ENTORNOS DE DESARROLLO

2013

Egibide

Entornos de Desarrollo

2015

Departamento de Informática

Elaboración de diagramas de clase

Forma parte de la vista estática del sistema. En el diagrama de clases definiremos las características de cada una de las clases, interfaces, colaboraciones y relaciones de dependencia y generalización. Es decir, daremos rienda suelta a nuestros conocimientos de diseño orientado a objetos, definiendo las clases e implementando las relaciones.

Clase

El elemento clase se representa en UML de la siguiente manera:

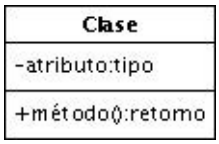


Figura 2.1: Esquema básico de una clase en UML.

El compartimento superior sirve para colocar el nombre de la clase. Tiene que ser único y es recomendable utilizar la notación camel para escribirlo. Esta notación capitaliza la primera letra de la palabra que escribamos, así como la de la cada una de las palabras que compongan el nombre de la clase, si es que son varias. No es recomendable utilizar acentos en los nombres de clases.

También podemos optar por no mostrar alguno de los compartimentos que componen nuestra clase, para simplificar el diagrama que mostramos en un momento dado. Esta funcionalidad es particularmente útil cuando tenemos diagramas complejos y los queremos mostrar a los usuarios sin abrumar con los detalles, lo que facilita mucho la tarea de explicar qué hace nuestra aplicación y cómo está construida.

Atributos

Ya hemos comentado que el compartimento superior de una clase está destinado a contener los atributos de esa clase.

Para escribir un atributo seguiremos la sintaxis:

[+|-]nombre:tipo

Las características básicas que definen un atributo son:

- **Visibilidad** Indica si el atributo es visible o no desde el exterior de la clase. Lo más habitual será encontrar un símbolo + cuando sea visible y un - cuando esté oculto. Es muy raro encontrar atributos que no estén “ocultos” dentro de la clase, por lo que por defecto todas las aplicaciones de modelado UML aplican el valor - a los atributos.
- **Nombre** A ser posible, utilizando notación camel, excepto en la primera letra. Esto permite distinguir rápidamente en el programa a las clases de los atributos. Sin acentos.
- **Tipo** Después de los : aparece el tipo de dato que almacena ese atributo. Puede ser cualquier tipo

del lenguaje u otra clase que exista en nuestro diagrama. Así tenemos la clase Vendedor, que tiene un atributo denominado porcentajeComision, de tipo int y con visibilidad oculta:

Métodos

Para detallar un método en un diagrama UML tendremos que utilizar esta otra sintaxis:

Figura 2.2: La clase Vendedor en notación UML.

[+|-]nombre([parametro:tipo][,parametro:tipo]):tipoRetorno*

La expresión regular contiene los siguientes elementos:

- **Visibilidad** Indica si el método es visible o no desde el exterior de la clase. Lo más habitual será encontrar un símbolo + cuando sea visible y un - cuando esté oculto. En el caso de los métodos lo normal es que sean visibles al exterior, aunque no es raro encontrar métodos ocultos.
- **Nombre** A ser posible, utilizando notación camel, excepto en la primera letra. Al igual que para los atributos, esto permite distinguir rápidamente en el programa a las clases de los métodos. También sin acentos, para evitar complicaciones en el código fuente.
- **Lista de parámetros** Contiene una lista de tamaño ilimitado y que puede estar vacía (es lo que indica el calificador * en la expresión regular). Cada uno de los parámetros es una pareja nombre:tipo y tendremos que separarlos usando comas. Los paréntesis que rodean a la lista de parámetros deben aparecer siempre, aunque esté vacía.
- **Tipo de retorno** Después de la lista de parámetros colocamos el símbolo : y a continuación el tipo de dato que devolverá nuestro método. Nuevamente, puede ser un tipo del lenguaje u otro objeto, etc.

Ejercicios

1. Crea el diagrama de clases de un programa que solicite al usuario dos números enteros y la operación matemática simple que se desea desarrollar sobre ellos (suma o resta). Tras la recepción de los datos, el programa mostrará el resultado de la operación.
2. Crea el diagrama de clases de un programa para calcular una línea de un ticket de supermercado. El usuario introduce el nombre del producto, el precio por unidad y el número de unidades. El programa sacará en pantalla el nombre del producto, las unidades vendidas y el precio total.
3. Crea un diagrama de clases de un programa que genera un listín de teléfonos con los datos de los alumnos de clase. De cada alumno guarda su código, nombre, domicilio y teléfono.

