

## Criterio C - Desarrollo

Clase Menu (): Esta clase es la MAIN, la misma es la única ejecutable, en la que se agrupan todos los componentes del sistema mediante el menú del mismo.

Además, como se observa en el código, se llama a los métodos ubicados en la clase Métodos Generales (): , quienes se encargan de verificar la existencia, leer y guardar los archivos externos del sistema y los que luego serán utilizados para serialización. En cuanto a los archivos externos se ha decidido que los mismos sean aquellos que agrupen la información de los Insumos y los Costos Extra ya que esta información es brindada previamente por mi cliente. Estos permitirán luego conformar parte del Producto, para que este pueda ser utilizado durante la ejecución.

Por otro lado, se ha decidido que aquellos objetos relacionados con historiales, como las ventas, compras, clientes, vendedores y proveedores sean serializados ya que el acceso a estos será únicamente mediante el sistema debido que el mismo cuenta con las funciones necesarias, que se detallarán más adelante, para que así lo sea.

```
public static void main(String[] args) throws IOException, ClassNotFoundException
{
    //LEEMOS ARCHIVOS
    MetodosGenerales.CheqLeerDatos();
    MetodosGenerales.leerDatos();
    //TOMAMOS FECHA DEL SISTEMA
    MetodosGenerales.tomarFecha ();
    //CORROBORA STOCK INSUMOS
    gesInsumos.alertStock();
}
```

Para uso de las distintas categorías del programa se han implementado Menús a través de Do While con el fin estos se ejecuten automáticamente desde el comienzo.

```
char opcMod;
do
{
    System.out.println("\n\n");
    System.out.println("\t\t\t +-----+");
    System.out.println("\t\t\t |                EMPRESA MJF                |");
    System.out.println("\t\t\t +-----+");
    System.out.println("\t\t\t |                MENÚ PRINCIPAL                |");
    System.out.println("\t\t\t |                -----                |");
    System.out.println("\t\t\t |                *seleccione una opcion*        |");
    System.out.println("\t\t\t |                -----                |");
    System.out.println("\t\t\t |    1- Proveedores                            |");
    System.out.println("\t\t\t |    2- Insumos                                |");
    System.out.println("\t\t\t |    3- Producto                               |");
    System.out.println("\t\t\t |    4- Vendedores                             |");
    System.out.println("\t\t\t |    5- Clientes                               |");
    System.out.println("\t\t\t |    6- Verificacion de stock                   |");
    System.out.println("\t\t\t |    7- Pedidos                                 |");
    System.out.println("\t\t\t |    8- Ventas                                  |");
    System.out.println("\t\t\t |                -----                |");
    System.out.println("\t\t\t +-----+");
    System.out.println("\t\t\t |    *- Guardar                                |");
    System.out.println("\t\t\t |    0- Salir del sistema                       |");
    System.out.println("\t\t\t +-----+");

    opcMod = IBIO.inputChar("Ingrese la opcion: ");

    switch (opcMod)
    {
        case '1' : gesProveedores.menuProveedor();
                    break;

        case '2' : gesInsumos.menuInsumos() ;
                    break;

        case '3' : gesProducto.menuProducto();
```

```

        case '3' : gesProducto.menuProducto();
        break;

        case '4' : gesVendedores.menuVendedor();
        break;

        case '5' : gesClientes.menuCliente();
        break;

        case '6' : gesInsumos.veriStock();
        break;

        case '7' : gesPedidos.menuVentas();
        break;

        case '8' : gesVentas.menuVentas();
        break;

        case '*' : MetodosGenerales.guardarDatos();
        break;

    }
}while (opcMod != '0');

