



**UNIVERSIDAD NACIONAL DEL ALTIPLANO
PUNO**



**FACULTAD DE
INGENIERIA ESTADISTICA E INORMTICA**

LENGUAJE DE PROGRAMACION II

ing. COYLA IDME LEONEL

EXAMEN PRACTICO

Presentado por:

Vidman Ruis Roque Mamani

```

from abc import ABC, abstractmethod
from typing import List

# ----- VEHICULOS -----
class Vehiculo(ABC):
    """
    Clase base abstracta para representar un vehículo.
    Aplica los principios SRP y LSP.
    """
    def __init__(self, marca: str, modelo: str):
        self.marca = marca
        self.modelo = modelo

    @abstractmethod
    def obtener_datos(self) -> str:
        """Método que debe ser implementado para obtener información específica del
        vehículo."""
        pass

class Auto(Vehiculo):
    """
    Representa un automóvil con número de pasajeros.
    """
    def __init__(self, marca: str, modelo: str, pasajeros: int):
        super().__init__(marca, modelo)
        self.pasajeros = pasajeros

    def obtener_datos(self) -> str:
        return f"Auto - Marca: {self.marca}, Modelo: {self.modelo}, Pasajeros: {self.pasajeros}"

class Camion(Vehiculo):
    """
    Representa un camión con capacidad de carga.
    """
    def __init__(self, marca: str, modelo: str, carga_max: float):
        super().__init__(marca, modelo)
        self.carga_max = carga_max

    def obtener_datos(self) -> str:
        return f"Camion - Marca: {self.marca}, Modelo: {self.modelo}, Carga Máx: {self.carga_max} kg"

class Moto(Vehiculo):
    """
    Representa una motocicleta con tipo.
    """
    def __init__(self, marca: str, modelo: str, tipo: str):
        super().__init__(marca, modelo)
        self.tipo = tipo

    def obtener_datos(self) -> str:
        return f"Moto - Marca: {self.marca}, Modelo: {self.modelo}, Tipo: {self.tipo}"

# ----- INTERFACES DE REPORTE -----
class Reporte(ABC):
    """
    Interfaz para los distintos tipos de reporte.
    """

```

```

Aplica el principio ISP.
"""

@abstractmethod
def generar(self, vehiculos: List[Vehiculo]) -> None:
    pass

class ReporteConsola(Reporte):
    """
    Implementación concreta del reporte por consola.
    """
    def generar(self, vehiculos: List[Vehiculo]) -> None:
        print("Reporte por Consola (Clasificado):")
        autos = [v for v in vehiculos if isinstance(v, Auto)]
        camiones = [v for v in vehiculos if isinstance(v, Camion)]
        motos = [v for v in vehiculos if isinstance(v, Moto)]

        print("\n--- AUTOS ---")
        for v in autos:
            print(v.obtener_datos())
        print("\n--- CAMIONES ---")
        for v in camiones:
            print(v.obtener_datos())
        print("\n--- MOTOS ---")
        for v in motos:
            print(v.obtener_datos())

class ReporteArchivoTexto(Reporte):
    """
    Implementación concreta del reporte en archivo de texto.
    """
    def generar(self, vehiculos: List[Vehiculo]) -> None:
        autos = sorted([v for v in vehiculos if isinstance(v, Auto)], key=lambda v: (v.marca,
v.modelo))
        camiones = sorted([v for v in vehiculos if isinstance(v, Camion)], key=lambda v:
(v.marca, v.modelo))
        motos = sorted([v for v in vehiculos if isinstance(v, Moto)], key=lambda v: (v.marca,
v.modelo))

        with open("reporte_vehiculos.txt", "w", encoding="utf-8") as archivo:
            archivo.write("Reporte en archivo de texto (Clasificado):\n")
            archivo.write("\n--- AUTOS ---\n")
            for v in autos:
                archivo.write(v.obtener_datos() + "\n")
                print(v.obtener_datos())
            archivo.write("\n--- CAMIONES ---\n")
            for v in camiones:
                archivo.write(v.obtener_datos() + "\n")
                print(v.obtener_datos())
            archivo.write("\n--- MOTOS ---\n")
            for v in motos:
                archivo.write(v.obtener_datos() + "\n")
                print(v.obtener_datos())

class ReportePDF(Reporte):
    """
    Implementación simulada del reporte en formato PDF.
    """
    def generar(self, vehiculos: List[Vehiculo]) -> None:

```

```

        print("(Simulado) Reporte generado como PDF:")
        for vehiculo in vehiculos:
            print(vehiculo.obtener_datos())

# ----- GENERADOR DE REPORTES -----
class GeneradorReportes:
    """
    Clase que utiliza inyección de dependencias para generar reportes.
    Aplica DIP y composición para delegar responsabilidades.
    """

    def __init__(self, reporte: Reporte):
        self.reporte = reporte

    def generar_reporte(self, vehiculos: List[Vehiculo]) -> None:
        self.reporte.generar(vehiculos)

# ----- EJECUCIÓN -----
if __name__ == "__main__":
    # Lista de vehículos con sus datos específicos
    vehiculos: List[Vehiculo] = [
        Auto("Toyota", "Corolla", 5),
        Camion("Volvo", "FH16", 25000),
        Moto("Yamaha", "FZ", "Deportiva"),
        Auto("Honda", "Civic", 4),
        Camion("Mercedes-Benz", "Actros", 18000),
        Moto("Suzuki", "GSX-R", "Deportiva"),
        Auto("Chevrolet", "Onix", 5),
        Camion("Scania", "R500", 30000),
        Moto("Kawasaki", "Ninja", "Sport"),
        Camion("MAN", "TGX", 20000),
        Moto("Honda", "CBR600RR", "Sport")
    ]

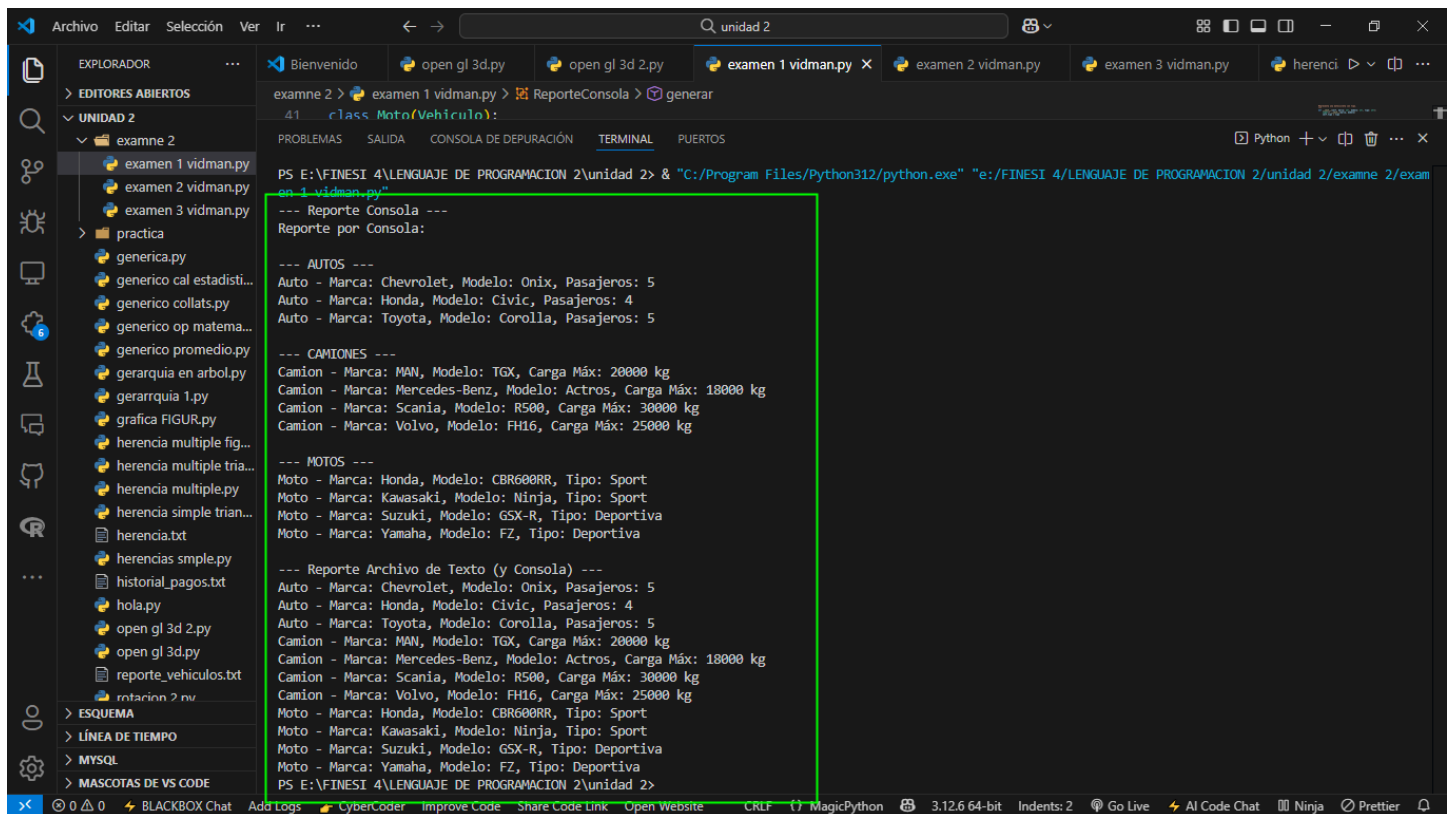
    # Ordenar por marca y luego por modelo
    vehiculos_ordenados = sorted(vehiculos, key=lambda v: (v.marca, v.modelo))

    print("--- Reporte Consola ---")
    GeneradorReportes(ReporteConsola()).generar_reporte(vehiculos_ordenados)

    print("\n--- Reporte Archivo de Texto (y Consola) ---")
    GeneradorReportes(ReporteArchivoTexto()).generar_reporte(vehiculos_ordenados)

