MANUAL TECNICO

A continuación, se explican los métodos que tiene que ver con la lógica del negocio en este caso relacionado a Activos y los métodos que llevan las estructuras de datos implementadas en el proyecto sean estas un Cubo disperso, Árbol Avl y una lista doble circular.

Métodos para la lógica del programa

```
void iniciar();
void menu_login();
void menu_admin();
void menu_usuario();
//admin----------------------
void crear_usurio();
void eliminar_usuario();
void reporte_de_activos_de_un_usuario();
//usuarios-------------------
void activos_que_yo_rento();
void agregar__Activo();
void eliminar__Activo();
void modificar__Activo();
void rentar__Activo();
void clientes__que_tengo();
void reporte_activos_empresa();
void reporte_activos_departamento();
void reporte_activos_rentados_porUsuario();
```

Explicación:

iniciar(): solo contiene impresione y un submenú para entrar al login o salir de la aplicación.

Login(): le permite al usuario ingresar como usuario o como admin por medio del usuario admin y contraseña admin.

Crear__usuario():este método es exclusivamente para el administrador, permite generar usuarios y luego ingresarlos a un Cubo disperso.

Eliminar_usuario(): este método es exclusivamente para el administrador, extrae al usuario del cubo disperso y lo elimina por medio del id.

Reporte_de_activos_de_un_usuario(): este método es exclusivamente para el administrador, manda a ejecutar un método propio del árbol que genera un grafo por medio de la herramienta graphviz.

Activo_que_yo_rento():este método es de los usuarios muestra los activos que el usuario esta rentanto y le permite devolverlos si desea.

Agregar__Activo():es un método solo para usuarios, permite ingresar un nuevo activo a su árbol avl.

Eliminar__Activo(): es un método solo para usuarios, permite quitar del árbol un activo por medio del id.

Modificar_Activo(): es un método para los usuarios , que le permite modificar el nombre y la descripción de un activo que este adentro de su árbol avl.

Rentar_activos():este método permite para los usuarios, permite rentar activos de otros usuarios por medio de un catálogo.

Clientes_que_tengo(): muestra a los activos que otros usuarios le han rentado al usuario propietario logeado.

Reporte_activos_empresa():este es un método para el administrador el cual ejecuta un metodo que recorre la matriz y revisa en cada usuario para saber si posee o no activos adentro de su árbol y grafica multiples arboles con el filtro de empresa.

Reporte_activos_departamento():este es un método para el administrador el cual ejecuta un metodo que recorre la matriz y revisa en cada usuario para saber si posee o no activos adentro de su árbol y grafica multiples arboles con el filtro de departamento.

Reporte_activos_rentados_porUsuario(): este sale de la lista doble circular es un método para el administrador , llama un archivo el cual es una imagen generada por graphviz con un filtro de el nombre de usuario por nombre, empresa, departamento.

CIRCULAR DOBLE

```
class CircularDobleTransacciones
{
private:
    NodoDobleC* inicio;
    NodoDobleC* ultimo;
    int tamanio;
public:
    CircularDobleTransacciones() { this->inicio = NULL; this->ultimo = NULL; this->tamanio = 0; };

    inline NodoDobleC* getInicio() { return this->inicio; }
    inline NodoDobleC* getUltimo() { return this->ultimo; }
    inline void vaciar() {this->inicio = NULL;this->ultimo = NULL;this->tamanio = 0;}
    void add_transaccion(string id_activo, string nombreU, string emp, string dep, string fecha, string ti);
    void add(Transa * nuevaTr);
    // para verificar si mis enlaces estan bien
    void getGraphviz();
    void imprimeparaAdelnate();
    void imprimeparaAtras();
    // metodos de ordenamiento aun faltan  y metodo para generar el id desde aca
    int compararAlfabeticamente(string, string);// 2 == igual , 0 menor , 1 mayor


    string generarId();
    bool yaExisteEl_id(string id_alfanumerico);
    string getLetra(int);

    void OrdenarAsc();
    void setMenor_como_primero();
    void setMayor_como_ultimo();

    void setMayor_como_inicio();
    void setMenor_como_ultimo();
    void OrdenarDescen();

    void cambiarPosiciones(NodoDobleC* , NodoDobleC*);

    CircularDobleTransacciones* reporteTransaccionesPorUsuario(string usuario , string empresa, string  depar
};
```