Relaciones

Normalmente las clases no se encuentran aisladas en nuestras aplicaciones. Precisamente, la facilidad de uso y desarrollo que ofrece la programación OO radica en la colaboración entre los objetos y cuando estamos modelando la vista estática del sistema, el diagrama de clases nos permite mostrar

cuales son las vías por las cuales los objetos se podrán comunicar.

En la figura vemos un diagrama de ejemplo que contiene prácticamente todos los elementos que encontraremos en los diagramas típicos.

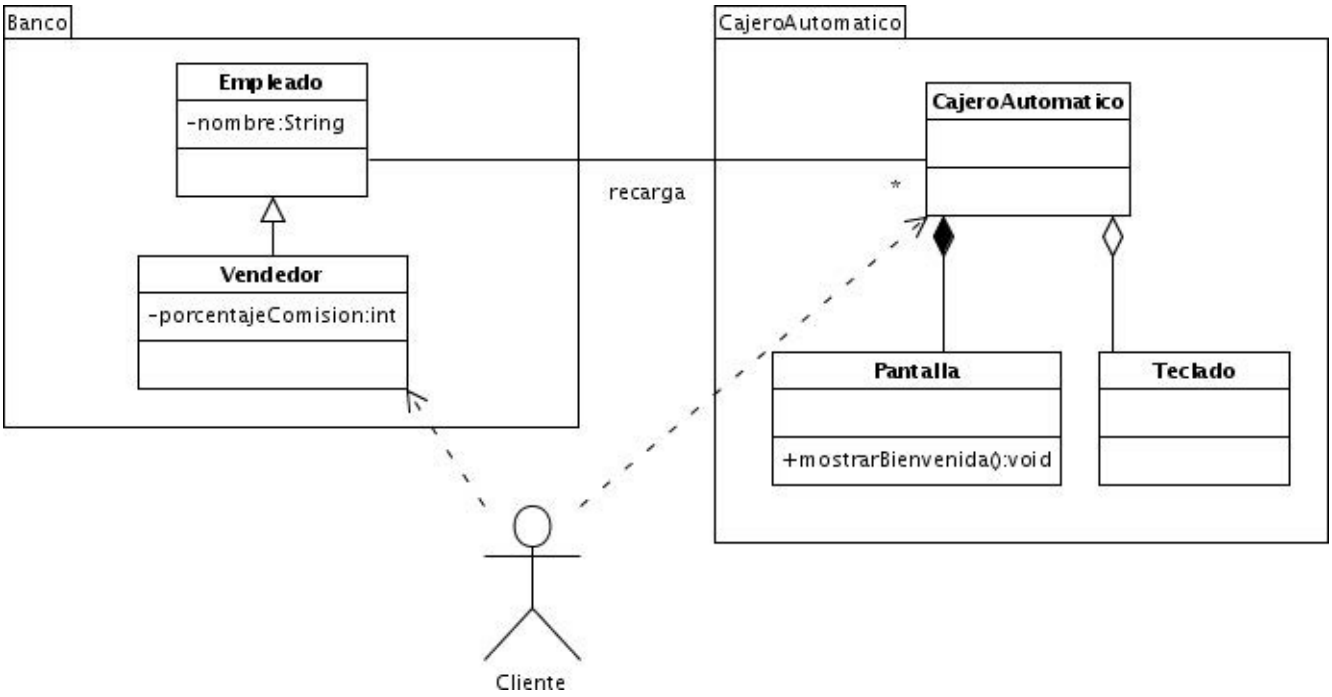


Figura 2.3: Diagrama de clases del Banco.

Podemos encontrar clases, una asociación con multiplicidad, dos dependencias, una generalización, una composición y una agregación, además de dos paquetes y un actor. Lo dicho, de todo un poco.

Vamos a estudiar las relaciones que aparecen en este diagrama una por una.

