

UNIDAD 5

Elaboración de diagramas de clases

Contenidos

- 5.1. Conceptos básicos de la orientación a objetos
- 5.2. El lenguaje UML
- 5.3. Clases, atributos, métodos y visibilidad
- 5.4. Relaciones entre clases
- 5.5. Tipos de clases de análisis
- 5.6. Herramientas para la creación de diagramas de clases
- 5.7. Generación de código a partir de diagramas de clases
- 5.8. Generación de diagramas de clases a partir de código (ingeniería inversa)

5.1. Conceptos básicos de la orientación a objetos

- ❑ Cambio de paradigma: atención a los procesos → atención a los datos.
- ❑ Características de los lenguajes orientados a objetos:



5.2. El lenguaje UML

■ UML es un lenguaje gráfico que se usa para visualizar, especificar, construir y documentar un sistema por medio de diferentes tipos de diagramas que se usan en distintas tareas de desarrollo de software.

UML incorpora varios tipos de diagramas con notaciones gráficas y textuales para mostrar el sistema con diferentes niveles de abstracción y distinto nivel de detalle.

5.2.1. Tipos de elementos en UML

Elementos estructurales

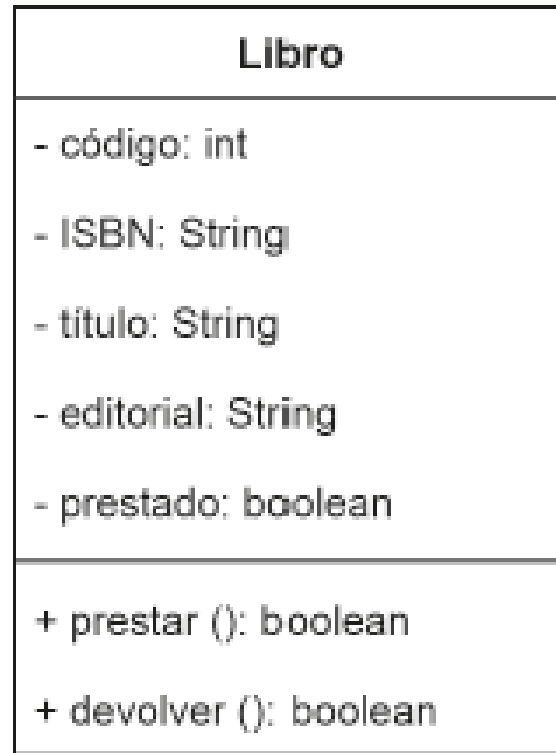


Figura 5.1. Clase llamada Libro con cinco atributos (*código*, *ISBN*, *título*, *editorial* y *prestado*) y dos métodos (*prestar* y *devolver*).



Figura 5.2. Interfaz *IVentana* con las operaciones que ofrece (abrir, cerrar, mover y dibujar).



Figura 5.3. Representación de una colaboración, en este caso *Generar factura*.



Figura 5.4. Representación del caso de uso *Registrar pedido*, que describe un conjunto de operaciones que realiza la aplicación y que son de interés para la persona que la utiliza.



Figura 5.5. Representación de la clase activa *GestorDeEventos* con los métodos *suspender* y *vaciarCola*.



Figura 5.6. Representación del componente *FormularioDeCliente*.



Figura 5.7. Representación de un artefacto llamado *NavegadorWeb*.



Figura 5.8. Nodo que representa un servidor web.

Elementos de comportamiento



Figura 5.9. Mensaje que solicita la impresión de una factura.



Figura 5.10. Representación de un estado en espera.

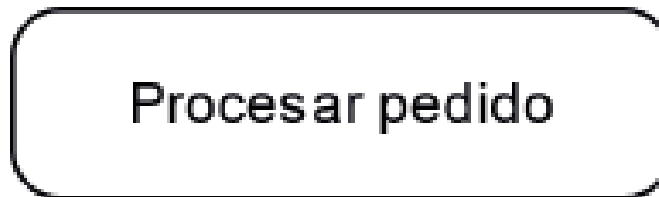


Figura 5.11. Acción que hace referencia al procesamiento de un pedido.
Las acciones se representan de igual modo que los estados, pero se distinguen de estos por el contexto en el que se usan.

Elementos de agrupación



Figura 5.12. Representación de un paquete llamado *Reglas del negocio*.

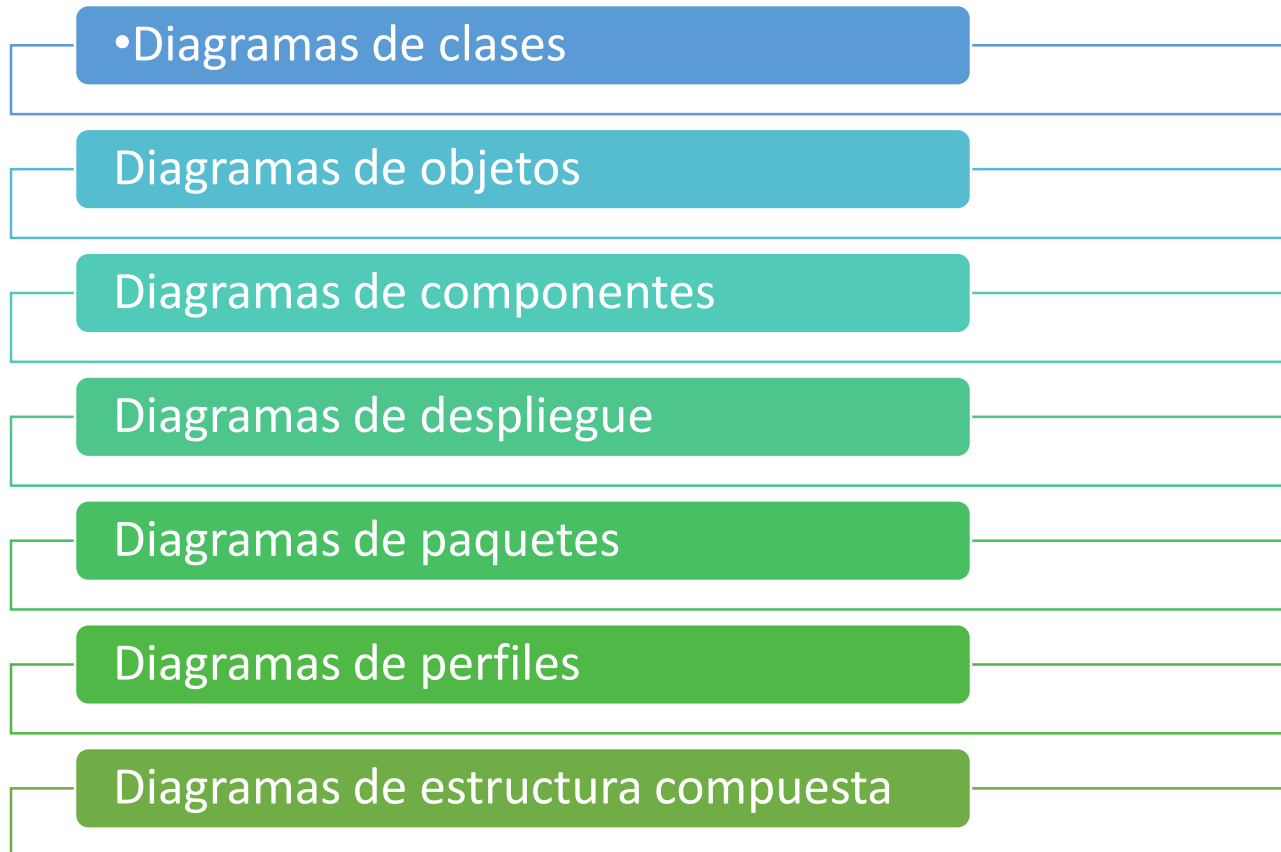
Elementos de anotación



Figura 5.13. Nota que indica que la clase *Ejemplar* es *débil* respecto a la clase *Libro*.

5.2.1. Tipos de diagramas en UML

Diagramas estructurales



Diagramas de comportamiento

- Diagramas de casos de uso

Diagramas de actividades

Diagramas de estados

Diagramas de interacción

Diagramas de secuencia

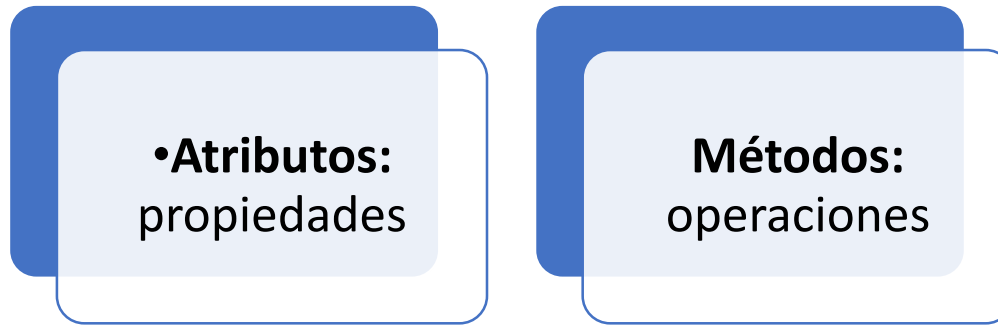
Diagramas de colaboración

Diagramas de tiempos

Diagrama global de interacciones

5.3. Clases, atributos, métodos y visibilidad

- ❑ Un objeto = una instancia de una clase.
- ❑ Una clase = un conjunto de objetos con la misma estructura y comportamiento:



- ❑ Las clases se representan mediante un rectángulo en los diagramas de clases con diferente nivel de detalle.

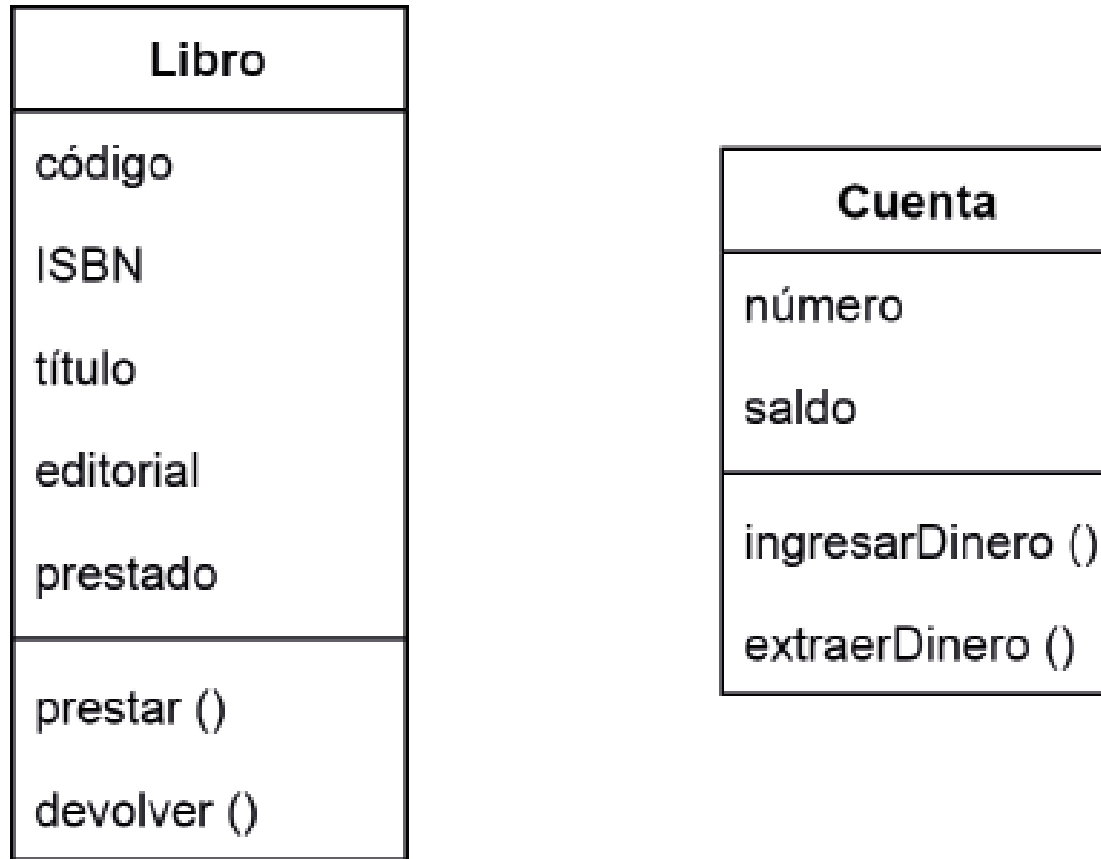


Figura 5.14. Representación de las clases *Libro* (con los atributos *código*, *ISBN*, *título*, *editorial* y *prestado* y los métodos *prestar* y *devolver*) y *Cuenta* (con los atributos *número* y *saldo* y los métodos *ingresarDinero* y *extraerDinero*).

- ❑ Para cada atributo se puede especificar:

modificador atributo: tipoDato

- ❑ Modificador de acceso o visibilidad:

- *Public* (+).
- *Protected* (#).
- *Private* (-).
- *Package* (~).

- ❑ Tipos de datos:

- *int* / *integer*.
- *String*.
- *boolean*.
- *float* / *double*.
- *char*.
- *Date*.
- *Time*.
- *DateTime*.

- ❑ Para cada método se puede especificar:

modificador método (parám₁: tipoDato₁, parám₂: tipoDato₂, ...) :
tipoDevuelto

Libro
- código: int - ISBN: String - título: String - editorial: String - prestado: boolean
+ prestar (): boolean + devolver (): boolean

Cuenta
- número: String - saldo: float
+ ingresarDinero (importe: float) + extraerDinero (importe: float)

Figura 5.15. Representación detallada de las clases *Libro* y *Cuenta*, indicando la visibilidad de cada atributo y método. En el caso de los atributos, se indica, además, su tipo de dato, y en el caso de los métodos, el tipo de dato devuelto, si lo hay, y por cada parámetro, si es el caso, su nombre y tipo de dato.

5.4. Relaciones entre clases

5.4.1. Agregación

- ❑ Relación entre un compuesto y sus componentes.
- ❑ El tiempo de vida de los componentes es diferente que el del agregado.
- ❑ Multiplicidades o cardinalidades para cualquier relación:

1

0..1

0..* / *

1..*

N

N..M

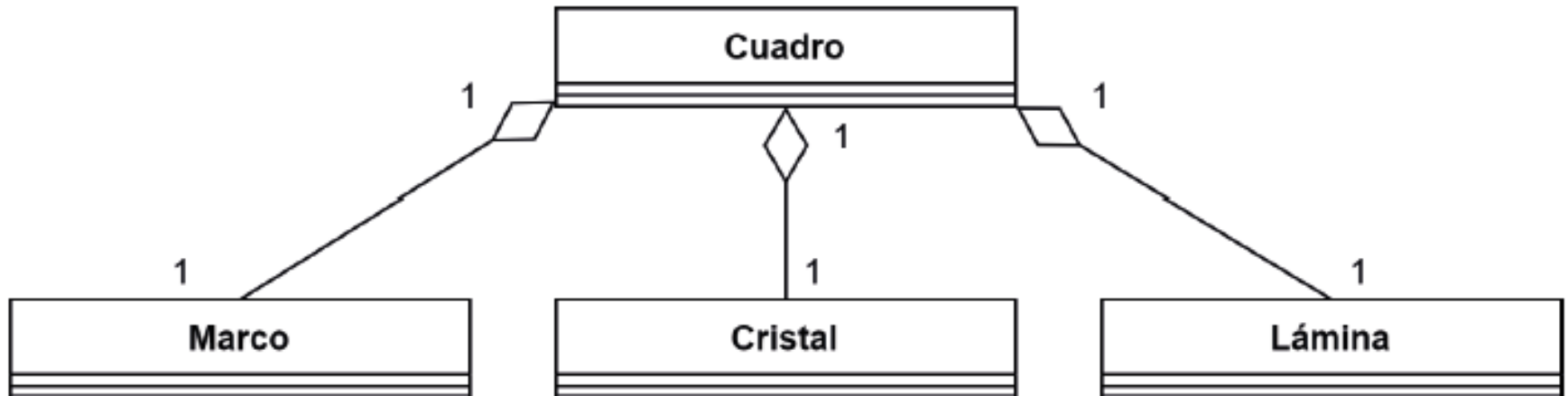


Figura 5.16. Diagrama *UML* que muestra una relación de agregación entre la clase de tipo compuesto *Cuadro* y sus partes o componentes *Marco*, *Cristal* y *Lámina*.

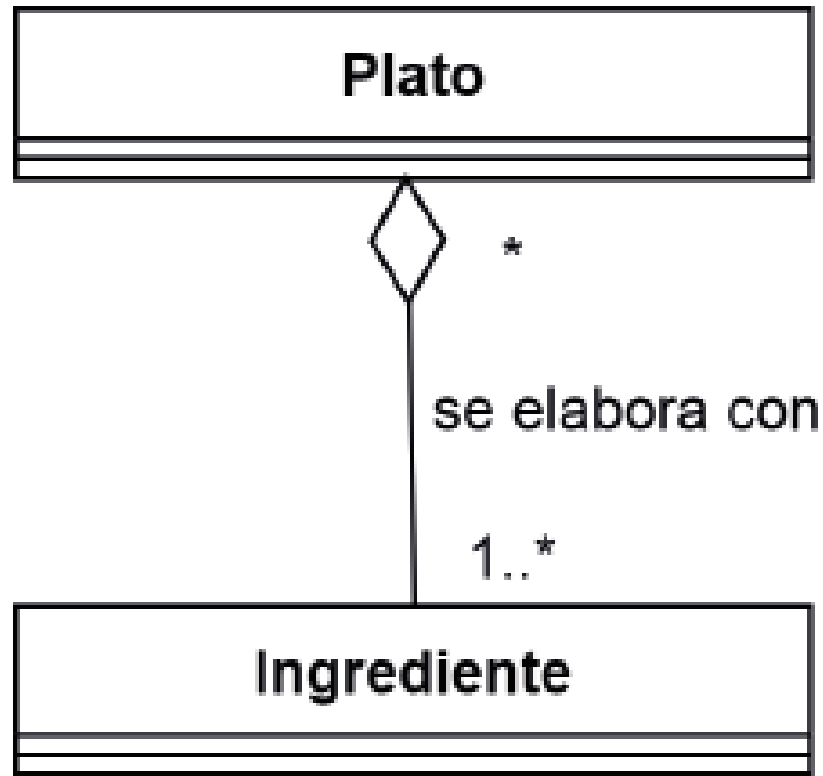


Figura 5.17. Diagrama UML que muestra una relación de agregación entre la clase de tipo compuesto *Plato* y sus ingredientes.

