UNIVERSIDAD
NACIONAL
DE COLOMBIA

ACTIVIDAD 3

Programación orientada a objetos

UNIVERSIDAD NACIONAL DE COLOMBIA
Profesor: Walter Arboleda

Daniel Felipe Zapata Patrón
Cc: 1000406800

PARTE 1

Código fuente:
https://github.com/danielzp2000/actividad3PooUnalMed/tree/660e38e25dcc489a73b455f4fb8b64a8c92cb91f/Parte%201

Capítlo 3: Estructura secuencial

Ejercicios propuestos

- 18. Se tiene la siguiente información de un empleado:

  - ✓ Código del empleado
  - ✓ Nombres
  - ✓ Número de horas trabajadas al mes
  - ✓ Valor hora trabajada
  - ✓ Porcentaje de retención en la fuente.

  Haga un algoritmo que muestre: código, nombres, salario bruto y salario neto.

Código:

```
class SalarioEmpleado:
    def __init__(self, horas, tarifa, retencion):
        self.horas_trabajadas = horas
        self.tarifa_hora = tarifa
        self.porcentaje_retencion = retencion

    def calcular_salario_bruto(self):
        bruto = self.horas_trabajadas * self.tarifa_hora
        return bruto

    def calcular_salario_neto(self):
        bruto = self.calcular_salario_bruto()
        neto = bruto - (self.porcentaje_retencion * bruto)
        return neto
```

Interfaz gráfica:

```
import tkinter as tk
from tkinter import messagebox

class VentanaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Salarios")

        # Campos de entrada
        tk.Label(self, text="Código Empleado:").grid(row=0, column=0, padx=10, pady=5)
        self.txt_codigo_empleado = tk.Entry(self)
        self.txt_codigo_empleado.grid(row=0, column=1, padx=10, pady=5)

        tk.Label(self, text="Nombre:").grid(row=1, column=0, padx=10, pady=5)
        self.txt_nombre = tk.Entry(self)
        self.txt_nombre.grid(row=1, column=1, padx=10, pady=5)

        tk.Label(self, text="Horas Trabajadas:").grid(row=2, column=0, padx=10, pady=5)
        self.txt_horas_trabajadas = tk.Entry(self)
        self.txt_horas_trabajadas.grid(row=2, column=1, padx=10, pady=5)

        tk.Label(self, text="Tarifa por Hora:").grid(row=3, column=0, padx=10, pady=5)
        self.txt_tarifa_hora = tk.Entry(self)
        self.txt_tarifa_hora.grid(row=3, column=1, padx=10, pady=5)

        tk.Label(self, text="Retención Fuente (%):").grid(row=4, column=0, padx=10,
pady=5)
        self.txt_retencion_fuente = tk.Entry(self)
```

```
        self.txt_retencion_fuente.grid(row=4, column=1, padx=10, pady=5)


        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=5, column=0, padx=10, pady=10)
        tk.Button(self, text="Borrar", command=self.borrar).grid(row=5, column=1, padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=6, column=0, columnspan=2, padx=10, pady=10)


        # Campos de salida
        tk.Label(self, text="Código:").grid(row=7, column=0, padx=10, pady=5)
        self.txt_codigo = tk.Entry(self, state="readonly")
        self.txt_codigo.grid(row=7, column=1, padx=10, pady=5)

        tk.Label(self, text="Nombre:").grid(row=8, column=0, padx=10, pady=5)
        self.txt_nombre_dos = tk.Entry(self, state="readonly")
        self.txt_nombre_dos.grid(row=8, column=1, padx=10, pady=5)

        tk.Label(self, text="Salario Bruto:").grid(row=9, column=0, padx=10, pady=5)
        self.txt_salario_bruto = tk.Entry(self, state="readonly")
        self.txt_salario_bruto.grid(row=9, column=1, padx=10, pady=5)

        tk.Label(self, text="Salario Neto:").grid(row=10, column=0, padx=10, pady=5)
        self.txt_salario_neto = tk.Entry(self, state="readonly")
        self.txt_salario_neto.grid(row=10, column=1, padx=10, pady=5)

    # Método para calcular los salarios
    def calcular(self):
```

```
try:
    codigo = self.txt_codigo_empleado.get()
    nombre = self.txt_nombre.get()
    horas = float(self.txt_horas_trabajadas.get())
    tarifa = float(self.txt_tarifa_hora.get())
    retencion = float(self.txt_retencion_fuente.get()) / 100

    empleado = SalarioEmpleado(horas,  tarifa, retencion)
    salario_bruto = empleado.calcular_salario_bruto()
    salario_neto = empleado.calcular_salario_neto()

    # Mostrar resultados en los campos de salida
    self.txt_codigo.config(state="normal")
    self.txt_codigo.delete(0, tk.END)
    self.txt_codigo.insert(0, codigo)
    self.txt_codigo.config(state="readonly")

    self.txt_nombre_dos.config(state="normal")
    self.txt_nombre_dos.delete(0, tk.END)
    self.txt_nombre_dos.insert(0, nombre)
    self.txt_nombre_dos.config(state="readonly")

    self.txt_salario_bruto.config(state="normal")
    self.txt_salario_bruto.delete(0, tk.END)
    self.txt_salario_bruto.insert(0, f"{salario_bruto:.2f}")
    self.txt_salario_bruto.config(state="readonly")

    self.txt_salario_neto.config(state="normal")
```

```
            self.txt_salario_neto.delete(0, tk.END)
            self.txt_salario_neto.insert(0, f"{salario_neto:.2f}")
            self.txt_salario_neto.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa valores válidos.")

    # Método para borrar los campos de entrada y salida
    def borrar(self):
        self.txt_codigo_empleado.delete(0, tk.END)
        self.txt_nombre.delete(0, tk.END)
        self.txt_horas_trabajadas.delete(0, tk.END)
        self.txt_tarifa_hora.delete(0, tk.END)
        self.txt_retencion_fuente.delete(0, tk.END)

        self.txt_codigo.config(state="normal")
        self.txt_codigo.delete(0, tk.END)
        self.txt_codigo.config(state="readonly")

        self.txt_nombre_dos.config(state="normal")
        self.txt_nombre_dos.delete(0, tk.END)
        self.txt_nombre_dos.config(state="readonly")

        self.txt_salario_bruto.config(state="normal")
        self.txt_salario_bruto.delete(0, tk.END)
        self.txt_salario_bruto.config(state="readonly")

        self.txt_salario_neto.config(state="normal")
        self.txt_salario_neto.delete(0, tk.END)
```

```
        self.txt_salario_neto.config(state="readonly")


    # Método para salir de la aplicación
    def salir(self):
        self.destroy()


if __name__ == "__main__":
    app = VentanaPrincipal()
    app.mainloop()
```