Asociación

Es una relación simple entre dos clases u objetos. Representa un camino de comunicación entre las dos clases. Sus características principales son:

- Se dibuja como una linea recta con flechas opcionales que permiten definir la “navegabilidad” de la asociación. Si no hay flechas los dos objetos situados a ambos lados pueden enviarse mensajes mutuamente. Si hay alguna flecha, los mensajes sólo pueden ir en el sentido de la flecha.
- Tiene “multiplicidad”, lo que indica el número de ocurrencias de una clase (objetos) que puede haber en cada lado. En el dibujo tenemos una asociación de “1” (no hay número) a “*” (0 o más). También es habitual encontrar asociaciones con un número concreto o con 1..* que significa “1 o más” (por lo menos 1).
- Suelen disponer de nombre a cada uno de los extremos, para clarificar su significado. En el ejemplo sólo hay un nombre, pero ya nos da a entender que vamos a usar esa asociación en caso de que el objeto de la clase Empleado quiera “recargar” uno de los múltiples objetos de la clase Cajero (enviándole un mensaje para que ejecute sumétodo recarga(importe:int)).

Las asociaciones son las relaciones más comunes entre clases y además son el caso general de los

otros dos tipos de relaciones que vamos a ver a continuación.

Agregación

La agregación es un tipo de asociación específico que tiene una semántica ligeramente distinta a la asociación normal.

Se representa como una linea recta con un rombo vacío en el extremo “agregado”.

Dispone de los mismos calificadores que una asociación normal aunque en el extremoagregado no suele ser habitual que la multiplicidad sea más de 1.

Su significado viene a ser “el agregado está formado por un conjunto de. . . ”, siendo los elemento que lo componen las clases que están unidas al agregado.

Un ordenador y sus componentes son un buen ejemplo de agregación.

Composición

La composición es un tipo de agregación con un significado todavía más fuerte: los componentes pertenecen sólo a un todo. Es más, si eliminamos alguno de los componentes el todo deja de existir.

Se representa igual que una agregación, pero con el rombo relleno.

Su significado suele ser **“se compone de. . . ”.**

Una *MesaCafe* compuesta de un TableroMesa y cuatro Patas sería un buen ejemplo.

Generalización (Herencia)

En el ejemplo también podemos ver el símbolo de una generalización. Se representa con una flecha de cabeza hueca, desde la clase derivada hasta la clase base.

Su significado siempre ha de ser tomado de abajo a arriba como “extiende o amplia la funcionalidad de. . . ”.

En el diagrama anterior aparecen además otros elementos nuevos:

Paquete

Tiene forma de ficha con un título en la parte superior izquierda. Sirve para agrupar partes de nuestro sistema y clarificar el diseño. Su finalidad es meramente organizativa.

Actor

Es la figura de un hombrecillo. Aunque no pertenece estrictamente al diagrama de clases, se puede

