



# UML (Lenguaje Unificado de Modelado).

Diagramas de clases.

Diagramas de secuencia.

Diagrama de Casos de uso.

Correspondencia entre elementos UML y elementos de lenguajes de programación.

Editores gráficos vs. Editores UML

\*\*\*\*\*

Cómo está definido UML?

# UML

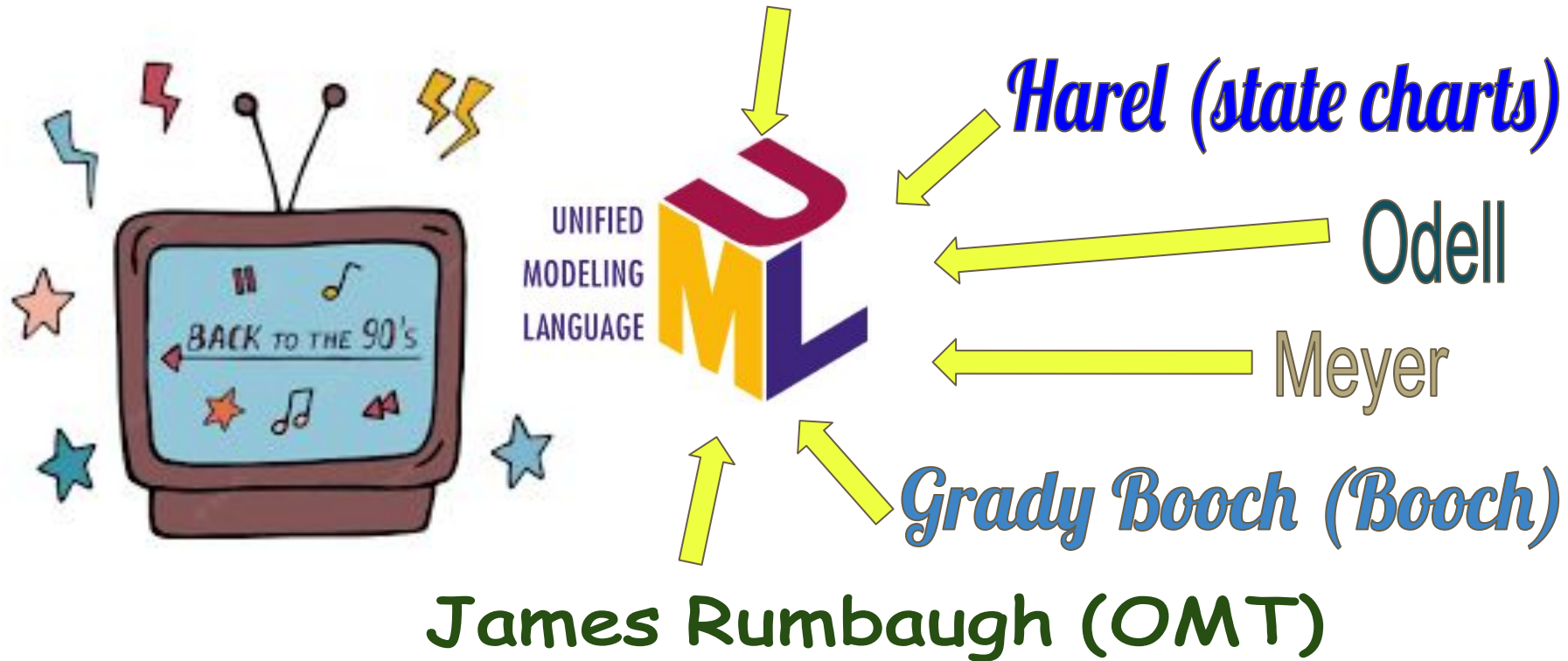
- Es un lenguaje de modelado visual que nos permite
  - especificar
  - visualizar
  - construir
  - documentar

...artefactos de un sistema de software.

- Permite capturar decisiones y conocimientos.

# Historia de UML

## Ivar Jacobson (Objectory)



# UML - Lenguaje de Modelado UNIFICADO

Grady Booch

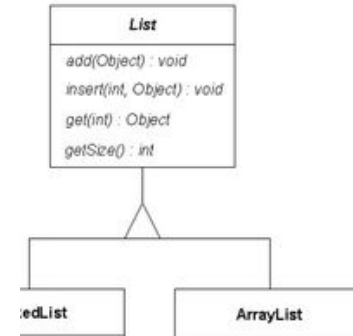
- Método Booch

Jim Rumbaugh

- Object-Modeling Technique (OMT) de Rumbaugh

Ivar Jacobson

- OOSE (Object-oriented Software Engineering)



# UML 2.5



**¿ Qué puedo hacer con UML?**

# Lenguaje UML



# Lenguaje UML

- Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software.
- 
- Permite capturar decisiones y conocimientos.

# Diagramas de estructura

- Diagrama de Clases
- Diagrama de Paquetes
- Diagrama de Componentes
- Diagrama de Objetos
- Diagrama de Despliegue

# Diagramas de estructura

- **Diagrama de Clases**
- **Diagrama de Paquetes**
- Diagrama de Componentes
- **Diagrama de Objetos**
- Diagrama de Despliegue

# Diagramas de Comportamiento

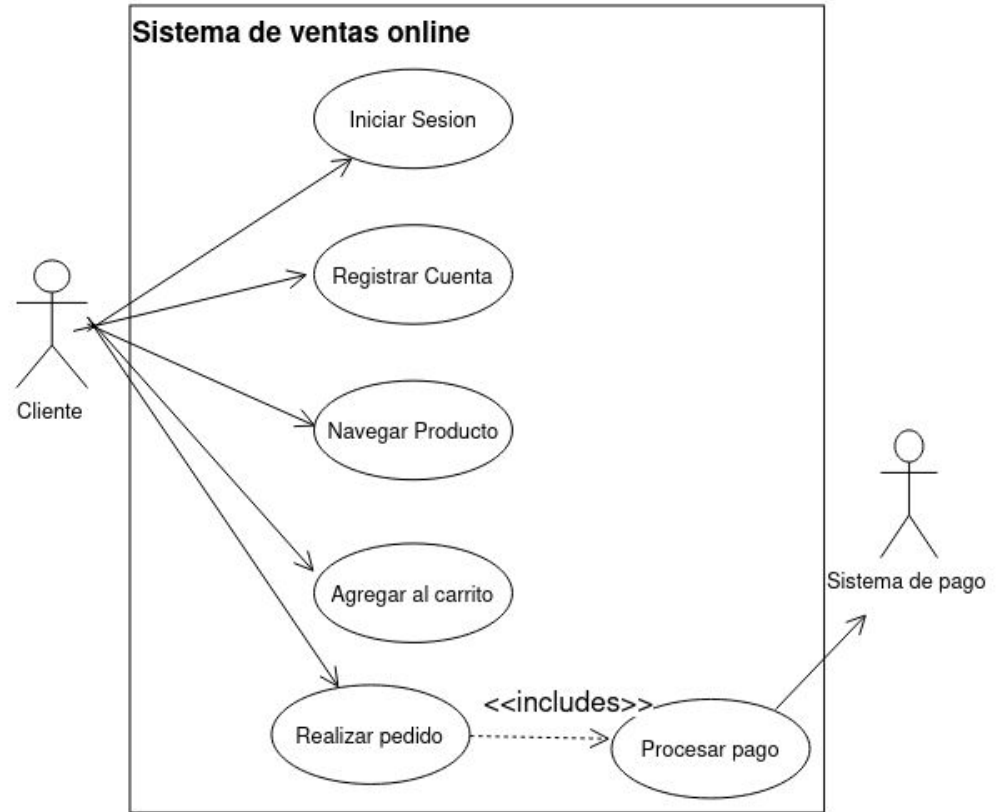
- Diagrama de Casos de Uso
- Diagramas de Interacción
  - Diagrama de Secuencia
  - Diagrama de Colaboración
- Diagrama de Máquinas de estado
- Diagrama de Actividades

# Diagramas de Comportamiento

- **Diagrama de Casos de Uso**
- Diagramas de Interacción
  - **Diagrama de Secuencia**
  - Diagrama de Colaboración
- Diagrama de Máquinas de estado
- Diagrama de Actividades

