5. Diseño orientado a objetos. Elaboración de diagramas estructurales

Autor	Xerach Casanova
Clase	Entornos de desarrollo
Fecha	@Feb 27, 2021 4:23 PM

1. Programación orientada a objetos

En la construcción de software, si queremos obtener un producto de calidad, es preciso realizar un proceso previo de análisis y especificación del proceso que vamos a seguir.

El enfoque estructurado

Los problemas se someten a un proceso de división en subproblemas reiteradamente hasta llegar a problemas elementales resueltos con una función. Todas las funciones resultantes se hilan y entretejen hasta formar una solución global. Este proceso está centrado en procedimientos que se codifican mediante funciones que actúan sobre estructuras de datos.

Enfoque orientado a objetos

Con este paradigma el proceso se centra en simular elementos de la realidad asociada al problema, de la forma más cercana posible. La abstracción que representa estos elementos se denomina objeto:

- Está formado por un conjunto de atributos, que son los datos que lo caracterizan.
- Realizan un conjunto de operaciones que definen su comportamiento. Estas actúan sobre sus atributos para modificar su estado. Cuando le decimos a un objeto que ejecute una operación determinada, se dice que se le pasa un mensaje.

Las aplicaciones orientadas a objetos se forman por un conjunto de objetos que interaccionan enviándose mensajes para producir resultados. Los objetos similares se abstraen en clases. Un objeto es una instancia de una clase.

Cuando se ejecuta un programa ocurren tres sucesos:

- Primero, los objetos se crean a medida que se necesitan.
- Segundo. Los mensajes se mueven de un objeto a otro, o del usuario a un objeto, a medida que el programa procesa información o responde a la entrada del usuario.

- Tercero, cuando los objetos ya no se necesitan se borran y liberan memoria.

1.1. Conceptos de orientación a objetos

- **Abstracción.** Permite capturar las características y comportamiento de un conjunto de objetos y darles una descripción formal. Es clave en el proceso de análisis y diseño orientado a objetos. Mediante ella, se puede armar el conjunto de clases para modelar la realidad.
- **Encapsulación.** Organiza los datos y métodos de una clase, evitando el acceso a datos por cualquier otro medio distinto a los definidos. El estado de los objetos solo debe ser modificado desde métodos de la propia clase. Este concepto y el principio de ocultación van dela mano.
- **Modularidad.** Permite subdividir una aplicación en partes más pequeñas, llamadas módulos. cada una de ellas debe ser lo más independiente posible, tanto de la aplicación en sí como de las partes restantes.
- **Principio de ocultación.** La implementación de la clase solo la conocen los responsables de su desarrollo, esta puede ser modificada para mejorar su algoritmo sin tener repercusión en el resto del programa. Principalmente se ocultan las propiedades del objeto y se reduce la propagación de efectos colaterales cuando se producen cambios.
- **Polimorfismo.** Se reúnen bajo el mismo nombre, comportamientos distintos. La selección de uno u otro depende del objeto que lo ejecute.
- **Herencia:** Relación entre objetos en los que unos utilizan las propiedades y comportamientos de otros, formando una jerarquía. Los objetos heredan propiedades y comportamiento de las clases a las que pertenecen.
- **Recolección de basura.** El entorno de objetos se encarga de destruir automáticamente objetos y desvincularlos de la memoria asociada.

1.2. Ventajas de la orientación a objetos

1. Permite desarrollar software en menos tiempo y con menos coste y de mayor calidad gracias a la reutilización de los módulos.
2. Consigue aumentar la calidad de los sistemas, haciéndolos más extensibles ya que es sencillo aumentar o modificar la funcionalidad de la aplicación modificando operaciones.

3. Facilidad para modificar y mantener gracias a la modularidad y encapsulación.
4. Facilita la adaptación al entorno y el cambio haciendo aplicaciones escalables. Se puede modificar la estructura y el comportamiento de los objetos sin cambiar la aplicación.

1.3. Clases, atributos y métodos

Una clase está formada por un conjunto de procedimientos y datos que resumen características similares a un conjunto de objetos. La clase tiene dos propósitos: definir abstracciones y favorecer modularidad.

Los elementos de una clase se denominan miembros y son:

- **Atributos.** Conjunto de características asociadas a una clase. Cuando se toman valores concretos dentro de su dominio, se define el estado del objeto. Se definen por su nombre y su tipo, pueden ser simples o compuestos como otra clase.
- **Protocolo:** Operaciones (métodos y mensajes) que manipulan el estado. Un método es el procedimiento o función que se invoca para actuar sobre un objeto. Un mensaje es el resultado de cierta acción efectuada sobre un objeto. Los métodos determinan como actúan los objetos cuando reciben un mensaje. El conjunto de mensajes a los cuales puede responder un objeto se le conoce como protocolo del objeto.

Los valores asignados a los atributos de un objeto concreto hacen a ese objeto ser único. La clase a la que pertenece define sus características generales y su comportamiento.

1.4. Visibilidad

El aislamiento protege los datos de que sean modificados por alguien que no tenga derecho a acceder a ellos. Se eliminan así efectos secundarios e interacciones.

1. **Interfaz:** captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.
2. **Implementación:** comprende como se representa la abstracción y los mecanismos que conducen al comportamiento deseado.

Existen distintos niveles de ocultación que se implementan en lo que se denomina visibilidad. Es una característica que define el tipo de acceso a atributos y métodos:

- **Público:** se puede acceder desde cualquier clase y cualquier parte del programa.
- **Privado:** solo pueden acceder desde operaciones de la clase.
- **Protegidos:** solo se puede acceder desde operaciones de la clase o derivadas en cualquier nivel.

