

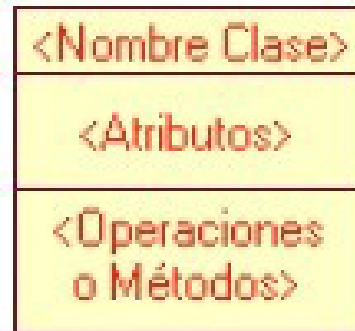
DIAGRAMAS DE CLASE

Introducción

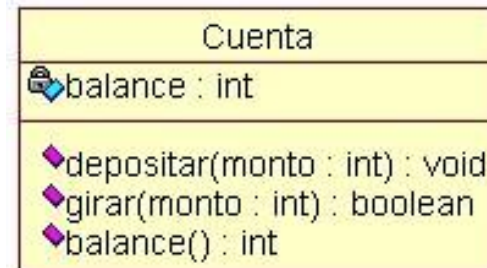
- Un diagrama de clases es un tipo de **diagrama de estructuras** que describe la estructura de un sistema mostrando sus clases y las asociaciones entre ellas
- Sirve para visualizar las relaciones entre las clases que componen el sistema
- Está compuesto por:
 - ▣ **Clases:** con sus atributos, métodos y visibilidad
 - ▣ **Relaciones:** con la asociación, herencia, agregación, composición, realización y dependencia

Clase

- En UML, una clase se representa por un rectángulo que posee tres divisiones:



- Ejemplo:
 - ▣ Una Cuenta Corriente que posee como característica:
 - Balance
 - ▣ Puede realizar las operaciones de:
 - Depositar
 - Girar
 - Balance



Atributos

- Un atributo representa alguna propiedad de la clase que se encuentra en todas las instancias de la clase
- Los atributos pueden representarse mostrando solo su nombre, nombre y tipo e incluso nombre, tipo y valor por defecto.
 - ▣ Ejemplos de Atributos:
 - Nombre, Salario, Código..
 - ▣ Tipos de atributos en UML: String, Integer, Boolean. También podemos indicar el tipo de cualquier lenguaje de programación

Atributos

- Al crear atributo se indicará su visibilidad:

Público (public)	+	Elemento no encapsulado visible para todos.
Protegido (protected)	#	Elemento encapsulado visible en la clase y las subclases de la clase.
Privado (private)	-	Elemento encapsulado visible solo en la clase.
Paquete (package)	~	Elemento encapsulado visible solo en las clases del mismo paquete.

Métodos

- Un método (también llamado operación) es la implementación de un servicio de la clase que muestra un comportamiento común a todos los objetos
- Definen la manera en que la clase interactúa con su entorno
- Las opciones de visibilidad de los métodos son las mismas que los atributos

Ejemplo

```
public class Cuenta
{
    private double balance = 0;
    private double limit;

    public void ingresar (double cantidad)
    {
        balance = balance + cantidad;
    }

    public void retirar (double cantidad)
    {
        balance = balance - cantidad;
    }
}
```

+ Cuenta
-balance : double =0 -limite : double
+depositar(cantidad : double) : void +retirar(cantidad : double) : void

Ejemplo

```
public class Motocicleta
{
    // Atributos (variables de instancia)
    private String matricula; // Placa de matrícula
    private String color;     // Color de la pintura
    private int velocidad;    // Velocidad actual (km/h)
    private boolean en_marcha; // ¿moto arrancada?

    // Operaciones (métodos)
    public void arrancar ()
    { ... }
    public void acelerar ()
    { ... }
    public void frenar ()
    { ... }
    public void girar (float angulo)
    { ... }
}
```

Motocicleta

-matricula
-color
-velocidad
-en_marcha

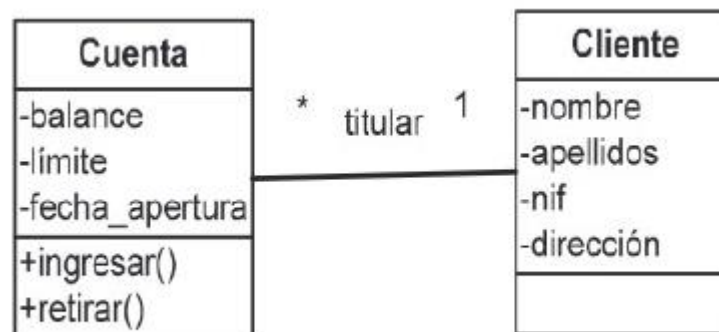
+arrancar()
+acelerar()
+frenar()
+girar()

Relaciones

- En el mundo real los objetos están relacionados entre sí
 - ▣ Ejemplo: vínculo entre alumno y curso matriculado
- En UML estos vínculos se describen mediante asociaciones
- La asociación tiene un **nombre** y una cardinalidad, llamada **multiplicidad**, que representa el número de instancias de una clase que se relacionan con las instancias de la otra clase (similar modelo E/R)
 - ▣ La multiplicidad situada en un extremo indica a cuántas instancias de la clase situada en ese mismo extremo está vinculada una instancia situada en el otro extremo
 - ▣ En un extremo es posible especificar cardinalidad mínima y máxima.