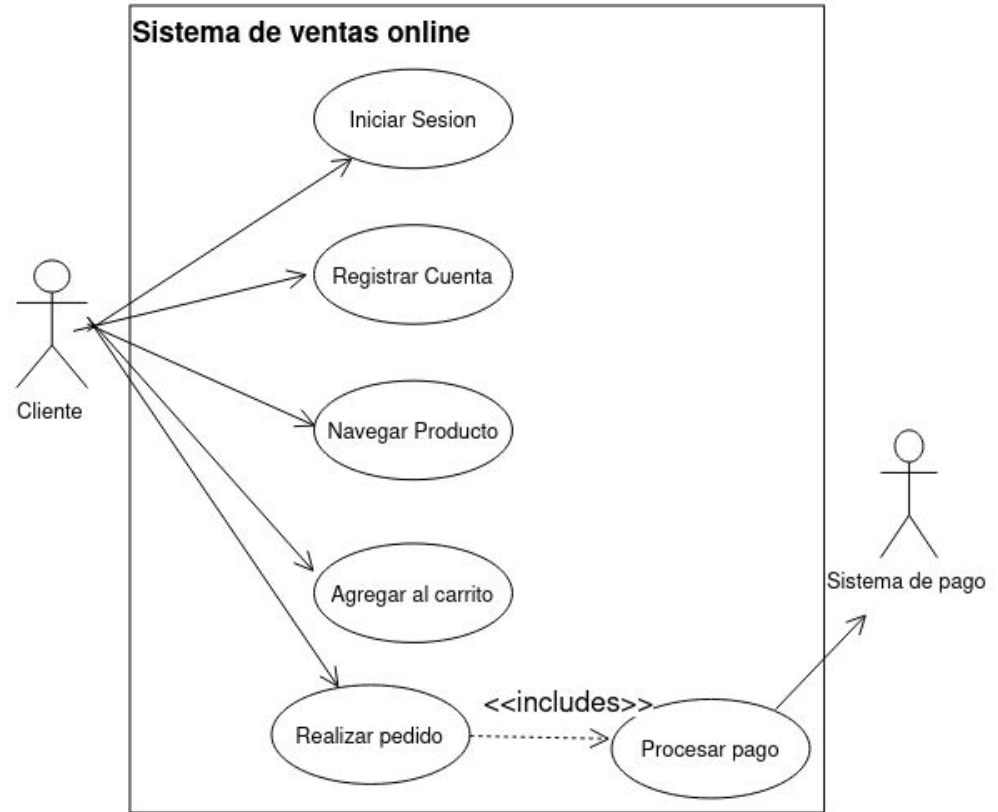
# Diagramas de Casos de Uso

- Un caso de uso es una representación del comportamiento de un sistema tal como se percibe por un usuario externo.
- Describe una interacción específica entre los actores y el sistema, proporcionando un proceso completo de cómo se utiliza el sistema en situaciones reales.



# Diagramas de Casos de Uso

- El término 'actor' engloba a personas, así como a otros sistemas que interactúan con el sistema
- Los elementos de un modelo de casos de uso son:
  - Actores
  - Casos de uso
  - Relaciones



# Diagramas de Casos de Uso



Actor: representa a un usuario externo, un sistema o una entidad que interactúa con el sistema que se está modelando.



Un caso de uso es una representación de una funcionalidad específica

Entre casos de uso pueden darse relaciones:  
extensión <<extends>>  
inclusión <<includes>>



# Diagramas de Casos de Uso - Relaciones

un caso de uso incluye a otro si el primero incorporará el comportamiento especificado en el segundo



un caso de uso extiende a otro si puede ser mejorado o ampliado con funcionalidad adicional en ciertos escenarios, pero no siempre se aplica.



# Diagramas de Casos de Uso - conversación

Curso Normal (Basico)			
1	Actor 1: Acción realizada por el actor		
2	Actor 2: Acción realizada por el actor	3	Acción realizada por el sistema
		N	Cuando se realiza la inclusión de otro caso de uso lo representaremos de la forma. Incluir (CU_identificador. CU_Nombre)
	<i>&lt;&lt; Se incluyen la secuencia de acciones realizadas por los actores que intervienen en el CU , se usaran, frases cortas, que describan el dialogo entre los actores y el sistema&gt;&gt;</i>  <i>&lt;&lt; Se pueden añadir referencias a elementos de un boceto del Interfaz del Usuario &gt;&gt;</i>		<i>&lt;&lt; Se incluyen la secuencia de acciones que realiza el sistema ante las acciones de los actores &gt;&gt;</i>

Cursos Alternos	
1a	Descripción de la secuencia de acciones alternas a la acción 1 del Curso Normal
1b	
	<i>&lt;&lt; Secuencia de los cursos alternos del CU &gt;&gt;</i>

acciones del actor

respuesta del sistema

# Diagramas de Casos de Uso - conversación

Comprar producto			
1-	Se indica el producto que se quiere comprar		
		2-	Se agrega el producto al carrito de compras
3.	Se procede a terminar la compra		
		4-	Se calcula el total
		5-	Se muestran las opciones de pago
6	Se elige una opción de pago		
		7-	Se crea una orden de compra, registrando el pago

**acciones del actor**

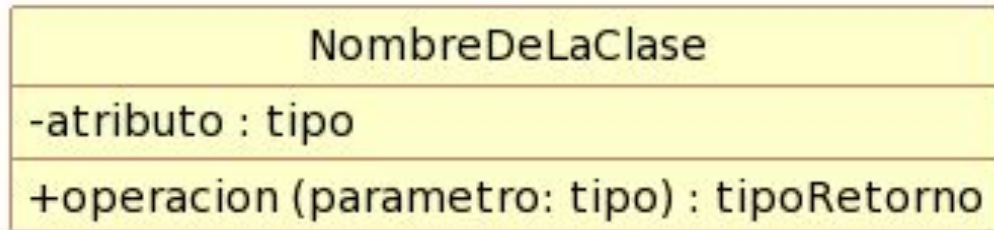
**respuesta del sistema**

# Diagrama de clases

# Diagrama de Clases

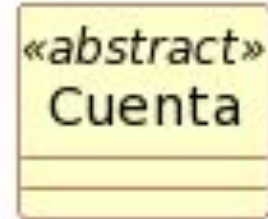
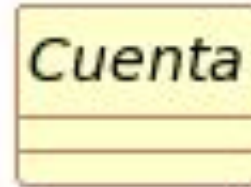
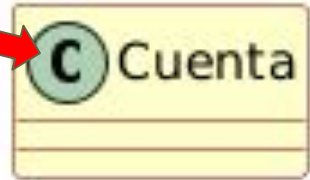
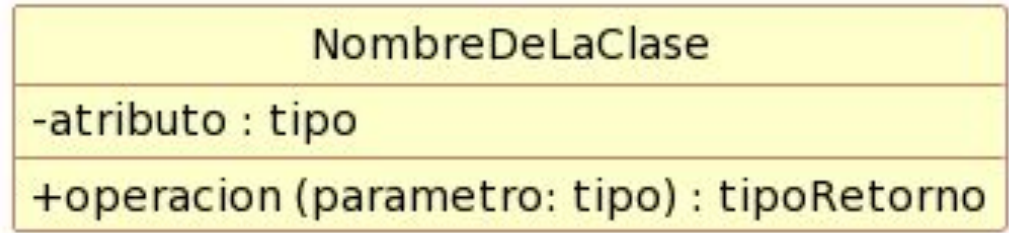
Una clase es una descripción de conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica.

Una clase es representada gráficamente por cajas con tres compartimientos



# Diagrama de Clases

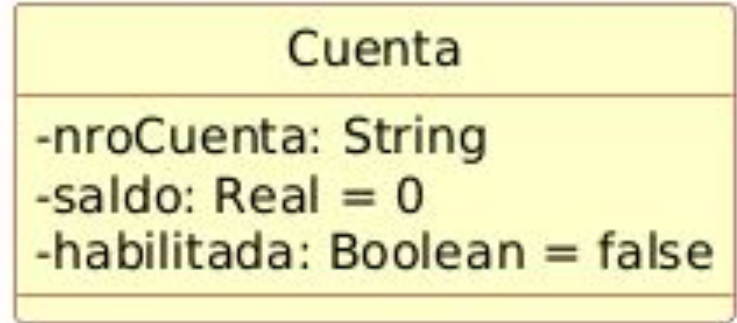
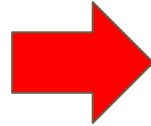
- Nombre de la clase
  - singular
  - debe comenzar con Mayúscula
  - estilo CamelCase
- Si la clase es abstracta
  - cursiva
  - estereotipo <<abstract>>



# Diagrama de Clases - Atributos

**visibilidad nombre: tipo = valor por defecto**

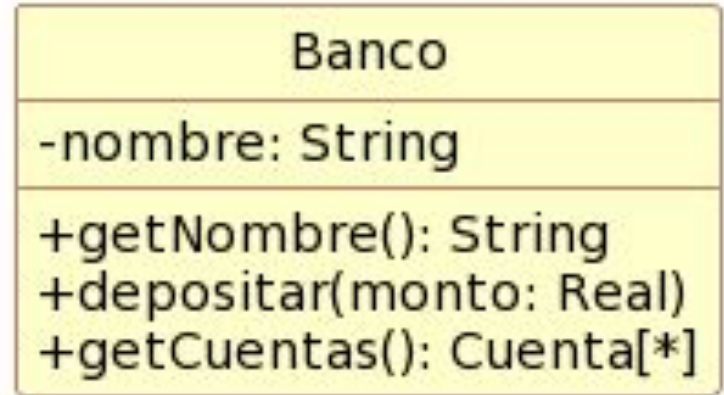
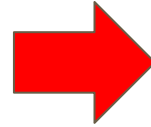
- visibilidad
  - privada (-)
  - protegida (#)
  - pública (+)
  - paquete (~)
- nombre
  - estilo camelCase
  - comienza con minúscula
- tipo
  - tipos UML: Integer, Real, Boolean, String
- valor por defecto



