UNIDAD 5: ELABORACIÓN DE DIAGRAMAS DE CLASES

Módulo Profesional: Entornos de desarrollo

ÍNDICE

RESUMEN INTRODUCTORIO

En la presente unidad se estudiarán conceptos relacionados con los diagramas de clase. El diseño de diagramas de clase permite a los desarrolladores comprender, de forma global, cómo debe ser la implementación del software en función de las clases que lo componen, junto con sus atributos y métodos. Para el diseño de diagramas de clase es necesario conocer la notación UML (cuyas siglas en inglés son Unified Modeling Language). Este lenguaje se creó para estandarizar todo el proceso de modelado de sistemas. A través de él se diseñan las clases y las relaciones entre ellas.

En la unidad se destacan las relaciones de Herencia, composición, agregación, asociación y uso, fundamentales para el diseño de culalquier software actual. Además, es importante destacar la existencia de herramientas que van a permitir realizar este tipo de diagramas. Principalmente se estudiarán 2 de ellas: ArgoUML y StarUML.

Una de las características de este tipo de herramientas es ofrecer la posibilidad de convertir el diagrama de clases diseñado en código fuente, de forma automática y viceversa. Este proceso será muy importante a la hora de implementar el software, ya que ayudará al equipo de programadores a realizar sus tareas de forma más rápida y eficiente. No obstante siempre será necesario llevar a cabo los casos de prueba pertinentes, con el objetivo de encontrar defectos que puedan ser solventados

INTRODUCCIÓN

En las unidades anteriores se ha comentado la importancia que tiene establecer una serie de requisitos y estructuras necesarias para crear un sistema software con anterioridad a la implementación del código. A través del lenguaje UML es posible diseñar una serie de diagramas y realizar los diferentes diseños previos al proceso de codificación.

Las factorías de software trabajan de forma constante con este tipo de diagramas, los cuales ayudan a darle sentido al ciclo de vida de la aplicación y facilitan la transición entre cada una de las fases.

CASO INTRODUCTORIO

En el equipo de proyecto de Juan, existen 2 analistas y 2 diseñadores. Uno de los clientes de la factoría de software decide aumentar la funcionalidad de un software que fue realizado por el equipo de Juan y que se encuentra actualmente en fase de mantenimiento preventivo.

Para ello contacta con Juan y presenta una serie de mejoras y novedades con respecto a la funcionalidad anterior. En este punto, se hace necesario retomar todos los diagramas de clase de la aplicación actual, con el objetivo de analizar de nuevo su funcionalidad y adaptar los cambios propuestos por Juan en la mayor brevedad posible. Sin la existencia de estos diagramas de clase, no sería viable abordar la actualización del software del cliente en un corto periodo de tiempo.

Al finalizar la unidad el alumnado:

- Conocerá los distintos diagramas UML.
- Conocerá la notación de los diagramas de clases.
- Será capaz de identificar las relaciones existentes entre clases en UML.
- Será capaz de realizar ingeniería inversa en los lenguajes de programación Java y C++.

1. DIAGRAMAS DE CLASES: NOTACIÓN

El Lenguaje de Modelamiento Unificado (**UML** - Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software.

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de agregación. Una clase es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica. Está compuesto por los siguientes elementos:

- **Clase**: Atributos, Métodos y Visibilidad.
- **Relaciones**: Herencia, Composición, Agregación, Asociación y Uso.

PARA SABER MÁS

Información sobre UML y los distintos diagramas que se pueden generar:

- Tema: Herramientas UML, Análisis y diseño UML

Para ampliar información sobre los diagramas de clases visite la web de Microsoft:

- Diagramas de clases de UML: Instrucciones

1.1 Notación de Clase

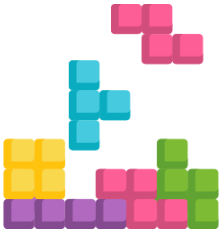
Las **clases** se representan por rectángulos que muestran el nombre de la clase y opcionalmente el nombre de las operaciones y atributos. Los compartimientos se usan para dividir el nombre de la clase, atributos y operaciones. Adicionalmente las restricciones, valores iniciales y parámetros se pueden asignar a clases. En UML, una clase es representada por un rectángulo que posee tres divisiones:

NOMBRE DE LA CLASE
ATRIBUTOS
OPERACIONES O MÉTODOS

Imagen: Representación UML de una clase

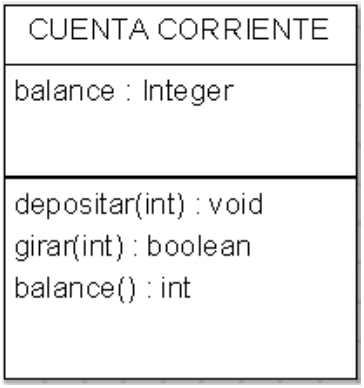
En donde cada una de las partes representa su cometido:

- **Superior**: Contiene el **nombre** de la Clase.
- **Intermedio**: Contiene los **atributos** (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).
- **Inferior**: Contiene los **métodos** u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).



EJEMPLO PRÁCTICO

Una Cuenta Corriente que posee como característica "Balance". Además, puede realizar las operaciones de: "Depositar", "Girar" y "Balance".
Solución: Diagrama de clases, en este caso la clase única con nombre, atributos y métodos, en este orden:



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

Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

- **public (+)**: El atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
- **private (-)**: El atributo solo será accesible desde dentro de la Clase.
- **protected (#)**: El atributo no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de las subclases que se deriven.

Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

- **public (+)**: El método será visible tanto dentro como fuera de la Clase, es decir, es accesible desde todos lados.
- **private (-)**: El método solo será accesible desde dentro de la clase (solo otros métodos de la clase lo pueden acceder).
- **protected (#)**: El método no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de métodos de las subclases que se deriven.

1.2 Relaciones: Herencia, composición, agregación, asociación y uso

Definido el concepto de Clase, se hace necesario detallar cómo se pueden interrelacionar dos o más clases (cada una con características y objetivos diferentes). Previamente, se procede a explicar el concepto de **cardinalidad** de relaciones.

En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, anotándose en cada extremo de la relación y pudiendo ser de tipo:

- **Uno o muchos**: 1..* (1..n).
- **0 o muchos**: 0..* (0..n).
- **Número fijo**: m (m denota el número).

1.2.1 Herencia (Especialización/Generalización)

Indica que una subclase hereda los métodos y atributos especificados por una superclase, por ende, la subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la superclase de tipo public y protected (ver ejemplo). En la figura se especifica que Coche y Camión heredan de Vehículo, es decir, Coche posee las Características de Vehículo (dueño, puertas, etc.) además posee algo particular, que es descapotable, en cambio Camión también hereda las características de Vehículo (dueño, puertas, etc.) pero posee como particularidad propia tara y carga. Cabe destacar que fuera de este entorno, lo único visible es el método característica aplicable a instancias de Vehículo, Coche y Camión, pues tiene definición pública, en cambio atributos como descapotable no son visibles por ser de tipo privado.

