Introducción Orientación a Objetos

La Orientación a Objetos (O.O.) surge en Noruega en 1967 con un lenguaje llamado Simula 67, desarrollado por Krinsten Nygaard y Ole-Johan Dahl, en el centro de cálculo noruego. Simula 67 introdujo por primera vez los conceptos de clases, corrutinas y subclases (conceptos muy similares a los lenguajes Orientados a Objetos de hoy en día).

El nacimiento de la Orientación a Objetos en Europa pasó inadvertido para gran parte de los programadores. Hoy tenemos la Orientación a Objetos como un niño de 33 años al que todos quieren bautizar.

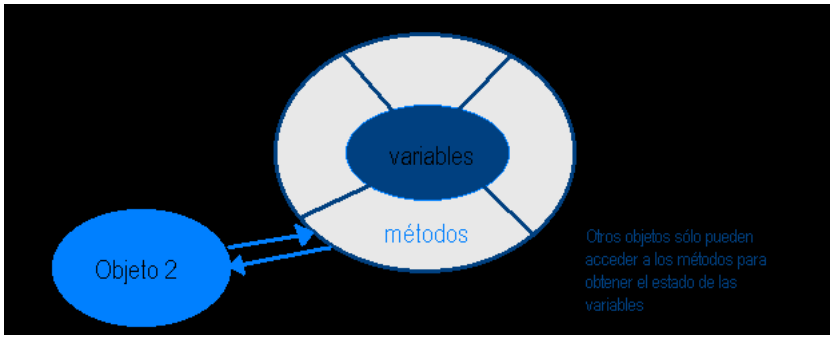
Uno de los problemas de inicio de los años setentas era que pocos sistemas lograban terminarse, pocos se terminaban con los requisitos iniciales y no todos los que se terminaban cumpliendo con los requerimientos se usaban según lo planificado. El problema consistía en cómo adaptar el software a nuevos requerimientos imposibles de haber sido planificados inicialmente.

Este alto grado de planificación y previsión es contrario a la propia realidad. El hombre aprende y crea a través de la experimentación, no de la planeación. La Orientación a Objetos brinda estos métodos de experimentación, no exige la planificación de un proyecto por completo antes de escribir la primer línea de código.

En los 70's científicos del centro de investigación en Palo Alto Xerox (Xerox park) inventaron el lenguaje Small talk que dio respuesta al problema anterior (investigar no planificar).

Small talk fue el primer lenguaje Orientado a Objetos puro de los lenguajes Orientados a Objetos, es decir, únicamente utiliza clases y objetos Hasta este momento uno de los defectos más graves de la programación es que las variables eran visibles desde cualquier parte del código y podían ser modificadas incluyendo la posibilidad de cambiar su contenido (no existen niveles de usuarios o de seguridad, o lo que se conoce como visibilidad).

Quien tuvo la idea fue D. Parnas cuando propuso la disciplina de ocultar la información. Su idea era encapsular cada una de las variables globales de la aplicación en un solo módulo junto con sus operaciones asociadas, sólo mediante las cuales se podí
tener acceso a esas variables.



El resto de los módulos (objetos) podían acceder a las variables sólo de forma indirecta mediante las operaciones diseñadas para tal efecto.

En los años 80's Bjarne Stroustrup de AT&T Labs., amplió el lenguaje C para crear C++ que soporta la programación Orientada a Objetos.

En esta misma década se desarrollaron otros lenguajes Orientados a Objetos como Objective C, Common Lisp Object System (CIOS), object Pascal, Ada y otros.

Posteriores mejoras en herramientas y lanzamientos comerciales de C++ por distintos fabricantes, justificaron la mayor atención hacia la programación Orientada a Objetos en la comunidad de desarrollo de software. El desarrollo técnico del hardware y su disminución del costo fue el detonante final. Con más computadoras al alcance de más personas más programadores, más problemas y más algoritmos surgieron.

En el inicio de los 90's se consolida la Orientación a Objetos como una de las mejores maneras para resolver problemas. Aumenta la necesidad de generar prototipos más rápidamente (concepto RAD Rapid Aplication Developments). Sin esperar a que los requerimientos iniciales estén totalmente precisos.

En 1996 surge un desarrollo llamado JAVA (extensión de C++). Su filosofía es aprovechar el software existente. Facilitar la adaptación del mismo a otros usos diferentes a los originales sin necesidad de modificar el código ya existente.

En 1997-98 se desarrollan herramientas 'CASE' orientadas a objetos (como el diseño asistido por computadora).

Del 98 a la fecha se desarrolla la arquitectura de objetos distribuidos RMI, Corba, COM, DCOM.

Actualmente la orientación a objetos parece ser el mejor paradigma, no obstante, no es una solución a todos los problemas. Trata de eliminar la crisis del software.

Entre los creadores de metodologías orientadas a objetos se encuentran: G. Booch, Rambaught, Ivar Jacobson y Peter Cheng.

CONCEPTOS FUNDAMENTALES DE LA ORIENTACIÓN A OBJETOS

– Programación estructurada y sus desventajas

La programación estructurada emplea la *técnica descenderte* o el *refinamiento sucesivo*, que comienza descomponiendo el programa en piezas manejables más pequeñas, conocidas como *funciones* (subrutinas, subprogramas o procedimientos), que realizan tareas menos complejas. Esta técnica introdujo el concepto de *abstracción* que se define como la capacidad para examinar algo sin preocuparse de sus datos internos.

