

# Módulo Profesional 05: Entornos de desarrollo

## UF3 Actividad B

CICLO FORMATIVO DE GRADO SUPERIOR EN  
DESARROLLO DE APLICACIONES MULTIPLATAFORMA  
MODALIDAD ONLINE

ALUMNA:

**MARÍA LAURA RONDINA IOBBI**

### Descripción de la actividad:

En la siguiente actividad teórica se valorarán los conceptos más importantes de la actividad. Por lo que deberás de dar respuesta a los siguientes ejercicios teóricos en el mismo documento y entregar por el campus. Poner nombre al documento y entregar en formato Word/pdf.

Dar respuesta a los siguientes ejercicios teórico/práctico en el mismo documento para entregar por el campus.

## 1. ¿Qué es y para qué sirve el Lenguaje Unificado de Modelado (UML)? [1,0 puntos]

El **lenguaje unificado de modelado** (UML ó *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, respaldado por el *Object Management Group* (OMG). Es un lenguaje gráfico para **visualizar, especificar, construir y documentar un sistema**. Ofrece un total de 14 diagramas que representan el comportamiento del software, los cuáles se pueden clasificar básicamente en:

- **Diagramas Estructurales:** Presentan modelos estáticos del objeto, como clases, paquetes o componentes, aunque también pueden ser utilizados como modelos en tiempo de ejecución.
- **Diagramas de Comportamiento:** aquellos que nos muestran la forma que tienen de reaccionar ante los estímulos externos. Son diagramas dinámicos, basados en su devenir en tiempo de ejecución.

UML ofrece un **estándar** para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, expresiones de lenguajes de programación, esquemas de bases de datos... En realidad el término lenguaje no es el más apropiado para definirlo, porque no es un lenguaje propiamente dicho, sino una serie de normas y estándares gráficos respecto a cómo se deben representar los esquemas relativos al software.

UML es una **herramienta propia de analistas funcionales** (definen qué debe hacer un programa sin escribir el código) y **analistas programadores** (dado un problema, estudian cómo resolverlo y escriben el código en un lenguaje determinado). Es decir, de personas con conocimientos relativamente avanzados de programación.

## 2. Explica y realiza el Diagrama UML de representación de los siguientes elementos. [1,0 puntos]:

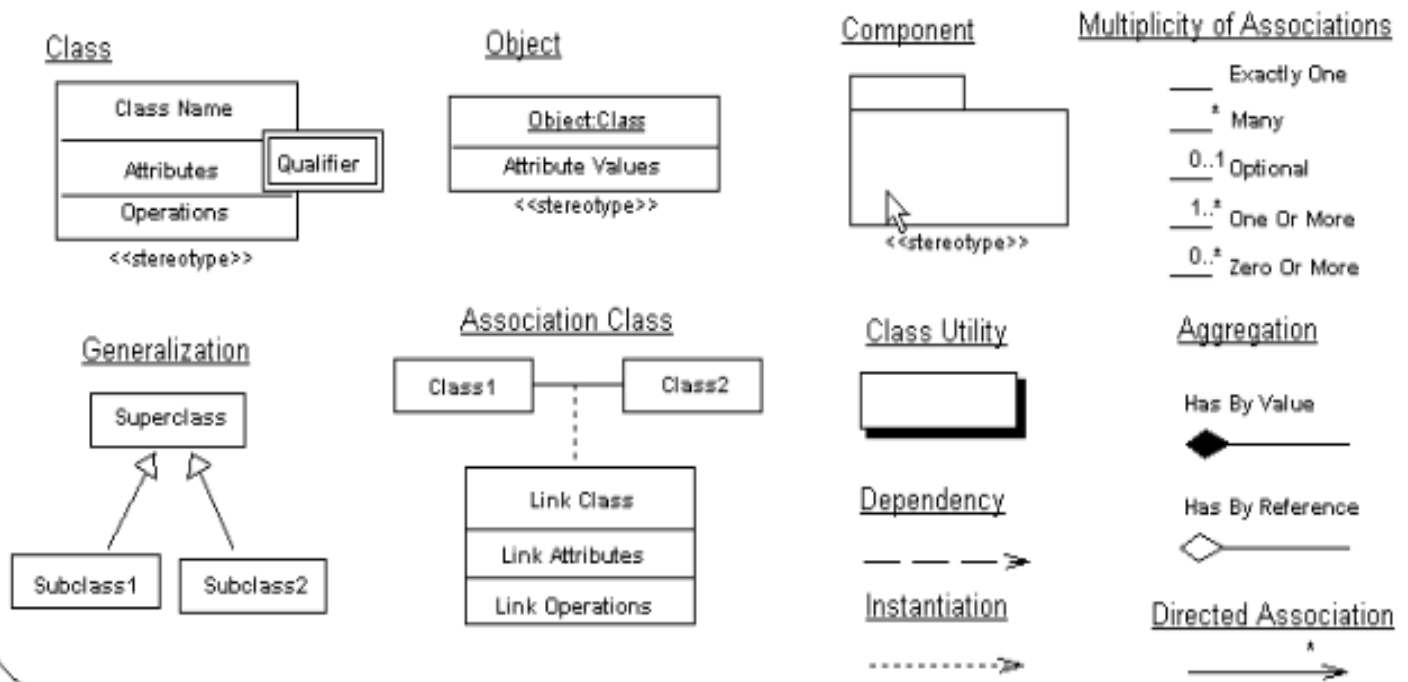
\*Una Clase

\*Un objeto

### ELEMENTOS DE UN DIAGRAMA DE CLASES:

Un **diagrama de clases** sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de agregación, ya que una clase es una descripción de conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica; mostrando un conjunto de elementos que son estáticos, como las clases y tipos junto con sus contenidos y relaciones. Un diagrama de clases está compuesto por los siguientes elementos: Clase: atributos, métodos y visibilidad. Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

# Class Diagram



## Notación de Clase:

Las clases se representan por rectángulos que muestran el nombre de la clase y opcionalmente el nombre de las operaciones y atributos. Los compartimientos se usan para dividir el nombre de la clase, atributos y operaciones. Adicionalmente las restricciones, valores iniciales y parámetros se pueden asignar a clases.

La clase contiene el nombre de la clase en el compartimiento más alto, el compartimiento siguiente detalla los atributos, y el último compartimiento muestra las operaciones.

La notación que precede el nombre del atributo u operación indica la visibilidad del elemento, si se usa el símbolo + el atributo y la operación tienen un nivel público de visibilidad, si se usa un símbolo – el atributo u operación es privado. Además, el símbolo # permite definir una operación o atributo como protegido y el símbolo ~ indica la visibilidad del paquete.

