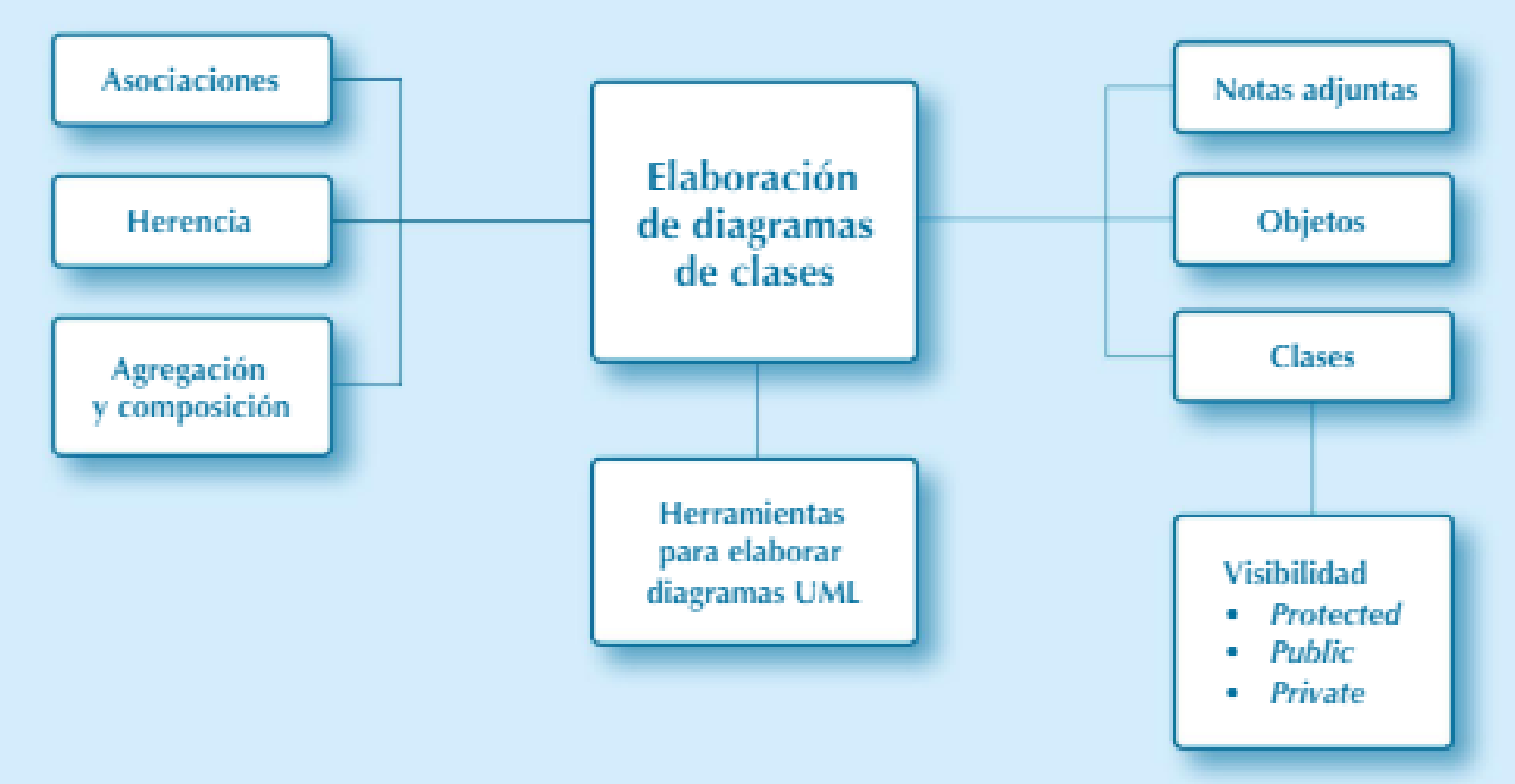
UNIDAD 6-1. ELABORACIÓN DE DIAGRAMAS DE CLASE

VICENT MARTÍ

OBJETIVOS

- En esta unidad se pretende que se aprenda a realizar y a comprender diagramas de clases, que son las herramientas UML más utilizadas en los proyectos de software. UML es el lenguaje que se utiliza en los proyectos orientados a objetos (OO) software.
- Reconocer y manejar las diferentes simbologías como asociaciones, herencia, agregación , etc.
- Realizar diferentes prácticas utilizando los diagramas UML, para consolidar el aprendizaje.

MAPA CONCEPTUAL



CONTENIDO

1. Introducción a la Orientación a Objetos
2. Conceptos de Orientación a Objetos
3. Ventajas de la Orientación a Objetos
4. UML
5. Notación del diagrama de clases
6. Herramientas para elaborar diagramas UML

1. INTRODUCCIÓN A LA ORIENTACIÓN A OBJETOS I

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. Para ello 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 (1) que actuaban sobre estructuras de datos (2), por eso a este tipo de programación se le llama programación estructurada (3).

(1) Funciones: *conjunto de sentencias escritas en un lenguaje de programación que operan sobre un conjunto de parámetros y producen un resultado.*
(2) 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.*
(3) 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.*

1. INTRODUCCIÓN A LA ORIENTACIÓN A OBJETOS II

Enfoque orientado a objetos (OO).
La orientación a objetos ha roto con el paradigma de la programación estructurada. Con este nuevo paradigma (4) el proceso se centra en simular los elementos de la realidad asociada al problema de la forma más cercana posible. La abstracción (5) 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**.

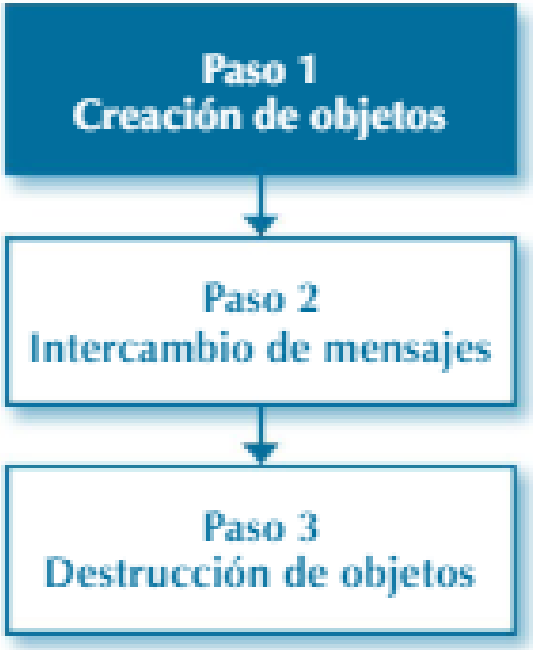
Las aplicaciones orientadas a objetos están formadas por un conjunto de objetos que interaccionan enviándose mensajes para producir resultados. Los objetos similares se abstraen en clases, se dice que un objeto es una instancia de una clase.

