



ESCUELA DE
INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



Día, Fecha:

Jueves, 25/07/2024

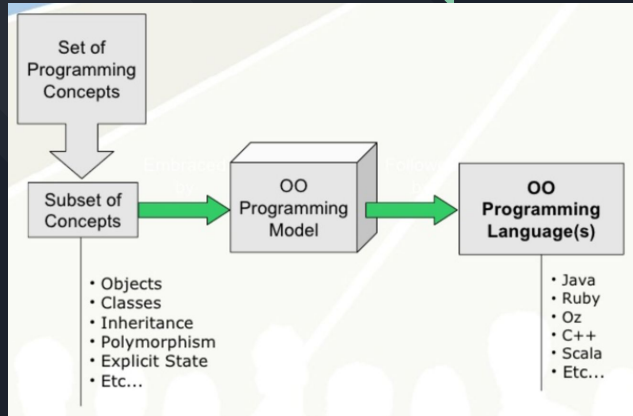
Hora de inicio:

10:40 - 12:20

Introducción a la Programación y Computación 2 [P]

Denilson Florentín de León Aguilar

Programación Orientada a Objetos (Object-oriented programming)



- Clases
- Objetos
- Métodos y Atributos
- Pilares
- Herencia
- Abstracción
- Polimorfismo
- Encapsulamiento

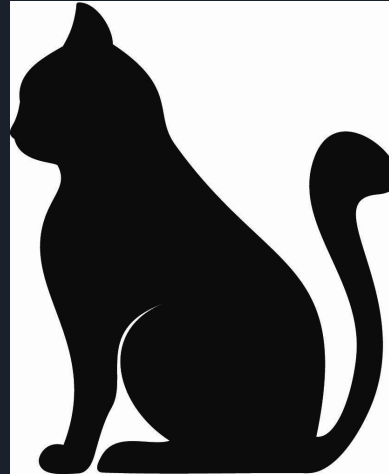
¿Qué es una clase en POO (OOP)?

DEFINICIÓN FORMAL:

Una "clase" es un modelo o plantilla que define la estructura y comportamiento de los objetos que se crearán a partir de ella. En otras palabras, una clase sirve como un plano para la creación de objetos.

La clase especifica las propiedades (atributos) y acciones (métodos) que los objetos derivados de ella poseerán

Los atributos representan las características del objeto, mientras que los métodos son las funciones o acciones que el objeto puede realizar.



Definición de un GATO:

- ser vivo
- tiene un color específico, puede ser más de uno
- hace ruidos
- puede tener nombre
- come
- respira
- foto

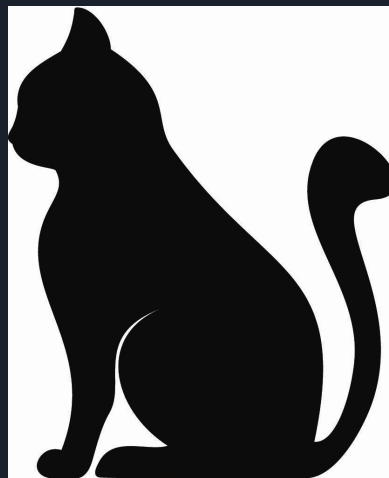
¿Qué es una clase en POO (OOP)?

DEFINICIÓN PERSONAL:

Es el concepto y un conjunto de atributos que forman un algo (ser vivo, cosas abstractas, información).

Concepto de qué es un GATO:

- ser vivo
- tiene un color específico, puede ser más de uno
- hace ruidos
- puede tener nombre
- come
- respira
- foto



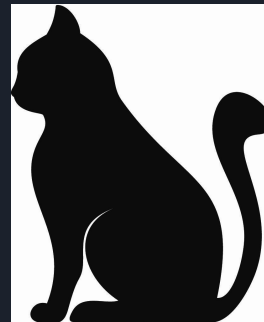
Entonces -> Clase GATO:

- estaVivo: boolean
- colores: list
- sonido:
- nombre:
- tieneHambre: boolean
- foto:

¿Qué es una Instancia en POO (OOP)?

DEFINICIÓN FORMAL:

Cuando se instancia una clase, se crea un objeto, también conocido como una instancia de esa clase. Cada objeto tiene sus propios valores para los atributos, pero comparte los mismos métodos definidos por la clase.



Clase GATO:

- estaVivo: boolean
- colores: list
- sonido:
- nombre:
- tieneHambre: boolean
- foto:

Instancia gato1:

- estaVivo: si
- colores: blanco, negro
- sonido: miau
- nombre: Beluga
- tieneHambre: si
- foto:



Instancia gato2:

- estaVivo: si
- colores: blanco
- sonido: miaaaaau
- nombre: Sinko peso
- tieneHambre: no
- foto:

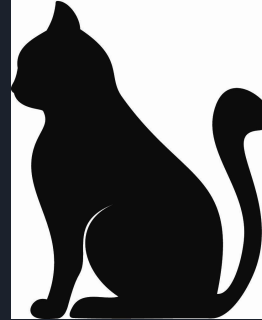


¿Qué es una Instancia en POO (OOP)?

