UNIVERSIDAD DE LOS LAGOS

DIPLOMADO
LENGUAJE DE PROGRAMACIÓN C#

MÓDULO 1 - ARQUITECTURA
CLASE 2 - PROGRAMACIÓN ORIENTADA A OBJETOS

JOEL TORRES CARRASCO
CRISTHIAN AGUILERA CARRASCO
CRISTIAN VALLEJOS VEGA

DEPARTAMENTO DE CIENCIAS DE LA INGENIERÍA
INGENIERÍA CIVIL EN INFORMÁTICA

Campus Osorno
Av. Fuchslocher 1305
Teléfono +56 64 2333 000
Fax +56 64 2333 774
Osorno, Chile

Campus Puerto Montt
Camino a Chinquihue Km 6
Teléfono +56 65 2322 536
Puerto Montt, Chile

Sede Santiago
República 517
Barrio Universitario
Teléfono +56 02 2675 3057
Santiago, Chile

Sede Chiloé
Ubaldo Mansilla Barrientos 131
Teléfono 56 65 2322 409
Castro, Chile
Eleuterio Ramírez 348
Teléfono +56 65 2322 476
Ancud, Chile

UNIVERSIDAD ACREDITADA
Septiembre de 2021 - Septiembre de 2026
Gestión Institucional · Docencia de Pregrado
Investigación · Vinculación con el Medio
AVANZADA

www.ulagos.cl

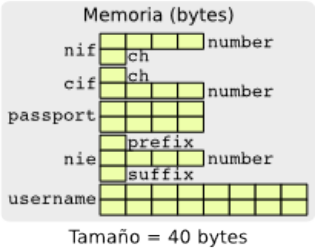
Tabla de Contenido

Sección Siguiente

El paradigma Estructurado y sus limitantes

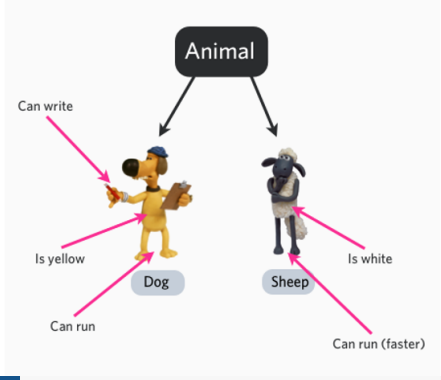
- Las interacciones entre las variables de estructuras son complejas
- No se modela el comportamiento de las estructuras
- El acceso a los datos es libre y modificable
- Problemas de escalabilidad y Rigidez + programas menos legibles

```
struct user_id {
    struct NIF {
        unsigned int number;        -> Definición
        char ch;
    } nif;                          -> Declaración
    struct CIF {
        char ch;                    -> Definición
        unsigned int number;
    } cif;                          -> Declaración
    char passport[8];
    struct NIE {
        char prefix;                -> Definición
        unsigned int number;
        char suffix;
    } nie;                          -> Declaración
    char username[16];
};
```



PARADIGMA DE ORIENTACIÓN A OBJETOS

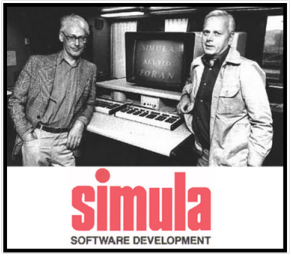
- Define los programas en términos de comunidades de objetos.
- Los objetos con características comunes conformas Clases
- Las Clases modelan las características y el comportamiento de los objetos.
- Los objetos se comunican entre ellos para realizar una tarea.
- Mucho más fácil de escribir, mantener y reutilizar.



Sección Siguiente

HISTORIA Y EVOLUCIÓN

- Nace en los 60's en Norwegian Computing Center por Kristen Nygaard y Ole Johan Dahl al crear el lenguaje SIMULA.



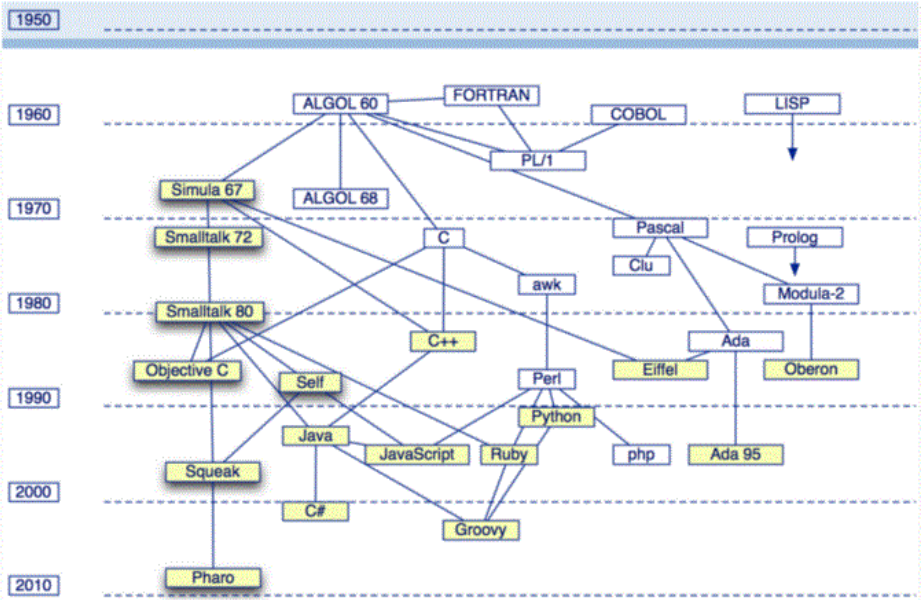

```
! todo programa empieza con un begin y termina con un end ;
Begin

  Class Saludos;
  Begin
    OutText("¡Hola Mundo!");
    OutImage;
  End of class saludos;

  REF(Saludos) objeto;
  objeto :- New Saludos;

End of module program;
```

