ÍNDICE

Unidad didáctica 6

ELABORACIÓN DE DIAGRAMAS DE CLASES

OBJETIVOS:

Esta unidad didáctica tiene como objetivo que el alumno sea capaz de elaborar diagramas de clases a partir de un sistema de información dado y aplicando los conceptos de la unidad didáctica 5.

6.1. CONCEPTOS INTRODUCTORIOS.

A continuación, introduciremos varios conceptos fundamentales (clases, atributos métodos, relaciones, herencia y clases abstractas) para entender que es un diagrama de clase.

En un diagrama de clase se representan las clases junto con sus relaciones, así como que para cada clase hay que indicar sus métodos y atributos y que las relaciones entre las clases pueden ser de asociación, de dependencia y de generalización o herencia.

6.1.1. Clases.

Es un plantilla que permite la **creación de objetos** y contiene la **descripción de las características comunes de esos objetos** y está formada por dos partes: estática (campo con nombre, apellidos) y dinámica (procedimientos o funciones, métodos…).

Está fuertemente relacionado con el **tipo abstracto de datos** que viene definido por una serie de operaciones y valores.

En **programación**, una clase es una **estructura de datos** que contiene datos y procedimientos o funciones capaces de operar sobre esos datos.

La clase “alumnos” representa una estructura de datos que contiene los datos de los alumnos y una serie de procedimientos tales como: buscar alumno, contar alumnos, etc.

6.1.2. Atributos.

Son las **propiedades de los objetos**, como, nombre, edad, etc., y que se corresponden con las “variables” de la programación estructurada.

Hay atributos **propios** (definidas dentro del objeto) y **heredados** (en un objeto diferente).

Los atributos de la clase “alumnos” del apartado anterior serían: nombre, apellido, edad, etc.

6.1.3. Métodos.

Son las **operaciones asociadas al objeto** (crear un alumno, añadir, borrar). Un objeto puede disponer de un método **propio** (incluidos dentro del objeto) y **heredado** (definidos en un objeto diferente). Los objetos deben ser creados o instanciados a través de operaciones (métodos) llamados **constructores** o inicializadores.

Las dos operaciones son: **constructor** (crea y/o inicializa un objeto) y **destructor** (libera y/o destruye el propio objeto).

Los métodos de la clase “alumnos” del apartado anterior serían: crear alumno, borrar alumno, buscar alumnos, contar alumno, etc.

6.1.4. Objetos.

Es una entidad que **contiene atributos y métodos** y que interactúan mediante mensajes por lo que el **paso de mensajes** es el protocolo de comunicación interobjetual.

Un objeto puede ser el objeto “profesor”, con atributos tales como: nombre, apellido, edad, etc. Y los métodos pueden: crear profesor, borrar profesor, buscar profesor, etc.

6.1.5. Relaciones.

Los tipos de relaciones son **asociación, dependencia y generalización**.

- Las relaciones de **asociación** son las típicas, igual que para los modelos entidad-relación (1 a N, 1 a 1, N a N y N a1).
- Las relaciones de **dependencia** son aquellas donde unas dependen de la existencia de otras.

- Las relaciones de **generalización** sirven para generalizar sobre un aspecto determinado así como para que las subentidades hereden de la entidad general sus métodos y atributos.

Supongamos que tenemos una clase “profesor” y una clase “alumno”, entonces una posible relación de asociación de N a N sería la relación “impartir clase”

6.1.6. Herencia.

Es un mecanismo de reutilización de código que permite a los programadores crear nuevas clases a partir de clases existentes, de modo que se permite a una nueva clase usar los métodos y atributos definidos en otra clase como si fueran propios.

Los tipos de herencia son:

- **Simple.** Una clase puede tener solo un descendiente.
- **Múltiple.** Cada clase puede tener una o más superclases inmediatas. Es aquella herencia en la que cada clase puede heredar métodos y/o atributos de cualquier número de superclases.

El lenguaje de programación C++ admite herencia simple y múltiple.

Supongamos que tenemos la clase “profesor” y la clase “alumno”, entonces tendríamos una clase “persona” de la cual heredan las otras dos clases, tanto sus atributos como sus métodos.