Relaciones

Notación	Cardinalidad / Multiplicidad
<i>0..1</i>	Cero o una vez
<i>1</i>	Una y solo una vez
<i>*</i>	De cero a varias veces
<i>1..*</i>	De una a varias veces
<i>M..N</i>	Entre M y N veces
<i>N</i>	N veces



Para ampliar... De UML a Java

- Cuando el diagrama se convierte en código, dependiendo de la herramienta de modelado que se utilice, las multiplicidades destino mayores que 1 se implementa como un atributo del tipo array o bien un atributo de algún tipo colección o set (en el ejemplo Vector)

Tipos de relaciones

Distinguimos los siguientes tipos de relaciones:

- **Asociación:** Permite asociar objetos que se relacionan entre si
- **Herencia** (generalización y especialización): Una subclase hereda los atributos y métodos de una superclase
 - persona superclase
 - alumno y empleado subclases
- **Composición:** Un objeto está compuesto por otros objetos
 - ordenador compuesto por placabase-memoria-teclado..
- **Agregación** : Es una composición débil. Un objeto está compuesto por otros objetos pero estos objetos pueden formar parte de otros
 - (Equipo - jugador)
- **Realización:** Relación entre clase abstracta y las clases la implementan
- **Dependencia:** Relación que se establece cuando una clase usa a la otra

Asociación

- Dentro de la relación cada clase juega un rol
- Dependiendo si ambas conocen la existencia de la otra o no, puede ser

- ▣ bidireccional o
- ▣ unidireccional



- La navegabilidad entre clases nos muestra que es posible pasar de un objeto de clase origen a uno (o más, dependiendo de la multiplicidad) de la de destino
- En el caso de unidireccionalidad la navegación irá en un solo sentido

Asociación. Ejemplo

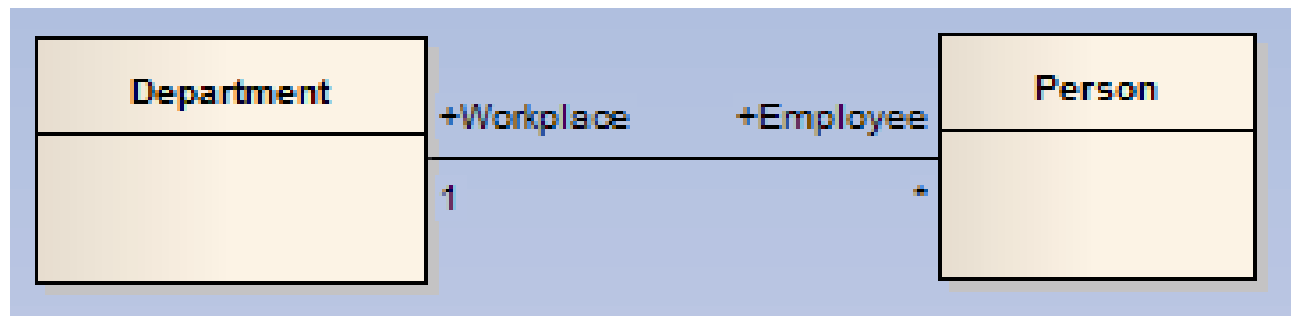


■ Asociación Tiene:

- Un cliente tiene muchas facturas, y la factura es de un cliente
- Al ser bidireccional ambas clases conocen la existencia de la otra, son navegables

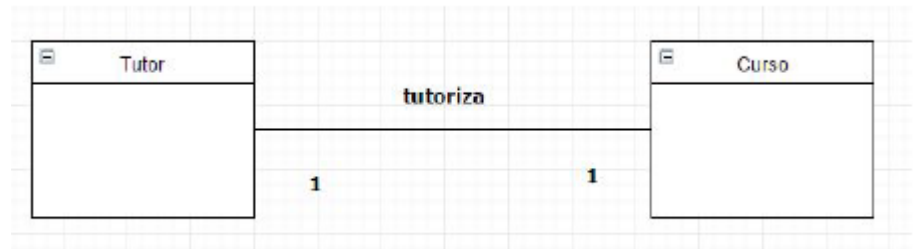
Asociación. Ejemplo

- un Departamento emplea a muchas Personas y una Persona pertenece a un Departamento.
- En este caso, es bidireccional y ambas clases conocen su existencia, ambas clases son navegables



Asociación. Ejemplo

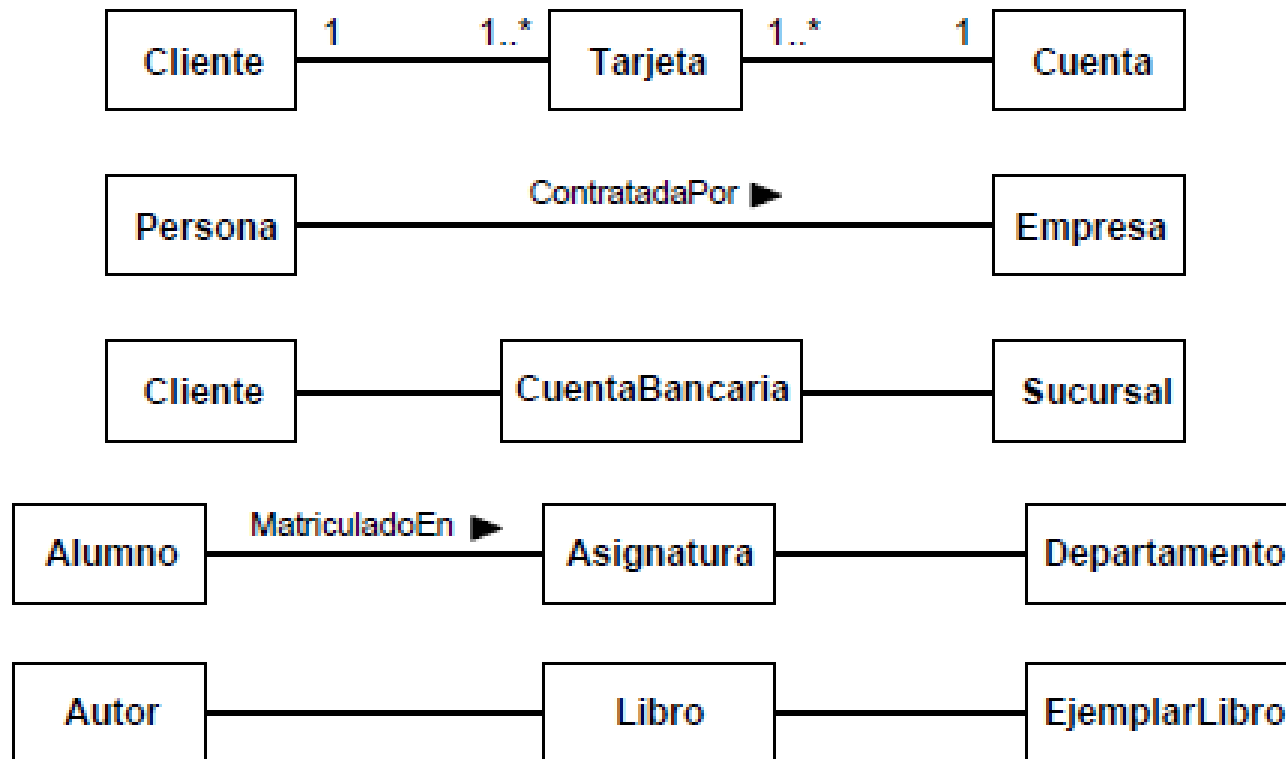
- Un tutor tutoriza a un curso y un curso es tutorizado por un profesor, es una relación 1 a 1