A la hora de de definir la visibilidad se tiene en cuenta que:

- El estado debe ser privado en los atributos de una clase. Se deben modificar mediante métodos de la clase creados a tal efecto.
- Las operaciones que definen la funcionalidad de la clase deben ser públicas.
- Las operaciones que ayudan a implementar parte de la funcionalidad deben ser privadas siempre que no se utilicen desde clases derivadas o privadas, si se utilizan desde clases derivadas.

1.5. Objetos. Instanciación

Cada vez que construimos un objeto en un programa a partir de una clase, se crea una instancia de ella. Cada instancia en el sistema sirve como modelo de un objeto del contexto del problema relevante para su solución. Puede realizar un trabajo, informar y cambiar su estado, y comunicarse con otros objetos del sistema, sin revelar como se implementan estas características.

Un objeto se define por:

- Su estado: cada atributo definido tiene un valor concreto.
- Su comportamiento: definido por los métodos públicos de su clase.
- Tiempo de vida: intervalo de tiempo en el programa, que el objeto existe. Comienza su creación en la instanciación y finaliza cuando se destruye.

La encapsulación y ocultamiento aseguran que los datos del objeto están ocultos y que no se pueden modificar accidentalmente por funciones externas.

Hay un caso particular, llamada clase abstracta, que no puede ser instanciada. Se suele usar para definir métodos genéricos relacionados con el sistema que

no son traducidos a objetos concretos, o para definir interfaces de métodos, cuya implementación se realiza en clases derivadas.

Ejemplos de objetos:

1. Objetos físicos: aviones en un sistema de control de tráfico aéreo, casas, parques...
2. Elementos de interfaces gráficas de usuario: ventanas, menús, teclado, cuadros de diálogo...
3. Animales: vertebrados, invertebrados...
4. Tipos de datos definidos por el usuario: Datos complejos, puntos de un sistema de coordenadas...
5. Alimentos: carnes, frutas, verduras...

2. UML

Unified Modeling Language o Lenguaje Unificado de Modelado. Es un conjunto de herramientas que permite modelar, construir y documentar los elementos que forman un sistema de software orientado a objetos. Es el estándar de facto de la industria, concebido por los autores de los tres métodos más usados de la orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh

- OMT - Object Modeling Tecnique (Rumbaugh et al.).
- Método - Booch (G. Booch)
- OOSE - Object-Oriented Software Engineering (I. Jacobson)

¿Por qué es útil modelar?

- Se permite utilizar un lenguaje que facilita la comunicación entre el equipo de desarrollo.
- Con UML podemos documentar todos los artefactos de un proceso de desarrollo: requisitos, arquitectura, pruebas, versiones...). Por lo que se dispone documentación que trasciende al proyecto.
- Hay estructuras que trascienden lo representable en un lenguaje de programación, como las que hacen referencia a la arquitectura del sistema, utilizando estas tecnologías podemos incluso indicar que módulos de

software vamos a desarrollar y sus relaciones, o en qué nodos hardware se ejecutan en sistemas distribuidos.

- Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose modelos precisos, no ambiguos y completos.

2.1 Algunos conceptos UML

2.1.1. Notación

Es un conjunto de símbolos y técnicas para combinarlos, que en el contexto UML permiten crear diagramas normalizados. Estas representaciones posibilitan al analista o desarrollador describir el comportamiento del sistema (análisis) y detalles de una arquitectura (diseño) de forma ambigua.

Que una notación sea detallada no significa que todos sus aspectos deban ser utilizados en todas las ocasiones. Utilizar UML debe facilitar el desarrollo y entendimiento de todos los participantes en el proyecto, pero no complicarlo. Utilizar todos los diagramas al máximo nivel puede liar más que aclarar.

Las notaciones UML deben ser independientes del lenguaje de programación.

Un diagrama es una representación gráfica de una colección de elementos de modelado (modelo), a menuda dibujada en un grafo con vértices conectados por arcos (notación.

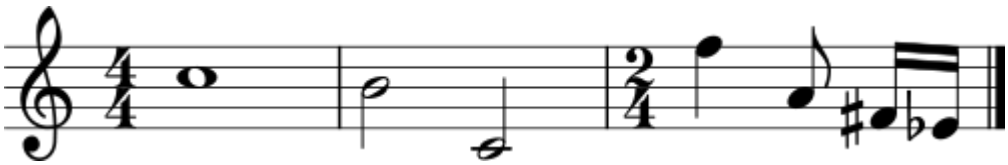
- **Notación musical -1.** La notación musical formal considera términos como pentagrama, nota redonda, blanca, negra, corchea, semicorchea, fusa, semifusa, clave de sol, fa, do...
- **Notación musical -2.** Otra alternativa útil para inexpertos es escribir la se cuencia de notas que forman la melodía: DO, RE, MI, FA, SOL, LA SI.

2.1.2. Modelos y herramientas

Un modelo captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema considerando un cierto propósito. El modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo y a un apropiado nivel de detalle.

Volviendo al ejemplo musical, la melodía se puede ver desde diferentes vistas:

- **Vista - 1.** Utilizando la primera notación del apartado anterior.



- **Vista -2** Usando la segunda notación del apartado anterior:

DO - SI - DO - FA - LA - FA# - MIB

- **Vista -3** La interpretación de la melodía puede ser una vista distinta, se pueden considerar vistas distintas en función del compás, instrumento...

En UML existen una serie de modelos o diagramas que nos proporcionan vistas en las fases de análisis y diseño.

Cada modelo es completo desde su punto de vista del sistema, pero existen relaciones de trazabilidad entre diferentes modelos.

Una herramienta de soporte automático de una notación, en nuestro ejemplo hablaríamos de un lápiz, un cuaderno musical, un programa de ordenador, un órgano...

Para el desarrollo de diagramas UML se usará la herramienta UMLet.