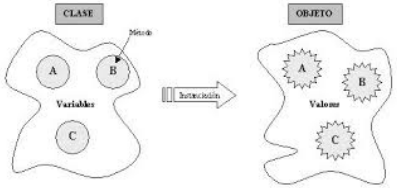
HISTORIA Y EVOLUCIÓN

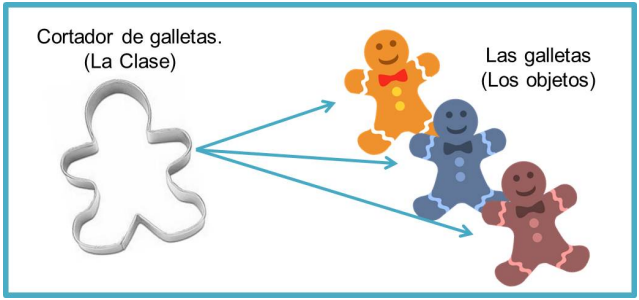


Sección Siguiente

IMPLEMENTACIÓN: CLASES VS OBJETOS

- Una **Clase** es una plantilla que modela la información (atributos) y el comportamiento (métodos)
- Un **Objeto** es una instancia de una clase con valores propios y sus acciones contribuyen a cumplir con una tarea.





Sección Siguiente

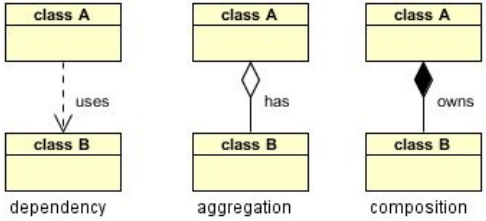
¿Qué es UML?

Diagramas UML:

- Diagrama de Clase
- Diagrama de Componentes
- Diagrama de Estados
- Diagrama de Casos de Uso
- Diagrama de Secuencia
- Diagrama de Actividad
- Diagrama de Despliegue
- Diagrama de Colaboración
- Diagrama de Objetos
- Diagrama de Interacción
- Diagrama de Estructura Compuesta
- Diagrama de Paquetes
- Diagrama de Comunicación
- Diagrama de Tiempo

DIAGRAMA DE CLASES

- Se definen **Clases**:
  - Atributos
  - Métodos
  - Visibilidad (-,+,#,~)
  - <<*Abstractas*>>
- Relaciones:
  - Herencia
  - Agregación
  - Composición
  - Asociación
  - Dependencia de Uso
- Multiplicidad
  (0..1, n, 0..*, 1..*, m..n)



