TEMA 5

Contenido

[DISEÑO ORIENTADO A OBJETOS. ELABORACIÓN DE DIAGRAMAS ESTRUCTURALES]

José Luis Comesaña Cabeza --- 2011/2012

Entornos de Desarrollo del curso de "*Desarrollo de Aplicaciones Web*"

Diseño orientado a objetos. Elaboración de diagramas estructurales.

Caso práctico
En la empresa siguen trabajando en diferentes aplicaciones con un nivel alto de complejidad, se desarrolla para diferentes plataformas, en entornos de ventanas, para la web, dispositivos móviles, etc. ***Ada*** *lleva un tiempo observando a su equipo, y a pesar de que ya han hablado de las diferentes fases de desarrollo del software, y que están descubriendo nuevos entornos de programación que han facilitado su trabajo enormemente, se ha dado cuenta de que todavía hay una asignatura pendiente, sus empleados no utilizan herramientas ni crean documentos en las fases previas del desarrollo de una aplicación, a pesar de ser algo tan importante como el resto de fases del proceso de elaboración de software. Tampoco construyen modelos que ayuden a hacerse una idea de cómo resultará el proyecto. Estos documentos y modelos son muy útiles para que todo el mundo se ponga de acuerdo en lo que hay que hacer, y cómo van a hacerlo.*
Como ***Ada*** *muy bien conoce, un proyecto de software tendrá éxito sólo si produce un software de calidad, consistente y sobre todo que satisfaga las necesidades de los usuarios que van a utilizar el producto resultante.*
Para desarrollar software de calidad duradera, hay que idear una sólida base arquitectónica que sea flexible al cambio.
Incluso para producir software de sistemas pequeños sería bueno hacer análisis y modelado ya que redunda en la calidad, pero lo que sí es cierto, es que cuanto más grande y complejos son los sistemas más importante es hacer un buen modelado ya que nos ayudará a entender el comportamiento del sistema en su totalidad. Y cuando se trata de sistemas complejos el modelado nos dará una idea de los recursos necesarios (tanto humanos como materiales) para abordar el proyecto. También nos dará una visión más amplia de cómo abordar el problema para darle la mejor solución.
Ada *se da cuenta de que el equipo necesita conocer procedimientos de análisis y diseño de software, así como alguna herramienta que permita generar los modelos y la documentación asociada, así que decide reunir a su equipo para empezar a tratar este tema...*

1.- Introducción a la orientación a objetos.

Caso práctico
Ya en la sala de reuniones...
—Deberíamos empezar por revisar cual es la situación actual. Como ya sabéis existen diferentes lenguajes de programación que se comportan de manera diferente, y esto determina en gran medida el enfoque que se le da al análisis previo. No es lo mismo un lenguaje estructurado que uno orientado a objetos. Tendríamos que conocer las características de ambos enfoques para entender un poco mejor cómo se analizan.
—Es cierto —contesta ***Juan*** *—desde que empecé en el mundo de la informática esto ha cambiado un poco, así que he tenido que ir investigando para adaptarme a los nuevos lenguajes de programación, si queréis, os pongo al día brevemente, ...*

La construcción de software es un proceso cuyo objetivo es dar solución a problemas utilizando una herramienta informática y tiene como resultado la construcción de un programa informático. Como en cualquier otra disciplina en la que se obtenga un producto final de cierta complejidad, si queremos obtener un producto de calidad, es preciso realizar un proceso previo de análisis y especificación del proceso que vamos a seguir, y de los resultados que pretendemos conseguir.

El enfoque estructurado.

Sin embargo, cómo se hace es algo que ha ido evolucionando con el tiempo, en un principio se tomaba el problema de partida y se iba sometiendo a un proceso de división en subproblemas más pequeños reiteradas veces, hasta que se llegaba a problemas elementales que se podía resolver

utilizando una función. Luego las funciones se hilaban y entretejían hasta formar una solución global al problema de partida. Era, pues, un proceso centrado en los procedimientos, se codificaban mediante funciones (*conjunto de sentencias escritas en un lenguaje de programación que operan sobre un conjunto de parámetros y producen un resultado*) que actuaban sobre estructuras de datos (*conjunto de una colección de datos y de las funciones que modifican esos datos, que recrean una entidad con sentido, en el contexto de un problema*), por eso a este tipo de programación se le llama programación estructurada (*paradigma de programación que postula que una programa informático no es más que una sucesión de llamadas a funciones, bien sean del sistema o definidas por el usuario*). Sigue una filosofía en la que se intenta aproximar qué hay que hacer, para así resolver un problema.

Enfoque orientado a objetos.

La orientación a objetos ha roto con esta forma de hacer las cosas. Con este nuevo paradigma (*modelo o patrón aplicado a cualquier disciplina científica u otro contexto epistemológico*) el proceso se centra en simular los elementos de la realidad asociada al problema de la forma más cercana posible. La abstracción (*aislar un elemento de su contexto o del resto de elementos que le acompañan para disponer de ciertas características que necesitamos excluyendo las no pertinentes. Con ello capturamos algo en común entre las diferentes instancias con objeto de controlar la complejidad del software*) que permite representar estos elementos se denomina objeto, y tiene las siguientes características:

- ✓ Está formado por un conjunto de **atributos**, que son los datos que le caracterizan y
- ✓ Un conjunto de **operaciones** que definen su comportamiento. Las operaciones asociadas a un objeto actúan sobre sus atributos para modificar su estado. Cuando se indica a un objeto que ejecute una operación determinada se dice que se le pasa un **mensaje**.

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

Cuando se ejecuta un programa orientado a objetos 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 se libera la memoria.

Todo acerca del mundo de la orientación a objetos se encuentra en la página oficial del Grupo de gestión de objetos:
http://www.omg.org/index.htm

2.- Conceptos de Orientación a Objetos.

Caso práctico
*—De acuerdo, buen resumen **Juan**, sin embargo, los últimos proyectos que han entrado a la empresa se han desarrollo en su totalidad mediante software orientado a objetos, hemos usado PHP con Javascript, pero sobre todo Java, que es un lenguaje basado en objetos, así que sería necesario que analizáramos con un poco más de detenimiento el enfoque orientado a objetos, que características presenta, y qué ventajas tiene sobre otros.*
*—¡Gracias, **Ada**!, también tengo alguna información sobre eso...*

Como hemos visto la orientación a objetos trata de acercarse al contexto del problema lo más posible por medio de la simulación de los elementos que intervienen en su resolución y basa su desarrollo en los siguientes conceptos:

- ✓ **Abstracción**: Permite capturar las características y comportamientos similares de un conjunto de objetos con el objetivo de darles una descripción formal. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad, o el problema que se quiere atacar.
- ✓ **Encapsulación**: Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.
- ✓ **Modularidad**: Propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. En orientación a objetos es algo consustancial, ya que los objetos se pueden considerar los módulos más básicos del sistema.
- ✓ **Principio de ocultación**: Aísla las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas. Reduce la propagación de efectos colaterales cuando se producen cambios.
- ✓ **Polimorfismo**: Consiste en reunir bajo el mismo nombre comportamientos diferentes. La selección de uno u otro depende del objeto que lo ejecute.
- ✓ **Herencia**: Relación que se establece entre objetos en los que unos utilizan las propiedades y comportamientos de otros formando una jerarquía. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
- ✓ **Recolección de basura**: Técnica por la cual el entorno de objetos se encarga de destruir automáticamente los objetos, y por tanto desvincular su memoria asociada, que hayan quedado sin ninguna referencia a ellos.

2.1.- Ventajas de la orientación a objetos.

Caso práctico
*—Además la orientación a objetos cuenta con una serie de ventajas que nos vienen muy bien a los que nos dedicamos a la construcción de software, sobre todo porque nos facilitan su construcción y mantenimiento al dividir un problema en módulos claramente independientes y que, además, cuando ya tenemos suficientemente probados y completos podemos utilizar en otras aplicaciones, la verdad que ahorra bastante tiempo y esfuerzo... —argumenta **Juan**.*

Este paradigma tiene las siguientes **ventajas** con respecto a otros:

1. Permite desarrollar software en mucho menos tiempo, con menos coste y de mayor calidad gracias a la reutilización (*utilizar artefactos existentes durante la construcción de nuevo software. Esto aporta calidad y seguridad al proyecto, ya que el código reutilizado ya ha sido probado*) porque al ser completamente modular facilita la creación de código reusable dando la posibilidad de reutilizar parte del código para el desarrollo de una aplicación similar.
2. Se consigue aumentar la calidad de los sistemas, haciéndolos más extensibles (*principio de diseño en el desarrollo de sistemas informáticos que tiene en cuenta el futuro crecimiento del sistema. Mide la capacidad de extender un sistema y*

el esfuerzo necesario para conseguirlo) ya que es muy sencillo aumentar o modificar la funcionalidad de la aplicación modificando las operaciones.

3. El software orientado a objetos es más **fácil de modificar y mantener** porque se basa en criterios de modularidad y encapsulación en el que el sistema se descompone en objetos con unas responsabilidades claramente especificadas e independientes del resto.
4. La tecnología de objetos facilita la adaptación al entorno y el cambio haciendo aplicaciones escalables (*propiedad deseable de un sistema, red o proceso que le permite hacerse más grande sin rehacer su diseño y sin disminuir su rendimiento*). Es sencillo modificar la estructura y el comportamiento de los objetos sin tener que cambiar la aplicación.

¿Cuál es la afirmación más adecuada al paradigma de orientación a objetos?

- ☐ Permite crear aplicaciones basadas en módulos de software que representan objetos del entorno del sistema, por lo que no son apropiados para dar solución a otros problemas.
- ☐ Tiene como objetivo la creación de aplicaciones basadas en abstracciones de datos estáticas y de difícil ampliación
- ☐ Permite crear aplicaciones cuyo mantenimiento es complicado porque las modificaciones influyen a todos los objetos del sistema
- ◉ **Permite crear aplicaciones basadas en módulos que pueden reutilizarse, de fácil modificación y que permiten su ampliación en función del crecimiento del sistema**

2.2.- Clases, atributos y métodos.

Caso práctico

—De acuerdo, ahora conocemos las características básicas y ventajas de usar la orientación a objetos, ¿qué más nos haría falta? ¿Quizá sus estructuras básicas?

