

INTERFACES
SIMULADOR DEL CLIMA

HANNA KATHERINE ABRIL GÓNGORA
KAROL ASLEY ORJUELA MAPE

UNIVERSIDAD MANUELA BELTRÁN

PROGRAMACIÓN ORIENTADA A OBJETOS

DOCENTE
DIANA MARCELA TOQUICA RODRÍGUEZ

FECHA

1. PREGUNTAS ORIENTADORAS

1.1. ¿Cuál es la diferencia clave entre una interfaz y una clase abstracta en Python y Java, y cuándo es más apropiado usar una sobre la otra en el diseño de un proyecto de software como un Simulador de Clima?

- Interfaz:

Python: En Python, una interfaz se define mediante una clase que contiene solo métodos sin implementaciones. Esto significa que una interfaz en Python no puede contener atributos ni métodos con implementaciones. Por ejemplo:

```
from abc import ABC, abstractmethod

class InterfaceExample(ABC):
    @abstractmethod
    def method1(self):
        pass

    @abstractmethod
    def method2(self):
        pass
```

Java: En Java, una interfaz es similar a la definición de Python, pero se usa la palabra clave interface en lugar de class y todos los métodos son automáticamente abstractos y públicos, por lo que no es necesario especificar abstract o public. Por ejemplo:

```
public interface InterfaceExample {
    void method1();
    void method2();
}
```

- Clase abstracta:

Python: Una clase abstracta en Python se define utilizando el módulo `abc` y se marca con la decoración `@abstractmethod`. Puede contener métodos abstractos (sin implementación) y métodos concretos (con implementación). Por ejemplo:

```
from abc import ABC, abstractmethod

class AbstractClassExample(ABC):
    @abstractmethod
    def method1(self):
        pass

    def method2(self):
        print("Implementación de method2")
```

Java: En Java, una clase abstracta se define con la palabra clave `abstract class`. Puede tener tanto métodos abstractos como concretos, y los métodos abstractos deben ser implementados por las clases que heredan de esta clase abstracta. Por ejemplo:

```
public abstract class AbstractClassExample {
    abstract void method1();

    void method2() {
        System.out.println("Implementación de method2");
    }
}
```

1.2. ¿Cómo pueden las interfaces ayudar a mantener la cohesión y la modularidad en el diseño del Simulador de Clima, especialmente cuando se trata de definir diferentes tipos de parámetros climáticos y sus comportamientos asociados?

En el diseño del Simulador de Clima en Python, las interfaces pueden ayudar enormemente a mantener la cohesión y la modularidad al definir diferentes tipos de parámetros climáticos y sus comportamientos asociados. Aquí hay algunas formas en que las interfaces pueden ser útiles en este contexto:

- Definición clara de comportamientos: Al utilizar interfaces, puedes definir claramente los comportamientos que deben implementar las clases que manejan diferentes tipos de parámetros climáticos. Por ejemplo, podrías tener una interfaz

ParametroClimatico que exija la implementación de métodos como `obtener_valor()` o `actualizar_valor()` para cada tipo de parámetro climático, como temperatura, humedad, presión atmosférica, etc.

- Flexibilidad en la implementación: Al definir interfaces para diferentes tipos de parámetros climáticos, puedes proporcionar una estructura flexible que permite agregar nuevos tipos de parámetros en el futuro sin cambiar la estructura básica del simulador. Por ejemplo, si en el futuro decides agregar un nuevo tipo de parámetro como `VelocidadViento`, simplemente puedes crear una nueva clase que implemente la interfaz `ParametroClimatico` sin afectar el código existente.
- Facilita la integración de módulos: Al utilizar interfaces, se facilita la integración de diferentes módulos dentro del simulador, ya que cada módulo puede implementar las interfaces necesarias sin depender directamente de las implementaciones concretas de otros módulos. Esto promueve una arquitectura modular y favorece la reutilización de código.

En resumen, al utilizar interfaces en Python para definir diferentes tipos de parámetros climáticos y sus comportamientos asociados, puedes mejorar la cohesión y modularidad del Simulador de Clima al proporcionar una estructura clara y flexible para la integración de módulos, la definición de comportamientos y la expansión futura del sistema.

1.3. ¿Qué estrategias y buenas prácticas se pueden utilizar al diseñar clases abstractas para encapsular comportamientos comunes entre los diferentes componentes del Simulador de Clima, como la simulación de la temperatura, la humedad y la presión atmosférica?