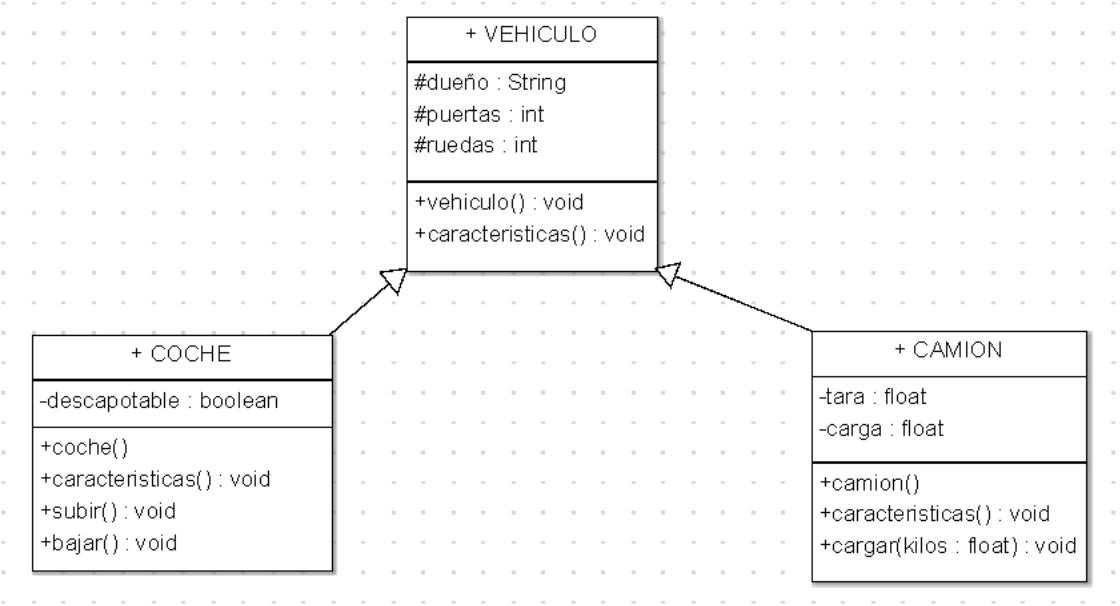


Imagen: Ejemplo de relación de herencia

Para que exista herencia clara, las subclases deben responder afirmativamente a la expresión "es un". Por ejemplo, Coche "es un" vehículo y Camión "es un" vehículo. Por tanto, la herencia existe. La notación la marca la flecha hueca.

1.2.2 Agregación

Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, se presentan dos posibilidades:

- **Por Valor**: Es un tipo de relación **estática**, en donde el tiempo de vida del objeto incluido está condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
- **Por Referencia**: Es un tipo de relación **dinámica**, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada **Agregación** (el objeto base utiliza al incluido para su funcionamiento).

En donde se destaca que:

- Un Almacén posee Clientes y Cuentas (los rombos van en el objeto que posee las referencias).

- Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

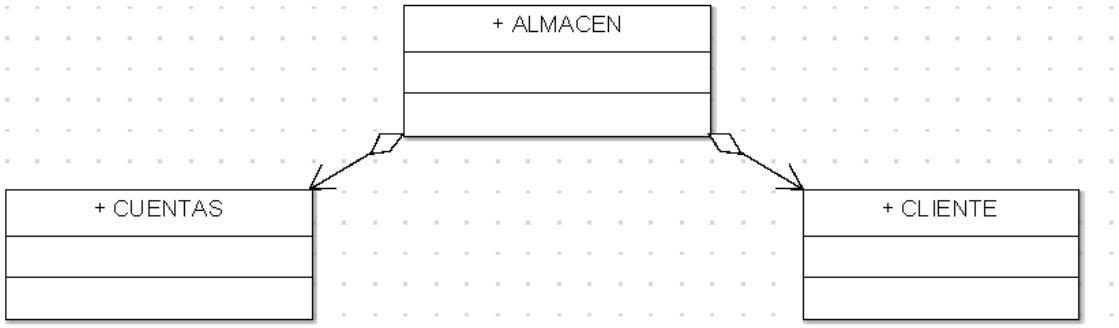


Imagen: Ejemplo de relación de agregación

1.2.3 Asociación

La relación entre clases conocida como Asociación permite asociar objetos que colaboran entre sí. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

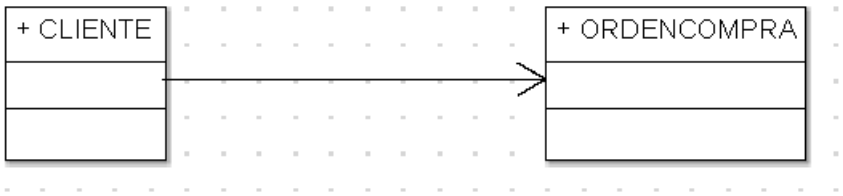


Imagen: Ejemplo de relación de asociación

Un cliente puede tener asociadas muchas Órdenes de Compra, en cambio una orden de compra solo puede tener asociado un cliente.

1.2.4 Dependencia o Instanciación (uso)

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase). Se denota por una flecha punteada.