2.1.3. Métodos

Un método es un proceso disciplinado para generar uno o varios modelos que describen aspectos del sistema de software en desarrollo, utilizando alguna notación bien definida.

2.2. Tipos de diagrama UML

UML define un sistema como una colección de modelos que describen sus diferentes perspectivas. Los modelos se implementan en una serie de diagramas que contienen una colección de elementos de modelado, dibujando como un grafo conexo de arcos (relaciones y vértices (otros elementos del modelo).

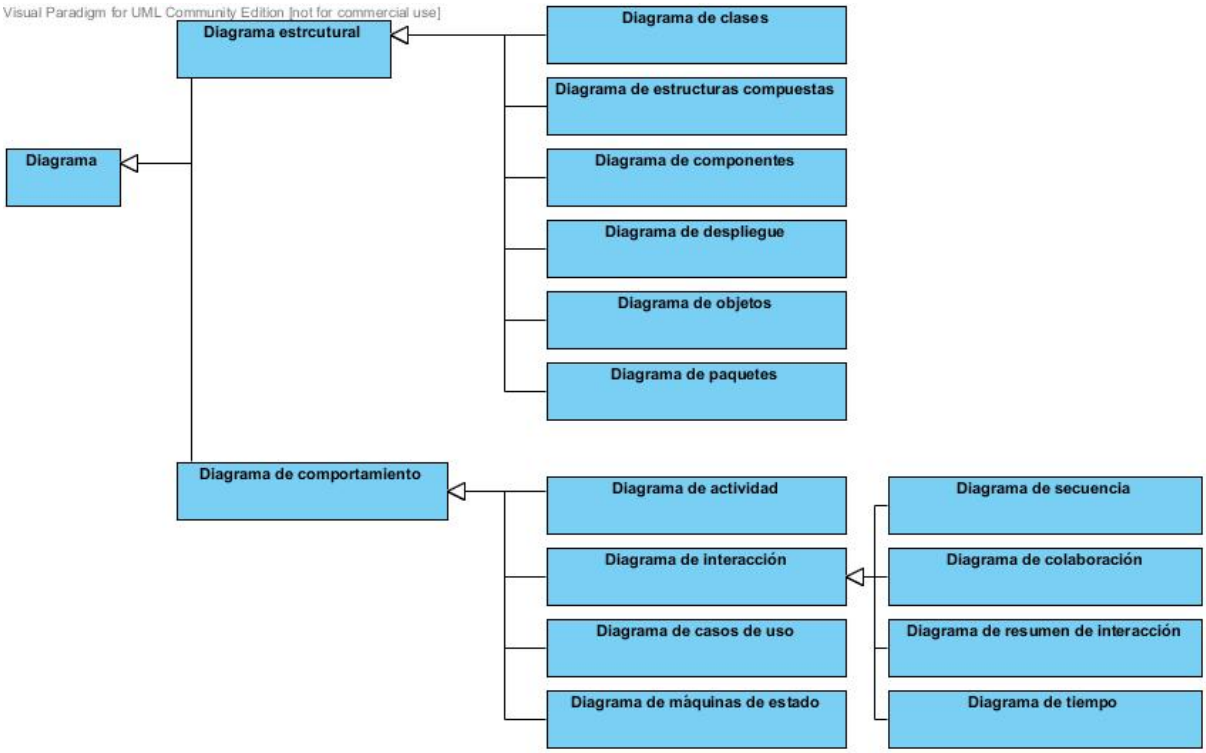
Se clasifican en

- **Diagramas estructurales.** Representan la visión estática del sistema. Especifican clases y objetos y como se distribuyen en el sistema físicamente.

- **Diagramas de clases.** Muestra los elementos del modelo estático abstracto, formado por un conjunto de clases y sus relaciones.
- **Diagramas de objetos.** Muestra los elementos del modelo estático en un momento concreto, habitualmente en casos especiales de un diagrama de clases o de comunicaciones. Está formado por el conjunto de objetos y sus relaciones.
- **Diagrama de componentes.** Especifica la configuración lógica de la implementación de una aplicación, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellas.
- **Diagrama de despliegue.** Representa la configuración del sistema en tiempo de ejecución, aparecen los nodos de procesamiento y sus componentes. Exhibe la ejecución de la arquitectura del sistema. Incluye nodos, ambientes operativos de hardware y software interfaces que las conectan.... Se utiliza en sistemas distribuidos.
- **Diagrama de estructuras compuestas.** Muestra la estructura interna de una clase, e incluye los puntos de interacción de esta con otras partes del sistema.
- **Diagrama de paquetes.** Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre ellos. Son útiles en sistemas de mediano o gran tamaño.

- **Diagramas de comportamiento**. muestran la conducta en tiempo de ejecución del sistema, tanto desde el punto de vista del sistema completo como de las instancias u objetos que lo integran.
  - **Diagrama de caso de uso.** Representa las acciones a realizar en el sistema desde el punto de vista de los usuario. Se representan las acciones, los usuarios y relaciones entre ellos. Especifican funcionalidad y comportamiento del sistema.
  - **Diagrama de estado de la máquina.** Describe el comportamiento de un sistema dirigido por eventos. Parecen los estados que puede tener un objeto o interacción.
  - **Diagrama de actividades.** Muestra el orden en el que se van realizando las tareas dentro del sistema. En él aparecen dos procesos de alto nivel de la organización. Incluye flujo de datos, o un modelo de la lógica compleja dentro del sistema.

- **Diagrama de secuencia.** Representa la ordenación temporal en el paso de mensajes. Modela la secuencia lógica a través del tiempo, de los mensajes de las instancias.
- **Diagrama de comunicación/colaboración.** Resalta la organización estructural de los objetos que se pasan mensajes. Ofrece las instancias de las clases, sus interrelaciones y el flujo de mensajes entre ellas. Comúnmente enfoca la organización estructural de los objetos que reciben y envían mensajes.
- **Diagrama de tiempos.** Muestra el cambio en un estado o una condición de una instancia o un rol a través del tiempo. Se usa normalmente para exhibir el cambio en el estado de un objeto en el tiempo, en respuesta a eventos externos.



