Trabajo Integrador de Estructura de datos

CLIENTE DE CORREO

Estudiantes:

Arguello, Joaquin

Garcia Torres, Julene

Hidalgo, Melisa

PRIMERA ENTREGA.

Proceso de formación del trabajo:

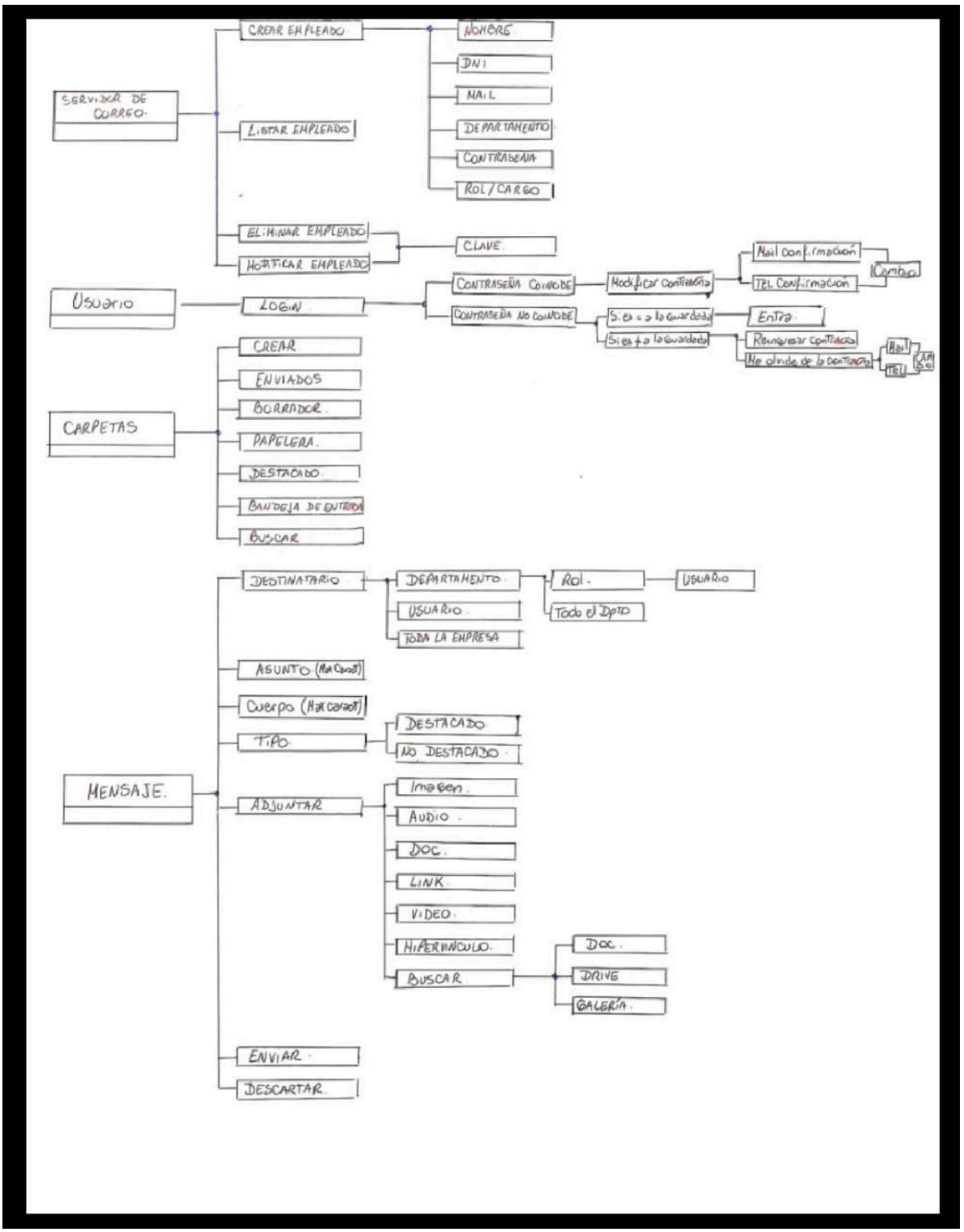
A partir de una reunión que se produjo por vía virtual, se debatió en grupo cuáles eran las necesidades a la hora de crear un cliente de correo.
La primera preocupación se centró en si solo se debía crear el cliente de este correo, es decir el usuario, o también el entorno en el cual ese usuario iba a funcionar.
Tras debatir sobre los distintos enfoques, se eligió avanzar con la segunda opción, aunque quedando abiertos a futuras modificaciones, o cambios de enfoques.

