

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**FACULTAD DE INGENIERÍA DE REDES Y COMUNICACIONES**

**Curso: PROGRAMACIÓN ORIENTADA A OBJETOS**

**Proyecto final**

**Docente:**

**Flores Orihuela, Carlos Alberto**

**Alumnos:**

**Casapia Quispe, Daniel Arturo**

**Chirinos Moreno, Erick Junior**

**Rincón Gómez, Yonin Jonathan**

**INDICE**

[**1.** **Resumen** 3](#_Toc20495854)

[**2.** **Introducción** 3](#_Toc20495855)

[**3.** **Diagrama de clases del modelo** 3](#_Toc20495856)

[**4.** **Diagrama de clases del modelo con Patrón de Diseño** 4](#_Toc20495857)

[**5.** **URL del Repositorio de Programa Fuente con Screens de commits realizados** 4](#_Toc20495858)

[**6.** **Resultado de pruebas unitarias** 5](#_Toc20495859)

[**7.** **Conclusiones:** 5](#_Toc20495860)

[**8.** **Recomendaciones:** 5](#_Toc20495861)

[**9.** **Glosario** 5](#_Toc20495862)

[**10.** **Evidencias del Trabajo en Equipo de cada Participante** 7](#_Toc20495863)

[**11.** **Bibliografía** 16](#_Toc20495864)

[**12.** **Objetivo del Estudiante (Student Outcome)** 16](#_Toc20495865)

# **Resumen**

La programación orientada a objetos nos ayuda a diseñar aplicaciones y programas, haciendo uso de paradigmas en forma secuencial. El objetivo de este proyecto es poner en práctica cada herramienta aprendida en clase y plasmarlo de la forma más asertiva para llegar a la solución.

# **Introducción**

En este proyecto se realizará un proceso (concurso de mascotas) donde los dueños, las mascotas y características son los datos brindados. Asimismo, se identifica los atributos, métodos y clases, lo que ayudará a realizar un proceso ágil y ordenado.

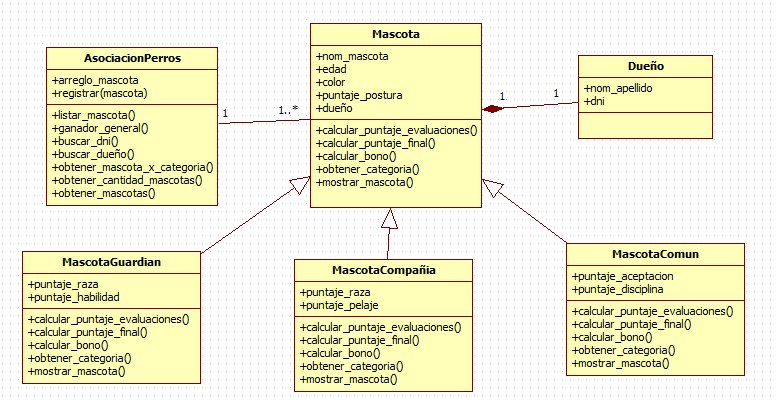
La gestión del manejo del Concurso de Mascotas se define de la siguiente manera:

* Con respecto a clases, nos enfocaremos en la clase Mascota, la cual estará constituida por los atributos y métodos que toda categoría (guardián, compañía y común) de perro va a manejar; con la clase Asociación, tendremos los métodos y crearemos un arreglo de mascota
* Aplicando herencia y polimorfismo, agilizaremos las subclases con referencia a las categorías de perros (guardianes, compañía y comunes).
* En la consulta de salida podremos imprimir la lista de mascotas con todos los atributos y calificación final de los perros con respecto a su categoría.
* El sistema nos permitirá controlar las validaciones de datos de entrada y el registro de ingreso de un DNI, para no ingresar otro dueño.
* Un reporte podrá dar detalle del perro ganador con su puntaje obtenido.