6.1.7. Clases abstractas.

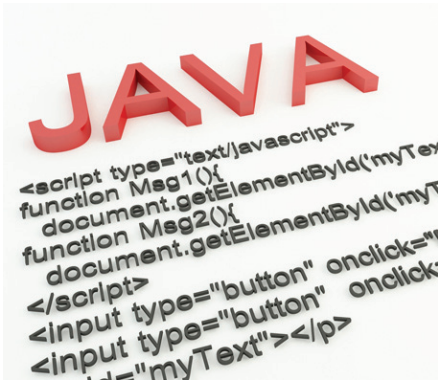
Hay ocasiones, cuando se desarrolla una jerarquía de clases, en que algún comportamiento está presente en todas ellas pero se materializa de forma distinta para cada una.

Por ejemplo, **pensemos en una estructura de clases para manipular figuras geométricas**. Podríamos pensar en tener una clase genérica, que podría llamarse **FiguraGeometrica** y una serie de clases que extienden a la anterior, que podrían ser círculo, polígono, etc.

Podría haber un **método dibujar** dado que sobre todas las figuras puede llevarse a cabo esta acción, pero las operaciones concretas para desarrollarlas dependen del tipo de figura en concreto (de su clase).

Por otra parte, la acción dibujar no tiene sentido para la clase genérica FiguraGeometrica, porque esta clase representa una abstracción del conjunto de figuras posibles.

Para resolver esta problemática **Java** proporciona las clases y métodos abstractos.



- Un **método abstracto** es un método declarado en una clase, para el cual esa clase no proporciona la implementación (el código).
- Una **clase abstracta** es una clase que tiene al menos un método abstracto. Una clase que extiende a una clase abstracta debe implementar los métodos abstractos (escribir el código), o bien volverlos a declarar como abstractos, con lo que ella misma se convierte también en clase abstracta.

El siguiente ejemplo muestra como **una clase abstracta** "Figura" contiene un método público "Figura", y una clase abstracta "área", así como dos clases abstractas "Circulo" y "Cuadrado" que heredan de la clase principal "Figura".

```
public abstract class Figura {

 protected double x;
 protected double y;

 public Figura (double x, double y) {
this.x = x;
this.y = y;
 }

 public abstract double area ();
}
public class Circulo extends Figura {
 private double radio;
 public Circulo (double x, double y, double radio) {
super(x,y);
this.radio = radio;
 }
 public double area () {
return Math.PI*radio*radio;
 }
}

public class Cuadrado extends Figura {
 private double lado;
 public Cuadrado (double x, double y, double lado) {
super(x,y);
this.lado = lado;
 }
 public double area () {
return lado*lado;
 }

}
```

De lo visto en este apartado, es necesario tener muy presente que una **clase** engloba la descripción de las características comunes de los objetos. Es fundamental saber

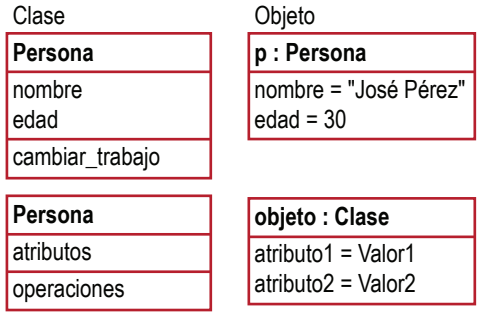
que los **atributos** son las propiedades de los objetos, y que los **métodos** son las operaciones asociadas a esos objetos. Por otro lado, el **objeto** en sí es una entidad que contiene atributos y métodos.

Al mismo tiempo, las **relaciones** son las distintas **uniones entre las clases**. Existen distintos **tipos** de relaciones y son: asociación, dependencia y generalización.

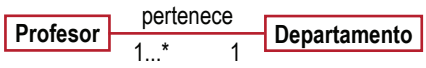
La **herencia**, por su parte, consiste en **reutilizar** código y **crear** nuevas clases a partir de clases existentes.

No olvidemos tampoco que una clase abstracta tiene al menos un método abstracto.

En el siguiente ejemplo **definiremos** lo que es una **clase** y un **objeto** así como las distintas **relaciones existentes** tales como son: **asociación, herencia y composición**.