*—Yo puedo contaros algo sobre eso, —comenta **Juan**— lo estudié en el Ciclo Formativo.*

Los objetos de un sistema se abstraen, en función de sus características comunes, en clases. Una clase está formada por un conjunto de procedimientos y datos que resumen características similares de un conjunto de objetos. La clase tiene dos propósitos: definir **abstracciones** y favorecer la **modularidad**.

Una clase se describe por un conjunto de elementos que se denominan **miembros** y que son:

- ✓ **Nombre**.
- ✓ **Atributos**: conjunto de características asociadas a una clase. Pueden verse como una relación binaria entre una clase y cierto dominio formado por todos los posibles valores que puede tomar cada atributo. Cuando toman valores concretos dentro de su dominio definen el estado del objeto. Se definen por su nombre y su tipo, que puede ser simple o compuesto como otra clase.
- ✓ **Protocolo**: Operaciones (métodos, 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 por un objeto. Los métodos determinan como actúan los objetos cuando reciben un mensaje, es decir, cuando se requiere que el objeto realice una acción descrita en un método se le envía un mensaje. El conjunto de mensajes a los cuales puede responder un objeto se le conoce como *protocolo del objeto*.

Por ejemplo, si tenemos un objeto icono, tendrá como atributos el tamaño, o la imagen que muestra, y su protocolo puede constar de mensajes invocados por el clic del botón de un ratón cuando el usuario pulsa sobre el icono. De esta forma los mensajes son el único conducto que conectan al objeto con el mundo exterior.

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

Un objeto es una concreción de una clase, es decir, en un objeto se concretan valores para los atributos definidos en la clase, y además, estos valores podrán modificarse a través del paso de mensajes al objeto.

- [x] **Verdadero**
- [ ] Falso

2.3.- Visibilidad.

Caso práctico

—Pues creo que ya lo tenemos todo...

*—No creas, —dice **Ada** que siempre sabe algo más, que el resto desconoce— en orientación a objetos, existe un concepto muy importante, que es el de visibilidad, permite definir hasta qué punto son accesibles los atributos y métodos de una clase, por regla general, cuando definimos atributos los ocultamos, para que nadie pueda modificar el estado del objeto, y dejamos los métodos abiertos, porque son los que permiten el paso de mensajes entre objetos...*

El principio de ocultación es una propiedad de la orientación a objetos que consiste en aislar el estado de manera que sólo se puede cambiar mediante las operaciones definidas en una clase. Este aislamiento protege a los datos de que sean modificados por alguien que no tenga derecho a acceder a ellos, eliminando efectos secundarios e interacciones. Da lugar a que las clases se dividan en dos partes:

1. **Interfaz**: captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.
2. **Implementación**: comprende cómo se representa la abstracción, así como 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 que se permite a atributos y métodos y que podemos establecer como:

- ✓ **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
- ✓ **Privado**: Sólo se pueden acceder desde operaciones de la clase.
- ✓ **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas (*cuando se utiliza la herencia es la clase que hereda los atributos y métodos de la clase base*) en cualquier nivel.

Como norma general a la hora de definir la visibilidad tendremos en cuenta que:

- ✓ El estado debe ser privado. 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 (si no se utilizan desde clases derivadas) o protegidas (si se utilizan desde clases derivadas).

¿Desde dónde se puede acceder al estado de una clase?

- [ ] Desde cualquier zona de la aplicación.
- [ ] Desde la clase y sus clases derivadas.
- [x] **Solo desde los métodos de la clase.**

2.4.- Objetos. Instanciación.

Caso práctico
Antonio *ha asistido a esta reunión como parte de su formación laboral, pero se encuentra algo perdido entre tantos conceptos:*
—A ver, estamos todo el tiempo hablando de que las clases tienen atributos y métodos, luego, que los objetos se pasan mensajes, que son los que modifican los atributos, entonces, ¿no son lo mismo?, ¿qué diferencia hay?

Una clase es una abstracción que define las características comunes de un conjunto de objetos relevantes para el sistema.

Cada vez que se construye un objeto en un programa informático a partir de una clase se crea lo que se conoce como instancia (*objeto de una clase, creado en tiempo de ejecución con un estado concreto*) de esa clase. Cada instancia en el sistema sirve como modelo de un objeto del contexto del problema relevante para su solución, que puede realizar un trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema, sin revelar cómo se implementan estas características.

Un objeto se define por:

- ✓ **Su estado**: es la concreción de los atributos definidos en la clase a un valor concreto.
- ✓ **Su comportamiento**: definido por los métodos públicos de su clase.
- ✓ **Su tiempo de vida**: intervalo de tiempo a lo largo del programa en el que el objeto existe. Comienza con su creación a través del mecanismo de **instanciación** y finaliza cuando el objeto se destruye.

La encapsulación y el ocultamiento aseguran que los datos de un objeto están ocultos, con lo que no se pueden modificar accidentalmente por funciones externas al objeto.

Mientras que un objeto es una entidad que existe en el tiempo y el espacio, una clase representa sólo una abstracción, "la esencia" del objeto, si se puede decir así.
Grady Booch

Ejemplo de objetos:

- ✓ **Objetos físicos**: aviones en un sistema de control de tráfico aéreo, casas, parques.
- ✓ **Elementos de interfaces gráficas de usuario**: ventanas, menús, teclado, cuadros de diálogo.
- ✓ **Animales**: animales vertebrados, animales invertebrados.
- ✓ **Tipos de datos definidos por el usuario**: Datos complejos, Puntos de un sistema de coordenadas.
- ✓ **Alimentos**: carnes, frutas, verduras.

Existe un caso particular de clase, llamada **clase abstracta**, que, por sus características, no puede ser instanciada. Se suelen usar para definir métodos genéricos relacionados con el sistema que no serán traducidos a objetos concretos, o para definir métodos de base para clases derivadas.

3.- UML.

Caso práctico
Ahora que el equipo conoce los fundamentos de la orientación a objetos llega el momento de ver cómo pueden poner en práctica los conocimientos adquiridos.
***Ada** está interesada, sobre todo, en que sean capaces de representar las clases de los proyectos que están desarrollando y como se relacionan entre ellas. Para ello decide comenzar comentando las características de un lenguaje de modelado de sistemas orientados a objetos llamado UML. Este lenguaje permite construir una serie de modelos, a través de diagramas de diferentes visiones de un proyecto.*
—Es importante apreciar como estos modelos, nos van a permitir poner nuestras ideas en común utilizando un lenguaje específico, facilitarán la comunicación, que como sabéis, es algo esencial para que nuestro trabajo en la empresa sea de calidad.

Una empresa de software con éxito es aquella que produce de manera consistente software de calidad que satisface las necesidades de los usuarios. El modelado es la parte esencial de todas las actividades que conducen a la producción de software de calidad.

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 software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh, de hecho las raíces técnicas de UML son:

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

UML permite a los desarrolladores y desarrolladoras visualizar el producto de su trabajo en esquemas o diagramas estandarizados denominados modelos (*representación gráfica o esquemática de una realidad, sirve para organizar y comunicar de forma clara los elementos que involucran un todo. Esquema teórico de un sistema o de una realidad compleja que se elabora para facilitar su comprensión y el estudio de su comportamiento*) que representan el sistema desde diferentes perspectivas.

¿Porqué es útil modelar?

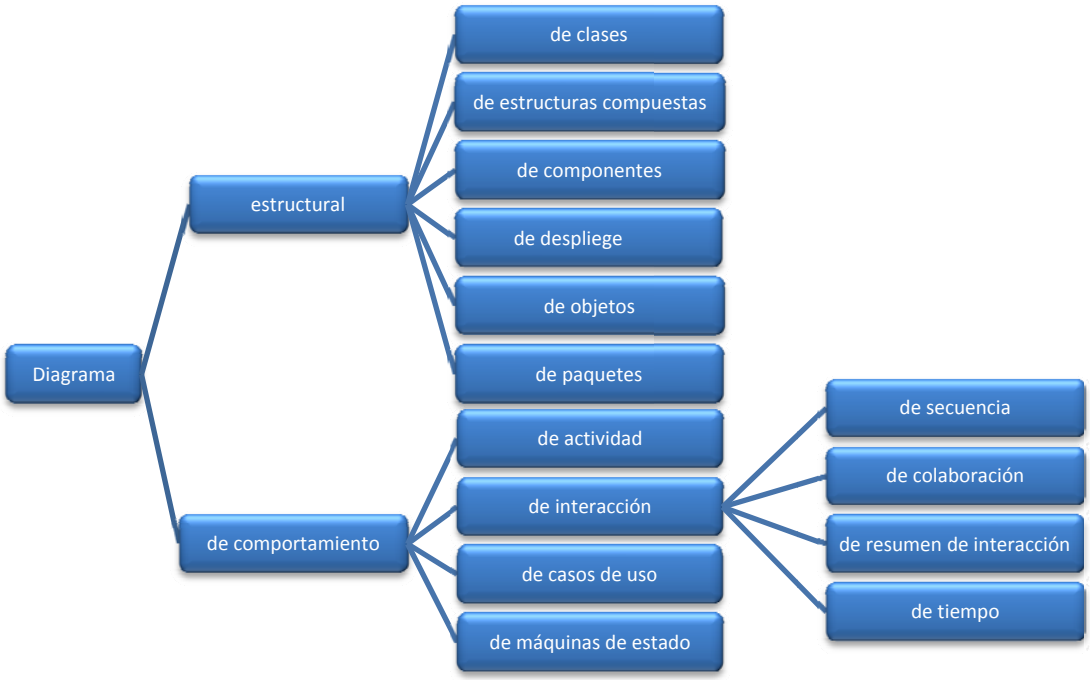
- ✓ Porque permite utilizar un lenguaje común que facilita la comunicación entre el equipo de desarrollo.
- ✓ Con UML podemos documentar todos los artefactos (*información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software*) de un proceso de desarrollo (requisitos *(condiciones que debe cumplir un proyecto software. Suelen venir definidos por el cliente. Permiten definir los objetivos que debe cumplir un proyecto software)*, arquitectura, pruebas, versiones,...) por lo que se dispone de 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 (*conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos. Junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores y el estilo arquitectónico que guía esta organización, estos elementos y sus interfaces, sus colaboraciones y su composición*), utilizando estas tecnologías podemos incluso indicar qué módulos de software vamos a desarrollar y sus relaciones, o en qué nodos hardware se ejecutarán cuando trabajamos con sistemas distribuidos.
- ✓ Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose modelos precisos, no ambiguos y completos.