- Asociación distribuyen: Un almacén distribuye Artículos en varias zonas. La asociación es unidireccional: sólo la clase Almacen conoce la existencia de la clase destino

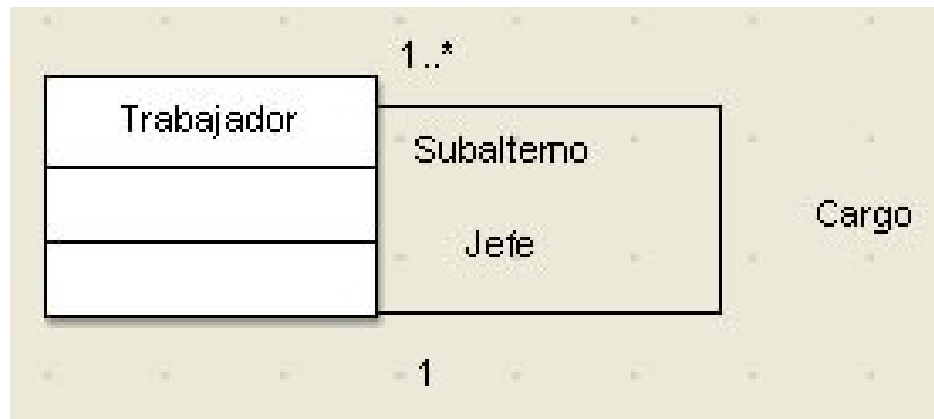


Asociación. Ejercicio



Asociación reflexiva

- Un trabajador jefe tiene a su cargo muchos subalternos y un subalterno tiene un único jefe

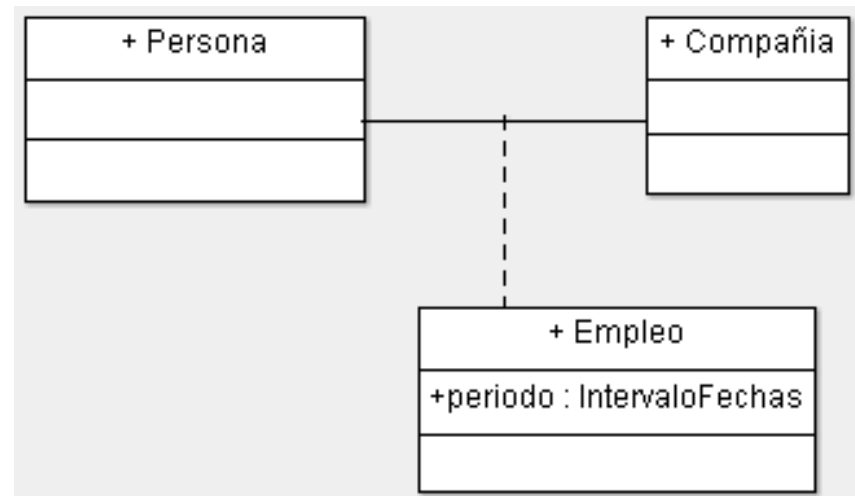


Clase asociativa

- Cuando una asociación entre dos clases lleva información necesaria para esa asociación
 - ▣ La clase asociativa recibe el estatus de clase y sus instancias son elementos de la asociación

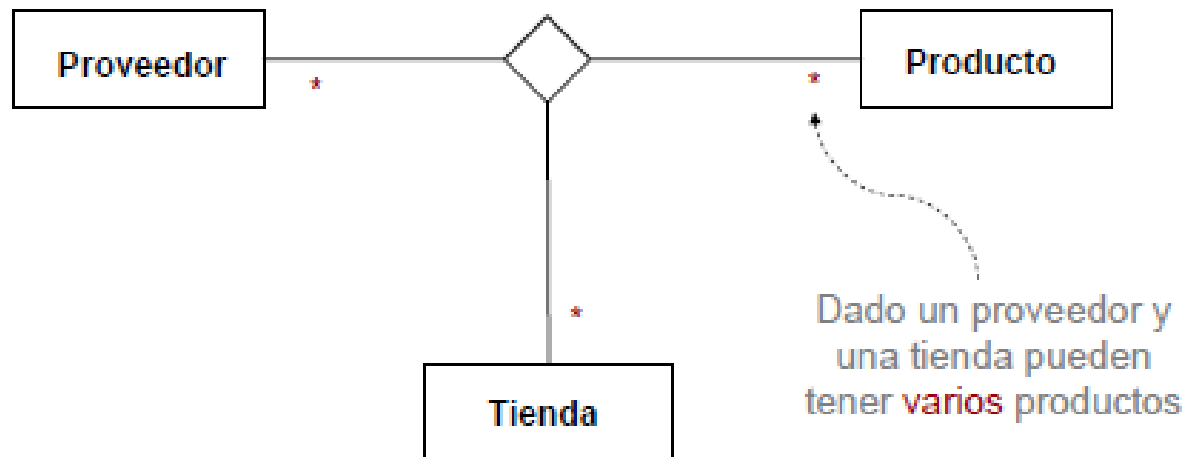
Ejemplo.

