

UT03. DISEÑO ORIENTADO A OBJETOS. ELABORACIÓN DE DIAGRAMAS ESTRUCTURALES.

Entornos de Desarrollo

1 DAW – C.I.F.P. Carlos III - Cartagena

Índice

1.- Introducción a la orientación a objetos

2.- Conceptos de orientación a objetos

2.1.- Ventajas de la orientación a objetos.

2.2.- Clases, atributos y métodos.

2.3.- Visibilidad

2.4.- Objetos. Instanciación.

3.- UML

3.1.- Tipos de diagramas UML.

3.2.- Herramientas para la elaboración de diagramas UML.

3.3.- Diagramas de clases.

3.4.- Relaciones entre clases.

3.5.- Paso de los requisitos de un sistema al diagrama de clases.

3.6.- Generación de código a partir del diagrama de clases.

3.7.- Generación de la documentación.

4.- Ingeniería inversa.

3. UML

- **UML-*Unified Modeling Language*** - *Lenguaje Unificado de Modelado*
- Conjunto de herramientas que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos.
- Modelos que representan el sistema desde diferentes perspectivas
- Usado por
 - **OMT** - Object Modeling Technique (Rumbaugh et al.)
 - **Método-Booch** (G. Booch)
 - **OOSE** - Object-Oriented Software Engineering (I. Jacobson)
- Existen dos grandes **versiones** de UML:
 - **UML 1.X** → (UML 1.1, 1.2, 1.3, 1.4, 1.5) Desde finales de los 90 hasta el 2005, aproximadamente
 - **UML 2.X** → (UML 2.0 hasta 2.5) la última revisión formal es la de 2.5.1

Padres de UML



Dr. James Rumbaugh



Grady Booch



Dr. Ivar Jacobson

- Razones de uso:
 - Uso de un lenguaje común
 - Documentación del proceso de desarrollo (requisitos, arquitectura, pruebas, versiones...)
 - Puede representar estructuras referentes a la arquitectura del sistema (relaciones entre módulos, nodos en los que se ejecuta en sistemas distribuidos)
 - Modelos precisos, no ambiguos y completos en las decisiones de análisis, diseño e implementación
 - Conexión a lenguajes de programación mediante ingeniería directa e inversa

Tipos de diagramas UML

- **Sistema**

- Conjunto de modelos que describen sus diferentes perspectivas, implementados en una serie de diagramas (representaciones gráficas de una colección de elementos de modelado), a menudo dibujado como un grafo conexo de arcos (relaciones) y vértices (otros elementos del modelo).

- **Elementos**

- **Estructuras:** Son los nodos del grafo y definen el tipo de diagrama.
- **Relaciones:** Son los arcos del grafo que se establecen entre los elementos estructurales.
- **Notas:** Se representan como un cuadro donde podemos escribir comentarios que nos ayuden a entender algún concepto que queramos representar.
- **Agrupaciones:** Se utilizan cuando modelamos sistemas grandes para facilitar su desarrollo por bloques.

Tipos de diagramas UML. II

- **Clasificación**

- **Diagramas de estructura**

- Representan la visión estática del sistema.
 - Se centran en los elementos que deben existir en el sistema modelado.

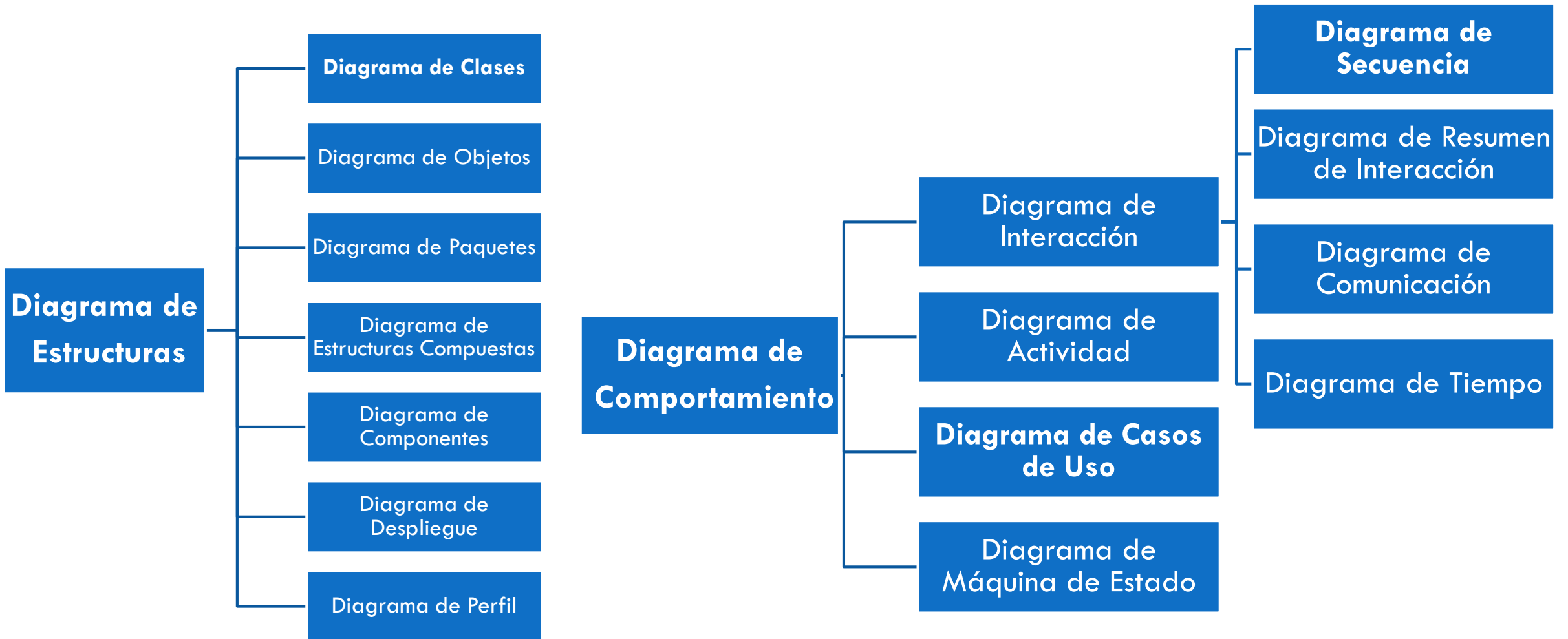
- **Diagramas de comportamiento**

- Representan la parte dinámica del sistema.
 - Se centran en lo que debe suceder en el sistema

- **Diagramas de interacción**

- Derivados de los diagramas de comportamiento más general
 - Se centran en el flujo de control y de datos entre los elementos del sistema modelado

Tipos de diagramas UML. II



Tipos de diagramas UML más utilizados

- **Diagramas de clases:** Muestra las diferentes clases del sistema y cómo se relacionan entre ellas.
- **Diagrama de objetos:** Muestra los objetos y sus relaciones del modelo estático en un momento concreto.
- **Diagramas de casos de uso:** se utiliza para entender el uso del sistema, mediante un conjunto de actores, acciones y relaciones entre ellos.
- **Diagramas de secuencia:** Representación temporal de los objetos y sus relaciones, interacción entre los objetos y los mensajes que intercambian entre sí junto con el orden temporal de los mismos.
- **Diagramas de estado:** analizan los cambios de estado de los objetos. Muestran los estados, eventos, transacciones y actividades de los diferentes objetos.
- **Diagrama de actividad:** se utiliza para mostrar la secuencia de actividades. Muestran el orden en el que se van realizando tareas dentro de un sistema, incluyendo decisiones que surgen en la progresión de los eventos contenidos en la actividad.
- **Diagramas de despliegue:** especifica el hardware sobre el que el sistema se va a ejecutar y cómo se despliega en ese hardware. Se compone de nodos, unidad material capaz de recibir y ejecutar software, y de vínculos físicos entre los nodos, correspondientes a las ramas de la red.
- **Diagrama de paquetes:** Muestra cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. El uso más común es para organizar diagramas de casos de uso y diagramas de clases.

Herramientas para la elaboración de diagramas UML

- Herramientas **CASE** que
 - Permiten desarrollo de los diagramas UML.
 - Cuentan con un entorno **WYSIWYG**
 - Permiten documentar los diagramas
 - Permiten integrarse con otros entornos de desarrollo.

Herramientas para la elaboración de diagramas UML

- Algunas de las herramientas más utilizadas:
 - **Modelio**: herramienta de código abierto compatible con UML2 y BPMN. Permite diseñar amplia variedad de diagramas UML y obtener el código correspondiente. Se puede descargar para Windows, Linux y MacOS
 - **Visual Paradigm for UML (VP-UML)**: Herramienta CASE concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. No es una herramienta gratuita pero incluye una versión para uso no comercial durante un mes
 - **Papyrus**. Plugin de Eclipse Proyecto de código abierto que proporciona un entorno integrado para la edición de modelos UML y SysUML.
 - **ArgoUML**: se distribuye bajo licencia Eclipse. Soporta los diagramas de UML 1.4, y genera código para Java y C++. Para poder ejecutarlo se necesita la plataforma Java. Admite ingeniería directa e inversa.
 - **StarUML**
 - **DrawMagic**

Herramientas para la elaboración de diagramas UML

- **Herramientas online**

- **VisualParadigm:** versión limitada en web. Editor intuitivo, potente y fácil de usar hecho para todos. Permite trabajo colaborativo y contiene gran cantidad de plantillas. Es necesario registrarse
- **Lucidchart**
- **Creately**
- **Draw.io**
- **Genmymodel**

- **Herramientas textuales:** utilizan una notación textual para “escribir” los modelos.

- **PlantUML:** proyecto Open Source
- **yUML**

Diagramas de clases

- Elementos:
 - **Clases.**
 - **Relaciones**, relaciones reales entre los elementos del sistema a los que hacen referencia las clases. Pueden ser de asociación, agregación y herencia.
 - **Notas:** comentarios que nos ayuden a entender algún concepto que queramos representar.
 - **Elementos de agrupación:** las clases y sus relaciones se agrupan en **paquetes**, que a su vez se relacionan entre sí.

Diagrama de clases. Clases

Nombre Clase
-lista de atributos
+lista de métodos()

■ Clase

- Se representa como un rectángulo dividido en tres filas
 - Nombre de la clase
 - Atributos con su visibilidad (privada, por defecto), representada por el signo menos "-"
 - Métodos con su visibilidad (pública, por defecto), representado por el signo más "+".

+	Público
-	Privado
#	Protegido
/	Derivado (se puede combinar con otro)
~	Paquete

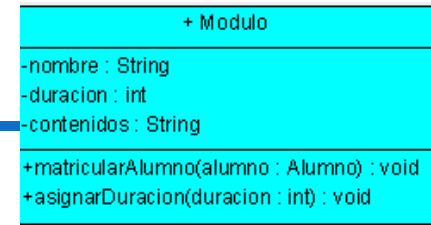
Diagrama de clases. Atributos

+ Modulo
-nombre : String
-duracion : int
-contenidos : String

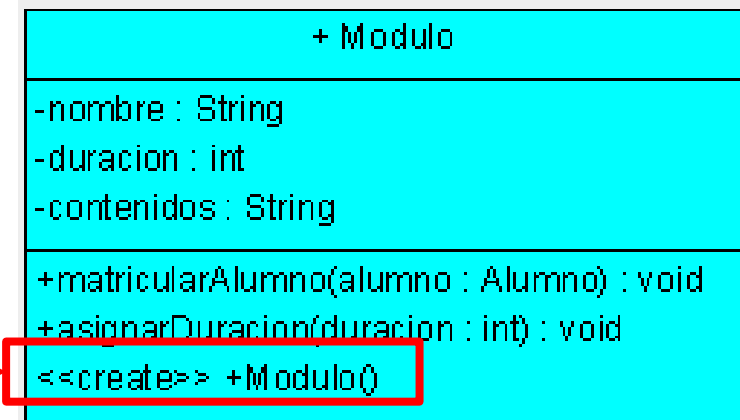
- **Nombre**
- **Tipo**
 - Simple (int, float...)
 - Compuesto, pudiendo incluir otra clase.
- **Visibilidad**
 - **Público:** acceso desde cualquier clase y cualquier parte del programa. Visible para todos
 - **Privado:** acceso desde operaciones de la clase. Visible solo en la clase.
 - **Protegido:** acceso desde operaciones de la clase o de clases derivadas en cualquier nivel. Visible en la clase y las subclases de la clase
 - **Paquete:** acceso desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Visible en las clases del mismo paquete

Diagrama de clases. Métodos

- Definir nombre, parámetros, el tipo que devuelve y su visibilidad y la descripción del método que aparecerá en la documentación que se genere del proyecto