Además UML puede conectarse a lenguajes de programación mediante ingeniería directa (*transformación de un modelo en código a través de su traducción a un determinado lenguaje de programación*) e inversa (*transformación del código en un modelo a través de su traducción desde un determinado lenguaje de programación*), como veremos.

3.1.- Tipos de diagramas UML.

Caso práctico
*Cuando **María** estudió el ciclo formativo no llegó a ver estas tecnologías con tanto detenimiento, así que está asimilándolo todo poco a poco:*
—De acuerdo, UML describe el sistema mediante una serie de modelos que ofrecen diferentes puntos de vista. Pero ¿qué tenemos que hacer para representar un modelo?, ¿en qué consiste exactamente?
—Utilizaremos diagramas, que son unos grafos en los que los nodos definen los elementos del diagrama, y los arcos las relaciones entre ellos.

UML define un sistema como una **colección de modelos** que describen sus diferentes perspectivas. Los modelos se implementan en una serie de diagramas que son 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).



Un diagrama UML se compone de cuatro tipos de 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.

y se clasifican en:

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

- **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. Dentro de este grupo están los diagramas de interacción.

En la imagen aparecen todos los diagramas organizados según su categoría. Se han destacado aquellos que pertenecen al estándar UML 2.0, más novedosos. En total se describen trece diagramas para modelar diferentes aspectos de un sistema, sin embargo no es necesario usarlos todos, dependerá del tipo de aplicación a generar y del sistema, es decir, se debe generar un diagrama sólo si es necesario.

Un 80% de las aplicaciones se pueden modelar con el 20% de los diagramas UML.

En el siguiente enlace tienes un documento con la descripción de los diagramas UML.
Diagramas UML

3.2.- Herramientas para la elaboración de diagramas UML.

Caso práctico
—Ahora que conocemos los diagramas que podemos generar para describir nuestro sistema, sería buena idea buscar alguna herramienta que nos ayude a elaborarlos. ¡No sería nada práctico andar todo el día con la libreta a cuestas!
—Lo que nos permite conocer a un buen desarrollador es que siempre hace un buen esquema inicial de cada proyecto, y eso puede hacerse en miles de soportes, desde una libreta a un servilleta, cualquier cosa que te permita hacer un pequeño dibujo, no obstante tienes razón. El uso de herramientas, además de facilitar la elaboración de los diagramas, tiene otras ventajas, como la integración en entornos de desarrollo, con lo que podremos generar el código base de nuestra aplicación desde el propio diagrama.
-¡Guau, eso sí es facilitar el trabajo!

La herramienta más simple que se puede utilizar para generar diagramas es lápiz y papel, hoy día, sin embargo, podemos acceder a herramientas CASE que facilitan en gran manera el desarrollo de los diagramas UML. Estas herramientas suelen contar con un entorno de ventanas tipo wysiwyg (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et), permiten documentar los diagramas e integrarse con otros entornos de desarrollo incluyendo la generación automática de código y procedimientos de ingeniería inversa.

Podemos encontrar, entre otras, las siguientes herramientas:

- **Rational Systems Developer de IBM**: Herramienta propietaria que permite el desarrollo de proyectos software basados en la metodología UML. Desarrollada en origen por los creadores de UML ha sido recientemente absorbida por IBM. Ofrece versiones de prueba, y software libre para el desarrollo de diagramas UML.

  Si sientes curiosidad puedes seguir este enlace a la página oficial de Rational Systems Developer:
  http://www-01.ibm.com/software/rational/

- Visual Paradigm for UML (VP-UML): Incluye una versión para uso no comercial que se distribuye libremente sin más que registrarse para obtener un archivo de licencia. Incluye diferentes módulos para realizar desarrollo UML, diseñar bases de datos, realizar actividades de ingeniería inversa y diseñar con Agile. Es compatible con los IDE de Eclipse, Visual Studio .net, IntellijDEA y NetBeans. Multiplataforma, incluye instaladores para Windows y Linux.
  Aquí tienes el enlace a la página oficial de Visual Paradigm.
  http://www.visual-paradigm.com/
- **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.

Aquí tienes el enlace a la página oficial de ArgoUML.

http://argouml.tigris.org/

3.2.1.- Visual Paradigm.

Para realizar el ejemplo de desarrollo de diagramas de clases que veremos a continuación se ha determinado usar la herramienta Visual Paradigm for UML por los siguientes motivos:

- ✓ Incluye una versión para uso no comercial, aunque se debe aclarar que viene con funcionalidad limitada, que se distribuye bajo licencia LGPL. Es posible solicitar una licencia de prueba para treinta días que utilizaremos cuando veamos la parte de ingeniería directa e inversa y generación de código.
- ✓ Es multiplataforma.
- ✓ Compatible con UML 2.0.
- ✓ Admite la generación de informes en formatos PDF, HTML y otros.
- ✓ Incluye un módulo para integrarse con NetBeans.
- ✓ Permite realizar actividades de ingeniería inversa y directa. Esto junto con la consideración anterior permite generar código en un proyecto NetBeans directamente a partir del diseño de clases, ahorrándonos trabajo.

En el siguiente enlace encontrarás información sobre dónde encontrar esta herramienta y su proceso de instalación.

Descarga e instalación de Visual Paradigm

Las herramienta CASE para la elaboración de diagramas UML sirven solo para la generación de los diagramas asociados al análisis y diseño de una aplicación.

- ○ Verdadero
- ◉ **Falso**

.

3.3.- Diagramas de clases.

Caso práctico
*En la empresa ya han instalado Visual Paradigm, **Juan** y **María** están empezando a investigar su funcionamiento, y como utilizarlo desde un proyecto de NetBeans.*
—Empecemos por los diagramas estructurales, entre ellos el más importante es el diagrama de clases, fíjate, representa la estructura estática del sistema y las relaciones entre las clases.

Dentro de los diagramas estructurales, y de todos en general, es el más importante porque representa los elementos estáticos del sistema, sus atributos y comportamientos, y como se relacionan entre ellos. Contiene las clases del dominio del problema, y a partir de éste se obtendrán las clases que formarán después el programa informático que dará solución al problema.

En un diagrama de clases podemos encontrar los siguientes elementos:

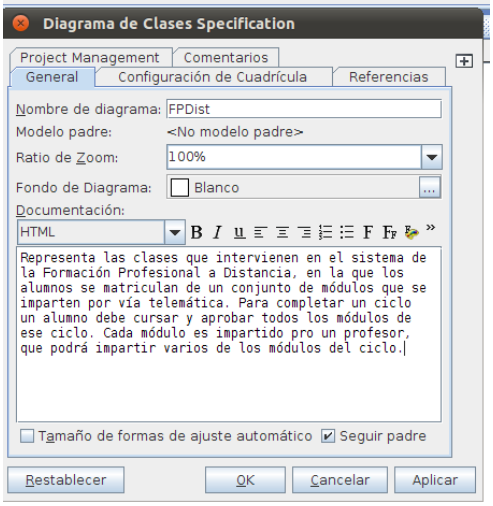
- ✓ **Clases**: recordemos que son abstracciones del dominio del sistema que representan elementos del mismo mediante una serie de características, que llamaremos atributos, y su comportamiento, que serán métodos. Los atributos y métodos tendrán una visibilidad que determinará quien puede acceder al atributo o método. Por ejemplo una clase puede representar a un coche, sus atributos serán la cilindrada, la potencia y la velocidad, y tendrá dos métodos, uno para acelerar, que subirá la velocidad, y otro para frenar que la bajará.

- ✓ **Relaciones**: en el diagrama representan relaciones reales entre los elementos del sistema a los que hacen referencia las clases. Pueden ser de asociación, agregación y herencia. Por ejemplo si tengo una clase persona, puedo establecer una relación conduce entre persona y coche.
- ✓ **Notas**: Se representan como un cuadro donde podemos escribir comentarios que nos ayuden a entender algún concepto que queramos representar.
- ✓ **Elementos de agrupación**: Se utilizan cuando hay que modelar un sistema grande, entonces las clases y sus relaciones se agrupan en paquetes, que a su vez se relacionan entre sí.

Ejercicio resuelto

Crear un diagrama de clases nuevo en Visual Paradigm UML que incluya su nombre y su descripción.

Para crear un diagrama de clases en VP-UML seleccionamos **Archivo >> Nuevo Diagrama** y seleccionamos Diagrama de clases. También podemos acceder al Navegador de diagramas, que se encuentra en el panel de la izquierda y en diagramas de clases hacer clic con el botón secundario y Seleccionar Nuevo diagrama de clases. Cuando generamos un diagrama nuevo tenemos que indicar su nombre y una descripción. Esto es importante para la generación de la documentación posterior.



Cuando creamos un diagrama nuevo aparece en blanco en el panel central de la aplicación. Si es necesario cambiar sus propiedades podemos hacerlo seleccionándolo en el Navegador de diagramas, a través de la opción "Abrir <nombre> Specification" del menú contextual. También podemos abrir el menú contextual, haciendo clic con el botón derecho del ratón sobre el panel central de la aplicación.

3.3.1.- Creación de clases.

Una clase se representa en el diagrama como un rectángulo divido en tres filas, arriba aparece el nombre de la clase, a continuación los atributos con su visibilidad y después los métodos con su visibilidad que está representada por el signo menos "–" para los atributos (privados) y por el signo más "+" para los métodos (públicos).

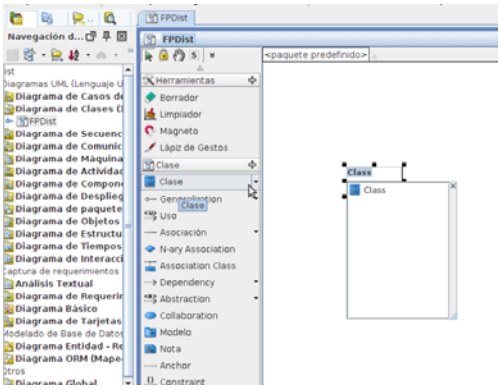
"Una clase es una descripción de un conjunto de objetos que manifiestan los mismos atributos, operaciones, relaciones y la misma semántica."
(Object Modelling and Design [Rumbaught et al., 1991])

"Una clase es un conjunto de objetos que comparten una estructura y un comportamiento comunes."
[Booch G., 1994]

Ejercicio resuelto

Crear una clase nueva en el diagrama de clases del punto anterior.

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.



Al crear una clase es obligatorio definir nombre, atributos y métodos.

- ☐ Verdadero
- ☒ **Falso**

3.3.2.- Atributos.