- 19. Dado el valor del lado en un triángulo equilátero, haga un algoritmo que obtenga el perímetro, el valor de la altura y el área del triángulo.

Código:

```
import math


class Pitagoras:
    def __init__(self, longitud):
        self.longitud = longitud


    def calcular_ultimo_lado(self):
        medio_lado = self.longitud / 2
        lado = math.sqrt(self.longitud**2 - medio_lado**2)
        return lado


class Operaciones:
    def __init__(self, longitud):
        self.longitud = longitud


    def calcular_area(self):
        pitagoras = Pitagoras(self.longitud)
```

```
        altura = pitagoras.calcular_ultimo_lado()
        area = self.longitud * altura / 2
        return area

    def calcular_perimetro(self):
        perimetro = self.longitud * 3
        return perimetro
```

INTERFAZ GRÁFICA:

```
import tkinter as tk
from tkinter import messagebox

class Ventana(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Triángulo Equilátero")

        # Campo para ingresar la longitud
        tk.Label(self, text="Longitud del lado:").grid(row=0, column=0, padx=10,
pady=10)
        self.txt_longitud = tk.Entry(self)
        self.txt_longitud.grid(row=0, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=1, column=0,
padx=10, pady=10)
```

```
        tk.Button(self, text="Borrar", command=self.borrar).grid(row=1, column=1, padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=2, column=0, columnspan=2, padx=10, pady=10)

        # Campos de salida para área y perímetro
        tk.Label(self, text="Área:").grid(row=3, column=0, padx=10, pady=10)
        self.txt_area = tk.Entry(self, state="readonly")
        self.txt_area.grid(row=3, column=1, padx=10, pady=10)

        tk.Label(self, text="Perímetro:").grid(row=4, column=0, padx=10, pady=10)
        self.txt_perimetro = tk.Entry(self, state="readonly")
        self.txt_perimetro.grid(row=4, column=1, padx=10, pady=10)

    def calcular(self):
        try:
            longitud = float(self.txt_longitud.get())
            operacion = Operaciones(longitud)
            area = operacion.calcular_area()
            perimetro = operacion.calcular_perimetro()

            # Mostrar resultados
            self.txt_area.config(state="normal")
            self.txt_area.delete(0, tk.END)
            self.txt_area.insert(0, f"{area:.2f}")
            self.txt_area.config(state="readonly")

            self.txt_perimetro.config(state="normal")
            self.txt_perimetro.delete(0, tk.END)
```

```
            self.txt_perimetro.insert(0, f"{perimetro:.2f}")
            self.txt_perimetro.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa un valor numérico válido.")

    def borrar(self):
        self.txt_longitud.delete(0, tk.END)
        self.txt_area.config(state="normal")
        self.txt_area.delete(0, tk.END)
        self.txt_area.config(state="readonly")

        self.txt_perimetro.config(state="normal")
        self.txt_perimetro.delete(0, tk.END)
        self.txt_perimetro.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = Ventana()
    app.mainloop()
```

Capítulo 4: Estructura decisión lógica.

