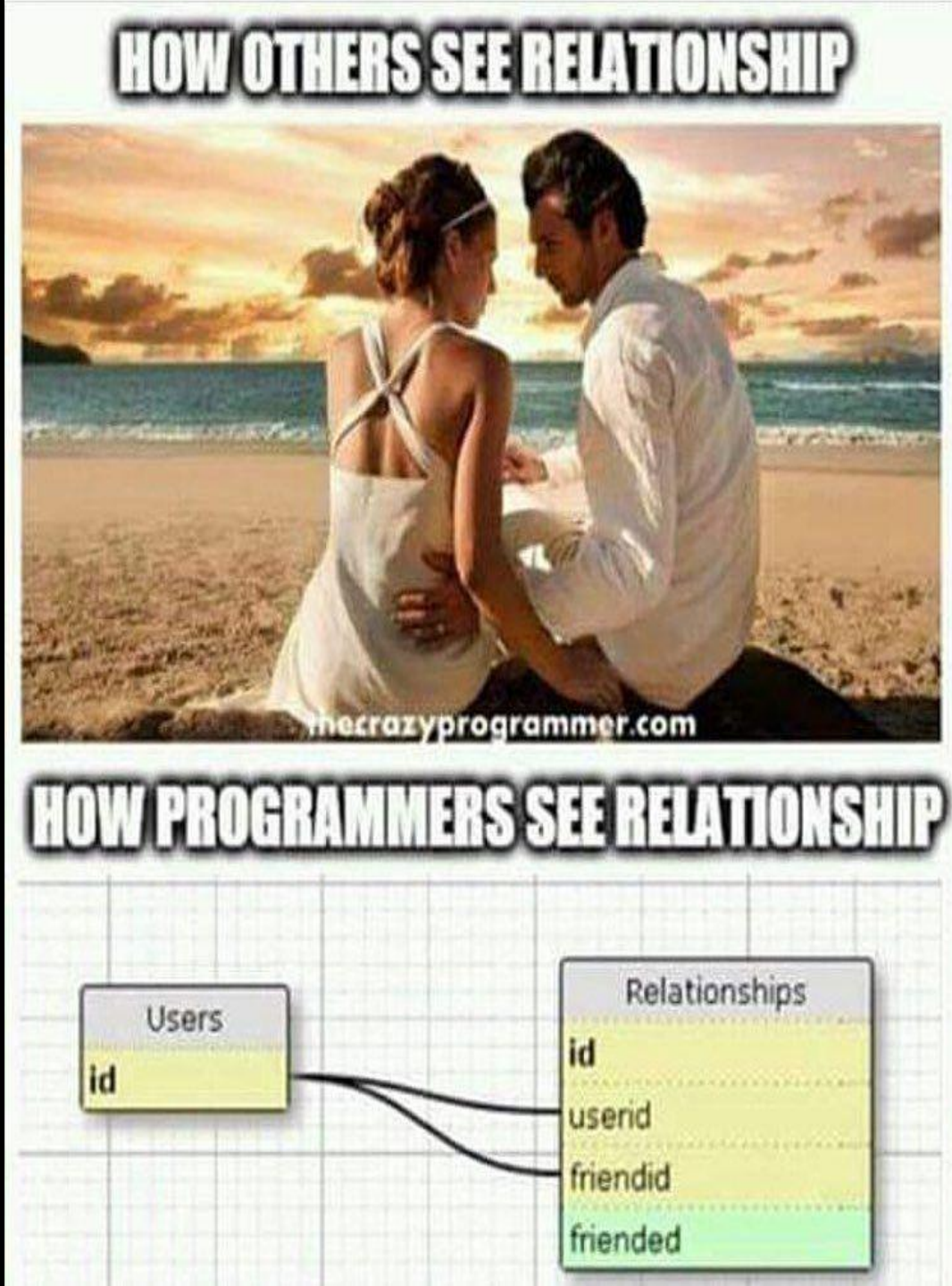
Relación entre Clases

Isis Bonet Cruz, PhD



Tipos básicos de relaciones entre clases

GENERALIZACIÓN/ESPECIALIZACIÓN

Denota una relación “ES-UN” (IS-A)

Por ejemplo:

- La ROSA es un tipo de FLOR.
- El DOVERMAN es un PERRO.
- El LADA es un AUTOMÓVIL LIGERO
- El PARGO es un PEZ.