5.4.2. Composición

- ❑ Agregación fuerte.
- ❑ La cardinalidad al lado del compuesto es siempre 1.
- ❑ Restricciones:

•Cada componente solo puede estar presente en un compuesto.

Si se elimina el compuesto, hay que eliminar todos sus componentes.



Figura 5.18. Diagrama UML que muestra una relación de composición entre la clase 'compuesto' *PlanEstudios* y sus asignaturas.

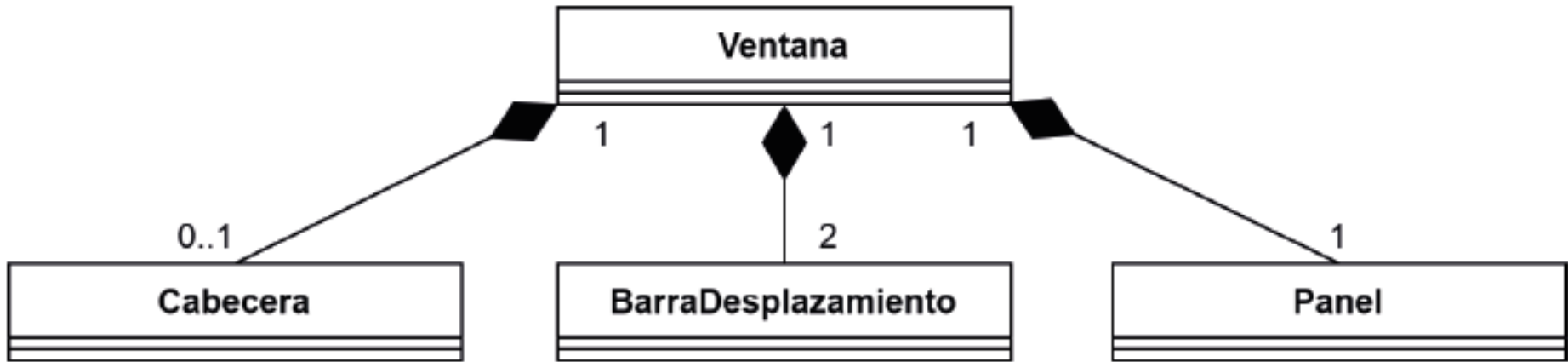
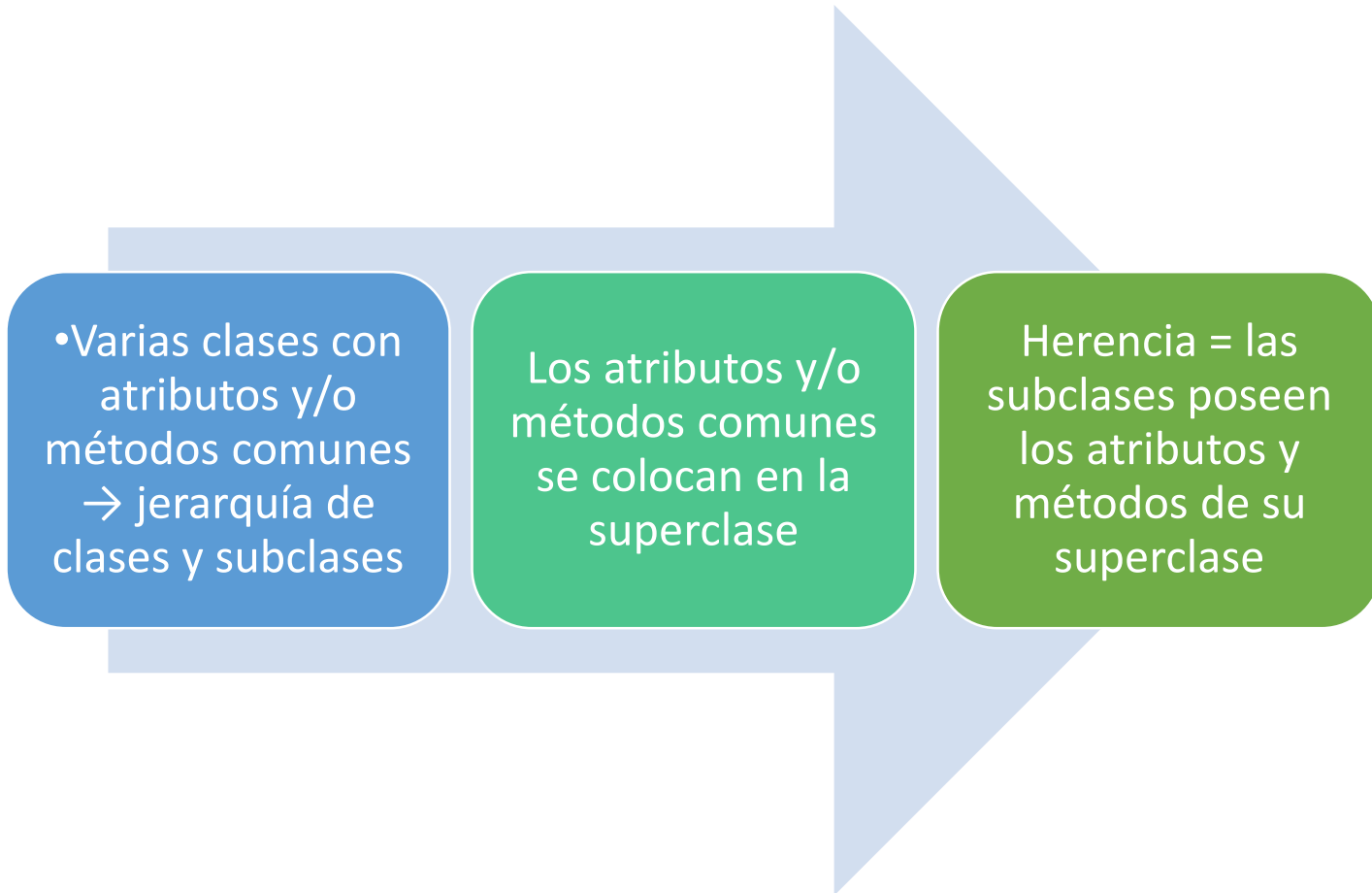


Figura 5.19. Diagrama UML que muestra una relación de composición entre la clase ‘compuesto’ *Ventana* y sus componentes *Cabecera*, *BarraDesplazamiento* y *Panel*.

5.4.3. Generalización y especialización



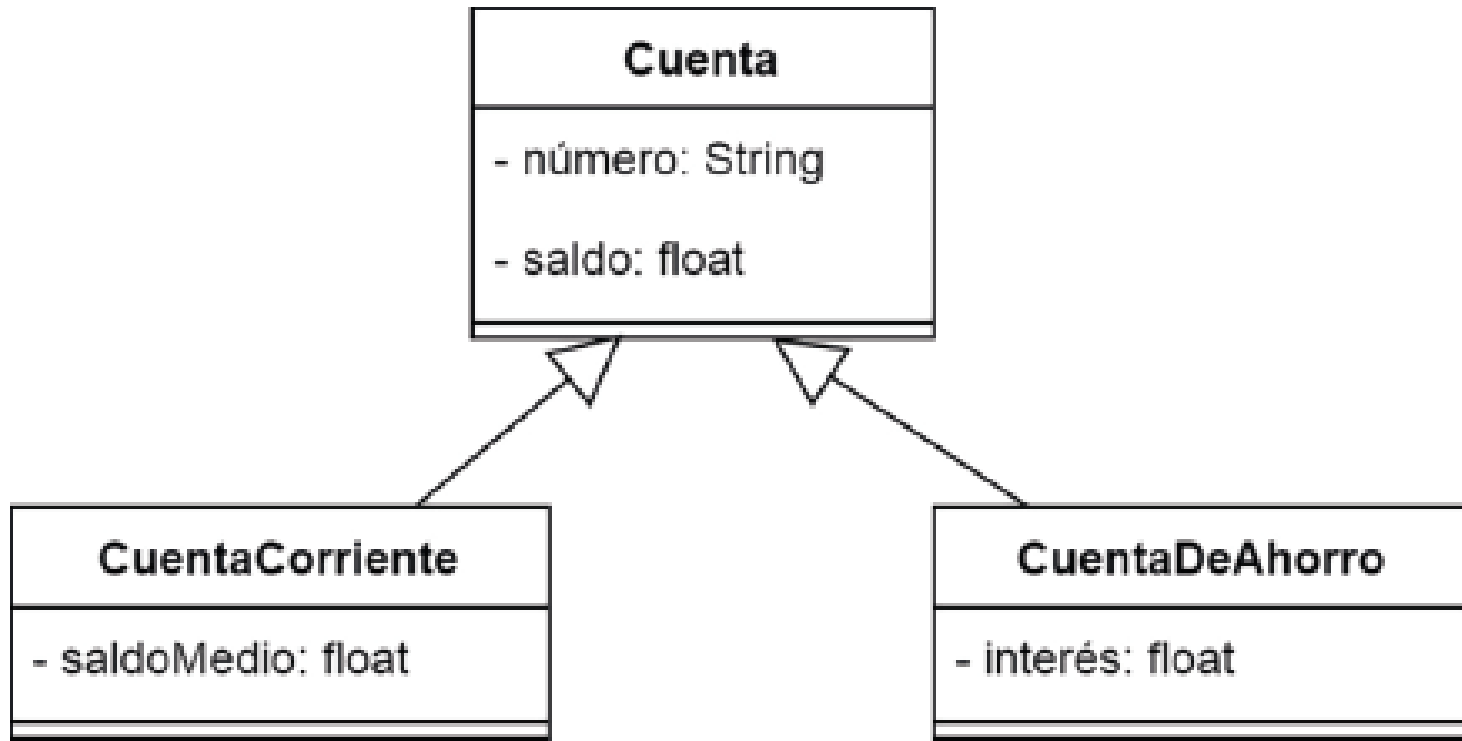


Figura 5.20. Diagrama UML que muestra una relación de generalización-especialización entre la superclase *Clase* y sus subclases *CuentaCorriente* y *CuentaDeAhorro*.

5.4.4. Asociación

- ❑ Asociación = relación genérica.
- ❑ En función del grado:
 - Reflexivas.
 - Binarias.
 - Ternarias / cuaternarias...

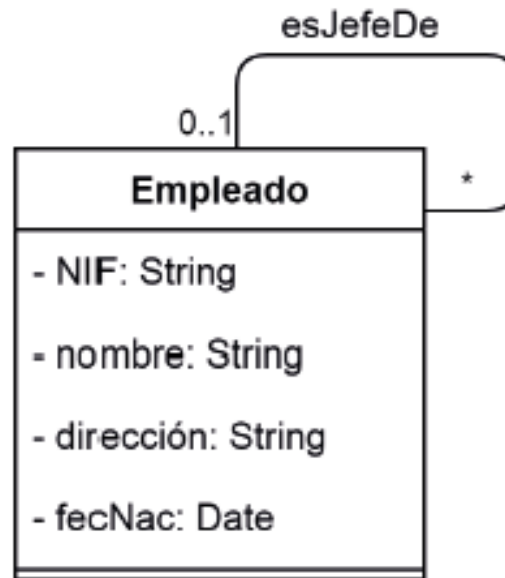


Figura 5.21. Diagrama UML que muestra una relación reflexiva de *Empleado*, la cual refleja el jefe directo de cada persona empleada, en caso de que lo tenga.

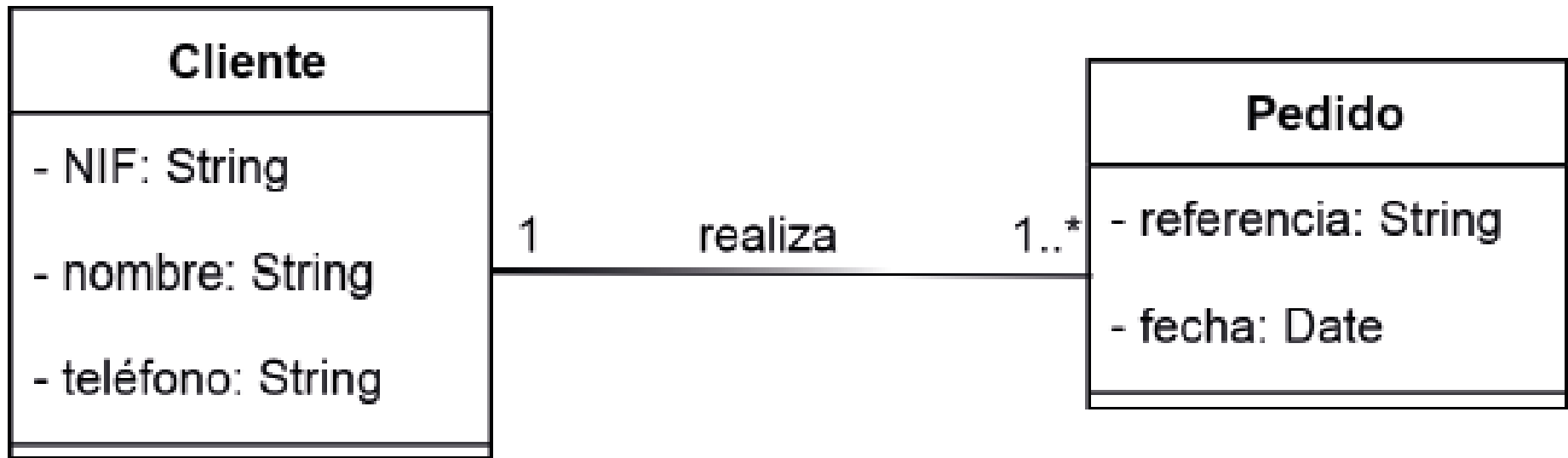


Figura 5.22. Diagrama UML que muestra una asociación binaria entre *Cliente* y *Pedido*, que refleja los pedidos que realiza cada cliente.

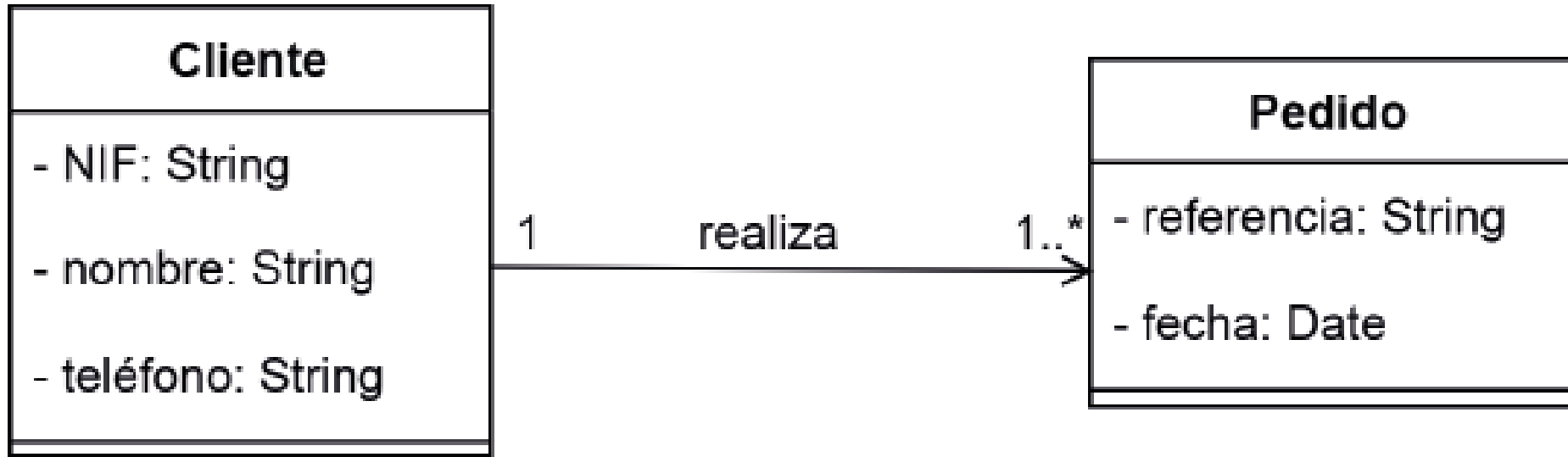


Figura 5.23. Diagrama UML que muestra una asociación binaria entre *Cliente* y *Pedido* navegable solo de *Cliente* a *Pedido*, lo que quiere decir que a partir de un cliente se puede acceder a todos los pedidos que este ha realizado.