En un programa estructurado, los *datos locales* se ocultan dentro de funciones y los *datos compartidos* se pasan como argumentos.

A medida que la complejidad de un programa crece, también crece su independencia de los tipos de datos fundamentales que procesa, provocando que el acceso a los mismos se convierta *crítico*.

Los programas basados en funciones son difíciles de diseñar. El problema es que sus componentes principales (funciones y estructuras de datos) no modelan bien el mundo real.

– **Pensando en objetos**

Supongamos que se desea construir una *computadora*, ¿qué componentes se necesitarían para construirla? Entre los componentes tenemos: una tarjeta madre, un chip CPU, una tarjeta de video, un disco duro, un teclado, etc. Lo ideal sería que cuando se ensamblen todos los componentes, se tenga un sistema en donde todos estos componentes se unan para crear un sistema más grande.

En su estructura interna, cada uno de estos componentes puede ser muy complicado y fabricarse por diferentes compañías con distintos métodos de diseño. Pero no se necesita saber como funcionan los componentes o que hace cada chip en la tarjeta para poder ensamblarla y que todas las unidades interactúen entre sí. ¿Encajara la tarjeta de video en las ranuras de la tarjeta madre?, ¿Funcionara el monitor con esta tarjeta de video?, una vez que conozca las interacciones que hay entre los componentes y la forma en que coincide con aquellas, es sencillo juntar el sistema completo.

¿Qué tiene que ver esto con la programación? Todo. La POO funciona de esta manera. Al utilizarla el programa general estará formado por componentes individuales (objetos) numerosos y diferentes; cada uno de los cuales realizara su papel en el programa y todos se comunicaran en formas predefinidas.

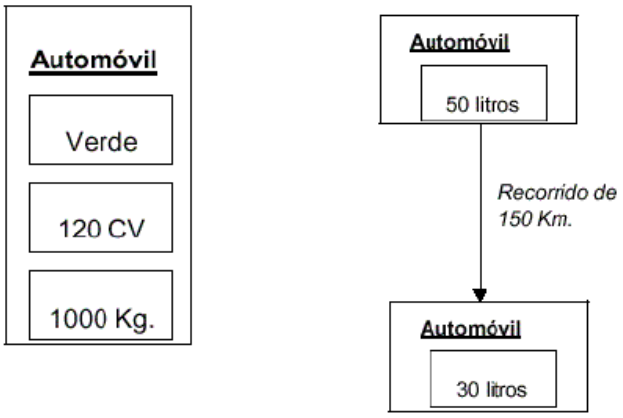
– **Concepto de objeto**

• Cualquier cosa, ocurrencia o fenómeno que puede ser identificado y caracterizado
• Entidad definida por un conjunto de atributos comunes y los servicios u operaciones asociados
• La representación abstracta del objeto informático es una imagen simplificada del objeto del mundo real.
• Se acostumbra a considerar los objetos como seres animados con vida propia (nacen, viven y mueren).
• Unidad atómica formada por la unión de *estado* y *comportamient*o.

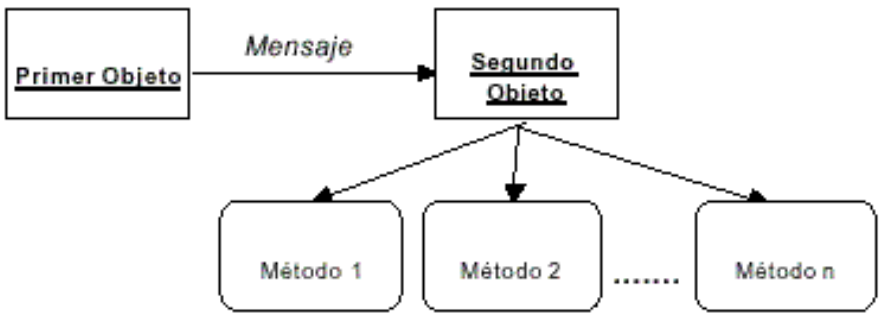
• La *encapsulación* proporciona una cohesión interna fuerte y un acoplamiento externo débil.
• Para manipular los objetos se utilizan los *mensaje*s.

– **El estado de un objeto**

• Contiene los valores de sus atributos (variables) cuya información cualifica al objeto.
• Existe un rango de valores para los atributos
• El estado en un instante dado corresponde a una selección de valores de entre todos los posibles en cada atributo.
• El estado evoluciona con el tiempo.
• Hay componentes constantes (marca del automóvil, CV, etc.).
• Generalmente el estado de un objeto es variable.

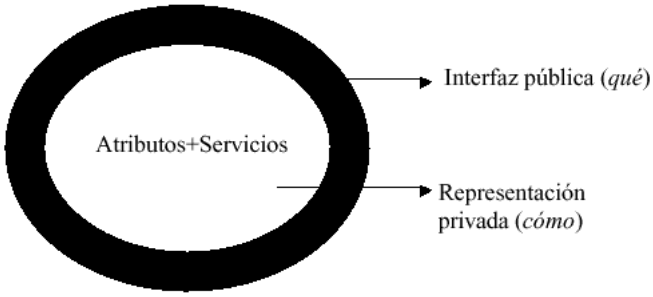


– **El comportamiento de un objeto**
• Describe las *acciones* y *reacciones* de los objetos.
• Cada operación es un átomo de comportamiento.
• Las operaciones se desencadenan por estímulos externos *(mensaje*s), enviados por otros objetos.
• El estado y el comportamiento están relacionados.