# Diagrama de Clases - Operaciones

**visibilidad nombre (parámetro: tipo): tipo retorno**

- visibilidad
  - privada (-)
  - protegida (#)
  - pública (+)
  - paquete (~)
- nombre
  - estilo camelCase
  - comienza con minúscula
- Parámetros
  - nombre: estilo camelCase
- tipo de retorno
  - Si no retorna nada, no se especifica
  - Si retorna un objeto, se indica de qué clase
  - Si retorna una colección, se indica el nombre de la clase [\*]

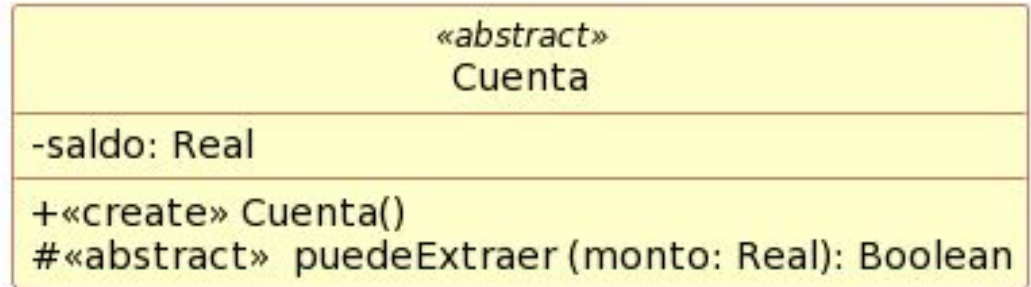
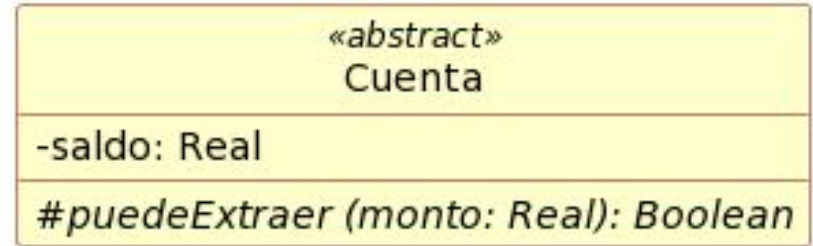




# Diagrama de Clases - Operaciones


visibilidad nombre (parámetro: tipo): tipo retorno

- 
- si el método es **abstracto**
  - *cursiva*
  - con estereotipo <<abstract>>
- si el método es un **constructor**
  - con estereotipo <<create>>



# Diagrama de Clases - Relaciones

Asociaciones 

Herencia (Generalización) 

Implementación 

Dependencia 

# Diagrama de Clases - Asociación

- **navegabilidad**
- multiplicidad
- nombre de rol
- tipo:
  - simple
  - agregación
  - composición

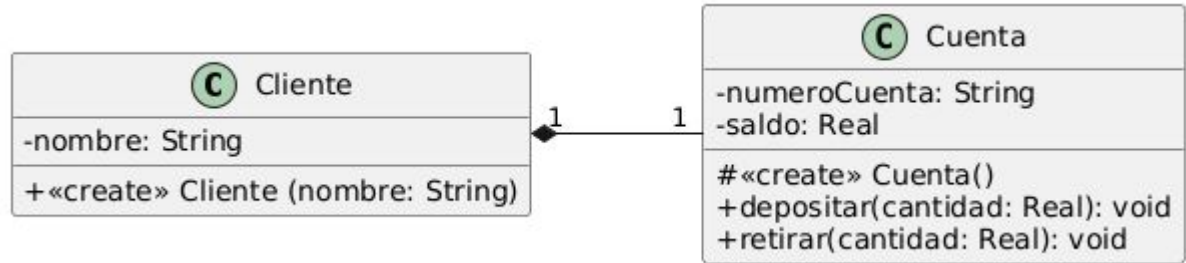


```
public class Cliente {
    private String nombre;
    private Cuenta cuenta;
}

public class Cuenta {
    private String numeroCuenta;
    private double saldo;
}
```

# Diagrama de Clases - Asociación

- **navegabilidad**
- multiplicidad
- nombre de rol
- tipo:
  - simple
  - agregación
  - composición

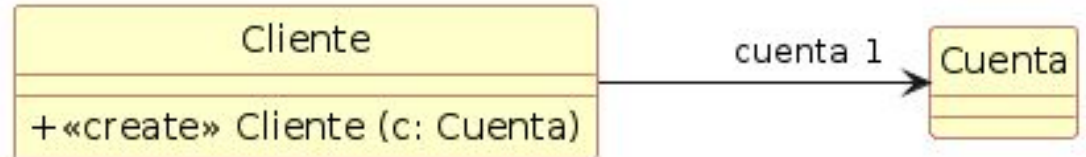
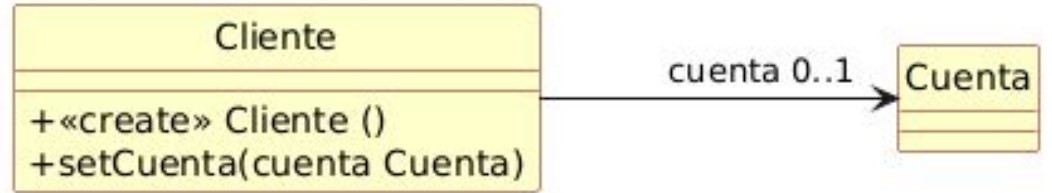


```
public class Cliente {
    private String nombre;
    private Cuenta cuenta;
}

public class Cuenta {
    private String numeroCuenta;
    private double saldo;
    private Cliente cliente;
}
```

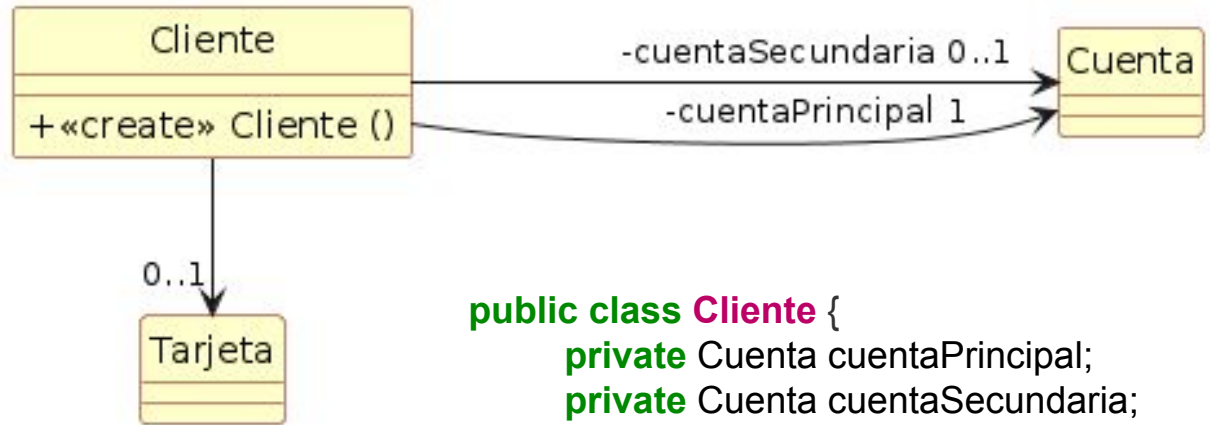
# Diagrama de Clases - Asociación

- navegabilidad
- **multiplicidad**
- nombre de rol
- tipo:
  - simple
  - agregación
  - composición



# Diagrama de Clases - Asociación

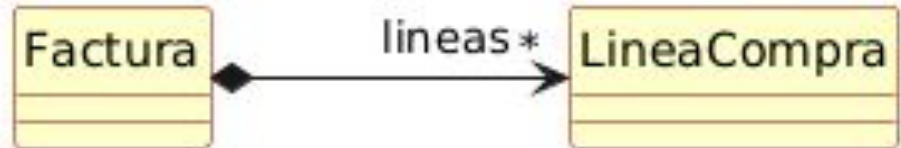
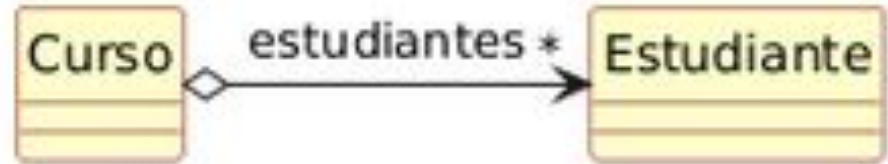
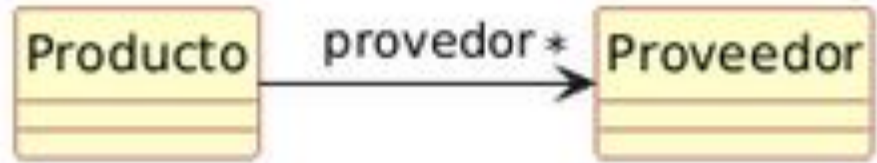
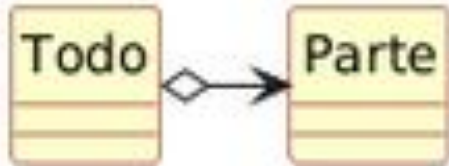
- navegabilidad
- multiplicidad
- **nombre de rol**
- tipo:
  - simple
  - agregación
  - composición



```
public class Cliente {  
    private Cuenta cuentaPrincipal;  
    private Cuenta cuentaSecundaria;  
    private Tarjeta tarjeta;  
}
```

# Diagrama de Clases - Asociación

- navegabilidad
- multiplicidad
- nombre de rol
- **tipo:**
  - simple
  - agregación
  - composición



# Diagrama de Clases - Interfaces

Una interfaz define un conjunto de operaciones que una clase o componente debe implementar.

