UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
Facultad de Ciencias Químicas e Ingeniería
Programas de Ingeniero en Computación e Ingeniero en Software y Tecnologías Emergentes

INFORMACIÓN DE LA MATERIA

Nombre de la materia y clave: Lenguajes de Programación Orientado a Objetos (40006). Grupo y periodo: 341 (2022-2) Profesor: Manuel Castañón Puga.

INFORMACIÓN DE LA ACTIVIDAD

Nombre de la actividad: Actividad de taller 4.2.1: Modelado de la cohesión y acoplamiento entre clases y objetos. Lugar y fecha: A 13 de noviembre de 2022 en el Edificio 6E, Salón 204. Carácter de la actividad: Individual Participante(es): Diego Andrés González Beltrán

REPORTE DE ACTIVIDADES

Objetivo de la actividad:
En esta actividad se tiene como objetivo el modelado de las relaciones entre clases utilizando polimorfismo, es decir, una clase que hereda de otra con los mismo métodos y atributos, sin embargo, la clase que hereda de la superclase puede ser creada como objeto de tipo inferior que apunta a una variable tipo de la superclase. Puede sonar confuso pero realmente es simple.

Feline implementará Animal y hereda de Organism, mientras que Cat implementa también Animal y hereda de Feline. En Feline, el método doAction imprime “Noise”, mientras que el doAction de Cat imprime “Meow!”. Al crear un objeto de tipo Cat que apunta a Feline se le conoce como polimorfismo ya que realmente no estamos haciendo un objeto de Cat que apunte a Cat, sin embargo esto es posible porque todo método y atributo tanto como Cat y en Feline permanecen, pero cambia la implementación del método del tipo de clase del objeto que se creó. Ejemplo: Feline cat = new Cat();

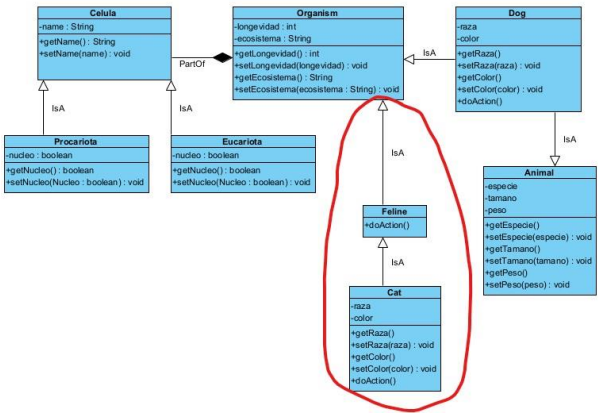

Figura 1. Diagrama actualizado con clase Cat que hereda de Feline e implementa de Animal

```
Color: Cafe
Especie: Felino
Tamano: 0
Peso: 0
Raza: Persa
Longevidad: 14 anios
Ecosistema: Terrestre

Meow!
```

Figura 2. Se puede observar que se imprime Meow! En vez de Noise, esto se debe a que se creó un objeto de tipo Cat y no Feline, sin embargo apunta a Feline porque tiene el mismo espacio de memoria que Cat por lo que lo hace compatible.

Ahora para C#, solo fue cuestión de utilizar las palabras reservadas virtual para el método doAction() en la clase Feline y override en doAction() de la clase Cat. Y creando objeto aplicando el concepto de Polimorfismo, Feline cat = new Cat().

```
        1 reference
 9      public virtual void doAction(){
10          Console.WriteLine("Noise");
11      }
        1 reference
12      public void setRaza(String raza){
13          this.raza = raza;
14      }
15
        0 references
16      public String getRaza(){
17          return raza;
18      }
19
        0 references
20      public string getColor()
21      {
22          return color;
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    JUPYTER

Name: Procariota
Existe nucleo? False
Name: Eucariota
Existe nucleo? True

Meow!
PS D:\Users\diego\Documents\Programming_Projects\LPOO\Cell\cellcs> dotnet run
Raza: Beagle
Color: Cafe y blanco
Especie: Perro
Tamano: 50
Peso: 45
Longevidad: 15 anios
Ecosistema: Terrestre
Name: Procariota
Existe nucleo? False
Name: Eucariota
Existe nucleo? True

Meow!
```

Figura 3. Prueba de código en C#

Feline.java > Feline

```java
public class Feline extends Organism implements Animal{
    private String raza;
    private String color;
    private String especie;
    private int tamano;
    private int peso;

    public Feline(String raza, String color, String especie, int tamano, int peso) {
        this.raza = raza;
        this.color = color;
        this.especie = especie;
        this.tamano = tamano;
        this.peso = peso;
    }

    public Feline(){}

    public String getEspecie() {
        return especie;
    }

    public void setEspecie(String especie) {
        this.especie = especie;
    }

    public int getTamano() {
        return tamano;
    }

```

Figura 4. Constructores

En cuanto el acoplamiento y cohesión, porque no existen métodos tan complicados ya que son setter y getters mayormente, decidí que debería de agregar constructores con parámetro y sin parámetro, esto para que el programador pueda decidir entre crear un objeto sin inicializar las variables de instancia y realizar esto con los setters, o bien inicializarlos ya creando el objeto. Considero que esto mejora el acoplamiento y cohesión del programa ya que ahora existen distintas formas de reutilizar el código.

Enlace del repositorio de GitHub:
https://github.com/DiegoAndresGlez/HDS

RESUMEN/REFLEXIÓN/CONCLUSIÓN

En esta actividad se utilizaron conceptos de relaciones entre clases de los talleres anteriores y ahora mejorando el acoplamiento y cohesión del programa.

Aprendí los conceptos de cohesión y acoplamiento para mejorar el código en cuanto su estructura y para su reutilización, eliminando además redundancias, etc.

Concluyó que la cohesión y acoplamiento de un programa se debe a que cualquier proyecto de software requiere de un código desarrollado de tal forma que sea mantenible, legible y reutilizable.

Doy fe de que toda la información dada es completa y correcta.	Diego Andres Gonzalez Beltran