# Taller dos. Clases, objetos, atributos y métodos.

Programación orientada objetos

Simón Cuartas Rendón – C.C. 1.037.670.103
Estudiante de estadística

**0.** Crear un proyecto de *Eclipse* llamado *Taller 2.*
✓

**1.** Agregar al proyecto de *Eclipse* las clases *Animal, Familia* y *FamiliaAnimales.*
✓

**A.** ¿Cuántas clases se están definiendo en este ejercicio?
En este ejercicio se están definiendo **tres** clases diferentes: **Animal, Familia** y **FamiliaAnimales.**

**B.** ¿En cuál de las tres clases se define el programa principal? Corra el programa principal.
La clase que define el archivo principal es **FamiliaAnimales,** ya que es esta la que incluye el método `main`(marcada en un recuadro rojo en la siguiente imagen).

```
public class FamiliaAnimales {
 public static void main(String args[]) {
 Animal animal1 = new Animal();
 Animal.totalAnimales++;
 Animal animal2 = new Animal();
 Animal.totalAnimales++;
```

**C.** ¿Cuántos objetos de la clase **Animal** se están creando en la clase que define el programa principal?
En el programa **FamiliaAnimales** se están definiendo dos objetos de la clase **Animal:** `animal1` y `animal2` (marcadas en violeta en la imagen anterior), y al ejecutar el método `tenerHijo()` se está creando un nuevo objeto.

**D.** ¿Cuáles objetos se están creando de la clase **Animal** en la clase que define el programa principal?
En la clase que define al programa principal, **FamiliaAnimales,** se están creando dos objetos de la clase **Animal: `animal1`** y **`animal2`**, y con el método `tenerHijo()` se está creando al objeto de la clase Animal que tiene como atributo de nombre **`"Cebrallo."`**

**E.** ¿Cuáles atributos tiene la clase **Animal?**

La clase Animal posee cinco atributos: `nombre` (de tipo String), `género` (de tipo String), `peso` (de tipo double), `pareja` (de tipo Animal) y `totalAnimales` (de tipo int).

**F.** ¿Cuáles atributos de la clase animal están haciendo referencia a tipos primitivos?

Hay dos atributos en la clase Animal que son de tipo primitivo: `peso`, que es **double,** y `totalAniamles`, que es **int.**

**G.** ¿Cuáles atributos de la clase **Animal** están haciendo referencia a objetos?

Hay solo un atributo en la clase Animal es que de tipo referencia: `pareja`, que es de tipo Animal.

**H.** ¿Con qué valor se inicializa el atributo `pareja` de la clase **Animal?**

Como no se le está definiendo nada explícitamente al atributo `pareja`, este se está inicializando con el valor **`null`**.

**I.** ¿Cuál es el nombre que tienen los objetos `animal1` y `animal2` antes de la línea siete en la clase **FamiliaAnimales**?

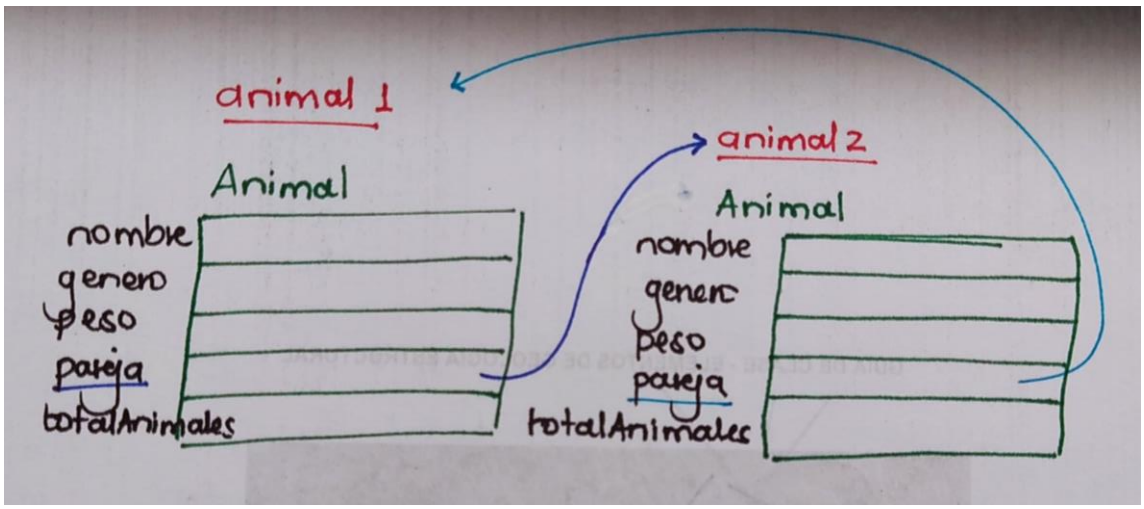
Al igual que en el caso anterior, como no se le está asignando ningún nombre o valor particular a los objetos `animal1` y `animal2`, Java les asigna **`null`** por defecto.

