



Introducción

Breve reseña. Nociones básicas UML.

Objetos. Clases. Mensajes. Métodos. Variables e Instancias.

Contenidos

Breve reseña	2
Introducción	4
Nociones básicas de UML	6
UML (Unified Modeling Language)	6
Diagramas de clases (atributos y operaciones)	7
Relaciones entre clases (dependencia, generalización y asociación)	11
Diagramas de objetos	15
Diagrama de secuencias	18
Objetos	18
Organización de los objetos	20
Atributos de los objetos	20
Clases	22
Relaciones entre clases	25
Mensajes, métodos, variables e instancias	25
Mensajes	25
Métodos	26
Variables	28
Declaración de variables de instancia	29

Breve reseña

La velocidad con la que avanza la tecnología del hardware en los últimos años es muy grande, con lo que se ha logrado tener computadoras más poderosas, baratas y compactas. Pero el Software no ha tenido el mismo comportamiento, ya que, al desarrollar aplicaciones, es frecuente que se excedan los tiempos de entrega, así como los costos de los sistemas de información (tanto de desarrollo, como de mantenimiento), además de ser poco flexibles. Se han creado diferentes herramientas de ayuda al desarrollo, para lograr aumentar la productividad en el Software:

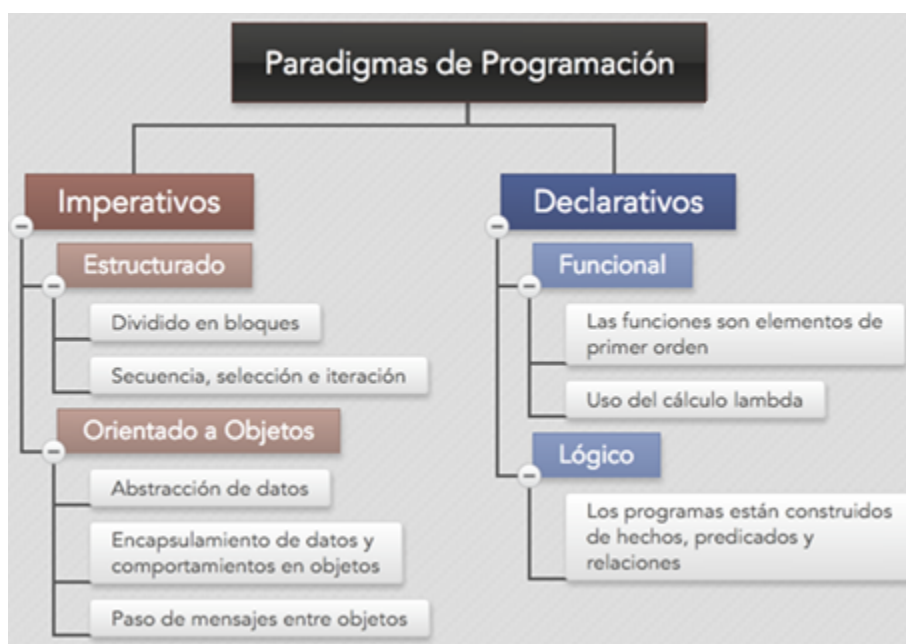
1. Técnicas como las del Diseño Estructurado y el desarrollo descendente (*topdown*).
2. Herramientas de Ingeniería de Software asistida por computadora conocida como CASE.
3. Desarrollo de lenguajes de programación más poderosos como los lenguajes de cuarta generación ([4GL](#)) y los orientados a objetos.
4. Diversas herramientas como gestión de proyectos, gestión de la configuración, ayuda en las pruebas, bibliotecas de clases de objetos, entre otras.

Haremos más énfasis en los lenguajes de programación, sobre todo en la Programación Orientada a Objetos (POO de aquí en adelante). Un lenguaje en términos generales, se puede entender como “.. *sistemas de símbolos y signos convencionales que son aceptados y usados individual y socialmente, con el fin de comunicar o expresar sentimientos, ideas, conocimientos, etc., por ejemplo, el lenguaje natural o articulado, el corporal, el artificial o formal, los sistemas de señalamiento, el arte, entre muchos otros tipos..*”. Existen también los lenguajes artificiales, que entre sus características, tenemos el que no es ambiguo y es universal, entre los que se encuentran los de las Matemáticas y los de Programación. En los lenguajes de programación, existen diferentes clasificaciones, una de ellas es la que a continuación se muestra:

1. **Programación Imperativa**, donde el programa es una serie de pasos, realizando en cada uno de ellos un cálculo (como ejemplos están: Cobol, Fortran entre otros).

2. **Programación Funcional**, el programa es un conjunto de funciones matemáticas que se combinan (como Lisp, Scheme, etc.).
3. **Programación Lógica** (también conocida como declarativa), aquí el programa es una colección de declaraciones lógicas (ejemplo Prolog).
4. **Programación Concurrente**, la programación consiste en un grupo de procesos corporativos, que llegan a compartir información ocasionalmente entre ellos (ejemplos LINDA y Fortran de alto rendimiento HPF 1995).
5. **Programación guiada por eventos**, el programa consiste en un ciclo continuo que va a responder a los eventos generados aleatoriamente (orden no predecible), ya que dichos eventos son originados a partir de acciones del usuario en la pantalla (ejemplos JAVA y Visual Basic).
6. **Programación Orientada a Objetos** (POO), el programa está compuesto por varios objetos que interactúan entre ellos a través de mensajes, los cuales hacen que cambien su estado (ejemplos C++, Eiffel y JAVA).

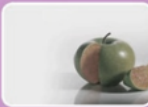


Existen muchas formas de clasificar los paradigmas de programación incluyendo más o menos programas y niveles, por ejemplo:



Esquema de los paradigmas de la programación

Introducción

La Programación Orientada a Objetos es un conjunto de reglas y principios de programación (o sea, un paradigma de programación) que busca representar las entidades u objetos del dominio (o enunciado) del problema dentro de un programa, en la forma más natural posible.

	<i>Es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas de computadora.</i>
	<i>Es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real.</i>
	<i>Se basa en la idea natural de la existencia de un mundo lleno de objetos, de modo que la resolución del problema se realiza en términos de objetos.</i>

Características de la POO

En el paradigma de programación tradicional o estructurado, que es el que se usaba anteriormente, el programador busca identificar los procesos (en forma de subproblemas) que aplicados sobre los datos permiten obtener los resultados buscados. Esta forma de proceder no es en absoluto incorrecta: la estrategia de descomponer un problema en subproblemas es una técnica elemental de resolución de problemas que los programadores orientados a objetos siguen usando dentro del nuevo paradigma. Pero entonces, ¿qué es lo nuevo del paradigma de la POO?.

La sola orientación a la descomposición en subproblemas no alcanza cuando el sistema es muy complejo. Se vuelve difícil de visualizar su estructura general y se hace complicado realizar hasta las más pequeñas modificaciones sin que estas repercutan en la lógica de un número elevado de otras rutinas. Finalmente, se torna una tarea casi imposible replantear el sistema para agregar nuevas funcionalidades complejas que permitan que ese sistema simplemente siga siendo útil frente a continuas nuevas demandas.