Ejercicios resueltos

- 7. Hacer un algoritmo que, dados dos valores numéricos A y B, escriba un mensaje diciendo si A es mayor, menor o igual a B.

Código:

```
import tkinter as tk
from tkinter import messagebox
```

```
# Clase que realiza la comparación entre dos números
class ComparacionValores:
    def __init__(self, numero_a, numero_b):
        self.numero_a = numero_a
        self.numero_b = numero_b

    def determinar_relacion(self):
        if self.numero_a > self.numero_b:
            return "A es mayor que B"
        elif self.numero_a < self.numero_b:
            return "A es menor que B"
        else:
            return "A es igual a B"

# Ventana principal con Tkinter
class Ventana(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Comparación de Valores")

        # Etiquetas y campos de entrada
        tk.Label(self, text="Número A:").grid(row=0, column=0, padx=10, pady=10)
        self.txt_numero_a = tk.Entry(self)
        self.txt_numero_a.grid(row=0, column=1, padx=10, pady=10)

        tk.Label(self, text="Número B:").grid(row=1, column=0, padx=10, pady=10)
        self.txt_numero_b = tk.Entry(self)
```

```
        self.txt_numero_b.grid(row=1, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=2, column=0, padx=10, pady=10)
        tk.Button(self, text="Borrar", command=self.borrar).grid(row=2, column=1, padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=3, column=0, columnspan=2, padx=10, pady=10)

        # Campo para mostrar el resultado
        self.txt_resultado = tk.Entry(self, state="readonly")
        self.txt_resultado.grid(row=4, column=0, columnspan=2, padx=10, pady=10)

    # Método para calcular la relación
    def calcular(self):
        try:
            numero_a = float(self.txt_numero_a.get())
            numero_b = float(self.txt_numero_b.get())
            comparacion = ComparacionValores(numero_a, numero_b)
            resultado = comparacion.determinar_relacion()
            self.txt_resultado.config(state="normal")
            self.txt_resultado.delete(0, tk.END)
            self.txt_resultado.insert(0, resultado)
            self.txt_resultado.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error", "Por favor ingresa números válidos.")

    # Método para borrar los campos de texto
```

```
    def borrar(self):
        self.txt_numero_a.delete(0, tk.END)
        self.txt_numero_b.delete(0, tk.END)
        self.txt_resultado.config(state="normal")
        self.txt_resultado.delete(0, tk.END)
        self.txt_resultado.config(state="readonly")

    # Método para salir de la aplicación
    def salir(self):
        self.destroy()

# Punto de entrada principal
if __name__ == "__main__":
    app = Ventana()
    app.mainloop()
```

- 10. Cierta universidad para liquidar el pago de matrícula de un estudiante le exige los siguientes datos:
  - ✓ Número de inscripción
  - ✓ Nombres
  - ✓ Patrimonio
  - ✓ Estrato social

  La universidad cobra un valor constante para cada estudiante de $50.000. Si el patrimonio es mayor que $2'000.000 y el estrato superior a 3, se le incrementa un porcentaje del 3% sobre el patrimonio. Hacer un algoritmo que muestre:

  - ✓ Número de inscripción
  - ✓ Nombres
  - ✓ Pago de matrícula

  Código:

```
class CalculoMatricula:
    def __init__(self, patrimonio, estrato):
        self.patrimonio = patrimonio
        self.estrato = estrato
        self.costo_matricula = 50000  # Costo base de la matrícula
```

```
    def calcular_pago_matricula(self):
        if self.patrimonio > 2000000 and self.estrato > 3:
            self.costo_matricula += 0.03 * self.patrimonio
        return self.costo_matricula
import tkinter as tk
from tkinter import messagebox

class Ventana(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Matrícula")

        # Etiquetas y campos de entrada
        tk.Label(self, text="Número de Inscripción:").grid(row=0, column=0,
padx=10, pady=10)
        self.txt_num_inscripcion = tk.Entry(self)
        self.txt_num_inscripcion.grid(row=0, column=1, padx=10, pady=10)

        tk.Label(self, text="Nombres:").grid(row=1, column=0, padx=10, pady=10)
        self.txt_nombres = tk.Entry(self)
        self.txt_nombres.grid(row=1, column=1, padx=10, pady=10)

        tk.Label(self, text="Patrimonio:").grid(row=2, column=0, padx=10, pady=10)
        self.txt_patrimonio = tk.Entry(self)
        self.txt_patrimonio.grid(row=2, column=1, padx=10, pady=10)

        tk.Label(self, text="Estrato:").grid(row=3, column=0, padx=10, pady=10)
        self.txt_estrato = tk.Entry(self)
        self.txt_estrato.grid(row=3, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=4,
column=0, padx=10, pady=10)
        tk.Button(self, text="Borrar", command=self.borrar).grid(row=4, column=1,
padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=5, column=0,
columnspan=2, padx=10, pady=10)

        # Campos de salida
        tk.Label(self, text="Resultado:").grid(row=6, column=0, padx=10, pady=10)
        self.txt_resultado = tk.Entry(self, state="readonly")
        self.txt_resultado.grid(row=6, column=1, padx=10, pady=10)
```

```
    def calcular(self):
        try:
            num_inscripcion  = self.txt_num_inscripcion.get()
            nombres = self.txt_nombres.get()
            patrimonio = float(self.txt_patrimonio.get())
            estrato = int(self.txt_estrato.get())

            matricula = CalculoMatricula(patrimonio,  estrato)
            pago_matricula  = matricula.calcular_pago_matricula()

            resultado = f"Número: {num_inscripcion}\nNombre:  {nombres}\nPago
Matrícula: ${pago_matricula:.2f}"
            self.txt_resultado.config(state="normal")
            self.txt_resultado.delete(0, tk.END)
            self.txt_resultado.insert(0, f"${pago_matricula:.2f}")
            self.txt_resultado.config(state="readonly")

        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa valores válidos para
patrimonio y estrato.")

    def borrar(self):
        self.txt_num_inscripcion.delete(0, tk.END)
        self.txt_nombres.delete(0, tk.END)
        self.txt_patrimonio.delete(0, tk.END)
        self.txt_estrato.delete(0, tk.END)
        self.txt_resultado.config(state="normal")
        self.txt_resultado.delete(0, tk.END)
        self.txt_resultado.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = Ventana()
    app.mainloop()
```

Ejercicios propuestos

- 22. Elaborar un algoritmo que entre el nombre de un empleado, su salario básico por hora y el número de horas trabajadas en el mes; escriba su nombre y salario mensual si éste es mayor de $450.000, de lo contrario escriba sólo el nombre.
  Código:

```
class CalculoNomina:
    def __init__(self, horas_trabajadas, valor_hora, nombre):
        self.horas_trabajadas = horas_trabajadas
        self.valor_hora = valor_hora
        self.nombre = nombre

    def calcular_nomina(self):
        salario = self.horas_trabajadas * self.valor_hora
        if salario < 450000:
            return f"{self.nombre}"
        else:
            return f"{self.nombre}\n{salario:.2f}"

#INTERFAZ GRÁFICA:

import tkinter as tk
from tkinter import messagebox

class Ventana(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Nómina")

        # Campos de entrada
        tk.Label(self, text="Nombre:").grid(row=0, column=0, padx=10, pady=5)
        self.txt_nombre = tk.Entry(self)
        self.txt_nombre.grid(row=0, column=1, padx=10, pady=5)

        tk.Label(self, text="Horas Trabajadas:").grid(row=1, column=0, padx=10,
pady=5)
        self.txt_horas_mes = tk.Entry(self)
        self.txt_horas_mes.grid(row=1, column=1, padx=10, pady=5)

        tk.Label(self, text="Valor por Hora:").grid(row=2, column=0, padx=10,
pady=5)
        self.txt_valor_hora = tk.Entry(self)
        self.txt_valor_hora.grid(row=2, column=1, padx=10, pady=5)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=3,
column=0, padx=10, pady=10)
        tk.Button(self, text="Borrar", command=self.borrar).grid(row=3, column=1,
padx=10, pady=10)
```

```
        tk.Button(self, text="Salir", command=self.salir).grid(row=4, column=0,
columnspan=2, padx=10, pady=10)

        # Campo de salida
        tk.Label(self, text="Respuesta:").grid(row=5, column=0, padx=10, pady=5)
        self.txt_respuesta = tk.Entry(self, state="readonly")
        self.txt_respuesta.grid(row=5, column=1, padx=10, pady=5)

    def calcular(self):
        try:
            nombre = self.txt_nombre.get()
            horas_trabajadas = float(self.txt_horas_mes.get())
            valor_hora = float(self.txt_valor_hora.get())

            nomina = CalculoNomina(horas_trabajadas, valor_hora, nombre)
            respuesta = nomina.calcular_nomina()

            # Mostrar respuesta en el campo de texto
            self.txt_respuesta.config(state="normal")
            self.txt_respuesta.delete(0, tk.END)
            self.txt_respuesta.insert(0, respuesta)
            self.txt_respuesta.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa valores numéricos
válidos para las horas trabajadas y el valor por hora.")

    def borrar(self):
        self.txt_nombre.delete(0, tk.END)
        self.txt_horas_mes.delete(0, tk.END)
        self.txt_valor_hora.delete(0, tk.END)

        self.txt_respuesta.config(state="normal")
        self.txt_respuesta.delete(0, tk.END)
        self.txt_respuesta.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = Ventana()
    app.mainloop()
```

- 23. Dados los valores A, B y C que son los parámetros de una ecuación de segundo grado, elaborar un algoritmo para hallar las posibles soluciones de dicha ecuación.