```

Resultados por consola



The screenshot shows a VS Code editor with a file explorer on the left and a terminal window at the bottom. The terminal displays the output of a Python script named `examen 1 vidman.py`. The script defines a `Moto` class and prints data for various vehicles. The output is as follows:

```
PS E:\FINESI 4\LENGUAJE DE PROGRAMACION 2\unidad 2> "C:/Program Files/Python312/python.exe" "e:/FINESI 4/LENGUAJE DE PROGRAMACION 2/unidad 2/examen 2/examen-1-vidman.py"
--- Reporte Consola ---
Reporte por Consola:

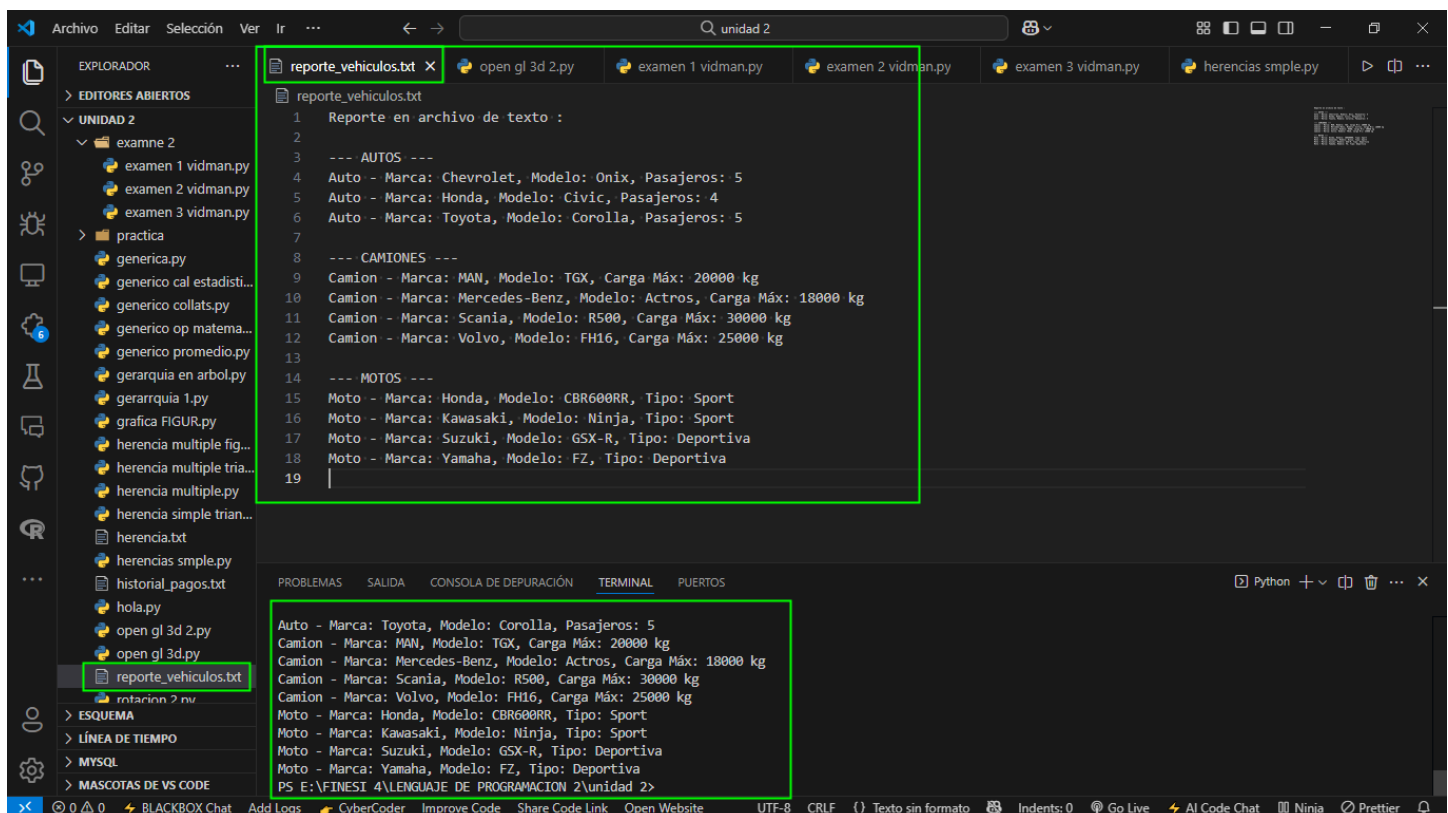
--- AUTOS ---
Auto - Marca: Chevrolet, Modelo: Onix, Pasajeros: 5
Auto - Marca: Honda, Modelo: Civic, Pasajeros: 4
Auto - Marca: Toyota, Modelo: Corolla, Pasajeros: 5

--- CAMIONES ---
Camion - Marca: MAN, Modelo: TGX, Carga Máx: 20000 kg
Camion - Marca: Mercedes-Benz, Modelo: Actros, Carga Máx: 18000 kg
Camion - Marca: Scania, Modelo: R500, Carga Máx: 30000 kg
Camion - Marca: Volvo, Modelo: FH16, Carga Máx: 25000 kg

--- MOTOS ---
Moto - Marca: Honda, Modelo: CBR600RR, Tipo: Sport
Moto - Marca: Kawasaki, Modelo: Ninja, Tipo: Sport
Moto - Marca: Suzuki, Modelo: GSX-R, Tipo: Deportiva
Moto - Marca: Yamaha, Modelo: FZ, Tipo: Deportiva

--- Reporte Archivo de Texto (y Consola) ---
Auto - Marca: Chevrolet, Modelo: Onix, Pasajeros: 5
Auto - Marca: Honda, Modelo: Civic, Pasajeros: 4
Auto - Marca: Toyota, Modelo: Corolla, Pasajeros: 5
Camion - Marca: MAN, Modelo: TGX, Carga Máx: 20000 kg
Camion - Marca: Mercedes-Benz, Modelo: Actros, Carga Máx: 18000 kg
Camion - Marca: Scania, Modelo: R500, Carga Máx: 30000 kg
Camion - Marca: Volvo, Modelo: FH16, Carga Máx: 25000 kg
Moto - Marca: Honda, Modelo: CBR600RR, Tipo: Sport
Moto - Marca: Kawasaki, Modelo: Ninja, Tipo: Sport
Moto - Marca: Suzuki, Modelo: GSX-R, Tipo: Deportiva
Moto - Marca: Yamaha, Modelo: FZ, Tipo: Deportiva
PS E:\FINESI 4\LENGUAJE DE PROGRAMACION 2\unidad 2>
```