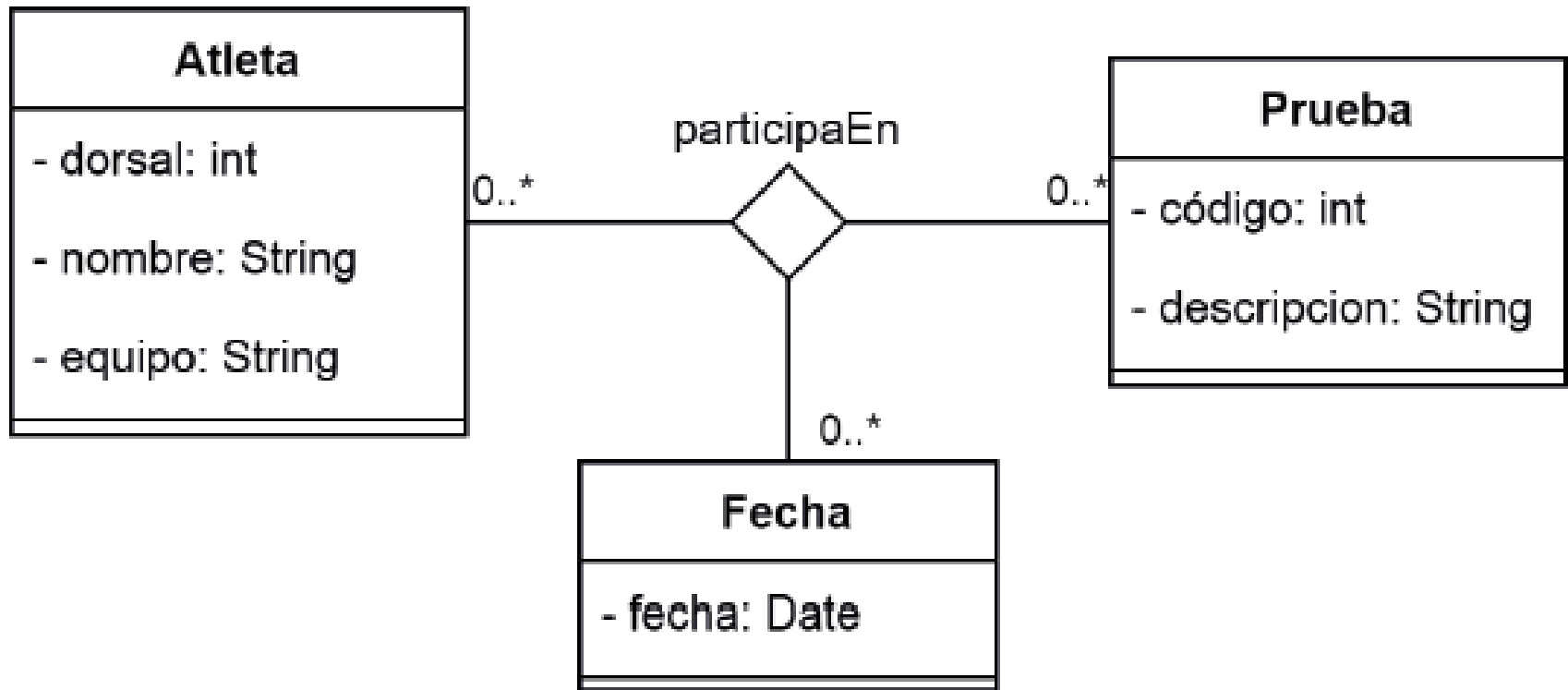


Figura 5.24. Diagrama UML que muestra una asociación ternaria entre *Prueba*, *Atleta* y *Fecha*, que refleja el día o los días en los que participa cada atleta en cada prueba.

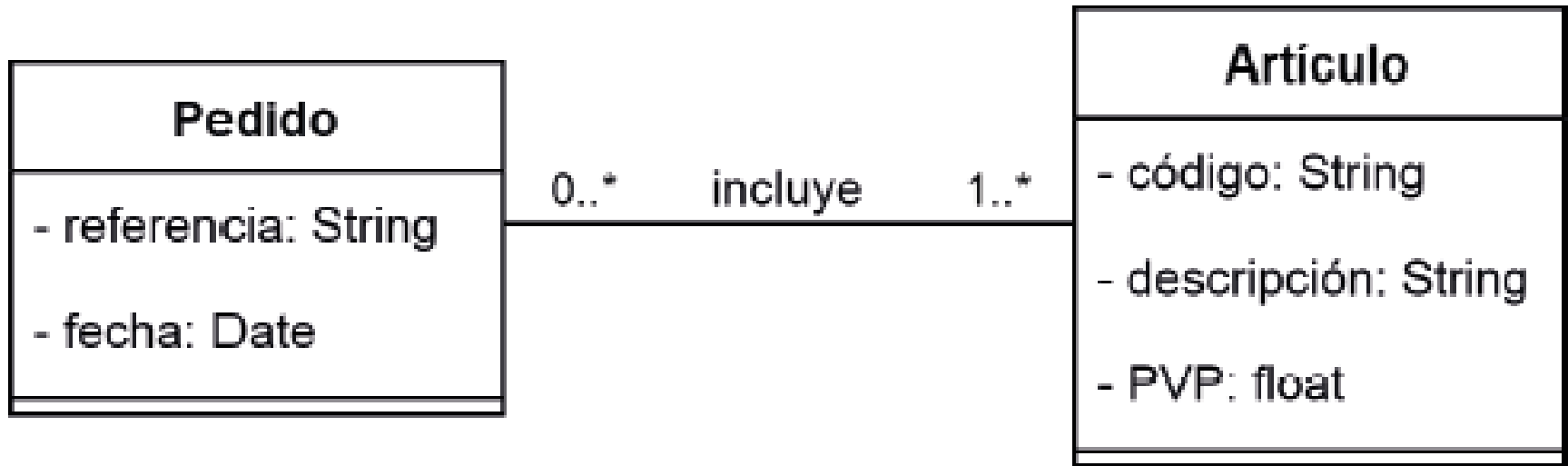


Figura 5.25. Diagrama UML que muestra una asociación binaria entre *Pedido* y *Artículo*, que refleja el artículo o los artículos solicitados en cada pedido.

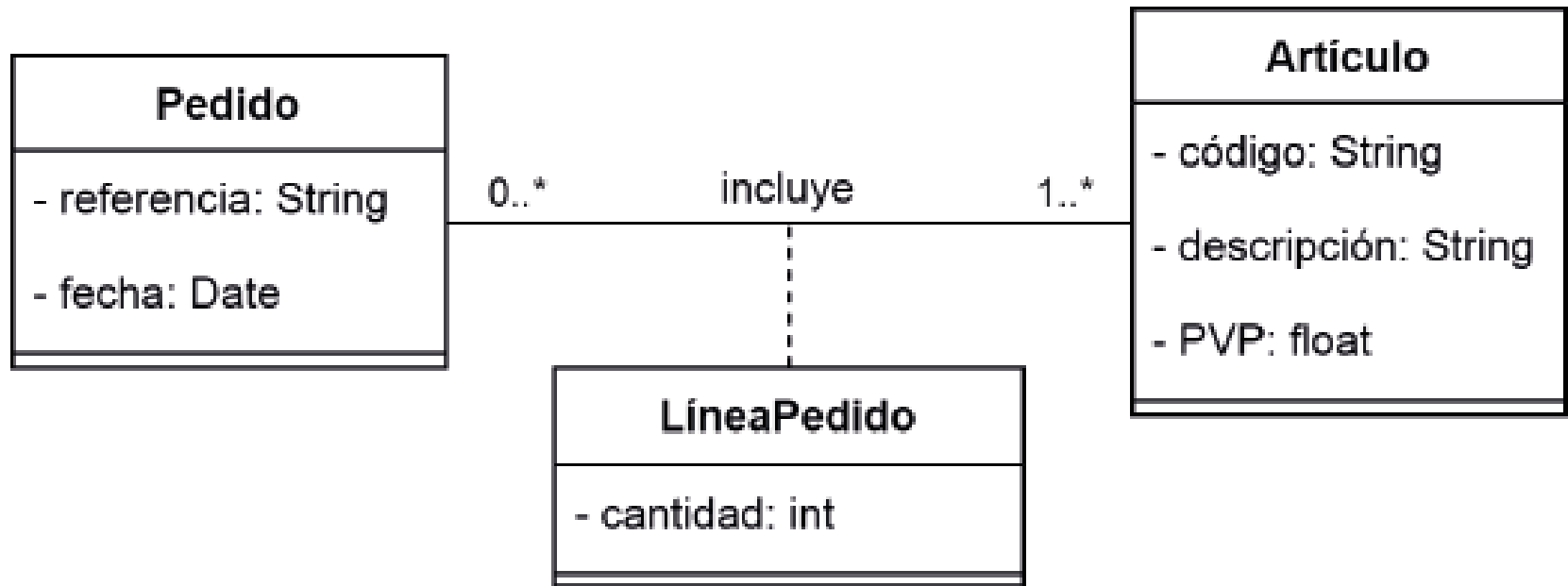


Figura 5.26. Diagrama UML que muestra, además de una asociación binaria entre *Pedido* y *Artículo*, una clase asociativa, a la que se ha llamado *LíneaPedido*. Esta indica, por artículo solicitado en cada pedido, la cantidad o número de unidades que de ese artículo se han solicitado en ese pedido.

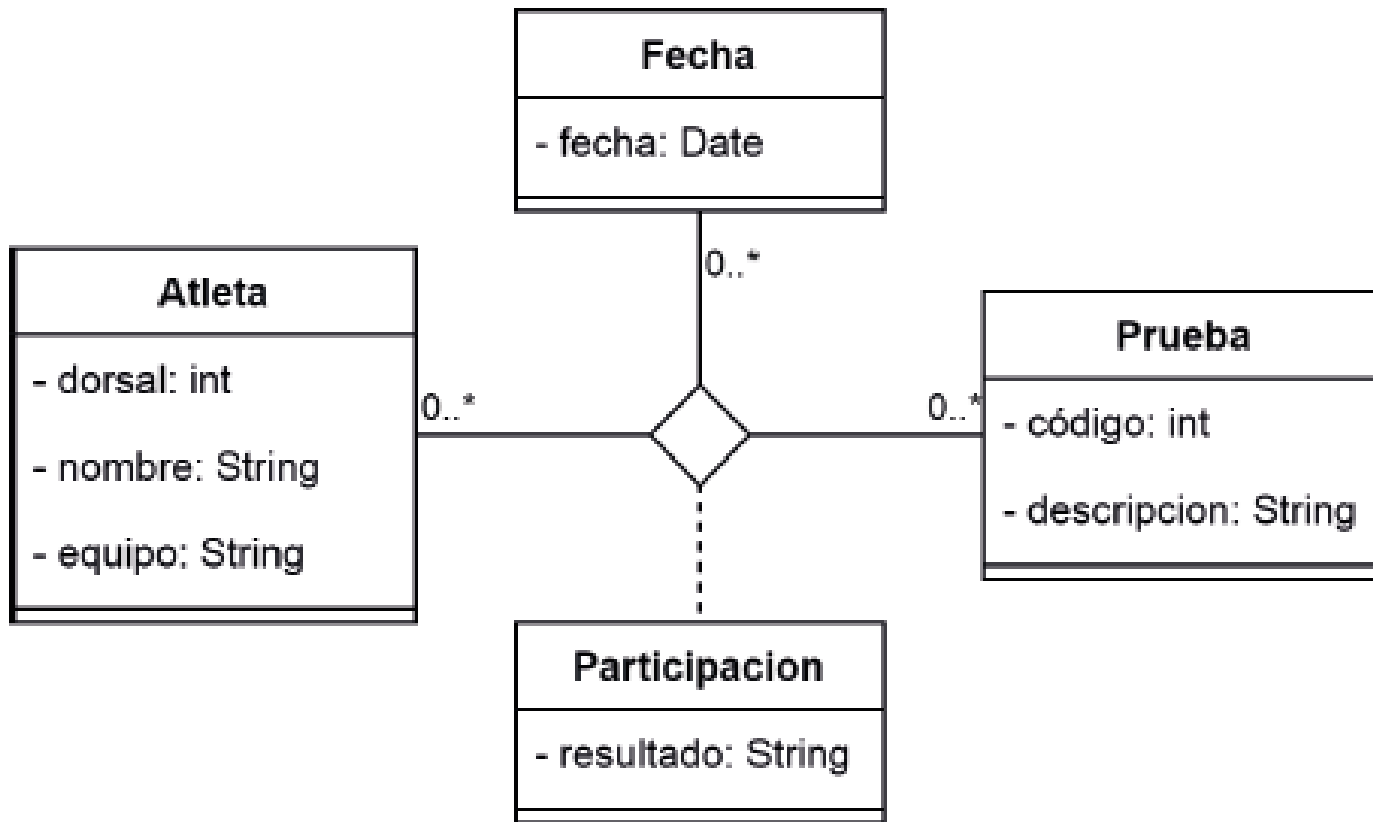


Figura 5.27. Diagrama UML que muestra una asociación ternaria entre *Prueba*, *Atleta* y *Fecha*, que refleja el día o los días que participa cada atleta en cada prueba. Además, se ha vinculado una clase asociativa a dicha relación ternaria con el objetivo de almacenar el resultado obtenido por cada atleta en cada prueba en la que ha participado.

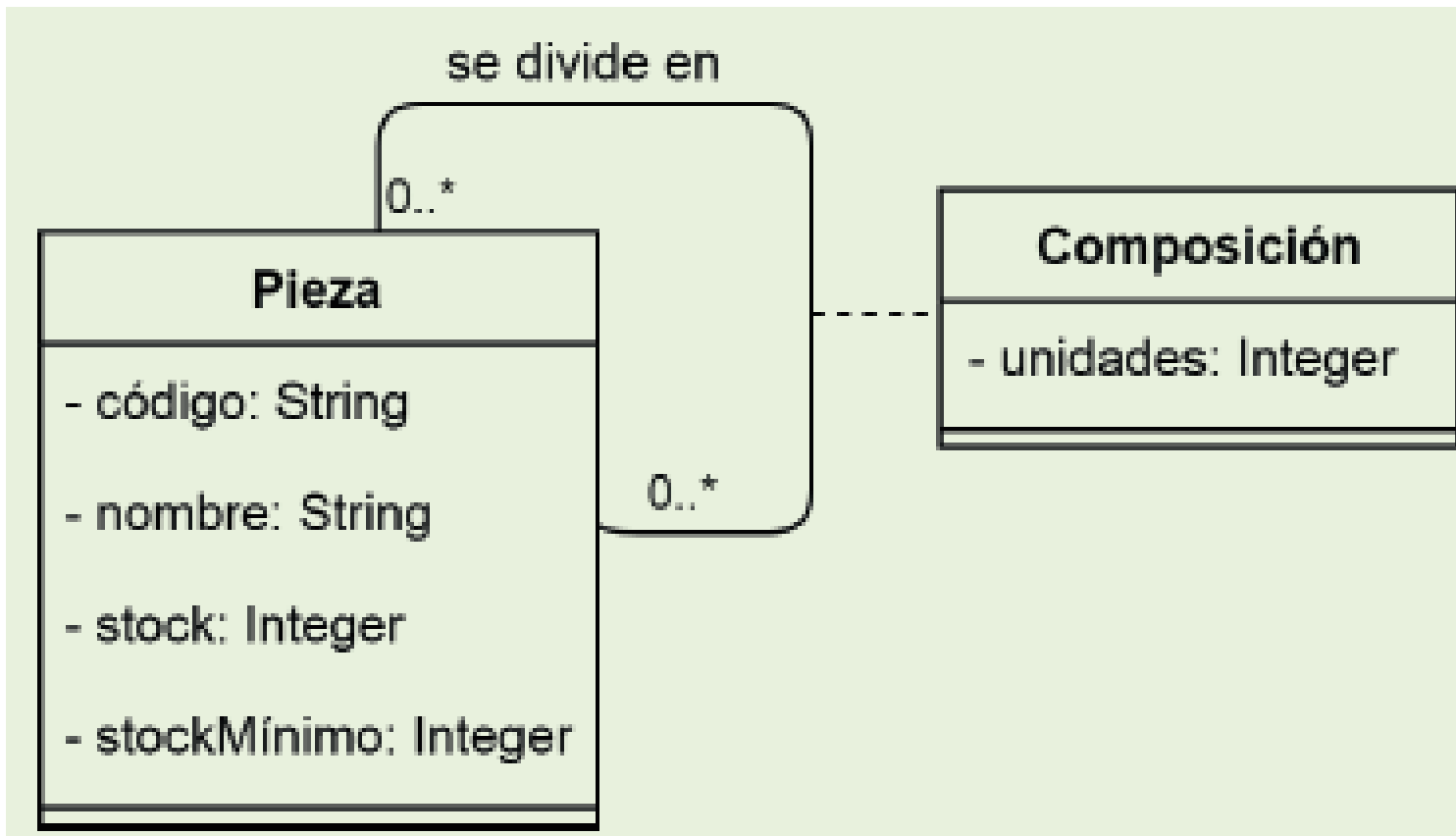


Figura 5.28. Diagrama UML que muestra una relación reflexiva de la clase *Pieza*, necesaria para conocer la relación jerárquica existente entre las piezas. Se precisa de una clase asociativa para conocer el número de subpiezas que contiene cada pieza.

5.4.5. Realización

- ❑ Relación entre una clase Interface y las clases que la implementan.
- ❑ Las interfaces contienen métodos comunes a varias clases.

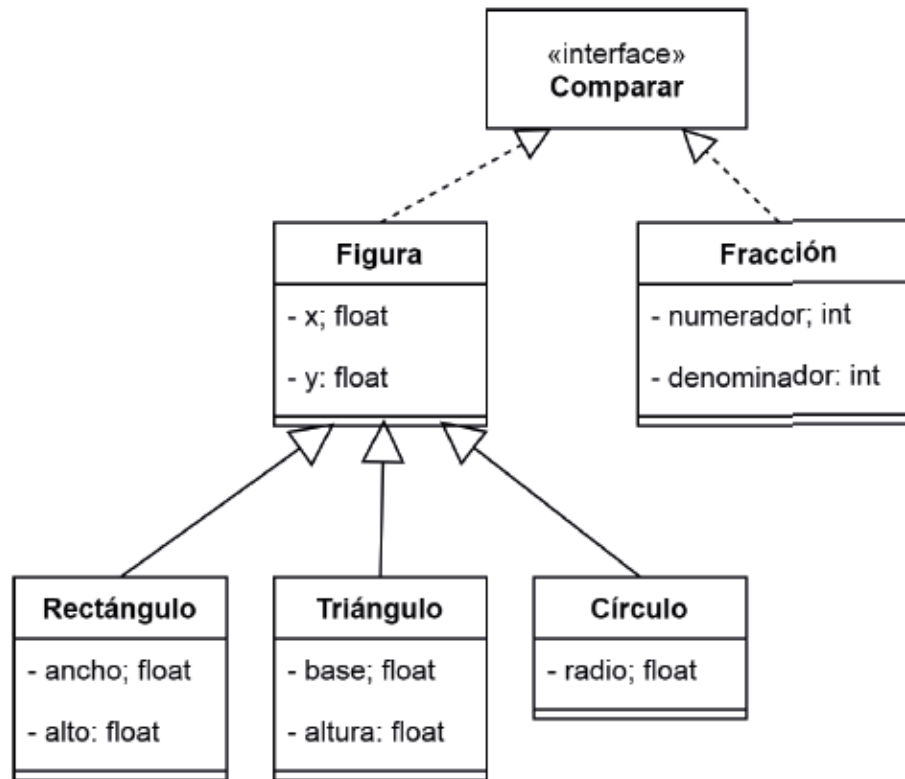
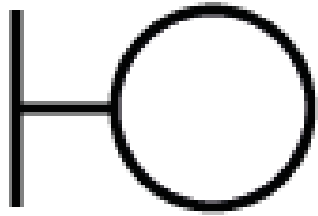


Figura 5.29. Diagrama UML que muestra una relación de realización entre *Figura* y la interfaz *Comparar* y otra relación de realización entre *Fracción* y la interfaz *Comparar*. Además, la clase *Figura* es la superclase de las subclases *Rectángulo*, *Triángulo* y *Círculo*.

