Trabajo Final

CLIENTE DE CORREO ELECTRÓNICO

ESTRUCTURA DE DATOS EN PYTHON

Melisa Gisela Hidalgo

Introducción

El proyecto consiste en simular un sistema de correo electrónico, se tomo como ejemplo el correo interno del campus, aunque tendría la misma aplicación en un sistema de comunicacion para empleados dentro de una empresas. El mismo está construido usando programación orientada a objetos y varias estructuras de datos. La idea principal aplicar conceptos de diseño, eficiencia, recursividad y algoritmos.

A lo largo del trabajo se fueron incorporando nuevas funcionalidades en cada entrega. Al principio solo estaban las clases básicas (Usuario, Mensaje, Carpeta, ServidorCorreo), todas bien encapsuladas y con sus getters/setters. Más adelante se sumó la estructura completa del árbol de carpetas, donde cada carpeta puede tener otras subcarpetas dentro, lo que nos permitió manejar la bandeja de entrada de manera recursiva.

Después agregamos funciones más avanzadas como filtros automáticos, una cola de prioridad para mensajes urgentes y un modelo de red con varios servidores conectados entre sí. Todo esto permitió que el sistema no solo guarde mensajes sino que también los clasifique, los ordene y hasta los envíe de un servidor a otro usando un recorrido BFS.

El resultado final es un proyecto completo donde se integran varios temas de la materia: encapsulamiento, estructuras lineales (listas, pilas, colas), recursividad con árboles generales, grafos, BFS y diseño orientado a objetos. Además, se incluye una interfaz por consola que permite manejar todas las funciones del sistema de manera interactiva.

OBJETIVOS

APLICAR CORRECTAMENTE LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

DEMOSTRAR EL USO DE CONCEPTOS FUNDAMENTALES COMO:

- Clases y objetos
- Encapsulamiento (ocultar datos internos)
- Interfaces (para estandarizar funciones como enviar, recibir y listar mensajes)
- Relaciones entre clases (composición, uso, dependencia, etc.)

IMPLEMENTAR ESTRUCTURAS DE DATOS PROPIAS PARA DEMOSTRAR MANEJO DE NODOS Y LÓGICA RECURSIVA COMO:

- Listas enlazadas (para almacenar mensajes)
- Pilas (para recuperación de eliminados)
- Colas de prioridad (para mensajes urgentes)
- Árbol general recursivo (para representar carpetas y subcarpetas)

Aplicar algoritmos y técnicas de recorrido

El proyecto debe mostrar:

- Recorridos recursivos por carpetas (preorder en tu implementación)
- Búsquedas por asunto o remitente
- Envío entre servidores usando BFS dentro de un grafo

Integrar todas las funcionalidades en un sistema completo

La última etapa del proyecto apunta a que todo funcione junto:

- Login y autenticación
- Filtros automáticos
- Manejo de mensajes prioritarios
- Estructura de carpetas
- Envío a usuarios y departamentos
- Comunicación en red

TODO ESTO PRESENTADO DESDE UN MENÚ INTERACTIVO SIMPLE

ADEMÁS DEL CÓDIGO, EL OBJETIVO INCLUYE:

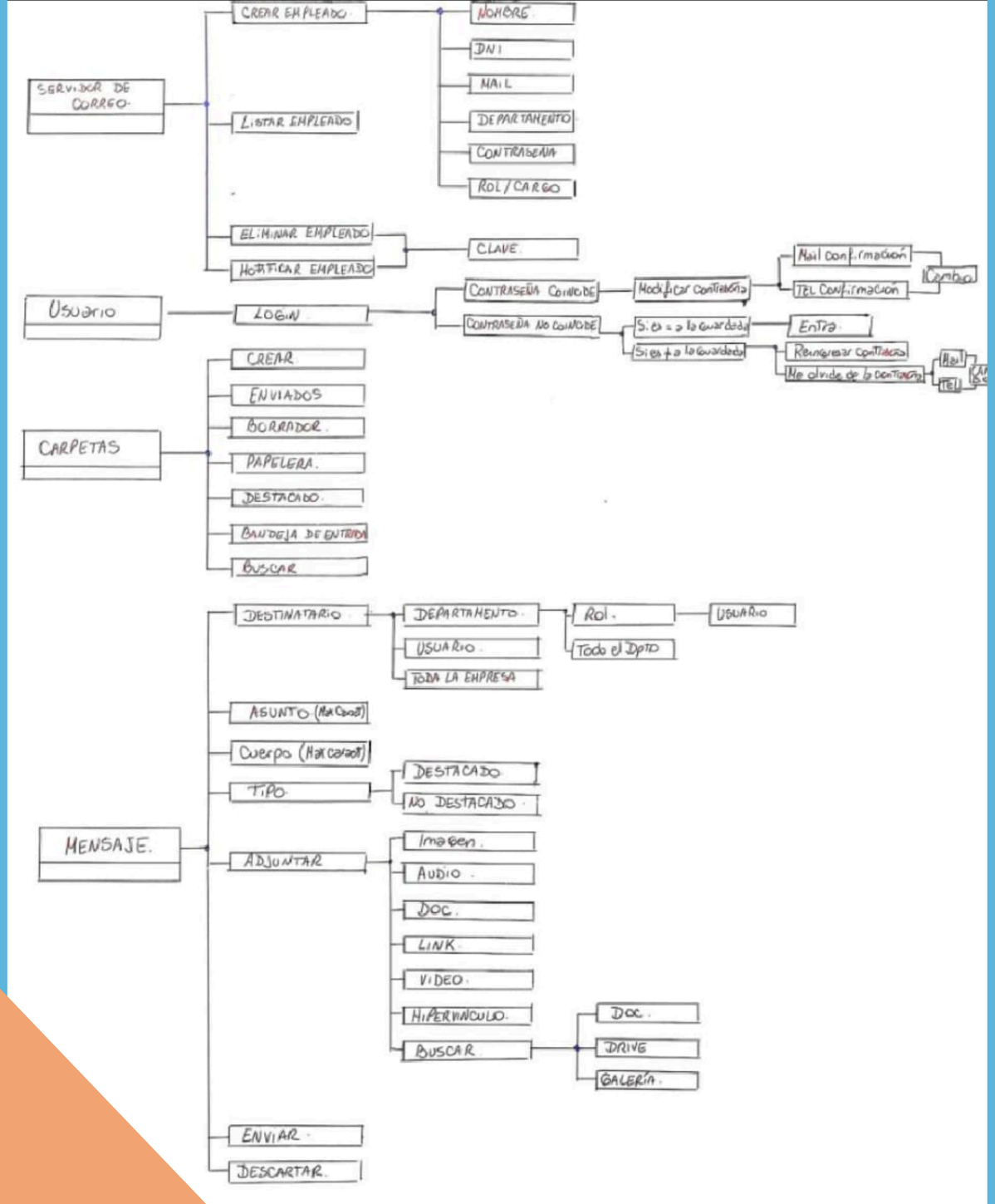
- DIAGRAMAS UML DE CLASES

- EXPLICACIONES DE POR QUÉ SE ELIGIÓ CADA ESTRUCTURA O DISEÑO

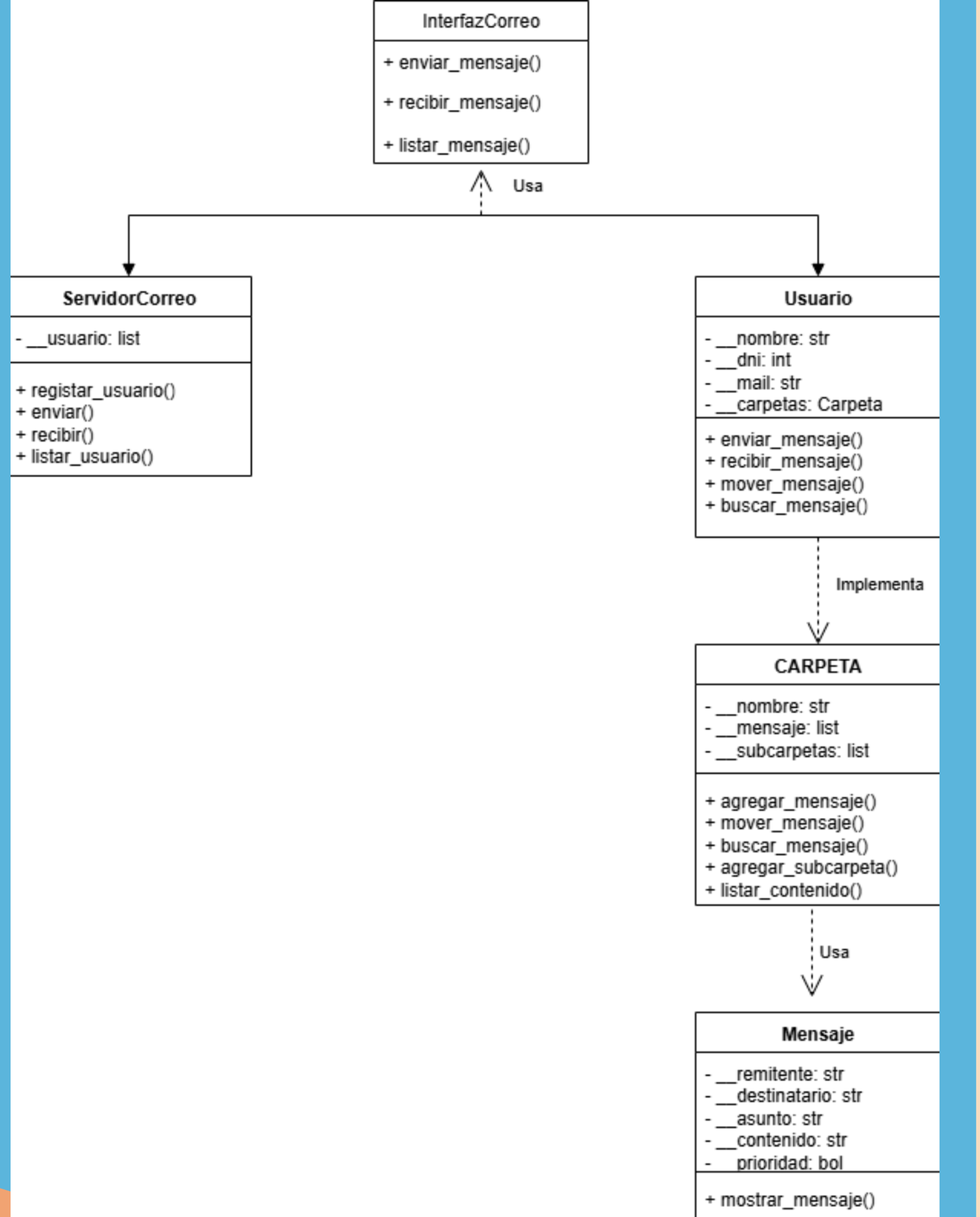
- ANÁLISIS DE EFICIENCIA

- CONCLUSIONES GENERALES

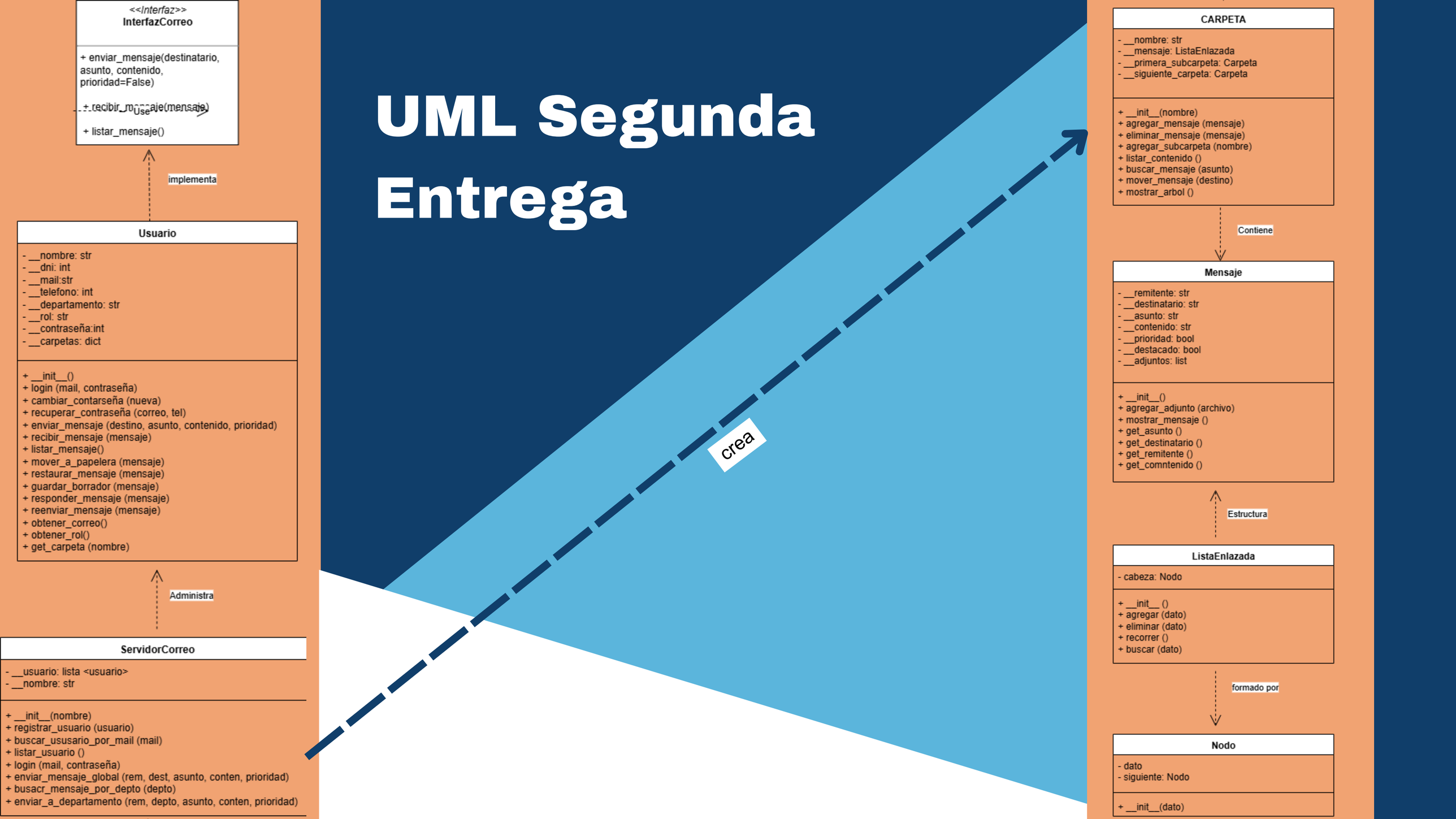