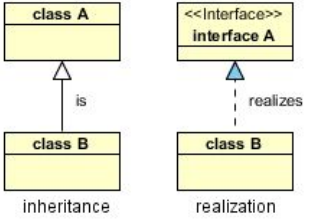


DIAGRAMA DE CLASES

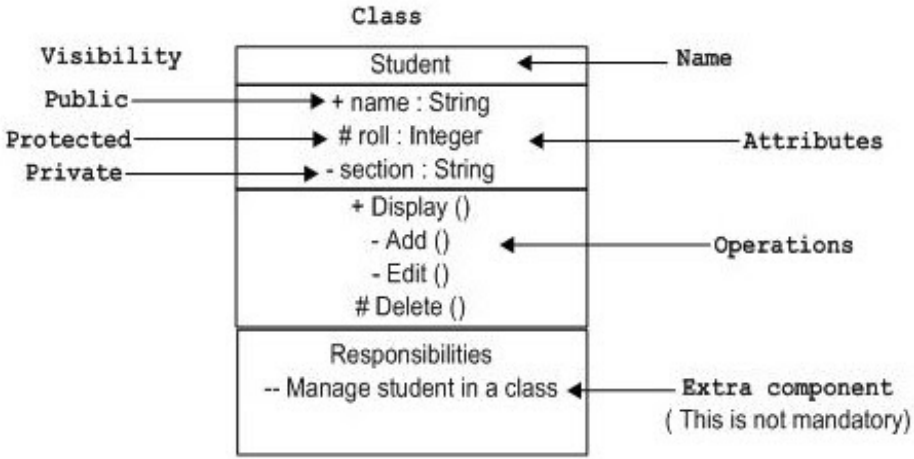


DIAGRAMA DE CLASES

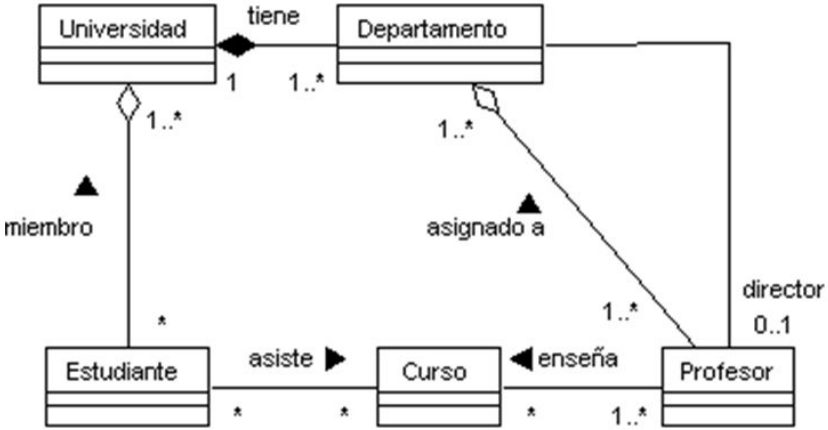
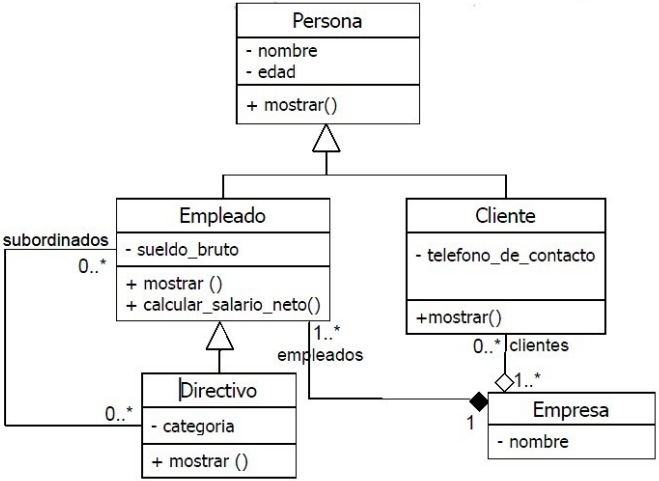
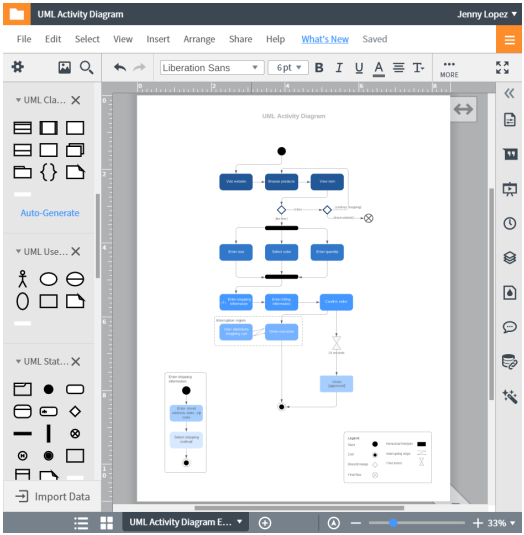


DIAGRAMA DE CLASES



HERRAMIENTAS DE DESARROLLO UML

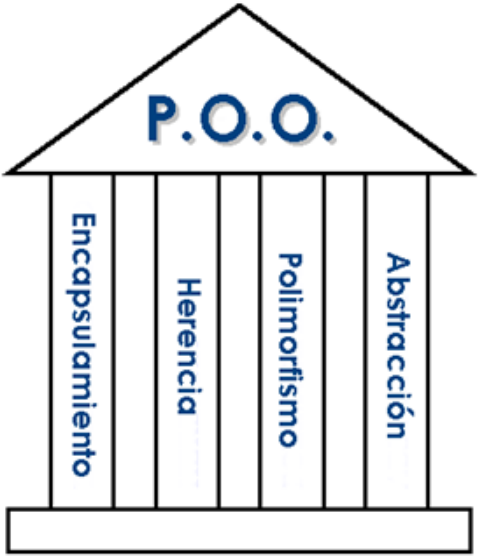
- Gliffy
- ArgoUML
- MagicDraw
- Lucidchart
- Visual-Paradigm
- Diagrams.net (Draw.io)
- StarUML
- IBM Rational Rhapsody
- MS Visio
- UMLEtino