```



Cada opción cuenta con su propio Menú

Método de Carga aplicado para los objetos; Proveedor, Cliente, Vendedor, Venta, Compra e Insumo Proveedor.

```

public void cargarProveedor ()
{
    System.out.println("\n|ALTA PROVEEDOR|\n ");
    String cuilCuit = IBIO.input("\n| Cuil/Cuit: ");
    veriE(cuilCuit);
    if (veriE(cuilCuit))
    {
        System.out.println("\n|El proveedor ya existe\n");
    }
    else
    {
        String razonSoc = IBIO.input("\n| Razon Social: ");
        String dni = IBIO.input("\n| DNI: ");
        String tel = IBIO.input("\n| Telefono/Celular: ");
        String mail = IBIO.input("\n| Mail: ");
        String direccion = IBIO.input("\n| Direccion: ");
        char estado = IBIO.inputChar("\n| Estado del proveedo [A por activo, D por desactivo]");
        InsumoProv insumo = cargarInsP(estado);

        Proveedor Nuevo = new Proveedor (razonSoc, dni, cuilCuit, tel, mail, direccion, insumo, estado);
        proveedores.add(Nuevo);
    }
}

```

En primer lugar se pide un número de cuil/cuit, quien actuará como identificador único de las personas, luego se somete a una prueba de existencia utilizando un ciclo For que recorre la lista buscando posibles igualdades. Para acceder a los atributos privados de la clase se utiliza el método GET que se encuentra dentro de ella. Se hace uso de una booleana para informar la situación.

```
public boolean veriE (String cuilCuit) //Verificacion de existencia de Proveedor
{
    for (int i = 0; i < proveedores.size(); i++)
    {
        if (proveedores.get(i).getCuilCuit().equalsIgnoreCase(cuilCuit))
        {
            return true;
        }
    }
    return false;
}
```

A partir de ello se procede a consultar por los datos necesarios para llevar a cabo la creación del nuevo objeto. Se realiza la creación del nuevo objeto con todos sus atributos previamente definidos en el constructor de su respectiva clase, luego se prosigue con la agregación del objeto a la lista mediante la función Add.

Luego de ello se prosigue con un ordenamiento por inserción, el cual resulta el más natural, ya que al generarse un nuevo cliente se recorre su lista comparándolo con el atributo considerado para el ordenamiento de cada uno de los objetos dentro de ella, si se halla que el mismo es mayor (en este caso por medio del código ASCII) se inserta allí mediante un add.

```

static void ordenamientoIns (Vendedor nuevo)
{
    //Ciclo de control principal, asume el primer elemento ubicado, inicia en el segundo
    for (int i = 1; i < vendedores.size(); i++)
    {
        if (vendedores.get(i).getRazonSoc().compareToIgnoreCase(nuevo.getRazonSoc()) > 0)
        {
            vendedores.add(i-1,nuevo);
            break;
        }
    }
}

```

Nos permite determinar si, según el valor del código ASCII, la palabra es mayor o menor

Método Modificar aplicado para los objetos; Proveedor, Cliente, Vendedor, Venta, Compra, Insumo Proveedor, Insumos y Costo Extra.

```

public void modificarCliente ()
{
    String ing = IBIO.input("\n| Cuil/cuit del cliente a modificar: ");
    for (int i = 0; i < clientes.size(); i++)
    {
        if (clientes.get(i).cuilCuit.equalsIgnoreCase(ing))
        {
            clientes.get(i).modCli();
        }
    }
}

```

Para modificar los objetos se pide por el identificador, luego se realiza una búsqueda mediante el uso de un FORO mediante el .size que nos permite obtener el tamaño de la lista. Al identificar el objeto se llama a un método de modificación dentro de la clase identidad.

```

switch (opcMod)
{
    case '1' : this.setRazonSoc(IBIO.input("\n| Ingrese la nueva Razon Social: "));
               break;

    case '2' : this.setDni(IBIO.input("\n| Ingrese el nuevo DNI: "));
               break;

    case '3' : this.setCuilCuit(IBIO.input("\n| Ingrese el nuevo Cuil/Cuit: "));
               break;

    case '4' : this.setTel(IBIO.input("\n| Ingrese el nuevo Telefono: "));
               break;

    case '5' : this.setMail(IBIO.input("\n| Ingrese el nuevo Mail: "));
               break;

    case '6' : this.setDireccion(IBIO.input("\n| Ingrese la nueva Direccion: "));
               break;

    case '7' : this.setCat(IBIO.inputChar("\n| Ingrese la nueva categoria(D en distri
               break;

}
}while (opcMod != '0');

```

Nuevamente se hace uso del Do While para el menú de modificaciones, las cuales se realizan mediante el método SET debido a el encapsulamiento de los atributos.