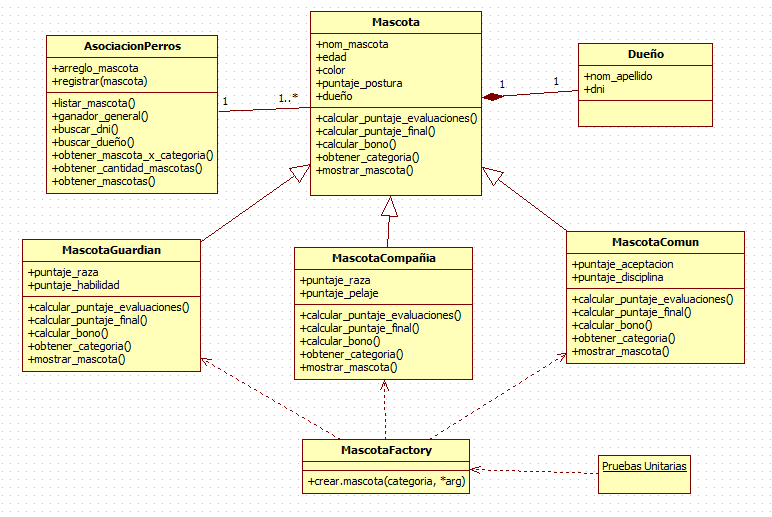
La Asociación de mascotas requiere lo siguiente requisitos de diseño:

* Desarrollar las clases y sus relaciones.
* Aplique las pruebas unitarias a todos sus métodos de negocio o comportamientos de sus clases.
* Aplicar el patrón de diseño Factory.
* Controlar con excepciones la validación de todos los datos de entrada.

# **Diagrama de clases del modelo**



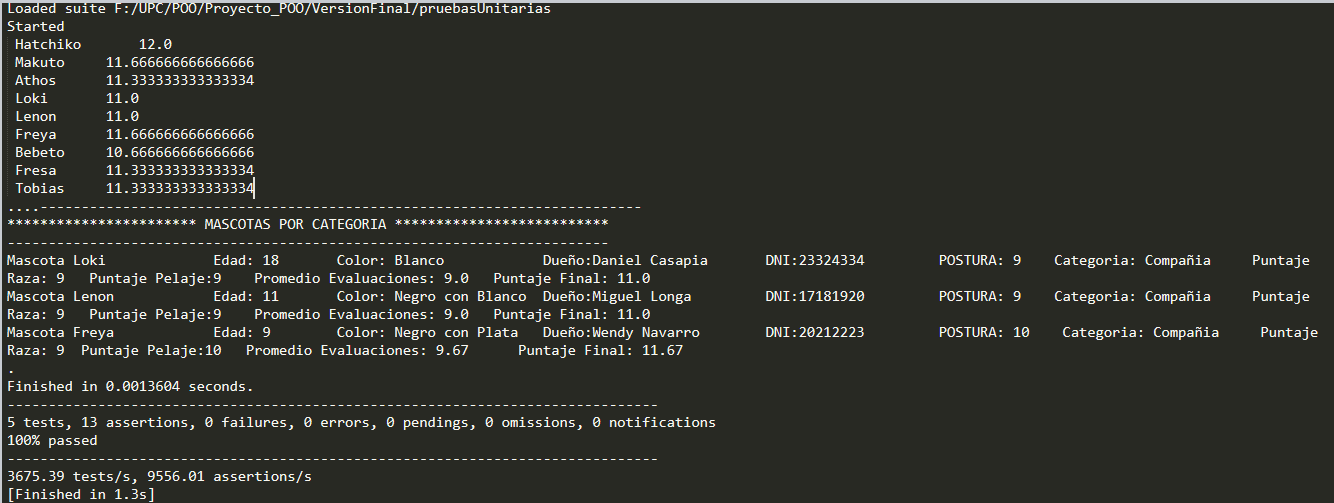
# **Diagrama de clases del modelo con Patrón de Diseño**



# **URL del Repositorio de Programa Fuente con Screens de commits realizados**

El programa fuente y los archivos (screenshots) se encuentra en el siguiente URL: https://github.com/jrincon2810/Proyecto\_POO

# **Resultado de pruebas unitarias**



# **Conclusiones:**

* Al aplicar herencia y polimorfismo, hacen que al crear las clases “hijas” (categoría Guardián, Compañía y Común), estas hereden todos los atributos y métodos (gracias al polimorfismo) ya existentes en la clase “padre”.
* Al realizar las pruebas unitarias comprobamos que nuestro código este en buen funcionamiento y al momento de cambiar el código mismo nos permita hacer pruebas sobre los cambios y saber que estos no tengan errores.