PLAN INICIAL

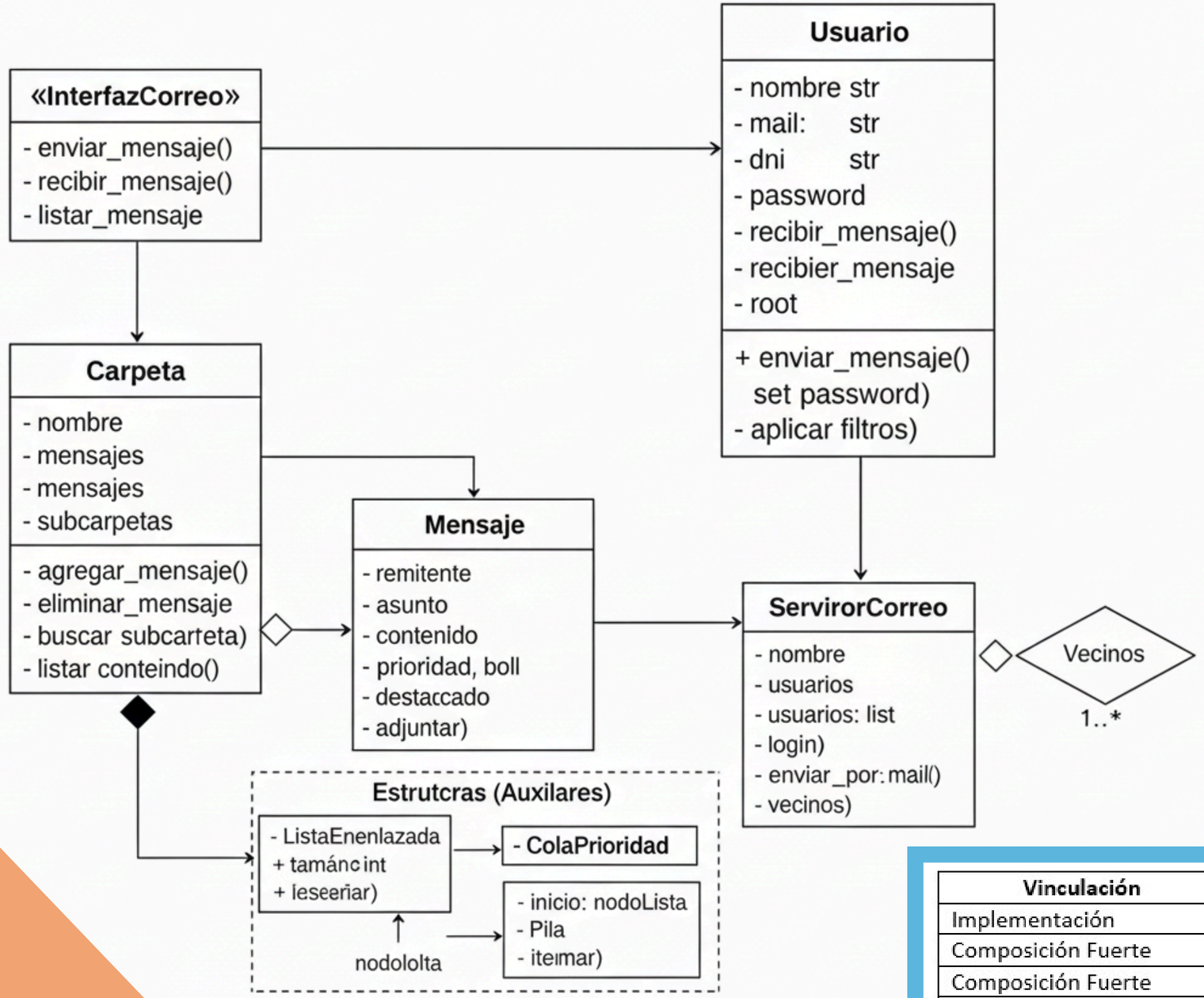


UML Primera Entrega



UML Segunda Entrega





UML Entrega Final

Vinculación	Clases Involucradas	Tipo (UML)
Implementación	Usuario ← InterfazCorreo	Herencia (Generalización)
Composición Fuerte	Usuario → Carpeta (Root, Inbox)	Rombo Negro (♦)
Composición Fuerte	Carpeta → ListaEnlazada / Pila	Rombo Negro (♦)
Agregación (Red)	ServidorCorreo ↔ ServidorCorreo	Rombo Blanco (◇)
Asociación (Gestión)	ServidorCorreo ↔ Usuario	Línea Simple
Asociación (Contención)	Carpeta ↔ Mensaje	Línea Simple