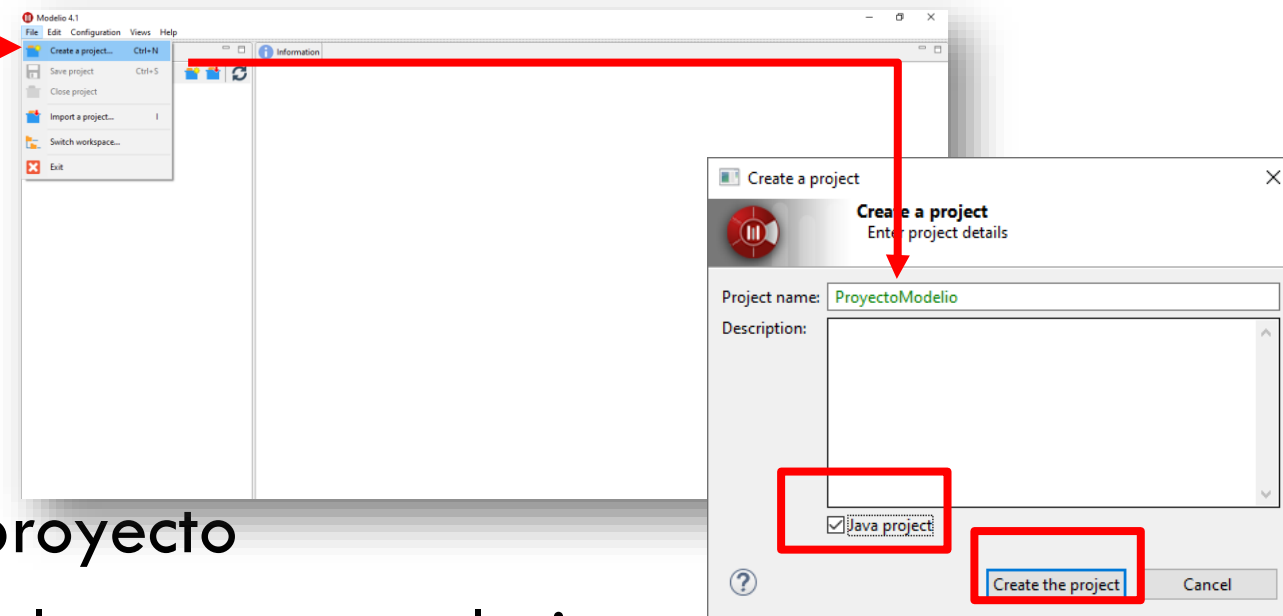


- **Constructor** de la clase
 - No devuelve ningún valor
 - Tiene el **mismo nombre** que la clase
 - **Uso:** Para instanciar objetos de clases
 - Existe una función especial para que el objeto deje de existir

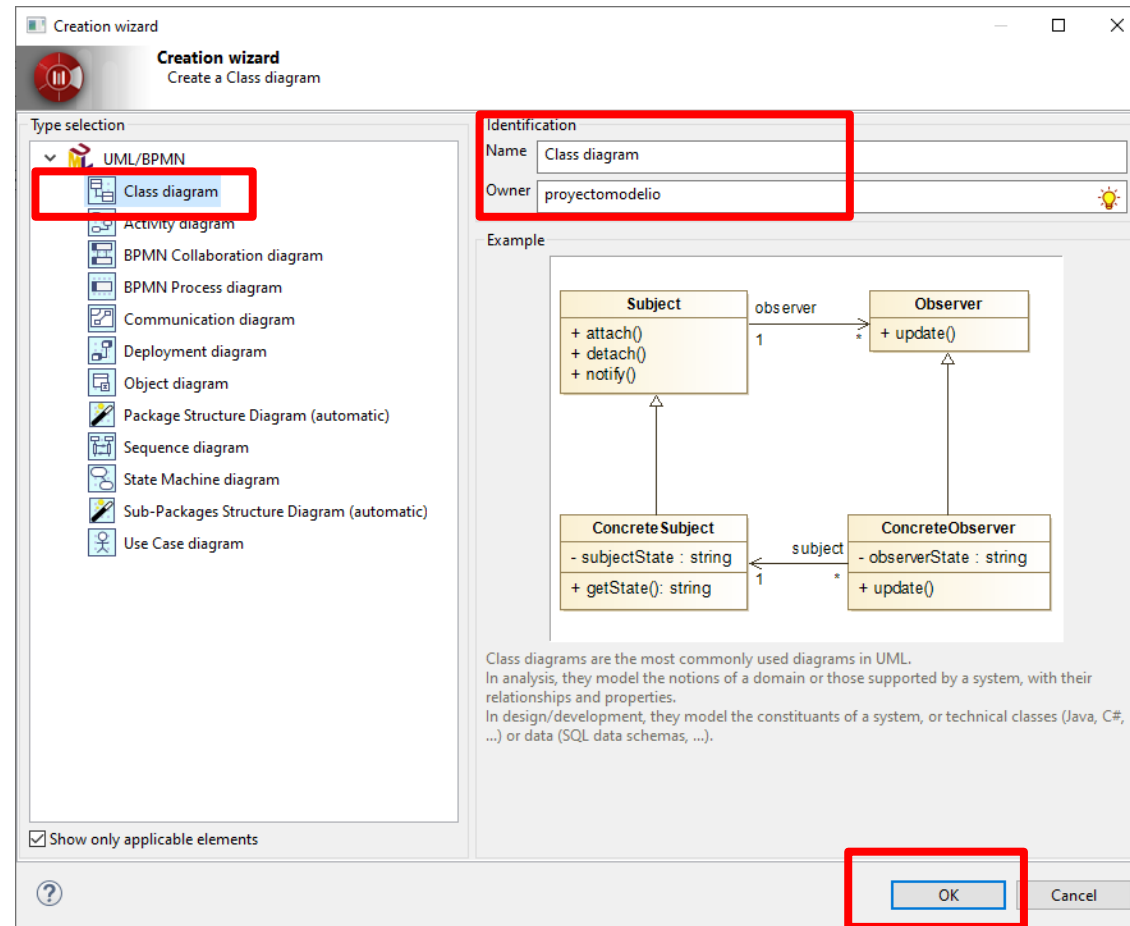
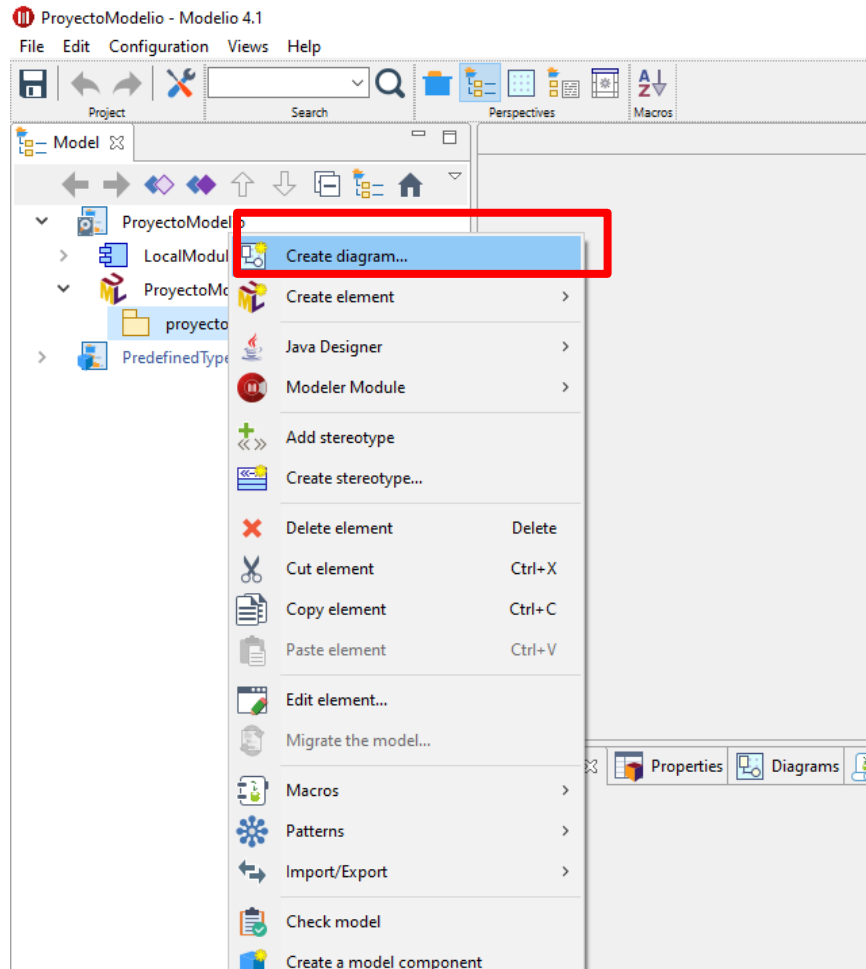


Creación de un diagrama de clases en Modelio

- Descargar de la página oficial de [Modelio](#)
- Es una herramienta UML de código abierto compatible con los estándares UML2 y BPMN.
- Crear proyecto
 - Nombre del proyecto
 - Activar Java Project
 - Pulsar **Create the Project**
- Para crear un diagrama en el proyecto
 - Menú contextual en la carpeta del proyecto con el mismo nombre que el proyecto

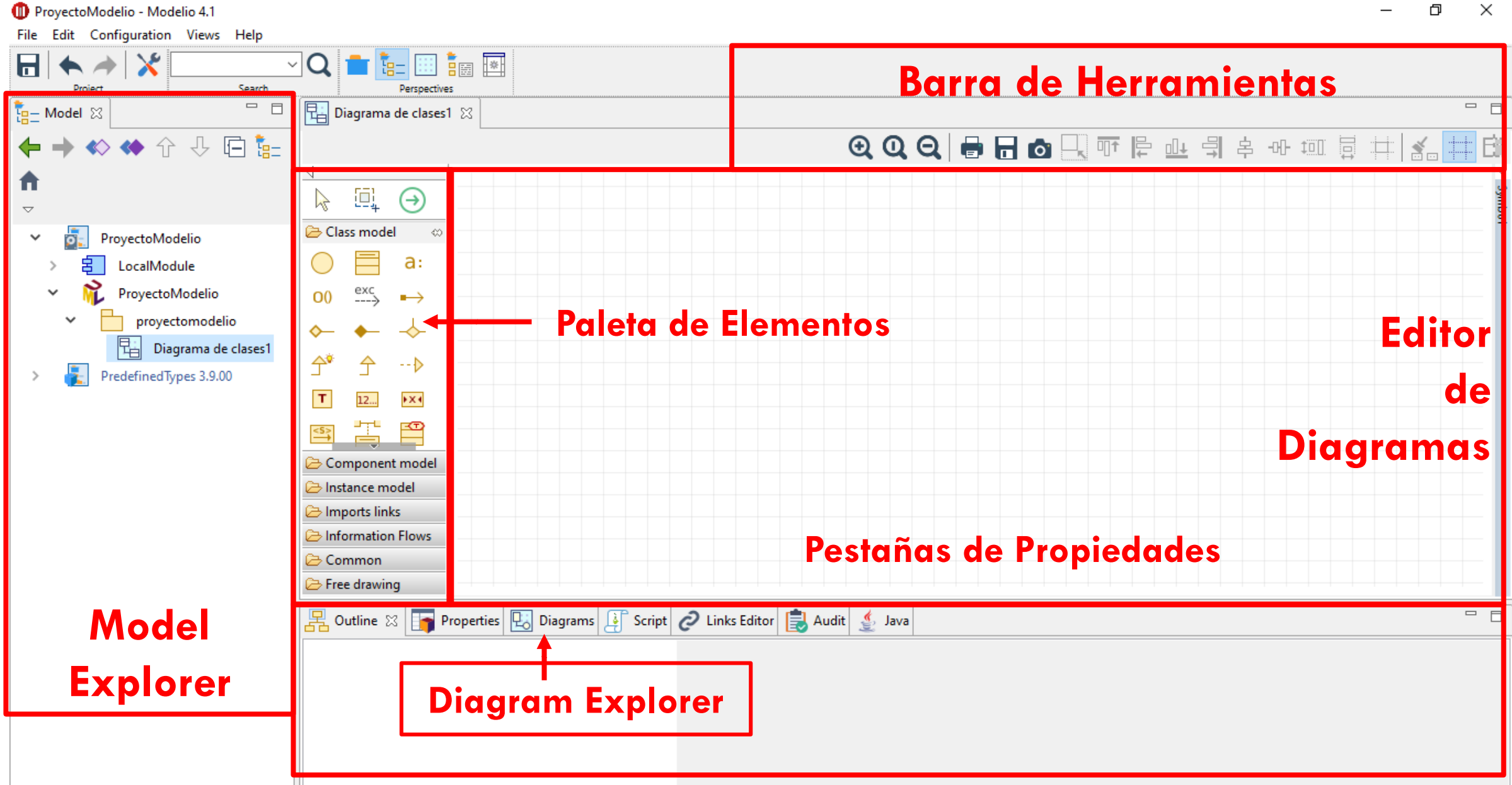


Creación de un diagrama de clases en Modelio



Selecciona **Crear diagram...** >> **Class diagram** >> escribir el nombre del diagrama y **OK**

