

Correspondencias UML – Java – C#

UML es un lenguaje de modelado visual, Java y C# son lenguajes de programación textuales. UML es más rico que los lenguajes de programación en el sentido que este ofrece medios de expresión más abstractos y más poderosos.

Sin embargo, existe por lo regular una forma privilegiada de traducir los conceptos UML en declaraciones Java o C# (en esta propuesta tratamos de respetar las convenciones de nombrado de cada lenguaje).

Este documento propone una síntesis de las correspondencias más importantes entre los conceptos de modelización UML y el mundo de la implementación en un lenguaje orientado a objetos.

En esta síntesis, solo se presentan los lenguajes C# y Java, pero es común encontrar literatura para otros lenguajes como PHP 5, Python o C++.


La estructura estática

Los conceptos estructurales (o estáticos) como las clases o las interfaces son fundamentales tanto en UML mismo, como en los lenguajes Java y C#. Estos están representados en UML en los diagramas de clases, y constituyen el esqueleto de un código orientado a objetos.

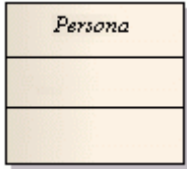
CLASE

La clase es el concepto fundamental de toda la tecnología de objetos. La palabra clave correspondiente también existe en Java y C#, y obvio en los demás lenguajes orientados a objetos.

Por defecto cada clase UML viene siendo un archivo .java (Java) o .cs (C#), pero el almacenamiento puede ser diferente si el diseñador lo desea.



UML	Java
	<pre>public class Catalogo { ... }</pre>
	C#
	<pre>public class Catalogo { ... }</pre>

Una clase abstracta es simplemente una clase que no se instancia directamente, sino que representa una abstracción pura que nos sirve para factorizar las propiedades. Esta se denota en *itálica*. La palabra clave `abstract` es igualmente válida en Java y en C#.

UML	Java
	<pre>public abstract class Persona { ... }</pre>
	C#
	<pre>public abstract class Persona { ... }</pre>

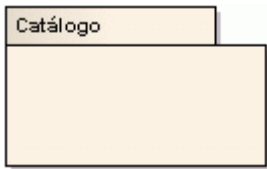
INTERFACE

La noción UML de interface (representada bajo las formas gráficas) se traduce por la palabra clave correspondiente de la misma forma tanto en Java como en C#.

UML	Java
	<pre>public interface IVisualizable { public void mostrar(); }</pre>
	C#
 IVisualizable	<pre>public interface IVisualizable { void mostrar(); }</pre>

PAQUETE

El paquete, que como sabemos agrupa clases o interfaces, existe tanto en Java como en C#, pero con una sintaxis diferente (hay que tener cuidado con las reglas de nombrado: minúsculas en Java).

UML	Java
	<pre>package catalogo; ... </pre>
	C#
	<pre>namespace Catalogo { ... }</pre>


ATRIBUTO

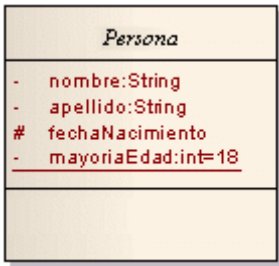
Los atributos vienen siendo variables en Java o en C#. Su tipo es ya sea un tipo primitivo (int, etc.) o ya sea una clase proporcionada por la plataforma (String, Date, etc.). Es importante no olvidar en este caso la directiva de importación del paquete correspondiente.

La visibilidad de los atributos se muestra haciéndolos anteceder un + para los públicos, # para los protegidos y – para los privados.

Los atributos de clase en UML vienen siendo los miembros estáticos en Java o en C#.

Los atributos de tipo referenciado a otro objeto o a una colección de objetos se discutirán posteriormente (ver Asociación).

UML	Java
 <pre> classDiagram class Catálogo { - nombre:String - fechaCreacion:Date } </pre>	<pre> import java.util.Date; public class Catálogo { private String nombre; private Date fechaCreacion; } </pre>
	C# <pre> using System; public class Catálogo { private string nombre; private DateTime fechaCreacion; } </pre>

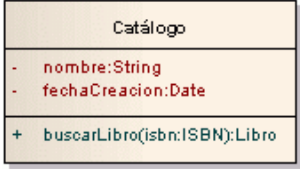
UML	Java
 <pre> classDiagram class Persona { - nombre:String - apellido:String # fechaNacimiento - mayoríaEdad:int=18 } </pre>	<pre> abstract public class Persona { private String nombre; private String apellido; protected Date fechaNacimiento; private static int mayoríaEdad=18; } </pre>
	C# <pre> abstract public class Persona { private string nombre; private string apellido; protected DateTime fechaNacimiento; private static int mayoríaEdad=18; } </pre>

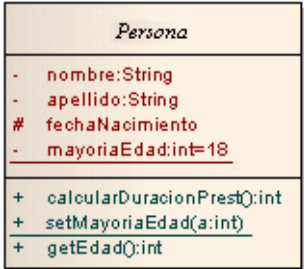
OPERACIÓN

Las operaciones vienen siendo los métodos en Java y en C# (pero en el caso de PHP 5, son las function).

