El Modelo de Objetos

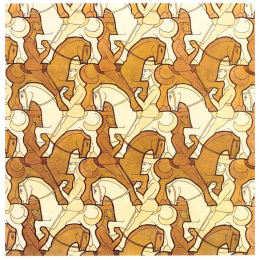
- Contenido
  - Introducción
  - Evolución del Modelo de Objetos
  - Fundamentos del Modelo de Objetos
  - Elementos del Modelo de Objetos
  - Aplicación del Modelo de Objetos

El Modelo de Objetos

- Contenido
  - **Introducción**
  - Evolución del Modelo de Objetos
  - Fundamentos del Modelo de Objetos
  - Elementos del Modelo de Objetos
  - Aplicación del Modelo de Objetos

Introducción

- La tecnología orientada a objetos se apoya en los sólidos fundamentos de la ingeniería, cuyos elementos reciben el nombre global de modelo de objetos.
- El modelo de objetos abarca los principios de abstracción, encapsulación, modularidad, jerarquía, tipos, concurrencia y persistencia.
- Ninguno de los principios anteriores es nuevo por sí mismo. Lo importante del modelo de objetos es el hecho de conjugar todos estos elementos en forma sinérgica.
- El diseño orientado a objetos es fundamentalmente diferente a los enfoques de diseño estructurado tradicionales ya que:
  - Requiere un modo distinto de pensar acerca de la descomposición.
  - Produce arquitecturas de software muy diferentes del diseño estructurado.



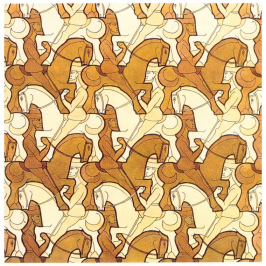
El Modelo de Objetos

- Contenido
  - Introducción
  - **Evolución del Modelo de Objetos**
  - Fundamentos del Modelo de Objetos
  - Elementos del Modelo de Objetos
  - Aplicación del Modelo de Objetos

Evolución del Modelo de Objetos (1)

Lenguajes de Primera Generación (1954-1958)	
FORTRAN I	Expresiones matemáticas.
ALGOL 58	Expresiones matemáticas.
Flowmatic	Expresiones matemáticas.
IPL V	Expresiones matemáticas.

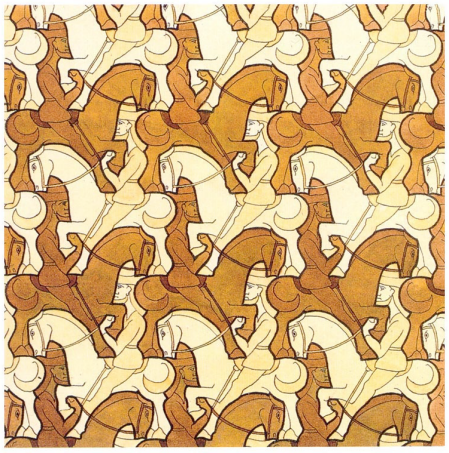
Lenguajes de Segunda Generación (1959-1961)	
FORTRAN II	Subrutinas, compilación separada.
ALGOL 60	Estructura en bloques, tipos de datos.
COBOL	Descripción de datos, manejo de archivos.
LISP	Manejo de listas, punteros y recolección de basura.



Evolución del Modelo de Objetos (2)

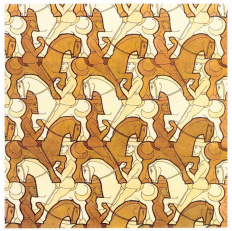
Lenguajes de Tercera Generación (1962-1970)	
PL/1	FORTRAN + ALGOL + COBOL.
ALGOL 68	Sucesor riguroso del ALGOL 60.
Pascal	Sucesor sencillo del ALGOL 60.
Simula	Clases, abstracciones de datos.

El Hueco Generacional (1970-1980)
Se inventaron muchos lenguajes diferentes, pero pocos perduraron.



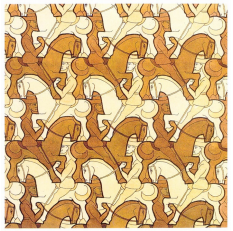
Evolución del Modelo de Objetos (3)

- La tendencia ha sido un desplazamiento desde los lenguajes que dicen a la computadora qué hacer (lenguajes imperativos), hacia lenguajes que describen las abstracciones clave en el dominio del problema (lenguajes declarativos).
- Los lenguajes de la primera generación se utilizaron principalmente para aplicaciones científicas y de ingeniería, y utilizaron un vocabulario y dominios de problema matemáticos casi por completo (FORTRAN I).
- Esta primera generación representó un paso de alejamiento de la máquina y un paso de acercamiento al dominio del problema.
- En los lenguajes de segunda generación, el énfasis se puso en las abstracciones algorítmicas (máquinas más potentes y baratas) y se le decía a la máquina qué hacer: leer fichas personales, ordenarlas y a continuación imprimir informe.
- Se acercaba a los desarrolladores un paso hacia el espacio del problema y los alejaba de la máquina subyacente.



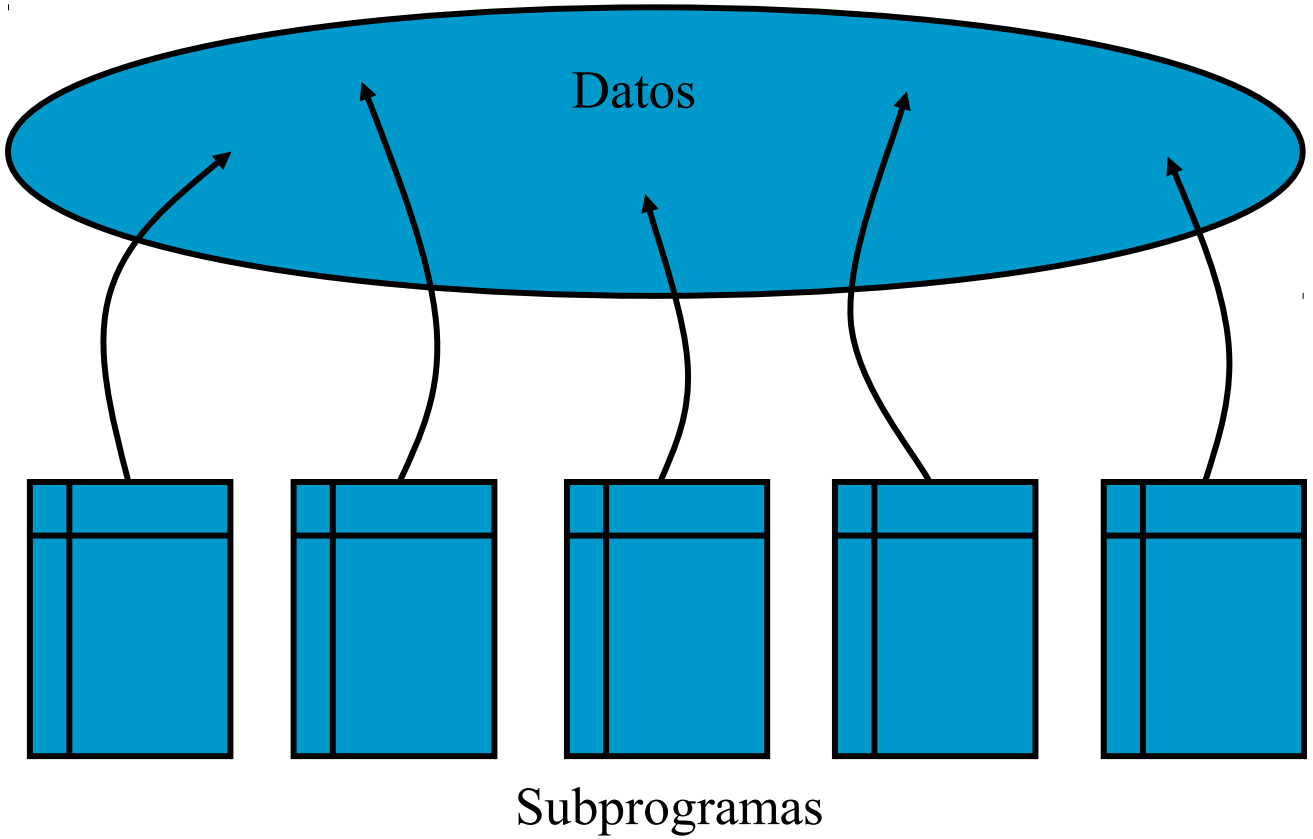
Evolución del Modelo de Objetos (4)

- En la tercera generación de lenguajes, se inventaron los transistores y la capacidad de procesamiento de las computadoras se incrementó exponencialmente, por lo que la evolución fue hacia las abstracciones de datos (más tipos de datos).
- Los lenguajes de esta época son el ALGOL 60 y el Pascal. Otro paso más de acercamiento hacia el dominio del problema.
- Los años setenta ofrecieron un frenesí de actividad investigadora que arrojó como resultado la creación de cerca de dos mil diferentes tipos de lenguajes con sus dialectos.
- En gran medida, la tendencia a escribir programas más y más grandes puso de manifiesto las deficiencias de los lenguajes más antiguos; así, sólo algunos de estos lenguajes han sobrevivido: Smalltalk, Ada, CLOS, C++ y Eiffel.
- Lo que resulta del mayor interés para nosotros es la clase de lenguajes que suelen llamarse basados en objetos y orientados a objetos. Estos son los que mejor soportan la descomposición orientada a objetos del software.



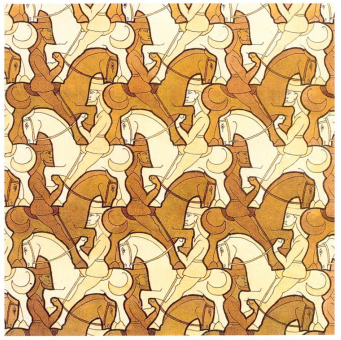
Evolución del Modelo de Objetos (5)

Lenguajes de la primera y principios de la segunda generación.



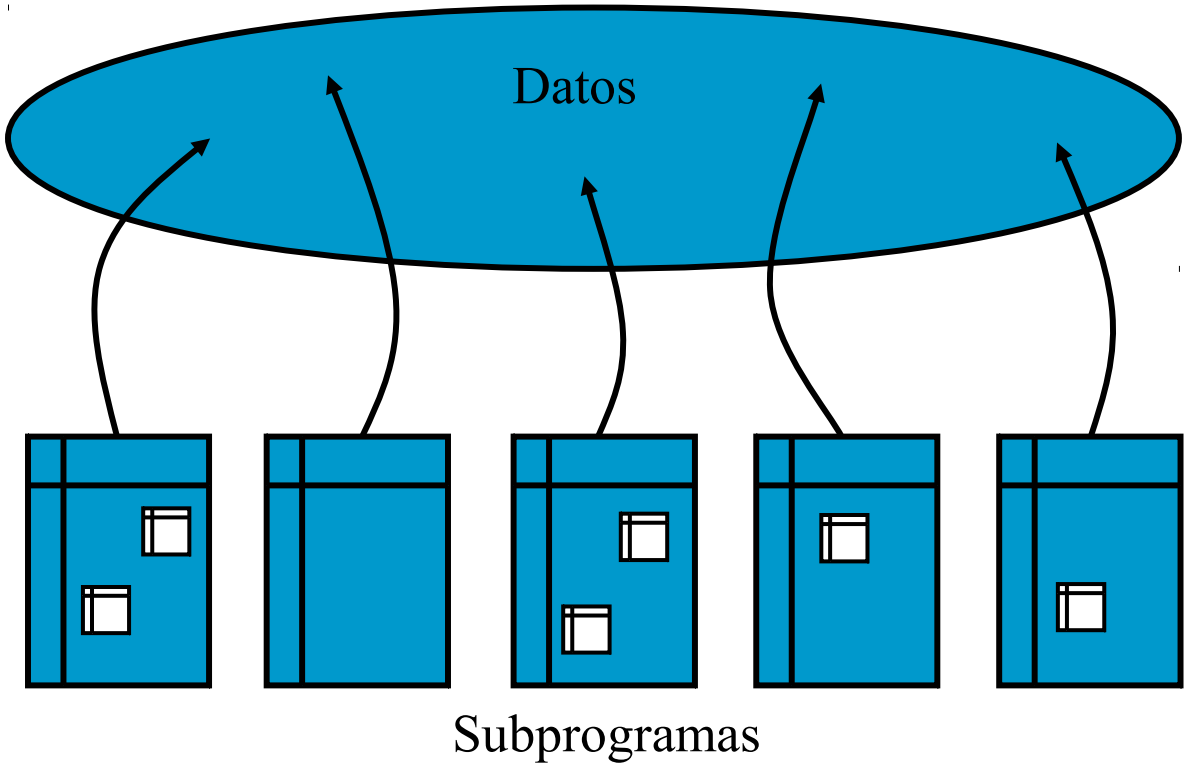
Evolución del Modelo de Objetos (6)

- Lenguajes de la primera y principios de la segunda generación
  - En estos lenguajes, el bloque básico de construcción de todas las aplicaciones es el subprograma.
  - Las aplicaciones escritas en estos lenguajes exhiben una estructura física relativamente plana, con varios subprogramas accediendo a datos globales.
  - Durante el diseño pueden separarse lógicamente diferentes clases de datos, pero existen pocos elementos en estos lenguajes para reforzar estas decisiones.
  - Un error en una parte de un subprograma se propaga con un efecto devastador por el resto de la aplicación pues comparten los datos globales.