(4) Paradigma: *modelo o patrón aplicado a cualquier disciplina científica u otro contexto epistemológico.*
(5) 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.*

1. INTRODUCCIÓN A LA ORIENTACIÓN A OBJETOS III

Cuando se ejecuta un programa orientado a objetos ocurren tres sucesos:

Paso 1: los objetos se crean a medida que se necesitan.
Paso 2: 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.
Paso 3: 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 (OMG): http://www.omg.org/index.htm

2. CONCEPTOS DE ORIENTACIÓN A OBJETOS I

Como hemos visto la orientación a objetos (OO) 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**
- **Encapsulación**
- **Modularidad**
- **Principio de ocultación**
- **Polimorfismo**
- **Herencia**
- **Recolección de basura**

2. CONCEPTOS DE ORIENTACIÓN A OBJETOS II

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.

2. CONCEPTOS DE ORIENTACIÓN A OBJETOS III

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.

3. VENTAJAS DE LA ORIENTACIÓN A OBJETOS

- Permite desarrollar software en **mucho menos tiempo, con menos coste y de mayor calidad** gracias a la reutilización (1) 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.
- Se consigue **aumentar la calidad de los sistemas**, haciéndolos más extensibles (2) ya que es muy sencillo aumentar o modificar la funcionalidad de la aplicación modificando las operaciones.
- El software OO 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.
- La tecnología de objetos facilita la **adaptación al entorno y el cambio** haciendo aplicaciones escalables (3). Es sencillo modificar la estructura y el comportamiento de los objetos sin tener que cambiar la aplicación.

(1) **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.*

(2) **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.*

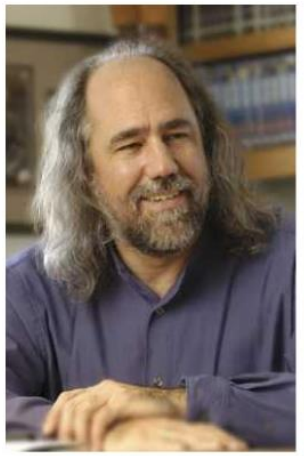
(3) **Escalables:** *propiedad deseable de un sistema, red o proceso que le permite hacerse más grande sin rehacer su diseño y sin disminuir su rendimiento*.

4. UML

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 James 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 visualizar el producto de su trabajo en diagramas estandarizados denominados modelos (1) que representan el sistema desde diferentes perspectivas.


grady booch


ivar jacobson


james rumbaugh

(1) 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.*

¿PORQUÉ ES ÚTIL MODELAR? I

Permite utilizar un lenguaje común que facilita la comunicación entre el equipo de desarrollo.

- Con UML podemos documentar todos los artefactos (1) de un proceso de desarrollo, 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 (2), 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.

(1) Artefactos: *información que es utilizada o producida mediante un proceso de desarrollo de software. Pueden ser artefactos un modelo, una descripción o un software.*

(2) Arquitectura del sistema: *conjunto de decisiones significativas acerca de la organización de un sistema software.*

¿PORQUÉ ES ÚTIL MODELAR? II

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

- **Ingeniería directa:** transformación de un modelo o un código a través de su traducción a un determinado lenguaje de programación.
- **Ingeniería inversa:** transformación del código en un modelo a través de su traducción desde un determinado lenguaje de programación.

Es importante apreciar como UML, permite poner las ideas de los desarrolladores en común utilizando un lenguaje específico, facilita la comunicación, que es algo esencial para que el desarrollo del software 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.

ELEMENTOS DE UN DIAGRAMA UML

UML define un sistema como una colección de modelos que describen sus diferentes perspectivas.

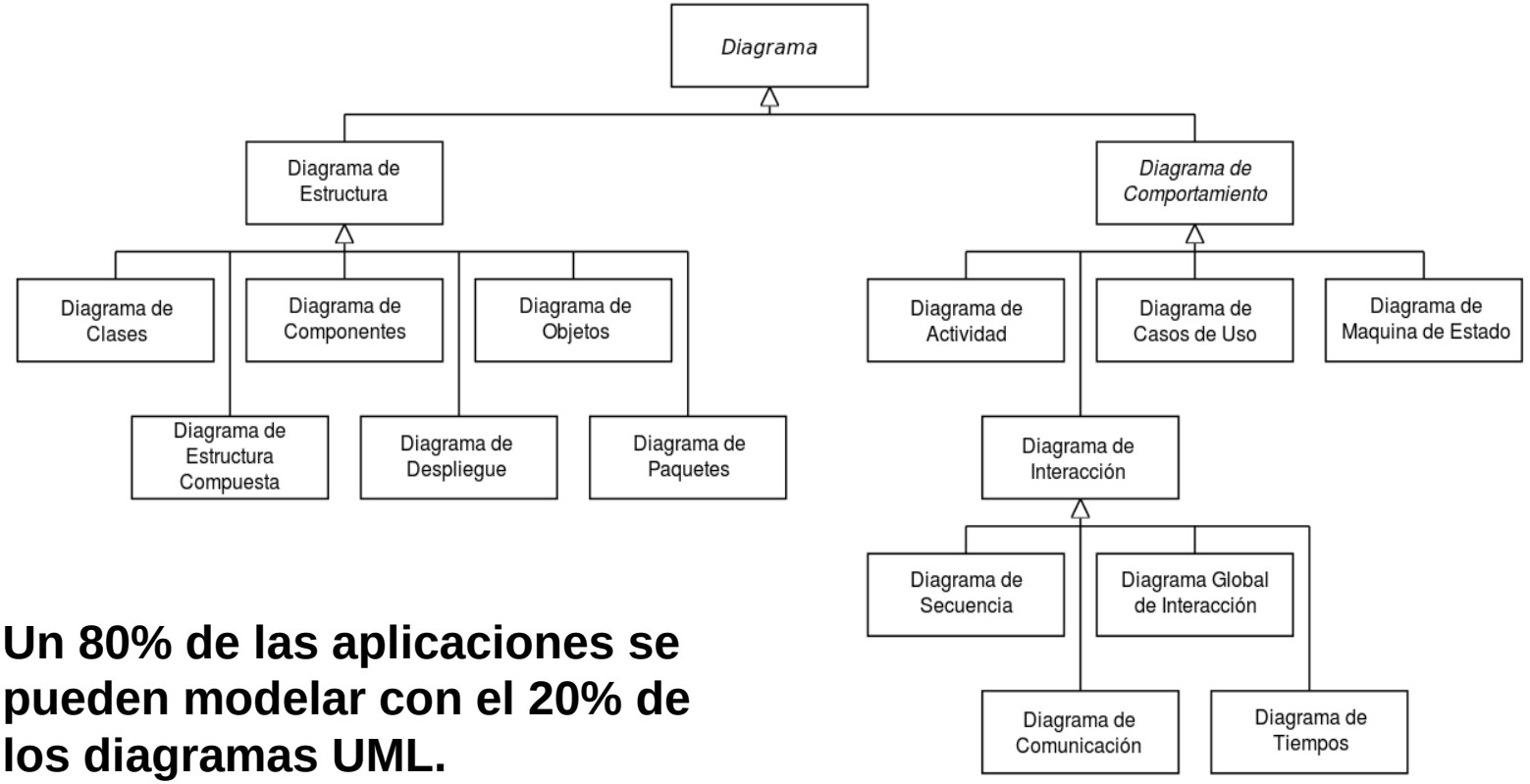
- Los modelos se implementan con una serie de diagramas que son representaciones gráficas de una colección de elementos de modelado, a menudo dibujado como un grafo conexo de arcos (relaciones) y vértices (otros elementos del modelo).

