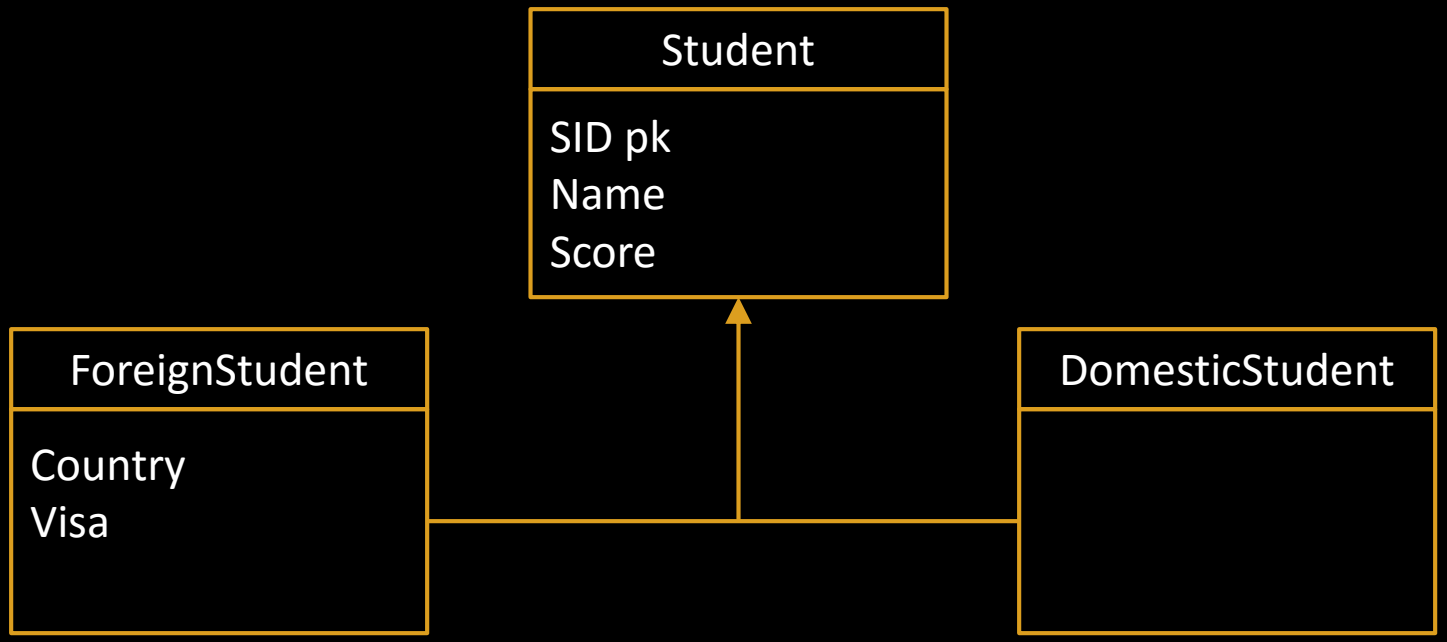
EJEMPLO DE TRADUCCIÓN CON HERENCIA INCOMPLETA Y DISJUNTA



DomesticStudent(SID, Name, Score)

ForeignStudent(SID, Country, Visa)