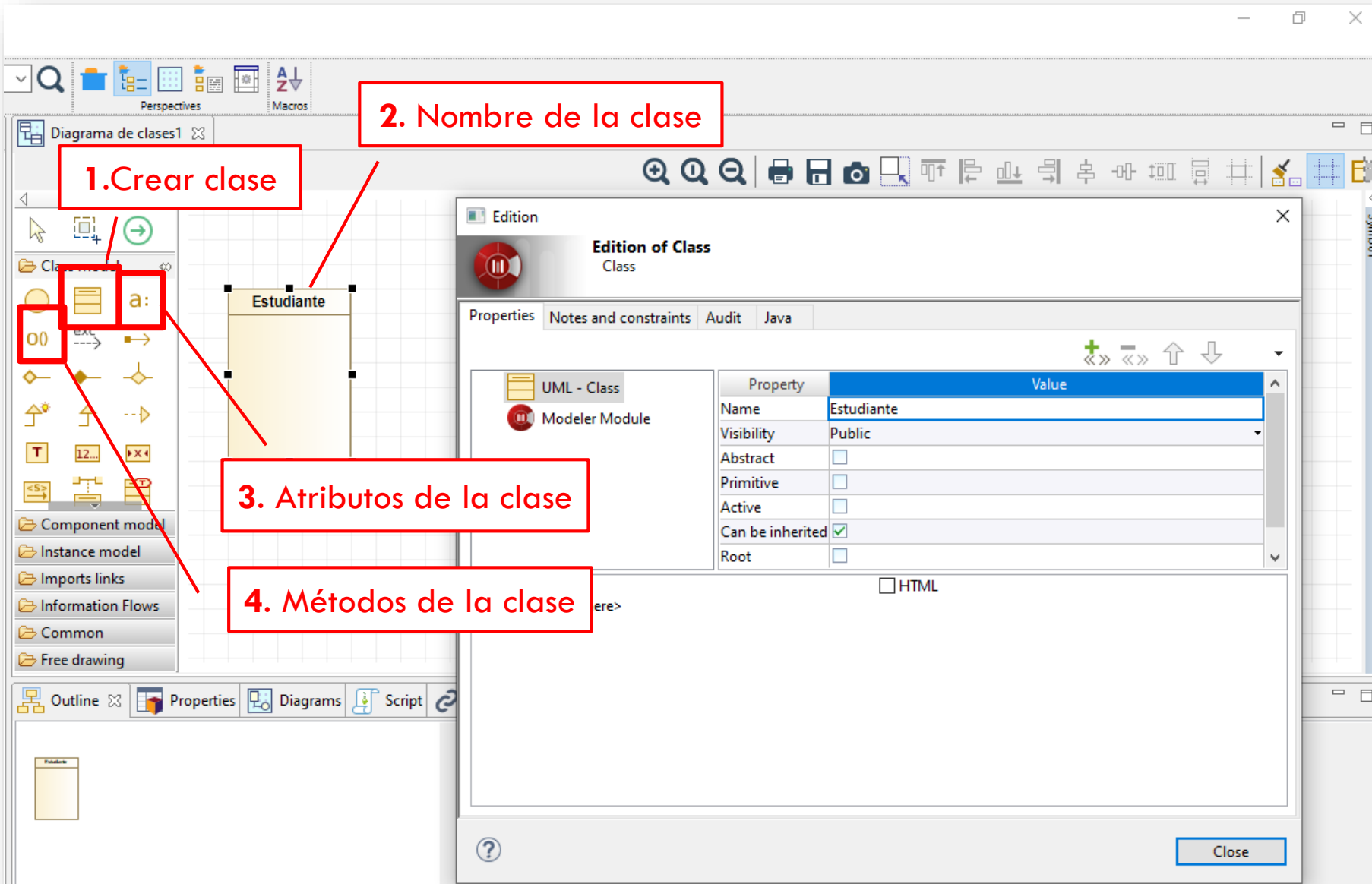
Creación de un diagrama de clases en Modelio



Creación de un diagrama de clases en Modelio

- **Área de edición** o editor de diagramas
- **Model explorer:** dispone de todos los elementos del diagrama. Haciendo doble clic sobre el elemento, se accede a sus propiedades
- **Diagram explorer:** lista de diagramas creados (ver, renombrar o eliminar)
- **Paleta de elementos:** que se pueden añadir a los diagramas
- Pestaña **properties:** propiedades del elemento seleccionado
- Pestaña **outline:** vista previa de los diagramas
- Pestaña **Audit:** vista de la auditoría (errores en los diagramas)
- Pestaña **Java:** vista de propiedades de Java, para generar código a partir del diagrama de clases

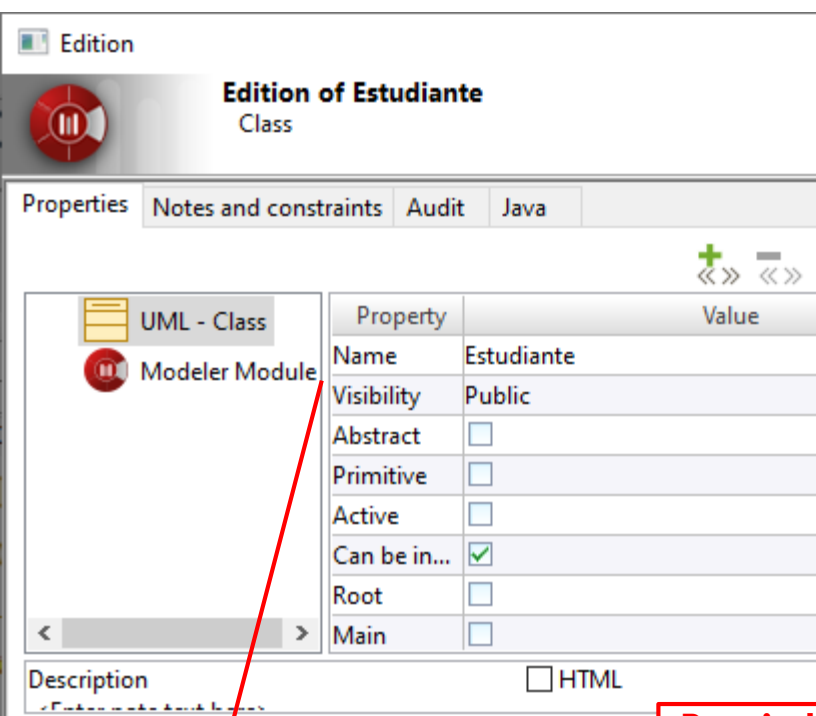
Creación de clase, atributos y métodos. Modelo



- (1) Crear la clase y pinchar en el área de edición. Para poner el nombre (2), clic en el nombre y escribir o doble clic y accedo a las propiedades de la clase
- (3) Atributos de la clase, clic en **a:** y pinchar en la clase.
- (4) Métodos de la clase, clic en **o()** y pinchar en la clase

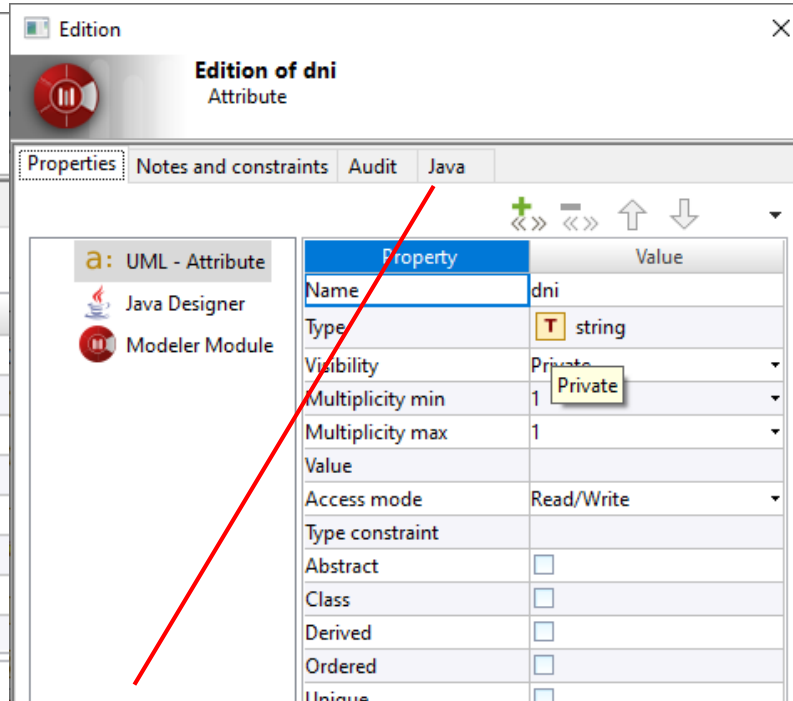
Creación de clase, atributos y métodos. Modelo

- Para acceder a las propiedades de la clase, atributo o método → doble clic en el elemento

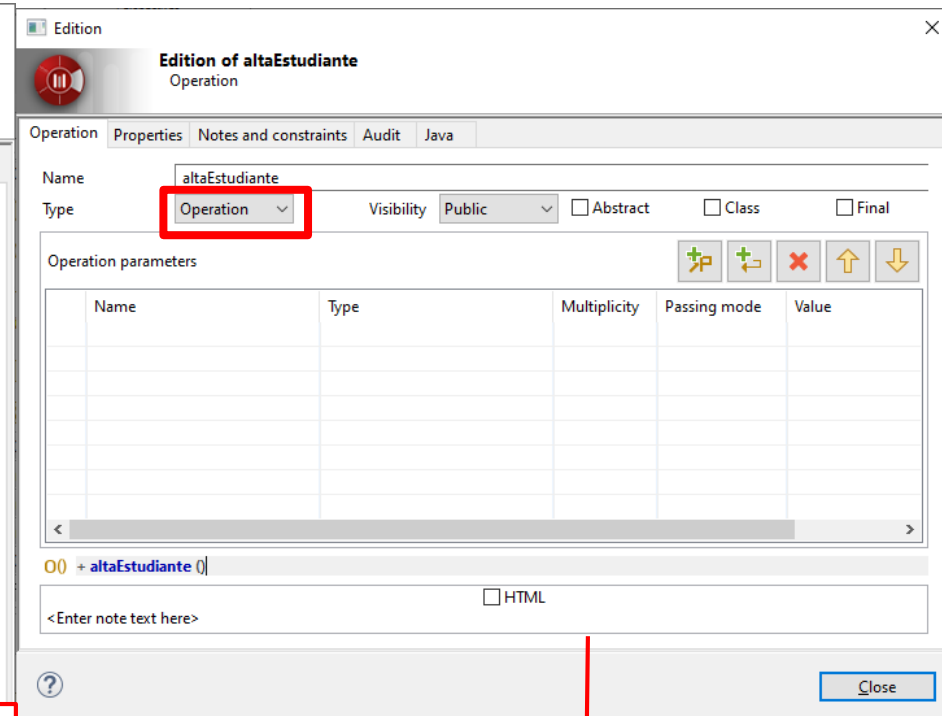


Propiedades de la clase:

Nombre: Estudiante
Visibilidad: Public



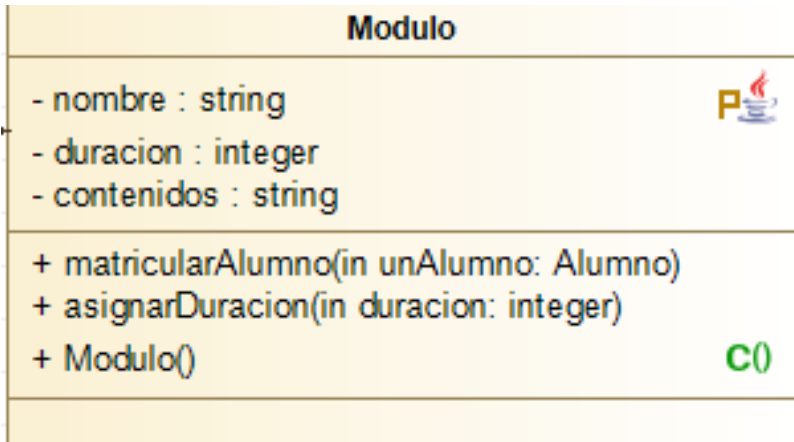
Propiedades de atributos. Los métodos setter y getter están en la pestaña Java para cada atributo (primero Java Property>>Getter visibility:Public>>Setter visibility: Public → aparece una P en los atributos):
Nombre: dni
Tipo: String
Visibilidad: private



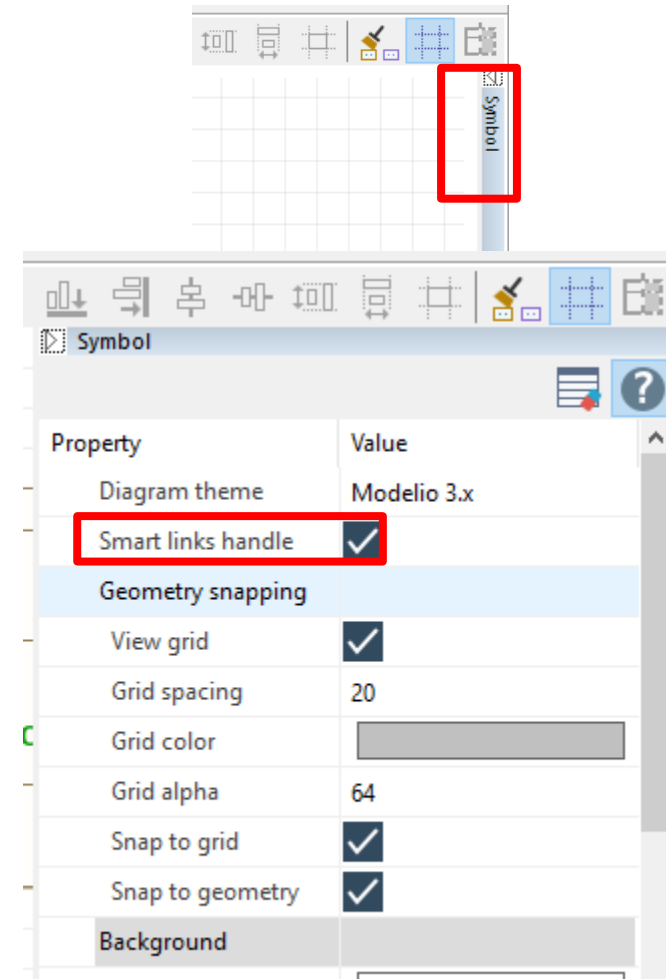
Propiedades de métodos

Creación de clase, atributos y métodos. Modelio

- Crea un diagrama con una clase, **Módulo**, con los siguientes **atributos** (nombre-string, duración-int, contenidos-string), y los siguientes **métodos** (matricularAlumno, asignarDuracion, y el constructor)



Al crear las clases y pasar el ratón por el diagrama aparece un círculo verde con una flecha en su interior, que dificulta realizar determinadas operaciones. Se trata de la herramienta **Smart links handle** y puede resultar más cómodo deshabilitarla para trabajar. Para ello, hacer clic en una zona vacía del editor y acceder a la opción **Symbol** que aparece en la parte superior derecha de la ventana en vertical.