Relación de Herencia

La herencia es una relación entre clases en la que una clase comparte la estructura y/o comportamiento definidos en una (herencia simple) o más clases (herencia múltiple)

Herencia

Una clase de las que otras heredan se llama SUPERCLASE.

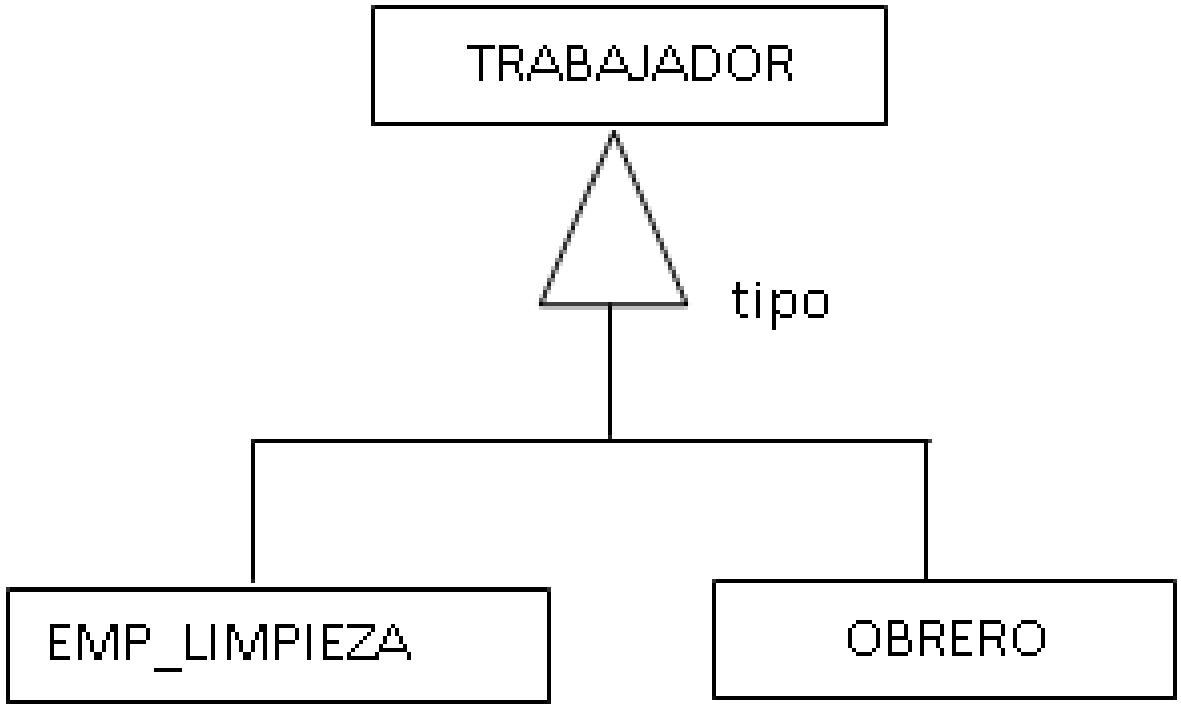
Una clase que hereda de otra se llama SUBCLASE.

Al decir que una subclase hereda de una superclase, se está diciendo que la funcionalidad y los datos que se asocian con la clase hija (o subclase) forman un **superconjunto** de la funcionalidad y datos asociados con la clase paterna o superclase.

Ejemplo

Se quiere construir un sistema para el cálculo del salario de una empresa. En la misma todos los trabajadores tienen un salario básico, el cual se paga por días trabajados, pero existen dos tipos de trabajadores:

- los de limpieza a los cuales se le paga un plus en dependencia del porcentaje de cumplimiento de una norma de metros cuadrados establecida para cada uno (si cumple toda la norma es un 10% del salario, sino es según lo que cumpla)
- y los obreros directos a la producción que reciben un plus en correspondencia con el cumplimiento de su plan de producción y la calidad del mismo.


