

## Programación: Actividad compleja

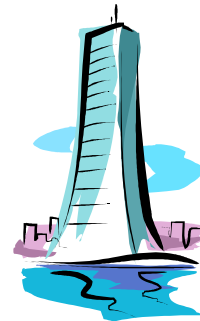
- ❑ Los problemas que se intentan resolver con el software implican elementos complejos propios del área a la que pertenecen.
- ❑ Además, es difícil gestionar el proceso de desarrollo de software.



3

## Programación: Actividad compleja (2)

- ❑ “Un constructor pensaría raramente en añadir un subsótano a un edificio ya construido de 100 plantas... Los usuarios de sistemas de software casi nunca lo piensan dos veces a la hora de solicitar cambios equivalentes...De todas formas (dicen ellos) es simplemente cosa de programar”



4

## Crisis del software



- La incapacidad humana de dominar la complejidad del software conlleva a:
  - Proyectos retrasados
  - Proyectos que exceden el presupuesto
  - Proyectos deficientes que no cumplen los requerimientos

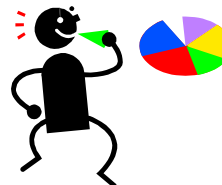
5

## Imponiendo orden al caos

- “La técnica de dominar la complejidad se conoce desde tiempos remotos: *divide et impera* (divide y vencerás)”.

[Dijkstra]

- “Para entender un nivel dado de un sistema, basta con comprender unas pocas partes (no necesariamente todas) a la vez”.
- Descomposición es la clave:
  - Descomposición algorítmica tradicional
  - Descomposición orientada a objetos



6

## Programación Orientada a Objetos

- ❑ POO es un conjunto de técnicas que pueden utilizarse para desarrollar programas eficientemente.
- ❑ Los objetos son los elementos principales de construcción.
- ❑ La Orientación a Objetos (OO) es el estilo dominante de programación, descripción y modelado de hoy en día.

7

## 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 miembros de una jerarquía de clases unidas mediante relaciones de herencia”

Grady Booch

8

## Origen de la POO



- ❑ Ole-Johan Dahl y Kristen Nygaard, científicos y profesores del Norwegian Computing Center son considerados los “padres” de la Orientación a Objetos.
- ❑ Ellos introdujeron los conceptos principales de OO, y crearon los lenguajes de programación SIMULA 1 (1961-1965) y SIMULA 67 (1965-1968) que los implementaban.

9

## Los conceptos de OO introducidos por SIMULA...

- ❑ Ya incluían los conceptos de clase, subclase, herencia, creación dinámica de objetos y ocultamiento de información.
- ❑ Proviene del mundo real.
- ❑ Tardaron aprox. 20 años en ganar entendimiento y popularidad.
- ❑ Han influenciado a los lenguajes de programación modernos, a las metodologías y a los lenguajes de modelado.
- ❑ Los han adoptado lenguajes como Smalltalk, C++, Java y C#



10

## El modelo de Objetos

- ❑ Objetos en el mundo real
  - Propiedades
  - Métodos
- ❑ Abstracción
- ❑ Clases y Objetos
- ❑ Encapsulamiento
- ❑ Mensajes
- ❑ Constructores
- ❑ Destructores
- ❑ Herencia
  - Simple
  - Múltiple
- ❑ Clases Abstractas
- ❑ Sobreescritura
- ❑ Sobrecarga
- ❑ Polimorfismo



11

## Objetos en el mundo real



Lavadora



Perro



Televisión



Persona



Factura

12



## Podemos darnos cuenta que...

- ❑ Los objetos poseen características que los distinguen entre sí.
- ❑ Los objetos tienen acciones asociadas a ellos.

13

## Ejemplo: PERRO



- ❑ Características:
  - Nombre: "FIDO"
  - Raza: "Chihuahua"
  - Color: "Café"
  - ....etc...
- ❑ Acciones:
  - Ladrar ["Guau Guau"]
  - Comer ["Chomp Chomp"]
  - Dormir ["Zzzzzzzz"]
  - ....etc...