Una Persona tiene un Empleo en una Compañía, la Compañía emplea muchas Personas, y del Empleo se necesita saber el periodo en el que está contratado

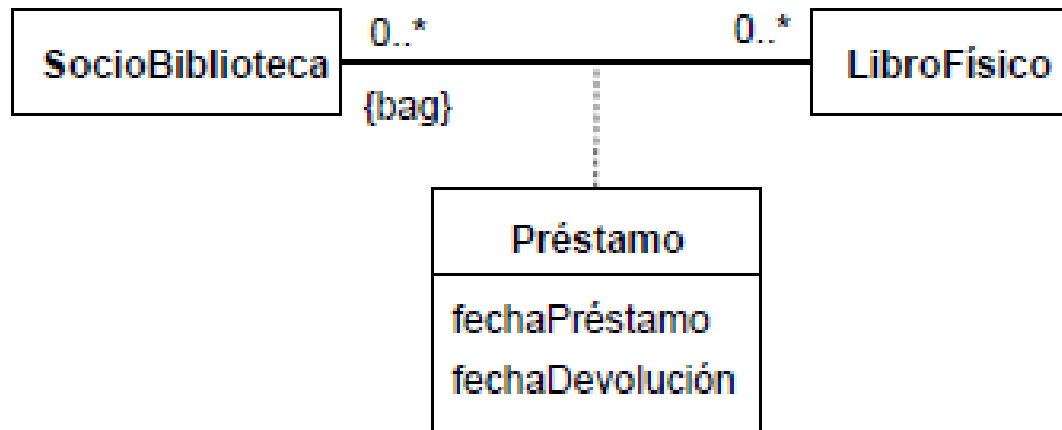


Asociaciones n-arias

- Asociación entre más de dos clases
- La representamos con un rombo



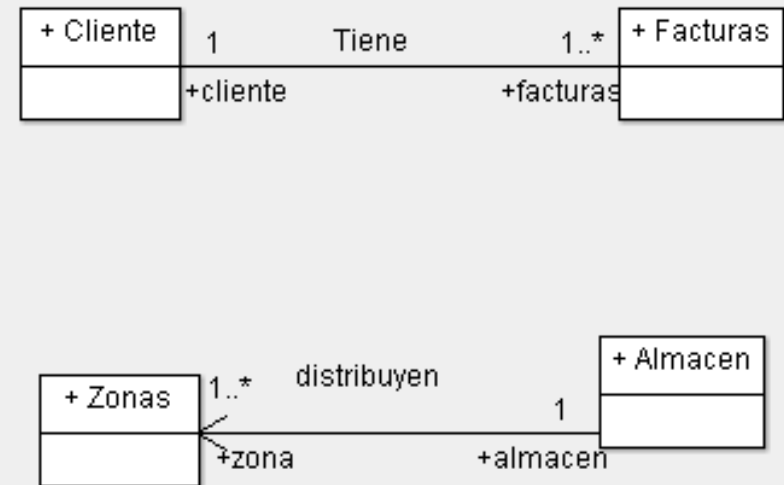
Ejemplo. Préstamo con UML 2.0



- un préstamo es la asociación entre un único socio y un único libro (por la propia definición de clase asociativa).
 - un socio puede haber cogido prestado varios libros.
 - un libro ha podido ser cogido prestado por socios varias veces.
 - un socio puede haber cogido prestado un mismo libro varias veces (correspondiendo a préstamos distintos).
-

Para ampliar... De UML a Java

- Si se convierten a Java dos clases unidas por una asociación bidireccional, cada una de las clases tendrá un objeto o colección de objetos, dependiendo de la multiplicidad entre ellos
- En cambio, en la asociación unidireccional, la clase destino no sabrá de la existencia de la clase origen, y la clase origen contendrá un objeto o colección de objetos de la clase destino



Para ampliar... De UML a Java

```
import java.util.Vector;
```

```
public class Ciente {
```

```
* @element-type Facturas  
public Vector facturas;
```

```
}
```

```
public class Facturas {
```

```
* @element-type Cliente  
    public Cliente cliente;
```

```
}
```

```
import java.util.Vector;
```

```
public class Almacen {
```

```
* @element-type Zona  
public Vector zona;
```

```
}
```

```
public class Zonas {
```

