Programación Orientada a Objetos

Introducción

Paradigmas de programación

Paradigma se refiere a una teoría o conjunto de teorías que sirve de modelo a seguir para resolver problemas o situaciones determinadas que se planteen. Es sinónimo de "patrón" o "ejemplo".

"La probabilidad de que se produzca un avance continuo en la programación requerirá de una invención continua además de la elaboración y colaboración de nuevos paradigmas" (Robert W. Floyd en The Paradigms of Programming)

Programación declarativa

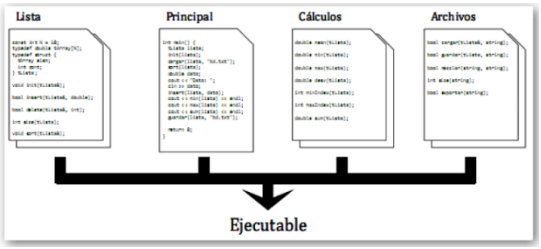
- Se describe la lógica de computación necesaria para resolver un problema sin describir un flujo de control de ningún tipo.
- La solución del problema se expresa especificando un conjunto de mecanismos internos de control.
- incluye la:
  - ✓ Programación funcional
  - ✓ Programación lógica
  - ✓ Programación restringida o con restricciones

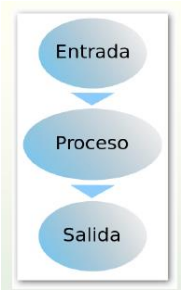
Programación imperativa

- Se describen sentencias que modifican el estado de un programa.
- La solución del problema se expresa especificando una secuencia de acciones a realizar a través de uno o más procedimientos denominados subrutinas o funciones.
- Incluye la:
  - ✓ Programación estructurada
  - ✓ Programación modular
  - ✓ **Programación orientada a objetos (POO)**

Programación Modular

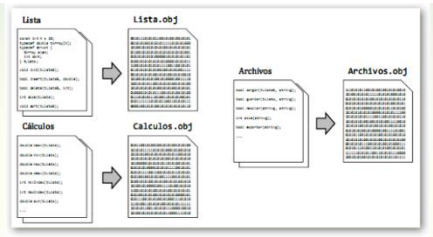
- Técnica de diseño Top-Down o de refinamiento sucesivo
- Utiliza el **Principio de Modularidad**
  - Consiste en dividir un programa en módulos o subprogramas, donde cada módulo resuelve una tarea específica
  - **!!! Módulo ≠ Función o Procedimiento**
  - Un módulo:
    - Es un archivo que contiene una parte del proyecto global
    - Es un archivo de código fuente de una unidad funcional



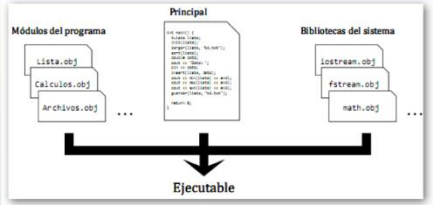


- Reciben datos de entrada, realizan un proceso y devuelven datos a la salida

- Se compila a código fuente de **forma independiente**



- Al compilar el programa principal, se adjuntan los módulos compilados



Desventajas

- Los programas de sistemas empresariales se van volviendo cada vez más grandes y complejos
- Aumenta la cantidad de reglas de negocios
- Muchas de esas reglas se presentan como volátiles.
- No existe uniformidad de representación entre las fases de análisis, diseño e implementación.
- Los mismos datos son gestionados desde diferentes porciones de código, lo cual vuelve poco flexibles las tareas de mantenimiento.
- Los datos estables y los datos que cambian con frecuencia se encuentran dispersos en el sistema.
- El Concepto de Módulo no resuelve naturalmente estos problemas

Programación Orientada a Objetos (POO)

Se basa en el concepto fundamental del “Objeto”

Propone una serie de técnicas y mecanismos:

- Herencia
- Abstracción
- Polimorfismo
- Encapsulamiento
- Paso de mensajes

Descomposición funcional (Procedimental)	Orientación a objetos
Enfoque principal: Módulos construidos alrededor de las operaciones.	Enfoque principal: Módulos construidos alrededor de las Clases

Presencia de Datos globales o distribuidos entre módulos.	Presencia de Clases débilmente acopladas, y **sin datos globales**
Perspectiva de Diseño: Entrada+Proceso+Salida.	Perspectiva de Diseño: Encapsulamiento+mensajes.
Subproducto básico: Organigramas de flujo de datos y/o control.	Subproducto básico: Diagramas jerárquicos de clases

Desarrollo OO

- Un buen desarrollo de software orientado a objetos requiere de:
  - Análisis OO
  - Diseño OO
  - Implementación en Lenguaje OO
- El modelamiento visual es clave para realizar AyDOO
- **UML se ha impuesto como un estándar para el modelamiento de sistemas OO**
- Cuando los modelos OO se construyen en forma correcta, son fáciles de comunicar, cambiar, expandir, validar y verificar

Ventajas de la POO

- Aumenta la Similitud del Programa al Mundo Real
- Maximiza la Modularidad y el Encapsulamiento
- Datos separados del Diseño
- Mejor entendimiento de la lógica del programa
- Fácil documentación y diseño del programa
- Dinamismo en el manejo de los datos
- Reutilización del código que facilita el
- Mantenimiento y la Extensibilidad del Software
- Facilita la creación de programas visuales
- Facilita la construcción de prototipos
- Facilita el trabajo en equipo

- Permite crear sistemas más complejos

Desventajas de la POO

- El aprendizaje inicial es más costoso
- Complejidad para adaptarse
- Se debe escribir mayor cantidad de código (al menos al principio)
- No es aplicable para resolver todos los tipos de problemas
- Requiere de mayor esfuerzo durante las fases de análisis y diseño
- La depuración en OO es más compleja que la depuración de programas estructurados
- Existe dependencia del lenguaje de programación
- Los estándares están en continua evolución
-