Método de Baja aplicado para los objetos; Proveedor, Cliente, Vendedor, Venta y Compra.

```

public void elimCom ()
{
    char opc;
    System.out.println("\n|ELIMINACION DE COMPRA|\n");
    int idCom = IBIO.inputInt("\n| Id de compra: ");
    for (int i = 0; i < compras.size(); i++)
    {
        if (compras.get(i).getId() == idCom)
        {
            compras.get(i).verComp();
            opc = IBIO.inputChar("\n| Ingrese E si desea proseguir con la eliminacion, de lo contrar
            if (opc == 'E')
            {
                compras.remove(i);
            }
        }
    }
}
}

```

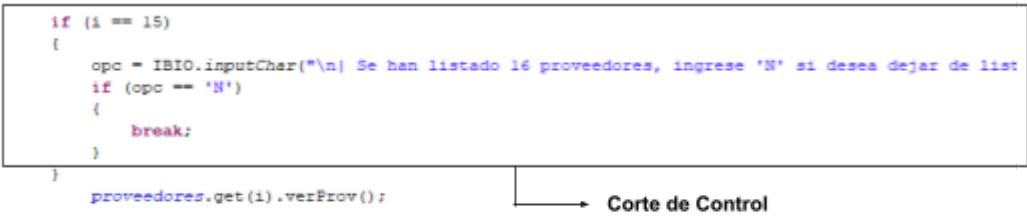
En este caso se ingresa el identificador, se busca mediante un FOR con el tamaño de la lista, se muestra mediante un método dentro de la clase identidad, se consulta

si definitivamente se desea borrar el objeto, se aplica un condicional, si la respuesta es a favor de la eliminación se finaliza con la función `.remove` eliminando de forma permanente el objeto.

Método de listar utilizados en los objetos; Proveedor, Cliente, Vendedor, Venta, Compra, Insumo Proveedor, Insumos y Costo Extra.

```
public void listarXproveedor ()
{
    char opc;

    for (int i = 0; i < proveedores.size(); i++)
    {
        if (i == 15)
        {
            opc = IBIO.inputChar("\n Se han listado 16 proveedores, ingrese 'N' si desea dejar de list
            if (opc == 'N')
            {
                break;
            }
        }
        proveedores.get(i).verProv();
    }
}
```



Corte de Control

En todos los casos se recorre la lista mediante un FOR, se aplica el corte de control que permite que el usuario elija, luego de una cierta cantidad de muestreos, si el desea continuar listando los, si así lo es, se realiza el muestreo mediante un método dentro de la clase identidad.

```

public void verProv()
{
    char opcC;

    System.out.println("\n| PROVEEDOR: " + this.razonSoc + " |");
    System.out.println("\n| Cuit/Cuit: " + this.cuilCuit);
    System.out.println("\n| Telefono: " + this.tel);
    System.out.println("\n| Mail: " + this.mail);
    System.out.println("\n| Direccion: " + this.direccion);
    System.out.println("\n| Estado: " + this.estado);
    System.out.println("\n| Insumo: ");
    this.verInsumo();//datos de INSUMOPROV
    opcC= IBIO.inputChar("\n| Ingrese S si desea ver la lista de compras al proveedor, de lo contrario c

    if (opcC == 'S')
    {
        for (int i = 0; i < compras.size(); i++)
        {
            compras.get(i).verComp();
        }
    }
}
}

```

Como se observa se hace referencia a los atributos mediante el “THIS” ya que el método se encuentra dentro de la misma clase, apuntando a un objeto particular de la lista.

Actualizaciones y cargas a partir de una nueva compra.