Arquitectura de diseño

CLASES PRINCIPALES

Class	Atributos (Tipo de Dato)	Métodos (Funciones)
Usuario	nombre: str dni: str/int mail: str rol: str root: Carpeta papelera: Pila cola_prioridad: ColaPrioridad	+login(mail, password) +set_password(nueva_pass) +verificar_password(pass) +enviar_mensaje(dest, asunto, cont) +recibir_mensaje(mensaje) +agregar_regla_filtro(palabra, carpeta) +procesar_mensajes_prioritarios()
Carpeta	nombre: str mensajes: ListaEnlazada subcarpetas: list<Carpeta> pila_auxiliar: Pila	+agregar_mensaje(mensaje) +eliminar_mensaje(mensaje) +recuperar_ultimo_eliminado() +buscar_subcarpeta(nombre) +listar_contenido()
ServidorCorreo	nombre:str usuarios: list<Usuario> vecinos: list<ServidorCorreo>	+registrar_usuario(usuario) +buscar_usuario_por_mail(mail) +login(mail, password) +enviar_mensaje_global() +bfs_ruta(servidor_destino)
Mensaje	remitente: str destinatarios: list<str> asunto: str prioridad: bool fecha:datetime	+mostrar_mensaje() +adjuntar(tipo, nombre, ruta) +es_prioritario() +marcar_destacado()

CLASES AUXILIARES (ESTRUCTURA DE DATOS)

Class	Atributos (Tipo de Dato)	Métodos (Funciones)
ListaEnlazada	Inicio: nodoLista	+insertar(dato) +eliminar(dato)
Pila	Ítem: list	+apilar(elemento) +desapilar()
Cola	Ítem: list	+encolar(elemento) +desencolar()

Árbol General

SIRVE PARA LA ORGANIZACIÓN DE CARPETAS

Se usa en :
Clase Carpeta (Carpeta.py)
Cada carpeta tiene subcarpetas (recursividad)
Se usa en métodos como listar_contenido(), buscar_subcarpeta(), buscar_mensaje_por_asunto()

Sirve para:
Representar la estructura de carpetas del usuario:
- **"Root"**
- **"Bandeja de Entrada"**
- **"Enviados"**
- **Cualquier carpeta que el usuario vaya creando**
- **Y dentro puede haber más subcarpetas**

Se eligió porque:
- **Permite recorrer todo de forma ordenada**
- **Permite hacer búsquedas recursivas**
- **No pierde formato si el ususario crea nuevas carpetas**
- **Formato similar al sistema de correo real**

```python
def agregar_subcarpeta(self, subcarpeta):
    self.__subcarpetas.append(subcarpeta)

def buscar_subcarpeta(self, nombre):
    for sub in self.__subcarpetas:
        if sub.get_nombre() == nombre:
            return sub

        resultado = sub.buscar_subcarpeta(nombre)
        if resultado is not None:
            return resultado

    return None
```

```python
def buscar_mensaje_por_asunto(self, palabra):
    encontrados = []

    mensajes = self.__mensajes.obtener_todos()
    for mensaje in mensajes:
        if palabra.lower() in mensaje.get_asunto().lower():
            encontrados.append(mensaje)

    for subcarpeta in self.__subcarpetas:
        encontrados += subcarpeta.buscar_mensaje_por_asunto(palabra)

    return encontrados
```

```python
class Carpeta:

    def __init__(self, nombre):
        self.__nombre = nombre
        self.__mensajes = ListaEnlazada()
        self.__subcarpetas = []
        self.__pila_auxiliar = Pila()
```

Cola de Prioridad

SIRVE PARA ESTABLECER MENSAJES COMO URGENTES SI LO SON.

Se usa en:
En Estructura, clase ColaPrioridad
En Usuario, atributo __cola_prioridad
Métodos recibir_mensaje() y procesar_mensajes_prioritarios()

```python
def recibir_mensaje(self, mensaje):
    # Si el mensaje es urgente → va a la cola de prioridad
    if mensaje.get_prioridad():
        self.__cola_prioridad.encolar(mensaje)
```

Sirve para:
-Cuando un mensaje llega marcado como urgente, no va directo a la bandeja, va primero a la cola de prioridad, esto permite que el usuario pueda abrir los mensajes urgentes antes que los normales.

```python
class ColaPrioridad(Cola):
    def encolar(self, elemento):
        self.items.append(elemento)

    def desencolar(self):
        if not self.esta_vacia():
            return self.items.pop(0)
        return None
```

Se eligió porque:
- Una cola respeta el orden en que llegaron los urgentes
- Orden FIFO: primero entra, primero sale
- Evita mezclar urgentes con los mensajes comunes
- Usar una pila no permitia orden FIFO, no era eficiente, al igual que árbol ya que hacia la tarea más complicada.

```python
#  MENSAJES PRIORITARIOS

def procesar_mensajes_prioritarios(self):
    if self.__cola_prioridad.esta_vacia():
        return None
    return self.__cola_prioridad.desencolar()
```

ESTRUCTURA DE DATOS

Grafo

SIRVE PARA COMUNICACIÓN ENTRE SERVIDORES

Se usa en:
Clase ServidorCorreo
Lista __vecinos (las conexiones entre servidores)
Método bfs_ruta() para buscar camino entre servidores
Método enviar_mensaje_red() para enviar un mensaje entre varios servidores

Sirve para:
Tener varios servidores conectados. Cada uno puede estar conectado con otros, y el mensaje tiene que viajar por esos enlaces hasta llegar al servidor donde está el destinatario.
En ese grafo los nodos son los servidores y las aristas las conexiones.

