Ingeniería en Sistemas Computacionales
PROGRAMACION ORIENTADA A OBJETOS
Oscar David Galvan Alvarez
TUE0096

PRACTICA#4 CLASES_GRAFICOS

En este reporte de practica de programación se expondrá lo realizado en la práctica #4 clases_grafico, así como también se incluirán imágenes del código realizado y lo que arrojan a la hora de ejecutarlos desglosando el código en 2 partes. La primera explicando las clases que se crearon mencionando sus atributos y sus funciones, así como también el cómo se enlazan entre sí. La segunda parte en la que se hablara del código principal que es el que ejecuta las clases y los métodos, así como también hace una gráfica comparativa utilizando la librería de matplotlib que ya se ha visto con anterioridad.

DEFINICION DE LAS CLASES, ATRIBUTOS Y METODOS

Para el inicio de esta practica primero se tubo que hacer un análisis sobre cuales serian las clases que se crearían y cuales serian sus atributos y métodos, teniendo primero una clase principal o clase padre de la cual se derivaran las clases hijo o subclases las cuales serán 4. (Este análisis previo se realizó en la 2da investigación). Las cuales son la CLASE_PADRE, CLASE_MOTOCICLETA, CLASE_SUPERMERCADO, CLASE_AUTOMOVIL y CLASE_DEPORTIVO.

CLASE_PADRE (vehículo)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - CLASE_PADRE.py

```python
#############Clase padre##########################
class vehiculo:
    def __init__(self, jugador, marca, modelo, velocidad_maxima, aceleracion, manejo):
        self.jugador = jugador
        self.marca = marca
        self.modelo = modelo
        self.velocidad_maxima = velocidad_maxima
        self.aceleracion = aceleracion
        self.manejo = manejo

    def acelerar(self):
        print(f"{self.jugador} acelera.")


    def frenar(self):
        print(f"{self.jugador} frena lentamente.")


    def direccionizq(self):
        print(f"{self.jugador} se mueve a la izquierda.")


    def direccionder(self):
        print(f"{self.jugador} se mueve a la derecha.")
```

Esta clase es la primera de todas y de la cual se derivarán las otras en este caso se le asignaron 6 atributos que serán los generales y con los cuales contarán las otras subclases. Primero asignamos el nombre de la clase y después pondremos sus atributos y los definiremos con un self el cual sirve para representar al propio objeto y este pueda ser llamado. Justo debajo de los atributos, definiremos los métodos que tendrá nuestra clase en este caso se le asignaron 4 métodos a la clase los cuales actúan como una función en si que solo arrojara algo cuando sean llamados cada uno mostrando un mensaje de acción diferente, y sin tener algún modificador de atributo en ellos.

“Subclase” CLASE_MOTOCICLETA (motocicleta)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - CLASE_MOTOCICLETA.py

```python
####################CLASE HIJO MOTOCICLETA##########################
import CLASE_PADRE as pad

class motocicleta(pad.vehiculo):
    def __init__ (self, jugador, marca, modelo, velocidad_maxima, aceleracion, manejo, casco, estabilidad):
        super().__init__(jugador, marca, modelo, velocidad_maxima, aceleracion, manejo)
        self.casco = casco
        self.estabilidad = estabilidad

    def caballito(self):
        print(f"{self.jugador} hace caballito con la motocileta para ganar velocidad.")
        MOTO.velocidad_maxima=MOTO.velocidad_maxima+MOTO.aceleracion

    def salto(self):
        print(f"{self.jugador} hace un salto con la motocileta por encima de un obstaculo.")
        MOTO.velocidad_maxima=MOTO.velocidad_maxima-10

MOTO=motocicleta("ruvik", "mortalica", "deportivo", 150, 20, "Cambios", "resistente","debil")
```