El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra como, por ejemplo, una aplicación gráfica que instancia una ventana (la creación del Objeto Ventana está condicionado a la instanciación proveniente desde el objeto Aplicación):

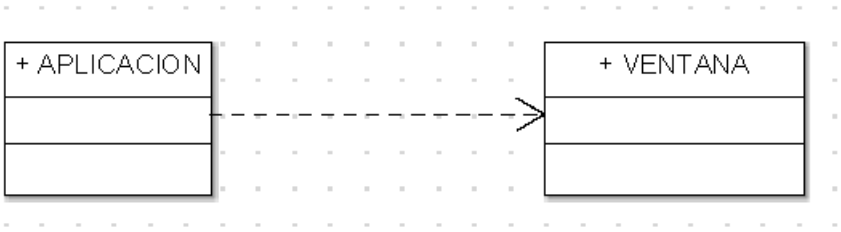


Imagen: Ejemplo de relación de uso

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

ARTÍCULO DE INTERÉS

Mecanismos de herencia y su implementación en Java:

- Relaciones entre clases: Diagramas de clases UML

2. HERRAMIENTAS PARA LA ELABORACIÓN DE DIAGRAMAS DE CLASES. INSTALACIÓN

UML incluye un conjunto de técnicas de notación gráfica para especificar, visualizar y documentar los modelos de sistemas de software, incluyendo su estructura y diseño, de manera que cumpla con todos estos requisitos. Existen muchas herramientas profesionales de diagramas populares, tanto comerciales como de software libre.

A continuación, se indican las principales herramientas UML de código abierto: StarUML, ArgoUML, Violet UML Editor, Astah Community, BOUML, Dia, UMLet, UMLGraph, MetaUML, Visual Paradigm for UML 10.1 Community Edition, T4 Editor plus modeling tools, NetBeans IDE UML y Eclipse UML2 Tools. Por su parte, algunos generadores Online de Diagramas UML son yUML y zOOML.

PARA SABER MÁS

Listado muy completo de herramientas para la elaboración de diagramas de clases:

- Las mejores herramientas UML

2.1 ArgoUML

ArgoUML es una herramienta implementada en Java, con la cual se tiene la posibilidad de crear modelos UML que serán compatibles con los estándares de la versión 1.4 de dicho lenguaje. Puede ser utilizado en cualquier plataforma de Java y además se encuentra disponible en 10 idiomas (inglés, italiano, francés y español, entre otros). Se pueden crear diferentes tipos de diagramas, entre ellos diagrama de clases, diagrama de estados, diagrama de actividad, diagrama de casos de uso, diagrama de colaboración, diagrama de despliegue y diagrama de secuencia. Se tiene la posibilidad de salvar esos diagramas en los siguientes formatos: PNG, GIF, SVG, PGML, PostScript y encapsulated PS.

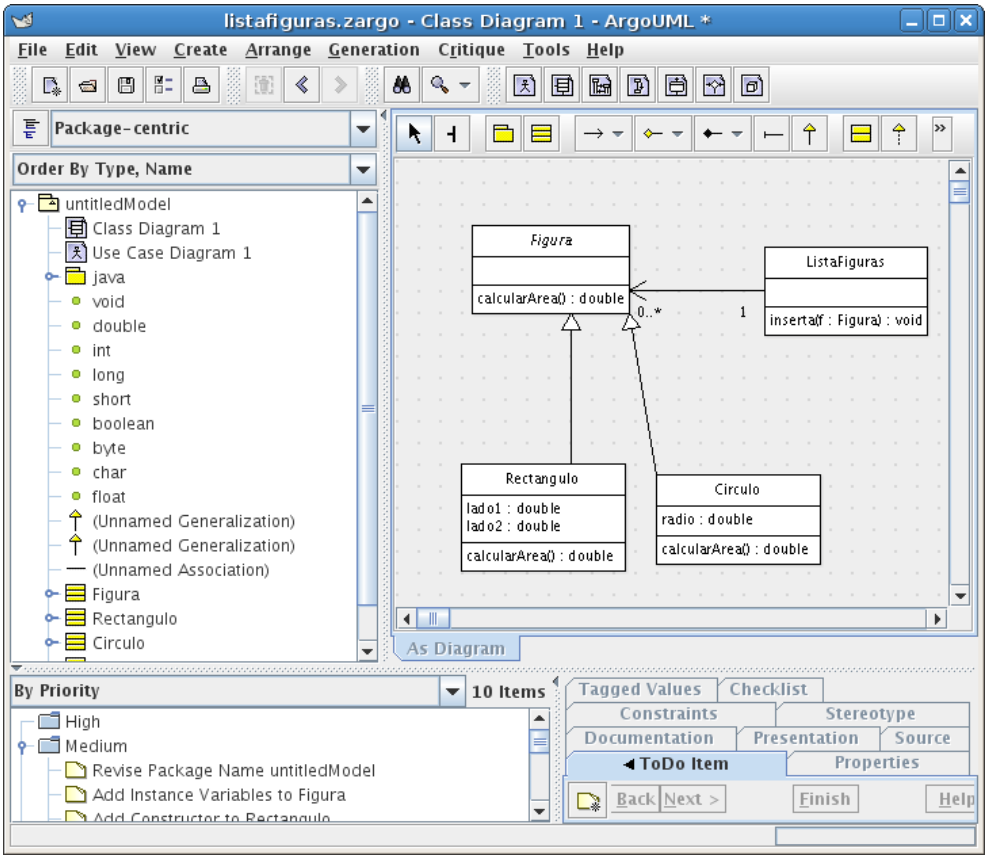
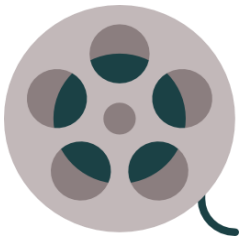