Al realizar una nueva compra se debe ingresar sobre qué proveedor se desea actuar se carga el Identificador del insumo en una nueva variable.

```

if (GestionProveedores.proveedores.get(i).getCuilCuit().equalsIgnoreCase(ccProv))
{
    provCom = GestionProveedores.proveedores.get(i);
    idInsProv = GestionProveedores.proveedores.get(i).getInsumo().getId();
}

```

Luego, a partir de ello se identifica el objeto completo de ese identificador de insumo, se guarda en una variable para luego ser asignado como atributo de la nueva compra. A partir de ello se actualiza el stock del mismo mediante el acceso a un GET en la posición guardada en una variable auxiliar y el método de actualizar el



stock dentro de la clase identidad. Por otro lado, se hace un condicional que determina si el precio será modificado o no lo será dependiente el valor antiguo y el nuevo. Si el mismo debe ser actualizado se accede a un nuevo método de actualización de precio de la misma forma que se prosiguió con el stock, además, se actualiza el precio dentro de los atributos encapsulados de el proveedor, también a través de un método dentro de la clase identidad.

```
for (int j = 0; j < MetodosGenerales.insumos.size(); j++)
{
    if ((MetodosGenerales.insumos.get(j).getId() == idInsProv))
    {
        insCom = MetodosGenerales.insumos.get(j);
        posIn = j;
    }
}
id = compras.size()+1;
Compra nueva = new Compra (id,fecha, estado, provCom, insCom, kgU, precio
compras.add(nueva);
GestionProveedores.proveedores.get(posPr).addCom(nueva);
MetodosGenerales.insumos.get(posIn).addCom(nueva);
if (estado == 'E')
{
    MetodosGenerales.insumos.get(posIn).actStock(kgU);
}

//si es mayor lo cambio si es menor no lo cambio
if (precioFi > MetodosGenerales.insumos.get(posIn).getPrecio() )
{
    MetodosGenerales.insumos.get(posIn).actPrecio(precioFi);
    GestionProveedores.proveedores.get(posPr).actPrecio(precioFi);
}
```

Método dentro de la clase Producto y de la clase Proveedor

```
public void actPrecio(double precioFi)
{
    this.insumo.setPrecio(precioFi);
}
```

## Verificación de Stock.

El sistema cuenta con un método de verificación de stock, el cual permite obtener cuanto stock es necesario de cada insumo para producir ciertos KG de producto, además advierte si el stock es insuficiente y permite redirigir a una nueva compra de los necesarios.

Se detalla en los comentarios del código el proceso.

```

public void veriStock ()
{
    System.out.println("\n|VERIFICACION DE STOCK|\n");
    int g = 0;
    int t = 0;
    int e = 0;
    int et = 0;
    double ingKg = IBIO.inputDouble("\n| Cantidad de KG a producir: ");
    //stock necesario por kg=
    double stGraf = 0.3;
    double stTalco = 0.7;
    double stEnv = 0.2;
    double stEti = 0.2;

    //Stock necesario teniendo en cuenta los KG a producir
    double stNgraf = ingKg * stGraf;
    double stNtalco = ingKg * stTalco;
    double stNenv = ingKg * stEnv;
    double stNeti = ingKg * stEti;

    //Muestra el stock necesario de cada insumo para producir los KG ingresados
    System.out.println("\n|El stock que se necesita de cada insumo para producir " + ingKg + "Kg");
    System.out.println("\n| GRAFITO: " + stNgraf);
    System.out.println("\n| TALCO: " + stNtalco);
    System.out.println("\n| ENVASES: " + stNenv);
    System.out.println("\n| ETIQUETAS: " + stNeti + "\n");

    //Consulta si hay el stock necesario
    //Grafito 1, talco 2, envases 3, etiquetas 4.

    if (MetodosGenerales.insumos.get(0).getStock() < stNgraf)
    {
        System.out.println("\n|EL STOCK DE GRAFITO ES INSUFICIENTE");
        g = 1;
    }

    if (MetodosGenerales.insumos.get(1).getStock() < stNtalco)
    {
        System.out.println("\n|EL STOCK DE TALCO INDUSTRIAL ES INSUFICIENTE");
        t = 1;
    }

    if (MetodosGenerales.insumos.get(2).getStock() < stNenv)
    {
        System.out.println("\n|EL STOCK DE ENVASES ES INSUFICIENTE");
        e = 1;
    }

    if (MetodosGenerales.insumos.get(3).getStock() < stNeti)
    {
        System.out.println("\n|EL STOCK DE ETIQUETAS ES INSUFICIENTE");
        et = 1;
    }

    if ((g == 1) || (t == 1) || (e == 1) || (et == 1))
    {
        char opc = IBIO.inputChar("\n|¿DESEA REALIZAR UNA NUEVA COMPRA DE INSUMOS? [S(en :
        if (opc == 'S')
        {
            compras.nuevaCom();
        }
    }
}

```

El stock necesario por KG de cada uno de los insumos proviene de las conversaciones previas con mi cliente

Variables auxiliares, utilizadas luego en el condicional.

