Preguntas de análisis

1. Si deseo modificar el mensaje del método ruido al crear un objeto Pajaro sin alterar la clase Animal ¿Qué debo agregarle al código? (Por ejemplo, al llamar el método ruido imprima, cantar y silbar).

RTA: Se debe sobrescribir (redefinir) el método ruido() en la clase Pajaro y dentro de este poner: print("cantar y silbar")

2. Si se crea una nueva clase Pez, y no se definen nuevos métodos, constructor y atributos. ¿Qué constructor tendrá esta clase, qué argumentos recibe? ¿Qué otros métodos y atributos tendrán estos mismos?

RTA: Si hereda de la clase Animal, entonces la clase Pez heredara:

- como atributos: _nombre, _raza, _edad;
- como métodos: los set() y los get() de raza y nombre, junto con los metodos caminar(), correr(), ruido(),
- además del __init__ que recibe los argumentos de nombre, edad y raza.

```
class Animal(SerVivo):
    _totalCreados = 0

    def __init__ (self, nombre, edad, raza):
        self._nombre = nombre
        self._edad = edad
        self._raza = raza
        Animal._totalCreados += 1

    def setRaza(self, raza):
        self._raza = raza

    def getRaza(self):
        return self._raza

    def setNombre(self, nombre):
        self._nombre = nombre

    def getNombre(self):
        return self._nombre

    def caminar(self):
        pass

    def correr(self):
        pass

    def ruido(self):
        pass
```

3. ¿Qué ocurre con el atributo nombre y edad de la clase SerVivo, al momento de definirse en la Clase Animal? ¿Cómo cambiaría el código del constructor para que estos atributos sean el mismo?

RTA: A pesar de que se definen en dos clases distintas, en python hay un solo espacio de memoria para los atributos de instancia que se llamen igual, sin embargo, se puede usar el super en la clase Animal de la siguiente manera:

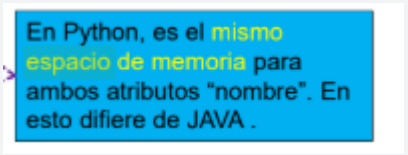
super().__init__(nombre,edad)

```
def __init__ (self, nombre, edad, raza):
    super().__init__(nombre,edad)
    #self._nombre = nombre
    #self._edad = edad
    self._raza = raza
    Animal._totalCreados += 1
```

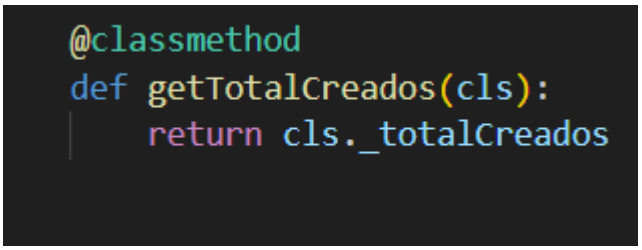
En Python, es el mismo espacio de memoria para ambos atributos "nombre". En esto difiere de JAVA .

4. En la clase Animal se sobrescribieron los métodos setNombre y getNombre, ¿Como modificaría estos métodos para que su funcionamiento no oculte algún valor de la clase padre? ¿Podría plantearse esta solución usando super()?
RTA: En teoría no se debería ocultar ningún valor porque es Python y hay un mismo espacio de memoria para atributos de instancia que se llamen igual, aun así para evitar confusiones haría lo siguiente en la clase Animal:

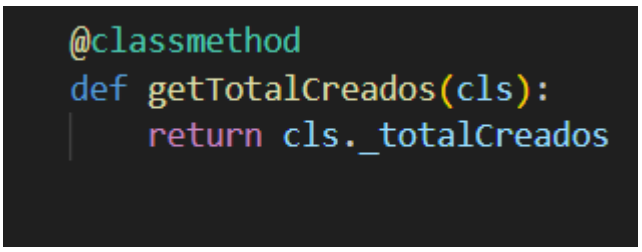
Python
```
def setNombre(self, nombre):
    super().setNombre(nombre)

def getNombre(self):
    return super().getNombre()
```



5. El atributo totalCreados de la clase SerVivo ¿es heredado por las demás clases? ¿En este caso ocurre ocultación de los atributos al definirlo de nuevo en las clases hijas?
RTA: Si, este atributo de clase se hereda, pero se reescribe en cada clase ocultando este atributo de la clase "padre", aun asi se puede acceder a estos porque cada clase tiene un método de clase para obtenerlo.



6. ¿Los métodos getTotalCreados sobrescriben al método de su padre?
RTA: se podría decir que si, porque este método se sobreescribe en cada clase para poder acceder al atributo de clase _totalCreados, propio de cada una de las clases.



7. ¿Qué métodos hereda una clase que hereda de la clase Persona?
RTA: se heredan los métodos de Persona como__init__, adueñarAnimal y getTotalCreados, por otro lado de la clase de SerVivo se hereda el setNombre, getNombre,setEdad y getEdad.

```
def __init__(self, nombre, edad):
    super.__init__(nombre, edad)
    self._animalAcargo = None
    Persona._totalCreados += 1

def aduenarAnimal(self, x):
    self._animalAcargo = x
    x.ruido()

@classmethod
def getTotalCreados(cls):
    return cls._totalCreados
```

```
def setNombre(self, nombre):
    self._nombre = nombre

def getNombre(self):
    return self._nombre

def setEdad(self, edad):
    self._edad = edad

def getEdad(self):
    return self._edad
```

8. ¿Qué tipo de objetos podrían pasársele al método aduenarAnimal de la clase Persona?¿Se le puede pasar un objeto serVivo?

RTA:

```
def aduenarAnimal(self, x):
    self._animalAcargo = x
    x.ruido()
```

Se podría pasar en el parámetro x todo objeto que tenga el método ruido, que en este caso sería la clase Animal y todos sus “hijos” (Pajaro,Perro o Gato), es decir, no se le puede pasar el objeto serVivo

9. ¿Cómo podría obtener la cantidad de seres vivos que se creen (objetos creados)?
RTA: se podría imprimir el método de clase getTotalCreados de la clase SerVivo
print(SerVivo.getTotalCreados())

10. Si se define el siguiente método en la clase Perro:
def getRaza(self, tipo):
return self._raza + " " + tipo
¿Qué ocurre con el método getRaza de la clase Animal? ¿Este método se sobreescribe o se sobrecarga?
RTA: El método getRaza en la clase Perro sobrescribe el método de la clase Animal, para saber cuál de los dos se ejecutara dependerá de qué clase es el objeto.