Resultados en archivo TXT



The screenshot shows the same VS Code editor, but now the file `reporte_vehiculos.txt` is open in the editor. The file contains the same vehicle data as the terminal output, formatted as a text report. The content of the file is as follows:

```
1 Reporte en archivo de texto :
2
3 --- AUTOS ---
4 Auto - Marca: Chevrolet, Modelo: Onix, Pasajeros: 5
5 Auto - Marca: Honda, Modelo: Civic, Pasajeros: 4
6 Auto - Marca: Toyota, Modelo: Corolla, Pasajeros: 5
7
8 --- CAMIONES ---
9 Camion - Marca: MAN, Modelo: TGX, Carga Máx: 20000 kg
10 Camion - Marca: Mercedes-Benz, Modelo: Actros, Carga Máx: 18000 kg
11 Camion - Marca: Scania, Modelo: R500, Carga Máx: 30000 kg
12 Camion - Marca: Volvo, Modelo: FH16, Carga Máx: 25000 kg
13
14 --- MOTOS ---
15 Moto - Marca: Honda, Modelo: CBR600RR, Tipo: Sport
16 Moto - Marca: Kawasaki, Modelo: Ninja, Tipo: Sport
17 Moto - Marca: Suzuki, Modelo: GSX-R, Tipo: Deportiva
18 Moto - Marca: Yamaha, Modelo: FZ, Tipo: Deportiva
19
```

EXAMEN 2:-----

```
import tkinter as tk
from tkinter import ttk
from abc import ABC, abstractmethod
import math

# ----- INTERFAZ BASE -----
class FiguraGeometrica(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimetro(self):
        pass

    @abstractmethod
    def descripcion(self):
        pass

# ----- FIGURAS -----
class Circulo(FiguraGeometrica):
    def __init__(self, radio):
        self.radio = radio

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

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

    def descripcion(self):
        return f"Círculo de radio {self.radio}"

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

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

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

    def descripcion(self):
        return f"Cuadrado de lado {self.lado}"

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

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

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