Si alguno de los STOCKS resulta ser insuficiente consulta si se desea realizar una nueva compra

Redirige al método para realizar una nueva compra

Proceso de cálculo de costo final:

- Se calcula el costo de los insumos por kg.

```
private double calCosIns()
{
    double costTalco = 0;
    double costGraf = 0;
    double costEnv = 0;
    double costEti = 0;
    double costFI = 0;

    for (int i = 0; i < insumos.size(); i++)
    {
        if (insumos.get(i).getId() == 00001)
        {
            costGraf = 30*insumos.get(i).getPrecio()/100;
        }

        if (insumos.get(i).getId() == 00002)
        {
            costTalco = 70*insumos.get(i).getPrecio()/100;
        }

        if (insumos.get(i).getId() == 00003)
        {
            costEnv = insumos.get(i).getPrecio()/5;//Bidones de 5kg.
        }

        if (insumos.get(i).getId() == 00004)
        {
            costEti = insumos.get(i).getPrecio()/5;//Una etiqueta por bidon.
        }
    }

    costFI = costTalco + costGraf + costEnv + costEti;
    return costFI;
    // 70 TALCO 30 GRAFITO, 1 ETIQUETA, 1 ENVASE
}
```

Método que devuelve un atributo de tipo double

Código de insumo pre establecido

Como menciona el precio se guarda el atributo correspondiente a cada uno de los objetos mediante las actualizaciones de stock surgientes de las compras

Suma de todos los costos finales de cada uno de los insumos por KG

Devuelve el costo final de insumos por KG

```
public double calCostF()
{
    double costFI = calCosIns();
    // 0 -- Ganancia en numero pero luego utilizada como %.
    double aux = 0;
    double costFAux = 0;

    for (int i = 0; i < costosExtra.size(); i++)
    {
        if ((costosExtra.get(i).getId() != 0) && (costosExtra.get(i).getEstado() == 'A'))
        {
            aux = aux + costosExtra.get(i).getImporte();
        }
    }

    costFAux = costFI + aux;

    for (int i = 0; i < costosExtra.size(); i++)
    {
        if (costosExtra.get(i).getId() == 0)
        {
            ganancia = costosExtra.get(i).getImporte()*costFAux/100;
        }
    }

    costFinal = costFAux + ganancia;
    return costFinal;
}
```

Se guarda dentro de una nueva variable el costo final de insumo que devuelve el método 2

Se recorre la lista de costos extra y se los suma si es que están ACTIVOS, y si son distintos de "0", ya que esta es la ganancia

Se suman los costos finales de insumos + los costos finales de costos extra

Se busca el número cargado en el costo extra "0" (ganancia) y se calcula la ganancia en referencia a el costFAux

Se suma el costFAux y la ganancia y se obtiene el costo final definitivo por KG

Devuelve el costo final por KG

```

public double [] calcCostosFdes (double kg, double Vdolar, double descFdis)
{
    calCostF(); → Se llama el metodo que obtiene el costo final por KG de producto
    double [] valores = new double [2]; → Se crea el arreglo donde se guardará el costo
    double costXkg = kg*costFinal; → final por pedido en dólares y en pesos
    double calcDes = descFdis*costXkg/100; → Se calcula cuánto hay que descontar en el precio final del pedido
    double cfinal = costXkg - calcDes; → debido a los descuentos
    System.out.println("\n|Costo en pesos: " + cfinal);
    double cfinalDl = cfinal/Vdolar;
    System.out.println("\n|Costo en Dolares: " + cfinalDl);
    valores [0] = cfinal; → Se cargan los costos tanto en pesos como en
    valores [1] = cfinalDl; → dólares en el arreglo, luego se lo devuelve
    return valores; → mediante un "return"
}