Sección Siguiente

PILARES POO

- **Abstracción**:
  Separa la vista del objeto
  a su implementación
- **Encapsulamiento**:
  Asegura el contenido de
  la información
- **Herencia**:
  Ordenar y clasificar las clases
  en Jerarquías
- **Polimorfismo**:
  Implementaciones diferentes para un mismo
  comportamiento

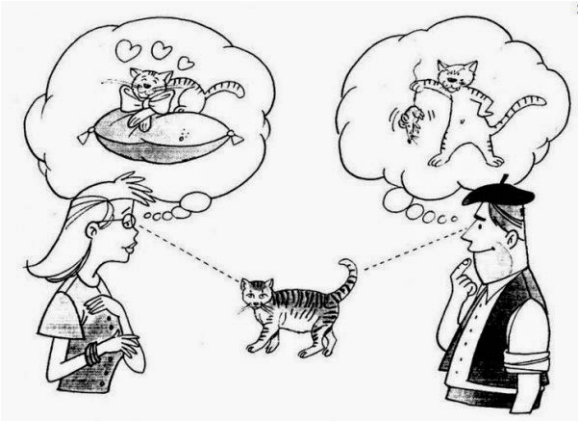


Sección Siguiente

ABSTRACCIÓN

La abstracción consiste en **AISLAR** al objeto de un contexto.

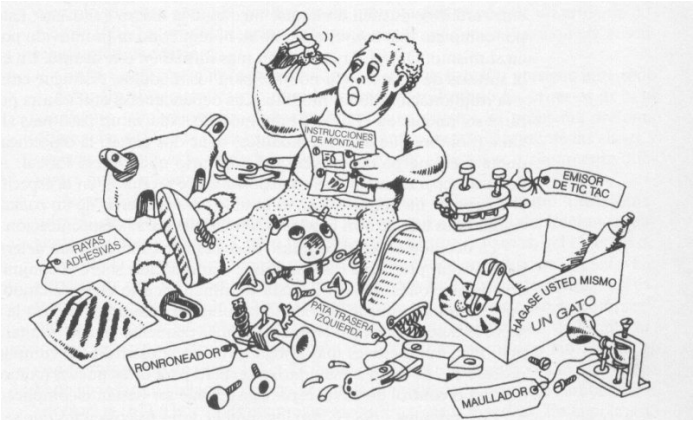
- Concepto de CAJA NEGRA
- No importa cómo está implementado, sino para qué funciona
- La implementación debe proporcionar todas las características y comportamientos necesarios
- Un programador externo puede ver la clase y utilizarla



ABSTRACCIÓN

Se debe **modularizar** los elementos que componen la clase

- Debe especificar cada atributo
- Debe especificar cada Método



Gato
atributo1 atributo2 ...
método1() método2() ...

ABSTRACCIÓN

Un mayor nivel de abstracción permitirá una mejor implementación, más escalabilidad, flexibilidad y mantenimiento.

Gato
name: String sex: String age: int weight: int color: String texture: String
Gato() eat(): void move(): void meow(): void purr(): void hunt_mice(): mouse

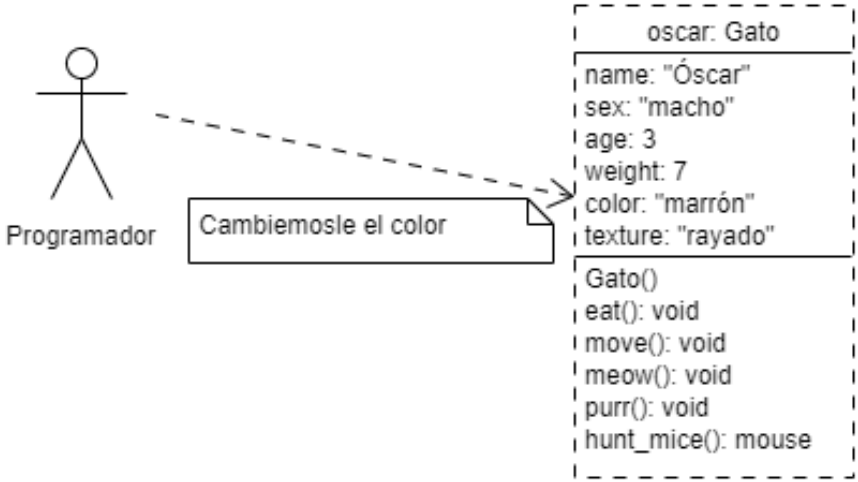
Óscar: Cat

```
name    = "Óscar"
sex     = "macho"
age     = 3
weight  = 7
color   = marrón
texture = rayada
```

Luna: Cat

```
name    = "Luna"
sex     = "hembra"
age     = 2
weight  = 5
color   = gris
texture = lisa
```

Caso Crítico



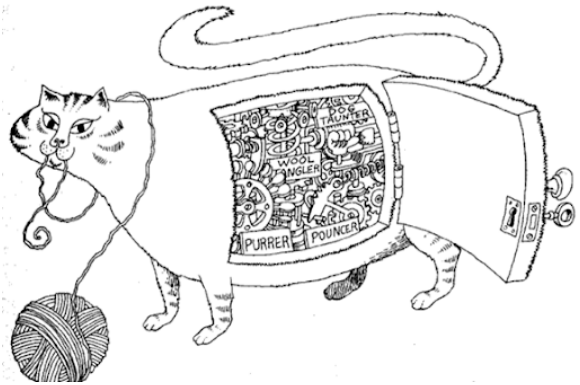
¿Es posible cambiarle el color sin problemas?
¿Debería permitirlo?