Un diagrama 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 cono un cuadro donde podemos escribir comentarios que nos aclaren algún concepto a representar.
- **Agrupaciones:** Se utilizan cuando modelamos sistemas grandes para facilitar su desarrollo por bloques.

TIPOS DE DIAGRAMAS UML I

Jerarquía de diagramas de UML 2.2. En UML 2.5 hay otro diagrama estructural más, el **diagrama de perfiles**. UML 2.5.1 fue lanzado en octubre de 2012 como una versión "En proceso" que fue formalmente liberada en junio de 2015 y su última revisión es de diciembre del 2017.



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

TIPOS DE DIAGRAMAS UML II

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.

No es necesario usar todos los diagramas, dependerá del tipo de aplicación a generar y del sistema, es decir, se debe generar sólo aquellos diagramas necesarios para el desarrollo.

DIAGRAMAS ESTRUCTURALES

Diagrama de clases: Muestra los elementos del modelo estático abstracto, y está formado por un conjunto de clases y sus relaciones. Prioridad ALTA.
Diagrama de objetos: Muestra los elementos del modelo estático en un momento concreto, está formado por un conjunto de objetos y sus relaciones. 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. 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. Se utiliza cuando tenemos sistemas distribuidos. 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. Prioridad BAJA.
Diagrama de paquetes: Exhibe cómo los elementos del modelo se organizan en paquetes, así como las dependencias entre esos paquetes. En sistemas de mediano o gran tamaño. 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. 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. 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. 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. 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. 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. 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. Prioridad BAJA.

5. NOTACIÓN DEL DIAGRAMA DE CLASES

Los diagramas de clases UML se componen de **clases**, **instancias** (objetos) e **interfaces** y visualizan relaciones jerárquicas, así como las asociaciones entre estos elementos. La notación de este tipo de diagrama es la base para la mayor parte del resto de diagramas estructurales.

UML 2 define a los diagramas de estructura como clasificadores. El diagrama de clases está indicado sobre todo como ejemplo para un diagrama de estructura. Otros diagramas de esta categoría utilizan componentes modificados del diagrama de clases para su notación.

Como diagrama de estructuras, muestra un sistema en estado estático de modo que el observador obtiene una perspectiva general de los elementos imprescindibles de un sistema, pero también visualizan las relaciones entre los componentes de esta arquitectura. Con un diagrama de clases UML se modelan desde objetos reales a clases abstractas con perfiles ampliables en cualquier lenguaje de programación. Con ellos se facilita la comunicación entre departamentos especializados en la realización de un proyecto.

CLASES, ATRIBUTOS Y MÉTODOS I

- 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:** identifica / **Atributos:** características / **Protocolo:** operaciones (métodos y mensajes)

3 zonas:

- Identificador
- Atributos
- Métodos

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.

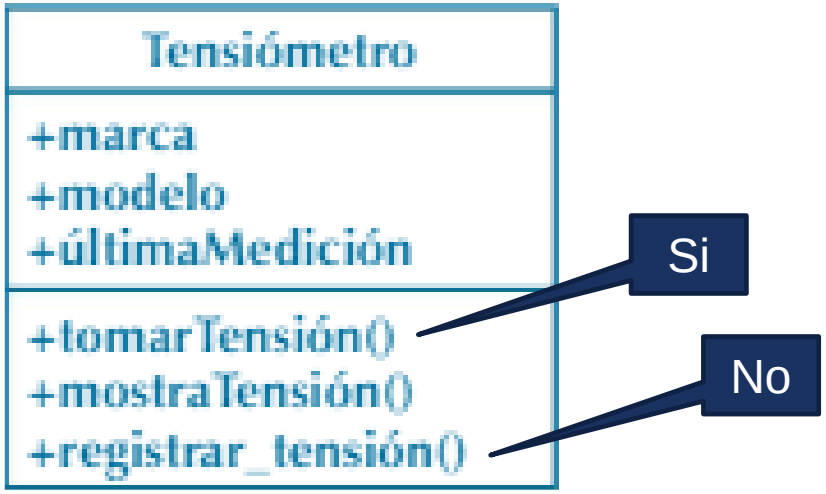
CLASES, ATRIBUTOS Y MÉTODOS II

Nombre. Identifica la clase.
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.

CLASES, ATRIBUTOS Y MÉTODOS III

Notación camelCase: **letra de caja camello** es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello.



Notación *camelCase*

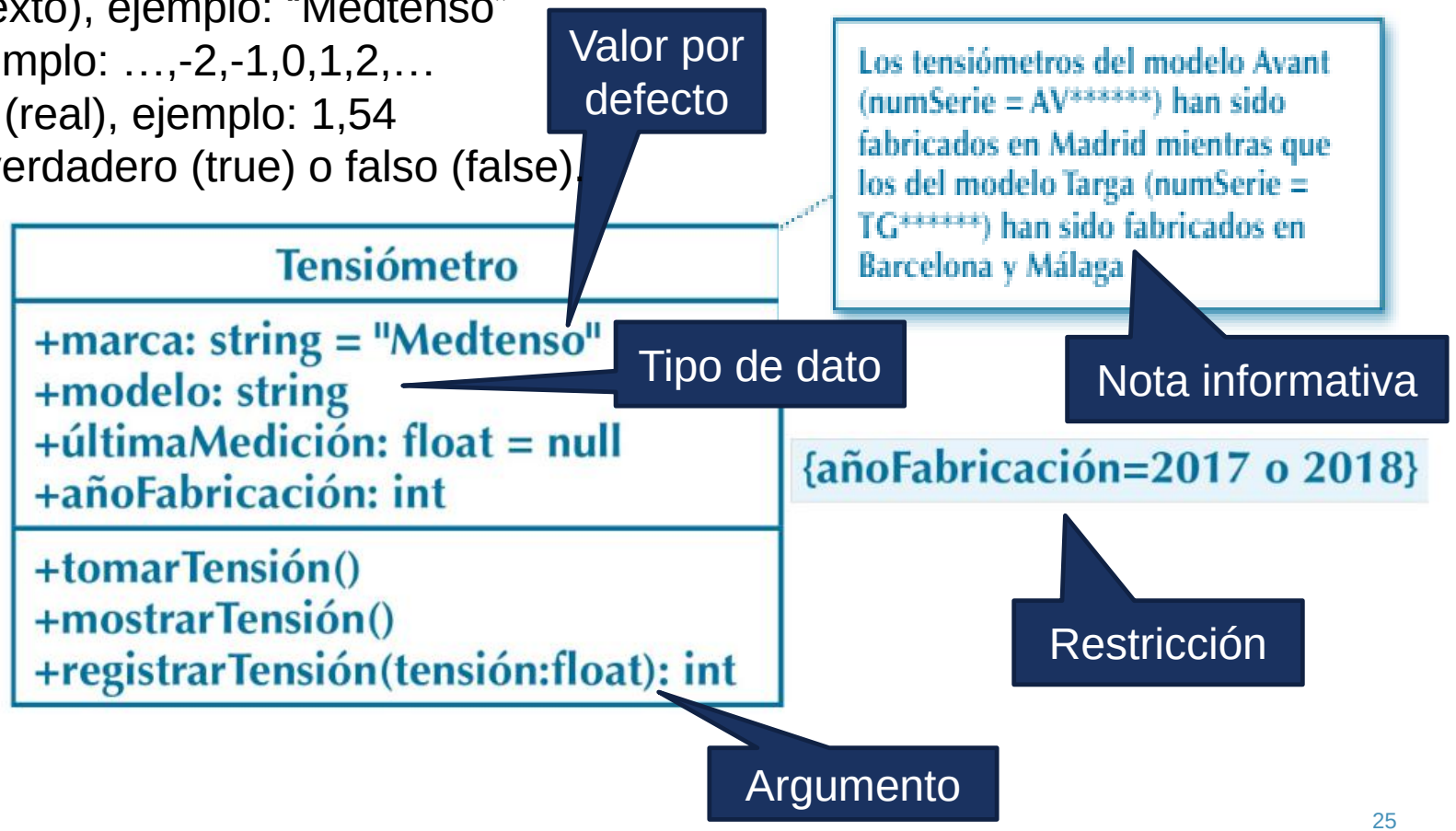
Muchos expertos recomiendan utilizar en la nomenclatura de los atributos y métodos el formato *camelCase*. Esto se aplica a palabras compuestas como pueden ser última medición o *tomar tensión*. Las mayúsculas serían como las jorobas de un camello. Las palabras compuestas comienzan con minúscula y la siguiente palabra del conjunto siempre comienza con mayúscula. Por ejemplo: últimaMedición, tomarTensión o lasJorobasDelCamello.

