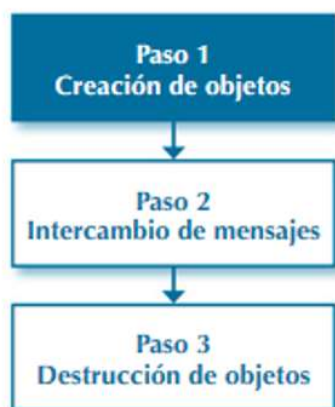


UT5- Diagramas de clases

1. INTRODUCCIÓN

En el **diseño orientado a objetos**, un sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos.

El ciclo de vida de un programa orientado a objetos sería como el siguiente:



Paso 1. Se crean los objetos (siempre, a medida que son necesarios)

Paso2: El usuario interactúa con el programa y los objetos van intercambiando mensajes entre ellos. En ocasiones, al procesar la información, suelen crearse y destruirse objetos.

Paso3: Los objetos ya no son necesarios y se destruyen. Bien es el programador el que se encarga de incluir la destrucción del objeto en el código, bien existe un proceso en el sistema encargado de ello.

Recuerda- Un objeto siempre es una entidad que tiene una serie de propiedades o atributos(datos) y un comportamiento o funcionalidad(métodos)



Un **objeto** consta de una estructura de datos y de una colección de métodos u operaciones que manipulan esos datos. Los datos definidos dentro de un objeto son sus atributos. Las operaciones definen el comportamiento del objeto y cambian el valor de uno o más atributos. Los objetos se comunican unos con otros a través del paso de mensajes.

Una **clase** no es más que una plantilla para la creación de objetos. Cuando se crea un objeto se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto.

Para el análisis y diseño orientado a objetos se utiliza **UML** (Unified Modeling Language- Lenguaje de Modelado Unificado). Es un lenguaje de modelado basado en diagramas que sirve para expresar modelos (un modelo es una representación de la realidad donde se ignoran detalles de menor importancia).

2. CONCEPTOS ORIENTADOS A OBJETOS

El paradigma OO se basa en el concepto de objeto. Un objeto es aquello que tiene estado (propiedades más valores), comportamiento (acciones y reacciones a mensajes) e identidad (propiedad que lo distingue de los demás objetos). La estructura y comportamiento de objetos similares están definidos en su clase común; los términos "**instancia y objeto**" son



 CIFP VIRGEN DE GRACIA	<p style="text-align: center;">UT5-Diagrama de clases</p> <p style="text-align: center;">Entornos de desarrollo (1º DAW)</p>	<p style="text-align: center;">Dpto. INFORMÁTICA</p>  <p style="text-align: center;">Curso: 2022-23</p>
--	--	---

intercambiables. **Una clase es un conjunto de objetos que comparten una estructura y comportamiento común.**

Clases y objetos pueden parecer conceptos similares, pero existe una clara diferencia conceptual entre ellos. Las **clases** son un concepto **estático** definido en el programa fuente, son una abstracción de la esencia de un objeto, mientras que los **objetos** son entes **dinámicos** que existen en tiempo y espacio, y que ocupan memoria en la ejecución de un programa.