Fundamentos de la Orientación a Objetos

En el mundo real:

- Fenómeno: es un "objeto" percibido en el mundo real
- Concepto: es una abstracción que describe las propiedades comunes a un conjunto de fenómenos.
  - ✓ **Nombre:** Lo distingue de otros conceptos
  - ✓ **Propósito:** Propiedades que determinan la pertenencia al concepto
  - ✓ **Miembros:** Conjunto de fenómenos que forman parte del concepto

En la informática o mundo virtual:

- Objeto (Fenómeno):
  - ✓ Es una abstracción de "algo" en un dominio de un problema
  - ✓ Refleja las capacidades de un sistema para llevar información acerca de ese "algo", interactuar con ese "algo", o ambas cosas.
  - ✓ Es una representación en computadora de alguna cosa o evento del mundo real.
- Clase (Concepto):
  - ✓ Es una categoría y un 'molde' de objetos similares.
  - ✓ Los objetos se agrupan en clases.
  - ✓ Una clase describe y define el conjunto de atributos y comportamientos compartidos que se encuentran en cada objeto de la clase.

Modelo de objetos

Es un Modelo de la Realidad

- ✓ Según directrices del Paradigma de Orientado a Objetos
- ✓ Las entidades se describen en términos del Constructor Objeto

Es un marco de Referencia Conceptual

- ✓ Establece el conjunto básico de los conceptos, la terminología asociada y el modelo de computación de los Sistemas Software soportados por la tecnología orientada a los objetos

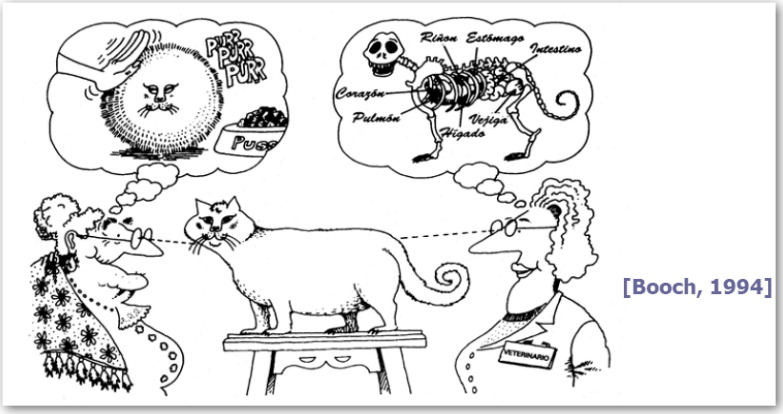
Es un Conjunto de Conceptos y Herramientas

- ✓ Con interacción dentro del sistema considerado
- ✓ Incluye mecanismos para manejar, como mínimo abstracción, encapsulamiento, jerarquía, modularidad y comunicación

Abstracción:

Obtiene una generalización conceptual de uno o más objetos y de sus características (desde un enfoque en particular).

La abstracción es un concepto sumamente útil, pero demasiado extenso, excepto para los casos triviales



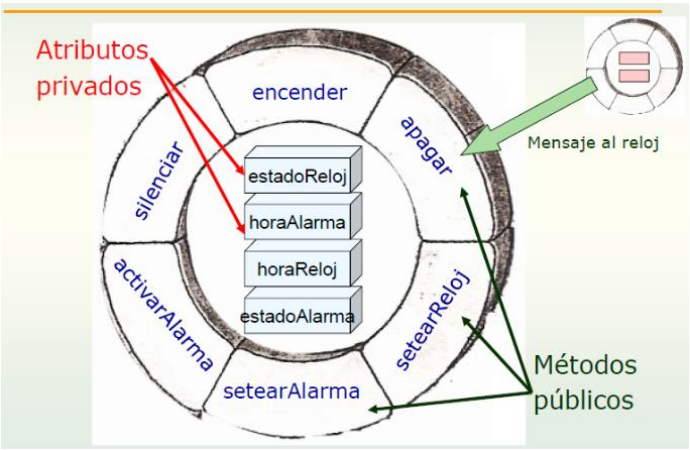
[Booch, 1994]

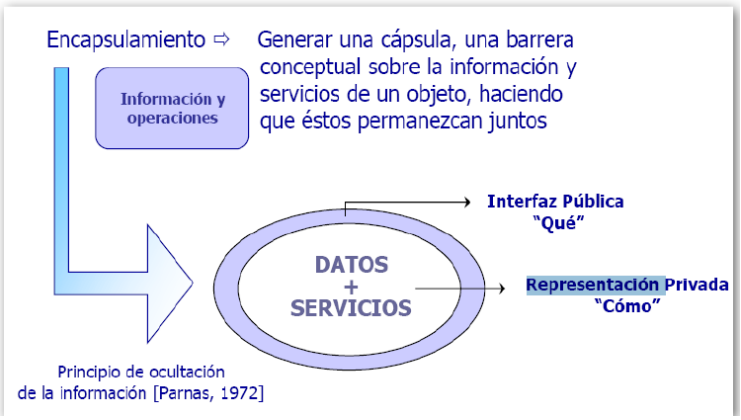
Encapsulamiento:

Oculta los detalles de la implementación de un objeto. La idea es ocultar el "cómo funciona" y solo permitir mostrar el "que hace" o "que se obtiene". Esto último se logra con una **interfaz**, la cual permite interactuar con el objeto, pero sin tocar los atributos internos.