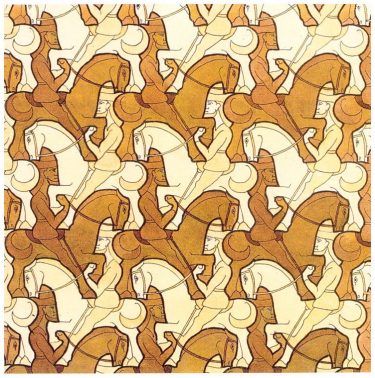
Evolución del Modelo de Objetos (7)

Lenguajes de fines de la segunda y principios de la tercera generación



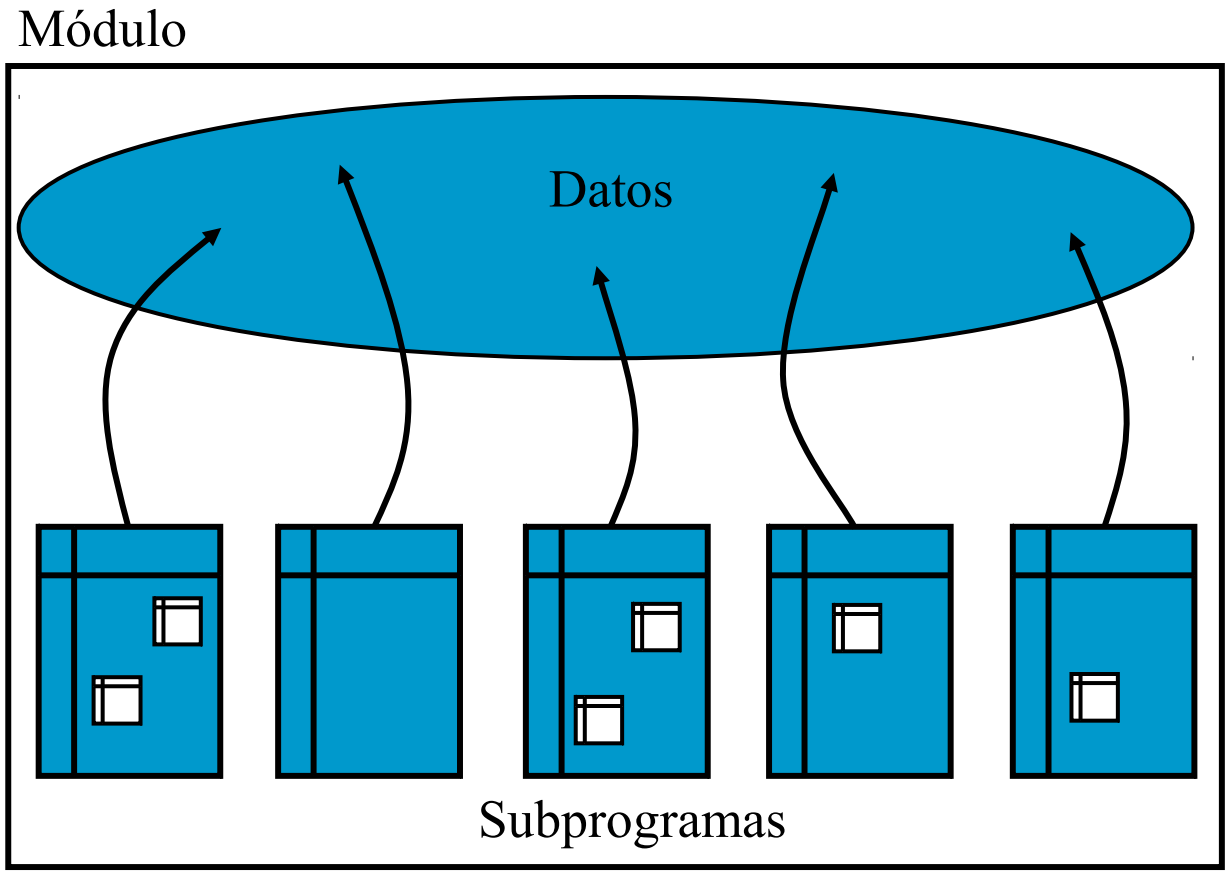
Evolución del Modelo de Objetos (8)

- Lenguajes de fines de la segunda y principios de la tercera generación
  - Aparece la abstracción procedimental, con subprogramas considerados como una forma de abstraer funciones del programa y no como un dispositivo para ahorrar trabajo.
  - Esto permite que se inventen lenguajes que soporten una gran variedad de mecanismos de pasos de parámetros.
  - Se asentaron los fundamentos de la programación estructurada y surgieron los métodos de diseño estructurado para construir grandes sistemas usando los subprogramas como bloques físicos básicos de construcción.



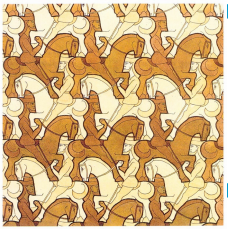
Evolución del Modelo de Objetos (9)

Lenguajes de finales de la tercera generación



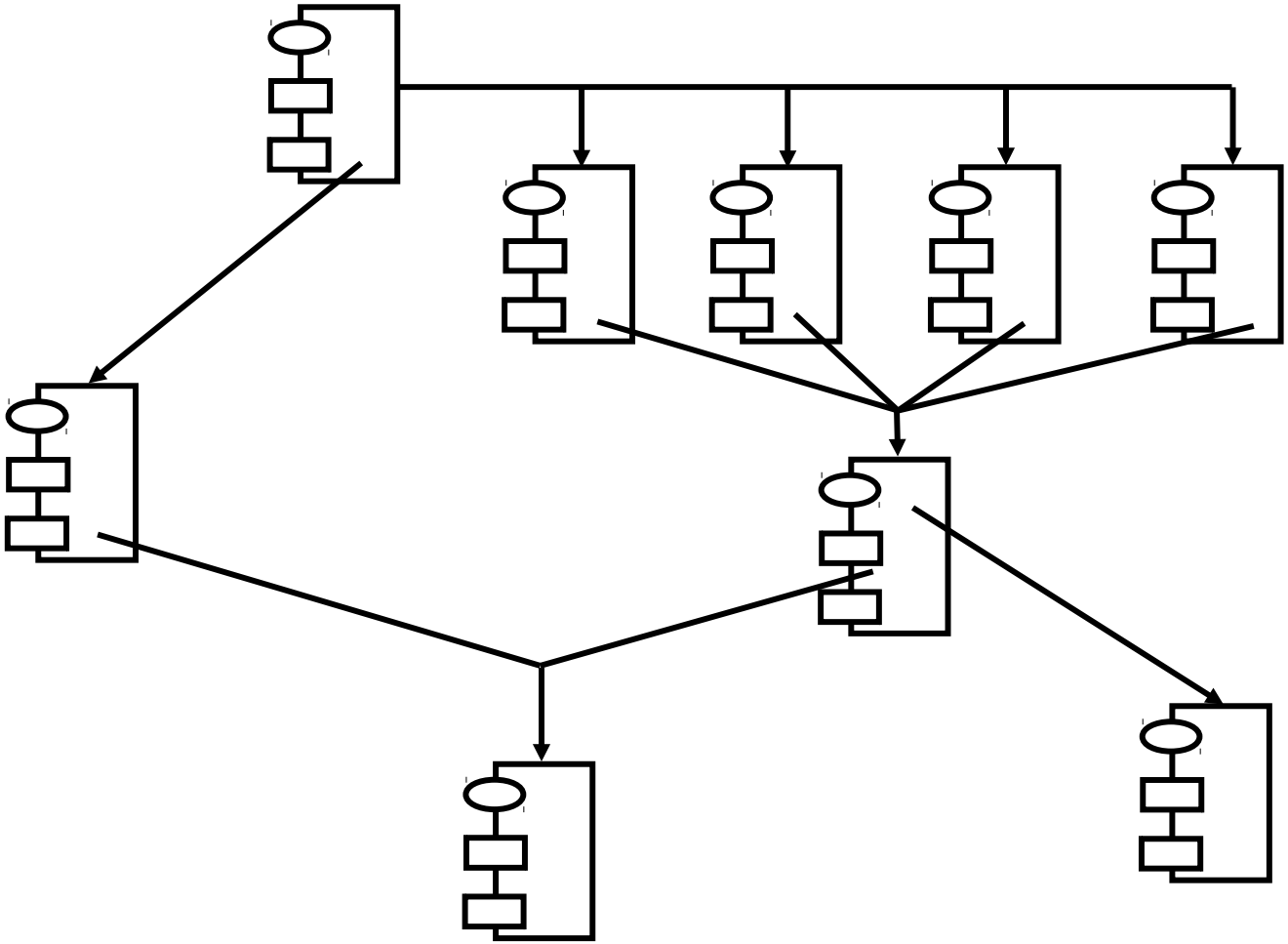
Evolución del Modelo de Objetos (10)

- Lenguajes de finales de la tercera generación
  - Surge otro importante mecanismo estructurador para resolver los proyectos de gran escala: el módulo compilado separadamente, que en su primera concepción no era mucho más que un contenedor arbitrario para datos y subprogramas.
  - Los módulos no solían conocerse como un mecanismo de abstracción importante.
  - En la práctica se utilizaban simplemente para agrupar subprogramas de los que cabría esperar que cambiasen juntos.
  - Tenían pocas reglas que exigiesen una consistencia semántica entre las interfaces de los módulos.
  - Permitía trabajar en equipos de trabajo independientes.



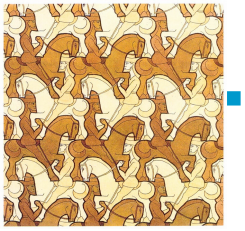
Evolución del Modelo de Objetos (11)

Lenguajes de programación basados en objetos y orientados a objetos



Evolución del Modelo de Objetos (12)

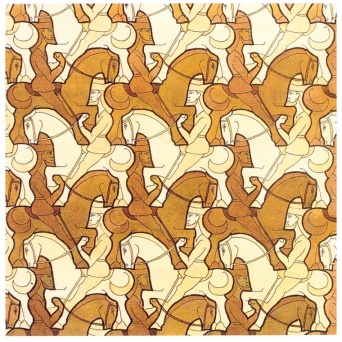
- Lenguajes de programación basados en objetos y orientados a objetos
  - Primero surgieron los métodos de diseño dirigido por los datos que proporcionaron una aproximación disciplinada a los problemas de realizar abstracciones de datos en lenguajes orientados algorítmicamente.
  - Luego aparecieron teorías acerca del tipo, que finalmente encontraron realización en lenguajes como Pascal.



  - La conclusión de estas ideas apareció en lenguajes como Simula, Smalltalk, Object Pascal, C++, CLOS, Ada y Eiffel, que recibieron l nombre de basados en objetos y orientados en objetos.
  - El bloque físico de construcción en estos lenguajes es el módulo, que representa una colección lógica de clases y objetos en lugar de subprogramas.
  - A diferencia de los lenguajes orientados a procedimientos, la estructura física de los lenguajes orientados a objetos tiene el aspecto de un grafo y no de un árbol y existen pocos o ningún dato a nivel global.

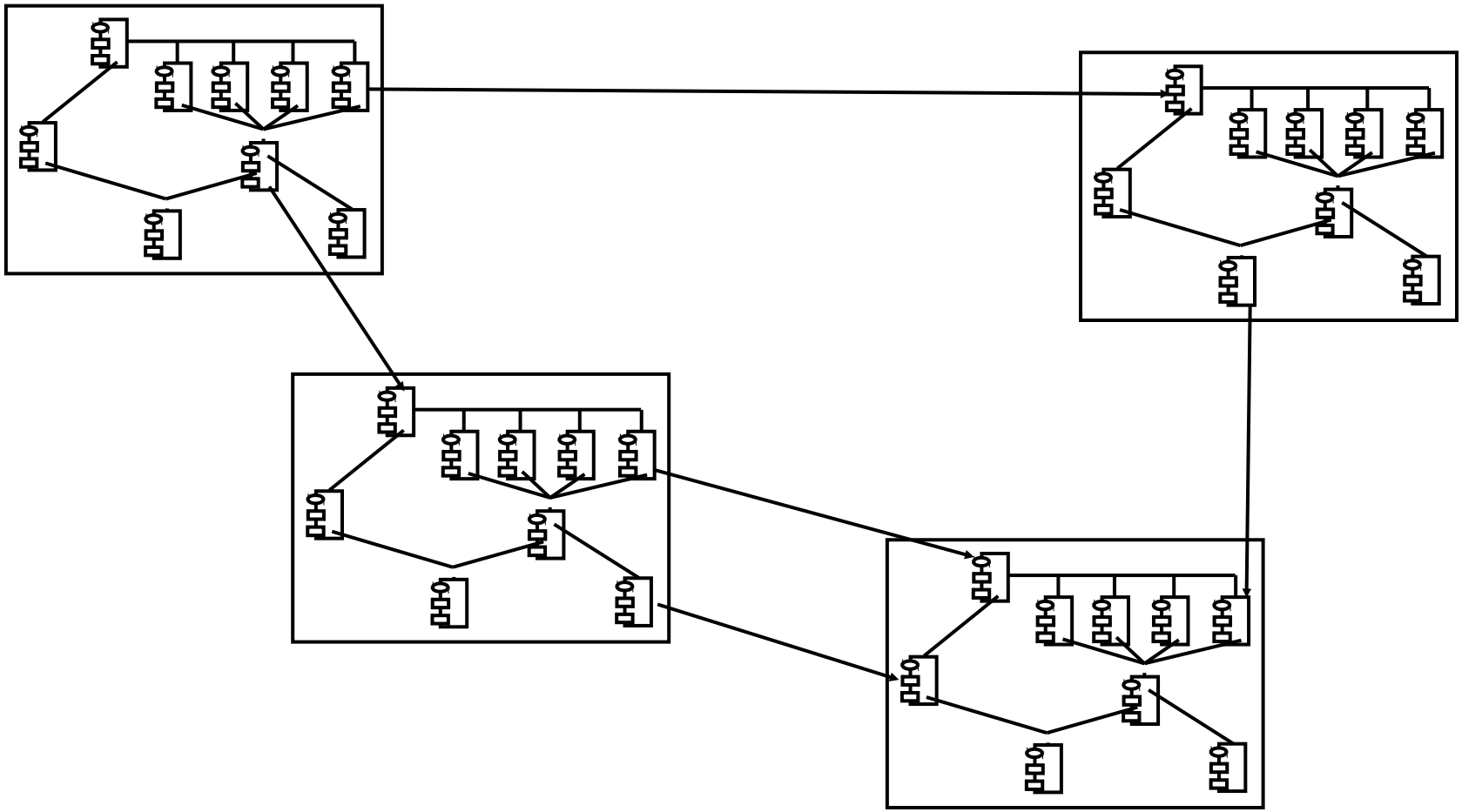
Evolución del Modelo de Objetos (13)

- Aplicaciones a gran escala
  - En sistemas muy complejos se encuentra que las clases, objetos y módulos proporcionan un medio esencial, pero aún así, insuficiente de abstracción.
  - Afortunadamente el modelo de objetos soporta el aumento de escala, con agrupaciones de abstracciones que se construyen en capas, una sobre otra.
  - A cualquier nivel de abstracción se encuentran colecciones significativas de objetos que colaboran para lograr un comportamiento de nivel superior.
  - Si se examina cualquier agrupación, se descubre que está formada por un nuevo conjunto de abstracciones cooperando. Ésta es exactamente la organización de la complejidad descripta anteriormente.



