Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

•••

Karla Ramírez Pulido

Orientación a Objetos

Paradigma Orientado a Objetos (P.O.O.)

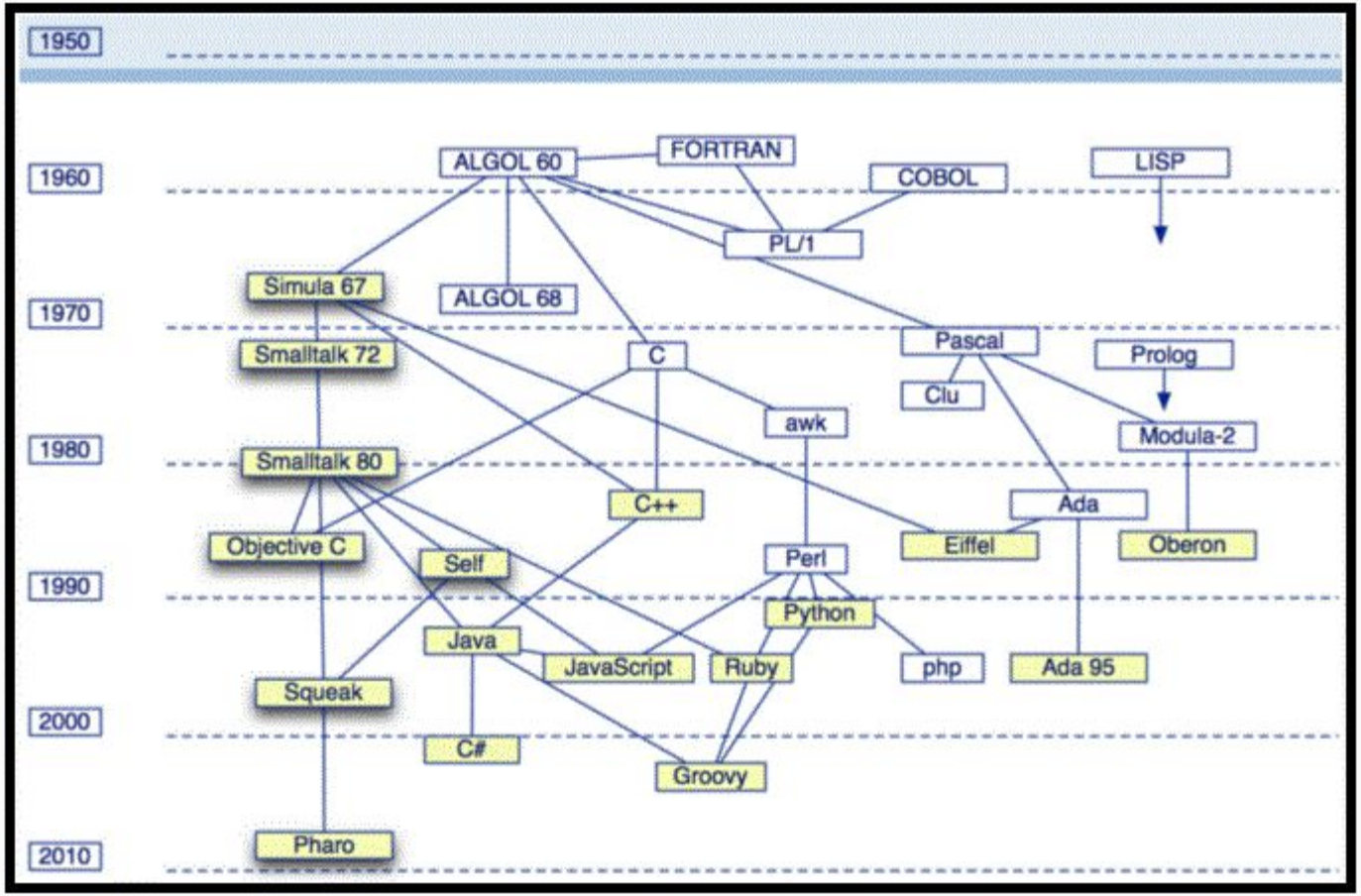
Modela usando:

- **Clases**
- **Métodos**
- **Objetos**
- **Polimorfismo**
- **Herencia**
- **Sobrecarga de operadores**