14

## ¿Cómo modelar un objeto real en un programa?

- ❑ Las “características” son PROPIEDADES o datos.
- ❑ Las “acciones” son METODOS u operaciones.



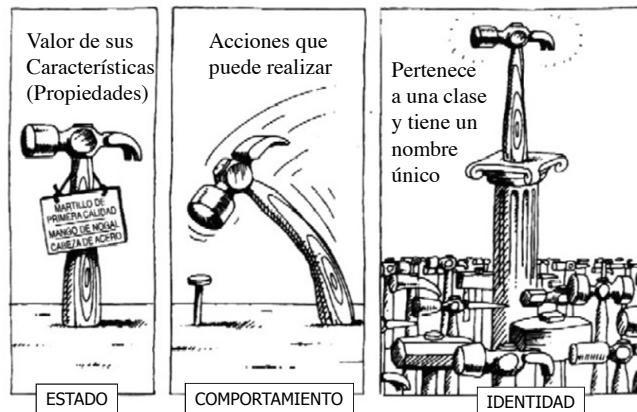
Objeto Perro “Real”

FIDO : Perro
Nombre: FIDO Raza: Chihuahua Color: Café
Ladrar() Comer() Dormir()

Abstracción de un objeto “Perro” en software

15

## Todos los objetos tienen Estado, Comportamiento e Identidad



16

# Abstracción

- Se refiere a “quitar” propiedades y métodos de un objeto y quedarse solo con aquellos que sean necesarios (relevantes para el problema a solucionar).



Objeto Perro “Real”:  
**Propiedades:**  
(Nombre, Raza, Color, **Edad, Tamaño, etc.**)  
**Acciones o métodos:**  
(Ladrar, Comer, Dormir, **Jugar, Caminar, etc.**)

FIDO : Perro
Nombre: FIDO
Raza: Chihuahua
Color: Café
Ladrar()
Comer()
Dormir()

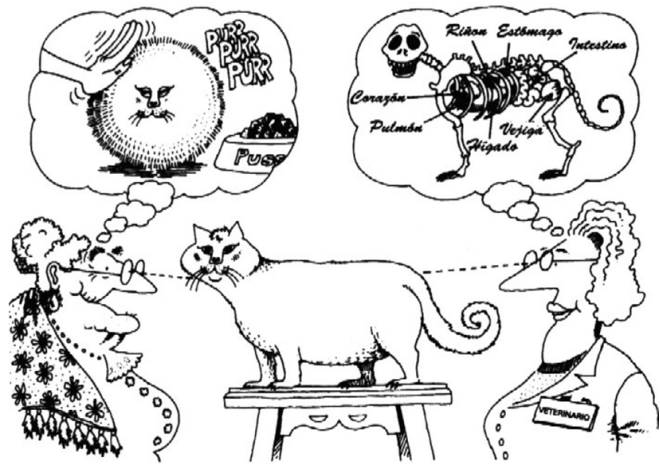
Nótese que en la “Abstracción” del perro quitamos varias Propiedades y acciones.

Abstracción de un “Perro”

17

# Abstracción

La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.



18



## Abstracción

Las clases y objetos deben estar al nivel de abstracción adecuado: ni demasiado alto ni demasiado bajo.



19

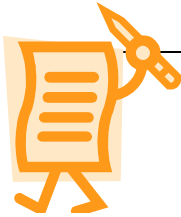
## Clases y Objetos

- ❑ "FIDO" es UN "PERRO"
- ❑ "FIDO" es del TIPO "PERRO"
- ❑ "FIDO" es un OBJETO
- ❑ "PERRO" es la CLASE de "FIDO"
  
- ❑ "CHESTER" es OTRO "PERRO"
- ❑ "CHESTER" también es del TIPO "PERRO"
- ❑ "CHESTER" es otro OBJETO
- ❑ "PERRO" también es la clase de "CHESTER"



20


## Clase



- ❑ Es una descripción de las características y acciones para un tipo de objetos.
- ❑ Una clase NO es un objeto. Es solo una plantilla, plano o definición para crear objetos.