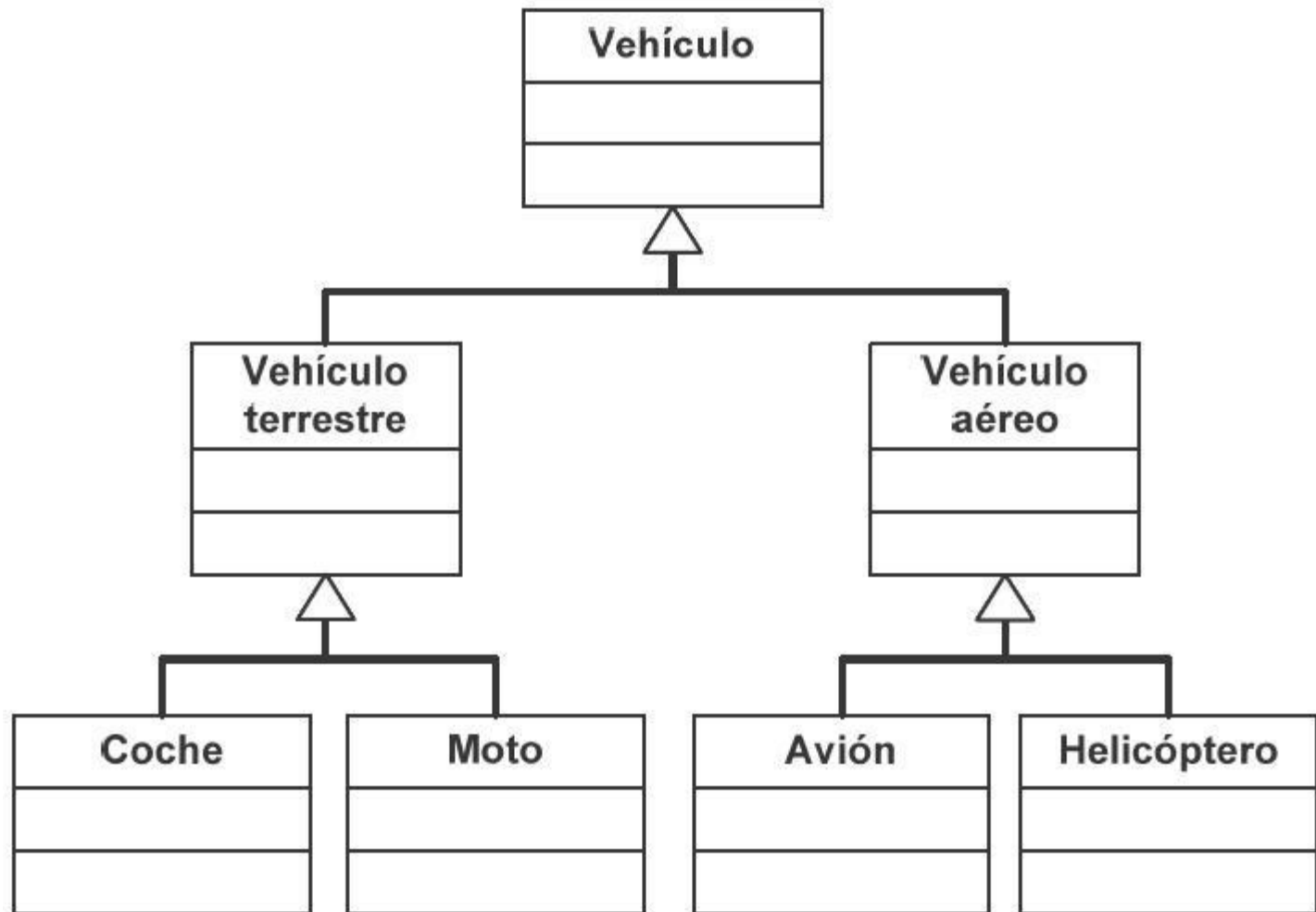
```
}
```

Almacén a zonas es navegable.
Al ser la multiplicidad de zonas de 1..*, en almacen se añade un vector de objetos

Herencia (generalización y especialización)

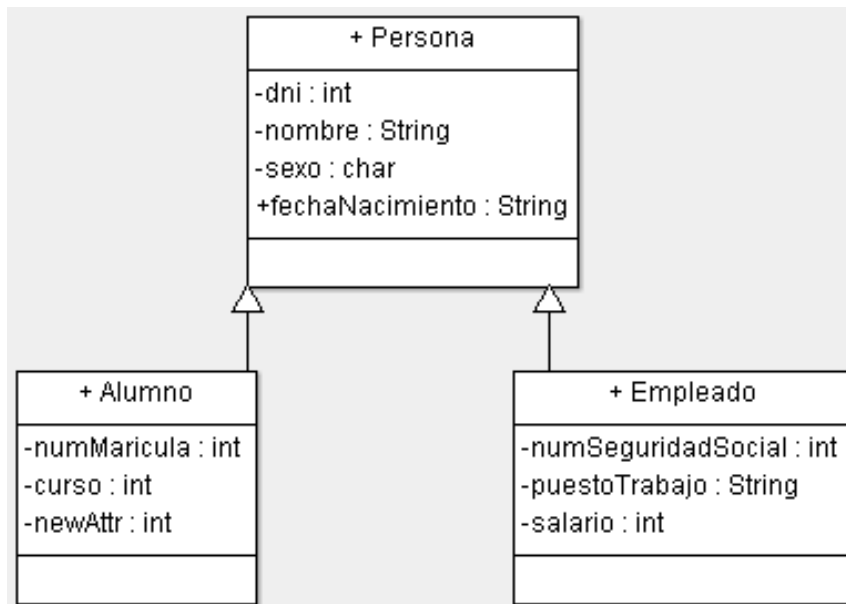
- La herencia nos permite compartir similitudes entre clases
 - ▣ Clase con atributos y métodos comunes → Superclase
 - ▣ Clases más refinadas → Subclases
- La superclase **generaliza** a sus subclases y las subclases **especializan** a la superclase
- La herencia se representa con una flecha donde el extremo de la flecha apunta a la superclase

