

## 5.B. UML.

Sitio: [ALONSO DE AVELLANEDA](#)

Curso: Entornos de desarrollo

Libro: 5.B. UML.

Imprimido por: Iván Jiménez Utiel

Día: martes, 17 de marzo de 2020, 01:13

## Tabla de contenidos

[1. Introducción.](#)

[2. Familiarizándonos con algunos conceptos UML.](#)

[2.1. Notación.](#)

[2.2. Modelos y herramientas.](#)

[2.3. Método.](#)

[3. Tipos de diagramas UML.](#)

[4. Herramientas para la elaboración de diagramas UML.](#)

[4.1. UMLet.](#)

[5. Ingeniería inversa.](#)

## 1. Introducción.

**Unified Modeling Language** o **Lenguaje Unificado de Modelado** es un conjunto de herramientas que permiten modelar 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 visualizar el producto en esquemas o diagramas estandarizados denominados **modelos** o **vistas** 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 y con el cliente.
- Hay estructuras que van más allá de lo representable en un lenguaje de programación, como las que hacen referencia a la arquitectura del sistema. 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 construir **modelos precisos, no ambiguos y completos**.

Además **UML** puede conectarse a lenguajes de programación mediante **ingeniería directa e inversa**.

## 2. Familiarizándonos con algunos conceptos UML.

A continuación se describen algunos de los términos típicamente usados en **UML**. Se acompaña de un ejemplo relacionado con la creación de una melodía musical.

## 2.1. Notación.

Una **notación** es un conjunto de **símbolos** y **técnicas para combinarlos**, que en el contexto **UML**, permiten crear **diagramas** normalizados. Estas representaciones posibilitan al analista o desarrollador describir el comportamiento del sistema (análisis) y los detalles de una arquitectura (diseño) de forma no ambigua.








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

Las notaciones UML deben ser independientes de los lenguajes de programación.

Un **Diagrama** es una representación gráfica de una colección de elementos de modelado (**modelo**), a menudo dibujada como un grafo con vértices conectados por arcos (**notación**).

- **Notación musical-1.** La notación musical formal considera términos como pentagrama; notas redonda, blanca, negra, corchea, semicorchea, fusa, semifusa; clave de sol, fa, do ...

*Tipos de silencios empleados en música*

						
Redonda	Blanca	Negra	Corchea	Semicorchea	Fusa	Semifusa
(Unidad)	(1/2)	(1/4)	(1/8)	(1/16)	(1/32)	(1/64)

- **Notación musical-2.** Otra alternativa, útil para inexpertos, podría ser escribir en una/varias líneas la secuencia de notas que forman una melodía.  
**DO - RE - MI - FA - SOL - LA - SI.**

## 2.2. Modelos y herramientas.

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

Volviendo al ejemplo musical, la melodía puede mostrarse desde diferentes vistas.

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



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

DO - DO - RE - DO - FA - MI - DO - DO - RE - DO - SOL - FA - FA - LA - DO - LA - FA - MI - RE

**Vista - 3.** La interpretación de la melodía sería otra vista distinta. Incluso se pueden considerar diferentes vistas en función del compás, instrumento ...



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

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

Una **herramienta** es el soporte automático de una notación. En nuestro ejemplo hablaríamos de un lápiz, un cuaderno musical, un programa de ordenador, un órgano ....

Para el desarrollo de diagramas **UML** en este curso se usará la aplicación/herramienta **UMLet**.

### 2.3. Método.

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

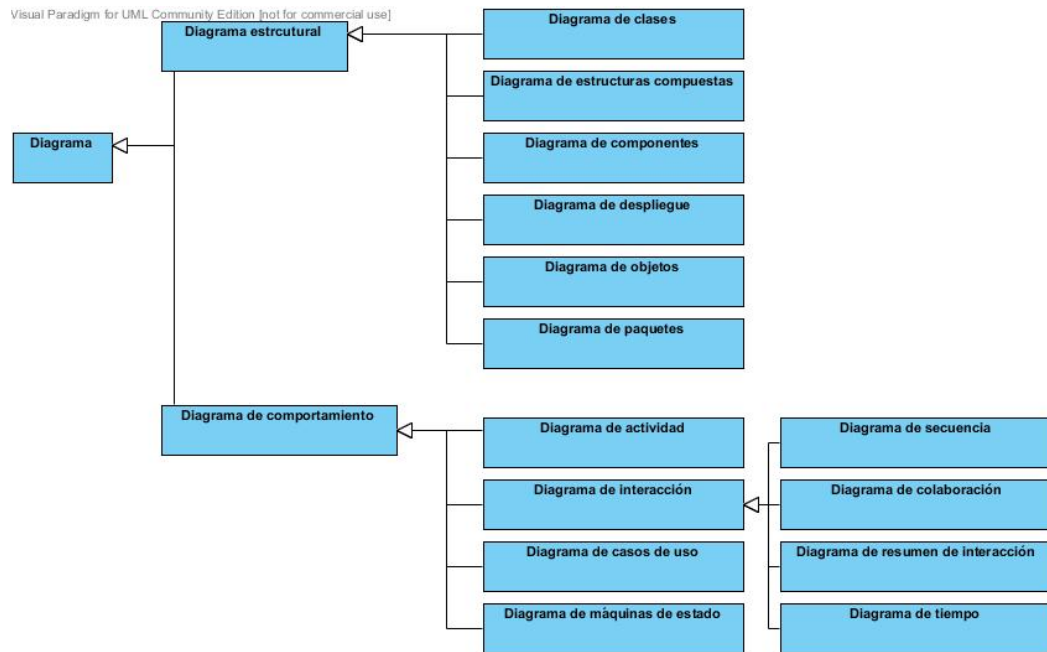
### 3. Tipos de diagramas UML.