Actúa como un contrato que establece qué métodos deben ser implementados



En UML, las interfaces se representan mediante un rectángulo con el nombre de la interfaz precedido por el estereotipo **<<interface>>**.

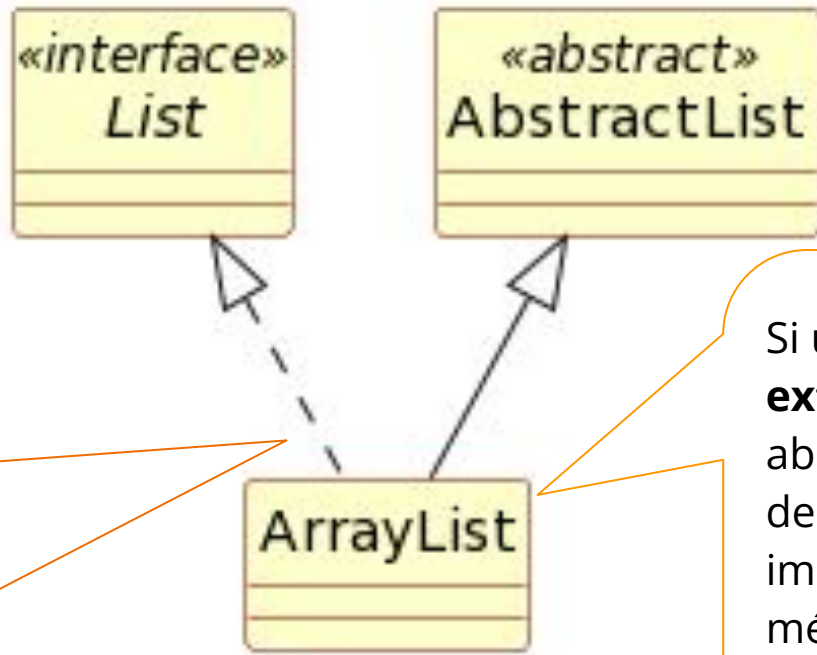
Dentro del rectángulo, se listan las operaciones o métodos que la interfaz define.

La implementación de los métodos definidos en la interfaz es responsabilidad de las clases que la implementan.



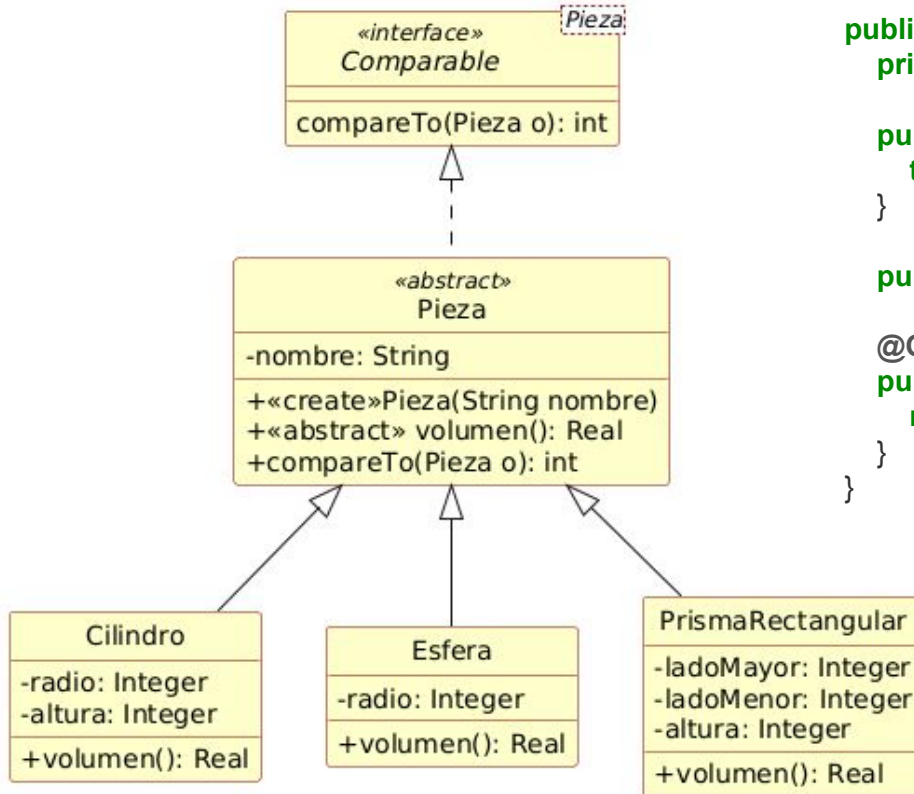
# Diagrama de Clases - Implementación y herencia

Una clase concreta **implementa** una interface. Esto significa que la clase proporciona una implementación concreta para todos los métodos (operaciones) definidos en esa interfaz.



Si una clase concreta **extiende** a una clase abstracta, significa que debe proveer las implementación de los métodos abstractos.

# Diagrama de Clases - Implementación y herencia



```
public abstract class Pieza implements Comparable<Pieza> {
    private String nombre;

    public Pieza(String nombre) {
        this.nombre = nombre;
    }

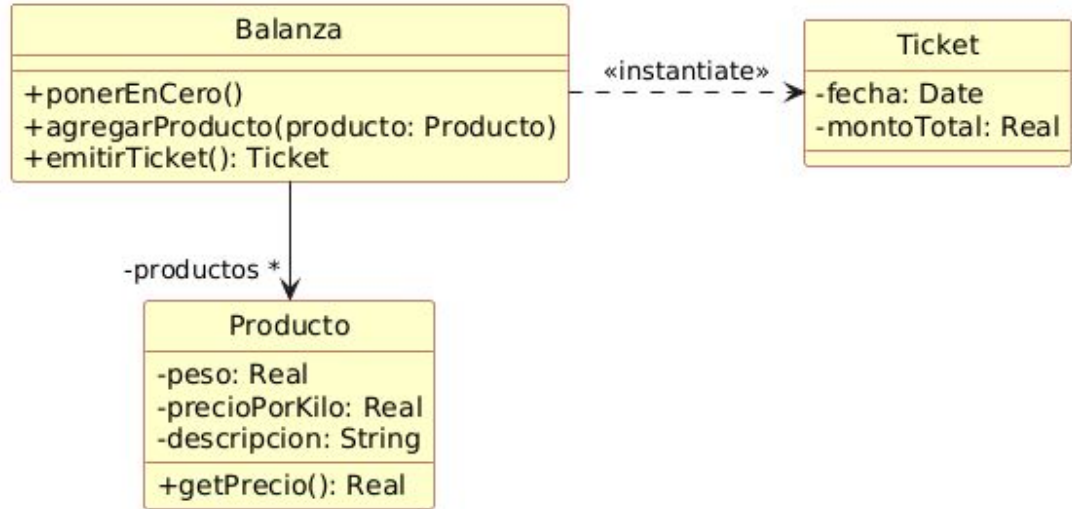
    public abstract double calcularVolumen();

    @Override
    public int compareTo(Pieza otra) {
        return Double.compare(this.calcularVolumen(), otra.calcularVolumen());
    }
}
```