2.3. Herramientas para la elaboración de diagramas UML

Las herramientas CASE facilitan en gran manera el desarrollo de diagramas UML, suelen contar con entorno de ventanas tipo wysiwyg, que permiten documentar los diagramas e integrarse con otros entornos de desarrollo, incluyendo la generación de código y procedimientos de ingeniería inversa.

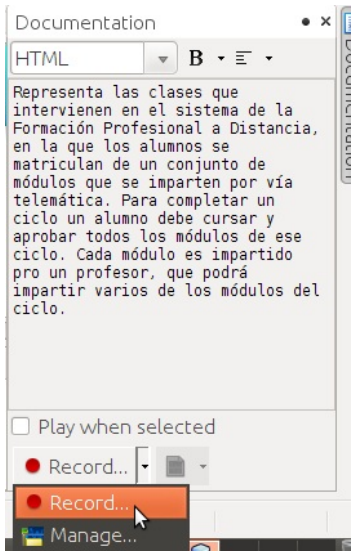
Entre otras podemos encontrar

- **Rational Systems Developer de IBM,** herramienta propietaria, permite desarrollo de proyectos software basados en UML. Creada en su origen por los creadores de UML y absorbida por IBM.
- **Visual Paradigm for UML (VP-UML).** Incluye versión para uso no comercial que se distribuye libremente al registrarse para obtener archivo de licencia (LGPL).
    - Incluye diferentes módulos para UML, diseñar bases de datos, realizar actividades de ingeniería inversa y diseñar con Agile.
    - Compatible con UML 2.0.
    - Admite generar informes PDF, HTML y otros.
    - Compatible con IDEs como Eclipse, Netbeans, Visual Studia ,net, IntellijDEA...
    - Multiplataforma
    - Disponible para Windows y Linux

2.3.1. Generación de la documentación

Se pueden hacer las anotaciones necesarias abriendo la especificación de cualquiera de los elementos, clases o relaciones, o bien del diagrama en sí mismo en la pestaña "Specification"

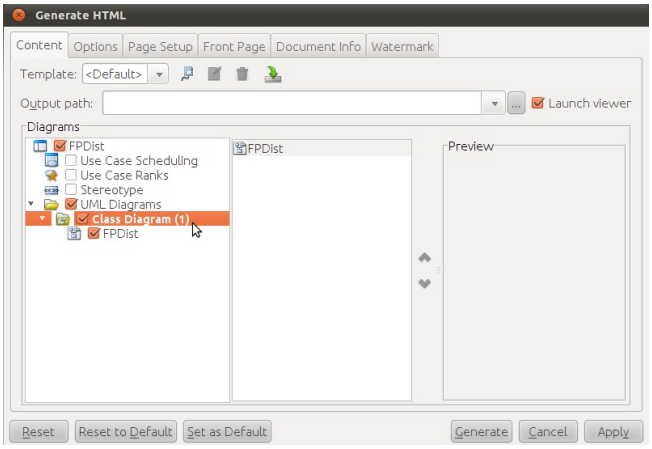
La ventana del editor cuenta con herramientas para formatear texto y añadir elementos como imágenes o hiperenlaces. También se puede grabar voz.



Generar informes.

Cuando los modelos están completos se puede generar un informe en varios formatos con la documentación escrita.

Desde VP-UML accedemos a tools/Reports/Report writer y seleccionamos el tipo de informe.



Desde SDE para NetBeans seleccionamos Modelin/reports/report writer.

En ambos, una vez elegido el tipo de informe se obtiene la siguiente ventana en la que se selecciona:

- Qué diagramas queremos que intervengan y donde se almacena el informe.
- La pestaña opciones permite configurar los elementos a añadir al informe: tablas de contenidos, títulos, etc.
- Propiedades de la página.
- Añadir marca de agua.

El resultado es un archivo html, pdf o doc en el directorio donde hayamos indicado con la documentación de los diagramas seleccionados.

2.3.2. - UMLet

Para los diagramas UML utilizaremos UMLet:

- Software gratuito
- Multiplataforma
- Incluye módulo de integración con eclipse
- Dispone de versión web que no requiere instalación: http://www.umletino.com/umletino.html

2.4. Ingeniería inversa

Se define como el proceso de analizar código, documentación y comportamiento de una aplicación para identificar sus componentes actuales y sus dependencias para extraer y crear una abstracción del sistema e

información del diseño. El sistema en estudio no es alterado, sino que se produce un conocimiento adicional del mismo.

Tiene como caso particular la reingeniería que es el proceso de extraer el código fuente de un archivo ejecutable.

Puede ser de varios tipos:

- **Ingeniería inversa de datos:** se aplica sobre algún código de bases de datos para obtener 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, o cualquier documento de diseño para obtener sus documentos de análisis o requisitos.
- **Ingeniería inversa de interfaces de usuario.** Consiste en estudiar la lógica interna de las interfaces para obtener los modelos y especificaciones que sirvieron para su construcción, con objeto de tomarlas como punto de partida en procesos de ingeniería directa que permitan su actualización.

3. Diagrama de clases

El diagrama de clases se considera el más importante de los existentes en UML. Representa los elementos estáticos del sistema, sus atributos y comportamiento, así como su relación entre ellos. Contiene las clases del dominio del problema y a partir de este se obtienen las clases que formarán el programa informático.