CLASES, ATRIBUTOS Y MÉTODOS IV

Atributos: Los atributos tienen asociado un tipo de dato, los más habituales son:

- String o cadena de caracteres (texto), ejemplo: “Medtenso”
- Integer (int) o número entero, ejemplo: …,-2,-1,0,1,2,…
- Float o número en coma flotante (real), ejemplo: 1,54
- Boolean o boleanos, puede ser verdadero (true) o falso (false).

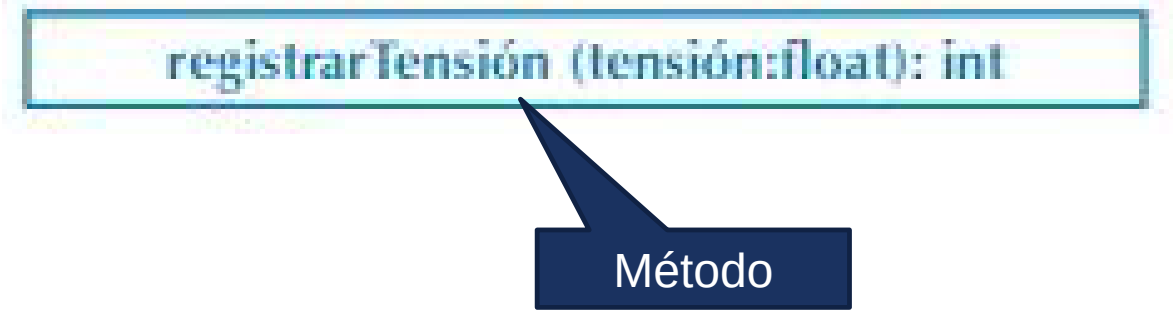
A los atributos se les puede asociar **valores por defecto** para toda la clase. Como la marca “Medtenso”. Podemos incluir **restricciones** como el año de fabricación. En los métodos podemos indicar los **argumentos** que requieren. En la clase también si se necesita podemos incluir **notas informativas**.



CLASES, ATRIBUTOS Y MÉTODOS IV

Métodos: utilizan la notación camelCase y en ocasiones suelen utilizar parámetros y devolver un valor una vez finalizadas las acciones correspondientes.

El método **registrarTensión()** acepta un parámetro de tipo **float** llamado **tensión** y devolverá un valor **entero (int).** Generalmente los métodos devuelven al menos un valor entero, 0 si todo ha ido bien y otro valor en caso contrario.



VISIBILIDAD I

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:

- **Interfaz**: captura la visión externa de una clase, abarcando la abstracción del comportamiento común a los ejemplos de esa clase.
- **Implementación**: comprende cómo se representa la abstracción, así como los mecanismos que conducen al comportamiento deseado.

VISIBILIDAD II

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 (public)**: símbolo (+). Se pueden acceder desde cualquier clase y cualquier parte del programa.
- **Privado (private)**: símbolo (-). Sólo se pueden acceder desde operaciones de la clase, no desde clases derivadas (*cuando se utiliza la herencia es la clase que hereda los atributos y métodos de la clase base*).
- **Protegido (protectec)**: símbolo (#). Sólo se pueden acceder desde operaciones de la clase o de clases derivadas, pero no desde derivadas de las derivadas.

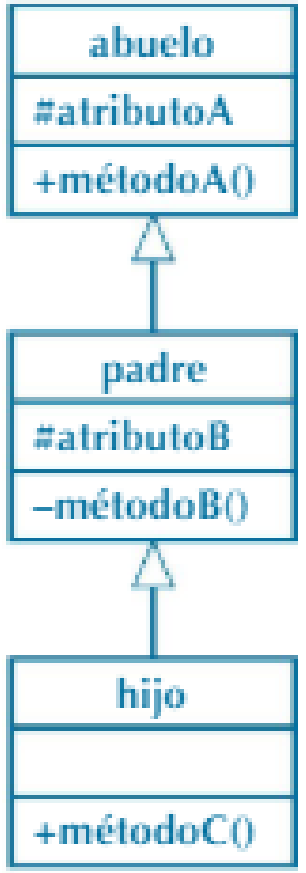
Como norma general a la hora de definir la visibilidad tendremos en cuenta que:

- 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 (acceder desde otras clases), pero en ocasiones, deben ser privadas (si no se utilizan desde clases derivadas) o protegidas (si se utilizan desde clases derivadas).

VISIBILIDAD III

Vemos como funciona la **visibilidad,** con un ejemplo concreto.

- El métodoA() puede acceder a atributoA.
- El métodoB() puede acceder a atributoA.
- El métodoB() puede acceder a métodoA().
- El métodoC() no puede acceder a métodoB(), pero sí que puede acceder a métodoA().
- El métodoC() puede acceder a atributoB, pero no a atributoA.



***Protected* en Java**

En Java, cuando un elemento es *protected,* es accesible a todas las subclases y todas las clases que estén dentro del mismo paquete (ojo, dentro del mismo paquete o *package*).

VISIBILIDAD IV

Dependiendo del problema que se plantee, el analista deberá escoger entre métodos y atributos privados, públicos y protegidos. Los programadores, generalmente, utilizan atributos privados y métodos llamados setters y getters para establecer el valor de un atributo y recuperarlo respectivamente.

Con setters y getters, el programador se asegura que ninguna clase o método ajeno pueda cambiar el valor interno de un objeto a su gusto. Siempre lo hará por medio del setter correspondiente cuando el atributo no es calculado. Esta será la única forma de cambiar o establecer el valor de un atributo y lo hará siguiendo las reglas determinadas en el setter, por lo que evitarán tener atributos con valores no permitidos. Con la creación de estos métodos se contribuye al **encapsulamiento y la ocultación de los atributos**.

Por ejemplo, podemos establecer que un atributo DNI, su setter tendrá en cuenta las reglas para evitar que se introduzca de manera incorrecta, cumpliendo que sea un número y la letra correspondiente.