Se eligió porque:
- Permite ubicar servidores.
- Con BFS encontrás rutas de forma más eficiente.
- Es similar a una red real.
- Se puede agregar más servidores y la estructura sigue funcionando.

```python
def bfs_ruta(self, servidor_destino):
    visitados = set()
    cola = deque([(self, [])])

    while cola:
        actual, ruta = cola.popleft()
        if actual == servidor_destino:
            return ruta + [actual]

        if actual not in visitados:
            visitados.add(actual)
            for vecino in actual.get_vecinos():
                cola.append((vecino, ruta + [actual]))
    return None
```

```python
def agregar_conexion(self, servidor_vecino):
    if servidor_vecino not in self.__vecinos:
        self.__vecinos.append(servidor_vecino)
        servidor_vecino.__vecinos.append(self)
```

```python
def enviar_mensaje_red(self, remitente_mail, destinatario_mail, asunto, contenido, prioridad=False):
    remitente = self.buscar_usuario_por_mail(remitente_mail)

    destino = None
    for servidor in self.__vecinos + [self]:
        candidato = servidor.buscar_usuario_por_mail(destinatario_mail)
        if candidato:
            destino = servidor
            break

    if destino is None:
        print("No se encontró el destinatario en la red.")
        return

    ruta = self.bfs_ruta(destino)
    if ruta is None:
        print("No existe ruta entre los servidores.")
        return

    print("Ruta encontrada: " + " , ".join(s.get_nombre() for s in ruta))

    destino_usuario = destino.buscar_usuario_por_mail(destinatario_mail)
    mensaje = Mensaje(remitente_mail, destinatario_mail, asunto, contenido, prioridad)
    remitente.get_enviados().agregar_mensaje(mensaje)
    destino_usuario.recibir_mensaje(mensaje)

    print("Mensaje enviado correctamente a través de la red.")
```

Recursividad en la estructura de carpetas

ESTRUCTURA DE DATOS

La recursividad aparece principalmente en la gestión de carpetas y subcarpetas, usando el árbol general de la clase nodoArbol, donde una carpeta puede contener otras carpetas, y esas carpetas pueden contener más carpetas, y así hasta donde el usuario quiera.

Se eligió porque:
Al no saber cuántos niveles de carpetas va a crear el usuario, usamos la recursión para que realice esa tarea automáticamente: se mete tan profundo como el árbol lo permita, sin que tengamos que escribir bucles anidados.

Eso genera una estructura natural de tipo árbol, donde cada nodo puede tener muchos hijos.
Para trabajar sobre esa estructura, las funciones más importantes son recursivas:

- buscar_subcarpeta(nombre)
    - Si el nombre coincide con la carpeta actual, la carpeta buscada es encontrada.
    - De lo contrario, se llama a sí misma dentro de cada subcarpeta.
- listar_contenido()
    - Muestra la carpeta actual.
    - Después vuelve a llamarse para cada subcarpeta.
    - Así se va recorriendo todo el árbol “hacia abajo”.

Grafos y sus formas de recorridos

ESTRUCTURA DE DATOS

En la entrega 3 necesitabas modelar la red de servidores como un grafo, y permitir que un mensaje viaje de un servidor a otro.
El método usado es: bfs_ruta()
Esto implementa BFS (Breadth-First Search), o sea: búsqueda en anchura.
Porque este modelo garantiza la ruta más corta entre dos servidores.

En una red de correo:
El mensaje debería viajar por el camino más corto disponible.
BFS explora niveles completos antes de seguir bajando.
Eso asegura que apenas encuentra el destino, lo hace con la cantidad mínima de saltos entre servidores.
Es más rápido, más eficiente

DFS en cambio:
Se mete por un camino profundo sin importar si es el más largo.
Podría recorrer media red antes de probar otro camino.
No garantiza la ruta más eficiente.

Relación entre recursividad y BFS
✓ La recursividad aparece en la estructura interna del usuario (árbol de carpetas).
✓ BFS aparece en la comunicación entre servidores (grafo de la red).

```python
class ServidorCorreo:

    def __init__(self, nombre="Servidor de Correo Central"):
        self.__nombre = nombre
        self.__usuarios = []
        self.__vecinos = []          # Servidores conectados (grafo)
```

Cada servidor guarda una lista de servidores vecinos.
Si A está conectado con B, ambos se agregan mutuamente en sus listas(lista de adyacencia).

```python
def agregar_conexion(self, servidor_vecino):
    if servidor_vecino not in self.__vecinos:
        self.__vecinos.append(servidor_vecino)
        servidor_vecino.__vecinos.append(self)
```

Sirve para:
Tener varios servidores conectados. Cada uno puede estar conectado con otros, y el mensaje tiene que viajar por esos enlaces hasta llegar al servidor donde está el destinatario.
En ese grafo los nodos son los servidores y las aristas las conexiones.

```python
def bfs_ruta(self, servidor_destino):
    visitados = set()
    cola = deque([(self, [])])

    while cola:
        actual, ruta = cola.popleft()
        if actual == servidor_destino:
            return ruta + [actual]

        if actual not in visitados:
            visitados.add(actual)
            for vecino in actual.get_vecinos():
                cola.append((vecino, ruta + [actual]))
    return None
```