En esta subclase lo primero que se hizo fue importar la clase padre usando modularidad para que así pudiéramos utilizar sus atributos, una vez hecho esto definiremos la subclase como normalmente lo haríamos primero indicaremos un nombre a la clase indicando que se deriva de la clase vehículo, después definiremos sus atributos agregando los atributos de la clase vehículos y los atributos propios de la subclase, y con la función super() accederemos a los atributos de la clase padre sin referenciarlos de nuevo. Y después de ello añadiremos sus respectivos que so propios de la subclase y que imprimen un texto indicando una acción y haciendo que modifique un atributo cuando estos son ejecutados en este la velocidad máxima sobrescribiendo su valor al sumarle la aceleración. Justo al final de esta clase tenemos los valores que le dimos a nuestra clase motocicleta esto se encuentra aquí ya que al utilizar modularidad no se podía hacer un circulo de importaciones ya que si un archivo importa primero otro y una variable no está definida y se intenta importar después no podrá importarla y esto a su vez hará que después no pueda ejecutarse el código correctamente.

“Subclase” CLASE_SUPERMERCADO (carro_de_supermercado)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - CLASE_SUPERMERCADO.py

```python
#######################CLASE HIJO CARRO DE SUPERMERCADO###############
import CLASE_PADRE as pad

class carro_de_supermercado(pad.vehiculo):
    def __init__(self, jugador, marca, modelo, velocidad_maxima, aceleracion, manejo, capacidad_de_carga, debil):
        super().__init__(jugador, marca, modelo, velocidad_maxima, aceleracion, manejo)
        self.capacidad_de_carga = capacidad_de_carga
        self.debil = debil

    def lanzar_productos(self):
        print(f"{self.jugador} lanza una tortilla de harina.")
        SUP.capacidad_de_carga=SUP.capacidad_de_carga-1
        SUP.velocidad_maxima=SUP.velocidad_maxima+SUP.aceleracion

    def ruedas(self):
        print("Las ruedas del carrito rechinan")
        SUP.velocidad_maxima=SUP.velocidad_maxima-20

SUP=carro_de_supermercado("David","soriana","De metal",100,100,"Impulso",10,"resistente")
```

Para esta subclase se utiliza la misma estructura importamos la clase padre, definimos la clase con su nombre, indicamos cuales son los atributos generales y únicos, usamos la función super () para acceder a los atributos sin referenciarlos y después definimos los atributos únicos. Ahora bien, una vez hecho lo principal se asignan métodos diferentes que modificaran los respectivos valores de sus atributos de esta clase, modificando velocidad y un valor único el cual es la capacidad de carga la cual es necesaria al referirse a un carro de supermercado, y por último tenemos los valores asignados a cada atributo.

“Subclase” CLASE_AUTOMOVIL (automovil)

Utilizaremos la misma estructura importamos la clase padre, definimos la clase con su nombre, indicamos cuales son los atributos generales y únicos, usamos la función super () para acceder a los atributos sin referenciarlos y después definimos los atributos únicos que en este caso haremos que sean atributos los cuales tengan un valor que pueda cambiar con sus método correspondiente como lo son el combustible y el modificador de velocidad que se utilizaran para cambiar los atributos de velocidad y de combustible. Ahora bien, definimos nuestros 2 métodos indicando su nombre, después que imprima un texto de acción y a su vez modifique los valores de combustible y velocidad sobrescribiendo el mismo en los 2 métodos. Y por ultimo los valores de la clase.