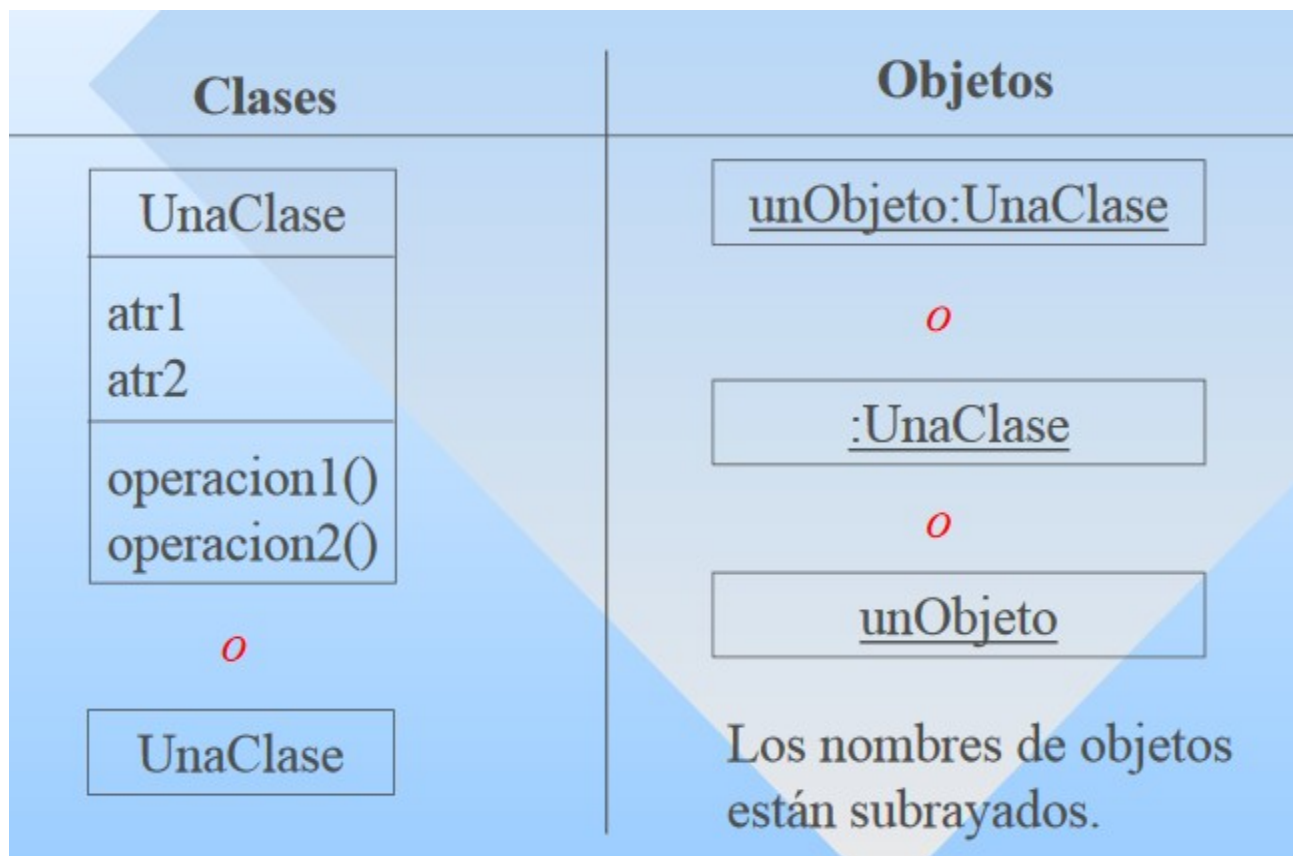
## Notación de Objeto:

El diagrama de objetos muestra las instancias creadas y los vínculos existentes entre ellas. Es por eso que este diagrama se utiliza una vez que el sistema está activo. Las instancias se representan dentro de un rectángulo con su nombre subrayado. Las relaciones entre las instancias se representan mediante líneas continuas.

## Diferencia de Diagramas:

El diagrama de clases da una representación estática del sistema. El diagrama de objetos es dinámico, depende del momento en el que lo observemos en el sistema y varía en función de las operaciones realizadas por el usuario.

## REPRESENTACIÓN DE CLASES Y OBJETOS:










**\*\*** En Visual Paradigm, represento a modo de ejemplo el diagrama UML de una **clase Vehiculo**, que tiene los siguientes atributos: una matrícula de tipo String, una marca de tipo String, un modelo de tipo String y un color de tipo String. También tiene una serie de operaciones, como la posibilidad de consultar la matrícula, la marca, el modelo y el color, devolviendo en todos los casos un resultado de tipo String. Y también contiene la operación de establecer o modificar la matrícula y el color.

Class Specification

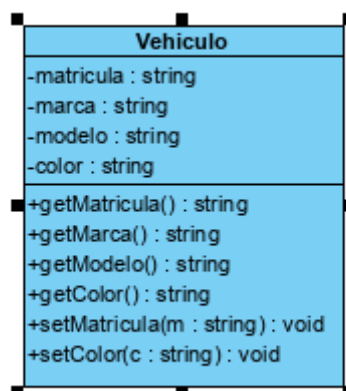
General | **Attributes** | Operations | Receptions

Name	Classifier	Visibility	Type	Initial Value
matricula	Vehiculo	private	string	
marca	Vehiculo	private	string	
modelo	Vehiculo	private	string	
color	Vehiculo	private	string	

 Class Specification

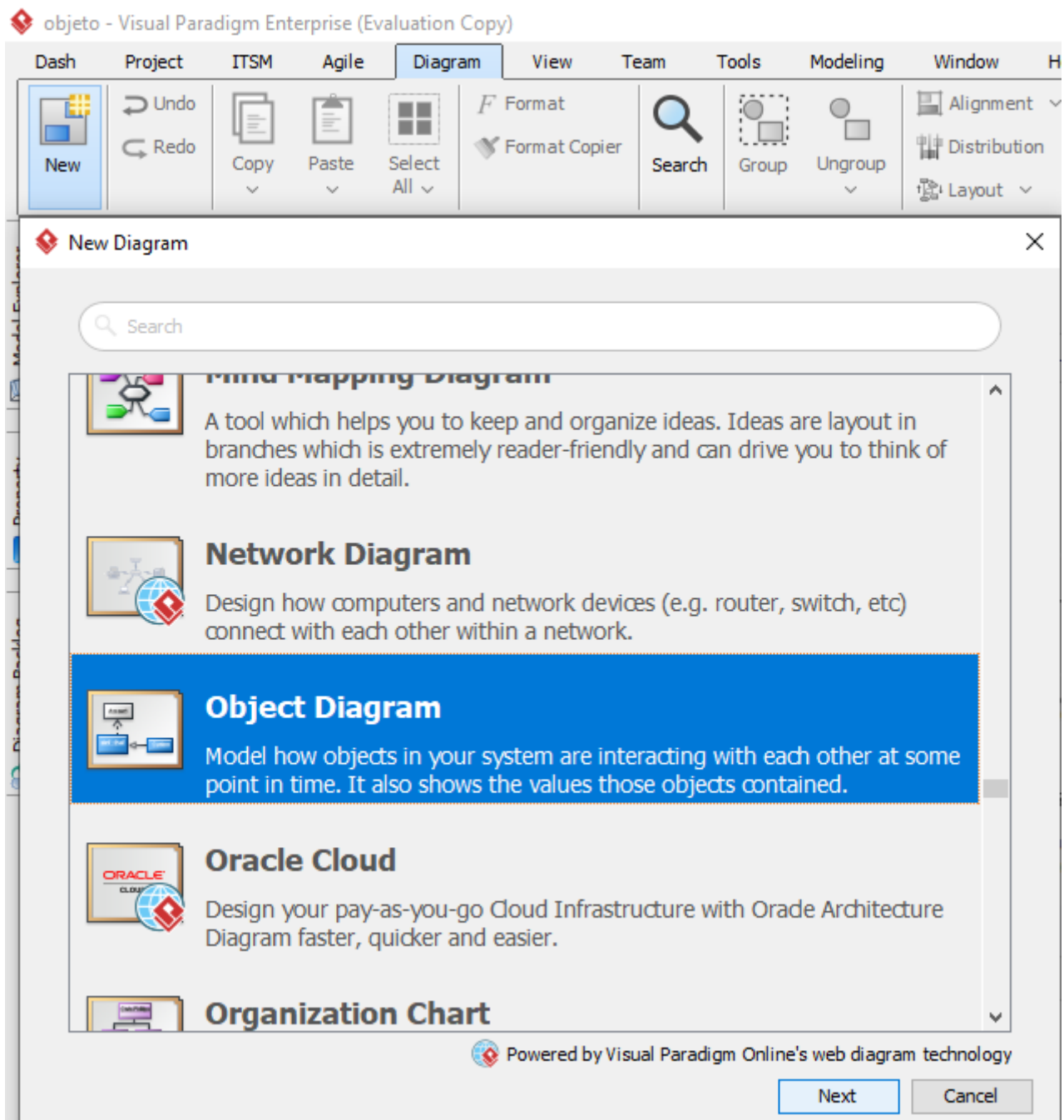
General		Attributes	Operations	Receptions
<input type="checkbox"/> Show parameters		<div>Override</div> <div>Overload</div>		
Name	Classifier	Visibility	Return type	
getMatricula	 Vehiculo	public	string	
getMarca	 Vehiculo	public	string	
getModelo	 Vehiculo	public	string	
getColor	 Vehiculo	public	string	
setMatricula	 Vehiculo	public	void	
setColor	 Vehiculo	public	void	

## **\*\*REPRESENTACIÓN DE LA CLASE Vehiculo:**



**\*\*** En Visual Paradigm, represento a modo de ejemplo el diagrama UML de un *objeto Coche* usando el Diagrama de objetos:

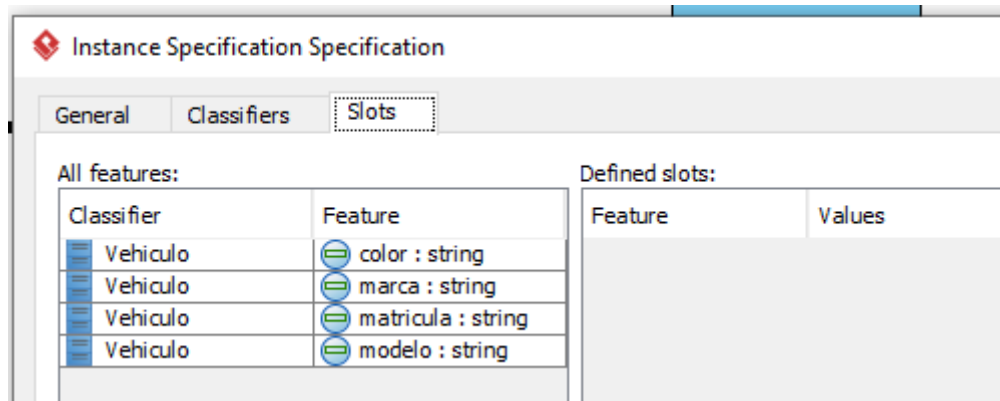
**Diagram → New → Object Diagram**



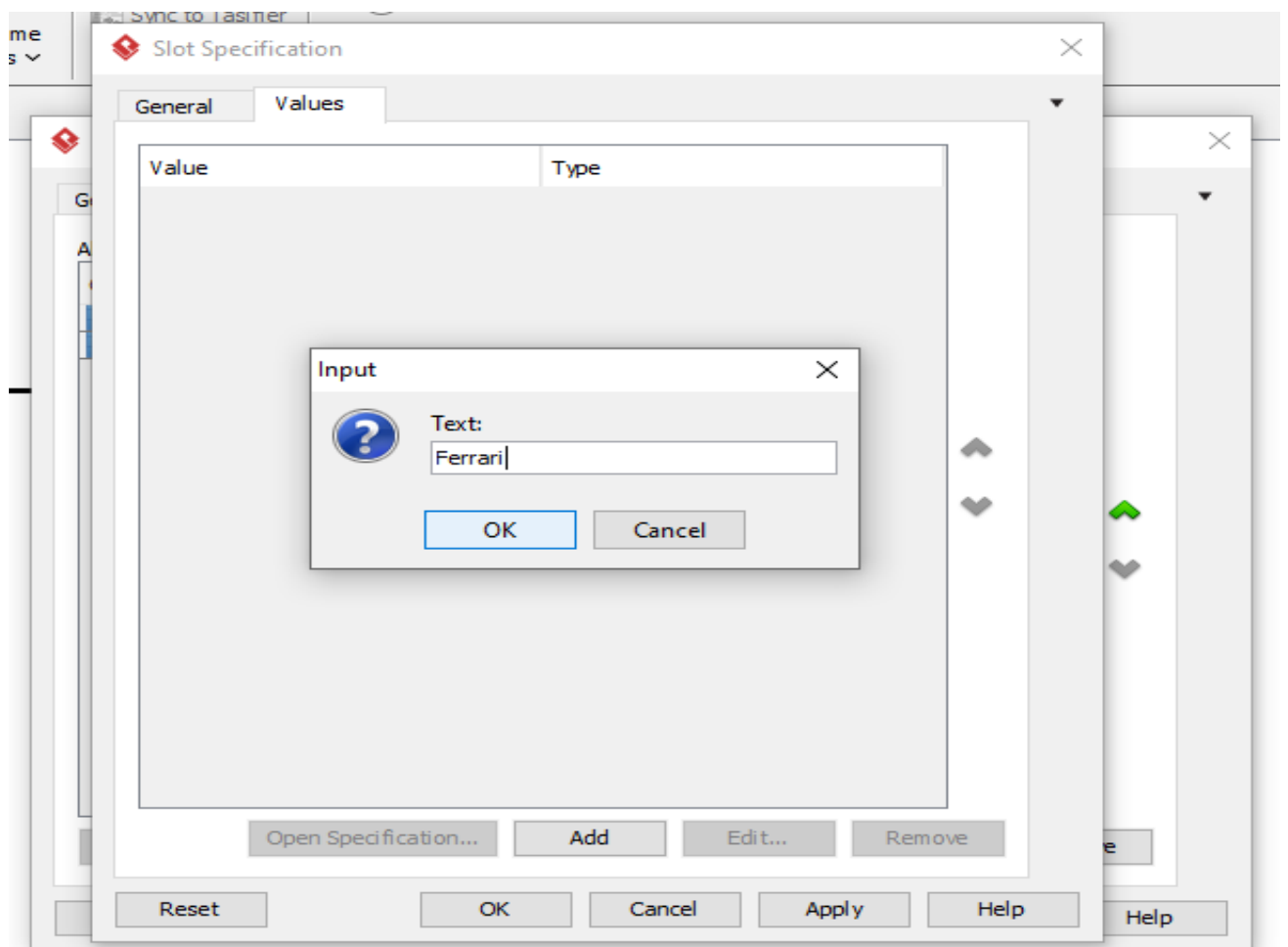
El objeto coche instanciado a partir de la clase Vehiculo hereda sus atributos y métodos, y además tiene su propia matrícula (XV1005L), marca (Ferrari) , modelo (California) y color (rojo).

En el panel de la izquierda clicamos en la opción **Instance Specification** para crear el objeto coche. En la pestaña **Classifiers** le asocio la clase a partir de la que se instancia (Vehiculo).

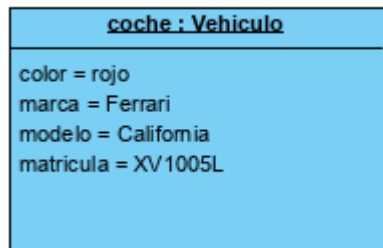




En la pestaña **Slots**, clicando sobre cada característica (**Feature**) , se activa el botón **Define Slot** donde debemos clicar, seguido de **Edit Values** → **Add** -> **Text**



**\*\*LA REPRESENTACIÓN DEL OBJETO Coche sería:**



3. Instalarse una de las herramientas de Diseño de diagramas de clase que hay de libre distribución para NetBeans/Eclipse y a partir de las especificaciones de un determinado escenario propuesto por ti realizar lo siguiente:

- Describir las especificaciones del escenario propuesto por ti. [2,0 puntos]
- Realiza el diagrama UML de clases basándote en las especificaciones del inciso a). [3,0 puntos]
- A partir del diagrama de clases del inciso b) genera el código. Te puedes ayudar de una de las herramientas como por ejemplo: "SDE de Visual Paradigma" que tiene una versión de prueba. [3,0 puntos]

Deberás explicar el proceso de los tres incisos y adjuntar capturas de pantalla.

a)

En el escenario propuesto para desarrollar el ejercicio, nos encontramos con una clase Empresa, entendiendo por Clase la unidad básica que encapsula toda la información de un objeto (instancia de esta clase). A través de ella se puede modelar el entorno de estudio. En UML, la clase Empresa es representada por un rectángulo que posee tres divisiones:

-El ámbito superior del rectángulo contiene el nombre de la clase (**Empresa**).

-El ámbito intermedio contiene los atributos o variables de instancia que caracterizan a la clase. En este escenario, dicha empresa tiene un nombre por atributo, almacenado en una variable llamada **nombreEmpresa** que es de tipo String.

-El ámbito inferior contiene los métodos u operaciones que permiten al objeto interactuar con su entorno, en este caso dos: un método que permite consultar el nombre de esta empresa, y por lo tanto, devuelve un String (**getNombreEmpresa()**), y otro método que, recibiendo un parámetro que almacena un valor de tipo String, permite establecer o modificar el nombre de la empresa (**setNombreEmpresa(String n)**).

De la misma forma contamos con una clase de nombre **Departamento**, que tiene un nombre por atributo (**nombreDepto**, de tipo String) y dos métodos para consultar y modificar dicho nombre (son **getNombreDepto()** y **setNombreDepto(String n)**, respectivamente).