Este método:
Implementa BFS (Breadth First Search)que es un algoritmo de busqueda en grafos.
Busca una ruta entre dos servidores de la red.

```python
def enviar_mensaje_red(self, remitente_mail, destinatario_mail, asunto, contenido, prioridad=False):
    remitente = self.buscar_usuario_por_mail(remitente_mail)

    destino = None
    for servidor in self.__vecinos + [self]:
        candidato = servidor.buscar_usuario_por_mail(destinatario_mail)
        if candidato:
            destino = servidor
            break

    if destino is None:
        print("No se encontró el destinatario en la red.")
        return

    ruta = self.bfs_ruta(destino)
    if ruta is None:
        print("No existe ruta entre los servidores.")
        return

    print("Ruta encontrada: " + " , ".join(s.get_nombre() for s in ruta))

    destino_usuario = destino.buscar_usuario_por_mail(destinatario_mail)
    mensaje = Mensaje(remitente_mail, destinatario_mail, asunto, contenido, prioridad)
    remitente.get_enviados().agregar_mensaje(mensaje)
    destino_usuario.recibir_mensaje(mensaje)

    print("Mensaje enviado correctamente a través de la red.")
```

El servidor:
Busca al destinatario en toda la red.
Usa BFS para encontrar la ruta más corta.
Entrega el mensaje siguiendo esa ruta virtual.

FUNCIONALIDADES PRINCIPALES DEL PROGRAMA

ESTRUCTURA DE DATOS

1. Gestión de usuarios
Crear (registrar) nuevos usuarios.
Iniciar sesión con verificación de contraseña.
Solicitud de cambio de contraseña en el primer login.
Recuperación mediante pregunta de seguridad.
Mostrar lista de usuarios (solo administrador).
Atributos encapsulados y modificables según permisos.

2. Envío y recepción de mensajes
Enviar mensajes individuales entre usuarios del mismo servidor.
Enviar mensajes a todos los usuarios de un mismo departamento.
Recepción automática de mensajes en la bandeja de entrada.
Soporte para mensajes con prioridad alta.
Fecha automática, adjuntos, destacado, etc.

3. Estructura de carpetas (Árbol general recursivo)
Cada usuario tiene:
Root
Bandeja de Entrada
Enviados
Carpetas nuevas creadas por reglas de filtro.
Posibilidad de:
Agregar subcarpetas recursivamente.
Buscar carpetas recursivamente.
Listar el contenido de todo el árbol (DFS preorden).

4. Gestión de mensajes dentro de carpetas
Insertar mensajes.
Eliminar mensajes y enviarlos a papelera (con pila LIFO).
Recuperar últimos mensajes eliminados.
Mover mensajes entre carpetas.
Buscar mensajes por asunto de manera recursiva.

FUNCIONALIDADES PRINCIPALES DEL PROGRAMA

ESTRUCTURA DE DATOS

5. Filtros automáticos
El usuario puede definir reglas de filtrado:
Por palabra clave.
Por carpeta destino.
Los filtros se aplican automáticamente al recibir mensajes.
Se pueden aplicar también manualmente a todos los mensajes existentes.

6. Cola de prioridad para mensajes urgentes
Los mensajes marcados como prioritarios se almacenan en una ColaPrioridad.
El usuario puede procesar estos mensajes en orden FIFO.

7. Envío de mensajes entre servidores (Grafo con busqueda BFS)
Los servidores pueden conectarse entre sí formando un grafo.
Se utiliza BFS para:
Buscar la ruta más corta entre servidores.
Enviar mensajes a usuarios que están en otros servidores.
Se muestra la ruta utilizada para el envío.

8. Interfaz de línea de comandos (Main)
Permite al usuario interactuar con todas las funcionalidades:
Menús.
Validación de permisos.
Operaciones del uso de correo.

Analisis de complejidad

ESTRUCTURA DE DATOS

EN EL PROGRAMA:

1)Lista enlazada
- **Insertar: O(1)**
- **Eliminar: O(n)**
- **Busqueda: O(n)**

2)Pila:
- **Apilar/desapilar: O(n)**

3) Arbol General:
- **Buscar subcarpeta: O(n)**
- **Listar contenidos recursivos: O(n)**
- **Buscar mensajes por asunto: O(c*m)**

donde c: carpetas. m: mensajes por carpetas

4) Movimientos de mensajes:
- **Eliminar: O(n)**
- **Insertar: O(1)**

O(1) → tiempo constante
O(n) → tiempo lineal
O(n^2) → tiempo cuadrático
O(log n) → tiempo logarítmico
O(n log n) → tiempo cuasi lineal

LA O ES "EL ORDEN DE CRECIMIENTO DEL TIEMPO DE EJECUCIÓN DE UNA TAREA"

Problematicas surgidas en el trabajo

Aprendizaje del manejo de Github, VisualCode, Draw Io

Comprensión de las consignas y el posterior trabajo grupal para llevarlas a cabo.