Herencia. Ejemplo



Herencia. Ejemplo

- Todas las clases comparten los atributos y métodos de la clase Persona



```
public class Persona {
    private int dni;
    private String nombre;
    private char sexo;
    public String fechaNacimiento;
}

public class Alumno extends Persona {
    private int numMaricula;
    private int curso;
}

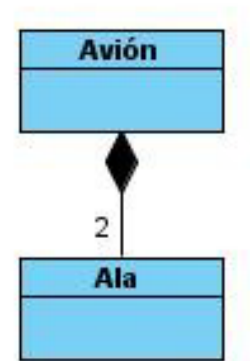
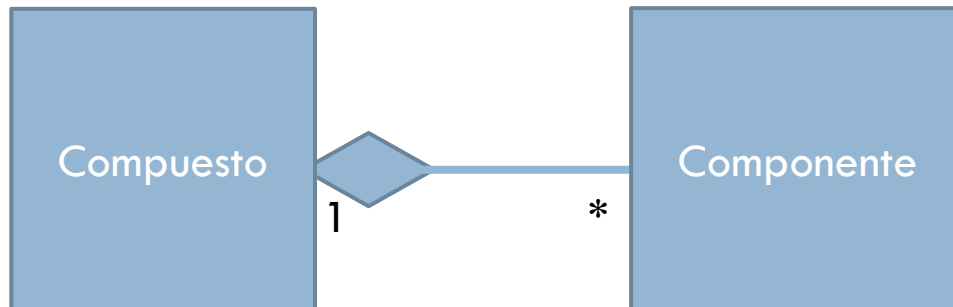
public class Empleado extends Persona
{
    private int numSeguridadSocial;
    private String puestoTrabajo;
    private int salario;
}
```

Composición

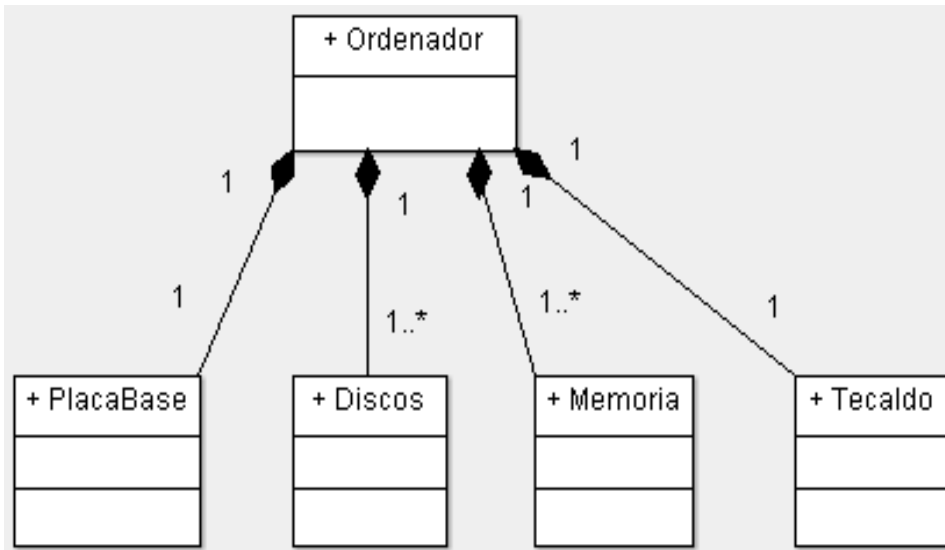
- Cuando un objeto está compuesto por otros objetos
 - ▣ Tenemos un objeto y sus componentes
- Tenemos dos formas de composición:
 - ▣ fuerte o **composición**
 - ▣ débil o **agregación**

Composición

- En la composición:
 - ▣ los componentes constituyen parte el objeto compuesto y no pueden ser compartidos por varios objetos compuestos
 - ▣ La supresión del objeto compuesto implica la de sus componentes



Composición. Ejemplo



```
import java.util.Vector;

public class Ordenador {
    public PlacaBase myPlacaBase;
    public Vector<Discos> myDiscos;
    public Vector <Memoria> myMemoria;
    public Teclado myTeclado;

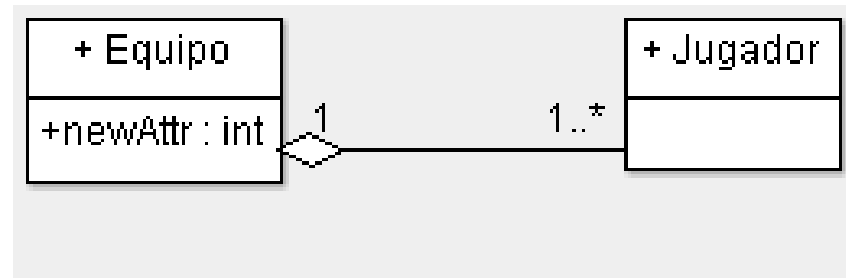
    public Ordenador(){
        myPlacaBase=new PlacaBase();
        myDiscos=new Vector<>();
        myMemoria=new Vector<>();
        myTeclado=new Teclado();
    }
}
```

Agregación

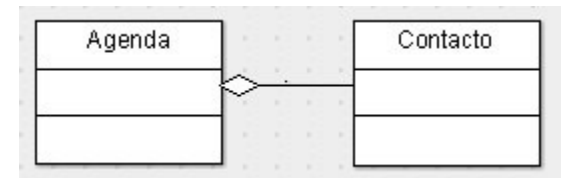
- Es la composición débil
- En este caso los componentes pueden ser compartidos por varios compuestos
- Se da con mayor frecuencia que la composición
 - ▣ En el modelado se puede iniciar el análisis estableciendo composición y luego refinar si es agregación

Agregación. Ejemplo


Un equipo está compuesto por jugadores, sin embargo, el jugador puede jugar también en otros equipos. Si desaparece el equipo, el jugador no desaparece.