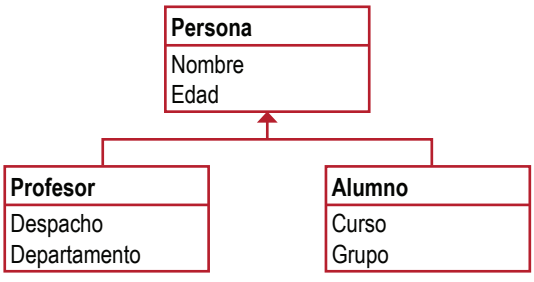
Relación de asociación



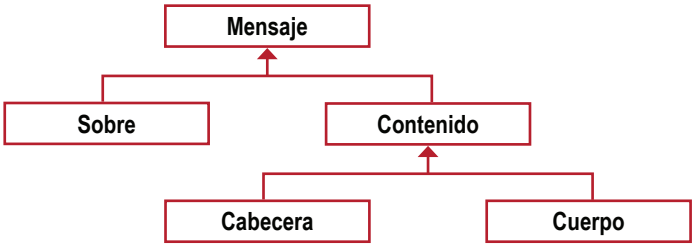
Multiplicidad

*	0 ó más
1...*	1 ó más
3...5	3, 4, ó 5
3, 5	3 ó 5

Relación de herencia



Relación de composición



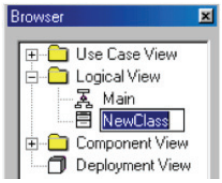
La siguiente presentación nos describe de un modo básico los conceptos anteriores: http://es.slideshare.net/NesMey/clases-objetos-y-herencia.

6.2. USO DE HERRAMIENTAS CASE PARA ELABORAR DIAGRAMAS DE CLASE.

En este apartado se van a explicar una serie de **pasos a seguir** para elaborar los diagramas de clase usando **herramientas CASE**. Estos pasos son los siguientes:

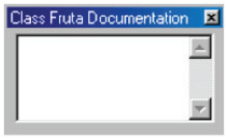
Paso 1. Creación del diagrama de clase

1. Hacer clic con el botón derecho del ratón sobre la Lógical View del Browser.
2. Seleccionar en el menú New Class.
3. La clase aparece en la ventana del Browser, de un [Enter] en la clase seleccionada para poder asignarle el nombre.



Paso 2. Creación de documentación de una clase

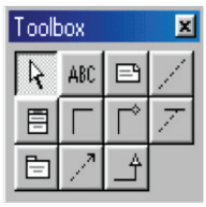
1. Seleccionar la clase en el Browser.
2. Posicionar el cursor en la ventana de documentación.



Paso 3. Creación de un diagrama de clase

1. Hacer doble clic con el botón izquierdo del ratón en el diagrama principal de la **Lógical View** en el Browser.
2. Aparece la ventana del diagrama de clases.
3. Mediante la técnica arrastrar-soltar llevar a la ventana del diagrama la clase seleccionada.
4. Repetir el paso 3 para agregar todas las clases necesarias.

El diagrama de clases también puede crearse utilizando la barra de herramientas.



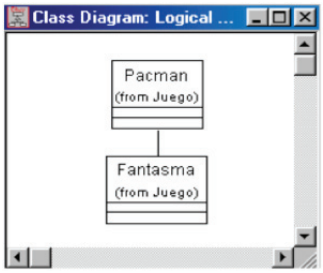
Paso 4. Creación de relaciones

Las relaciones que se tratarán son las siguientes:

- Relación de asociación.
- Relación de agregación.
- Multiplicidad.
- Relación de herencia.

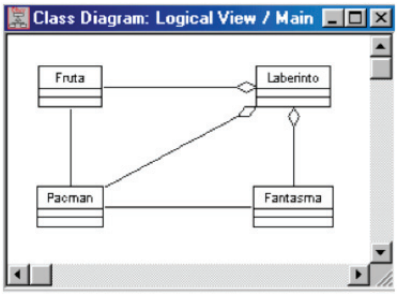
Paso 5. Creación de relación de asociación

1. Hacer clic con el botón izquierdo del ratón en el icono de asociación de la barra de herramientas.
2. Pulsar con el botón izquierdo del ratón en la clase asociada en el diagrama de clases.
3. Arrastrar la línea de asociación a la otra clase asociada.



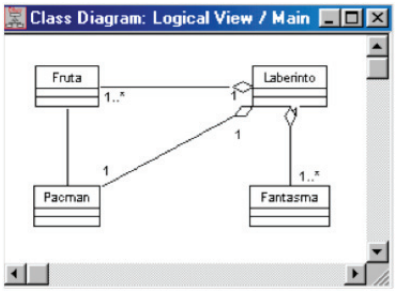
Paso 6. Creación de relación de agregación

1. Seleccionar el icono de agregación de la barra de herramientas.
2. Hacer clic con el botón izquierdo del ratón en la clase que juega el rol de “parte” en el diagrama de clases y soltar la línea de agregación en la clase que juega el rol de agregado.