Evolución del Modelo de Objetos (14)

Aplicaciones a gran escala

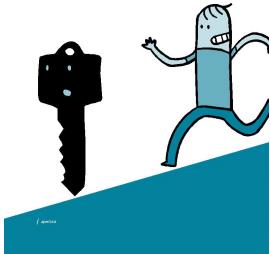


El Modelo de Objetos

- Contenido
  - Introducción
  - Evolución del Modelo de Objetos
  - **Fundamentos del Modelo de Objetos**
  - Elementos del Modelo de Objetos
  - Aplicación del Modelo de Objetos

Fundamentos del Modelo de Objetos (1)

- Los métodos de diseño orientados a objetos han surgido para ayudar a los desarrolladores a explotar la potencia expresiva de los lenguajes de programación basados y orientados a objetos.
- Se utilizan las clases y objetos como bloques básicos de construcción.
- El modelo de objetos ha demostrado ser un concepto unificador en la informática, aplicable no sólo a los lenguajes de programación, sino también al diseño de interfaces de usuario, bases de datos e incluso arquitecturas de computadoras.
- La orientación a objetos ayuda a combatir la complejidad inherente a muchos tipos de sistema diferentes. El diseño orientado a objetos representa así un desarrollo evolutivo, no revolucionario; no rompe con los avances del pasado sino que se basa en avances ya probados.



Fundamentos del Modelo de Objetos (2)

- Existen límites para la cantidad de complejidad que se puede manejar utilizando sólo descomposición algorítmica (en la que la mayoría de nosotros fue formado), por lo tanto, hay que adentrarse en la descomposición orientada a objetos.

- Existe un gran embrollo terminológico . Para minimizar la confusión, se definirá qué es orientado a objetos y qué no lo es.

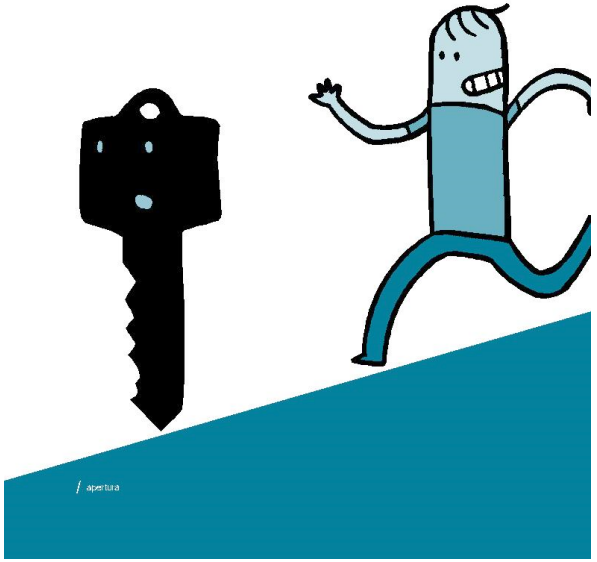
- **Objeto:** ***"Una entidad tangible que muestra algún comportamiento bien definido."***

- Sirven para unificar las ideas de las abstracciones algorítmica y de datos.

- Un objeto sólo puede cambiar de estado, actuar, ser manipulado o permanecer en relación con otros objetos de maneras apropiadas para ese objeto.

Fundamentos del Modelo de Objetos (3)

- Existen **propiedades invariantes** que caracterizan un objeto y su comportamiento.
  - Por ejemplo, un ascensor se caracteriza por propiedades invariantes que incluyen que sólo se desplaza arriba y abajo por su hueco.
  - Una silla se caracteriza porque permite sentarse a las personas.
- Cualquier simulación de estos objetos debe incorporar estos invariantes, porque son intrínsecos al concepto mismo del objeto.



Fundamentos del Modelo de Objetos (4)

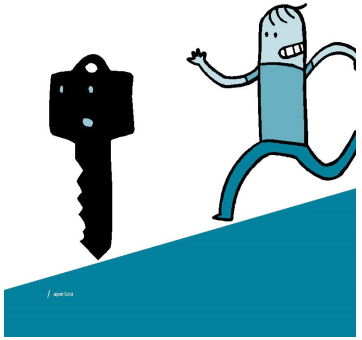
Programación Orientada a Objetos (POO)

"La POO es un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia."

- Hay tres partes importantes en la definición de la POO:
  - La programación orientada a objetos utiliza objetos, no algoritmos, como sus bloques lógicos de construcción fundamentales (jerarquía "parte-de").
  - Cada objeto es una instancia de alguna clase.
  - Las clases están relacionadas con otras clases por medio de relaciones de herencia (jerarquía de clases o "es-un").

Fundamentos del Modelo de Objetos (5)

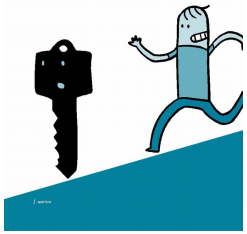
- Un programa puede parecer orientado a objetos, pero si falta cualquiera de los elementos anteriores, no es un programa orientado a objetos.
- La programación sin herencia, por ejemplo, NO es orientada a objetos, es programación con tipos abstractos de datos.
- Para que un lenguaje sea orientado a objetos, debe satisfacer los siguiente:
  - Soportar objetos, que son abstracciones de datos con un interfaz de operaciones con nombre y un estado local oculto.
  - Los objetos deben tener un tipo asociado (clase).
  - Los tipos (clases) pueden heredar atributos de los supertipos (superclases).



Fundamentos del Modelo de Objetos (6)

Diseño Orientado a Objetos (DOO)

"El DOO es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña."



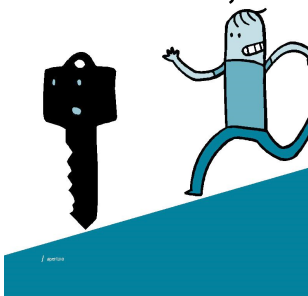
- Hay dos partes importantes en esta definición:
  - El diseño orientado a objetos da lugar a una descomposición orientada a objetos.
  - Se utiliza diversas notaciones para expresar diferentes modelos del diseño lógico y físico de un sistema, además de los aspectos estáticos y dinámicos del mismo.
- Resumiendo, se utiliza el término diseño orientado a objetos para referirse a cualquier método que encamine a una descomposición orientada a objetos.

Fundamentos del Modelo de Objetos (7)

Análisis Orientado a Objetos (AOO)

"El AOO es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema."

- El Análisis Orientado a Objetos (AOO), que enfatiza la construcción de modelos del mundo real, utiliza una visión del mundo orientada a objetos.
- Los productos del AAO sirven como modelos de los que se puede partir para un DOO.
- Los productos del DOO pueden utilizarse entonces como anteproyectos para la implementación completa de un sistema utilizando métodos de POO, siguiendo el camino del ciclo de vida de un desarrollo software.



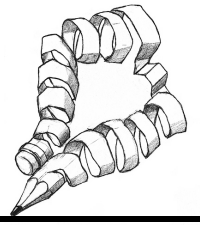
El Modelo de Objetos

- Contenido
  - Introducción
  - Evolución del Modelo de Objetos
  - Fundamentos del Modelo de Objetos
  - **Elementos del Modelo de Objetos**
  - Aplicación del Modelo de Objetos

Elementos del Modelo de Objetos (1)

- La mayoría de los programadores trabajan en un lenguaje y utilizan sólo un estilo de programación. Programan bajo un paradigma apoyado por el lenguaje que usan.

- No les han mostrado las alternativas para pensar sobre un problema, y por lo tanto, tienen dificultades para apreciar las ventajas de elegir un estilo más apropiado para el problema en cuestión.



- Hay cinco tipos principales de estilos de programación:

Estilo de Programación	Tipo de Abstracción
Orientados a procedimientos	Algoritmos
Orientados a objetos	Clases y Objetos
Orientados a lógica	Objetivos, a menudo expresados como cálculo de predicados
Orientados a reglas	Reglas si-entonces (if-then)
Orientados a restricciones	Relaciones invariantes

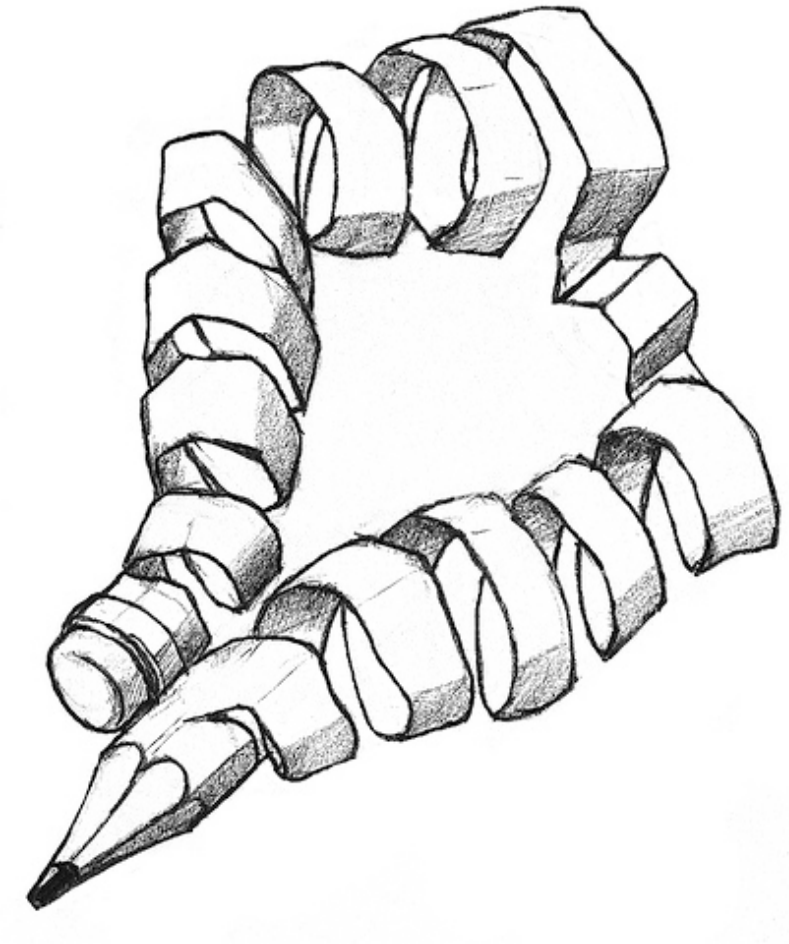
Elementos del Modelo de Objetos (2)

- No hay un estilo de programación que sea el mejor para todo tipo de aplicaciones.
- El ingeniero de software tiene su “caja de herramientas” de estilos y sólo emplea la herramienta adecuada de acuerdo al tipo de problema.
- Cada uno de los estilos de programación se basa en su propio marco de referencia conceptual.
- Cada uno requiere una actitud mental diferente, una forma distinta de pensar en el problema.
- Los elementos del modelo de objetos pueden dividirse en dos grupos:

  - **Elementos Fundamentales**: un modelo que carezca de cualquiera de estos elementos no es orientado a objetos.
  - **Elementos Secundarios**: cada uno de ellos es una parte útil del modelo de objetos, pero no es esencial

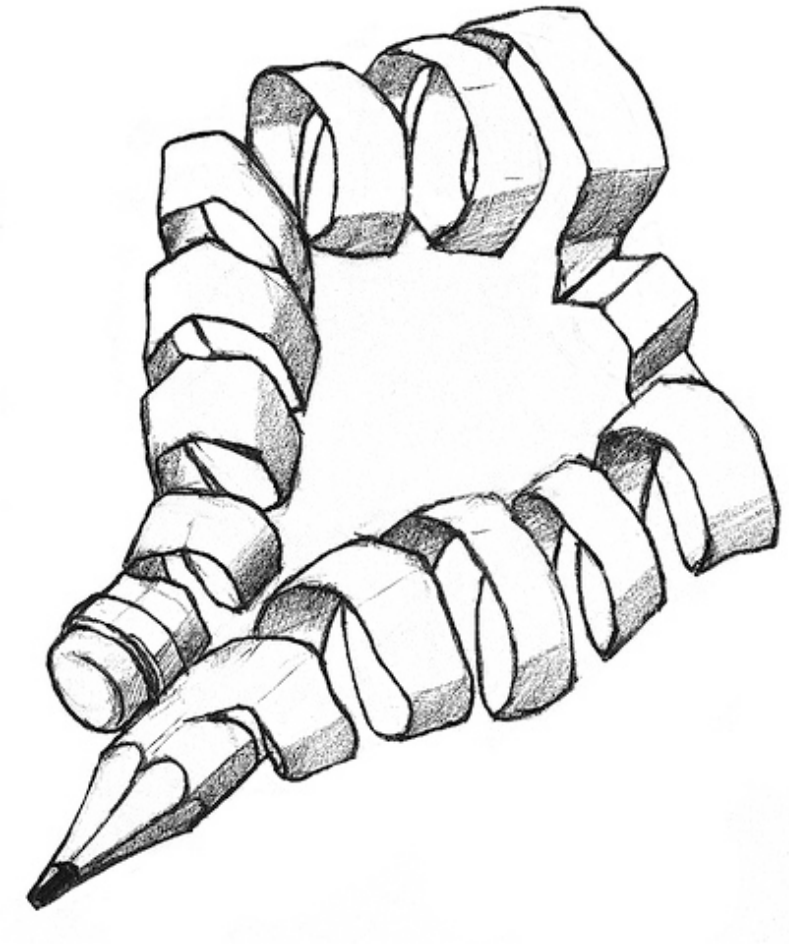
Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - Persistencia



Elementos del Modelo de Objetos (3)

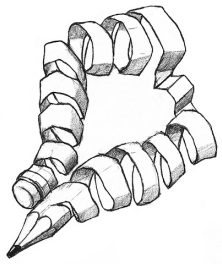
- Elementos Fundamentales:
  - **Abstracción**
  - Encapsulamiento
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - Persistencia