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - CLASE_AUTOMOVIL.py

```python
##################CLASE AUTOMOVIL##############################
import CLASE_PADRE as pad

class automovil(pad.vehiculo):
    def __init__(self, jugador, marca, modelo, velocidad_maxima, aceleracion, manejo, combustible, ModificadorVel):
        super().__init__(jugador, marca, modelo, velocidad_maxima, aceleracion, manejo)
        self.combustible = combustible
        self.ModificadorVel = ModificadorVel

    def aceleracion_por_objeto(self):
        print(f"{self.jugador} obtuvo 10 de aceleracion al agarrar Modificador de Velocidad.")
        AUTO.combustible=AUTO.combustible-500
        AUTO.velocidad_maxima=AUTO.velocidad_maxima+AUTO.aceleracion

    def tomar_combustible(self):
        print(f"{self.jugador} obtuvo 200 de gasolina al agarrar Lata de combustible.")
        AUTO.combustible=AUTO.combustible+200

AUTO=automovil("Felipe","KIA","Estandar",150,10,"Estandar",3000,20)
```

“Subclase” CLASE_DEPORTIVO (automovil_deportivo)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - CLASE_DEPORTIVO.py

```python
#########################CLASE DEPORTIVO####################
import CLASE_PADRE as pad

class automovil_deportivo(pad.vehiculo):
    def __init__(self, jugador, marca, modelo, velocidad_maxima, aceleracion, manejo, modo_turbo, color):
        super().__init__(jugador, marca, modelo, velocidad_maxima, aceleracion, manejo)
        self.modo_turbo = modo_turbo
        self.color = color

    def activar_turbo(self):
        print(f"{self.jugador} activo el {self.modo_turbo}.")
        AUTODEP.velocidad_maxima=AUTODEP.velocidad_maxima+AUTODEP.aceleracion

    def drift(self):
        print(f"{self.jugador} hizo un drift.")
        AUTODEP.velocidad_maxima=AUTODEP.velocidad_maxima-50

    def pintura(self):
        print(f"{self.jugador} se metio a un taller de pintura.")
        AUTODEP.color="Verde"

AUTODEP=automovil_deportivo("Shadow","Ferrari","Deportivo",300,50,"Estandar","Oxido nitroso","Rojo")
```

Para esta subclase se utiliza la misma estructura importamos la clase padre, definimos la clase con su nombre, indicamos cuales son los atributos generales y únicos, usamos la función super () para acceder a los atributos sin referenciarlos y después definimos los atributos únicos y justo después de esto definimos nuestros métodos en este caso siendo 3 métodos los primeros 2 modificando la velocidad máxima en el caso del turbo aumentándola y en el caso del drift disminuyéndola y en el 3er método que modifica el color del auto. Y al final los valores de la clase.

IMPRESIÓN DE METODOS Y GRAFICA COMPARATIVA

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - PRINCIPAL.py

```python
import matplotlib.pyplot as plt
from CLASE_MOTOCICLETA import MOTO
from CLASE_SUPERMERCADO import SUP
from CLASE_AUTOMOVIL import AUTO
from CLASE_DEPORTIVO import AUTODEP
```

Primero en el archivo principal importaremos la librería matplotlib que sirve para graficar, así como también desde los archivos importar las variables que tienen almacenadas los valores de los atributos no es necesario importar el archivo completo.

Para mandar a llamar algún método o atributo que necesitemos ocupamos escribir lo siguiente:

ATRIBUTOS: nombre de la variable . nombre del atributo EJEMPLO: MOTO.velocidad_maxima

METODOS: nombre de la variable . el nombre del metodo() EJEMPLO: MOTO.acelerar()

CLASE MOTOCICLETA (VARIABLE: MOTO)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - PRINCIPAL.py

```python
print(MOTO.__class__.__name__)

print("Velocidad:",MOTO.velocidad_maxima)

MOTO.caballito()
print("Velocidad:",MOTO.velocidad_maxima)
MOTO.acelerar()
MOTO.direccionder()
MOTO.direccionizq()
MOTO.salto()
print("Velocidad:",MOTO.velocidad_maxima)
MOTO.frenar()
```

```
motocicleta
Velocidad: 150
ruvik hace caballito con la motocileta para ganar velocidad.
Velocidad: 170
ruvik acelera.
ruvik se mueve a la derecha.
ruvik se mueve a la izquierda.
ruvik hace un salto con la motocileta por encima de un obstaculo.
Velocidad: 160
ruvik frena lentamente.
```