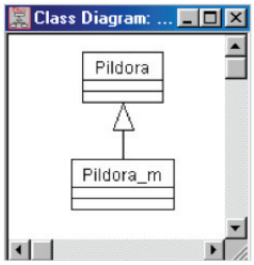
Paso 7. Indicadores de multiplicidad

Hacer un clic con el botón derecho del ratón en la línea de la relación y seleccionar la multiplicidad deseada.



Paso 8. Herencia

1. Seleccionar el icono de herencia de la barra de herramientas.
2. Pulsar el botón izquierdo del ratón en la subclase y soltar la línea de herencia en la superclase.



En el siguiente enlace puedes ver distintos tipos de herramientas para UML: http://goo.gl/x91FV.

Lo primero de todo, cuando se quiere elaborar un diagrama de clase, es crear dicho diagrama y, a continuación, la documentación de la clase.

6.3. MÓDULOS INTEGRADOS PARA ELABORAR DIAGRAMAS DE CLASE.

El módulo integrado que se usa actualmente para elaborar los diagramas de clase son las conocidas "**herramientas CASE**". Existe una gran variedad de éstas con características específicas. A continuación describiremos algunas de ellas, desde las más actuales hasta otras que ya no se usan tanto.

Microsoft Project

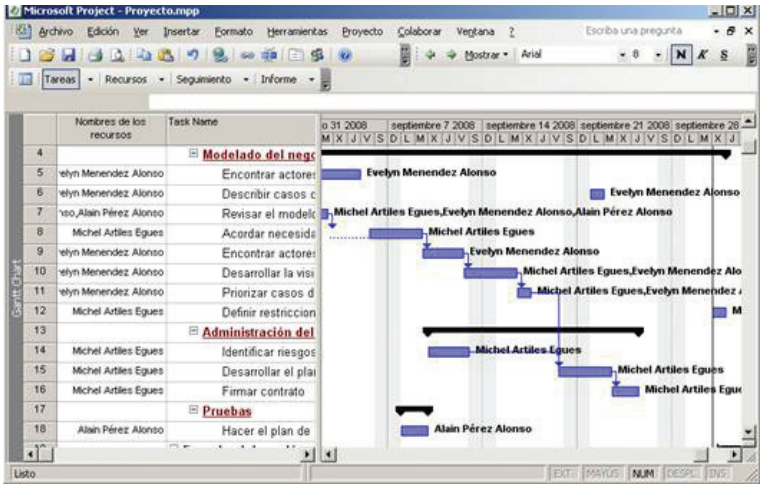
En general, Microsoft Project es un ***software* de administración de proyectos** que básicamente se usa para dar seguimiento al progreso, administrar presupuesto y analizar cargas de trabajo usando diagramas como: diagrama de Gantt y diagrama de Pert (diagrama de red).

Ademas, tiene las siguientes **características**:

1. Permite el aprendizaje rápido.

2. Posibilita la organización y seguimiento de las tareas y recursos.
3. Da la opción de comparar versiones de planes de proyectos.
4. Permite evaluar los cambios.
5. Sirve para realizar un seguimiento del rendimiento.
6. Facilita la opción de generar informes predefinidos.
7. Con él se pueden compartir planes de proyecto.
8. Permite la colaboración entre grupos de trabajo.
9. Es útil para la gestión de proyectos.

A continuación incluimos una **imagen que muestra la interfaz de Microsoft Project**:



Fuente: http://tiny.cc/o25gbx.

Rational Rose

Rational Rose es una **herramienta de producción y comercialización** que tiene la capacidad de crear, ver, modificar y manipular.