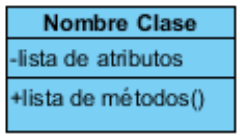
En él encontramos los siguientes elementos:

- **Clases:** agrupan conjuntos de objetos con características comunes, que llamamos atributos y su comportamiento que serán métodos. Atributos y métodos tienen una visibilidad que determina quien puede acceder a ellos.
- **Relaciones:** en el diagrama se representan las relaciones reales entre los distintos elementos a los que hacen referencia las clases. Pueden ser de asociación, agregación, composición y generalización.
- **Notas:** se representan como un cuadro donde podemos escribir comentarios que ayuden al entendimiento del diagrama.

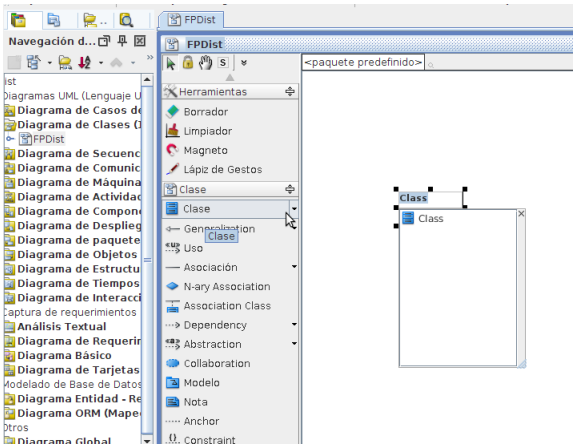
- **Elementos de agrupación:** se utilizan cuando hay que modelar un sistema grande. Las clases y sus relaciones se agrupan en paquetes que se relacionan entre sí.

3.1. Creación de clases

Se representa en el diagrama como un cuadro dividido en tres filas: nombre de la clase, atributos con su visibilidad y por último los métodos y su visibilidad.



Cuando generamos un diagrama nuevo aparece un panel con los elementos que podemos añadir al diagrama. Si hacemos clic sobre el icono que genera una clase nueva y a continuación sobre el lienzo aparecerá un cuadro para definir el nombre de la nueva clase.



Posteriormente, cuando la clase esté creada si hacemos clic con el botón secundario sobre la clase y seleccionamos "Abrir Especificación" podremos añadir atributos y métodos.

3.2. Atributos

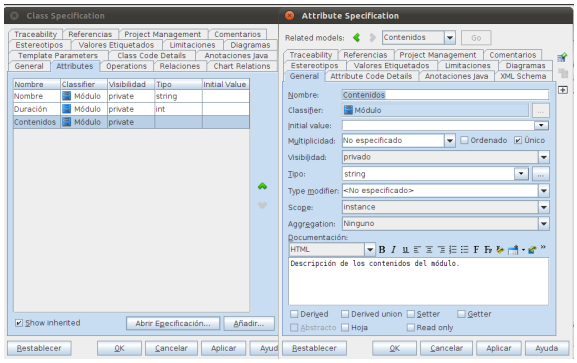
Forman la parte estática de la clase. Son un conjunto de variables para la que es preciso definir su nombre y su tipo, que puede ser simple o compuesto.

Se puede indicar su valor inicial o su visibilidad, que puede ser:

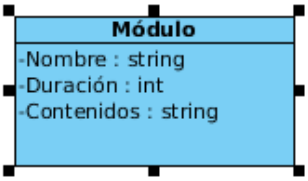
- **Público (+).** Se accede desde cualquier clase y cualquier parte del programa
- **Privado (-).** Solo se accede desde operaciones de la clase.
- **Protegido(#).** Se accede desde clases derivadas en cualquier nivel.

- **Paquete(~).** Se accede desde operaciones de las clases que pertenecen al mismo paquete.

Crear la clase como hemos visto en el punto anterior y modificar su nombre a "Módulo". Para añadir un atributo a una clase basta con seleccionar **Añadir atributo** del menú contextual y escribir su nombre.



Si queremos añadir más información podemos hacerlo desde la especificación de la clase en la pestaña Atributos, en la imagen vemos la especificación de una clase llamada Módulo, y de su atributo Contenidos para el que se ha establecido su tipo (string) y su descripción.



Por defecto la visibilidad de los atributos es privado y no se cambia a menos que sea necesario. Así queda la representación de la clase, los guiones al lado del atributo significan visibilidad privada.

Tenemos la posibilidad de añadir, desde el menú contextual de la clase, con el atributo seleccionado dos métodos llamados getter y setter que se utilizan para leer y establecer el valor del atributo cuando el atributo no es calculado, con la creación de estos métodos se contribuye al encapsulamiento y la ocultación de los atributos.

3.3. Métodos

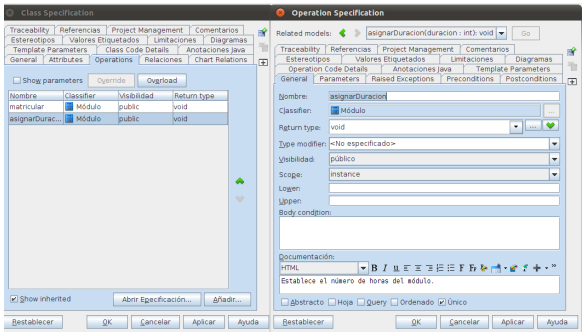
Representan la funcionalidad de la clase. Lo que puede hacer. Para definir un método se indica como mínimo su nombre, parámetros, tipo que devuelve y visibilidad. También se incluye descripción del método que aparece en la documentación que se genere del proyecto. Existen dos métodos particulares:

- **Constructor:** no devuelve ningún valor, tiene el mismo nombre de la clase y ejecuta acciones necesarias cuando se instancia un nuevo objeto.
- **Destructor:** Cuando no se va a utilizar más el objeto se puede utilizar el destructor que libere recursos del sistema que tenía asignado. Se utiliza de

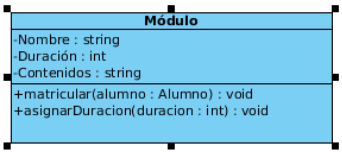
forma diferente según el lenguaje de programación.