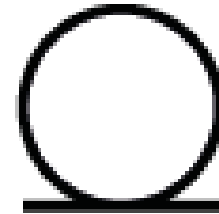
5.5. Tipos de clases de análisis



Clase de interfaz



Clase de control



Clase de entidad

Figura 5.30. Representación de los tipos de clases de interfaz, de control y de entidad.

5.6. Herramientas para la creación de diagramas de clases

5.6.1. Creación de diagramas de clases con diagrams.net

<https://app.diagrams.net>

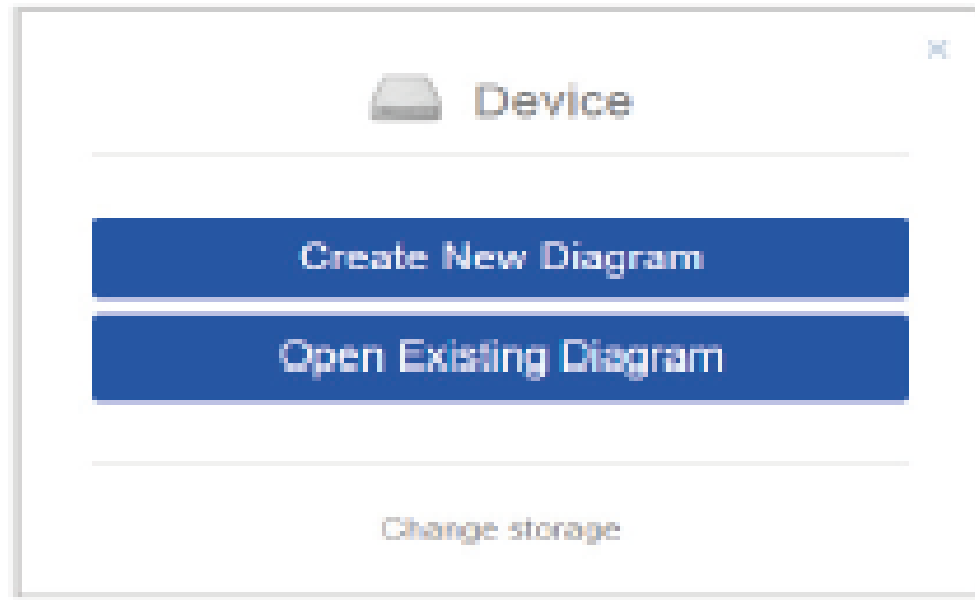


Figura 5.31. Pantalla de inicio de diagrams.net, en la que se muestran dos opciones: crear un nuevo diagrama o abrir un diagrama existente.

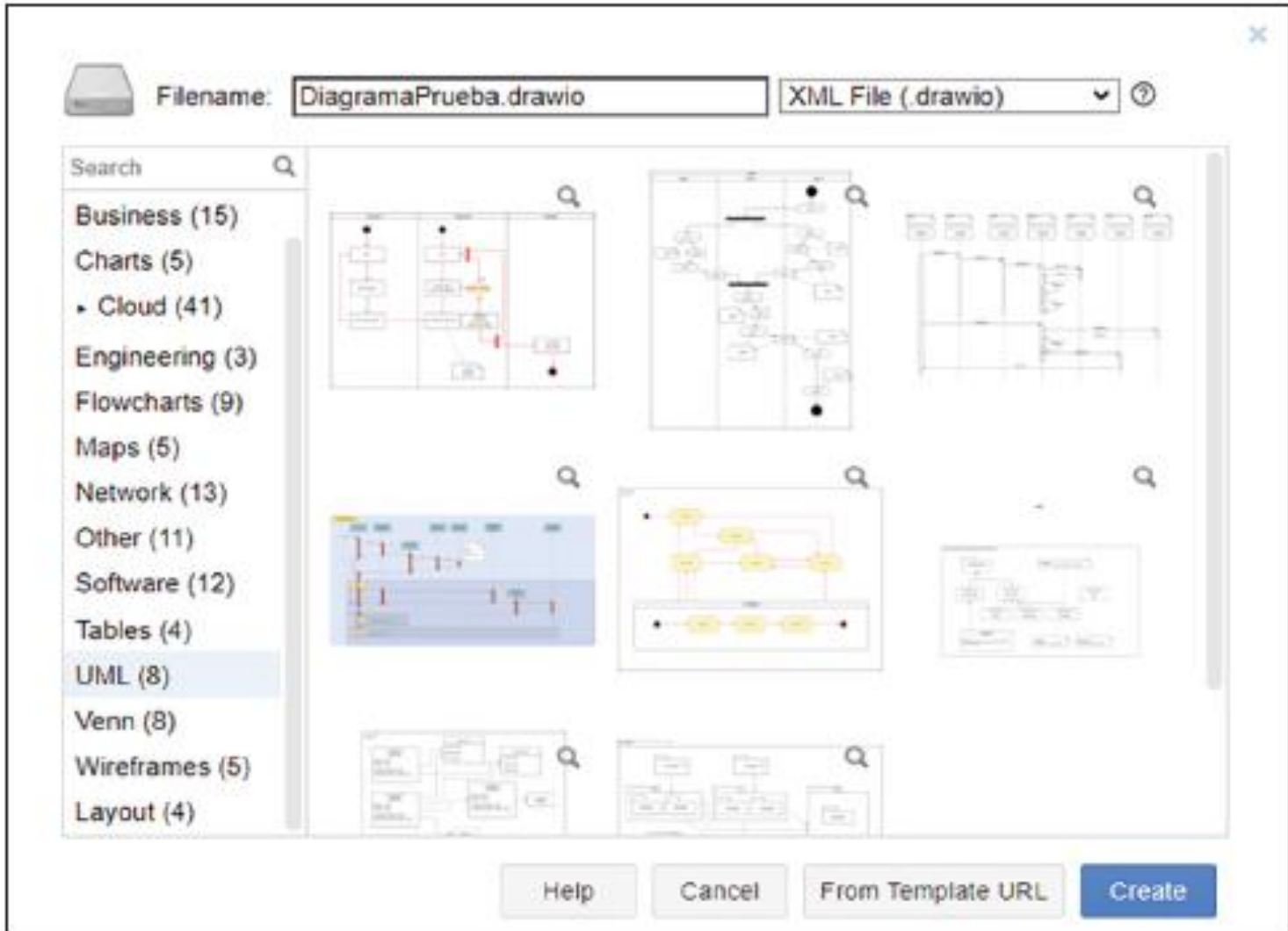


Figura 5.32. Ventana en la que se da un nombre al diagrama y se deja el tipo y la extensión por defecto.

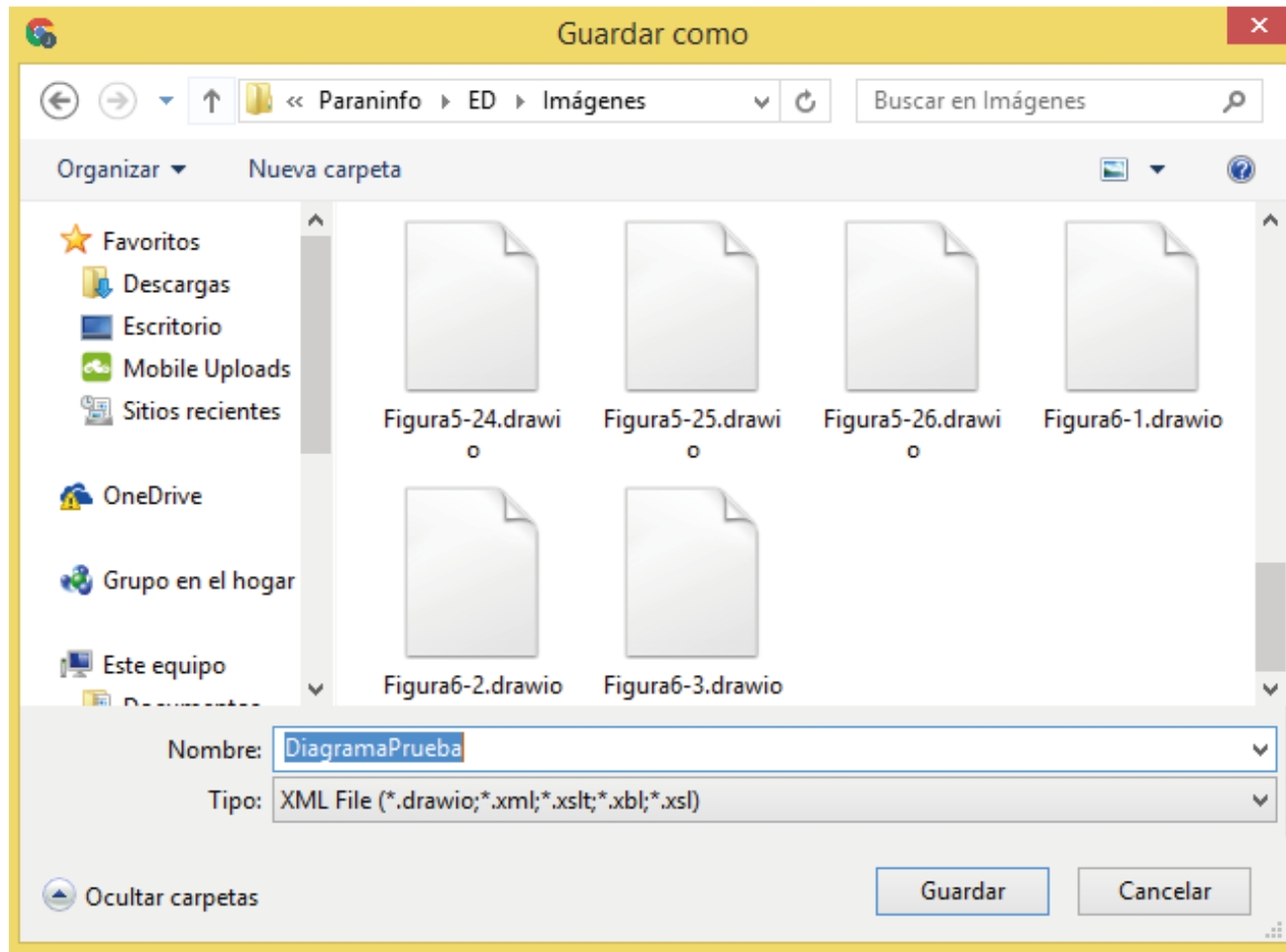


Figura 5.33. Ventana en la que se indica la ubicación donde se desea guardar el diagrama en el equipo y se asigna su nombre.

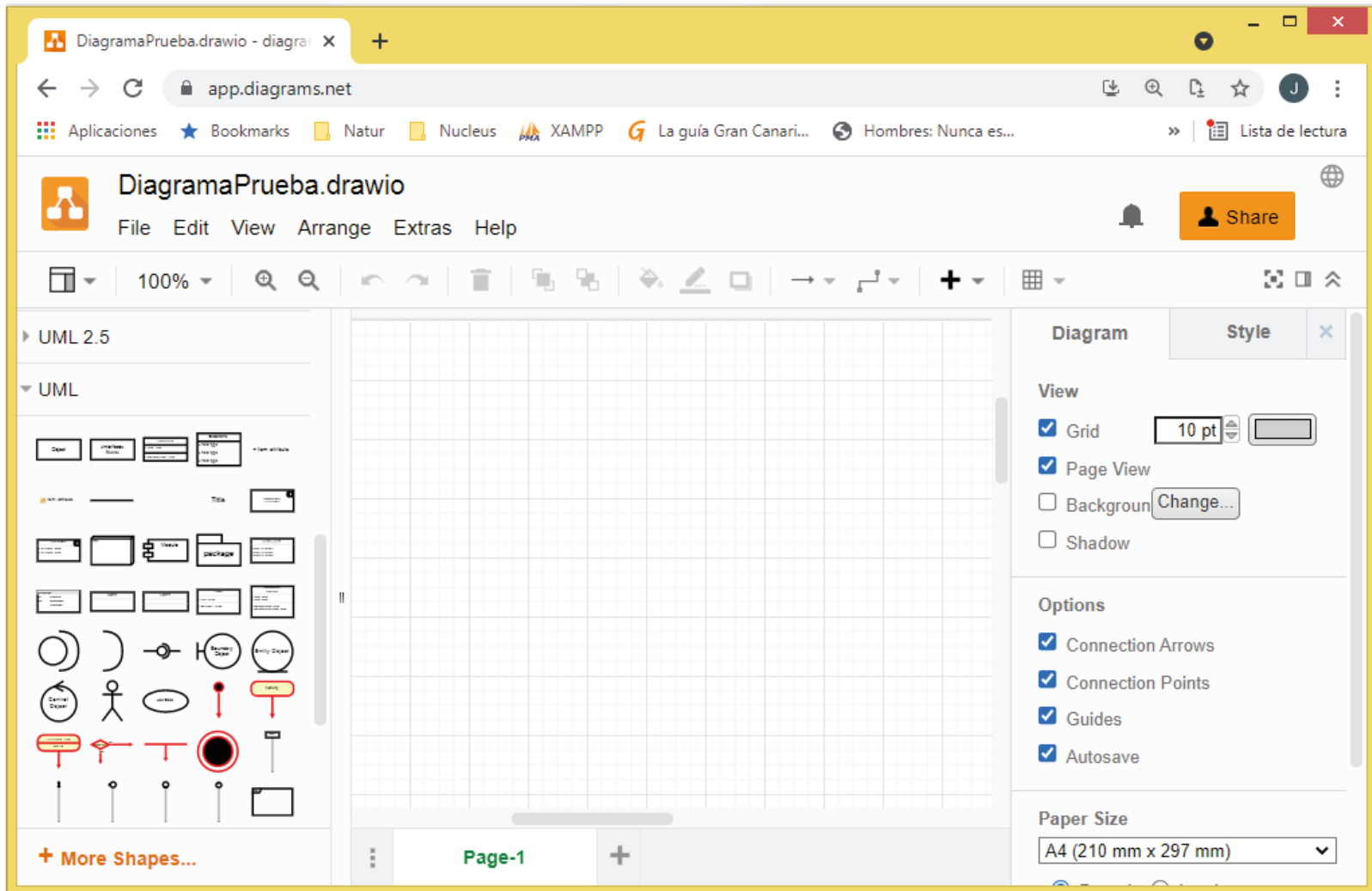


Figura 5.34. Pantalla principal de diagrams.net, donde se distinguen tres áreas de izquierda a derecha: área de paletas, área de edición y área de propiedades.

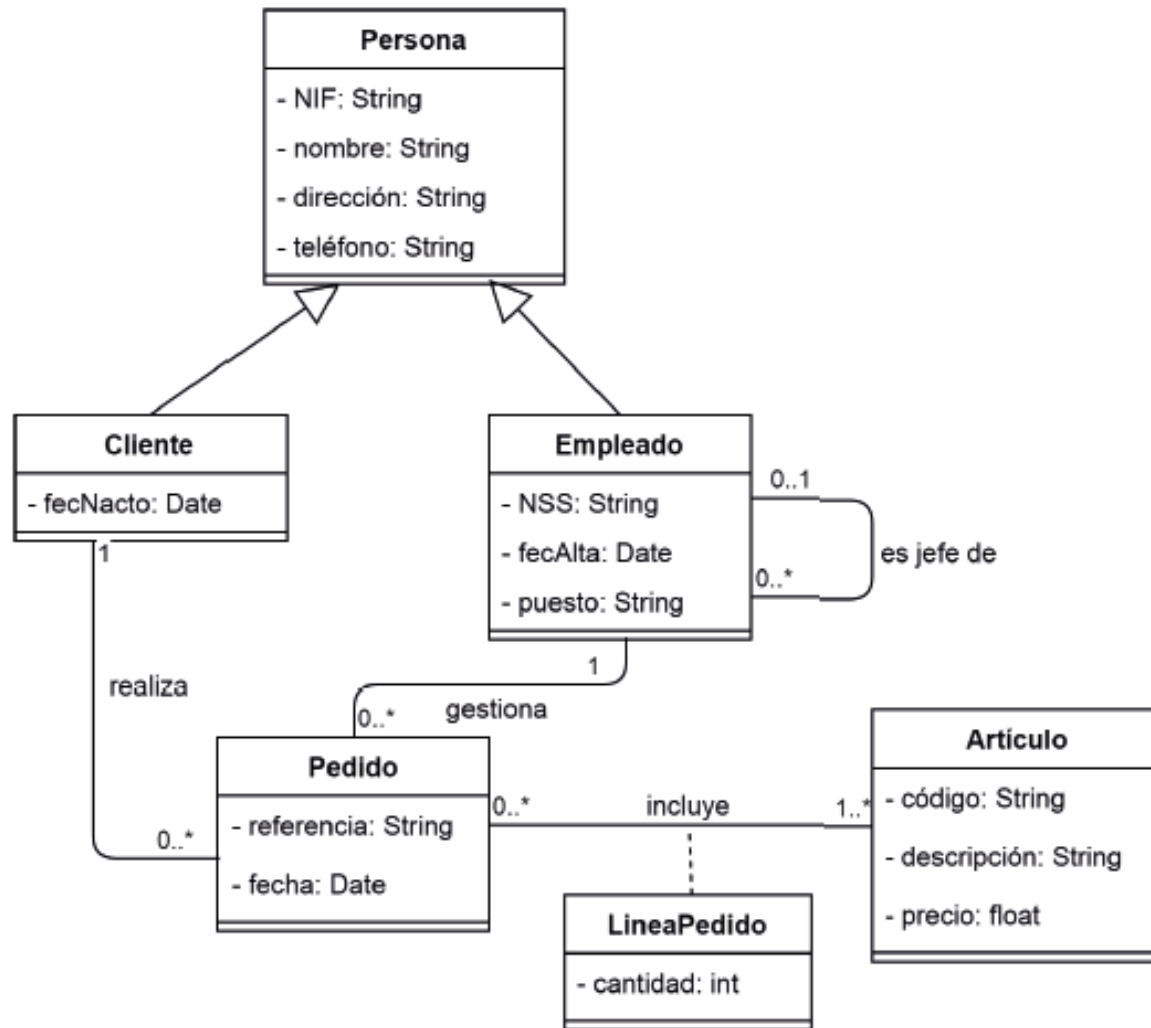


Figura 5.35. Diagrama de clases creado con diagrams.net que refleja el personal y los clientes de una empresa y los pedidos que estos hacen de los artículos que vende la empresa.