TRABAJADOR
tipo
EMP_LIMPIEZA
OBRERO

Implementación en Java

class <nombre clase> **extends** <nombre clase ancestra>**{**

...

}

Consideraciones

Al hacer la declaración de herencia de la subclase:

- La subclase tiene todos los atributos y los métodos de la superclase
- Queda declarar los elementos específicos de la subclase

- No es posible, en este caso, eliminar nada de la superclase
- Se pueden redefinir atributos y métodos (o sea incorporar a la subclase atributos y métodos con el mismo nombre que en la superclase, en este caso se dice que el atributo o método de la subclase anula el atributo o método del ancestro

Modificador protected

Los atributos y métodos que aparecen con el modificador **protected** pueden ser accedidos desde la clase en que se declaran y desde cualquier clase que herede de ésta

Ejemplo

```
public class Trabajador{
  protected String nombre;
  protected float sb;     // salario básico
  protected float dt;     // días trabajados

  public Trabajador(String nombre, float sb, int dt){
    this.nombre=nombre;
    this.sb=sb;
    this.dt=dt;
  }

  public float salario(){
    return sb*dt/24;
  }
}
```

Ejemplo

```
class EmpLimpieza extends Trabajador{

  private float norma;
  private float cump;

  public EmpLimpieza(String nombre, float sb, int dt,
                     float norma, float cump){
    super(nombre, sb, dt); //llamar constructor ancestro
    this.norma=norma;
    this.cump=cump;
  }

  public float salario(){
    return
super.salario()+super.salario()*0.1*cump/norma;
  }

}
```

Los objetos se crean de la misma forma:

```
class TestTrabajador{

public static void main(String[] arg){

Trabajador t=new Trabajador("Juan Gonzalez", 400, 23);
// se ha creado un nuevo trabajador
System.out.println("El Salario es: "+t.salario());

EmpLimpieza l= new EmpLimpieza("Maria Linares", 250, 19,50, 40);
// se ha creado un nuevo Empleado de Limpieza
System.out.println("El salario es:"+l.salario());
}
}
```

Una variable declarada de tipo de una superclase puede ser inicializada con un objeto de esa superclase o de que cualquier subclase que herede de ella

```
class TestTrabajador{

public static void main(String[] arg){

Trabajador t=new Trabajador("Juan Gonzalez", 400, 23);
// se ha creado un nuevo trabajador
System.out.println("El Salario es: "+t.salario());

Trabajador l= new EmpLimpieza("Maria Linares", 250, 19,50, 40);
// se ha creado un nuevo Empleado de Limpieza
System.out.println("El salario es:"+l.salario());
}
}
```

Clases abstractas

Se le pone el modificador **abstract** delante de la palabra **class.**

En estos casos no se pueden crear objetos de la clase.

En Java se puede especificar que no se desea que de una clase hereden otras. Esto puede ser por razones de seguridad o diseño. Para ello basta anteponer a la declaración de la clase el modificador **final.**

ASOCIACIÓN

Denota alguna dependencia semántica entre las clases, que de otro modo serían independientes

Por ejemplo:

- un PROFESOR imparte una ASIGNATURA.
- un BECADO vive en un CUARTO.

Relación de Asociación

- Esta relación es la más general entre clases, pero también la de mayor debilidad semántica. También constituye la relación que se puede representar en la programación de diversas maneras.

- Cada vez que en los problemas haya una oración que involucre dos clases de las que hemos determinado, estas clases tienen una relación de asociación (siempre que no existe entre ellas relaciones de herencia o de parte de).

Ejemplos

Ejemplos:

- PROFESOR imparte ASIGNATURA.
- ARTICULO adquirido de un PROVEEDOR.
- CAMION transporta un EMBARQUE.