CLASE CARRO_DE_SUPERMERCADO (VARIABLE: SUP)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - PRINCIPAL.py

```python
print(SUP.__class__.__name__)

print("Productos en el carrito:",SUP.capacidad_de_carga)
print("Velocidad:",SUP.velocidad_maxima)

SUP.lanzar_productos()
print("Productos en el carrito:",SUP.capacidad_de_carga)
print("Velocidad:",SUP.velocidad_maxima)
SUP.acelerar()
SUP.direccionder()
SUP.direccionizq()
SUP.ruedas()
print("Velocidad:",SUP.velocidad_maxima)
SUP.frenar()
```

```
carro_de_supermercado
Productos en el carrito: 10
Velocidad: 100
David lanza una tortilla de harina.
Productos en el carrito: 9
Velocidad: 200
David acelera.
David se mueve a la derecha.
David se mueve a la izquierda.
Las ruedas del carrito rechinan
Velocidad: 180
David frena lentamente.
--------------------------------------------------------------------------------
```

CLASE AUTOMOVIL (VARIABLE: AUTO)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - PRINCIPAL.py

```python
print(AUTO.__class__.__name__)

print("Velocidad:",AUTO.velocidad_maxima)
print("Combustible:",AUTO.combustible)

AUTO.aceleracion_por_objeto()
print("Velocidad:",AUTO.velocidad_maxima)
print("Combustible:",AUTO.combustible)
AUTO.direccionder()
AUTO.tomar_combustible()
print("Combustible:",AUTO.combustible)
AUTO.direccionizq()
AUTO.frenar()
print("Combustible:",AUTO.combustible)
```

```
automovil
Velocidad: 150
Combustible: 3000
Felipe obtuvo 10 de aceleracion al agarrar Modificador de Velocidad.
Velocidad: 160
Combustible: 2500
Felipe se mueve a la derecha.
Felipe obtuvo 200 de gasolina al agarrar Lata de combustible.
Combustible: 2700
Felipe se mueve a la izquierda.
Felipe frena lentamente.
Combustible: 2700
--------------------------------------------------------------------------------
```

CLASE AUTOMOVIL_DEPORTIVO (VARIABLE: AUTODEP)

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - PRINCIPAL.py

```python
print(AUTODEP.__class__.__name__)

print("Velocidad:",AUTODEP.velocidad_maxima)
print("Color:",AUTODEP.color)

AUTODEP.activar_turbo()
print("Velocidad:",AUTODEP.velocidad_maxima)
AUTODEP.acelerar()
AUTODEP.direccionder()
AUTODEP.drift()
print("Velocidad:",AUTODEP.velocidad_maxima)
AUTODEP.direccionizq()
AUTODEP.pintura()
print("Color:",AUTODEP.color)
AUTODEP.frenar()
```

```
automovil_deportivo
Velocidad: 300
Color: Rojo
Shadow activo el Oxido nitroso.
Velocidad: 350
Shadow acelera.
Shadow se mueve a la derecha.
Shadow hizo un drift.
Velocidad: 300
Shadow se mueve a la izquierda.
Shadow se metio a un taller de pintura.
Color: Verde
Shadow frena lentamente.
```