```

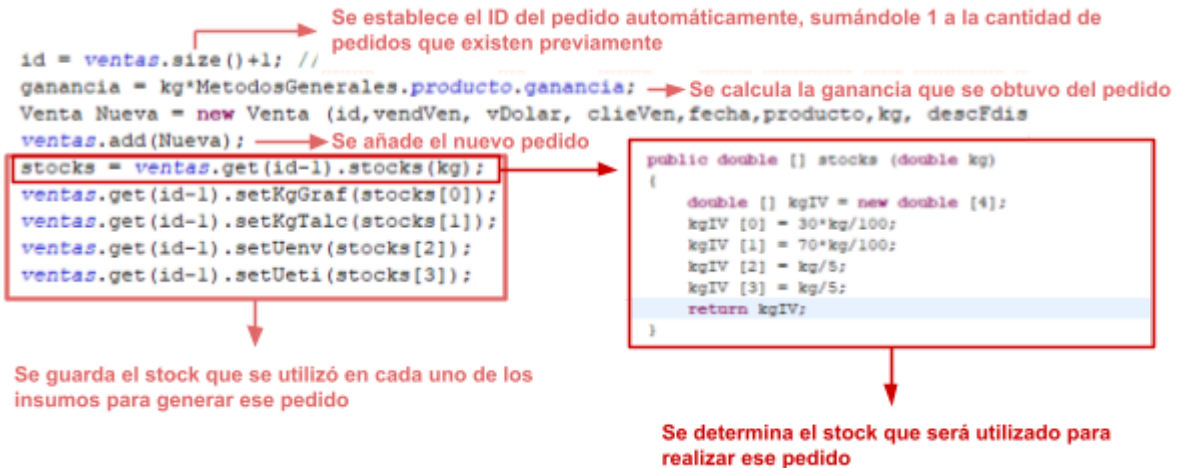
- Si no se presentan descuentos existe otro método que realiza el mismo procedimiento sin aplicar la debida parte de la resta por descuentos.

Se calcula el costo de los costos extra por KG y se lo suma a el costo obtenido anteriormente de los insumos por KG. Además, a partir de ello se calcula la ganancia y se la suma al costo obtenido anteriormente de la suma de los insumos y de los costos extra, obteniendo así el costo final por KG de producto y guardandolo en el atributo costoFinal.

Nuevo pedido:

Se le asigna un cliente y vendedor, se ingresa la cantidad de KG y a partir de ello se utilizan los métodos desarrollados anteriormente para establecer el precio del mismo, y la ganancia que se obtuvo.

A mi parecer es crucial mencionar el siguiente código:



Luego se calculan los atributos necesarios para el objeto.

Cuando se realiza un nuevo pedido el estado de pago y de entrega se ponen automáticamente en PENDIENTE, por lo que existe una opción donde estos pedidos pueden ser visualizados y modificados.



Tanto para los pedidos como para las ventas se utiliza la misma lista, ya que la única diferencia entre estos es el estado de las diferentes categorías.

Cuando los pedidos dejan de estar pendientes en ambas categorías pasan a ser ventas.

```

for (int i = 0; i < GestionPedidos.ventas.size(); i++)
{
    if ((GestionPedidos.ventas.get(i).getEstadoEnv() == 'C') && (GestionPedidos.ventas.get(i).getEstadoPag() == 'C'))
    {
        GestionPedidos.ventas.get(i).verH();
    }
}

public void verH ()
{
    System.out.println("\nId: " + this.id);
    System.out.println("\nFecha: " + this.fecha);
    System.out.println("\nCliente: " + this.cliente.getRazonSoc() + " - " + this.cliente.getCuilCuit());
    System.out.println("\nVendedor: " + this.vendedor.getRazonSoc() + " - " + this.vendedor.getCuilCuit());
    System.out.println("\nKg vendidos: " + this.kg);
    System.out.println("\nDescuentos: " + this.desc);
    System.out.println("\nPrecio final pesos : " + this.vprecFinal);
    System.out.println("\nPrecio final dolares : " + this.vprecFinalDl);
    System.out.println("\nGanancia: " + this.ganancia);
}

```

Se utiliza la siguiente condición para determinar si estamos en frente de un pedido o una venta

Este método muestra unicamente los datos relevantes (según mi cliente) de las ventas