DEFINICIÓN PERSONAL:

La “Clase” de “GATO” es un CONCEPTO, pues define qué es un gato, qué características posee y qué puede realizar.

Ahora las INSTANCIAS “gato1” y “gato2”, pasan del concepto a algo REAL, pues pueden tener características individuales entre ellos pero no dejarán de ser un gato



Clase GATO:

- estaVivo: boolean
- colores: list
- sonido:
- nombre:
- tieneHambre: boolean
- foto:

Instancia gato1:

- estaVivo: si
- colores: blanco, negro
- sonido: miau
- nombre: Beluga
- tieneHambre: si
- foto:



Instancia gato2:

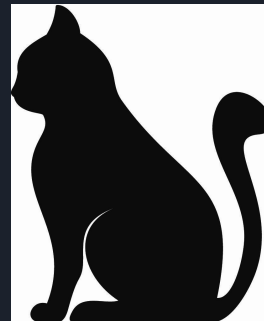
- estaVivo: si
- colores: blanco
- sonido: miaaaaau
- nombre: Sinko peso
- tieneHambre: no
- foto:



¿Qué es un Objeto en POO (OOP)?

Un objeto es el nombre que recibe cada una de las instancias, por ende, el flujo de la programación orientada a objetos es el siguiente:

- Definición de algo: Clase
 - Atributos
 - Acciones: Métodos y funciones
- Se crean objetos (usables) de la Clase:
 - Instancias



Clase GATO:

- estaVivo: boolean
- colores: list
- sonido:
- nombre:
- tieneHambre: boolean
- foto:

Instancia gato1:

- estaVivo: si
- colores: blanco, negro
- sonido: miau
- nombre: Beluga
- tieneHambre: si
- foto:



Instancia gato2:

- estaVivo: si
- colores: blanco
- sonido: miaaaaau
- nombre: Sinko peso
- tieneHambre: no
- foto:

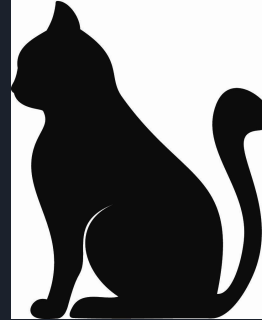


Métodos y Atributos en POO (OOP)

Siguiendo la definición dada al inicio *“La clase especifica las propiedades (atributos) y acciones (métodos) que los objetos derivados de ella poseerán”*.

En este caso, los atributos de un GATO son: ¿está vivo?, ¿qué colores tiene?, ¿qué sonido hace?; mientras que las acciones son: saltar, comer, correr, dormir, arañar, etc.

Todos los gatos tienen esas características comunes, pero así como los humanos son únicos, los gatos también (objetos).



Clase GATO:

- estaVivo: boolean
- colores: list
- sonido:
- nombre:
- tieneHambre: boolean
- foto:

Instancia gato1:

- estaVivo: si
- colores: blanco, negro
- sonido: miau
- nombre: Beluga
- tieneHambre: si
- foto:

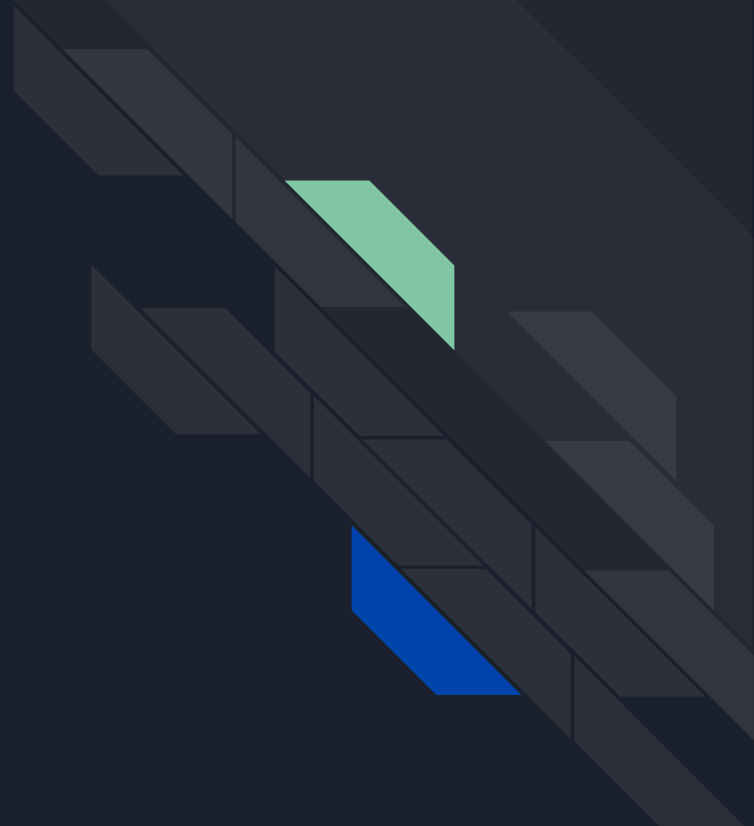


Instancia gato2:

- estaVivo: si
- colores: blanco
- sonido: miaaaaau
- nombre: Sinko peso
- tieneHambre: no
- foto:



En Python



Código en Python de Gato y gato1, gato2

```
class Gato:
    def __init__(self, esta_vivo, colores, sonido, nombre, tiene_hambre, foto):
        self.esta_vivo = esta_vivo
        self.colores = colores
        self.sonido = sonido
        self.nombre = nombre
        self.tiene_hambre = tiene_hambre
        self.foto = foto
```

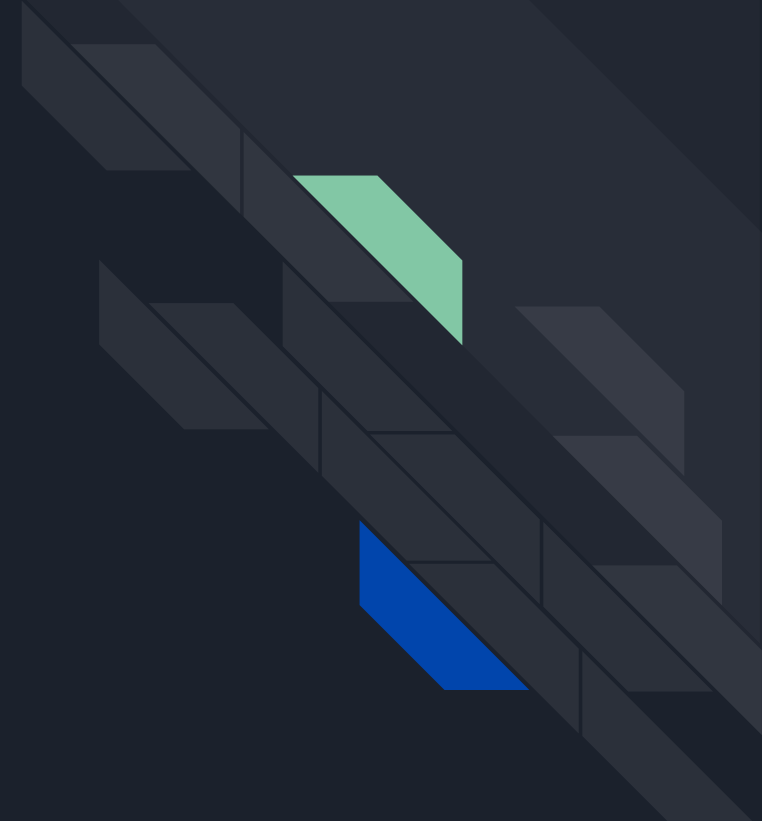
Instancia gato1

```
gato1 = Gato(esta_vivo=True, colores=['blanco', 'negro'], sonido='miau',
nombre='Beluga', tiene_hambre=True, foto=None)
```

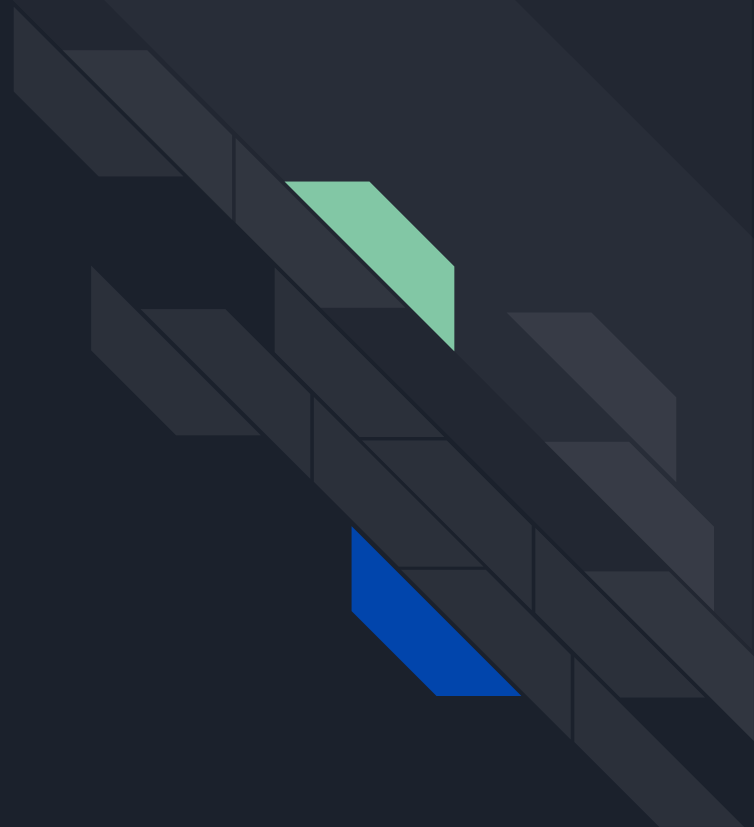
Instancia gato2

```
gato2 = Gato(esta_vivo=True, colores=['blanco'], sonido='miaaaau',
nombre='Sinko peso', tiene_hambre=False, foto=None)
```