```

    def descripcion(self):
        return f"Rectángulo de base {self.base} y altura {self.altura}"

class Triangulo(FiguraGeometrica):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def area(self):
        s = self.perimetro() / 2
        return math.sqrt(s * (s - self.a) * (s - self.b) * (s - self.c))

    def perimetro(self):
        return self.a + self.b + self.c

    def descripcion(self):
        return f"Triángulo de lados {self.a}, {self.b}, {self.c}"

# ----- VISUALIZADOR -----
class VisualizadorFiguras:
    def __init__(self, figuras, textbox):
        self.figuras = figuras
        self.textbox = textbox

    def mostrar(self):
        self.textbox.delete("1.0", tk.END)
        for figura in self.figuras:
            self.textbox.insert(tk.END, f"{figura.descripcion()}\n", "desc")
            self.textbox.insert(tk.END, f"Área: {figura.area():.2f}\n", "res")
            self.textbox.insert(tk.END, f"Perímetro: {figura.perimetro():.2f}\n\n", "res")

# ----- INTERFAZ PRINCIPAL -----
class App:
    def __init__(self, root):
        self.root = root
        self.root.title("📐 Calculadora de Figuras Geométricas")
        self.root.configure(bg="#f2f2f2")
        self.figuras = []

        title = tk.Label(root, text="Sistema de Figuras Geométricas", font=("Helvetica", 16,
"bold"), bg="#f2f2f2", fg="#2e7d32")
        title.grid(row=0, column=0, columnspan=2, pady=(10, 5))

        # Selector de figura
        tk.Label(root, text="Selecciona una figura:", font=("Helvetica", 11),
bg="#f2f2f2").grid(row=1, column=0, sticky="w", padx=10)
        self.figura_var = tk.StringVar()
        self.figura_menu = ttk.Combobox(root, textvariable=self.figura_var, state="readonly",
values=["Círculo", "Cuadrado", "Rectángulo",
"Triángulo"], width=20)
        self.figura_menu.grid(row=1, column=1, pady=5)
        self.figura_menu.bind("<<ComboboxSelected>>", self.actualizar_parametros)

        # Marco de parámetros
        self.param_frame = tk.Frame(root, bg="#e8f5e9", bd=2, relief="groove")
        self.param_frame.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky="we")

```

```

self.param_labels = []
self.param_entries = []

# Botones
tk.Button(root, text="+ Agregar Figura", bg="#81c784", fg="white",
font=("Helvetica", 10, "bold"),
        command=self.agregar_figura).grid(row=3, column=0, pady=5, padx=10)
tk.Button(root, text="👁 Mostrar Figuras", bg="#4caf50", fg="white",
font=("Helvetica", 10, "bold"),
        command=self.mostrar_figuras).grid(row=3, column=1, pady=5)

# Área de resultados
self.textbox = tk.Text(root, width=60, height=15, bg="ffffff", fg="#333",
font=("Courier New", 10))
self.textbox.grid(row=4, column=0, columnspan=2, padx=10, pady=(5, 10))
self.textbox.tag_config("desc", foreground="#1a237e", font=("Courier New", 10,
"bold"))
self.textbox.tag_config("res", foreground="#004d40")

def actualizar_parametros(self, event=None):
    for lbl in self.param_labels:
        lbl.destroy()
    for ent in self.param_entries:
        ent.destroy()
    self.param_labels.clear()
    self.param_entries.clear()

    figura = self.figura_var.get()
    campos = []

    if figura == "Círculo":
        campos = ["Radio"]
    elif figura == "Cuadrado":
        campos = ["Lado"]
    elif figura == "Rectángulo":
        campos = ["Base", "Altura"]
    elif figura == "Triángulo":
        campos = ["Lado A", "Lado B", "Lado C"]

    for i, nombre in enumerate(campos):
        lbl = tk.Label(self.param_frame, text=nombre + ":", font=("Helvetica", 10),
bg="#e8f5e9")
        lbl.grid(row=i, column=0, padx=5, pady=3, sticky="w")
        ent = tk.Entry(self.param_frame, font=("Helvetica", 10), width=20)
        ent.grid(row=i, column=1, padx=5, pady=3)
        self.param_labels.append(lbl)
        self.param_entries.append(ent)

def agregar_figura(self):
    tipo = self.figura_var.get()
    try:
        valores = [float(e.get()) for e in self.param_entries]
        figura = None
        if tipo == "Círculo" and len(valores) == 1:
            figura = Circulo(valores[0])
        elif tipo == "Cuadrado" and len(valores) == 1:
            figura = Cuadrado(valores[0])
        elif tipo == "Rectángulo" and len(valores) == 2:

```



```

        figura = Rectangulo(valores[0], valores[1])
    elif tipo == "Triángulo" and len(valores) == 3:
        figura = Triangulo(valores[0], valores[1], valores[2])