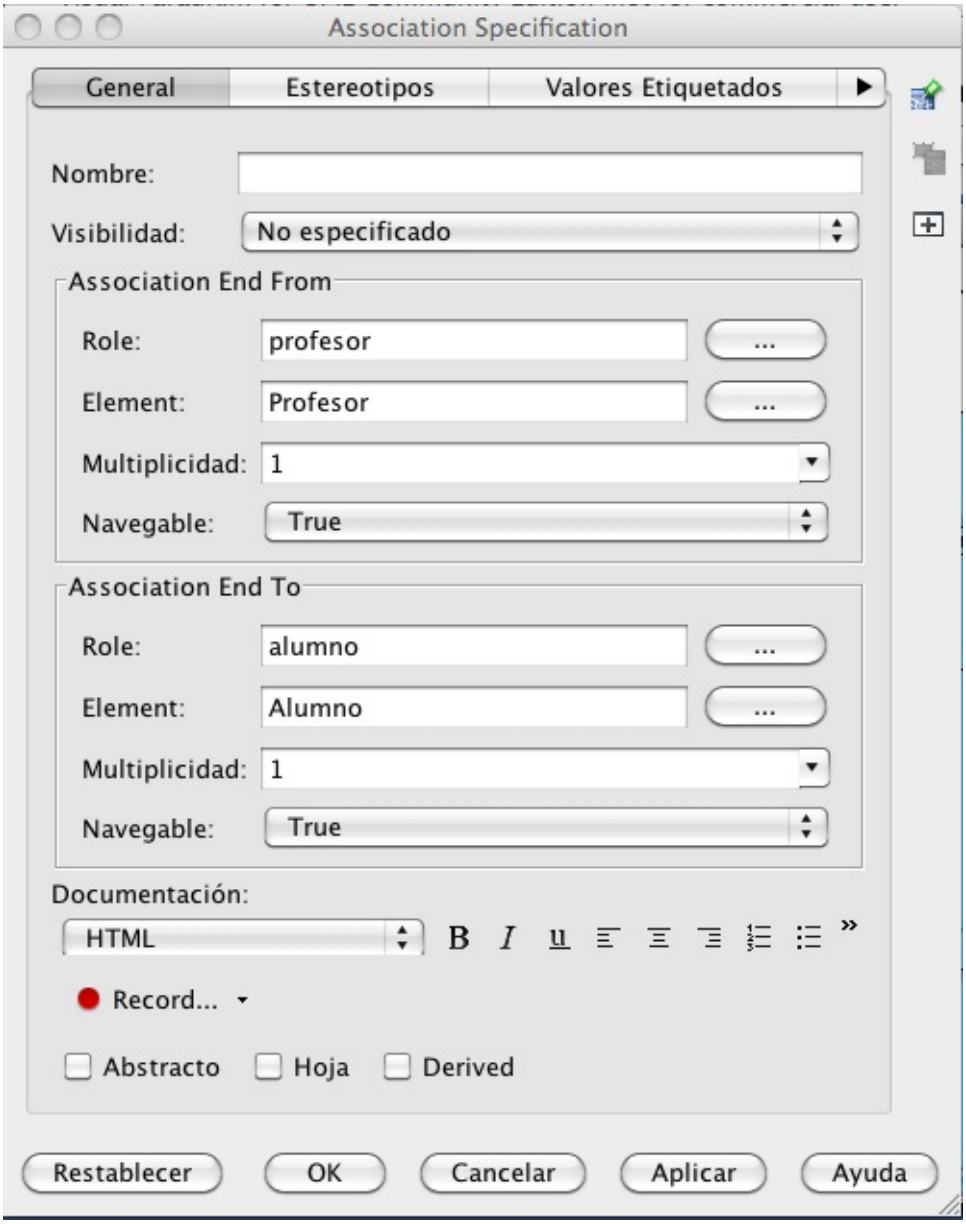
incluir cuando necesitamos clarificar cómo se produce la interacción entre el usuario del sistema y las clases de nuestra aplicación. Para representar esto podemos utilizar las relaciones de dependencia (las flechas punteadas) que aparecen en el diagrama.