Su visibilidad está definida con las mismas convenciones que los atributos (hay que tener cuidado con los parámetros en Java, ya que no soporta las dirección in y return de UML).

Las operaciones de clase vienen siendo los métodos estáticos; las operaciones abstractas (en *itálica*) se traducen por la palabra clave correspondiente en Java, PHP o en C#.

UML	Java
 <pre> classDiagram class Catalogo { - nombre:String - fechaCreacion:Date + buscarLibro(isbn:ISBN):Libro } </pre>	<pre> public class Catalogo { private String nombre; private Date fechaCreacion; public Libro buscarLibro(ISBN isbn) { ... } ... } </pre>
	C#
	<pre> public class Catalogo { private string nombre; private DateTime fechaCreacion; public Libro buscarLibro(ISBN isbn) { ... } ... } </pre>

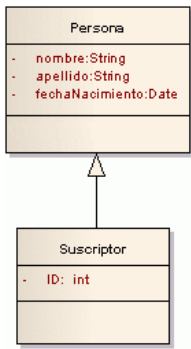
UML	Java
 <pre> classDiagram class Persona { - nombre:String - apellido:String # fechaNacimiento - mayoriaEdad:int=18 + calcularDuracionPrest():int + setMayoriaEdad(a:int) + getEdad():int } </pre>	<pre> abstract public class Persona { private String nombre; private String apellido; protected Date fechaNacimiento; private static int mayoriaEdad=18; public abstract int calcularDuracionPrest(); public static void setMayoriaEdad(int mEdad) { } Public int getEdad() { ... } } </pre>
	C#
	<pre> abstract public class Persona { private string nombre; private string apellido; protected DatTimeee fechaNacimiento; private static int mayoriaEdad=18; public abstract int calcularDuracionPrest(); public static void setMayoriaEdad(int mEdad) { ... } public int getEdad() { ... } } </pre>

Las relaciones

Las relaciones UML entre conceptos estáticos son muy ricas y no se traducen todas de manera simple mediante una palabra clave en los lenguajes orientados a objetos.

GENERALIZACION

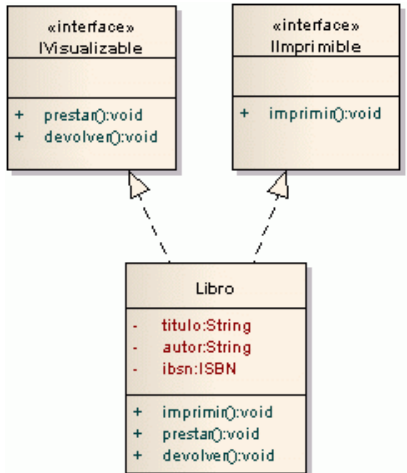
El concepto UML de generalización se traduce directamente por el mecanismo de herencia en los lenguajes orientados a objetos. La sintaxis es diferente en Java y en C#. Como referencia, el mecanismo de herencia de PHP 5 es muy parecido a Java.

UML	Java
 <pre> classDiagram class Persona { nombre:String apellido:String fechaNacimiento:Date } class Suscriptor { ID:int } Persona < -- Suscriptor </pre>	<pre> public class Suscriptor extends Persona { private int ID; } </pre>
	<p>C#</p> <pre> public class Suscriptor : Persona { private int ID; } </pre>

REALIZACION

Una clase UML puede implementar varias interfaces. Contrariamente a C++, los lenguajes Java, C# y PHP 5 proponen directamente este mecanismo, pero no de forma directa la herencia múltiple entre clases.

C# utiliza la sintaxis de C++ para la herencia. La misma palabra clave se utiliza para la herencia de implementación y de interface, de forma distinta a Java y PHP 5 que especifican *extends* e *implements*.