**J.** ¿Cuál es el peso de `animal1` en la clase **FamiliaAnimal?**

En ningún punto de la clase FamiliaAnimal se define explícitamente cuál es el valor del peso de `animal1`, por lo cual se le asigna un valor de cero (0.0) por defecto cuando se le asigna un espacio en la memoria.

**K.** Dibuje el estado de memoria luego de establecer como pareja del caballo a la cebra y viceversa.

En la imagen que está a continuación se ilustra el estado de memoria luego de realizar esta asignación particular del atributo **`pareja`**:



**L.** ¿Cuál es el género de animal2 en la clase **FamiliaAnimales?**
En ninguna parte de esta clase se está definiendo explícitamente el género de `animal2`, por lo que asume el valor que se le dio en clase **Animal,** esto es: **M.**

**M.** ¿Qué sucede si ocurren cada una de las siguientes situaciones?

*i.* *Se comenta la línea quince de la clase **FamiliaAnimales**.*
No se le va establecer a `animal1` que `animal2` es su pareja, lo cual implicaría que el método `procrear()` que es utilizado más adelante se ejecutaría en el primer condicional y saldría en pantalla `"Cebra no tiene pareja"`, y en consecuencia, no se forma una familia y el `totalAnimales` se conservaría en dos antes de la muerte de `animal2`.

*ii.* *Se comenta la línea dieciséis de la clase **FamiliaAnimales**.*
No se le va establecer a `animal2` que `animal1` es su pareja, pero aún así podría formarse una familia ya que previamente se estableció a `animal2` como pareja de `animal1`, siendo este el objeto utilizado para ejecutar el método `procrear()`.

*iii.* *Se comenta las líneas quince y dieciséis de la clase **FamiliaAnimales**.*
Ninguno de los dos animales va a tener pareja y por lo tanto no se puede formar una familia, similar al caso del literal ***i.***

**N.** ¿En el contexto de cuál objeto se está ejecutando el método `procrear` cuando es invocado en la línea dieciocho de la clase **FamiliaAnimales**?
El método `procrear()` se está ejecutando en el contexto del objeto **`animal1`,** por lo que se ejecuta el `else` de la línea 45 de la clase **Animal,** haciendo que `animal1` sea definido como el papá de *Cebrallo* y `animal2` sea la mamá, y es `animal2` quien ha tenido el hijo.

**O.** ¿Qué sucede si al atributo pareja de la clase Animal se le coloca el modificador final?
`pareja` es un atributo de tipo de referencia, por lo que una vez se le asigne un objeto, no se le podrá asignar ningún otro objeto, y como en este caso no estamos asignándole ningún objeto a este atributo, entonces asume `null` por defecto, y esto hará que no sea posible ejecutar los métodos `setPareja()` y `procrear()` más adelante, por lo que no se podrá formar una familia.

**P.** ¿Se puede eliminar el modificador static del método `morir` de la clase **Animal** sin que se afecte el funcionamiento del programa?
En un principio no es posible eliminar el `static` al método `morir`, pues al ejecutar el programa se va a generar un error en **FamiliaAnimales,** pues en una de las últimas líneas de esta clase se quiere registrar la muerte de `animal2`, para lo cual se debe acceder primero a la clase **Animal,** especificar que se desea ejecutar el método `morir` y luego indicar a cuál objeto de esta clase se le va a aplicar el método, y sin el `static` habría un error en la sentencia pues no es posible usar métodos de instancia (sin el `static`) en el contexto de una clase, sino que debe ser en el contexto de una propia instancia de la clase. Sin embargo,

para lograr que el programa funcione, podemos cambiar el acceso con cualquier objeto de la clase animal (por ejemplo, escribiendo `animal2.morir(animal2)`).

