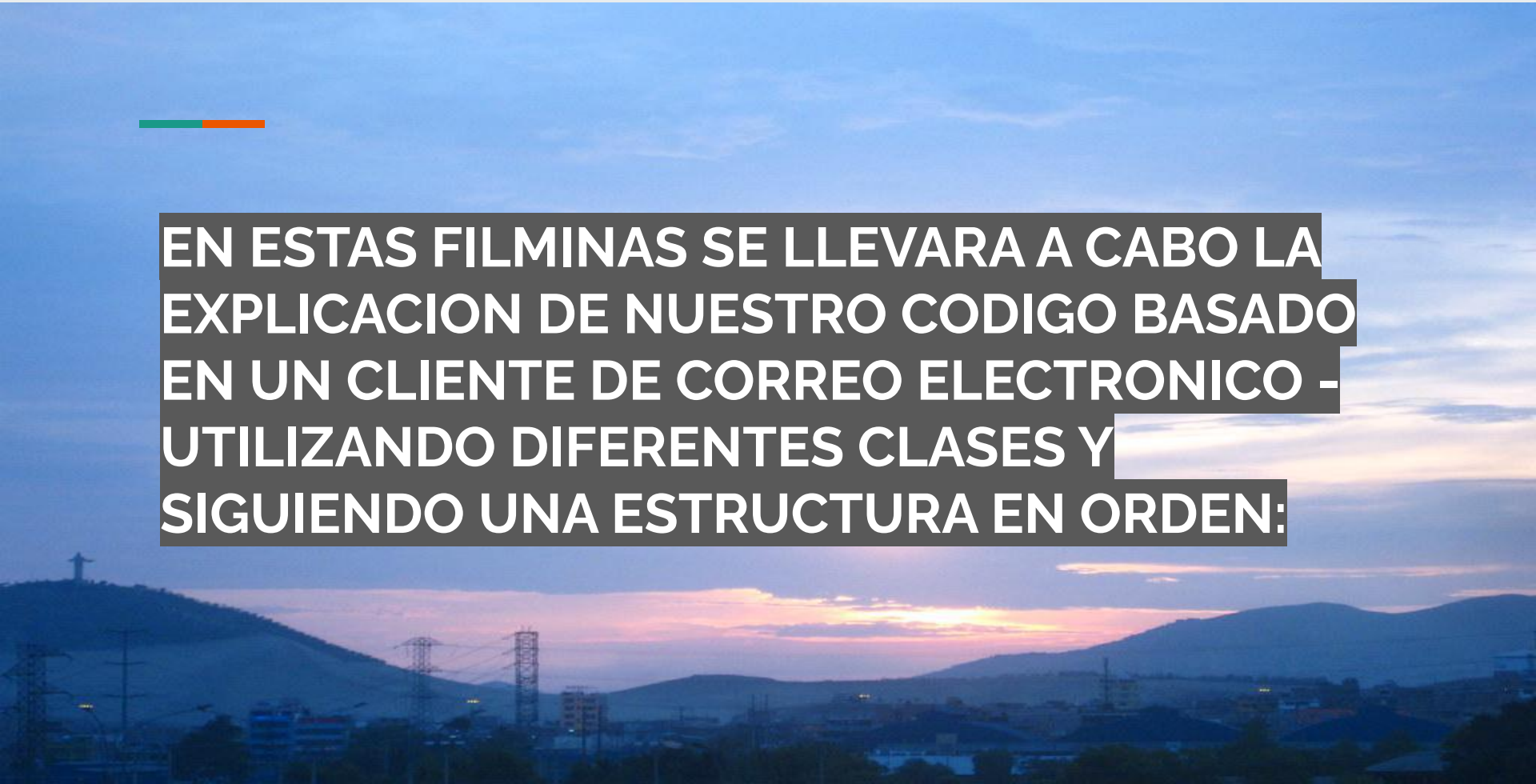
PROYECTO DE ESTRUCTURAS DE DATOS

Yapura Tomas
Cristaldo Giuliana
Aaron Lara

UNIVERSIDAD NACIONAL
GUILLERMO BROWN



EN ESTAS FILMINAS SE LLEVARA A CABO LA EXPLICACION DE NUESTRO CODIGO BASADO EN UN CLIENTE DE CORREO ELECTRONICO - UTILIZANDO DIFERENTES CLASES Y SIGUIENDO UNA ESTRUCTURA EN ORDEN:

PRIMERO COMENZAMOS CON LA IMPORTACIÓN DE HERRAMIENTAS

```
# Importamos herramientas para clases abstractas y anotaciones de tipo
from abc import ABC, abstractmethod
from typing import List
```

from abc import ABC, abstractmethod
Importa herramientas para crear clases abstractas e interfaces en Python.