Al diseñar clases abstractas para encapsular comportamientos comunes entre los diferentes componentes del Simulador de Clima, como la simulación de la temperatura, la humedad y la presión atmosférica, puedes aplicar varias estrategias y buenas prácticas para mejorar la cohesión, la modularidad y la reutilización del código. Aquí tienes algunas sugerencias:

- **Identifica comportamientos comunes:** Analiza los diferentes componentes del Simulador de Clima para identificar los comportamientos que son comunes entre ellos. Por ejemplo, la mayoría de los componentes podrían tener métodos para iniciar la simulación, actualizar valores climáticos y obtener información actualizada.
- **Crea una clase abstracta base:** Basándote en los comportamientos comunes identificados, crea una clase abstracta base que encapsule estos comportamientos. Esta clase abstracta servirá como punto de partida para los diferentes componentes del simulador.
- **Implementa métodos concretos si es posible:** Si algunos de los comportamientos comunes tienen implementaciones comunes, puedes proporcionar métodos concretos en la clase abstracta base. Esto reduce la duplicación de código en las clases hijas.
- **Utiliza métodos abstractos para comportamientos específicos:** Define métodos abstractos en la clase abstracta base para comportamientos específicos que deben ser implementados por las clases hijas. Por ejemplo, podrías tener métodos abstractos como `simular_temperatura()`, `simular_humedad()`, etc.
- **Crea clases hijas especializadas:** Para cada tipo de componente climático (temperatura, humedad, presión atmosférica, etc.), crea clases hijas que hereden de la clase abstracta base y que implementen los métodos abstractos según sus necesidades específicas.

2. EXPLICACIÓN DEL CÓDIGO EN PYTHON

```
1 import tkinter as tk
2 from tkinter import messagebox
3 from abc import ABC, abstractmethod
4
```

`tkinter as tk:` Importa la biblioteca Tkinter bajo el alias `tk`, que se usa para crear interfaces gráficas.

`from tkinter import messagebox:` Importa la función `messagebox` de Tkinter para mostrar mensajes de error.

from abc import ABC, abstractmethod: Importa ABC y abstractmethod para definir clases y métodos abstractos.

```
7 class Clima(ABC):
8     def __init__(self, temperatura, humedad, presion):
9         self.temperatura = temperatura
10        self.humedad = humedad
11        self.presion = presion
12
13        @abstractmethod
14        def determinar_clima(self):
15            pass
16
17
```

Clima: Es una clase abstracta que define un constructor y un método abstracto determinar_clima que será implementado por sus subclases.

```
17
18 class CalculoClima(Clima):
19 >     def determinar_clima(self):...
110
```

CalculoClima: Subclase de Clima que implementa el método abstracto determinar_clima, donde se establecen las condiciones para determinar el tipo de clima basándose en los valores de temperatura, humedad y presión. Tener en cuenta que si los parámetros dados son irrealistas o no correspondientes a un clima existente en el programa el programa de notificara con None.

```
112 class SimuladorClimaApp(tk.Tk):
113     def __init__(self):
114         super().__init__()
115
116         self.attributes("-fullscreen", True)
117         self.bind("<Escape>", self.salir_pantalla_completa)
118         self.crear_contenido()
119         self.title("SkyScape")
120         self.configure(bg="white")
121
```

SimuladorClimaApp: Subclase de tk.Tk que representa la interfaz de la aplicación. En su inicialización, se configura la ventana y se crean los elementos visuales.

```

122
123     def salir_pantalla_completa(self, event=None):
124         self.attributes("-fullscreen", False)
125