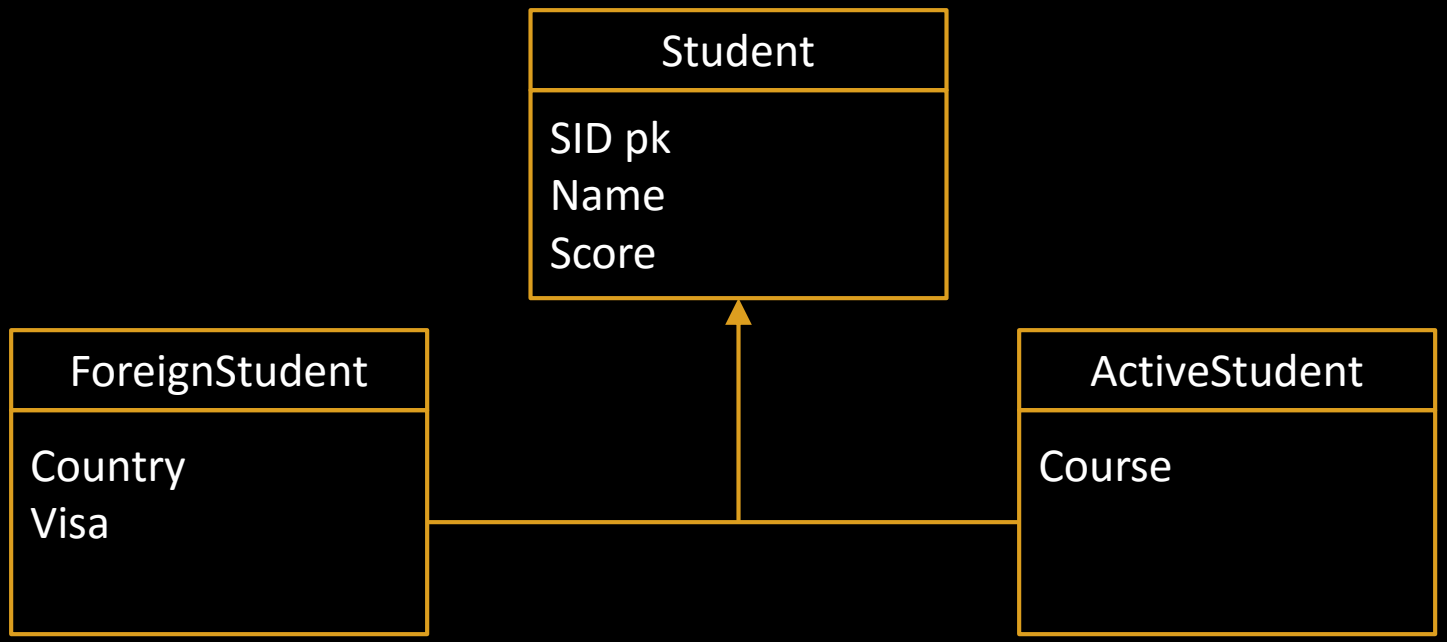
EJEMPLO DE TRADUCCIÓN CON HERENCIA COMPLETA Y DISJUNTA



ForeignStudent(SID, Name, Score, Country, Visa)

DomesticStudent(SID, Name, Score)

EJEMPLO DE TRADUCCIÓN CON HERENCIA SUPERPUESTA



Student(<u>SID</u>, Name, Score, Country, Visa, Course)

UML Y JAVA

UML Y JAVA

- Al estar UML diseñado para representar información en lenguajes orientados a objetos, su traducción a Java es inmediata.
- **La mayoría de correspondencias son directas:**
  - Clases UML a clases Java.
  - Asociaciones como uso de objetos de otra clase.
  - Composición como una asociación creada en el constructor.
  - Agregación como una asociación con la multiplicidad correcta.
  - Herencia como herencia en Java vía “extends”.
  - Realización como herencia en Java vía “implements”.
- Hay software que permite **generar automáticamente** código básico Java a partir de UML, y viceversa.
  - El más popular es el *framework* EMF.

RESUMEN DEL TEMA

- UML es un lenguaje gráfico, nacido de la orientación a objetos.
  - Adecuado para una visualización fácil de la estructura de la información.
  - El que usamos al programar.
- El modelo relacional es el lenguaje en el que están implementados los DBMS.
  - Eficiente y optimizado para su manejo por máquinas.
  - El que usamos para manejar datos.
- La parte de modelado de datos de UML presenta 7 conceptos fundamentales:
  - Clase, asociación, clase de asociación, composición, agregación, herencia y la realización.
- Se puede traducir semiautomáticamente de UML al modelo relacional.
  - Salvo la realización, que no se utiliza para modelar datos.
- La traducción entre Java y UML es inmediata, con correspondencias directas.