

Actividad 2 grupal POO

Estudiantes

David Esteban Cartagena Mejia

C.C: 1001361568

Josué González Otálvaro

C.C: 1001236911

Juan Manuel Saldarriaga V.

C.C: 1001226798

Docente

Walter Hugo Arboleda Mazo

Asignatura

Programación Orientada a Objetos (POO)



Universidad Nacional de Colombia Sede Medellín
Mayo de 2025

Tabla de contenidos

Ejercicio 2.1	4
Ejercicio 2.2	7
Ejercicio 2.3	9
Ejercicio 2.4	12
Ejercicio 2.5	16

Listado de Figuras

1 Diagrama UML de la clase Persona	5
2 Planeta Tierra	7

Ejercicio 2.1

Se requiere un programa que modele el concepto de una persona. Una persona posee nombre, apellido, número de documento de identidad y año de nacimiento. La clase debe tener un constructor que inicialice los valores de sus respectivos atributos. La clase debe incluir los siguientes métodos:

Definir un método que imprima en pantalla los valores de los atributos del objeto.
En un método main se deben crear dos personas y mostrar los valores de sus atributos en pa-

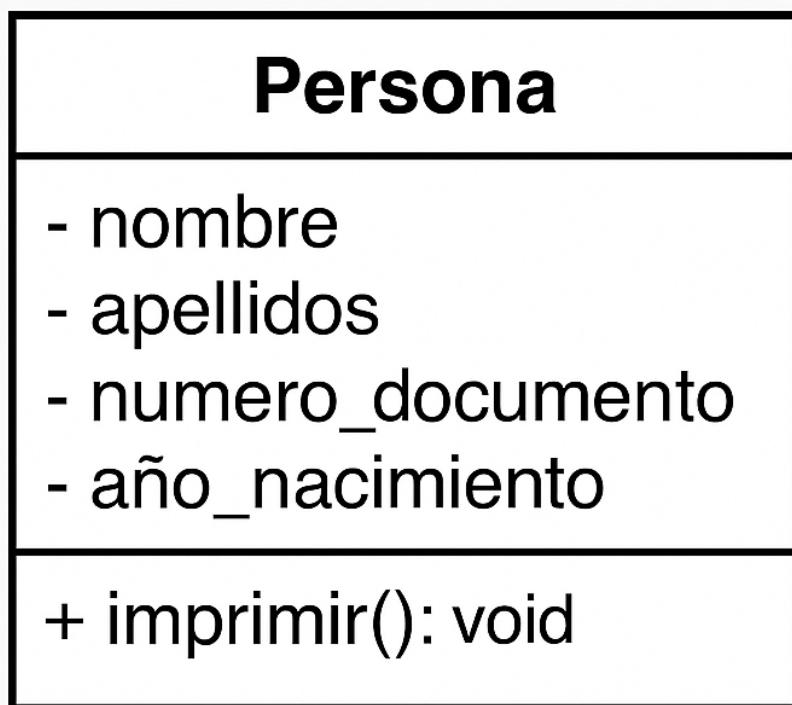


Figura 1: Diagrama UML de la clase Persona

```
class Persona:  
    def __init__(self, nombre,  
                 apellidos, numero_documento, ano_nacimiento):  
        self.nombre = nombre  
        self.apellidos = apellidos  
        self.numero_documento = numero_documento  
        self.ano_nacimiento = ano_nacimiento
```

```
def imprimir(self):
    print(f"Nombre = {self.nombre}")
    print(f"Apellidos = {self.apellidos}")
    print(f"Número de documento de identidad = {self.numero_documento}")
    print(f"Año de nacimiento = {self.ano_nacimiento}")
    print()

# Método main simulado
if __name__ == "__main__":
    p1 = Persona("Pedro", "Pérez", "1053121010", 1998)
    p2 = Persona("Luis", "León", "1053223344", 2001)

    p1.imprimir()
    p2.imprimir()
```

Nombre = Pedro
Apellidos = Pérez
Número de documento de identidad = 1053121010
Año de nacimiento = 1998

Nombre = Luis
Apellidos = León
Número de documento de identidad = 1053223344
Año de nacimiento = 2001