# **Recomendaciones:**

* Al crear una subclase (clase “hija”), tener en cuenta que al usar **super,** su fin de este es llamar a los atributos creados en la clase principal (class Mascota) y por ende no se le agrega los nuevos atributos de la subclase.

# **Glosario**

**Polimorfismo**. - La capacidad que tienen los objetos de comportarse de múltiples formas sin olvidar que para esto se requiere de la herencia, en si consiste en hacer referencia a objetos descendientes.

**Herencia**. -La herencia es el proceso por el cual un objeto adquiere las propiedades de otro.

**Clase Padre**. - Se refiere a la clase que es el ancestro más directo, así como a todas las clases ascendentes.

**Clase hija**. - Heredan el estado y el comportamiento en forma de las variables y los métodos de la clase padre.

**Atributos**. - Son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades.

**Variables de objeto**. - @nom\_mascota, @edad\_mascota, @color, las variables de objeto (o variables de instancia) sólo pueden ser leídas/escritas por el propio objeto.

**Variables de clase**. - nom\_mascota, edad\_mascota, color, las variables de clase no pertenecen a los objetos, si no a la clase. Existen únicamente en la clase, y son compartidas por todos los objetos.

**Símbolos**: :nom\_mascota, :edad\_mascota, :color, los símbolos son identificadores únicos que los encontraremos en varios sitios.

**Self**. - Es un término que se refiere al objeto que se crea al iniciar el intérprete.

**Break**. - Finaliza un while o un until loop, o un método dentro de un bloque.

**Def**. - Inicia la definición de un método; se cierra con end.

**Do**. - Comienza un bloque; se cierra con end.

**Else**. - Ejecuta el código que continua si la condición previa no es true. Funciona con if,

elsif, unless o case.

**Elsif**. - Ejecuta el código que continua si la condicional previa no es true. Funciona con if o elsif.

**End**. - Finaliza un bloque de código.

**For**. - Comienza un loop for. Se usa con in.

**If**. - Ejecuta un bloque de código si la declaración condicional es true. Se cierra con end.

**In**. - Usado con el loop for.

**Next**. - Salta al punto inmediatamente después de la evaluación del loop condicional

**Nil**. - Vacío, no inicializado, invalido. No es igual a cero.

**Return**. - Regresa un valor de un método o un bloque.

**Super**. - Llamada a método del mismo nombre en la superclase.

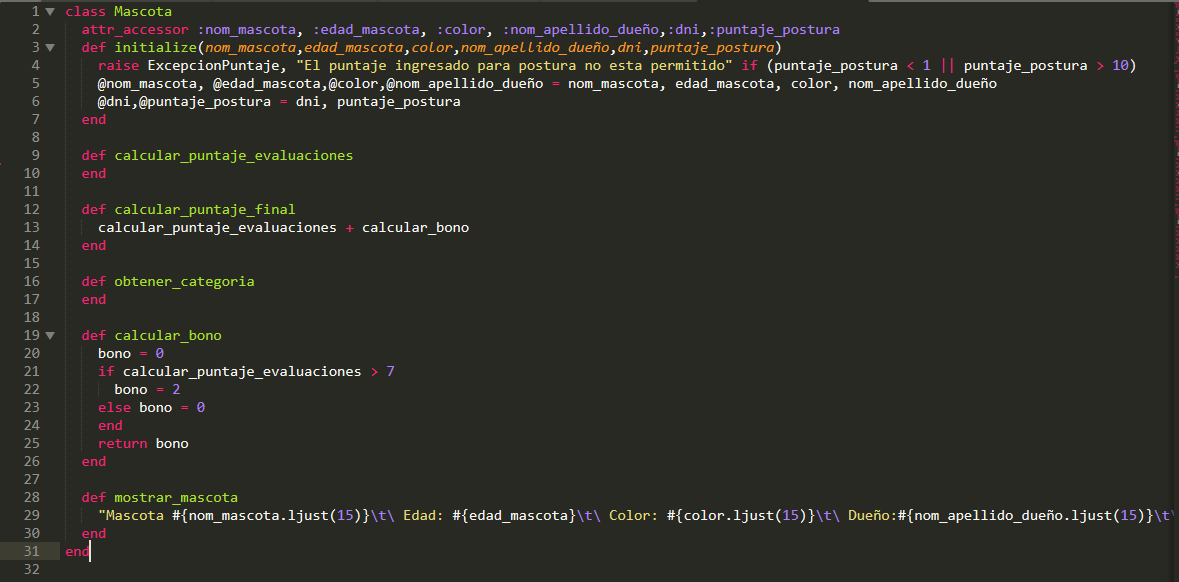
**When**. - Inicia una clausula debajo de under.

