PROGRAMACIÓN ORIENTADA A OBJETOS - UNALMED 2024 -2

Preguntas de análisis

Andrea Merino Mesa 1035991011

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).**

Se puede aplicar la sobre-escritura de métodos, definiendo un método ruido () con comportamientos diferentes (**cantar y silbar**).

```
def ruido(self):
        return "cantar y silbar"
```

2. **Si se crea una nueva clase Pez que herede de animal, 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?**

Inicializador: Como cuando una clase no crea su propio __init__, lo hereda de su padre que en este caso es Animal, entonces la clase Pez heredaría el inicializador de Animal (def __init__ (self, nombre, edad, raza)) de forma automática

Atributos: La clase Pez hereda todos los atributos de la clase Animal (ya que para herencia en Python no tenemos en cuenta los UnderScore), tanto los de instancia: self._raza como los de clase: _totalCreados. Además, como la clase Animal hereda de SerVivo, también hereda self._nombre, self._edad.

Métodos: La clase Pez hereda todos los métodos de la clase Animal, tanto los de instancia: setRaza(self, raza), getRaza(self),setNombre(self, nombre), getNombre(self), caminar(self), correr(self), ruido(self); como el de clase: getTotalCreados(cls). Además, como la clase Animal hereda de SerVivo, también hereda setEdad(self, edad), getEdad(self). Por último tomar en cuenta que también hereda de Object.

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?**

Lo que ocurre con el atributo nombre y edad es una **redefinición de atributos**, pues en Python es el mismo espacio de memoria el que se usa para los atributos de instancia. Para que estos atributos sean el mismo, en vez de definirlos por separado en ambas clases, se podría **llamar al constructor de la clase SerVivo** para que inicialice esos atributos, de esta manera:

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

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()?**

Con super() no es posible acceder a los atributos de instancia de la clase padre ya que super() es para acceder a elementos de clase, sin embargo, con el super si podemos acceder al método sobreescrito de la clase padre y así evitar que se oculte algún valor de la clase padre.

Usando super():

```
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?**

Los atributos de clase **se heredan** en Python a las clases hijas y se pueden acceder usando tanto el nombre de la clase padre como el de las hijas. Sin embargo, si se **define de nuevo** un atributo en la clase hija con el mismo nombre que el de la clase padre, este **redefine** el atributo heredado, pues en Python ocupan diferente espacio de memoria. Cabe aclarar que Python no es tan estricto como Java, por lo que en este no se suele hablar estrictamente de Ocultamiento sino que se habla de Redefinición.

6. **¿Los métodos getTotalCreados sobrescriben al metodo de su padre?**

Si, **se sobreescriben** porque En Python la sobre-escritura de métodos debe **tener el mismo nombre** que el método heredado, **lo cual cumple**. Aunque parezca que en este caso no hay sobreescritura porque el método aparece igual, **el sólo volverlo a definir ya lo sobre-escribe.** Por tanto, para acceder al del padre debemos usar el nombre de la clase padre (porque el super() no es tan recomendable en estos casos).

7. **¿Qué métodos hereda una clase que hereda de la clase Persona?**

   Hereda el inicializador de Persona (**__init__ (self, nombre, edad)**), el método de instancia **aduenarAnimal(self, x)** y el de clase **getTotalCreados(cls)**. Además, como **Persona hereda de SerVivo**, la clase que herede de Persona también heredará el inicializador de SerVivo (**__init__ (self, nombre, edad)**), el método de clase de SerVivo **getTotalCreados(cls)** y los métodos de instancia de dicha clase: **setNombre(self, nombre), getNombre(self), setEdad(self, edad) y getEdad(self).** Por último, todas las clases **heredan de Objet** por lo que también hereda sus métodos.

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

En este caso **se podría pasar cualquier objeto de alguna clase que posea el método ruido()** para que no genere error, lo que serían objetos de clase menor o igual a Animal (Animal, Pez, Pajaro y Perro). SerVivo no se puede pasar porque **no tiene el método ruido ()** y generaría error. Sin embargo, de no ser por esto, si se podría porque a diferencia de JAVA, Python no es tipado y por tanto no ocurre lo de que por ser una clase mayor a Persona no se puede pasar.

9. **¿Cómo podría obtener la cantidad de seres vivos que se creen (objetos creados)?**

   Podemos acceder al método de clase getTotalCreados(cls), llamándolo desde la clase: **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?

En Python no hay sobrecarga como tal, aunque se puede simular, pero en este caso lo que **sucede es una sobreescritura,** ya que en Python para **sobre-escribir solo se necesita definir nuevamente el método usando el mismo nombre** del método del padre.