Una Agenda está compuesta por Contactos , sin embargo, los Contactos pueden estar también en otras Agendas. Si desaparece la Agenda, el Contacto no desaparece



Diferencias entre agregación y composición

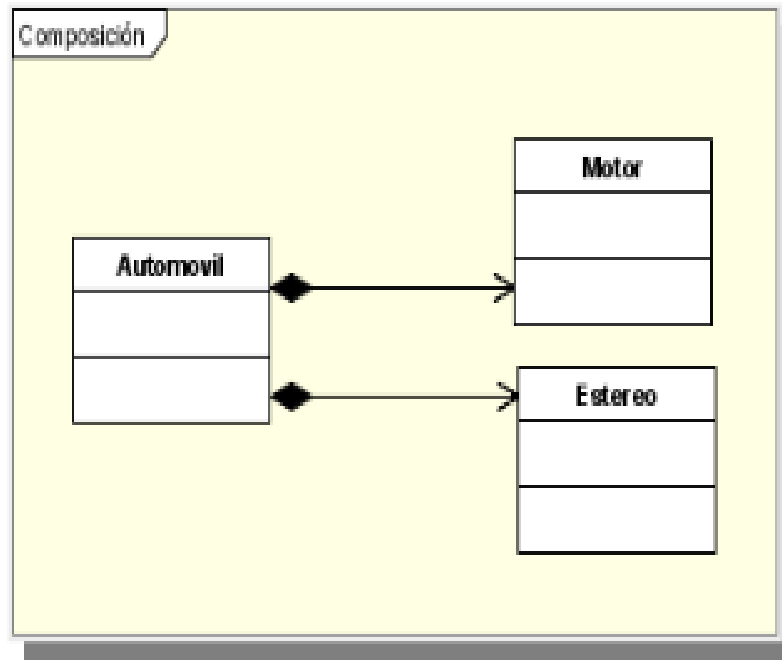
	Agregación	Composición
Símbolo		
Varias asociaciones comparten los componentes	SI	NO
Destrucción de los componentes al destruir el compuesto	No	SI
Cardinalidad del compuesto	Cualquiera	0..1 o 1

De UML a Java

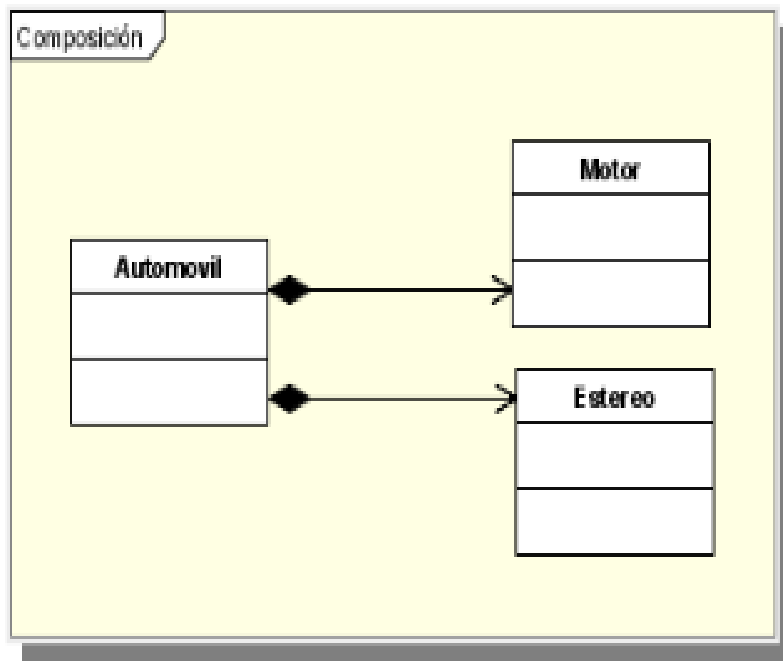
- La forma de traducir ambos tipos de relación a código es tener un atributo en la clase compuesta donde almacenaremos una colección de los objetos que la componen
 - ▣ podremos utilizar diferentes estructuras de datos que nos permitan almacenar la colección de objetos: arrays, listas...
 - ▣ Además debemos de proporcionar un método para agregar elementos a la colección

De UML a Java. Actividad

- ¿Cómo sería el código java para el siguiente ejemplo de composición?