Al armar la estructura recursiva de carpetas, el árbol fallaba, en varios lados.

Manejo de recuperación de contraseña: En primer lugar se había pensado en código aleatorio de recuperación de contraseña, pero no lo pudimos llevar a cabo, en su lugar se desarrolló una pregunta de seguridad. En este aspecto también surgió al duda donde alojar esa información, se decidió que el servidor no guardara la contraseña, sino que sea el usuario quien las almacene, y el servidor de ser necesario hacia una llamada a la información alojada en Usuario, para mantener la privacidad de los datos de cada usuario. Sin embargo, queda pendiente, la resolución de un segundo problema, ya que si el usuario se olvida la respuesta exacta de la pregunta de seguridad, no hay forma de recuperación, y quedaría bloqueado.

Manejo de los mensajes prioritarios. No se utilizó una heap, sino una cola, falto tiempo de desarrollo

Implementación del grafo. La forma de escritura para hacerlo. Es difícil visualizar los protocolos de conexión entre servidores, sobre todo cuando el externo no se rige por las mismas construcciones que el servidor tuyo.

Grafos: Qué tipo de búsqueda usar. En la abstracción pareciera que para el recorrido es más eficiente en DFS. Al seguir analizándolo, decidimos utilizar los 2 modelos porque habían 2 necesidades, la primera era el recorrido entre carpetas y subcarpetas, el árbol, en ese caso utilizamos recorrido preorder, con DFS, sin embargo al momento de la utilización de grafos en el servidor nos pareció más eficiente el modelo de BFS, ya que en diferentes fuentes consultadas así lo sugerían.

Errores de importación por nombres de archivo

Errores típicos de escritura

Estado inicial de los usuarios

Nos hubiera gustado tener un poco más de tiempo para hacer restricciones, ya que actualmente: Mail solo requiere tener @ y (.); y celular solo se valida con isdigit() y largo 10.

CONCLUSIÓN

ARRANCAR ESTE PROYECTO FUE BASTANTE MÁS DIFÍCIL DE LO QUE IMAGINABA. AL PRINCIPIO NO TENÍA BASES SÓLIDAS DE PROGRAMACIÓN ORIENTADA A OBJETOS NI DE ESTRUCTURAS DE DATOS COMPLEJAS, ASÍ QUE MUCHAS COSAS SE SINTIERON FRUSTRANTES. PERO FRUSTRACIÓN QUE VENÍA JUSTAMENTE DE LAS GANAS DE APRENDER Y QUE NO SALGA.

PERO, JUSTAMENTE POR ESO, EL TRABAJO TERMINÓ SIENDO UN PROCESO DE APRENDIZAJE GRADUAL: AL PRINCIPIO TODO ERA CONFUSO, DESPUÉS ALGUNAS PIEZAS EMPEZARON A ENCAJAR, Y HACIA LA MITAD YA PODÍA RECONOCER PATRONES, JUSTIFICAR DECISIONES Y ENTENDER POR QUÉ ALGO FUNCIONABA O DEJABA DE FUNCIONAR. HUBO MOMENTOS EN LOS QUE CORREGIR ALGO ROMPÍA OTRA COSA, Y TOCÓ REARMAR CÓDIGO VARIAS VECES.

TODO ESO LLEVÓ TIEMPO. Y SINCERAMENTE, AUNQUE EL PROYECTO SE TRABAJÓ DE MANERA PROGRESIVA, LA MAYOR PARTE DEL ESFUERZO FUERTE SE CONCENTRÓ EN LAS ÚLTIMAS SEMANAS. LA BAJA DE COMPAÑEROS DEL GRUPO HIZO QUE SE DEBA REORGANIZAR EL TRABAJO, AVANZAR MÁS RÁPIDO Y RESOLVER COSAS CON MENOS MARGEN.

AUN ASÍ, EL PROYECTO SALIÓ. NO ESTÁ PERFECTO NI ES DEFINITIVO, PERO SÍ ES UNA VERSIÓN SÓLIDA TENIENDO EN CUENTA EL PUNTO DE PARTIDA. SOBRE TODO, ES UNA BASE QUE SE PUEDE SEGUIR MEJORANDO.

EN RESUMEN: FUE UN TRABAJO DESAFIANTE, FRUSTRANTE POR MOMENTOS, PERO MUY FORMATIVO. AHORA ENTIENDO MUCHO MÁS SOBRE CÓDIGO, ESTRUCTURAS DE DATOS, RECURSIÓN, GRAFOS Y CÓMO SE INTEGRAN EN UN SISTEMA REAL. Y AUNQUE ME HUBIERA GUSTADO TENER MÁS TIEMPO PARA DEJARLO AÚN MÁS PROLIJO, ME QUEDO CON QUE APRENDÍ MUCHÍSIMO Y QUE EN EL FUTURO PUEDO SEGUIR CONSTRUYENDO SOBRE ESTA VERSIÓN INICIAL.