Representación de Asociación

Cada vez que aparece una relación de asociación es importante determinar la cardinalidad de la misma. La cardinalidad es el elemento que nos dice cuantos objetos de una clase se relacionan con la de la otra. La cardinalidad se expresa de la siguiente forma:

Ejemplo

Un trabajador trabaja en UN departamento

En un departamento trabajan 1 o más trabajadores

En la implementación

La forma de representar una asociación en un lenguaje de programación es variada:

- Una clase puede ser instanciada en algún método de la otra.
- Objetos de una clase pueden ser pasados como parámetros de métodos de la otra.

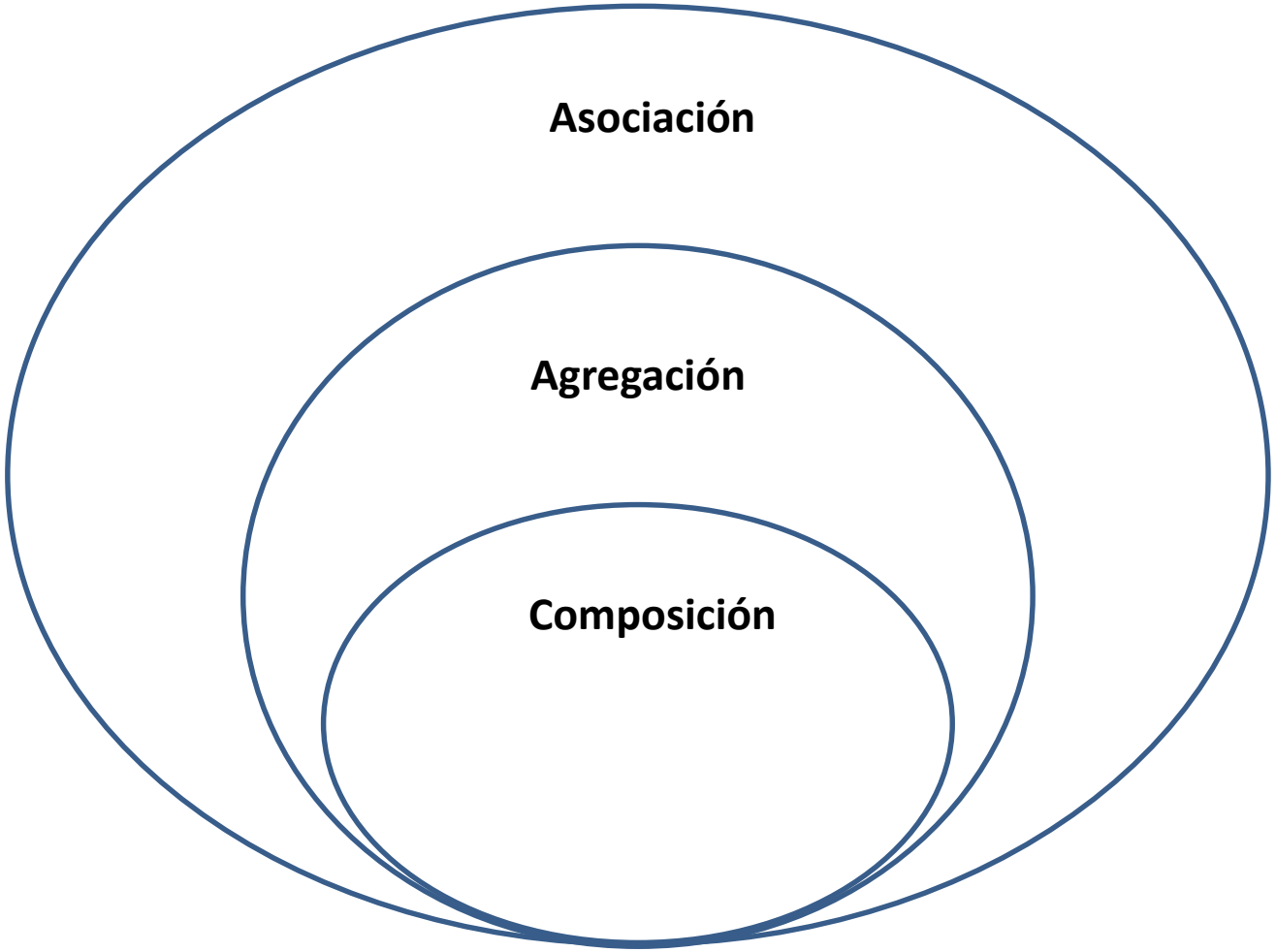
La forma en que se trabajará depende mucho del problema que se está resolviendo.