El método que se explicara a continuación es el método de inserción:

```
void CircularDobleTransacciones::add_transaccion(string id_activo, string nombreUsuario, string emp, string dep, string fecha, string ti) {
    string  id_transaccion_nueva= this->generarId();
    while (yaExisteEl_id(id_transaccion_nueva)) {
        id_transaccion_nueva = this->generarId();
    }
    cout << "ID de Transaccion GENERADO: " + id_transaccion_nueva << endl;
    this->add(new Transa(id_transaccion_nueva ,id_activo , nombreUsuario, emp, dep, fecha , ti));
}

void CircularDobleTransacciones::add(Transa * nuevaTr ) {
    NodoDobleC* nuevo = new NodoDobleC();
    nuevo->tra = nuevaTr;
    if (this->inicio == NULL) {
        this->inicio = nuevo;
        this->ultimo = nuevo;
        this->inicio->sig = nuevo;
        this->ultimo->sig = nuevo;
        this->tamanio++;
    }
    else { // insercion al final
        nuevo->ant = this->ultimo;
        this->ultimo->sig = nuevo;
        this->ultimo = nuevo; // refrescamos cual es el ultimo ahora

        this->ultimo->sig = this->inicio;
        this->inicio->ant = this->ultimo;
        this->tamanio++;
    }
}
```

En una lista circular doble se poseen 2 punteros un ant que es mi anterior un sig que es mi siguiente , para insertar llamo previamente a mi método add_transaccion para poder generar un id único y luego ya lo envio al insertar de la lista doble circular ahí reservo memoria para un nuevo nodo seguido pregunto si la lista esta vacia pues este será el ultimo y a su vez el primero , sino la inserto al final en ese paso lo que se hace es que el puntero anterior del nuevo nodo que se creo va estar apuntando al ultimo seguido de eso el siguiente del ultimo va ser mi nuevo nodo , y ahora actualizo quien es el ultimo , el ultimo será mi nuevo nodo que acabo de ingresar ahora para terminar el algoritmo solo hacemos que el nuevo nodo con su puntero siguiente apunte al inico de mi lista y que el puntero anterior del inicio de mi lista apunte al ultimo nodo de mi lista (el cual es el nuevo) y se tiene una variable de tamaño que crece en cualquiera de los dos casos en 1.

METODO PARA ODENAR LA LISTA CIRCULAR DOBLE

```
void CircularDobleTransacciones::OrdenarDescen() {
    if (this->tamanio == 0 ) {
        return;
    }
    if (this->tamanio == 2) {
        this->setMayor_como_inicio();
        this->setMenor_como_ultimo();
        return;
    }
    else {
        this->setMayor_como_inicio();
        this->setMenor_como_ultimo();

        NodoDobleC* puntero1 = this->inicio; // como mi inicio siempre es el menor es el unico nodo que no podria intercambiarse , ya que lo c
        NodoDobleC* puntero2 = NULL;
        puntero2 = puntero1->sig;

        for (int i = 0; i < this->tamanio; i++) {


            for (int j = 0; j < this->tamanio - 1; j++) {

                if (compararAlfabeticamente(puntero2->tra->getId_tra(), puntero2->sig->tra->getId_tra()) == 0 && puntero2->sig != this->inicio)
                    /*cout << "entra a ordenar ASc: " << puntero2->tra->getId_tra() << "<" << puntero2->sig->tra->getId_tra() << endl;
                    this->imprimeparaAdelnate();
                    system("pause");*/

                    this->cambiarPosiciones(puntero2, puntero2->sig);

                }
                puntero2 = puntero2->sig;
            }

            puntero1 = puntero1->sig;
        }
    }
}
```

El método esta compuesto por otros 3 métodos que le ayudan setMayor_como_inicio() , setMenor_como_ultimo() y cambiarPosiciones().

MATRIZ

```
using namespace std;
class Matriz_dispersa
{
private:
    nMatrix* root;
    int contadorFilas = -1;
    int contadorColumnas = -1;
public:
    Matriz_dispersa() {
        this->root = new nMatrix("-1" , "-1");
        this->root->setDepartamento("-1");
        this->root->setEmpresa("-1");
        this->root->setPos_x(-1);
        this->root->setPos_y(-1);
    }
    inline nMatrix* getRoot() {
        return this->root;
    }
    //------------------------                    SOLO PARA VER SI EXISTE
    nMatrix* existeColumna(string);
    nMatrix* existeFIla(string);
    //------------------------

    //------------------------                   GENERADOR DE COLUMNAS Y FILAS
    nMatrix* generaColumna(string);
    nMatrix* generaFila(string);


    //---- update positions
    nMatrix* actualizaPosiciones(nMatrix* , nMatrix*, nMatrix*);
    nMatrix* heredarPunteros(nMatrix* , nMatrix*);



    // ------- buscar en el CUBO :'v
    void modificarUsuario(string dep_fila, string  emp_col, Usuario* usuario);// enviando
    nMatrix* BuscarNodo(string empresa, string dep, string nombreUser);// recibiendo
    nMatrix* isInsercion3D(string empresa, string dep);
    bool UsuarioRepetido(nMatrix* cordenada ,string nombre);
    void eliminarInterno(string empresa, string dep, string nombreUser);
    nMatrix* reemplazarCara(nMatrix* adelante, nMatrix* atras);
```

