

## Documento Soporte

Juan Diego Acosta Molina, Laura Daniela Cubillos Escobar, Santiago Sánchez Moya

### Diseño de la solución:

Se plantea una solución al problema utilizando una estructura multilista, listas enlazadas y colas, estas se utilizan para poder organizar la información dentro del programa. Para realizar la implementación de las consultas se hace uso de árboles roji-negros para poder recolectar y organizar la información almacenada en la multilista según el objetivo de la consulta. Se hace uso de archivos planos para realizar la lectura y escritura de la información en una memoria secundaria de largo plazo. Las librerías para las diversas estructuras de datos se crearon con anterioridad, la estructura multilista fue diseñada especialmente para solucionar el problema planteado

En primer lugar, se definen las clases para organizar la información del programa, se hace uso de clases para las sucursales, los empleados, los hijos. Estos serán los objetos base con los cuales se construyan las diferentes listas presentes en la memoria principal.

Para poder consignar la información de los objetos en memoria no volátil, se hace uso de las clases “Archivos” la cual es capaz de escribir una línea en un archivo y leerla, la clase “ObjetosToarchivos” se encarga de tomar una línea leída del archivo y transformarla en un objeto. Por último, la clase “gestionPersistencia” tomará una cola de objetos elaborada tras leer todos los registros de un archivo y los asignará a la lista o multilista respectiva.

Para permitir un fácil uso del programa existen las clases “Controlador” y “Vista” las cuales se encargan de gestionar la entrada y salida de información en la terminal de texto

Existe la librería “Edad” para permitir calcular la edad de las diferentes personas presentes en los archivos basadas en su fecha de nacimiento.

### Estructura del programa:

#### Estructura de los archivos planos:

Existen los archivos planos “Hijos” “Empleados” y “Sucursales” los cuales están estructurados dependiendo de la información que contienen

Empleado (nombre, apellido, tipo de documento, correo electrónico, fecha de nacimiento, ciudad de nacimiento, país de nacimiento, ciudad de residencia, dirección, barrio, actividad laboral sucursal de trabajo, numero de documento, sexo, teléfono fijo, teléfono celular, edad)

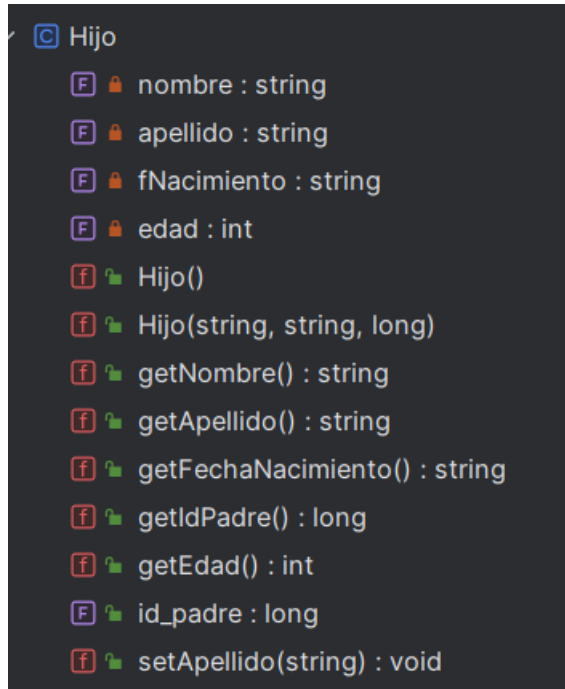
Sucursal (Nombre, dirección, barrio, gerente)

Hijo (nombre, fecha de nacimiento, ID del padre)

Clases que representan información dentro del programa:

Hijo.

Los atributos y métodos de la clase Hijo son los siguientes



```
class Hijo {
    nombre : string
    apellido : string
    fNacimiento : string
    edad : int
    Hijo()
    Hijo(string, string, long)
    getNombre() : string
    getApellido() : string
    getFechaNacimiento() : string
    getIdPadre() : long
    getEdad() : int
    id_padre : long
    setApellido(string) : void
}
```

Empleado:

Los atributos de la clase empleado son los siguientes:

```
C Empleado
(F) nombre : string
(F) apellido : string
(F) tipold : string
(F) correo : string
(F) fNa : string
(F) ciudadNa : string
(F) paisNa : string
(F) ciudadRe : string
(F) direccion : string
(F) barrio : string
(F) actividadLaboral : string
(F) nomSucursal : string
(F) numld : long
(F) sexo : char
(F) cel : int
(F) fijo : int
(F) edad : int
(F) hijos : Lista<Hijo>

(F) sigSexo : Empleado *
(F) sigCiudadNa : Empleado *
(F) sigPaisNa : Empleado *
(F) sigCiudadRe : Empleado *
(F) sigBarrio : Empleado *
(F) sigAct : Empleado *
(F) sigEdad : Empleado *
(F) sigNumHijos : Empleado *
(F) sigNomSucursal : Empleado *
(F) antSexo : Empleado *
(F) antCiudadNa : Empleado *
(F) antPaisNa : Empleado *
(F) antCiudadRe : Empleado *
(F) antBarrio : Empleado *
(F) antAct : Empleado *
(F) antEdad : Empleado *
(F) antNumHijos : Empleado *
(F) antNomSucursal : Empleado *
```