Código:

```
import math

class CalculoEcuacion:
    def __init__(self, a, b, c):
        self.valor_a = a
        self.valor_b = b
        self.valor_c = c

    def calcular_solucion_ecuacion(self):
        discriminante = self.calcular_discriminante()
        if discriminante > 0:
            x1 = (-self.valor_b + math.sqrt(discriminante)) / (2 * self.valor_a)
            x2 = (-self.valor_b - math.sqrt(discriminante)) / (2 * self.valor_a)
            return f"Las soluciones son: {x1:.2f} y {x2:.2f}"
        elif discriminante == 0:
            x = -self.valor_b / (2 * self.valor_a)
            return f"La solución es: {x:.2f}"
        else:
            return "No tiene soluciones reales"

    def calcular_discriminante(self):
        return (self.valor_b ** 2) - (4 * self.valor_a * self.valor_c)


#INTERFAZ GRÁFICA:

import tkinter as tk
from tkinter import messagebox

class Ventana(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Ecuación Cuadrática")

        # Campos de entrada para los valores A, B y C
        tk.Label(self, text="Valor A:").grid(row=0, column=0, padx=10, pady=5)
        self.txt_a = tk.Entry(self)
        self.txt_a.grid(row=0, column=1, padx=10, pady=5)

        tk.Label(self, text="Valor B:").grid(row=1, column=0, padx=10, pady=5)
```

```
        self.txt_b = tk.Entry(self)
        self.txt_b.grid(row=1, column=1, padx=10, pady=5)

        tk.Label(self, text="Valor C:").grid(row=2, column=0, padx=10, pady=5)
        self.txt_c = tk.Entry(self)
        self.txt_c.grid(row=2, column=1, padx=10, pady=5)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=3,
column=0, padx=10, pady=10)
        tk.Button(self, text="Borrar", command=self.borrar).grid(row=3, column=1,
padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=4, column=0,
columnspan=2, padx=10, pady=10)

        # Campo de salida para mostrar el resultado
        tk.Label(self, text="Resultado:").grid(row=5, column=0, padx=10, pady=5)
        self.txt_resultado = tk.Entry(self, state="readonly", width=50)  # Se agrega
width=50
        self.txt_resultado.grid(row=5, column=1, padx=10, pady=5)

    def calcular(self):
        try:
            a = float(self.txt_a.get())
            b = float(self.txt_b.get())
            c = float(self.txt_c.get())

            if a == 0:
                messagebox.showerror("Error",  "El valor de 'A' no puede ser cero en una
ecuación cuadrática.")
                return

            ecuacion = CalculoEcuacion(a, b, c)
            resultado = ecuacion.calcular_solucion_ecuacion()

            # Mostrar el resultado
            self.txt_resultado.config(state="normal")
            self.txt_resultado.delete(0, tk.END)
            self.txt_resultado.insert(0, resultado)
            self.txt_resultado.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa valores numéricos
válidos.")
```

