1. `std::list` (Las "Listas Maestras" o Bases de Datos en RAM)

- **Estructuras:**
    1. `std::list<Cliente> listaMaestraClientes`
    2. `std::list<Cuenta> listaMaestraCuentas`
    3. `std::list<Ventanilla> todasLasVentanillas`
- **Ubicación:** Todas son atributos privados en `SistemaBancario.h`.
- **Funcionamiento (Por qué se usó):** Estas tres listas son la **fuente única de verdad** de tu programa. Actúan como la base de datos que vive en la memoria RAM. Su único propósito es **almacenar los objetos completos** (`Cliente`, `Cuenta`, `Ventanilla`) que se cargan desde los archivos `.txt`. Elegimos `std::list` (una lista doblemente enlazada) porque es extremadamente eficiente (tiempo constante, O(1)) para añadir nuevos elementos al final (`emplace_back`) a medida que los cargamos o creamos.
- **Énfasis en Punteros (El Concepto Clave):** Este es el punto más importante: estas listas **NO almacenan punteros**, almacenan los **objetos reales**. Esta es una decisión de diseño fundamental para la **gestión de la memoria** (evitar *memory leaks*).
    1. Cuando cargamos un cliente, el objeto `Cliente` "vive" físicamente dentro de un nodo de la `listaMaestraClientes`.
    2. Debido a que los objetos viven aquí permanentemente (mientras el banco exista), podemos obtener de forma segura un **puntero** (ej. `Cliente*`) que apunta a esa dirección de memoria.
    3. Este puntero es el que pasamos a otras estructuras (como las colas).
    4. Cuando el programa se cierra, el destructor de `SistemaBancario` llama al destructor de `std::list`, que automáticamente destruye todos los objetos `Cliente` que contiene. **Esto nos da manejo automático de memoria sin usar `delete`.**

---

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

- **Estructura:** `std::priority_queue<Cliente*, std::vector<Cliente*>, ComparadorClientePtr> filaDeEspera`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.
- **Funcionamiento (Por qué se usó):** Esta es la cola de atención principal. Es una **Cola de Prioridad** (implementada internamente como un *Heap*). Su función es ordenar automáticamente a los clientes. Cada vez que añadimos un cliente

(`push()`), la estructura usa nuestra lógica (`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 cola almacena **punteros (`Cliente*`)**.
    1. **Eficiencia:** Es muchísimo más rápido (eficiente) ordenar y mover punteros (que son solo direcciones de memoria, 8 bytes) que copiar y mover los objetos `Cliente` completos.
    2. **Conexión:** El puntero `Cliente*` que almacenamos aquí **apunta al objeto real** que "vive" en la `listaMaestraClientes`. No estamos gestionando copias, estamos gestionando "referencias" a los clientes reales. Cuando llamamos a `pop()`, obtenemos el puntero al cliente más importante, y podemos pasar ese puntero 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 (Por qué se usó):** Esta es una **Cola simple FIFO** (First-In, First-Out). Su función es gestionar las ventanillas que no están ocupadas. Cuando el banco inicia, todas las ventanillas se añaden (`push()`) aquí. Cuando un cliente necesita ser atendido (`procesarFila`), tomamos una ventanilla del frente (`front()`). Cuando esa ventanilla termina, se "forma" de nuevo al final de la cola (`push()`). Esto garantiza una distribución de trabajo justa (tipo *Round-Robin*).
- **Énfasis en Punteros (Modificación del Objeto Real):** Al igual que la cola de prioridad, esta almacena **punteros (`Ventanilla*`)**. Los punteros apuntan a los objetos `Ventanilla` reales que viven en la `std::list todasLasVentanillas`. Usamos punteros para poder modificar el estado del objeto original (ej. `ventanilla->atenderCliente(cliente)`), y no una copia.

---

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

- **Estructura:** `std::stack<Transaccion> pilaDeshacer`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.
- **Funcionamiento (Por qué se usó):** Esta es una **Pila LIFO** (Last-In, First-Out). Su propósito es implementar la función "Deshacer última transacción". Cuando se realiza una operación *reversible* (como un Retiro o Transferencia), "empujamos" (`push()`) una **copia** de la transacción a la pila. Si el usuario selecciona "Deshacer",

simplemente miramos la cima (`top()`) para obtener los datos y revertirlos, y luego la quitamos (`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. Queremos que la pila contenga una "foto" (un *snapshot*) de la transacción tal como ocurrió. No queremos un puntero a una transacción que podría cambiar; queremos el registro exacto de lo que se debe revertir.

---

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

- **Estructura:** `std::deque<Transaccion> historialTransacciones`
- **Ubicación:** Atributo privado en `SistemaBancario.h`.
- **Funcionamiento (Por qué se usó):** Esta es una **Cola de Doble Extremo** (Deque). La usamos como el registro histórico principal. Elegimos `std::deque` en lugar de `std::vector` o `std::list` por su eficiencia única: nos permite añadir elementos al final (`push_back()`) y *también* eliminar elementos del inicio (`pop_front()`) en tiempo constante (O(1)). Esto nos permite implementar un "historial rotativo": si el historial supera el `MAX_HISTORIAL` (ej. 50 transacciones), podemos 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. Almacenamos los datos reales de lo que sucedió, no referencias.