La orientación a objetos 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:** Denota las características esenciales de un objeto, donde se capturan sus comportamientos. El objeto es obtener una descripción formal. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a componer un conjunto de clases que permitan modelar la realidad o el problema que quiere resolver.
- **Encapsulación:** La encapsulación es el proceso de ocultar todos los detalles de un objeto que no contribuyen a sus características esenciales, es decir, separar el aspecto externo del objeto accesible por otros objetos, del aspecto interno del mismo que será inaccesible para los demás. O lo que es lo mismo: la encapsulación consiste en ocultar los atributos y métodos del objeto a otros objetos, estos no deben estar expuestos a los objetos exteriores. Una vez encapsulados, pasan a denominarse atributos y métodos privados del objeto.
- **Modularidad:** Es la propiedad de una aplicación o de un sistema que ha sido descompuesto en un conjunto de módulos o partes más pequeñas coherentes e independientes. Estos módulos se pueden compilar por separado, pero tiene conexiones con los otros módulos.
- **Jerarquía o Herencia:** La programación orientada a objetos introduce la posibilidad de extender clases, produciendo nuevas definiciones de clases que heredan todo el comportamiento y código de la clase extendida. La clase original se denomina clases padre, base o superclase. La nueva clase que se define como una extensión se denomina clase hija, derivada o subclase. La extensión de una clase se denomina herencia, porque la nueva clase hija hereda todos los métodos y atributos de la clase padre que se extiende. Cada subclase estaría formada por un grupo de objetos más especializados con características comunes que compartirían datos y operaciones. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
- **Polimorfismo:** Consiste en reunir con el mismo nombre comportamientos diferentes. Es la propiedad por la cual un mismo mensaje puede originar conductas completamente diferentes al ser recibido por diferentes objetos. De un modo más preciso: dos instancias u objetos, pertenecientes a distintas clases, pueden responder a la llamada a métodos del mismo nombre, cada uno de ellos con distinto comportamientos encapsulados, pero que responden a una interfaz común (marcada a través del mecanismo de la herencia)

 <p>CIFP VIRGEN DE GRACIA</p>	<p>UT5-Digrama de clases</p> <p>Entornos de desarrollo (1º DAW)</p>	<p>Dpto. INFORMÁTICA</p>  <p>Curso: 2022-23</p>
--	---	--

- **Tipificación:** Es la definición precisa de un objeto, de tal forma que objetos de diferentes tipos no puedan ser intercambiados o, puedan intercambiarse de manera muy restringida.
- **Concurrencia:** Es la propiedad que distingue un objeto que está activo de uno que no lo está. El objeto activo está haciendo algo, se utilizan sobre todo en la programación concurrente o multihilo
- **Persistencia.** Es la propiedad de un objeto a través de la cual su existencia trasciende de tiempo (es decir, el objeto continúa existiendo después de que su creador ha dejado de existir) y/o el espacio. Se refiere a objetos de clases asociadas a Bases de Datos Orientas a Objetos (BDOO) o a Bases de Datos Objetos Relacionales (BDOR).

3. QUE ES UML



- Lenguaje
- Modelado- Crear una representación de la realidad
- Unificado

El Lenguaje de Modelado Unificado (UML- Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de SW. Este lenguaje se puede utilizar para modelar tanto sistemas de Sw, como de HW, como organizaciones del mundo real. Para ello, utiliza una serie de diagramas en los que se representan los distintos puntos de vista de modelado. Podemos decir que UML es un lenguaje que se utiliza para documentar.

UML fue creado por los padres de la OO (Grady Booch, JimRumbaugh e Ivar Jacobson) es el estándar utilizado por la industria del software.