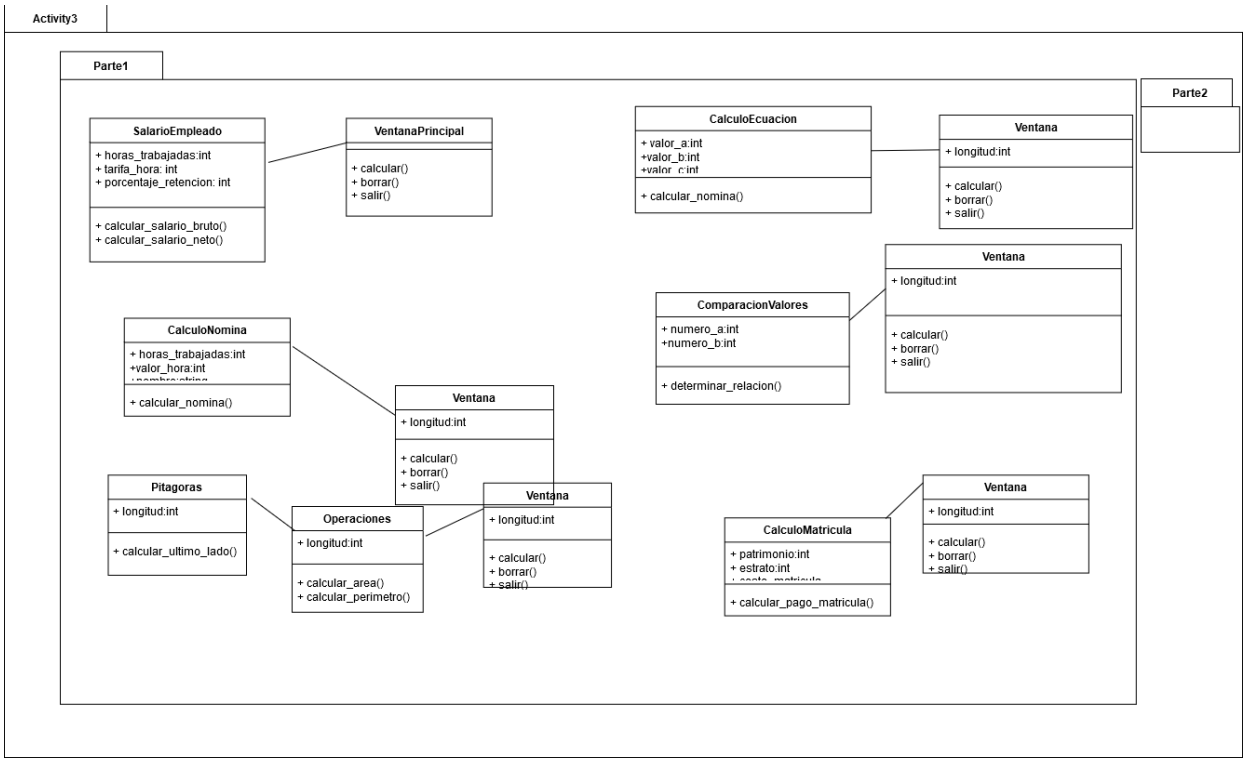
```
    def borrar(self):
        self.txt_a.delete(0, tk.END)
        self.txt_b.delete(0, tk.END)
        self.txt_c.delete(0, tk.END)

        self.txt_resultado.config(state="normal")
        self.txt_resultado.delete(0, tk.END)
        self.txt_resultado.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = Ventana()
    app.mainloop()
```

Diagrama UML



PARTE 2

Código fuente:
https://github.com/danielzp2000/actividad3PooUnalMed/tree/660e38e25dcc489a73b455f4fb8b64a8c92cb91f/Parte%202

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.

Círculo

```
import math


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


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


    def calcular_perimetro(self):
        return 2 * math.pi * self.radio
```

```
#INTERFAZ GRÁFICA

import tkinter as tk
from tkinter import messagebox

class InterfazCirculo(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Círculo")

        # Campo para ingresar el radio
        tk.Label(self, text="Radio:").grid(row=0, column=0, padx=10, pady=10)
        self.txt_radio = tk.Entry(self)
        self.txt_radio.grid(row=0, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=1, column=0, padx=10, pady=10)
        tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=1, column=1, padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=2, column=0, columnspan=2, padx=10, pady=10)

        # Campos de salida para área y perímetro
        tk.Label(self, text="Área:").grid(row=3, column=0, padx=10, pady=10)
        self.txt_area = tk.Entry(self, state="readonly")
        self.txt_area.grid(row=3, column=1, padx=10, pady=10)

        tk.Label(self, text="Perímetro:").grid(row=4, column=0, padx=10, pady=10)
```

```
        self.txt_perimetro = tk.Entry(self, state="readonly")
        self.txt_perimetro.grid(row=4, column=1, padx=10, pady=10)

    def calcular(self):
        try:
            radio = float(self.txt_radio.get())
            if radio < 0:
                messagebox.showerror("Error",  "El radio no puede ser negativo.")
                return

            circulo = Circulo(radio)
            area = circulo.calcular_area()
            perimetro = circulo.calcular_perimetro()

            # Mostrar resultados
            self.txt_area.config(state="normal")
            self.txt_area.delete(0, tk.END)
            self.txt_area.insert(0, f"{area:.2f}")
            self.txt_area.config(state="readonly")

            self.txt_perimetro.config(state="normal")
            self.txt_perimetro.delete(0, tk.END)
            self.txt_perimetro.insert(0, f"{perimetro:.2f}")
            self.txt_perimetro.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa un valor numérico válido para el
radio.")

    def limpiar(self):
```

```
        self.txt_radio.delete(0, tk.END)

        self.txt_area.config(state="normal")
        self.txt_area.delete(0, tk.END)
        self.txt_area.config(state="readonly")

        self.txt_perimetro.config(state="normal")
        self.txt_perimetro.delete(0, tk.END)
        self.txt_perimetro.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = InterfazCirculo()
    app.mainloop()
```