De UML a Java. Actividad. Solución



```
public class Automovil {

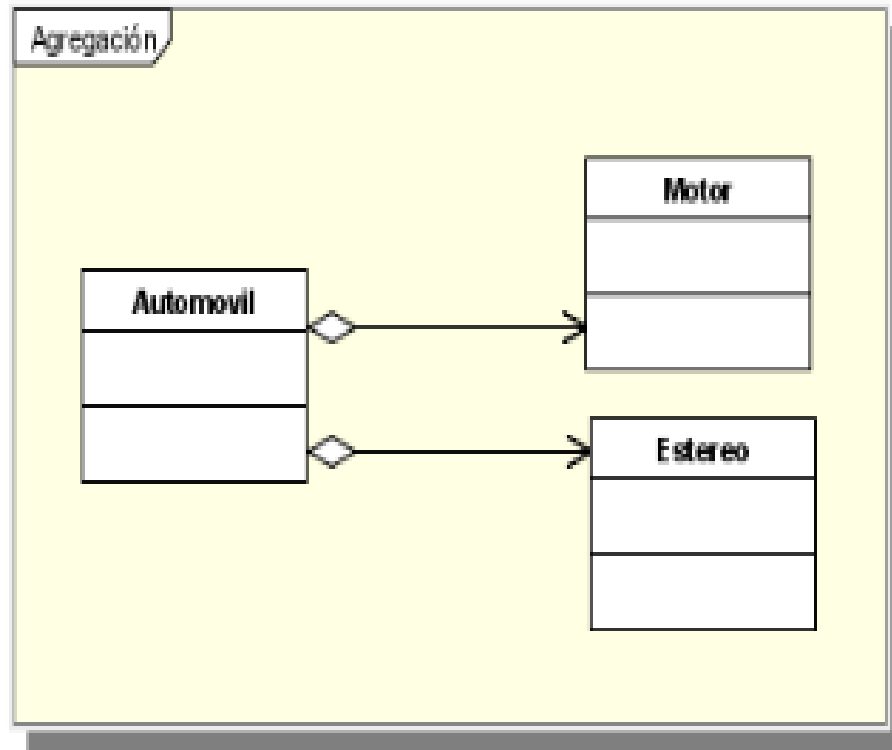
    public Estereo estereo;
    public Motor motor;

    public Automovil() {
        estereo = new Estereo();
        motor = new Motor();
    }

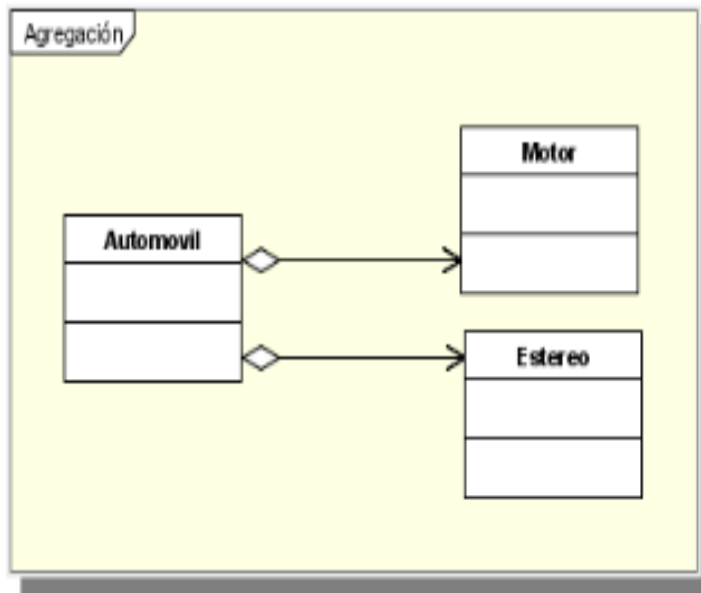
}
```

De UML a Java. Actividad

- ¿y si fuera una relación de Agregación?



De UML a Java. Actividad. Solución

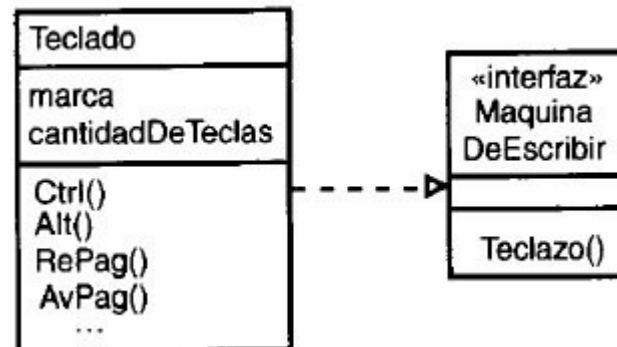


```
public class Automovil {  
  
    public Estereo estereo;  
    public Motor motor;  
  
    public Automovil() {  
    }  
  
    public void ensamblar(Estereo e, Motor m) {  
        estereo = e;  
        motor = m;  
    }  
}
```

Si los componentes no desaparecen al desaparecer el compuesto, sería más correcta esta implementación

Realización

- Una realización es la relación de herencia existente entre una clase interfaz y la subclase que implementa esa interfaz
 - ▣ **RECORDAR QUE..** Una **interfaz** es una clase totalmente abstracta, es decir, no tiene atributos y todos sus métodos son abstractos y públicos, sin desarrollar. Estas clases no desarrollan ningún método. Gráficamente se representan como una clase con el estereotipo “interface”.
- Se representa mediante una flecha con línea discontinua en lugar de una línea completa.




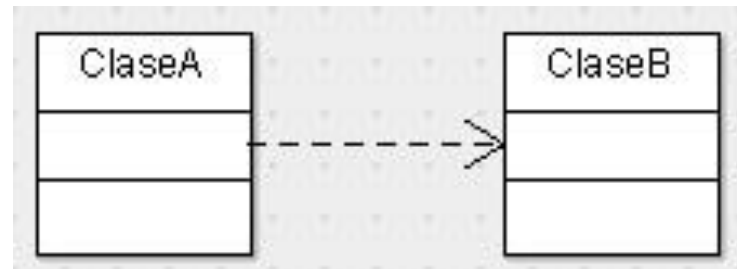
Realización. Ejemplo

- Se considera que cualquier animal come, se comunica y se reproduce, sin embargo cada tipo de animal lo hace de manera diferente. Cada subclase implementará los métodos de la interfaz.

<pre>public interface Animal { public void comer(); public void comunicarse(); public void reproducirse(); }</pre>	<pre>public class Calamar implements Animal { public Calamar(){ } public void reproducirse(){ } public void comunicarse(){ } public void comer(){ } }</pre>
<pre>public class Perro implements Animal { public Perro(){ } public void reproducirse(){ } public void comunicarse(){ } public void comer(){ } }</pre>	<pre>public class Gallina implements Animal { public Gallina(){ } public void reproducirse(){ } public void comunicarse(){ } public void comer(){ } }</pre>

Dependencia

- Se establece entre dos clases cuando una clase usa a la otra, es decir la necesita para su cometido
- Esta relación es la más básica entre clases y comparada con los demás tipos de relación, la mas débil.
- Se representa con una flecha sin relleno discontinua  que va desde la clase utilizadora a la clase utilizada



Dependencia. Ejemplo

- Relación clase Impresora y clase Documento. La impresora imprime documentos, necesita el documento para imprimirlo.
- Relación clase Viajero y Equipaje. El viajero necesita su equipaje para viajar (el viajero depende de la clase Equipaje porque la necesita)