Para trabajar sobre este segundo enfoque, se trabajó sobre la idea de cómo armar la mensajería de una empresa y qué era necesario para eso.
Surgieron las siguientes cuestiones:

- Primero debía existir una empresa que tenga una una base de datos que incluya una lista apropiada de clientes y sus datos personales. Este sería nuestro servidor de correo.
- Sobre esa cuestión, se pensó que si bien los correos eran internos, deben estar protegidos con contraseña.
- Para que los datos y correos sean protegidos por contraseña, al momento de ingresar un nuevo empleado se le debería asignar una cuenta corporativa, la cual tendría su número de documento como usuario y una contraseña designada por default, la cual se debería modificar luego de ingresar al sistema por primera vez.
- Para lograr esto se enviaría de forma automática un mensaje de correo electrónico o un mensaje telefónico.
- Al momento de que el nuevo usuario cambie la contraseña, esta también debería modificarse de forma automática en la base de datos creada en un principio.
- De esta manera, para entrar al sistema y poder escribir un correo el usuario debe loguearse con su dni y la contraseña modificada.
- También se consideró que debía haber alguien que tenga una contraseña especial que permita la creación y modificación de los datos de cada empleado para el caso de incorporaciones, despidos o promociones, por ejemplo RRHH.
- Se pensó en la idea de crear una forma de recuperación en caso de olvido de contraseña.

Luego de eso se analizó qué datos de los empleados debían formar parte del usuario, y se consideraron diferentes especificaciones que servirían de utilidad para la selección de un destinatario, como por ejemplo dni, mail, departamento de trabajo, rol o cargo.
Para la visualización de la página de correo se pensó en crear diferentes carpetas.
Para el mensaje propiamente dicho hay muchas especificaciones a tener en cuenta, por ejemplo en destinatario, existe la opción de filtrar por departamento, rol, o bien se puede optar directamente por agregar el nombre si lo conociese.
Se optó por agregar funciones de envío a todo el departamento y a toda la empresa para mensajes generales de difusión.
También se tuvo en cuenta que tanto el asunto del mensaje, como el cuerpo, deberían estar limitados en caracteres, para que sea eficiente, y se podrían optar por diferentes opciones en cuanto a tipo de información adjuntada, como por ejemplo documentos, imágenes, etc.

El primer boceto del trabajo quedó de la siguiente manera:



Clasificando en clases, atributos y métodos por el momento quedamos en estos:

Class:

ServidorCorreo

Atributo:

Usuario:

lista de usuarios (o empleados)

Métodos:

CrearUsuario

nombre

dni

mail

contraseña

departamento

rol

ListarEmpleados

EliminarEmpleado

contraseña

ModificarEmpleado

contraseña

Class:

Usuario

Atributo:

nombre: str

dni: int

mail: str

contraseña: str

departamento: str

rol: str

carpeta: lista de carpetas

Métodos:

login

mail

contraseña

recuperarContraseña

contraseñaErronea

listarCarpeta (por nombre)

crearCarpeta

Class:

Carpeta

Atributos:

nombre: str

mensaje:

lista de mensajes

Método:

bandejaDeEntrada
listarMensaje
crearMensaje
eliminarMensaje
listarMensaje
enviadosMensajes
listarMensaje
borrador
destadados
buscarMensaje