Sección Siguiente

ENCAPSULAMIENTO

El encapsulamiento consiste en **PROTEGER** la información y acciones del objeto

- Refuerza el concepto de CAJA NEGRA en la clase
- Cada variable y método tiene una **VISIBILIDAD** asociada
- Impide que un programador externo acceda a los datos o acciones del objeto
- Permite acceso restringido y supervisado de acuerdo a lo establecido en la clase



ENCAPSULAMIENTO

La **VISIBILIDAD** permite identificar el nivel de acceso a los datos

- **Public (+)**: Todos tienen acceso a los datos
- **Protected (#)**: Solo en el paquete y sub-clases
- **internal ()**: Solo las clases del mismo paquete
- **Private (-)**: Solamente la misma clase puede modificar los datos

Access Modifiers	Inside Assembly		Outside Assembly	
	With Inheritance	With Type	With Inheritance	With Type
Public	✓	✓	✓	✓
Private	x	x	x	x
Protected	✓	x	✓	x
Internal	✓	✓	x	x
Protected Internal	✓	✓	✓	x

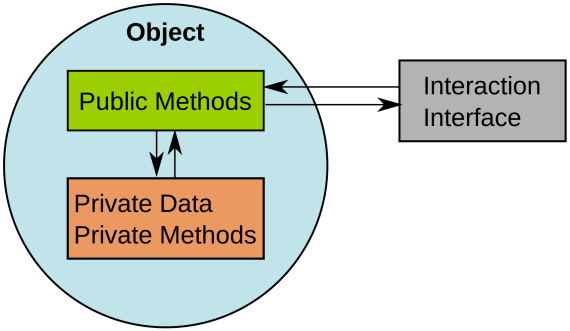
ENCAPSULAMIENTO

Se debe **ENCAPSULAR** los datos críticos a través de los métodos

- Todos los datos importantes debes ser privados
- La única forma de hacer una acción sobre ellos es con los métodos permitidos
- Evita datos erroneos y conflictos

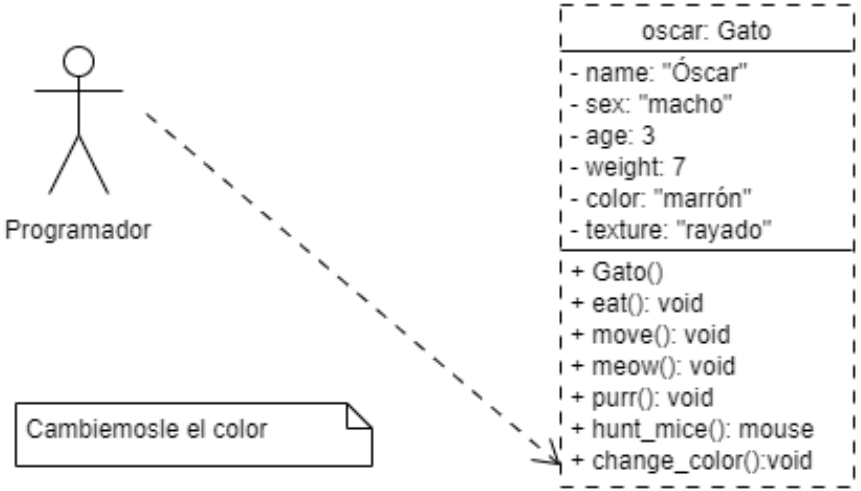
Métodos Getters y Setters:

- **Getters**: son métodos que permiten obtener el valor de un atributo del objeto.
- **Setters**: son métodos que permiten modificar el valor de un atributo del objeto por uno dado.