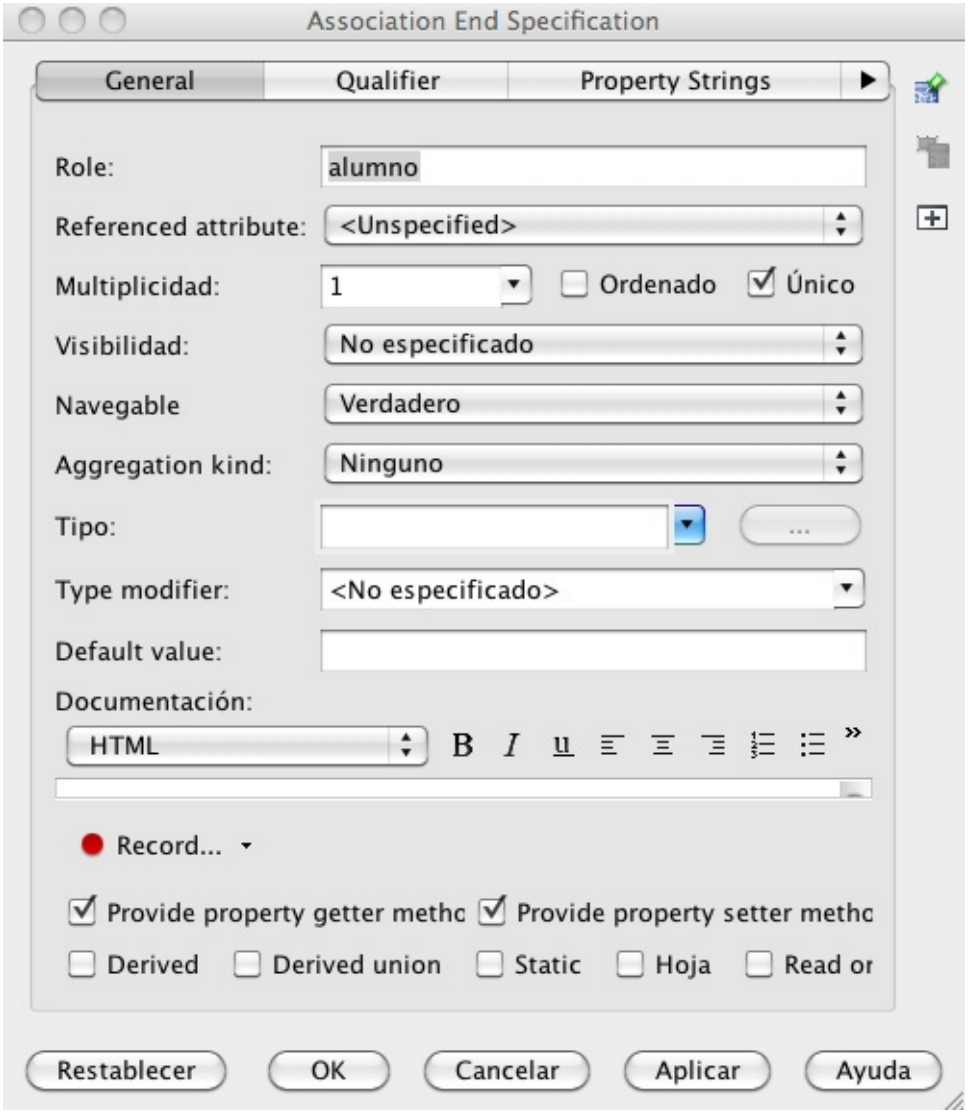
Ejercicios

1. **Ejercicio 1:** Modela dos clases con una relación de 1 a 1 entre ellas. Observa el código que se genera.
2. **Ejercicio 2:** Modela dos clases con una relación de 1 a * entre ellas. Observa el código que se genera.
3. **Ejercicio 3:** Modela dos clases con una relación de * a * entre ellas. Observa el código que se genera.
4. **Ejercicio 4:** Modela dos clases con una relación de 1 a 0..* entre ellas. Observa el código que se genera.
5. **Ejercicio 5:** Modela cuatro clases con una relación de **herencia** entre ellas (persona, trabajador,cliente,proveedor). Observa el código que se genera.
6. **Ejercicio 6:** Modela cuatro clases con una relación de **agregación** entre ellas (ordenador, teclado, pantalla, ratón). Observa el código que se genera.
7. **Ejercicio 7:** Modela dos clases con una relación de tipo **composición** entre ellas (empresa, departamentos). Observa el código que se genera.
8. **Ejercicio 8:** Tras observar el código que nos genera y donde lo deja. Une dos proyectos (uno Java y otro UML) y aprende a utilizar el concepto de la reingenieria inversa en los dos sentidos.
9. **Ejercicio 9:** Crea y utiliza desde el método main, alguna clase generada automáticamente a través del diagrama de clases.

Opciones para resolver los ejercicios en el entorno de desarrollo (Visual Paradigm)