La mayoría de métodos realizan algo relacionado con su nombre pero a continuación se explicaran brevemente los que se consideran más importantes.

```
    nMatrix* generaFila(string);


    //---- update positions
    nMatrix* actualizaPosiciones(nMatrix* , nMatrix*, nMatrix*);
    nMatrix* heredarPunteros(nMatrix* , nMatrix*);



    // ------- buscar en el CUBO :'v
    void modificarUsuario(string dep_fila, string  emp_col, Usuario* usuario);// enviando
    nMatrix* BuscarNodo(string empresa, string dep, string nombreUser);// recibiendo
    nMatrix* isInsercion3D(string empresa, string dep);
    bool UsuarioRepetido(nMatrix* cordenada ,string nombre);
    void eliminarInterno(string empresa, string dep, string nombreUser);
    nMatrix* reemplazarCara(nMatrix* adelante, nMatrix* atras);


    // aux visuales nada mas
    void imprimirFondo(nMatrix*);
    void imprimirSolo3D();

    // grafo
    void getGraphviz();


    // agregar
    void add(string empresa_fila, string  dep, Usuario* usuario);


    // reportes
    void reporteEmpresa(string empresa_fila);
    void reporteDepartamento(string departamento_colum);

    // para mostrar el catalogo

    CatalogoLsimple* recolectaProductos(CatalogoLsimple* CATALOGO);

};
```

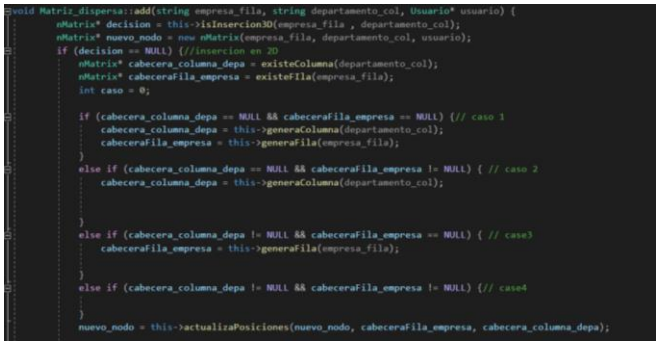
generaFila() y generaColumna() estos métodos reciben como parámetro un atributo string con el nombre del nuevo cabezal que se formará en la matriz.

```
// GENERANDO LAS FILAS Y COLUMNAS NUEVAS
nMatrix* Matriz_dispersa::generaFila(string empresaNueva) {
    this->contadorFilas++;
    nMatrix* NUEVA_FILA = new nMatrix(empresaNueva, "-1");
    nMatrix* aux = this->root;
    while (aux->getDown() != NULL) {
        aux = aux->getDown();
    }
    NUEVA_FILA->setPos_x(-1);
    NUEVA_FILA->setPos_y(this->contadorFilas);
    NUEVA_FILA->setArriba(aux);
    aux->setAbajo(NUEVA_FILA);
    return NUEVA_FILA;
}


nMatrix* Matriz_dispersa::generaColumna(string depNuevo) {
    this->contadorColumnas++;
    nMatrix* NUEVA_COLUMNA = new nMatrix("-1", depNuevo);
    nMatrix* aux = this->root;
    while (aux->getDer() != NULL) {
        aux = aux->getDer();
    }
    NUEVA_COLUMNA->setPos_x(this->contadorColumnas);
    NUEVA_COLUMNA->setPos_y(-1);
    NUEVA_COLUMNA->setIzq(aux);
    aux->setDerecha(NUEVA_COLUMNA);
    return NUEVA_COLUMNA;
}
```

ExisteFila() ExisteColumna() estos dos métodos verificaban si ya existían adentro de la matriz esos cabezales pues retornaban el cabezal si existían y si no retornaban un NULL