En C#, se puede modificar a través de las

CASO CRÍTICO



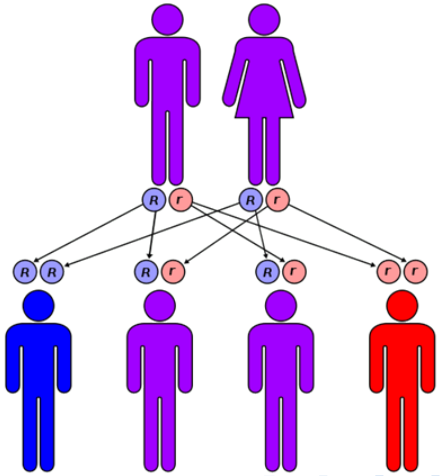
El cambio es admitido por el objeto

Sección Siguiente

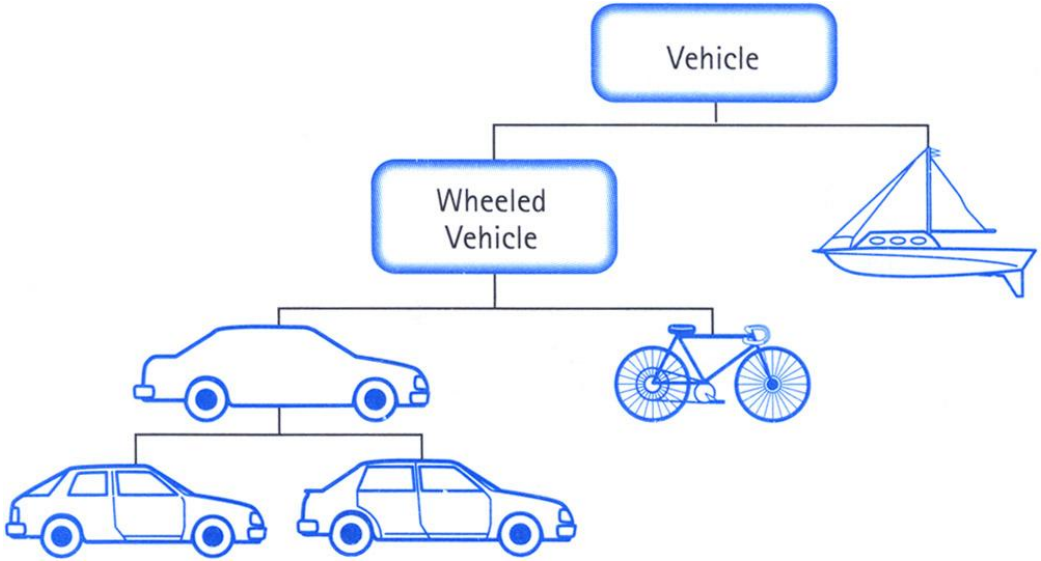
HERENCIA

La Herencia consiste en compartir la estructura y el acceso a los datos de una clase superior

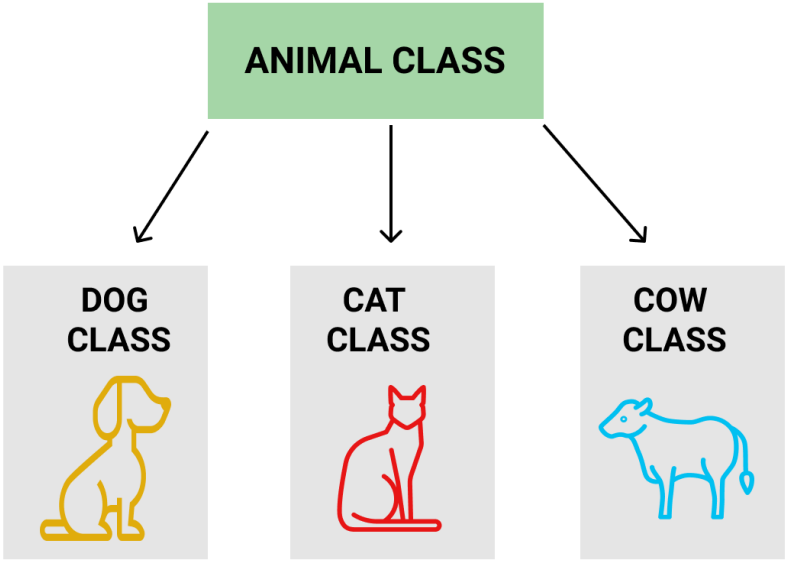
- Una **Super Clase** es la que comparte su estructura
- Una **Sub Clase** es la que hereda la estructura de la Super Clase
- La función **base()** permite acceder al Constructor de la Super Clase
- La clausula **Protected** permite acceder a los datos de la Super Clase