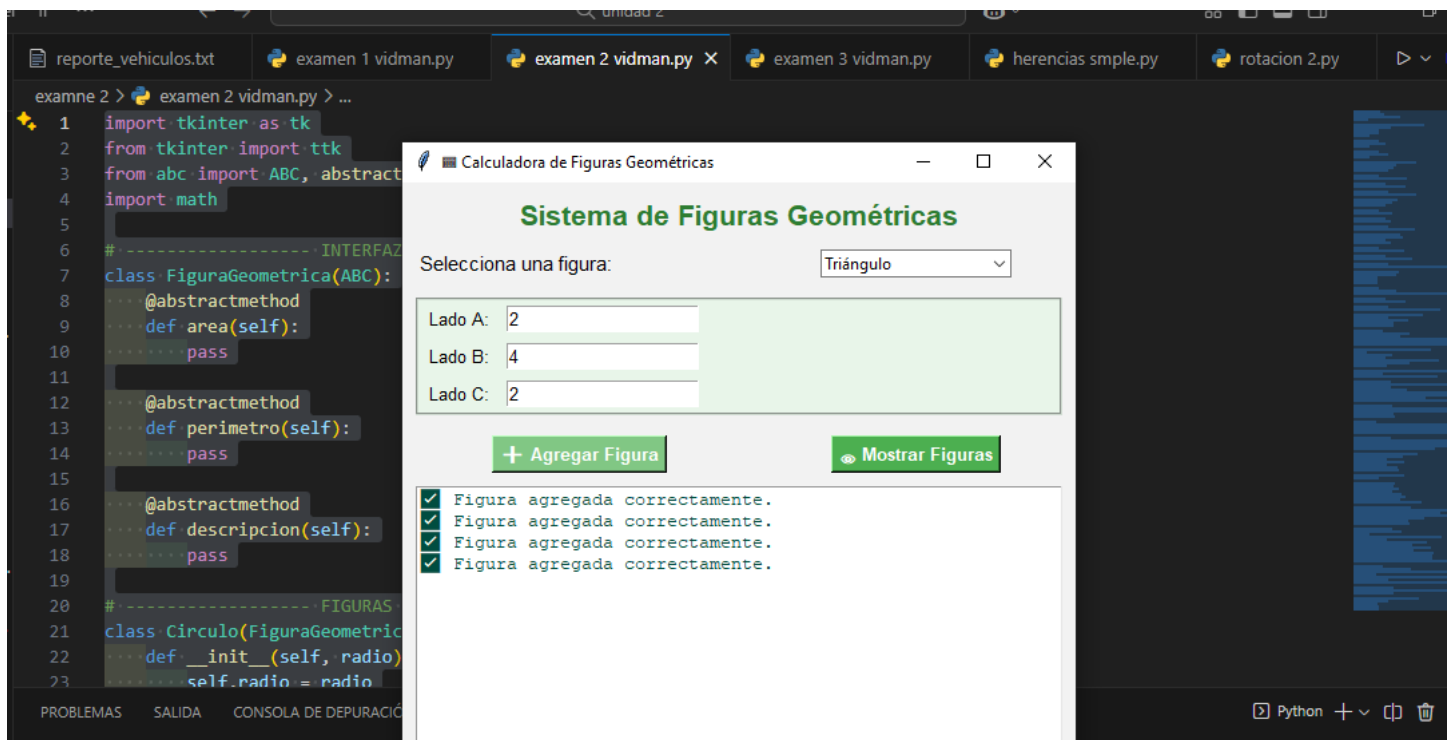
    if figura:
        self.figuras.append(figura)
        self.textbox.insert(tk.END, "✅ Figura agregada correctamente.\n", "res")
    else:
        self.textbox.insert(tk.END, "❌ Parámetros insuficientes o inválidos.\n",
"res")
    except ValueError:
        self.textbox.insert(tk.END, "⚠ Error: Ingresa solo números válidos.\n", "res")

def mostrar_figuras(self):
    visualizador = VisualizadorFiguras(self.figuras, self.textbox)
    visualizador.mostrar()

