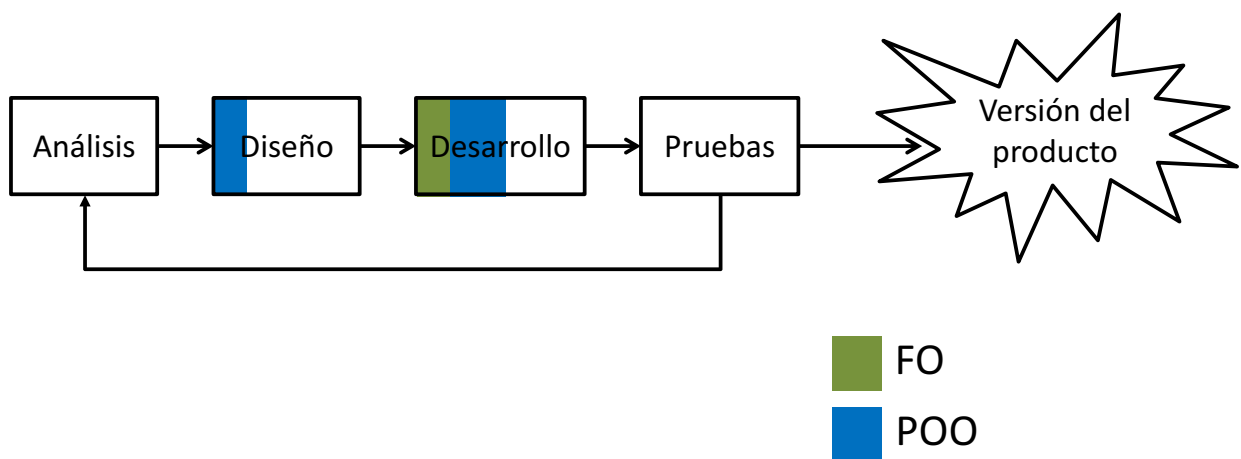


UML: Diagramas de clases

Programació Orientada a Objectes

El proceso de creación de software (un tanto simplificado)



Contenidos

1. Presentación de la asignatura
2. Introducción al concepto de Programación Orientada a Objetos y a Java
3. Clases y objetos en Java
 - Atributos y métodos
 - Constructores
4. Encapsulación de datos
5. Atributos de clase. Definición de constantes
6. Contenedores
7. Lenguaje Unificado de modelado: Diagramas de clase
8. Herencia
9. Polimorfismo

--- Examen parcial ---

10. Excepciones
11. Entrada/Salida de datos

Programación Java

Diseño

UML

- UML: *Unified Modeling Language* (Lenguaje Unificado de Modelado)
- Estándar para visualizar, especificar, construir y documentar un sistema.
 - Los “planos” de un arquitecto del software!
- UML especifica diferentes tipos de diagramas para describir el sistema desde diferentes puntos de vista
 - Diagramas de clases
 - Diagramas de componentes
 - Diagramas de secuencia
 - Diagramas de objetos
 - etc, etc etc...
- Antes de programar un sistema de cierta envergadura, es imprescindible diseñarlo y modelarlo en UML.

Diagramas de clases

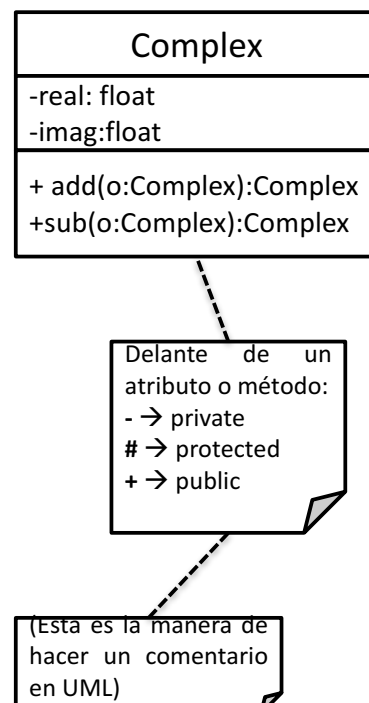
- Un tipo de diagrama UML que permite visualizar gráficamente las clases de un programa, así como las relaciones entre estas.
- Información de una clase
 - Nombre
 - Atributos
 - Tipo: int, String, short, char...
 - Nombre
 - Visibilidad: privada, protegida, pública
 - Otros modificadores: static, final...
 - Métodos
 - Tipo de retorno
 - Nombre
 - Parámetros
 - Otros modificadores: static, final...
- Un diagrama de clases pretende dar una visión rápida y concisa del sistema
 - no es necesario que tenga tanto detalle como un código java
 - especificaremos aquellas clases, atributos, métodos y relaciones que sean de relevancia a la hora de describir la solución