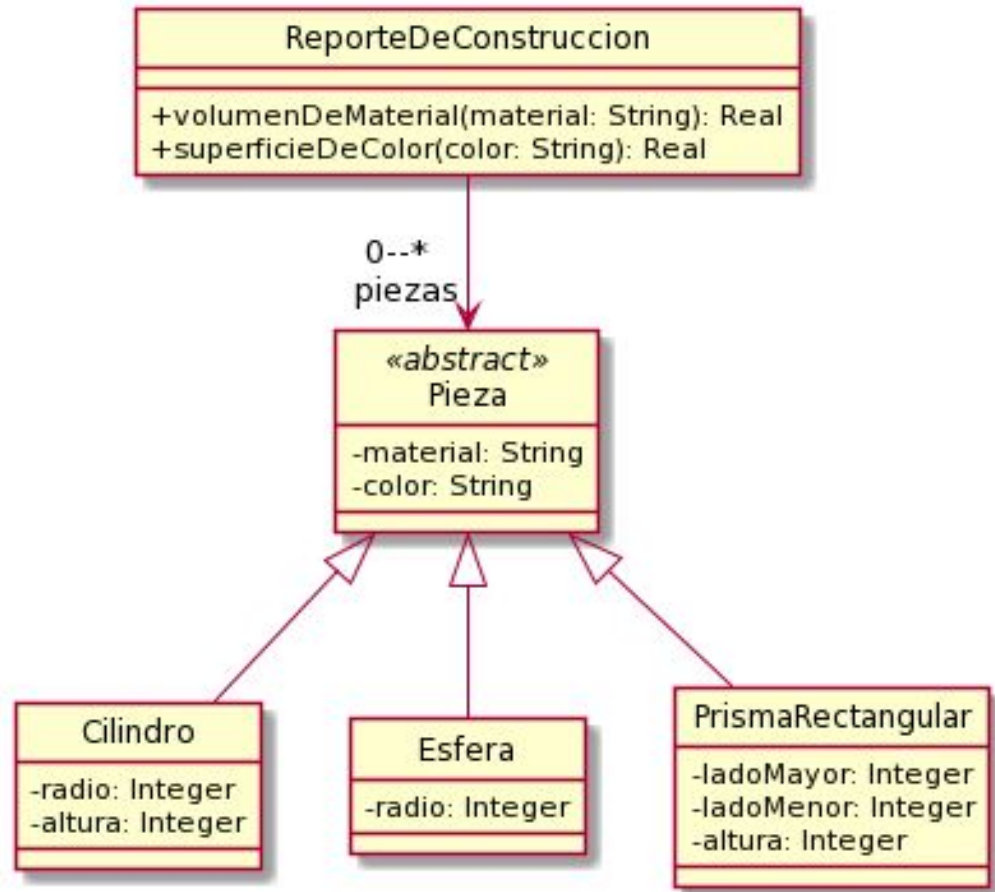
# Diagrama de Clases - Dependencia

Indica que un cambio en la especificación de una clase puede afectar a otra clase que depende de ella.

Esta relación se utiliza para mostrar que una clase o componente utiliza o necesita otro para funcionar correctamente, pero sin mantener una referencia duradera a él.



# Diagrama de Clases

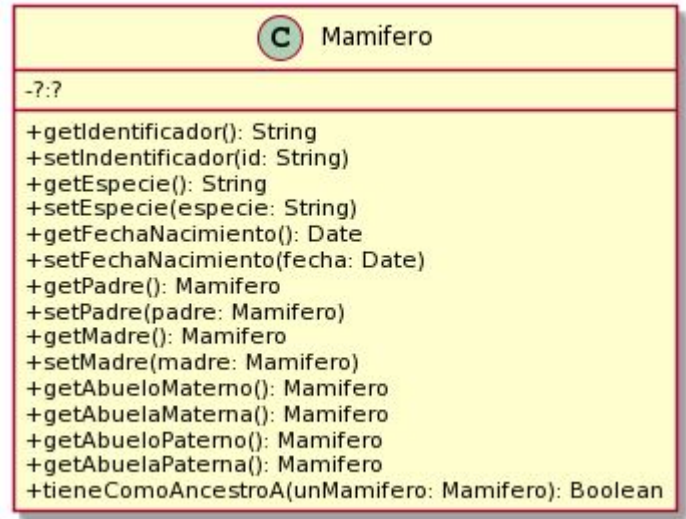
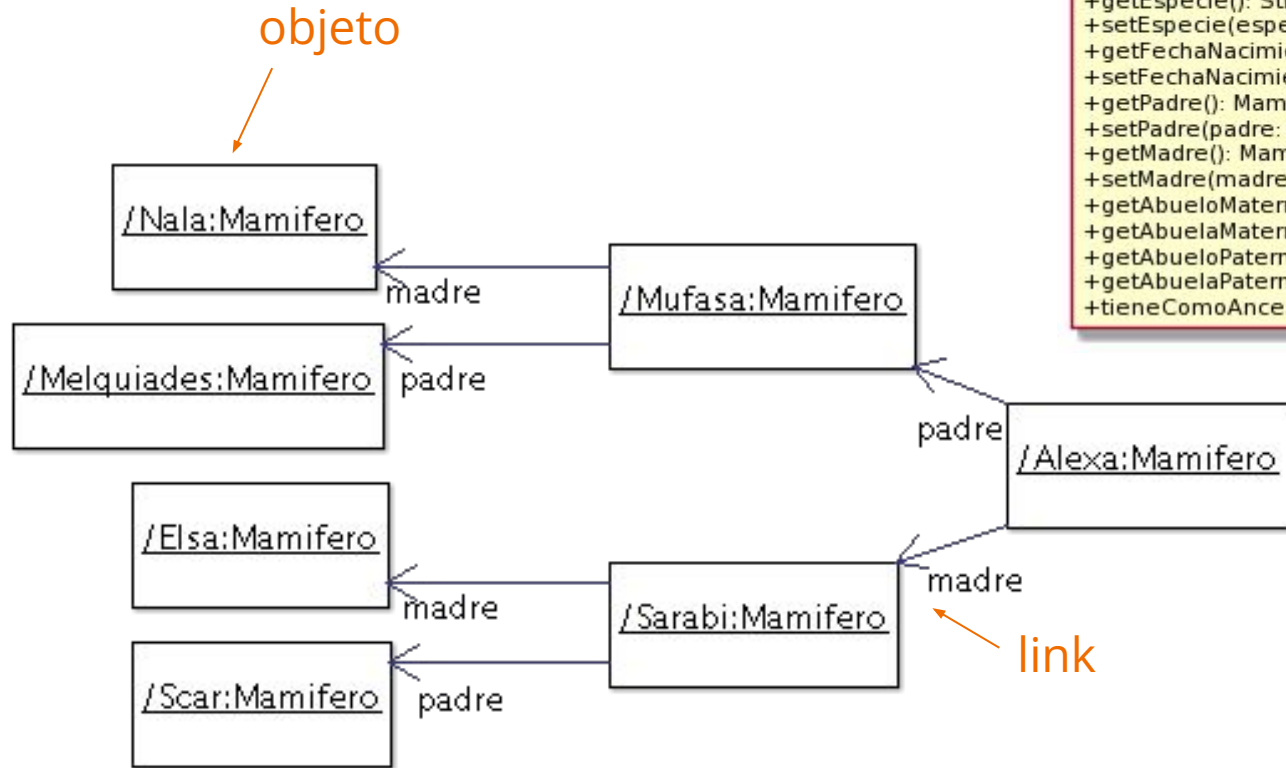


## Diagrama de objetos

Permiten visualizar una instancia específica de un sistema en un momento dado.

Se pueden mostrar los valores de los atributos y los links que son las referencias a otros objetos.

# Diagrama de objetos



## Diagrama de paquetes

Permiten la agrupación de clases

Son útiles para mostrar la organización de un sistema y cómo los elementos se agrupan y relacionan entre sí.