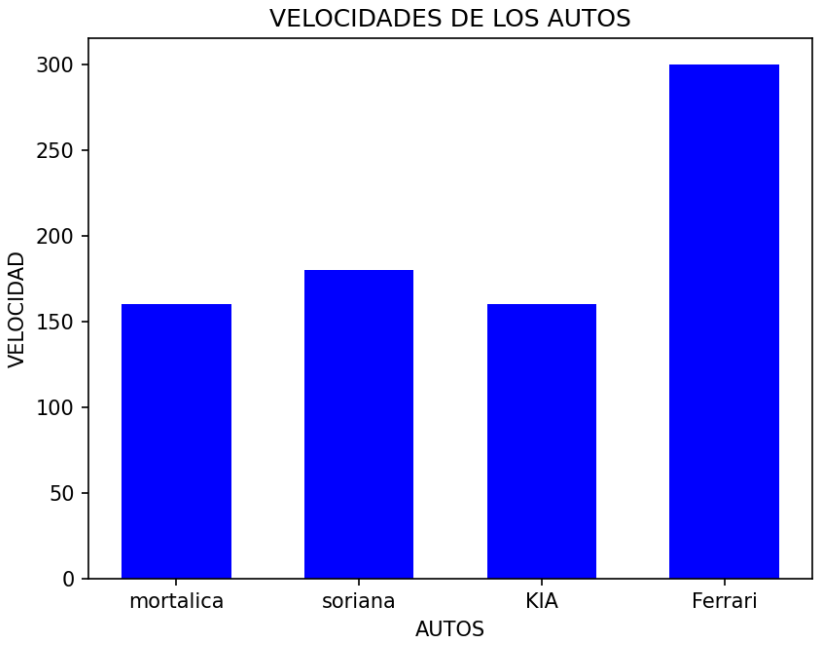
GRAFICO COMPARATIVO DE VELOCIDADES

CLASES (ATRIBUTOS Y METODOS) HERENCIA Y MODULARIDAD - PRINCIPAL.py

```python
##GRAFICA EN MATPLOTLIP
AUTOS=[MOTO,SUP,AUTO,AUTODEP]
nombre=[]
velocidad=[]
for i in AUTOS:
    NAME=i.marca
    nombre.append(NAME)
    VEL=i.velocidad_maxima
    velocidad.append(VEL)

plt.bar(nombre,velocidad,color="blue",width=0.6)
plt.title("VELOCIDADES DE LOS AUTOS")
plt.xlabel("AUTOS")
plt.ylabel("VELOCIDAD")
plt.show()
```

Para hacer este grafico importamos la librería matplotlib como plt para abreviarlo más fácilmente y no batallar a la hora de implementar alguna función. Ahora bien, para poder hacer una grafica de barras tenemos que extraer 2 valores las velocidades y los nombres de nuestros vehículos en este caso, para poder hacer eso se almacenaron primero las variables en una lista para así en un ciclo for poder recorrerla con cada iteración que da con cada dato de la lista. Para poder almacenar los resultados del ciclo for crearemos listas vacías almacenadas a variables las cuales son nombre y velocidad las cuales utilizaremos una vez contengan los datos para graficar al recorrer la lista y con un .append almacenar cada dato que se manda a llamar.



Para graficar usamos el plt que previamente indicamos para mandar a llamar las funciones que nos ayudaran a graficar en este caso el plt.bar que nos ayuda a tener una gráfica de barras añadiendo las listas con los datos que ocupamos indicándole un color y un ancho, así como también el titulo de nuestro grafico con plt.title y los nombres de el ejey y el ejex con plt.ylabel y plt.xlabel y por ultimo y lo más importante el plt.show que muestra la grafica que creamos.

CONCLUSIONES:

El crear este código al utilizar modularidad fue difícil ya que tiene sus limitaciones que puede solucionar correctamente para que el código se ejecutara bien, ahora bien, el hacer clases con sus atributos y métodos es complejo de hacer ya que tienes que pensar correctamente el cómo enlazar cada uno de ellos y elegir correctamente métodos que sean modificables sin que presente algún problema, aunque es difícil no es imposible ya que no es cuestión de saber como programarlo sino que más bien es cuestión de la lógica de como enlazarlo ya que sin lógica no podría obtenerse un resultado satisfactorio esto mismo sirviéndome como experiencia a la hora de hacer una práctica similar o diferente.

GITHUB: https://github.com/D4V1D216/PROGRAMACION-ORIENTADA-A-OBJETOS.git