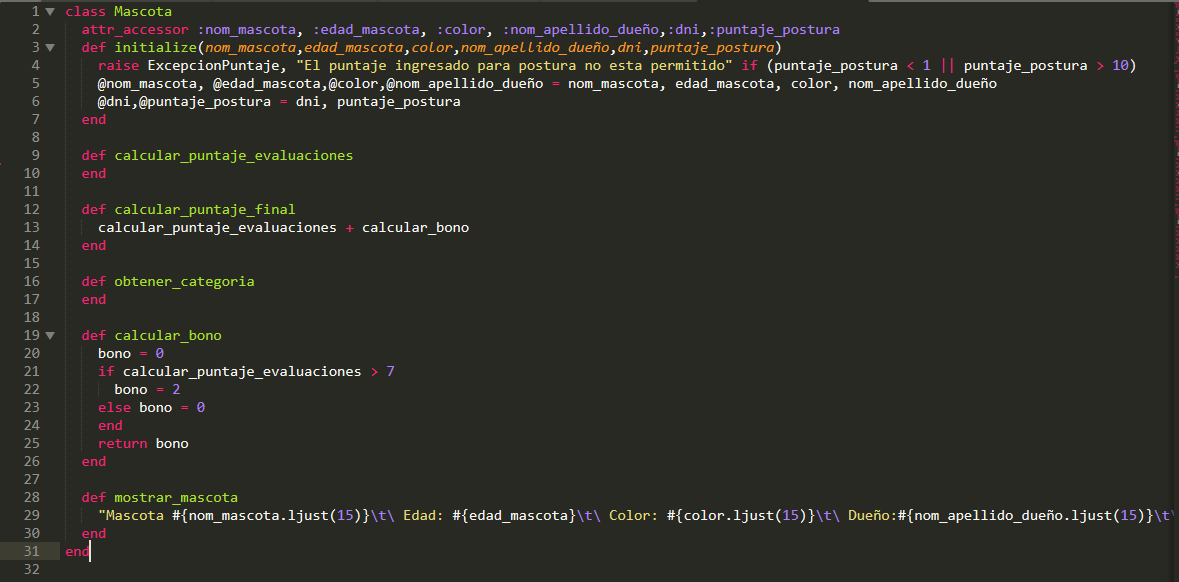
**Class**. - Define una clase; se cierra con end.