21

## Clase



- ❑ Contiene todas las características comunes de ese conjunto de objetos
- ❑ Clase = Modelo = Plantilla = Esquema = Descripción de la anatomía de los objetos.
- ❑ A partir de una clase se pueden crear muchos objetos independientes con las mismas características.

22

## Objeto

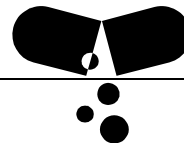


- ❑ Unidad que combina datos y funciones.
  - Datos = Propiedades = Atributos = Características
  - Funciones = Métodos = Procedimientos = Acciones
- ❑ Un objeto es creado a partir de una clase.
- ❑ Los datos y funciones están **Encapsulados**.
- ❑ Posee un nombre único (identificador).
- ❑ Un objeto es del tipo de una clase
- ❑ “Un objeto es la instancia de una clase”
- ❑ Un objeto es un ejemplar específico creado con la estructura de una clase.

23



## Encapsulamiento

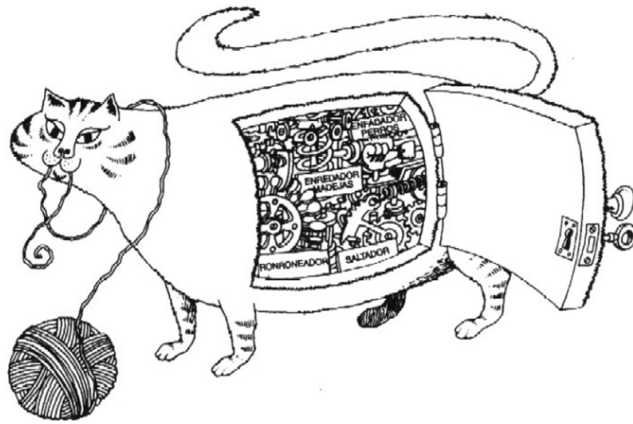


- ❑ Permite incluir en una sola entidad información y operaciones que controlan dicha información.
- ❑ Permite:
  - Componentes públicos [Accesibles, Visibles].
  - Componentes privados [No accesibles, Ocultos].
  - Restricción de accesos indebidos.

24

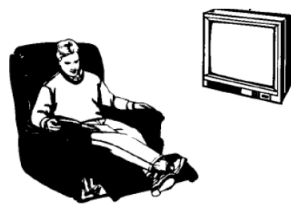
## Encapsulamiento

El encapsulamiento oculta detalles de la implementación de un objeto.



25

## Ejemplo: Encapsulamiento

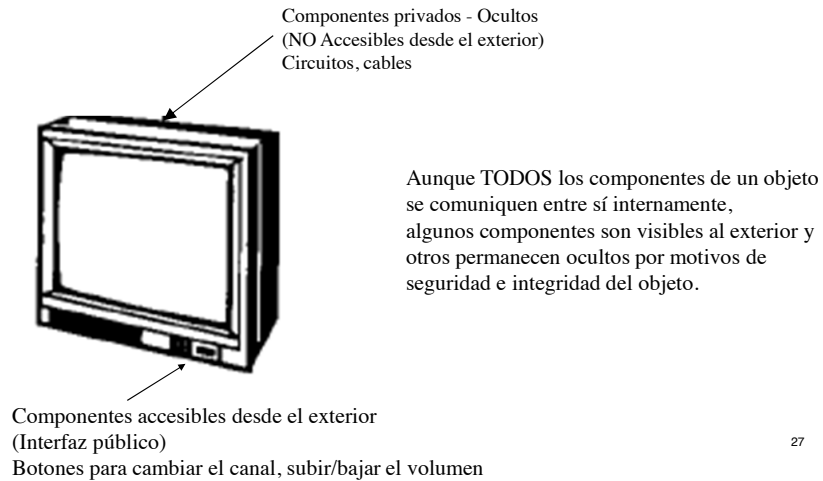


La Televisión oculta algunos componentes y operaciones de la persona que la ve.