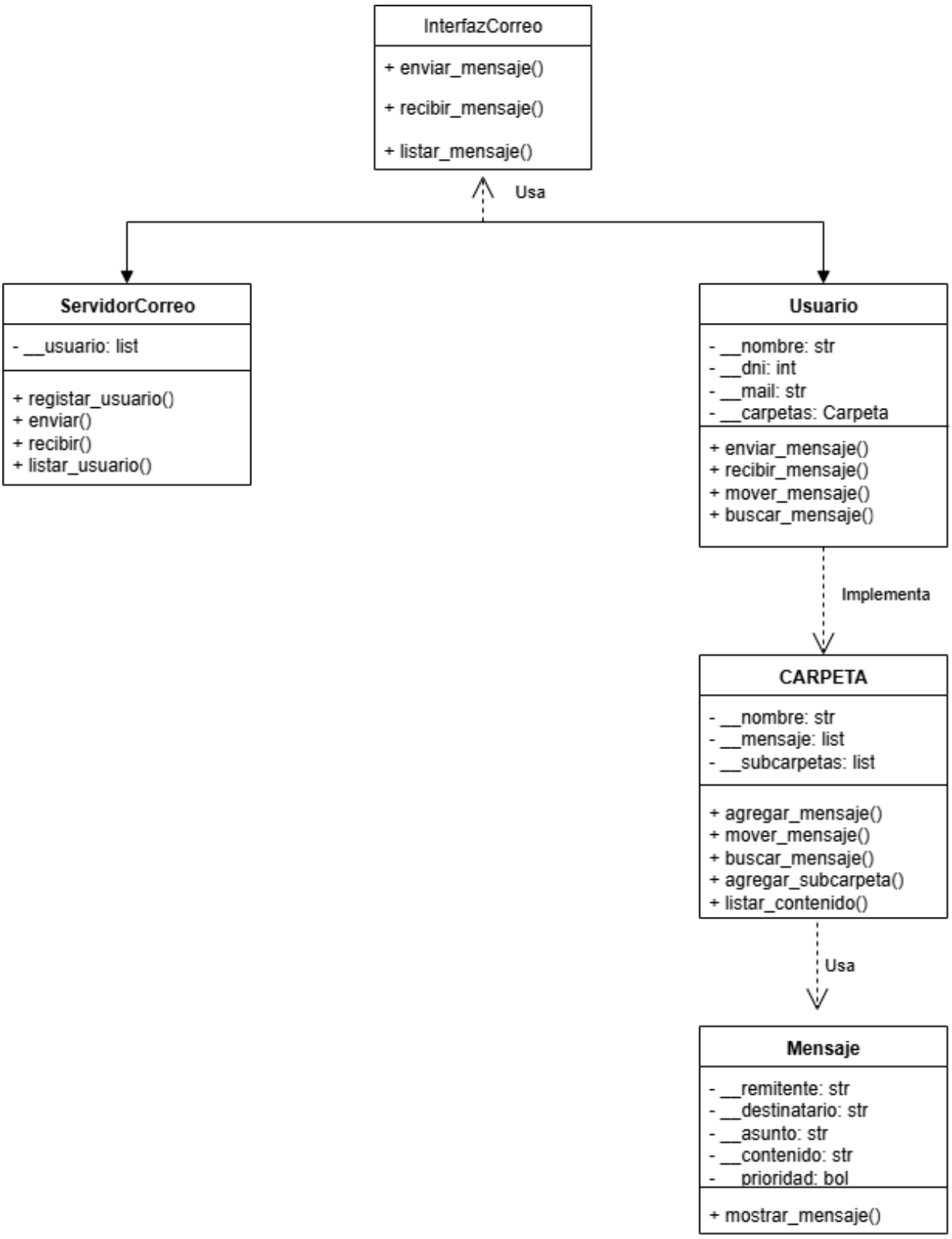
Class:
Mensaje
Atributos:
contraseña
remitente
usuario
destinatario
usuario
lista de usuarios
departamento
asunto
cuerpo
tipo
adjuntar
estado
destacado
no destacado
Método:
enviar
descartar
marcar destacado
adjuntar

En esta primera versión, el grupo elaboró un modelo con el concepto imaginario que teníamos, de acuerdo a la funcionalidad y a la propia experiencia a la hora de usar un correo de este tipo. Se lo tipificó similar en características al correo del campus. La idea ea definir qué atributos, clases, métodos debían tener cada una.

Las clases con las que iniciamos nuestro proyecto fueron: ServidorCorreo, Usuario, Carpeta y Mensaje según lo requerido en las consignas.
No teníamos aún el modelo UML, pero lo agregamos en la segunda entrega.
Se pensó desde el lugar de lo que debía hacer, pero aún no tenía una estructura definida. No tenía tampoco encapsulamientos o modularidad, por lo tanto y así como se indicó en la retroalimentación faltaba la parte de estructura de orientación a objetos.
También estaban sugeridas las dependencias de cada clase, pero no estaba bajada a un código que las reglara.

SEGUNDA ENTREGA:

A partir de los pendientes de la primera entrega, y de la devolución del docente, surgieron problemáticas hacia adentro de nuestro grupo y hacia el trabajo.
En primer lugar, tuvimos la baja de una de nuestras compañeras, Julene Garcia Torres a causa de estar pasando un momento personal difícil, por lo cual se tuvo que reajustar la distribución de las tareas para llegar con progresos significativos a la fecha de entrega.
Al ser ambos nuevos usuarios de Github, tuvimos que sortear varias problemáticas que se desprenden del uso de la plataforma por no estar familiarizada con ella. Como por ejemplo, si las branch se utiliza una por usuario, o si es para diferentes posibles formatos de desarrollo del proyecto principal.
Hacia dentro del proyecto, se nos marcó que faltaba proteger las propiedades, y encapsulamiento. Así como también el diagrama de clases, la interfaz y se requería mejor modularidad.
Buscamos y diseñamos el diagrama de clases según el progreso que iba teniendo el proyecto, hicimos uno al comenzar la segunda etapa, que era el correspondiente a la etapa 1, y una al finalizar, donde se pueden apreciar mejor organización que el modelos anterior.
El siguiente es el primer modelo realizado, el cual corresponde a la entrega 1.