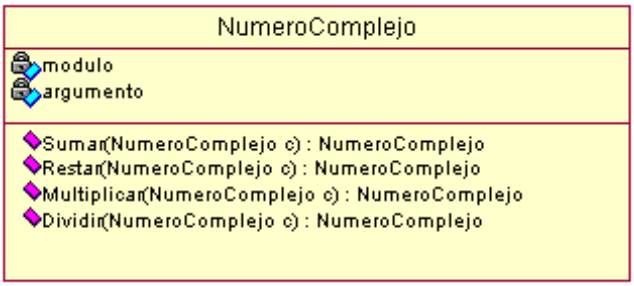


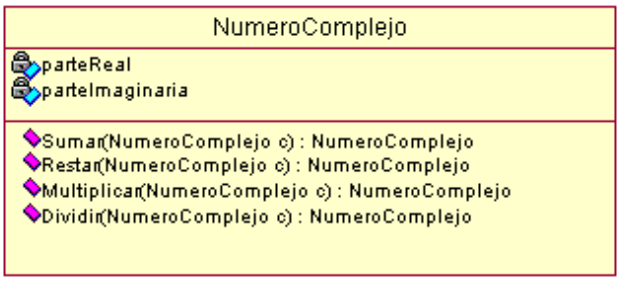
– **La identidad**
• Caracteriza la existencia de un objeto.
• Le distingue de los demás de manera no ambigua independientemente de su estado (dos objetos se pueden distinguir a pesar de tener atributos idénticos).
• En el modelado no se representa la identidad de manera específica.
• El concepto de identidad es independiente del de estado.

– **Encapsulación**
• Ocultar al exterior los detalles de implementación, de manera que el "mundo"sólo vea una interfaz inteligible (la *parte pública*).



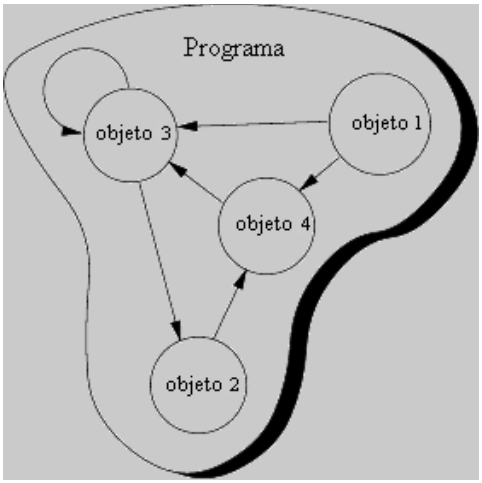
• Los clientes de una clase sólo conocen la interfaz de la misma, es decir, conocen los prototipos de las operaciones pero no cómo están implementadas.





– **¿Cómo es un programa orientado a objetos?**
• Ver un programa como una colección de objetos interactuantes es un principio fundamental en la programación orientada a objetos.

• Los objetos en esta colección reaccionan al recibir mensajes, cambiando su estatus de acuerdo a la invocación de métodos que a su vez pdrían causar que otros mensajes fueran enviados a otros objetos.

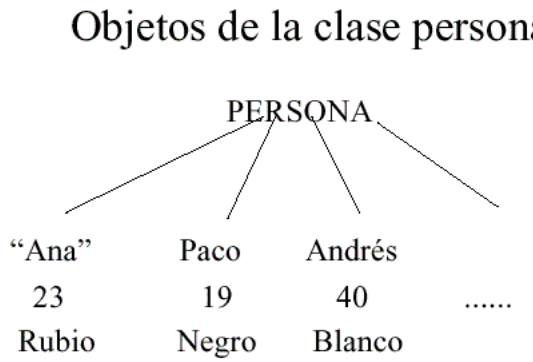


– **Clase**

- Plantilla a partir de la que se crean objetos. Contiene una definición del estado y los métodos del objeto.
- Módulo software que encapsula atributos, operaciones, excepciones y mensajes.

Un ejemplo: La clase "Persona"

• Atributos
– Nombre: string
– Fecha de nacimiento: fecha
– Color del pelo: (negro, blanco, pelirrojo, rubio)
• Métodos
– Nacer
– Crecer
– Morir