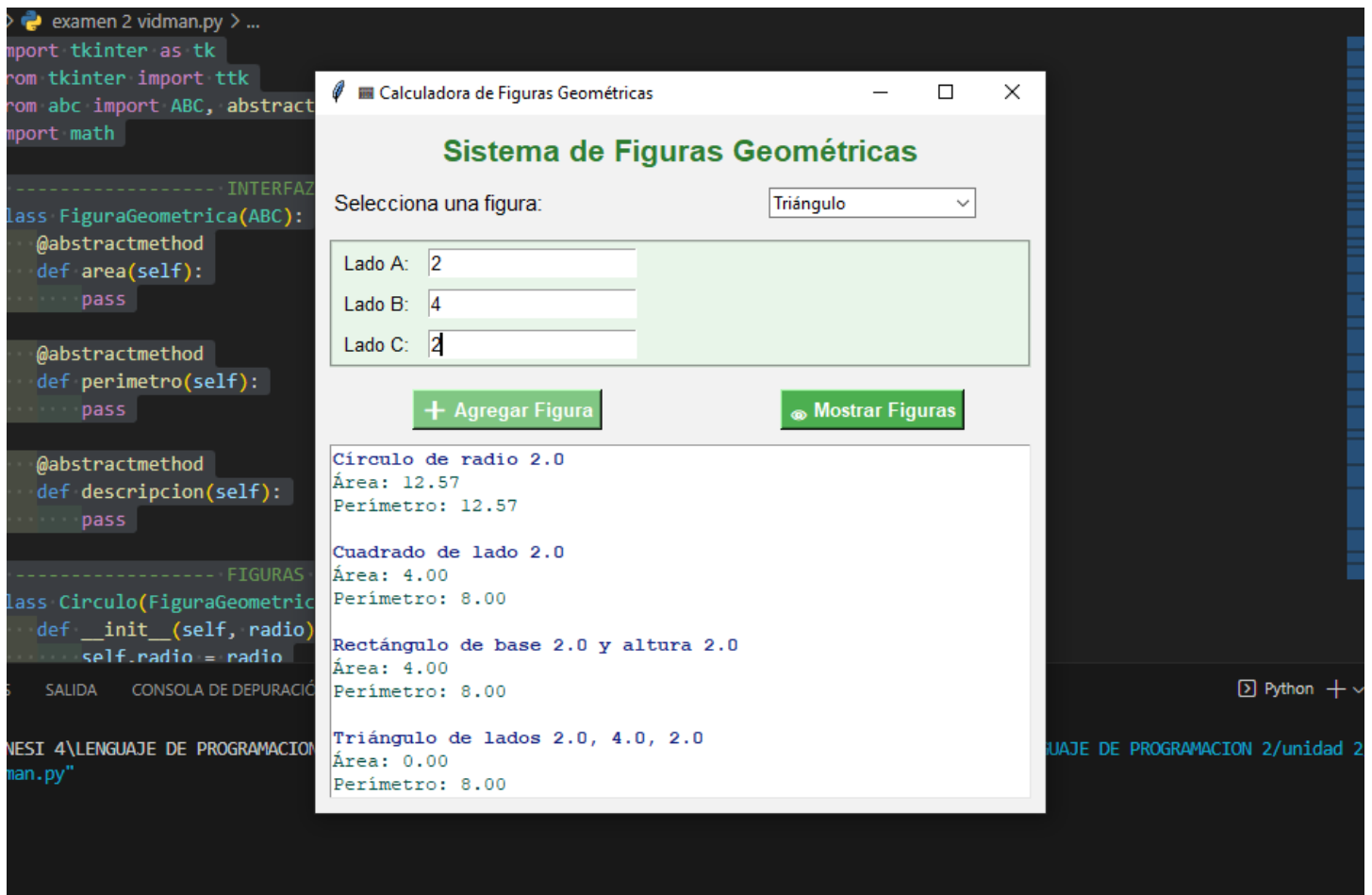
# ----- EJECUCIÓN -----
if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
    root.mainloop()

```

de acuerdo a la figura te pide que ingreses sus respectivos parámetros. Una vez echo se agrega la figura para posteriormente mostrar resultados



Una vez echo clic en mostrar figura, se mostrará todos los resultados de las figuras agregados



Torre de hanoi-----

```
import tkinter as tk
from tkinter import ttk
import time

# ----- LÓGICA DE NEGOCIO (S, O, L, I, D) -----

class HanoiGame:
    def __init__(self, n_discos):
        self.n_discos = n_discos
        self.movimientos = 0
        self.historial = []
        self.torres = {
            "A": [i for i in range(n_discos, 0, -1)],
            "B": [],
            "C": []
        }
```

```

def mover_disco(self, origen, destino):
    if origen == destino:
        return False
    if self.torres[origen] and (not self.torres[destino] or self.torres[origen][-1] <
self.torres[destino][-1]):
        disco = self.torres[origen].pop()
        self.torres[destino].append(disco)
        self.movimientos += 1
        self.historial.append((origen, destino))
        return True
    return False

def esta_resuelto(self):
    return len(self.torres["C"]) == self.n_discos

def resolver_hanoi(self, n=None, origen="A", auxiliar="B", destino="C"):
    pasos = []
    n = n if n is not None else self.n_discos
    if n > 0:
        pasos += self.resolver_hanoi(n - 1, origen, destino, auxiliar)
        pasos.append((origen, destino))
        pasos += self.resolver_hanoi(n - 1, auxiliar, origen, destino)
    return pasos