Las relaciones principales que vemos en este gráfico son:

- El ServidorCorreo/Usuario: mantiene una lista y puede agregarlos.
- Usuario/Carpeta: Cada usuario tiene su árbol carpeta.
- Carpeta/Carpeta: Cada carpeta puede tener subcarpetas(recursividad)
- Carpeta/Mensaje: En cada Carpeta hay Mensajes.

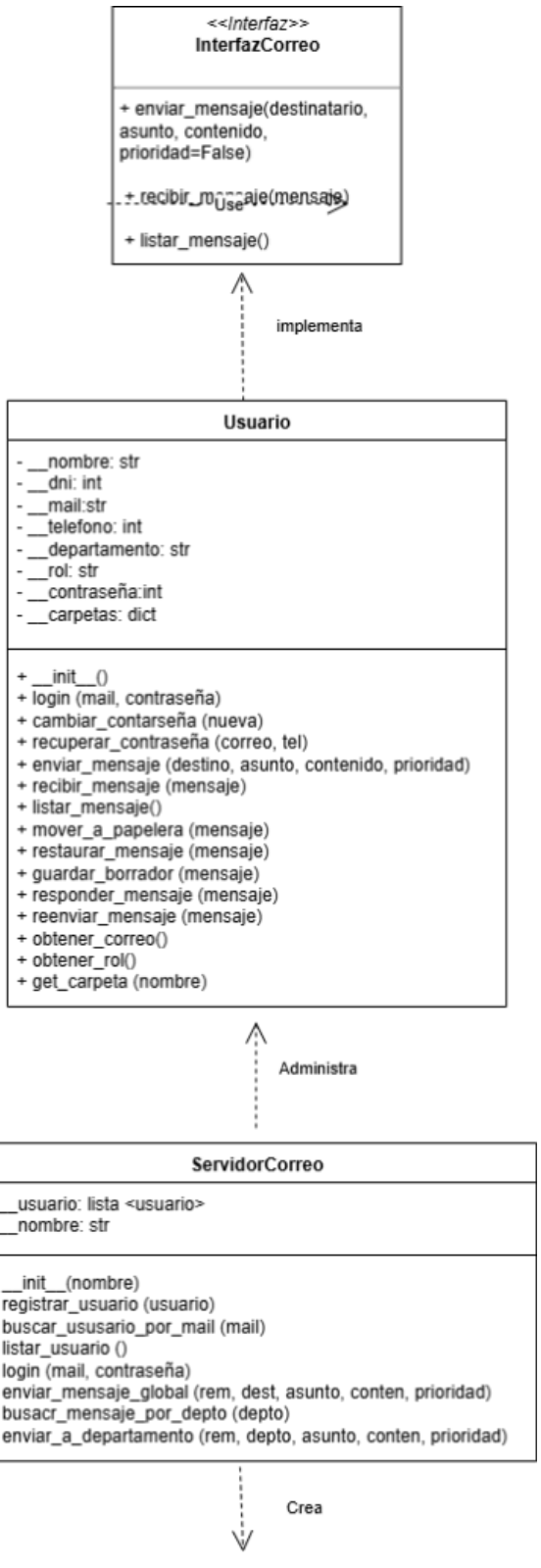
- Usuario/ServidorCorreo: implementa InterfazCorreo.