INSTANCIACIÓN I

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:

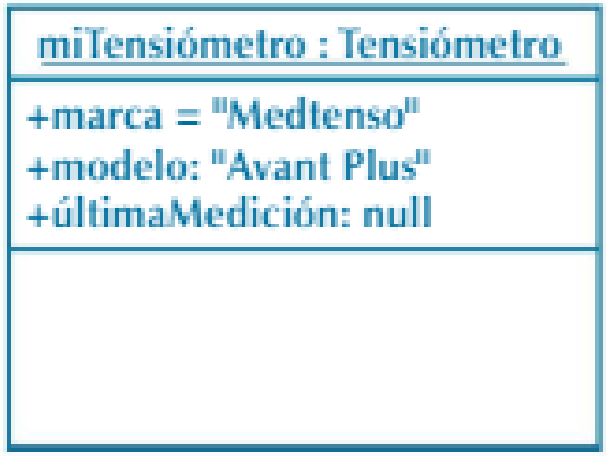
- **Su estado**: es la concreción de los atributos definidos en la clase a un valor concreto.
- **Su comportamiento**: definido por los métodos públicos de su clase.
- **Su tiempo de vida**: intervalo de tiempo a lo largo del programa en el que el objeto existe. Comienza con su creación a través del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

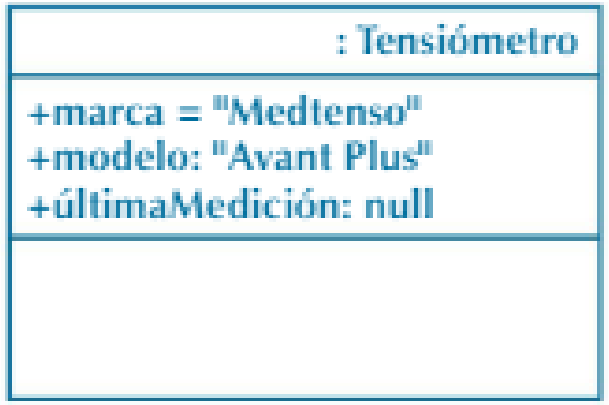
La encapsulación y el ocultamiento aseguran que los datos de un objeto están ocultos, con lo que no se pueden modificar accidentalmente por funciones externas al objeto.

INSTANCIACIÓN II

Un objeto es una instancia de una clase, pero con una serie de valores específicos en los atributos. Vemos el ejemplo de la clase Tensiómetro, con el objeto miTensiómetro que es del tipo tensiómetro con una marca y modelo determinados. El null indica que no se han realizado aún mediciones.

Pueden existir **instancias anónimas**, es decir objetos sin nombre, sin identidad ninguna. Ejemplo: la instancia anónima de la clase Tensiómetro.





INSTANCIACIÓN III

Grady Booch dice:
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.

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.

RELACIONES: ASOCIACIÓN I

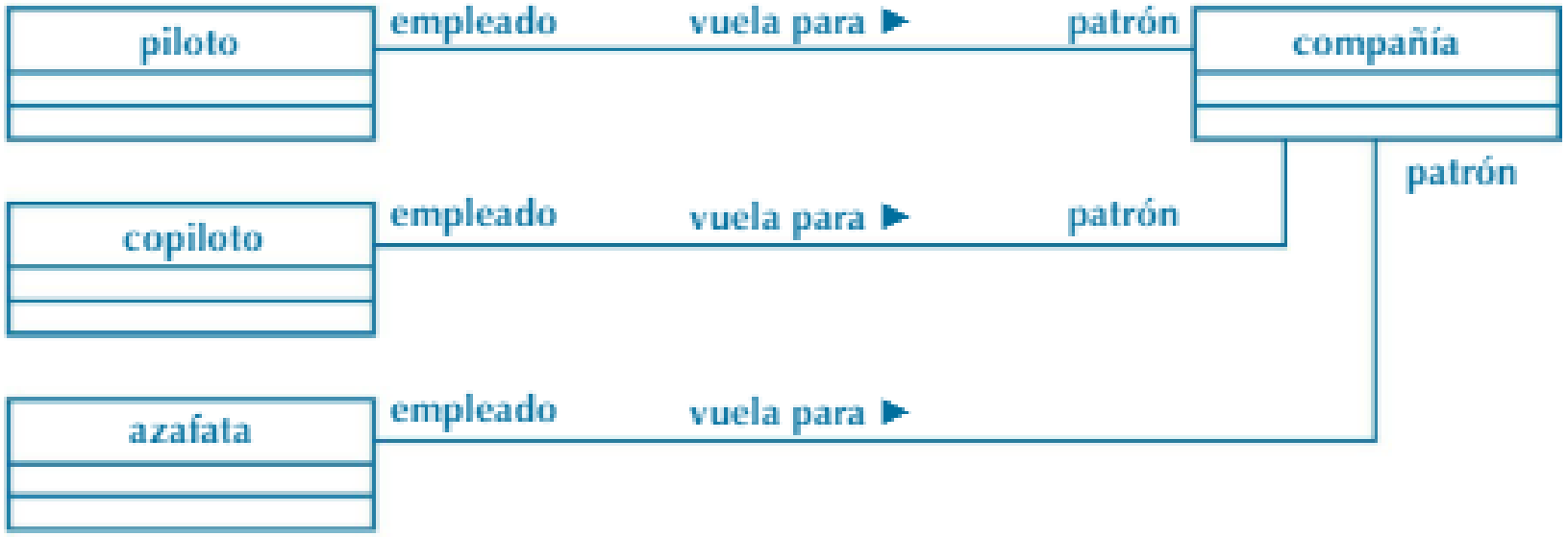
Las clases, generalmente están conectadas unas con otras de forma conceptual. Ejemplo: en un sistema de vuelos en los que compañías y pilotos de avión están conectados y a esa relación se denomina **asociación.**

Las flecha indica que un piloto vuela para una compañía, suele haber un rol entre ambas entidades (piloto y compañía), piloto es el empleado, la compañía el patrón.

Una asociación también puede representarse de forma inversa, una compañía puede emplear a uno o varios pilotos. Incluso podrían representarse las dos asociaciones con las dos entidades, pero por eficiencia y simplicidad se representa sólo una.