Ejercicio 2.2

Planeta

- nombre: Optional[str]
- cantidad_satelites: int
- masa: float
- volumen: float
- diametro_sol: float
- tipo: TipoPlaneta
- es_observable: bool

- + imprimir()
- + calcular_densidad(Optional)
- + es_planeta_exterior() bool

Figura 2: Planeta Tierra

```
from enum import Enum

class TipoPlaneta(Enum):
    GASEOSO = "GASEOSO"
```

```

TERRESTRE = "TERRESTRE"
ENANO = "ENANO"

class Planeta:
    def __init__(self, nombre=None, cantidad_satelites=0, masa=0, volumen=0,
                 diametro=0, distancia_sol=0, tipo=TipoPlaneta.TERRESTRE, es_observable=False):
        self.nombre = nombre
        self.cantidad_satelites = cantidad_satelites
        self.masa = masa
        self.volumen = volumen
        self.diametro = diametro
        self.distancia_sol = distancia_sol
        self.tipo = tipo
        self.es_observable = es_observable

    def imprimir(self):
        print(f"Nombre del planeta = {self.nombre}")
        print(f"Cantidad de satélites = {self.cantidad_satelites}")
        print(f"Masa del planeta = {self.masa}")
        print(f"Volumen del planeta = {self.volumen}")
        print(f"Diámetro del planeta = {self.diametro}")
        print(f"Distancia al Sol = {self.distancia_sol}")
        print(f"Tipo de planeta = {self.tipo.value}")
        print(f"Es observable = {self.es_observable}")

    def calcular_densidad(self):
        if self.volumen == 0:
            return None
        return self.masa / self.volumen

    def es_planeta_exterior(self):
        UA = 149_597_870 # km
        limite_exterior = 3.4 * UA
        return self.distancia_sol > limite_exterior

if __name__ == "__main__":
    tierra = Planeta("Tierra", 1, 5.9736e24, 1.08321e12, 12742, 150_000_000, TipoPlaneta.TIERRA)
    for planeta in [tierra]:
        planeta.imprimir()
        densidad = planeta.calcular_densidad()
        print(f"Densidad del planeta = {densidad:.2f} kg/km³" if densidad else "Densidad no calculada")
        print(f"Es planeta exterior = {planeta.es_planeta_exterior()}")

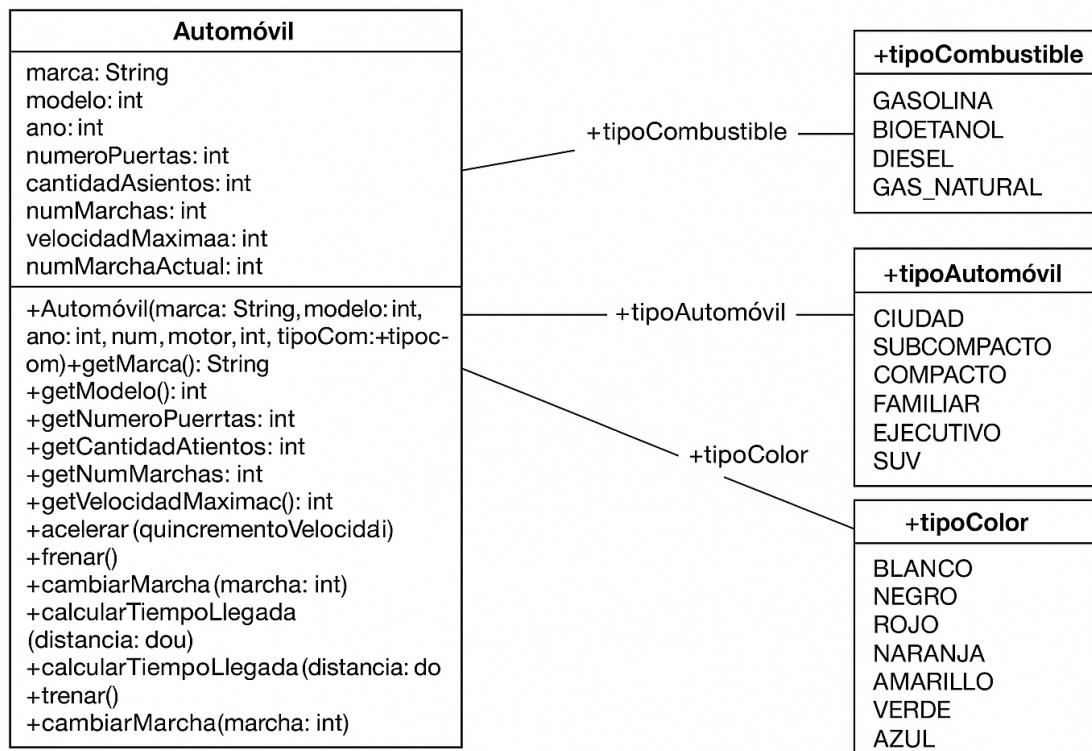
```

```
print()
```

```
Nombre del planeta = Tierra
Cantidad de satélites = 1
Masa del planeta = 5.9736e+24
Volumen del planeta = 1083210000000.0
Diámetro del planeta = 12742
Distancia al Sol = 150000000
Tipo de planeta = TERRESTRE
Es observable = True
Densidad del planeta = 5514720137369.48 kg/km3
Es planeta exterior = False
```