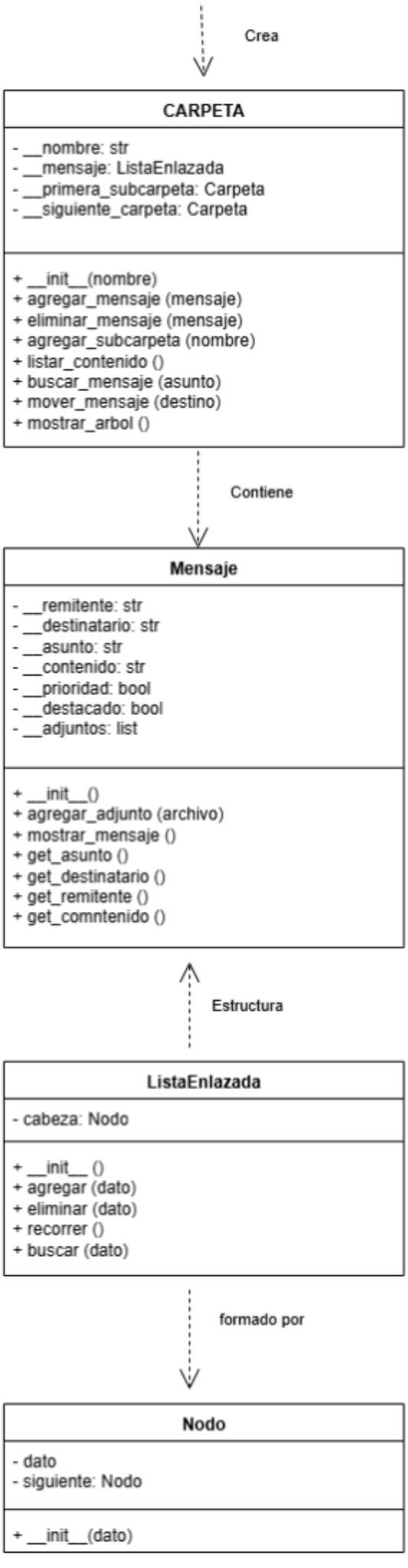
El motivo por el que iniciamos con esta estructura fue que podíamos incorporar recursividad en Carpeta ya sea con movimientos o búsquedas, ServidorCorreo lo incorporamos para gestionar envíos y recepción de mensajes, y que cada uno de los Usuarios podían tener un espacio propio de trabajo, con su Carpeta y Mensajes personales.
Sobreescribiendo ésto, al final de nuestro proyecto podemos darnos cuenta a partir de la evolución de nuestro trabajo cuáles fueron las falencias de este modelo:

- la estructura binaria de las subcarpetas limitaban la recursividad, y la cantidad de subniveles que podían generarse.
- algunas clases dependen de los atributos de otras, por ejemplo ServidorCorreo accedía a atributos privados de Usuario.
- la búsqueda no era recursivas ni jerárquicas, sino lineales lo que sacaba eficiencia.

Cubiertos todos los faltantes de la primera entrega pudimos empezar a desarrollar la parte del código correspondiente a la segunda parte.
Nos encontramos que muchas de las propuestas que habíamos pensado para el proyecto en el diseño original carecían de desarrollo con las herramientas que teníamos, no por falta de material, sino de comprensión de los mismos.
La compañera Melisa Hidalgo, asistió a clases de apoyo para poder organizar un poco los contenidos y así aplicarlos al proyecto.
Tuvimos inconvenientes a la hora de trabajar el repositorio como colaboración, por lo tanto se acordó hacerlo en esta entrega en un documento auxiliar, de forma colaborativa, para después subir los adelantos logrados.
Se empezó a trabajar en el código propiamente dicho, tuvimos que realizar un diagrama de relaciones entre las carpetas para que al modificar una no sea necesario ir y volver en cada una de las carpetas porque se rompía el código en otro lado al hacer una modificación o agregado. Esto costó bastante, porque dependía del gráfico y las conexiones que teorizabamos.
Se creó la main, carpetas y subcarpetas para estructurar mejor en módulos, creamos una carpeta para los documentos de seguimiento y progreso del proyecto y otra para la interfaz, dando por cubiertos esos faltantes. En un principio creamos: main, servidor de correo, usuario, mensaje, carpeta, interfaz_correo, read me y una carpeta de documentos. Pero más adelante nos dimos cuenta que necesitábamos agregar una carpeta extra que contuviera las diferentes estructuras de datos que se iban a utilizar, como por ejemplo, nodos, pilas, etc.

El orden que elegimos para trabajarlas fue: interfaz, mensaje, carpeta, usuario, servidor de correo y finalmente la main.
Comenzamos con la interfaz porque esta se utiliza desde el módulo principal pero no forma parte de la estructura del árbol en sí, en ella estan nombradas clases, pero no definida,es una abstracción. Se deja que las clases concretas definan e implementen la interfaz.
Luego se definieron todas las clases, para finalmente organizar el main, el cual debía inicializar el sistema, crear usuarios, simular acciones de clientes como enviar, recibir, mover o buscar mensajes y mostrar resultados de búsquedas de mensajes o carpetas. El orden responde también a que para utilizar Usuario, es necesario que exista anteriormente Carpeta; para ServidorCorreo se necesita que Usuario exista; y para usar el Main, se necesita que todo esté definido con anterioridad.
Los requisitos para esta segunda etapa eran: gestionar carpetas y subcarpetas como estructura recursiva, mover carpetas, búsqueda recursiva de mensajes por asunto o remitente, analizar la eficiencia de las operaciones, crear una infografía para el árbol de carpetas.
Para ésto como antes mencionamos, diseñamos el main como raíz ejecutable; las demás clases, como por ejemplo, Usuario, Carpeta, Mensaje, etc., serían ramas hijas. Agregamos una carpeta de documentos para la parte de seguimiento del proyecto y colocación de adjuntos, y el read me.
Con el encapsulamiento, se incorporaron los métodos setters y getter para tener acceso a los atributos protegidos de forma privada, tanto para listarlos como para modificarlos. Se utilizan en los atributos __nombre, __mensajes, y __subcarpetas. Se inician al crear cada carpeta y su acceso se hace mediante los métodos getters.
Luego del modelado nuevo, nuestro formato de UML quedo de la siguiente manera:


<<Interfaz>>
InterfazCorreo
+ enviar_mensaje(destinatario, asunto, contenido, prioridad=False)
+ recibir_mensaje(mensaje)
+ listar_mensaje()
implementa
Usuario
- __nombre: str
- __dni: int
- __mail:str
- __telefono: int
- __departamento: str
- __rol: str
- __contraseña:int
- __carpetas: dict
+ __init__()
+ login (mail, contraseña)
+ cambiar_contarseña (nueva)
+ recuperar_contraseña (correo, tel)
+ enviar_mensaje (destino, asunto, contenido, prioridad)
+ recibir_mensaje (mensaje)
+ listar_mensaje()
+ mover_a_papelera (mensaje)
+ restaurar_mensaje (mensaje)
+ guardar_borrador (mensaje)
+ responder_mensaje (mensaje)
+ reenviar_mensaje (mensaje)
+ obtener_correo()
+ obtener_rol()
+ get_carpeta (nombre)
Administra
ServidorCorreo
- __usuario: lista <usuario>
- __nombre: str
+ __init__(nombre)
+ registrar_usuario (usuario)
+ buscar_ususario_por_mail (mail)
+ listar_usuario ()
+ login (mail, contraseña)
+ enviar_mensaje_global (rem, dest, asunto, conten, prioridad)
+ busacr_mensaje_por_depto (depto)
+ enviar_a_departamento (rem, depto, asunto, conten, prioridad)
Crea


Crea
CARPETA
- __nombre: str
- __mensaje: ListaEnlazada
- __primera_subcarpeta: Carpeta
- __siguiente_carpeta: Carpeta
+ __init__(nombre)
+ agregar_mensaje (mensaje)
+ eliminar_mensaje (mensaje)
+ agregar_subcarpeta (nombre)
+ listar_contenido ()
+ buscar_mensaje (asunto)
+ mover_mensaje (destino)
+ mostrar_arbol ()
Contiene
Mensaje
- __remitente: str
- __destinatario: str
- __asunto: str
- __contenido: str
- __prioridad: bool
- __destacado: bool
- __adjuntos: list
+ __init__()
+ agregar_adjunto (archivo)
+ mostrar_mensaje ()
+ get_asunto ()
+ get_destinatario ()
+ get_remitente ()
+ get_comntenido ()
Estructura
ListaEnlazada
- cabeza: Nodo
+ __init__ ()
+ agregar (dato)
+ eliminar (dato)
+ recorrer ()
+ buscar (dato)
formado por
Nodo
- dato
- siguiente: Nodo
+ __init__(dato)