```

Salir_pantalla_completa: Es un método cuya función es básicamente al correr el programa adaptarse a la pantalla completa de cualquier tipo de pantalla, esto con el propósito de evitar incompatibilidad y distorsión del programa.

```

126     def crear_contenido(self):
127         label_bienvenida = tk.Label(
128             self,
129             text="Bienvenido a SkyScape: Simulación del Clima",
130             font=("Arial", 24, "bold"),
131             pady=10,
132             bg="white",
133         )
134         label_bienvenida.pack(side=tk.TOP, fill=tk.X)
135
136         texto = """SkyScape es una herramienta avanzada de simulación del clima que
137         los diferentes fenómenos meteorológicos de forma interactiva."""
138         label_bienvenida = tk.Label(
139             self, text=texto, font=("Arial", 14), pady=10, bg="white"
140         )
141         label_bienvenida.pack(side=tk.TOP, fill=tk.X)
142
143         self.frame_ingreso()
144
145         boton_calcular = tk.Button(
146             self,
147             text="Calcular Clima",
148             relief=tk.RAISED,
149             command=self.calcular_clima,
150             width=15,
151             font=("Arial", 10, "bold"),
152         )
153         boton_calcular.pack(pady=10)
154
155         self.resultado_label = tk.Label(
156             self, text="El pronóstico del clima es: ", font=("Arial", 14), pady=10,
157         )
158         self.resultado_label.pack(side=tk.TOP, fill=tk.X)
159
160         self.frame_resultado() # Agregar el marco para mostrar el resultado
161
162         boton_salir = tk.Button(
163             self,
164             text="Salir",
165             relief=tk.RAISED,
166             command=self.salir_programa,
167             width=10,
168             font=("Arial", 10, "bold"),
169         )
170         boton_salir.place(relx=0.98, rely=0.98, anchor="se")

```

Crear_contenido: Es básicamente la configuración de la ventana principal en donde se verán los ingresos de datos y los resultados, en esta parte de código se establecen los parámetros de

configuración de la ventana, desde el color, mensajes, iconos etc.

```
171
172     def salir_programa(self):
173         self.destroy()
174
```

Salir_programa: Este método le da la funcionalidad al botón de salir, este con la utilidad de salir por completo del código.

```
176     def frame_ingreso(self):
177         frame = tk.Frame(self, bg="white")
178         frame.pack()
179
180         label_temperatura = tk.Label(
181             frame, text="Temperatura:", font=("Arial", 10, "bold"),
182         )
183         label_temperatura.grid(row=4, column=0, padx=10, pady=5)
184         self.entrada_temperatura = tk.Entry(frame, width=5)
185         self.entrada_temperatura.grid(row=4, column=1, padx=10, pady=5)
186         unidades_temperatura = tk.Label(frame, text="°C", bg="white")
187         unidades_temperatura.grid(row=4, column=2, pady=5)
188
189         label_humedad = tk.Label(
190             frame, text="Humedad:", font=("Arial", 10, "bold"), bg="
191         )
192         label_humedad.grid(row=5, column=0, padx=10, pady=5)
193         self.entrada_humedad = tk.Entry(frame, width=5)
194         self.entrada_humedad.grid(row=5, column=1, padx=10, pady=5)
195         unidades_humedad = tk.Label(frame, text="%", bg="white")
196         unidades_humedad.grid(row=5, column=2, pady=5)
197
198         label_presion_at = tk.Label(
199             frame,
200             text="Presión Atmosférica:",
201             font=("Arial", 10, "bold"),
202             bg="white",
203         )
204         label_presion_at.grid(row=6, column=0, padx=10, pady=5)
205         self.entrada_presion_at = tk.Entry(frame, width=5)
206         self.entrada_presion_at.grid(row=6, column=1, padx=10, pady=5)
207         unidades_presion_at = tk.Label(frame, text="hPa", bg="white")
208         unidades_presion_at.grid(row=6, column=2, pady=5)
```

Frame_ingreso: se definen y se dan las configuraciones de los espacios en donde los usuarios van a ingresar los datos para proceder a realizar el calculo correspondiente para predecir el clima.


```

210     def frame_resultado(self):
211         frame = tk.Frame(
212             self,
213             bg="grey",
214             width=900,
215             height=350,
216             borderwidth=1,
217             relief=tk.FLAT,
218             highlightbackground="grey",
219             highlightthickness=1,
220         )
221         frame.pack()
222

```

Frame_resultado: En esta área se planea dar el resultado de la predicción del clima, mediante texto e imágenes, además también se piensa mostrar los datos ingresados que generaron dicho clima además de una posible sugerencia personalizada según el tipo de clima, por si el usuario va a salir, para que tenga en cuenta.

```

223     def calcular_clima(self):
224         try:
225             temperatura = int(self.entrada_temperatura.get())
226             humedad = int(self.entrada_humedad.get())
227             presion = int(self.entrada_presion_at.get())
228
229             clima = CalculoClima(temperatura, humedad, presion)
230             tipo_clima = clima.determinar_clima()
231             self.resultado_label.config(text=f"El clima es: {tipo_clima}")
232         except ValueError:
233             messagebox.showerror(
234                 "Error",
235                 "Ingrese valores numéricos válidos para temperatura y humedad",
236             )
237

```

calcular_clima: Obtiene los valores ingresados por el usuario, crea un objeto CalculoClima y muestra el tipo de clima calculado.

```

239 if __name__ == "__main__":
240     app = SimuladorClimaApp()
241     app.mainloop()

```

if __name__ == "__main__": Verifica si el script se ejecuta directamente. Si es así, crea una instancia de SimuladorClimaApp y comienza el bucle principal de la interfaz gráfica.