Cuadrado

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

    def calcular_area(self):
        return self.lado ** 2

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

#INTERFEZ GRÁFICA

```
import tkinter as tk
from tkinter import messagebox

class InterfazCuadrado(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Cuadrado")

        # Campo para ingresar el lado
        tk.Label(self, text="Lado:").grid(row=0, column=0, padx=10, pady=10)
        self.txt_lado = tk.Entry(self)
        self.txt_lado.grid(row=0, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=1, column=0, padx=10, pady=10)
        tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=1, column=1, padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=2, column=0, columnspan=2, padx=10, pady=10)

        # Campos de salida para área y perímetro
        tk.Label(self, text="Área:").grid(row=3, column=0, padx=10, pady=10)
        self.txt_area = tk.Entry(self, state="readonly")
        self.txt_area.grid(row=3, column=1, padx=10, pady=10)
```

```
        tk.Label(self, text="Perímetro:").grid(row=4, column=0, padx=10, pady=10)
        self.txt_perimetro = tk.Entry(self, state="readonly")
        self.txt_perimetro.grid(row=4, column=1, padx=10, pady=10)

    def calcular(self):
        try:
            lado = float(self.txt_lado.get())
            if lado < 0:
                messagebox.showerror("Error",  "El lado no puede ser negativo.")
                return

            cuadrado = Cuadrado(lado)
            area = cuadrado.calcular_area()
            perimetro = cuadrado.calcular_perimetro()

            # Mostrar resultados
            self.txt_area.config(state="normal")
            self.txt_area.delete(0, tk.END)
            self.txt_area.insert(0, f"{area:.2f}")
            self.txt_area.config(state="readonly")

            self.txt_perimetro.config(state="normal")
            self.txt_perimetro.delete(0, tk.END)
            self.txt_perimetro.insert(0, f"{perimetro:.2f}")
            self.txt_perimetro.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa un valor numérico válido para el
lado.")
```

```
    def limpiar(self):
        self.txt_lado.delete(0, tk.END)

        self.txt_area.config(state="normal")
        self.txt_area.delete(0, tk.END)
        self.txt_area.config(state="readonly")

        self.txt_perimetro.config(state="normal")
        self.txt_perimetro.delete(0, tk.END)
        self.txt_perimetro.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = InterfazCuadrado()
    app.mainloop()
```

Rectángulo

```
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) + (2 * self.altura)

# InTERFAZ GRÁFICA

import tkinter as tk
from tkinter import messagebox

class InterfazRectangulo(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Rectángulo")

        # Campos para ingresar la base y la altura
        tk.Label(self, text="Base:").grid(row=0, column=0, padx=10, pady=10)
        self.txt_base = tk.Entry(self)
        self.txt_base.grid(row=0, column=1, padx=10, pady=10)

        tk.Label(self, text="Altura:").grid(row=1, column=0, padx=10, pady=10)
        self.txt_altura = tk.Entry(self)
        self.txt_altura.grid(row=1, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=2, column=0,
padx=10, pady=10)
        tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=2, column=1,
padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=3, column=0,
columnspan=2, padx=10, pady=10)
```

```
    # Campos de salida para área y perímetro
    tk.Label(self, text="Área:").grid(row=4, column=0, padx=10, pady=10)
    self.txt_area = tk.Entry(self, state="readonly")
    self.txt_area.grid(row=4, column=1, padx=10, pady=10)

    tk.Label(self, text="Perímetro:").grid(row=5, column=0, padx=10, pady=10)
    self.txt_perimetro = tk.Entry(self, state="readonly")
    self.txt_perimetro.grid(row=5, column=1, padx=10, pady=10)

def calcular(self):
    try:
        base = float(self.txt_base.get())
        altura = float(self.txt_altura.get())

        if base < 0 or altura < 0:
            messagebox.showerror("Error",  "La base y la altura deben ser valores positivos.")
            return

        rectangulo = Rectangulo(base, altura)
        area = rectangulo.calcular_area()
        perimetro = rectangulo.calcular_perimetro()

        # Mostrar resultados
        self.txt_area.config(state="normal")
        self.txt_area.delete(0, tk.END)
        self.txt_area.insert(0, f"{area:.2f}")
        self.txt_area.config(state="readonly")
```

```
            self.txt_perimetro.config(state="normal")
            self.txt_perimetro.delete(0, tk.END)
            self.txt_perimetro.insert(0, f"{perimetro:.2f}")
            self.txt_perimetro.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa valores numéricos válidos para la base y la altura.")

    def limpiar(self):
        self.txt_base.delete(0, tk.END)
        self.txt_altura.delete(0, tk.END)

        self.txt_area.config(state="normal")
        self.txt_area.delete(0, tk.END)
        self.txt_area.config(state="readonly")

        self.txt_perimetro.config(state="normal")
        self.txt_perimetro.delete(0, tk.END)
        self.txt_perimetro.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = InterfazRectangulo()
    app.mainloop()
```