Ejemplo

• Java

```
public class Complex {  
    private float real, imag;  
    public Complex add(Complex o) {  
        Complex ret = new Complex();  
        ret.real = real + o.real;  
        ret.imag = imag + o.imag;  
        return ret;  
    }  
    public Complex sub(Complex o) {  
        Complex ret = new Complex();  
        ret.real = real - o.real;  
        ret.imag = imag - o.imag;  
        return ret;  
    }  
}
```

• UML



Especificación de atributos y métodos en UML

- Atributos
 - ó # ó + nombreAtributo : TipoAtributo
- Métodos
 - ó # ó + nombreMétodo(nombreParámetro1 : TipoParámetro1, ... , nombreParámetroN : TipoParámetroN) : TipoRetorno
- Atributos o métodos “static”: subrayado
- Ejemplos

protected String nombre → #nombre: String

public static void main(String[] args) →

+main(args:String[]):void

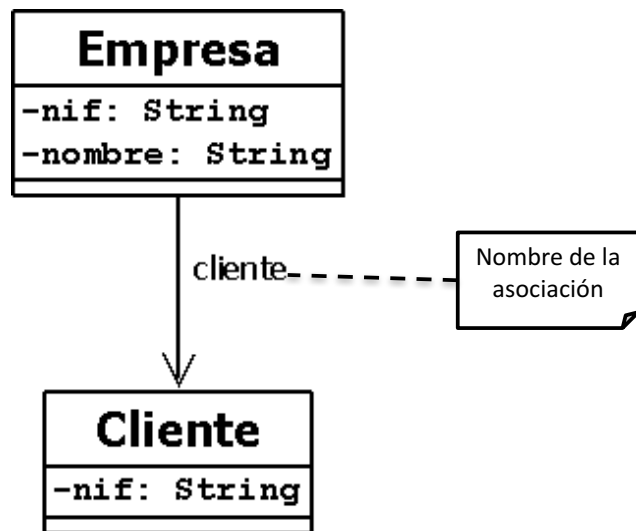
Relaciones entre clases

- Una clase por sí sola no tiene mucha utilidad si no la relacionamos con otras clases.
- El verdadero poder de los diagramas de clases es mostrar las relaciones entre las clases de un sistema/software.
- Varios tipos
 - Asociación
 - Dependencia
 - Herencia/generalización

Asociación (I)

```
public class Empresa {  
    private String nombre;  
    private String nif;  
    private Cliente cliente;  
}
```

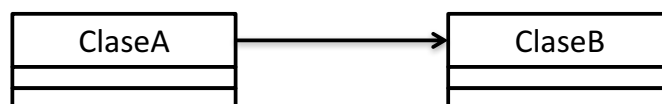
```
public class Cliente {  
    private String nif;  
}
```



Navegabilidad

- Propiedad que indica que es posible navegar desde un objeto de una clase al objeto de la otra clase con la que está relacionada.

- Unidireccional: ClaseA tiene un atributo del tipo ClaseB



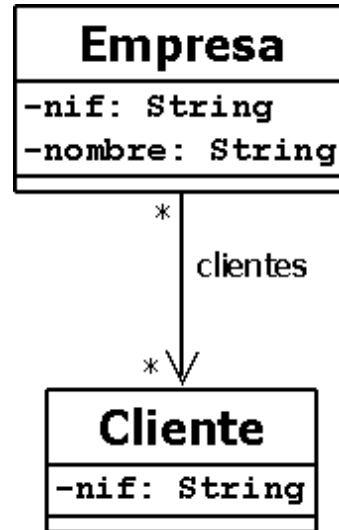
- Bidireccional: ClaseA tiene un atributo del tipo ClaseB, y ClaseB tiene un atributo del tipo ClaseA



Asociación (II) : cardinalidad

- En realidad, una empresa no tiene un solo cliente, y un cliente puede serlo de varias empresas. Esto lo indicaremos en cada extremo de la asociación.

```
public class Empresa {  
    private String nombre;  
    private String nif;  
    private Set<Cliente> clientes;  
}  
  
public class Cliente {  
    private String nif;  
}
```



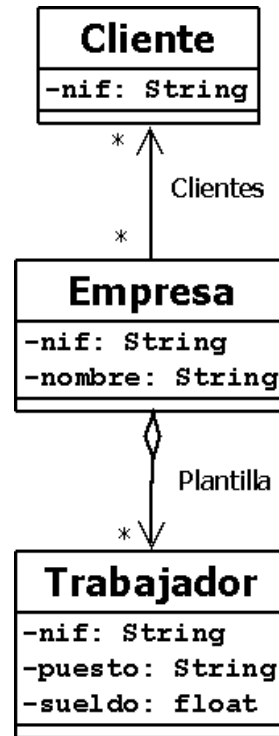
Cardinalidad

- Ausencia de indicación: cardinalidad de 1.
- 8: exactamente 8.
- 4,6,9: exactamente 4, 6 o 9.
- * : 0 o varios (sin determinar límite superior).
- Rangos: utilizan “..”; Ej: 2..4 (entre 2 y 4)
- En un rango, “*” indica varios siempre (sin límite superior)
- 0..1: uno o ninguno.
- 1..* : 1 o varios (sin determinar límite superior).