Imagen: Interfaz de ArgoUML



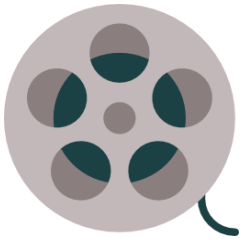
VIDEO DE INTERÉS

Instalación de ArgoUML:

- Tutorial de instalación ArgoUML

2.2 StarUML

StarUML es una herramienta de programación escrita en código abierto y de distribución libre que genera diagramas UML (Diagramas de clases, estructuras, componentes, paquetes, objetos, actividades, módulos, comunicación, estados, actividades, secuencias, etc.). Estos diagramas tienen como función explicar el proceso que hace cada objeto y elemento de la aplicación, de modo que convierte el diseño gráfico en una serie de esquemas y códigos necesarios para el buen funcionamiento del programa. Es compatible con lenguaje C++ o Java. Presenta plantillas predeterminadas que ayudan a entender cómo funciona este sistema, posibilitando crear diseños propios a partir de modelos propuestos.



VIDEO DE INTERÉS

Crear código Java a partir de diagramas de clases con StarUML:

- Crear código JAVA con modelos de clases en starUML 2.0

3. GENERAR CÓDIGO A PARTIR DE DIAGRAMAS DE CLASES

Son muchas las herramientas que permiten generar código a partir de diagramas de clase y viceversa. A continuación, se presentan un ejemplo con Visual Studio Ultimate.