RELACIONES: ASOCIACIÓN II

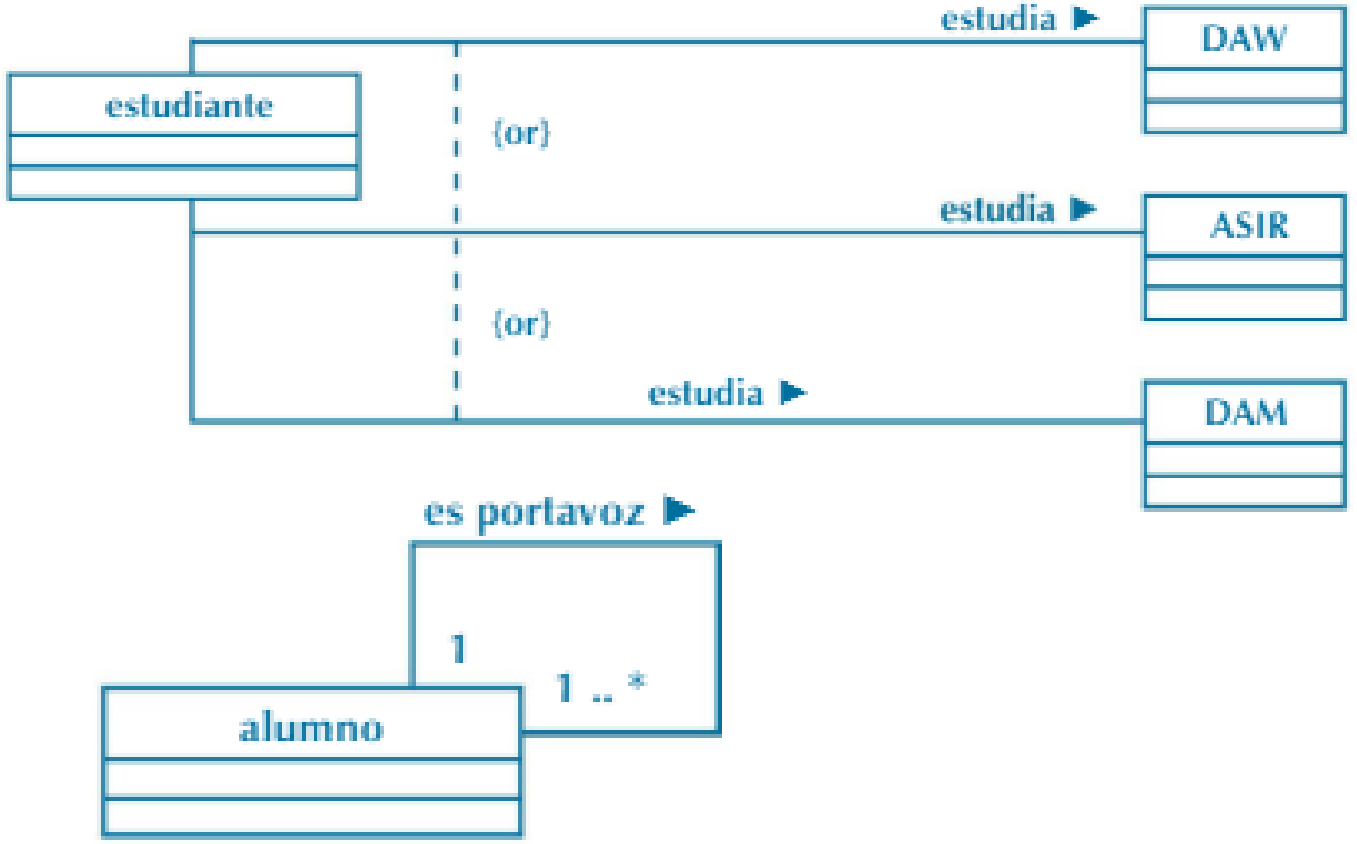
Es posible que una entidad o clase esté asociada con muchas otras clases. Por ejemplo, un piloto, copiloto, azafata (o azafato) podrían volar con una compañía aérea.



RELACIONES: ASOCIACIÓN III

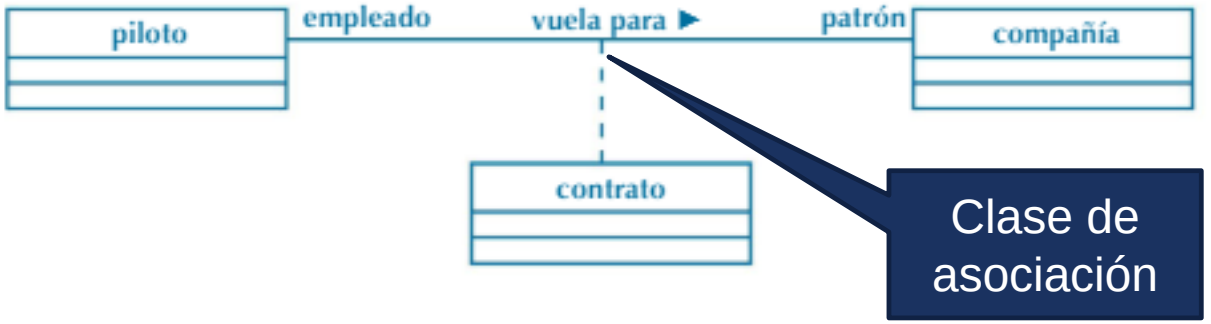
En ocasiones, hay que establecer ciertas restricciones en las asociaciones. Por ejemplo, para expresar que un estudiante de grado superior puede estudiar un ciclo de DAW, DAM o ASIR, pero no dos o más a la vez). La línea discontinua expresa la cláusula OR.

En ocasiones se utilizan asociaciones reflexivas, por ejemplo, alumnos que pueden ser portavoces de su grupo. Aunque algunos analistas prefieren utilizar la herencia y evitar asociaciones reflexivas.

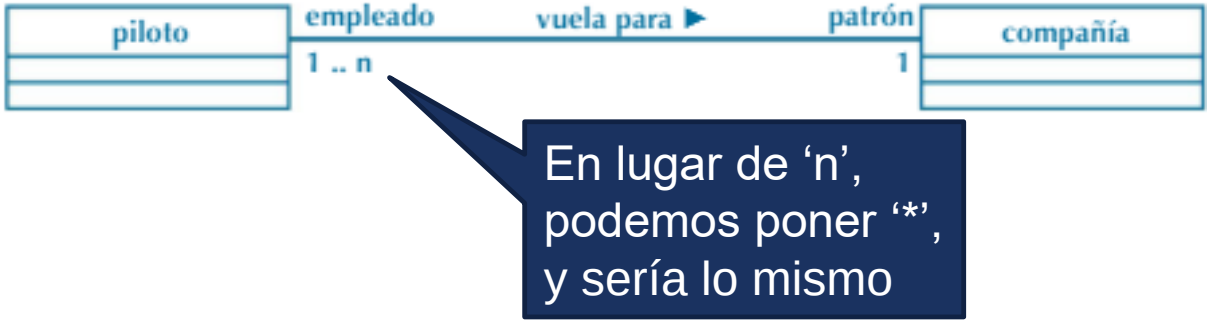


RELACIONES: ASOCIACIÓN IV

También es posible tener una clase de asociación, eso quiere decir que una asociación podría tener atributos y métodos, por tanto puede comportarse como una clase y asociarse con otras clases a su vez. Puede observarse que la asociación **“vuela para”** tiene una clase de asociación denominada **contrato**. Se asocia con una línea discontinua.



En las asociaciones se expresan las **multiplicidades o cardinalidades**, se indica el número de entidades o cantidad de objetos que participan en ella. Los números se colocan en los extremos de la asociación. 1 o varios pilotos vuelan para 1 compañía aérea.



RELACIONES: ASOCIACIÓN V

Pero podemos tener diferentes cardinalidades. Por ejemplo, si tengo la relación (1), 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 (2), en la que un profesor puede impartir uno o varios módulos, mientras que un módulo es impartido sólo por un profesor.