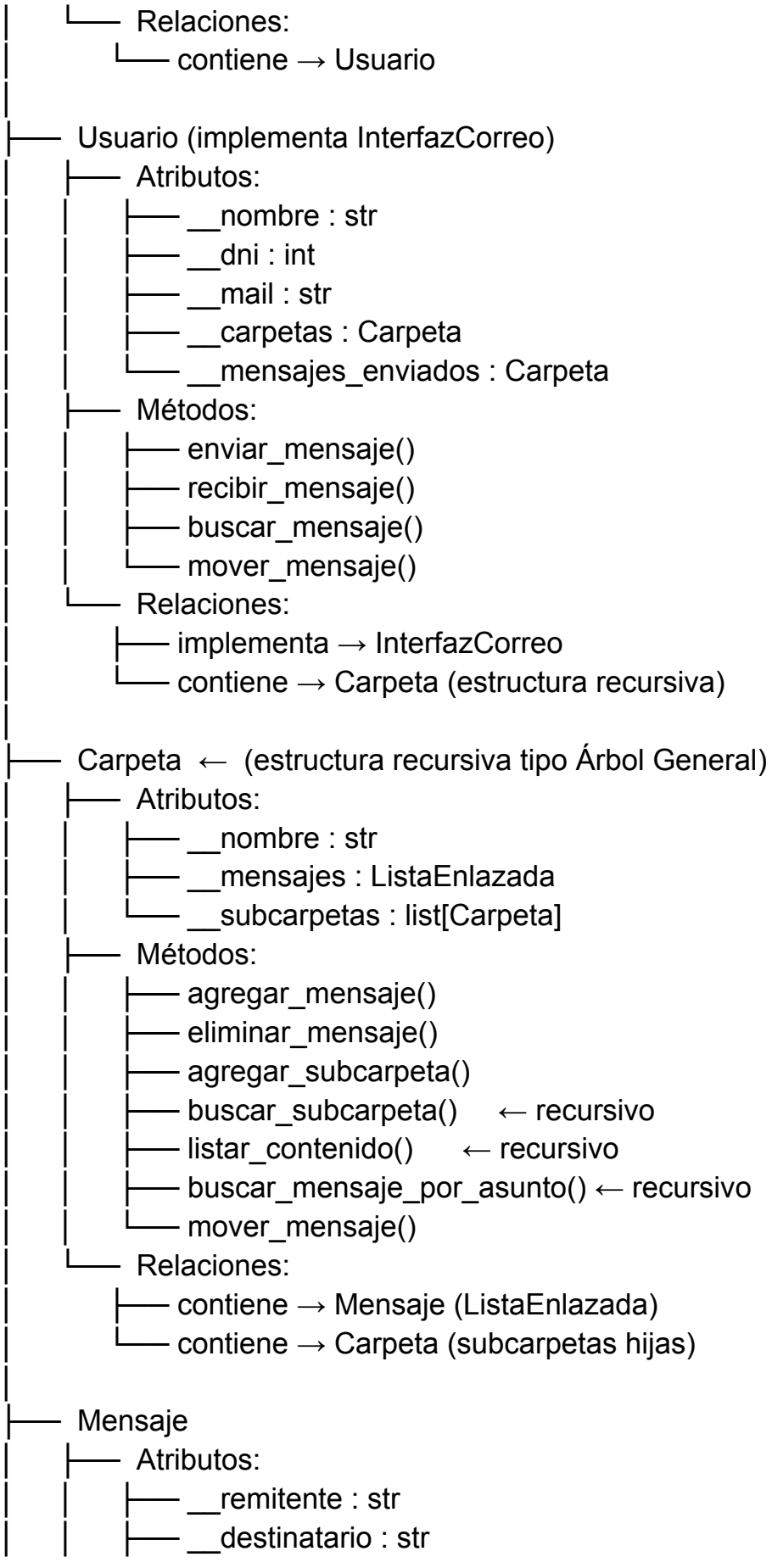
Este nuevo diagrama cumple con el modelo orientado a objetos, se muestra la interfaz; la clase Usuario implementa esa interfaz; ServidorCorreo, Carpeta, Mensaje, ListaEnlazada y nodo aparecen relacionadas más eficientemente. En cuanto a la recursividad, la carpeta está relacionada consigo misma, lo que forma un árbol general recursivo.
Para relacionar sus componentes, en este modelo:

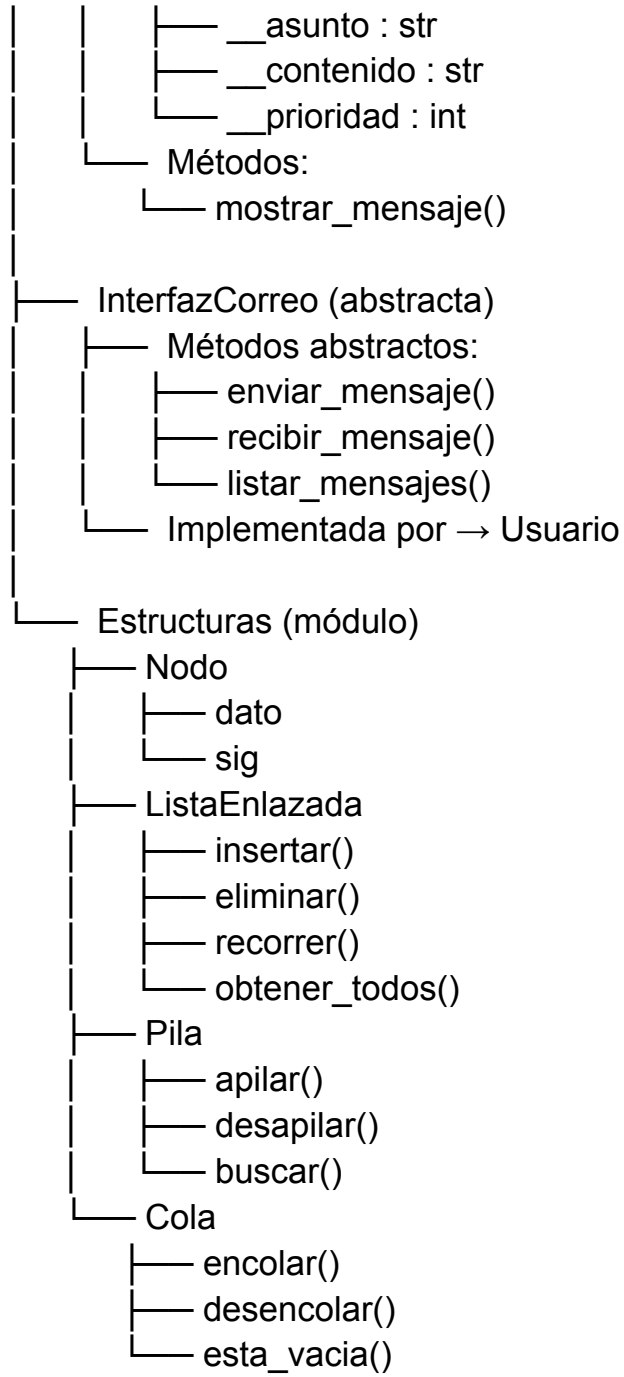
- ServidorCorreo administra usuarios y carpetas
- Usuario contiene carpetas y mensajes
- Carpeta contiene mensajes y otras carpetas
- ListaEnlazada estructura los mensajes dentro de cada carpeta.

