**Documentación del Sistema de Gestión para Asesores de Seguros**

**1. Introducción y Objetivo del Sistema**

El proyecto es un **Sistema de Gestión para Asesores de Seguros**, desarrollado como un Trabajo Práctico Integrador. El objetivo principal del sistema es proporcionar a los productores de seguros una herramienta eficaz para administrar su cartera de clientes, así como llevar un registro y seguimiento integral de las pólizas y los siniestros asociados.

El sistema está diseñado para centralizar la información y automatizar tareas comunes como el control de vencimientos de pólizas y la generación de informes de recaudación, permitiendo al usuario realizar altas, bajas y modificaciones (ABM) de los datos críticos para su negocio.

**2. Arquitectura y Diseño del Sistema**

El sistema sigue un diseño orientado a objetos, donde la lógica se encapsula en distintas clases, cada una con una responsabilidad específica. La arquitectura general se puede dividir en tres capas principales:

1. **Clases de Entidad (Modelo de Datos):** Representan los objetos del mundo real que el sistema gestiona. Estas son: Cliente, Poliza, Vehiculo, Siniestro, Compania y Fecha.
2. **Clases de Acceso a Datos (Gestores de Archivos):** Se encargan de la persistencia de los datos. Cada clase de entidad tiene una clase "Archivo" asociada (ej. ArchivoCliente) que maneja la lectura y escritura de los datos en archivos binarios (.dat). Esto separa la lógica de negocio de la manipulación de archivos.
3. **Capa de Presentación (Interfaz de Usuario):** Implementada a través de menús de consola en main.cpp. Esta capa interactúa con el usuario y utiliza las clases de acceso a datos para ejecutar las operaciones solicitadas.

El siguiente diagrama de clases ilustra las relaciones entre las principales entidades del sistema:

**3. Descripción Detallada de Clases y Métodos**

A continuación, se detalla el propósito y funcionamiento de cada clase y sus métodos más importantes.

**3.1. Clases de Entidad**

Estas clases definen la estructura de los datos que maneja el sistema.

**Clase Fecha**

* **Descripción:** Representa una fecha con día, mes y año. Es utilizada por otras clases como Poliza para manejar fechas de inicio y vencimiento.
* **Archivos:** Fecha.h, Fecha.cpp
* **Atributos:**
  + \_dia, \_mes, \_anio (enteros).
* **Métodos Principales:**
  + Fecha(): Constructor que inicializa la fecha en cero.
  + cargar(): Pide al usuario que ingrese día, mes y año por consola.
  + toString() const: Devuelve la fecha como una cadena de texto en formato "DD/MM/AAAA".

**Clase Cliente**

* **Descripción:** Almacena toda la información personal de un cliente.
* **Archivos:** Cliente.h, Cliente.cpp
* **Atributos:** idCliente, nombre, apellido, dni, fechaNacimiento, domicilio, email, telefono y un booleano activo para bajas lógicas.
* **Métodos Principales:**
  + cargarId(): Solicita el ID del cliente, que funciona como clave principal.
  + cargarDatos(): Pide al usuario el resto de los datos del cliente y establece el estado activo en true.
  + mostrar() const: Imprime en pantalla todos los datos del cliente de forma formateada.

**Clase Vehiculo**

* **Descripción:** Contiene los datos específicos de un vehículo asegurado.
* **Archivos:** vehiculo.h, vehiculo.cpp
* **Atributos:** idVehiculo, marca, version, anio, patente y activo.
* **Métodos:** Similar a Cliente, posee los métodos cargarId(), cargarDatos() y mostrar() para la entrada y salida de datos.

**Clase Compania**

* **Descripción:** Representa a una compañía de seguros.
* **Archivos:** Compania.h, Compania.cpp
* **Atributos:** \_id, \_nombre, \_comision y \_activo.
* **Métodos:** Incluye cargarId(), cargarDatos() y mostrar() para la gestión de su información.

**Clase Poliza**

* **Descripción:** Es una de las clases centrales del sistema. Vincula a un cliente (\_dni, \_nombre, \_apellido) con un vehículo (\_patente) y define los términos del seguro.
* **Archivos:** Poliza.h, Poliza.cpp
* **Atributos:** \_numeroPoliza, \_dni, \_patente, \_fechaInicio, \_fechaVencimiento, \_nombre, \_apellido, \_detalleCobertura, \_importeMensual y \_activo.
* **Métodos:** También implementa cargarId(), cargarDatos() y mostrar(). cargarDatos() reutiliza la clase Fecha para cargar las fechas de la póliza.