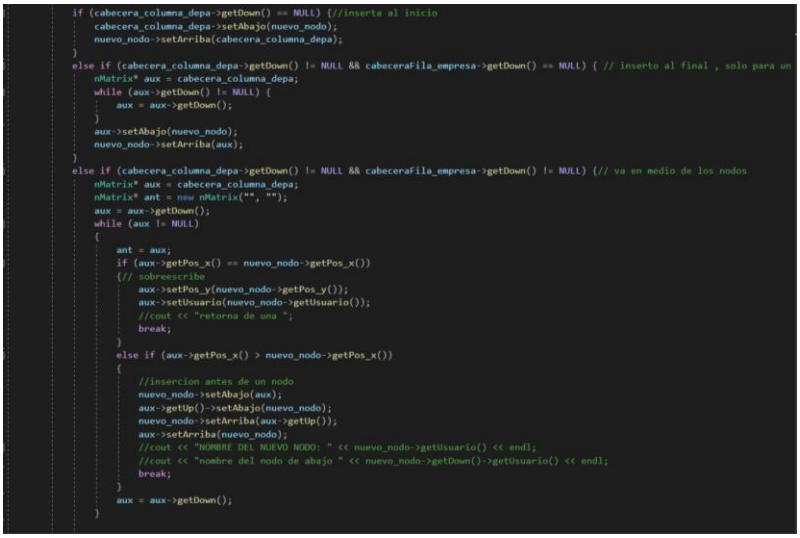
```
nMatrix* Matriz_dispersa::generaFila(string empresaNueva) {
    this->contadorFilas++;
    nMatrix* NUEVA_FILA = new nMatrix(empresaNueva, "-1");
    nMatrix* aux = this->root;
    while (aux->getDown() != NULL) {
        aux = aux->getDown();
    }
    NUEVA_FILA->setPos_x(-1);
    NUEVA_FILA->setPos_y(this->contadorFilas);
    NUEVA_FILA->setArriba(aux);
    aux->setAbajo(NUEVA_FILA);
    return NUEVA_FILA;
}


nMatrix* Matriz_dispersa::generaColumna(string depNuevo) {
    this->contadorColumnas++;
    nMatrix* NUEVA_COLUMNA = new nMatrix("-1", depNuevo);
    nMatrix* aux = this->root;
    while (aux->getDer() != NULL) {
        aux = aux->getDer();
    }
    NUEVA_COLUMNA->setPos_x(this->contadorColumnas);
    NUEVA_COLUMNA->setPos_y(-1);
    NUEVA_COLUMNA->setIzq(aux);
    aux->setDerecha(NUEVA_COLUMNA);
    return NUEVA_COLUMNA;
}
```

El metodo add() este era el encargado de ingresar un nuevo usuario a la matriz , para este método tome en cuenta 5 posibles casos de inserción fila existe columna existe , fila no existe columna si , columna no existe fila si , ninguno de los dos cabezales existe y el de inserción en 3D



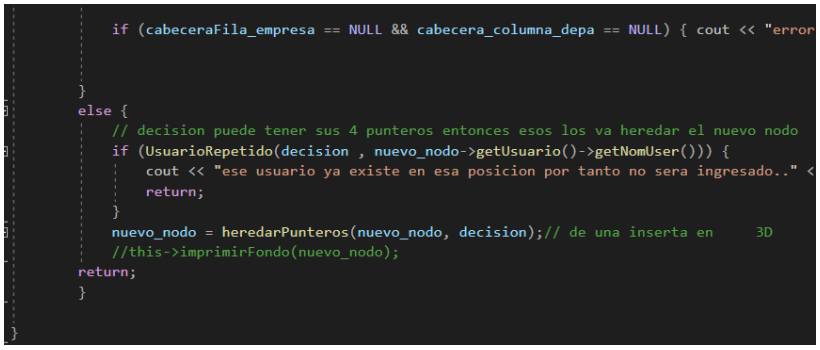
```
void Matriz_dispersa::add(string empresa_fila, string departamento_col, Usuario* usuario) {
        nMatrix* decision = this->isInsercion3D(empresa_fila , departamento_col);
        nMatrix* nuevo_nodo = new nMatrix(empresa_fila, departamento_col, usuario);
        if (decision == NULL) {//insercion en 2D
            nMatrix* cabecera_columna_depa = existeColumna(departamento_col);
            nMatrix* cabeceraFila_empresa = existeFila(empresa_fila);
            int caso = 0;

            if (cabecera_columna_depa == NULL && cabeceraFila_empresa == NULL) {// caso 1
                cabecera_columna_depa = this->generaColumna(departamento_col);
                cabeceraFila_empresa = this->generaFila(empresa_fila);
            }
            else if (cabecera_columna_depa == NULL && cabeceraFila_empresa != NULL) { // caso 2
                cabecera_columna_depa = this->generaColumna(departamento_col);


            }
            else if (cabecera_columna_depa != NULL && cabeceraFila_empresa == NULL) { // case3
                cabeceraFila_empresa = this->generaFila(empresa_fila);

            }
            else if (cabecera_columna_depa != NULL && cabeceraFila_empresa != NULL) {// case4

            }
            nuevo_nodo = this->actualizaPosiciones(nuevo_nodo, cabeceraFila_empresa, cabecera_columna_depa);
```