Ejercicio 2.3

Encontraremos en la parte inferior el desarrollo del código fuente para la resolución del ejercicio 2.3, se hizo uso del paquete Enum y se definió la clase Automóvil con sus distintos atributos y métodos, posterior a ello se hace uso del método main.



```

from enum import Enum

class TipoCombustible(Enum):
    GASOLINA = "Gasolina"
    BIOETANOL = "Bioetanol"
    DIESEL = "Diésel"
    BIODIESEL = "Biodiésel"
    GAS_NATURAL = "Gas Natural"

class TipoAutomovil(Enum):
    CIUDAD = "Ciudad"
    SUBCOMPACTO = "Subcompacto"
    COMPACTO = "Compacto"
    FAMILIAR = "Familiar"
    EJECUTIVO = "Ejecutivo"
    SUV = "SUV"

class TipoColor(Enum):
    BLANCO = "Blanco"
    NEGRO = "Negro"
    ROJO = "Rojo"
    NARANJA = "Naranja"
    AMARILLO = "Amarillo"
    VERDE = "Verde"
    AZUL = "Azul"
    VIOLETA = "Violeta"

class Automovil:
    def __init__(self, marca, modelo, motor, tipo_combustible, tipo_auto,
                 num_puertas, asientos, velocidad_max, color):
        self.marca = marca
        self.modelo = modelo
        self.motor = motor
        self.tipo_combustible = tipo_combustible
        self.tipo_auto = tipo_auto
        self.num_puertas = num_puertas
        self.asientos = asientos
        self.velocidad_max = velocidad_max
        self.color = color
        self.velocidad_actual = 0

    def acelerar(self, incremento):

```

```

        if self.velocidad_actual + incremento <= self.velocidad_max:
            self.velocidad_actual += incremento
        else:
            print("No se puede superar la velocidad máxima.")

    def desacelerar(self, decremento):
        if self.velocidad_actual - decremento >= 0:
            self.velocidad_actual -= decremento
        else:
            print("No se puede desacelerar a velocidad negativa.")

    def frenar(self):
        self.velocidad_actual = 0

    def calcular_tiempo_llegada(self, distancia):
        if self.velocidad_actual > 0:
            return distancia / self.velocidad_actual
        else:
            print("La velocidad actual es cero, no se puede calcular el tiempo.")
            return None

    def imprimir(self):
        print("Marca:", self.marca)
        print("Modelo:", self.modelo)
        print("Motor:", self.motor)
        print("Tipo de combustible:", self.tipo_combustible.value)
        print("Tipo de automóvil:", self.tipo_auto.value)
        print("Número de puertas:", self.num_puertas)
        print("Cantidad de asientos:", self.asientos)
        print("Velocidad máxima:", self.velocidad_max)
        print("Color:", self.color.value)

# --- MAIN ---
if __name__ == "__main__":
    auto1 = Automovil("Ford", 2018, 3, TipoCombustible.DIESEL, TipoAutomovil.EJECUTIVO,
                      5, 6, 250, TipоСolor.NEGRO)

    auto1.imprimir()
    auto1.velocidad_actual = 100
    print("Velocidad actual:", auto1.velocidad_actual)