3. EXPLICACION DEL CODIGO UML

```
1  @startuml Simulador
2
3  abstract class Clima {
4      -temperatura: int
5      -humedad: int
6      -presion: int
7      +determinar_clima(): string
8  }
9  class CalculoClima {
10     +determinar_clima(): string
11 }
12 class SimuladorClimaApp {
13     -entrada_temperatura: Entry
14     -entrada_humedad: Entry
15     -entrada_presion_at: Entry
16     -resultado_label: Label
17     +salir_pantalla_completa()
18     +crear_contenido()
19     +salir_programa()
20     +frame_ingreso()
21     +frame_resultado()
22     +calcular_clima()
23 }
24
25 Clima <|-- CalculoClima
26 SimuladorClimaApp -- Clima
27
28 @enduml
29
```

3.1. @startuml SimuLador: Esta línea inicia la creación de un diagrama UML llamado “SimuLador”.

3.2. abstract class Clima {: Declara una clase abstracta llamada “Clima”.

3.3. -temperatura: int: Define un atributo privado de tipo entero llamado “temperatura” en la clase “Clima”.

3.4. -humedad: int: Agrega otro atributo privado de tipo entero llamado “humedad” en la misma clase.

3.5. -presion: int: Añade un tercer atributo privado de tipo entero llamado “presion” a la clase “Clima”.

3.6. +determinar_clima(): string: Declara un método público llamado “determinar_clima” que devuelve una cadena.

3.7. } : Cierra la definición de la clase abstracta “Clima”.

3.8. class CalcuLoClima { : Comienza la definición de otra clase llamada “CalcuLoClima”.

3.9. +determinar_clima(): string: Similar a la línea 6, declara un método público con el mismo nombre pero dentro de esta nueva clase.

3.10. } : Finaliza esta segunda definición de clase.

3.11. class SimuLadorClimaApp { : Inicia la definición de una tercera clase, “SimuLadorClimaApp”.

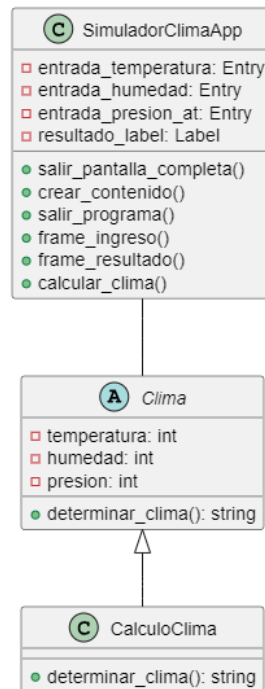
3.12. -entrada_temperatura: Entry, -entrada_humedad: Entry, -entrada_presion.at: Entry, -resultado_Label: Label: Estas líneas declaran cuatro atributos privados con tipos especificados como “Entry” o “Label”.

3.12. +salir_pantalla_completa(): Una declaración de método público sin tipo de retorno especificado, probablemente void o similar.

3.15. } : Cierra esta tercera definición de clase.

3.16. @enduml: Marca el final del diagrama UML.

4. DIAGRAMA DEL CODIGO EN UML



5. EXPLICACION Y USO

Este programa es un simulador de clima desarrollado con Tkinter en Python. Utiliza una estructura orientada a objetos con clases como **Clima** y **CalculoClima**, donde esta última hereda y define condiciones para determinar el tipo de clima en función de la temperatura, humedad y presión atmosférica ingresadas por el usuario. La clase principal **SimuladorClimaApp** crea la interfaz gráfica, permitiendo al usuario ingresar datos y calcular el clima correspondiente. Es una herramienta interactiva que facilita la comprensión de los fenómenos meteorológicos, esto nos ayuda a comprender mejor la programación orientada a objetos, la cual se implementó en este código.