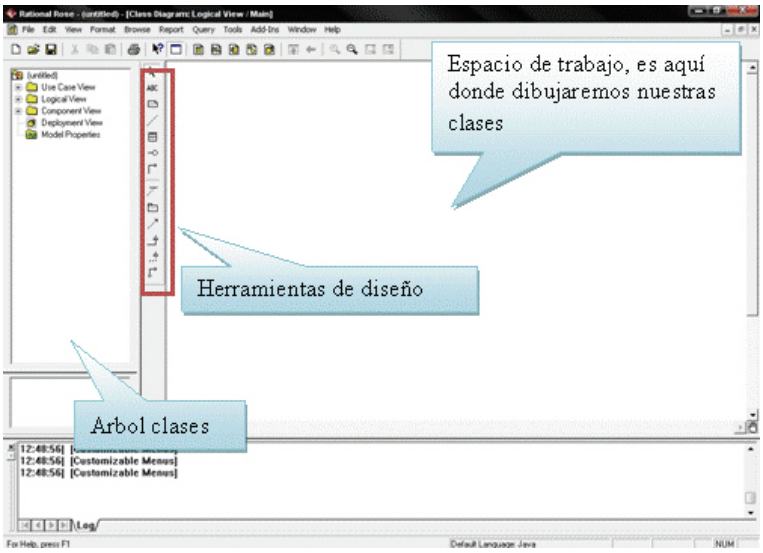
Conviene destacar que no es un *software* gratuito, que se usa para base de datos DB2, Oracle, SQL 92, SQL Server, Sybase y aplicaciones WEB.

Además, Rational Rose habilita asistentes para crear clases, se pueden aplicar los patrones de diseño así como admite la integración con otras herramientas de desarrollo (IDEs).

Para poder usar Rational Rose se necesita:

- Windows 2000 Professional, Service Pack 4.
- Windows XP Professional, Service Pack 2.
- Windows 2000 and 2003 Server and Advanced Server, Service Pack 3 and 4.
- Windows Vista.
- Linux.

Seguidamente se puede ver una **imagen que muestra la interfaz de Rational Rose**:



Fuente: http://tiny.cc/g45gbx.

El siguiente enlace nos muestra un monográfico sobre las herramientas CASE en el proceso de desarrollo del *software*: http://tiny.cc/f75gbx.

Existen dos módulos integrados para elaborar diagramas de comportamiento, tales como son Microsoft Project y Rational Rose.

6.4. CREACIÓN DE CÓDIGO A PARTIR DE DIAGRAMAS DE CLASES.

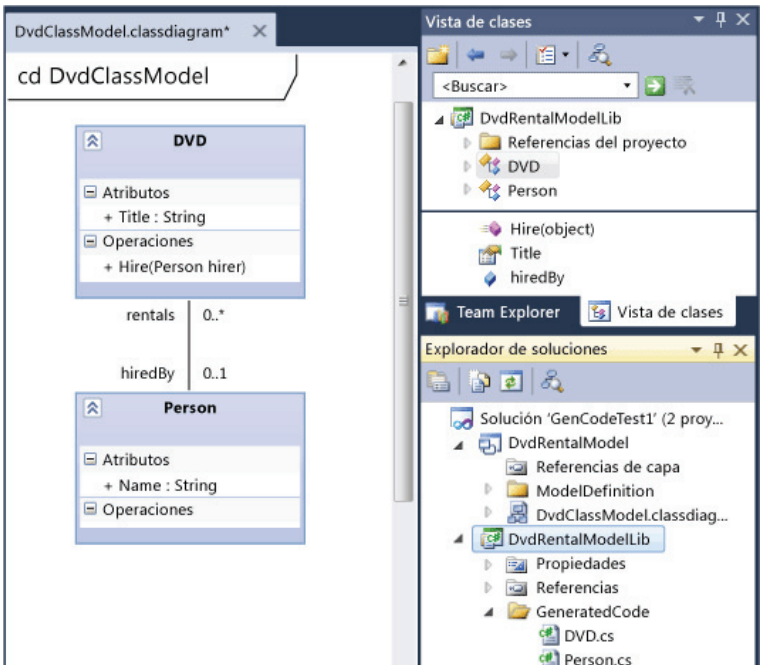
Para crear código a partir de diagramas de clases, se usará el *software* Feature Pack de Visual Studio 2010 el cual **permite crear código a partir de elementos UML en Visual Studio 2010 Ultimate mediante el comando Generar código**.

El *software* anterior:

- Puede modificar y extender este comportamiento si modifica o copia las plantillas de texto que generan código.
- Puede especificar un comportamiento diferente para los tipos contenidos en los diferentes paquetes de su 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.

La siguiente pantalla nos **muestra su interfaz principal**:



Fuente: http://tiny.cc/1d6gbx.