Por ejemplo, en la imagen de la derecha del funcionamiento de un reloj. Por medio de la interfaz se puede interactuar con el reloj, pero los atributos internos, tales como "estadoReloj" "estadoAlarma" no se pueden tocar directamente. No modificamos directamente los estados de memoria interno, sino que eso lo realiza el objeto por sí solo según lo que pidamos por la interfaz

El encapsulamiento y la modularidad son medios para manejar las abstracciones





Jerarquía:

Con frecuencia, un conjunto de abstracciones forma una jerarquía. La identificación de estas jerarquías en el diseño simplifica la comprensión del problema

La jerarquía es una clasificación u ordenamiento de abstracciones

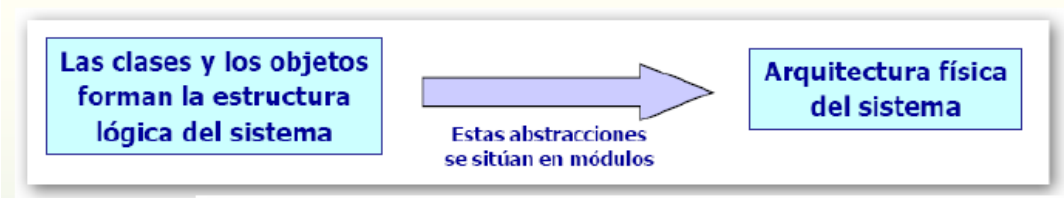
Modularidad:

La modularizarían consiste en dividir un programa en módulos que pueden compilarse de forma separada, pero que tienen conexiones con otros módulos.

"Las conexiones entre módulos son las suposiciones que cada módulo hace acerca de todos los demás"

Es una forma de lograr que los conceptos se encuentren **encapsulados** en archivos compilables de manera independiente

El uso de módulos es esencial para el manejo de la complejidad



Lenguaje de Modelado Unificado (UML)

Lenguaje basado en una notación gráfica (símbolos gráficos) que permite especificar, construir, visualizar y documentar los objetos de un sistema de software

Lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema

¿Qué NO es UML?

- Un Enfoque Teórico del Paradigma de Objetos
- Un Lenguaje de Programación, **Es un lenguaje de MODELADO**
- Una Metodología de Desarrollo
- Una Técnica para el Aprendizaje del Paradigma de Objetos

¿Qué es un modelo?

Modelo: Es una abstracción de un sistema del mundo real, considerando un cierto propósito.

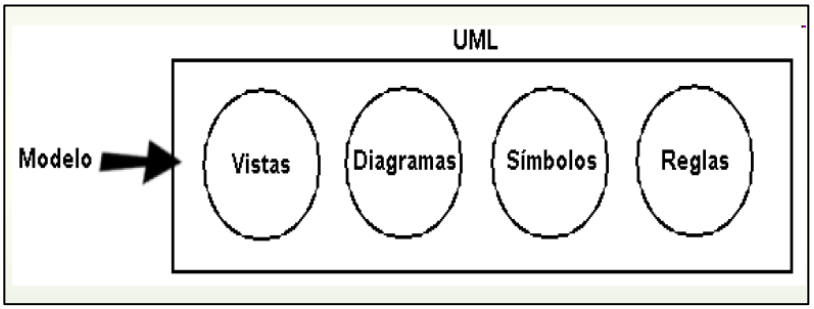
- Modelo estructural
- Modelo del comportamiento
- Modelo arquitectónico

- Modelo del negocio

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

Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés.

UML permite entonces un conjunto de *vistas*, *diagramas*, *símbolos* y *reglas* para generar un modelo.

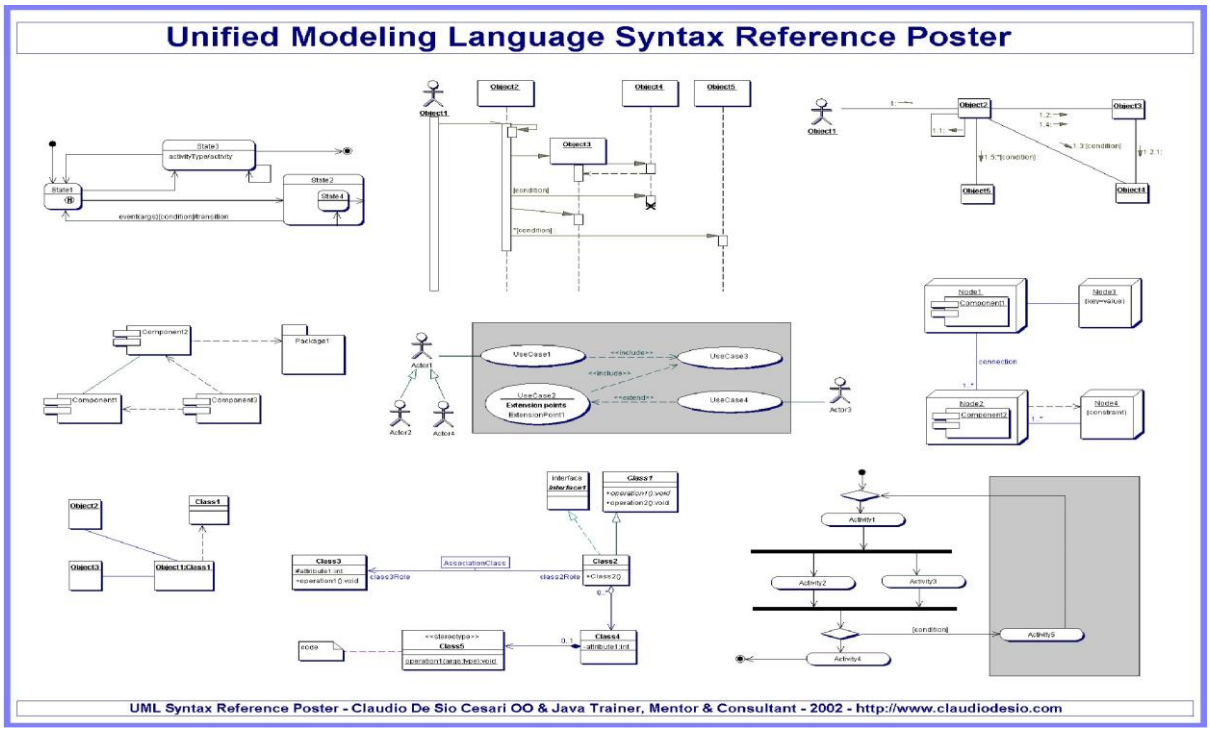


Vistas

- Las vistas muestran diferentes aspectos del sistema modelado.
- Cada vista es una abstracción que consiste en un número de diagramas.
- Modelo 4+1