Métodos en Modelio

Nombre del método y tipo:

En el tipo es donde se especifica si se trata de un constructor

Visibilidad: del método (por defecto public)

Parámetro ejemplo: unAlumno de tipo Alumno, de entrada (in)

Parámetros: para añadir parámetros (primer botón), y valor devuelto (return)

Operation Properties Notes and constraints Audit Java

Name: matricularAlumno

Type: Operation

Visibility: Public

Abstract: ☐ Class: ☐ Final: ☐

Operation parameters

Name	Type	Multiplicity	Passing mode	Value
>P unAlumno	Alumno	1	In	

00 + matricularAlumno (unAlumno in : Alumno)

<Enter note text here>

Close

Para crear los getter y los setter de los atributos

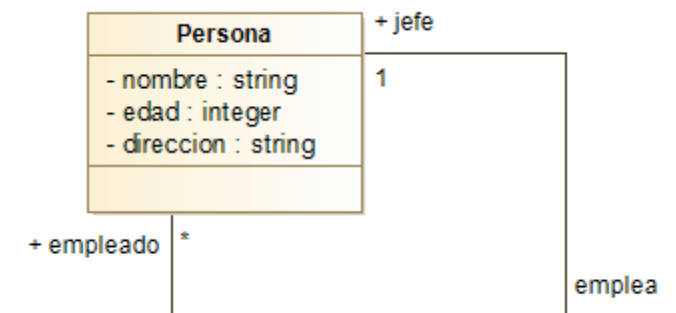
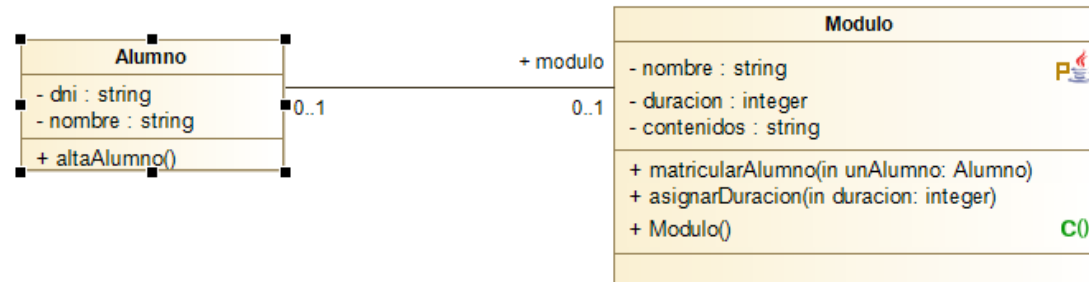
The screenshot shows the UML class editor for a class named 'Alumno'. The class has four attributes: `- nombre : string`, `- dni : string`, `- direccion : string`, and `- tlf : string`. It also has a constructor: `+ Alumno(in nombre: string, in dni: string, in dir: string, in telef: string)`. The 'Java' tab is selected in the bottom toolbar, and the 'Properties' panel is open, showing configuration for getters and setters.

Property	Value
No code	<input type="checkbox"/>
Java Property	<input checked="" type="checkbox"/>
Wrapper	<input type="checkbox"/>
Getter	<input checked="" type="checkbox"/>
Getter Visibility	Public
Setter	<input checked="" type="checkbox"/>
Setter Visibility	Public
Static	<input type="checkbox"/>
Final	<input type="checkbox"/>

- Los métodos get y set de cada atributo se crea a través de la pestaña **Java** del atributo
 - Marcar las casillas **Java Property**, **Getter** y **Setter**
 - En **Getter visibility** y **Setter visibility**, seleccionar **Public**
 - Aparecerá un icono en los atributos

Relaciones entre clases

- Conexión entre dos clases que incluimos en el diagrama cuando aparece algún tipo de relación entre ellas en el dominio del problema.
- Representación mediante una línea continua
- Se caracterizan por su **cardinalidad** llamada **multiplicidad**
- Pueden ser:
 - De herencia
 - De composición
 - De agregación
 - De asociación

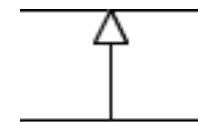
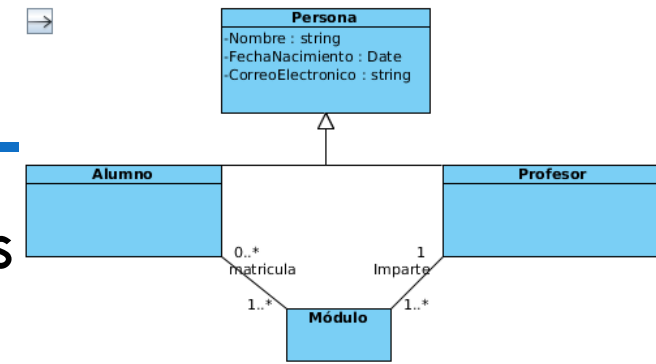


Tipos de relación

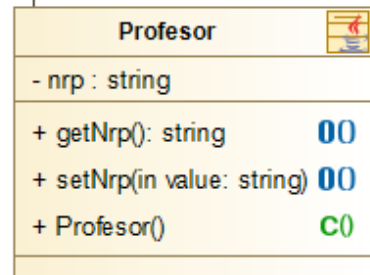
- Herencia
- Composición
- Agregación
- Asociación

Tipos de Relación. Herencia

- Permite a los objetos ser contruidos a partir de otros objetos
- Objetivo: **reutilización**
- Una **clase base** y una jerarquía de clases que contiene las **clases derivadas**.
- Tipos:
 - **Herencia simple:** Una clase puede tener sólo un ascendente.
 - **Herencia múltiple:** Una clase puede tener más de un ascendente inmediato.
- Representación:
 - En el diagrama de clases se representa como una asociación en la que el extremo de la clase base tiene un triángulo.



Ejemplo de relación de herencia

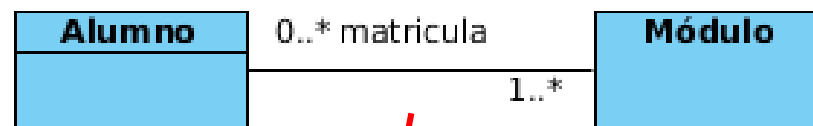


```
public class Persona {  
    private String nombre;  
    private String direccion;  
    private String telefono;  
    private String alias;  
    private String mail;  
    public String getNombre() {return this.nombre;}  
    public synchronized void setNombre(String value) {this.nombre = value;}  
    public String getDireccion() {return this.direccion;}  
    public void setDireccion(String value) {this.direccion = value;}  
    public String getTelefono() {return this.telefono;}  
    public void setTelefono(String value) {this.telefono = value;}  
    public String getAlias() {return this.alias;}  
    public void setAlias(String value) {this.alias = value;}  
    public String getMail() {return this.mail;}  
    public void setMail(String value) {this.mail = value;}  
    public Persona() {}  
}  
  
public class Alumno extends Persona {  
    private float notaMedia;  
    public float getNotaMedia() {return this.notaMedia;}  
    public void setNotaMedia(float value) {this.notaMedia = value;}  
    public Alumno() {}  
}  
  
public class Profesor extends Persona {  
    private String nrp;  
    public String getNrp() {return this.nrp;}  
    public void setNrp(String value) {this.nrp = value;}  
    public Profesor() {}  
}
```

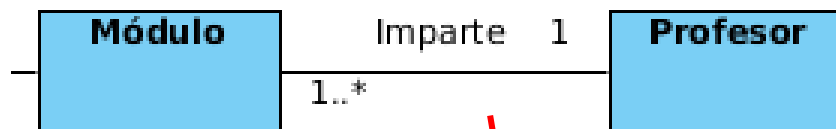
Cardinalidad o multiplicidad de la relación

- Cuántos objetos de una clase se van a relacionar con objetos de otra clase

Significado de las cardinalidades	
Cardinalidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)



Un alumno se matricula en uno o varios módulos y en un módulo están matriculados 0 o n alumnos



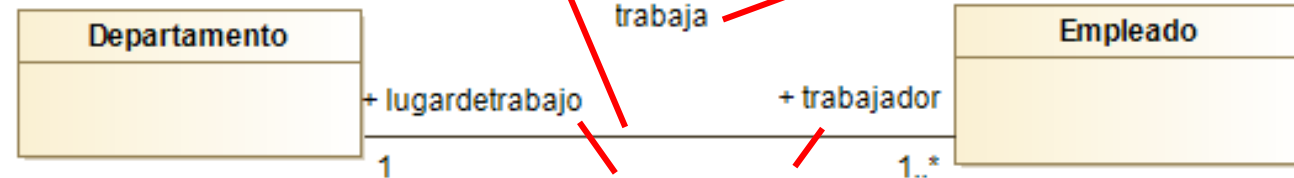
Un profesor imparte uno o varios módulos y un módulo es impartido por un único profesor

Cardinalidad o multiplicidad de la relación en Modelo

- Abrir la especificación de la relación >> doble-clic en la conexión >> establecer los apartados
 - **Multiplicidad mínima y máxima** (un empleado trabaja en 1 **Departamento** y un **Departamento** tiene a 1..* **Empleados**)
 - Nombre de la asociación
 - Si es o no navegable
 - Roles
 - Tipo de asociación: se puede modificar a agregación o composición (ver + adelante)

Doble-click en la asociación

Nombre de asociación



Roles

Edition

Edition of trabajador
Association End

Properties Notes and constraints Audit Java

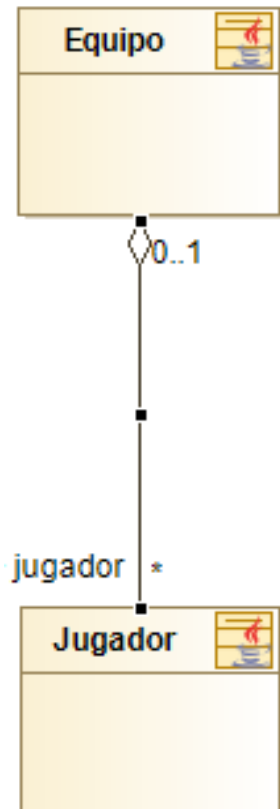
UML Java Model

Property	From: Departamento	To: Empleado
Association name	trabaja	
Navigable	<input type="checkbox"/>	<input type="checkbox"/>
Role	lugarde trabajo	trabajador
Target	<null>	<null>
Association type	Association	Association
Multiplicity min	1	1
Multiplicity max	1	*
Visibility	N/A	N/A
Is changeable	N/A	N/A
Accessors	N/A	N/A
Abstract	N/A	N/A
Class	N/A	N/A
Ordered	N/A	N/A
Unique	N/A	N/A

Description ☐ HTML

Tipos de Relación. Agregación y Composición

- La **agregación** es una asociación binaria que representa una relación **todo-parte** (pertenece a, tiene un, es parte de).
- Es una **relación débil**, y los componentes pueden ser compartidos por varios compuestos y la destrucción del compuesto **no implica la destrucción** de los componentes.



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

Edition

Edition of jugador
Association End

Multiplicidad de Equipo a Jugador

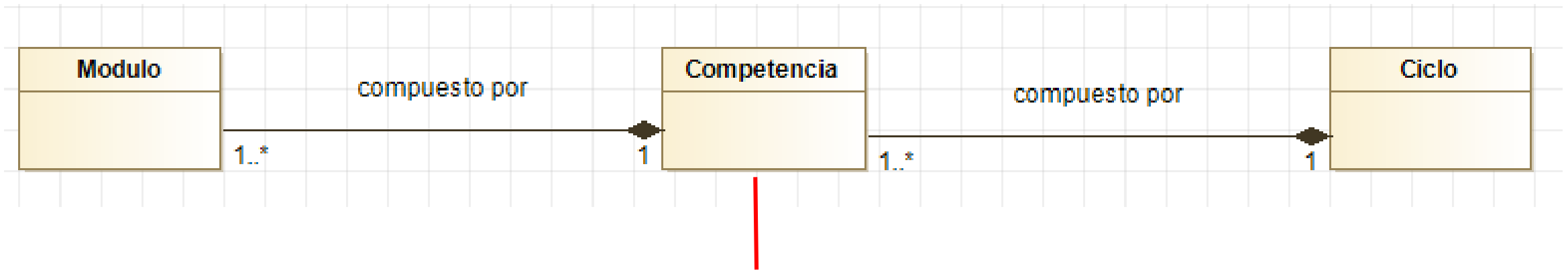
Properties Notes and constraints Audit Java

UML - AssociationEnd
Java Designer
Modeler Module

Property	From: Equipo	To: Jugador
Association name		
Navigable	<input type="checkbox"/>	<input type="checkbox"/>
Role		jugador
Target	<null>	<null>
Association type	Association	Aggregation
Multiplicity min	0	0
Multiplicity max	1	*
Visibility	N/A	N/A
Is changeable	N/A	N/A
Accessors	N/A	N/A
Abstract	N/A	N/A
Class	N/A	N/A
Ordered	N/A	N/A
Unique	N/A	N/A


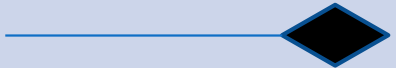
Tipos de relación. Agregación y composición

- La **composición** es una *agregación fuerte* en la que una instancia 'parte' está relacionada, como máximo, con una instancia 'todo' en un momento dado, de forma que cuando un objeto 'todo' es eliminado, **también son eliminados** sus objetos 'parte'.
- La **cardinalidad máxima** del objeto compuesto es uno obligatoriamente.



Un **Ciclo** está compuesto por varias **Competencias**, y una competencia está formada por varios **Modulos**

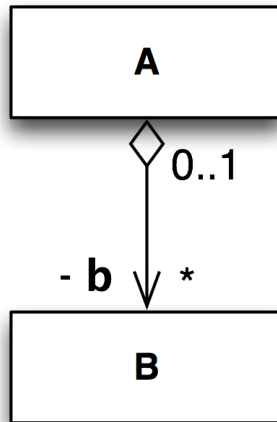
Diferencias entre agregación y composición

Agregación vs. 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

¿Agregación o composición?