El método más directo para crear un método es en el menú contextual seleccionar "Añadir operación" y escribir la signatura del método:

+nombre(<lista_parámetros>) : tipo_devuelto



También se puede añadir desde la especificación de la clase en la pestaña *Operations*.



En la imagen vemos la especificación de la clase y del método "asignarDuración" que asigna el número de horas del módulo.

El signo + en la signatura del método indica que es público. Así queda la clase en el diagrama:

3.4. Relaciones entre clases

Una relación es una conexión entre dos clases que incluimos en el diagrama.

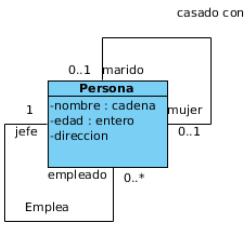
Se representan con una línea continua. Los mensajes navegan por las relaciones entre clases y objetos relacionados, normalmente en ambas direcciones, aunque a veces es necesario navegar en una sola dirección (punta de flecha).

Las relaciones se caracterizan por su cardinalidad, que representa cuantos objetos de una clase se pueden involucrar en la relación.

Creamos la clase como hemos visto en puntos anteriores. Para crear una relación utilizamos el elemento asociación de la paleta o bien el icono Association >> Class del menú contextual de la clase. Otra forma consiste en hacer clic sobre la clase Alumno, seleccionar Association >> Class y estirar la linea hasta la clase Módulo, aparecerá un recuadro para nombrar la relación.

Es posible establecer relaciones unarias de una clase consigo misma. En el ejemplo se ha rellenado

en la especificación de la relación los roles y la multiplicidad.

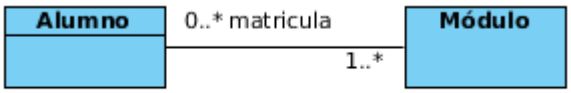


3.4.1. Cardinalidad o multiplicidad de la relación

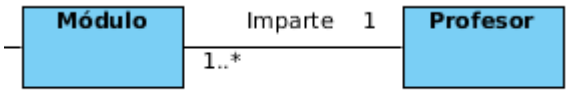
Representa cuantos objetos de una clase se relacionan con objetos de otra clase. En una relación hay una cardinalidad para cada extremo de la relación:

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)

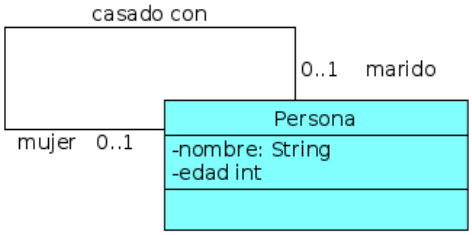
Por ejemplo, en la siguiente relación un alumno se relaciona con la clase módulo. En un módulo pueden matricularse varios alumnos pero puede no tener alumnos matriculados y un alumno puede estar matriculado en varios módulos pero mínimo debe estar matriculado en uno:



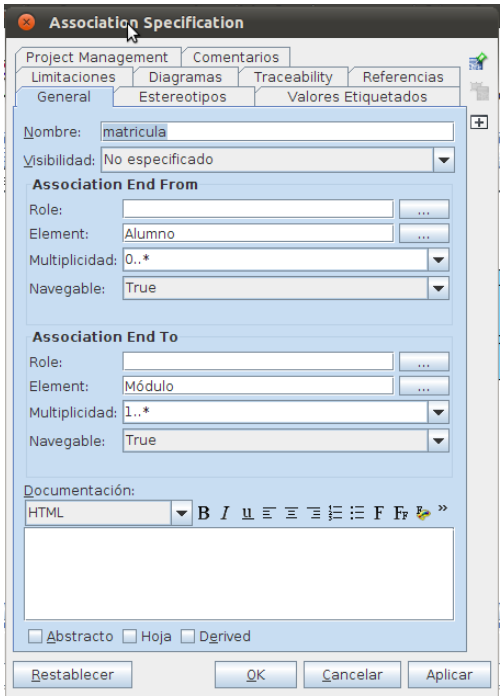
En esta otra quiere decir que la clase Profesor relaciona con la clase Módulo debido a que los profesores imparten diferentes módulos y un módulo es impartido por un profesor. La cardinalidad indicada quiere decir que todo profesor imparte al menos un módulo pudiendo impartir varios y, todo módulo es impartido por un profesor y sólo uno:



En este otro: Indica que una persona se relaciona con otra persona por la relación "casado con". La cardinalidad indica que una mujer puede estar soltera o casada con una persona y los maridos igual:



Si queremos establecer la cardinalidad abrimos la especificación de la relación y establecemos el apartado Multiplicidad a alguno de los valores que indica, si necesitamos utilizar algún valor concreto también podemos escribirlo nosotros mismos. En el caso que nos ocupa seleccionaremos la cardinalidad 0..* para los alumnos y 1..* para los módulos.



3.4.2. Relación de herencia (generalización)

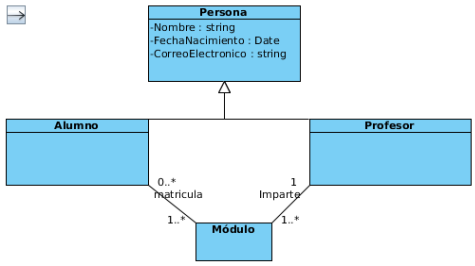
La herencia o generalización es una propiedad que permite a los objetos ser construidos a partir de otros objetos, utilizando estructuras de datos y métodos presentes en sus antepasados. Gracias a ello se puede reutilizar código realizado con anterioridad.