La clase llamada **Oficina** posee una dirección por atributo (String **direccion**) y contiene dos métodos para consultar y modificar la dirección ( **getDireccion()** y **setDireccion(String n)** , respectivamente).

La clase **Empleado** posee dos atributos, **nombreEmpleado** y **apellido**, ambos de tipo String. Y contiene cuatro métodos que permiten consultar o modificar, tanto el nombre, como el apellido del empleado. Estos métodos son: **getNombreEmpleado()**, **setNombreEmpleado(String n)**, **getApellido()** y **setApellido(String t)** .

Por último, tenemos una subclase **Sede** , que hereda los métodos y atributos especificados en la Superclase Oficina (y que podría, además, contener los suyos propios). Esta relación de herencia la determina una flecha de punta blanca que apunta a la clase Padre. Sólo una oficina de todas las existentes podrá ser sede en la empresa.

Tanto Departamento como Oficina tienen una **relación de composición** con la clase Empresa, lo cuál se representa por medio de un diamante cerrado coloreado de negro que señala a la clase principal de esta relación: Si la empresa desapareciera, ni departamento, ni oficina, podrían seguir existiendo por sí mismos.

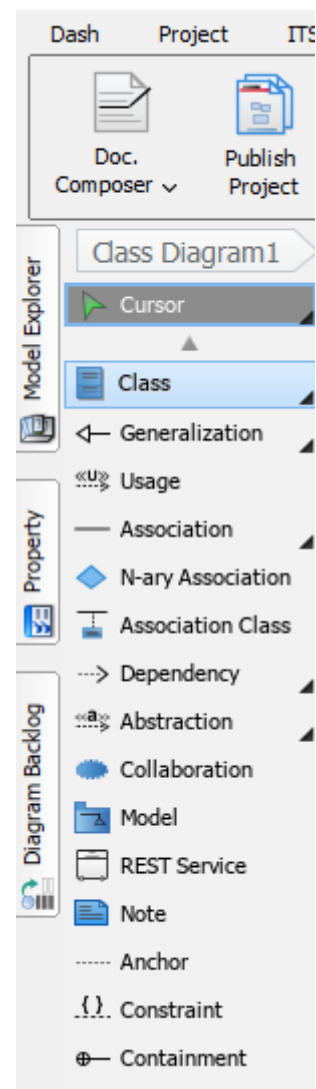
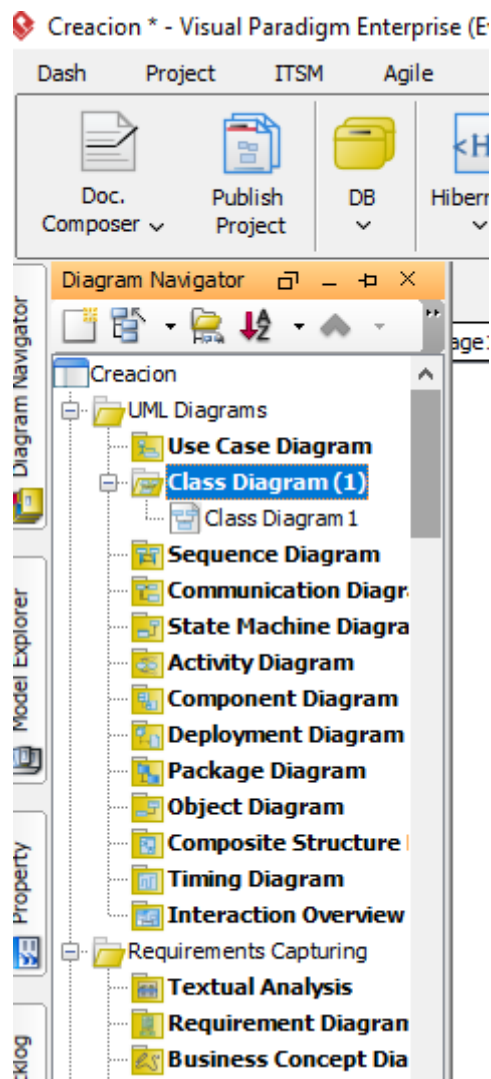
La empresa puede contener varios departamentos y oficinas. A su vez, cada departamento puede integrar varias oficinas (por razones de espacio necesario) y una oficina o local puede albergar varios departamentos diferentes. Esta **relación de multiplicidad** se representa con \* y la asociación entre las clases que se conectan viene marcada por una línea continua. Además, un departamento puede tener uno o más empleados (**1..\***) y sólo puede ser gestionado o dirigido por un único empleado con el rol de mando (**1**).

b)

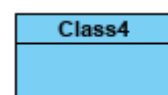
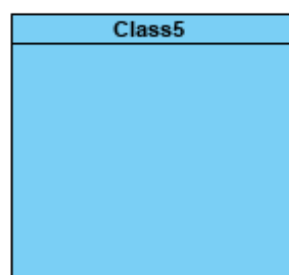
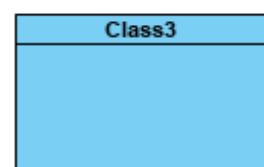
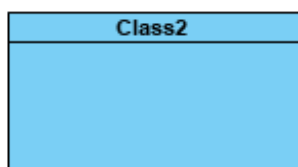
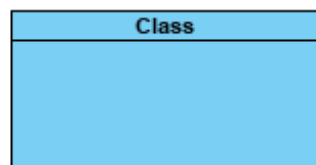
El siguiente diagrama de Clases UML lo realicé en la plataforma Visual Paradigm Enterprise.

En el panel de la izquierda, **Diagram Navigator**, se realiza clic derecho sobre el tipo diagrama deseado, en este caso, **Class Diagram** → **New Class Diagram**

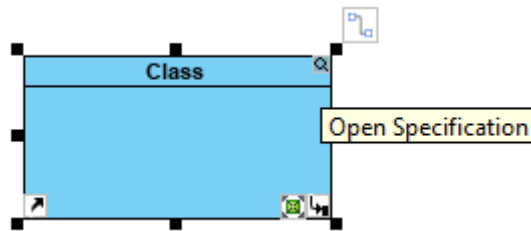
*En la paleta que se despliega, se localizan las herramientas para crear el Diagrama. Clic en Class para crear los rectángulos de clase:*