Usar el comando Generar código

En general, los pasos para usar el comando Generar código son:

1. Generar un **archivo independiente** para cada elemento.
2. Crear **un modelo de UML** que contenga clases.
3. Se **crean archivos que contienen el código de C#**.
4. Aplicar al modelo **algunas pruebas de validación** para asegurarse de que se puede traducir a C#. Si **se produce un error en las pruebas**, se muestra un mensaje de error y no se realiza la generación de código.
5. Si ha creado un **comando de menú de validación**, no se genera código para ningún elemento cuyo comando de validación produzca errores.

Transformación de generación de código predeterminado

En la transformación de generación de código predeterminado, hay que tener en cuenta las siguientes **consideraciones**:

1. Se resumen los **resultados obtenidos del comando Generar código**, salvo que se personalice el comando.
2. Se **genera un tipo de C# para cada tipo seleccionado en el modelo de UML**. Si el tipo UML está incluido en un paquete, el tipo de C# generado se coloca dentro de un espacio de nombres y el archivo se guarda en una carpeta que tiene el mismo nombre que el espacio de nombres.
3. Se **genera una propiedad de C# para cada Attribute** de una clase UML.
4. Se **genera un método de C# para cada Operation** de un tipo UML.
5. Se **genera un campo de C# para cada asociación navegable** en la que participa la clase.

6. Si **agrega un estereotipo a cada tipo UML**, puede controlar más propiedades del tipo de C# generado.

Modificar el código generado

Para modificar el código generado, hay que **tener en cuenta** que:

1. Varía según las propiedades de cada tipo, atributo u operación. Por ejemplo, si establece la propiedad Is Abstract (abstracto) de una clase en true, aparecerá la palabra clave *abstract* en la clase generada. Si establece la Multiplicity (multiplicidad) de un atributo en 0.. *, la propiedad generada tendrá un tipo IEnumerable<>.
2. Cada estereotipo **proporciona varias propiedades adicionales** las cuales se establecen seleccionando la clase u otro elemento en el diagrama. En la ventana Propiedades, hay que expandir Stereotypes (Estereotipos) y, después el estereotipo de C#, por ejemplo, Clase de C.
3. Cada atributo y operación tiene también propiedades de estereotipo que puede establecer.

En el siguiente enlace se puede ampliar información sobre los temas que estamos abordando en esta unidad:
http://msdn.microsoft.com/es-es/library/ff657795%28v=vs.100%29.aspx.

Personalizar el comando Generar código

Para personalizar el **comando Generar código** hay que **tener en cuenta** las siguientes consideraciones:

1. Funciona transformando los elementos del modelo mediante un conjunto de plantillas de texto, las cuales se especifican en un conjunto de enlaces de plantilla de texto donde éstos enlaces especifican, qué plantilla se debería aplicar, dónde se debería colocar el resultado generado y otros parámetros del comando Generar código.
2. Para inspeccionar los enlaces de plantilla adjuntados a un elemento de modelo, hay que hacer clic en los puntos suspensivos [...] en la propiedad Text Template Bindings (enlaces de plantilla de texto) en la ventana **Propiedades**.
3. Podemos aplicar más de una plantilla a un elemento modelo así como podemos crear más de un archivo a partir de cada elemento del modelo.
4. Para adjuntar los enlaces de plantilla de texto a un paquete o a la raíz del modelo, accedemos al **Explorador de modelos UML**, pulsamos sobre el botón secundario en el elemento del modelo y después entramos en Propiedades.
5. En la **ventana Propiedades**, nos vamos al botón de puntos suspensivos ([…]) en la propiedad Text Template Bindings (enlaces de plantilla de texto).
6. Aparece el cuadro de diálogo Text Template Bindings y pulsamos sobre “Agregar” para **crear un nuevo enlace de plantilla de texto** o nos vamos a un enlace existente para editarlo.
7. Las propiedades del enlace de plantilla de texto se ponen en el cuadro de diálogo, y se puede adjuntar tantos enlaces a un elemento como queramos.

Escribir una plantilla de texto

Para escribir una plantilla de texto, se debe comenzar por **modificar las copias de las plantillas estándar**. Para ello, podemos copiar las plantillas de los sitios web siguientes:

- Feature Pack de visualización y modelado de Visual Studio 2010: http://msdn.microsoft.com/es-es/vstudio/ff655021.aspx.
- Feature Pack 2 de Visual Studio 2010: http://bit.ly/1hAN7YI.