Inserción en 2D es una inserción en dos listas dobles pero en simultaneo o de esa manera es como lo veo. En este caso se muestra la inserción desde el punto de vista de la lista doble de la columna , para la fila es lo mismo pero cabe destacar que en una podría ser una inserción al inicio y en la otra al final , que sea simultaneo no significa que se inserte se la misma forma para ambas listas dobles.



```
if (cabecera_columna_depa->getDown() == NULL) {//inserta al inicio
    cabecera_columna_depa->setAbajo(nuevo_nodo);
    nuevo_nodo->setArriba(cabecera_columna_depa);
}
else if (cabecera_columna_depa->getDown() != NULL && cabeceraFila_empresa->getDown() == NULL) { // inserto al final , solo para un
    nMatrix* aux = cabecera_columna_depa;
    while (aux->getDown() != NULL) {
        aux = aux->getDown();
    }
    aux->setAbajo(nuevo_nodo);
    nuevo_nodo->setArriba(aux);
}
else if (cabecera_columna_depa->getDown() != NULL && cabeceraFila_empresa->getDown() != NULL) {// va en medio de los nodos
    nMatrix* aux = cabecera_columna_depa;
    nMatrix* ant = new nMatrix("", "");
    aux = aux->getDown();
    while (aux != NULL)
    {
        ant = aux;
        if (aux->getPos_x() == nuevo_nodo->getPos_x())
        {// sobreescribe
            aux->setPos_y(nuevo_nodo->getPos_y());
            aux->setUsuario(nuevo_nodo->getUsuario());
            //cout << "retorna de una ";
            break;
        }
        else if (aux->getPos_x() > nuevo_nodo->getPos_x())
        {
            //insercion antes de un nodo
            nuevo_nodo->setAbajo(aux);
            aux->getUp()->setAbajo(nuevo_nodo);
            nuevo_nodo->setArriba(aux->getUp());
            aux->setArriba(nuevo_nodo);
            //cout << "NOMBRE DEL NUEVO NODO: " << nuevo_nodo->getUsuario() << endl;
            //cout << "nombre del nodo de abajo " << nuevo_nodo->getDown()->getUsuario() << endl;
            break;
        }
        aux = aux->getDown();
    }
```

Ahora la inserción en 3D , se hace una búsqueda previa del punto [empresa – departamento] y si es se encuentra se procede a verificar si el nuevo usaurio ya existe si no existe se procede a hacer la inserción de la siguente manera apoyándose de 1 método extra heredarPunteros()



```
        if (cabeceraFila_empresa == NULL && cabecera_columna_depa == NULL) { cout << "error


    }
    else {
        // decision puede tener sus 4 punteros entonces esos los va heredar el nuevo nodo
        if (UsuarioRepetido(decision , nuevo_nodo->getUsuario()->getNomUser())) {
            cout << "ese usuario ya existe en esa posicion por tanto no sera ingresado.." <
            return;
        }
        nuevo_nodo = heredarPunteros(nuevo_nodo, decision);// de una inserta en     3D
        //this->imprimirFondo(nuevo_nodo);
    return;
    }

}
```

El metodo heredarPunteros() da todos sus punteros a la antigua cara y se los hereda a la nueva cara por tanto se procede a ingresar atrás la antigua cara para no perder los datos.

```
nMatrix* Matriz_dispersa::heredarPunteros(nMatrix* nuevo_nodo , nMatrix* decision) {
    // puede que sea una esquina  , o de los ultimos solo daria clavo derecha y abajo
    decision->getUp()->setAbajo(nuevo_nodo);
    decision->getIzq()->setDerecha(nuevo_nodo);
    if (decision->getDer() != NULL) {
        decision->getDer()->setIzq(nuevo_nodo);
    }
    if (decision->getDown() != NULL) {
        decision->getDown()->setArriba(nuevo_nodo);
    }


    // 0 clavos
    nuevo_nodo->setAbajo(decision->getDown());
    nuevo_nodo->setArriba(decision->getUp());
    nuevo_nodo->setIzq(decision->getIzq());
    nuevo_nodo->setDerecha(decision->getDer());
    // actualiza posiciones
    nuevo_nodo->setPos_x(decision->getPos_x());
    nuevo_nodo->setPos_y(decision->getPos_y());
    // limpieza de mi nodo que antes era la CARA
    decision->setAbajo(NULL);
    decision->setArriba(NULL);
    decision->setDerecha(NULL);
    decision->setIzq(NULL);

    // insercion :O
    nuevo_nodo->setBehind(decision);
    decision->setFront(nuevo_nodo);

    return nuevo_nodo;
}
```