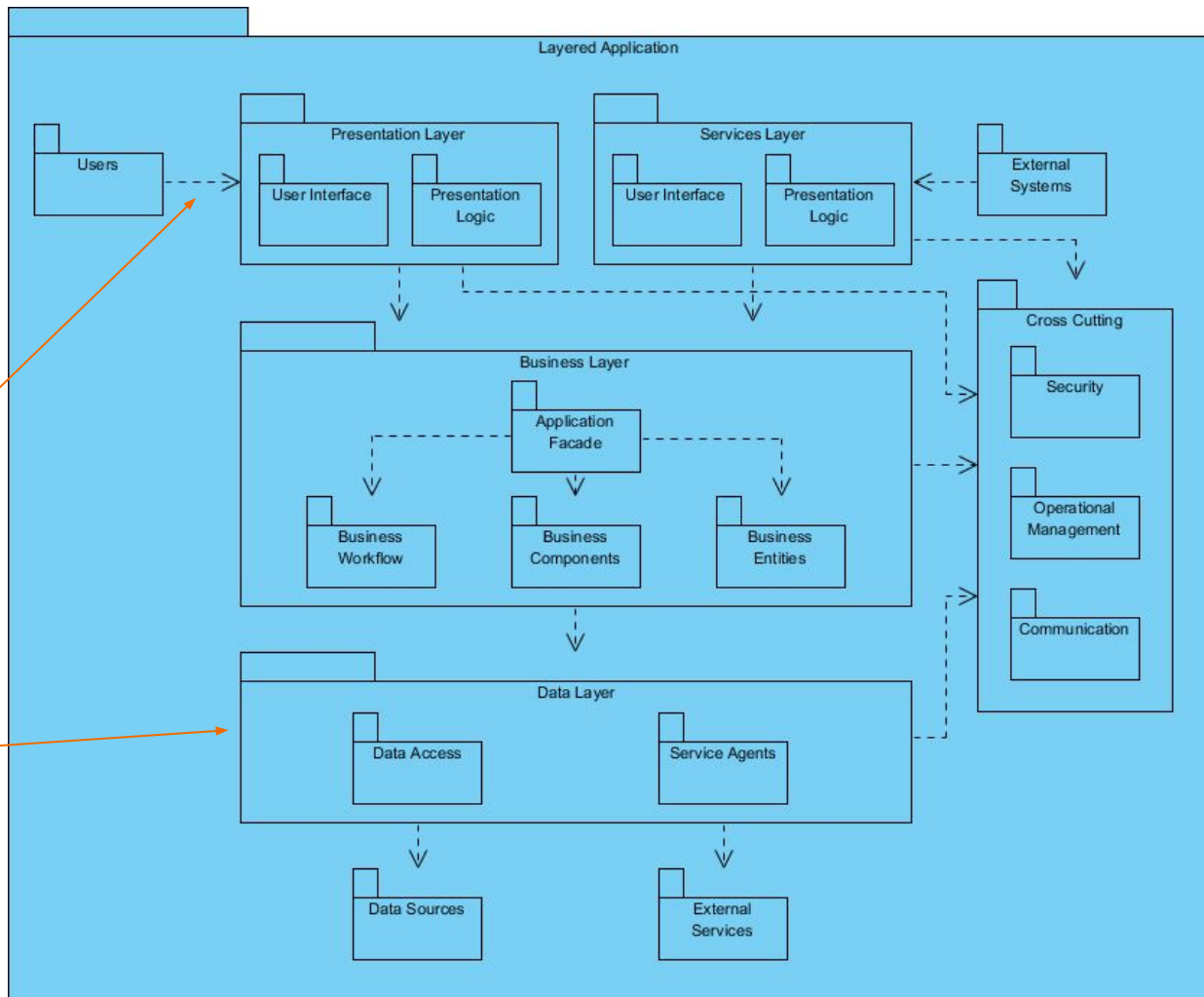
Se quiere

- una alta cohesión dentro de un paquete. Los elementos dentro de un paquete están relacionados
- poco acoplamiento entre ellos (exportando sólo aquellos elementos necesarios e importando solo lo necesario)

# Diagrama de paquetes

dependencia

paquete





## Diagrama de secuencia

Un diagrama de secuencia es un tipo de diagrama de interacción porque describe cómo —y en qué orden— colabora un grupo de objetos.

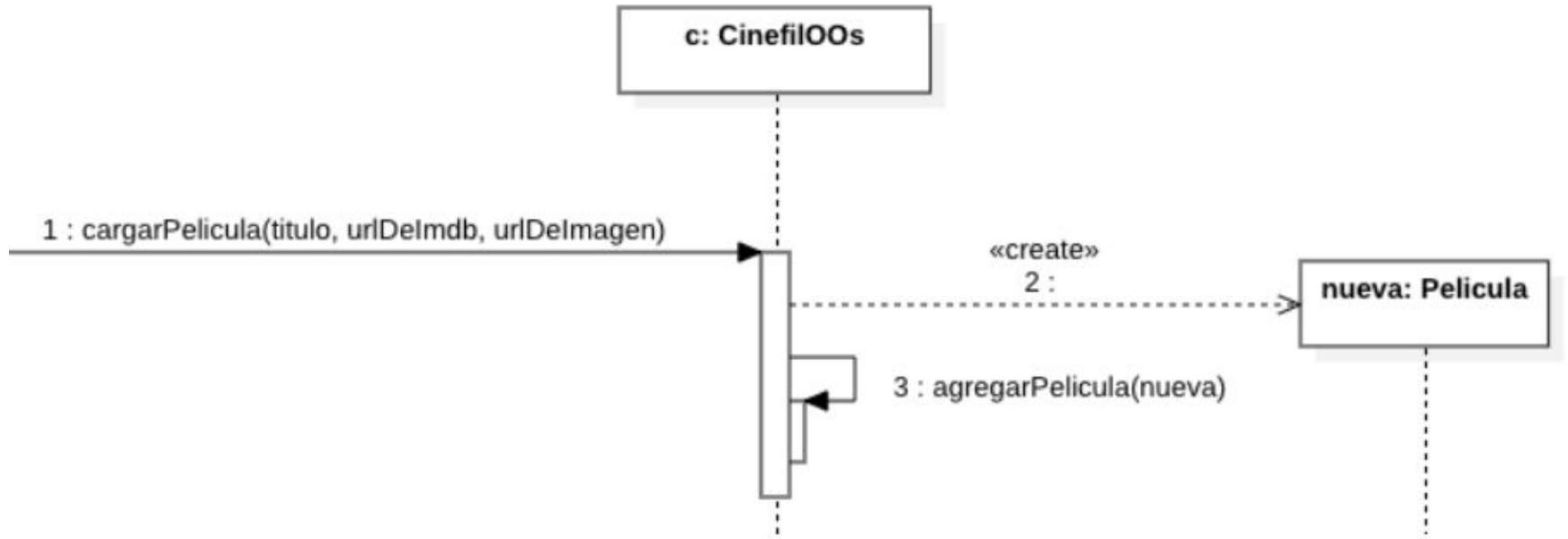
- Muestra claramente cómo interactúan distintos objetos en un sistema a lo largo del tiempo.

En un diagrama de secuencia

- los objetos se representan en la parte superior del diagrama
- el tiempo avanza de arriba hacia abajo.
-

# Diagrama de secuencia

Diagrama de secuencia del mensaje cargarPelicula de CinefilOOs



# Diagrama de secuencia

- Las flechas horizontales muestran las interacciones entre los objetos, indicando quién envía un mensaje a quién y en qué orden.

las flechas indican el envío de mensajes

Líneas de vida, representan la existencia temporal de un objeto

1 : cargarPelicula(titulo, urlDelmdb, urlDelmagen)

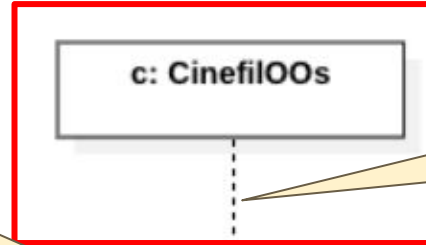
«create»

2 :

nueva: Pelicula

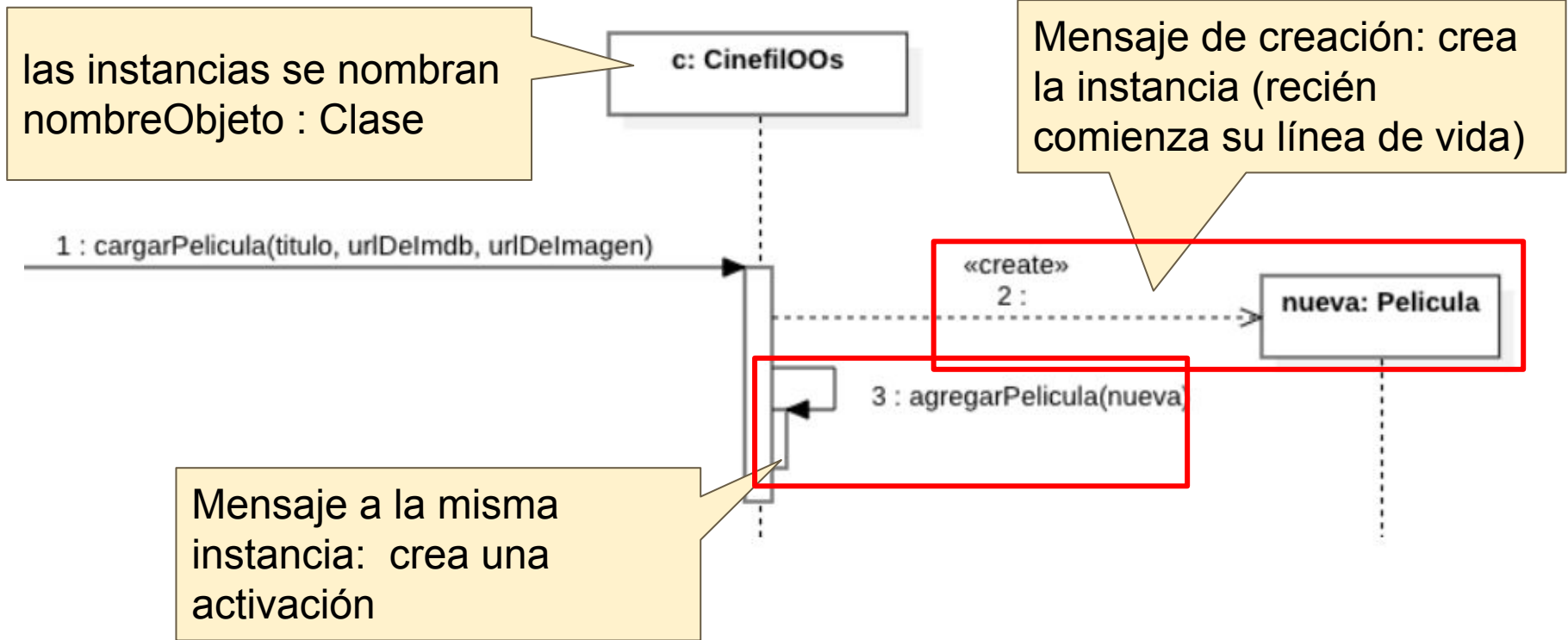
3 : agregarPelicula(nueva)

Activación: se representa mediante una barra vertical sobre la línea de vida



# Diagrama de secuencia

- Las flechas horizontales muestran las interacciones entre los objetos, indicando quién envía un mensaje a quién y en qué orden.