UML define un sistema como una **colección de modelos** que describen sus diferentes perspectivas. Los modelos se implementan en una serie de **diagramas** que son representaciones gráficas de una colección de elementos de modelado, a menudo dibujado como un grafo con **arcos** (relaciones) y **vértices** (otros elementos del modelo).

Los diagramas UML 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.

En la siguiente imagen aparecen todos los diagramas organizados según su categoría.



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 cuando sea necesario.

#### Diagramas estructurales.

- **Diagrama de clases.** Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones.
- **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.
- **Diagrama de componentes.** Especifica la organización lógica de la implementación de una aplicación, indicando sus componentes, sus interrelaciones, interacciones y sus interfaces públicas y las dependencias entre ellos.
- **Diagrama de despliegue.** Representa la configuración del sistema en tiempo de ejecución. Aparecen los nodos de procesamiento y sus componentes. Exhibe la ejecución de la arquitectura del sistema. Incluye nodos, ambientes operativos tanto de hardware como de 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.
- **Diagrama de estructuras compuestas.** Muestra la estructura interna de una clase, e incluye los puntos de interacción de esta clase con otras partes del sistema.
- **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.

#### Diagramas de comportamiento.

- **Diagrama de casos de uso.** Representa 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.
- **Diagrama de estado de la máquina.** Describe el comportamiento de un sistema dirigido por eventos. En el que aparecen los estados que puede tener un objeto o interacción, así como las transiciones entre dichos estados. También se denomina diagrama de estado, diagrama de estados y transiciones o diagrama de cambio de estados.
- **Diagrama de actividades.** Muestra 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.
- **Diagramas de interacción.**
  - **Diagrama de secuencia.** Representa la ordenación temporal en el paso de mensajes. Modela la secuencia lógica, a través del tiempo, de los



mensajes entre las instancias.

- **Diagrama de comunicación/colaboración.** Resalta la organización estructural de los objetos que se pasan mensajes. Ofrece las instancias de las clases, sus interrelaciones, y el flujo de mensajes entre ellas. Comúnmente enfoca la organización estructural de los objetos que reciben y envían mensajes.
- **Diagrama de interacción.** Muestra un conjunto de objetos y sus relaciones junto con los mensajes que se envían entre ellos. Cada nodo de actividad dentro del diagrama puede representar otro diagrama de interacción.
- **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.

## 4. Herramientas para la elaboración de diagramas UML.

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 (**what you see is what you get**), permiten documentar los diagramas e integrarse con otros entornos de desarrollo incluyendo la generación automática de código y en algunos casos, 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 absorbida por **IBM**.
- [UMLet](#): herramienta UML de código abierto y libre distribución. Dispone de un interfaz de usuario sencillo de utilizar.
- [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. Dispone de diferentes módulos para realizar desarrollo UML, diseñar bases de datos, realizar actividades de ingeniería inversa y diseñar con Agile. Es compatible con los IDEs Eclipse, Visual Studio .net, IntelliJDEA y NetBeans.
- [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










## 4.1. UMLet.

Para los **diagramas UML** que se van a realizar a lo largo del curso se va a usar la herramienta **UMLet**. Algunas de sus características son:

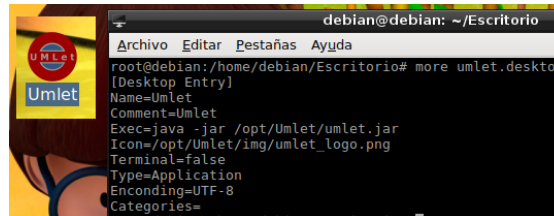
- Es un software gratuito.
- Es multiplataforma.
- Incluye un módulo para integrarse con Eclipse.
- Dispone de la versión web **UMLetino** <http://www.umletino.com/umletino.html>, que no precisa instalación.

La instalación de **UMLet** como herramienta de escritorio es muy sencilla:

- Descargar **UMLet standalone** del sitio oficial de descargas <https://www.umlet.com/changes.htm>, actualmente se encuentra en versión 14.3. El fichero descargado es *umlet-standalone-14.3.0.zip*, también disponible en la sección [recursos de contenidos](#).
- Descomprimir el fichero, el paquete proporciona un conjunto de ficheros bajo el directorio *uml* con ejecutables para diferentes plataformas. En particular, usaremos el archivo comprimido **Java umlet.jar** para lanzarlo desde nuestro entorno de trabajo **Debian**.

	custom_elements	08/11/2018 0:45	Carpeta de archivos	
	img	08/11/2018 0:45	Carpeta de archivos	
	lib	08/11/2018 0:45	Carpeta de archivos	
	palettes	08/11/2018 0:45	Carpeta de archivos	
	LICENCE	03/04/2016 15:14	Documento de tex	35 KB
	umlet	16/11/2017 10:39	Archivo DESKTOP	1 KB
	Umlet	28/05/2018 19:20	Aplicación	59 KB
	umlet	05/08/2018 11:26	Executable Jar File	43 KB
	umlet.sh	06/04/2018 9:59	Archivo SH	3 KB

- Copiar la carpeta descomprimida en alguna ruta del sistema de archivos, por ejemplo */opt* y crear un acceso directo en el escritorio **umlet.desktop** con la información mostrada en la siguiente figura.



Utilizando **UMLet** dibujaremos diferentes diagramas a lo largo de este tema y del siguiente.

## 5. Ingeniería inversa.

La **ingeniería inversa** se define como el proceso de analizar código, documentación y comportamiento de una aplicación para extraer información del diseño. El sistema en estudio no se modifica, 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:** consiste en estudiar la lógica interna de las interfaces para obtener los modelos y especificaciones que sirvieron para su construcción, con objeto de tomarlas como punto de partida en procesos de ingeniería directa que permitan su actualización.