- Los objetos encapsulan lo que hacen. Ocultan la funcionalidad interna de sus operaciones, de otros objetos y del mundo exterior.

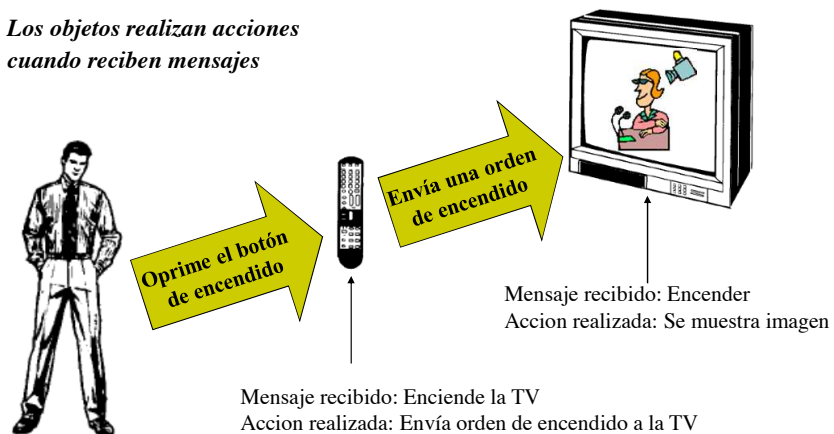
26

## Ejemplo Encapsulamiento



## Mensajes entre Objetos

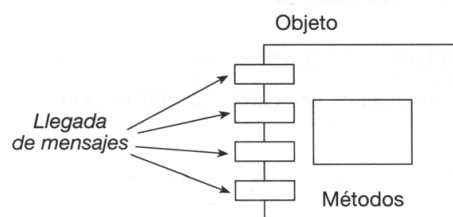
*Los objetos realizan acciones cuando reciben mensajes*



## Mensajes: Comunicación entre objetos



- ❑ **Mensaje.-** Orden que se envía al objeto para indicarle realizar una acción.
- ❑ **Mensaje.-** Llamada a un método (o función) del objeto.



Al conjunto de mensajes a los cuales puede responder un objeto se llama **“Protocolo del Objeto”**

29

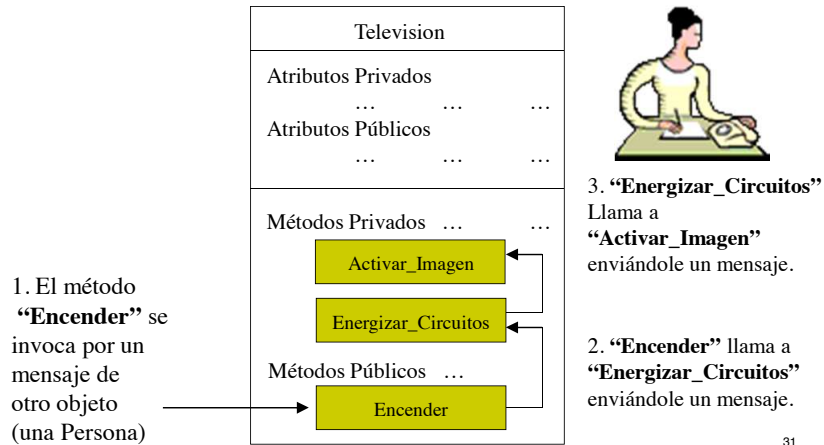


## Anatomía de un mensaje

- Identidad del receptor
  - Método que ha de ejecutar
  - Información especial (argumentos o parámetros)
- 
- ❑ Ejemplos:
    - miTelevision.Encender( )
    - miTelevision.Apagar( )
    - miTelevision.CambiarCanal( 45 )
    - miPerro.Comer(“Croquetas”)
    - miEmpleado.Contratar ( “Juan”, 3500)
    - miFactura.Imprimir( )