Triangulo

```
import math
```

```
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_perimetro(self):
        return self.base + self.altura + self.calcular_hipotenusa()

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

    def determinar_tipo_triangulo(self):
        hipotenusa = self.calcular_hipotenusa()
        if self.base == self.altura == hipotenusa:
            return "Equilátero"
        elif self.base != self.altura and self.base != hipotenusa and self.altura != hipotenusa:
            return "Escaleno"
        else:
            return "Isósceles"


#Interfaz Gráfica:

import tkinter as tk
```

```
from tkinter import messagebox

class InterfazTriangulo(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Cálculo de Triángulo Rectángulo")

        # Campos para ingresar la base y la altura
        tk.Label(self, text="Base:").grid(row=0, column=0, padx=10, pady=10)
        self.txt_base = tk.Entry(self)
        self.txt_base.grid(row=0, column=1, padx=10, pady=10)

        tk.Label(self, text="Altura:").grid(row=1, column=0, padx=10, pady=10)
        self.txt_altura = tk.Entry(self)
        self.txt_altura.grid(row=1, column=1, padx=10, pady=10)

        # Botones
        tk.Button(self, text="Calcular", command=self.calcular).grid(row=2, column=0, padx=10, pady=10)
        tk.Button(self, text="Limpiar", command=self.limpiar).grid(row=2, column=1, padx=10, pady=10)
        tk.Button(self, text="Salir", command=self.salir).grid(row=3, column=0, columnspan=2, padx=10, pady=10)

        # Campos de salida para área, perímetro, hipotenusa y tipo de triángulo
        tk.Label(self, text="Área:").grid(row=4, column=0, padx=10, pady=10)
        self.txt_area = tk.Entry(self, state="readonly")
        self.txt_area.grid(row=4, column=1, padx=10, pady=10)
```

```
        tk.Label(self, text="Perímetro:").grid(row=5, column=0, padx=10, pady=10)
        self.txt_perimetro = tk.Entry(self, state="readonly")
        self.txt_perimetro.grid(row=5, column=1, padx=10, pady=10)

        tk.Label(self, text="Hipotenusa:").grid(row=6, column=0, padx=10, pady=10)
        self.txt_hipotenusa = tk.Entry(self, state="readonly")
        self.txt_hipotenusa.grid(row=6, column=1, padx=10, pady=10)

        tk.Label(self, text="Tipo de Triángulo:").grid(row=7, column=0, padx=10, pady=10)
        self.txt_tipo_triangulo = tk.Entry(self, state="readonly")
        self.txt_tipo_triangulo.grid(row=7, column=1, padx=10, pady=10)

    def calcular(self):
        try:
            base = float(self.txt_base.get())
            altura = float(self.txt_altura.get())

            if base <= 0 or altura <= 0:
                messagebox.showerror("Error", "La base y la altura deben ser mayores que cero.")
                return

            triangulo = TrianguloRectangulo(base, altura)
            area = triangulo.calcular_area()
            perimetro = triangulo.calcular_perimetro()
            hipotenusa = triangulo.calcular_hipotenusa()
            tipo_triangulo = triangulo.determinar_tipo_triangulo()

            # Mostrar resultados
```

```
            self.txt_area.config(state="normal")
            self.txt_area.delete(0, tk.END)
            self.txt_area.insert(0, f"{area:.2f}")
            self.txt_area.config(state="readonly")

            self.txt_perimetro.config(state="normal")
            self.txt_perimetro.delete(0, tk.END)
            self.txt_perimetro.insert(0, f"{perimetro:.2f}")
            self.txt_perimetro.config(state="readonly")

            self.txt_hipotenusa.config(state="normal")
            self.txt_hipotenusa.delete(0, tk.END)
            self.txt_hipotenusa.insert(0, f"{hipotenusa:.2f}")
            self.txt_hipotenusa.config(state="readonly")

            self.txt_tipo_triangulo.config(state="normal")
            self.txt_tipo_triangulo.delete(0, tk.END)
            self.txt_tipo_triangulo.insert(0, tipo_triangulo)
            self.txt_tipo_triangulo.config(state="readonly")
        except ValueError:
            messagebox.showerror("Error",  "Por favor ingresa valores numéricos válidos.")

    def limpiar(self):
        self.txt_base.delete(0, tk.END)
        self.txt_altura.delete(0, tk.END)

        self.txt_area.config(state="normal")
        self.txt_area.delete(0, tk.END)
```

```
        self.txt_area.config(state="readonly")

        self.txt_perimetro.config(state="normal")
        self.txt_perimetro.delete(0, tk.END)
        self.txt_perimetro.config(state="readonly")

        self.txt_hipotenusa.config(state="normal")
        self.txt_hipotenusa.delete(0, tk.END)
        self.txt_hipotenusa.config(state="readonly")

        self.txt_tipo_triangulo.config(state="normal")
        self.txt_tipo_triangulo.delete(0, tk.END)
        self.txt_tipo_triangulo.config(state="readonly")

    def salir(self):
        self.destroy()

if __name__ == "__main__":
    app = InterfazTriangulo()
    app.mainloop()
```

Diagrama UML