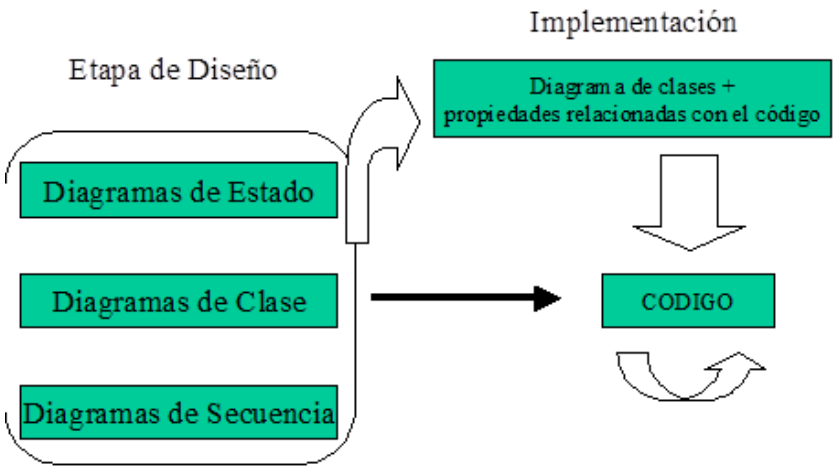


Imagen: Esquema generación de código a partir de diagramas

En Visual Studio Ultimate se puede generar código a partir de los diagramas de clases UML mediante el comando de **Generar código**. De forma predeterminada, el comando genera un tipo de C# para cada tipo UML que seleccione. Se puede modificar y extender este comportamiento modificando o copiando las plantillas de texto que generan código. Se puede especificar un comportamiento diferente para los tipos contenidos en los diferentes paquetes del modelo. El comando Generar código es especialmente adecuado para generar código a partir de la selección de elementos del usuario y para generar un archivo para cada clase UML u otro elemento. Por ejemplo, la captura de pantalla muestra dos archivos de C# generados a partir de dos clases UML.

Como alternativa, para generar código en el que los archivos generados no tienen una relación de 1:1 con los elementos UML, se puede considerar la posibilidad de escribir plantillas de texto que se invocan con el comando Transformar todas las plantillas.

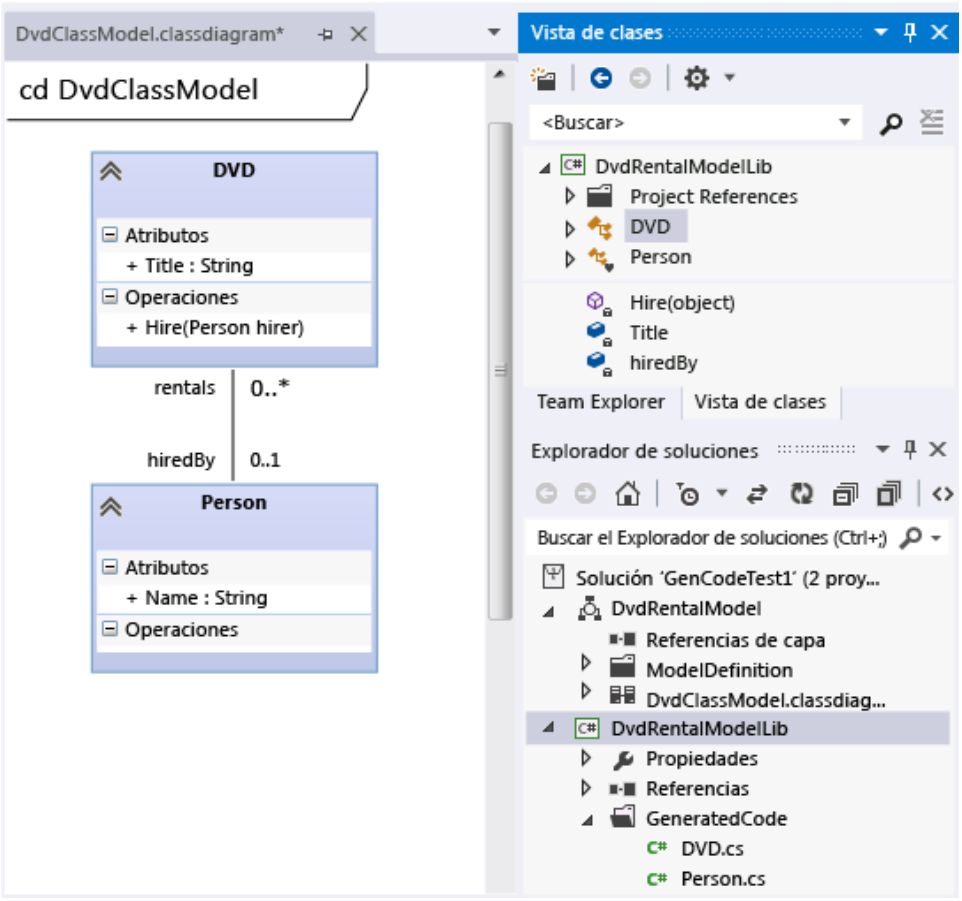


Imagen: Generación de código a partir de diagrama de clases

4. GENERAR DIAGRAMAS DE CLASES A PARTIR DE CÓDIGO

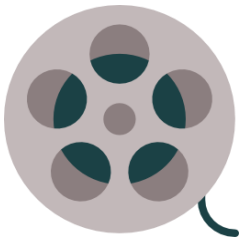
Se utiliza el mismo software para realizar el proceso inverso: generar diagramas de clases a partir de código. Para agregar clases de C# desde el código al diagrama de clases de UML en Visual Studio Ultimate, se arrastran dichas clases o espacios de nombres desde el Explorador de soluciones, desde un gráfico de dependencias o desde el Explorador de arquitectura al diagrama de clases de UML en cuestión. Las clases de las que dependen también aparecerán en el Explorador de modelos UML.