30

## Ejemplo de envío de mensajes



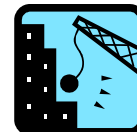
## Constructores y Destruidores

- Los objetos ocupan espacio en memoria; existen en el tiempo y deben crearse [instanciarse] y destruirse:

- **Constructor.-** Operación que crea un objeto y/o inicializa su estado.



- **Destructor.-** Operación que libera el estado de un objeto y/o destruye el propio objeto.

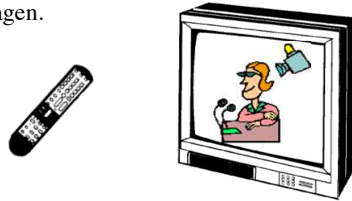


32

## Ejemplo de constructor y destructor

Cada vez que se **enciende** la Television...

- Se deben energizar los circuitos
- Se debe activar el cinescopio
- ...Para posteriormente mostrar la imagen.

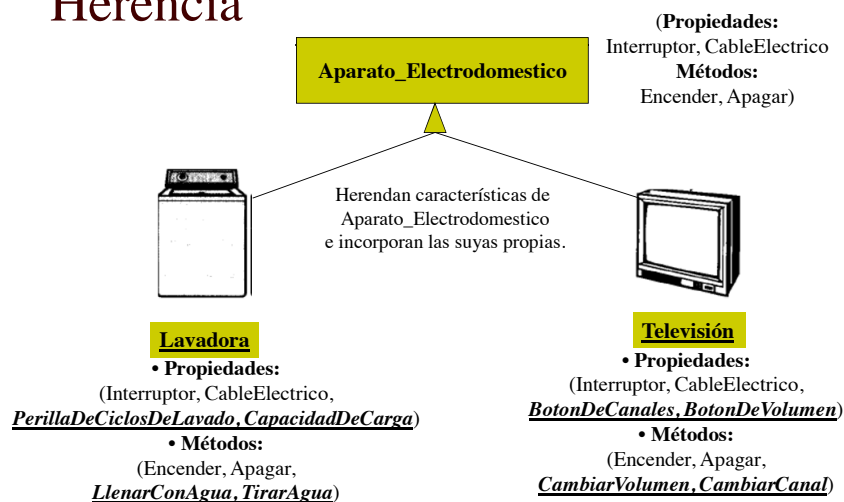


Cada vez que se **apaga** la Television...

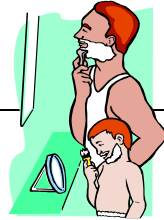
- Se deben des-energizar los circuitos
- Se debe des-activar el cinescopio
- ...Para posteriormente apagar la imagen

33

## Herencia







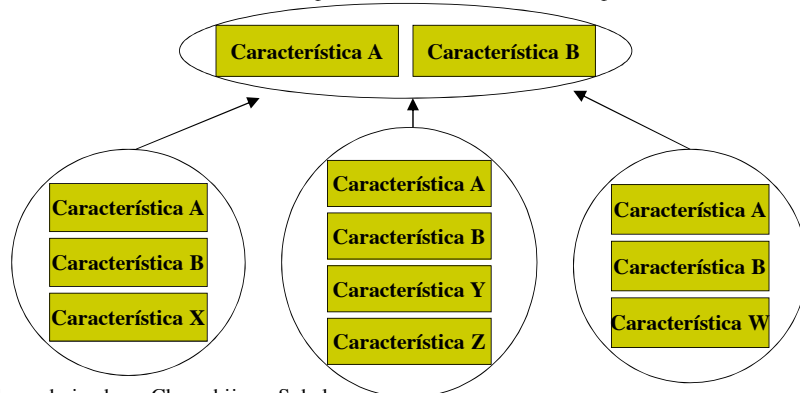
## Herencia

- ❑ Capacidad para utilizar características previstas en antepasados o ascendientes.
- ❑ Permite construir nuevas clases a partir de otras ya existentes, permitiendo que éstas les “transmitan” sus propiedades.
- ❑ Objetivo: Reutilización de código.