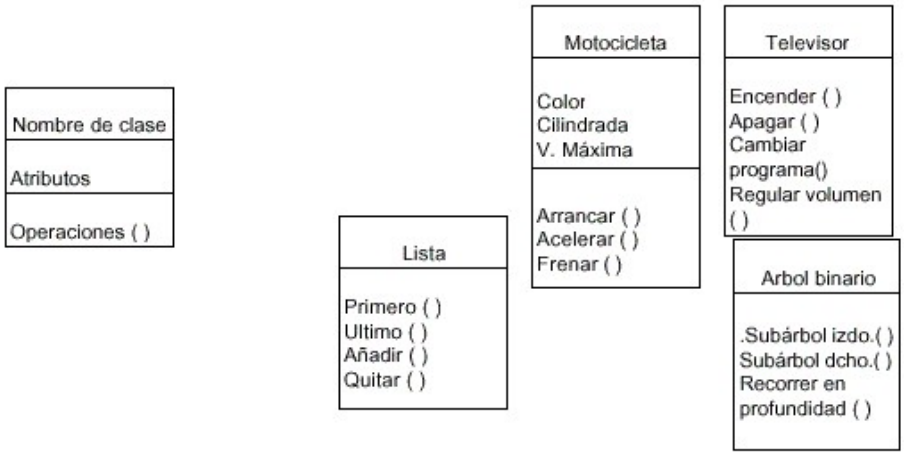


– **Más definiciones de clase**

• Conjunto de objetos que comparte una estructura y comportamiento comunes
• Conjunto de cosas que tienen el mismo comportamiento y características
• Instrumentación que puede ser instanciada para crear múltiples objetos que tienen el mismo comportamiento inicial
• La clase define el ámbito de definición de un conjunto de objetos.
• Cada objeto pertenece a una clase.
• Las generalidades están contenidas en las clases y las particularidades en los objetos.

• Un objeto se construye a partir de una clase instanciándolo.

– **Representaciones gráficas de clases**

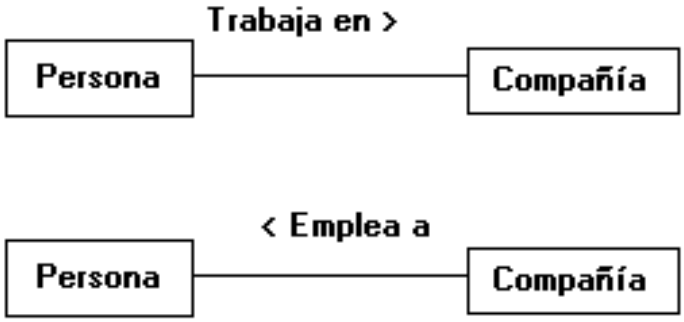


– **Relaciones entre clases**

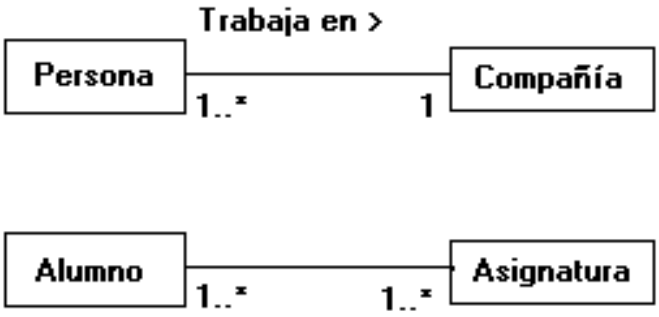
• Existen dos relaciones entre clases:
- La asociación
- La agregación

– **Asociación**

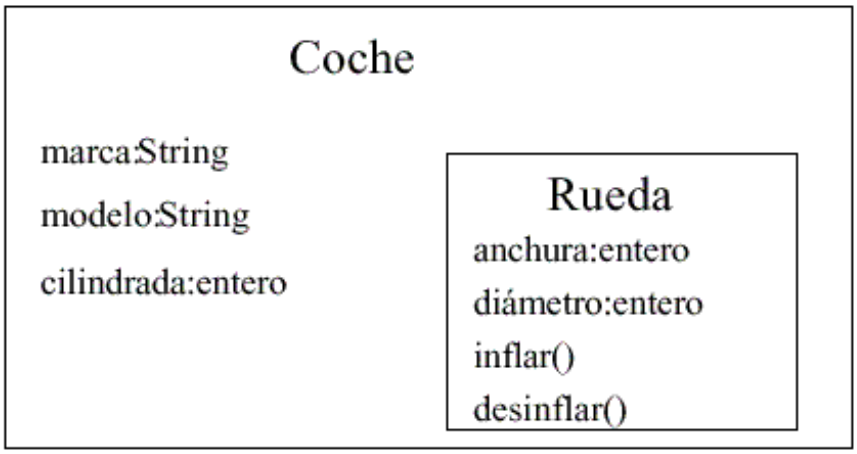
• Expresa una conexión semántica bidireccional entre clases.
• Se representan como los enlaces y se diferencian por el contexto del diagrama.

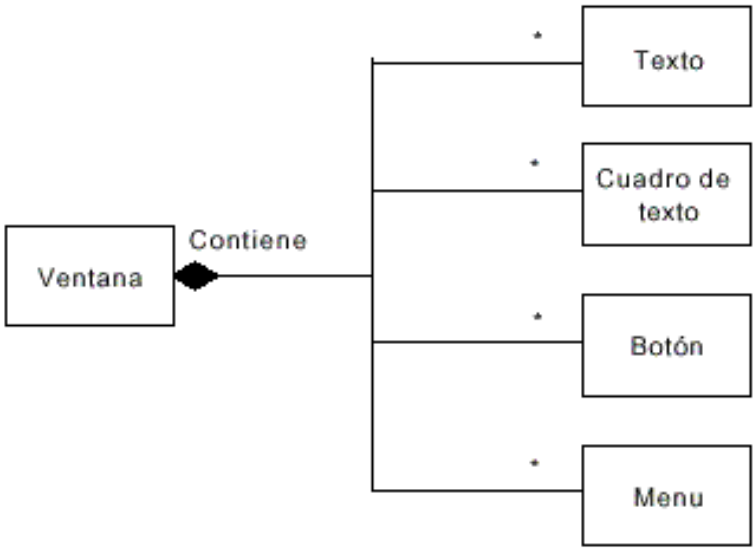


• Cardinalidad de las asociaciones:

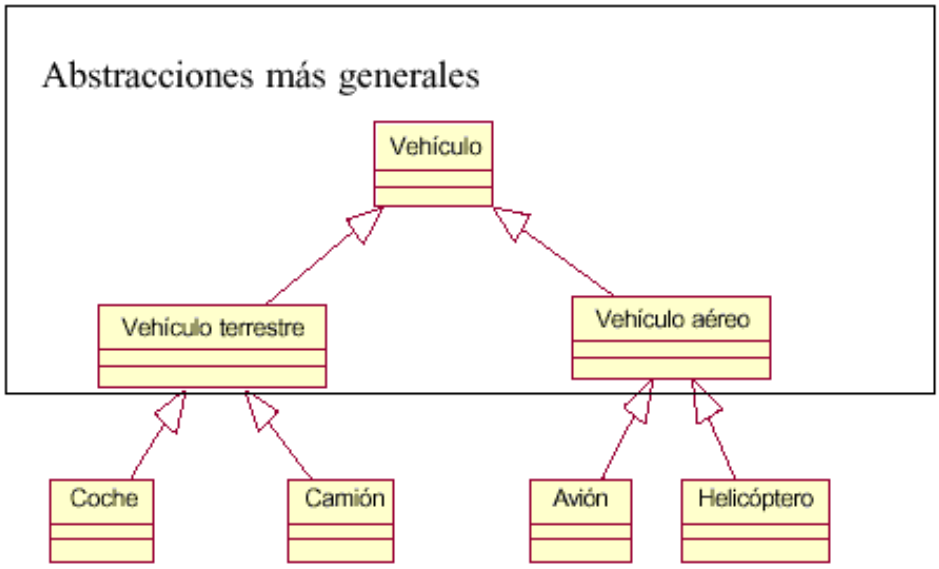


– **Agregación**
• Es una forma particular de asociación pero con acoplamiento ***fuerte y asimétrico***
• Una de las clases cumple una función más importante que la otra.
• Permite representar asociaciones amo/esclavo, todo/partes, compuestos / componentes.