- El objeto parte (Jugador) es compartido por más de un objeto agregado (Equipo) → **agregación**
- El objeto parte (Jugador) **no** puede ser compartido por más de un objeto agregado (Equipo) → **composición**
- Un objeto parte puede existir sin ser componente de un objeto agregado → **agregación**
- Un objeto parte **no** puede existir sin ser componente de un objeto agregado → **composición**

Implementación de la agregación



```
class A {
    private Vector<B> b = new Vector<B>();

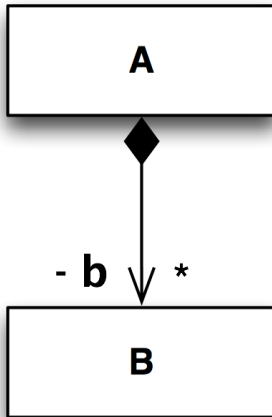
    public A() {}

    public addB(B unB) {
        b.add(unB);
    }
}
```

- El **objeto B** puede ser creado fuera de A, de forma que pueden existir referencias externas ('objB') al objeto agregado.
- Cuando 'objA' desaparece, 'objB' sigue existiendo

```
// En otro lugar (código cliente),
// quizás fuera de A...
B objB = new B();
if (...) {
    A objA = new A();
    objA.addB(objB);
}
```

Implementación de la composición



```
class A {
    private Vector<B> b = new Vector<B>();

    public A() {}
    public addB(...) {
        b.add(new B(...));
    } '...' es la información necesaria para
    crear B
}
```

- El **objeto B** es creado **dentro de A**, de forma que A es el único que mantiene referencias a su componente B.
- Cuando 'objA' desaparece, también desaparecen los objetos B que forman parte de él

```
// En otro lugar (código cliente),
// quizás fuera de A... B
if (...) {
    A objA = new A();
    objA.addB(...);
}
```

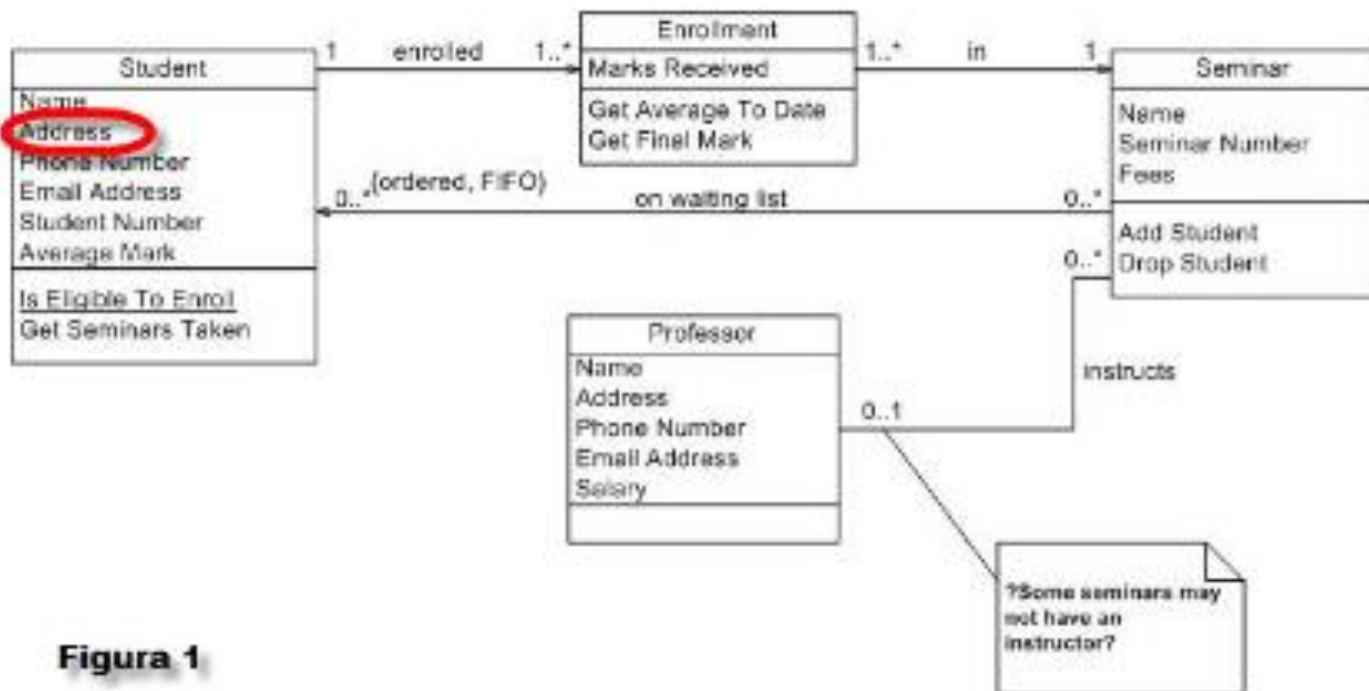


Figura 1

La dirección de un estudiante puede ser un atributo de la clase estudiante o puede ser una clase, teniendo en cuenta la naturaleza de los datos de una dirección (está formada por calle, número, piso, puerta...)

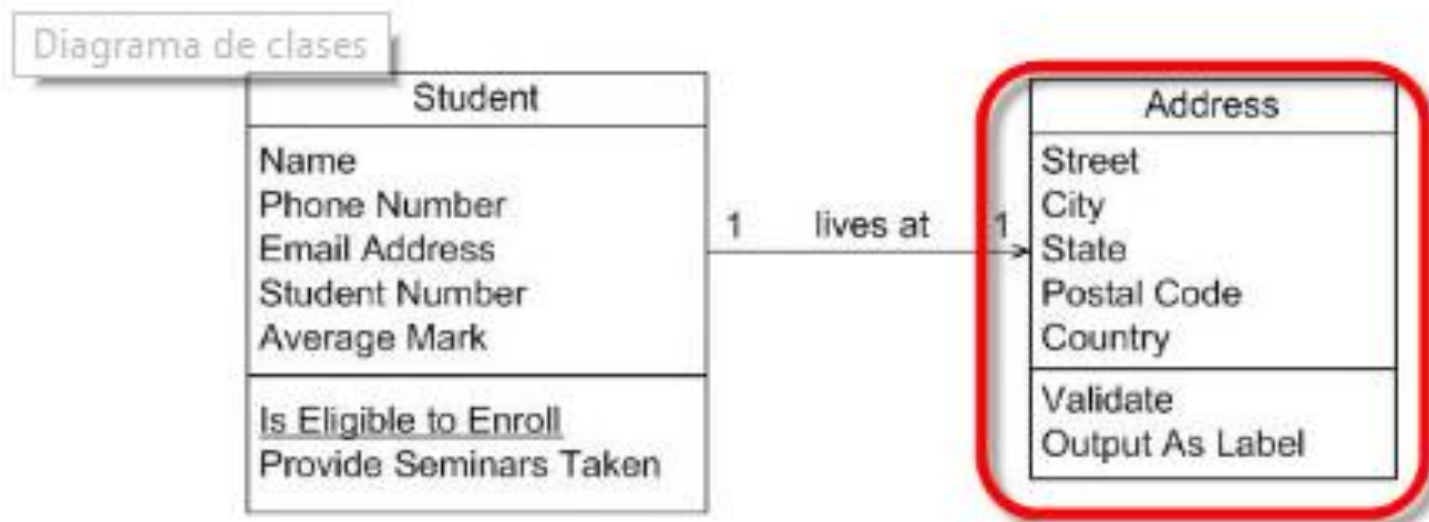
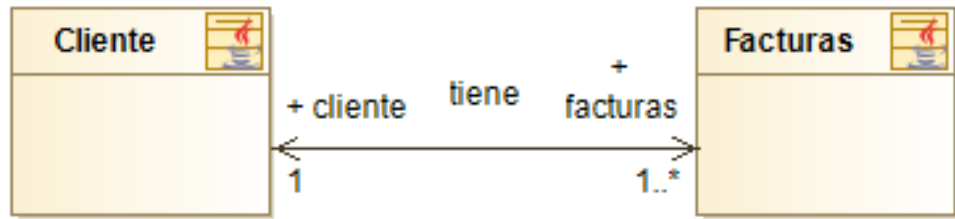


Figura 2

Tipos de Relación. Asociación

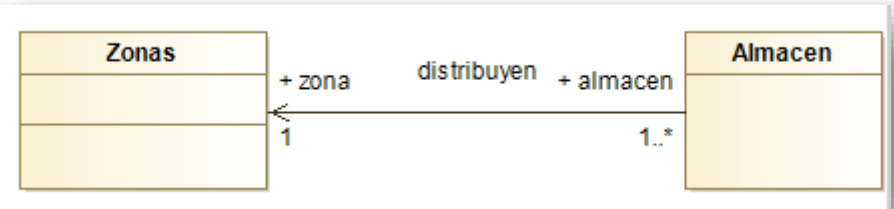
- Permite asociar objetos que colaboran entre sí.
- Si se convierten a Java dos clases unidas por una **asociación bidireccional**, cada una de las clases tendrá un objeto o un set de objetos dependiendo de la multiplicidad entre ellas.
- En la **asociación unidireccional** la clase destino no sabrá de la existencia de la clase origen, y la clase origen contendrá un objeto o set de objetos de la clase destino.
- **Navegabilidad.** Establece las asociaciones unidireccionales o bidireccionales.
 - En la bidireccional, ambas clases conocen la existencia de la otra, son navegables
 - En la unidireccional, la clase origen conoce la existencia de la clase destino (es navegable)

Ejemplo Asociación



La asociación tiene: un cliente tiene facturas y una factura es de un cliente. Es **bidireccional** → son navegables. Ambas clases tienen conocimiento de la otra

Se desactiva el checkbox **Navegable** en rol almacen (no navegable)



La asociación distribuyen: un almacén distribuye en varias zonas y una zona es distribuida por un almacén. Es **unidireccional** → es navegable de Almacén a Zona, es decir, Almacen conoce la existencia de Zona

Property	From: Cliente	To: Facturas
Association name	tiene	
Navigable	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Role	cliente	facturas
Target	Cliente	Facturas
Association type	Association	Association
Multiplicity min	1	1
Multiplicity max	1	*
Visibility	Public	Public
Is changeable	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Accessors	Read/Write	Read/Write
Abstract	<input type="checkbox"/>	<input type="checkbox"/>

Property	From: Zonas	To: Almacen
Association name	distribuyen	
Navigable	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Role	zona	almacen
Target	Zonas	<null>
Association type	Association	Association
Multiplicity min	1	1
Multiplicity max	1	*
Visibility	Public	N/A
Is changeable	<input checked="" type="checkbox"/>	N/A
Accessors	Read/Write	N/A
Abstract	<input type="checkbox"/>	N/A

Tipos de Relación. Asociación. Código generado

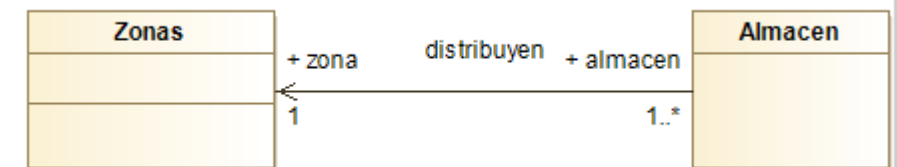
```
public class Cliente {  
    public List<Facturas> facturas =  
    new ArrayList<Facturas> ();  
}  
  
public class Facturas {  
    public Cliente cliente;  
}
```



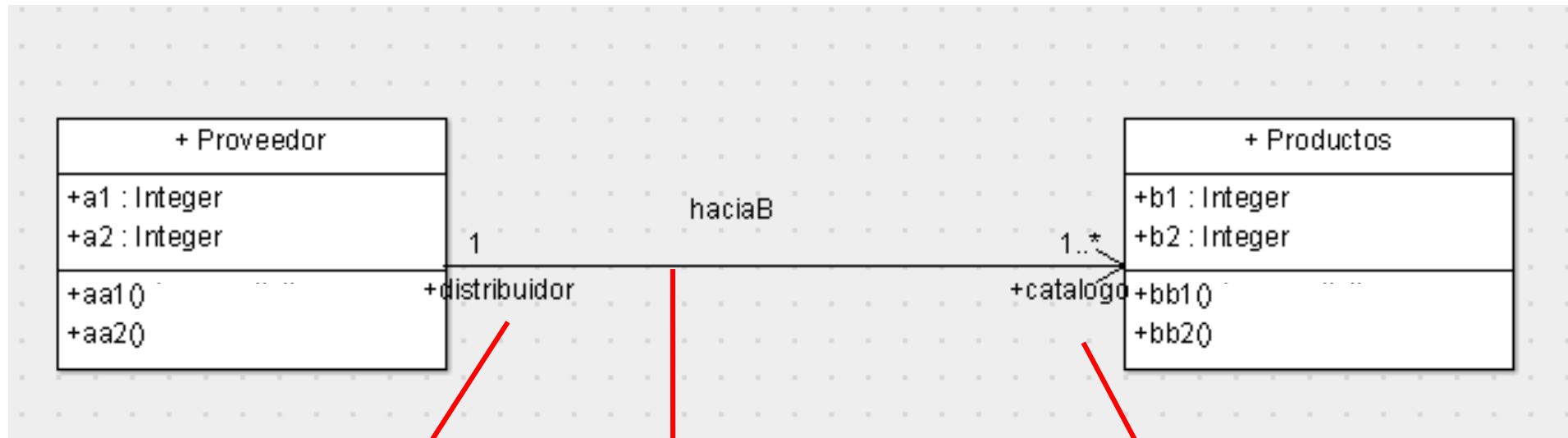
-En el caso de **Cliente-Factura**, son bidireccionales, Cliente contiene una **Lista de facturas** y Facturas un objeto **cliente**.