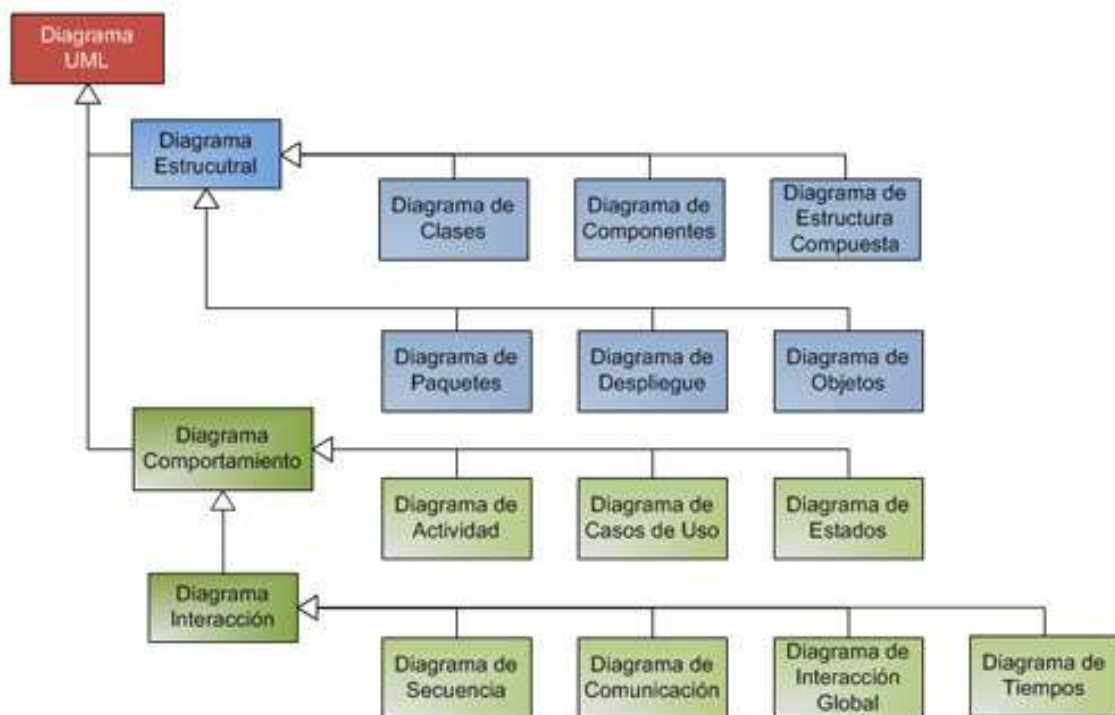
¿Por qué es necesario un estándar como el UML? Cuando se desarrolla un sistema software, muchas veces, un equipo de personas (muchas o pocas dependiendo de la envergadura del proyecto) tienen que comunicarse durante su creación. Se generará un volumen ingente de documentación e información y todo esto tiene que estar normalizado de alguna manera para que sea comprensible para todo el grupo.

El UML soluciona este problema y puede ser útil tanto en proyectos pequeños como en proyectos muy grandes.

La primera versión de UML apareció en 1997, y tras varias revisiones y modificaciones, es un estándar de facto en la industria del software y lo seguirá siendo a corto y medio plazo.

UML 2.0 define 13 tipos de diagramas, divididos en 3 categorías: 6 tipos de diagramas representan la estructura estática de la aplicación o del sistema, 3 representan tipos genera de comportamiento y 4 representan diferentes aspectos de las interacciones:

- **Diagrama de estructura** (parte estática del modelo): Incluyen el **diagrama de clases**, diagrama de objetos, diagrama de componentes, diagrama de estructura compuesta, diagrama de paquetes y diagrama de implementación o despliegue. Se centran en los elementos que deben existir en el sistema modelado.
- **Diagramas de comportamiento** (parte dinámica del modelo): incluyen el **diagrama de casos de uso** (usando por algunas metodologías durante la recopilación de requisitos), diagrama de actividad y diagrama de estado. Se centran en lo que debe suceder en el sistema.
- **Diagramas de interacción**: todos derivados del diagrama de comportamiento más general. Incluyen el **diagrama de secuencia**, diagrama de comunicación, diagrama de tiempos y diagrama de vista de interacción. Se centran en el flujo de control y de datos entra los elementos del sistema modelado.



Notación de los diagramas de clases

Las clases y objetos suelen representarse en diagramas mediante la notación UML, como se ha mencionado anteriormente, UML es un conjunto de herramientas que van a permitir modelar, documentar y, posteriormente, construir un sistema o aplicación orientada a objetos.

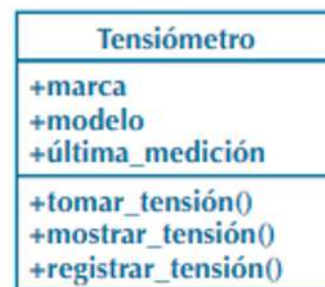
CLASES

Las clases, como ya se ha explicado, pueden ser categorías o tipos de cosas y se modelan de la forma que se observa en la figura, donde una clase está dividida en tres áreas. En el área superior, se encuentra el nombre de la clase, que en este caso, será tensiómetro.

En el área central, se almacenará el estado de la clase (los atributos). Podría almacenarse mucha más información (tanta como sea necesaria).

Por último, en el área inferior, puede observarse cómo está registrado el comportamiento de la clase (qué es lo que hace).