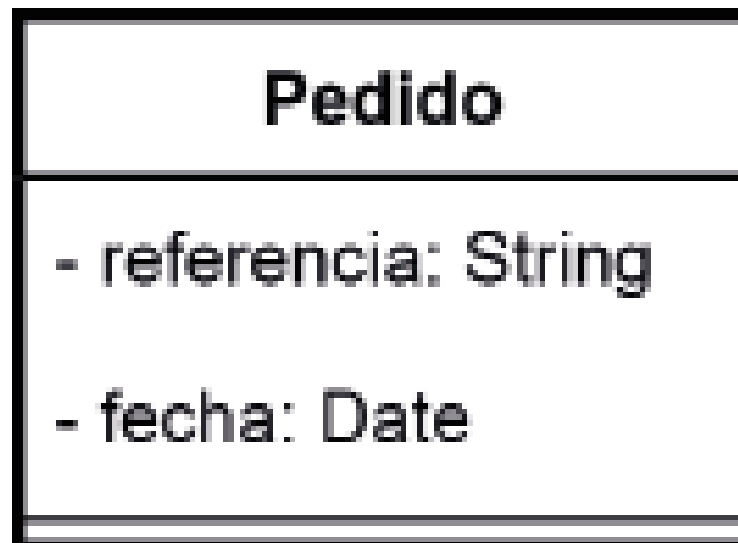


Figura 5.36. Clase Pedido creada con diagrams.net con dos atributos (*referencia* y *fecha*) y sin métodos.



Figura 5.37. Elemento de la paleta UML para crear una asociación entre dos clases.

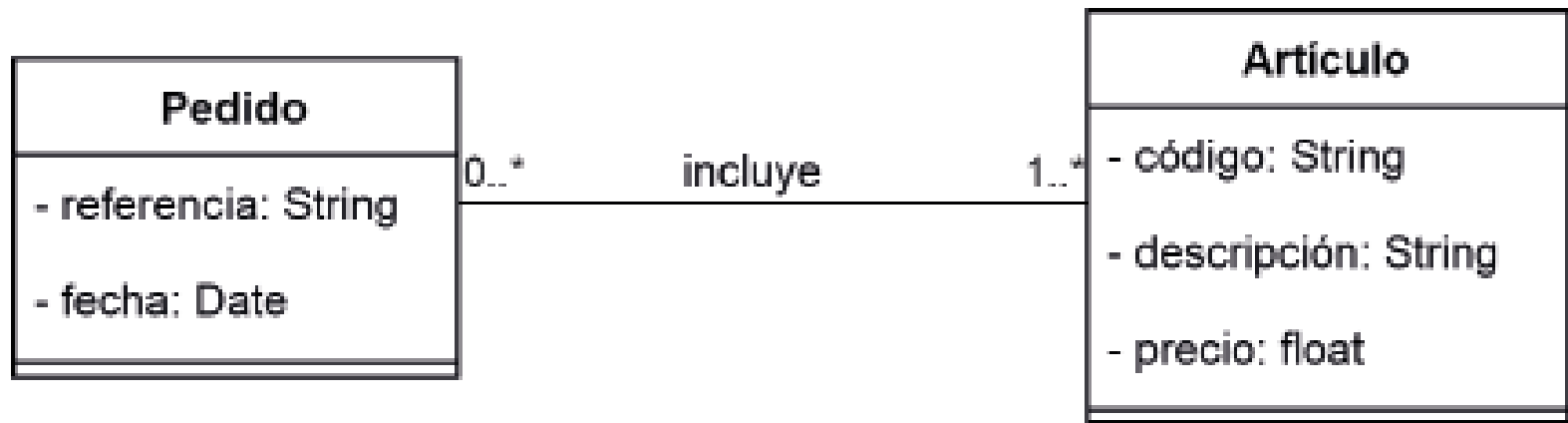


Figura 5.38. Parte del diagrama de clases en el que se muestran las clases *Pedido* y *Artículo* unidas por medio de una asociación llamada *incluye*.

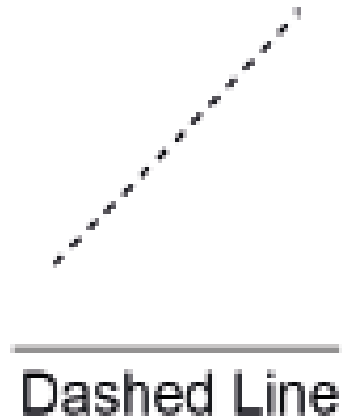


Figura 5.39. Elemento de la paleta *General* para crear una línea discontinua.



Figura 5.40. Elemento de la paleta *UML* para crear una relación de generalización-especialización.

5.6.1. Creación de diagramas de clases con Papyrus

- ❑ Para ver la vista Papyrus en Eclipse: Window → Perspective → Open Perspective → Other → <Papyrus>
- ❑ Para crear un proyecto: File → New → Papyrus Project

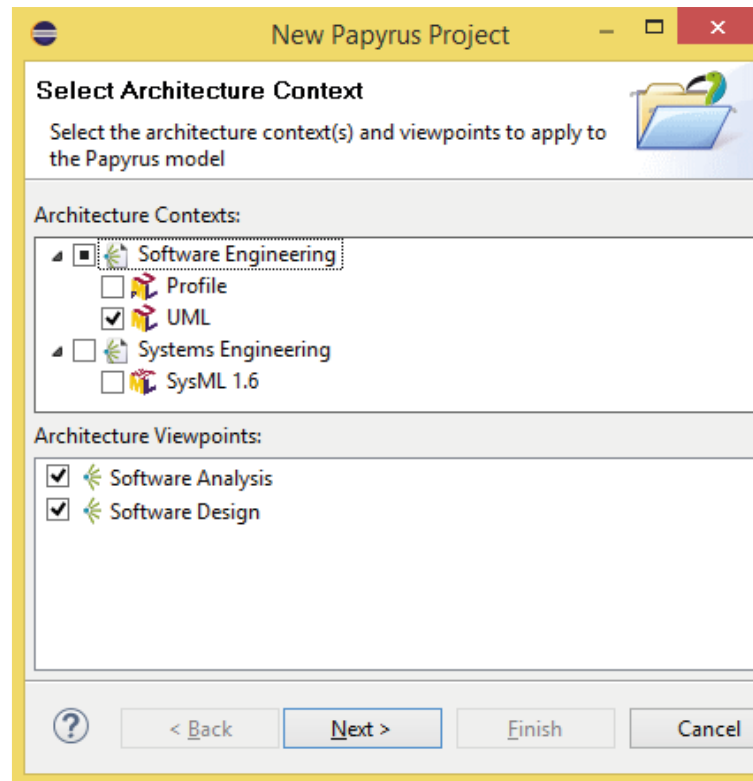


Figura 5.41. Al crear un proyecto en Papyrus para elaborar un diagrama de clases, se debe seleccionar el contexto *UML*.

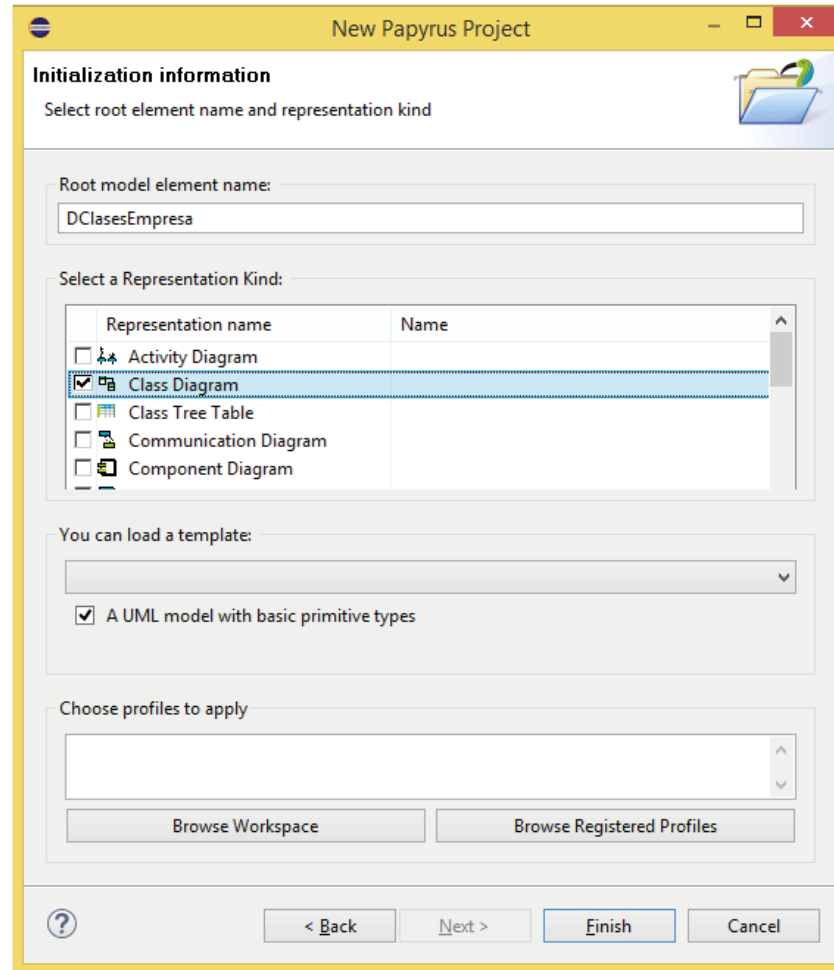


Figura 5.42. Como último paso en el proceso de creación de un proyecto Papyrus, se han de seleccionar el tipo o los tipos de diagramas UML que se desean crear en el proyecto.

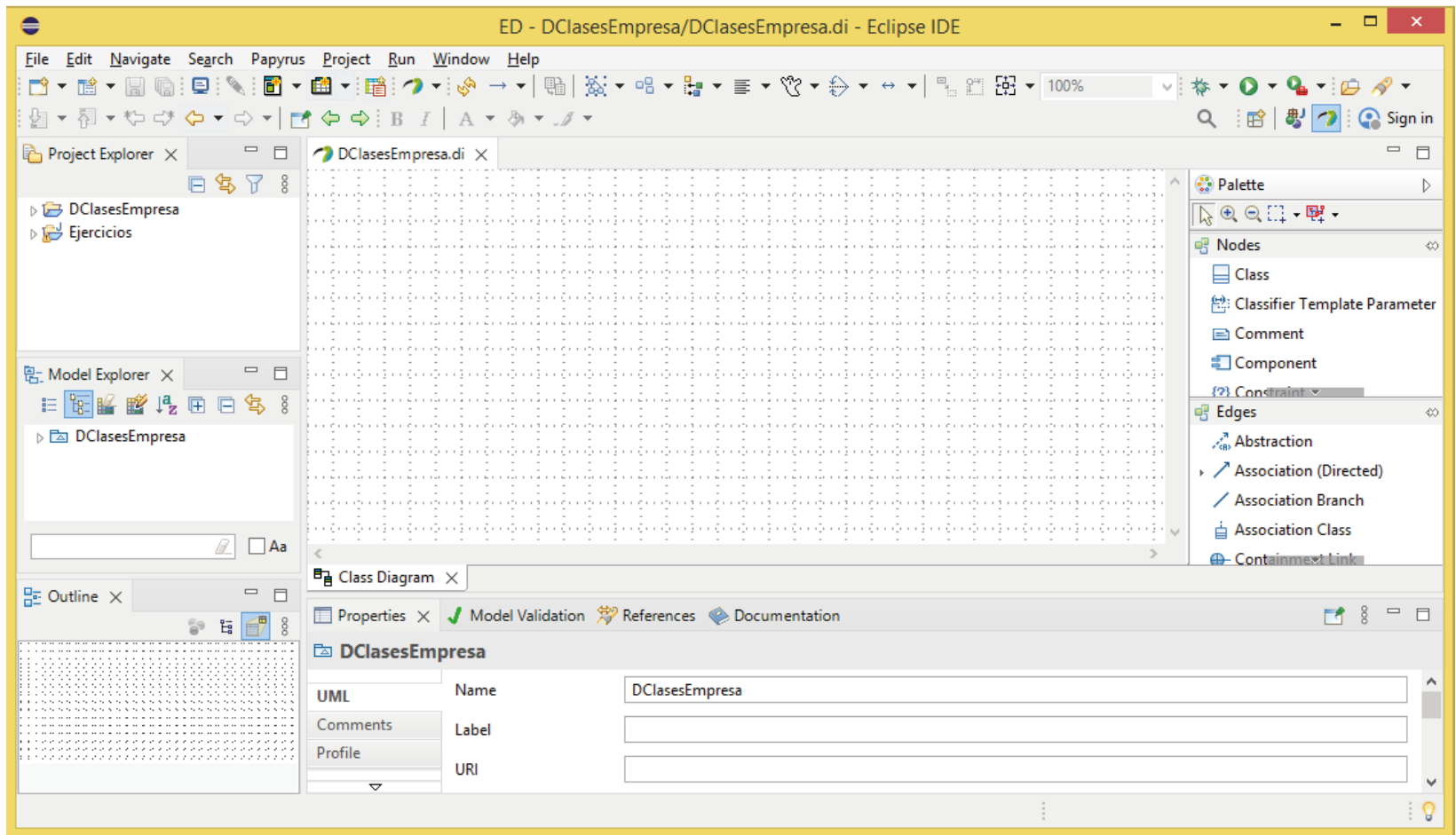


Figura 5.43. Vista *Papyrus* de Eclipse con los elementos necesarios para crear diagramas de clases repartidos en diferentes áreas.

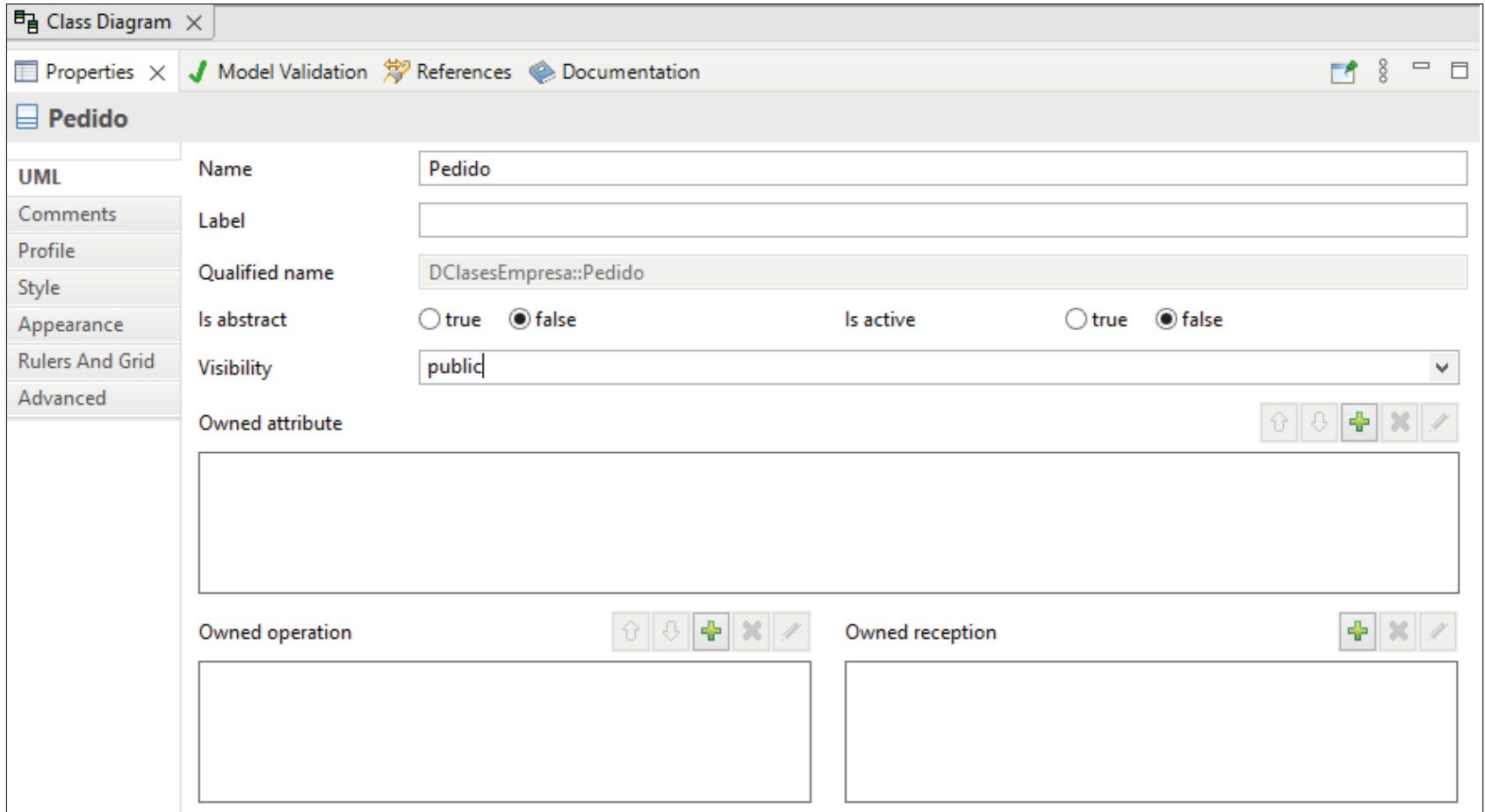


Figura 5.44. En la sección *Properties*, se pueden ver y modificar las propiedades de una clase (su nombre, visibilidad, etc.).