Diagramas

Los diagramas son las gráficas que describen cada vista



Símbolos y Reglas de un Diagrama

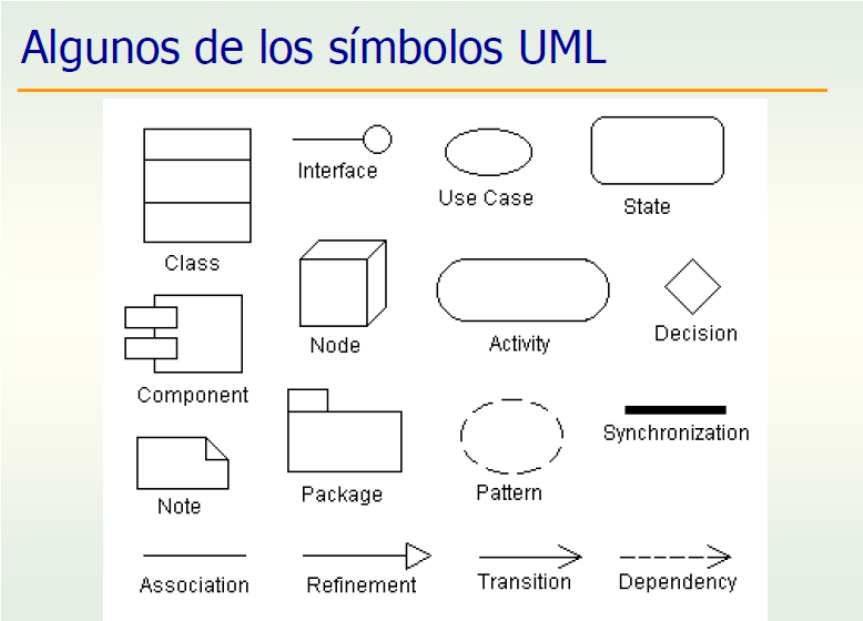
Elementos de modelo o Símbolos

- **Son los conceptos utilizados en los diagramas.**
- Representan conceptos comunes orientados a objetos, tales como clases, objetos y mensajes
- Representan también a las relaciones entre estos conceptos incluyendo la asociación, dependencia y generalización.
- Un elemento de modelo es utilizado en varios diagramas diferentes, pero siempre tiene el mismo significado y simbología.

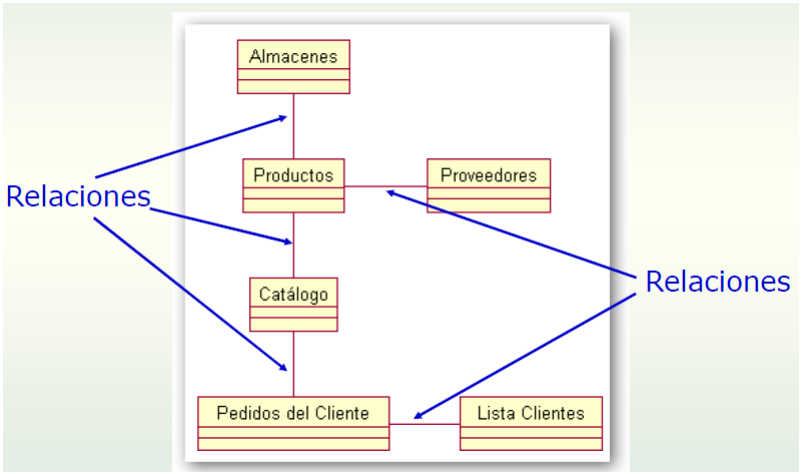
Reglas o Mecanismos generales:

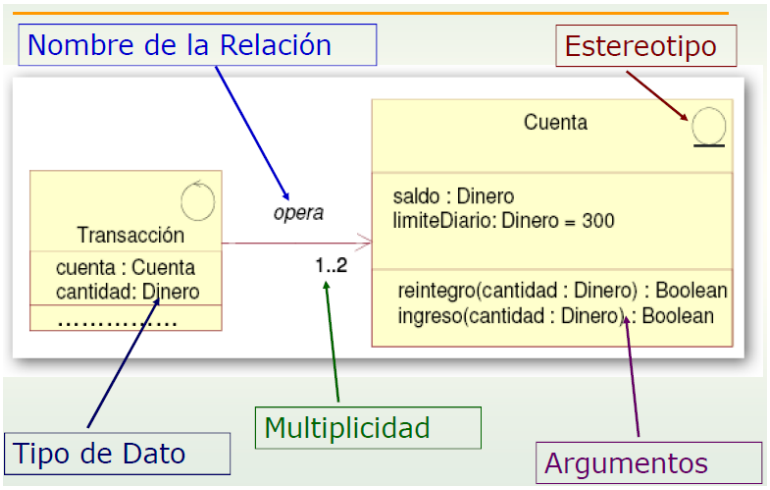
- Proveen comentarios extras, información o semántica acerca del elemento de modelo; además proveen mecanismos de extensión para adaptar o extender UML a un método o proceso específico, organización o usuario.