Los Métodos de la clase empleado son los siguientes:

```

f ➤ Empleado()
f ➤ Empleado(string, string, string, string, string, s
f ➤ agregarHijo(Hijo) : void
f ➤ getNombre() : string
f ➤ getApellido() : string
f ➤ getTipold() : string
f ➤ getNumld() : long
f ➤ getSexo() : char
f ➤ getCorreo() : string
f ➤ getFechaNacimiento() : string
f ➤ getCiudadNacimiento() : string
f ➤ getPaisNacimiento() : string
f ➤ getCiudadResidencia() : string
f ➤ getDireccion() : string
f ➤ getBarrio() : string
f ➤ getCelular() : int
f ➤ getTelefonoFijo() : int
f ➤ getEdad() : int
f ➤ getActividadLaboral() : string
f ➤ getNombreSucursal() : string
f ➤ setNombre(const string &) : void
f ➤ setApellido(const string &) : void
f ➤ setTipolD(const string &) : void
f ➤ setCorreo(const string &) : void
f ➤ setFechaNacimiento(const string &) : void
f ➤ setCiudadNacimiento(const string &) : void
f ➤ setPaisNacimiento(const string &) : void
f ➤ setCiudadResidencia(const string &) : void
f ➤ setDireccion(const string &) : void
f ➤ setBarrio(const string &) : void
f ➤ setActividadLaboral(const string &) : void
f ➤ setNombreSucursal(const string &) : void
f ➤ setNumerolD(long) : void
f ➤ setSexo(char) : void
f ➤ setTelefonoCelular(int) : void
f ➤ setTelefonoFijo(int) : void
f ➤ getHijos() : Lista<Hijo>

```

## Clase Sucursal

Los métodos de la clase sucursal son los siguientes:

```
f Sucursal()  
f Sucursal(string, string, string, string)  
f getNombre() const : string  
f getDireccion() const : string  
f getBarrio() const : string  
f getNomGerente() const : string
```

Los atributos de esta clase son:

```
F nombre : string  
F direccion : string  
F barrio : string  
F nomGerente : string
```

Las estructuras de datos empleadas son:

```
▼ estructuras  
H cola.h  
H estructura.h  
H Lista.h  
H multilistaEmpleado.h  
H pila.h
```

A continuación, se muestran las especificaciones de cada una de las estructuras de datos, atributos y métodos.

Cola:

```
▼ Cola<T>  
f Cola()  
f InsCola(T) : void  
f AtenderCola() : T  
f ImprimirCola(int) : T  
f ColaVacia() : bool  
F cab : nodo<T> *  
F fin : nodo<T> *  
F tam : int
```

Lista:

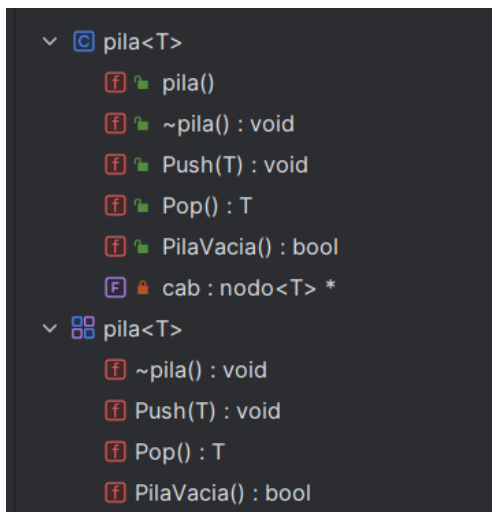
```
f Lista()
f lista_vacia() : bool
f insertar_final(T) : void
f insertar_inicio(T) : void
f insertar_pos(T, int) : void
f eliminar(int) : bool
f editarInfo(int, T) : void
f get_tam() : int
f set_tam(int) : void
f get_info(int) : T
f get_info_ap(int) : T *
f get_nodo_ap(int) : Nodo<T> *
F cab : Nodo<T> *
F fin : Nodo<T> *
F tam : int
```

MultilistaEmpleados:

```
f MultiListaEmpleado()
f encontrarCabecera(string) : bool
f crearCabecera(string) : void
f insertarCabNomSucursal(Empleado *) : void
f insertarCabSexo(Empleado *) : void
f insertarCabCiudadNa(Empleado *) : void
f insertarCabPaisNa(Empleado *) : void
f insertarCabCiudadRe(Empleado *) : void
f insertarCabBarrio(Empleado *) : void
f insertarCabAct(Empleado *) : void
f insertarCabEdad(Empleado *) : void
f eliminarCabSexo(Empleado *) : void
f eliminarCabNomSucursal(Empleado *) : void
f eliminarCabCiudadNa(Empleado *) : void
f eliminarCabCiudadRe(Empleado *) : void
f eliminarCabPaisNa(Empleado *) : void
f eliminarCabBarrio(Empleado *) : void
f eliminarCabAct(Empleado *) : void
f eliminarCabEdad(Empleado *) : void
f eliminarCabNumHijos(Empleado *) : void
f insertarCabNumHijos(Empleado *) : void
f llenarCab() : void
f insertar(Empleado) : void
f eliminar(Nodo<Empleado> *) : void
f actualizarEmp(Empleado) : void
f buscarAnt(long) : Nodo<Empleado> *
f buscarPorID(long) : Empleado *
```

```
f buscarPorID(long) : Empleado *
F multiEmpleados : Lista<Empleado>
F nomCabeceras : Lista<Cab>
```