# ----- INTERFAZ GRÁFICA (Tkinter) -----

class HanoiApp:
    def __init__(self, master):
        self.master = master
        master.title("Torres de Hanoi - SOLID")
        self.juego = None
        self.tiempo_inicio = None
        self.torre_seleccionada = None

        # UI
        self.crear_widgets()

    def crear_widgets(self):
        self.label_discos = ttk.Label(self.master, text="Número de discos:")
        self.label_discos.pack()
        self.entry_discos = ttk.Entry(self.master)
        self.entry_discos.insert(0, "3")
        self.entry_discos.pack()

        self.boton_iniciar = ttk.Button(self.master, text="Iniciar Juego",
command=self.iniciar_juego)
        self.boton_iniciar.pack()

        self.canvas = tk.Canvas(self.master, width=400, height=300, bg="lightblue")
        self.canvas.pack()

```

```

self.canvas.bind("<Button-1>", self.click_canvas)

self.label_movimientos = ttk.Label(self.master, text="Movimientos: 0")
self.label_movimientos.pack()

self.label_tiempo = ttk.Label(self.master, text="Tiempo: 00:00")
self.label_tiempo.pack()

self.boton_solucion = ttk.Button(self.master, text="Mostrar Solución",
command=self.mostrar_solucion)
self.boton_solucion.pack()

self.label_historial = ttk.Label(self.master, text="Historial:")
self.label_historial.pack()

def iniciar_juego(self):
    try:
        n_discos = int(self.entry_discos.get())
        if n_discos <= 0:
            raise ValueError
        self.juego = HanoiGame(n_discos)
        self.tiempo_inicio = time.time()
        self.actualizar_tiempo()
        self.dibujar_torres()
    except ValueError:
        print("Número inválido de discos.")

def actualizar_tiempo(self):
    if self.tiempo_inicio:
        t = int(time.time() - self.tiempo_inicio)
        self.label_tiempo.config(text=f"Tiempo: {t//60:02}:{t%60:02}")
        self.master.after(1000, self.actualizar_tiempo)

def dibujar_torres(self):
    self.canvas.delete("all")
    espacio_entre_torres = 130
    ancho_base = 60
    altura_disco = 20
    alto_torre = 150

    for i, torre in enumerate(["A", "B", "C"]):
        x = 80 + i * espacio_entre_torres
        y_base = 280
        color = "orange" if self.torre_seleccionada == torre else "brown"

        self.canvas.create_rectangle(x - 5, y_base - alto_torre, x + 5, y_base,
fill=color)
        self.canvas.create_text(x, y_base + 10, text=torre)

        if self.juego:

```

```

        for j, disco in enumerate(self.juego.torres[torre]):
            disco_ancho = ancho_base * (disco / self.juego.n_discos)
            x0 = x - disco_ancho / 2
            y0 = y_base - (j + 1) * (altura_disco + 4)
            self.canvas.create_rectangle(x0, y0, x0 + disco_ancho, y0 + altura_disco,
fill=self.color_disco(disco))

    if self.juego:
        self.label_movimientos.config(text=f"Movimientos: {self.juego.movimientos}")
        self.label_historial.config(text="\n".join([f"{o} → {d}" for o, d in
self.juego.historial[-5:]]))

def click_canvas(self, event):
    x = event.x
    for i, torre in enumerate(["A", "B", "C"]):
        x_torre = 80 + i * 130
        if abs(x - x_torre) < 50:
            if self.torre_seleccionada is None:
                self.torre_seleccionada = torre
            else:
                if self.juego.mover_disco(self.torre_seleccionada, torre):
                    self.dibujar_torres()
                    if self.juego.esta_resuelto():
                        print("¡Juego completado!")
                    self.torre_seleccionada = None
                break
    self.dibujar_torres()

def mostrar_solucion(self):
    if not self.juego:
        return
    pasos = self.juego.resolver_hanoi()
    self.ejecutar_pasos(pasos)

def ejecutar_pasos(self, pasos, idx=0):
    if idx >= len(pasos):
        return
    origen, destino = pasos[idx]
    self.juego.mover_disco(origen, destino)
    self.dibujar_torres()
    self.master.after(500, lambda: self.ejecutar_pasos(pasos, idx + 1))

def color_disco(self, disco):
    colores = ["red", "green", "blue", "cyan", "magenta", "yellow"]
    return colores[disco % len(colores)]

# ----- MAIN -----
if __name__ == "__main__":
    root = tk.Tk()
    app = HanoiApp(root)

```

```
root.mainloop()
```