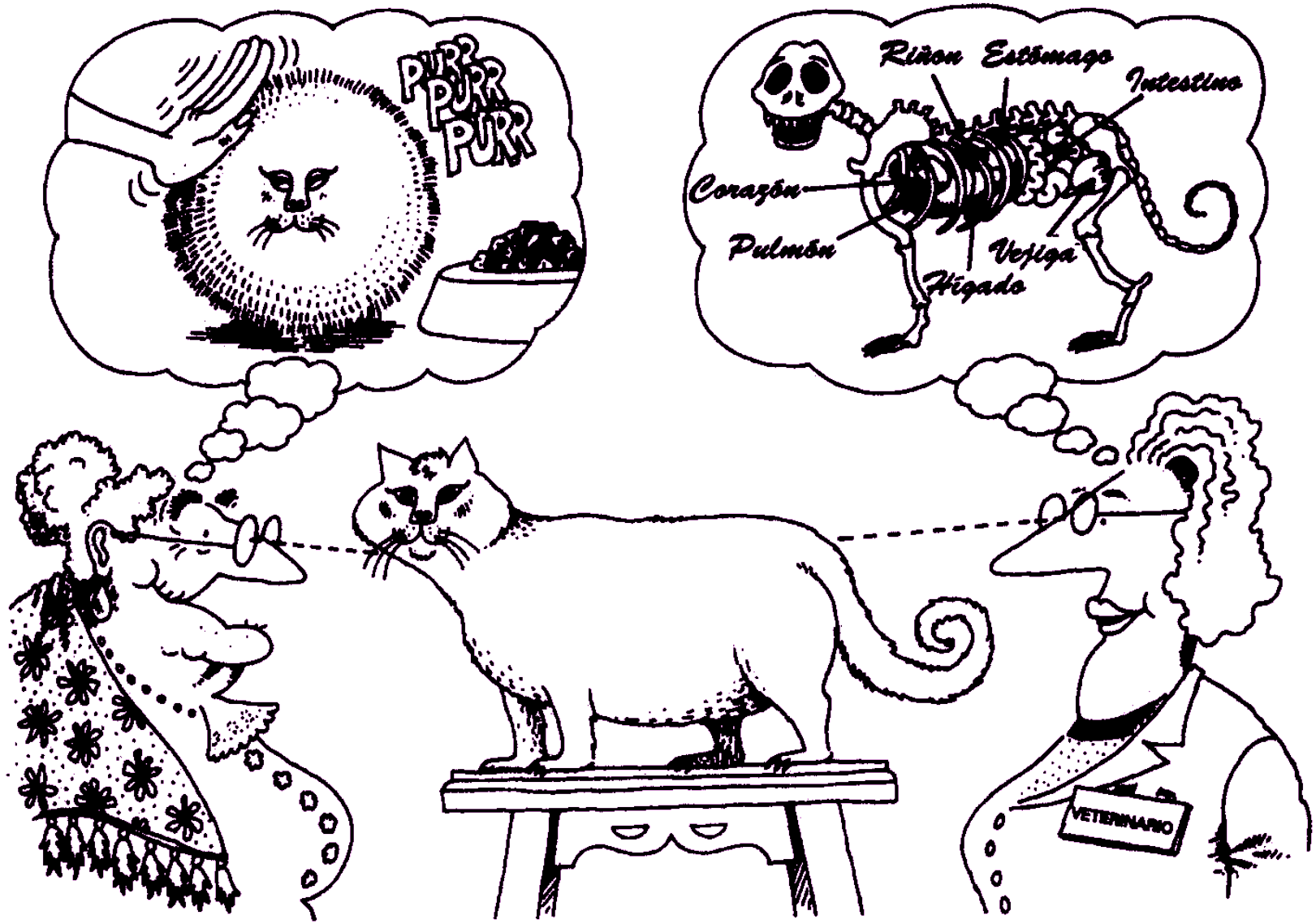
Abstracción (1)

- Es una de las vías fundamentales por la que los humanos combatimos la complejidad.
- Surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias.
- Una buena abstracción es aquella que enfatiza detalles significativos al lector o usuario y suprime detalles que son, al menos por el momento, irrelevantes o causa de distracción.

"Una abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objeto y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador."

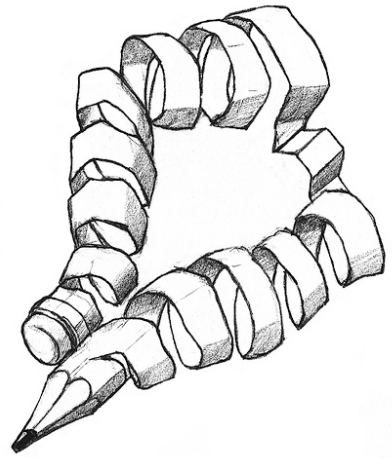


Abstracción (2)



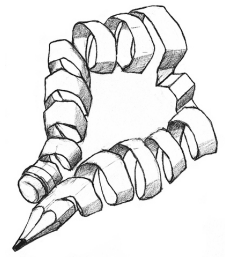
Abstracción (3)

- Una abstracción se centra en la visión externa de un objeto, y, por tanto, sirve para separar el comportamiento esencial de un objeto de su implantación.
- Esta **barrera de abstracción** se consigue aplicando el principio de mínimo compromiso, mediante el cual la interfaz de un objeto muestra su comportamiento esencial y nada más.
- La decisión sobre el conjunto adecuado de abstracciones para determinado dominio es el problema central del diseño orientado a objetos.



Abstracción (4)

- Existen variados tipos de abstracción, que incluyen:
  - **Abstracción de entidades**: un objeto que representa un modelo útil de una entidad del dominio del problema o del dominio de la solución.
  - **Abstracción de acciones**: un objeto que proporciona un conjunto generalizado de operaciones, todas ellas desempeñan funciones del mismo tipo.
  - **Abstracción de máquinas virtuales**: un objeto que agrupa operaciones, todas ellas virtuales utilizadas por algún nivel superior de control, u operaciones que utilizan todas algún conjunto de operaciones de nivel inferior.
  - **Abstracción de coincidencia**: un objeto que almacena un conjunto de operaciones que no tienen relación entre sí.

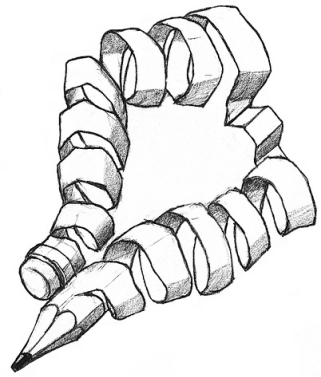


Abstracción (5)

- Se persigue construir abstracciones de entidades, porque imitan directamente el vocabulario de un determinado dominio del problema.
- Se puede caracterizar el comportamiento de un objeto considerando los servicios que presta a otros objetos, así como las operaciones que puede realizar sobre otros objetos.
- **Modelo contractual**: la vista exterior de cada objeto define un contrato del que pueden depender otros objetos (caja negra), y que a su vez debe ser llevado a cabo por la vista interior del propio objeto (a menudo en colaboración con otros objetos).
- Al conjunto completo de operaciones que puede realizar un cliente sobre un objeto, junto con las órdenes que admite, se le denomina su **protocolo**.
- Un protocolo constituye la visión externa completa, estática y dinámica de la abstracción.

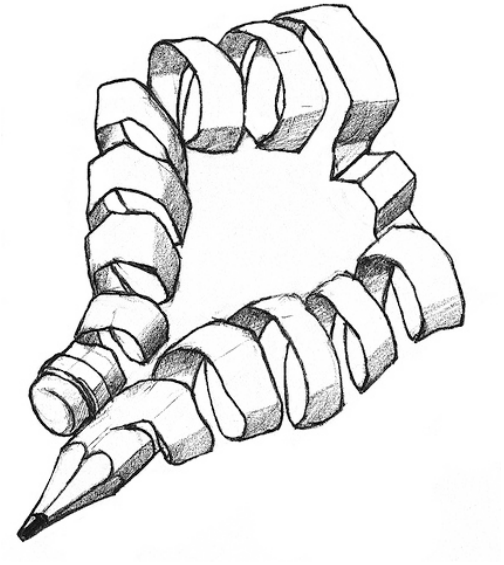
Abstracción (6)

- Todas las abstracciones tienen tanto propiedades estáticas como dinámicas.
  - Por ejemplo, un objeto archivo ocupa una cierta cantidad de espacio en un dispositivo de memoria concreto, tiene un nombre y un contenido; todas estas son propiedades estáticas.
  - El valor de estas propiedades es dinámico, relativo al tiempo de vida del objeto: un objeto archivo puede aumentar o disminuir su tamaño, puede cambiar su nombre y también su contenido.
- En un estilo de programación estructurado, la actividad que cambia el valor dinámico de los objetos es la parte central de todos los programas: ocurren cosas cuando se llama a subprogramas y se ejecutan instrucciones.



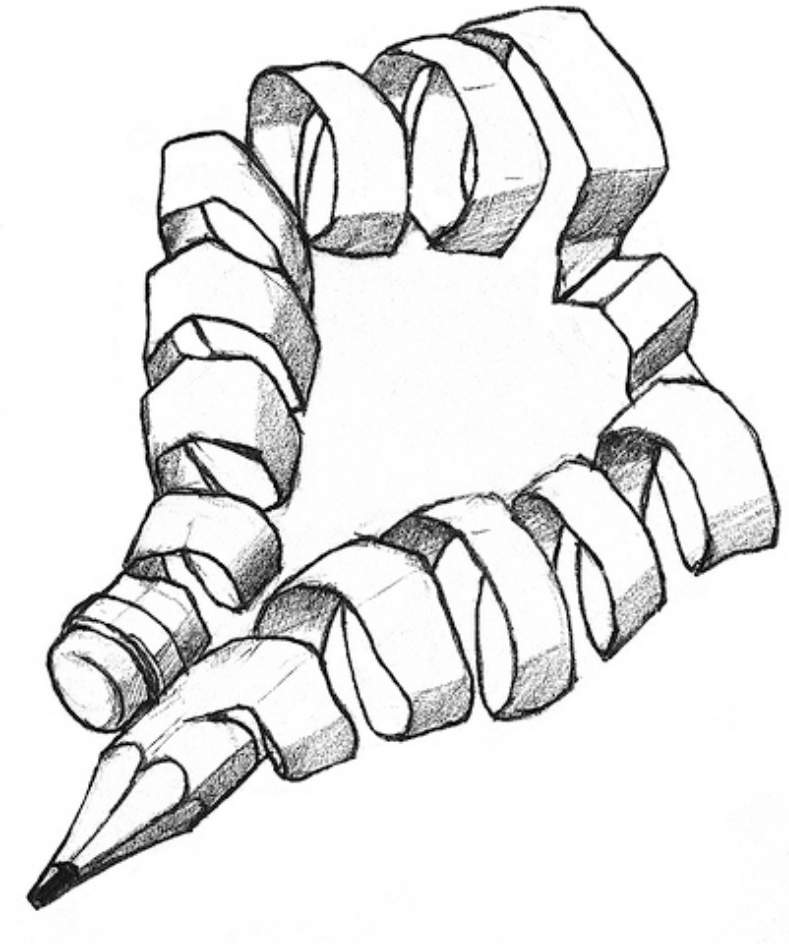
Abstracción (7)

- En un estilo de programación orientado a reglas, ocurren cosas cuando hay eventos que disparan reglas, que a su vez, pueden disparar otras reglas, y así sucesivamente.
- En un estilo de programación orientado a objetos, ocurren cosas cuando se opera sobre un objeto (envío de mensajes a un objeto).
- Las operaciones significativas que pueden realizarse sobre un objeto y las reacciones de este ante ellas constituyen el **comportamiento** completo del objeto.



Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - **Encapsulamiento**
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - Persistencia



Encapsulamiento (1)

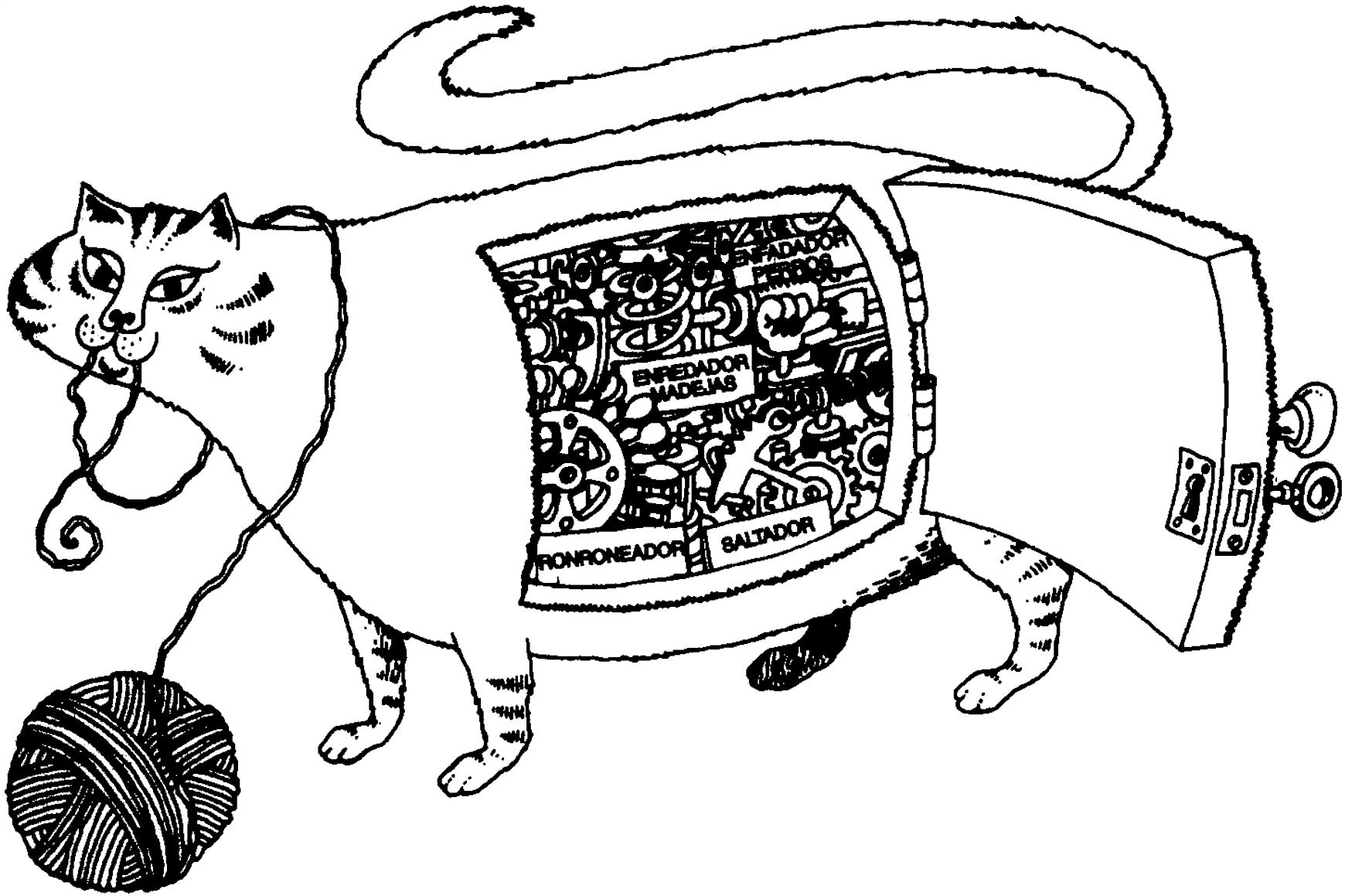
"El encapsulamiento es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar la interfaz de una abstracción y su implementación."