¿Qué es una **plantilla de texto**? Es un prototipo del archivo resultante y contiene el texto resultante y el código de programa que lee el modelo.

Veamos las siguientes **consideraciones**, que hay que tener en cuenta, para usar una **plantilla de texto**:

1. Para **navegar por el modelo de UML** hay que usar la API (funciones de librerías) de UML.
2. Para utilizar las plantillas con el comando Generar código, hay que incluir la **directiva de modelado**.
3. Para **depurar una plantilla de texto**, hay que poner dentro de un segmento de instrucción: System.Diagnostics.Debugger.Launch();
4. Podemos establecer la extensión de nombre de archivo del resultado en la directiva Output, una directiva en cada plantilla de texto.
5. Hay que usar la directiva Assembly para la utilización de ensamblados, como por ejemplo:

```
<#@ Assembly Name="%ProgramFiles%\Microsoft Visual Studio
10.0\Common7\IDE\PublicAssemblies\Microsoft.VisualStudio.
ArchitectureTools.Extensibility.dll" #>
```

6. Para usar los espacios de nombres como System se importan automáticamente al código de programa, para otros espacios usamos la directiva Import como por ejemplo:

```
<#@ Import Namespace="Microsoft.VisualStudio.Uml.Classes" #>
<#@ Import Namespace="Microsoft.VisualStudio.
ArchitectureTools.Extensibility.Uml" #>
```

7. Usamos la **directiva Include** para hacer referencia al texto de otro archivo teniendo en cuenta que el comando Generar código ejecuta las partes de la plantilla encerradas entre corchetes <# ... #>.

El siguiente enlace explica cómo crear diagramas de clases UML desde el código: http://msdn.microsoft.com/es-es/library/ff657806.aspx.

Para crear código a partir del diagrama de clases se usa Visual Studio 2010 Ultimate que permite crear código a partir de elementos UML.

6.5. GENERACIÓN DE DIAGRAMAS DE CLASES A PARTIR DE CÓDIGO (INGENIERÍA INVERSA).

Podemos agregar clases de C# o espacios de nombres de Explorador de arquitectura o gráficos de dependencias a un diagrama de clases UML, así como podemos agregar clases de C# de Explorador de soluciones.

Para **agregar clases desde código de programa** a un modelo UML tenemos que abrir un proyecto de C#:

1. Crear un proyecto de modelado UML.
2. Agregar un diagrama de clases UML al proyecto de modelado.
3. En el menú de **Arquitectura**, elija **Nuevo diagrama**.
4. En el cuadro de diálogo Agregar nuevo diagrama, seleccionar diagrama de clases UML.
5. Abrir el Explorador de arquitectura: en el menú de Arquitectura, elegir Ventanas, Explorador de arquitectura.
6. Arrastrar los espacios de nombres o tipos del Explorador de arquitectura a la superficie del diagrama de clases UML así como **arrastrar espacios de nombres** o tipos de gráficos de dependencia.

En el siguiente enlace encontraréis un artículo sobre ingeniería inversa de modelos UML a partir de código Java, C# y Visual Basic.NET: http://tiny.cc/1d6gbx.

Usaremos el Visual Studio Ultimate para Generar código a partir de diagramas de clases.

6.6. RESUMEN.

En esta unidad, hemos podido ver todo el **proceso de elaboración de los diagramas de clases de modo que lo principal que no hay que olvidar es**:

1. En un diagrama de clase intervienen básicamente las **clases y las relaciones**.

    - Clase son la **descripción de las características comunes de esos objetos**.
    - Las relaciones son las distintas uniones entre las clases. Los tipos de relaciones son **asociación, dependencia y generalización**.

2. Los módulos integrados para elaborar diagramas de clase son el **Microsoft Project y el Rational Rose**.

    - Microsoft Project es un *software* usado para el **desarrollo de planes, asignación de recursos a tareas, facilitar el seguimiento de proyectos, administrar presupuesto y analizar cargas de trabajo**.
    - Rational Rose es una **herramienta de producción y comercialización** establecidas por Rational Software Corporation (actualmente parte de IBM).

3. Para la creación de código a partir de diagramas de clases usamos el **Feature Pack de Visual Studio 2010**.

4. Para generación diagramas de clases a partir de código usamos el **Visual Studio Ultimate**.