La POO significó una nueva visión en la forma de programar, buscando aportar claridad y naturalidad en la manera en que se plantea un programa. Ahora, el objetivo primario no es identificar procesos sino identificar actores: las entidades u objetos que aparecen en el escenario o dominio del problema, tales que esos objetos tienen no sólo datos asociados sino también algún comportamiento que son capaces de ejecutar.

Pensemos en un objeto como en un robot virtual: el programa tendrá muchos robots virtuales (objetos de software) que serán capaces de realizar eficiente y prolijamente ciertas tareas en las que serán expertos, e interactuando con otros robots virtuales (objetos) serán capaces de resolver el problema que el programador esté planteando.

En síntesis, podemos afirmar que, “La programación Orientada a Objetos es una metodología que basa la estructura de los programas en torno a los objetos”.

Los lenguajes de POO ofrecen medios y herramientas para describir los objetos manipulados por un programa. Más que describir cada objeto individualmente, estos lenguajes proveen una construcción (Clase) que describe a un conjunto de objetos que poseen las mismas propiedades y comportamientos.

Se deja a continuación un link explicativo:

<https://youtu.be/r0KjrzbsRec>

Algunas ventajas del paradigma orientado a objetos son:

- Es una forma más natural de modelar los problemas.
- Permite manejar mejor la complejidad.
- Facilita el mantenimiento y extensión de los sistemas.
- Es más adecuado para la construcción de entornos GUI.
- Fomenta el re-uso, con gran impacto sobre la productividad y la confiabilidad.

Nociones Básicas UML

Antes de comenzar a entender los aspectos más importantes de POO, vamos a conocer un recurso que nos será útil durante todo el recorrido.

UML (Unified Modeling Language)

El lenguaje unificado de modelado (UML), sostiene la idea de que una imagen vale más que mil palabras. Por ello, nos permite crear y generar diagramas para forjar un lenguaje visual común en el complejo mundo del análisis y desarrollo de software. Estos diagramas también son comprensibles para los usuarios y quien desee entender un sistema.

Es importante aclarar que UML no es un lenguaje de programación, pero existen herramientas que se pueden usar para generar código en diversos lenguajes usando los diagramas UML. Además, UML guarda una relación directa con el análisis y el diseño orientados a objetos.

“No es una metodología, sino una notación para desarrollar modelos (...) UML es el estándar para visualizar, especificar, construir y documentar los artefactos de un sistema de software”

Se deja a continuación un link explicativo:

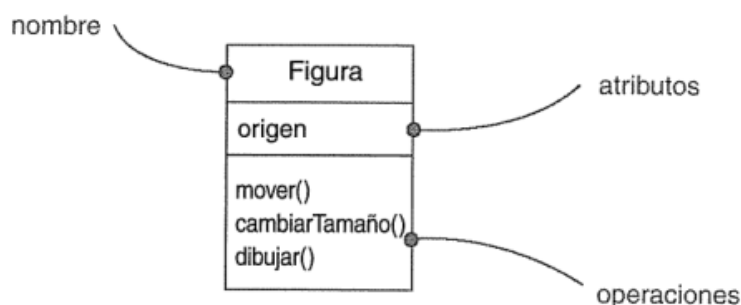
<https://youtu.be/MLytVFh6T1Y>

Un software que nos permite realizar Diagramas en UML, entre varios otros, es Lucidchart. Puedes registrarte en el siguiente link: <https://www.lucidchart.com>

Más adelante visualizaremos con más detalle todas las características de UML. Por ahora, vamos a conocer 2 tipos de diagramas para entender la POO (los Diagramas de Clases y los Diagramas de Secuencias).

Diagrama de Clases

El diagrama de clases es el diagrama UML más comúnmente usado, y la base principal de toda solución orientada a objetos. En él se presentan las clases dentro de un sistema, atributos y operaciones, y la relación entre cada clase. Las clases se agrupan para crear diagramas de sistemas grandes. *Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones.* Las relaciones pueden ser: de **dependencia**, **generalización** o **asociación**. Gráficamente un diagrama de clases es una colección de nodos y arcos. Una clase no es un objeto individual, sino que representa un conjunto de objetos. UML proporciona una representación gráfica de las clases, como muestra la siguiente figura.



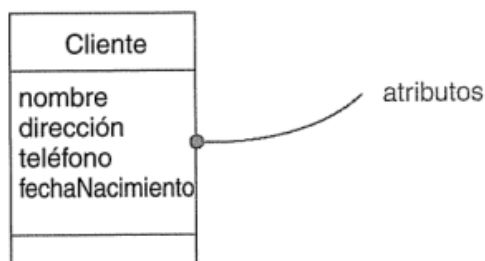
Representación gráfica de una clase en UML

Esta notación permite visualizar la abstracción de una clase independientemente de cualquier lenguaje de programación específico y de forma que permite resaltar las partes más importantes de la abstracción: su **nombre**, sus **atributos** y sus **operaciones**.

El nombre de una clase puede ser texto formado por cualquier número de letras, números y ciertos signos de puntuación. En la práctica, los nombres de clase son nombres cortos. Por convención, el nombre de la clase se pone en mayúsculas la primera letra de cada palabra, como *Cliente* o *SensorDeTemperatura*.

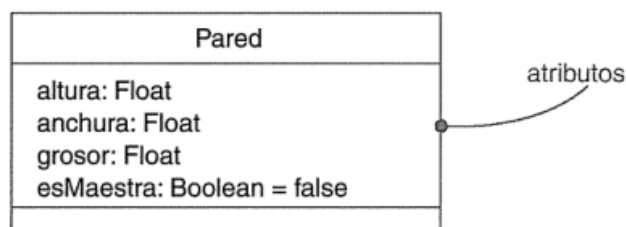
Atributos. Un atributo, es una propiedad de una clase, identificado con un nombre. Una clase puede tener cualquier número de atributos o ninguno. Un atributo representa alguna propiedad del elemento que se está modelando que es compartida por todos los objetos de

esa clase. Un atributo es, por tanto, una abstracción de un tipo de dato o estado que puede incluir un objeto de la clase. Los atributos se pueden representar mostrando solo sus nombres, como se ve en la siguiente figura.



Atributos de una clase

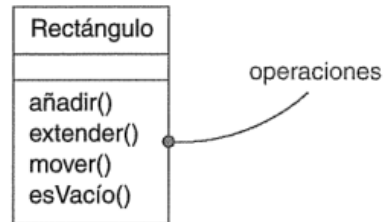
El nombre puede ser texto igual que el nombre de la clase, pero comienza con minúsculas. Por ejemplo: *nombre* o *fechaNacimiento*. Un atributo se puede especificar aún más indicando su tipo de dato y quizás un valor inicial por defecto, como se muestra en la siguiente figura.



Atributos con tipo de dato y valor inicial

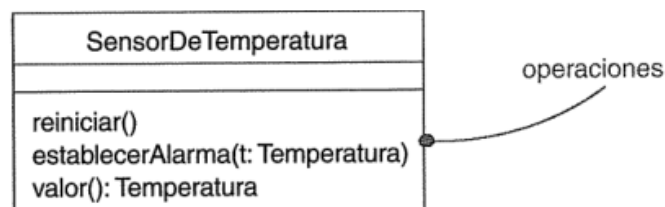
Como se ve hay tres atributos de tipo Float, uno de tipo Boolean que además está inicializado en *false*.