Consiste en una clase base y una jerarquía de clases que contiene las clases derivadas, que heredan código y datos de su clase base, además de añadir su propio código especial y datos, o incluso, cambiando elementos de la clase base que necesita que sean distintos.

- **Herencia simple:** Una clase solo tiene un ascendente.
- **Herencia múltiple:** se tienen más de un ascendente inmediato y se adquieren datos y métodos de más de una clase.

En el diagrama de clases se representa como una asociación en la que el extremo de la clase base tiene un triángulo.

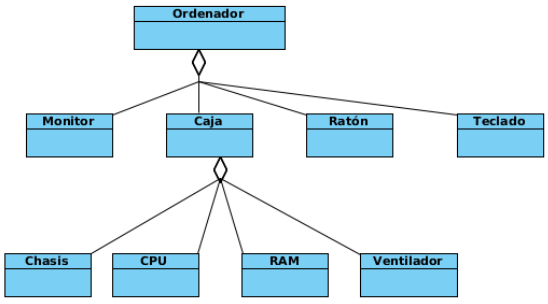
Podemos utilizar la relación de herencia para crear una clase nueva que se llame Persona y que recoja las características comunes de profesor y alumno. Persona será la **clase base** y Profesor y Alumno las **clases derivadas**.



Como los atributos Nombre, FechaNacimiento y correoElectronico se heredan de la clase base no hace falta que aparezcan en las clases derivadas, por lo que las hemos eliminado. Después podemos añadir atributos o métodos propios a las clases derivadas. La relación se añade de igual manera que una relación de asociación, pero seleccionando la opción Generalization.

3.4.3. Agregación y composición

Agregación es una asociación binaria que representa una relación todo-parte (pertenece a, tiene un, es parte de). Los elementos parte pueden existir sin el elemento contenedor y no son propiedad suya.



Por ejemplo. Un centro comercial tiene clientes o un equipo tiene unos miembros. El tiempo de vida de los objetos no tiene por qué coincidir.

En el siguiente caso, tenemos un ordenador que se compone de piezas sueltas que pueden ser sustituidas y que tienen entidad por si mismas, por lo que se

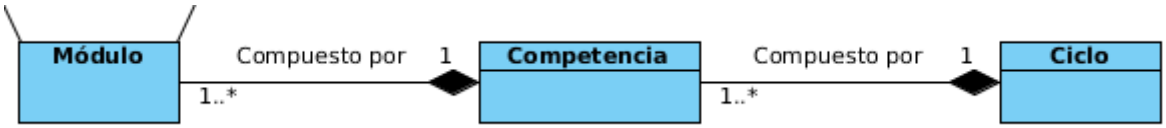
representa mediante relaciones de agregación. Utilizamos la agregación porque es posible que una caja, ratón o teclado o una memoria RAM existan con independencia de que pertenezcan a un ordenador o no.

Composición es una agregación fuerte en la que una instancia "parte" está relacionada como máximo con una instancia "todo" en un momento dado, cuando el objeto "todo" es eliminado, también son eliminados sus objetos "parte". Por ejemplo, un rectángulo tiene 4 vértices, un centro comercial está organizado mediante un conjunto de secciones de venta.

Para modelar la estructura de un ciclo formativo vamos a usar las clases Módulo, Competencia y Ciclo que representan lo que se puede estudiar en Formación Profesional y su estructura lógica. Un ciclo formativo se compone de una serie de competencias que se le acreditan cuando supera uno o varios módulos formativos.

Dado que si eliminamos el ciclo, las competencias no tienen sentido, y lo mismo ocurre con los módulos hemos usado relaciones de composición. Si los módulos o competencias pudieran seguir existiendo sin su contenedor habríamos utilizado relaciones de agregación.

Estas relaciones se representan con un rombo en el extremo de la entidad contenedora. En el caso de la agregación es de color blanco y para la composición negro. Como en toda relación hay que indicar la cardinalidad.



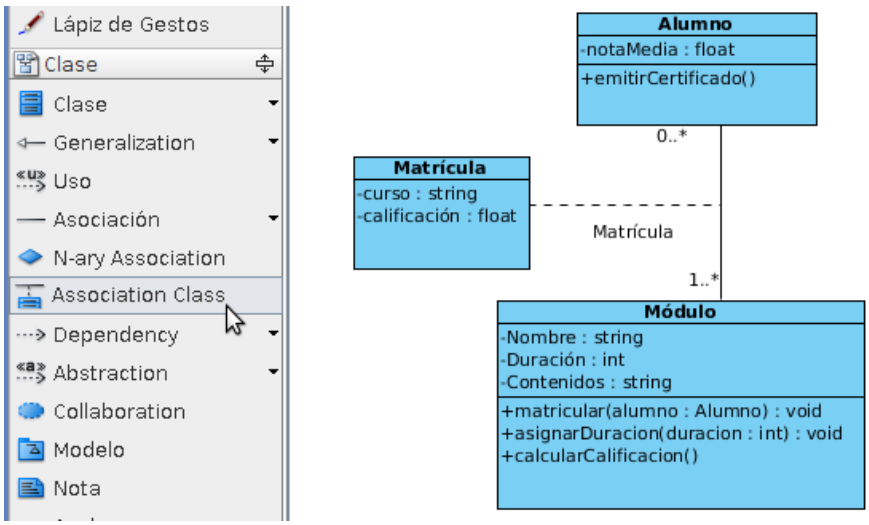
3.4.4. Atributos de enlace

Se pueden añadir atributos a relaciones para añadir algún tipo de información que la complete de alguna manera.