# **Evidencias del Trabajo en Equipo de cada Participante**

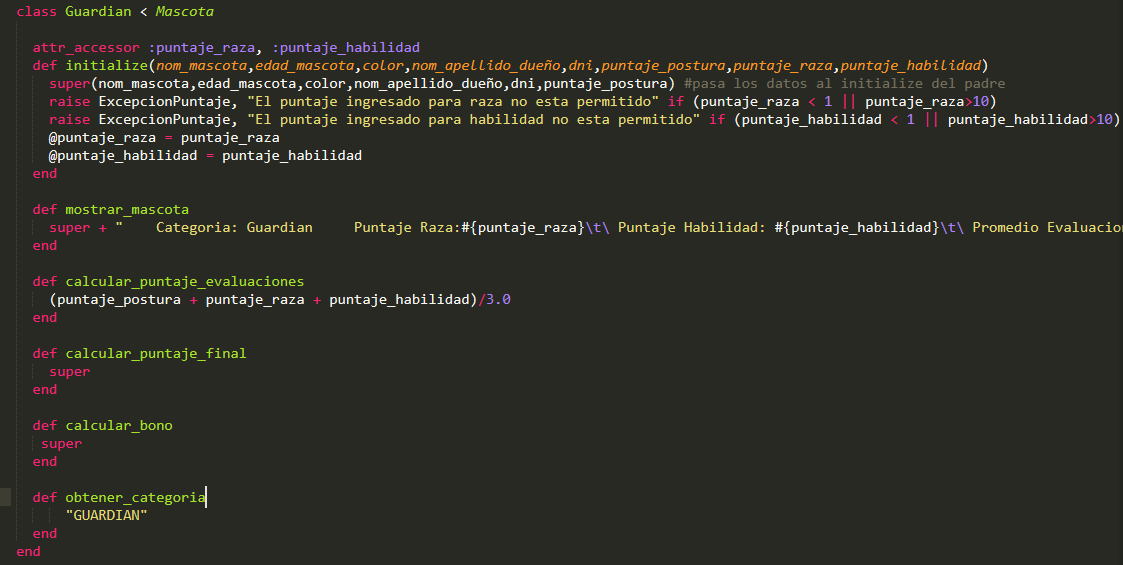
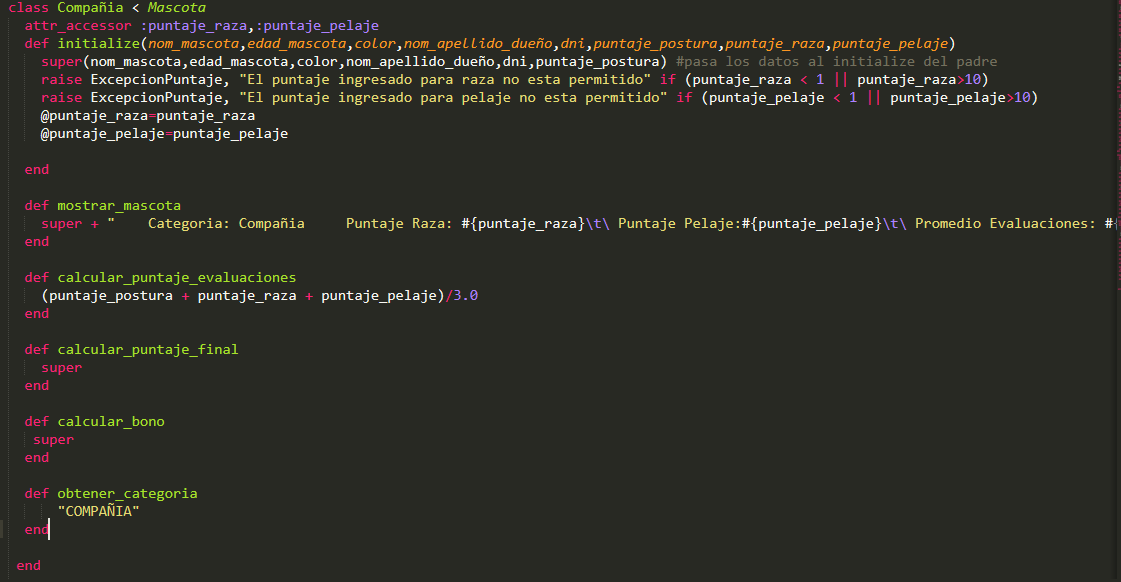
* 1. Avance del gráfico UML (realizado por Erick Chirinos)



* 1. Realización de la clase Mascota en conjunto con sus atributos y definiciones (realizado por Erick Chirinos)

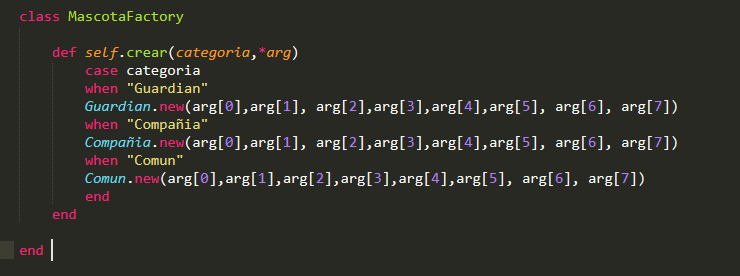


* 1. Realización de herencia y polimorfismo para cada categoría de perro (realizado por Erick Chirinos)