Asociación (III): Agregación

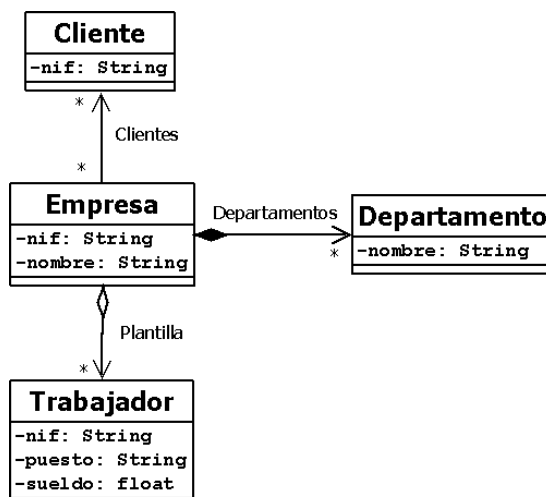
- Tipo de relación que indica que un extremo es “parte” del otro.

```
public class Cliente {  
    private String nif;  
}  
  
public class Trabajador {  
    private String nif;  
    private String puesto;  
    private float sueldo;  
}  
  
public class Empresa {  
    private String nombre;  
    private String nif;  
    private Set<Cliente> clientes;  
    private Set<Trabajador> plantilla;  
}
```



Asociación (IV): Composición

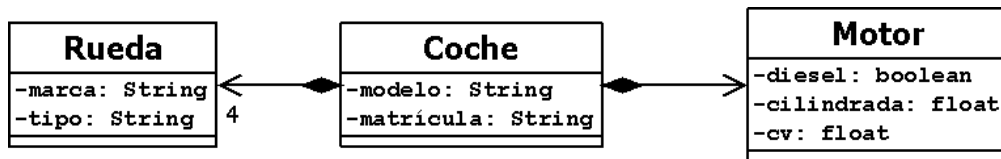
- También se llama “Agregación Fuerte”.
- Los elementos no solo son parte de una clase, sino que deben su existencia a ésta.
- Si la clase que contiene a otra mediante composición fuera eliminada, todos los elementos que la componen también se eliminarían.



```
public class Empresa {  
    private String nombre;  
    private String nif;  
    private Set<Cliente> clientes;  
    private Set<Trabajador> plantilla;  
    private Set<Departamento> departamentos;  
}
```

¡OJO!

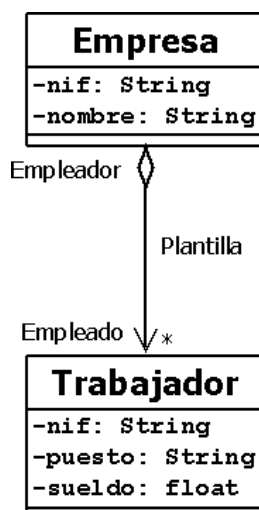
- La de agregación o composición no implica necesariamente una cardinalidad de uno a muchos, o de muchos a muchos.



- Una cardinalidad a muchos se puede implementar de diferentes maneras:
 - Array estático
 - `Rueda ruedas[] = new Rueda[4];`
 - Conjunto
 - `HashSet<Rueda> ruedas = new HashSet<Rueda>();`
 - Lista
 - `ArrayList<Rueda> ruedas = new ArrayList<Rueda>();`

Asociación (V): rol de las clases

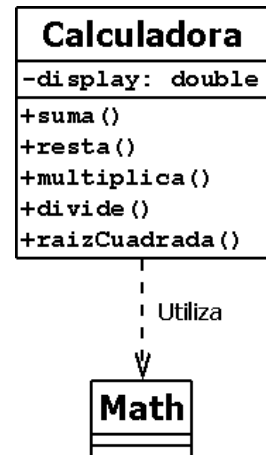
- A veces se puede especificar el rol que cada clase tiene en la asociación, para dar más información



Dependencia

- La clase X usa la clase Y, pero Y no es ningún atributo de X, ni forma parte de ésta.
- Ejemplos:
 - X usa un método estático de Y.
 - Y es un parámetro de un método de X.
 - Y es una variable local de algún método de X.

```
public class Calculadora {  
    double display;  
    //...  
    void raizCuadrada() {  
        display = Math.sqrt(display);  
    }  
}
```



Ejercicio

- Hacer el diagrama UML del siguiente código Java.

```
public class Departamento {  
    protected String nombre;  
    private Set<Medico> plantilla;  
}  
public class Paciente {  
    private String nombre;  
    private Medico medico;  
}  
public class Clínica {  
    private String dirección;  
    private Set<Medico> plantilla;  
    private Set<Paciente> pacientes;  
    private Set<Departamento> departamentos;  
}  
Public class Medico {  
    private String nombre;  
    Departamento departamento;  
}
```