TODO/PARTE

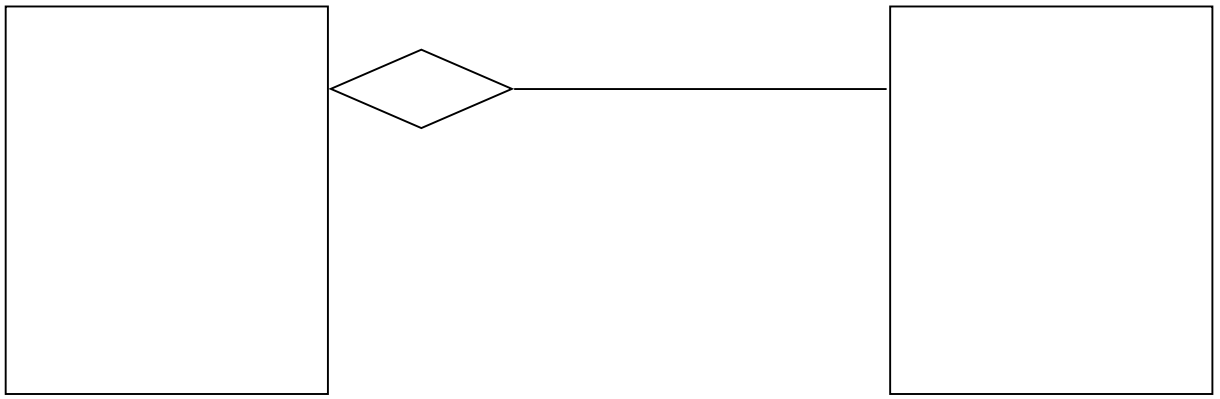
Denota una relación “parte de”

Por ejemplo:

- un PÉTALO es parte de una FLOR.
- el CARBURADOR es parte de un AUTOMÓVIL LIGERO.


Asociación
Agregación
Composición

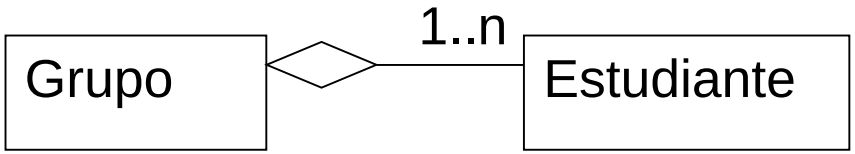
Relación de Agregación



¿Existen las PARTES independientes del TODO?

Si las partes NO existen independientemente del todo se está en presencia de la composición

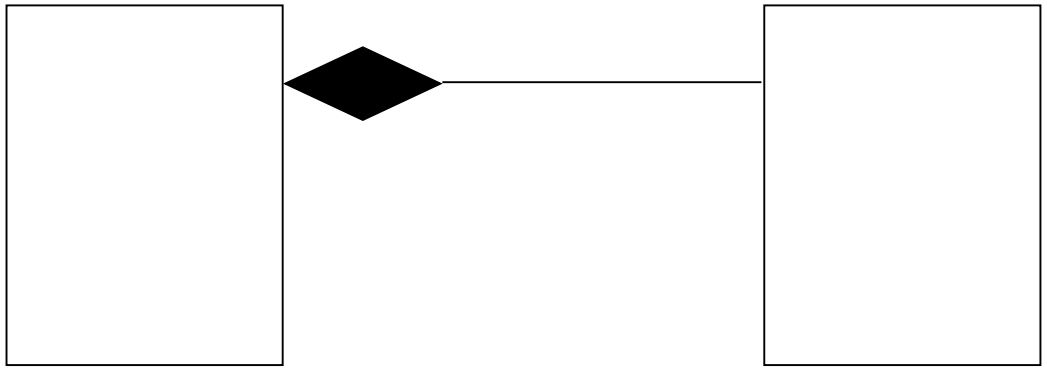
En el diagrama se puede incluir en el lado de la parte la cantidad mínima y máxima de ellas que puede tener el todo.



