Justificación Estructuras

1. `std::list` (La "Base de Datos" en Memoria)

- **Estructuras:**
    1. `std::list<Cliente> listaMaestraClientes`
    2. `std::list<Cuenta> listaMaestraCuentas`
    3. `std::list<Ventanilla> todasLasVentanillas`
- **Ubicación:** Todas son atributos privados de `SistemaBancario.h`.
- **Funcionamiento y Propósito:** Estas tres listas actúan como la **fuente única de verdad** del programa. Son la "base de datos" que reside en la memoria RAM. Su propósito es **almacenar los objetos completos** (`Cliente`, `Cuenta`, `Ventanilla`) que se cargan desde los archivos `.txt` al iniciar el programa. Se eligió `std::list` (una lista doblemente enlazada) sobre `std::vector` por una razón clave de estabilidad de punteros: cuando se añaden elementos a una `std::list` (`emplace_back`), los elementos que ya están en la lista **no cambian su dirección de memoria**.
- **Énfasis en Punteros (El Concepto Clave):** Este es el pilar del diseño. Estas listas **NO almacenan punteros**, almacenan los **objetos reales**. Esto es fundamental para una gestión de memoria segura y automática (evitando `new` y `delete` manuales).
    1. Cuando se carga un cliente, el objeto `Cliente` "vive" físicamente dentro de un nodo de la `listaMaestraClientes`.
    2. Como la dirección de ese objeto nunca cambiará, podemos obtener un **puntero** (`Cliente*`) que apunta directamente a él.
    3. Este puntero seguro es el que se comparte con otras estructuras (como las colas).
    4. Al cerrar el programa, el destructor de `std::list` se encarga automáticamente de destruir todos los objetos que contiene, previniendo cualquier fuga de memoria (*memory leaks*).

---

2. `std::priority_queue` (La Fila de Espera)

- **Estructura:** `std::priority_queue<Cliente*, std::vector<Cliente*>, ComparadorClientePtr> filaDeEspera`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.

- **Funcionamiento y Propósito:** Esta es la cola de atención principal. No es una cola simple, sino una **Cola de Prioridad**, implementada internamente como un *Max-Heap*. Su función es **ordenar automáticamente** a los clientes. Cada vez que se añade un cliente (`push()`), la estructura utiliza la lógica del `ComparadorClientePtr` para reordenarse, asegurando que el cliente con la prioridad más alta (ej. `VIP = 1`) y el tiempo de llegada (`time_t`) más antiguo esté siempre en la cima (`top()`).
- **Énfasis en Punteros (Eficiencia y Conexión):** Esta estructura almacena **punteros (`Cliente*`)**.
    1. **Eficiencia:** Es computacionalmente mucho más "barato" y rápido ordenar y mover punteros (direcciones de memoria de 8 bytes) que copiar y mover objetos `Cliente` completos.
    2. **Conexión:** El puntero `Cliente*` que se almacena aquí **apunta al objeto real** que "vive" en la `listaMaestraClientes`. No se gestionan copias; se gestionan "referencias" a los clientes reales. Al llamar a `pop()`, se obtiene el puntero al cliente más importante, el cual se pasa a una `Ventanilla` para que atienda al objeto original.

---

3. `std::queue` (La Fila de Ventanillas Libres)

- **Estructura:** `std::queue<Ventanilla*> ventanillasLibres`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.
- **Funcionamiento y Propósito:** Esta es una **Cola simple FIFO** (First-In, First-Out). Su función es administrar las ventanillas que no están ocupadas. Al iniciar el banco, todas las ventanillas se añaden (`push()`) a esta cola. Cuando un cliente necesita ser atendido, se toma una ventanilla del frente (`front()`). Cuando esa ventanilla termina su atención (`liberarCliente()`), se "forma" de nuevo al final de la cola (`push()`). Esto garantiza una distribución de trabajo equitativa (similar a *Round-Robin*).
- **Énfasis en Punteros (Gestión de Estado):** Al igual que la cola de clientes, esta almacena **punteros (`Ventanilla*`)**. Los punteros apuntan a los objetos `Ventanilla` reales que viven en la `std::list todasLasVentanillas`. Se usan punteros para poder **modificar el estado del objeto original** (ej. `ventanilla->atenderCliente(cliente)` cambia el estado `libre` a `false`), en lugar de modificar una copia inútil.

---

4. `std::stack` (La Pila de Deshacer)

- **Estructura:** `std::stack<Transaccion> pilaDeshacer`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.
- **Funcionamiento y Propósito:** Esta es una **Pila LIFO** (Last-In, First-Out). Su único propósito es implementar la función "Deshacer última transacción". Cuando se realiza una operación *reversible* (Retiro o Transferencia), se "empuja" (`push()`) una **copia** de la transacción a la pila. Si se selecciona "Deshacer", el sistema mira la cima (`top()`) para obtener los datos de la transacción a revertir, y luego la elimina (`pop()`).
- **Énfasis en Punteros (¡Importante! NO se usan punteros):** A diferencia de las colas, esta estructura almacena **objetos `Transaccion` completos** (copias), no punteros. Esta es una decisión de diseño intencional. Se necesita una "foto" (*snapshot*) de la transacción tal como ocurrió. No se desea un puntero a una transacción que podría cambiar, sino el registro exacto de los datos que deben ser revertidos.

---

5. `std::deque` (El Historial de Transacciones)

- **Estructura:** `std::deque<Transaccion> historialTransacciones`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.
- **Funcionamiento y Propósito:** Esta es una **Cola de Doble Extremo** (Deque). Se utiliza como el registro histórico principal. Se eligió `std::deque` en lugar de `std::vector` o `std::list` por su eficiencia única: permite añadir elementos al final (`push_back()`) y *también* eliminar elementos del inicio (`pop_front()`) en tiempo constante (O(1)). Esto permite implementar un "historial rotativo": si el historial supera el `MAX_HISTORIAL`, se puede eliminar la transacción más antigua del frente (`pop_front()`) de forma muy eficiente.
- **Énfasis en Punteros (Almacenamiento de Copias):** Al igual que la Pila de Deshacer, el historial almacena **copias de objetos `Transaccion`**, no punteros. El motivo es el mismo: el historial es un registro de eventos pasados. Se almacenan los datos reales de lo que sucedió, no referencias.
-