Operaciones. Una operación es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento. En otras palabras, una operación (o método) es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase. A menudo (pero no siempre), la invocación de una operación sobre un objeto cambia los datos o el estado del objeto. Las operaciones se pueden representar mostrando solo sus nombres, como se ilustra en la siguiente figura.



Operaciones de una clase.

Nótese que la convención de nombres es similar al de los atributos. Además, se puede incluir el tipo y valores por defecto de todos los parámetros y en el caso de las funciones un tipo de retorno, como se muestra en la siguiente figura.



Operaciones y sus signatures

Nótese que la operación *establecerAlarma* recibe un parámetro llamado "t" que es de tipo "Temperatura". La operación *valor* por su parte retorna un dato de tipo: Temperatura.

Cuando se dibuja una clase, no hay porque mostrar todos los atributos y todas las operaciones de una vez. De hecho, en la mayoría de los casos, no se puede (hay demasiados para ponerlos en la figura) y posiblemente no se debería hacer (probablemente solo será relevante un subconjunto de esos atributos y operaciones para una vista determinada). Un compartimento vacío no significa necesariamente que no haya operaciones o atributos, solo se ha decidido no mostrarlos. Se puede especificar explícitamente que más atributos u operaciones que los mostrando acabando la lista con puntos suspensivos.

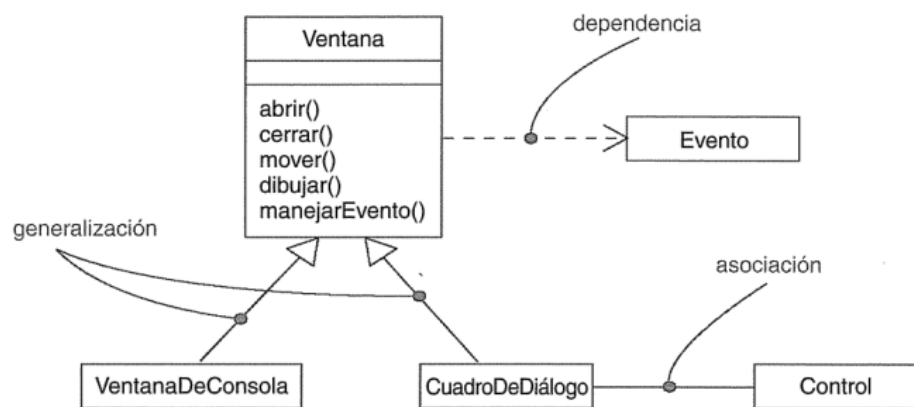
Algunas clases pueden ser abstractas, porque en nuestro sistema no se crearía una instancia de ella. El nombre de una clase abstracta puede colocarse en Cursiva, o entre corchetes angulares: <<nombre>>.

Relaciones entre clases

Al realizar abstracciones, uno se da cuenta de que muy pocas clases se encuentran aisladas. La mayoría colaboran con otras de varias maneras. Por tanto, al modelar un sistema, no solo hay que identificar los elementos que conforman el vocabulario del sistema, también hay que modelar cómo se relacionan estos elementos entre sí.

En el modelado orientado a objetos hay tres tipos de relaciones especialmente importantes: **dependencias**, que representan relaciones de uso entre clases; **generalizaciones**, que conectan clases generales con sus especializaciones, y **asociaciones**, que representan relaciones estructurales entre objetos.

UML proporciona una representación gráfica para cada uno de estos tipos de relaciones, como se muestra en la siguiente figura.



Relaciones entre clases

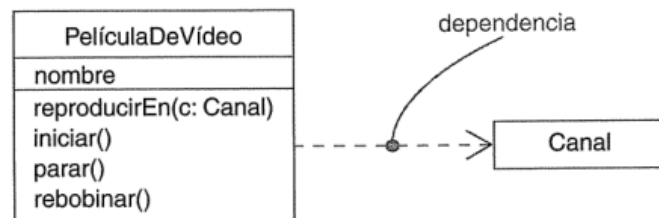
Esta notación permite visualizar *relaciones* independientemente de cualquier lenguaje de programación específico, y de forma que permite destacar las partes más importantes de una relación: su nombre, los elementos que conecta y sus propiedades.

Dependencia

Una *dependencia* es una relación de uso que declara que un elemento (por ejemplo, la clase **Ventana**) utiliza la información y los servicios de otro elemento (por ejemplo, la clase

Evento), pero no necesariamente a la inversa. Gráficamente, una dependencia se representa como una línea discontinua dirigida hacia el elemento del cual se depende. Las dependencias se usarán cuando se quiera indicar que un elemento utiliza a otro.

La mayoría de las veces, las dependencias se utilizarán en el contexto de las clases, para indicar que una clase utiliza las operaciones de otra o que utiliza variables o parámetros cuyo tipo viene dado por la otra clase. Véase la siguiente figura.

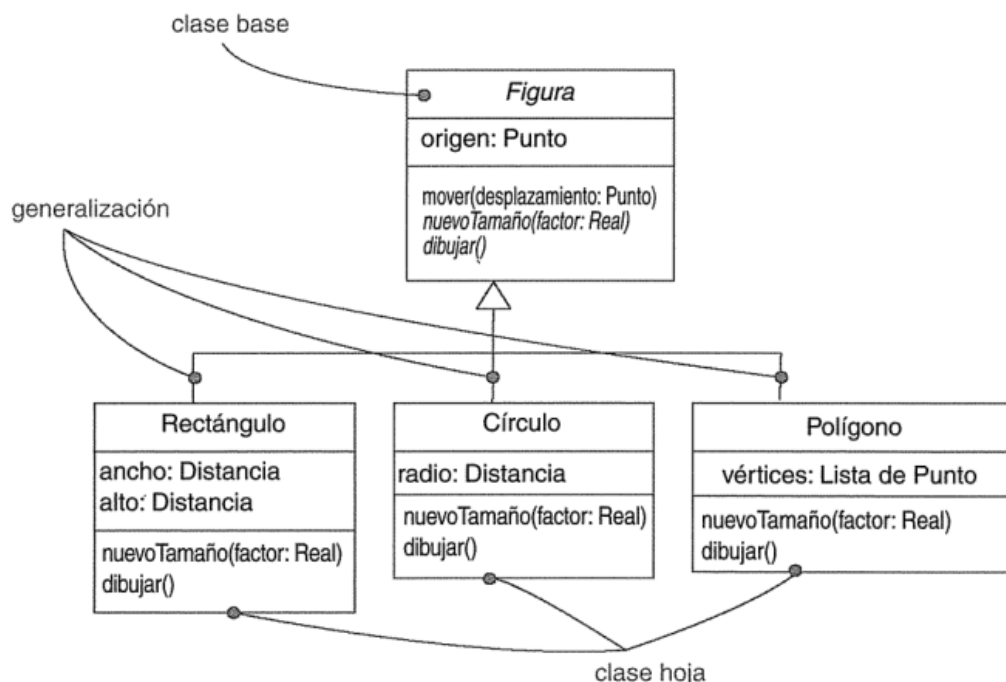


Dependencia entre clases

Esto es claramente una relación de uso. Si la clase utilizada cambia, la operación de la otra clase puede verse también afectada, porque la clase utilizada puede presentar ahora una interfaz o un comportamiento diferentes.