Create a new Property

Name: referencia

Label:

Is derived: ☐ true ☒ false

Is ordered: ☐ true ☒ false

Is static: ☐ true ☒ false

Visibility: private

Multiplicity: 1

Aggregation: none

Is derived union: ☐ true ☒ false

Is read only: ☐ true ☒ false

Is unique: ☒ true ☐ false

Type: String

Default value: <Undefined>

Subsetting property:

Redefined property:

UML Comments

OK Cancel

Figura 5.45. Ventana que muestra las propiedades del atributo referencia de la clase *Pedido*: su nombre, tipo de dato y visibilidad son las más importantes.

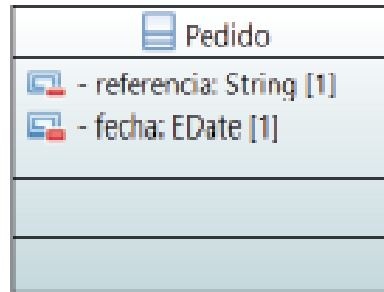


Figura 5.46. Representación de la clase *Pedido* en un diagrama de clases UML creado con Papyrus SysML.

The screenshot shows the 'Properties' window for the association 'LíneaPedido'. The window is divided into several sections:

- UML**: Contains the 'Name' field set to 'LíneaPedido'.
- Comments**: Empty.
- Profile**: Empty.
- Style**: Empty.
- Appearance**: Empty.
- Advance**: Contains 'Is abstract' (radio buttons for true/false, false selected), 'Is derived' (radio buttons for true/false, false selected), and 'Visibility' (dropdown set to 'public').
- Rulers And Grid**: Empty.
- Advanced**: Contains 'Owned attribute' (list with 'cantidad : Integer') and two 'Member End' sections.

The first 'Member End' section is for the 'Artículo' class:

- Name: 1..*
- Label: (empty)
- Type: Artículo
- Owner: Association
- Navigable: (radio buttons for true/false, false selected)
- Aggregation: none
- Multiplicity: 1..*

The second 'Member End' section is for the 'Pedido' class:

- Name: 0..*
- Label: (empty)
- Type: Pedido
- Owner: Association
- Navigable: (radio buttons for true/false, false selected)
- Aggregation: none
- Multiplicity: *

Figura 5.47. Propiedades de la asociación con clase asociativa *LíneaPedido* entre *Pedido* y *Artículo*.

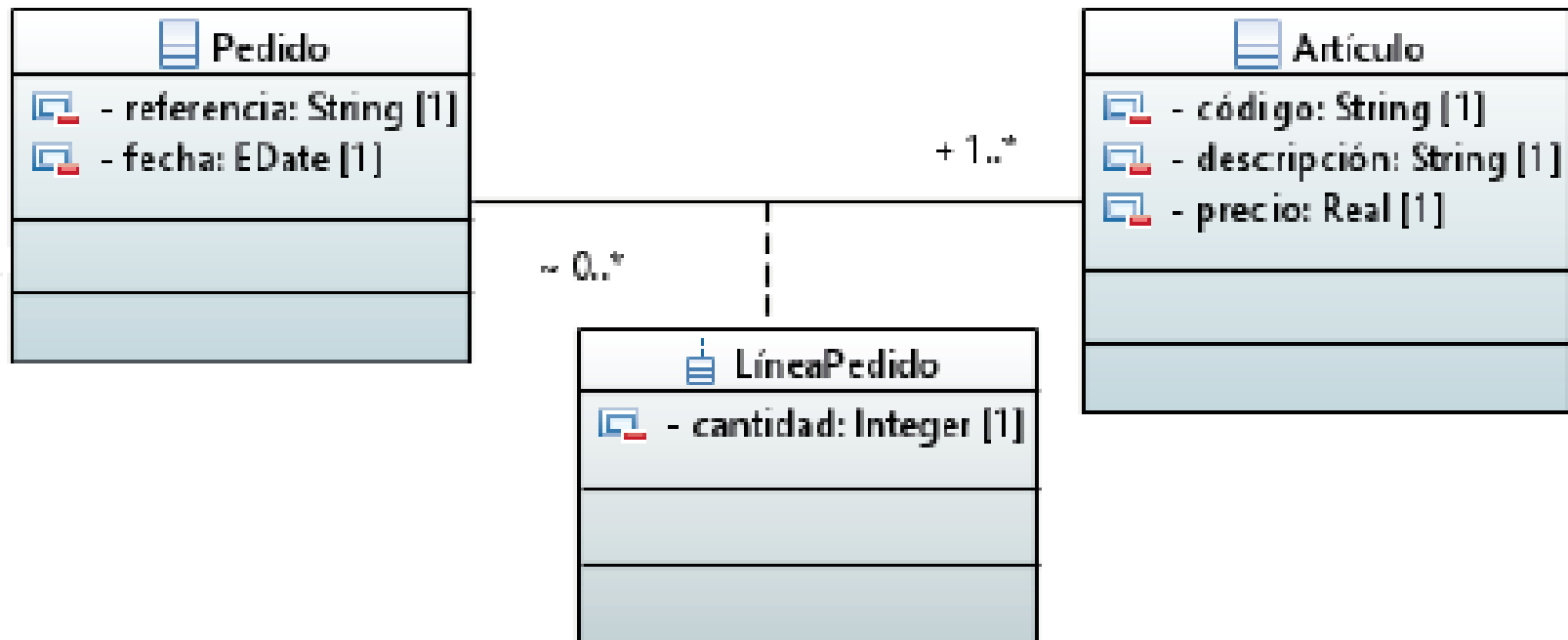


Figura 5.48. Representación de las clases *Pedido* y *Artículo* unidas por medio de una asociación con una clase asociativa vinculada llamada *LíneaPedido*.

Properties X

/ realiza

UML

Comments

Profile

Style

Appearance

Advance

Rulers And Grid

Advanced

Name	realiza	
Label		
Visibility	public	
Member End		
Name	pedido	
Label		
Type	Pedido	
Owner	Association	
Navigable	<input type="radio"/> true <input checked="" type="radio"/> false	
Aggregation	none	
Multiplicity	*	

Member End		
Name	cliente	
Label		
Type	Cliente	
Owner	Association	
Navigable	<input type="radio"/> true <input checked="" type="radio"/> false	
Aggregation	none	
Multiplicity	1	

Figura 5.49. Propiedades de la asociación realiza entre *Cliente* y *Pedido*.

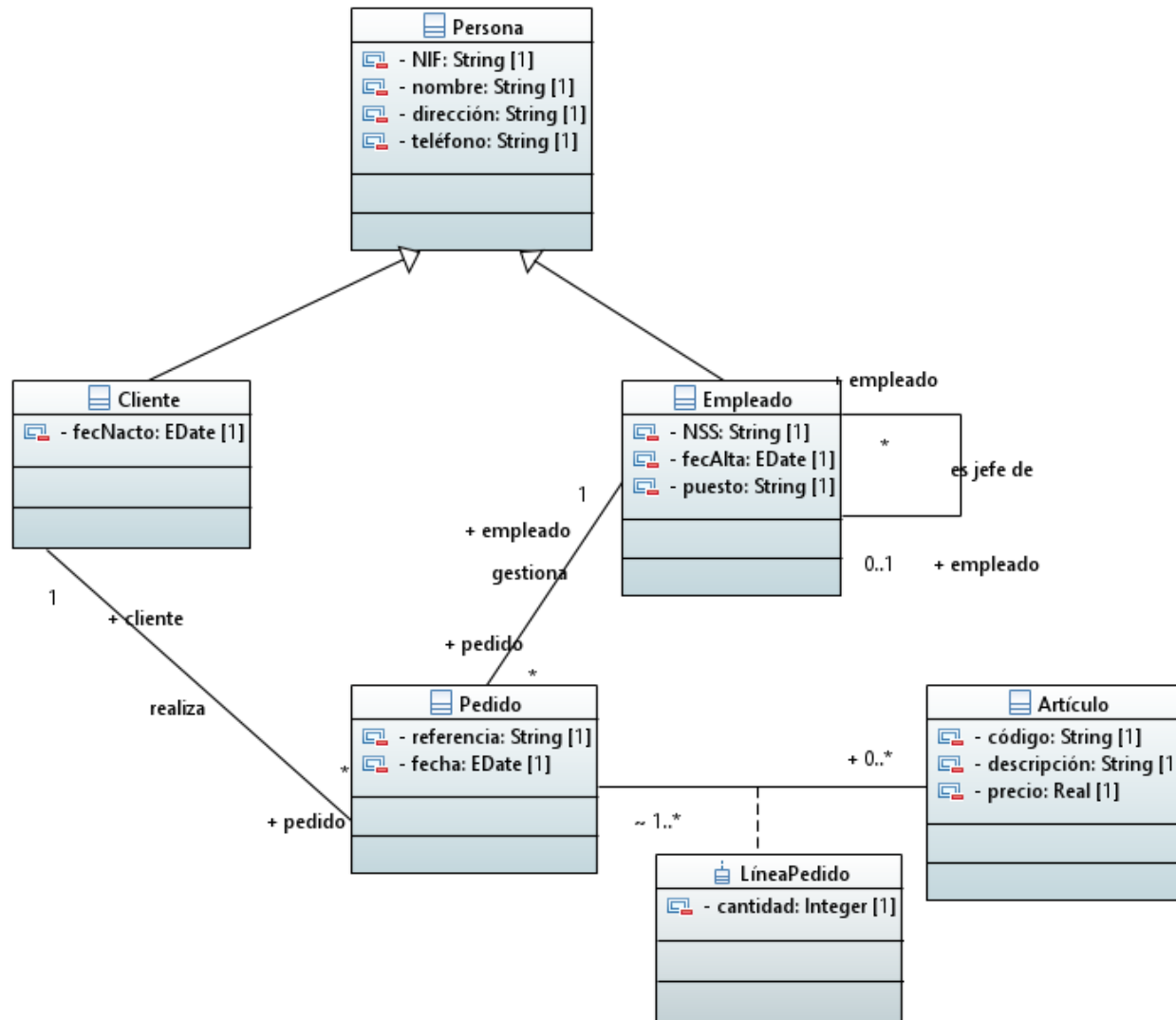


Figura 5.50. Diagrama de clases creado con Papyrus en Eclipse que refleja los empleados y clientes de una empresa y los pedidos que estos hacen de artículos que vende la empresa.

5.6.2. Creación de diagramas de clases con Modelio

<https://www.modelio.org/downloads/download-modelio.html>

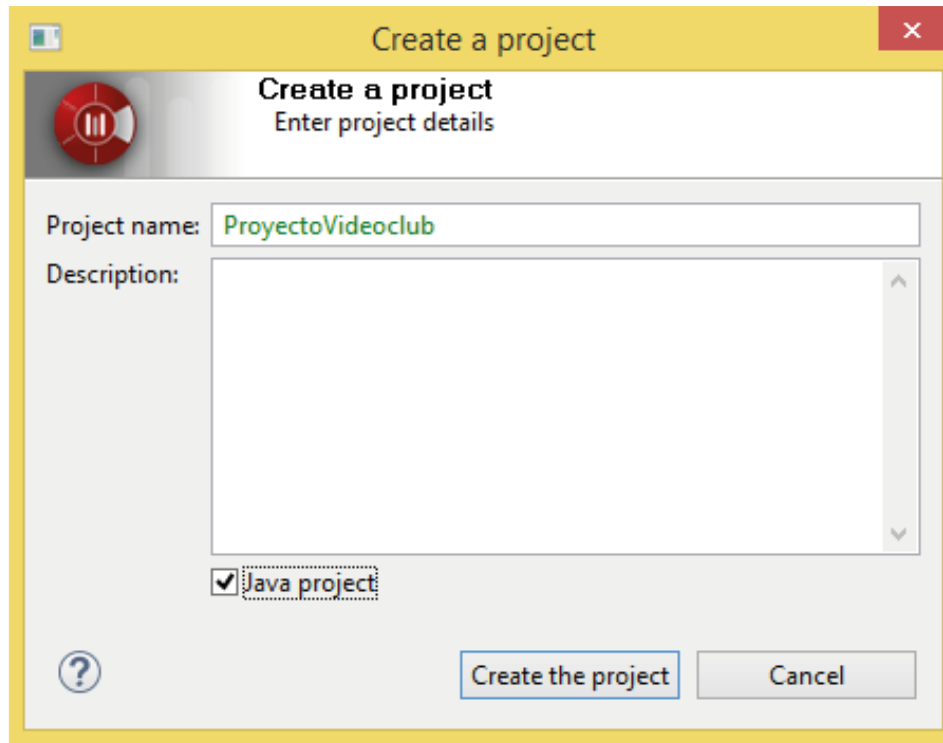


Figura 5.51. Ventana de creación de un proyecto en Modelio, en la que se debe dar un nombre al proyecto que se está creando.

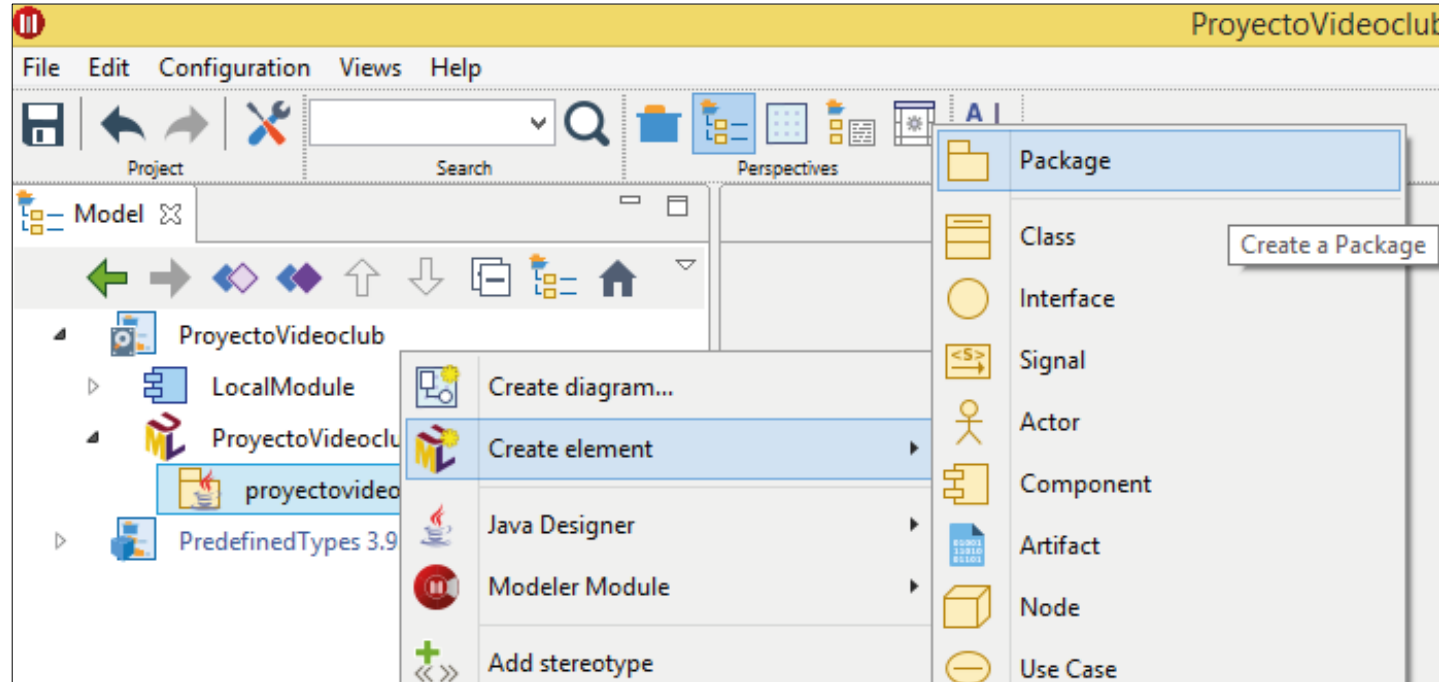


Figura 5.52. Para crear un diagrama en Modelio, una vez creado un proyecto, hay que seleccionar la carpeta correspondiente al proyecto y la opción del menú contextual *Create element > Package*.

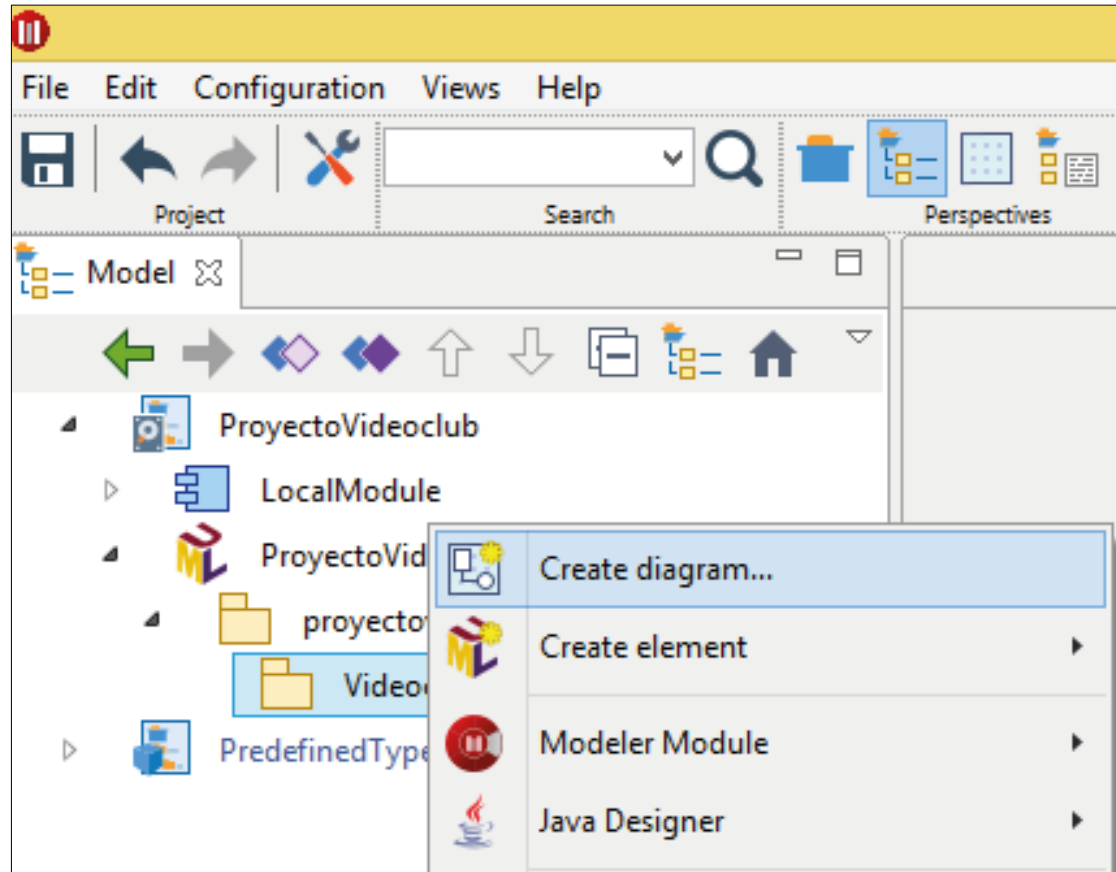


Figura 5.53. Para crear un diagrama en Modelio, tras crear un paquete, hay que seleccionar el paquete y elegir la opción del menú contextual *Create diagram*.