HERENCIA

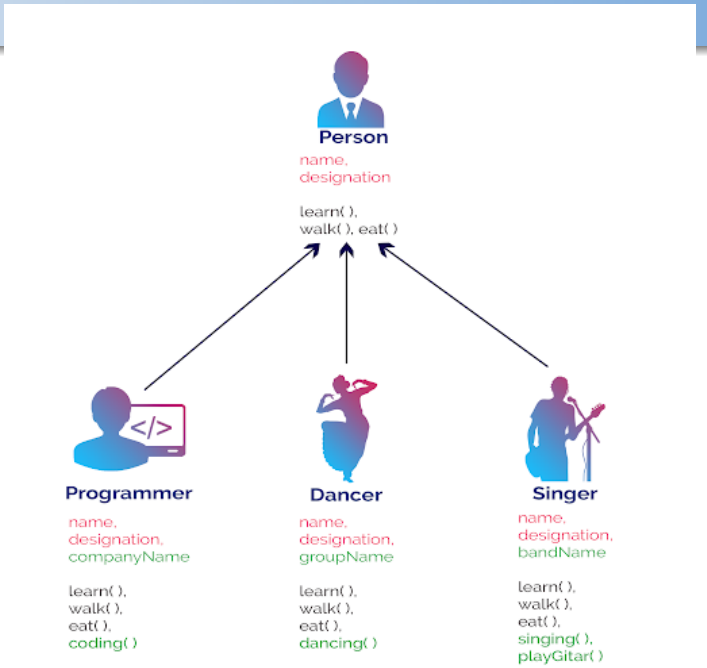


HERENCIA



HERENCIA

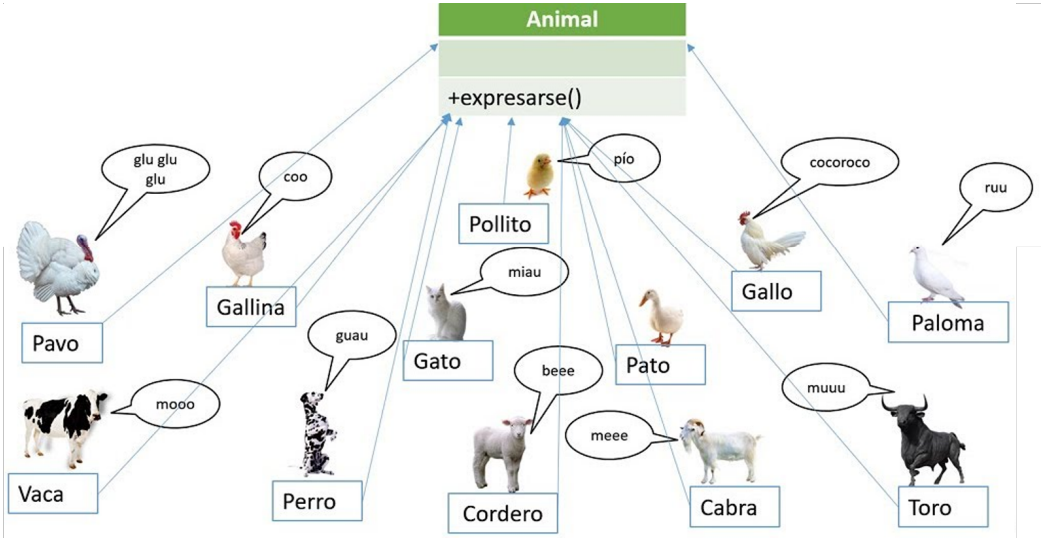
- La clase Person tiene una estructura base
- Las clases Programmer, Dancer y Singer heredan la estructura de Person, pero añade su propia estructura individual



Sección Siguiente

POLIMORFISMO

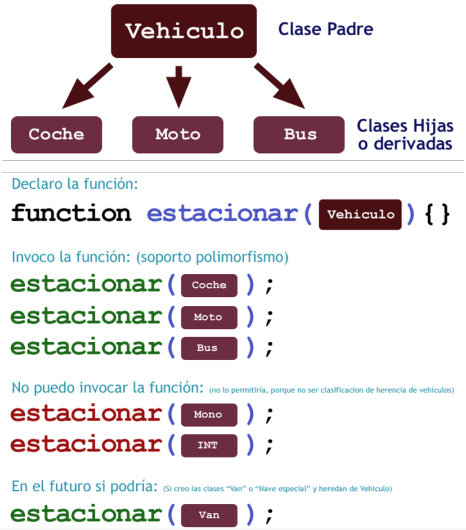
¿Cómo permito que cada subclase ocupe su propia expresión?



POLIMORFISMO

El polimorfismo consiste en modificar la estructura de la Super Clase de acuerdo a la manera que más acomode a la Sub Clase

- **Sobrecarga**: Existen varias métodos con el mismo nombre en clases que son completamente diferentes
- **Paramétrico**: Existen métodos con el mismo nombre pero acepta distintos tipos de parámetros
- **Inclusión**: Se llama al mismo método sin saber en detalle la clase a la que pertenece (@Override)