Forman la parte estática de la clase. Son un conjunto de variables para las que es preciso definir:

- ✓ Su nombre.
- ✓ Su tipo, puede ser un tipo simple, que coincidirá con el tipo de dato que se seleccione en el lenguaje de programación final a usar, o compuesto, pudiendo incluir otra clase.

Además se pueden indicar otros datos como un valor inicial o su visibilidad. La visibilidad de un atributo se puede definir como:

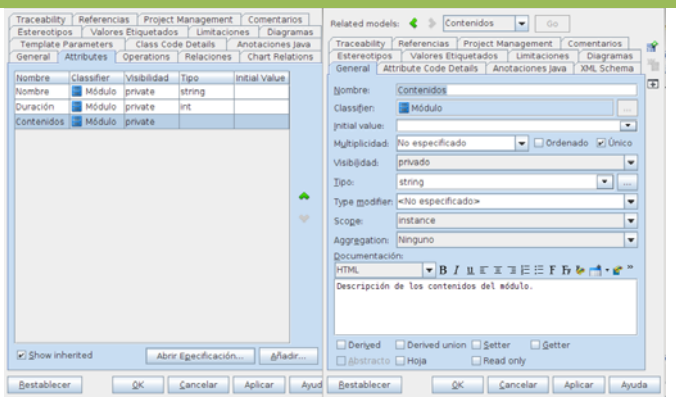
- ✓ **Público**: Se pueden acceder desde cualquier clase y cualquier parte del programa.
- ✓ **Privado**: Sólo se pueden acceder desde operaciones de la clase.
- ✓ **Protegido**: Sólo se pueden acceder desde operaciones de la clase o de clases derivadas en cualquier nivel.
- ✓ **Paquete**: Se puede acceder desde las operaciones de las clases que pertenecen al mismo paquete que la clase que estamos definiendo. Se usa cuando el lenguaje de implementación es Java.

Ejercicio resuelto

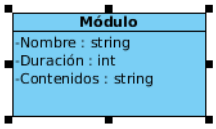
Crear una clase de nombre "Módulo" y que tenga tres atributos:

- ✓ **Nombre, de tipo string.**
- ✓ **Duración de tipo Int.**
- ✓ **Contenidos de tipo string.**

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.

¿Cómo sabemos que los atributos tienen visibilidad privada en el diagrama?

- ○ Porque aparecen acompañados del símbolo más "+"
- ○ Porque aparecen acompañados del símbolo almohadilla "#"
- ○ Porque aparecen acompañados del símbolo "~"
- ◉ **Porque aparece acompañado del símbolo menos "-"**

3.3.3.- Métodos.

Representan la funcionalidad de la clase, es decir, qué puede hacer. Para definir un método hay que indicar como mínimo su nombre, parámetros, el tipo que devuelve y su visibilidad. También se debe incluir una descripción del método que aparecerá en la documentación que se genere del proyecto.

Existen un caso particular de método, el **constructor** de la clase, que tiene como característica que no devuelve ningún valor. El constructor tiene el mismo nombre de la clase y se usa para ejecutar las acciones necesarias cuando se instancia un objeto de la clase. Cunado haya que destruir el objeto se podrá utilizar una función para ejecutar las operaciones necesarias cuando el objeto deje de existir, que dependerán del lenguaje que se utilice.

Ejercicio resuelto

Añadir a la clase creada anteriormente los métodos:

- ✓ matricular(alumno : Alumno) : void
- ✓ asignarDuración(duracion : int) : void

