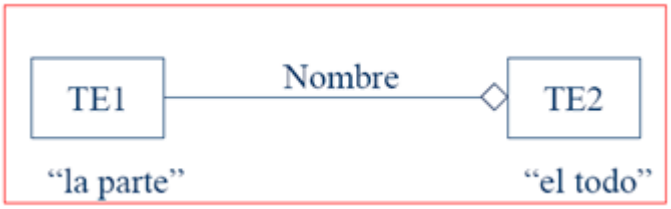
RELACIONES DE ASOCIACIÓN

Agregación:

Es una relación que se derivó de la asociación, por ser igualmente estructural, es decir que contiene un atributo, y además de ello la clase que contiene debe tener un método que asigne valor al objeto agregado. Por ejemplo: un medio de transporte tiene varios pasajeros.
Nos está diciendo que los objetos pasajero forman parte del objeto medio de transporte. ***Pero***, ***su ciclo de vida no está atado*** al del objeto medio de transporte. Es decir si el Autobus se destruye los pasajeros pueden seguir existiendo independientemente, (o por lo menos por eso rezaríamos).
También se la identifica como “es parte de” o “tiene un”.



Composición

Al igual que en la agregación, es una relación estructural pero se le suma, que tiene un método de destrucción de los objetos. Y a diferencia de la asociación, el ciclo de vida del objeto área está relacionado con el del objeto ruta. Es decir que si la ruta de viaje se levanta, las áreas que surgían a partir de ella desaparecen. También se puede leer como que una ruta tiene varias áreas de cobertura.

Clase de Asociación

Es una Clase que surge de una multiplicidad de muchos a muchos, y fue incorporada en UML para dar soporte a este caso. Se sacan los atributos de las clases involucradas y se los incorpora a una clase a parte. Al igual que las anteriores hace referencia a una relación estructural.

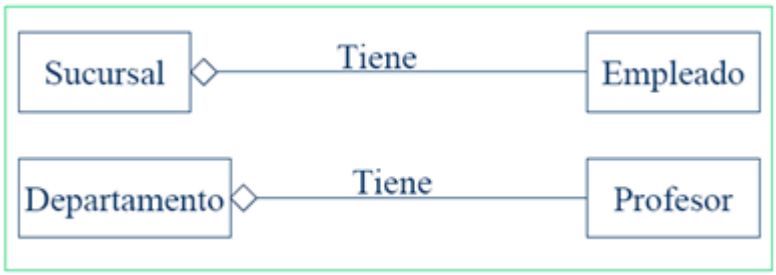
Dependencia

Es una relación de uso, es decir que una clase utiliza a otra. Y si esta ultima se altera, la anterior se puede ver afectada.
En código se suelen traducir principalmente como las clases donde se hace la instanciación de un objeto.

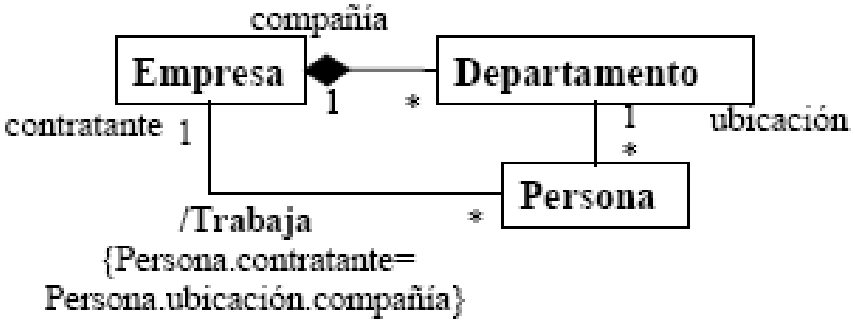
Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

Ejemplos de Asociación – Agregación - Composición

1)



2)



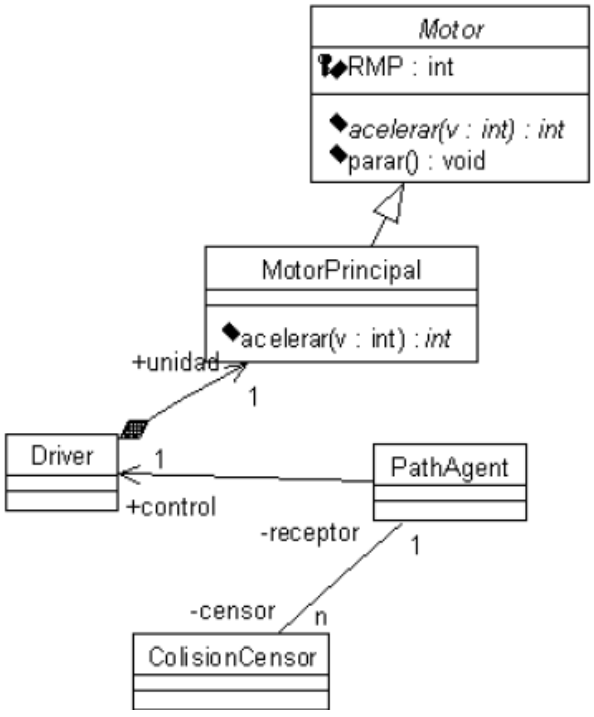
Algunos constructores:

→ public Departamento(Empresa compañia){ this. compañia = compañia;}

→ public Empresa(){ Departamento[] ubicación = new Departamento[10];}

Hay cuatro relaciones más: ¿cuáles son?

3)



class PathAgent {
private ColisionCensor [] censor;
public Driver control; }

class MotorPrincipal extends Motor {}

abstract class Motor {
protected int RMP; }

class Driver {
public MotorPrincipal unidad; }

class ColisionCensor {
private PathAgent receptor; }

Definir el constructor de Motor para inicializar sus atributos:
public Motor(int n) {
RPM = n; }
Definir el constructor de MotorPrincipal para inicializar sus atributos:
public MotorPrincipal(int n){
super(n); }

Definir el constructor de Driver para inicializar sus atributos:
public Driver(MotorPrincipal x){
unidad = x; }
Definir el constructor de PathAgent para inicializar sus atributos:
public PathAgent(ColisionCensor [] c, Driver t) {
censor = c;
control = t; }
Indique cómo acceder al método parar() desde un objeto X de la clase PathAgent:
X.control.unidad.RPM
Definir el constructor de ColisionCensor para inicializar sus atributos:

public ColisionCensor(PathAgent p) {
receptor = p; }
Definir el método acelerar(), el cual incrementa el valor de
RPM, en lo que indique el argumento que se le pasa y regresa
el nuevo valor:
public int acelerar(int v){
RPM += v;
return RPM; }
Definir el método parar(), pone a cero el valor de RPM:
public void parar(){
RPM = 0; }
Describa la creación de un objeto para Motor x;
x = new MotorPrincipal(12); //el argumento cualquier valor
Indique los nombres de las clase de objetos que es necesario
crear para una nueva instancia de un objeto PathAgent;
ColisionCensor, MotorPrincipal, Driver

4) **Ejercicio:**

Control de préstamos en una Biblioteca:
Se desea modelar semánticamente los datos relativos al control de préstamos en una Biblioteca con las siguientes restricciones:

• Cada libro puede estar escrito por más de un autor.
• Un autor puede escribir más de un libro.
• Cada libro puede tratar más de un tema.
• Hay muchos libros de cada tema
• No existe más que un ejemplar de cada libro.
• Cada persona sólo puede tomar un libro prestado a la vez

a) Realice es esquema en UML.
b) Desarrolle los constructores según las relaciones.
c) Agregue valor a los objetos de las relaciones.