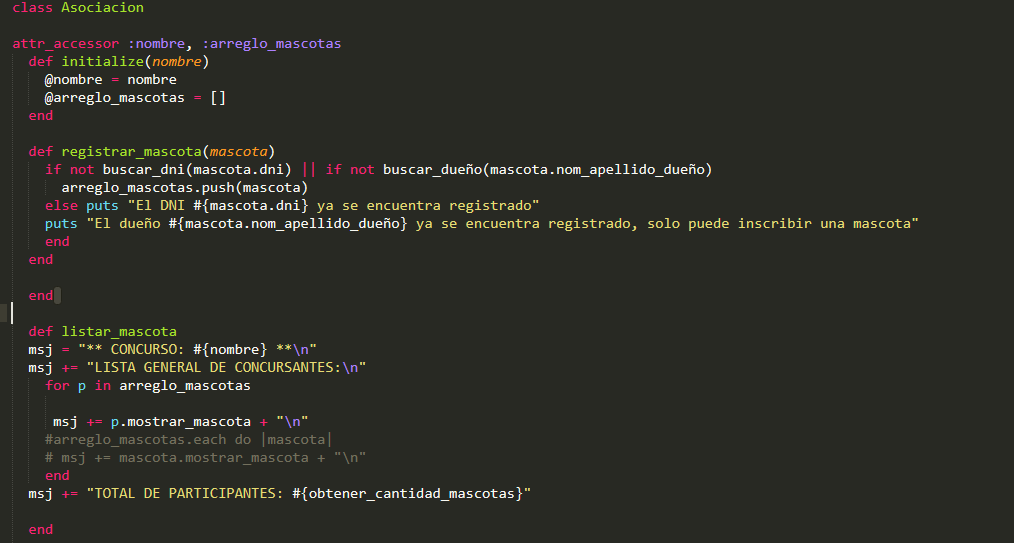


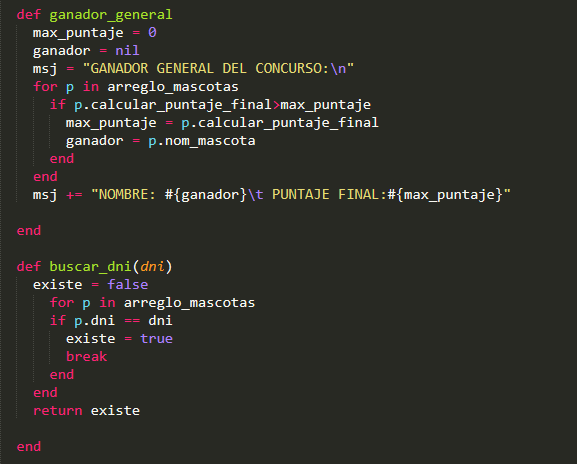


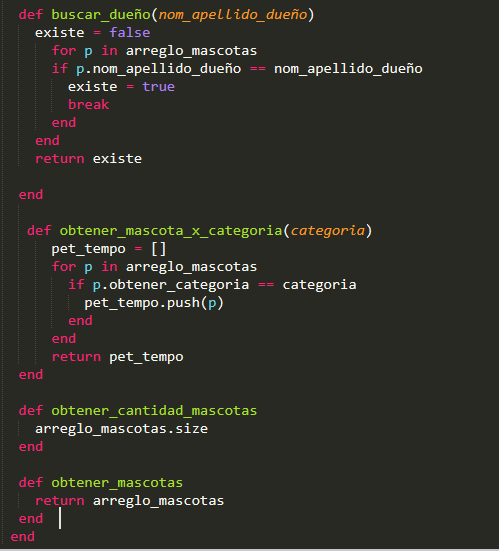
* 1. Realización de la clase MascotaFactory con la finalidad de poder almacenar cada categoría de la clase donde se hace la petición. (realizado por Erick Chirinos)