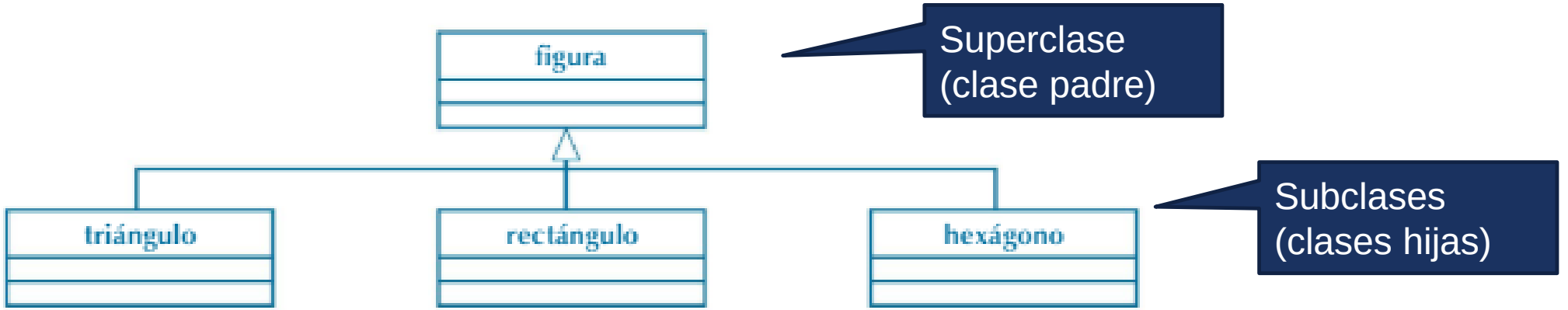
Significado de las cardinalidades.	
Cardinalidad	**Significado**
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

1

2

RELACIONES: HERENCIA I

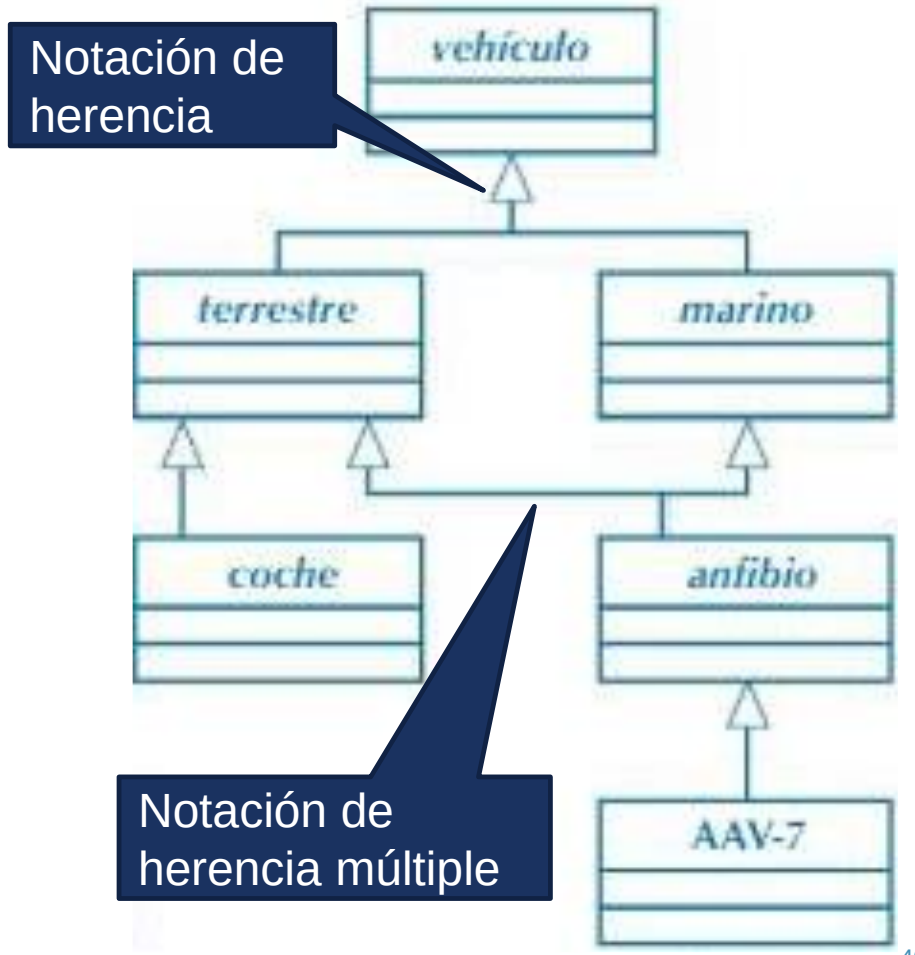
La herencia o generalización es uno de los aspectos mas utilizados en la OO. Así por ejemplo es fácil pensar que un triángulo, un rectángulo o un hexágono tienen aspectos o características en común. De hecho son figuras geométricas. Todas ellas comparten atributos, como el número de lados, el área, el perímetro, etc. Y en cuanto comportamientos (métodos), como desplazarse, rotar, agrandarse, etc.
Por tanto, lo que se tiene es una clase principal (figura) y subclases (clases secundarias o subclases), que heredan los atributos y operaciones de la clase principal (o superclase). No puede existir un objeto de la clase figura, siempre será un triángulo, rectángulo, etc. , de lo contrarío no sería herencia, sería una asociación. Las clases que no tienen objetos directamente son clases abstractas.



RELACIONES: HERENCIA II

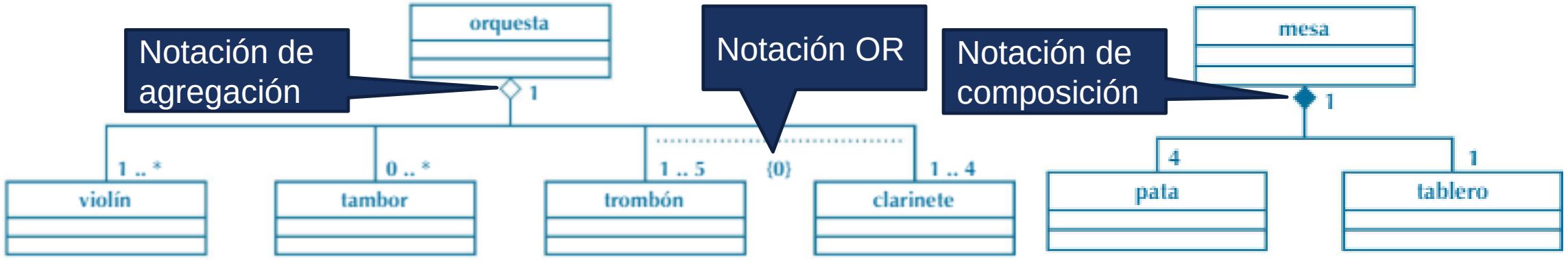
Las jerarquías pueden tener diferentes niveles observamos en el ejemplo hay una clase base o superclase principal llamada ***vehículo*****,** y luego otras dos como ***terrestre*** (vehículo terrestre) y ***marino*****,** (vehículo marino). También tenemos la subclase ***anfíbio*** que tiene características de terrestre y marino (herencia multiple), y esta a su vez tiene otra subclase **AAV-7**, que sería más concreta, es un vehículo de la armada española. Las superclases (clases padres) se colocan encima de las subclases (clases hijas). Todas las clases menos AAV-7 son abstractas, y suelen ir en cursiva.

