Respuestas Ejercicio 2 – Taller 5 – Python

Fabián Andrés Hurtado Arango

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**).

Para cambiar el mensaje del método ruido en la clase **Pajaro** sin afectar a la clase **Animal**, puedes redefinir el método ruido dentro de la clase **Pajaro**, agregando las instrucciones necesarias, por ejemplo, un print que diga "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?

Una clase como **Pez**, que no define nuevos métodos, constructores o atributos, hereda todo de la clase base **object**. Su constructor por defecto no recibe parámetros ni realiza inicialización adicional, y los métodos que hereda incluyen __del__(), __str__(), y __init__(), entre otros.

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?

Cuando la clase **Animal** hereda los atributos nombre y edad de la clase **SerVivo**, pero también los redefine, se crean nuevos atributos que son independientes de los originales. En el constructor, sería necesario usar super() para inicializar los atributos heredados antes de agregar los nuevos.

4. En la clase **Animal** se sobrescribieron los métodos **setNombre** y **getNombre**, ¿Cómo modificaría estos métodos para que su funcionamiento no oculte algún valor de la clase padre? ¿Podría plantearse esta solución usando **super()**?

Para que los métodos sobrescritos setNombre y getNombre no oculten los valores de la clase padre, puedes utilizar super() dentro de estos métodos. Esto asegura que la funcionalidad de la clase padre permanezca accesible y no se sobrescriba.

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?

El atributo totalCreados de la clase **SerVivo** no es heredado automáticamente. En cambio, se oculta en las clases hijas si estas lo redefinen. Esto significa que cada clase hija tiene su propia versión independiente del atributo.

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

En Python, los métodos como getTotalCreados no sobrescriben los métodos de la clase padre, sino que permanecen independientes.

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

Una clase que hereda de **Persona** recibe métodos como __init__, adueñarAnimal, getNombre, setNombre, getEdad, setEdad, getTotalCreados, __str__, y __del__, entre otros.

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

Cualquier tipo de objeto podría pasar al método adueñarAnimal, ya que Python no impone restricciones estrictas sobre tipos. Sin embargo, el código podría fallar si el objeto no tiene los atributos esperados, como ruido.

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

Para obtener la cantidad de seres vivos creados, debes usar el método getTotalCreados desde las clases adecuadas, asegurándote de acceder al atributo correspondiente si se encuentra en una clase base como **Animal**.

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 sobrescribe o se sobrecarga?

En Python, cuando un método como getRaza es redefinido en una clase hija, el método de la clase padre queda oculto. Esto no se considera sobrecarga, ya que en Python solo importa el nombre del método, no su firma.