Generalización

Una generalización es una relación entre un elemento general (llamado superclase o padre) y un caso más específico de ese elemento (llamado subclase o hijo). Un hijo hereda las propiedades de sus padres, especialmente sus atributos y operaciones (métodos). A menudo (pero no siempre) el hijo añade atributos y operaciones a los que hereda de sus padres. Una implementación de una operación en un hijo redefine la implementación de la misma operación en el padre; esto se conoce como polimorfismo. Gráficamente, la generalización se representa como una línea dirigida continua, con una gran punta de flecha vacía que señala a la superclase. Véase la siguiente figura.



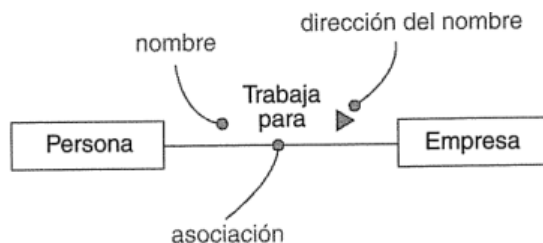
Generalización o herencia entre clases

Asociación

Una *asociación* es una relación estructural que especifica que los objetos de un elemento están conectados con los objetos de otro. Dada una asociación entre dos clases, se puede establecer una relación desde un objeto de una clase hasta algunos objetos de la otra clase. Gráficamente, una asociación se representa como una línea continua que conecta la misma o diferentes clases.

Además de la forma básica, hay adornos que pueden aplicarse a las asociaciones como el nombre y la multiplicidad.

Nombre. Una asociación puede tener (o no) un nombre, que se utiliza para describir la naturaleza de la relación. Para que no haya ambigüedad en su significado, se puede dar una dirección al nombre por medio de una flecha que apunte en la dirección en la que se pretende que lea el nombre, como se muestra en la siguiente figura.



Nombres de asociaciones

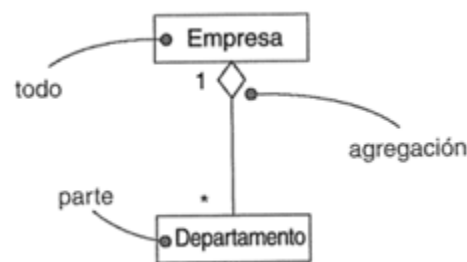
Multiplicidad. Una asociación representa una relación estructural entre dos objetos. En muchas situaciones de modelado, es importante señalar cuántos objetos pueden conectarse a través de una instancia de una asociación. Este “cuántos” se denomina multiplicidad de rol de la asociación, y representa un rango de enteros que especifican el tamaño posible del conjunto de objetos relacionados. La multiplicidad se escribe como una expresión con un valor mínimo y un valor máximo, que pueden ser iguales; se utilizan dos puntos consecutivos para separar ambos valores. Cuando se indica una multiplicidad en un extremo de una asociación, se está especificando cuántos objetos de la clase de ese extremo puede haber para cada objeto de la clase en el otro extremo. Se puede indicar una multiplicidad de exactamente uno (1), cero o uno (0..1), muchos (0..*), o uno o más (1..*). Se puede dar un rango de enteros (como 2..5). Incluso se puede indicar un número exacto (por ejemplo, 3, que equivale a 3..3).



Multiplicidad

En el gráfico anterior, se muestra que cada objeto empresa tiene como empleados a uno o más objetos persona (multiplicidad 1..*); cada objeto persona tiene como patrón a cero o más objetos empresa (multiplicidad *, lo que equivale a 0..*).

Agregación. Una asociación normal entre dos clases representa una relación estructural entre iguales, es decir, ambas clases están conceptualmente en el mismo nivel, sin ser ninguna más importante que la otra. A veces, se desea modelar una relación “todo/parte”, en la cual una clase representa una cosa grande (el “todo”), que consta de elementos más pequeños (las “partes”). Este tipo de relación se denomina agregación, la cual representa una relación del tipo “tiene-un”, o sea, un objeto del todo tiene objetos de la parte. En realidad, la agregación es solo un tipo especial de asociación y se especifica añadiendo a una asociación normal un rombo vacío en la parte del todo, como se muestra en la siguiente figura.

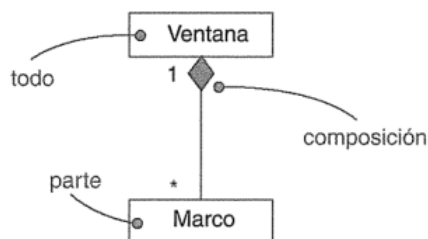


Agregación

El rombo vacío distingue el “todo” de la “parte”, ni más ni menos. Esto significa que la agregación no cambia el significado de la navegación a través de la asociación entre el todo y sus partes, ni liga la existencia del todo y sus partes.

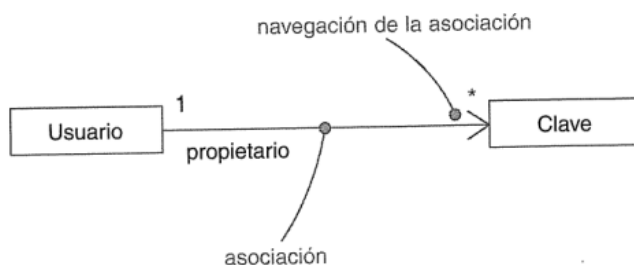
Composición. La agregación resulta ser un concepto simple con una semántica profunda. Es puramente conceptual y no hace más que distinguir un “todo” de una “parte”. Sin embargo, existe una variación de la agregación: la composición.

Las partes con una multiplicidad no fijada pueden crearse después de la parte compuesta a la que pertenecen, pero una vez creadas viven y mueren con ella. Es decir que en esta relación, un objeto derivado no puede existir sin su objeto principal. Como se ve en la siguiente figura, la composición se especifica adornando una asociación simple con un rombo relleno en el extremo del todo.



Composición

Navegación. Dada una asociación simple, sin adornos, entre dos clases, como Libro y Biblioteca, es posible navegar de los objetos de un tipo a los del otro tipo. A menos que se indique lo contrario, la navegación a través de una asociación es bidireccional. Pero hay circunstancias en las que se desea limitar la navegación a una sola dirección. Por ejemplo como se ve en la siguiente figura.



Navegación

Al modelar los servicios de un sistema operativo, se encuentra una asociación entre objetos Usuario y Clave. Dado un Usuario, se pueden encontrar las correspondientes Claves; pero, dada una Clave, no se podrá identificar al Usuario correspondiente. Se puede representar de forma explícita la dirección de la navegación con una flecha que apunte en la dirección de recorrido.

Diagrama de objetos

El diagrama de objetos muestra la relación entre objetos por medio de ejemplos del mundo real e ilustra cómo se verá un sistema en un momento dado. Representa a un objeto instanciado (creado en tiempo de ejecución) dentro de una clase, estableciendo valores a

los atributos. Dado que los datos están disponibles dentro de los objetos, estos pueden usarse para clarificar relaciones entre objetos.