- Las reglas pueden ser: sintácticas, semánticas, pragmáticas



Diagramas de Clases (Modelo Conceptual, porque relaciona conceptos)





Relaciones entre Clases

Relaciones entre conceptos

- Relaciones Simples: de dependencia de uso o de interacción
- Relaciones de Asociación: más complejas porque requieren mayor información
- Relaciones de Jerarquías (Herencia y Agregación): tratan de modelar diferentes tipos jerarquías

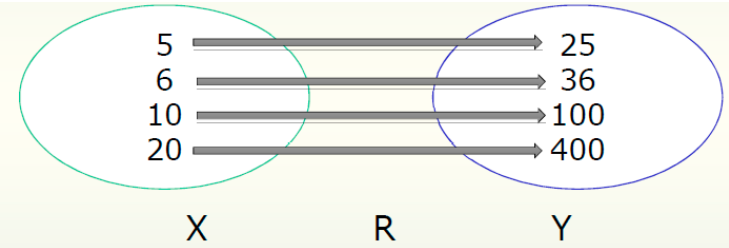
Vínculos

Es la relación que se establece entre 2 objetos al **comunicarse entre sí mediante uno o más mensajes**

5 al cuadrado es 25
6 al cuadrado es 36
10 al cuadrado es 100
20 al cuadrado es 400

Relación

Es la abstracción del conjunto de vínculos.

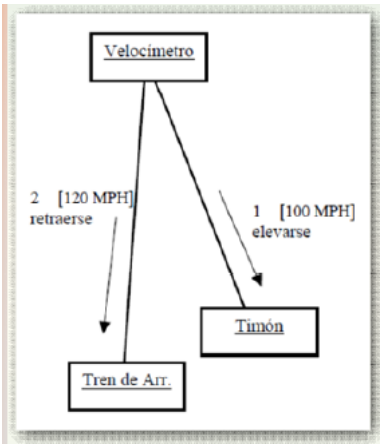


ó $Y = f(X) = X^2$

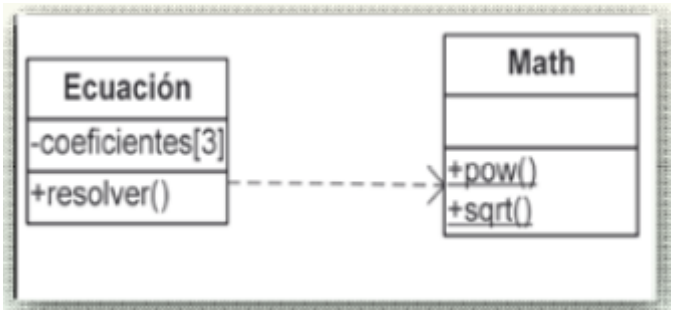
Asociación

Expresa una conexión bidireccional entre objetos. Es una abstracción de la relación que surge por los enlaces (vínculos) entre los objetos.

Simple o de uso/dependencia



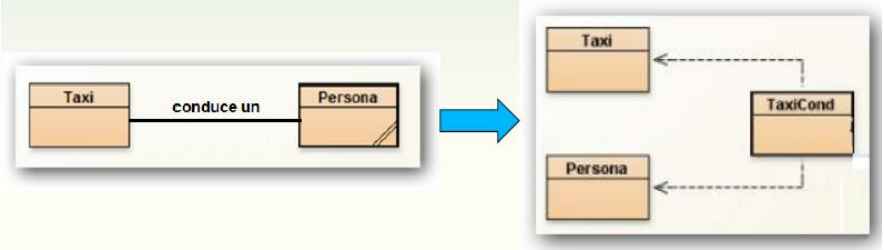
El velocímetro a los 120 MPH le dice al tren de aterrizaje que descienda



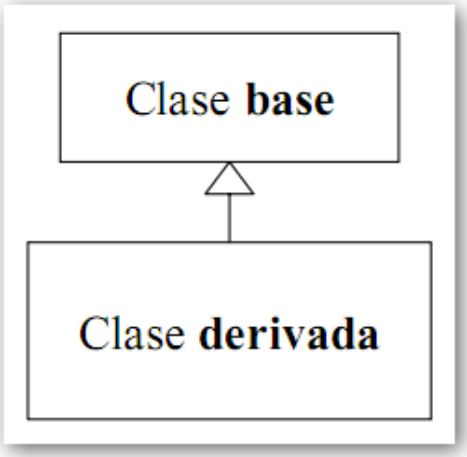
La clase ecuación llama o hace uso de operaciones de la clase match

Con clases

Hay casos donde para que un módulo haga uso de otro se necesitan rescatar datos de otro modulo. Por ejemplo, si la "persona conduce un taxi" el modelo no siempre se puede describir con solo esas dos clases: "persona", "taxi", ya que, se necesitan datos adicionales como: "consumo de combustible", "horas de manejo", etc. En resumen, en una estructura donde originalmente se definieron solo dos clases no siempre se puede resolver solo con esas dos clases

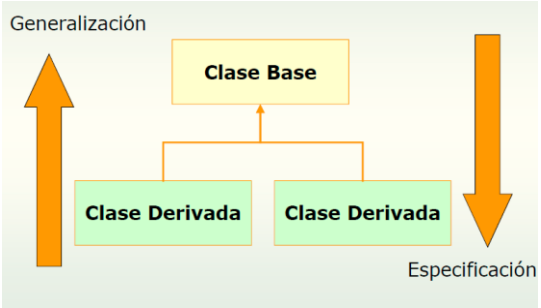


Herencia



- Es un mecanismo de la OO que permite clasificar los tipos de datos (abstracciones) por variedad
- Acerca la programación al modo de razonar humano
- Permite definir una clase modificando una o más clases ya existentes
- Reduce los tiempos y esfuerzos de programación y prueba
- La clase de la que se parte en este proceso recibe el nombre de **clase base** (madre o superclase)
- La nueva clase obtenida se llama **clase derivada** (hija o subclase)
    - ✓ La clase derivada hereda todas las características de la clase base.
    - ✓ La clase derivada puede definir características adicionales.
    - ✓ La clase derivada puede redefinir características heredadas de la clase base.
    - ✓ La clase derivada puede anular características heredadas de la clase base.
    - ✓ **El proceso de herencia no afecta de ningún modo a los objetos de la clase base. Solo sus características**

Esto permite generar **ontologías**, es decir, una jerarquía de frases donde los elementos nuevos van incorporando elementos nuevos respecto al anterior