Un poco de historia

Nace en los años 60 en el Norwegian Computing Center con la implementación de dos lenguajes de programación de simulación:

Simula 1 y *Simula 67*


1950
1960
1970
1980
1990
2000
2010
ALGOL 60
FORTRAN
COBOL
LISP
PL/1
Simula 67
ALGOL 68
Smalltalk 72
C
Pascal
Prolog
Clu
awk
Modula-2
Smalltalk 80
C++
Ada
Objective C
Self
Perl
Eiffel
Oberon
Python
Java
JavaScript
Ruby
php
Ada 95
Squeak
C#
Groovy
Pharo

Características

- Una buena abstracción de las clases, objetos y atributos nos brinda una implementación más detallada, puntual y coherente.
- **Reutilización de código:** cuando se han diseñado adecuadamente las clases, se pueden usar en distintas partes del programa y en numerosos proyectos (*herencia*), lo cual permite:
  - Rápido desarrollo.
  - Alta calidad del código.
  - Bajo costo en fases de desarrollo.

Características

- **Fácil adaptación:** la facilidad de añadir o suprimir nuevos objetos nos permite hacer modificaciones de una forma muy sencilla. Se trabaja bajo la idea de “divide y vencerás”.
- **Encapsulamiento:** nos permite proteger la integridad de los datos.
- **Fiabilidad:** al dividir el problema en partes más pequeñas podemos probarlas de manera independiente y aislar mucho más fácilmente los posibles errores que puedan surgir.
- **Bajo acoplamiento y alta cohesión:** las clases son independientes entre sí, y hacen las tareas específicas para las cuales fueron diseñadas.

Características

- Sencillo mantenimiento de software.
- Amplia documentación.
- Modularidad.
- Fácil depuración de código.
- Los lenguajes en este paradigma muchas veces presentan depuradores, IDEs, marcos de trabajo (*frameworks*), bibliotecas, etc, lo cual facilita el desarrollo de programas en estos lenguajes.

Objeto

Unidad que combina datos y funciones.

1. **Datos = Atributos = Características**
2. **Métodos = 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.

Objetos tienen:

Características que los distinguen entre sí.

Ejemplo: Perro

- Características
  - Nombre: Cheiser
  - Raza: Pug
  - Color: Cervato
  - ...
- Acciones asociadas a ellos.
  - Volumen de ladrido: bajo
  - Comer (# al día): 2
  - Dormir (hrs al día): 16
  - ...

Usando la terminología del P.O.O.

Características: ATRIBUTOS o datos

Acciones: MÉTODOS u operaciones

OTRO EJEMPLO:

Modelar un coche

¿En qué coche pensaron?

Coches

Seguramente pensaron en

alguno de éstos:

Ejemplo:

Atributos:

- **Marca:**
- **Nombre:**
- **Modelo:**
- **No. de cilindros:**
- **...**

Métodos:

- Encender:
- Reversa:
- Avanzar:
- Frenar:
- **...**

Modificadores de acceso

- **Públicos:** El modificador de acceso público denota campos y métodos que son de libre acceso desde cualquier otra parte de un programa.
- **Privados:** Especifica campos y métodos de una clase que no son accesible fuera de la unidad donde se declara la clase.
- **Protegidos:** El modificador de acceso protegido se utiliza para indicar métodos y campos con visibilidad sólo en la clase actual y sus clases derivadas (o subclases).

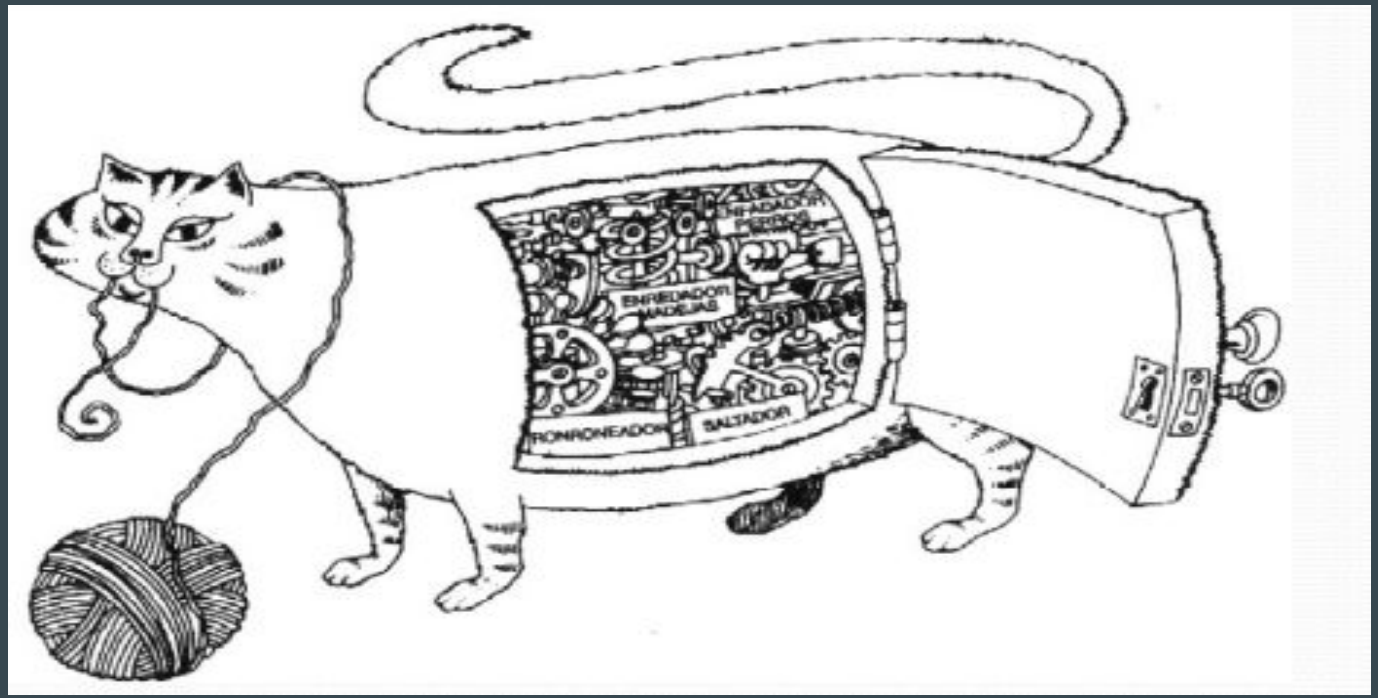
Clases

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

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

Encapsulamiento

El **encapsulamiento** oculta detalles de la implementación de un objeto.



Mensajes entre objetos

Mensajes:

1. Los objetos reciben acciones cuando reciben mensajes.
2. Llamada a un método del objeto.

Anatomía de un mensaje

- Identidad del receptor
- Método que ha de ejecutar
- Información especial (argumentos o parámetros)

Ejemplos:

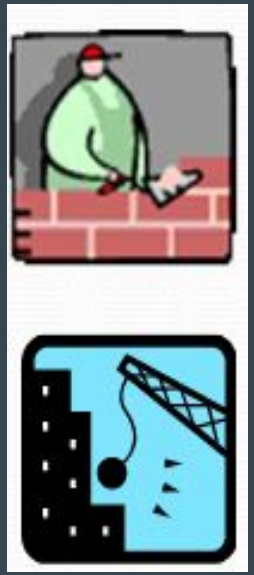
- miTelevision.Encender()
- miTelevision.Apagar()
- miPerro.Comer(“Croquetas”)