- Una vez que se ha seleccionado una implantación, debe tratarse como un secreto de la abstracción oculto para la mayoría de los objetos clientes.

- Ninguna parte de los sistemas complejos debe depender de los detalles internos de otras partes.

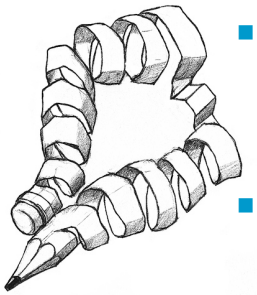
- Mientras la abstracción ayuda a las personas a pensar sobre lo que están haciendo, el encapsulamiento permite que los cambios hechos en los programas sean fiables con el menor esfuerzo.

Encapsulamiento (2)



Encapsulamiento (3)

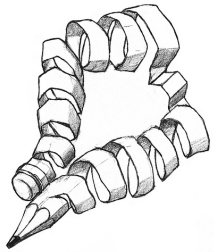
- La abstracción y el encapsulamiento son conceptos complementarios:
  - La abstracción se centra en el comportamiento observable del objeto.
  - El encapsulamiento se centra en la implementación que da lugar a este comportamiento.
- El encapsulamiento se consigue mediante la ocultación de la información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales.



  - Ejemplo: dispositivo electrónico integrado que se vende en el mercado (un contador).
  - Este dispositivo, cuyo comportamiento incluye devolver un valor a la salida que se va incrementando por una señal a la entrada, oculta los detalles de su implementación: no me preocupo, por ejemplo, de cómo están conectadas las diferentes compuertas lógicas que lo componen.

Encapsulamiento (4)

- El encapsulamiento proporciona barreras explícitas entre abstracciones diferentes y por lo tanto conduce a una clara separación de intereses.

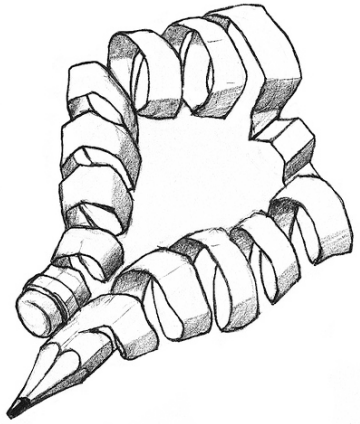


Otro ejemplo: al diseñar una aplicación de bases de datos, es práctica común el escribir programas de forma que no se preocupen de la representación física de los datos, sino que dependan sólo de un esquema que denota la vista lógica de los mismos. Los objetos a un nivel de abstracción están protegidos de los detalles de implementación a niveles más bajos de abstracción.

- Se puede llegar a la conclusión de que para que la abstracción funcione, la implementación debe estar encapsulada. Cada clase debe tener dos partes: una interfaz y una implementación.
- La **interfaz** de una clase captura sólo la vista externa, abarcando la abstracción que se ha hecho del comportamiento común de todas las instancias de una clase.
- La **implementación** de una clase comprende la representación de la abstracción como los mecanismos que consiguen el comportamiento deseado.

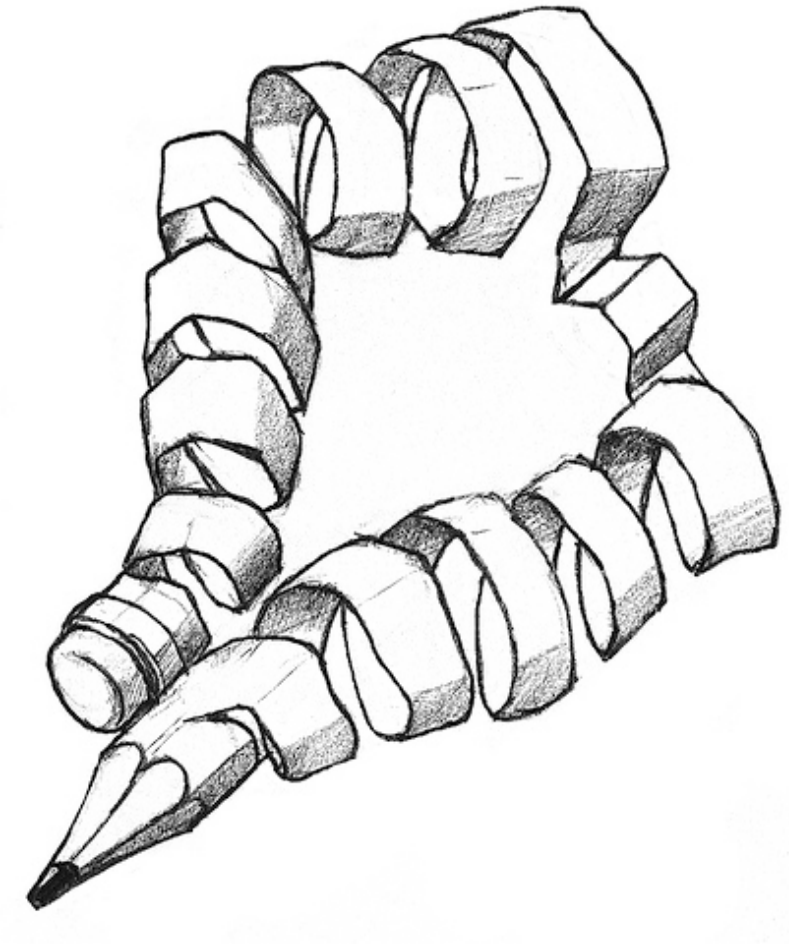
Encapsulamiento (5)

- En la interfaz de una clase es en el único lugar en el que se declaran todas las suposiciones que un cliente que va a usar esa clase puede hacer acerca de todas las instancias de la misma
- La implantación encapsula detalles acerca de los cuales ningún cliente puede realizar suposiciones.
- La ocultación es un concepto relativo: lo que está oculto a nivel de abstracción puede representar la visión externa a otro nivel de abstracción.
- En la práctica, hay ocasiones en las que es necesario estudiar la implantación de una clase para comprender realmente su significado, especialmente si la documentación externa es deficiente.



Elementos del Modelo de Objetos (3)

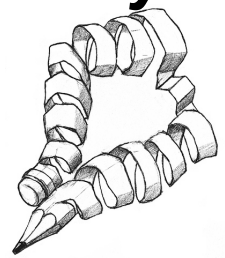
- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - **Modularidad**
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - Persistencia



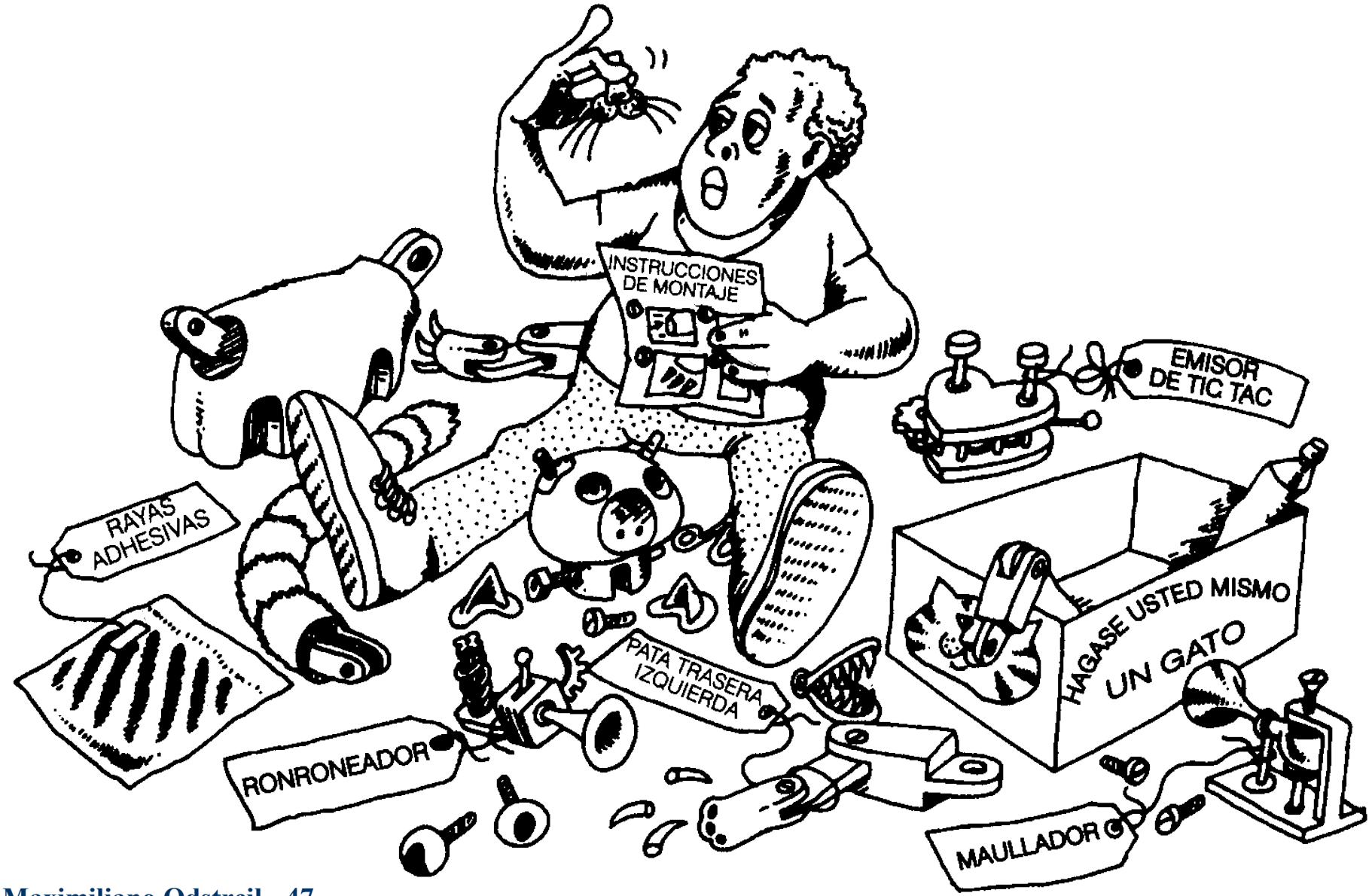
Modularidad (1)

- El acto de fragmentar un programa en componentes individuales puede reducir su complejidad en algún grado.
- Esta fragmentación crea una serie de fronteras bien definidas y documentadas dentro del programa.
- Estas fronteras o interfaces, tienen un incalculable valor de cara a la comprensión del programa.
- Especialmente para aplicaciones industriales, en las que puede haber varios cientos de clases, el uso de módulos es esencial para ayudar a manejar la complejidad.

"La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados.



Modularidad (2)



Modularidad (3)

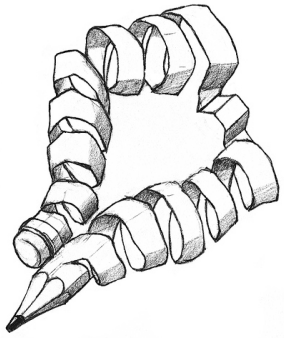
- La modularización consiste en dividir un programa en módulos que pueden compilarse separadamente, pero que tienen conexiones con otros módulos.
- La mayoría de los lenguajes distinguen entre la interfaz y la implementación del módulo, es decir que la modularidad y el encapsulamiento van de la mano.
- La decisión sobre el conjunto adecuado de módulos para determinado problema, es tan difícil como decidir sobre el conjunto adecuado de abstracciones.
- Los módulos sirven como contenedores físicos en los que se declaran las clases y objetos del diseño lógico realizado.
- Para problemas muy pequeños el desarrollador podría decidir declarar todas las clases y objetos en el mismo paquete.

Modularidad (4)

- Para problemas no triviales, es mejor agrupar las clases y objetos que se relacionan lógicamente en el mismo módulo, y exponer solamente los elementos que otros módulos necesitan ver con absoluta necesidad.
- En el diseño estructurado tradicional, los criterios de agrupamiento son los de **acoplamiento** y **cohesión** .
- En el diseño orientado a objetos, la tarea es decidir dónde empaquetar físicamente las clases y objetos a partir de la estructura lógica del diseño.
- El objetivo de fondo de la modularización es la reducción del costo del software al permitir que los módulos se diseñen y revisen independientemente.
- El desarrollador debe equilibrar dos intereses técnicos opuestos:
  - El deseo de encapsular abstracciones
  - La necesidad de hacer a ciertas abstracciones visibles para otros módulos.

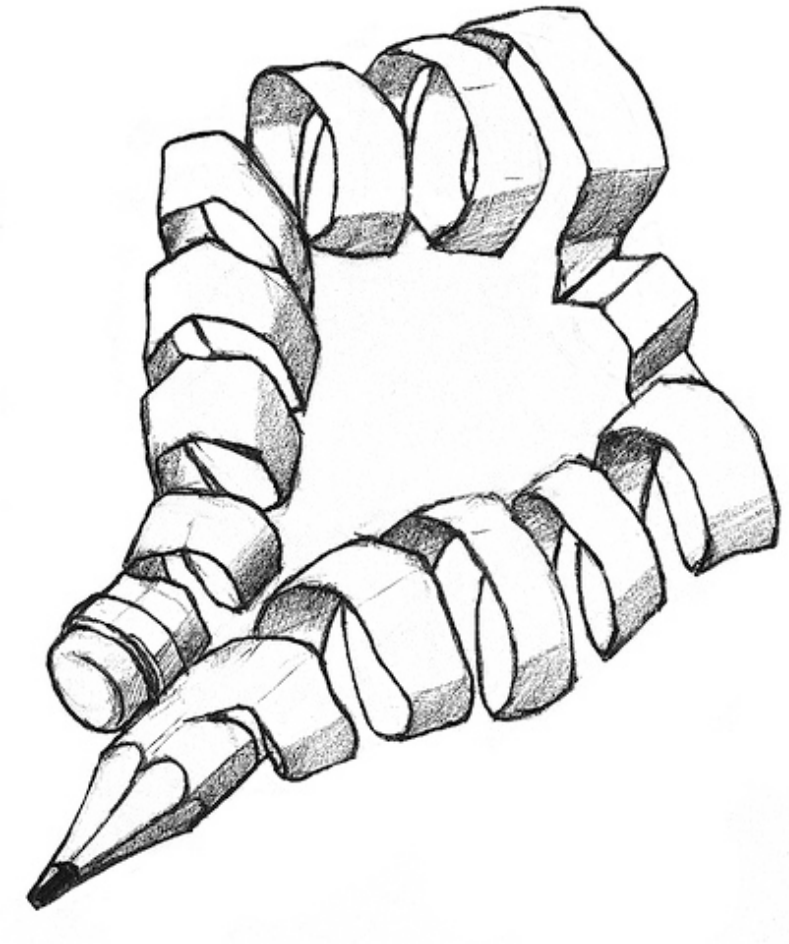
Modularidad (5)

- Hay que hacer lo posible, entonces, por construir:
  - **Módulos cohesivos**: agrupando abstracciones que guarden cierta relación lógica.
  - **Débilmente acoplados**: minimizando la dependencia entre módulos.
- Los principios de abstracción, encapsulamiento y modularidad son sinérgicos: un objeto proporciona una frontera bien definida alrededor de una abstracción, y tanto el encapsulamiento como la modularidad proporcionan barreras que rodean a esta abstracción.
- Las causas que pueden afectar a las decisiones sobre la modularidad son:
  - La necesidad de reutilización de código por parte de un equipo de desarrolladores.
  - La asignación de trabajo típica de un equipo de desarrolladores.
  - La seguridad, condicionada por los requisitos del problema.



Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - **Jerarquía**
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - Persistencia

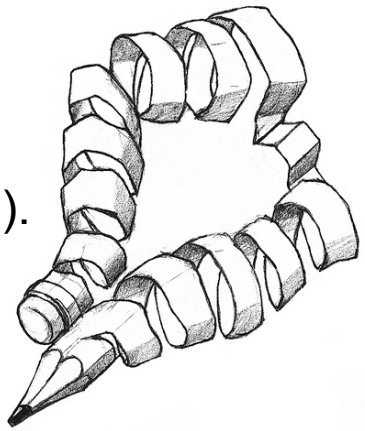


Jerarquía (1)

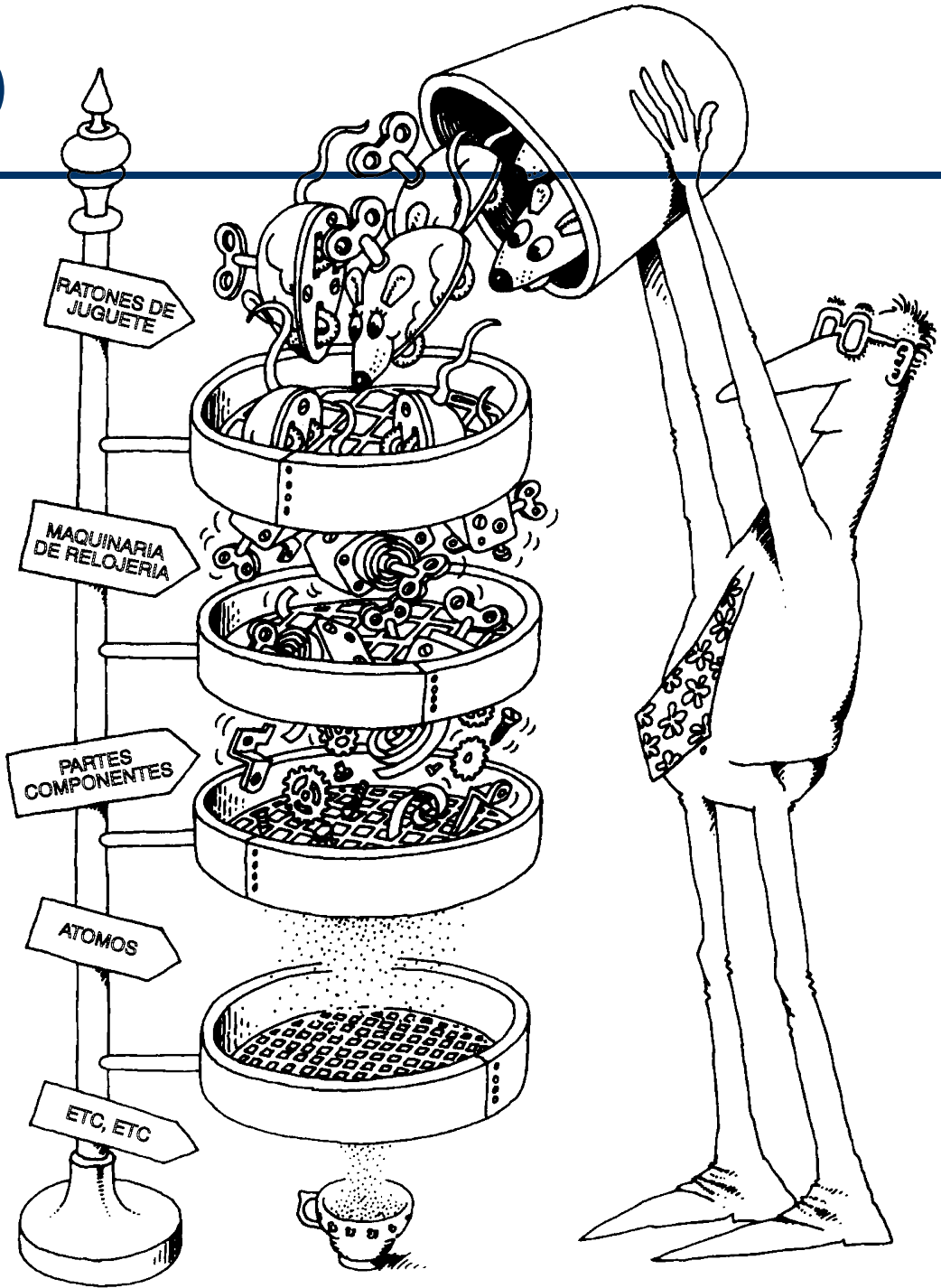
- La abstracción, el encapsulamiento y la modularidad son algo bueno, pero excepto en las aplicaciones más triviales, puede haber muchas más abstracciones diferentes de las que se puede comprender simultáneamente.
- Frecuentemente, un conjunto de abstracciones forma una jerarquía, y la identificación de esas jerarquías en el diseño simplifica en gran medida la comprensión del problema.

"La jerarquía es una clasificación u ordenación de abstracciones."

- Las dos jerarquías más importantes en un sistema complejo son:
  - Su estructuras de clases (jerarquía de clases o jerarquía "es-un")
  - Su estructura de objetos (jerarquía de partes o jerarquía "parte-de").



Jerarquía (2)


RATONES DE JUGUETE
MAQUINARIA DE RELOJERIA
PARTES COMPONENTES
ATOMOS
ETC, ETC

Jerarquía (3)

- La herencia es la jerarquía de clases más importante y es un elemento esencial en los sistemas orientados a objetos.
- La herencia define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida en una o más clases. Así tenemos:
  - **Herencia Simple** (comparte la estructura de una clase)
  - **Herencia Múltiple** (comparte la estructura de una o más clases)
- Una subclase hereda de una o más superclases, aumentando o redefiniendo generalmente la estructura y el comportamiento de sus superclases.
  - Por ejemplo un oso es-un tipo de mamífero, una casa es-un tipo de inmueble y un círculo es-un tipo de figura geométrica.
- Así la herencia implica una jerarquía de generalización/especialización en la que una subclase especializa el comportamiento o estructura de sus superclases.

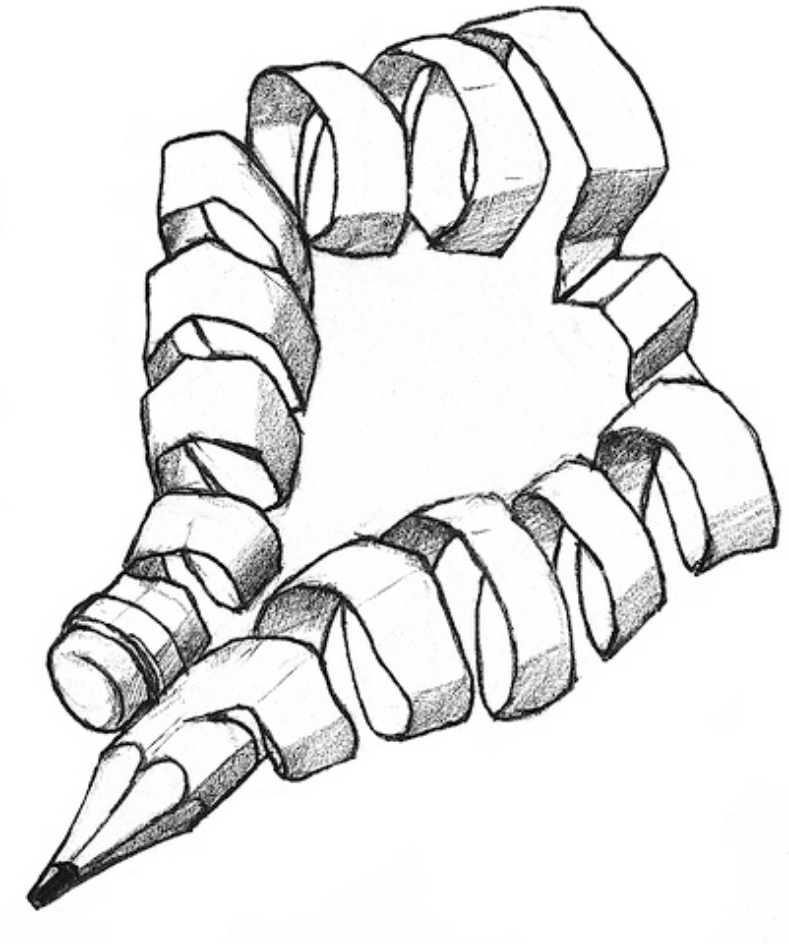
Jerarquía (4)

- La estructura y comportamiento comunes a diferentes clases, tenderá a migrar hacia superclases comunes. De este modo, la herencia permite declarar las abstracciones con economía de expresión.
  - Por ejemplo si en un problema se tienen las abstracciones alumnos y profesores, se podrá definir una superclase de ambas llamada personas, que agrupará cierto comportamiento y estructura común de ambos (apellidos, nombres, etc.)
- Las jerarquías “es-un” denotan relaciones de generalización/especialización.
- Las jerarquías “parte-de” describen relaciones de agregación.
- Estas jerarquías, se conocen a menudo como niveles de abstracción, donde una abstracción de alto nivel está generalizada (todo) y una abstracción de bajo nivel está especializada (parte).

  - Por ejemplo, podemos decir que un ser humano está compuesto por cabeza, tronco y extremidades, estando la clase “ser humano” a nivel más alto de abstracción y las clases cabeza, tronco y extremidades a nivel más bajo de abstracción.

Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - **Tipos (Tipificación)**
  - Concurrencia
  - Persistencia

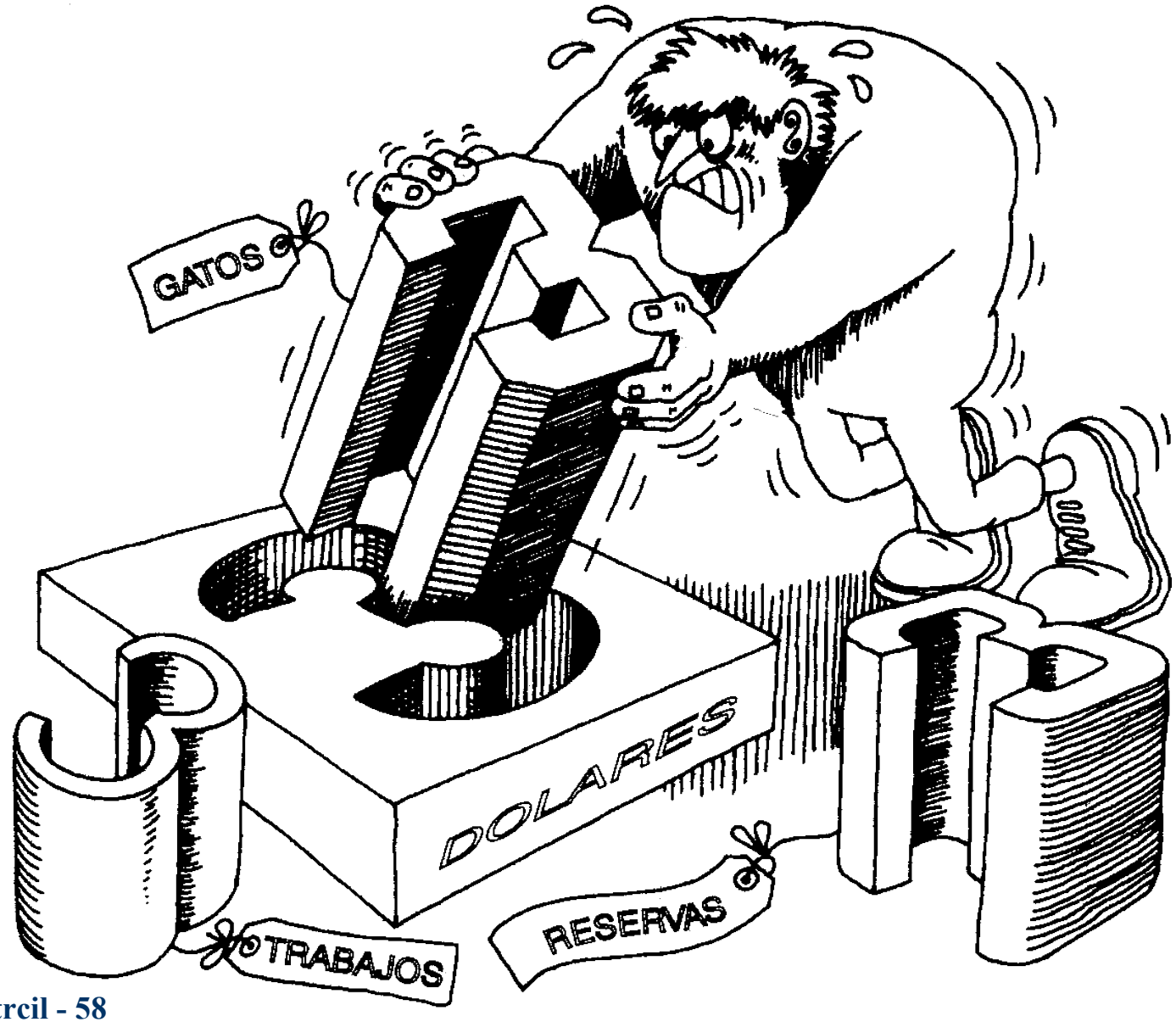


Tipos (1)

"Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de distintos tipos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas."

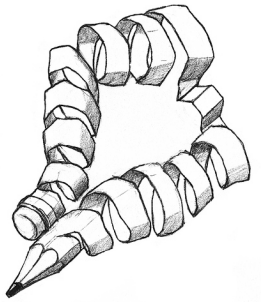
- El concepto de tipo se deriva en primer lugar de la teoría de los tipos de datos abstractos (ADT).
- Un tipo es una caracterización precisa de propiedades estructurales o de comportamiento que comparten una serie de entidades.
- Se usarán los términos tipo y clase de manera intercambiable, aunque se incluyen los tipos como elemento separado del modelo de objetos porque el concepto de tipo pone énfasis en el significado de la abstracción en un sentido muy distinto.