-En el caso de **Zonas-Almacen**, es unidireccional (solo es navegable zona). Por esa razón **Almacen** contiene la **zona** en la que se encuentra el Almacen

```
public class Zonas {  
}  
  
public class Almacen {  
    public Zonas zona;  
}
```



Tipos de Relación. Asociación. Navegable



Proveedor es navegable a Productos

La asociación es unidireccional,
solo la clase origen Proveedor
conoce la existencia de la clase
destino Productos

**Productos no es navegable a
Proveedor**

Tipos de Relación. Asociación. Código generado

```
import java.util.Vector;

public class Proveedor {

    public Integer a1;

    public Integer a2;

    /**
     *
     * @element-type Productos
     */
    public Vector catalogo;

    public void aa1() {
    }

    public void aa2() {
    }

}
```

Navegable

```
public class Productos {

    public Integer b1;

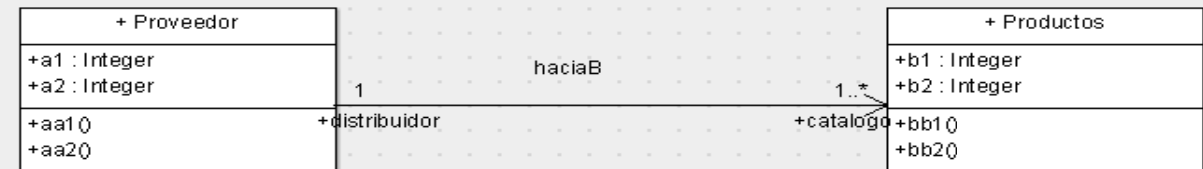
    public Integer b2;

    public void bb1() {
    }

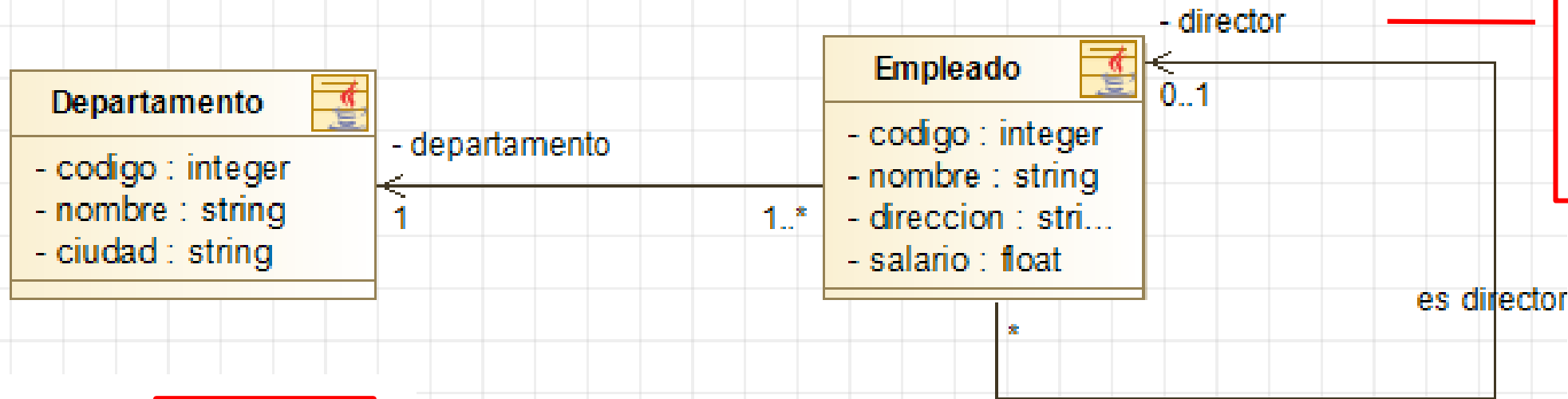
    public void bb2() {
    }

}
```

No navegable



Tipos de Relación. Asociación. Reflexiva



Empleado no es navegable, director es navegable

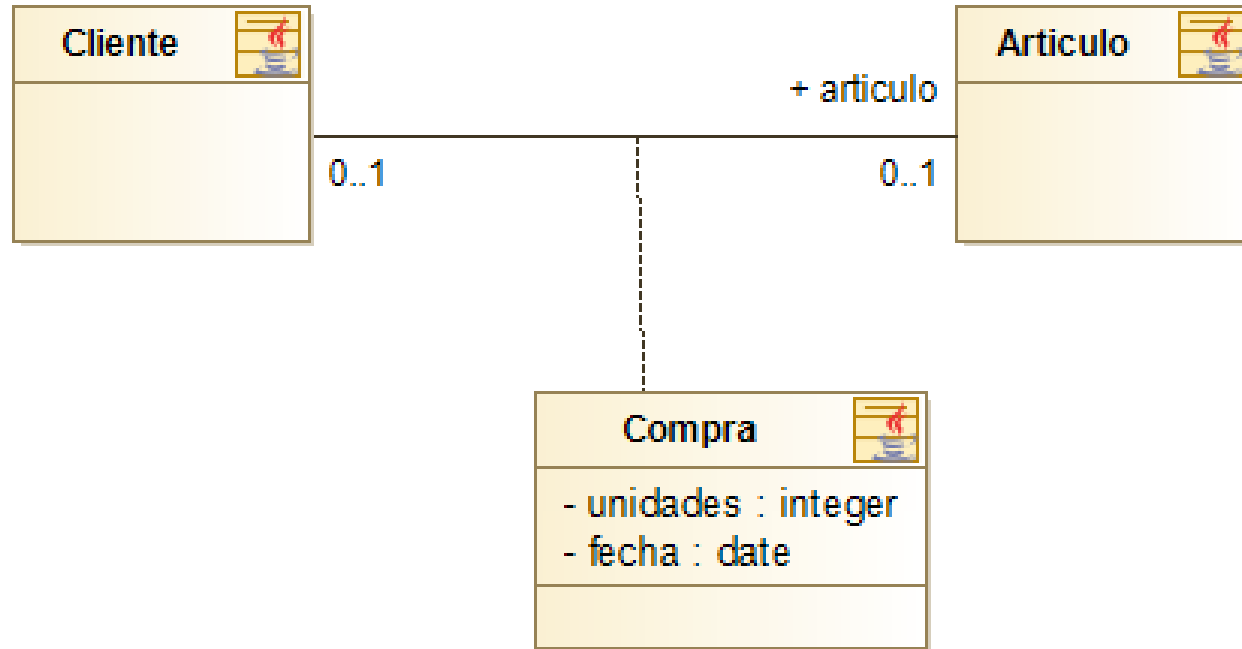
```
public class Departamento {
    private int codigo;
    private String nombre;
    private String ciudad;
}
```

Lectura: Empleado es jefe de 0 o n empleados y un empleado tiene un único jefe (excepto el jefe superior)

```
public class Empleado {
    private int codigo;
    private String nombre;
    private String direccion;
    private float salario;
    private Empleado director;
    private Departamento departamento;
}
```

Tipos de Relación. **Asociación.** Clase asociación

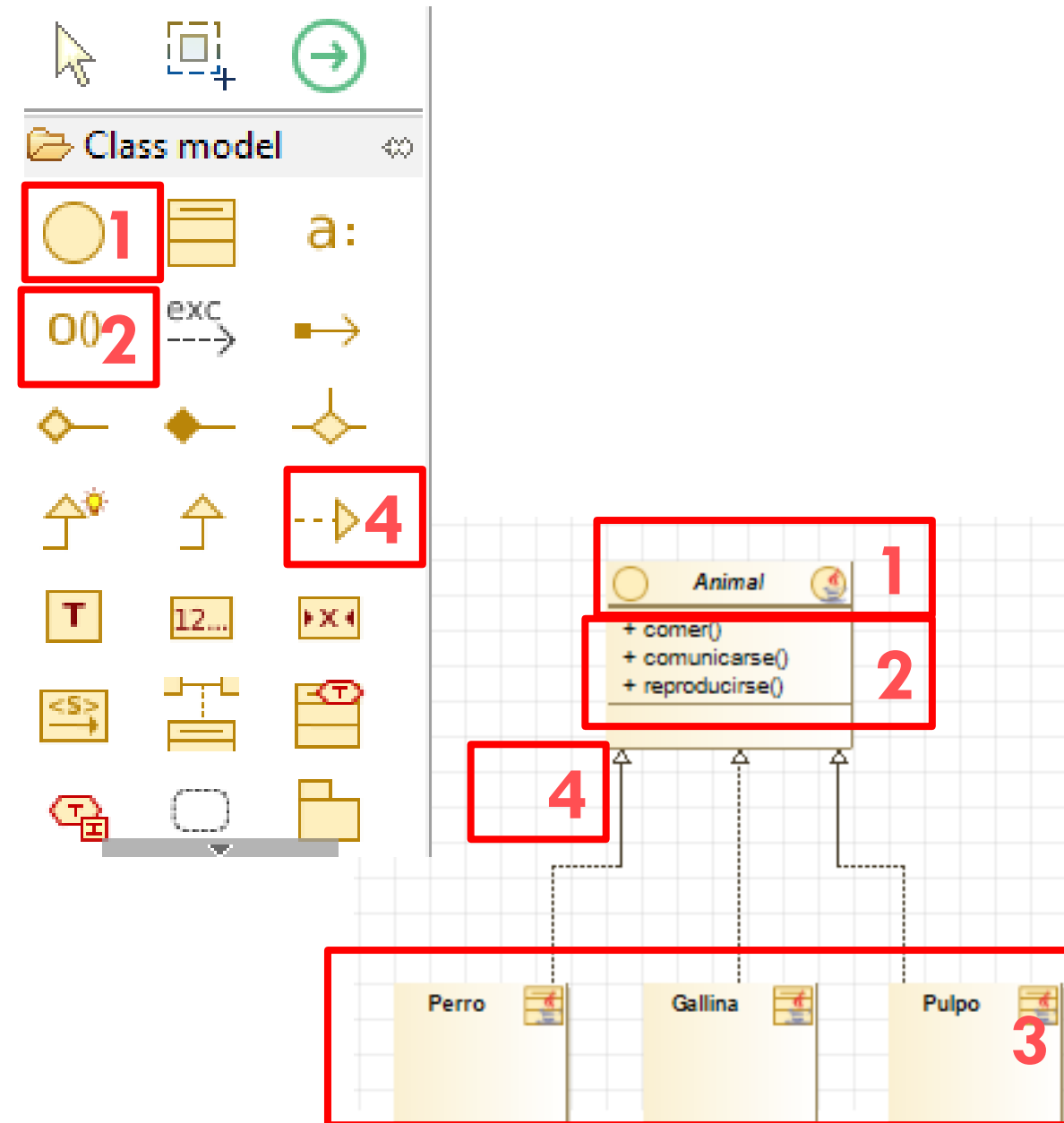
- Asociación entre dos clases que pueden llevar información
- Similar a los atributos en relaciones **N:M**



1. **Crear las tres clases**
 2. **Relacionar con una asociación Cliente-Articulo**
 3. **Botón "Class Association" y unir la asociación Cliente-Articulo con Compra**
- Tener en cuenta, que en **Modelio** indica con el color verde las asociaciones posibles y en rojo, las selecciones no admitidas

Relación de Realización

- Relación de herencia entre una clase **Interfaz** y la subclase que implementa esa interfaz.
- Interfaz:** clase totalmente abstracta. No tiene atributos y todos sus métodos son abstractos
- Para crear un interfaz con Modelio:
 - Botón Interfaz y crear el elemento en el tapiz
 - Añadir operaciones al interfaz
 - Crear las clases que implementan la interfaz
 - Relacionar las clases con la interfaz

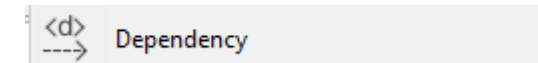


Dependencia

- Relación entre dos clases en la que una usa la otra, es decir, que la necesita para su cometido
- Ejemplo. La clase Impresora imprime Documentos (la clase *Impresora* usa la clase *Documento*). Un *Viajero* necesita un *Equipaje* para viajar

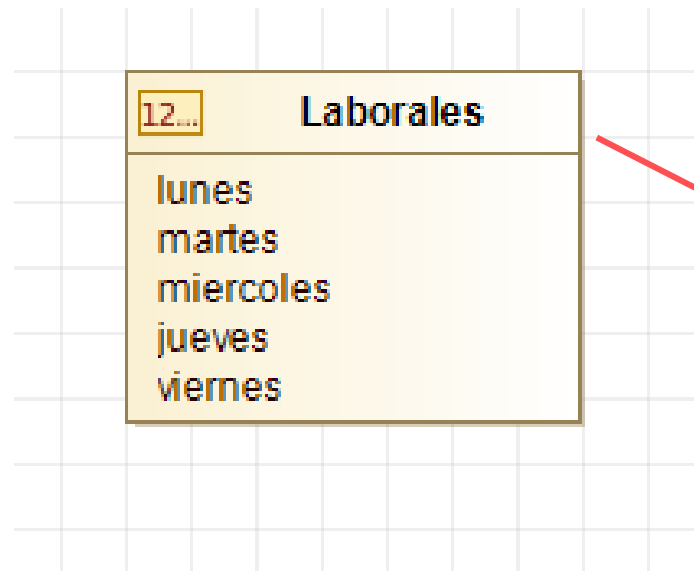
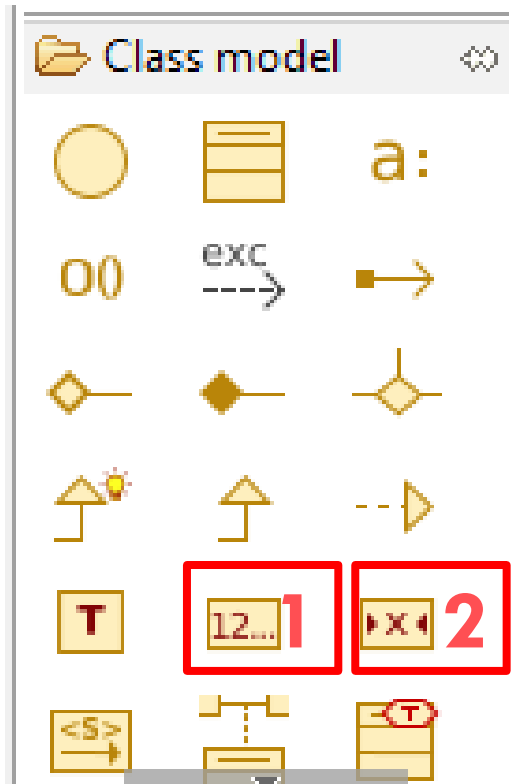


