UNIVERSIDAD NACIONAL DE ALMIRANTE BROWN

Materia: Estructura de Datos

Alumnos: Octavio Melgarejo y Juan Pablo Alonso

Profesor: Ángel Leonardo Bianco

Fecha de entrega: 11/10/2025

DOCUMENTACIÓN

Guía lista

- *Descripción general,*
- *Diagrama de clases,*
- *Análisis de complejidad,*
- *Casos límite,*
- *Ejemplos de uso,*

Descripción General

El sistema simula un servicio básico de correo electrónico con interfaz gráfica.
Permite:

- Crear usuarios con nombre y correo.
- Iniciar sesión.
- Enviar y recibir mensajes entre usuarios registrados.
- Visualizar los mensajes recibidos.

Cada usuario tiene una bandeja de entrada propia (implementada como lista), donde se almacenan los mensajes recibidos.

Diagrama de Clases

```
classDiagram

    class persona {

        - nombre : str

        - mail : str

        - bandeja_entrada : BandejaEntrada

        + __init__(nombre, mail)

    }
```

```
 class mensaje {
- remitente : str
- destinatario : str
- asunto : str
- cuerpo : str
+ __init__(remitente, destinatario, asunto, cuerpo)
}
class BandejaEntrada {
- mensajes : list[mensaje]
+ agregar_mensaje(mensaje)
+ ver_todos() : list[mensaje]
+ __len__() : int
+ __getitem__(idx)
}
class Enviar_mensaje {
- remitente_mail : str
+ __init__(remitente_mail)
+ enviar(destinatario_mail, asunto, cuerpo) : str
}
persona "1" --> "1" BandejaEntrada
BandejaEntrada "1" --> "*" mensaje
Enviar_mensaje --> mensaje
```

Estructura del Sistema

Componente	Descripción
`persona`	Representa un usuario del sistema. Contiene nombre, correo y su bandeja de entrada.
`mensaje`	Modela un mensaje con remitente, destinatario, asunto y cuerpo.
`BandejaEntrada`	Administra los mensajes recibidos. Implementada mediante una lista de objetos `mensaje`.
`Enviar_mensaje`	Gestiona el envío de mensajes, verificando que el destinatario exista.
`usuarios` (diccionario global)	Almacena los usuarios registrados (`mail` → `persona`).
`main()`	Inicia la interfaz principal para crear o iniciar sesión.
`abrir_ventana_usuario()`	Muestra los mensajes del usuario y permite enviar nuevos.

Análisis de Complejidad Algorítmica

Operación	Descripción	Complejidad
`BandejaEntrada.agregar_mensaje()`	Agrega un mensaje al final de la lista.	**O(1)**
`BandejaEntrada.ver_todos()`	Retorna la lista completa de mensajes.	**O(n)**
`Enviar_mensaje.enviar()`	Verifica destinatario y agrega el mensaje.	**O(1)** promedio (acceso a diccionario)
`crear_usuario()`	Verifica existencia en el diccionario `usuarios`.	**O(1)** promedio
`ingresar_usuario()`	Busca usuario y verifica credenciales.	**O(1)** promedio

Casos Límite y Manejo de Errores

Situación	Comportamiento Actual	Posible Mejora
Enviar mensaje a usuario inexistente	Devuelve `"El destinatario no existe."`	Correcto
Crear usuario con mail vacío	Muestra error en la interfaz	Correcto
Crear usuario con mail repetido	Muestra mensaje `"Ese mail ya está registrado."`	Correcto
Iniciar sesión con datos incorrectos	Muestra error	Correcto
Bandeja sin mensajes	Muestra `"No tienes mensajes nuevos."`	Correcto
Intentar enviar mensaje a sí mismo	Actualmente permitido	Se podría agregar validación

Ejemplo de Uso (flujo típico)

Crear usuario

```
user1 = persona("Octavio", "octavio@mail.com")

usuarios[user1.mail] = user1
```

1.

Enviar mensaje

```
emisor = Enviar_mensaje("octavio@mail.com")

print(emisor.enviar("maria@mail.com", "Hola", "¿Cómo estás?"))

# Output: "Mensaje enviado a maria@mail.com"
```

2.

Ver bandeja de entrada

```
for msg in usuarios["maria@mail.com"].bandeja_entrada.ver_todos():
```

3. `    print(msg.asunto, "-", msg.cuerpo)`

JUSTIFICACIÓN DE DECISIONES

Nos enfocamos en que el sistema pueda cumplir con las funciones básicas de un cliente de correo: crear usuarios, enviar mensajes y leer los mensajes recibidos, intentando hacerlo lo más simple posible

- **Clase Persona:** creamos esta clase que serían los usuarios. Guarda el nombre y el mail. Y que se pueda crear usuarios desde el teclado.

- **Clase Mensaje:** esta clase es para enviar y recibir mensajes. También utiliza un diccionario para almacenar los mensajes recibidos por cada dirección de correo, porque los diccionarios permiten acceder rápido usando la clave (el mail del usuario).

- **Estructura de datos:** elegimos diccionarios (`dict`) porque son fáciles de manejar en Python y nos permiten asociar un mail con los mensajes o usuarios correspondientes.

- **Menús y opciones:** usamos menús por consola para que sea sencillo de probar y de ver.

En cuanto al **diagrama de clases**, mostramos tres elementos principales:

- **Persona** (usuario),
- **Mensaje** (envío y recepción),
- **SistemaCorreo** (que organiza usuarios y mensajes).

La idea es que el diagrama muestre de forma sencilla cómo se relacionan las clases que tenemos en el código actual y que nos ayude para futuro.