ABC: Permite definir una clase como abstracta (no se puede instanciar directamente).
abstractmethod: acepta declarar forma que deben ser implementados por las subclases.
from typing import List
Importa el tipo List para indicar que una variable es una lista de elementos de cierto tipo (por ejemplo, List[Mensaje]).

¿Para qué sirven estas importaciones?

Permiten crear interfaces y clases base que obligan a las subclases a ejecutar ciertos métodos.
Facilitan la escritura de código más seguro y legible, usando apunte de tipo.

AHORA , PASAMOS A LA CLASE MENSAJE

```
# Clase que representa un mensaje de correo
class Mensaje:
    def __init__(self, remitente: str, destinatario: str, contenido: str):
```

1. **class Mensaje:**
   EN ESTA PRIMERA OCASIÓN definimos una nueva clase llamada Mensaje. Esta clase la uitilizaremos para crear objetos que representen mensajes de correo.
2. **def __init__(self, remitente: str, destinatario: str, contenido: str):**
   Es el método constructor. Se ejecuta automaticamente al crear un nuevo objeto de la clase.
   Recibe tres parámetros:
   a. remitente: el nombre del usuario que envia el mensaje.
   b. destinatario: el nombre del usuario que recibe el mensaje.
   c. contenido: el texto del mensaje

CLASE MENSAJE

```python
class Mensaje:
    def __init__(self, remitente: str, destinatario: str, contenido: str):
        # Guardamos el remitente, destinatario y contenido como atributos privados
        self._remitente = remitente
        self._destinatario = destinatario
        self._contenido = contenido
        self._info_extra = ("importante", "urgente")  # Tupla: ejemplo de información adicional
```

3 . self._remitente = remitente
Guarda el valor del parámetro envia en el atributo privado _remitente del objeto.
El prefijo _ indica que es privado y no debería modificarse directamente desde fuera de la clase.

4. self._destinatario = destinatario
Guarda el valor del parámetro destinatario en el atributo privado _destinatario.

5. self._contenido = contenido
Guarda el valor del parámetro contenido en el atributo privado _contenido.

6. self._info_extra = ("importante", "urgente")
Crea un atributo privado _info_extra que almacena una tupla con información adicional sobre el mensaje.
En este ejemplo, la tupla contiene los valores "importante" y "urgente".

CLASE MENSAJE

SEGUIMOS CON LOS PASOS DE LA CLASE MENSAJE
DONDE UTILIZAREMOS @property
PERO ¿ que es @property?
@property

- Es un decorador de Python que transforma un metodo en una propiedad. Permite acceder a sistema como si fueran atributos, sin necesidad de usar parentesis

```python
# Propiedad para acceder al remitente de forma segura
@property
def remitente(self):
    return self._remitente

# Propiedad para acceder al destinatario de forma segura
@property
def destinatario(self):
    return self._destinatario

# Propiedad para acceder al contenido de forma segura
@property
def contenido(self):
    return self._contenido

# Propiedad para acceder a la tupla de información extra
@property
def info_extra(self):
    return self._info_extra
```

1. **def remitente(self):**
   - Define una propiedad llamada remitente.
   - Cuando accedes a mensaje.remitente, se ejecuta este metodo y devuelve el valor de self._remitente.
2. **def destinatario(self):**
   - Define una propiedad llamada destinatario.
   - Al aceptar a mensaje.destinatario, devuelve el valor de self._destinatario.
3. **def contenido(self):**
   - Define una propiedad llamada contenido.
   - Al ceder a mensaje.contenido, devuelve el valor de self._contenido.
4. **def info_extra(self):**
   - Define una propiedad llamada info_extra.
   - Al acceder a mensaje.info_extra, devuelve el valor de self._info_extra, que es una tupla con informacion adicional.