Cuando un alumno se matricula de un módulo es preciso especificar el curso al que pertenece la matrícula, las notas obtenidas en el examen y la tarea y la calificación final obtenida. Estas características no pertenecen totalmente al alumno ni al módulo sino a la relación específica que se crea entre ellos, que además será diferente si cambia el alumno o el módulo. Añade estos atributos al enlace entre Alumno y Módulo.

Para modelar esto en Visual Paradigm creamos una clase nueva (Matrícula) junto a Alumno y Módulo, y la unimos a la relación utilizando el icono de la

paleta "Association class", el diagrama queda así:



3.4.5. Restricciones

La relación entre dos clases puede estar condicionada al cumplimiento de algún requisito, o un parámetro de una clase tiene un valor constante.

Las restricciones se incluyen mediante una descripción textual encerrada entre llaves.

3.5. Pautas para crear diagramas de clase

La clave es hacer una buena elección de las clases que sugiere el programa.

Para identificar las clases candidatas a formar parte del diagrama es recomendable subrayar cada nombre o sintagma nominal que aparece en el enunciado.

Una vez teniendo la lista completa habrá que estudiar cada clase potencial para ver si se incluye en el diagrama. Para ello se utilizan los siguientes criterios y deben cumplirse todos o casi todos estos criterios:

1. La información de la clase es necesaria para que el sistema funcione.
2. La clase posee un conjunto de atributos que podemos encontrar en cualquier ocurrencia de sus objetos. Si solo aparece un atributo se suele rechazar y será añadido como atributo de otra clase.
3. La clase tiene un conjunto de operaciones identificables que pueden cambiar el valor de sus atributos y son comunes en sus objetos.

4. Es una entidad externa que consume o produce información esencial para la producción de cualquier solución en el sistema.

3.5.1. Obtención de atributos y operaciones

Atributos

Definen al objeto en el contexto del sistema, es decir, el mismo objeto en sistema diferentes tendría diferentes atributos. Debemos contestar a la pregunta: ¿Qué elementos compuesto o simples definen completamente en el contexto del problema actual.

Operaciones

Describen el comportamiento del objeto y modifican sus características:

- Manipulan los datos.
- Realizan algún cálculo.
- Monitorizan un objeto frente a la ocurrencia de un suceso de control.

Se obtienen analizando verbos en el enunciado del problema.

Relaciones

Se debe estudiar de nuevo el enunciado para obtener como los objetos descritos se relacionan entre sí. Para facilitar el trabajo podemos buscar mensajes que se pasen entre objetos y relaciones de composición y agregación. Las relaciones de herencia se suelen encontrar al comprar objetos semejantes entre sí y se contrasta que tengan atributos y métodos comunes.

3.6. Generación de código a partir del diagrama de clases

La generación automática de código consiste en la creación del código fuente de manera automatizada a través de herramientas CASE.

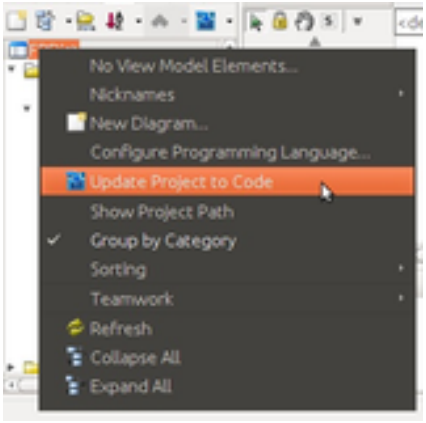
El proceso pasa por establecer correspondencia entre los elementos formales de los diagramas y la estructura de un lenguaje en concreto.

Utilizando el SDE integrado de VP-UML en NetBeans:

Se abre el modelo desde NetBeans, usando el SDE, se crea un proyecto nuevo y se importa el proyecto VP-UML que hemos creado.

Se hace de dos formas:

- **Sincronizar con el código:** el código fuente eliminado no se recupera, solo se actualiza el existente.



- **Forzar sincronizado a código.** Se actualiza todo el código, incluido el de código eliminado del proyecto NetBeans

Para generar todas las clases y paquetes, abrimos el proyecto CE-NB y desplegamos el menú contextual y seleccionamos Update Projecto to Code.

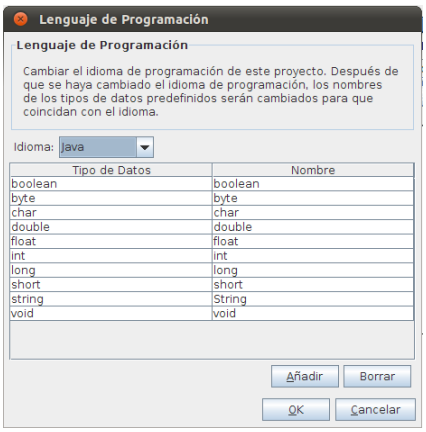
Si se produce algún problema se muestra en la ventana de mensaje. Este procedimiento produce archivos .java necesarios para implementar las clases del diagrama.

Desde VP-UML

Para generar el código java de un diagrama de clases utilizamos el menú herramientas/generación instantánea/java. Se muestra la ventana en la que configuramos el idioma, las clases y otras características básicas con la nomenclatura de atributos y métodos.

3.6.1. Elección del lenguaje de programació. Orientaciones para el lenguaje java.

El lenguaje final de implementación de la aplicación influyen en algunas decisiones a tomar cuando estamos creando el diagrama ya que el proceso de traducción es inmediato. Por ejemplo, si queremos utilizar la herramienta de generación de código tendremos que asegurarnos de utilizar tipos de datos simples apropiados, es decir, si usamos Java el tipo de dato para las cadenas de caracteres será String en lugar de string o char*.



Podemos definir el lenguaje de programación final desde el menú Herramientas >> Configurar lenguaje de programación. Si seleccionamos Java automáticamente cambiará los nombres de los tipos de datos al lenguaje escogido.

Mapa conceptual