**Q.** ¿Qué sucede si se modifica la línea tres de la clase **Animal** como se indica a continuación?

**Línea original:** `String`
```
genero = "M";
```

**Nueva línea:** `static final String`
```
genero = "M";
```

Habría algunos errores señalados por el IDE, pues en el programa se están asignado géneros en diferentes momentos y esto no tendría sentido al usar el `static_final`, ya que inicialmente estamos obligando a todas las instancias de la clase a tener el mismo género siempre a través del `static` y junto con el `final` los estamos forzando a todas las instancias de la clase Animal a tener al género "M" como su atributo de género; en otras palabras, no sería posible tener objetos del tipo Animal con género "F", lo cual también haría imposible formar una familia, ya que esto requiere que sean de diferente género.

**R.** ¿Por qué no es necesario asignarle el valor inicial al atributo totalAnimales de la clase **Animal?** Explique.

En este caso, se tiene que Java va a asignarle el valor por defecto de cero, lo cual está bien considerando que es una variable que se utiliza como contadora en el programa, lo cual tiene sentido para que comience en cero con el fin de no distorsionar la cantidad total de animales que se tienen.

**S.** ¿Por qué razón, si se reemplaza la línea 28 de la clase **FamiliaAnimales** por la línea indicada a continuación el resultado se mantiene igual?

**Línea original:**
```
System.out.println("Nuevo total de animales: " + Animal.totalAnimales);
```

**Nueva línea:**
```
System.out.println("Nuevo total de animales: " + animal1.totalAnimales);
```

Esto es posible ya que el atributo `totalAnimales` es de tipo estático, por lo que puede ser accedido y modificado a través de clases o instancias de la clase produciendo el mismo cambio en cualquier escenario.

**T.** ¿Por qué razón no se afecta el resultado si se cambia la línea 27 de la clase **FamiliaAnimales** por la mostrada a continuación?

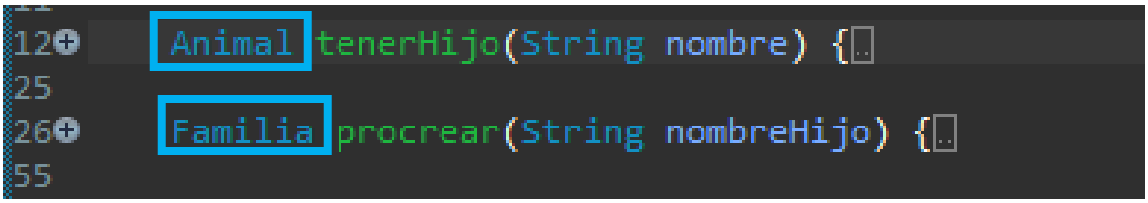
**Línea original:**
```
Animal.morir(animal2);
```

**Nueva línea:**
```
animal1.morir(animal2);
```

Semejante al caso anterior, como `morir` es un método **estático,** es posible emplearlo en el contexto de un objeto de su misma clase o a través de la misma clase; a la final, lo importante es que quede claro a cuál objeto le será aplicado el método y ambos casos se está realizando sobre `animal2`.

**U.** ¿Cuántos métodos tienes la clase **Animal**?
La clase **Animal** posee **cinco** métodos: `setPareja`, `tenerHijo`, `procrear`, `morir` y `toString`.

**V.** ¿Cuál es el tipo de retorno de los métodos `procrear()` y `tenerHijo()` de la clase **Animal**?
El método `tenerHijo()` retorna objetos de tipo **Animal,** mientras que `procrear()` los retorna de tipo de **Familia.** Esto se puede ver en la siguiente imagen:




**W.** ¿Al método `setPareja()` se le está pasando el parámetro por valor o por referencia?
Al método `setPareja()` se le está pasando el parámetro `pareja` por **referencia,** pues en este caso le estamos asignando un apuntador al parámetro pareja para realizar alguna modificación en esta zona de memoria.