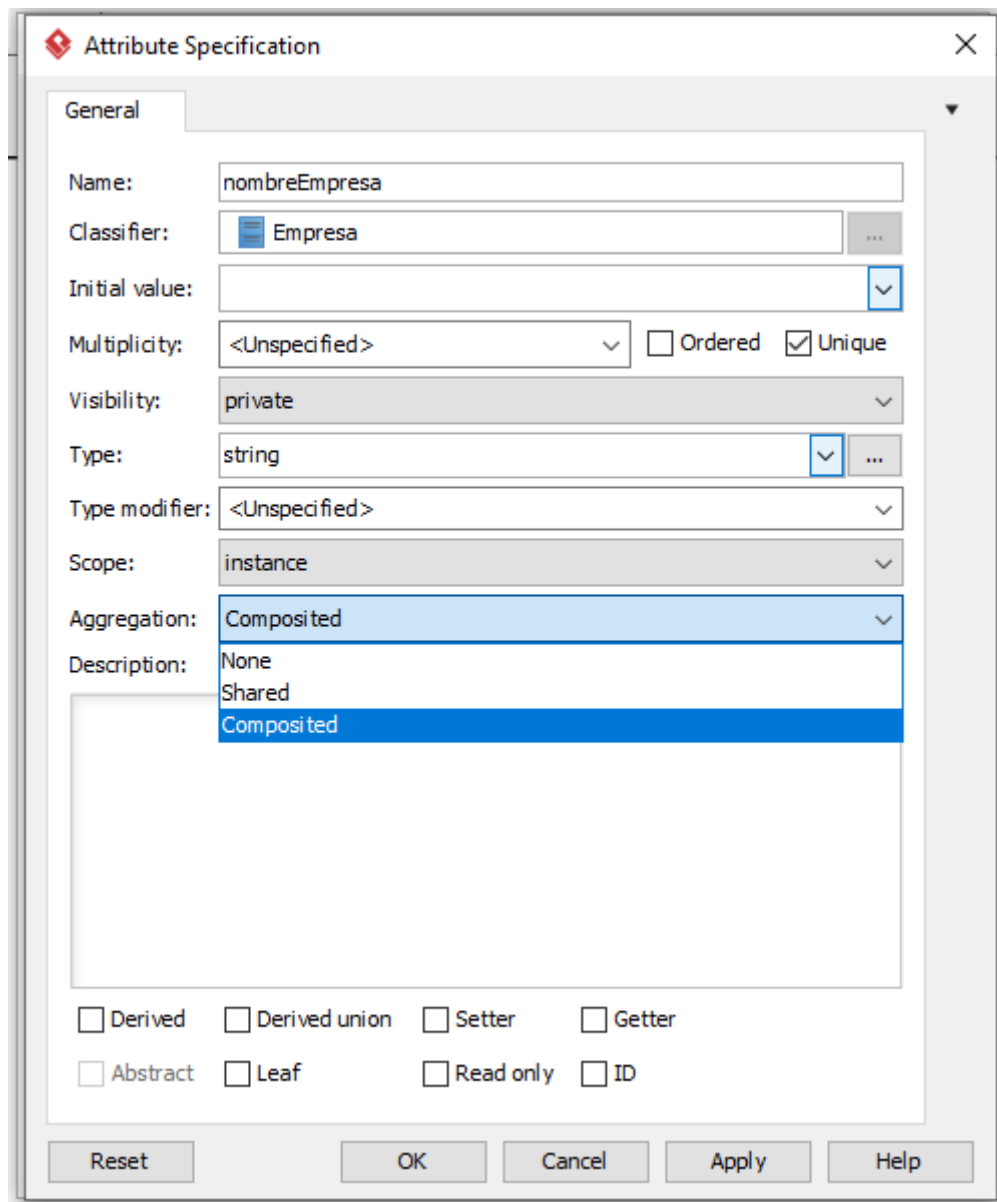
En el panel blanco central, clic izquierdo y arrastrar sin soltar, para dar forma a los **rectángulos de clase**.



Pinchando en la esquina superior derecha del rectángulo, aparece una especie de lupa sobre la que es posible clicar para desplegar un panel (***Open Specification***) en el que establecer las especificaciones de clase: nombre, visibilidad, atributos, operaciones, si hereda de alguna otra clase padre..

Una ventana de diálogo titulada "Class Specification" con una barra de cierre (X) en la esquina superior derecha. La ventana contiene cuatro pestañas: "General" (seleccionada), "Attributes", "Operations" y "Receptions".  
En la pestaña "General", hay los siguientes campos:  
- "Name:" con un campo de texto que contiene "Class".  
- "Parent:" con un campo de texto que contiene "<None>" y un botón de tres puntos a la derecha.  
- "Visibility:" con un menú desplegable que muestra "public".  
- "Description:" con un área de texto grande y vacía.  
En la parte inferior de la pestaña "General", hay una fila de casillas de verificación con las siguientes etiquetas: "Abstract", "Leaf", "Root", "Active" y "Business model".  
En la parte inferior de la ventana, hay una fila de botones: "Reset", "OK", "Cancel", "Apply" y "Help".

En dicha ventana, completamos las especificaciones de los atributos de cada una de las clases: nombre, multiplicidad, visibilidad, tipo, relación de agregación/composición, etc...



The image shows a 'UML Attribute Specification' dialog box. It has a 'General' tab. The fields are as follows:

- Name: nombreEmpresa
- Classifier: Empresa
- Initial value: (empty)
- Multiplicity: <Unspecified>, Ordered: ☐, Unique: ☒
- Visibility: private
- Type: string
- Type modifier: <Unspecified>
- Scope: instance
- Aggregation: Composit ed
- Description: A list box containing 'None', 'Shared', and 'Composit ed' (which is selected).

At the bottom, there are checkboxes for:

- Derived: ☐ Derived union: ☐ Setter: ☐ Getter: ☐
- Abstract: ☐ Leaf: ☐ Read only: ☐ ID: ☐

Buttons at the bottom: Reset, OK, Cancel, Apply, Help.

Rellenamos las operaciones de la clase: nombre, tipo de retorno, visibilidad, cuerpo de la operación, descripción de la operación.. y además, si el método recibe un parámetro, también se rellenan las especificaciones del mismo.

**Operation Specification**

General Parameters

Name:

Classifier:  ...

Return type:  ...

Type modifier:

Visibility:

Scope:

Lower:

Upper:

Body condition:

Description:

☐ Abstract ☐ Leaf ☐ Query ☐ Ordered ☒ Unique

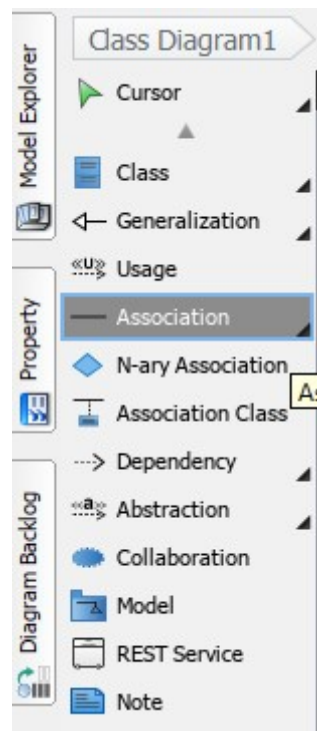
Reset OK Cancel Apply Help

Al pinchar en **Apply** → **Ok** obtenemos el resultado:

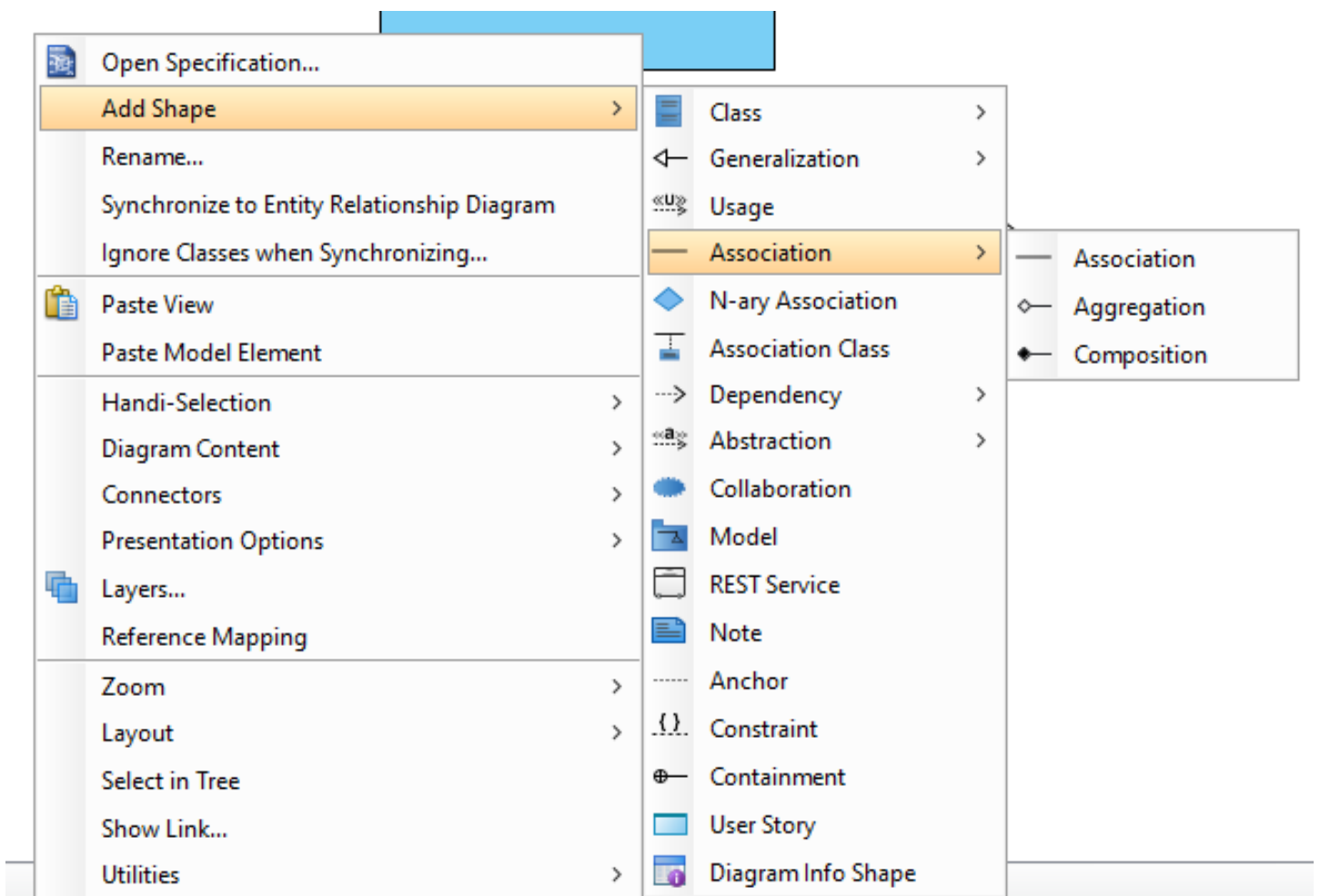
Empresa
-nombreEmpresa : string
+getNombreEmpresa() : string