Una clase puede heredar tanto de una superclase como de varias (herencia múltiple). En Java, no existe la herencia múltiple (sí en otros lenguajes como C++), pero existen mecanismos como las interfaces para hacer que una clase herede el comportamiento de varias clases.



RELACIONES: AGREGACIÓN Y COMPOSICIÓN

Se entiende por **agregación** una relación en la que varios elementos se combinan para formar un todo. Imaginar una orquesta, en ella puede tener un violín, una trompeta, un tambor, etc.
Las agregaciones pueden tener cardinalidades, como vemos en el ejemplo. Puede tener uno o muchos violines, ninguno o muchos tambores, entre uno y cinco trombones o entre uno y cuatro clarinetes. Es decir si tiene trombones no tienen clarinetes o al revés, si tiene clarinetes no tiene trombones.
Existe otro tipo de relación llamada **composición**, es muy parecida a la agregación, pero con la diferencia que los objetos de la composición no pueden formar parte de otras composiciones. Así un mismo violinista podría estar en dos orquestas (una por la mañana y otra por las tardes), pero lo pata de una mesa no puede quitarse para ponerla en otra mesa.



6. HERRAMIENTAS PARA ELABORAR DIAGRAMAS UML

Existen muchas herramientas en el mercado para la elaboración de diagramas UML. Para la elaboración de esta unidad, se ha elegido DIA, que es una herramienta libre con licencia GNU. También podemos utilizar una herramienta profesional como Visual Paradigm durante 30 días de manera gratuita o con la versión no comercial. Otra herramientas no gratuitas son MagicDraw o Visio de Microsoft.

- Muchas de estas herramientas son **independientes del lenguaje de programación**, obviamente, y también de la metodología de desarrollo.
- Las herramientas CASE para el diseño de diagramas UML deben ser **intuitivas y abiertas** a la incorporación de plugins que aumenten las funcionalidades.
- Este tipo de herramientas tienen que adaptarse al **trabajo colaborativo** y al acceso simultaneo de un equipo de desarrollo sin que se produzca ningún tipo de conflictos.
- Algunas herramientas permiten la **ingeniería inversa**, generando diagramas a partir del código. Esto permite que se conozca mejor el funcionamiento de un proyecto que estuviese mal documentado.

EJERCICIO 1

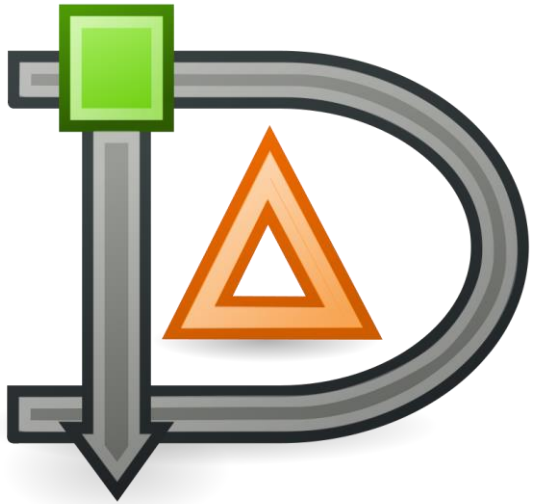
- Realizar la práctica del AULA VIRTUAL: P01-Explicación del Diagrama de Clases.

Escribe una documento con un esquema o mapa conceptual (en una página)

- Debes detallar todos los pasos que se deben realizan al hacer un análisis para preparar el diagrama de clases. No debes explicar ninguna herramienta.
- Súbelo al aula virtual (AV).

DIA

Dia es una aplicación informática de propósito general para la creación de diagramas, creada originalmente por Alexander Larsson, y desarrollada como parte del proyecto GNOME. Está concebido de forma modular, con diferentes paquetes de formas para diferentes necesidades.



Se puede utilizar para dibujar diferentes tipos de diagramas. Actualmente se incluyen diagramas entidad-relación, **diagramas UML**, diagramas de flujo, diagramas de redes, diagramas de circuitos eléctricos, etc. Nuevas formas pueden ser fácilmente agregadas, dibujándolas con un subconjunto de SVG e incluyéndolas en un archivo XML. Gracias al paquete dia2code, es posible generar el esqueleto del código a escribir, si se utiliza con tal fin un UML.

El formato para leer y almacenar gráficos es XML (comprimido con gzip, para ahorrar espacio). Puede producir salida en los formatos EPS, SVG y PNG. **En el aula virtual disponemos de una guía de uso**. http://dia-installer.de/index.html.es

VISUAL PARADIGM

Obtenemos los archivos desde la página de Visual Paradigm, ofrece dos versiones:

Visual Paradigm for UML (VP-UML), gratis durante 30 días mediante registro.
https://www.visual-paradigm.com/download/?platform=windows&arch=64bit
Versión Community-Edition, para uso no comercial (gratuito).
https://www.visual-paradigm.com/download/community.jsp
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.
Disponemos de una versión on-line gratuita
https://online.visual-paradigm.com/es/

MICROSOFT VISIO

Microsoft Visio es un software de dibujo vectorial para Microsoft Windows. Microsoft compró la compañía Visio en el año 2000.
Las herramientas que lo componen permiten realizar diagramas de oficinas, diagramas de bases de datos, diagramas de flujo de programas, **UML**, y más, que permiten iniciar al usuario en los lenguajes de programación.
Aunque originalmente apuntaba a ser una aplicación para dibujo técnico para el campo de Ingeniería y Arquitectura; con añadidos para desarrollar diagramas de negocios, su adquisición por Microsoft implicó drásticos cambios de directrices de tal forma que a partir de la versión de Visio para Microsoft Office 2003 el desarrollo de diagramas para negocios pasó de añadido a ser el núcleo central de negocio, minimizando las funciones para desarrollo de planos de Ingeniería y Arquitectura que se habían mantenido como principales hasta antes de la compra.

https://www.microsoft.com/es-es/microsoft-365/visio/flowchart-software

MAGICDRAW

MagicDraw UML es una herramienta CASE desarrollada por No Magic. La herramienta es compatible con el estándar UML, desarrollo de código para diversos lenguajes de programación (Java, C++ y C#, entre otros) así como para modelar datos.

https://www.nomagic.com/products/magicdraw

La herramienta cuenta con capacidad para trabajar en equipo y es compatible con las siguientes IDE's:

- NetBeans 6.X or later.
- Eclipse 3.1 o superior (versión Java).
- Sun Java Studio 8.
- Borland CaliberRM 6.0, 6.5 herramienta de requisitos.
- Oracle Workshop 8.1.2.
- E2E Bridge 4.0
- IntelliJ IDEA 4.X or later.
- IBM Rational Application Developer, etc.

EJERCICIO 2

- Realizar la práctica del AULA VIRTUAL: P02-Diagrama de Clases. Para realizarlo utiliza la herramienta DIA o alguna similar que sea gratuita.

Escribe una documento con portada, índice, url consultadas, etc.

- Debes incluir el enunciado de cada ejercicio, razonando como llegas a obtener el diagrama de clases, pon el diagrama de clases en el documento como una imagen.
- No es necesario que expliques como usas la herramienta DIA o similar.
- Súbelo al aula virtual (AV).