35

## Herencia - Jerarquía de clases

Clase Base = Super clase = Clase madre = Clase padre

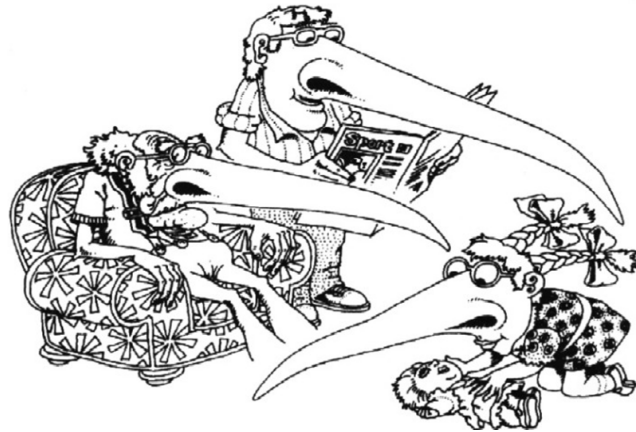


Clases derivadas = Clases hijas = Subclases

36

## Herencia

Una subclase hereda el comportamiento y la estructura de su Super Clase

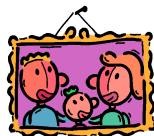


37

## Tipos de Herencia



- **Herencia Simple.-** Una clase puede tener sólo un ascendiente. [Una subclase puede heredar de una única clase].



- **Herencia múltiple (en malla).-** Una clase puede tener más de un ascendiente. [Heredar de más de una clase].

38

# Herencia simple

Artículo

Video

Audio

Altavoces


Radio

Casete

CD

Amplificador

Ejemplo 1



Ejemplo 2

Figura


Círculo

Rectángulo

Triángulo

Rectángulo redondeado

39



# Herencia múltiple

A

B

C

D

E

Ejemplo 1

Persona

Profesor

Investigador

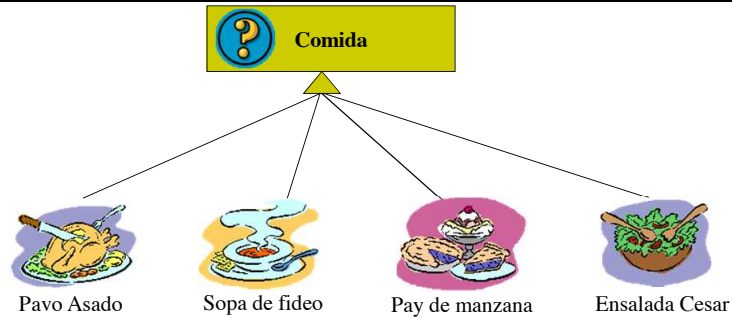
Profesor universitario

Ejemplo 2

40

20

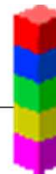
## Clase abstracta



La “Comida” como tal, es solo un concepto abstracto que NO puede instanciarse.  
Existen muchos alimentos que heredan sus características y  
ellos SI pueden existir por sí mismos.  
“Comida” es una clase Abstracta.

41

## Clase abstracta



- ❑ Es una clase que sirve como clase base común, pero NO puede tener instancias.
- ❑ Una clase abstracta solo puede servir como clase base (solo se puede heredar de ella).
- ❑ Sus clases “hijas” SI pueden tener instancias.