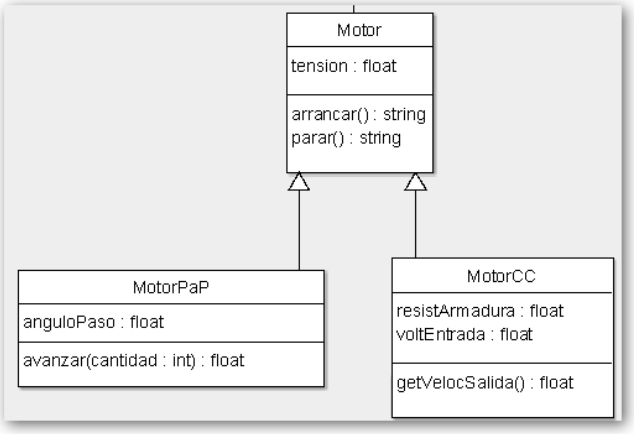
Esto se puede modelar de dos maneras:

1. Partiendo de las clases derivadas conocidas y de a poco ir "**generalizando**" hasta crear la clase base

2. Partir de la clase base conocida y **especificar**, creando así las clases derivadas

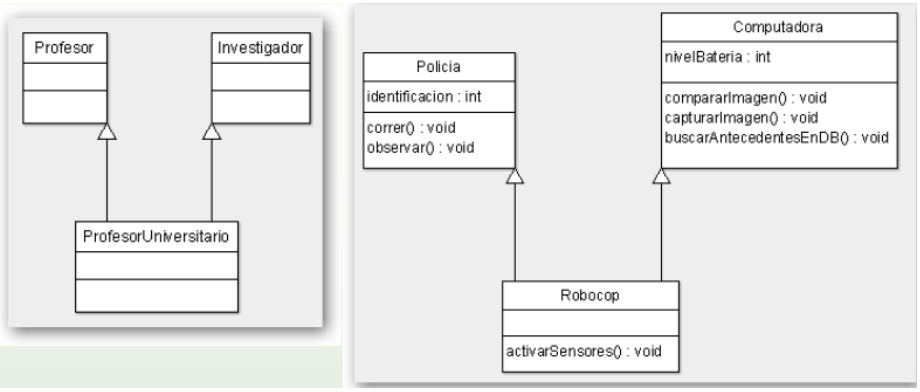
Herencias Simples

Cuando una o más Clase Derivada se crea a partir de una **Única Clase Base**



Herencias Múltiples

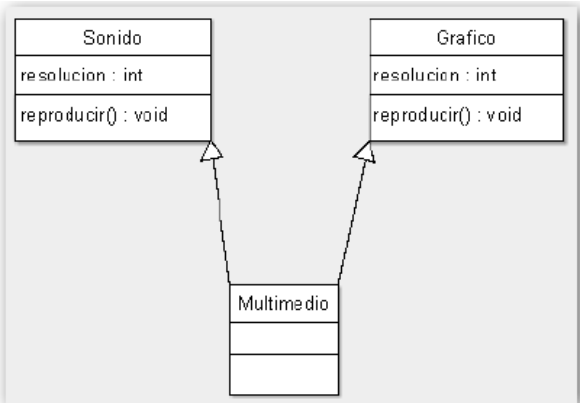
Cuando la Clase Derivada se obtiene de **Dos o Más Clases Base**



Dificultades de la Herencia Múltiple

Herencia repetida

- Por ej.: “Multimedio hereda 2 veces el atributo resolución"
- Produce ambigüedad respecto a los atributos o los métodos.
- Aumenta el tiempo de ejecución como consecuencia de tener que resolverse las colisiones.



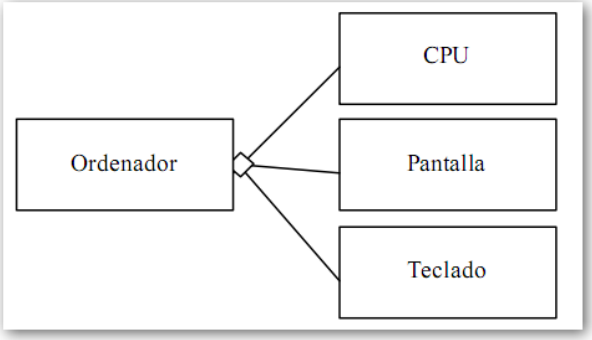
Agregación

Representa una relación de tipo “parte de” entre objetos.

En UML el símbolo de agregación es un rombo colocado en el extremo donde está la clase que representa el "todo".

Características:

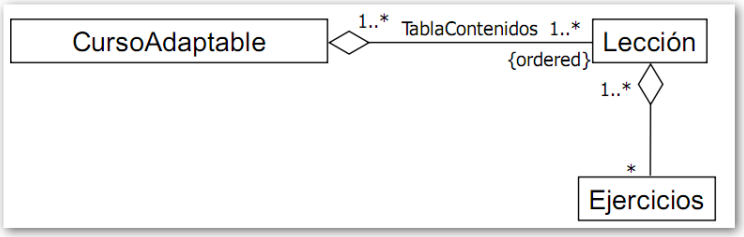
- ✓ Objeto agregado = Objeto construido a partir de otros.



- ✓ El agregado es mayor que la suma de sus partes
- ✓ Las interacciones del conjunto de objetos agregados (CPU, Pantalla, Teclado) se realizan a través de la interfaz del objeto agregado (Ordenador)
- ✓ Objetos componentes encapsulados dentro del agregado
- ✓ Multiplicidad en las dos partes de la relación
- ✓ Las partes pueden existir incluso después de que el agregado sea "desmontado" o destruido
- ✓ Las partes pueden cambiar de un agregado a otro

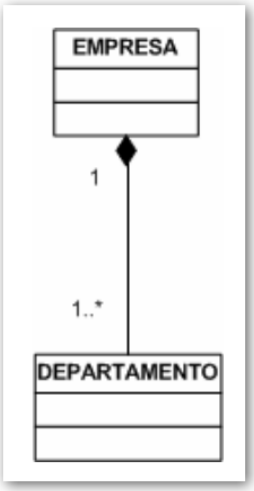
Ejemplo de Modelo con Agregación

Cursos adaptables:



- La lección aparece en la tabla de contenidos para cada curso que la usa
- Las lecciones se desarrollan para un curso, pero se pueden usar para construir otros cursos
- Los ejercicios se desarrollan inicialmente para un curso, pero pueden ser utilizados con otras lecciones para otros cursos.