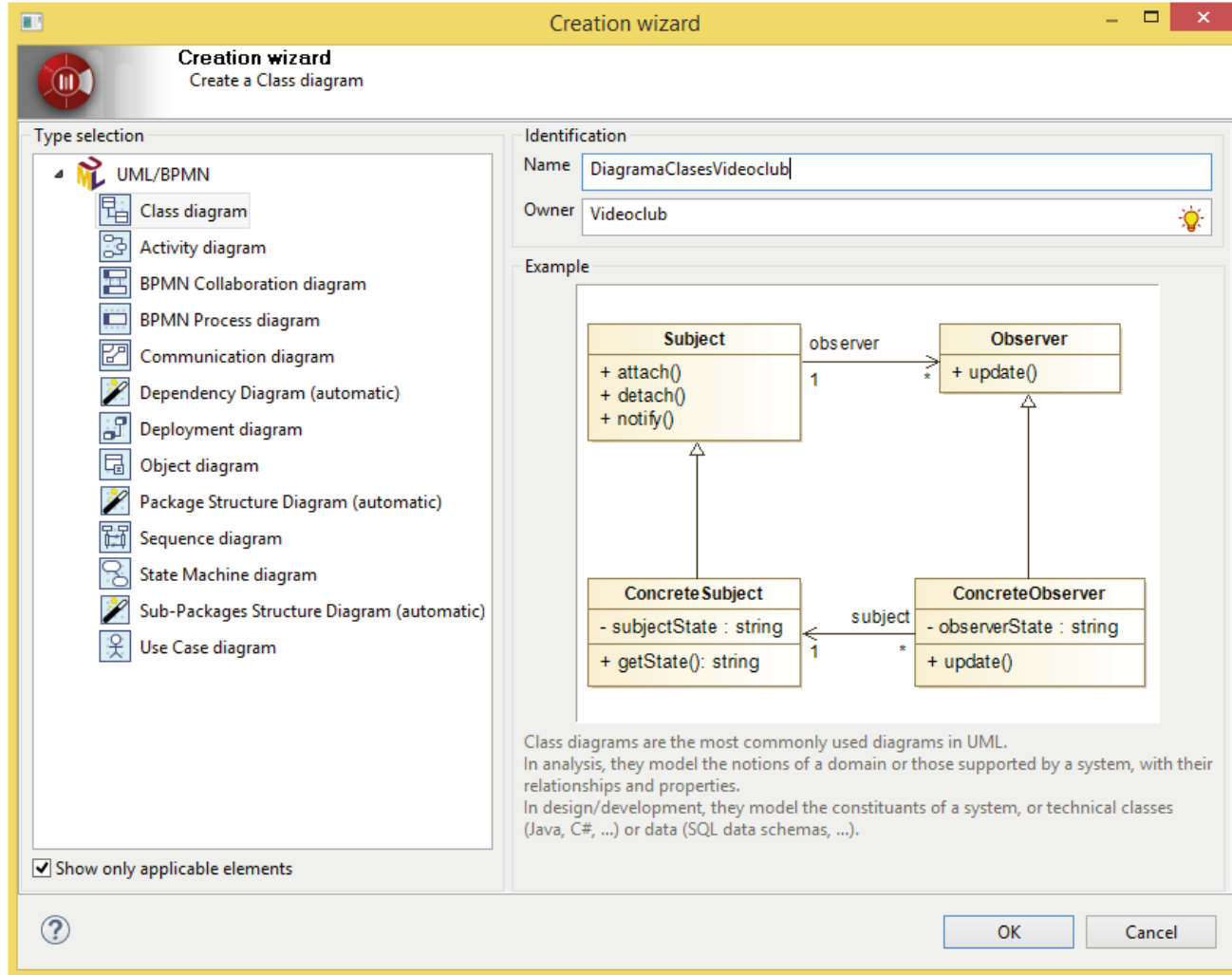


Figura 5.54. Ventana que se muestra antes de proceder a la creación de un diagrama en la que se selecciona el tipo de diagrama que se desea crear y se le da un nombre.


Property	
Name	código
Type	 integer
Visibility	Private
Multiplicity min	1
Multiplicity max	1
Value	
Access mode	Read/Write
Type constraint	
Abstract	<input type="checkbox"/>
Class	<input type="checkbox"/>
Derived	<input type="checkbox"/>
Ordered	<input type="checkbox"/>
Unique	<input type="checkbox"/>
Target is class	<input type="checkbox"/>

Figura 5.55. Ventana en la que se visualizan y editan las propiedades de un atributo en la pestaña *Properties*. En este caso, se muestran las propiedades del atributo código de la clase *Película*.

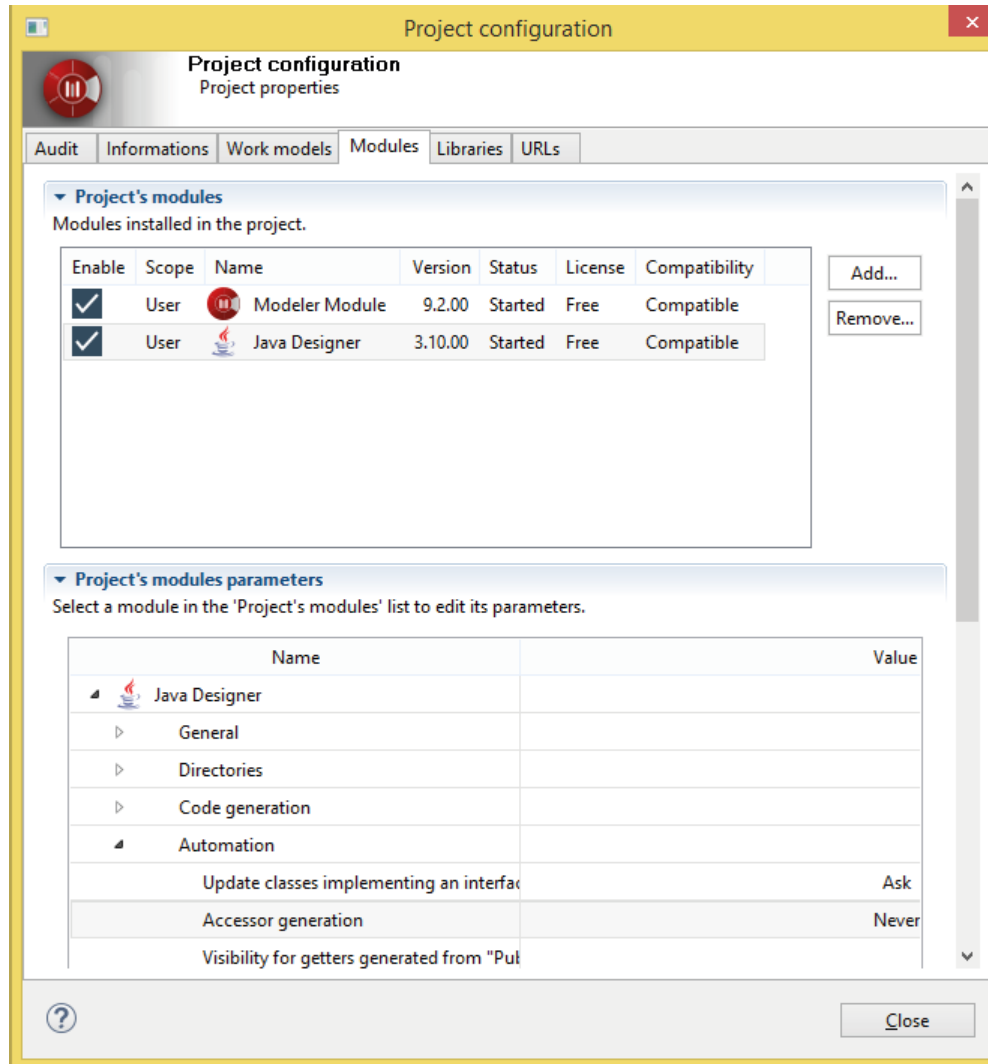


Figura 5.56. Se puede indicar a Modelio que no cree automáticamente métodos de consulta y acceso para todos los atributos de una clase modificando la propiedad *Accessor generation* del módulo Java Designer al valor *Never*.

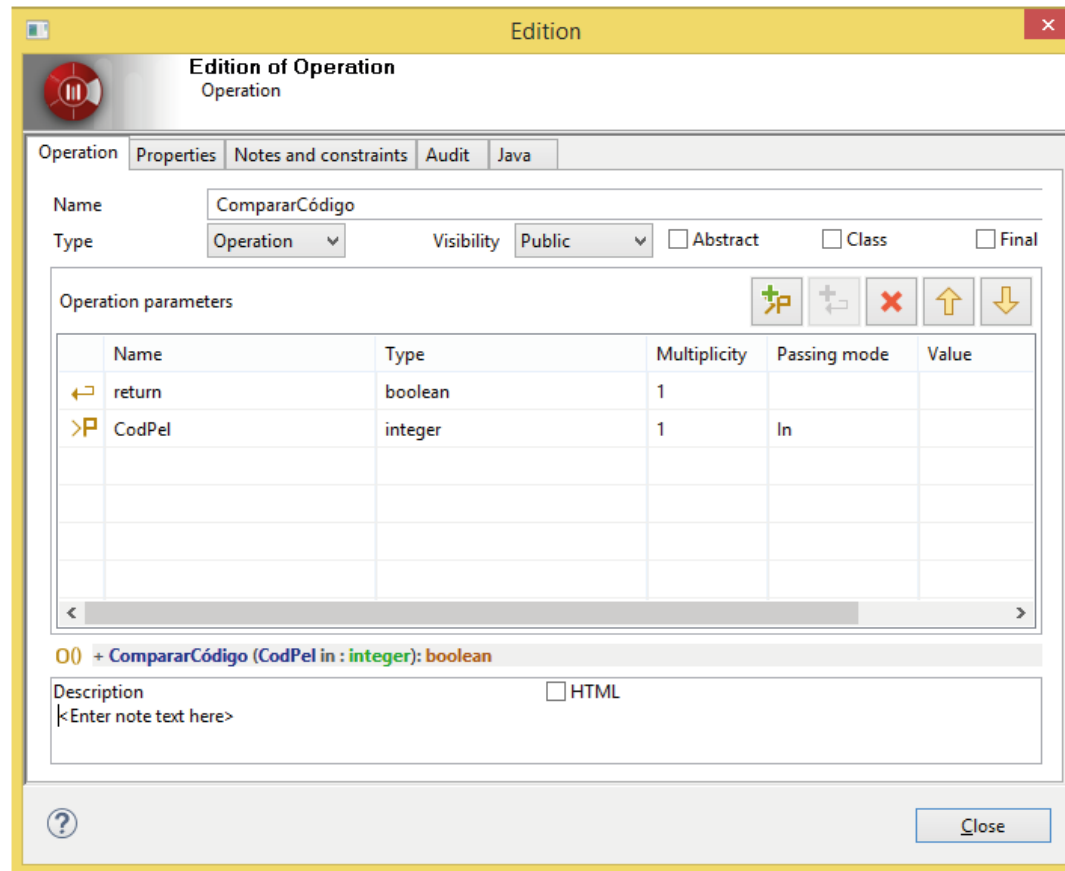


Figura 5.57. Ventana en la que se visualizan y modifican las propiedades de los métodos de una clase al seleccionar el método correspondiente y la opción del menú contextual *Edit element*. En la pestaña *Operation*, se debe asignar al método su nombre, tipo y visibilidad, y se deben definir sus parámetros y valor de retorno, si procede.

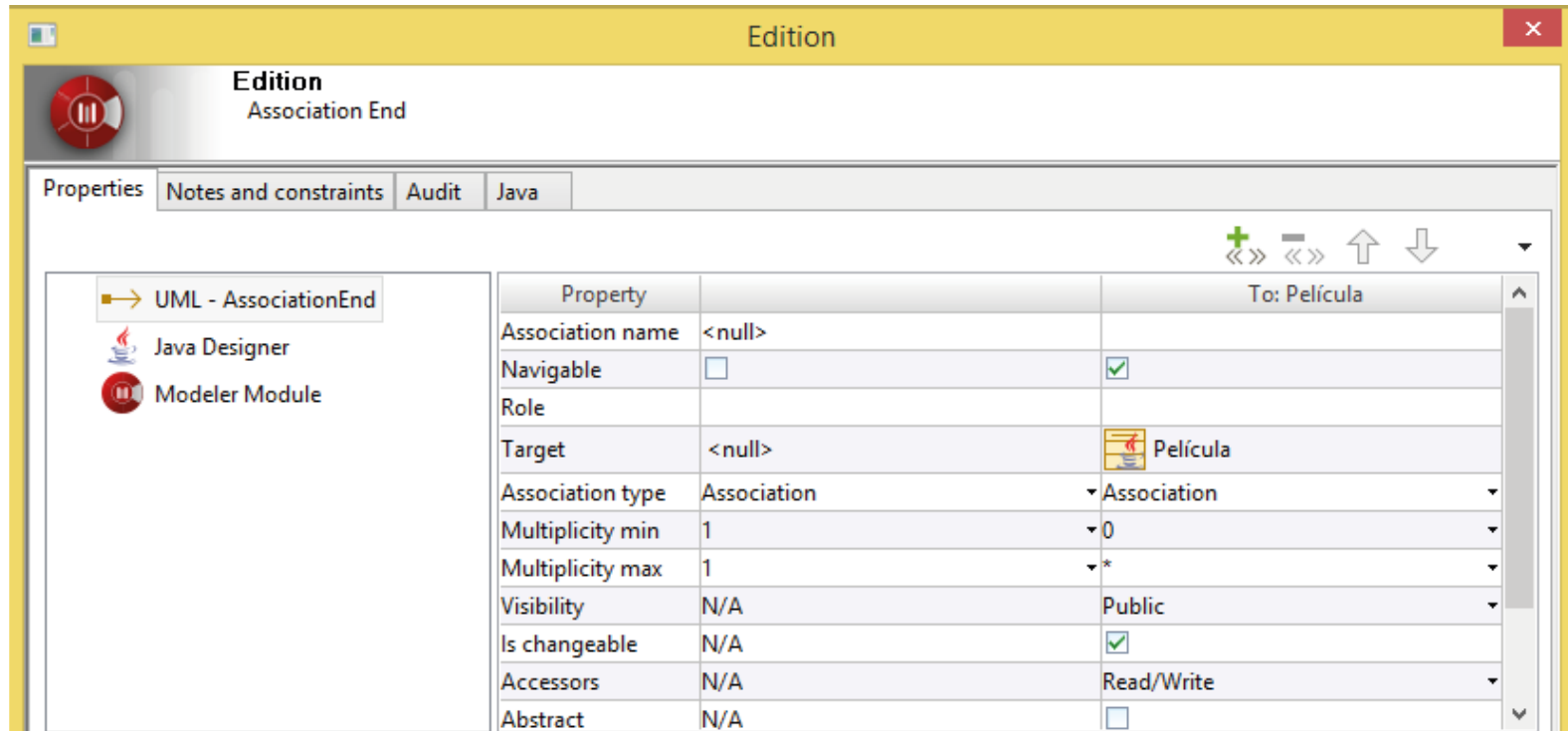


Figura 5.58. Ventana en la que se visualizan y modifican las propiedades de las relaciones establecidas entre clases, seleccionando la relación correspondiente y la opción del menú contextual *Edit element*. En la pestaña *Properties* se puede asignar un nombre a la relación, los roles, cambiar el tipo de asociación e indicar las multiplicidades mínimas y máximas al lado de cada clase.

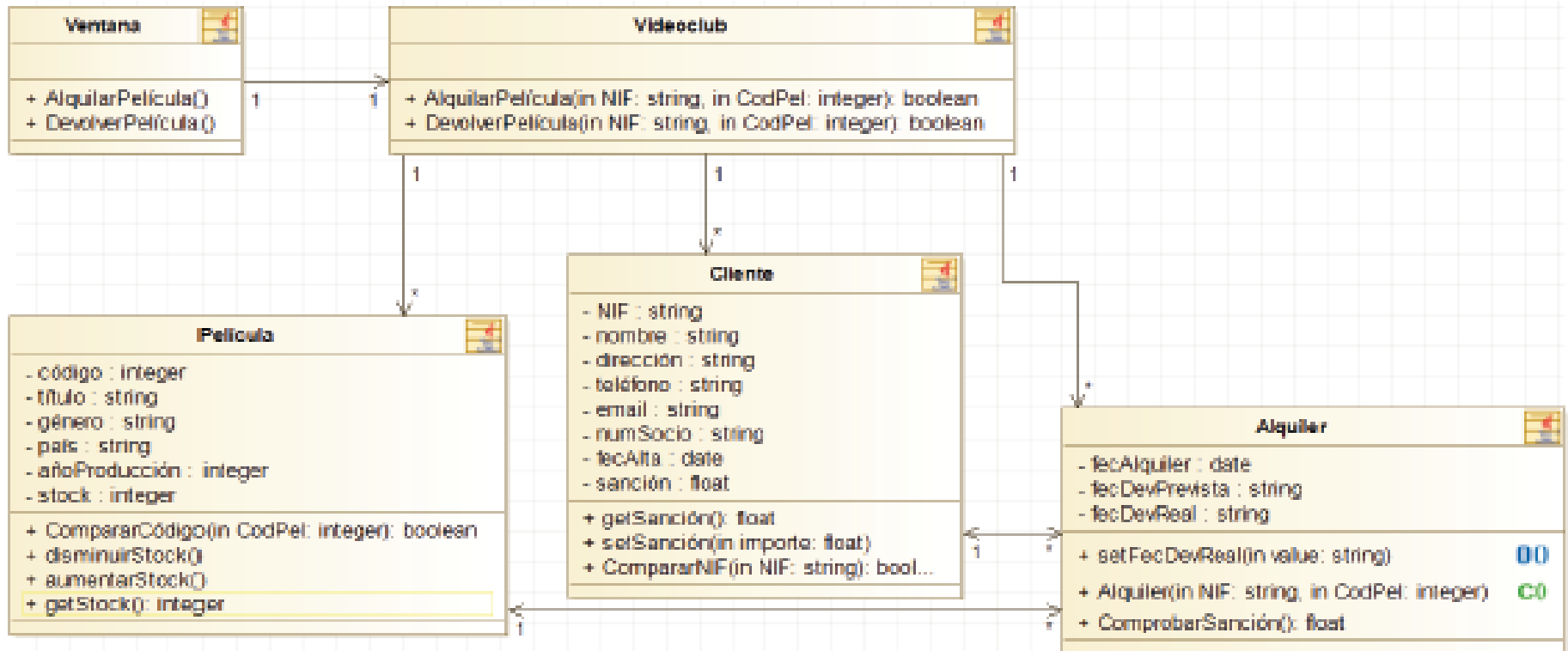


Figura 5.59. Diagrama de clases creado en Modelio que refleja la información que es necesario gestionar en un videoclub.