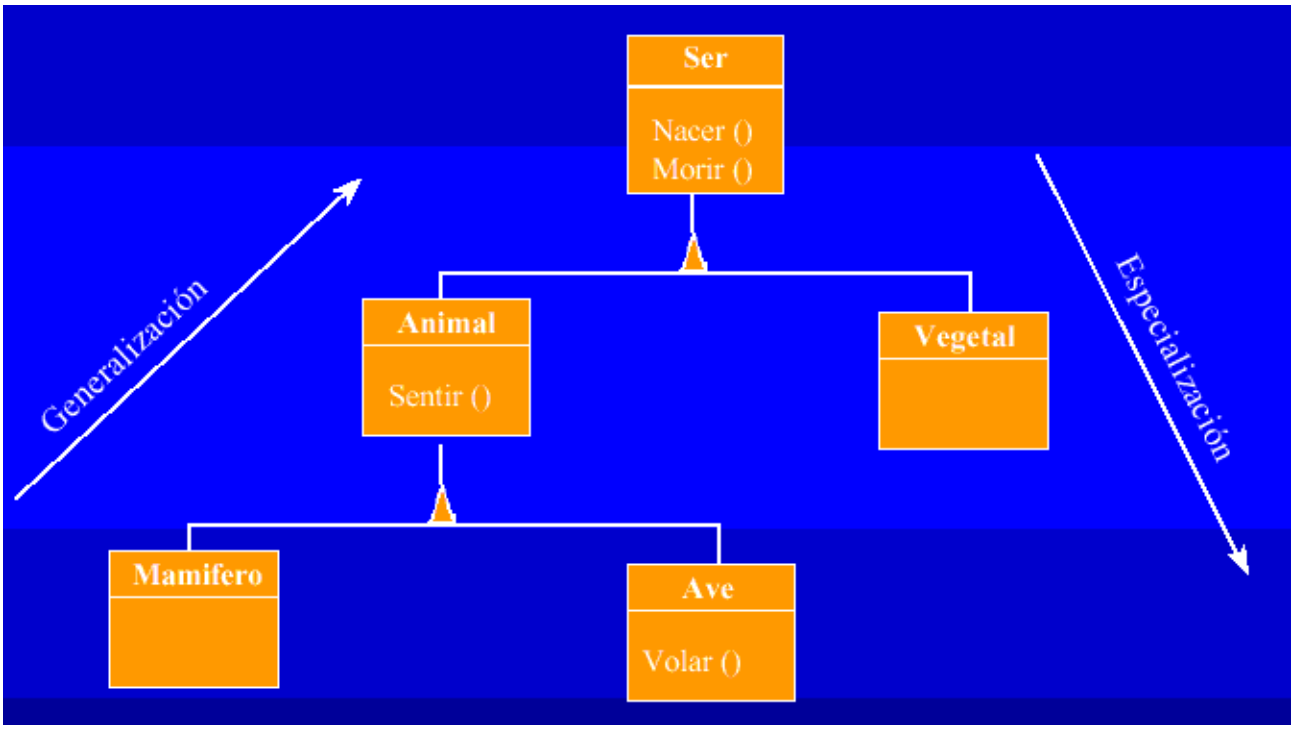




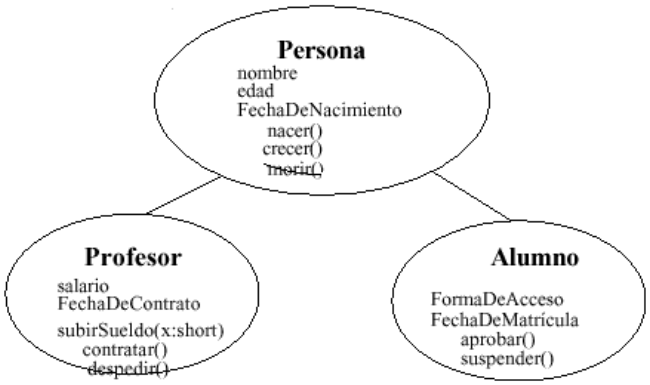
– **Las jerarquías de clases**
• La generalización: consiste en factorizar los elementos comunes (atributos, operaciones y restricciones) de un conjunto de clases en una clase más general llamada superclase.
• La especialización: permite capturar particularidades de un conjunto de objetos no discriminados por las clases ya identificadas. Las nuevas características se representan por una nueva clase, subclase de una de las clases existentes.



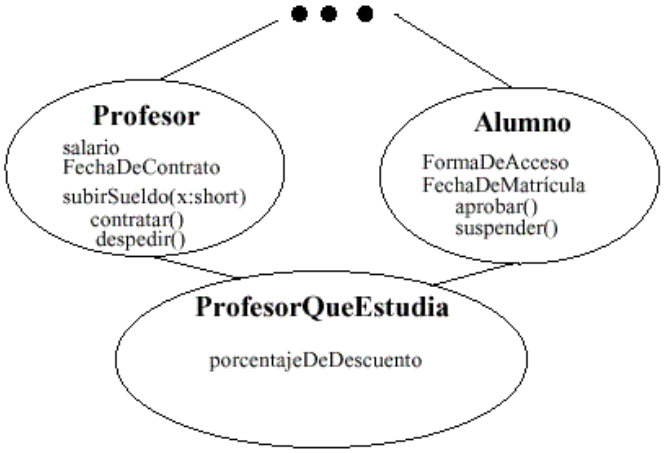
– **Herencia/Generalización/Especialización**
• Las subclases heredan las propiedades y métodos de las superclases
• Las subclases especializan a la superclase
• La superclase es una definición generalista de las subclases



– **Herencia Simple**



– **Herencia múltiple**



– **Herencia**

• Las subclases pueden...

– Añadir

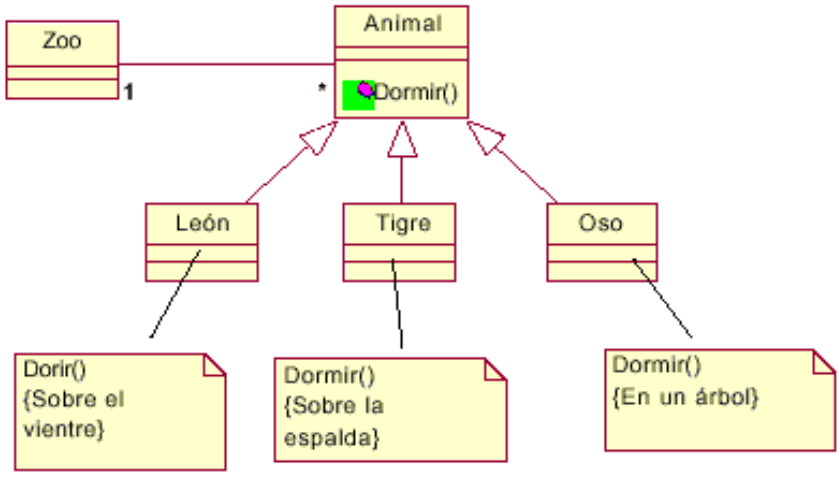
– Inhibir

– Redefinir

– **Polimorfismo**

• Capacidad de una operación de tener más de una interpretación en más de una clase.

• Facultad de un método de poder aplicarse a objetos de clases diferentes



– **Conclusiones: VENTAJAS**
• El análisis captura mayor semántica
• La herencia permite mayor extensibilidad
• El encapsulamiento y las interfaces reducen problemas de mantenimiento
• La OO es una herramienta para gestionar la complejidad
• Objetos bien diseñados, constituyen la base para módulos reutilizables, aumentando la productividad

• El paso de mensajes facilita la descripción de interfaces
• Particionamiento natural de los sistemas
• El ocultamiento de la información ayuda a construir sistemas más seguros
– **Conclusiones DESVENTAJAS**
• Diseñar módulos reutilizables añade un coste
• Beneficios a medio/largo plazo

Panorámica de metodologías de análisis y diseño orientado a objetos

– **Metodologías de desarrollo de software**
• La utilización de metodologías para el desarrollo de software no es un hecho reciente en el campo de la informática.
• Una metodología define **quién** debe hacer **Qué**, **Cuándo** y **Cómo** debe hacerlo. Es decir, la metodología define todo el **proceso** de desarrollo del software.

• A lo largo de la historia del desarrollo de software, se ha comprobado que es imprescindible la aplicación de metodologías en el desarrollo para poder conseguir un producto fiable y de calidad.
• El campo de la ingeniería del software ha buscado la definición de metodologías que permitiesen guiar todo el proceso del desarrollo de un sistema informático, desde su concepción inicial hasta su implantación y explotación.
• Sin embargo, las metodologías de desarrollo basadas en programación estructurada son muy poco operativas, y su implantación práctica ha sido siempre pequeña.
• Las técnicas de orientación a objetos han permitido la aparición de nuevas metodologías que sí cubren, de forma eficiente, cómoda y natural.
• Algunas metodologías orientadas a objetos:
• OSA
• FUSION
• OOM
• OO-JSD
• OOSD
• OBA
• OORA
• OSDI
• OMT

– **El proceso Unificado de desarrollo**

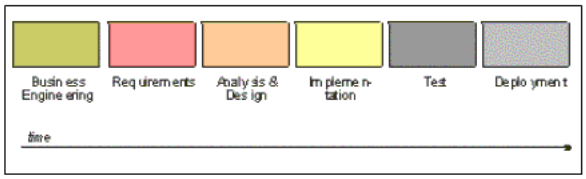
• El Proceso Unificado es la metodología de desarrollo orientado a objetos más extendida actualmente. Fue definido por Rational Rose.

• Define las fases a seguir en el desarrollo de un sistema software, los trabajadores que intervienen en cada una de las fases y los documentos y productos a generar.

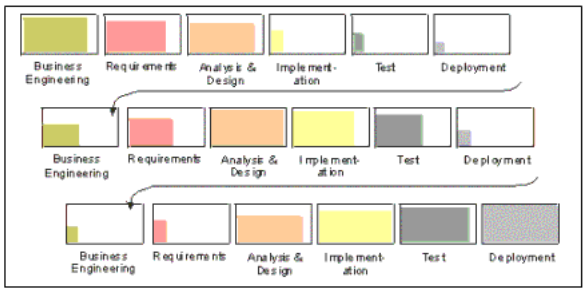
• Una de las características principales de este proceso es que se trata de un proceso iterativo e incremental, de tal forma que se realizan distintas iteraciones en las que se van abordando cada vez un mayor número de funciones del sistema.

• Este proceso se diferencia claramente del proceso clásico de desarrollo en cascada

Enfoque Cascada



Enfoque Iterativo e Incremental



Descripción del lenguaje de modelado UML

– **El lenguaje de notación UML**

• Para llevar a cabo todo el desarrollo de un sistema informático necesitaremos definir el **proceso** a seguir y utilizar una **notación**. El lenguaje **UML** (Unified Modeling Language) se ha convertido en el lenguaje estándar para el modelado orientado a objetos.

• El establecimiento de un lenguaje de notación estándar permite que el intercambio de información, la portabilidad entre herramientas y el aprendizaje sea más sencillo.

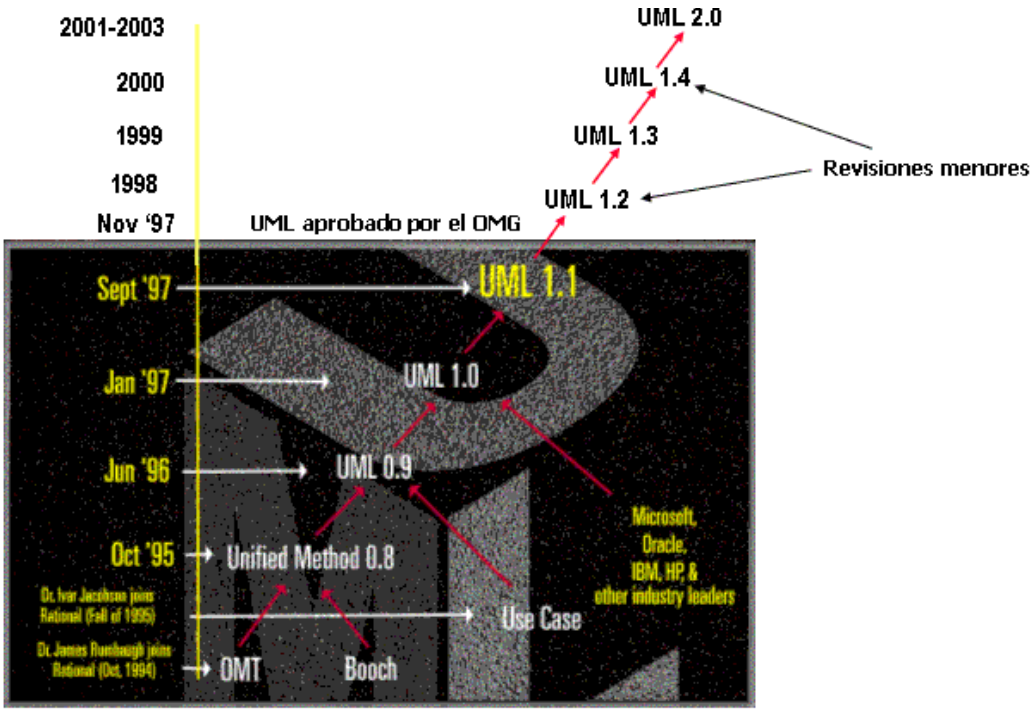
• UML es un lenguaje de notación, **no** es una metodología. Por lo tanto, UML no define cómo hay que realizar el proceso de desarrollo del sistema; simplemente nos proporciona la notación que nos permite llevar a cabo este proceso.