Luego de rellenar las especificaciones de todas las clases, procedo a establecer las relaciones de asociación, generalización, composición y multiplicidad entre las clases que contiene mi modelo, con las herramientas del panel izquierdo o bien haciendo **click derecho en el panel central** → **Add Shape**

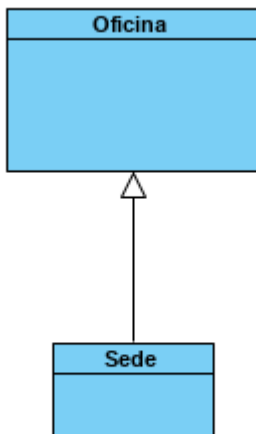
Panel Izquierdo:



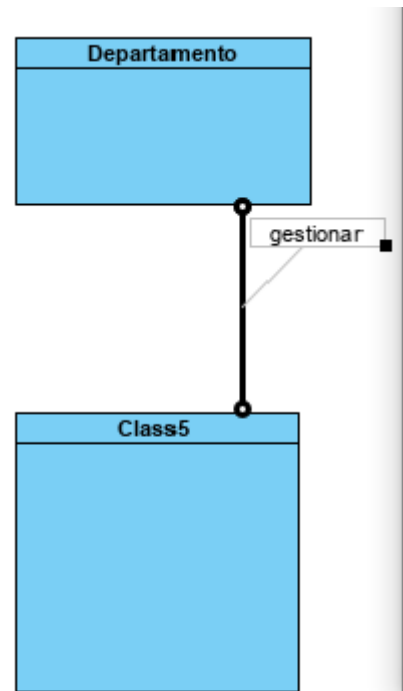
o Panel central:



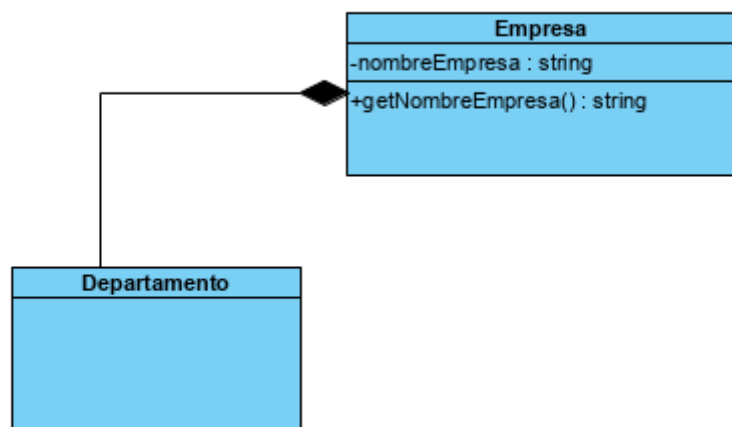
→ **Generalización:**



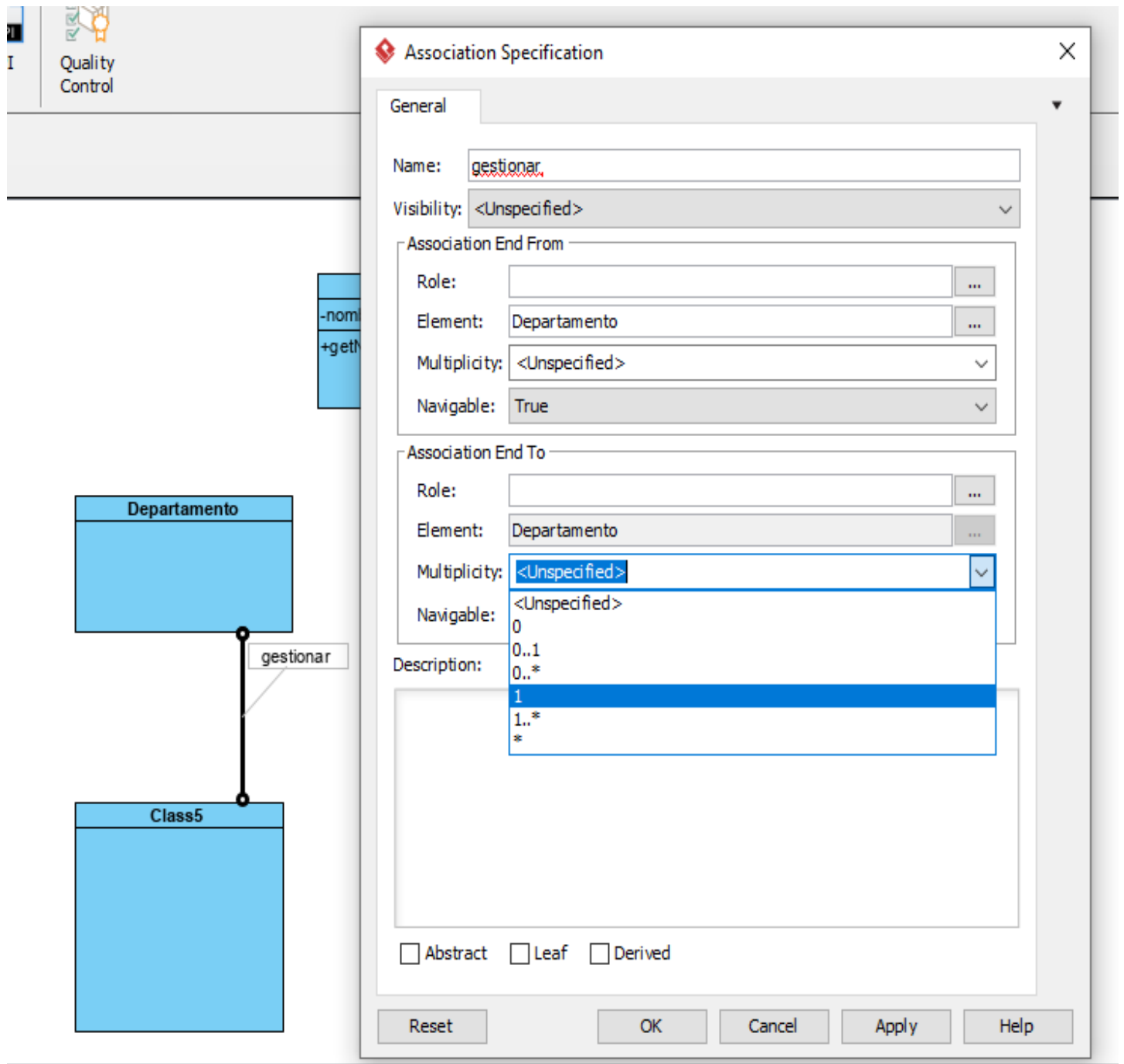
→ **Asociación:**



→ **Composición:**

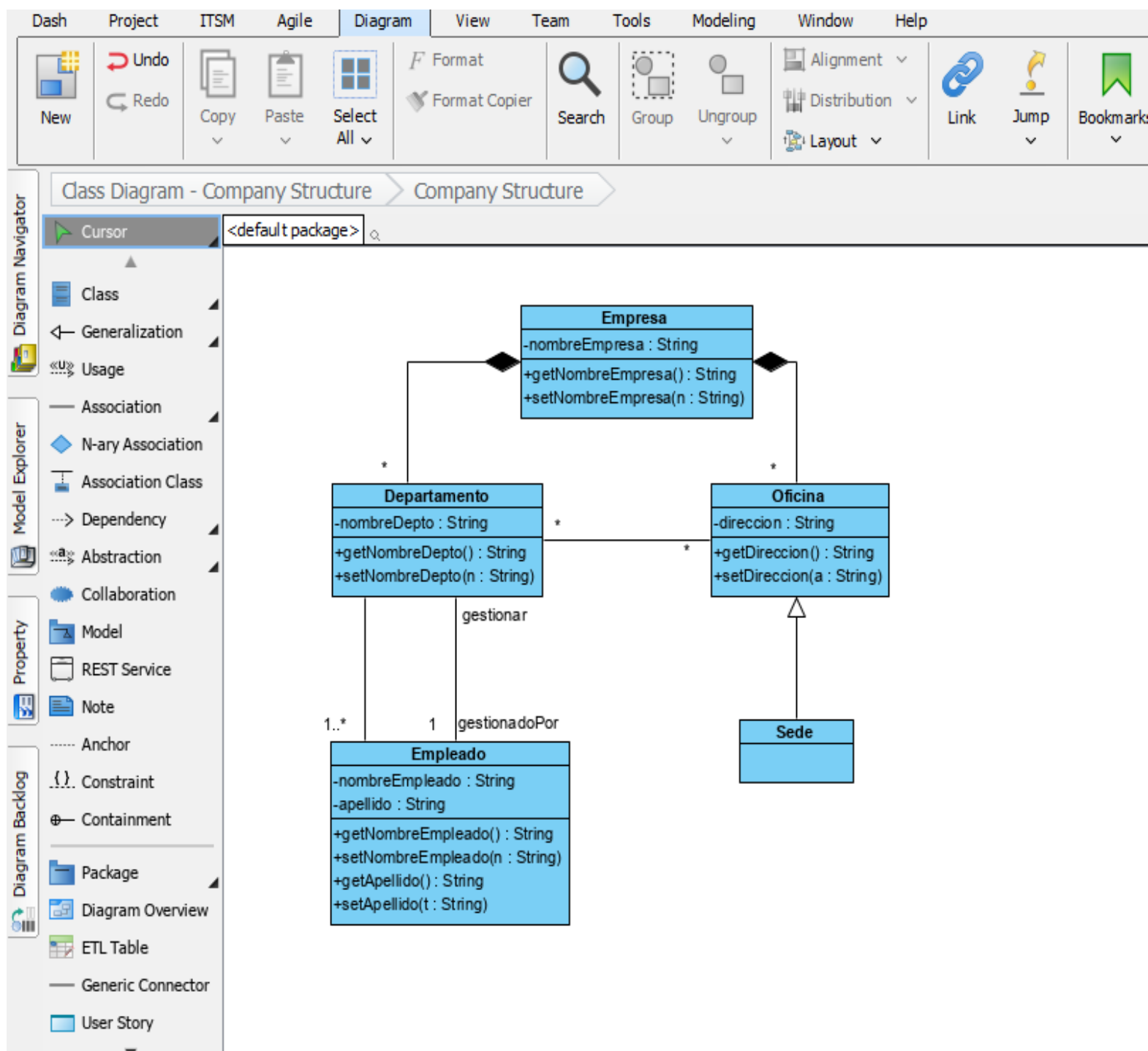


Si realizamos clic con botón derecho sobre la línea de relación creada, por ejemplo, la de asociación, se pueden establecer las especificaciones de la misma, incluida la relación de multiplicidad.

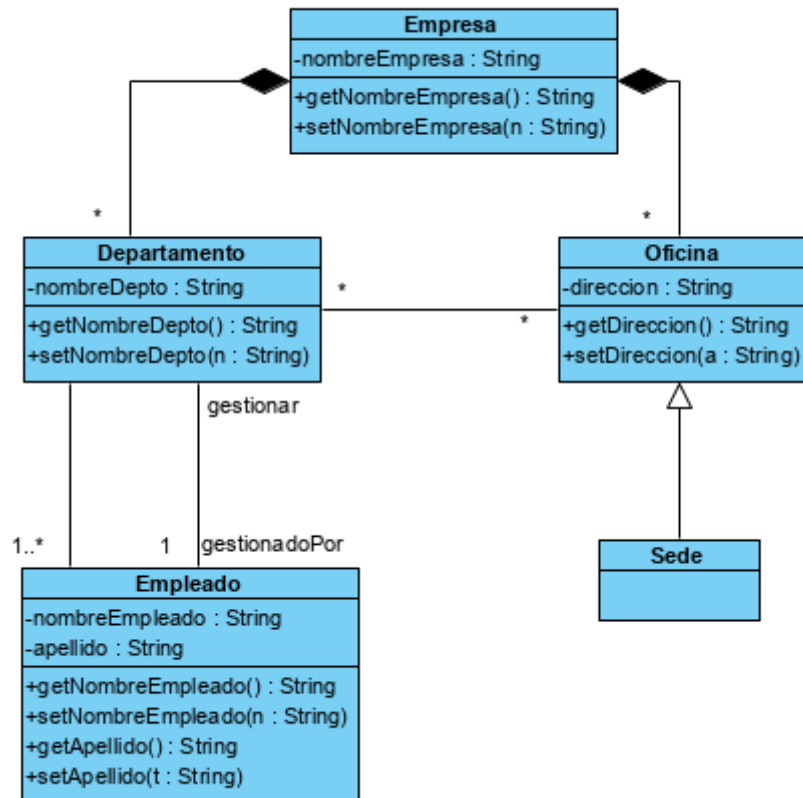


*El resultado final, aplicando todas las herramientas disponibles para construir mi modelo UML, es:*