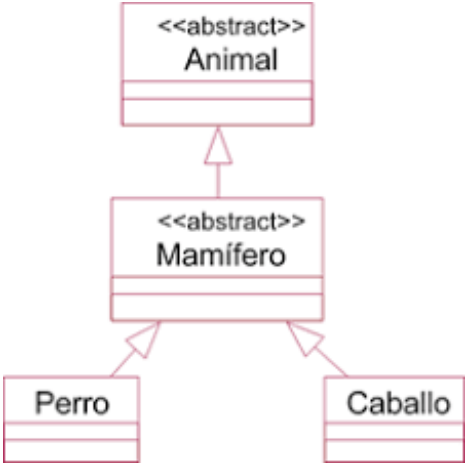
Sección Siguiente

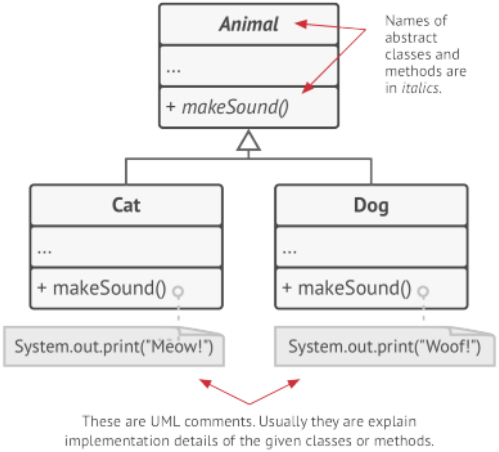
Clases y Métodos Abstractos

Data Abstraction es el proceso de ocultar ciertos detalles y mostrar solamente la información esencial al usuario.

- La abstracción se puede lograr a través de *clases abstractas* y a *interfaces*
- Se puede aplicar en clases y métodos:
  - **Clases Abstractas:** Es una clase restringida que no puede ser usada para crear objetos. Para usarla, es necesario utilizar herencia desde otra clase.
  - **Métodos Abstractos:** Solamente puede ser usados en clases abstractas, y no tienen descripción de código. La implementación de este método lo realizará otra clase a través de herencia.

Clases y Métodos Abstractos





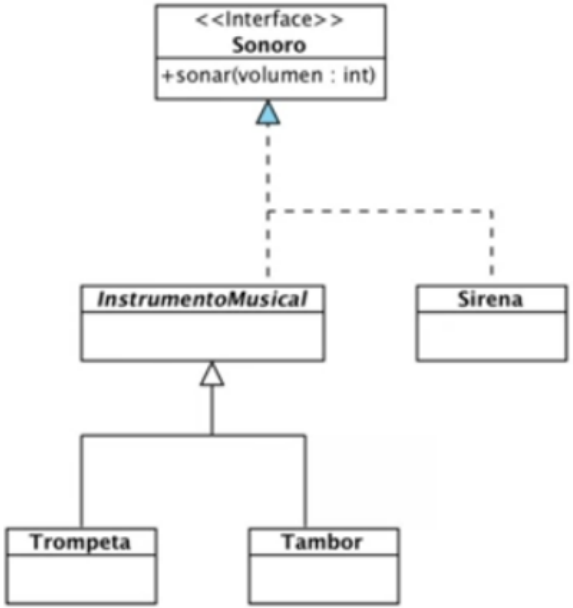
Sección Siguiente

INTERFACES

Otra firma de implementar Abstracción es a través de Interfaces.
Una *Interface* es una clase completamente Abstracta que es usada para agrupar métodos sin descripción.

- Para acceder a la Interface, esta debe ser *implementada*, que es muy similar a la herencia.
- La descripción de los métodos se realizará en las clases que implementa la interface.
- Las interfaces son por defecto Abstractas y Públicas.
- Las interfaces no contienen constructor
- Es una solución para lograr la herencia múltiple.

INTERFACES


<<Interface>>
Sonoro
+sonar(volumen : int)
InstrumentoMusical
Sirena
Trompeta
Tambor

Sección Siguiente

EJERCICIOS

TIPOS DE MONEDA

En una casa de cambio se realizan transacciones para cambiar una divisa extranjera por otra, a cambio de una comisión. Cada divisa tiene su equivalencia con las otras divisas, la que es cambiante por el mercado. Elabore una solución que permita calcular el cambio de divisa con interface divisas. Utilice de referencia la página:
https://www.xe.com/es/currencyconverter/

EJERCICIOS

DATOS PRIVADOS

En una compañía de búsqueda de trabajos, se visualizan las capacidades de cada participante, pero no se debe tomar en cuenta su rol en la compañía, para no crear problemas de vicios en el proceso o filtración de datos. Cree una interface candidato, para enrolar a gerentes, jefes de proyectos y empleados en la búsqueda. El que tenga más capacidades gana el puesto de trabajo.

EJERCICIOS

MULTINACIONAL

En una empresa multinacional, existen muchas tiendas de distintos rubros y distintos software. Sin embargo, es necesario realizar un recuento de los valores activos, pasivos para determinar las ganacias globales de todo el conglomerado. Cree una solución que permita acceder a estos montos de cada empresa, sin indagar a profundidad, y realizar una suma por cada monto.

EJERCICIOS

DOS PROGRAMAS

Un juego en línea requiere los datos de rendimiento de cada jugador, *score*, para poder realizar una lista ordenada de *ranking* del videojuego. Desarrolle una interface que permita entregar estos datos al programa del servidor sin problemas y sin interferir en el programa a nivel local.

EJERCICIOS

ORQUESTA

Se debe crear un programa que permita interpresar cada pista de audio de un instrumento, para escucharlo en general. Existen varios tipos de instrumentos: de viento, de cuerdas y de percusión. Es necesario conocer la pista de audio de cada uno.