Constructores y Destructores

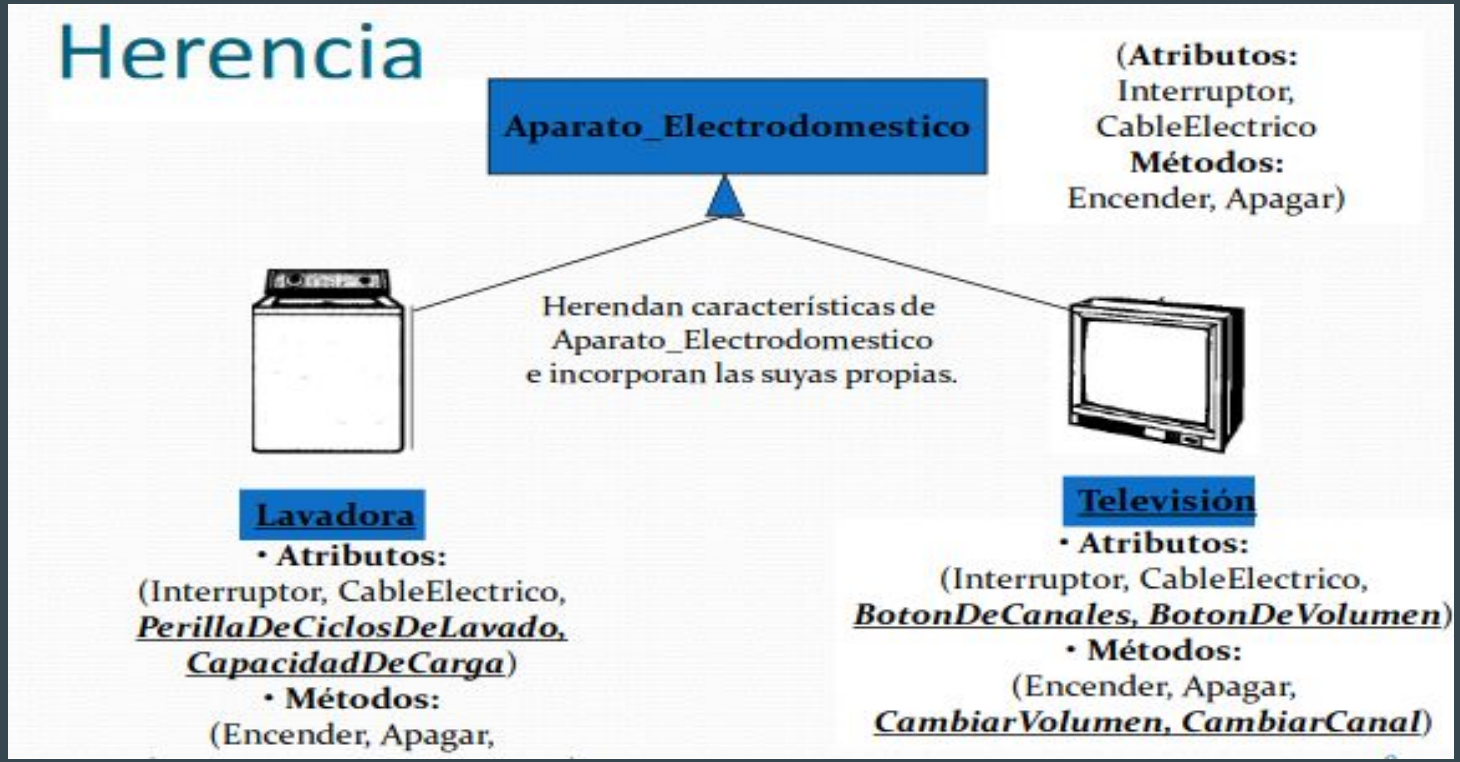
Los objetos ocupan espacio en memoria; existen en el tiempo y deben crearse [instanciarse] y destruirse:

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

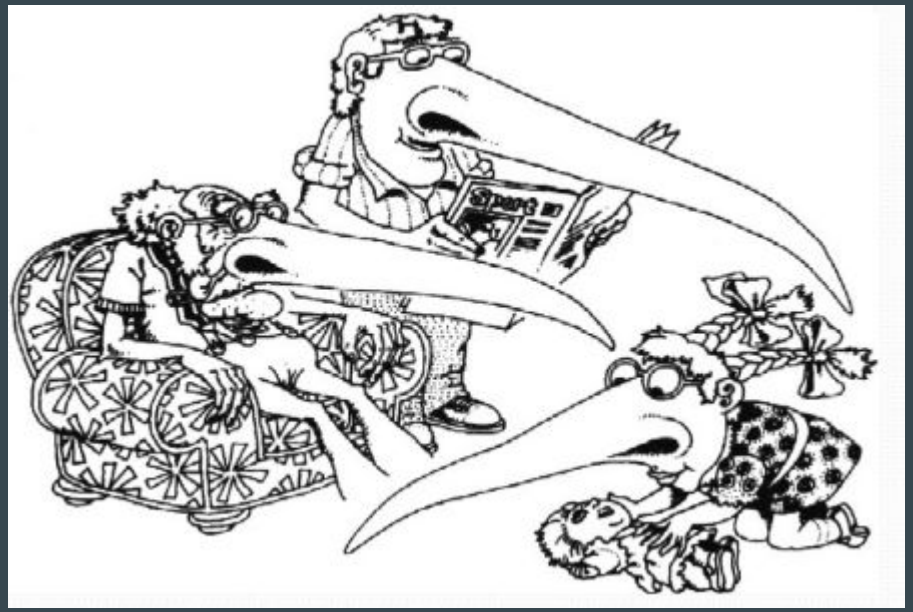


Herencia


Herencia
Aparato_Electrodomestico
(Atributos:
Interruptor,
CableElectrico
Métodos:
Encender, Apagar)
Herendan características de
Aparato_Electrodomestico
e incorporan las suyas propias.
Lavadora
• Atributos:
(Interruptor, CableElectrico,
PerillaDeCiclosDeLavado,
CapacidadDeCarga)
• Métodos:
(Encender, Apagar,
Televisión
• Atributos:
(Interruptor, CableElectrico,
BotonDeCanales, BotonDeVolumen)
• Métodos:
(Encender, Apagar,
CambiarVolumen, CambiarCanal)

Herencia: super clases y sub clases

Una subclase hereda el comportamiento y la estructura de su super clase.



Tipos de herencia

Herencia simple

Una clase puede tener sólo un ascendiente.

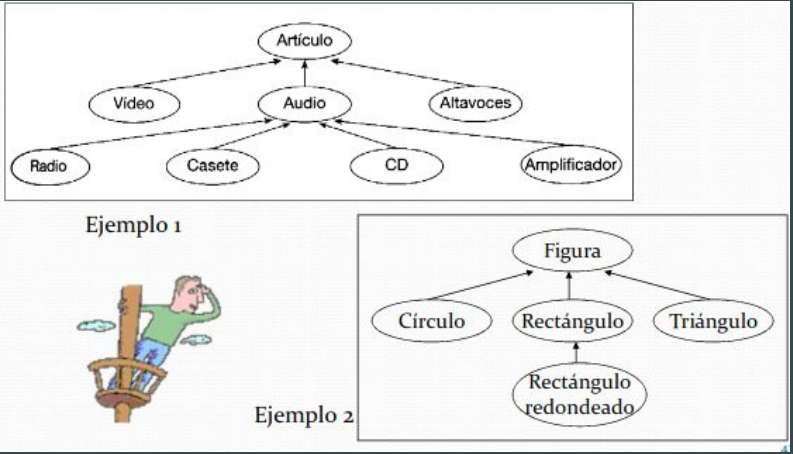
Una subclase puede heredar de una única clase.

Herencia múltiple

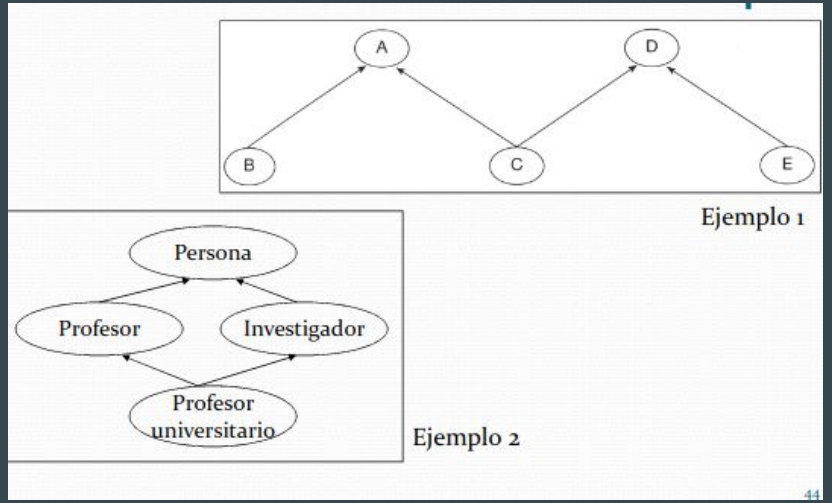
Una clase puede tener más de un ascendiente inmediato.

