**Solución punto 2 – Taller 5 Python:**

1. **Si deseo modificar el mensaje del método ruido al crear un objeto Pajaro sin alterar la clase Animal ¿Qué debo agregarle al código?**

Para modificar el mensaje del método ruido al crear un objeto Pajaro sin alterar la clase Animal, se debe "sobrescribir" el método ruido directamente en la clase Pajaro. Esto se hace definiendo nuevamente el método dentro de la clase Pajaro y permite cambiar su comportamiento específico solo para los objetos de tipo Pajaro.

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?**

Si se crea una nueva clase Pez y no se le definen nuevos métodos, constructor ni atributos, se heredará el comportamiento y los atributos de la clase SerVivo (su "padre"). Mientras no se le defina un comportamiento propio:

- Se usará el constructor (el inicializador / método init) de SerVivo que requiere dos argumentos: nombre, edad.
- La clase Pez tendrá los métodos: setNombre(nombre) que modifica el atributo _nombre, getNombre() que obtiene el valor de _nombre, setEdad(edad) que modifica el atributo _edad, getEdad() que obtiene el valor de _edad, y el método de clase getTotalCreados() que devuelve el total de instancias creadas de SerVivo o sus subclases.
- Sus atributos heredados son: _nombre que se define y asigna en el constructor de SerVivo, _edad que también se inicializa en el constructor de SerVivo, y el atributo de clase _totalCreados que lleva el conteo de todas las instancias de SerVivo y sus subclases.

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?**

Al momento de definirse en la clase Animal, los atributos _nombre y _edad son replicados dentro de la instancia de la clase Animal, pero no están directamente vinculados a los atributos de SerVivo, es decir, que cada clase maneja sus propias copias de estos atributos.

Para hacer que estos atributos sean los mismos y compartan espacio de memoria con los de SerVivo, se debe llamar explícitamente al constructor de SerVivo dentro del constructor de Animal usando super().__init__(nombre, edad). Esto asegura que _nombre y _edad sean gestionados por el constructor del padre y no duplicados.

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()?**

Para modificar los métodos setNombre y getNombre en la clase Animal sin ocultar los valores de la clase padre, se puede hacer uso de super() para invocar los métodos correspondientes de la clase SerVivo. Esto asegura que el comportamiento de la clase padre se mantenga intacto y reutilizado.

Se reemplazaría la línea del método set por super().setNombre(nombre) y la del método get por return super().getNombre(), para que se actualice el atributo _nombre directamente en la clase SerVivo y que se recupere su valor desde la clase padre, respectivamente.

Con estas modificaciones no se perdería ningún valor definido en SerVivo ni se modificaría el comportamiento de Animal (se puede añadir código para acciones específica para esta subclase).

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 es heredado por las clases hijas (Animal, Persona, etc.) porque es un atributo de clase. Sin embargo, si se vuelve a definir en una clase hija (como en Animal o Pajaro), ocurriría una ocultación del atributo de la clase padre (cada clase hija tendrá su propio contador independiente de instancias, y no se estará actualizando el contador de la clase SerVivo).

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

Sí, los métodos getTotalCreados de las clases hijas sobrescriben al método de la clase padre SerVivo. Aunque el método getTotalCreados en la clase base (SerVivo) está marcado como un método de clase que devuelve el total de instancias creadas, aunque tengan el mismo nombre, al redefinirlo en las clases hijas, se crea una nueva versión de este método para cada clase hija (cada clase hija manejará su propio contador de instancias, en lugar de usar el contador de la clase base).

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

Los métodos que hereda una clase que hereda de la clase Persona son (los de su padre SerVivo y sus propios):

- *setNombre(nombre):* Establece el nombre de la persona.
- *getNombre():* Obtiene el nombre de la persona.
- *setEdad(edad):* Establece la edad de la persona.
- *getEdad():* Obtiene la edad de la persona.
- *getTotalCreados():* Obtiene el total de instancias creadas de Persona o sus subclases.
- *aduenarAnimal(x):* Permite a la persona hacerse cargo de un animal.

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

El método aduenarAnimal de la clase Persona puede recibir objetos de tipo Animal o de cualquiera de sus subclases, como Gato, Pajaro, o Perro. Pero, no se le puede pasar un objeto de tipo SerVivo directamente, porque el método está diseñado específicamente para trabajar con objetos de tipo Animal y sus derivados, no con instancias genéricas de SerVivo.

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

Para obtener la cantidad de seres vivos creados (objetos de la clase SerVivo o sus subclases), se puede acceder al atributo de clase _totalCreados de la clase SerVivo, es decir, SerVivo.getTotalCreados(). Este atributo cuenta las instancias de SerVivo y sus subclases y usando se devuelve el total de instancias de SerVivo y sus subclases creadas, ya que _totalCreados se incrementa cada vez que se crea una nueva instancia.

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?**

El método getRaza(self, tipo) definido en la clase Perro oculta al método getRaza de la clase Animal, ya que tiene el mismo nombre pero con una firma diferente (añade el parámetro tipo). Esto no es sobrecarga, porque la sobrecarga implica definir el mismo método con diferentes firmas, pero dentro de la misma clase. Tampoco es sobrescritura, ya que la clase Perro está cambiando la funcionalidad del método, pero con un parámetro adicional.