

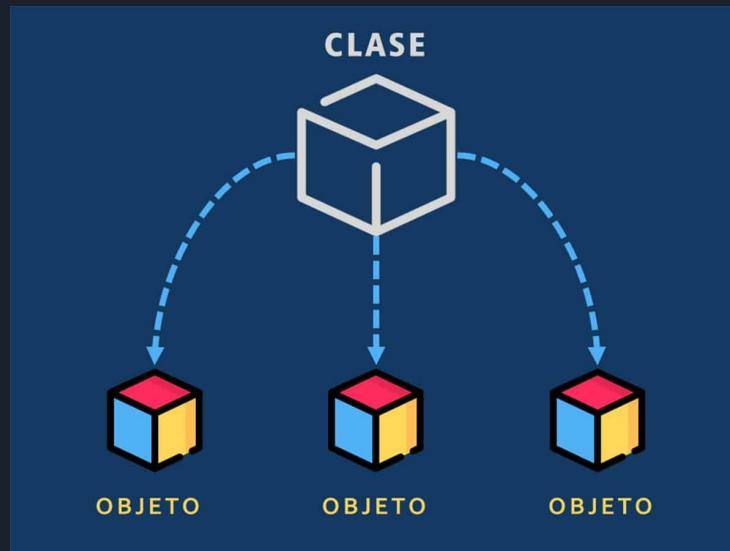


Infografía TP Cliente Correo

UNaB 2025 - Grupo 44

Justificación del diseño

La idea es crear un sistema de gestión de correos con diseño que poseen una arquitectura modular, orientada a objetos. La idea sería que pueda realizar las correspondientes operaciones de forma escalable y mantenible. A continuación se detallan las decisiones clave de diseño y el análisis de eficiencia de las operaciones implementadas.





Estructura general del sistema

- mensaje: Se trata de una clase simple que encapsula los datos de un correo (remitente, destinatario, asunto, cuerpo). No contiene lógica interna lo que facilita su manipulación por las demás clases sin necesitar acoplamiento.
- carpeta: Representa una carpeta de mensajes. Puede contener subcarpetas, formando una estructura recursiva (árbol n-ario). Incluye métodos que permiten agregar, eliminar, mover y buscar mensajes de forma recursiva. Es un sistema que simula de una manera más “cercana” cómo funcionan los correos reales.
- usuarios: Representa cada usuario que se encuentra registrado en el sistema, compuesto por atributos como *nombre*, *email* y *contraseña* con la idea de que cubra de manera simple la autenticación. Cada usuario posee una bandeja de entrada (carpeta) que le permite tener su árbol de mensajes y subcarpetas.
- ServidorCorreo: Administra todos los usuarios que se encuentren registrados, permitiendo su acceso por email mediante un diccionario con la idea de que fuera lo más eficiente posible en la autenticación y envío de mensajes.
- funciones_usuario: Agrupa todas las operaciones que los usuarios pueden realizar (iniciar sesión, enviar mensajes, crear subcarpetas, buscar, etc.), separando la lógica de interacción de la lógica de datos, este separamiento nos permite que en caso de expandir sobre las funciones no se afecte la estructura interna de las clases principales.
- menú: Controla la navegación del sistema en consola, gestiona sesiones y opciones disponibles según el estado del usuario. Es el punto de entrada del sistema, se separó para que no se mezcle la presentación con la lógica.

Complejidad de las operaciones (parte 1)

Clase	Método	Complejidad	Justificación breve
mensaje	constructor	O(1)	Solo guarda datos
usuarios	existente	O(1)	Comparación directa
usuarios	recibir	O(1)	Llama a agregar
ServidorCorreo	acceso por email	O(1)	Diccionario como estructura de búsqueda
funciones_usuario	búsqueda, envío, movimiento	Depende de carpeta	Delegan operaciones a métodos recursivos

Complejidad de las operaciones (parte 2)

Clase	Método	Complejidad	Justificación breve
carpeta	agregar_mensaje	O(1)	Solo guarda datos
carpeta	eliminar_mensaje	O(n)	Comparación directa
carpeta	mover_mensaje	O(n)	Llama a agregar
carpeta	buscar	O(n + m)	Diccionario como estructura de búsqueda
carpeta	listar	O(k)	Delegan operaciones a métodos recursivos



Complejidad de las operaciones explicadas

- agregar_mensaje: $O(1)$ Inserta el mensaje al final de la lista. Se trata de una operación constante.
- eliminar_mensaje: $O(n)$ Busca el mensaje en la lista y lo elimina. La búsqueda y la eliminación son lineales con respecto respecto al número de mensajes.
- mover_mensaje: $O(n)$ Depende de eliminar_mensaje ($O(n)$) y agregar_mensaje ($O(1)$). La operación completa en si es lineal.
- agregar_subcarpeta: $O(k)$ Recorre la lista de subcarpetas para evitar duplicaciones antes de agregar. La búsqueda es lineal respecto al número de subcarpetas existentes.
- obtener_subcarpeta: $O(k)$ Búsqueda lineal por nombre en la lista de subcarpetas.
- eliminar_subcarpeta: $O(k)$ Búsqueda lineal y eliminación por índice.
- buscar(asunto, remitente): $O(n + m)$ Recorre todos los mensajes en la carpeta actual (n) y llama recursivamente a buscar en cada subcarpeta (m). La complejidad total depende del número de mensajes existentes en la estructura completa.
- listar(nivel): $O(k)$ Recorre recursivamente todas las subcarpetas para imprimir la jerarquía. La complejidad depende del número total de carpetas.

Árbol de clases



- La estructura de carpetas implementa un árbol n-ario, donde cada nodo (carpeta) puede tener múltiples hijos (subcarpetas). Esta decisión permite representar jerarquías de forma y flexible, como se muestra en la imagen de ejemplo.
- Las operaciones que recorren el árbol, como buscar y listar, se implementan de manera recursiva. Nos permite aplicar la misma lógica a cada nivel sin tener que estar repitiendo código.