Herencia y Polimorfismo



Recordatorio: Captura de pantalla





Herencia en Programación Orientada a Objetos (OOP):

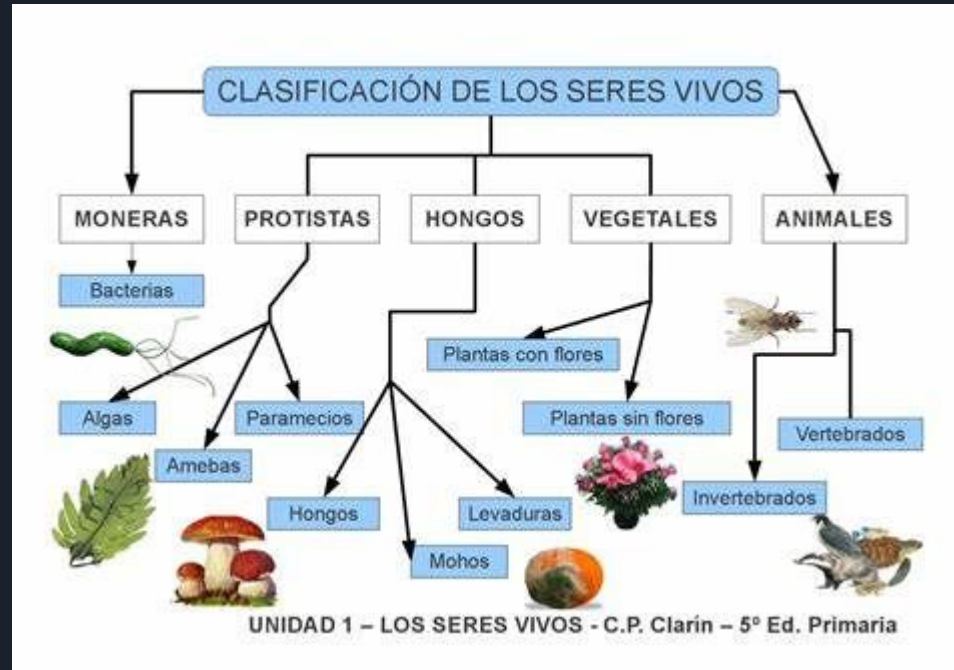
Definición Formal:

La herencia es un principio fundamental en OOP que permite a una clase heredar propiedades y comportamientos de otra clase. La clase que hereda se conoce como clase derivada o subclase, y la clase de la que se hereda se llama clase base o superclase. Este concepto promueve la reutilización de código y la creación de jerarquías de clases, donde las subclases pueden extender y especializar las características de la superclase.

Herencia en Programación Orientada a Objetos (OOP):

Si hacemos un análisis de los seres vivos nos daremos cuenta que existen 5 reinos:

- Moneras
- Protistas
- Hongos
- Vegetales
- Animales

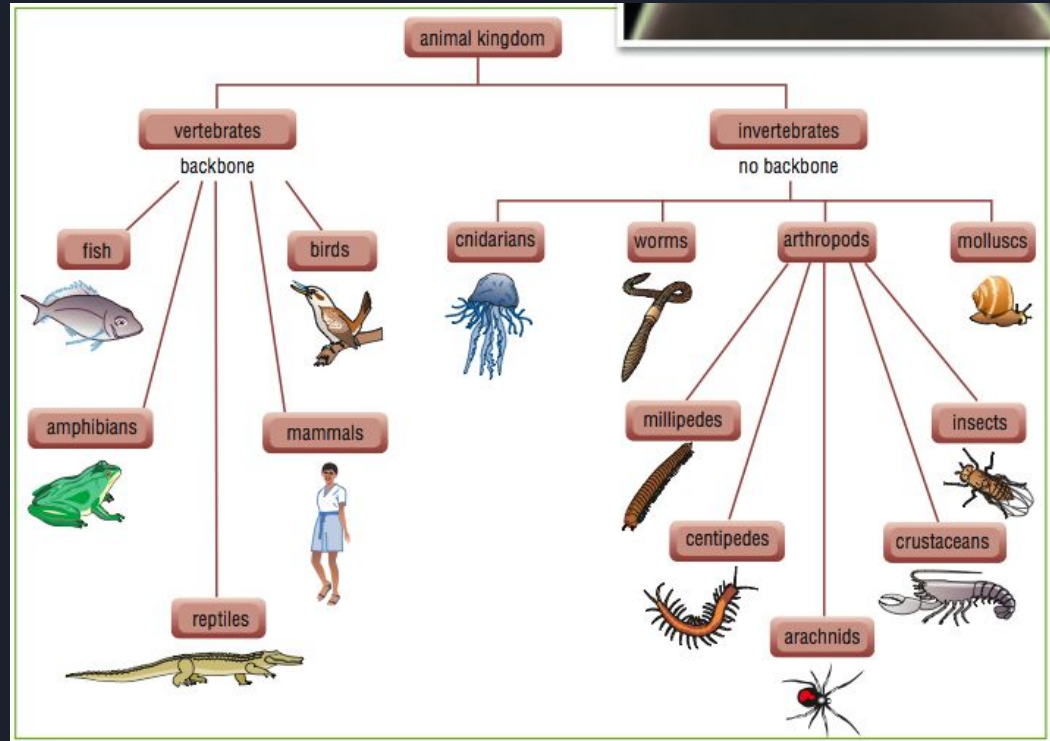


Herencia en Programación Orientada a Objetos (OOP):

Si nos enfocamos en el reino animal observaremos que se divide en 2 sub categorías:

- Vertebrados
- Invertebrados

Y a su vez, estos se dividen más y más, sin embargo hay que notar que dependiendo de cada sub categoría, estos comparten ciertas acciones (funciones, métodos) o características (atributos)



Herencia en Programación Orientada a Objetos (OOP):

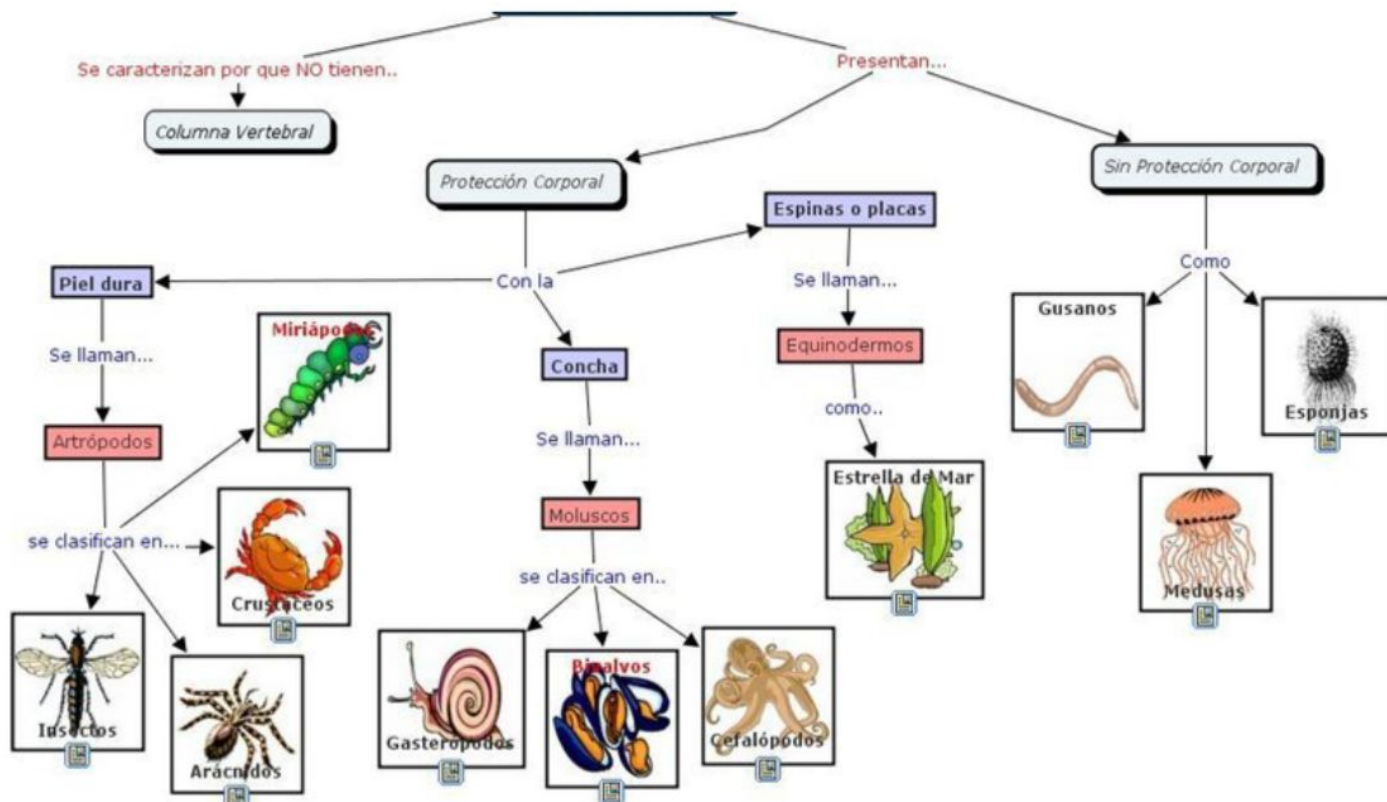
Los animales invertebrados:

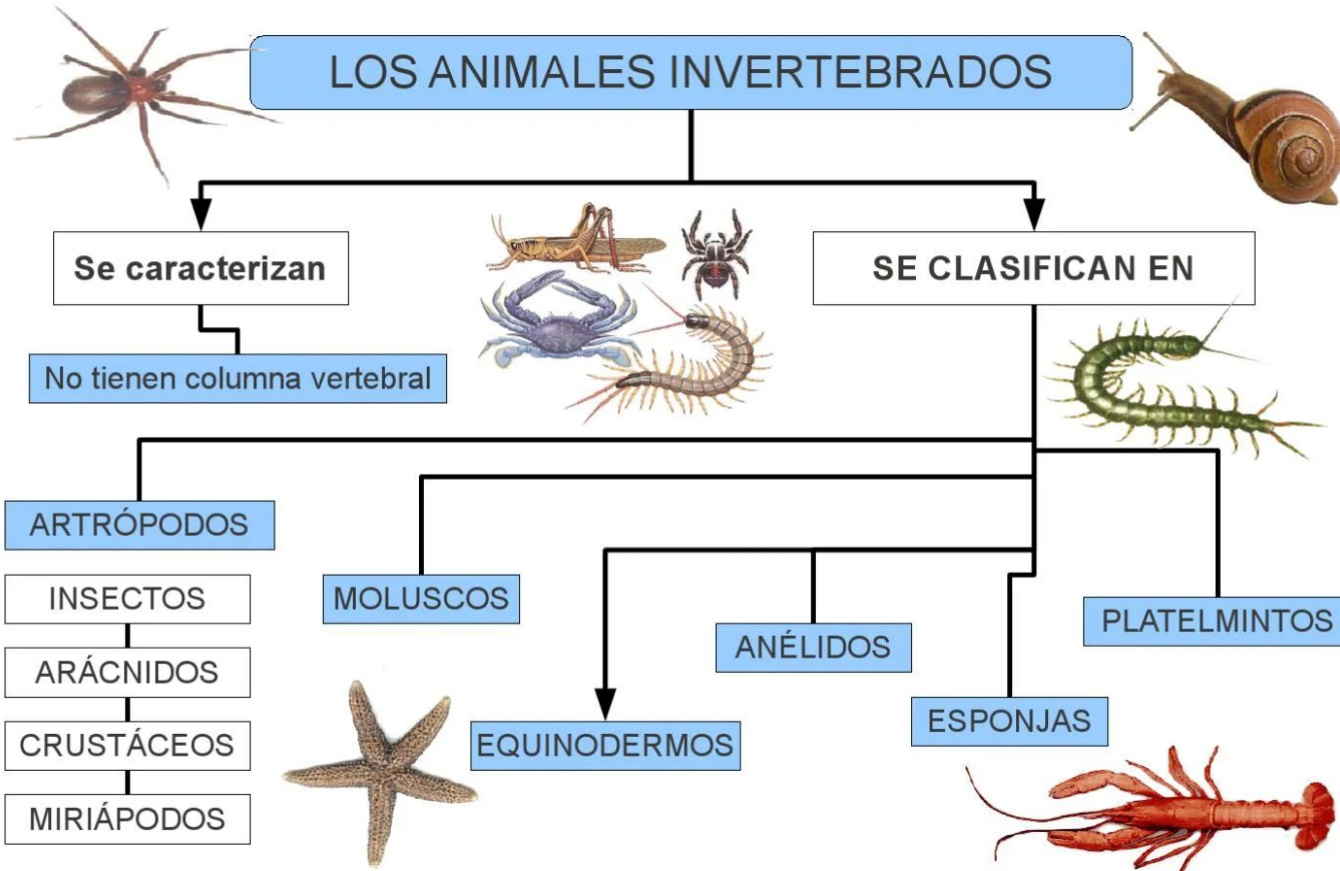
Los invertebrados son los animales que no tienen columna vertebral; es decir, carecen de vertebración.