# Diagrama de secuencia

Los corchetes indican que es opcional

Sintaxis del mensaje:

`[atributo:= ] nombre del mensaje (parámetros) [:valor de retorno]`

**Ejemplos:**

`agregarPelicula (nueva)`

`hayMonto:= hayMontoSuficienteParaExtraer(monto)`

# Diagrama de secuencia - CombinedFragment

Un fragmento combinado permite representar la lógica y las condiciones en la interacción entre objetos.

A través de ellos se pueden especificar bloques para repetición, opcionales y alternativos, entre otros.

Fragmentos más utilizados:

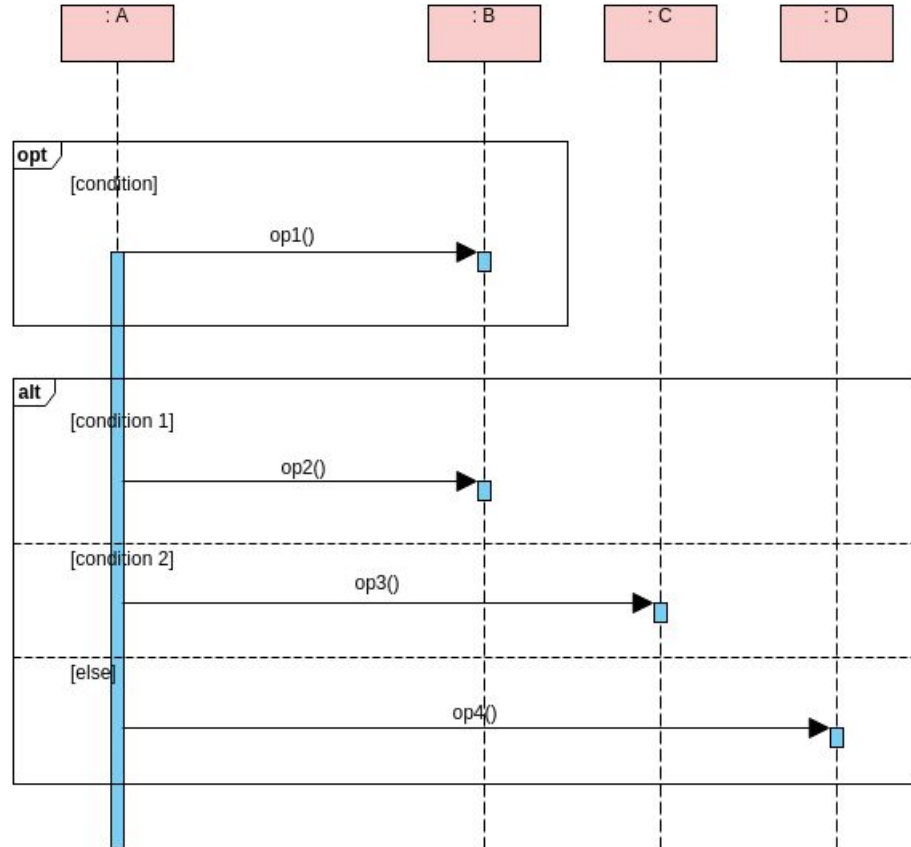
- opt: opcional
- alt: alternativa
- loop: bucle
- ref

# Diagrama de secuencia - CombinedFragment

Fragmento	Significado
alt	fragmento alternativo: tiene varias condiciones. Solo se ejecuta el fragmento cuya condición es verdadera.
opt	fragmento opcional: tiene un solo camino. Se ejecuta si esa condicion es verdadera. Caso particular de alt.
loop	Bucle: el fragmento puede ejecutarse varias veces, y la condición indica la base de la iteración.
ref	Referencia: se refiere a una interacción definida en otro diagrama. El marco abarca las líneas de vida involucradas en la interacción.

# Diagrama de secuencia - CombinedFragment

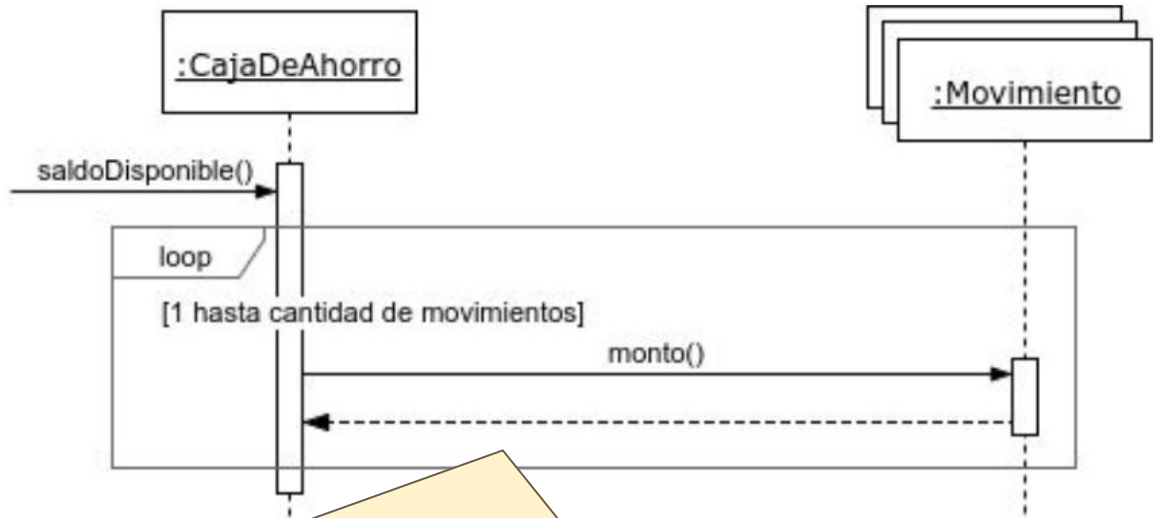
- **opt (Opcional):** Representa una parte de la secuencia de interacción que puede o no ejecutarse, dependiendo de una condición booleana. Si la condición es verdadera, se ejecuta la parte opcional; de lo contrario, se omite.
- **alt (Alternativa):** Este tipo de fragmento se utiliza para modelar una elección entre diferentes opciones de interacción. En cada opción se evalúa una condición booleana para determinar cuál de las opciones se ejecutará.





# Diagrama de secuencia - CombinedFragment

**loop (Bucle):** Se utiliza para modelar repeticiones de una secuencia de interacción. Puede especificar el número de repeticiones o utilizar una condición para controlar la terminación del bucle.



Mensaje de retorno: se dibuja con una línea discontinua que apunta desde el objeto receptor al objeto que envió el mensaje. Su uso es opcional