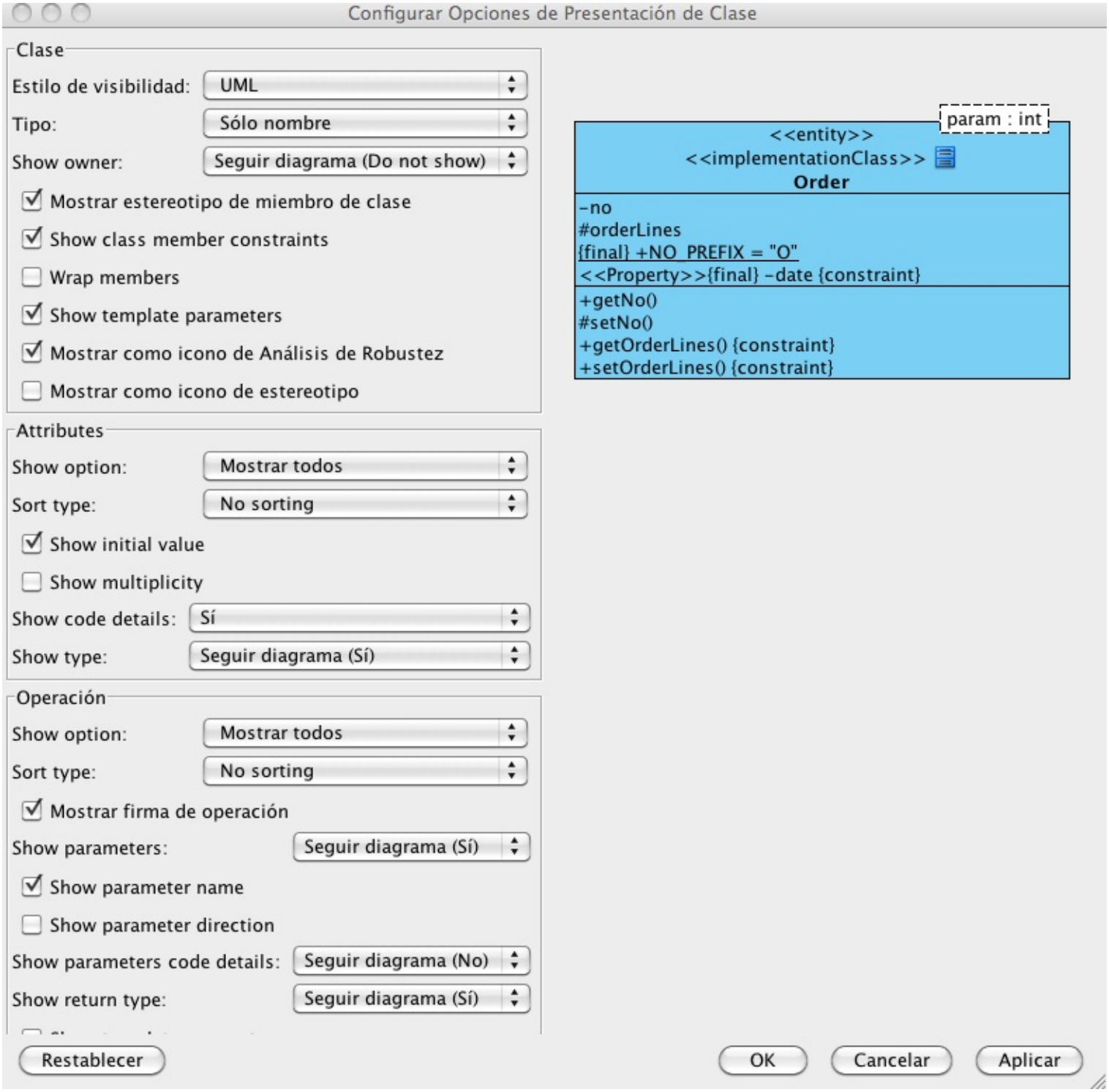
- Para que cuando creemos una relación entre dos clases el entorno de desarrollo genere los atributos de relación así como los métodos get y set:
  - *en la relación -> Abrir Especificación -> Role (poner un nombre al role) y desplegar*. Marcar las casillas de verificación "Provide property getter method" y "Provide property setter method".


Association Specification
General
Estereotipos
Valores Etiquetados
Nombre:
Visibilidad:
No especificado
Association End From
Role:
profesor
Element:
Profesor
Multiplicidad:
1
Navegable:
True
Association End To
Role:
alumno
Element:
Alumno
Multiplicidad:
1
Navegable:
True
Documentación:
HTML
Record...
Abstracto
Hoja
Derived
Restablecer
OK
Cancelar
Aplicar
Ayuda



- con esta opción, aunque en el diagrama de clases no aparezcan los métodos, cuando generamos el código a partir del diagrama nos permite indicar el tipo de objeto que queremos crear para la relación y genera los métodos en función de ese tipo. (p.e.: si indicamos que queremos que sea un arraylist, nos genera los métodos para añadir, eliminar, ...) (porque así lo hace de forma específica; carga los posibles tipos de datos existentes en cada lenguaje de programación).

- Para que en las clases me muestre todos los atributos y métodos (también los de las relaciones), tenemos una opción que es "Opciones de presentación", donde podemos configurar lo que queremos mostrar en el diagrama.



- Si queremos que en el diagrama aparezcan tanto los atributos de relación como los métodos get y set, podemos hacerlo nosotros.
  - *en la relación -> Abrir Especificación -> Role -> Referenced attribute -> Crear nuevo atributo*. En esta opción debemos configurar la multiplicidad, la visibilidad y la mavegabilidad.
  - una vez que tenemos creado el atributo, podemos crear los métodos get y set a partir de él.
  - con esta opción nos aparece todo en el diagrama, pero si generamos el código a partir del diagrama, lo que obtenemos siempre es un array normal de objetos (porque así lo hace de forma genérica; para todos los lenguajes de programación).