Para agregar clases desde código de programa a un modelo UML:

- Abrir un proyecto de C#.
- Agregar un diagrama de clases UML a la solución:
    - En el menú de Arquitectura, elegir Nuevo diagrama. En el cuadro de diálogo → Agregar nuevo diagrama, seleccionar Diagrama de clases UML. Se creará un proyecto de modelado si no existiera ninguno.

- Abrir el Explorador de arquitectura:
  - En el menú Arquitectura → Ventanas → Explorador de arquitectura.
- Arrastrar los espacios de nombres o tipos del Explorador de arquitectura a la superficie del diagrama de clases UML.

Para ver un tipo, expandir la vista de clases en la primera columna del Explorador de arquitectura y, a continuación, expandir un espacio de nombres en la columna siguiente. Se observarán los tipos en la tercera columna.



VIDEO DE INTERÉS

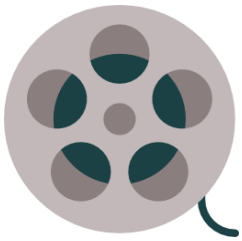
En este videotutorial obtendrá una base para seguir investigando sobre cómo generar diagramas de clases a partir de código Java con NetBeans:

- Generar UML a partir del código en Netbeans

Se han detallado los procedimientos para realizar ingeniería inversa a través del IDE VisualStudio. Se procede ahora cómo hacerlo con el software NetBeans y el lenguaje de programación Java. Los diagramas de clases se pueden generar desde NetBeans, y con base en esos diagramas se puede generar el código para el programa. Para esto se debe instalar el plugin UML de acuerdo a las siguientes instrucciones:

- Lo primero es acceder a "Herramientas → Complementos"
- Luego seleccionar la Pestaña Configuración, clic en agregar.
- Y ahora en el nuevo cuadro de dialogo que aparecerá en Nombre poner UML y en URL: http://ea.ddns.com.br:8090/netbeans6.8/UML/catalog.xml
- Ahora ir a Plugins Disponibles y buscar la entrada con el Nombre UML, o con el nombre que se ha escrito en la ventana anterior, marcar e instalar.
- Con esto ya está instalado el plugin, ahora para probarlo crear un nuevo proyecto y en el menú para escoger se tiene la opción de UML.
- Luego ya se puede modelar, solo basta arrastrar las clases al espacio de trabajo y modificarlas con un par de clics, Java automáticamente genera todos los get & set de los campos que se añadan.

- Y ahora para generar el código es mucho más fácil, solo se pulsa clic derecho a la clase y pulsar Generate Code, aparecerá un dialogo en donde seleccionar en qué proyecto y en que paquete se desea generar el código, se selecciona, se pulsa en aceptar y listo la clase ya está hecha.
- Una vez instalado el plugin se pueden crear los diagramas y de paso el código asociado a los mismos. A esto se le conoce como ingeniería inversa.



VIDEO DE INTERÉS

En este videotutorial puede ver el proceso anterior:

- Instalar plugin de UML en Netbeans

RESUMEN FINAL

En esta unidad se pretende inculcar el diseño UML para el modelo de diagramas de clase. Una clase va a permitir crear un esqueleto a través de una abstracción del mundo real que tendrá un nombre, unos atributos y unos métodos. Estas 3 partes son tenidas en cuenta en el diseño UML. Normalmente, entre las clases de un software existen relaciones, que serán de herencia, agregación, asociación, instanciación o/y dependencia, según la funcionalidad del mismo. A través de una serie de herramientas como ArgoUML o StarUML va a ser posible realizar diseños UML de diagramas de clases. A su vez, existe la posibilidad, en algunas de ellas de generar código a partir de dichos diagramas y viceversa.