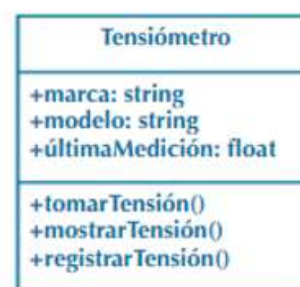
Con las clases, puede modelarse el mundo real. Una clase sería como en repostería el molde de un bizcocho y los bizcochos realizado con el molde serán los objetos de dicha clase. Por ejemplo, un objeto de nuestra clase Tensiómetro podría ser un tensiómetro de la marca Medtenso, modelo Avant Plus y con la última medición vacío, puesto que es nuevo y nunca ha sido utilizado.



ATRIBUTOS

Los atributos de una clase suelen tener un tipo determinado. UML permite indicar el posible formato de un atributo especificado su tipo:

1. *String o cadena de caracteres* (texto).
2. *Integer/int o número entero*. Posibles valores: 128,32, -4,etc
3. *Float o número en coma flotante* (numero real). Posibles valores :3,1428252637 , etc
4. *Boolean o booleano*. Puede especificarse verdadero o falso.



En la figura, puede verse como se ha incluido el tipo de los atributos en la clase.



UT5-Diagrama de clases

Entornos de desarrollo (1º DAW)

Dpto. INFORMÁTICA



Curso: 2022-23

Tensiómetro: miTensiómetro
+marca: string = "Medtenso"
+modelo: string
+últimaMedición: float = null
+tomarTensión()
+mostrarTensión()
+registrarTensión()

Los atributos pueden tener valores por defecto o valores predeterminados. Por ejemplo, puede determinarse que todos los tensiómetros que se generen tendrán como marca el valor MedTenso y que el valor por defecto para el atributo últimaMedición sea nulo.

Notación camelCase- Nomenclatura de los atributos y métodos. Se aplica a palabras compuestas como pueden ser última medición o toma tensión. Las palabras compuestas comienzan con minúscula y la siguiente palabra del conjunto siempre comienza con mayúsculas.

Al crear un atributo se indica su visibilidad (relacionada con encapsulamiento).

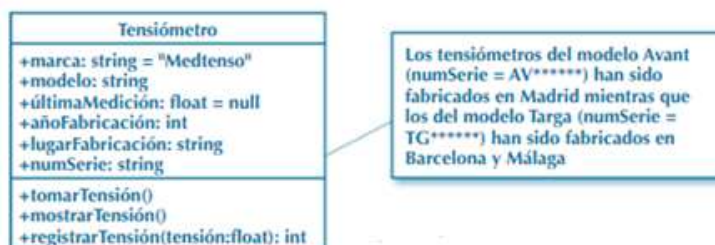
Los tipos de visibilidad son:

Público (public)	+	elemento no encapsulado visible para todos
Protegido (protected)	#	elemento encapsulado visible en la clase y las subclases de la clase
Privado (private)	-	elemento encapsulado visible solo en la clase
Paquete (package)	~	elemento encapsulado visible solo en las clases del mismo paquete

NOTAS ADJUNTAS

En ocasiones, es necesario añadir información extra a la clase. Imagínese que nuestro tensiómetro tiene un número de serie y también un lugar de fabricación. Dicho número de serie tiene un formato determinado que depende del modelo. Además, según el modelo, el lugar de fabricación será diferente.

Este tipo de información extra o aclaraciones irán en notas adjuntas.



Las notas adjuntas pueden llevar incluso imágenes y va a permitir al analista incluir información extra en el diagrama de clases que, luego, habrá que tener en cuenta en posteriores fases como la de codificación.