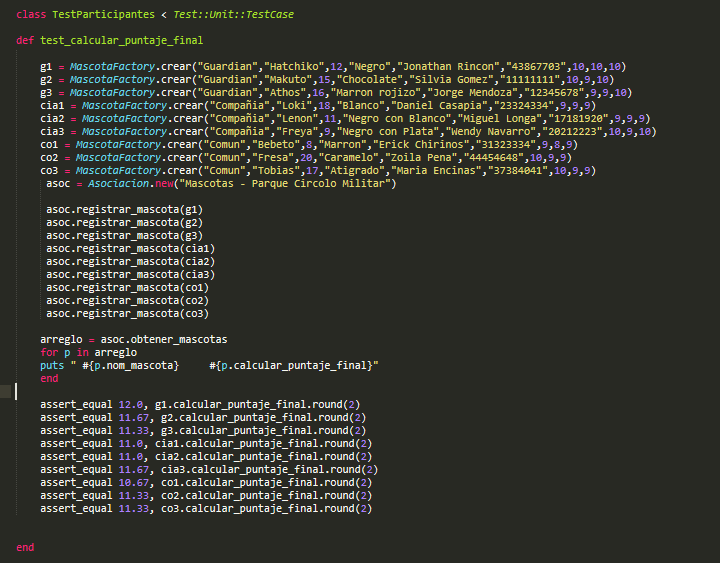
* 1. Realización de la clase Asociación con los siguientes métodos: listado de mascota, búsqueda de DNI, obtención de la mascota por categoría, la cantidad de mascotas (realizado por Daniel Casapia)