Tipos (2)



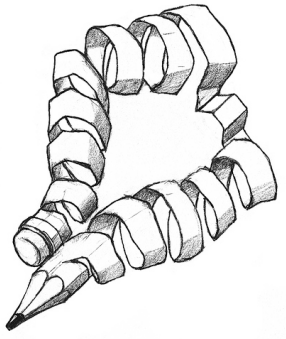
Tipos (3)

- Los tipos permiten expresar las abstracciones de manera que el lenguaje de programación en el que se implantan puede utilizarse para apoyar las decisiones de diseño.
- La idea de congruencia es central a la noción de tipos. Las reglas del dominio dictan y refuerzan ciertas combinaciones correctas de las abstracciones:
  - Por ejemplo, consideremos las unidades de medida en física: si se divide distancia en tiempo, se espera un valor que denote velocidad, no peso; multiplicar temperatura por peso no tiene sentido, pero sí lo tiene multiplicar masa por aceleración.
- Un lenguaje de programación puede tener comprobación estricta de tipos, comprobación débil de tipos, o incluso no tener tipos, y aún así ser considerado como orientado a objetos



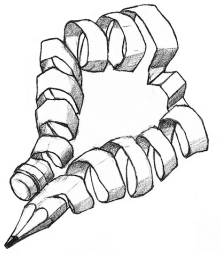
Tipos (4)

- Lenguajes con comprobación estricta de tipos:
  - Puede detectarse en tiempo de compilación cualquier violación a la concordancia de tipos.
  - Sirve para imponer ciertas decisiones de diseño.
  - Son particularmente relevantes a medida que aumenta la complejidad del sistema.
- La principal desventaja del uso de tipos estrictos es la pérdida de flexibilidad.
- Además introducen dependencias semánticas donde un pequeño cambio en la interfaz de la clase puede requerir la recompilación de todas sus subclases.



Tipos (5)

- Las principales ventajas que aportan los tipos estrictos son:
  - Los programas no “explotan” de forma misteriosa en ejecución.
  - El ciclo editar/compilar/depurar se minimiza debido a la detección temprana de errores.
  - Ayuda a auto-documentar los programas.
  - Los compiladores generan un código más eficiente.



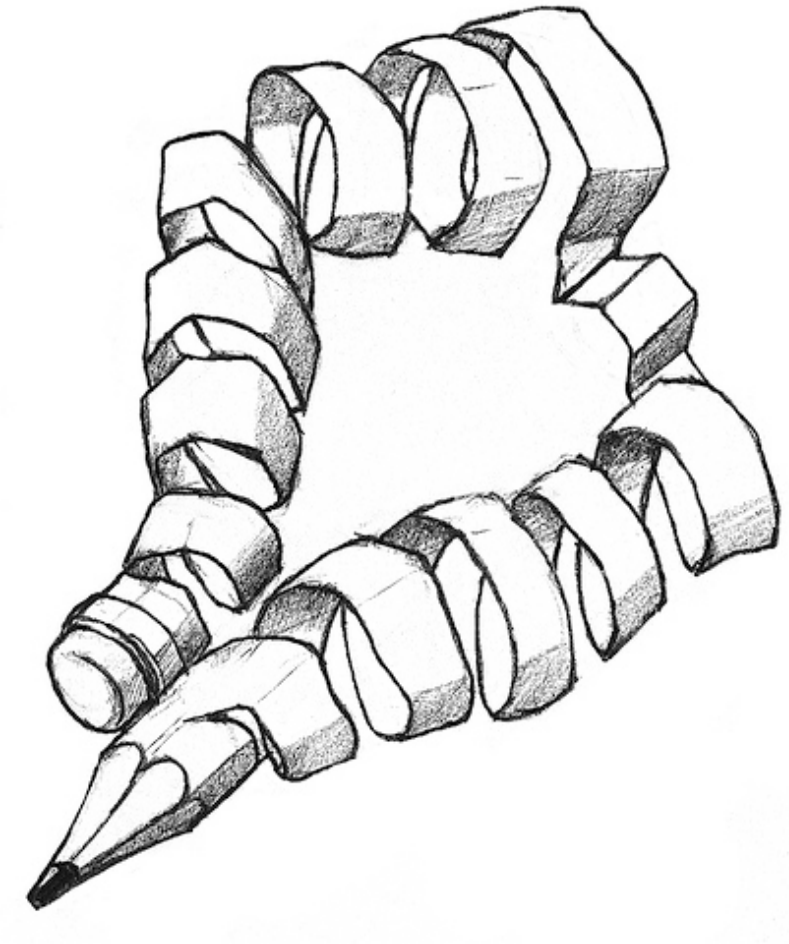
- Un concepto muy importante asociado a la teoría de tipos es el de polimorfismo.
- El **polimorfismo** permite que un solo nombre (tal como la declaración de una variable) pueda denotar objetos de muchas clases diferentes que se relacionan por alguna superclase común.
- Cualquier objeto denotado por ese nombre es, por lo tanto, capaz de responder a algún conjunto común de operaciones.

Tipos (6)

- El polimorfismo existe cuando interactúan las características de herencia y el enlace dinámico y es quizá la característica más potente de los lenguajes orientados a objetos.
  - Por ejemplo, en un problema se cuenta con la superclase Figuras, cuyas subclases son Puntos, Rectas, Rectángulos y Círculos.
  - El comportamiento de las figuras incluye su posibilidad de ser dibujadas, pero no todas las figuras se dibujan de igual manera.
  - Así la operación dibujar se define a nivel figura, pero la respuesta a dicha operación será distinta de acuerdo al tipo denotado por el nombre de la variable.

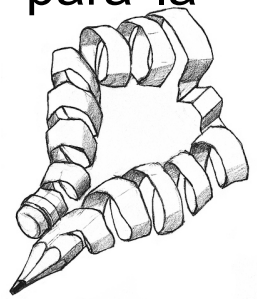
Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - **Concurrencia**
  - Persistencia



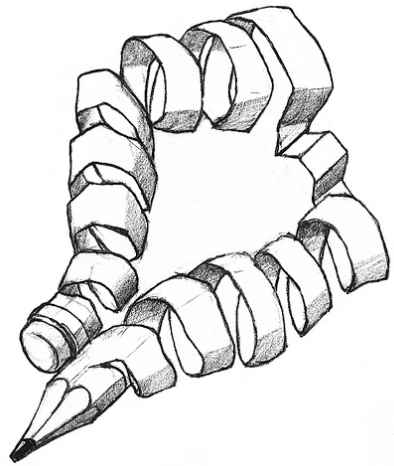
Concurrencia (1)

- Para ciertos tipos de problema, un sistema puede manejar muchos eventos diferentes simultáneamente.
- Otros problemas pueden implicar tantos cálculos que excedan la capacidad de cualquier procesador individual.
- Es natural considerar el uso de un conjunto distribuido de computadoras o utilizar procesadores capaces de realizar multitarea, que implica la **concurrencia** de los procesos , es decir permitir a diferentes procesos actuar al mismo tiempo, o al menos, conseguir la ilusión (en sistemas monoprocesadores) de que lo está haciendo.
- Muchos sistemas operativos actuales proporcionan soporte directo para la concurrencia, y por lo tanto existe una gran oportunidad (y demanda) para la concurrencia en sistemas orientados a objetos.



Concurrencia (2)

- En la POO, el manejo de la concurrencia no es diferente al de otro tipo de lenguajes, ya que hay que existen los mismos problemas:
  - Abrazo mortal
  - Inanición
  - Exclusión mutua
  - Condiciones de competencia.

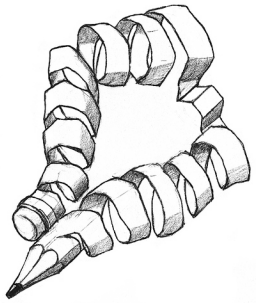


- A los niveles más altos de abstracción, la POO puede aliviar el problema de la concurrencia mediante la ocultación de la misma dentro de abstracciones reutilizables.
- Implícitamente un modelo de objetos define las unidades de distribución y movimiento y las entidades que se comunican.

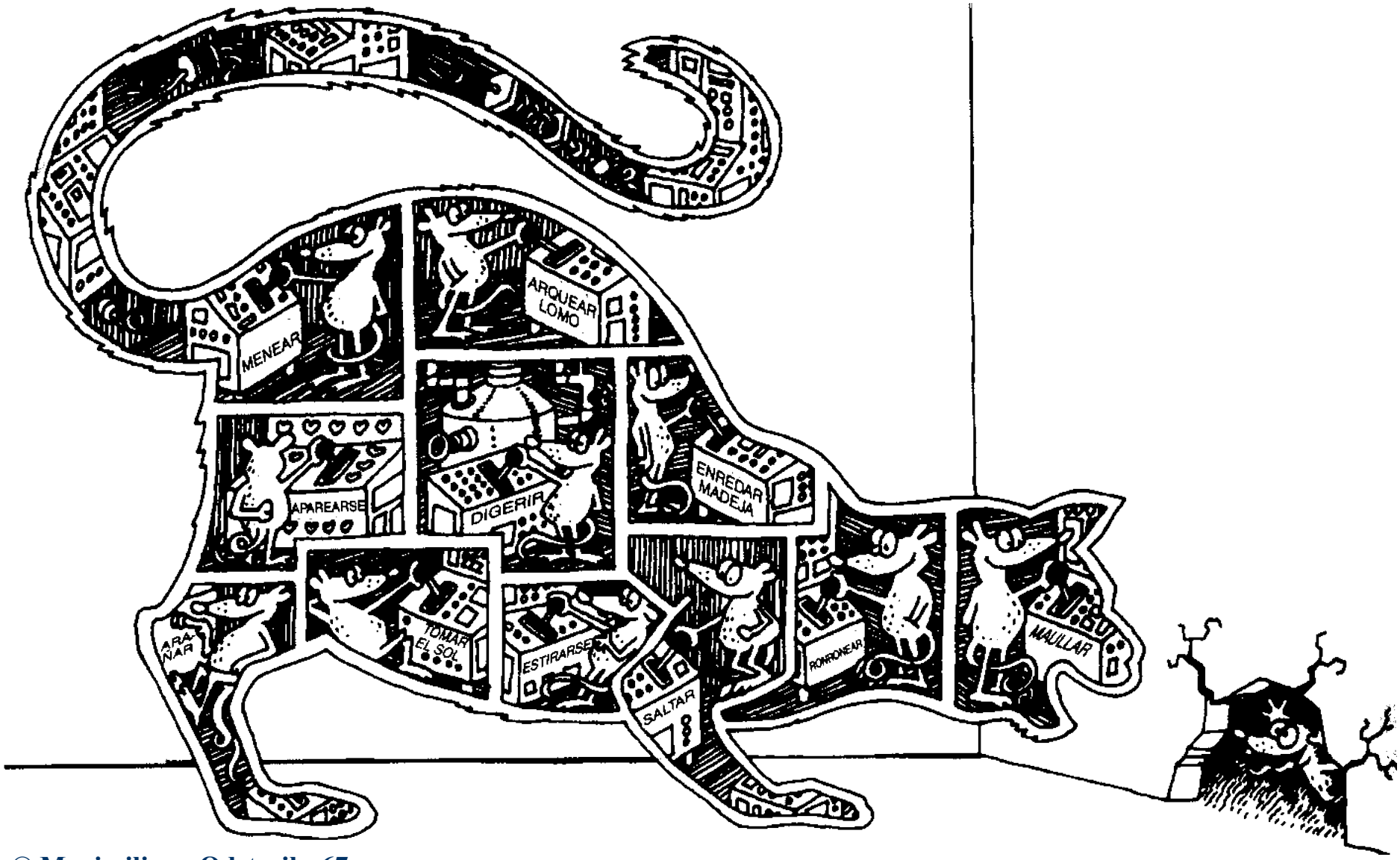
Concurrencia (3)

- Mientras que la programación orientada a objetos se centra en la abstracción de datos, encapsulamiento y herencia, la concurrencia se centra en la abstracción de procesos y la sincronización.
- Cada objeto puede representar un hilo separado de control (una abstracción de un proceso); tales objetos se llaman **activos**.
- En un sistema basado en objetos, se puede conceptualizar el mundo como un conjunto de objetos corporativos, algunos de los cuales son activos y sirven así como centros de actividad independiente.

"La concurrencia es la propiedad que distingue un objeto activo de uno que no está activo."



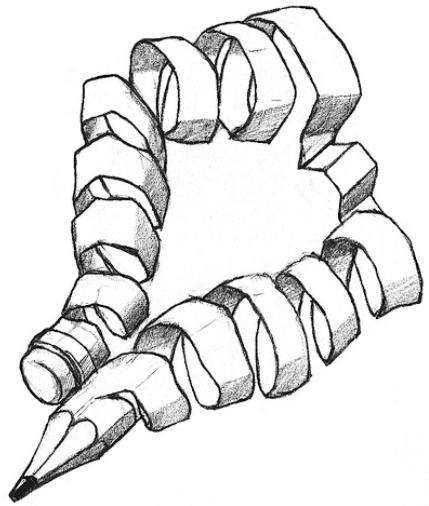
Concurrencia (4)



Concurrencia (5)

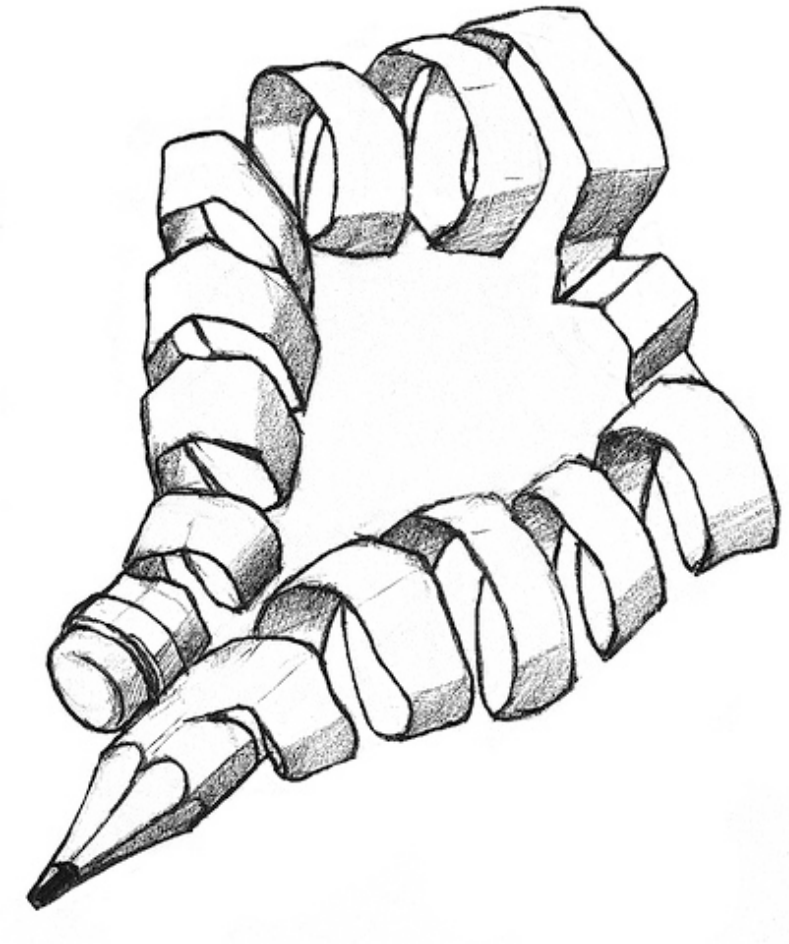
- Una de las realidades acerca de la concurrencia es que, una vez que se la introduce en un sistema, hay que considerar cómo los objetos activos sincronizan sus actividades con otros, así como con objetos puramente secuenciales.