ARBOL AVL METODOS

```
private:
    string Graph;
    int indice;
    Navl* raiz;
//  bool yaId;
    void mostrar_activos(Navl*);
    CatalogoLsimple* recolectaProRecursivo(Navl* ,CatalogoLsimple* CATALOGO , string nombreUser, string empresa, string departamento);
public:

    ArbolBin() {
            this->raiz = NULL;
            this->Graph = "";
            this->indice = 0;
            //this->yaId = false;
    }
    inline Navl* getRoot() { return this->raiz; };
    void add(Navl*);// le reservo memoria de un nodo antes de meterlo al arbol
    void add_Activo(string nom_, string descripcion_, bool disp);
    Navl* recursive_add(Navl*, Navl*);// root temporal , nuevo
    int compararAlfabeticamente(string, string);// 2 == igual , 0 menor , 1 mayor

    // para ver el resultado
    void recorrido_inOrder();
    void recursive_inOrder(Navl*);

    void eliminar(string);
    Navl* eliminarRecursivo(Navl*, string);
    Navl* getNodoMinimo(Navl*);


    //++++++++++++++++++++++++ GRAFICAR MI ARBOL ALV
    void getGraphviz();
    void getGraphviz(Navl*); // recolecto en preorder       raiz izq derecha
    void getGraphvizRepUsuarios(string nombreUsuario);

    //++++++++++++++++++++++++ BUSCAR
    Navl* buscar(string);
    Navl* buscarRecursivo(Navl*, string);

    // para le id alfa
    string generarId();
```

```
    void recorrido_inOrder();
    void recursive_inOrder(Navl*);

    void eliminar(string);
    Navl* eliminarRecursivo(Navl*, string);
    Navl* getNodoMinimo(Navl*);


    //++++++++++++++++++++++++ GRAFICAR MI ARBOL ALV
    void getGraphviz();
    void getGraphviz(Navl*); // recolecto en preorder       raiz izq derecha
    void getGraphvizRepUsuarios(string nombreUsuario);

    //++++++++++++++++++++++++ BUSCAR
    Navl* buscar(string);
    Navl* buscarRecursivo(Navl*, string);

    // para le id alfa
    string generarId();
    bool yaExisteEl_id(string id_alfanumerico);
    string getLetra(int);


    // para calcular el balance

    int GET_MI_FE(Navl* padre);
    int GET_Al(Navl*); // por si mi nodo izq , der cualquiera es null
    void dame_la_altura_y_balance(Navl*);


    // rotaciones
    Navl* rot_s_derecha(Navl*);
    Navl* rot_s_izquierda(Navl*);


    void mostrar_activos();

    // recolecar productos
    CatalogoSimple* recolectaPro(CatalogoSimple* CATALOGO , string nombreUser , string empresa , string departamento);

};
```

METODO ADD_ACTIVO() EN EL ARBOL AVL : por el momento lo que se muestra a continuacion solo es para un arbol binario normal no tienen nada de extraordinario solo que antes de ingresarse se le genera un ID unico y se va guardando un registro de los activos que ya se crearon con anterioridad para guardar la integridad de los activos y asi que no aparezcan repetidos igualmente quitando ambigüedad a la hora de buscar o eliminar datos.

```
void ArbolBin:: add_Activo(string nom_, string descripcion_, bool disp) {
    string  id_act = this->generarId();
    while (yaExisteEl_id(id_act) || Estatica::CONTROL_IDS->yaExisteEl_id(id_act)) {
        id_act = this->generarId();;
    }
    cout << "ID GENERADO: " + id_act << endl;
    Estatica::CONTROL_IDS->add(new Navl(new Act(id_act, nom_, descripcion_, disp)));
   // Estatica::CATALOGO->getGraphviz();
    this->add(new Navl(new Act(id_act , nom_ , descripcion_ , disp)));
}

void ArbolBin::add(Navl* nuevo) { // todo los nombre que agregue deben de ir en minusculas :o
    if (this->raiz == NULL) {
        this->raiz = nuevo;
    }
    else {
       this->raiz = recursive_add(this->raiz, nuevo);
    }
}


Navl* ArbolBin::recursive_add(Navl* actual, Navl* nuevo) {
    if ((compararAlfabeticamente(actual->acti->id_activ, nuevo->acti->id_activ)) == 1) {
        if (actual->iz != NULL) {
            actual->iz = recursive_add(actual->iz, nuevo);
        }
        else {
            actual->iz = nuevo;
        }
    }
    else if ((compararAlfabeticamente(actual->acti->id_activ, nuevo->acti->id_activ)) == 0) {

        if (actual->de != NULL) {
            actual->de = recursive_add(actual->de, nuevo);
        }
        else {
            actual->de = nuevo;
        }
    }
    else {
        cout << "xxxxxxxxxxxxxxxxxxxxxxxxxxxxx" << endl;
        cout << "REPETIDO: " + nuevo->acti->id_activ << endl; // no lo inserta
```