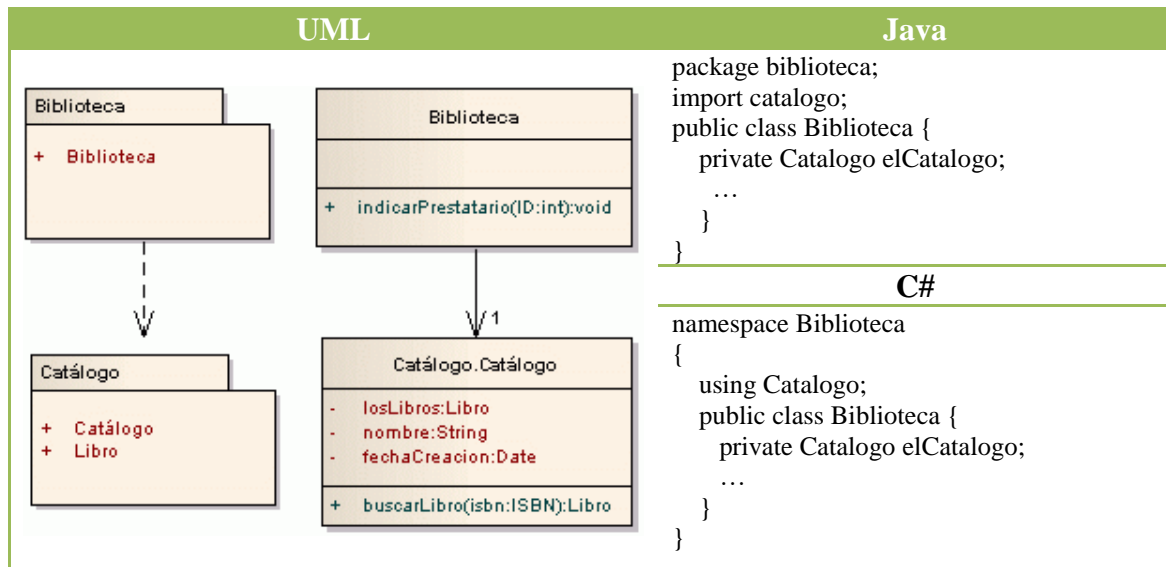
UML	Java
 <pre> classDiagram class IVisualizable { <<interface>> +prestar():void +devolver():void } class IImprimible { <<interface>> +imprimir():void } class Libro { -titulo:String -autor:String -isbn:ISBN +imprimir():void +prestar():void +devolver():void } IVisualizable <.. Libro IImprimible <.. Libro </pre>	<pre> public class Libro implements IVisualizable, IImprimible { private String titulo; private String autor; private ISBN isbn; public void imprimir() { ... } public void prestar() { ... } public void devolver() { ... } } </pre>
	<p>C#</p> <pre> public class Libro : IVisualizable, IImprimible { private string titulo; private string autor; } </pre>

```
private ISBN isbn;
public void imprimir() {
    ...
}
public void prestar() {
    ...
}
public void devolver() {
    ...
}
}
```

DEPENDENCIA

La dependencia es un concepto muy general en UML. Una dependencia entre una clase A y una clase B existe por ejemplo si A posee un método que toma como parámetro una referencia sobre una instancia de B, o si A utiliza una operación de la clase B. No existe una palabra clave correspondiente en Java, PHP 5 o en C#.

La dependencia entre paquetes se traduce de manera indirecta por medio de las directivas de importación o de uso en Java y en C#.



ASOCIACION

Las asociaciones navegables que se traducen a código objeto dependen notablemente de la multiplicidad de la extremidad en cuestión, pero igualmente de la existencia de una restricción `{ordered}` o de un calificativo.

Una asociación navegable con una multiplicidad 1 se traduce mediante una variable de instancia, es decir como un atributo, pero con un tipo referenciado hacia una instancia de clase del modelo en lugar de un tipo simple.

Una multiplicidad «*» se va a traducir mediante un atributo de tipo colección de referencia de objetos en lugar de una simple referencia hacia un objeto.

La dificultad consiste en elegir la colección adecuada entre las muchas propuestas de clases de base que proponen los diferentes lenguajes, en este caso Java y C#. O también si es posible crear arreglos de objetos, lo cual no siempre es una buena solución.

En este caso, se prefiere más bien recurrir a las colecciones, entre las cuales se tienen las siguientes (que de hecho son de las más utilizadas):


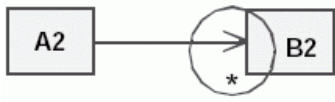
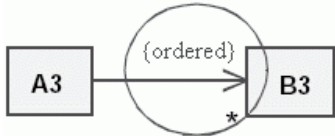

- En Java: ArrayList (antiguamente se llamaba Vector) y HashMap (llamado anteriormente Hashtable). Se utiliza ArrayList si se debe respetar un orden y recuperar los objetos a partir de un índice entero; se utiliza HashMap si se desean recuperar los objetos a partir de una llave arbitraria (En el JDK 1.5 se introdujeron las colecciones tipeadas llamadas «generics»).
- En C#: ArrayList, SortedList y Hashtable. Se utiliza ArrayList si se debe respetar un orden y recuperar los objetos a partir de un índice entero; se utiliza Hashtable o SortedList si se desean recuperar los objetos a partir de una llave arbitraria.
A partir de C# 2.0, se utilizará más bien List y Dictionary.