```

```

auto1.acelerar(20)
print("Velocidad actual:", auto1.velocidad_actual)

auto1.desacelerar(50)
print("Velocidad actual:", auto1.velocidad_actual)

auto1.frenar()
print("Velocidad actual:", auto1.velocidad_actual)

auto1.desacelerar(20)

```

Marca: Ford
 Modelo: 2018
 Motor: 3
 Tipo de combustible: Diésel
 Tipo de automóvil: Ejecutivo
 Número de puertas: 5
 Cantidad de asientos: 6
 Velocidad máxima: 250
 Color: Negro
 Velocidad actual: 100
 Velocidad actual: 120
 Velocidad actual: 70
 Velocidad actual: 0
 No se puede desacelerar a velocidad negativa.

Ejercicio 2.4

Se requiere un programa que modele varias figuras geométricas: el círculo, el rectángulo, el cuadrado y el triángulo rectángulo.

El círculo tiene como atributo su radio en centímetros.

El rectángulo, su base y altura en centímetros.

El cuadrado, la longitud de sus lados en centímetros.

El triángulo, su base y altura en centímetros.

Se requieren métodos para determinar el área y el perímetro de cada figura geométrica. Además, para el triángulo rectángulo se requiere:

Un método que calcule la hipotenusa del rectángulo.

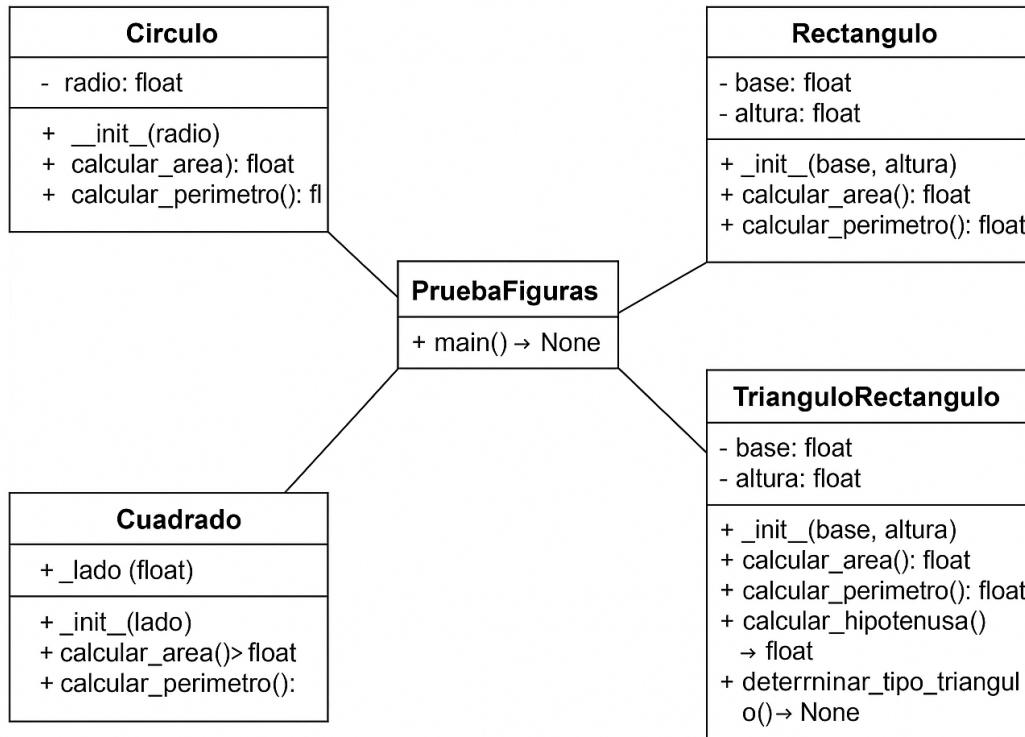
Un método para determinar qué tipo de triángulo es:

Equilátero: todos sus lados son iguales.

Isósceles: tiene dos lados iguales.

Escaleno: todos sus lados son diferentes.