Composición (o Agregación Inclusiva)



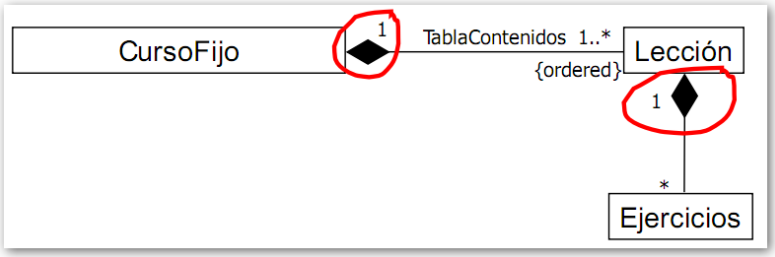
- Es una agregación con restricciones.
- **Si un objeto parte no puede comunicarse directamente con objetos externos distintos al objeto agregado se dice que la misma es una relación de agregación inclusiva.**
- En UML se simboliza este caso con un rombo negro.
- El ejemplo indica una agregación inclusiva debido a que no le está permitido en esta solución a un departamento tener relación por fuera del objeto agregado.

Características:

- ✓ La composición es una forma fuerte de agregación
- ✓ El ciclo de vida de las partes depende del ciclo de vida del agregado
- ✓ Las partes no existen fuera de su participación en el agregado
- ✓ La pertenencia fuerte implica objetos físicos que se unen para formar el compuesto
- ✓ La multiplicidad en el "extremo" del compuesto (Empresa) es 1 ó 0..1
- ✓ Multiplicidad en el "extremo" de las partes del compuesto (Departamento)
- ✓ Si el agregado se "desmonta" o se destruye las partes no tienen existencia propia
- ✓ Las partes no se pueden mover de una composición a otra

Ejemplo de Modelo con Composición

Cursos fijos:



- Se crean y se entregan. Para crear un nuevo curso todo el material se crea desde de cero
- La tabla de contenidos hace referencia a cada lección del curso. Cada lección solamente aparece en la tabla de contenidos del curso para el que fue desarrollada
- Las lecciones se utilizan únicamente en el curso para el que fueron desarrolladas
- Los ejercicios se utilizan únicamente en las lecciones para las que fueron desarrollados

¿Cómo obtener el Diagrama de Clases?

1. Obtener el Conjunto de Clases Conceptuales Candidatas
2. Depurar el conjunto y obtener Clases Finales
3. Representar las Clases Conceptuales identificadas
4. Representar las Relaciones más importantes
5. Escribir un Nombre para cada relación
6. Representar Multiplicidad
7. Incorporar Atributos
8. Incorporar Operaciones
9. Incorpore información sobre Tipos de Datos y Argumentos

1. Obtener las Clases Conceptuales

Identificar frases nominales en descripciones del dominio

Frase nominal: Un sustantivo o un conjunto de palabras que actúan como tal

Por ejemplo, en el texto: “Una cinta transportadora es un sistema de transporte continúo formado básicamente por una banda continua que se mueve entre dos cilindros.” Se observan los objetos: *Cinta, Sistema de Transporte, Banda, Cilindros.*

O se puede usar una lista de categorías:

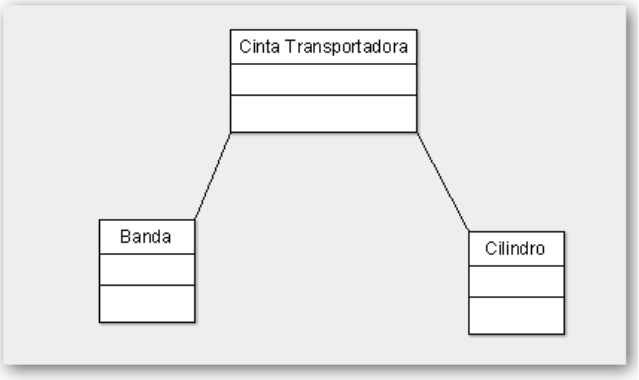
Categoría	*Ejemplo*
Objetos tangibles o físicos	Casa, Avión
Especificaciones, diseños o descripciones de las cosas	PlanoDeLaCasa, EspecificaciónDelProducto, DescripciónDelVuelo
Lugares	Tiendo, Aula
Transacciones	Venta, Pago, Reserva, Tranferencia
Líneas de la transacción	LíneaDeVenta
Roles de la gente	Cajero, Piloto, Jefe
Contenedores de otras cosas	Aula, Ciolectivo, Lata, Mochila
Contenidos	Pasajero, Artículo, ÚtilEscolar
Otros sistemas informáticos o electromecánicos externos al sistema	SistemaDeAutorización, ControlDeTra´ficoAéreo
Conceptos abstractos	Amor, Celos, Ansia, Acrofobia
Organizaciones	DepartamentoDeVentas, CompañiaAérea
Hechos	Reunión, Vuelo, Aterrizaje, Venta, Pago
Procesos (normalmente no se representan como conceptos, pero podría ocurrir)	VentaDeUnProducto, ReservaDeUnAsiento (no confundir con trasacciones)
Reglas y políticas	PolíticaDeReintegro, PolíticaDeCancelación
Catálogos	CatálogoDeProductos
Registros de finanzas, trabajo, contratos, cuestiones legales	Recibo, Remito, Factura, ContratoDeEmpleo, Expediente
Instrumentos y servicios financieros	LíneaDeCrédito, Stock
Manuales, documentos, artículos de referencia, libros	ManualDeReparaciones, ListaDeCambios
Relaciones	Amistad, Parentesco (podría estar en conceptos abstractos pero es bueno destacarlo ya que las relaciones no suelen ser consideradas al modelar)