Heredar de más de una clase.

Herencia Simple

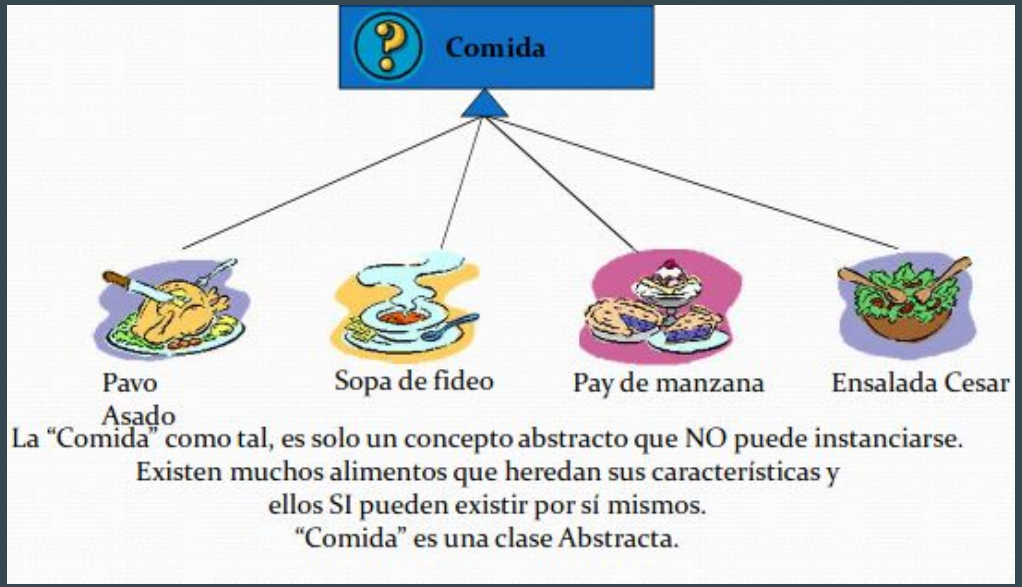


Herencia múltiple



Clase Abstracta

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



Dudas que surgieron en clase:

1. ¿Cómo son los modificadores de acceso que están en un paquete en JAVA?

R:= En Java cuando no pones el tipo del modificador, por defecto ese paquete es privado (package-private)

1. Otro ejemplo de clase abstracta sería:

R:= Una clase “molde” y la instancia podría ser una “galleta”. Recuerden que las clases abstractas NO se instancian.

Diferencia entre clase abstracta e interfaz en JAVA.

R:= Las interfaces modelan comportamiento que puede caracterizar clases no relacionadas, y una clase abstracta es una abstracción de un conjunto de clases que se derivan de esta. Y la clase abstracta NO se instancía.

Además la interfaz no debes implementar nada, además solo puede extender de otras interfaces, no heredar

Diferencias entre clase abstracta e interfaz

Una *interfaz* puede parecer similar a una *clase abstracta*, pero existen una serie de diferencias entre una interfaz y una clase abstracta:

- **Todos** los métodos de una interfaz se declaran implícitamente como abstractos y públicos.
- Una interfaz no declara variables de instancia.
-

- Una clase abstracta no puede *implementar* los métodos declarados como abstractos, una interfaz no puede *implementar* ningún método (ya que todos son abstractos).
- Una clase puede implementar varias interfaces, pero sólo puede tener una clase ascendiente directa.

Diferencias entre clase abstracta e interfaz

- Una clase abstracta pertenece a una jerarquía de clases mientras que una interfaz **no** pertenece a una jerarquía de clases. En consecuencia, clases sin relación de herencia pueden implementar la misma interfaz.

Bibliografía

- http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/POO/Apuntes/01.-%20Introduccion%20a%20la%20POO.pdf
- https://ferestrepoca.github.io/paradigmas-de-programacion/poo/poo_teoria/concepts.html
- https://www.javatpoint.com/access-modifiers
- https://www.arkaitzgarro.com/java/capitulo-18.html

¿Dudas?

Avisos para terminar el semestre

Gracias