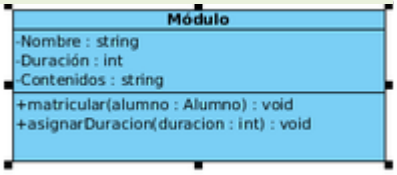
El método más directo para crear un método es en el menú contextual seleccionar "Añadir operación" y escribir la signatura del método:

```
+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:



¿Cuál es el método que no devuelve ningún tipo de dato?

- El constructor
- Todos los métodos devuelven algo, aunque sea void
- ~<nombre_clase>

3.4.- Relaciones entre clases.

Caso práctico

—Es fácil, ¿lo ves?, por ejemplo, para la aplicación de venta por Internet, tendríamos como clases socio, pedido o artículo, los socios se caracterizan por sus datos personales, los pedidos por su número, fecha, o localidad de destino y los artículos por el código o su descripción.
—Si eso lo veo claro, pero, ¿cómo lo ponemos todo junto? ¿Cómo se conecta el socio con el pedido y el artículo?

Una **relación** es una conexión entre dos clases que incluimos en el diagrama cuando aparece algún tipo de relación entre ellas en el dominio del problema.

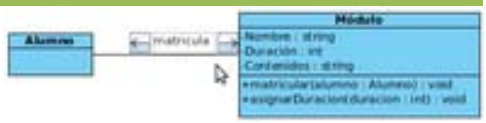
Se representan como una línea continua. Los mensajes "navegan" por las relaciones entre clases, es decir, los mensajes se envían entre objetos de clases relacionadas, normalmente en ambas direcciones, aunque a veces la definición del problema hace necesario que se navegue en una sola dirección, entonces la línea finaliza en punta de flecha.

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

- ✓ De herencia.
- ✓ De composición.
- ✓ De agregación.

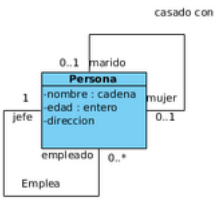
Ejercicio resuelto

Crea una clase nueva llamada Alumno y establece una relación de asociación con el nombre "matrícula" entre ésta y la clase Módulo.



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 línea 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.



Para obtener las relaciones de un diagrama nos basamos en la descripción de los requisitos del dominio, pero, ¿se pueden crear relaciones en el diagrama que no aparezcan especificadas en la lista de requisitos del problema?

No se puede, las relaciones se deben extraer de la descripción del problema, si no lo hiciéramos así nos estaríamos inventando información.

Sí se puede, a veces se infiere información o se conocen cosas del problema que no aparecen en la descripción de los requisitos.

3.4.1.- Cardinalidad o multiplicidad de la relación

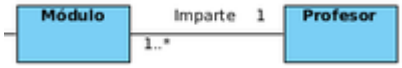
Un concepto muy importante es la **cardinalidad de una relación**, representa cuantos objetos de una clase se van a relacionar con objetos de otra clase. En una relación hay dos cardinalidades, una para cada extremo de la relación y pueden tener los siguientes valores:

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)

Por ejemplo, si tengo la siguiente relación:

quiere decir que los alumnos se matriculan en los módulos, en concreto, que un alumno se puede matricular en uno a más módulos y que un módulo puede tener ningún alumno, uno o varios.

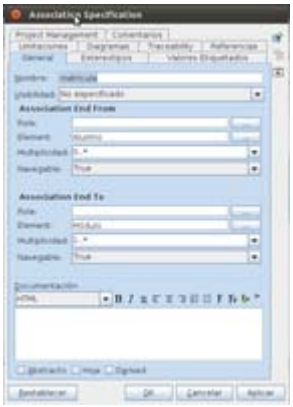
O esta otra:



en la que un profesor puede impartir uno o varios módulos, mientras que un módulo es impartido sólo por un profesor.

Ejercicio resuelto

Establece la cardinalidad de la relación que has creado en el punto anterior para indicar que un alumno debe estar matriculado en al menos un módulo, o varios y que para cada módulo se puede tener ningún alumno, uno o varios.



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.

La **herencia** es una propiedad que permite a los objetos ser construidos a partir de otros objetos, es decir, la capacidad de un objeto para utilizar estructuras de datos y métodos presentes en sus antepasados.

El objetivo principal de la herencia es la *reutilización*, poder utilizar código desarrollado con anterioridad. La herencia supone una clase base y una jerarquía de clases que contiene las clases

derivadas. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos, incluso cambiar aquellos elementos de la clase base que necesitan ser diferentes, es por esto que los atributos, métodos y relaciones de una clase se muestran en el nivel más alto de la jerarquía en el que son aplicables.

Tipos:

1. **Herencia simple**: Una clase puede tener sólo un ascendente. Es decir una subclase puede heredar datos y métodos de una única clase base.
2. **Herencia múltiple**: Una clase puede tener más de un ascendente inmediato, adquirir datos y métodos de más de una clase.

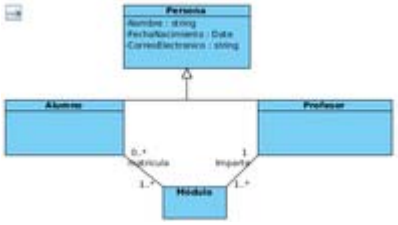
Representación:

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

Ejercicio resuelto

En nuestro diagrama tenemos Alumnos y Profesores. Aún no hemos hablado de su definición y estructura, pero en nuestro sistema tanto un alumno como un profesor tienen unas características comunes como el nombre, la fecha de nacimiento o el correo electrónico por el hecho de ser personas:

Transforma este diagrama para hacer uso de la herencia añadiendo una clase "Persona".



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.

He creado una clase persona cuyos atributo son Nombre, fechaContratación y numeroCuenta. De esta clase derivan por herencia la clase Empleado y JefeDepartamento. ¿Cómo debe declararse un método en la clase Persona que se llame CalculaAntigüedad que se usa sólo para calcular el sueldo de los empleados y jefes de departamento?

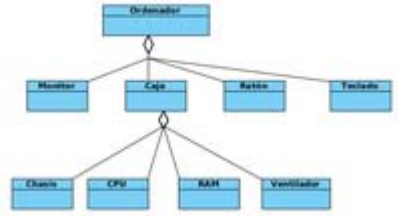
- ☐ Público
- ☐ Privada
- ◉ **Protegida**
- ☐ Paquete

3.4.3.- Agregación y composición.

Muchas veces una determinada entidad existe como un conjunto de otras entidades. En este tipo de relaciones un objeto componente se integra en un objeto compuesto. La orientación a objetos recoge este tipo de relaciones como dos conceptos: la agregación y la composición.

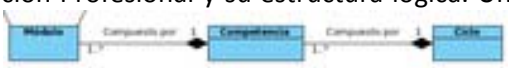
La **agregación** es una asociación binaria que representa una relación todo-parte (pertenece a, tiene un, es parte de). Los elementos parte pueden existir sin el elemento contenedor y no son propiedad suya. Por ejemplo, un centro comercial tiene clientes o un equipo tiene unos miembros. El tiempo de vida de los objetos no tiene porqué coincidir.

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



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'. Por ejemplo: un rectángulo tiene cuatro 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.

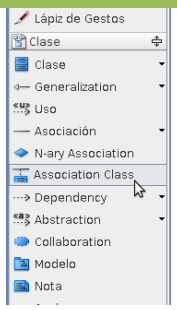
3.4.4.- Atributos de enlace.

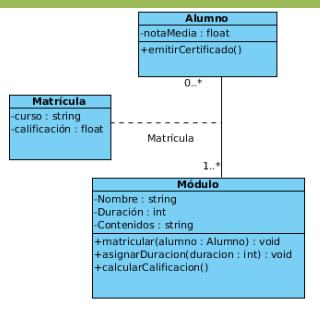
Es posible que tengamos alguna relación en la que sea necesario añadir algún tipo de información que la complete de alguna manera. Cuando esto ocurre podemos añadir atributos a la relación.

Ejercicio resuelto

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:





Siguiendo con el ejemplo anterior, para modelar el cálculo de la nota media de un alumno se añade el método *calcularNotaMedia* a la clase Alumno que realiza la media de las calificaciones de los módulos en los que el alumno se encuentra matriculado para este curso. ¿Qué visibilidad se debería poner a este método?

- (•) **Público**
- ( ) Privado
- ( ) Protegido
- ( ) Paquete

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

Caso práctico
María *y* ***Juan*** *siguen comentando la creación de diagramas de clases.*
—Las reservas se utilizan para relacionar los clientes y las habitaciones, eso es sencillo de ver, pero si tenemos un enunciado un poco más largo, puede no ser tan obvio. Quizá podrías darme algún consejo sobre cómo pasar de los requisitos iniciales de una aplicación a un primer diagrama de clases.
—Es verdad, la cosa se complica un poco cuando tenemos más requisitos, pero la clave está en analizar el texto para obtener nombres y continuar el desarrollo a partir de ahí.

Empezamos identificando objetos que serán las clases del diagrama examinando el planteamiento del problema. Los objetos se determinan subrayando cada nombre o cláusula nominal e introduciéndola en una tabla simple. Los sinónimos deben destacarse. Pero, ¿qué debemos buscar una vez que se han aislado todos los nombres? Buscamos sustantivos que puedan corresponder con las siguientes categorías:

- ✓ **Entidades externas** (por ejemplo: otros sistemas, dispositivos, personas) que producen o consumen información a usar por un sistema computacional.
- ✓ **Cosas** (por ejemplo: informes, presentaciones, cartas, señales) que son parte del dominio de información del problema.
- ✓ **Ocurrencias o sucesos** (por ejemplo: una transferencia de propiedad o la terminación de una serie de movimientos en un robot) que ocurren dentro del contexto de una operación del sistema.
- ✓ **Papeles o roles** (por ejemplo: director, ingeniero, vendedor) desempeñados por personas que interactúan con el sistema.
- ✓ Unidades organizacionales (por ejemplo: división, grupo, equipo) que son relevantes en una aplicación.
- ✓ **Lugares** (por ejemplo: planta de producción o muelle de carga) que establecen el contexto del problema y la función general del sistema.
- ✓ **Estructuras** (por ejemplo: sensores, vehículos de cuatro ruedas o computadoras) que definen una clase de objetos o, en casos extremos, clases relacionadas de objetos.

Cuando estemos realizando este proceso debemos estar pendientes de no incluir en la lista cosas que no sean objetos, como operaciones aplicadas a otro objeto, por ejemplo, "inversión de imagen" producirá un objeto en el ámbito del problema, pero en la implementación dará origen a un método. También es posible detectar dentro de los sustantivos **atributos** de objetos, cosa que también indicaremos en la tabla.

Cuando tengamos la lista completa habrá que estudiar cada objeto potencial para ver si, finalmente, es incluido en el diagrama. Para ayudarnos a decidir podemos utilizar los siguientes criterios:

1. La información del objeto es necesaria para que el sistema funcione.

2. El objeto posee un **conjunto de atributos** que podemos encontrar en cualquier ocurrencia del objeto. Si sólo aparece un atributo normalmente se rechazará y será añadido como atributo de otro objeto.
3. El objeto tiene un **conjunto de operaciones** identificables que pueden cambiar el valor de sus atributos y son comunes a cualquier ocurrencia del objeto.
4. Es una **entidad externa** que consume o produce información esencial para la producción de cualquier solución en el sistema.

El objeto se incluye si cumple todos (o casi todos) los criterios.

Se debe tener en cuenta que la lista no incluye todo, habrá que añadir objetos adicionales para completar el modelo y también, que diferentes descripciones del problema pueden provocar la toma de diferentes decisiones de creación de objetos y atributos.

3.5.1.- Obtención de atributos y operaciones.

Atributos

Definen al objeto en el contexto del sistema, es decir, el mismo objeto en sistemas diferentes tendría diferentes atributos, por lo que debemos buscar en el enunciado o en nuestro propio conocimiento, características que tengan sentido para el objeto en el contexto que se analiza. Deben contestar a la pregunta "*¿Qué elementos (compuestos y/o simples) definen completamente al objeto en el contexto del problema actual?*"

Operaciones

Describen el comportamiento del objeto y modifican sus características de alguna de estas formas:

- ✓ 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

Por último habrá que estudiar de nuevo el enunciado para obtener cómo los objetos que finalmente hemos descrito se relacionan entre sí. Para facilitar el trabajo podemos buscar mensajes que se pasen entre objetos y las relaciones de composición y agregación. Las relaciones de herencia se suelen encontrar al comparar objetos semejantes entre sí, y constatar que tengan atributos y métodos comunes.

Cuando se ha realizado este procedimiento no está todo el trabajo hecho, es necesario revisar el diagrama obtenido y ver si todo cumple con las especificaciones. No obstante siempre se puede refinar el diagrama completando aspectos del ámbito del problema que no aparezcan en la descripción recurriendo a entrevistas con los clientes o a nuestros conocimientos de la materia.

En el siguiente enlace tienes un ejemplo de obtención del diagrama de clases a partir de la descripción de un problema.

Generación del diagrama de clases de un problema dado.

El archivo al proyecto VP-UML resultante:

Enlace al proyecto de Visual Paradigm. (0.17 MB)

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

Caso práctico