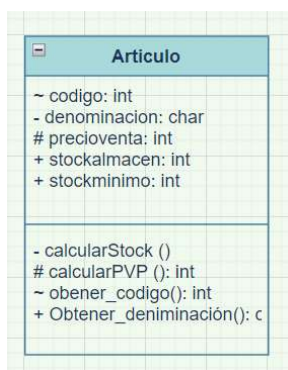
MÉTODOS

Las operaciones o métodos de una clase son su parte dinámica. Al igual que con los atributos, se utilizan la notación CamelCase y, en ocasiones, suelen aceptar una serie de parámetros y devolver un valor. Este valor lo devolverá una vez que finalice las acciones correspondientes.

registrarTensión (tensión:float): int

En la figura, puede observarse cómo el método registrarTensión() acepta un parámetro de tipo float llamado tensión y devolverá un valor entero (integer)

Generalmente, los métodos suelen devolver al menos un valor entero. Devuelven 0 si todo ha ido bien y otro valor en caso contrario.



Los métodos también llamados operaciones o funciones, muestran el comportamiento de todos los objetos.

Definen la interacción de una clase con su entorno.

Al igual que los atributos, los métodos pueden ser:

Public

Private

Protect

Package

RELACIONES: ASOCIACIONES

Las clases, generalmente, están conectadas unas con otras de forma conceptual. Imagínese que desea diseñarse un sistema de vuelos en los que hay compañías y pilotos de avión. Está claro que existe una conexión entre ambas entidades (piloto y avión) y esa relación se denomina en **UML asociación**.



La asociación tiene una dirección mostrada por la flecha que indica que un piloto vuela para una compañía (aérea).

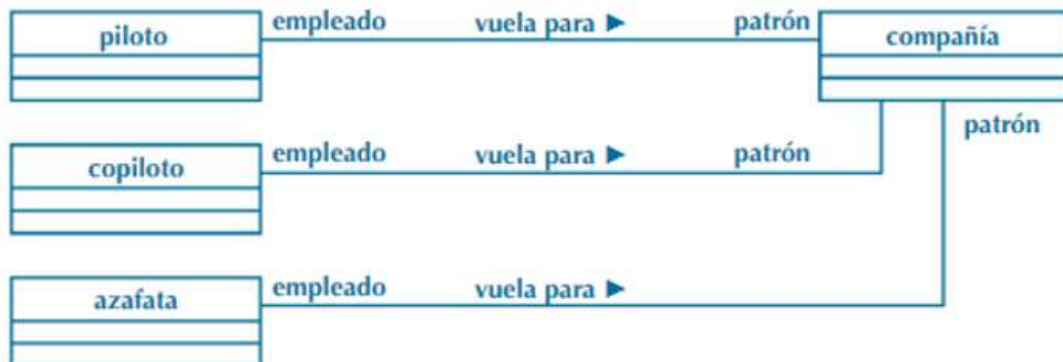
En estas asociaciones, siempre suele haber un rol entre ambas entidades (piloto y compañía). El piloto es el empleado, mientras que la compañía es el patrón.



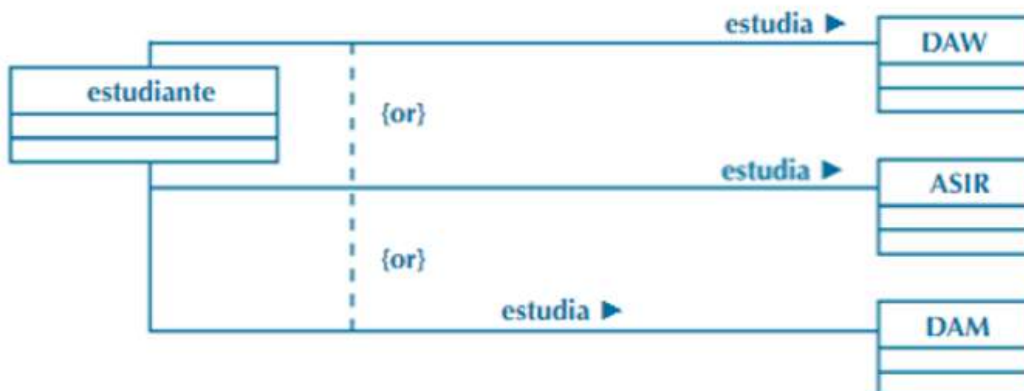
Como puede verse en la figura (asociación entre las clases Piloto y compañía con roles) los roles de cada una de las entidades se colocan en los extremos de la asociación. Obviamente, el rol está al lado de la clase que lo representa.