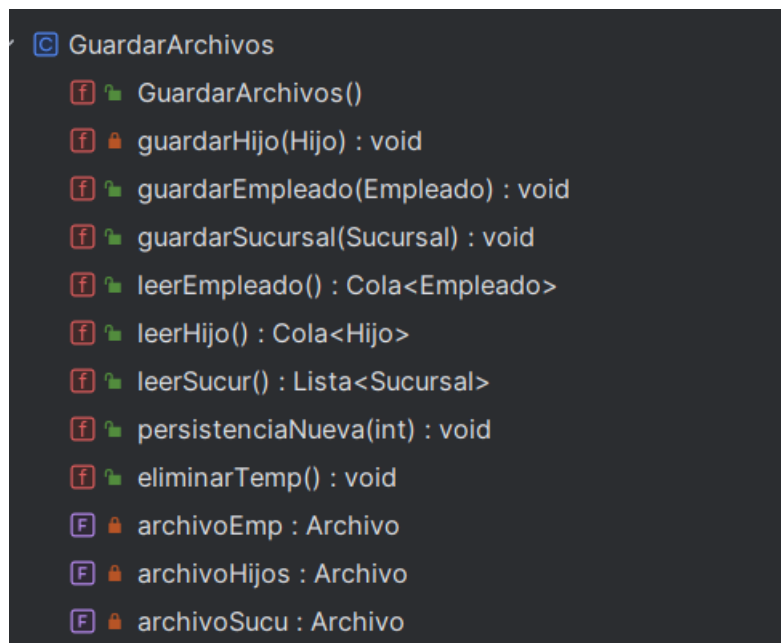
Pila:



Adicionalmente se definieron estructuras que facilitaran la realización de búsquedas:

Para la gestión de archivos, lectura y carga de los mismos se tienen las clases

- GuardarArchivos



- Archivo

```
▼ [C] Archivo
  [f] [g] Archivo()
  [f] [g] anadir(string) : void
  [f] [g] leerLinea() : string
  [f] [g] crearTemporal(string) : void
  [f] [g] borrarTemp(string) : void
  [f] [g] setRuta(string) : void
  [F] [l] nombreArchivo : string
  [F] [l] rutaArchivo : string
```

- GestionPersistencia

```
[f] [g] gestionPersistencia()
[f] [g] asignarHijos() : void
[f] [g] construirMulti() : MultiListaEmpleado
[f] [g] guardarMulti(MultiListaEmpleado) : void
[f] [g] construirSucursales() : Lista<Sucursal>
[f] [g] guardarSucursales(Lista<Sucursal>) : void
[F] [l] hijos : Cola<Hijo>
[F] [l] empleados : Cola<Empleado>
[F] [l] objAr : GuardarArchivos
```

La clase controlador está compuesta de la siguiente manera:

```
▼ [C] Controlador
  [F] [l] mul : MultiListaEmpleado
  [F] [l] gP : gestionPersistencia
  [F] [l] gs : gestionSucursales
  [F] [l] sucursales : Lista<Sucursal>
  [F] [l] vista : Vista
  [F] [l] consultar : Consultas
  [f] [g] ingresarEmpleado() : void
  [f] [g] ingresarSucursal() : void
  [f] [g] mostrarMenuP() : void
  [f] [g] realizarConsultas() : void
  [f] [g] eliminarEmpleado() : void
  [f] [g] editarInformacionEmpleado() : void
  [f] [g] Consulta1() : void
  [f] [g] Consulta2() : void
  [f] [g] Consulta3() : void
  [f] [g] Consulta4() : void
  [f] [g] Consulta5() : void
  [f] [g] Consulta6() : void
  [f] [g] ejecutar() : void
  [f] [g] finalizar() : void
```

La vista se compone de los siguientes métodos:

```
▼ Vista
  mostrarMenu() : int
  mostrarMenuConsultas() : void
  buscarempleado() : int
  mostrarSubMenuInsercion() : Empleado
  mostrarSubMenuInsercionS() : Sucursal
  editarEmp(Empleado *) : Empleado
```

Se implementó la librería “Edad” con los siguientes métodos:

```
Edad()
convertirFecha(string &, int &, int &, int &) : void
calcularEdad(string) : int
esBisiesto(int) : bool
esFechaValida(int, int, int) : bool
# EDAD_H
```

Diagrama de Clases

\*ver documento anexo

Diagrama de casos de uso:

