- 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/ Si de desea modificar el método ruido de la clase Pajaro sin alterar la clase Animal, se debe sobrescribir el método ruido dentro de la clase Pajaro, para redefinir el comportamiento del método solo en la clase pájaro.

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

- 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 se crea una clase Pez sin definir nada, esta clase no tendrá nada nuevo, por lo que heredará el constructor de su clase padre, en este caso Animal con los argumentos nombre, edad y raza. También heredará los atributos _nombre, _edad, _raza, y los métodos de SerVivo y Animal, como setRaza, getRaza, setNombre, getNombre, caminar, correr, ruido y getTotalCreados.

- ¿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/ Si en la clase Animal, los atributos nombre y edad se definen nuevamente, se ocultarían al redefinirlos. Para evitar esto y usar los atributos de la clase padre, el constructor de Animal debe modificarse usando super():

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

- 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/ Para evitar que los métodos setNombre y getNombre en Animal oculten los de la clase padre, se debe llamar los métodos de la clase padre, utilizando el super():

```
def setNombre(self, nombre):
```

```
    super().setNombre(nombre)

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

- 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/Sí, el atributo _totalCreados se hereda por las clases hijas. Sin embargo, cuando se define nuevamente en las clases hijas (Animal, Gato, etc.), ocurre ocultación, es decir, cada una redefine este atributo, creando una instancia independiente con un contador propio para cada clase (Animal, Gato, Pajaro, Perro, Persona, etc.). Por lo tanto, cada clase tiene su propio espacio de memoria para _totalCreados.

- ¿Los métodos getTotalCreados sobrescriben al metodo de su padre?

Rta/ Sí, los métodos getTotalCreados sobrescriben al método de la clase padre. Ya que, al tener cada clase (Animal, Gato, etc.) su propio atributo _totalCreados, al momento de implementar el método getTotalCreados, devuelve su respectivo atributo.

- ¿Qué métodos hereda una clase que hereda de la clase Persona?

Rta/ Los métodos que heredaría una clase que hereda de Persona incluyen el inicializador __init__ de Persona y el método getTotalCreados. Además, heredaría métodos de la clase padre de Persona, SerVivo, como setNombre, getNombre, setEdad, getEdad y getTotalCreados.

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

Rta/ Los objetos que podrían pasársele al método aduenarAnimal de la clase Persona son los que tengan el método ruido, es decir, los objetos de las clases Animal, Gato, Perro y Pajaro. Por lo que, no se puede pasar un objeto SerVivo ya que este no tiene el método ruido.

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

Rta/ Se puede obtener la cantidad total de seres vivos creados, utilizando el método de clase getTotalCreados de SerVivo, el cual retorna el valor del atributo _totalCreados de la clase SerVivo.

- 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/ Si se define el método getRaza con una nueva implementación diferente en la clase Perro, esta versión oculta la versión del método en Animal, es decir, este método sobreescribe la versión heredada. Por lo tanto, para los objetos de tipo Perro, solo se usará la versión específica del método getRaza.