5.7. Generación de código a partir de diagramas de clases

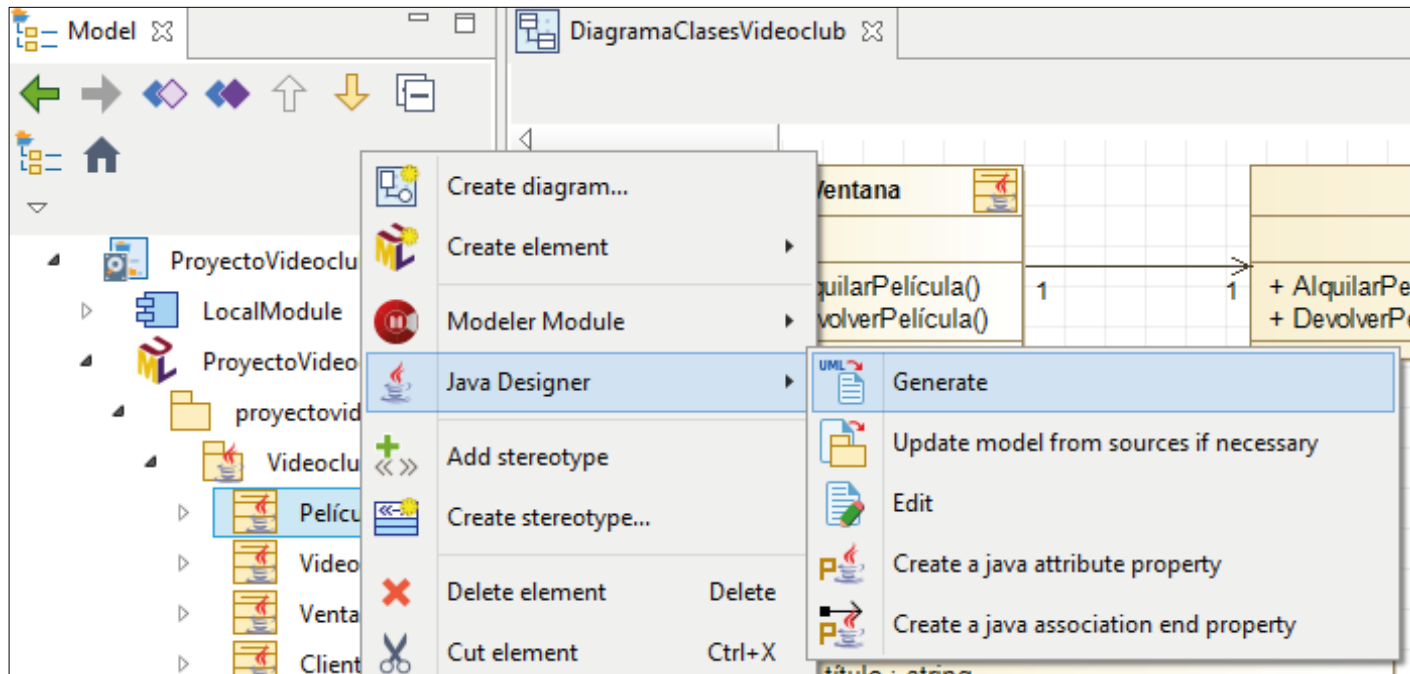


Figura 5.60. En Modelio es posible generar el código Java correspondiente a una clase activando la opción del menú contextual *Java Designer > Generate*.



The screenshot shows the Modelio IDE interface. At the top, there are two tabs: 'DiagramaClasesVideoclub' and 'Película'. The 'Película' tab is active, displaying the Java code for the 'Película' class. The code is as follows:

```
package Videoclub;

import java.util.ArrayList;
import java.util.List;
import com.modeliosoft.modelio.javadesigner.annotations.objid;

@objid ("0ae7ce52-fd0f-4663-9f0f-f8652b91f153")
public class Película {
    @objid ("5eac3cfc-b310-4790-b543-52759fc5108b")
    private int código;

    @objid ("29e1c7a0-899f-44b6-93b1-bb79c4c6b198")
    private String título;

    @objid ("a0864089-17c6-435f-b950-29b991a027ef")
    private String género;

    @objid ("8a25c2c8-1c3e-4417-9980-8f0555191f8d")
    private String país;

    @objid ("cc58639b-cb6d-4e5e-81c4-5d8d9b8b636d")
    private int añoProducción;

    @objid ("d62bce6a-1ef1-461b-97c0-2f253ed51fc0")
    private int stock;

    @objid ("8874f9ba-2127-43d3-929d-b2dbe028137c")
    public List<Alquiler> = new ArrayList<Alquiler> ();

    @objid ("333be3d4-3e77-48d0-b7a9-67c2b0df7f47")
    public boolean CompararCódigo(int CodPel) {
    }

    @objid ("c51f22d1-b788-4298-b882-8e81cc9e7ca4")
```

Figura 5.61. En Modelio se puede visualizar el código Java correspondiente a una clase seleccionando la opción del menú contextual *Java Designer > Edit*.

5.8. Generación de diagramas de clases a partir de código (ingeniería inversa)

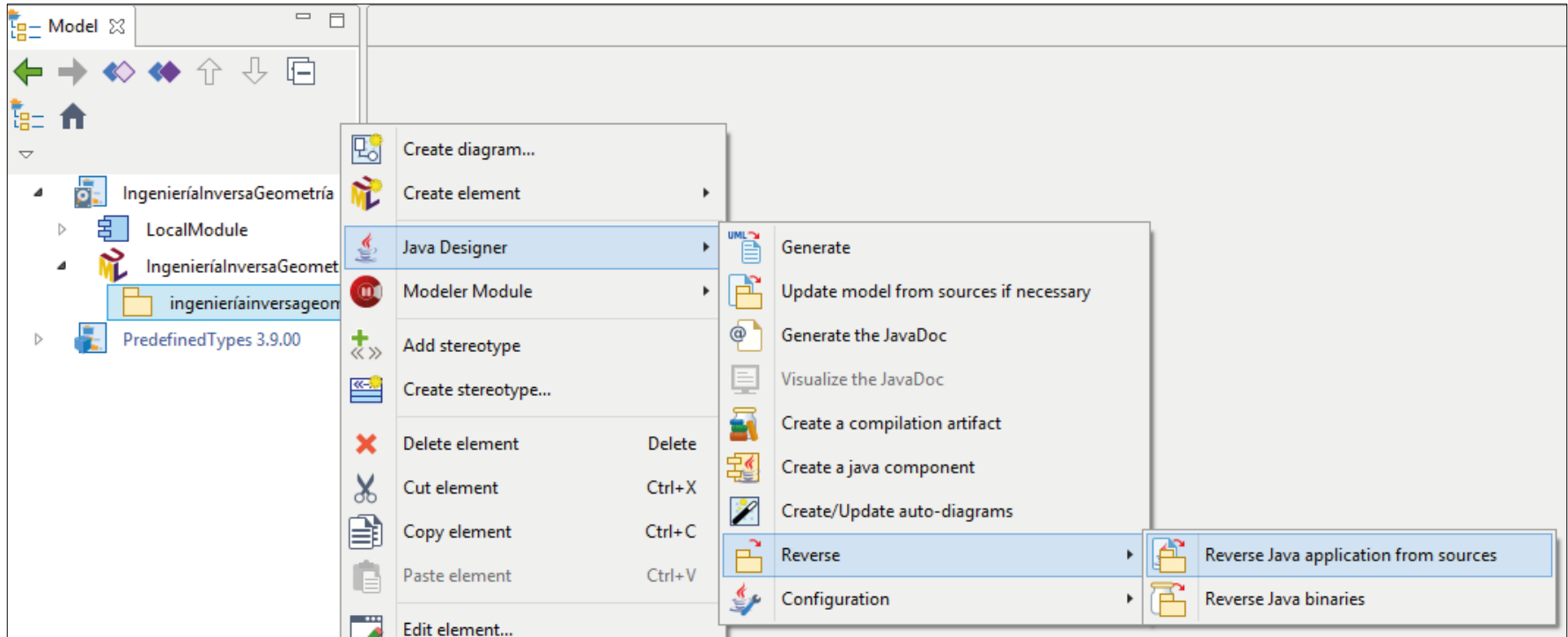


Figura 5.62. Para realizar ingeniería inversa a partir de archivos con código fuente de Java en Modelio, se debe crear un proyecto Java y activar la opción del menú contextual *Java Designer* > *Reverse* > *Reverse Java application from sources*.

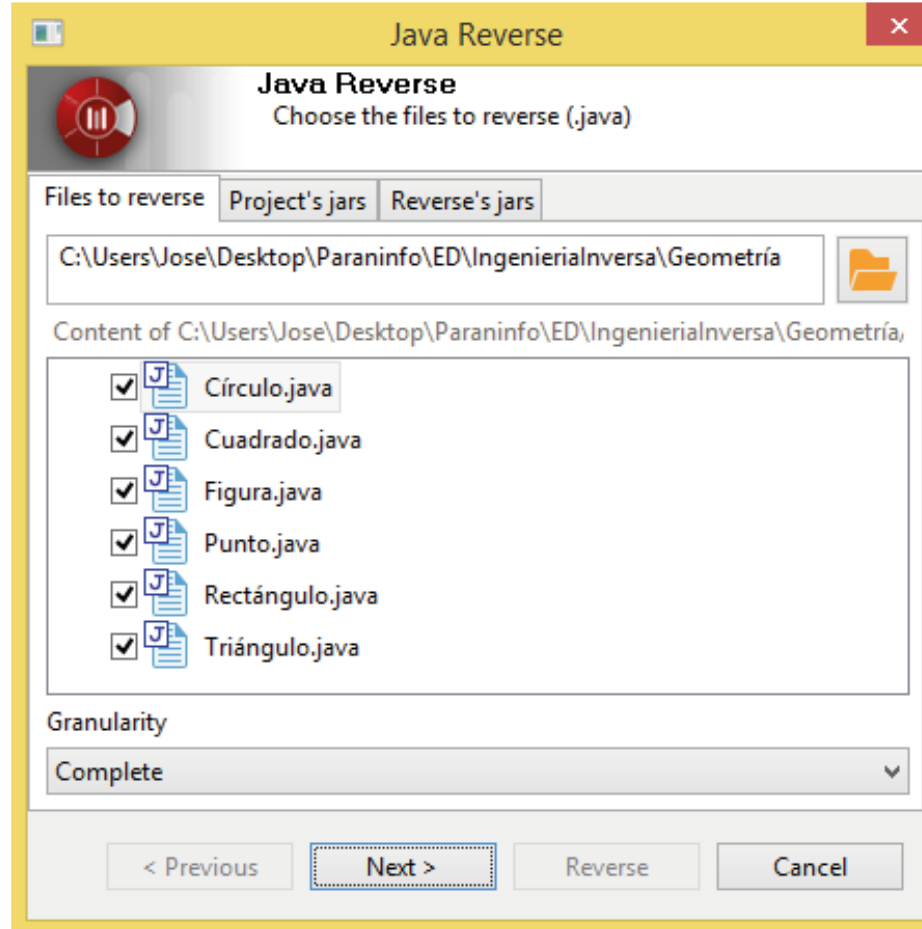


Figura 5.63. Ventana en la que se seleccionan las clases para las que se desea aplicar la ingeniería inversa.

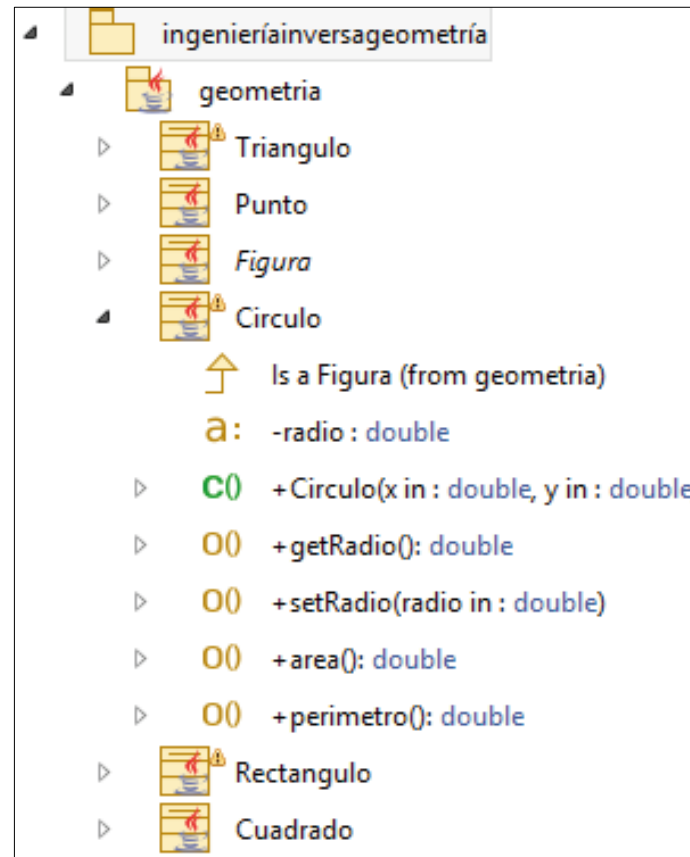


Figura 5.64. En el explorador del modelo se muestran todas las clases que se han creado a partir del código fuente. Estas se pueden desplegar para visualizar sus elementos (atributos, operaciones y relaciones).

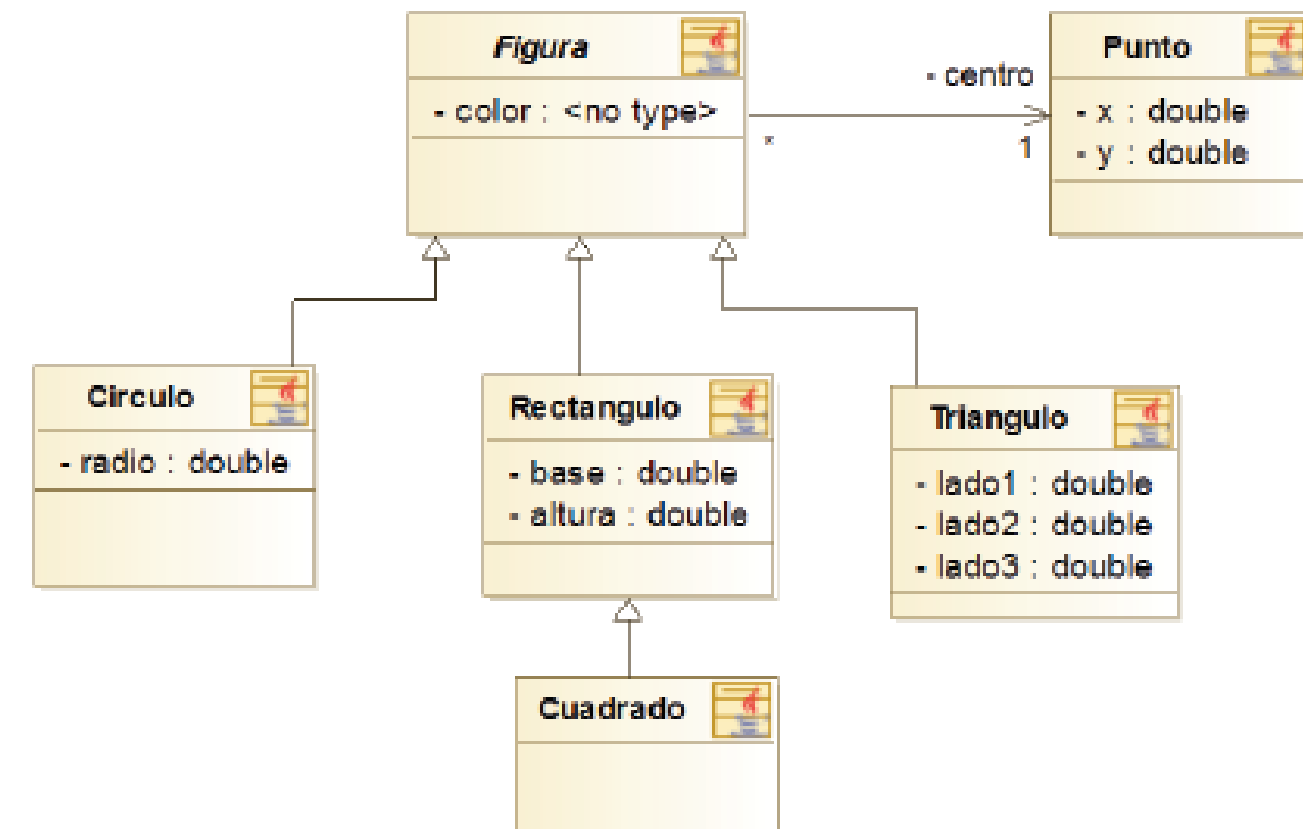


Figura 5.65. Diagrama de clases sin métodos resultado del proceso de ingeniería inversa en Modelio.