Una asociación también puede interpretarse de forma inversa, una compañía puede emplear a uno o varios pilotos. Podrían representarse incluso las dos asociaciones con las dos entidades, pero por simplicidad y eficiencia, suele representarse solamente una.

Es posible también que una entidad o clase está asociada con muchas otras clases (asociación entre una serie de clases). Ej: Se sobreentiende que un piloto, copiloto o azafata (azafato) podrían volar con una compañía aérea.



En ocasiones, hay que establecer ciertas restricciones en las asociaciones (asociación con restricciones entre una serie de clases) . Imagínese que un estudiante de grado superior puede estudiar un ciclo de DAW,DAM o ASIR (nunca dos o más a la vez). Esto se representaría como puede verse en la siguiente figura con una línea discontinua y las cláusulas {or}. En el ejemplo, un estudiante estudiaría uno y solo uno de los ciclos anteriores.

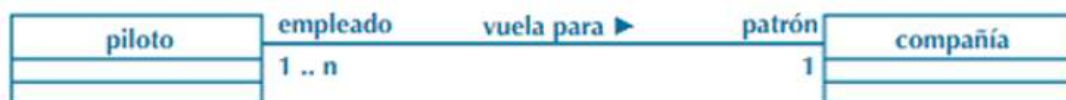


También existe la posibilidad de que exista una **clase de asociación**. Eso quiere decir que una asociación podría tener atributos y operaciones o métodos, por tanto, una asociación puede comportarse como una clase y estar también asociada con otras clases a su vez.



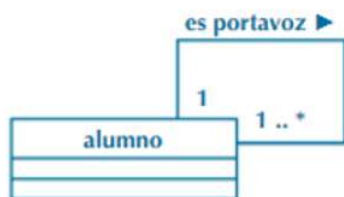
En la figura de arriba, puede observarse que la asociación “vuela para” tiene una clase de asociación denominada contrato. Esta clase de asociación contrato, como puede observarse, está conectada con la asociación mediante una línea discontinua.

En las asociaciones de UML, suelen especificarse las multiplicidades, es decir, que se especifica el número de entidades o cantidad de objetos que participan en ella. El número de objetos suele colocarse en los extremos de la asociación.



Analizando la asociación de arriba, se observa que uno o varios pilotos pueden volar para una compañía aérea. No existiría ninguna compañía aérea que no tuviese ningún piloto, puesto que la cardinalidad mínima sería 1. 1..n en el ejemplo significaría “uno o varios pilotos” (no se sabe el número). En ocasiones, en vez de n, se puede poner un asterisco (*), que significaría lo mismo.

En ocasiones, un analista puede encontrarse con diseños en los que existen asociaciones reflexivas. No suele ser muy común, podría darse el caso dependiendo del sistema que desee modelarse.



En el ejemplo, puede darse el caso de que existan una serie de alumnos, algunos de los cuales pueden ser portavoces de esos grupos.

Cardinalidad:

Las relaciones muestran la vinculación entre las clases.

Las relaciones tienen un nombre y una cardinalidad (multiplicidad) que indica el nº de instancias de una clase que se relacionan con instancias de la otra clase.

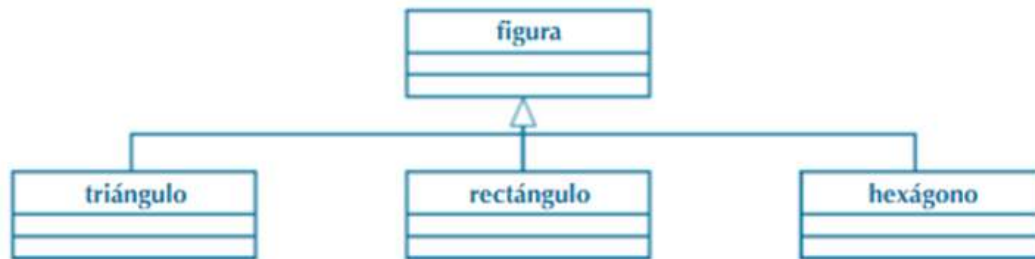
Para representar la multiplicidad mínima y máxima se utiliza la notación:

Notación	Cardinalidad/Multiplicidad
0..1	Cero o una vez
1	Una y solo una vez
*	De cero a varias veces
1..*	De una a varias veces
M..N	Entre M y N veces
N	N veces

RELACIONES: HERENCIA

La herencias o generalización es uno de los aspectos más utilizados en la OO. Todos pueden deducir que un triángulo, un rectángulo o un hexágono tiene aspectos o característica en común. Todas estas figuras compartirían atributos, como puede ser el número de lados, el área, el perímetro, etc., y comportamiento (métodos), como desplazarse, rotar, agrandarse o disminuirse de forma proporcional, etc.

Por lo tanto, lo que se tiene es una clase principal (en nuestro caso figura) y luego una serie de subclases más específicas llamadas clases secundarias o subclases, las cuales heredan tanto los atributos como las operaciones de la clase principal (o superclase)



Las jerarquías no solamente tienen dos niveles, sino que pueden tener múltiples niveles.

A) Asociación y herencia

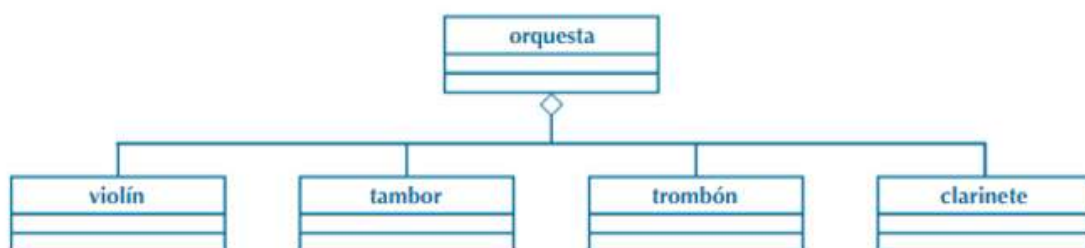
Si la subclase es un subtipo de la clase base (comparte atributos y comportamiento o métodos), generalmente es herencia, en caso contrario, sería una asociación. Muchas veces, es sencillo descubrirlo porque, por ejemplo, nunca va a existir un objeto del tipo figura, siempre será un triángulo, un rectángulo, etc.

B) Clases abstractas

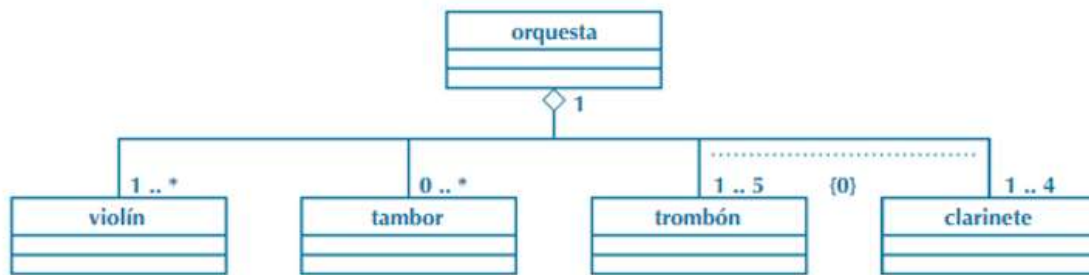
Existen clases de las que nunca va a existir un objeto, puesto que son genéricas. Siempre van a existir objetos, pero de sus subclases. En ese caso, se dice que dicha clase es abstracta y suele representarse con su nombre en cursiva.

Relaciones: Composición y agregación

Se entiende por agregación una relación que la que varios elementos se combinan para formar un todo. Imagínese una orquesta



Las agregaciones se muestran de forma parecida a la herencia, de tal manera que la clase que agrupa a las demás suele colocarse en la parte superior. A diferencia de la herencia, que se dibuja con un triángulo vacío, la agregación usa un rombo vacío. Como se puede observar las agregaciones también pueden tener cardinalidades.



Una **composición** es muy parecido a la agregación, pero con una diferencia: los objetos que forma la composición no pueden formar parte de otras composiciones.