- Como conclusión, en presencia de la concurrencia, no es suficiente con definir simplemente los métodos de los objetos; hay que asegurarse también de que la semántica de estos métodos se mantiene a pesar de la existencia de múltiples hilos de control.



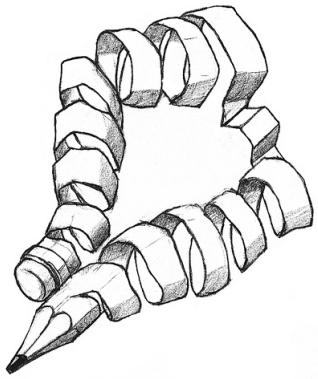
Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - **Persistencia**



Persistencia (1)

- Un objeto ocupa una cierta cantidad de espacio, y existe durante una cierta cantidad de tiempo.
- Hay un espacio continuo de existencia del objeto, que abarca desde los objetos transitorios, que surgen de la evaluación de una expresión, hasta los objetos de una base de datos, que sobreviven a la ejecución de un único programa.
- Este espectro de persistencia incluye, además, los siguientes objetos:
  - Resultados transitorios de la evaluación de expresiones.
  - Variables locales en la activación de procedimientos.
  - Variables propias, variables locales y elementos del stack (pila) cuya duración difiere de su ámbito.
  - Datos que existen entre ejecuciones de un programa.
  - Datos que existen entre varias versiones de un programa.
  - Datos que sobreviven al programa.



Persistencia (2)

- Los lenguajes de programación tradicionales suelen tratar solamente los tres primeros tipos de persistencia de objetos.
- La persistencia de los otros tres pertenece al dominio de la tecnología de las bases de datos.
- La introducción de la persistencia en el modelo de objetos da lugar a la aparición de bases de datos orientadas a objetos.
- En la práctica, tales bases de datos se construyen sobre tecnología contrastada, como modelos de bases de datos secuenciales, indexadas, jerárquicas, en red o relacionales.
- Ofrecen al programador la abstracción de una interfaz orientada a objetos, a través de la cual las consultas y otras operaciones se llevan a cabo en términos de objetos cuyo ciclo de vida trasciende el ciclo de vida de un programa individual.

Persistencia (3)

- El almacenar objetos en simples archivos es una solución ingenua para la persistencia, una solución que no resiste bien los cambios de escala.
- La persistencia se consigue a través de un pequeño número de bases de datos orientadas a objetos disponibles comercialmente.
- Otro enfoque razonable para la persistencia es proporcionar una cáscara orientada a objetos, bajo la cual subyace una base de datos relacional
- Esta última solución es bastante atractiva dada la inversión de capital en tecnología de bases de datos relacionales que existe y es la que se tratará en este curso.
- La persistencia abarca algo más que la mera duración de los datos. No sólo debe persistir el estado de un objeto, sino que su clase debe transcender también a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado almacenado.

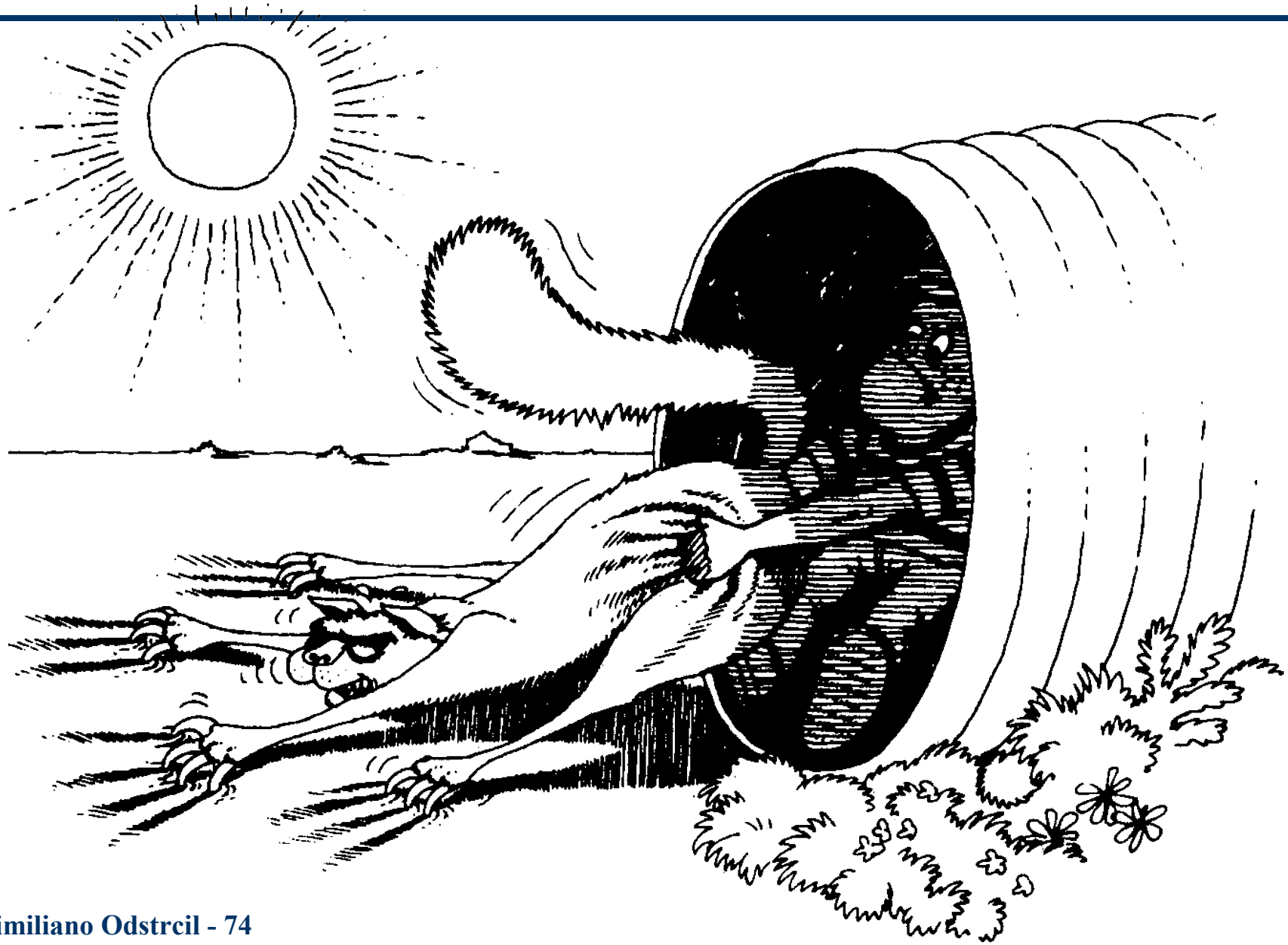
Persistencia (4)

- En la mayoría de los sistemas, nos preocupamos por la persistencia del objeto en el tiempo, pero en sistemas que se ejecutan en un conjunto distribuido de procesadores, a veces hay que preocuparse de la persistencia del objeto en el espacio.

- En estos sistemas es útil pensar que los objetos pueden llevarse de una máquina a otra y que incluso pueden tener representaciones diferentes en máquinas diferentes.

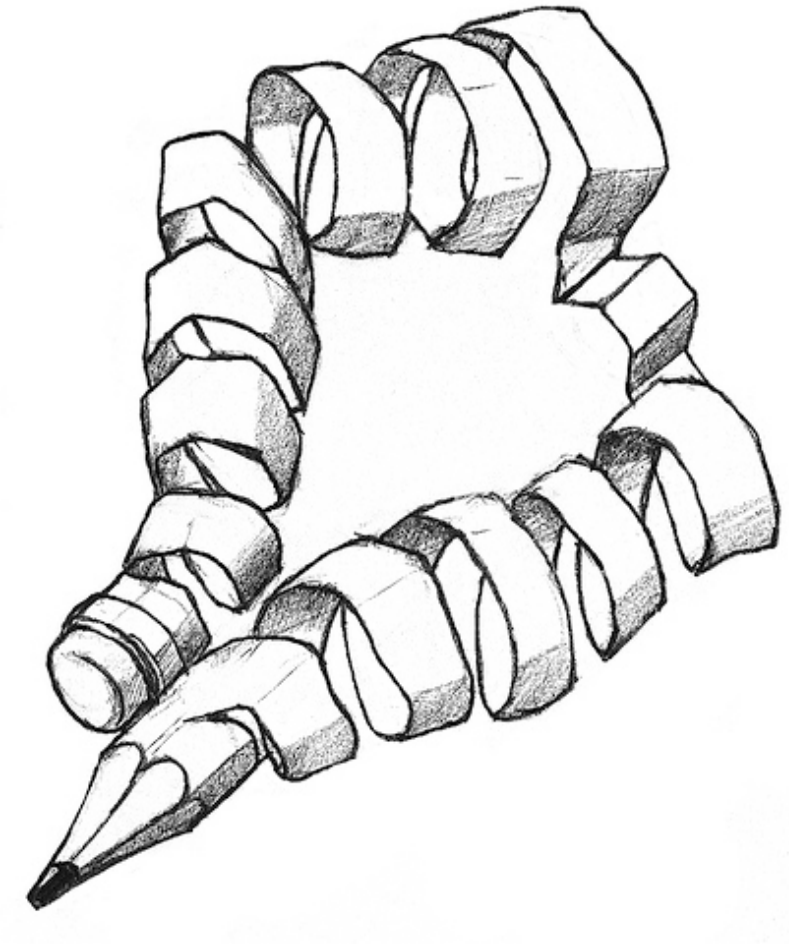
La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo (el objeto continúa después de que su creador deja de existir) y/o el espacio (la posición del objeto varía con respecto al espacio de direcciones en el que fue creado).

Persistencia (5)



Elementos del Modelo de Objetos (3)

- Elementos Fundamentales:
  - Abstracción
  - Encapsulamiento
  - Modularidad
  - Jerarquía
- Elementos Secundarios:
  - Tipos (Tipificación)
  - Concurrencia
  - Persistencia



El Modelo de Objetos

- Contenido
  - Introducción
  - Evolución del Modelo de Objetos
  - Fundamentos del Modelo de Objetos
  - Elementos del Modelo de Objetos
  - **Aplicación del Modelo de Objetos**

Aplicación del Modelo de Objetos (1)

- El modelo de objetos es fundamentalmente diferente de los modelos adoptados por los métodos de análisis, diseño y programación estructurada.
- Esto no significa que el modelo de objetos abandone todos los sanos principios y experiencias de métodos más viejos. En lugar de eso:
  - Introduce varios elementos novedosos que se basan en estos modelos anteriores.
  - Proporciona una serie de beneficios significativos que otros modelos simplemente no ofrecen.
  - Su uso conduce a la construcción de sistemas que incluyen los cinco atributos de los sistemas complejos bien estructurados.

Aplicación del Modelo de Objetos (2)

- Los cinco beneficios que se derivan de la aplicación del modelo de objetos:
  - El uso del modelo de objetos ayuda a explotar la potencia expresiva de los lenguajes de programación orientados a objetos. (Productividad)
  - Promueve la reutilización no sólo de software, sino de diseños enteros, conduciendo a la creación de marcos de desarrollo de aplicaciones reusables. (Beneficios de costo y planificación).
  - Produce sistemas que se construyen sobre formas intermedias estables, que son flexibles al cambio (Evolución en el tiempo).
  - Reduce los riesgos inherentes al desarrollo de sistemas complejos.
  - Resulta atractivo para el funcionamiento del conocimiento humano ya que las personas que no saben cómo funciona una computadora, encuentran bastante natural la idea de los sistemas orientados a objetos.

Aplicación del Modelo de Objetos (3)

- El modelo de objetos ha demostrado ser aplicable a una amplia variedad de dominios de problema.
- Puede ser que el diseño orientado a objetos sea el único método entre los disponibles hoy en día, que pueda emplearse para atacar la complejidad innata a muchos sistemas grandes.
- NO EXISTE la herramienta perfecta para todos los problemas. El uso del paradigma objetos no es aconsejable en ciertos dominios, no por razones técnicas, sino porque no hay personal con el entrenamiento adecuado o bien porque existe otro enfoque cuya aplicación resulta más productiva.

Aplicación del Modelo de Objetos (3)

Aeronáutica	Fabricación integrada por computadora
Análisis matemático	Hipermedia
Animación	Ingeniería petroquímica
Automatización de oficinas	Preparación de documentos
Bases de datos	Preparación de películas y escenarios
Componentes de software reutilizables	Proceso de datos de negocios
Composición de música	Reconocimiento de imágenes
Control de procesos químicos	Robótica
Control de tráfico aéreo	Simulación de cohetes y aviones
Diseño asistido por computadora	Sistemas de dirección y control
Diseño de interfaces de usuario	Sistemas de telemetría
Diseño VLSI	Sistemas expertos
Electrónica médica	Sistemas operativos
Enseñanza asistida por computadora	Software de banca y seguros
Entornos de desarrollo de software	Software de estaciones espaciales

El Modelo de Objetos

- Contenido
  - Introducción
  - Evolución del Modelo de Objetos
  - Fundamentos del Modelo de Objetos
  - Elementos del Modelo de Objetos
  - Aplicación del Modelo de Objetos