Resolución de las relaciones en programación

Si las relaciones tienen navegabilidad bidireccional, se resuelven de la siguiente manera:

- **Relación 1 a 1:** En cada una de las clases existe un atributo que será la referencia a un objeto de la otra clase.
- **Relación 1 a *:** En la clase de la parte 1 existe un atributo que será un array de referencias a objetos de la otra clase. En la clase de la parte * existe un atributo que será la referencia a un objeto de la otra clase.
- **Relación * a *:** En cada una de las clases existe un atributo que será un array de referencias a objetos de la otra clase.
- **Relación 1 a 0..*:** En la clase de la parte 1 existe un atributo que será un array de referencias a objetos de la otra clase. En la clase de la parte * existe un atributo que será la referencia a un objeto de la otra clase. En la parte 0..* no tendremos que controlar que exita la referencia al objeto (el valor del atributo podría ser null).
- **Relación de herencia:** En las subclases debemos poner la clausula de herencia hacia la clase superior. (*p.e.: en java: public class Cliente extends Persona*)
- **Relación de agregación:** En la clase superior existen atributos referenciando a objetos de las subclases y en las subclases existe un atributo referenciando a la clase superior. El tipo de los atributos depende de la multiplicidad de la relación.
- **Relación de composición:** En la clase superior existen atributos referenciando a objetos de las subclases y en las subclases existe un atributo referenciando a la clase superior. El tipo de los atributos depende de la multiplicidad de la relación. Tendremos que controlar que exista por lo menos una referencia a un objeto de la subclase para poder crear objetos de la clase superior.

Ejercicios

1. Ejercicio Matrícula Universitaria

Obtener el diagrama de clases de un sistema que gestiona las matriculas de los estudiantes en una universidad. Una persona viene caracterizada por su DNI, nombre, dirección y estado civil, y ésta puede convertirse en estudiante al darse de alta como tal en la universidad. Como estudiante podrá matricularse de las asignaturas que se imparten en la universidad, que tendrán un código, un nombre, un profesor responsable y un curso asignado. Una vez matriculado, el estudiante podrá recibir una beca y en su nueva condición de becario tendrá asignado un nuevo código y se conocerá el importe de la misma; al finalizar el curso, la condición de becario se acabará. Una vez el estudiante se matricula, tanto si recibe beca como si no, deberá examinarse de las asignaturas en las que se encuentra matriculado hasta que finalice el curso y vuelva a matricularse de nuevo o bien deje la universidad y con ello deje de ser estudiante.

2. Ejercicio Clínica Veterinaria

Un veterinario tiene como pacientes animales y como clientes familias. Un cliente es un conjunto de personas que suele corresponderse con una familia. Cada cliente tiene un código, el primer apellido del cabeza de familia, un número de cuenta bancaria, una dirección, un teléfono y los nombres y DNI de las personas correspondientes. No existe límite en el número de personas asociadas a una entidad cliente. Además, una persona puede estar dada de alta en varios clientes (p.e.: un hombre que vive con su esposa tiene un gato y como tal pertenece a un cliente, pero también está dado de alta en el cliente

asociado con el perro de sus padres). Los clientes pueden tener varias mascotas, cada mascota tiene un código, un alias, una especia, una raza, color de pelo, fecha de nacimiento aproximada, peso medio del animal en las últimas 10 visitas y el peso actual del animal. Asimismo, se guardará un historial médico con cada enfermedad que tuvo y la fecha en la que enfermó. Adicionalmente cada mascota tiene un calendario de vacunación, en el que se registrará la fecha de cada vacuna y la enfermedad de la que se vacuna.

3. Ejercicio Empresa

Representa mediante un diagrama de clases la siguiente especificación:

- Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes.
- Ambos se caracterizan por su nombre y edad.
- Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.
- De los clientes además se necesita conocer su teléfono de contacto.
- La aplicación necesita mostrar los datos de empleados y clientes.