6. DEMOSTRACION DE PRUEBAS Y FUNCIONAMIENTO DEL CODIGO

6.1. Prueba 1.



En esta prueba se puede apreciar, que el programa abre al instante en pantalla completa, se puede salir de esta mediante la teclas ESC y se puede salir del programa mediante el botón salir.

Además, en esta prueba podemos ver la versión beta por decirlo así de la interfaz grafica y su funcionamiento parcial, ya que se le dan unos parámetros de temperatura, humedad y presión, y se le da al usuario el clima correspondiente a esos parámetros, en este caso parcialmente nublado. Tener en cuenta que los cálculos realizados en este programa son según un clima templado, el clima cálido aun no se ha contemplado.

6.2. Prueba 2.

Bienvenido a SkyScape: Simulación del Clima

SkyScape es una herramienta avanzada de simulación del clima que te permite explorar y comprender los diferentes fenómenos meteorológicos de forma interactiva.

Temperatura: °C

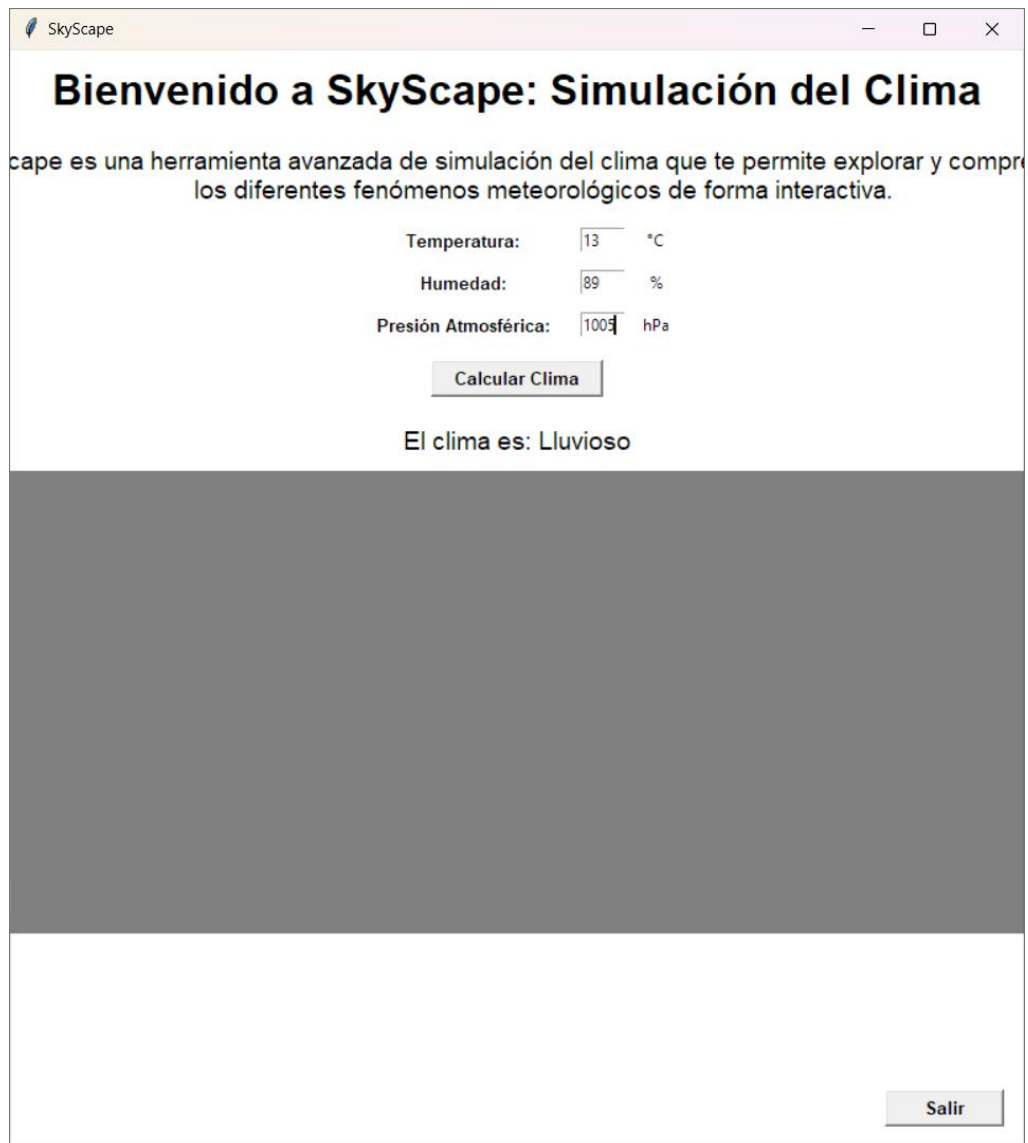
Humedad: %

Presión Atmosférica: hPa

El clima es: Calor extremo

En este caso se realiza la prueba, pero se adapta la pantalla a media pantalla para ver su responsividad, las áreas de ingreso de información se adaptan perfectamente al cambio, el texto no, pero se trabajará en eso futuramente, además de eso se prueba con otro clima.

6.3. Prueba 3.



The screenshot shows a web application window titled "SkyScape". The main heading is "Bienvenido a SkyScape: Simulación del Clima". Below this, a descriptive sentence states: "SkyScape es una herramienta avanzada de simulación del clima que te permite explorar y comprender los diferentes fenómenos meteorológicos de forma interactiva." The interface includes three input fields for climate parameters: "Temperatura:" with a value of 13 °C, "Humedad:" with a value of 89 %, and "Presión Atmosférica:" with a value of 1005 hPa. A "Calcular Clima" button is positioned below these inputs. The result of the simulation is displayed as "El clima es: Lluvioso". A large, solid gray rectangular area occupies the center of the page, likely representing a visualization of the simulated weather. A "Salir" button is located in the bottom right corner.

Bienvenido a SkyScape: Simulación del Clima

SkyScape es una herramienta avanzada de simulación del clima que te permite explorar y comprender los diferentes fenómenos meteorológicos de forma interactiva.

Temperatura: 13 °C

Humedad: 89 %

Presión Atmosférica: 1005 hPa

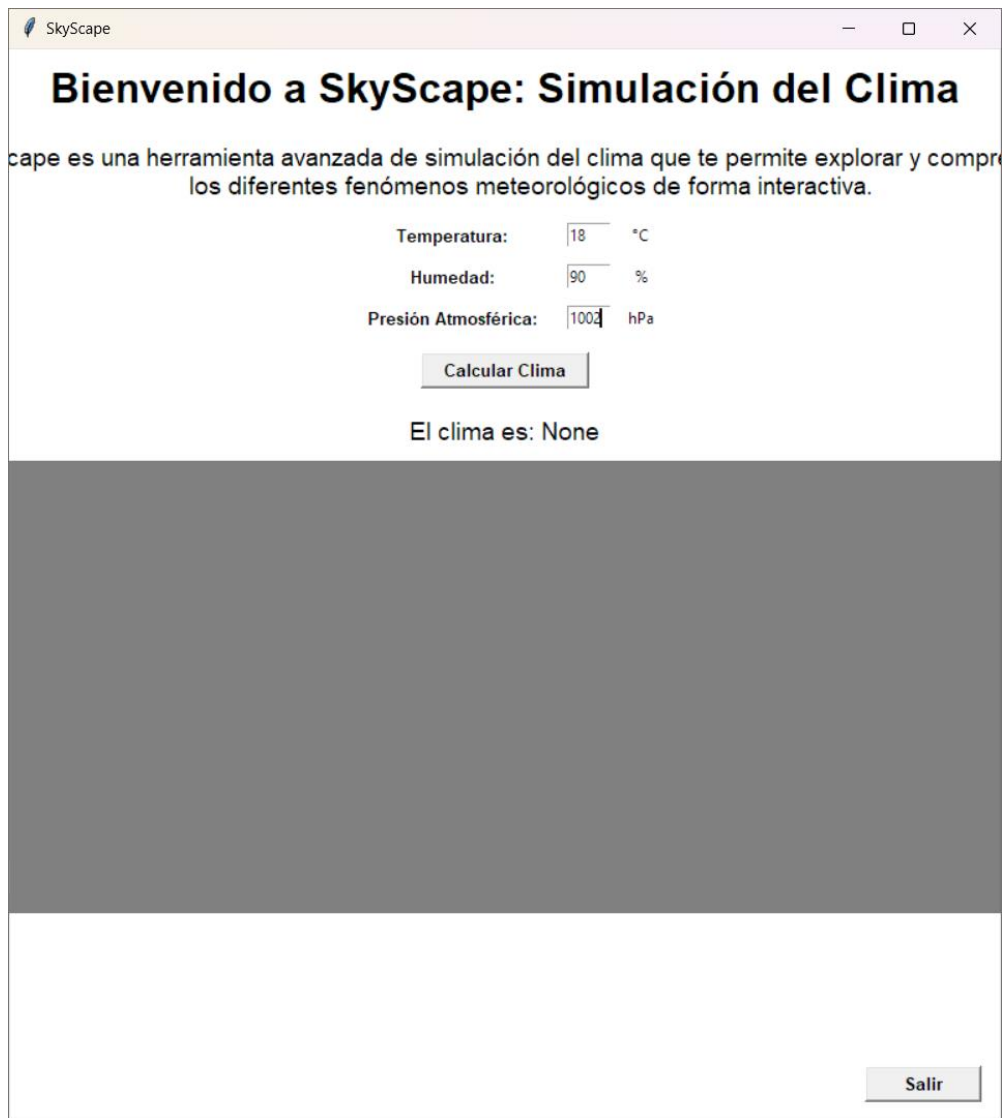
Calcular Clima

El clima es: Lluvioso

Salir

En esta prueba se demuestra únicamente la predicción del clima con otros parámetros, los cuales arrojan un clima lluvioso.

6.4. Prueba 4.



The screenshot shows a window titled "SkyScape" with a standard macOS-style title bar (minimize, maximize, close buttons). The main content area has a white background with the following elements:

- Header:** "Bienvenido a SkyScape: Simulación del Clima" in bold black text.
- Description:** "SkyScape es una herramienta avanzada de simulación del clima que te permite explorar y comprender los diferentes fenómenos meteorológicos de forma interactiva."
- Input Fields:**
 - "Temperatura:" with a text box containing "18" and a unit label "°C".
 - "Humedad:" with a text box containing "90" and a unit label "%".
 - "Presión Atmosférica:" with a text box containing "1003" and a unit label "hPa".
- Button:** A button labeled "Calcular Clima" below the input fields.
- Output:** The text "El clima es: None" is displayed below the button.
- Visual Representation:** A large, solid gray rectangular area occupies the lower half of the window, representing a simulated sky or landscape that is currently blank.
- Footer:** A button labeled "Salir" is located in the bottom right corner of the window.

En esta prueba se demuestra lo que pasa si se ingresan valores irrealistas o que no están dentro de las predicciones de clima del programa, para ese ejemplo se utilizan los parámetros temperatura, humedad y presión de 3 climas diferentes, por tal motivo nos da como resultado None.

7. DECISIÓN DE DISEÑO O CONSIDERACION IMPORTANTE

Para el diseño e implementación de una interfaz gráfica, (GUI) para interactuar con un sistema de simulación, hemos tomado una decisión clave: priorizar el modularidad y la reutilización del código. Para lograr este objetivo, hemos integrado clases abstractas en la estructura del proyecto. Estas clases abstractas actúan como plantillas flexibles que pueden ser extendidas por clases concretas para implementar funcionalidades específicas. Esta estrategia nos permite adaptarnos

fácilmente a cambios futuros y agregar nuevas funcionalidades sin necesidad de reescribir gran parte del código. Además, facilita la comprensión y el mantenimiento del sistema, ya que promueve una estructura clara y organizada.

8. PILARES DE POO IMPLEMENTADOS

La idea de nuestra implementación es que abarque los cuatro pilares fundamentales de la programación orientada a objetos al finalizar el proyecto. Si bien en esta etapa inicial nos hemos enfocado en integrar conceptos de herencia y polimorfismo, nuestra planificación contempla también la incorporación de encapsulamiento y abstracción en etapas posteriores. Esto nos permitirá construir un simulador climático sólido y adaptable, preparado para escalar y evolucionar con la adición de nuevas funcionalidades y la optimización del rendimiento.

9. CONCLUSIONES

Al desarrollar un simulador del clima no solo nos impulsa el crecimiento técnico, sino que también fomenta el desarrollo de habilidades creativas y de resolución de problemas. Esta iniciativa representa una emocionante oportunidad para aplicar el conocimiento adquirido en el aula en un proyecto práctico y significativo. Al diseñar e implementar una interfaz gráfica para interactuar con nuestro sistema de simulación, no solo estamos creando una herramienta funcional, sino que también estamos explorando nuevas formas de comunicar información compleja de manera clara y accesible.