Relación de dependencia: se crean las dos clases, se activa Smart Links Handle (el círculo verde con flecha) y se enlaza con la otra clase. Se selecciona la relación de dependencia



Enumerados

- Para crear un enumerado en el diagrama de clases:
 - Botón para crear el Enumerado
 - Botón para crear cada uno de los elementos del enumerado



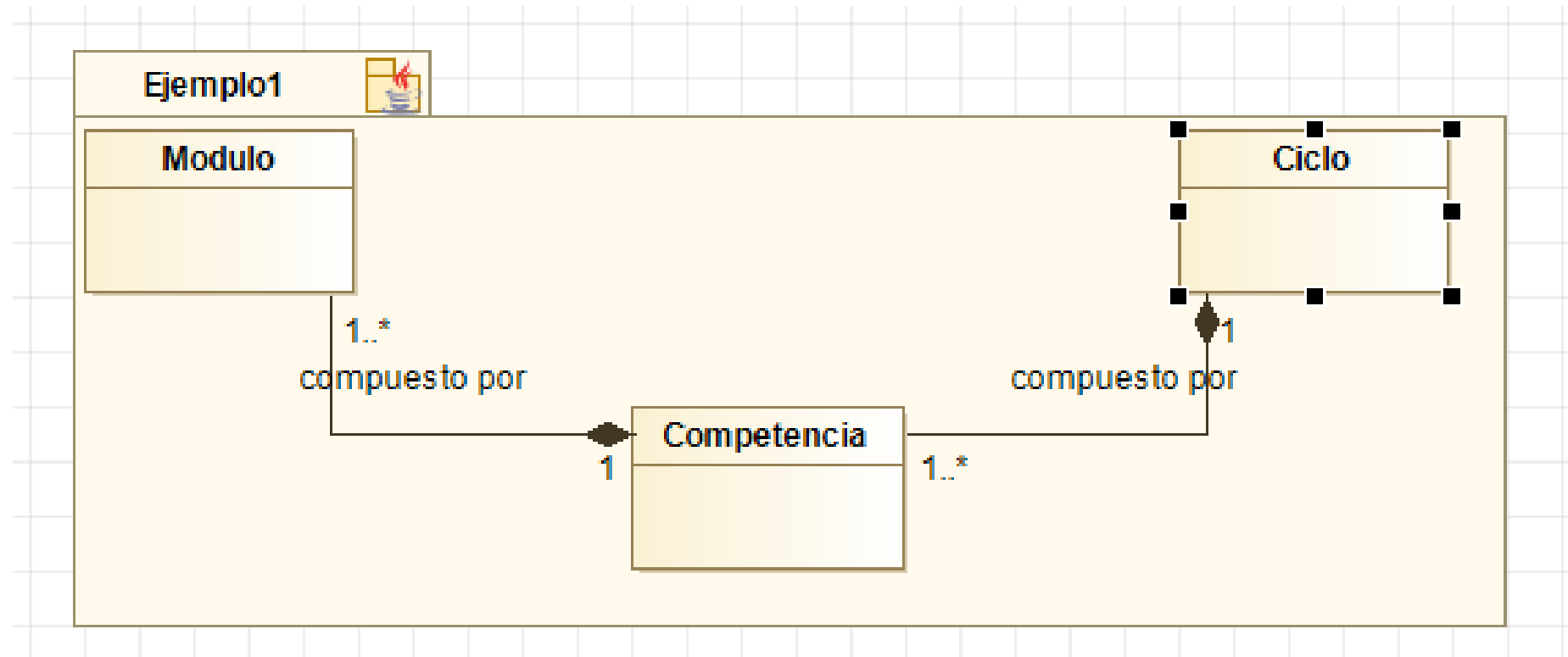
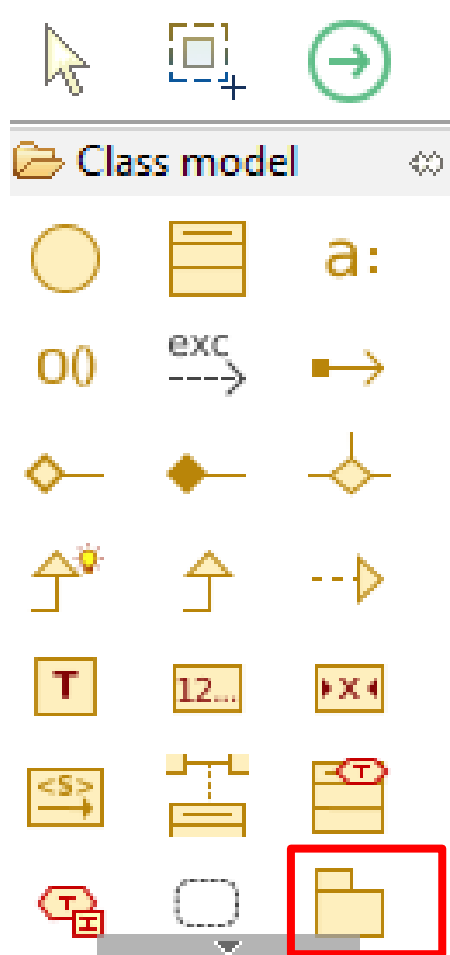
Ya se pueden definir atributos en una clase de tipo Laborales

Ejercicios

- Se trata de realizar un diagrama de clases para representar las relaciones entre empleados y departamentos.
 - Consideramos que un empleado trabaja en un departamento y en el departamento trabajan muchos empleados
 - Datos de los empleados son código, nombre, oficio y salario
 - Datos de los departamentos son código, nombre y localidad
 - Además, un empleado puede ser jefe de varios empleados
 - Se necesita crear los métodos para asignar datos a los empleados y departamentos y devolverlos (setter and getter).

Creación de paquetes

- Para crear paquetes en Modelio se utiliza el botón carpeta
- Se puede arrastrar en el editor de Modelio las clases a la carpeta creada

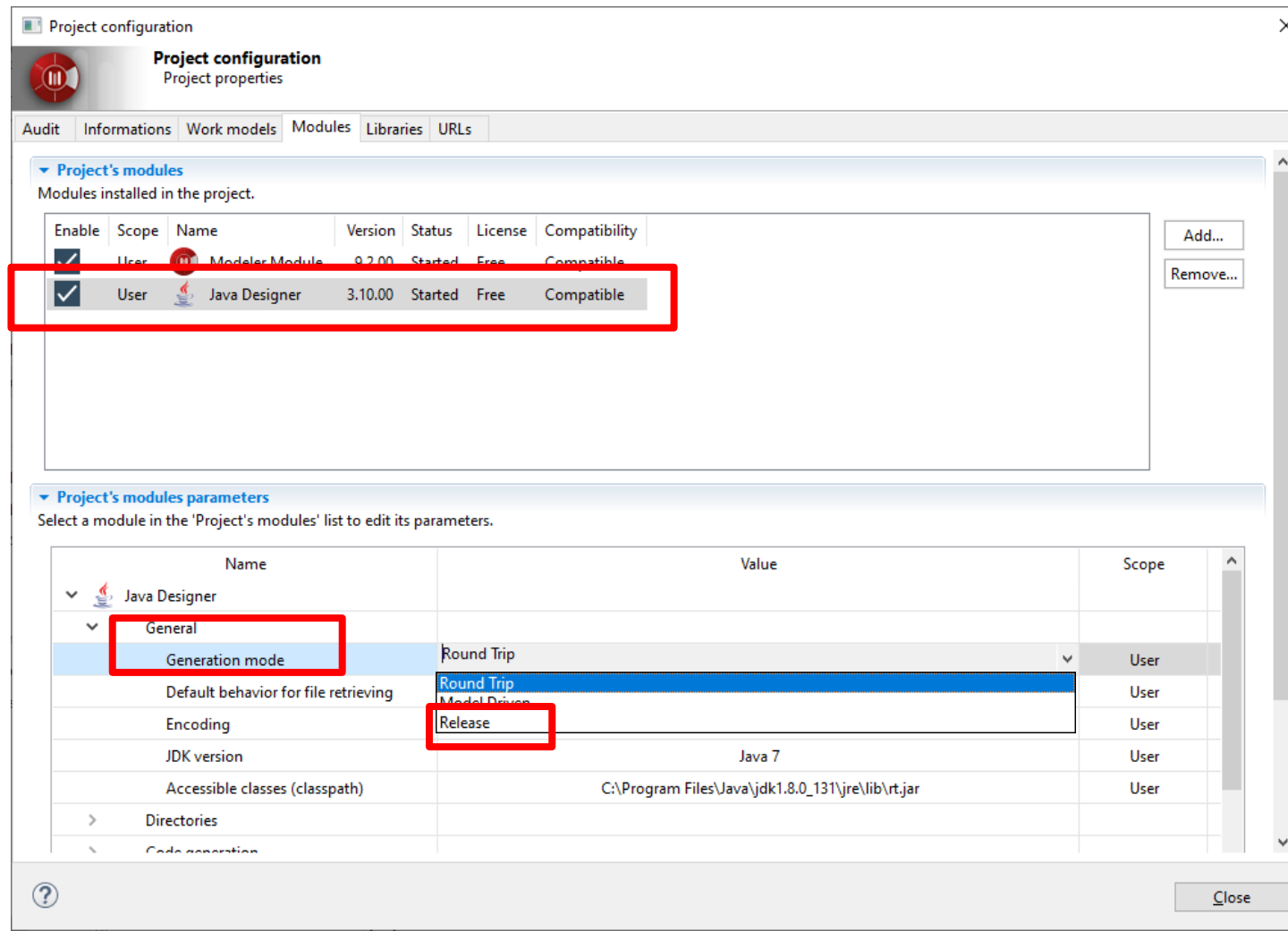


Ejercicio

- Realizar un diagrama de clases para representar las relaciones entre empresa, empleado y clientes. Utilizaremos asociaciones de composición y generalización en el diagrama.
 - La empresa se compone de clientes y de empleados. Utilizaremos para estas relaciones asociaciones de composición
 - Datos de la empresa son: CIF, razón social, dirección y teléfono
 - Datos de clientes: código, nombre, fecha de nacimiento, teléfono, empresa para la que trabaja y comisión
 - Datos de empleados: código, nombre, fecha de nacimiento, teléfono, fecha de alta en la empresa y salario
 - Como los atributos nombre, fecha de nacimiento y teléfono, son comunes para clientes y empleados se creará una clase persona para esos atributos y las clases cliente y empleado heredarán la clase persona
 - Un empleado puede ser director de varios empleados. De este director se necesita saber también la categoría y la fecha de alta como director. El director heredará de empleado y tendrá una asociación con empleado.
 - Para todas las clases se crean dos métodos, uno par asignar datos a los atributos, con tantos parámetros como atributos

Configuración para la generación de código en Modelio

- M.Configuration>>Modules...>>seleccionar Java Designer>>General>>Genaration Mode>>modificar a **Release**



Configuración para la generación de código en Modelio

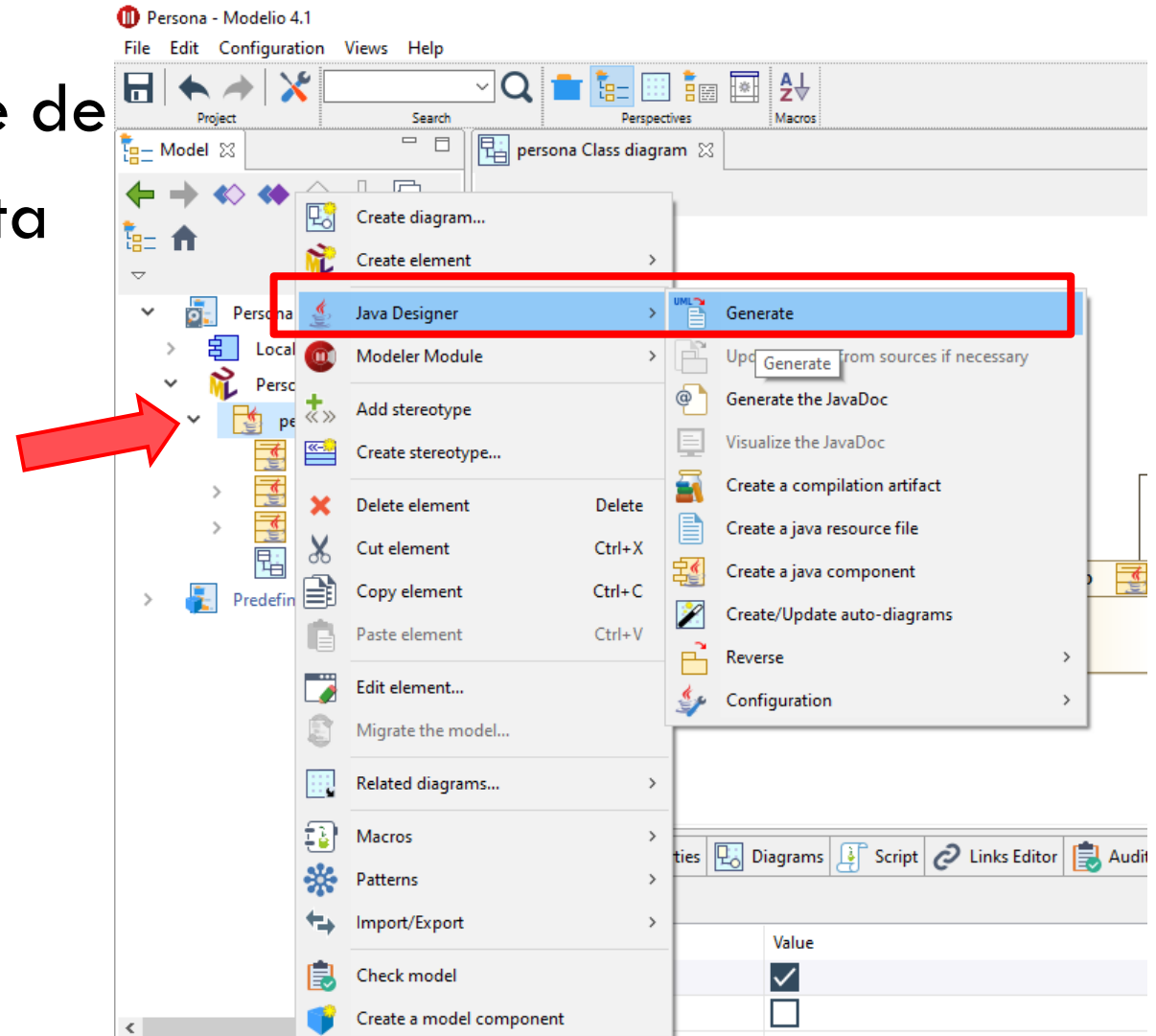
- Comprobar que **todas** las carpetas y clases tienen activadas la opción **Java element** de la pestaña Java

The screenshot shows the Modelio software interface. On the left, a tree view displays the project structure: 'Persona' (LocalModule) containing 'persona' (Persona, Alumno, Profesor) and 'persona Class diagram'. The main workspace shows a class diagram with 'Persona' at the top, and 'Alumno' and 'Profesor' below it, all connected by inheritance arrows. The bottom panel shows the 'Properties' view for the selected 'Java element' property, with a red box highlighting the 'Value' column, which contains an unchecked checkbox. A red callout box with the text 'Activar en todas las clases y carpetas' points to this checkbox.