Esto quiere decir que en un grupo puede haber como mínimo 1 estudiante y máximo muchos. Si el máximo estuviera determinado por un número, digamos 30 se escribe 1..30.

En Java la implementación de Todo/Parte se realiza poniendo como atributo del todo la parte. En el caso que sea varias partes es necesario en la clase que representa el TODO poner un arreglo o alguna otra variante que indique una colección de las partes.
Es importante la programación correcta del constructor del TODO, pues hay que tener en cuenta las partes de alguna manera.
La destrucción del compuesto no conlleva la destrucción de los componentes.

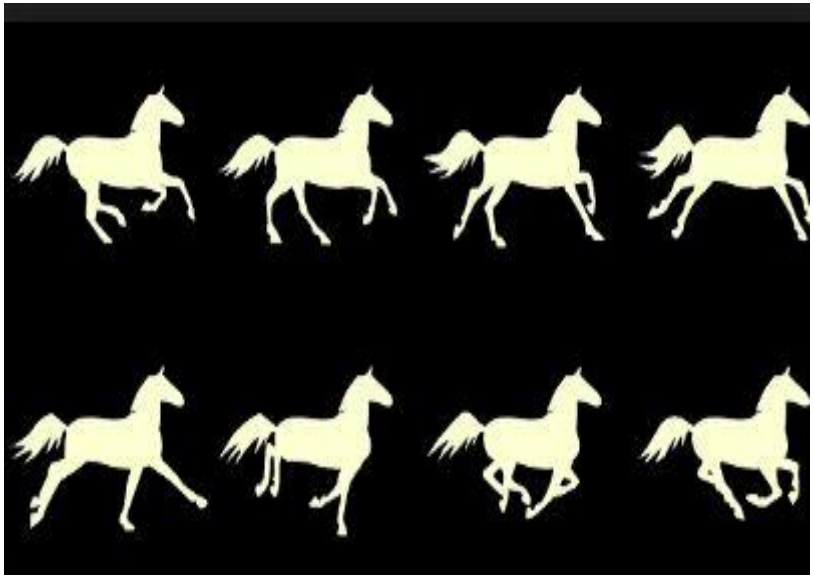
Composición



Esta decisión influye también en la programación en Java , sobre todo de los constructores.

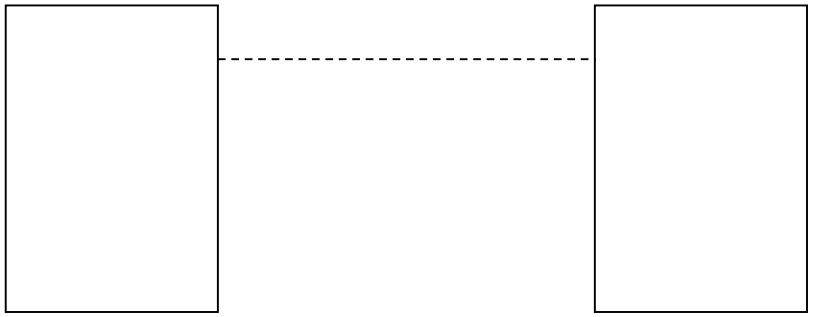
- Si no existen independientes (composición) posiblemente el constructor del TODO mande a construir las PARTES.
- Si existen independientes es posible que el constructor reciba como parámetros las PARTES.

Agregación vs. Composición



Tomado de: https://www.dokry.com/3162

Dependencia



- Es una relación de uso, es decir, una clase utiliza otra. Es la relación más débil.
- Puede ser aquellas clases en que se hace la instanciación de otros objetos.
- También puede ser cuando una clase tiene métodos que tienen como parámetros, o como retorno objetos de otras clases

Propuesto
(Termine las relaciones del diagrama)