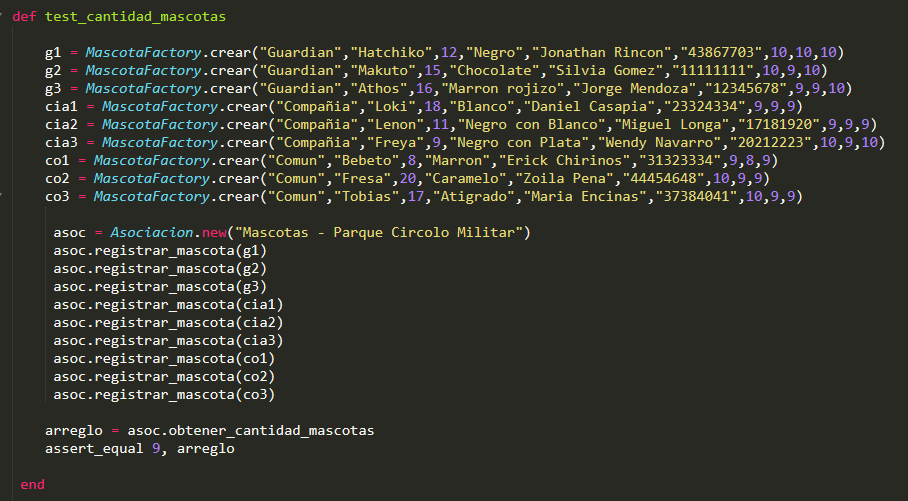


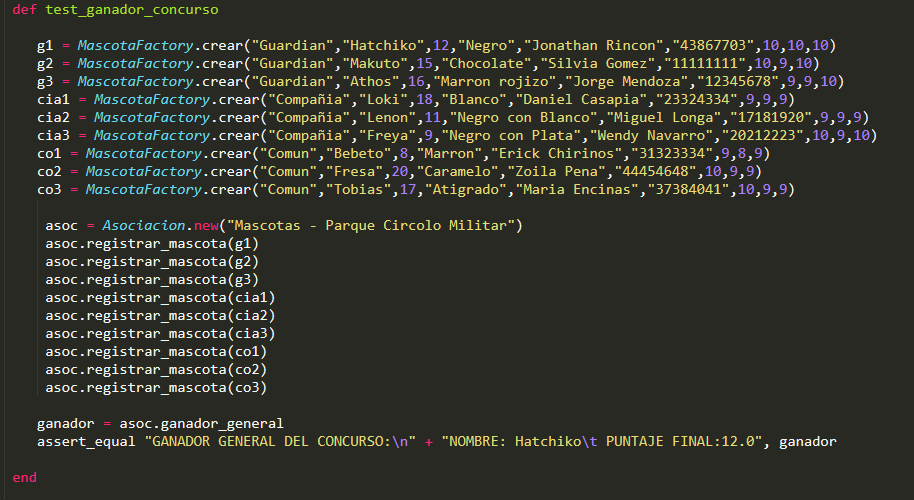


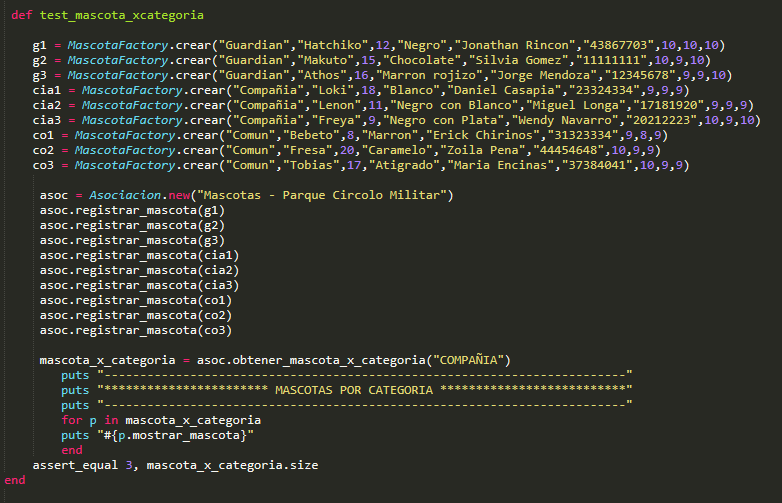


* 1. Realización de las pruebas unitarias del cálculo del puntaje final, de la cantidad de mascotas, ganador del concurso, la mascota por categoría, y la cantidad de mascota por categoría (realizado por Jonathan Rincón)





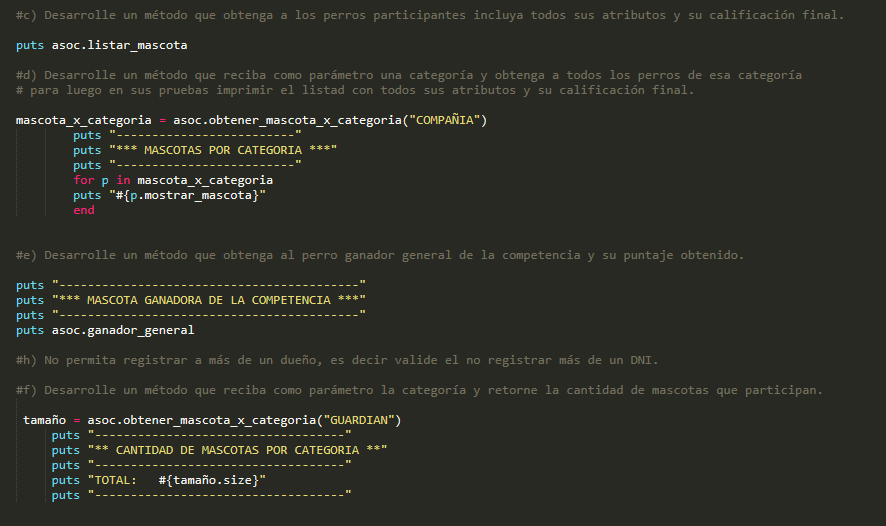






* 1. Se crea la impresión de los registros de todas las mascotas, calificación final, impresión de perros por categoría, ganador de la competencia con su puntaje final y el parámetro por categoría y su retorno a la cantidad de mascotas que participan (realizado por Jonathan Rincón).





# **Bibliografía**

En el presente trabajo se realizo las consultas a los siguientes links. Esto ayudo a reforzar los conceptos para poder aplicarlo en el proyecto. A continuación, se brinda el material usado:

<http://rubytutorial.wikidot.com/clases-objetos>

<https://ruby-doc.org/core-1.9.3/Array.html>

<http://rubytutorial.wikidot.com/excepciones>

<http://rubysur.org/aprende.a.programar/capitulos/metodos.html>

# **Objetivo del Estudiante (Student Outcome)**

El principal objetivo fue realizar el diseño y construcción del programa brindado en este proyecto con los fundamentos adecuados en RUBY. El material alcanzado en clase ayudo a realizarlo. Asimismo, las fuentes buscadas en internet ya sea de lectura o videos fueron de gran apoyo. Se aplico el uso de herencia, polimorfismo, patrones de diseño (Factory) y excepciones que ayudo a agilizar el desarrollo del programa. Uno de los puntos de inicio, en la programación, fue crear de manera correcta el diagrama de clases (StarUML) que nos dio la orientación necesaria para iniciar la estructura de código de manera sencilla. Asimismo, se programó los métodos de acuerdo a los requerimientos del proyecto. El llevar el curso de POO ayudara en nuestro desarrollo como profesional ya que entenderemos como funciona este paradigma de programación y como los códigos usados te facilitan el proceso de programar.