*Haciendo un zoom sobre el diagrama anterior, para apreciarlo mejor, tenemos:*

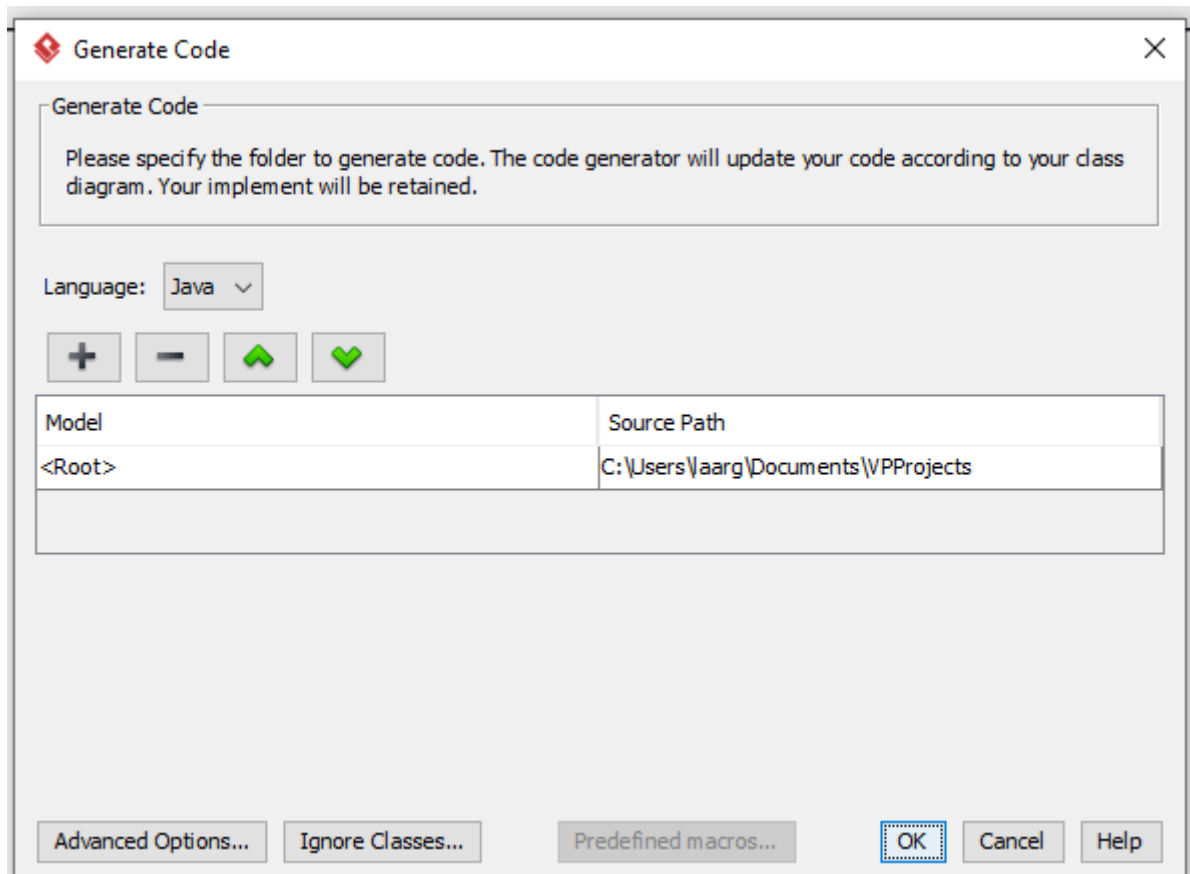
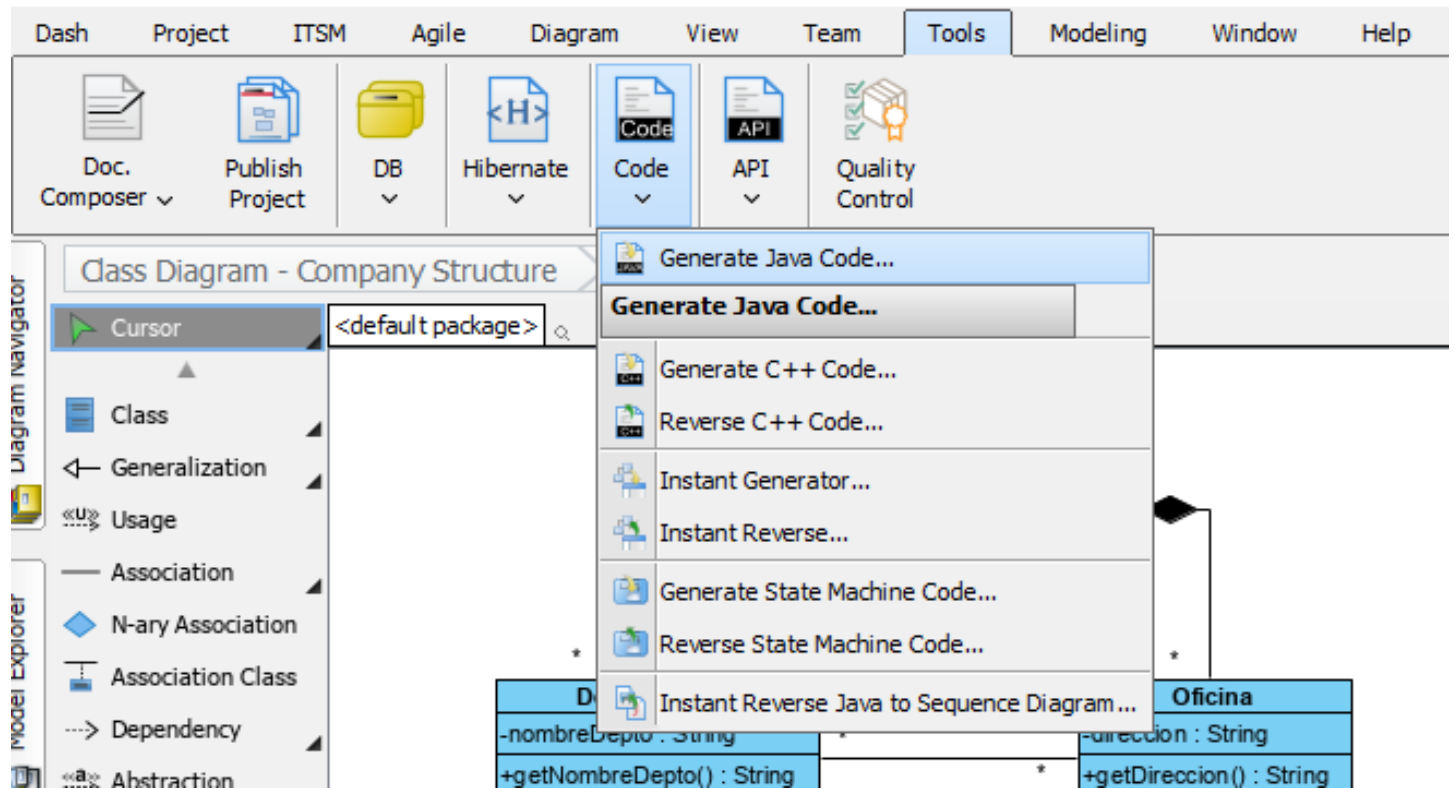


c)









A través de Visual Paradigm, procedemos a generar el código del anterior diagrama de clases UML, por medio de la siguiente ruta:

**Tools → Code → Generate Java Code → Ok**

EjercicioED \* - Visual Paradigm Enterprise (Evaluation Copy)



Sigo la ruta o Source Path para localizar el código generado y encuentro los 5 archivos .java correspondientes a las clases del diagrama: Empresa, Departamento, Empleado, Oficina y Sede.

Este equipo > Documentos > VPPProjects				
	Nombre	Fecha de modifica...	Tipo	Tamaño
	 .EjercicioED.vpp.lck	15/11/2019 23:26	Archivo LCK	0 KB
	 Departamento	15/11/2019 23:37	Archivo JAVA	1 KB
	 EjercicioED	15/11/2019 23:38	Visual Paradigm P...	652 KB
5	 EjercicioED.vpp.bak_000f	15/11/2019 21:24	Archivo BAK_000F	609 KB
	 Empleado	15/11/2019 23:37	Archivo JAVA	1 KB
	 Empresa	15/11/2019 23:37	Archivo JAVA	1 KB
	 Oficina	15/11/2019 23:37	Archivo JAVA	1 KB
	 Sede	15/11/2019 23:37	Archivo JAVA	1 KB

### **CÓDIGO DE LA CLASE “Empresa”:**

```
public class Empresa {  
  
    private String nombreEmpresa;  
  
    public String getNombreEmpresa() {  
  
        return this.nombreEmpresa;  
  
    }  
  
    /**  
  
    *  
  
    * @param n  
  
    */
```

```
public void setNombreEmpresa(String n) {  
  
    this.nombreEmpresa = n;  
  
}  
  
}
```

### **CÓDIGO DE LA CLASE “Departamento”:**

```
public class Departamento {  
  
    Empleado gestionadoPor;  
  
    private String nombreDepto;  
  
    public String getNombreDepto() {  
  
        return this.nombreDepto;  
  
    }  
  
    /**  
  
    *  
  
    * @param n  
  
    */  
  
    public void setNombreDepto(String n) {  
  
        this.nombreDepto = n;  
  
    }  
  
}
```

```
}
```

### **CÓDIGO DE LA CLASE “Empleado”:**

```
public class Empleado {  
  
    Departamento gestionar;  
  
    private String nombreEmpleado;  
  
    private String apellido;  
  
  
    public String getNombreEmpleado() {  
  
        return this.nombreEmpleado;  
  
    }  
  
    /**  
  
    *  
  
    * @param n  
  
    */  
  
    public void setNombreEmpleado(String n) {  
  
        this.nombreEmpleado = n;  
  
    }  
  
    public String getApellido() {  
  
        return this.apellido;  
  
    }  
}
```

```
}

/**

*

* @param t

*/

public void setApellido(String t) {

    this.apellido = t;

}

}
```

### **CÓDIGO DE LA CLASE “Oficina”:**

```
public class Oficina {

    private String direccion;

    public String getDireccion() {

        return this.direccion;

    }

    /**

    *

    * @param a

    */

}
```

```
public void setDireccion(String a) {
```

```
    this.direccion = a;
```

```
}
```

```
}
```

### **CÓDIGO DE LA CLASE “Sede”:**

```
public class Sede extends Oficina {
```

```
}
```