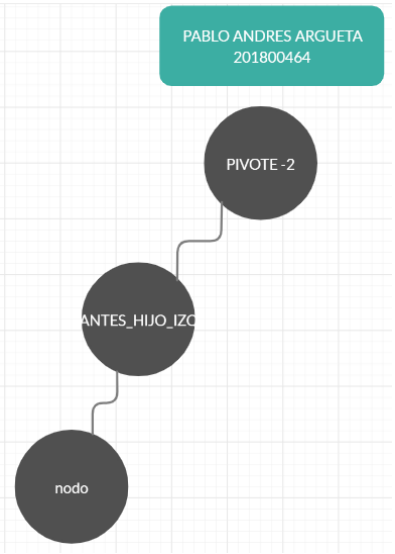
A continuación se muestran las rotaciones lo que lo hace un AVL:

```
actual->set_Mayor_Altura(GET_Al(actual->de), GET_Al(actual->iz));



int fact = GET_MI_FE(actual);
// this->dame_la_altura_y_balance(actual);// solo imprime mis datos de informacion
if (fact == -2 && GET_MI_FE(actual->iz) == -1) {// si es -2 fijo tengo hijo izquierdo entonces pregunto
  // cout << "rotacion simple a la derecha :)" << endl;
   actual = rot_s_derecha(actual);
}
else if (fact == 2 && GET_MI_FE(actual->de) == 1) {
   //cout << "rotacion simple a la izquierda :)" << endl;
   actual = rot_s_izquierda(actual);
}
else if (fact == 2 && GET_MI_FE(actual->de) == -1) {
  // cout << "rotacion doble a la derecha , izquierda :)" << endl;
   /*      X fe == 2
                 X fe == -1
             X
   */
   actual->de = rot_s_derecha(actual->de); // es como mas externa por eso se manda el derecho
   actual = rot_s_izquierda(actual);// esta es mas directa porque el arbol ya esta así , entonces lo que retorna ya es la nueva raiz
   /*
         X fe == 2
            X fe == 1
               X
   */
}
else if (fact == -2 && GET_MI_FE(actual->iz) == 1) {
  // cout << "rotacion doble izquierda , derecha :)" << endl;
   /*     9 fe == -2
   2 fe == +1
        5 fe == 0
   */
   actual->iz = rot_s_izquierda(actual->iz); // es como mas externa por eso se manda el IZquierdo
   actual = rot_s_derecha(actual);//mas directa , entonces lo que retorna ya es la nueva raiz

   /*      X
      X
   X
   */
```

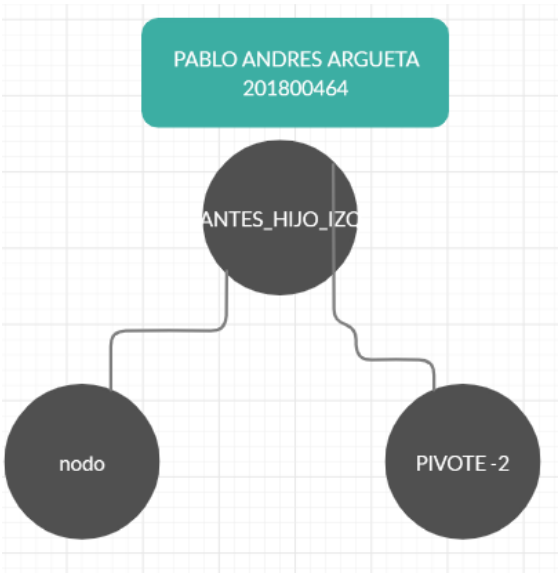
Rotacion simple Derecha

```
void ArbolBin::dame_la_altura_y_balance(Navl* nodoPrueba) {
    if (nodoPrueba != NULL)     cout << "ID: " << nodoPrueba->acti->id_activ << endl;
    else cout << "nodo NULL" << endl;


    cout << "altura :" << GET_Al(nodoPrueba) << endl;
    cout << " FE: " << GET_MI_FE(nodoPrueba) << endl;
}



Navl* ArbolBin:: rot_s_derecha(Navl* pivote_menos_2) { // OJO SIEMPRE VA TENER HIJO IZQUIERDO
    Navl* antes_hijo_izquierdo = pivote_menos_2->iz;
    pivote_menos_2->iz = antes_hijo_izquierdo->de; // lo derecho de mi hijo anterior izz
    antes_hijo_izquierdo->de = pivote_menos_2; // el pivote menos 2 deja de SER LA RAIZ Y AHORA LO ES EL HIJO IZQUIEDO


    // actualiza las  alturas de mi pivote y del que antes era su hijo izquierdo
    pivote_menos_2->set_Mayor_Altura(  GET_Al(pivote_menos_2->de), GET_Al(pivote_menos_2->iz)  );// indiferente el orden en que mande las altu
    antes_hijo_izquierdo->set_Mayor_Altura(GET_Al(antes_hijo_izquierdo->de), GET_Al(antes_hijo_izquierdo->iz));
    /*
    dame_la_altura_y_balance(pivote_menos_2);
    dame_la_altura_y_balance(antes_hijo_izquierdo);
    */

    return antes_hijo_izquierdo; // retorno la nueva root
}
```