**Clase Siniestro**

* **Descripción:** Registra un siniestro ocurrido, asociándolo a una póliza existente.
* **Archivos:** siniestro.h, siniestro.cpp
* **Atributos:** idSiniestro, desc\_siniestro, monto\_reclamo, id\_poliza (para vincularlo a una Poliza) y activo.
* **Métodos:** cargarId(), cargarDatos() y mostrar() para la gestión de su información.

**3.2. Clases de Acceso a Datos**

Estas clases implementan la lógica de persistencia, guardando y recuperando los objetos de las clases de entidad en archivos .dat. Todas siguen un patrón similar.

**Clase ArchivoCliente**

* **Archivos:** ArchivosClientes.h, ArchivosClientes.cpp
* **Métodos Principales:**
  + ArchivoCliente(const char \*n): Constructor que establece el nombre del archivo (por defecto "clientes.dat").
  + agregarRegistro(): Crea un nuevo objeto Cliente, verifica que el ID no exista llamando a buscarCliente(), carga sus datos y lo escribe al final del archivo.
  + buscarCliente(int idCliente): Busca un cliente por su ID. Devuelve la posición en el archivo si lo encuentra, o un valor negativo si no existe o hay un error.
  + listarRegistros(): Lee el archivo secuencialmente y muestra todos los clientes cuyo estado es activo.
  + modificarDatosCliente(): Pide un ID, lo busca, y si lo encuentra, permite al usuario volver a cargar los datos del cliente para sobreescribir el registro existente.
  + bajaLogica(int idCliente): Busca un cliente por ID y cambia su atributo activo a false, en lugar de borrarlo físicamente del archivo.

Las clases ArchivoPoliza, ArchivoVehiculo, ArchivoSiniestro y ArchivoCompania funcionan de manera análoga a ArchivoCliente, cada una gestionando su respectivo tipo de objeto y archivo .dat.

**Métodos Destacados de ArchivoPoliza**

Además de las operaciones ABM, ArchivoPoliza contiene la lógica para los informes:

* **Archivos:** ArchivosPoliza.h, ArchivosPoliza.cpp
* listarProximosVencimientos(int diasAnticipacion): Compara la fecha de vencimiento de cada póliza activa con la fecha actual del sistema y lista aquellas que están por vencer en el rango de días especificado.
* listarPolizasConSiniestros(): Cruza información entre polizas.dat y siniestros.dat. Por cada póliza, recorre el archivo de siniestros buscando coincidencias por ID de póliza y las muestra.
* calcularRecaudacionAnual(int anio): Suma el \_importeMensual (multiplicado por 12) de todas las pólizas que estuvieron activas durante el año consultado.
* calcularRecaudacionMensual, Semanal, y Personalizada: Filtran las pólizas según el período de tiempo (mes, semana o un rango de fechas) y suman sus importes para calcular la recaudación.

**4. Interfaz de Usuario y Flujo del Programa**

El flujo del programa es controlado desde el archivo main.cpp.

* **main():** La función principal que únicamente llama a menuPrincipal().
* **menuPrincipal():** Muestra las opciones principales del sistema (Clientes, Pólizas, Siniestros, etc.). Utiliza un switch para derivar al submenú correspondiente según la opción del usuario.
* **Submenús (menuClientes(), menuPolizas(), etc.):** Cada función de submenú se encarga de una entidad específica. Crean una instancia del gestor de archivos correspondiente (ej. ArchivoCliente archivoCli;) y llaman a sus métodos según la opción elegida por el usuario (Listar, Agregar, Modificar, Eliminar).
* **preguntarVolverMenuPrincipal():** Una función de utilidad que, después de cada operación, pregunta al usuario si desea volver al menú principal o realizar otra operación en el mismo submenú, mejorando la usabilidad.

**5. Conclusión**

El sistema presenta una arquitectura robusta y bien segmentada que cumple con los objetivos planteados. La separación entre la lógica de datos, la persistencia en archivos y la interfaz de usuario facilita el mantenimiento y la escalabilidad del proyecto. Los métodos implementados cubren todas las funcionalidades requeridas para una gestión básica pero completa de una cartera de seguros, incluyendo operaciones ABM, control de vencimientos y generación de informes clave para el negocio.