Mariquita

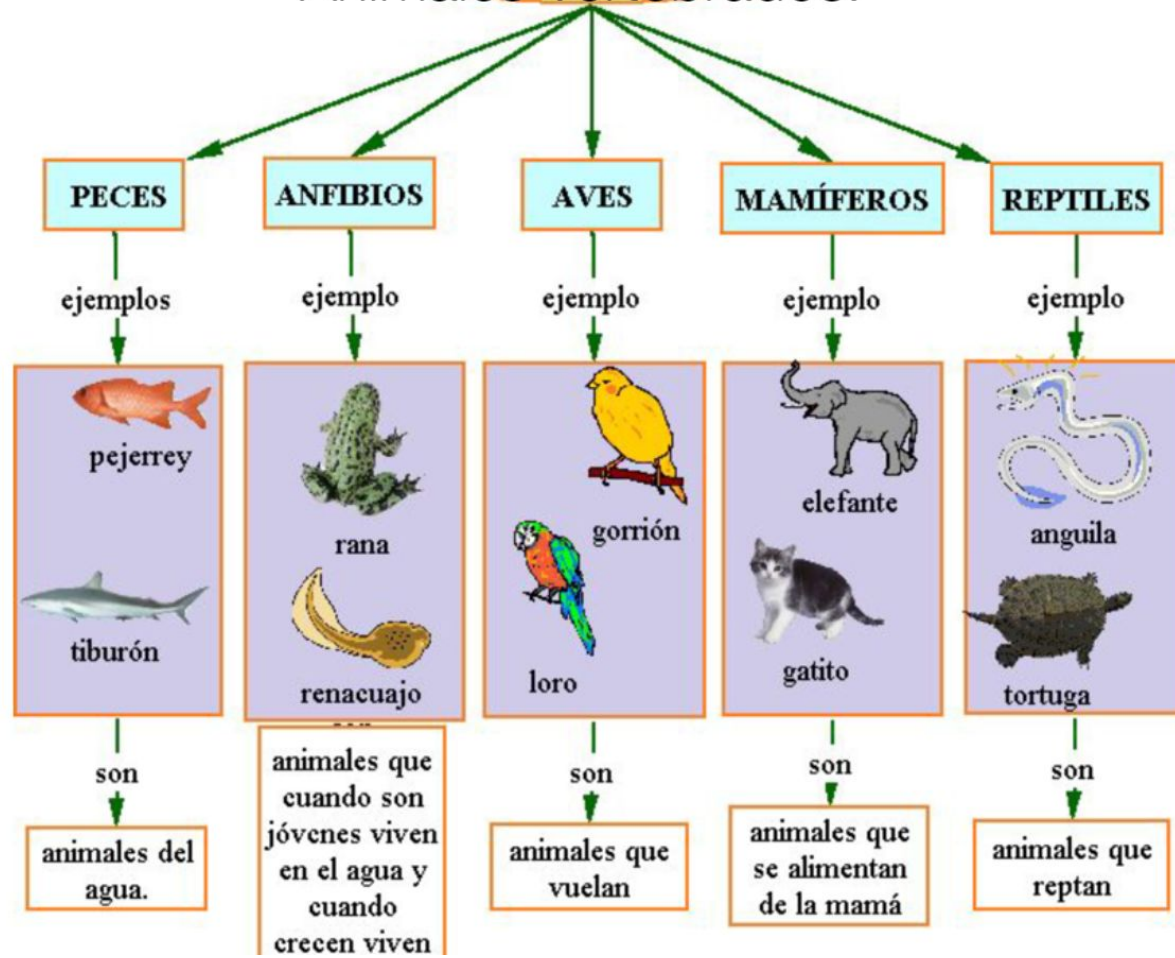


Animales Invertebrados:





Animales Vertebrados:

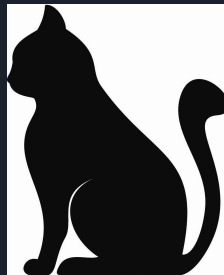


Herencia en Programación Orientada a Objetos (OOP):

Ejemplo con la Clase GATO:

Supongamos que tenemos una clase base llamada "Animal" que tiene atributos y métodos generales para cualquier tipo de animal. La clase "Gato" podría heredar de la clase "Animal", obteniendo así características generales de los animales, como respirar, moverse, etc.

En este ejemplo podríamos partir desde ser vivo, reino animal, etc. Sin embargo podría resultar en algo complejo, por ende haremos un ejercicio simple.




Herencia en Programación Orientada a Objetos (OOP):

```
class Animal:
    def __init__(self, esta_vivo):
        self.esta_vivo = esta_vivo

    def respirar(self):
        print("El animal está respirando")

class Gato(Animal):
    def __init__(self, esta_vivo, colores, sonido, nombre, tiene_hambre, foto):
        super().__init__(esta_vivo)
        self.colores = colores
        self.sonido = sonido
        self.nombre = nombre
        self.tiene_hambre = tiene_hambre
        self.foto = foto
```


Además de los métodos de la clase Animal, se pueden añadir métodos específicos para los gatos



Abstracción en Programación Orientada a Objetos (OOP):

Definición Formal:

La abstracción es el proceso de simplificar y centrarse en los aspectos esenciales de un objeto, ignorando detalles menos relevantes. En OOP, se logra a través de la creación de clases abstractas que definen atributos y métodos comunes sin proporcionar una implementación específica. Las clases concretas derivadas de las clases abstractas llenan los detalles particulares.



Abstracción en Programación Orientada a Objetos (OOP):

Ejemplo con la Clase GATO:

Podemos tener una clase abstracta "Mamifero" que define características compartidas por todos los mamíferos, pero sin implementar completamente cada detalle. La clase "Gato" hereda de "Mamifero" y completa los detalles específicos de un gato.