—Bueno, ya tenemos el diagrama, es cierto que es bastante útil para aclarar ideas en el equipo y establecer un plan de trabajo inicial. Además es más fácil empezar a programar porque ya tenemos la línea a seguir, ahora solo falta que empecemos a crear clases y a rellenarlas, ¿Verdad?
—Pues sí, pero aún no lo sabes todo, el diagrama de clases aún te va a dar más facilidades...

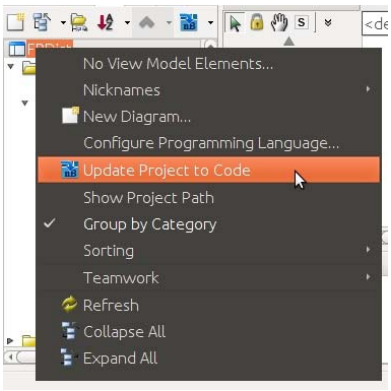
La Generación Automática de Código consiste en la creación utilizando herramientas CASE de código fuente de manera automatizada. El proceso pasa por establecer una correspondencia entre los elementos formales de los diagramas y las estructuras de un lenguaje de programación concreto. El diagrama de clases es un buen punto de partida porque permite una traducción bastante directa de las clases representadas gráficamente, a clases escritas en un lenguaje de programación específico como Java o C++.

Normalmente las herramientas de generación de diagramas UML incluyen la facilidad de la generación, o actualización automática de código fuente, a partir de los diagramas creados.

Ejercicio resuelto

Traduce el diagrama de clases generado en el punto anterior, tanto desde el SDE para NetBeans de Visual Paradigm como desde VP-UML.

Utilizando el SDE integrado de VP-UML en NetBeans:



Antes de hacerlo tendremos que abrir el modelo desde NetBeans, usando el SDE, crear un proyecto nuevo e importar el proyecto VP-UML que hemos creado.

Se puede hacer de dos formas:

- ✓ **Sincronizar con el código**: El código fuente eliminado no se recuperará. Solo se actualizará el código existente.
- ✓ **Forzar sincronizado a código**: Se actualizará todo el código que pueda partir del modelo, incluido el de código eliminado del proyecto NetBeans.

Para generar todas las clases y paquetes de un proyecto VP-UML en NetBeans abrimos el proyecto CE-NB y desplegamos el menú contextual y seleccionamos **Update Project to Code**. También existe la posibilidad de hacerlo directamente desde una clase en particular.

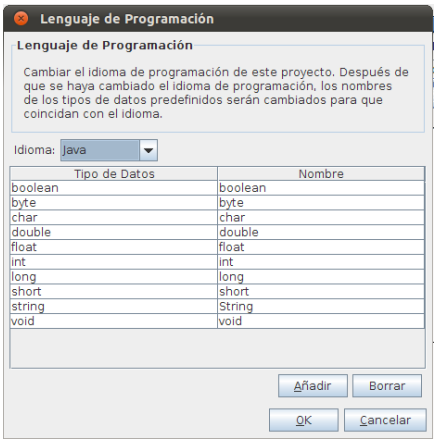
Si se produce algún problema se muestra en la ventana de mensajes, una vez corregido se vuelve a actualizar.

Este procedimiento produce los 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 una ventana en la que podemos configurar el idioma, las clases a generar, y otras características básicas relacionadas con la nomenclatura de atributos y métodos. También permite seleccionar la forma en que se va a implementar la asociación de composición, en nuestro caso hemos elegido la opción por defecto que es a través de un vector.

3.6.1.- Elección del lenguaje de programación. 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. Si existe algún problema en los nombres de clases, atributos o tipos de datos porque no puedan ser utilizados en el lenguaje final o no existan la generación dará un fallo y no se realizará. 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.

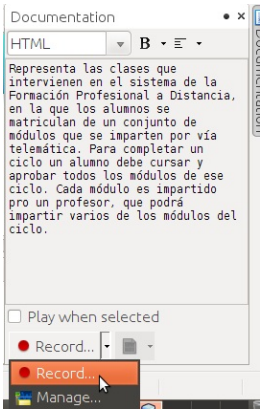
3.7.- Generación de la documentación.

Caso práctico

Los chicos siguen estudiando características de la herramienta VP-UML...

—Sería estupendo que después de generar un diagrama, en el que verdaderamente te has esforzado, pudieras hacer anotaciones sobre la importancia de cada clase, atributo o relación, para que, posteriormente, pudieras compartir esa información o recordarlo rápidamente cuando estuvieras programando el sistema en caso de tener que consultar algo.

—No solo eso, además de generar documentación exhaustiva, la herramienta permite crear informes con ella, pasándola a un formato más cómodo de leer e interpretar en papel por el equipo de desarrollo.

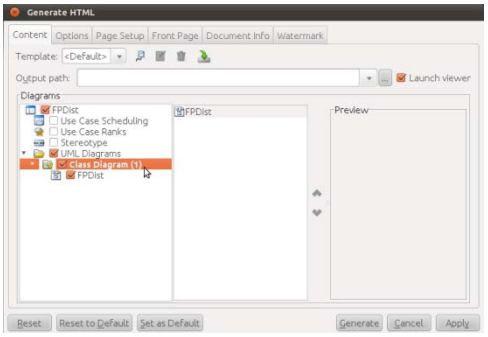


Como en todos los diagramas UML, podemos hacer las anotaciones que consideremos necesarias abriendo la especificación de cualquiera de los elementos, clases o relaciones, o bien del diagrama en sí mismo en la pestaña "Specification".

La ventana del editor cuenta con herramientas para formatear el texto y darle un aspecto bastante profesional, pudiendo añadir elementos como imágenes o hiperenlaces.

También se puede grabar un archivo de voz con la documentación del elemento usando el icono Grabar.

Generar informes



Cuando los modelos están completos podemos generar un informe en varios formatos diferentes (HTML, PDF o Word) con la documentación que hemos escrito. Para generar un informe hacemos:

Desde VP-UML accedemos a Tools >> Reports >> Report writer y seleccionamos el tipo de informe que queremos.

Desde el SDE para NetBeans seleccionamos Modelin >> Reports >> Report writer.

En ambos casos, una vez que elegimos el tipo de informe, obtendremos la siguiente ventana en la que seleccionamos entre otros:

- ✓ Qué diagramas queremos que intervengan y donde se almacenará el informe.
- ✓ La pestaña opciones (Options) permite configurar los elementos que se añadirán al informe, como tablas de contenidos, títulos, etc.
- ✓ Las propiedades de la página.
- ✓ Si se va a añadir una marca de agua.

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

4.- Ingeniería inversa.

Caso práctico

Ada *está muy satisfecha con el cambio que percibe en su equipo, ahora, cuando tiene que enfrentarse a un proyecto nuevo se esfuerzan en escribir los requerimientos antes y comenzar con un proceso de análisis y creación de un diagrama de clases. No obstante, ahora le surge un reto nuevo, una empresa les ha contratado para reconstruir una aplicación que hay que adaptar a nuevas necesidades. Así que los chicos continúan investigando si pueden aplicar el uso de diagramas en este caso también.*

La **ingeniería inversa** se define como el proceso de analizar código, documentación y comportamiento de una aplicación para identificar sus componentes actuales y sus dependencias y 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.

La ingeniería inversa puede ser de varios tipos:

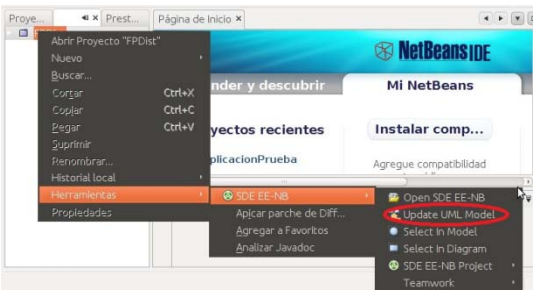
- ✓ **Ingeniería inversa de datos**: Se aplica sobre algún código de bases datos (aplicación, código SQL, etc.) para obtener los modelos relacionales o sobre el modelo relacional para obtener el diagrama entidad-relación.
- ✓ **Ingeniería inversa de lógica o de proceso**: Cuando la ingeniería inversa 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.

Ejercicio resuelto

A partir del código que has obtenido en el ejercicio anterior genera el diagrama de clases correspondiente.

Desde el **SDE de VP-UML para NetBeans** tendremos acceso a una herramienta que nos permite la transformación de código Java en diagramas de clases. Para ello:

1. Abrimos el SDE.
2. Seleccionamos el proyecto en el panel de proyectos y abrimos la herramienta SDE-CE NB.
3. Abrir un proyecto nuevo VP-UML en el proyecto de NetBeans.
4. Desde el nodo del proyecto seleccionamos en el menú contextual la opción "**Update UML Model**" lo que iniciará el proceso de ingeniería inversa.



Desde VP-UML

Haremos el proceso de ingeniería inversa desde **Herramienta >> Java Bidireccional >> Código de Inversión**. Al seleccionar esta opción nos preguntará donde se localiza el código fuente. El proceso no genera el diagrama exactamente igual que el original, es capaz de obtener las clases y las relaciones de herencia. El resto de relaciones tendremos que establecerlas nosotros a mano.

Tienes una buena descripción teórica sobre ingeniería inversa en el siguiente documento:

Ingeniería Inversa. (0.53 MB)

Anexo I.- Diagramas UML.

Diagramas estructurales.

- ✓ **Diagramas de clases**: Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones. Tiene una prioridad ALTA.
- ✓ **Diagrama de objetos**: Muestra los elementos del modelo estático en un momento concreto, habitualmente en casos especiales de un diagrama de clases o de comunicaciones, y está formado por un conjunto de objetos y sus relaciones. Tiene una prioridad ALTA.
- ✓ **Diagrama de componentes**: Especifican la organización lógica de la implementación de una aplicación, sistema o empresa, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos. Tiene una prioridad MEDIA.
- ✓ **Diagramas de despliegue**: Representan 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 sea de hardware o software, así como las interfaces que las conectan, es decir, muestra como los componentes de un sistema se distribuyen entre los ordenadores que los ejecutan. Se utiliza cuando tenemos sistemas distribuidos. Tiene una prioridad MEDIA.
- ✓ **Diagrama integrado de estructura** (UML 2.0): Muestra la estructura interna de una clasificación (tales como una clase, componente o caso típico), e incluye los puntos de interacción de esta clasificación con otras partes del sistema. Tiene una prioridad BAJA.
- ✓ **Diagrama de paquetes**: Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. Suele ser útil para la gestión de sistemas de mediano o gran tamaño. Tiene una prioridad BAJA.

Diagramas de comportamiento.

- ✓ **Diagramas de casos de uso**: Representan las acciones a realizar en el sistema desde el punto de vista de los usuarios. En él se representan las acciones, los usuarios y las relaciones entre ellos. Sirven para especificar la funcionalidad y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. Tiene una prioridad MEDIA.
- ✓ **Diagramas de estado de la máquina**: Describen el comportamiento de un sistema dirigido por eventos. En él aparecen los estados que pueden tener un objeto o interacción, así como las transiciones entre dichos estados. Se lo denomina también diagrama de estado, diagrama de estados y transiciones o diagrama de cambio de estados. Tiene una prioridad MEDIA.
- ✓ **Diagrama de actividades**: Muestran el orden en el que se van realizando tareas dentro de un sistema. En él aparecen los procesos de alto nivel de la organización. Incluye flujo de datos, o un modelo de la lógica compleja dentro del sistema. Tiene una prioridad ALTA.

Diagramas de interacción.

- ✓ **Diagramas de secuencia**: Representan la ordenación temporal en el paso de mensajes. Modela la secuencia lógica, a través del tiempo, de los mensajes entre las instancias. Tiene una prioridad ALTA.
- ✓ **Diagramas de comunicación/colaboración (UML 2.0)**: Resaltan 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. Tiene una prioridad BAJA.

- ✓ **Diagrama de interacción**: Muestra un conjunto de objetos y sus relaciones junto con los mensajes que se envían entre ellos. Es una variante del diagrama de actividad que permite mostrar el flujo de control dentro de un sistema o proceso organizativo. Cada nodo de actividad dentro del diagrama puede representar otro diagrama de interacción. Tiene una prioridad BAJA.
- ✓ **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. Tiene una prioridad BAJA.

Anexo II.- Descarga e instalación de Visual Paradigm.

Descarga e instalación de Visual Paradigm

Obtenemos los archivos desde la página de Visual Paradigm:

http://www.visual-paradigm.com/download/vpuml.jsp?edition=ce

Ofrece dos versiones:

- ✓ **Visual Paradigm for UML (VP-UML)**, versión de prueba de 10 días, ampliable a 30 días mediante registro.
- ✓ **Versión Community-Edition**, para uso no comercial (gratuito).

En cualquier caso necesitamos un código de activación que conseguiremos registrándonos. Se envía al correo electrónico que se indique en el registro.

La versión **Community-Edition** incluye algunas de las funcionalidades de la versión completa, entre las que no se encuentra la generación de código ni la ingeniería inversa, que se verán al final de la unidad por lo que se recomienda empezar por la versión completa de prueba por 30 días, para los que se necesita un código de tipo, conseguiremos el código de activación, que es un archivo de tipo `zvpl`, en este caso llamado `vpsuite.zvpl`.

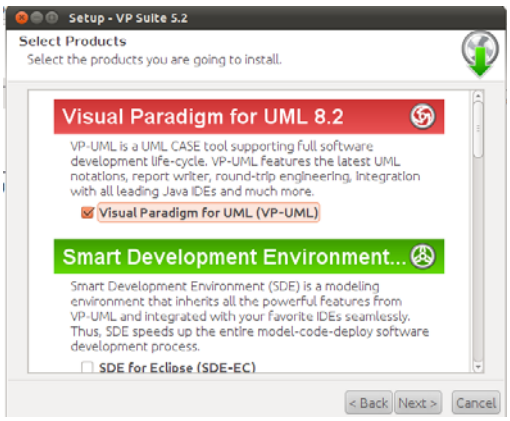
Para la siguiente unidad también usaremos VP-UML, de modo que si fuera necesario tendríamos que conseguir un nuevo código de activación, esta vez de tipo Community-Edition.

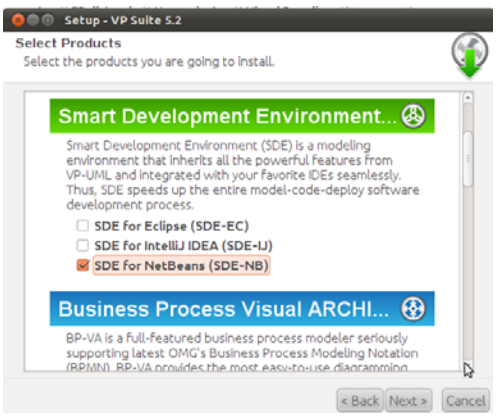
Proceso de instalación

Ejecutaremos el archivo de instalación, que tendrá diferente extensión si es para Windows o para Linux. En nuestro caso suponemos que lo hacemos en un equipo con Ubuntu Desktop 10.10. Se debe tener en cuenta que en el nombre se incluye la versión y la fecha en la que apareció, por lo que estos datos pueden cambiar con el tiempo. Si hacemos la instalación en Windows bastará con hacer doble clic sobre el archivo `.exe`.