– **Historia de UML**

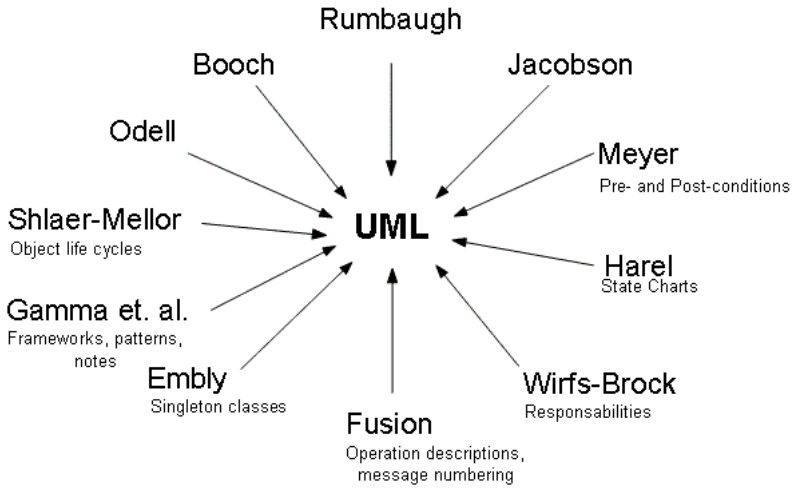
• El origen del UML se debe a los “tres amigos”; Grady Booch, Jim Rumbaugh e Ivar Jacobson. Los tres son socios de la compañía Rationa Software.

• Pronto se unieron al proyecto empresas de la talla de Hewlett-Packard, IBM y Oracle, lo cual favoreció enormemente la expansión del lenguaje.

• Posteriormente, el grupo OMG (*Object Management Group*) aprobó el estándar de UML en el documento “OMG Unified Modeling Language Specification”.



• UML aglutina distintos enfoques orientados a objetos existentes.



Componentes del lenguaje UML

• Un **modelo** captura una vista de un sistema del mundo real. Es una abstracción de dicho sistema, considerando un cierto propósito.

• El modelo describe completamente aquellos aspectos del sistema que son relevantes al propósito del modelo, y a un apropiado nivel de detalle. Los modelos más destacados en la metodología orientada a objetos son:

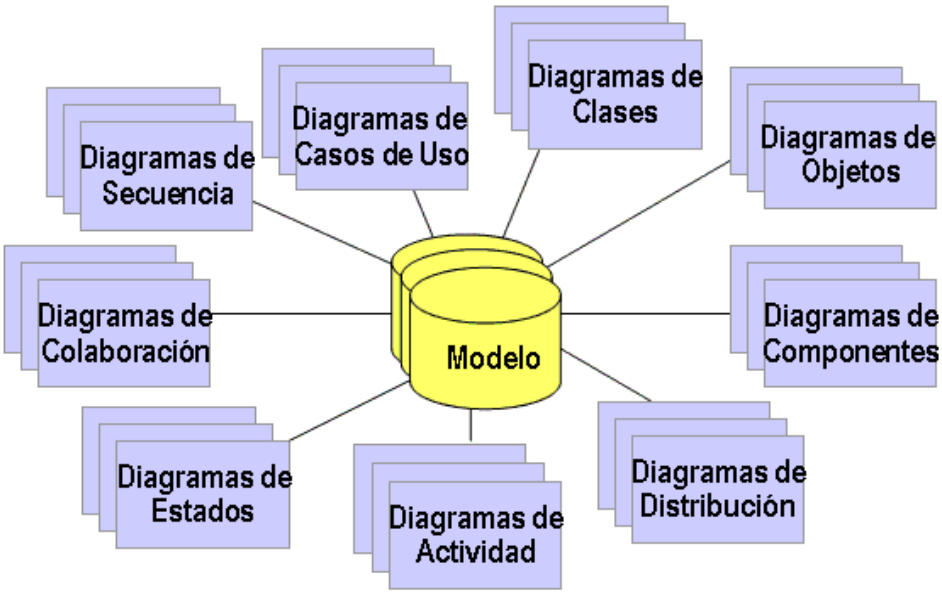
• Modelo de análisis

• Modelo de diseño

• Modelo de implementación

• Un **diagrama** es una representación gráfica de una colección de elementos de modelado, a menudo dibujada como un grafo con vértices conectados por arcos.

• Los diagramas expresan gráficamente partes de un modelo



• El modelo estructural de UML se irá viendo a lo largo del curso a la vez que se describe los procesos de análisis y diseño orientado a objetos.

IDENTIFICACIÓN DE OBJETOS Y CLASES

- **Objetivos del análisis**

• El desarrollo de un sistema informático es un proceso complejo.
• Se realiza por un equipo de personas que deben estar eficientemente organizadas.
• Debe seguir un proceso bien definido.
• Se debe apoyar en el uso de herramientas sofisticadas que nos permitan minimizar el coste del proceso.
• El modelo de análisis captura las partes esenciales del sistema, independientemente del lenguaje de implementación.

Visual Paradigm

Instalación Visual Paradigm Community Edition