Activar en todas las clases y carpetas

Configuración para la generación de código en Modelio

- Botón derecho en la carpeta del proyecto >> Java Designer >> Generate
- Hay que comprobar el workspace de Modelio y habrá creado una carpeta src con el código generado.





Project configuration

Project properties

Audit

Informations

Work models

Modules

Libraries

URLs

General

Name: Persona

Project images:



64 * 64



24 * 24

Contact:

Description:

Storage

Path: C:\Users\mcruz\modelio\workspace\Persona

Last modification: 2022-01-17T01:37:40.752009Z

Size: 6 MB

M.Configuration>>Modules>>Pestaña
Informations>>en Path se encuentra la
ubicación del workspace del proyecto

Models

Models composing the project:

Name	Status	Path
Persona	Operational	
PredefinedTypes 3.9.00	Operational	

Project's modules

Modules installed in the project.

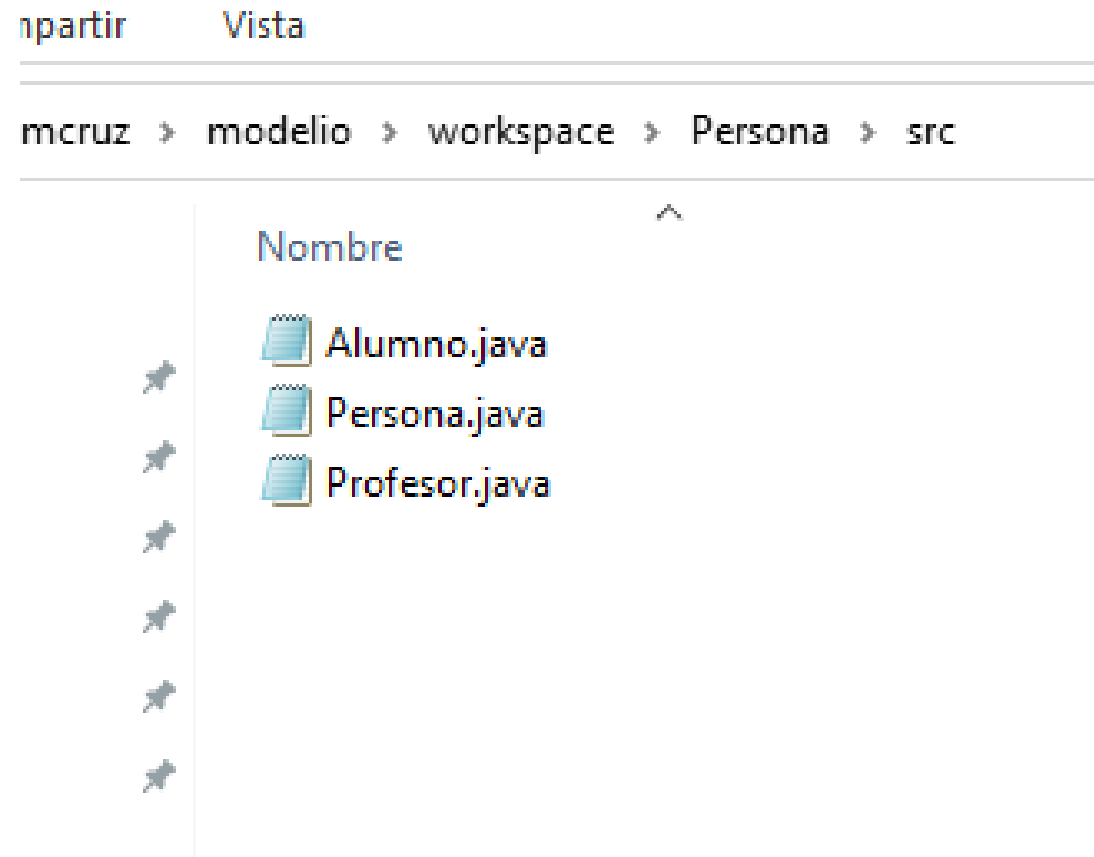
Name	Version	Status
	3.9.00	Operational



Close

Configuración para la generación de código en Modelio

- Y en la carpeta src del proyecto se encuentran los archivos .java generados



Documentación

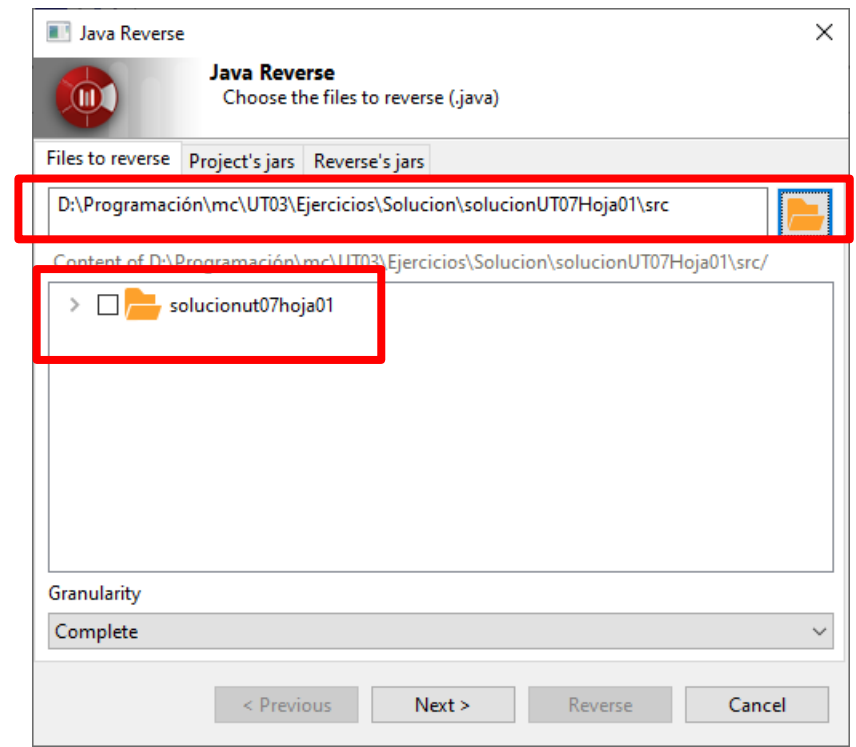
- Si se está creando el diseño, y no se cumple con las reglas establecidas, aparecerán las críticas y las sugerencias en el apartado de **críticas** (*clic en las hojas de las clases*) que nos ayudarán a corregir fallos.
- **Para borrar, no solo de la vista, hay que borrar del modelo (Botón Papelera) de la barra de herramientas.**

Ingeniería inversa

- *“Proceso de analizar un sistema para crear una representación del mismo, pero a un nivel más elevado de abstracción.”*-**E.J. Chikofsky, J.H. Cross**
- *“Proceso que recorre hacia atrás el ciclo de desarrollo de software.”*-**P. Hall**

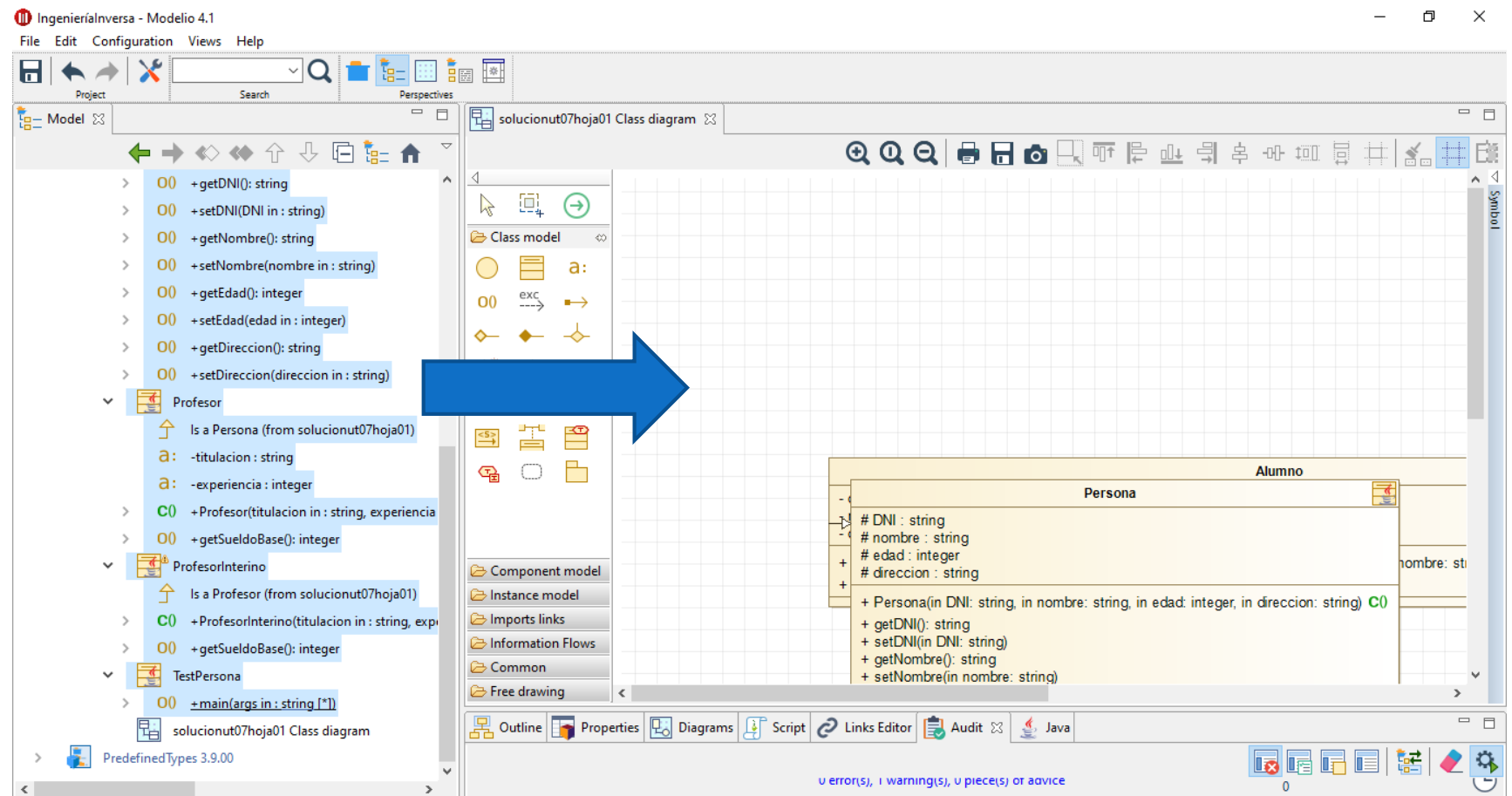
Ingeniería inversa en Modelio

- Crear un proyecto activando la casilla Java Project
- Menú contextual del proyecto creado >>Java Designer >>Reverse >> Reverse Java Application from sources
- Seleccionar el src del proyecto en la carpeta y a continuación los paquetes que se quieran incluir en el diagrama
- Pulsar Next y Reverse.



Ingeniería inversa en Modelio

- Aparece en el explorador de Modelio las clases
- Pinchar y arrastrar al editor todos los elementos que se quieran incluir en el diagrama
- Colocar



Ingeniería inversa

- **Ingeniería inversa de datos:** Se aplica sobre algún código de bases datos (aplicación, código SQL, etc.) para obtener los modelos relacionales o sobre el modelo relacional para obtener el diagrama entidad-relación.
- **Ingeniería inversa de lógica o de proceso:** Cuando la ingeniería inversa se aplica sobre el **código** de un programa para averiguar su lógica (reingeniería), o sobre cualquier documento de diseño para obtener documentos de análisis o de requisitos.
- **Ingeniería inversa de interfaces de usuario:** Se aplica con objeto de mantener la lógica interna del programa para obtener los modelos y especificaciones que sirvieron de base para la construcción de la misma, con objeto de tomarlas como punto de partida en procesos de ingeniería directa que permitan modificar dicha interfaz.