**Cómo se define UML?**

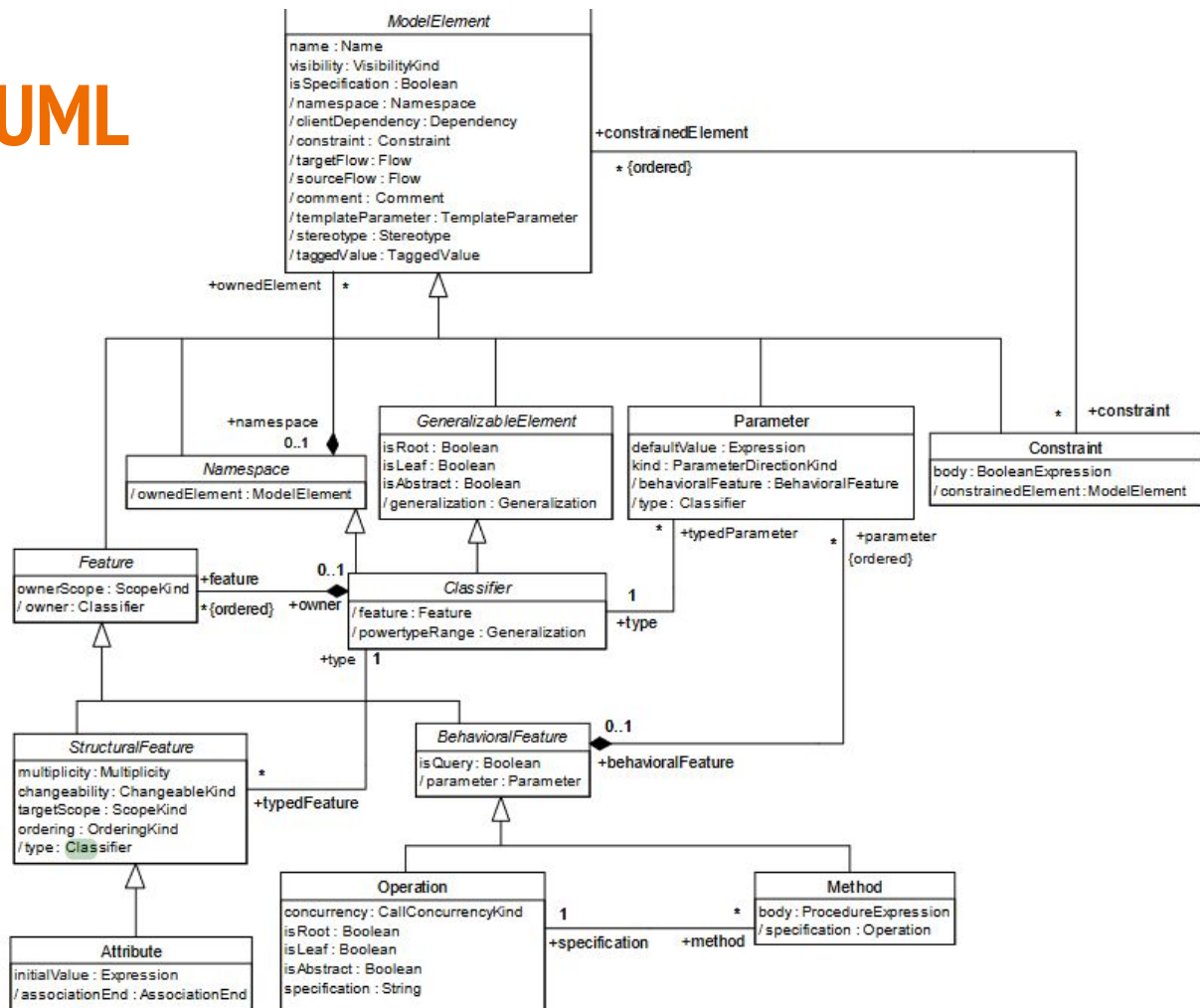
## Cómo se define UML?

Documento de especificación de UML

Metamodelo de UML

- es una especificación que define las construcciones y elementos básicos que pueden utilizarse para crear diagramas en UML.
- es esencialmente una descripción formal de cómo se estructuran y relacionan los elementos

# Metamodelo de UML



## Herramientas de modelado

Las herramientas de modelado UML pueden ser tanto gráficas como basadas en el metamodelo UML

Herramienta gráfica: permite dibujar elementos con la misma imagen que UML

Herramientas basadas en el metamodelo UML: Estas herramientas se centran más en la manipulación directa de los elementos del metamodelo UML.

# Herramientas UML



# PlantUML

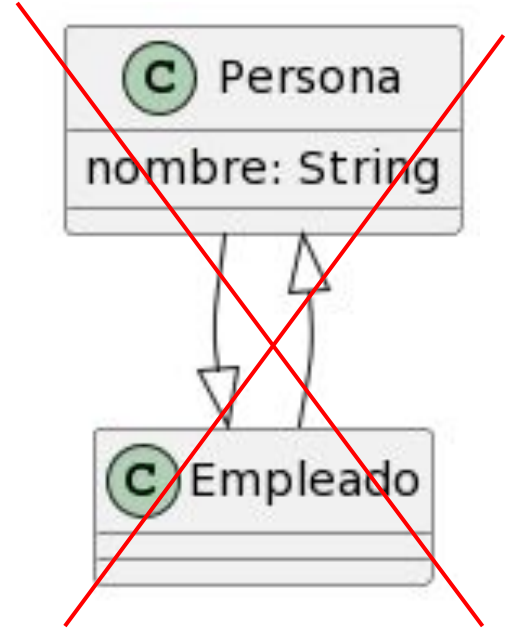
Herramienta que pasa de texto a imagen

no tiene chequeo del metamodelo.

Permite dibujar cosas ajenas a UML, o con errores semánticos

Versión online

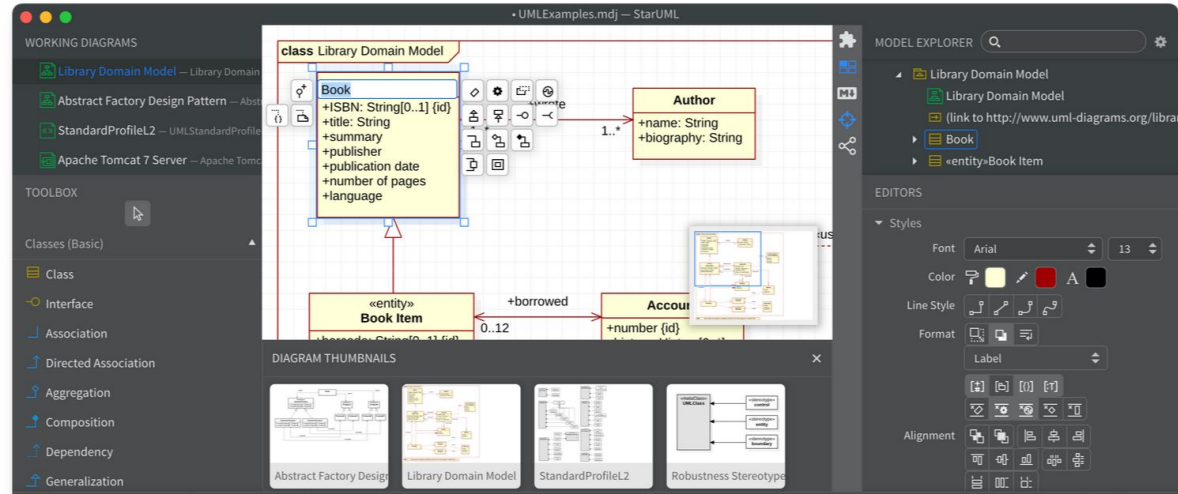
En UML no es posible una jerarquía en ciclo.





Realiza el chequeo de algunas reglas definidas  
en el metamodelo

Versión instalable

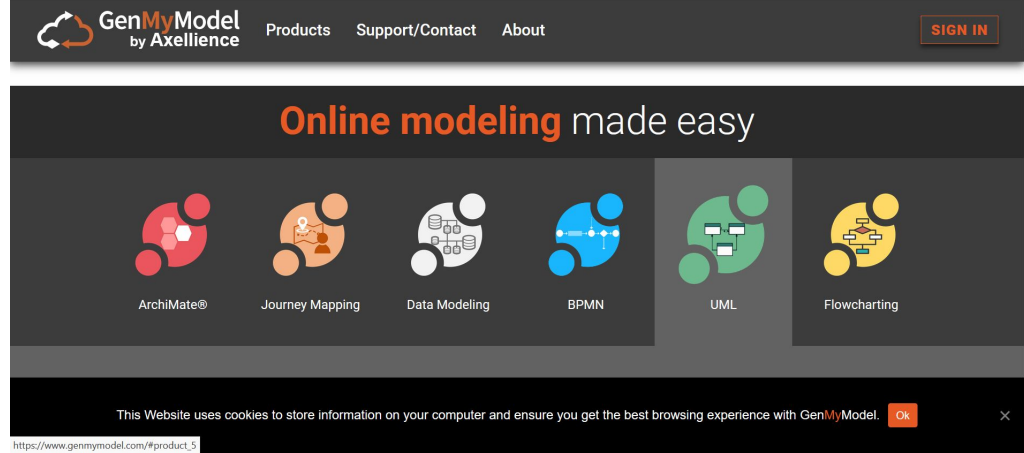
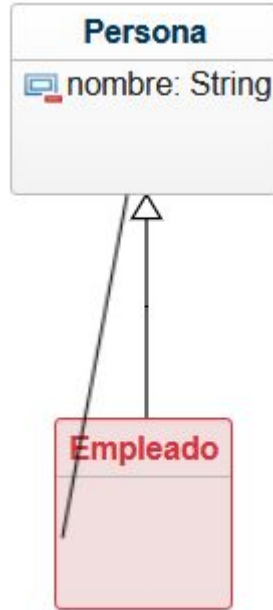




# GenMyModel

Realiza el chequeo de algunas reglas definidas en el metamodelo

Versión online



En UML no es posible una jerarquía en ciclo.

No es posible dibujarlo.

# Para repasar

- Sitio oficial UML - <https://www.uml.org/>
- Documento de especificación - Metamodelo (<https://www.omg.org/spec/UML/2.5/PDF>)
  
- Grady Booch Reflects on UML 1.1 20th Anniversary ( [LINK](#) )
- PlantUML (<https://plantuml.com/es/>)
- GenMyModel (<https://app.genmymodel.com>)

¿ Preguntas ?