```
usuario@equipo:~/VP/ chmod +x VP_Suite_Linux_5_2_20110611.sh
usuario@equipo:~/VP/sudo  ./VP_Suite_Linux_5_2_20110611.sh
```

Durante la instalación tendremos que indicar qué módulos queremos instalar, seleccionaremos Visual Paradigm for UML y el SDE (Smart Development Environment o Entorno de Desarrollo Inteligente), de NetBeans que es el que vamos a usar.





A continuación tendremos que indicar que vamos a utilizar la versión **Enterprise** de ambas herramientas y en que directorio está NetBeans:

Es importante destacar que la instalación debe hacerse sobre una instalación limpia de NetBeans, es decir, que solo podremos instalarlo en el directorio que indicamos una vez.

A continuación se pide un archivo con la **licencia** de la herramienta. Al iniciar la descarga nos pedirá que nos registremos, tras hacerlo podremos solicitar este archivo. Lo insertamos ahora, como hemos instalado dos herramientas nos pedirá dos archivos, pero podemos usar la opción de archivo de licencia combinado, de modo que nos sirva para los dos casos. Si nos lo pide, tendremos que volver a añadirlo después al iniciar Visual Paradigm, con una copia nueva del archivo de clave.

Por último indicamos dónde queremos que ponga los archivos con los proyectos y finalizamos la instalación indicando que no queremos que abra ninguna aplicación.

Iniciar Visual Paradigm

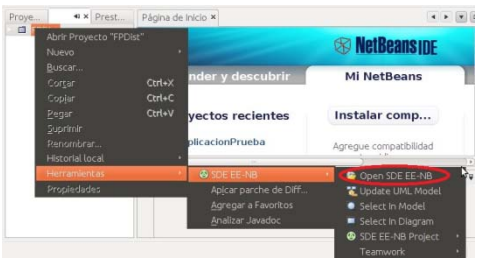
Una vez realizada la instalación tendremos una entrada en el menú **Aplicaciones** llamada **Otras**, si trabajamos con Linux o bien una entrada de menú en el botón Inicio para Visual Paradigm, si es que trabajamos en Windows. En cualquiera de los casos para abrir la herramienta buscamos la opción Visual Paradigm for UML, que se abrevia como **VP-UML**. Al hacer clic se abrirá el programa, y nos preguntará cual es el directorio por defecto para guardar los proyectos, podemos dejar la opción por defecto o seleccionar nuestro propio directorio.

Iniciar VP-UML desde NetBeans

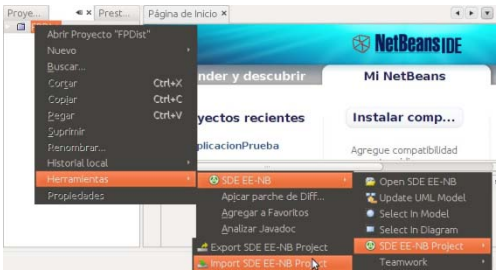
Al hacer la instalación hemos indicado, marcado, que se instale también el SDE para NetBeans, por lo que también tenemos la opción de iniciar la herramienta para usarla integrada con NetBeans. Para abrirlo buscamos dentro del menú de Visual Paradigm la opción **SDE for NetBeans**.

Esto abre la aplicación NetBeans, a la que se ha incorporado una pequeña diferencia, y es que podemos añadir a un proyecto en desarrollo existente un proyecto VP-UML. ¿Cómo lo hacemos?

Estando en la ventana de **Proyectos**, si hacemos clic con el botón secundario sobre un proyecto vemos una serie de opciones, como compilar o construir, ahora, además, abrir el SDE desde **Open SDE EE-NB**, que abre el SDE. La primera vez nos pedirá que importemos un archivo de clave, que podremos obtener con el botón Request Key desde la página oficial. Para ello necesitaremos el correo de registro que hemos utilizado al hacer la instalación.

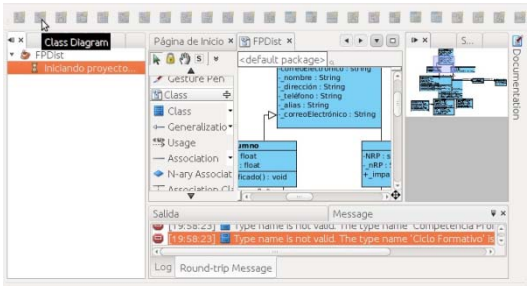


Los proyectos de Visual Paradigm se podrán almacenar en el directorio por defecto, que se denomina `vpproject` y cuelga del directorio principal del proyecto NetBeans, o en otra ubicación. Nosotros nos quedaremos con la opción por defecto.



También podemos importar un proyecto VP-UML que tengamos ya creado seleccionándolo al crear el proyecto existente.

Una vez creado o importado el proyecto, tendremos una serie de botones en la zona superior derecha que nos permitirán crear los diferentes diagramas de UML, y que queden asociados al proyecto NetBeans.



Anexo III.- Generación del diagrama de clases de un problema dado.

Descripción del problema

El Ministerio de Educación ha encargado a **BK Programación** que desarrolle una plataforma de aprendizaje electrónico para que los alumnos de ciclos formativos a distancia tengan acceso a los materiales y puedan comunicarse con sus profesores. Para que los chicos puedan empezar a crear los primeros diagramas de la aplicación Ada les pasa la siguiente descripción del ámbito del problema:

"Los alumnos y alumnas de Ciclos Formativos a Distancia se matriculan de varios módulos formativos al año. Los módulos formativos son impartidos por profesores y profesoras que pondrán los contenidos del módulo a disposición de los alumnos y alumnas. Para superar un módulo hay que hacer una tarea y un examen que se calificarán de uno a diez, y sacar en ambos casos una puntuación superior a cinco. Los exámenes se componen de 30 preguntas que se eligen y ordenan al azar. Las preguntas tienen un enunciado y cuatro posibles respuestas, sólo una de ellas válida. Un ciclo formativo se compone de una serie de competencias profesionales, que tienen una descripción y que, a su vez, están formadas por uno o varios módulos, que tienen un nombre, y un número de horas. Cuando un alumno o alumna supera los módulos correspondientes a una capacidad se le certifica esa capacidad. Cuando se han superado todos los módulos (y por tanto se han adquirido todas las competencias profesionales) se aprueba el ciclo. Cuando un alumno o alumna finaliza el ciclo se emite un certificado de competencias a su nombre donde aparece la descripción de las competencias que forman el ciclo y la nota media obtenida. Si un alumno o alumna no termina de cursar el ciclo completo puede pedir un certificado que acredite las competencias que sí tenga adquiridas. El alumnado y el profesorado se identifican con un alias en el sistema y se comunican a través de correo electrónico. Por motivos administrativos es necesario conocer el nombre y apellidos, dirección completa y teléfono de todas las personas que participan en el sistema, sea como profesores o como alumnos. Para el profesorado, además, se debe conocer su número de registro personal (NRP)"

Extracción de los sustantivos de la descripción del problema.

Primero subrayamos los sustantivos de la descripción del problema (sin repeticiones) y los pasamos a una tabla: (usaremos sólo alumnos y sólo profesores para simplificar el diseño)

Los alumnos de Ciclos Formativos a Distancia se matriculan de varios módulos formativos al año. Los módulos formativos son impartidos por profesores que pondrán los contenidos del módulo a disposición de los alumnos. Para superar un módulo hay que hacer una tarea y un examen que se calificarán de uno a diez, y sacar en ambos casos una puntuación superior a cinco. Los exámenes se componen de 30 preguntas que se eligen y ordenan al azar. Las preguntas tienen un enunciado y cuatro posibles respuestas, sólo una de ellas válida. Un ciclo formativo se compone de una serie de competencias profesionales, que tienen una descripción y que, a su vez, están formadas por uno o varios módulos, que tienen un nombre, y un número de horas. Al sumar las horas de un ciclo obtenemos las horas del módulo. Cuando un alumno supera los módulos correspondientes a una competencia se le certifica esa competencia. Cuando se han superado todos los módulos (y por tanto se han adquirido todas las competencias profesionales) se aprueba el ciclo. Cuando un alumno finaliza el ciclo se emite un certificado de competencias a su nombre donde aparece la descripción de las competencias que forman el ciclo y la nota media obtenida. Si un alumno no termina de cursar el ciclo completo puede pedir un certificado que acredite las competencias que si tenga adquiridas. Los alumnos y profesores se identifican con un alias en el sistema y se comunican a través de correo electrónico. Por motivos administrativos es necesario conocer el nombre y apellidos, dirección completa y teléfono de todas las personas que participan en el sistema, sea como profesores o como alumnos. Para los profesores, además, se debe conocer su número de registro personal (NRP).

Tabla de sustantivos	
Clase/objeto potencial	**Categoría**
Alumno	Entidad externa o rol
Ciclo Formativo a Distancia	Unidad organizacional
Modulo Formativo	Unidad organizacional
Año	Atributo
Profesor	Entidad externa o rol
Contenidos	Atributo
Tarea	Cosa
Examen	Cosa
Uno	Atributo
Diez	Atributo
Pregunta	Cosa
Enunciado	Atributo
Respuesta	Atributo
Competencia Profesional	Unidad organizacional
Descripción	Atributo
Horas	Atributo
Certificado de competencias	Cosa
Nombre	Atributo
Nota media	Atributo
Alias	Atributo
Sistema	Estructura
Nombre	Atributo
Dirección	Atributo
Teléfono	Atributo
Persona	Rol o entidad externa
Número de registro personal	Atributo

Selección de sustantivos como objetos/clases del sistema