Abstracción en Programación Orientada a Objetos (OOP):

```
from abc import ABC, abstractmethod

class Mamifero(ABC):
    @abstractmethod
    def amamantar(self):
        pass

class Gato(Mamifero):
    def amamantar(self):
        print("Los gatos no amamantan directamente, pero son mamíferos")

# Otros métodos y atributos específicos de la clase Gato
```

Polimorfismo en Programación Orientada a Objetos (OOP):

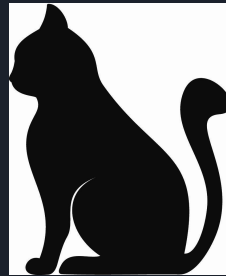
Definición Formal:

El polimorfismo es un principio en OOP que permite que un mismo método o función tenga diferentes implementaciones según el tipo de objeto al que se aplique. En otras palabras, un objeto puede tomar múltiples formas y responder de manera diferente a la misma operación dependiendo de su clase.

Polimorfismo en Programación Orientada a Objetos (OOP):

Ejemplo con la Clase GATO y Herencia:

Supongamos que tenemos una clase base llamada "Animal" que tiene un método llamado "sonido". La clase "Gato" hereda de "Animal" y proporciona su propia implementación del método "sonido". Cuando llamamos al método "sonido" en una instancia de "Gato", se ejecutará la implementación específica de la clase "Gato".



Polimorfismo en Programación Orientada a Objetos (OOP):

```
class Animal:
    def sonido(self):
        print("Sonido genérico de un animal")

class Gato(Animal):
    def sonido(self):
        print("Miau")

# Crear instancias
animal_generico = Animal()
mi_gato = Gato()

# Llamadas al método sonido
animal_generico.sonido() # Imprimirá "Sonido genérico de un animal"
mi_gato.sonido()        # Imprimirá "Miau"
```

Polimorfismo en Programación Orientada a Objetos (OOP):

```
class Animal:
    def sonido(self):
        print("Sonido genérico de un animal")

class Gato(Animal):
    def sonido(self):
        print("Miau")

# Crear instancias
animal_generico = Animal()
mi_gato = Gato()

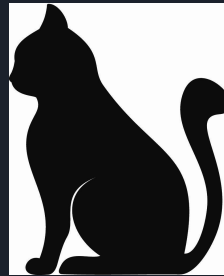
# Llamadas al método sonido
animal_generico.sonido() # Imprimirá "Sonido genérico de un animal"
mi_gato.sonido()         # Imprimirá "Miau"
```

En este ejemplo, tanto la clase base "Animal" como la subclase "Gato" tienen un método llamado "sonido", pero cada una tiene su propia implementación. Cuando llamamos al método "sonido" en un objeto, se ejecuta la implementación correspondiente a la clase del objeto. Esto es un ejemplo de polimorfismo, ya que un mismo método, "sonido", se comporta de manera diferente según el tipo de objeto al que se aplique.

Encapsulamiento en Programación Orientada a Objetos (OOP):

Definición Formal:

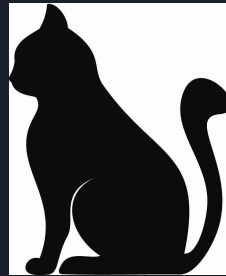
El encapsulamiento es un principio en OOP que consiste en ocultar los detalles internos de implementación de un objeto y restringir el acceso directo a ciertos componentes. Se logra utilizando modificadores de acceso (como públicos, privados y protegidos) para controlar la visibilidad de los atributos y métodos de una clase. El encapsulamiento ayuda a proteger la integridad del objeto y promueve la seguridad y la eficiencia del código.



Encapsulamiento en Programación Orientada a Objetos (OOP):

Ejemplo con la Clase GATO y Encapsulamiento:

Supongamos que queremos encapsular algunos atributos de la clase "Gato" para que no puedan ser accedidos directamente desde fuera de la clase.



Encapsulamiento en Programación Orientada a Objetos (OOP):

```
class Gato:
    def __init__(self, esta_vivo, colores, sonido, nombre, tiene_hambre, foto):
        self._esta_vivo = esta_vivo
        self._colores = colores
        self._sonido = sonido
        self._nombre = nombre
        self._tiene_hambre = tiene_hambre
        self._foto = foto
```

Métodos getter para acceder a los atributos encapsulados

```
def get_esta_vivo(self):
    return self._esta_vivo
```

```
def get_colores(self):
    return self._colores
```

Otros métodos setter o getter según sea necesario

Encapsulamiento en Programación Orientada a Objetos (OOP):

```
# Crear una instancia
```

```
mi_gato = Gato(esta_vivo=True, colores=['blanco', 'negro'], sonido='miau', nombre='Beluga',  
tiene_hambre=True, foto=None)
```

```
# Acceder a los atributos encapsulados mediante métodos getter
```

```
print("¿El gato está vivo?", mi_gato.get_esta_vivo())
```

```
print("Colores del gato:", mi_gato.get_colores())
```

En este ejemplo, los atributos de la clase "Gato" están marcados como encapsulados usando un guion bajo antes de sus nombres. Además, se proporcionan métodos getter para acceder a estos atributos de manera controlada y segura desde fuera de la clase. Esto ilustra cómo el encapsulamiento ayuda a gestionar el acceso a los detalles internos del objeto y brinda una interfaz controlada para interactuar con ellos