4. Ejercicio Venta de Coches

Realizar el diagrama de clases correspondiente al siguiente sistema. Se trata de una empresa de venta de coches de segunda mano con las siguientes características:

- Los coches los suministran distintos proveedores, nos interesa conocer la marca, modelo, matrícula, precio de compra, de venta.
- Los coches pueden ser turismos, industriales y todo terrenos. Además pueden necesitar ser reparados, por lo que se debe tener un control de las reparaciones hechas, que pueden ser mecánicas, eléctricas o de chapa.
- En la empresa habrá dos tipos de vendedores: asalariados y por comisión. De los asalariados nos interesa saber también el salario y de los que van con comisión los coches que se han venido.
- Además se tendrá un control de los clientes tanto de los que han comprado un coche, como de los interesados en algún tipo de coche que podrán hacer reserva.
- Los coches pueden estar en distintas exposiciones, y debemos saber en todo momento dónde se encuentra cada coche.
- Se necesitan operaciones para realizar una venta de un coche, para reparar los coches que los necesiten, para comprar nuevos coches a los proveedores, etc.
- También interesa tener operaciones que nos devuelvan qué cliente compró un cierto coche, que se realicen listados de los coches que se encuentran en stock en un momento dado.

5. Ejercicio Biblioteca

Una biblioteca tiene copias de libros. Estos últimos se caracterizan por su nombre, tipo (novela, teatro, poesía, ensayo), editorial, año y autor.

- Los autores se caracterizan por su nombre, nacionalidad y fecha de nacimiento.
- Cada copia tiene un identificador, y puede estar en la biblioteca, prestada, con retraso o en

reparación.
- Los lectores pueden tener un máximo de 3 libros en préstamo.
- Cada libro se presta un máximo de 30 días, por cada día de retraso, se impone una “multa” de dos días sin posibilidad de coger un nuevo libro.
- Realiza un diagrama de clases y añade los métodos necesarios para realizar el prestamo y devolución de libros.

6. Ejercicio Alquiler automóviles

Se desea diseñar un diagrama de clases sobre la información de las reservas de una empresa dedicada al alquiler de automóviles, teniendo en cuenta que:

- Un determinado cliente puede tener en un momento dado hechas varias reservas.
- De cada cliente se desea almacenar su DNI, nombre, dirección y teléfono. Además dos clientes se diferencian por un código único.
- Cada cliente puede ser avalado por otro cliente de la empresa.
- Una reserva la realiza un único cliente pero puede involucrar varios coches.
- Es importante registrar la fecha de inicio y final de la reserva, el precio del alquiler de cada uno de los cohces, los litros de gasolina en el depósito en el momento de realizar la reserva, el precio total de la reserva y un indicador de si el coche o los coches han sido entregados.
- Todo coche tiene siempre asignado un determinado garaje que no puede cambiar. De cada coche se quiere la matrícula, el modelo, el color y la marca.
- Cada reserva se realiza en una determinada agencia.

7. Ejercicio Zoo

Un zoo necesita una aplicación informática para llevar su organización respecto a las especies que posee, los empleados (cuidadores y guías), y los distintos itinerarios de visita que ofrece. La información está estructurada de la siguiente manera:

- Especies: de las especies interesa saber el nombre en español, el nombre científico y una descripción general. Hay que tener en cuenta que una especie puede vivir en diferentes hábitats naturales y que un hábitat puede ser ocupado por diferentes especies. Las especies se encuentran en distintas zonas del parque de manera que cada especie está en una zona y en una zona hay varias especies.
- Hábitats: los diferentes hábitats naturales vienen definidos por el nombre, el clima y el tipo de vegetación predominantes, así como el continente o continentes en los que se encuentran.
- Zonas: las zonas del parque en las que se encuentran las distintas especies vienen definidas por el nombre y la extensión que ocupan.
- Itinerarios: los itinerarios discurren por ditintas zonas del parque. La información de interés para los itinerarios es: código de itinerario, la duración del recorrido, la longitud del itinerario, el máximo número de visitantes autorizado y el número de distintas especies que visita. Hay que tener en cuenta que un itinerario recorre distintas zonas del parque y que una zona puede ser recorrida por diferentes itinerarios.
- Guías: los guías del parque vienen definidos por el nombre, dirección, teléfono y fecha en la que comenzaron a trabajar en el zoo. Interesa saber qué guías llevan qué itinerarios, teniendo en cuenta que un guía puede llevar varios itinerarios y que un itinerario puede ser asignado a

diferentes guías en diferentes horas, siendo éstas un dato de interés.

- Cuidadores: los cuidadores vienen definidos por el nombre, dirección, teléfono y fecha de ingreso en el parque. Hay que tener en cuenta que un cuidador puede estar a cargo de varias especies y que una especie puede ser atendida por varios cuidadores, siendo de interés la fecha en la que un cuidador se hace cargo de una especie.