Ahora aplicamos los criterios de selección de objetos. En este apartado es necesario destacar que aunque algunos de los sustantivos que tenemos en el anunciado podrían llegar a convertirse en clases y objetos, como los contenidos de un módulo formativo, se descartan en esta fase porque el enunciado no da suficiente información. El proceso de creación de diagramas no es inmediato, sino que está sujeto a revisiones, cambios y adaptaciones hasta tener un resultado final completo.

Tabla de elección de sustantivos como objetos o clases del sistema.	
Clase/objeto potencial	**Criterios aplicables**
Alumno	2,3,4
Ciclo Formativo a Distancia	1,2,3
Módulo Formativo	1,2,3
Profesor	2,3,4
Tarea	1,2,3
Examen	1,2,3
Competencia Profesional	1,2,3
Pregunta	1,2,3
Certificado de competencias	Falla 2,3 rechazado
Sistema	Falla 1,2,3,4 rechazado
Persona	2,3,4

Obtención de los atributos de los objetos.

Buscamos responder a la pregunta ¿Qué elementos (compuestos y/o simples) definen completamente al objeto en el contexto del problema actual?

Tabla de relación de las clases u objetos con sus atributos.	
Clase/objeto potencial	**Atributos**
Alumno	Nombre, dirección, teléfono, alias, correo electrónico.
Ciclo Formativo a Distancia	Nombre, descripción, horas.
Módulo Formativo	Modulo Formativo
Profesor	Nombre, dirección, teléfono, alias, correo electrónico, NRP.
Tarea	Descripción, calificación.
Examen	Descripción, calificación.
Competencia Profesional	Nombre, descripción.
Pregunta	Enunciado, respuestas, respuesta válida.
Persona	Nombre, dirección, teléfono, alias, correo electrónico.

Obtención de los métodos

Buscamos o inferimos en el enunciado verbos, y actividades en general que describan el comportamiento de los objetos o modifiquen su estado.

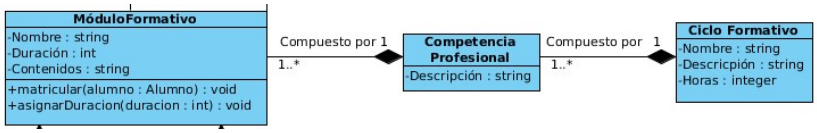
Tabla de clases u objetos del sistema con sus posibles métodos.	
Clase/objeto potencial	**Métodos**
Alumno	CalcularNotaMedia() : void emitirCertificado() : void
Ciclo Formativo a Distancia	

Módulo Formativo	Matricular(Alumno : alumno) : void asignarDuracion(horas: int) : void
Profesor	
Tarea	
Examen	Calificar() añadirPregunta() ordenarPreguntas() crearExamen()
Competencia Profesional	
Pregunta	
Persona	

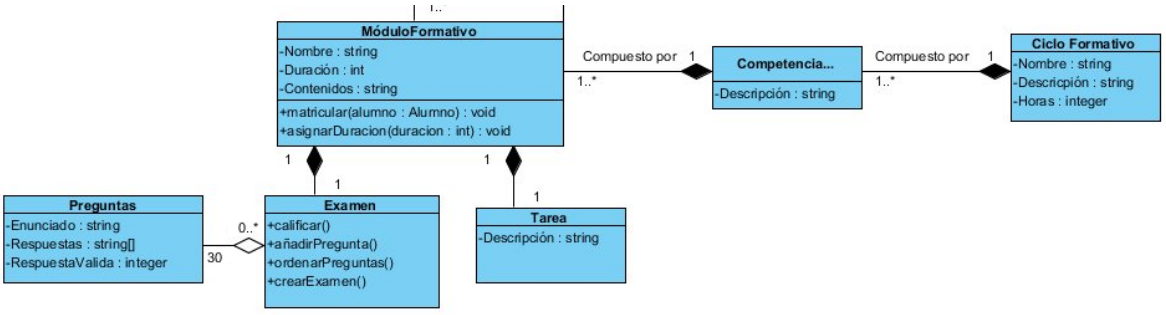
Obtener relaciones

Con las clases ya extraídas y parcialmente definidas (aún faltan por añadir métodos y atributos inferidos de posteriores refinamientos y de nuestro conocimiento) podemos empezar a construir relaciones entre ellas.

Comenzaremos por las clases que hacen referencia a la estructura de los Ciclos, cada Ciclo se compone de una o más competencias profesionales, que no tienen la capacidad de existir por sí mismas, es decir, la competencia no tiene sentido sin su ciclo, por lo que vamos a crear una relación entre ambas clases de composición. De igual manera una competencia profesional se compone de un conjunto de módulos formativos (1 o más) por lo que relacionaremos ambas, también mediante composición.

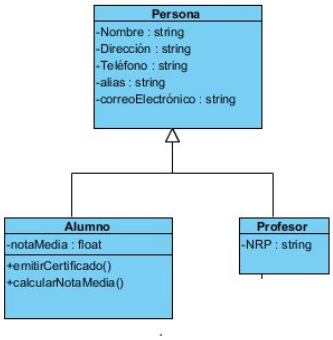


Un módulo formativo a su vez, contiene un examen y una tarea, que tampoco tienen sentido por sí mismos, de modo que también los vamos a relacionarlos mediante composición. El examen por su parte se compone de 30 preguntas, pero éstas pueden tener sentido por sí mismas, y pertenecer a diferentes exámenes, además, el hecho de eliminar un examen no va a dar lugar a que las preguntas que lo forman se borren necesariamente, si leemos con atención el enunciado, podemos deducir que las preguntas se seleccionan de un repositorio del que pueden seguir formando parte [... [Los exámenes se componen de 30 preguntas que se eligen y ordenan al azar...], así que en este caso usaremos la relación de agregación para unirlos.



Por otra parte alumnos y profesores comparten ciertas características, por necesidad del sistema, como son los datos personales, o el correo electrónico, esto induce a pensar que podemos crear una abstracción con los datos comunes, que de hecho, ya hemos obtenido del enunciado en la clase

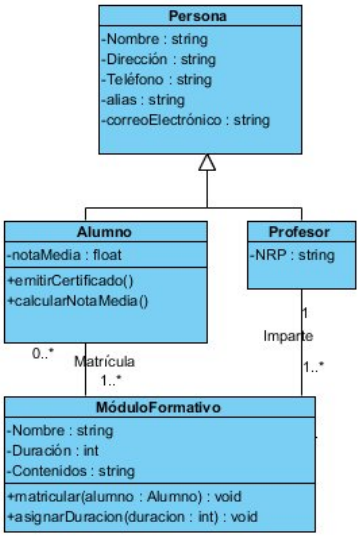
persona, que recoge las coincidencias entre alumnos y profesores y añadir una relación de herencia de la siguiente manera:



Por último queda relacionar a alumnos y profesores con los módulos formativos. Un alumno se matricula de un conjunto de módulos formativos, y un profesor puede impartir uno o varios módulos formativos.

Más concretamente, de cara a la cardinalidad, un alumno puede estar matriculado en uno o varios módulos, mientras que un módulo puede tener, uno o varios alumnos matriculados. Por su parte un profesor puede impartir uno o varios módulos, aunque un módulo es impartido por un profesor.

Éste análisis da como resultado lo siguiente:



Añadir Getters, Setters y constructores

Por último añadimos los métodos que permiten crear los objetos de las clases (constructores) así como los que permiten establecer los valores de los atributos no calculados y leerlos (getters y setters), recuerda que para tener éstos métodos completos es necesario que el atributo tenga establecido su tipo, para que sea tenido en cuenta.

Para añadir los getters y setters en Visual Paradigm, basta con desplegar el menú contextual del atributo y seleccionar la opción Create Getter and Setter.

También hay que añadir los métodos que no se infieren de la lectura del enunciado, por ejemplo los que permiten añadir módulos a las competencias, o competencias a los ciclos. Para comprobar estos métodos puedes descargar el diagrama de clases en un proyecto VP-UML un poco más adelante.

Añadir documentación

Por último se debe rellenar la documentación de cada clase, atributo y método con una descripción de los mismos que será necesaria para la generación de informes posterior. A continuación se listan las clases con su documentación:

- ✓ **Persona**: Generalización para agrupar las características comunes de alumnos y profesores como personas que interactúan con el sistema. De una persona interesa conocer su nombre, dirección, teléfono, alias y correo electrónico.
- ✓ **Alumno**: Es un tipo de persona. Representa a las personas que se matriculan de un ciclo formativo. Un alumno puede estar matriculado durante varios años en un ciclo. Está matriculado de un ciclo siempre que está matriculado en algún módulo que forme parte del ciclo. Para aprobar un Ciclo hay que superar todos los módulos que lo componen. Para superar un módulo hay que realizar la tarea y aprobar el examen, que está compuesto de 30 preguntas de tipo test, con cuatro respuestas posibles una de las cuales es la correcta. De un alumno interesa almacenar su nota media.
- ✓ **Profesor**: Es un tipo de persona. Representa a las personas que imparten los módulos formativos. Evalúan las tareas que realizan los alumnos y se encargan de poner los contenidos a disposición de los alumnos. De un profesor interesa almacenar su número de registro personal.
- ✓ **Ciclo Formativo a Distancia**: Es uno de los núcleos centrales del sistema. Representa los estudios que se pueden realizar a distancia. Un ciclo formativo se compone de un conjunto de competencias profesionales que se componen a su vez de módulos formativos. Se aprueba un ciclo formativo cuando se adquieren todas las competencias que lo forman. De un ciclo formativo se almacena su nombre, descripción y horas totales.
- ✓ **Competencia Profesional**: Representan el conjunto de conocimientos generales que se adquieren cuando se completa un ciclo formativo. Se componen de módulos profesionales y se adquiere una competencia cuando se superan los módulos que la componen. De una competencia se almacena su descripción.
- ✓ **Módulo Formativo**: Unidades formativas que cursa un alumno. Un módulo formativo tiene una serie de contenidos que el alumno debe estudiar, y una tarea y un examen que el alumno debe hacer. Cuando se aprueban la tarea y el examen se supera el módulo. De un módulo se almacena su nombre, duración y contenidos.
- ✓ **Tarea**: Actividad relacionada con los contenidos de un módulo que debe realizar un alumno. Tiene una puntuación de uno a diez y es evaluada por el profesor. La nota se asigna a cada alumno para la matrícula del módulo al que pertenece la tarea. De una tarea se almacena su descripción.
- ✓ **Examen**: Conjunto de treinta preguntas que se evalúa de uno a diez. Un examen puede desordenarse y calificarse calculando cuantas preguntas son correctas.
- ✓ **Pregunta**: Forman los exámenes de un módulo. Se compone del enunciado y cuatro posibles respuestas de las cuales sólo una es válida.