ESto permitio que sea más limpio el diagrama, más legible, no mezcla niveles, mantiene atributos y métodos claves, muestra jerarquías, dependencias y relaciones estructurales.
Esto facilitó el desarrollo de nuestro árbol en este formato:

Para la implementación del Árbol general de carpetas, cada Carpeta se modela como un nodo de un árbol, donde el nodo contiene mensajes directamente, y además mantiene una lista de referencia a sus subcarpetas. Así cada carpeta puede tener dentro otras carpetas y estas a su vez, otras subcarpetas más, por eso se convierte en recursiva.
El árbol también se utiliza para búsquedas, donde la estructura recursiva permite formar jerarquías sin límite de profundidad, donde cada nodo pued tener n hijos. Su eficiencia es O(N+M) siendo N= número de carpetas y M(número de Mensajes). Luego cuando lleguemos a la parte de árbol desglosamos su eficiencia.
Lo siguiente, es el

```
ClienteCorreo
│
├── main.py
│   └── instancia y coordina todos los módulos del sistema
│
├── ServidorCorreo
│   ├── Atributos:
│   │   ├── __usuarios : list[Usuario]
│   │   └── __dominio  : str
│   ├── Métodos:
│   │   ├── registrar_usuario()
│   │   ├── enviar_mensaje()
│   │   ├── recibir_mensaje()
│   │   └── listar_usuarios()
```


└── Relaciones:
└── contiene → Usuario
├── Usuario (implementa InterfazCorreo)
├── Atributos:
├── __nombre : str
├── __dni : int
├── __mail : str
├── __carpetas : Carpeta
└── __mensajes_enviados : Carpeta
├── Métodos:
├── enviar_mensaje()
├── recibir_mensaje()
├── buscar_mensaje()
└── mover_mensaje()
└── Relaciones:
├── implementa → InterfazCorreo
└── contiene → Carpeta (estructura recursiva)
├── Carpeta ← (estructura recursiva tipo Árbol General)
├── Atributos:
├── __nombre : str
├── __mensajes : ListaEnlazada
└── __subcarpetas : list[Carpeta]
├── Métodos:
├── agregar_mensaje()
├── eliminar_mensaje()
├── agregar_subcarpeta()
├── buscar_subcarpeta() ← recursivo
├── listar_contenido() ← recursivo
├── buscar_mensaje_por_asunto() ← recursivo
└── mover_mensaje()
└── Relaciones:
├── contiene → Mensaje (ListaEnlazada)
└── contiene → Carpeta (subcarpetas hijas)
├── Mensaje
├── Atributos:
├── __remitente : str
├── __destinatario : str



Lectura del árbol

Raíz: main.py : es el punto de ejecución del sistema.
Primer nivel: ServidorCorreo: administra usuarios, centraliza envíos y recepciones.
Segundo nivel: Usuario: representa cada cuenta de correo individual, implementa la interfaz común.
Tercer nivel: Carpeta: estructura recursiva tipo árbol general (subcarpetas ilimitadas).

Cuarto nivel: Mensaje: unidades de información contenidas en las carpetas.
Módulo auxiliar: Estructuras: contiene las estructuras lineales de soporte (lista enlazada, pila y cola).

Aspectos recursivos
Relación autorreferencial de Carpeta:
Carpeta: [lista de subcarpetas: Carpeta, Carpeta, ...]
Esto es lo que forma el árbol general.

Métodos recursivos funcionales:

buscar_subcarpeta()
listar_contenido()
buscar_mensaje_por_asunto()

Estas operaciones aplican DFS (Depth-First Search) para recorrer la jerarquía.