La siguiente figura muestra estos detalles.


Una asociación bidireccional se traduce naturalmente por un par de referencias, una en cada clase involucrada en la asociación. Los nombres de los roles en las extremidades de una asociación sirven para nombrar las variables de tipo referencia.

Hay que tener cuidado, en los lenguajes orientados a objetos, no es posible precisar que dos referencias pertenecen a dos clases que corresponden a la misma asociación y que una es opuesta a la otra.

El concepto de asociación UML no existe de hecho verdaderamente en los lenguajes orientados a objetos!!.

UML	Java	C#
	<pre>public class A1 { private B1 laB1; ... }</pre>	<pre>public class A1 { private B1 laB1; ... }</pre>
	<pre>public class A2 { private B2 lasB2[]; ... }</pre>	<pre>public class A2 { private B2[] lasB2; ... }</pre>
	<pre>public class A3 { private List<B3> las B3 = new ArrayList<B3>(); ... }</pre>	<pre>public class A3 { private IList lasB3 = new List<B3>(); ... }</pre>
	<pre>public class A4 { private Map<Q,B4> las B4 = new HashMap<Q,B4>(); ... }</pre>	<pre>public class A4 { private IDictionary <Q,B4> las B4 = new Dictionary <Q,B4>(); ... }</pre>

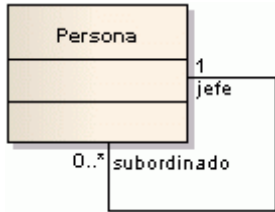
A continuación se muestran los detalles de la asociación bidireccional.

UML	Java	C#
	<pre>public class Hombre { private Mujer esposa; ... } public class Mujer { private Hombre esposo; ... }</pre>	<pre>public class Hombre { private Mujer esposa; ... } public class Mujer { private Hombre esposo; ... }</pre>

La siguiente figura muestra una asociación reflexiva que se traduce por una referencia sobre un objeto de la misma clase.

AGREGACION y COMPOSICION

La agregación es un caso particular de asociación no simétrica que expresa una relación de contenido. Las agregaciones no necesitan estar nombradas: implícitamente estas significan «contienen», «están compuestas de».

UML	Java	
 <pre> classDiagram class Persona { +jefe } class subordinado { +subordinado } Persona "1" -- "0..*" subordinado </pre>	<pre> public class Persona { private Persona subordinado[]; private Persona jefe; ... } </pre>	
	<th>C#</th>	C#
	<pre> public class Persona { private Persona[] subordinado; private Persona jefe; ... } </pre>	

La semántica de las agregaciones no es fundamentalmente diferente de las de las asociaciones simples, estas se traducen por lo tanto como se indicó anteriormente en Java o en C#.

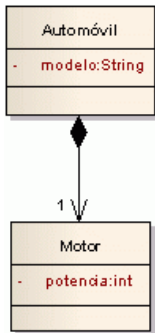
La única restricción es que una asociación no puede contener marca de agregación más que en una de las extremidades.

Una composición es una agregación más fuerte que implica:

- una parte no puede pertenecer más que a un solo compuesto (agregación no compartida);
- la destrucción del compuesto resulta en la destrucción de todas sus partes (el compuesto es responsable del ciclo de vida de las partes).

En ciertos lenguajes orientados a objetos, por ejemplo C++, la composición implica la propagación del destructor, pero esto no se aplica a Java ni a C#.

Por el contrario, la noción de clase anidada puede ser muy interesante para traducir la composición. Sin embargo esto, es en absoluto obligatorio.

UML	Java	C#
 <pre> classDiagram class Automóvil { +modelo:String } class Motor { +potencia:int } Automóvil "1" *-- "1" Motor </pre>	<pre> public class Automovil { private String modelo; private Motor motor; private static class Motor { private int potencia; } ... } </pre>	<pre> public class Automovil { private string modelo; private Motor motor; private class Motor { private int potencia; } ... } </pre>

CLASE DE ASOCIACION

Se trata de una asociación promovida al rango de la clase. Tiene las características de una asociación y de una clase y puede por lo tanto tener atributos que se aprecian para cada vínculo. Este concepto UML avanzado no existe en los lenguajes de programación orientados a objetos, por lo tanto podríamos traducirlo, transformando en clase normal y agregando las variables de tipo referencia.