Se debe desarrollar una clase de prueba con un método main para crear las cuatro figuras y probar los métodos respectivos.



```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio

    def calcular_area(self):
        return math.pi * math.pow(self.radio, 2)

    def calcular_perimetro(self):
```

```

        return 2 * math.pi * self.radio

class Cuadrado:
    def __init__(self, lado):
        self.lado = lado

    def calcular_area(self):
        return self.lado * self.lado

    def calcular_perimetro(self):
        return 4 * self.lado

class Rectangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def calcular_area(self):
        return self.base * self.altura

    def calcular_perimetro(self):
        return 2 * (self.base + self.altura)

class TrianguloRectangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def calcular_area(self):
        return (self.base * self.altura) / 2

    def calcular_hipotenusa(self):
        return math.sqrt(self.base ** 2 + self.altura ** 2)

    def calcular_perimetro(self):
        return self.base + self.altura + self.calcular_hipotenusa()

    def determinar_tipo_triangulo(self):
        hipotenusa = self.calcular_hipotenusa()

```

```

if self.base == self.altura == hipotenusa:
    print("Es un triángulo equilátero")
elif self.base != self.altura and self.base != hipotenusa and self.altura != hipotenusa:
    print("Es un triángulo escaleno")
else:
    print("Es un triángulo isósceles")

# Clase de prueba
if __name__ == "__main__":
    figura1 = Circulo(2)
    figura2 = Rectangulo(1, 2)
    figura3 = Cuadrado(3)
    figura4 = TrianguloRectangulo(3, 5)

    print("El área del círculo es =", figura1.calcular_area())
    print("El perímetro del círculo es =", figura1.calcular_perimetro())
    print()

    print("El área del rectángulo es =", figura2.calcular_area())
    print("El perímetro del rectángulo es =", figura2.calcular_perimetro())
    print()

    print("El área del cuadrado es =", figura3.calcular_area())
    print("El perímetro del cuadrado es =", figura3.calcular_perimetro())
    print()

    print("El área del triángulo es =", figura4.calcular_area())
    print("El perímetro del triángulo es =", figura4.calcular_perimetro())
    print("La hipotenusa del triángulo es =", figura4.calcular_hipotenusa())
    figura4.determinar_tipo_triangulo()

```

El área del círculo es = 12.566370614359172
 El perímetro del círculo es = 12.566370614359172

El área del rectángulo es = 2
 El perímetro del rectángulo es = 6

El área del cuadrado es = 9
 El perímetro del cuadrado es = 12

El área del triángulo es = 7.5

El perímetro del triángulo es = 13.8309518948453

La hipotenusa del triángulo es = 5.830951894845301

Es un triángulo escaleno

Ejercicio 2.5

Se requiere un programa que modele una cuenta bancaria que posee los siguientes atributos:

Nombres del titular.

Apellidos del titular.

Número de la cuenta bancaria.

Tipo de cuenta: puede ser una cuenta de ahorros o una cuenta corriente.

Saldo de la cuenta.

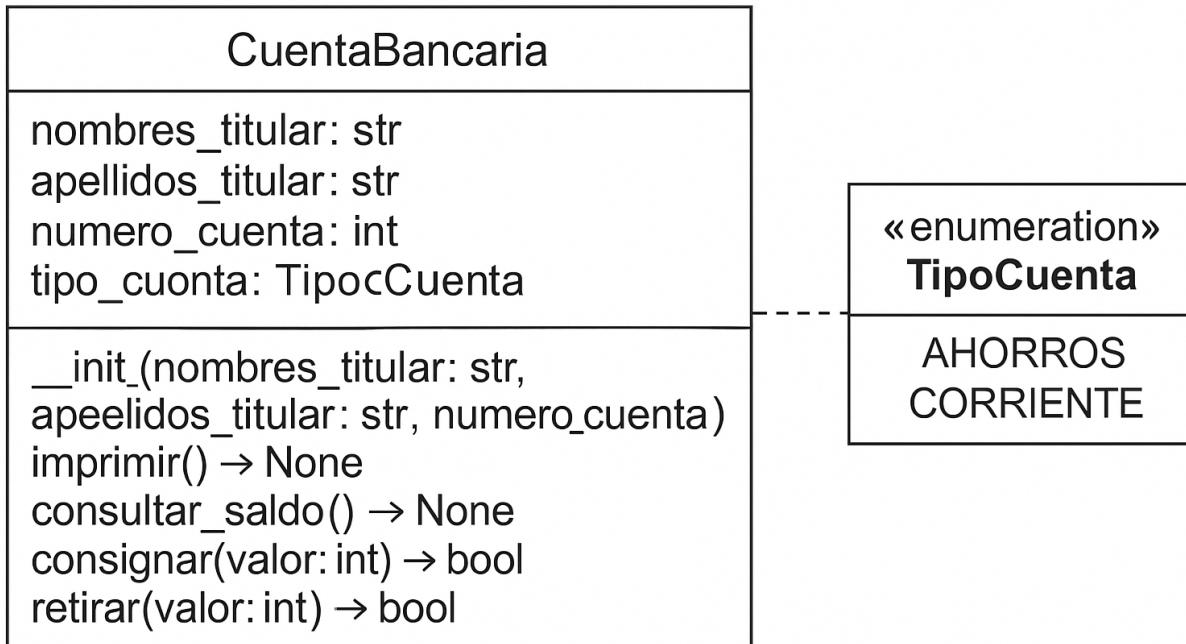
Se debe definir un constructor que inicialice los atributos de la clase. Cuando se crea una cuenta bancaria, su saldo inicial tiene un valor de cero. En una determinada cuenta bancaria se puede:

Imprimir por pantalla los valores de los atributos de una cuenta bancaria.

Consultar el saldo de una cuenta bancaria.

Consignar un determinado valor en la cuenta bancaria, actualizando el saldo correspondiente.

Retirar un determinado valor de la cuenta bancaria, actualizando el saldo correspondiente. Es necesario tener en cuenta que no se puede realizar el retiro si el valor solicitado supera el saldo actual de la cuenta



```

from enum import Enum

class TipocCuenta(Enum):
    AHORROS = "AHORROS"
    CORRIENTE = "CORRIENTE"

class CuentaBancaria:
    def __init__(self, nombres_titular, apellidos_titular, numero_cuenta, tipo_cuenta):
        self.nombres_titular = nombres_titular
        self.apellidos_titular = apellidos_titular
        self.numero_cuenta = numero_cuenta
        self.tipo_cuenta = tipo_cuenta
        self.saldo = 0

    def imprimir(self):
        print(f"Nombres del titular = {self.nombres_titular}")
        print(f"Apellidos del titular = {self.apellidos_titular}")
        print(f"Número de cuenta = {self.numero_cuenta}")
        print(f"Tipo de cuenta = {self.tipo_cuenta.value}")
        print(f"Saldo = {self.saldo}")

```

```

def consultar_saldo(self):
    print(f"El saldo actual es = {self.saldo}")

def consignar(self, valor):
    if valor > 0:
        self.saldo += valor
        print(f"Se ha consignado ${valor} en la cuenta. El nuevo saldo es ${self.saldo}")
        return True
    else:
        print("El valor a consignar debe ser mayor que cero.")
        return False

def retirar(self, valor):
    if valor > 0 and valor <= self.saldo:
        self.saldo -= valor
        print(f"Se ha retirado ${valor} en la cuenta. El nuevo saldo es ${self.saldo}")
        return True
    else:
        print("El valor a retirar debe ser menor que el saldo actual.")
        return False

if __name__ == "__main__":
    cuenta = CuentaBancaria("Pedro", "Perez", 123456789, TipoCuenta.AHORROS)
    cuenta.imprimir()
    cuenta.consignar(200000)
    cuenta.consignar(300000)
    cuenta.retirar(400000)

```

Nombres del titular = Pedro
 Apellidos del titular = Perez
 Número de cuenta = 123456789
 Tipo de cuenta = AHORROS
 Saldo = 0
 Se ha consignado \$200000 en la cuenta. El nuevo saldo es \$200000
 Se ha consignado \$300000 en la cuenta. El nuevo saldo es \$500000
 Se ha retirado \$400000 en la cuenta. El nuevo saldo es \$100000