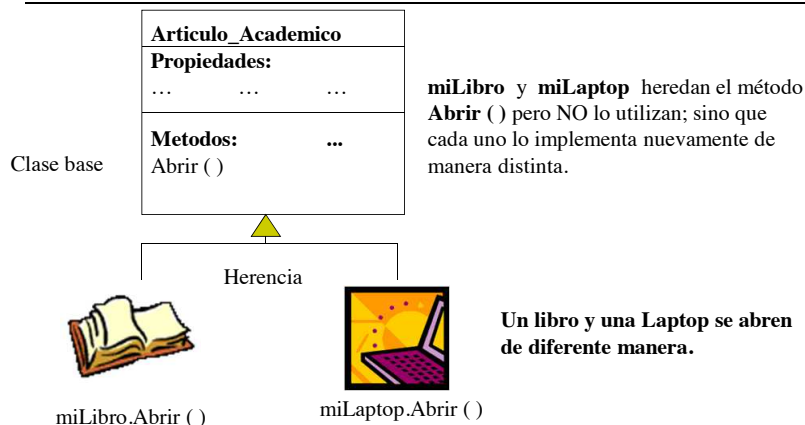
42

## Anulación / Sustitución / Sobreescritura [ Overriding ]

- ❑ Sucede cuando una clase “B” hereda características de una clase “A”, pero la clase “B” re-define las características heredadas de “A”.
- ❑ Propiedades y métodos pueden heredarse de una superclase. Si estas propiedades y métodos son re-definidos en la clase derivada, se dice que han sido “Sobreescritos”.

43

## Anulación / Sustitución / Sobreescritura [ Overriding ]



44

## Sobrecarga [ Overload ]

- ❑ La sobrecarga representa diferentes maneras de realizar una misma acción.
- ❑ En los programas se usa el mismo nombre en diferentes métodos con diferentes firmas [número, orden y tipo de los parámetros].
- ❑ El código de programación asociado a cada sobrecarga puede variar.
- ❑ Ejemplos:
  - `miEmpleado.Contratar("Juan", "Ventas", 2500)`
  - `miEmpleado.Contratar("Juan")`
  - `miEmpleado.Contratar("Juan", 2500)`

45

## Ejemplo de Sobrecarga [ Overload ]



`miPuerta.Abrir ( Adentro, Afuera)`



`miPuerta.Abrir ( Afuera, Adentro)`



`miPuerta.Abrir ( )`

46



## Polimorfismo

Se refiere a:

1. Es el uso de un mismo nombre para representar o significar más de una acción.
  - La sobrecarga es un tipo de Polimorfismo.
2. Que un mismo mensaje pueda producir acciones totalmente diferentes cuando se recibe por objetos diferentes del mismo tipo.
  - Un usuario puede enviar un mensaje genérico y dejar los detalles de la implementación exacta para el objeto que recibe el mensaje en tiempo de ejecución.
  - Para este caso, se utiliza herencia y sobreescritura (Override).

47

## Polimorfismo

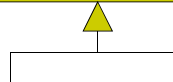


POLI = Múltiples MORFISMO = Formas



miRefrigerador.Abrir( "Puerta de Abajo" )  
miRefrigerador.Abrir( "Puerta de Arriba" , "Mitad" )

ObjetoEnFormaDeCaja



miRegalo.Abrir( )



miCofre.Abrir( )

48

## Bibliografía recomendada

- **Fundamentos de programación: Algoritmos, Estructuras de datos y Objetos.**  
Luis Joyanes Aguilar  
Mc Graw Hill  
Tercera Edición  
ISBN: 84-481-3664-0
- **Teach yourself Object Oriented programming in 21 days**  
2nd edition Ed. SAMS, 2002. ISBN: 0-672-32109-2
- **OOP demystified: A self-teaching guide**  
Jim Keogh, Mario Gianni  
Ed. McGrawHill, 2004.  
ISBN: 0-07-225363-0
- **Modelado y Diseño Orientados a Objetos**  
(Metodología OMT)  
James Rumbough, Michael Blaha, et. al.  
Editorial Prentice Hall  
ISBN 0-13-24-0698-5