2. Depurar el Conjunto de Clases

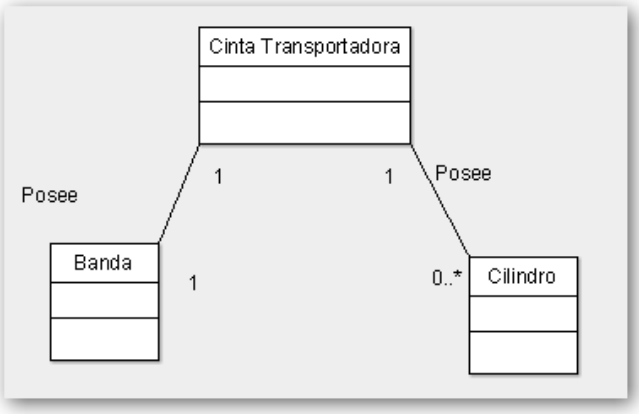
- Incluir en el modelo aquellas que cumplan una o más de las siguientes propiedades:
  - ✓ Guardar información
  - ✓ Necesidad de proveer un servicio
  - ✓ Atributos múltiples
  - ✓ Atributos comunes
  - ✓ Operaciones comunes
  - ✓ Requisitos esenciales
- Eliminar las clases sinónimas
- Eliminar las clases superfluas

En el ejemplo se suprime Sistema de Transporte

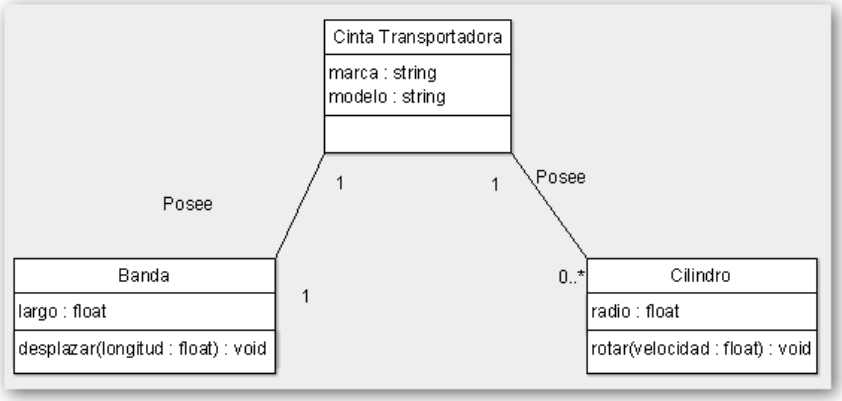
34. Representar Clases y Relaciones



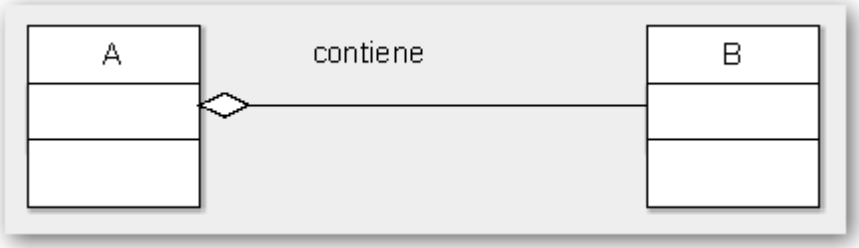
56. Indicar Relaciones y Multiplicidad



789. Añadir atributos, métodos y tipos

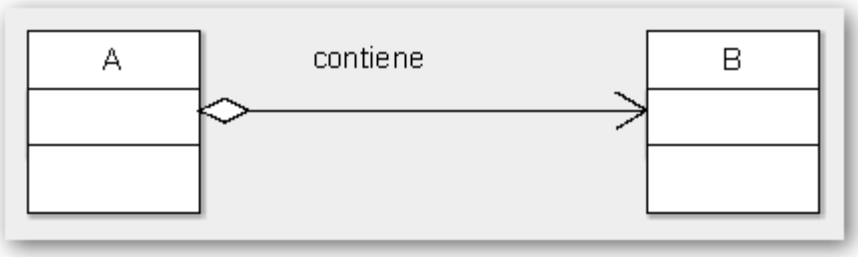


Agregación Bidireccional



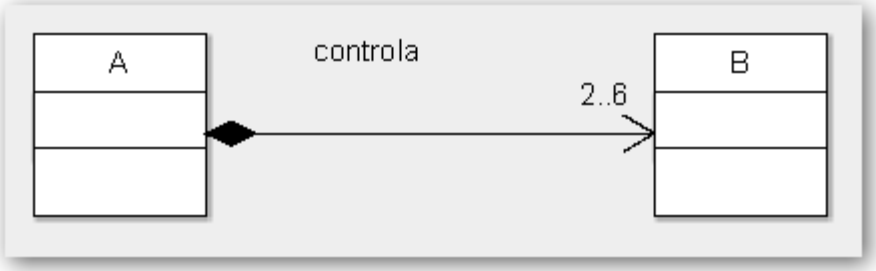
- A se puede comunicar con B y viceversa. Para esto A debe contener como un atributo a B, al igual que B debe tener como atributo a A (para poder comunicarse)
- El concepto componente se representa **siempre** como un atributo del concepto agregado.
- El acceso del componente hacia el agregado depende del diseño (con atributo o método).

Agregación de Navegación Restringida



- A se puede comunicar con B, pero B no puede mandarle mensaje a A. Solo A contiene como atributo a B
- El concepto componente se representa **siempre** como un atributo del concepto agregado.
- El concepto componente **no accede** al agregado.

Composición Unidireccional



- **Solo A puede controlar a B**, es decir que no hay ninguna otra clase que tenga la capacidad de hacerlo