Los componentes básicos son:

Objetos: Los objetos son instancias de una clase. Por ejemplo, si "Auto" es una clase, un 208 compacto de Peugeot es un objeto de una clase.

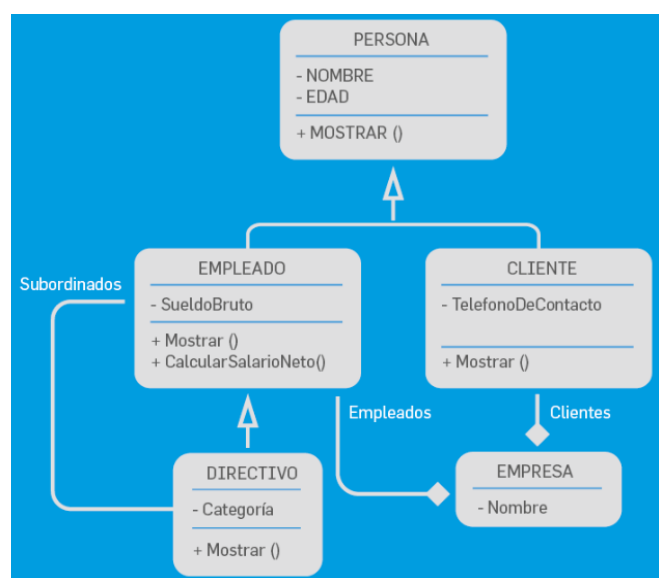
Títulos de clase: Los títulos de clases son los atributos específicos de una clase dada. Se pueden listar títulos de clases como elementos en el objeto o incluso en las propiedades del propio objeto (como el color).

Atributos: Los atributos de clases se representan por medio de un rectángulo con dos pestañas que indica un elemento de software.

Enlaces: Los enlaces son líneas que conectan dos figuras de un diagrama de objetos entre sí (debe nombrarse la interacción).

También, al igual que en el diagrama de clases, se puede agregar una tercera sección donde estén indicados los métodos u operaciones.

"El diagrama de objetos también es conocido como diagrama de instancias".



Elementos de un Diagrama de Objetos

Diagrama de secuencias

El diagrama de secuencia es un esquema conceptual que permite representar el comportamiento de un sistema. Muestra cómo los objetos interactúan entre sí y el orden de la ocurrencia. Representan interacciones para un escenario concreto.

Los componentes básicos son:

1. **Objetos:** se representa del modo usual (como en el diagrama de objetos)
2. **Líneas de vida:** representa un participante individual en un diagrama de secuencia
3. **Mensaje:** el mensaje que va de un objeto a otro pasa de la línea de vida de un objeto al de otro. Hay diferentes tipos de mensajes como: simple, sincrónico, asincrónico.
4. **Dimensiones:** Horizontal, que es la disposición de los objetos, y vertical que muestra el paso del tiempo.

Objetos

Ya presentamos el funcionamiento básico de UML que nos ayudará a representar mejor algunos conceptos básicos de POO. Antes de desarrollar con profundidad cada uno de los conceptos, veamos algunas características de los Lenguajes Orientados a Objetos.

Según [Alan Kay](#) (1993), las características son seis:

1. Todo es un objeto.
2. Cada objeto es construido a partir de otros objetos.
3. Todo objeto es instancia de una clase.
4. Todos los objetos de la misma clase pueden recibir los mismos mensajes (realizar las mismas acciones). La clase es el lugar donde se define el comportamiento de los objetos y su estructura interna.
5. Las clases se organizan en una estructura arbórea de raíz única, llamada jerarquía de herencia. *Ejemplo:* puesto que un círculo es una forma, un círculo siempre aceptará todos los mensajes destinados a una forma.

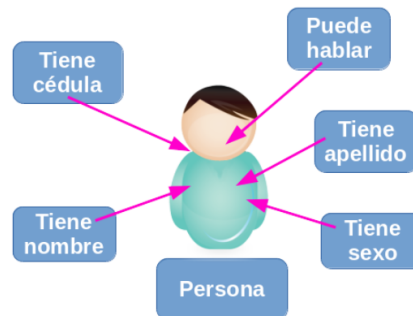
6. Un programa es un conjunto de objetos que se comunican mediante el paso de mensajes.

Ahora analizaremos en profundidad cada término mencionado con más detalle.

Objetos

Los Objetos, se pueden definir como las unidades básicas de construcción, para la conceptualización, diseño o programación. Son instancias agrupadas en clases con características en común que son los atributos y procedimientos, conocidos como operaciones o métodos. También se puede decir, que un objeto es una abstracción encapsulada genérica de datos y los procedimientos para manipularlos.

"Un objeto es una cosa o entidad, que tiene atributos (propiedades) y de formas de operar sobre ellos que se conocen como métodos."



Atributos y Métodos de un objeto

De una forma más simple se puede decir que un objeto es un ente que tiene características y comportamiento. Los objetos pueden modelar diferentes cosas como; dispositivos, roles, organizaciones, sucesos, ventanas (de Windows), iconos, etc.

Las características generales de los objetos son:

1. Se identifican por un nombre o identificador único que los diferencia de los demás.
2. Poseen estados.
3. Poseen un conjunto de métodos.
4. Poseen un conjunto de atributos.
5. Soportan el encapsulamiento.
6. Tienen un tiempo de vida.

7. Son instancias de una clase.

Organización de los objetos

Los objetos forman siempre una organización jerárquica, existiendo varios tipos de jerarquía, por ejemplo:

- Simples: cuando su estructura es representada por medio de un árbol (estructura de datos).
- Compleja: cualquier otra diferente a la de árbol.

Sea cual fuere la estructura se tienen en ella los siguientes tres niveles de objetos:

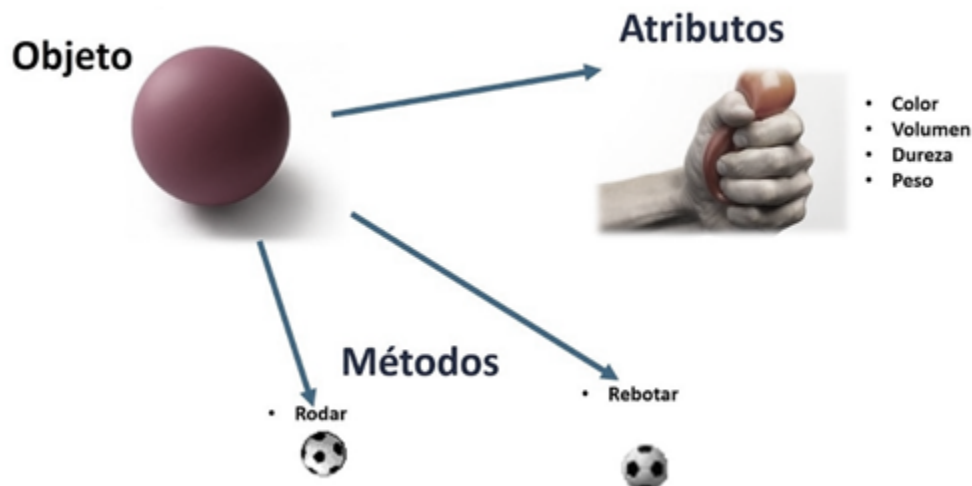
1. La raíz de la jerarquía: Es un objeto único, está en el nivel más alto de la estructura y se le conoce como objeto Padre, Raíz o Entidad.
2. Los objetos intermedios: Son los que descienden directamente de la raíz y que a su vez tienen descendientes (tienen ascendencia y descendencia) y representan conjuntos de objetos, que pueden llegar a ser muy generales o especializados, de acuerdo con los requerimientos de la aplicación.
3. Los objetos terminales: Son todos aquellos que tienen ascendencia, pero que no tienen descendencia.

Atributos de los objetos





Los atributos son las características del objeto (qué tiene, de qué consta), que se definen a través de tipos de datos o variables, los cuales se dividen en:

- ☐ Tipos de Datos Primitivos (TDP): nos sirven para representar tipos de datos como números enteros, caracteres, números reales, booleanos (verdadero/falso), etc. Una variable de tipo primitivo nos permite almacenar en ella un tipo primitivo como por ejemplo un valor numérico.
- ☐ Tipos de Datos por Referencia: indican que vamos a trabajar con instancias de clases, no con tipos primitivos. De esta manera, una variable de tipo referencia

establece una conexión hacia un objeto, y a través de esta conexión podemos acceder a sus métodos y atributos.



Atributos y Métodos de un objeto

Clase Persona		Objeto Persona		Objeto Persona 2		Objeto Persona 3	
Atributos	nombre	Atributos	nombre = Leo	Atributos	nombre = Lucas	Atributos	nombre = Blaise
	edad		edad = 32		edad = 30		edad = 34
	colorRemera		colorRemera = Azul		colorRemera = Rojo		colorRemera = Amarilla
	colorPantalon		colorPantalon = Azul		colorPantalon = Negro		colorPantalon = Blanco
	colorBotines		colorBotines = Negro		colorBotines = Verde		colorBotines = Negro
	colorPelo		colorPelo = Negro		colorPelo = Rubio		colorPelo = Blanco
	colorPiel		colorPiel = Clara		colorPiel = Clara		colorPiel = Oscura
Clase Molde o Plantilla		Objeto 1		Objeto 2		Objeto 3	
							

Clase e instancias

Más adelante desarrollaremos con mayor profundidad los tipos de variables y datos, viendo una clasificación más detallada. Se toma la convención de que, al utilizar estos atributos,



UNJu

Universidad
Nacional de Jujuy



Argentina
programa
4.0

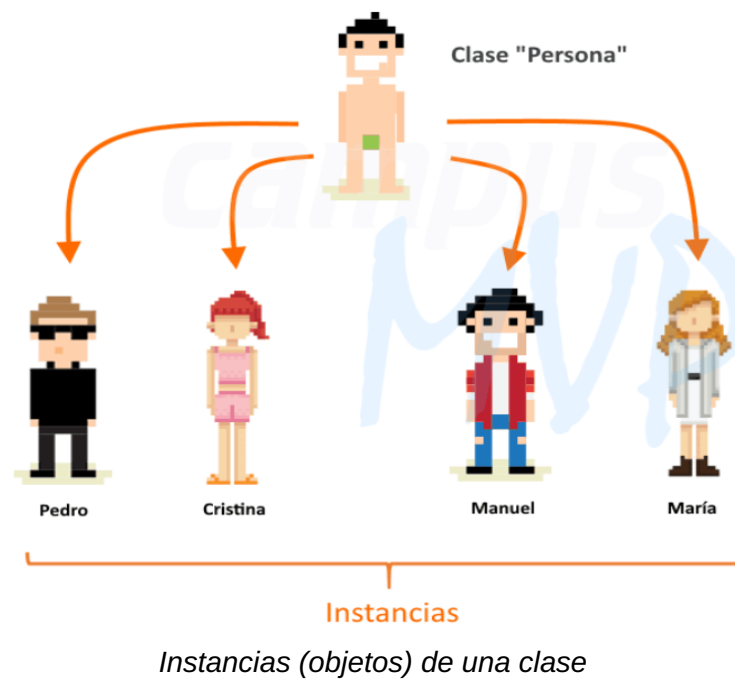
se utilicen siempre con letras minúsculas y sólo si se trata de más de una palabra, entonces será con mayúscula cada primera letra de cada palabra adicional y siempre juntas, sin espacios en blanco, por ejemplo: colorDelPelo.

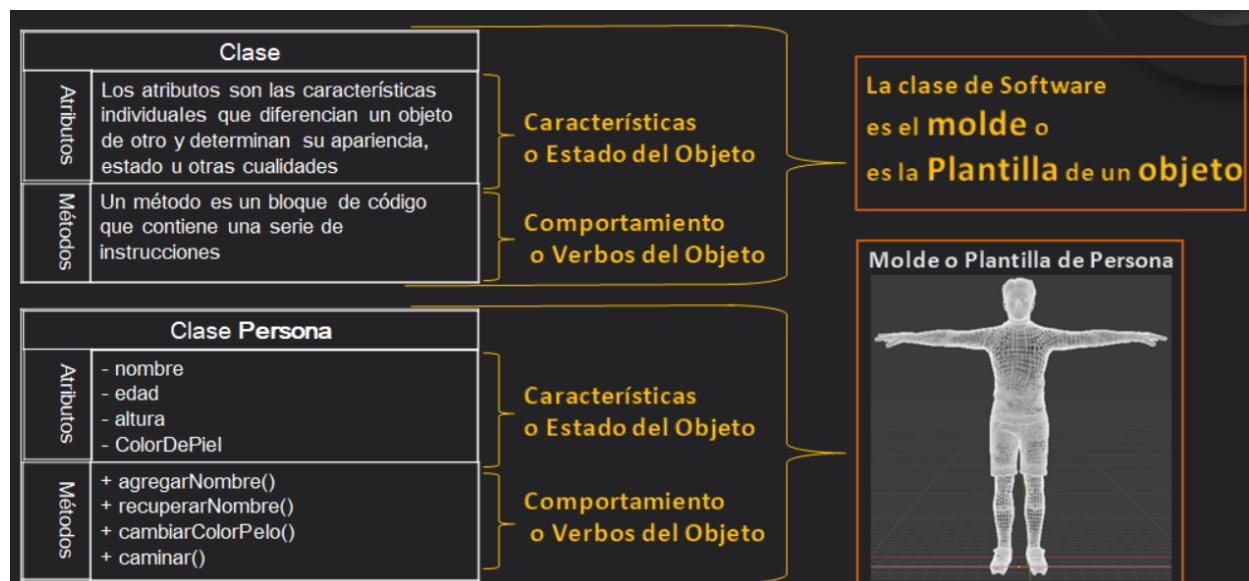
Clases

Es la generalización de un tipo específico de objetos, como el conjunto de características (atributos) y comportamientos de todos los objetos que componen a la clase. Una *clase* es una *plantilla* mediante la cual se crean los diferentes objetos requeridos para la solución del problema. Los *objetos* son instancias de las clases.

Ejemplo 1, la clase “Plumón” tiene todas las características (tamaño, color, grosor del punto, etc) y todos los métodos o acciones (pintar, marcar, subrayar, etc), que pueden tener todos los plumones existentes en la realidad. Un plumón en especial como un “Marcador” para discos compactos (CDs) de color negro y punto fino, es un objeto (o instancia) de la clase plumón.

Ejemplo 2: Se puede crear un objeto llamado *Pedro*. Este objeto es creado a partir de la clase *Persona*. Se puede crear otro objeto llamado *María* el cual pertenece a la clase *Persona*. Significa que a partir de la clase se pueden crear los objetos que se deseen.





Clase e instancia

Cuando se define una clase se especifica cómo serán los objetos de dicha clase, esto quiere decir, de qué variables (o atributos) y de qué métodos (comportamiento) estará constituida. Las clases proporcionan una especie de plantilla o molde para los objetos, como podría ser el molde para hacer helados de hielo, los objetos que se crean son en sí los helados de hielo, estos helados van a tener como atributos (características), su color, tamaño, sabor, etc y como acciones o métodos, que se pueden congelar, derretir, etc.

Características generales de las clases:

1. Poseen un nivel de abstracción alto.
2. Pueden relacionarse mediante jerarquías.
3. Los nombres de las clases deben estar en singular.

La representación gráfica de una o varias clases se hará mediante los denominados Diagramas de Clase. Para los diagramas de clase se utilizará la notación que provee el Lenguaje de Modelación Unificado (UML, que vimos previamente) a saber:

Las clases se denotan como rectángulos divididos en tres partes. La primera contiene el nombre de la clase, la segunda contiene los atributos y la tercera los métodos.

Los modificadores de acceso a datos y operaciones, a saber: público, protegido y privado; se representan con los símbolos +, # y – respectivamente, al lado izquierda del atributo. (+ *público*, # *protegido*, - *privado*).

Se deja a continuación el link de un video explicativo:

<https://youtu.be/aSUvGPB6oE8>

Relaciones entre clases

Las clases no se construyen para que trabajen de manera aislada, la idea es que ellas se puedan relacionar entre sí, de manera que puedan compartir atributos y métodos sin necesidad de volver a escribirlos.

La posibilidad de establecer jerarquías entre las clases es una característica que diferencia esencialmente la programación orientada a objetos de la programación tradicional, debido fundamentalmente a que permite extender y reutilizar el código existente sin tener que volver a escribirlo cada vez que se necesite.

Los tipos de relaciones entre clases que veremos son:

☐ **Dependencia** (Usa un)

☐ **Herencia** (generalización / especialización o Es-un).

Superclase: se puede definir en términos sencillos como la clase padre de alguna clase específica y puede tener cualquier número de subclases.

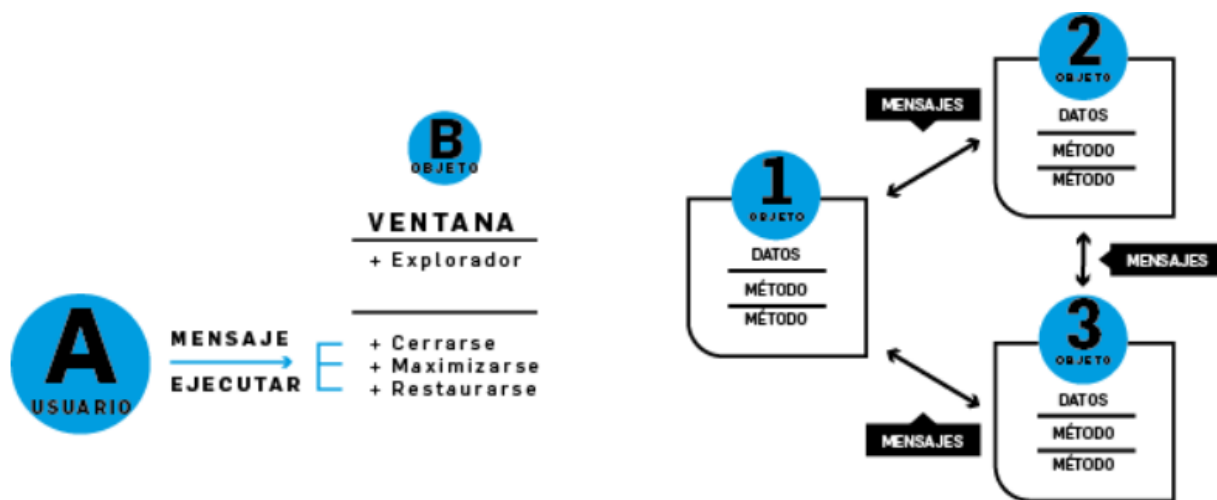
Subclase: es la clase hija de alguna clase específica.

☐ **Asociación** (entre otras, la relación Tiene-un): **Agregación** (todo / parte o Forma-parte-de) y **Composición** (Es parte elemental de).

Mensajes. Métodos. Variables e instancias

Mensajes

En POO, los objetos descritos anteriormente se comunican a través de señales o mensajes, siendo estos mensajes los que hacen que los objetos respondan de diferentes maneras, por ejemplo, un objeto en Windows como una ventana de alguna aplicación, puede cerrarse, maximizarse o restaurarse (métodos) de acuerdo con el mensaje que le sea enviado. En otras palabras, un mensaje es una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos pudiendo o no llevar algunos parámetros



Ejemplo de comunicación entre objetos

Métodos

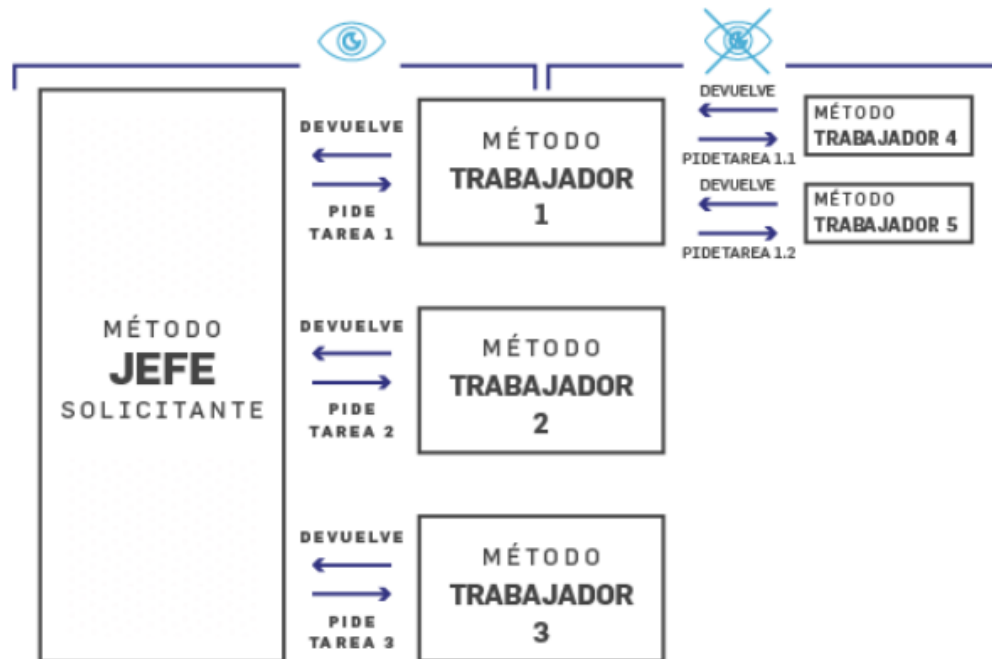
Los métodos (también conocidos como funciones o procedimientos en otros lenguajes y como operaciones en UML) permiten al programador dividir un programa en módulos, por medio de la separación de sus tareas en unidades autónomas. Las instrucciones en los cuerpos de los métodos se escriben sólo una vez, y se reutilizan tal vez desde varias ubicaciones en un programa; además, están ocultas de otros métodos. Una razón para dividir un programa en módulos mediante los métodos es la metodología “divide y vencerás”, que hace que el desarrollo de programas sea más fácil de administrar, ya que

se pueden construir programas a partir de piezas pequeñas y simples. Otra razón es la reutilización de software (usar los métodos existentes como si fueran bloques de construcción para crear nuevos programas). A menudo se pueden crear programas a partir de métodos estandarizados, en vez de tener que crear código personalizado. Una tercera razón es para evitar la repetición de código. El proceso de dividir un programa en métodos significativos hace que el programa sea más fácil de depurar y mantener.

Un método se invoca mediante una llamada, y cuando el método que se llamó completa su tarea, devuelve un resultado, o simplemente el control al método que lo llamó. Una analogía a esta estructura de programa es la forma jerárquica de la administración: Un jefe (el solicitante) pide a un trabajador (el método llamado) que realice una tarea y que le reporte (devuelva) los resultados después de completar la tarea. El método jefe, no sabe cómo el método trabajador, realiza sus tareas designadas. Tal vez el trabajador llame a otros métodos trabajadores, sin que lo sepa el jefe. Este “ocultamiento” de los detalles de implementación fomenta la buena ingeniería de software.

Los métodos interactúan con los mensajes, determinando cuáles son los que un objeto puede recibir. Haciendo una analogía con la programación procedural, se podría comparar con las funciones o subrutinas.

En el gráfico siguiente se muestra al método jefe comunicándose con varios métodos trabajadores en forma jerárquica. El método jefe divide las responsabilidades entre los diversos métodos trabajadores. Observe que trabajador 1 actúa como “método jefe” de trabajador 4 y trabajador 5.



Ejemplo de comunicación entre objetos

Un método cuenta con las siguientes partes:

- Nombre del método.
- Valor que regresa.
- Argumentos opcionales.
- Modificadores en la declaración del método (como los de visibilidad)
- Cuerpo del método.

Variables

Las variables son espacios en la memoria del ordenador, en las que se almacenan los datos, cada una tiene su propio nombre, tipo y valor. Los tipos de variables que vamos a ver son de:

- **Instancia:** se ocupan para definir los atributos de un objeto.
- **Clase:** son variables en las que sus valores son idénticos para todas las instancias de la clase.
- **Locales:** son las variables que se declaran y se utilizan cuando se definen los métodos.

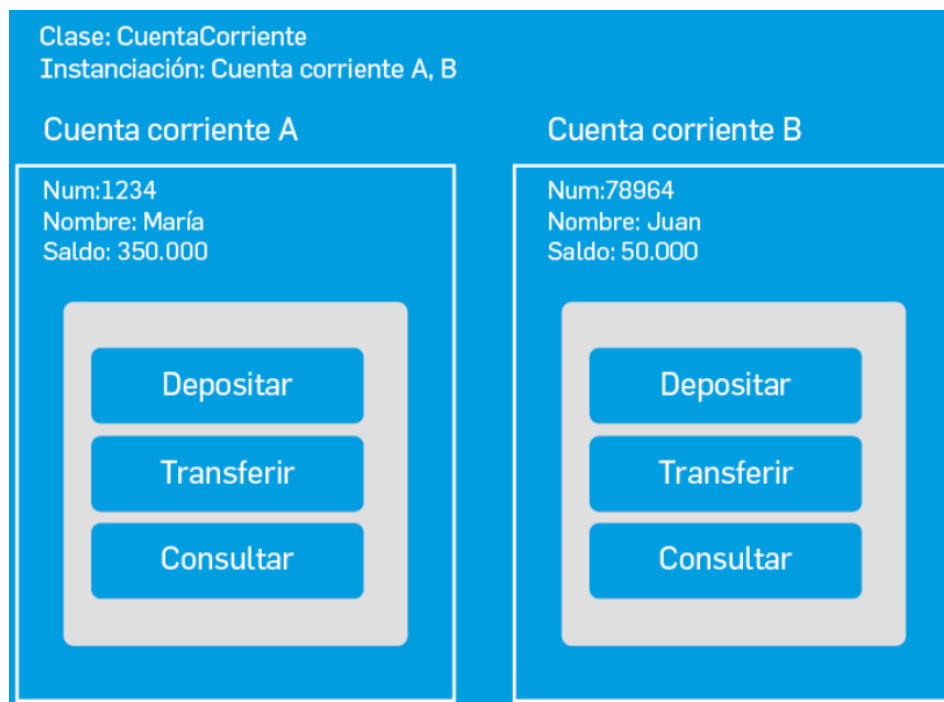
El nombre de la variable será algún identificador válido (de acuerdo con lo explicado anteriormente) y con él se hace la referencia a los datos que contienen la variable. El tipo de una variable indica qué valores puede manejar y a las operaciones que se pueden hacer con ella, por ejemplo, si es una variable de tipo entero, se pueden realizar con ella las operaciones aritméticas, en tanto que, si fueran de tipo carácter, no se podrían hacer este tipo de operaciones. El valor que puede tomar es en sí la literal, que va a depender precisamente de su tipo.

Instancias

Hasta el momento se han abordado algunos aspectos concernientes a cómo se deben implementar las clases y, cómo expresar sus atributos y métodos.

Aún cuando todos los tipos de datos son clases, se denominan *objetos* a las *instancias* de las clases que excluyen los tipos de datos básicos del lenguaje. Cabe recordar que en el caso de los tipos de datos básicos sus valores pueden asociarse a la definición de variables y luego a través del operador de asignación, éstas cambiarían su estado.

¿Es factible seguir esta misma idea para manejar objetos? Para manipular objetos o instancias de tipos de datos que no sean básicos también se utilizan variables y éstas utilizan semántica por referencia, solo que con una diferencia con respecto a los tipos básicos, los objetos tienen que ser creados (al crear un objeto se reserva memoria para almacenar los valores de todos sus campos) o al menos contener la referencia de otro objeto creado con anterioridad.



Ejemplo de instancia de objetos

Declaración de variables de instancia

El estado de un objeto se representa por sus variables (conocidas como variables de instancia). Estas son declaradas dentro del cuerpo de la clase. A un objeto se le conoce como instancia de una clase y para crearlo se llama a una parte de líneas de código conocidas con el nombre constructor de una clase que tienen el mismo nombre de la clase. Una vez que ya no se ocupa el objeto, se ejecuta el recolector de basura (garbage collector). Estos conceptos de objeto, constructor y de recolector de basura, se abordarán más adelante con más detalle.