5 . def info_extra(self):

- Define una propiedad llamada info_extra.
- Al acceder a mensaje.info_extra, devuelve el valor de self._info_extra, que es una tupla con informacion adicional

CONTINUAMOS CON LA CLASE CARPETA

DONDE REPRESENTAREMOS UNA ESPECIE DE BANDEJA DE ENTRADA

```python
# Clase que representa una carpeta de mensajes (como bandeja de entrada o enviados)
class Carpeta:
    def __init__(self, nombre: str):
        self._nombre = nombre
        self._mensajes: List[Mensaje] = []  # Lista para almacenar mensajes
```

1 . class Carpeta:

Definimos la clase Carpeta, que describe una carpeta de mensajes (por ejemplo, "bandeja entrada" o "enviados").

2. def __init__(self, nombre: str):

- Es el constructor de la clase.
- Recibe el parámetro nombre, que indica el nombre de la carpeta.
- Luego comienza dos atributos privados:

A. self._nombre: que lo que hace es almacenar el nombre de la carpeta.
B. self._mensajes: crea una lista vacía para guardar objetos de tipo Mensaje.

SEGUIMOS CON LA CLASE CARPETA

```python
@property
def nombre(self):
    return self._nombre

# Método para agregar un mensaje a la carpeta
def agregar_mensaje(self, mensaje: Mensaje):
    self._mensajes.append(mensaje)

# Método para listar todos los mensajes de la carpeta
def listar_mensajes(self):
    return self._mensajes
```

EN CONCLUSION : La clase Carpeta sirve para organizar mensajes en listas separadas por nombre (por ejemplo, bandeja de entrada o enviados). Permite agregar mensajes y consultar todos los mensajes guardados en esa carpeta.

1. **@property def nombre(self):**
   - Permite acceder al nombre de la carpeta de forma segura usando carpeta.nombre.
2. **def agregar_mensaje(self, mensaje: Mensaje):**
   - Método para agregar un mensaje a la carpeta.
   - Recibe un objeto Mensaje y lo añade a la lista self._mensajes.
3. **def listar_mensajes(self):**
   - forma para obtener todos los mensajes almacenados en la carpeta.
   - Devuelve la lista

AHORA EN NUESTRO CODIGO CONTINUAMOS CON LA CLASE ICORREO

EN ESTA CLASE SE HACE LA REFERENCIA A LA INTERFAZ QUE DEBE CONTENER UN CORREO

```python
class ICorreo(ABC):
    @abstractmethod
    def enviar_mensaje(self, mensaje: Mensaje):
        pass

    @abstractmethod
    def recibir_mensaje(self, mensaje: Mensaje):
        pass

    @abstractmethod
    def listar_mensajes(self, carpeta: str):
        pass
```

en conclusion o en resumen : La clase ICorreo es una interfaz que define los metodos que debe tener cualquier clase que represente un sistema de correo. Sirve para asegurar que todas las clases derivadas tengan estos metodos implementados.

1) class ICorreo(ABC):

- Define una clase llamada ICorreo que hereda de ABC (Abstract Base Class).
- Esto significa que es una clase abstracta, usada como interfaz, y no se puede crear un objeto directamente de ella.

2) @abstractmethod

- Es un decorador que indica que el sistema es abstracto.
- Obliga a que cualquier clase que herede de ICorreo implemente estos metodos.

3) def enviar_mensaje(self, mensaje: Mensaje):

- Metodo abstracto para enviar un mensaje.
- las clases derivada deben definir como se envia un mensaje.

4) def recibir_mensaje(self, mensaje: Mensaje):

- Metodo abstracto para recibir un mensaje.
- Las clases derivada deben definir como se recibe un mensaje.

5) def listar_mensajes(self, carpeta: str):

- Metodo abstracto para listar los mensajes de una carpeta especifica.

. Las clases derivadas deben definir como se listan los mensajes.

EN EL CODIGO CONTINUAMOS CON LA CLASE USUARIO

La clase Usuario representa a cada persona en el sistema de correo. Cada usuario tiene su nombre y dos carpetas para organizar sus mensajes. Puede enviar, recibir y consultar mensajes de forma ordenada.

```python
# Clase que representa a un usuario del sistema de correo
class Usuario(ICorreo):
    def __init__(self, nombre: str):
        self._nombre = nombre
        # Diccionario para almacenar las carpetas del usuario
        self._carpetas = {
            "bandeja_entrada": Carpeta("bandeja_entrada"),
            "enviados": Carpeta("enviados")
        }
```

1 . class Usuario(ICorreo):

- Definimos la clase Usuario, que representa a una persona que usa el sistema de correo.
- Hereda de la clase ICorreo, por lo que debe implementar los metodos definidos en esa interfaz.

2) def __init__(self, nombre: str):

- es el Constructor de la clase.
- Recibe el nombre del usuario y lo guarda en el atributo privado _nombre.
- Crea un diccionario privado _carpetas con dos carpetas: "bandeja_entrada" y "enviados", cada una es un objeto de la clase Carpeta.

SEGUNDA PARTE DE LA CLASE USUARIO

DONDE UTILIZAREMOS EL @PROPERTY Y METODOS PARA ENVIAR , RECIBIR Y LISTAR MENSAJES

```python
@property
def nombre(self):
    return self._nombre

# Método para enviar un mensaje (lo guarda en la carpeta de enviados)
def enviar_mensaje(self, mensaje: Mensaje):
    self._carpetas["enviados"].agregar_mensaje(mensaje)

# Método para recibir un mensaje (lo guarda en la bandeja de entrada)
def recibir_mensaje(self, mensaje: Mensaje):
    self._carpetas["bandeja_entrada"].agregar_mensaje(mensaje)

# Método para listar los mensajes de una carpeta especifica
def listar_mensajes(self, carpeta: str):
    if carpeta in self._carpetas:
        return self._carpetas[carpeta].listar_mensajes()
    return []
```

1) @property def nombre(self):

- Permite acceder al nombre del usuario de forma segura usando usuario.nombre.

2) def enviar_mensaje(self, mensaje: Mensaje):

- Metodo para enviar un mensaje.
- Guarda el mensaje en la carpeta "enviados" del usuario.

3) def recibir_mensaje(self, mensaje: Mensaje):

- Metodo para recibir un mensaje.
- Guarda el mensaje en la carpeta "bandeja_entrada" del usuario.

4) def listar_mensajes(self, carpeta: str):

- Manera para listar los mensajes de una carpeta especifica.
- Si la carpeta existe en el diccionario, devuelve la lista de mensajes de esa carpeta. Si no existe, devuelve una lista vacia.

Llegamos a la ultima la clase de nuestro codigo la cual es la clase SERVIDORCORREO

La clase ServidorCorreo es el núcleo del sistema. Permite registrar usuarios y resolver el envío de mensajes entre ellos, asegurando que cada mensaje llegue tanto al destinatario como quede registrado en los enviados del remitente.

CLASE SERVIDORCORREO

```python
# Clase que representa el servidor de correo, encargado de gestionar usuarios y mensajes
class ServidorCorreo:
    def __init__(self):
        self._usuarios = {}  # Diccionario para almacenar usuarios por nombre

    # Método para registrar un nuevo usuario en el servidor
    def registrar_usuario(self, usuario: Usuario):
        self._usuarios[usuario.nombre] = usuario

    # Método para enviar un mensaje entre usuarios registrados
    def enviar(self, mensaje: Mensaje):
        if mensaje.destinatario in self._usuarios:
            self._usuarios[mensaje.destinatario].recibir_mensaje(mensaje)
        if mensaje.remitente in self._usuarios:
            self._usuarios[mensaje.remitente].enviar_mensaje(mensaje)
```

CLASE SERVIDORCORREO PASO A PASO

1) class ServidorCorreo:

-Define la clase que representa el servidor de correo, encargado de gestionar los usuarios y el envío de mensajes.

2) def __init__(self):

- Constructor de la clase.

- comienza el atributo privado _usuarios como un diccionario vacío.

- Este diccionario almacenará los usuarios registrados, usando su nombre como clave.

3) def registrar_usuario(self, usuario: Usuario):

- procede para registrar un nuevo usuario en el servidor.

- Recibe un objeto Usuario y lo agrega al diccionario _usuarios, usando el nombre del usuario como clave.

4) def enviar(self, mensaje: Mensaje):

- Método para enviar un mensaje entre usuarios registrados.

- Verifica si el destinatario está registrado en el servidor:

- Si está, llama al uso recibir_mensaje del usuario destinatario para que reciba el mensaje en su bandeja de entrada.

- Verifica si el remitente está registrado:

- Si está, llama al método enviar_mensaje del usuario remitente para que el mensaje se guarde en su carpeta de enviados.