**X.** ¿A quién está haciendo referencia la variable `this` en la línea 29 de la clase **Animal** cuando se ejecuta el programa principal? ¿Podría omitirse el uso de la variable `this` en este caso?
El parámetro `this` hace referencia al objeto de tipo **Animal** que se tiene como atributo `pareja`, y a ese línea se le podría retirar el `this`, ya que al invocar el objeto se tiene que referenciar a partir de cual objeto de tipo **Animal** se va a invocar al método `procrear()` para así poder extraer su pareja y los géneros según corresponda en el desarrollo de tal método.

**Y.** ¿Por qué no se afecta la ejecución del programa si se reemplaza la línea 38 de la clase **Animal** por la indicada a continuación?

**Línea original:**
```
System.out.println(nombre + " y " + pareja.nombre + " van a tener un
hijo");
```

**Nueva línea:**
```
System.out.println(this.nombre + " y " + this.pareja.nombre + " van a
tener un hijo");
```

En la línea original no es necesario usar el `this` porque cuando se vaya a invocar el método `procrear()`, se debe dejar claro con cuál objeto se va a realizar (por ejemplo,

`animal1.procrear("XXXX")` (donde `XXXX` representa el nombre del hijo), por lo que se entiende de manera inmediata a qué objeto se le va a obtener el nombre.

**Z.** ¿Podría eliminarse el modificador `static` del método `morir()` la clase **Animal** sin que se afecte el programa?

En un principio no es posible eliminar el `static` al método `morir`, pues al ejecutar el programa se va a generar un error en **FamiliaAnimales,** pues en una de las últimas líneas de esta clase se quiere registrar la muerte de `animal2`, para lo cual se debe acceder primero a la clase **Animal,** especificar que se desea ejecutar el método `morir` y luego indicar a cuál objeto de esta clase se le va a aplicar el método, y sin el `static` habría un error en la sentencia pues no es posible usar métodos de instancia (sin el `static`) en el contexto de una clase, sino que debe ser en el contexto de una propia instancia de la clase. Sin embargo, para lograr que el programa funcione, podemos cambiar el acceso con cualquier objeto de la clase animal (por ejemplo, escribiendo `animal2.morir(animal2)`).

**AA.** ¿A quién hace referencia la variable `this` en las líneas 44 y 46 de la clase **Animal** cuando se ejecuta el programa principal? ¿Por qué es necesario?

Se refiere al objeto de la clase **Animal** que se está usando como referencia para poder correr al método **procrear,** y es necesario ya que con este podemos referirnos de manera explícita a tal objeto. Se debe notar que, a diferencia de las situaciones por las cuales ya se han preguntado antes y no era necesario usar el `this`, en este caso estamos llamando al objeto en sí y no a uno de sus atributos (lo que permitía omitirlo en esos casos).

**BB.** Modifique el método `imprimir` de la clase **Familia** para que sea un método de la clase.

A continuación, se muestran y se resaltan las modificaciones hechas:

Asiento.java | Auto.java | Motor.java | Animal.java | Familia.java | FamiliaAnimales.java

```
1  public class Familia {
2      static Animal papa;
3      static Animal mama;
4      static Animal hijo;
5
6      static void imprimir() {
7          String genero;
8          if (hijo.genero.equals("M"))
9              genero = "masculino";
10         else
11             genero = "femenino";
12         System.out.println(papa.nombre + " es el papá, " + mama.nombre + " es la mamá, y " + hijo.nombre
13                 + " es el hijo de genero " + genero);
14     }
15 }
```

**CC.** ¿Por qué es útil que el atributo `totalAnimales` sea un atributo de clase y no un atributo de ejemplar?

Este atributo está siendo empleado en el programa para realizar el conteo de animales ha medida que estos son introducidos, bien sea de manera manual como nuevos objetos o a través del método `procrear()`, y también cuando son eliminados con el método `muerte()`. En este sentido, se hace útil este atributo sea estático para que pueda ser

llamado o invocado junto con la clase **Animal** y para que, al realizar cambios, suceda en general y no en particular para cada objeto.

**DD.** ¿Se puede colocar el modificar `static` al método `tenerHijo()` de la clase **Animal** sin que se afecte el funcionamiento del programa? Explicar.

Sí, ya que este método aún podría ser invocado por medio de una instancia por ser estático.

**EE.** ¿Qué hace el método `toString()` de la línea 59 de la clase **Animal**?

Este método básicamente está imprimiendo una cadena de caracteres (un String) con los atributos asociados al objeto de la **Animal** llamado `Cebrallo`.