Se explicará el código con un poco de ayuda grafica.



Para que se realice este método se tiene que cumplir que un nodo tenga un factor de equilibrio de -2 , tener en cuenta que si el “antes hijo izq” tuviera un hijo derecho este siempre sería menor al “pivote-2” por tanto ese como acarreo le queda a él después de hacer la rotación. A continuación, se muestra como quedaría la rotación.



La rotación a la izquierda es lo mismo pero para el otro lado y las otras dos retaciones son combinaciones de estas dos.

ELIMINAR EN EL AVL

```
Nav1* ArbolBin::eliminarRecursivo(Nav1* raiz, string id) {
    if (raiz == NULL) { // la llave para salir de la recursividad es que mi raiz sea NULL
        return raiz;
    }

    int res = this->compararAlfabeticamente(raiz->acti->id_activ, id);

    switch (res) {
    case 1:// 1 lo uso para saber si id1 es mayor a id2
        raiz->iz = this->eliminarRecursivo(raiz->iz, id);
        break;
    case 0: // 0 para saber si id1 es menor a id2 (segundo parametro)
        raiz->de = this->eliminarRecursivo(raiz->de, id);
        break;
    default:
        // AQUI ES YA ASEGURO HABER ENCONTRADO EL NODO CON EL MISMO ID
        if ((raiz->iz == NULL) || (raiz->de == NULL)) { // si uno de los dos de sus punteros es null entonces se puede reemplazar
            Nav1* revisar = NULL;

            if (revisar == raiz->iz) revisar = raiz->de;
            else  revisar = raiz->iz;

            if (revisar == NULL) raiz = NULL;// Hoja , solo la borrar directo
            else raiz = revisar;//reemplaza

           //  cout << "Activo eliminado :D" << endl;// si llega aca se que se va eliminar
        }
        else {
            Nav1* revisar = this->getNodoMinimo(raiz->de); // menor de lOs mayores
            raiz->acti->setTodoEl_objeto(revisar->acti->id_activ, revisar->acti->nombre, revisar->acti->descripcion , revisar->acti->dispo);//
            raiz->de = eliminarRecursivo(raiz->de, revisar->acti->id_activ);// le mando la parte derecha porque ahora en mi arbol hay simultane
        }
        break;
    }


    if (raiz == NULL) return raiz;// cuando se elimina la hoja puede dar null pointer si baja por eso se retorna de una vez

    int fact = GET_MI_FE(raiz);

    if (fact == -2 && GET_MI_FE(raiz->iz) == -1) {// si es -2 fijo tengo hijo izquierdo entonces pregunto
```

Para la eliminación de un avl es la elimiacion de un árbol binario pero con condiciones de rotación , como se puede ver el se pregunta y cuando el resultado da 2 en mi metodo compararalfabeticamente lo que sucede es que entra al caso de eliminación el case default, en ese caso si es hoja pueden pasar 2 cosas que se reemplace o que se elimine definitivamente.
Pregunta si la izquierda del que quiero eliminar es nulo , un nodo aux se vuelve la derecha de ese nodo esa el “revisar”será diferente de NULL sino es el izquierdo el null en mi nodo auxiliar revisar se vuelve la parte izquierda del nodo que quiero eliminar luego pregunta:
si revisar contiene algo solo lo cambio por ya sea su hijo izquierdo o derecho ahora si los dos fueron NULL significa que es una hoja y pues solo elimino directo la raíz , ahora en el caso de que el nodo que quiera eliminar tenga dos hijos lo que hace es un intercambio de nodos sube el menor de los mayores y de el extraigo todos los datos y los pongo en el que quería borrar ósea que los datos del que quería borrar ya no existen pero en ese punto me todo que tengo dos nodos con le mismo id pero el menor de los mayores es una hoja por tanto lo mando a eliminar.

CASO EN EL QUE EL NODO QUE QUIERO ELIMINAR TIENE DOS HIJOS

Eliminando el 5: