MEMORIA ESCRITA PRÁCTICA 1
DISTRIBUIDORA JMLMJ SAS

Presentado por:
Mónica Sofía Restrepo León - *morestrepol@unal.edu.co*
Maria Fernanda Calle Agudelo - *mcalleag@unal.edu.co*
Luis Alejandro Varela Ojeda - *luvarelao@unal.edu.co*
Jaider Castañeda Villa - *jcastanedavi@unal.edu.co*
Joan Sebastian Salazar Montoya - *josalazarmo@unal.edu.co*

Grupo 1 - Equipo 4

Profesor:
Jaime Alberto Guzman Luna
jaguzman@unal.edu.co

Jueves 25 de Mayo

Universidad Nacional de Colombia
Facultad de Minas
2023-1

CONTENIDO

I. DESCRIPCIÓN GENERAL DE LA SOLUCIÓN

Análisis, diseño e implementación

Distribuidora JMLMJ SAS es un programa diseñado para llevar el control de una empresa de distribuciones de diferentes tipos de productos, vista desde el punto de un administrador que es el encargado de controlar todos los procesos, las principales partes del programa son:

- **Fábrica:** Existe solo una y acá sucede la creación de los productos que se envían a las diferentes sucursales para abastecerlas.

- **Tiendas:** Desde acá se realizan las ventas a los clientes de los productos que hayan disponibles en esa tienda, la empresa cuenta con 3 principales sucursales

- **Productos:** Distribuidora SAS maneja 3 categorías de productos, Consumibles o alimentos, limpieza y productos de construcción.

- **Transportes:** la distribuidora cuenta con una lista predeterminada de transportes que se usan para enviar los productos ya sea desde la fábrica a las tiendas, como de las tiendas a los clientes .
- **Trabajadores:** tenemos 3 tipos de trabajadores, operarios (trabajan en la fábrica) sólo existe uno ya que hay una sola fábrica,, Vendedores (trabajan en las tiendas) tenemos 3 vendedores, uno por cada tienda y por último, conductores (manejan los transportes).

- **Cuentas Bancarias:** para la empresa se maneja una sola cuenta general, es la misma para las 3 tiendas existentes y para la única fábrica, luego están las cuentas de los clientes de donde se les descuenta el dinero al momento de hacer un envío, y por último las de cada trabajador de la distribuidora.

a continuación una breve descripción de cada parte disponible del programa:

1. **Enviar Pedidos:** Esta es la principal herramienta de la aplicación en donde el administrador podrá realizar envíos desde alguna tienda a diferentes clientes, seleccionar productos y hasta el tipo de transporte que se necesitará.

2. **Abastecer Tiendas**: esta opción permite enviar productos desde una fábrica principal a 3 principales tiendas según el tipo de producto y la capacidad de de cada tienda.

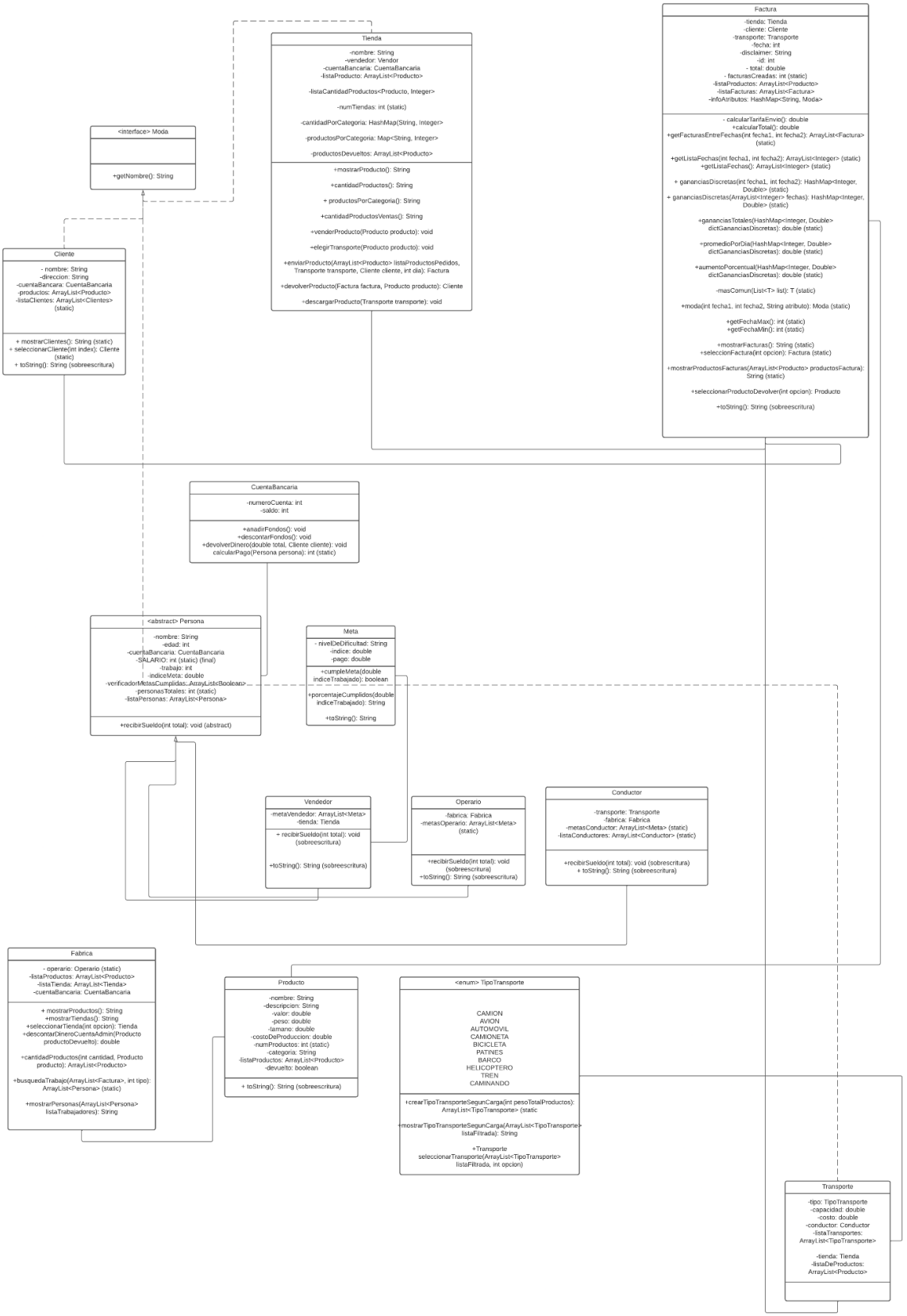
3. **Pago Trabajadores:** Le permite al encargado pagar un sueldo base y bonos a la cuenta dependiendo de las metas logradas a los diferentes trabajadores de la empresa: operarios, vendedores y conductores.

4. **Realizar Devoluciones:** Como encargado de manejar la aplicación pueden ocurrir errores al momento de hacer un envió a un cliente, por lo tanto en base a la factura que se genera en un envío puede devolver productos.

5. **Estadísticas:** este apartado permite observar el rendimiento de la empresa para saber que productos abastecer y así poder vender más, desde las ganancias discretas, totales, promedios y aumento porcentual.

II. DESCRIPCIÓN DEL DISEÑO ESTÁTICO DEL SISTEMA

Diagrama de clases y objetos



Link Diagrama de clases y objetos

El sistema de distribución SAS está compuesto por 12 clases, sumadas a 1 interfaz y 1 enumerado. Lo anterior está dividido en dos paquetes, los cuales se encuentran en el paquete *gestorAplicación:*

➔ Paquete *gestión*:
Contiene las clases relacionadas con la mano de obra de la empresa, los clientes y su proceso de pago, como lo son las cuentas bancarias y las facturas de los envíos.

- Persona: clase abstracta que proporciona atributos y métodos comunes para la creación de los distintos tipos de trabajadores de la empresa.
    - Cada una tiene asociada una cuenta bancaria.

- Conductor: clase que hereda de Persona, que representa al trabajador que transporta los envíos en determinado transporte desde la tienda hasta los clientes o desde la fábrica hasta la tienda en el momento de abastecer.
    - Cada uno está asociado a un transporte.

- Vendedor: clase que hereda de Persona, que representa al trabajador que realiza la venta en su respectiva tienda.
    - Cada uno está asociado a una tienda diferente.

- Operario: clase que hereda de Persona, que representa al trabajador que controla la producción de los productos en las fábricas.
    - Está asociado a la única fábrica que tiene la empresa.

- Cliente: clase que representa al comprador de productos de la empresa.

- CuentaBancaria: clase que representa una cuenta bancaria la que se utiliza para realizar los respectivos pagos y descuentos tanto a los trabajadores y clientes, como a las cuentas de la fábrica y las tiendas.

- Factura: clase que permite guardar la información importante de los pedidos que se han realizado como lo son el cliente, la tienda, la lista de productos y el transporte utilizado.

- Meta: clase que representa las metas que se le pueden dar a los trabajadores para recibir incentivos monetarios por el cumplimiento de ciertos objetivos.

- Moda: interfaz que permite marcar objetos para obtener modas en la funcionalidad de estadísticas.

➔ Paquete *producción*:
Contiene las clases relacionadas con la producción y distribución de los productos de la empresa.

- Fabrica: clase que representa el lugar de fabricación de los productos y de donde salen para ser abastecidos a las diferentes tiendas.
    - Hay una única fábrica, que abastece a varias tiendas.
    - La fábrica tiene asociada 1 operario.
    - Tiene su propia cuenta bancaria.

- Tienda: clase que representa la tienda donde se venden los productos.
    - Cada una tiene su propia cuenta bancaria.
    - Cada una tiene un vendedor diferente.

- Producto: clase que representa el producto que se genera en la fábrica y se vende en las tiendas.

- Transporte: clase que representa un medio de transporte que se puede utilizar para mover productos entre una fábrica y una tienda o una tienda y el cliente.
    - Cada uno está asociado a un conductor.

- TipoTransporte: enumerado que establece el conjunto de valores constantes disponibles para los tipos de transportes en los que se pueden transportar los productos.

III. DESCRIPCIÓN DE LA IMPLEMENTACIÓN DE CARACTERÍSTICAS DE PROGRAMACIÓN ORIENTADA A OBJETOS

Clase abstracta
Se implementa la clase abstracta Persona en el paquete **gestorAplicacion.gestion;**
de ella heredarán las clases Vendedor, Operario y Conductor.

```
11
12  public abstract class Persona implements Serializable {
13
14
15      /*-------------------Atributos-------------------------*/
16
17      private static final long serialVersionUID = 12736217L;
18
19
20      private String nombre;
21      private int edad;
22      private int cedula;
23      private CuentaBancaria cuentaBancaria;
24      private static final int SALARIO=1000;
25      private int trabajo;
26      private double indiceMeta;
27      private ArrayList<Boolean> verificadorMetasCumplidas = new ArrayList<Boolean>();
28      private static int personasTotales;
29      private static  ArrayList<Persona> listaPersonas = new ArrayList<Persona>();
30
```

Como es una clase abstracta no puede ser instanciada directamente, sino que sirve como base para otras clases. (En este caso Vendedor, Operario y Conductor.) Se utiliza para establecer métodos y propiedades que deben ser implementados por las clases derivadas. Se utiliza la palabra clave "abstract" para probarla antes de la declaración de la clase, y se pueden incluir métodos abstractos (sin implementación) que deben ser definidos en las clases derivadas.

Método abstracto
Se implementa el método abstracto recibirSueldo en la línea 57 de la clase abstracta Persona explicada anteriormente

```
54
55      /*-------------------- Metodos -------------------------*/
56
57          public abstract void recibirSueldo(int total);
58
59
```

Y su implementación en las clases hijas se ve en las clases Vendedor, Operario y Conductor. El siguiente ejemplo es de la clase Vendedor, en la línea 47.

```java
    /*-------------------Metodos-------------------------*/
    @Override
    public void recibirSueldo(int total) {
        tienda.getCuentaBancaria().descontarFondos(total);
        super.getCuentaBancaria().anadirFondos(total);
    }

```

Interfaz
Se implementa la interfaz Moda.

La interfaz permite marcar objetos para obtener modas en la funcionalidad de estadísticas.

src > gestorAplicacion > gestion > Moda.java > ...

```java
package gestorAplicacion.gestion;

public interface Moda {

    public String getNombre();

}

```

Podemos verla implementada en la clase Transporte.

```java
import java.util.ArrayList;
import gestorAplicacion.gestion.Moda;


import gestorAplicacion.gestion.Conductor;

public class Transporte implements Moda{
    //Atributos

    private TipoTransporte tipo;
    private Double capacidad;
    private double costo;
    private Conductor conductor;
    private ArrayList<TipoTransporte> listaTransportes;
```

En la siguiente imagen, se da uso al método de la interfaz moda dentro de la clase Transporte.

```java
//Getters y setters


//de la interfaz Moda
public String getNombre(){
   return tipo.getNombre();
}
//para tipo
public TipoTransporte getTipo() {
   return tipo;
}
```

Dentro de la funcionalidad Estadística, esto sirve para saber cual es el transporte usado más frecuentemente por los clientes, información valiosa para una empresa distribuidora.

Herencia

Se implementa la herencia con la clase Persona, con sus subclases Vendedor, Operario y Conductor. Esto para agrupar sus similitudes y así facilitando la reutilización de código, un ejemplo de esto son los atributos heredados de la clase padre que son generales para sus hijos, los cuales cada uno tiene un atributo singular dependiendo del trabajo que realizan.

- Clase padre Persona

```java
public abstract class Persona implements Serializable {


    /*-------------------Atributos-------------------------*/

    private static final long serialVersionUID = 12736217L;


    private String nombre;
    private int edad;
    private int cedula;
    private CuentaBancaria cuentaBancaria;
    private static final int SALARIO=1000;
    private int trabajo;
    private double indiceMeta;
    private ArrayList<Boolean> verificadorMetasCumplidas = new ArrayList<Boolean>();
    private static int personasTotales;
    private static  ArrayList<Persona> listaPersonas = new ArrayList<Persona>();

```

- Clase Hija Vendedor

```
13
14  public class Vendedor extends Persona {
15      /*-------------------Atributos-------------------------*/
16
17      private static ArrayList<Meta> metasVendedor = new ArrayList<Meta>(List.of(new Meta(nivelDeDificultad:
18                                                                                  new Meta(nivelDeDificultad:
19      private Tienda tienda;
20
21      /*-------------------Constructores-------------------------*/
22
23
24
25      /*Constructor de la clase conductor que recibe todos los parametros*/
26
27      public Vendedor(String nombre, int edad, int cedula, CuentaBancaria cuentaBancaria,Tienda tienda) {
28          super(nombre, edad, cedula, cuentaBancaria);
29          this.tienda=tienda;
30      }
```

En el constructor de la línea 27, podemos ver que la clase Vendedor hereda los atributos nombre, cuentaBancaria y cédula de la clase padre. El único atributo que se declara en la misma clase es la tienda, declarado en la línea 19.

La siguiente foto es de la clase serializada Load, podemos ver un objeto de esta clase donde se evidencia la herencia:

```
129             //vendedores
130
131             Vendedor vendedor1 = new Vendedor(nombre:"Maria Beatriz",edad:20,cedula:57793,cuentaVendedor1,tienda:null);
132             Vendedor vendedor2 = new Vendedor(nombre:"Adriana Alexia Putellas",edad:21,cedula:89235,cuentaVendedor2,tienc
133             Vendedor vendedor3 = new Vendedor(nombre:"Lionel Andres Messi",edad:22,cedula:14720,cuentaVendedor3,tienda:nu
134
```

- Clase Hija Operario

```
14  public class Operario extends Persona {
15
16      /*-------------------Atributos-------------------------*/
17
18      private static ArrayList<Meta> metasOperario = new ArrayList<Meta>(List.of(new Meta(nivelDeDificultad:"Fáci
19                                                                                  new Meta(nivelDeDificultad:"Dif
20      private Fabrica fabrica;
21
22
23      /*-------------------Constructores-------------------------*/
24
25
26      /*Constructor de la clase operario que recibe todos los parametros*/
27
28      public Operario(String nombre, int edad, int cedula, CuentaBancaria cuentaBancaria,Fabrica fabrica) {
29          super(nombre, edad, cedula, cuentaBancaria);
30          this.fabrica=fabrica;
31      }
```

La siguiente foto es de la clase serializada Load, podemos ver un objeto de esta clase donde se evidencia la herencia: :

```
168             //conductores
169             Conductor conductor1 = new Conductor(nombre:"Pablo Ramirez", edad:45, cedula:544764513, cuentaConductor1,tra
170             Conductor conductor2 = new Conductor(nombre:"Hernando Cruz", edad:50, cedula:645541321, cuentaConductor2,tra
171             Conductor conductor3 = new Conductor(nombre:"Mario Casas", edad:25, cedula:13216531, cuentaConductor3,transp
```

- Clase Hija Conductor

```
16  public class Conductor extends Persona {
17
18      private Transporte transporte;
19      private Fabrica fabrica;
20      private static ArrayList<Meta> metasConductor = new ArrayList<Meta>(List.of(new Meta(nivelDeDificultad:"Fácil",indi
21                                                                                  new Meta(nivelDeDificultad:"Dificil",in
22      private static ArrayList<Conductor> listaConductores = new ArrayList<Conductor>();
23      /*------------------Constructores-------------------------*/
24
25
26
27      /*Constructor de la clase conductor que recibe todos los parametros*/
28
29      public Conductor(String nombre, int edad, int cedula, CuentaBancaria cuentaBancaria,Transporte transporte) {
30          super(nombre, edad, cedula, cuentaBancaria);
31          this.transporte=transporte;
32          listaConductores.add(this);
33      }
```

En todas las clases hijas se usan los atributos de la clase padre. Podemos evidenciar esto en los objetos.

La siguiente foto es de la clase serializada Load, donde se evidencia la herencia:

```
169             Conductor conductor1 = new Conductor(nombre:"Pablo Ramirez", edad:45, cedula:544764513, cuentaConductor1,trans
170             Conductor conductor2 = new Conductor(nombre:"Hernando Cruz", edad:50, cedula:645541321, cuentaConductor2,trans
171             Conductor conductor3 = new Conductor(nombre:"Mario Casas", edad:25, cedula:13216531, cuentaConductor3,transpor
172             conductor1.setFabrica(fabrica);
173             conductor2.setFabrica(fabrica);
174             conductor3.setFabrica(fabrica);
175
```

Nombre, edad y cédula son atributos de la clase padre, que se manifiestan en los objetos de sus clases hijas.

Ligadura dinámica

- Caso 1: Interfaz moda

```
public interface Moda {

    public String getNombre();

}
```

Todos los objetos que implementan la clase Moda, deben hacer la implementación de un método getNombre() aquí es donde se ve reflejada la ligadura dinámica.

```
case 1:

    System.out.println("La tienda más usada ha sido "
    + Factura.moda(fecha1, fecha2, atributo:"tienda").getNombre());

    break;

case 2:

    System.out.println("El transporte más usado ha sido "
    + Factura.moda(fecha1, fecha2, atributo:"transporte").getNombre());

    break;

case 3:

    System.out.println("El cliente al que más se le ha vendido ha sido "
    + Factura.moda(fecha1, fecha2, atributo:"cliente").getNombre());

    break;
```

Se sigue que en UiEstadísticas se utiliza el método moda. Como este método devuelve un objeto que implementa Moda, este objeto debe implementar el método getNombre(). La ligadura dinámica está en el hecho de que cuando se llama getNombre() aquí, se está yendo es a la implementación específica de cada objeto de este método (Tienda, Cliente, Transporte)

- Caso 2: mostrarPersonas

```
public static String mostrarPersonas(ArrayList<Persona> listaTrabajadores){
    String texto = "";
    int indice = 1;

    for (Persona i: listaTrabajadores) {
        texto += "\n" + "Trabajador "+ indice + i.toString();  //Uso de ligadura dinámica
        indice++;
    }

    return texto;
}
```

Este método de la clase Fabrica en la línea 207 cuenta con ligadura dinámica debido a que se usan referencias de una clase abstracta(Persona) hacia las subclases(Conductor,Operario, Vendedor) de esta.

Atributo de clase
En la clase Clientes en la línea 21 se declara e inicia un ArrayList de tipo cliente el cual se llama listaClientes y tiene como función guardar todos los clientes creados el objeto es de tipo static debido a que se necesita que todos los clientes compartan la lista.

```
21    static private ArrayList<Cliente> listaClientes = new ArrayList<Cliente>();
```

Método de clase
El método es gananciasTotales de la clase factura en la línea 238 este método es estático debido al atributo estático que recibe.

```java
public static double gananciasTotales(HashMap<Integer, Double> dictGananciasDiscretas){

    double suma = 0;

    for(int fecha: dictGananciasDiscretas.keySet()){
        suma += dictGananciasDiscretas.get(fecha);
    }

    return suma;

}
```

Uso de Constante

Se implementa el uso de constante en la línea 16 la clase Persona del paquete gestión, con el atributo salario de clase entero; esto nos permite tener un salario base fijo para luego a partir de este calcular el de las diferentes clases que heredan de Persona (Operario, Vendedor y Trabajador) dependiendo de trabajo desempeñan y la cantidad.

```java
16      private static final int SALARIO=1000;
```

Encapsulamiento

Privados

```java
//Atributos
private String nombre;
private String descripcion;
private double valor;
private double peso;
private double tamano;
private double costoDeProduccion;
private static int numProductos = 0;
private String categoria;
```

```java
//Get and Set
public String getNombre() {
    return this.nombre;
}

public boolean isDevuelto() {
    return devuelto;
}

public void setDevuelto(boolean devuelto) {
    this.devuelto = devuelto;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getDescripcion() {
    return this.descripcion;
}

public void setDescripcion(String descripcion) {
    this.descripcion = descripcion;
}
```

En la clase Producto se definen los atributos privados en la línea 15, luego al final en la línea 60 podemos observar los getters and setters public para poder acceder a estos atributos y así obtener o modificar sus valores desde otras clases

Protected

```java
36      /*Constructor de la clase conductor que recibe todos los parametros*/
37      protected Persona(String nombre, int edad, int cedula, CuentaBancaria cuentaBancaria)
```

En la clase Persona línea 37, el constructor de persona es protegido ya que solo lo utilizan las subclases, operario, vendedor y conducto

Sobrecarga de métodos

```
189
190  public static HashMap<Integer, Double> gananciasDiscretas(int fecha1, int fecha2){
191
```

```
225  public static HashMap<Integer, Double> gananciasDiscretas(ArrayList<Integer> fechas){
226
```

En la clase Factura en las líneas 190 y 225 podemos ver como 2 métodos se llaman igual y retornan lo mismo, con la diferencia de que uno recibe 2 enteros y el otro recibe un ArrayList de enteros, ocurriendo así la sobrecarga, dependiendo del parámetro con el que se llame entrará a alguno de los 2 métodos.

Sobrecarga de constructores

```
52      public Factura(Tienda tienda, Cliente cliente, Transporte transporte, ArrayList<Producto> listaProductos,
53                     int fecha, String disclaimer, Operario operario) {
```

```
74      public Factura(Tienda tienda, Cliente cliente, Transporte transporte, ArrayList<Producto> listaProductos,
75                     int fecha, Operario operario) {
```

En la clase Factura en las líneas 52 y 74 observamos los respectivos constructores que reciben parámetros muy similares con la diferencia de que el segundo no recibe un "disclaimer" o descripción donde el otro constructor si, así cuando se cree un nuevo objeto de la clase Factura se le puede pasar o no como parámetro ese atributo.

Manejo de referencias this para desambiguar y this()

- this para desambiguar

  ➔ Caso 1

```
31      public Fabrica(ArrayList<Producto> listaProductos, ArrayList<Tienda> listaTienda,Cuent
32          this.listaProductos = listaProductos;
33          this.listaTienda = listaTienda;
34          this.cuentaBancaria = cuentaBancaria;
35          this.operario = operario;
36      }
```

En la línea 32 de la clase Fábrica se usa el this para desambiguar con los nombres de los parámetros del constructor.

  ➔ Caso 2

```
52      public Factura(Tienda tienda, Cliente cliente, Transporte transporte, ArrayList<Producto> listaProductos,
53                     int fecha, String disclaimer, Operario operario) {
54          this.tienda = tienda;
55          this.cliente = cliente;
56          this.transporte = transporte;
57          this.listaProductos = listaProductos;
58          this.fecha = fecha;
59          this.disclaimer = disclaimer;
60          this.operario = operario;
61
62          infoAtributos.put(key:"tienda", tienda);
63          infoAtributos.put(key:"transporte", transporte);
64          infoAtributos.put(key:"cliente", cliente);
65
66
67          this.total = calcularTotal();
68          listaFacturas.add(this);
69
70          this.id = ++facturasCreadas;
71      }
```

En la clase Factura de la línea 54 hasta la 70 se usa el this para desambiguar los nombre de los parámetros que tiene el constructor con los atributos que pertenecen a la clase.

- this()

  ➔ Caso 1

```
74      public Factura(Tienda tienda, Cliente cliente, Transporte transporte, ArrayList<Producto> listaProductos,
75                     int fecha, Operario operario) {
76          this(tienda, cliente, transporte, listaProductos, fecha, disclaimer:"SIN DISCLAIMER",operario);
77      }
```

En la clase Factura, línea 76, se usa el this() para llamar el otro constructor y pasarle el parámetro del disclaimer.

  ➔ Caso 2

```
public Producto(String nombre, double valor, double peso, double tamano, String categoria){
    this(nombre, descripcion:"Sin descripción", valor, peso, tamano, costoDeProduccion:10.0, categoria);
}
```

En la clase Producto en la línea 44 se usa el this para llamar a otro constructor donde se le pasan los parámetros de descripción y costo de producción por defecto.

Enumeración

Se implementa el enumerado TipoTransporte, en el cual se definen una lista fija de transportes posibles que se pueden escoger para el envío de productos:

```
1   package gestorAplicacion.produccion;
2
3   import java.util.ArrayList;
4
5   public enum TipoTransporte {
6
7       //los tipos de transporte que hay
8       CAMION(precioEnvio:15000, capacidadMax:16329, nombre:"Camion"),
9       AVION(precioEnvio:30000, capacidadMax:640000,nombre:"Avion"),
10      AUTOMOVIL(precioEnvio:9000,capacidadMax:500,nombre:"Automovil"),
11      CAMIONETA(precioEnvio:12000, capacidadMax:650,nombre:"Camioneta"),
12      BICICLETA(precioEnvio:5000,capacidadMax:35,nombre:"Bicicleta"),
13      PATINES(precioEnvio:3000,capacidadMax:20,nombre:"Patines"),
14      BARCO(precioEnvio:20000, capacidadMax:33565835,nombre:"Barco"),
15      HELICOPTERO(precioEnvio:70000, capacidadMax:29000,nombre:"Helicoptero"),
16      TREN(precioEnvio:20000,capacidadMax:30000,nombre:"Tren"),
17      CAMINANDO(precioEnvio:5000, capacidadMax:15,nombre:"Caminando");
18
19
```

IV. DESCRIPCIÓN DE FUNCIONALIDADES

Funcionalidad 1: Envío de pedidos

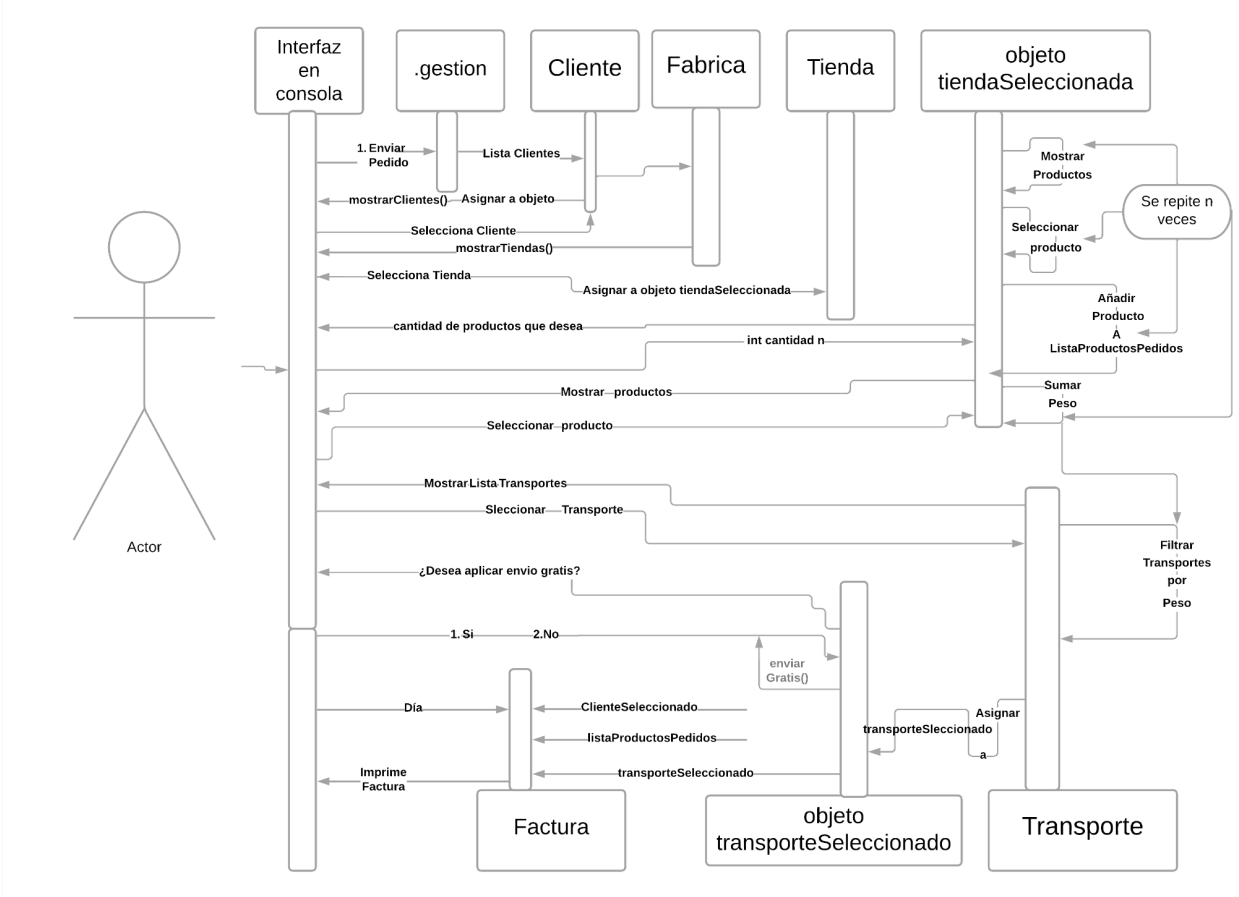
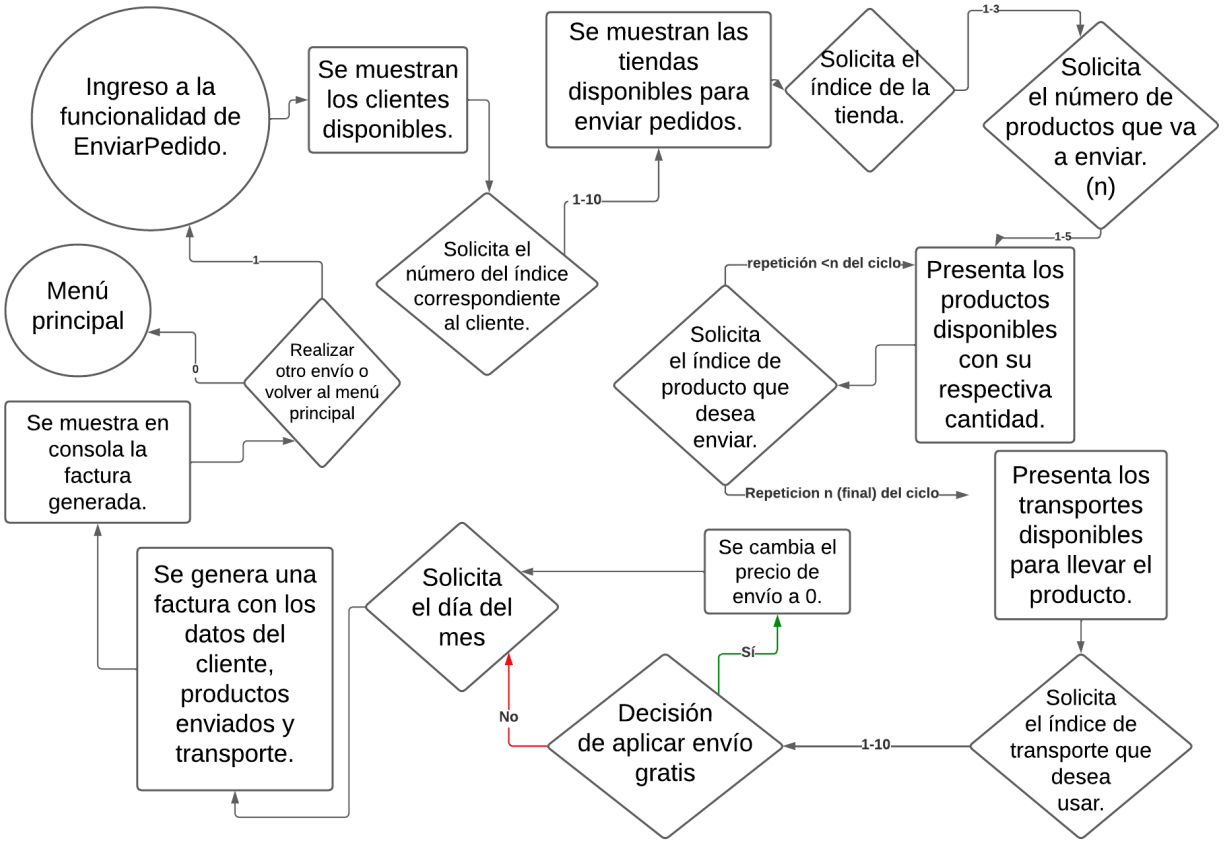


Diagrama de interacción con el usuario:

[Link diagrama interacción Enviar Pedido]

Diagrama de secuencia



[Link Diagrama de secuencia Funcionalidad Enviar Pedido]

La funcionalidad EnviarPedido en Java se basa en menús desplegables que permiten al administrador seleccionar: el cliente al que desea enviar un pedido, la tienda de donde se tomarán los productos, los productos que se enviarán al cliente y el tipo de transporte que se utilizará para enviar el pedido.

En todos los casos hay una opción (con el número 0) para devolverse del menú, es útil en caso de que el administrador se equivoque al introducir una opción.

También hay otra restricción para opciones no válidas, esta detectará cuando el número introducido no tiene objeto correspondiente y desplegará un mensaje de esta manera:

```
Número de cliente inválido, por favor seleccione un cliente en la lista
```

Y repetirá el mismo menú.

Para los casos de switch se utiliza un bucle *while* que se mantiene en ejecución hasta que el usuario decide salir del menú (con la opción 0). Dentro de este bucle se utiliza un switch para identificar en qué elección o paso del proceso se encuentra el usuario en el menú y así realizar la acción correspondiente. Para cada elección se muestra un mensaje en pantalla pidiendo al usuario que seleccione la opción deseada.

Case 1: Permite al usuario seleccionar el cliente al que desea enviar el pedido. Para ello, se muestra una lista de clientes disponibles (con el método **Cliente.mostrarClientes()** y se solicita al usuario que seleccione un número correspondiente al cliente deseado. Usa el método

Cliente.getListaClientes().size() para validar si el número está en el rango deseado. Luego **Cliente.getListaClientes().get(numClienteSeleccionado - 1)**, que selecciona un cliente de la lista y lo asigna al objeto clienteSeleccionado. Imprime en consola el número de elección del cliente, y su nombre. Cuando esto se logra, nos lleva al caso de elección 2.

Ejemplo:

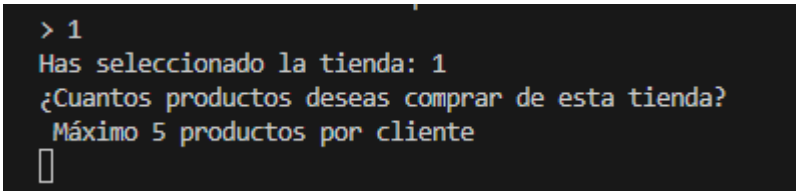
```
Seleccione el cliente al que desea enviar:
1. Juan Pérez Dir. Calle 123
2. María García Dir. Avenida 456
3. Pedro Gómez Dir. Carrera 789
4. Luis Gómez Dir. Carrera 789
5. Josesito Gómez Dir. Carrera 789
6. Julián Araña Alvarez Dir. Carrera 101
7. Emiliano Dibu Martinez Dir. Carrera 23
8. Diego Armando Maradona Dir. Carrera 10
```

Case 2: Permite al usuario seleccionar la tienda de donde se tomarán los productos. Para ello, se muestra una lista de tiendas disponibles (usando el método de instancia **UiMenu.fabrica.mostrarTiendas()**. Después se le pide al administrador que seleccione una de estas tiendas mediante su índice en la lista desplegada. Usa el método **UiMenu.fabrica.getListaTienda().size()**
para validar si el número está en el rango deseado. Luego
tiendaSeleccionada=UiMenu.fabrica.getListaTienda().get(numTiendaSeleccionada - 1);
que selecciona un cliente de la lista y lo asigna al objeto tiendaSeleccionada. Imprime en consola el número de elección de la tienda, y su nombre. Cuando esto se logra, nos lleva al caso de elección 3.

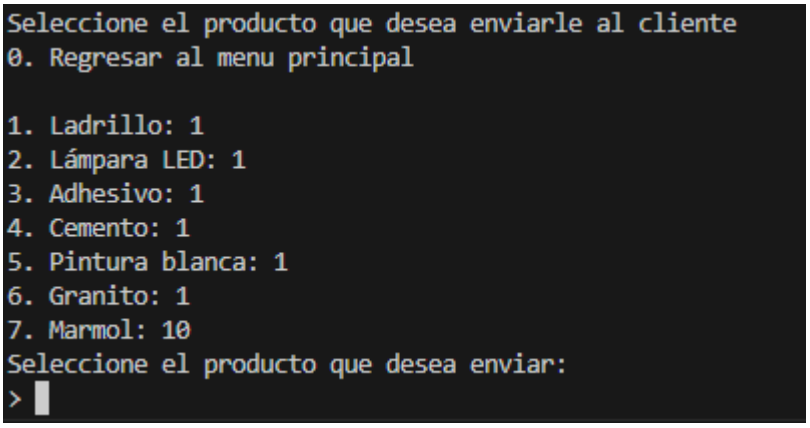
```
Su pedido se enviará desde alguna de estas tiendas, por favor seleccione una:
0. Volver al menu principal
1. Hefesto Construcciones-PRODUCTOS:
     Adhesivo: 1
     Pintura blanca: 1
     Granito: 1
     Lámpara LED: 1
     Cemento: 1
     Marmol: 1
     Ladrillo: 1

2. Consumibles de la Abuela Tata-PRODUCTOS:
     Sandwich de pollo: 1
     Hojas de Te: 1
     Jamon: 1
     Carne de cerdo: 1
     Carne de res: 1
     Galletitas: 1
     Pastel de cumpleaños: 1

3. Miss Músculo Aseo-PRODUCTOS:
     Jabon: 1
     Jabón líquido: 1
     LimpiaMax: 1
     Lavadora: 1
     Papel higiénico: 1
     Detergente: 1
     Bolsas de basura: 1
     Cepillo de dientes: 1
```

Case 3: Seleccionar los productos que se enviarán al cliente. Primero se solicita al usuario que indique cuántos productos desea comprar, y se verifica que el número sea válido. El número máximo de productos posibles es 5.

Así se ve en consola esta comprobación:



Luego, usando el método **tiendaSeleccionada.cantidadProductosVentas()** se muestra una lista de productos disponibles en la tienda seleccionada y se solicita al usuario que seleccione el número correspondiente al producto deseado. Esta acción se repetirá dentro de un bucle n veces, n siendo la cantidad de productos que quiera enviar el administrador, así podrá seleccionar los diferentes productos que desee enviar. La selección se hace uno por uno.
Una repetición de este bucle para seleccionar el producto se ve así:



Por cada uno de estos bucles con un número válido se hace lo siguiente:

productoSeleccionado = tiendaSeleccionada.getListaProductos().get(numProductoSeleccionado - 1);

Esto asigna el producto a la instancia de productoSeleccionado, luego, como son varios productos: **listaProductosPedidos.add(productoSeleccionado);** Que añade la instancia a una lista con los productos pedidos, luego esta lista se usará para generar la factura, y que se genere con más de un producto.

También se usa el método **tiendaSeleccionada.venderProducto(productoSeleccionado);**

Este método retira el producto de los productos disponibles, para que no aparezca como si aún estuviera disponible durante la selección.
Esto significa que la lista puede cambiar con cada iteración, por ejemplo, siguiendo la imagen de ejemplo arriba, si compra la última unidad disponible de Lámpara Led, y una unidad de Marmol, para la cuarta iteración la lista se vería así:

```
Ha seleccionado el producto # 5 Nombre del producto: Marmol
Seleccione el producto que desea enviarle al cliente
0. Regresar al menu principal

1. Ladrillo: 1
2. Adhesivo: 1
3. Cemento: 1
4. Granito: 1
5. Marmol: 9
Seleccione el producto que desea enviar:
>
```

Podemos apreciar que la Lámpara LED, que antes tenía el índice 2 ya no está, y la cantidad de unidades disponibles de Mármol ha bajado de 10 a 9. Los índices de los productos han sido arreglados de acuerdo a este cambio.

Durante estas iteraciones sumamos el peso de los productos para luego filtrar el transporte que pueda cargarlo en el caso 4.

Al terminar estas n iteraciones y llenar la lista de productos del cliente., pasamos a la elección de casos 4 con la linea 145, **eleccion=4;**

Case 4: Seleccionar el tipo de transporte que se utilizará para enviar el pedido.

Primero con **TipoTransporte.crearTipoTransporteSegunCarga(PesoTotalProductos)**;
se crea una lista de transportes únicamente con los transportes que puedan sostener el peso de los productos enviados.

Con **TipoTransporte.mostrarTipoTransporteSegunCarga(listaTransFiltrada)** se muestra una lista de tipos de transporte disponibles con sus respectivos precios y se solicita al usuario que seleccione un número correspondiente al tipo de transporte deseado. Si el usuario selecciona un número inválido, se muestra un mensaje de error y se solicita que vuelva a intentar.

La lista completa de transportes que saldría en consola con un producto de poco peso se vería así:

```
Seleccione en que medio de transporte quiere enviar este producto
Junto a cada tipo de transporte se encuentra su precio.

Advertencia: Los tipos de transporte han sido filtrados de manera que solo puede seleccionar los que puedan soportar el peso de su producto.
Su pedido pesa 1 kilogramos
0. Regresar al menu principal
1. Camion $5.0
2. Avion $10.0
3. Automovil $3.0
4. Camioneta $4.0
5. Bicicleta $1.0
6. Patines $1.0
7. Barco $15.0
8. Helicoptero $22.0
9. Tren $33.0
10. Caminando $1.0
```

Siguiendo el ejemplo que llevamos, si elegimos un bloque de mármol y una Lampara LED, la lista se ve así:

```
Seleccione en que medio de transporte quiere enviar este producto
Junto a cada tipo de transporte se encuentra su precio.

Advertencia: Los tipos de transporte han sido filtrados de manera que solo puede seleccionar los que puedan soportar el peso de su producto.
Su pedido pesa 6000003 kilogramos
0. Regresar al menu principal
1. Barco $15.0

Seleccione el número del tipo de transporte:
>
```

La lista inicial de transportes tiene 10 elementos, entre ellos Bicicleta, Patines, Caminando y Automóvil, estos transportes (entre otros) no tienen la capacidad de carga para transportar un producto de 6000003 kilos, así que no se han añadido a la lista.

Para un producto de peso intermedio (43 kg), la lista puede verse así:

```
Advertencia: Los tipos de transporte han sido filtrados de manera que solo puede seleccionar los que puedan soportar el peso de su producto.
Su pedido pesa 43 kilogramos
0. Regresar al menu principal
1. Camion $5.0
2. Avion $10.0
3. Automovil $3.0
4. Camioneta $4.0
5. Barco $15.0
6. Helicoptero $22.0
7. Tren $33.0

Seleccione el número del tipo de transporte:
>
```

En cualquier caso, la lista de transportes disponibles se filtra con el peso de cada uno.

Después de este paso se preguntará si desea aplicarle envío gratis al pedido.

```
El pedido se enviará por Avion
Desea aplicar envío gratis?
 1. Si   2. No
>
```

Al seleccionar el int 1 para sí, se activa el método **Transporte.enviarGratis(transporteSeleccionado),** que cambia el precio de envío a 0.
También hay dos métodos pertinentes para este caso:
transporteSeleccionado.recordarPrecioTransporte();
El cual guarda el precio original del transporte para ser usado luego, y

transporteSeleccionado.reestablecerPrecioTrans();
El cual se llama después de terminar el envío y generar la factura, su función es restablecer el precio de envío originalmente declarado al transporte que se usó.
Ahora vamos al caso 5, el último paso.

Case 5:
Iniciamos el día en el que fue hecha la venta (solo el día, sin mes ni año) por razones de estadística y contabilidad de la empresa.

```
Digite el día del mes entre 1 y 30:
```

Aquí se valida que el día sea válido en el rango que se pida.

Finalmente, el método **tiendaSeleccionada.enviarPedido(listaProductosPedidos, transporteSeleccionado, clienteSeleccionado, dia)**, retorna un objeto de la clase Factura y lo guarda.

También imprime una factura en consola usando este mismo método y un poco de información extra.

```
**************************************
Factura generada en la tienda Hefesto Construcciones
A nombre del cliente: Diego Armando Maradona
Nombre del producto: Lámpara LED
Precio del producto: 80.0
Nombre del producto: Marmol
Precio del producto: 20345.0
Tipo de transporte: BARCO
Tarifa de envio: 0.0
Total a pagar: 20425.0
**************************************
```

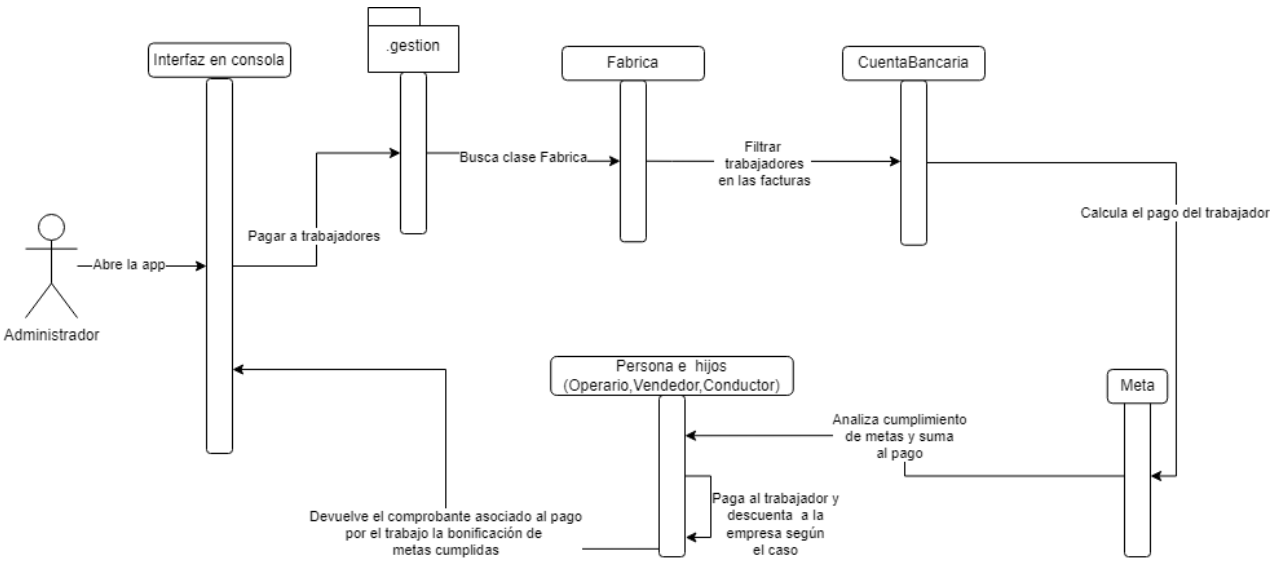
Arriba el ejemplo de una factura donde se pidieron 2 productos de la misma tienda, una Lámpara LED y un bloque de mármol, y se aplicó envío gratis.

En este punto la funcionalidad ha cumplido su propósito, ahora desplegamos un menú con únicamente dos opciones, salir o repetir la acción.

```
¿Desea hacer otro envio o volver al menu principal?

0. Volver al menu principal
1. Realizar otro  envio
> 
```

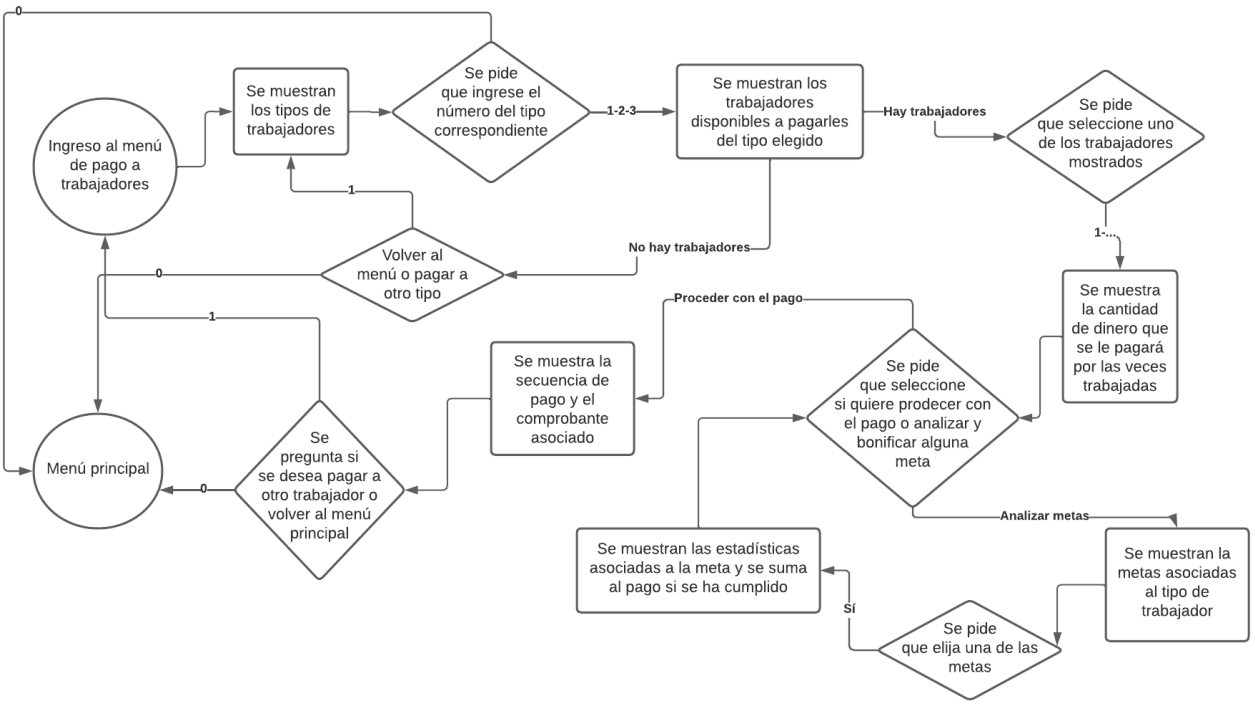
Nota: Todas las instancias se inician al inicio de la función para evitar conflictos con el alcance.

Funcionalidad 2: Pago a trabajadores



Link Funcionalidad Pago a trabajadores

Diagrama de secuencia



[Link Diagrama Funcionalidad Pago a trabajadores]

La funcionalidad pago a trabajadores consiste en menús desplegables que permiten al administrador seleccionar alguno de los trabajadores que haya trabajado (en los envíos), revisar el pago que le corresponde y dar bonificaciones por el cumplimiento de ciertas metas que fueron establecidas previamente tomando como índice los productos vendidos (para los vendedores y el operario) y el peso de los productos que transporta (para el conductor).

Objetos que intervienen en su implementación: objetos de tipo Persona, Conductor, Vendedor, Operario, Meta y Factura.

Las líneas que se presentarán a continuación son del archivo UiPagarTrabajadores.java del paquete uiMain.
Luego de escoger la opción 2 en el menú principal se despliega las opciones de los tipos de los trabajadores a los que se tienen disponibles para pagar:

```
Ingrese el número de la opción que desea utilizar
> 2

Has escogido la opción pagar a trabajadores

¿A qué tipo de trabajador desea pagarle?

1. Operarios
2. Conductores
3. Vendedores
0. Volver al menú principal
>
```

Para continuar, la clase Factura guarda el histórico de todas las facturas que se han creado con la funcionalidad enviar pedido en un arreglo estático:

```
private static ArrayList<Factura> listaFacturas
```

Esta lista, sumado al entero pedido anteriormente que se escoge dependiendo del tipo de trabajador al que se le desea pagar, son los argumentos del método **busquedaTrabajo** de la clase Fabrica. Este método busca los vendedores, conductores, u operario según sea el caso que estén en las facturas de los envíos que se han realizado,además, verifica que hayan trabajado (atributo trabajo>0) pues es posible que anteriormente ya se le haya pagado por un trabajo realizado. El método devuelve una lista de Personas:
Línea 52 **ArrayList<Persona> listaTrabajadores = Fabrica.busquedaTrabajo(Factura.getListaFacturas(), opcion);**

Esta lista de personas es pasada al método **mostrarPersonas** de la clase Fabrica, el cúal retorna un String compuesto de las descripciones de cada Persona que fue añadida a la lista, esto haciendo uso de la ligadura dinámica pues ejecuta el método toString de la clase Vendedor, Operario o Conductor según sea el caso.
Línea 73 **System.out.println(Fabrica.mostrarPersonas(listaTrabajadores));**
Así se ve por pantalla si en este caso se escogiera la opción 3 (vendedores):

```
Esta es la lista de Vendedores que han trabajado

Trabajador 1
Nombre: Lionel Andres
Edad: 22
Cedula: 14720
Tienda: Zaha Hadid Arquitectura

Trabajador 2
Nombre: Maria Beatriz
Edad: 20
Cedula: 57793
Tienda: Hefesto Construcciones

Trabajador 3
Nombre: Adriana
Edad: 21
Cedula: 89235
Tienda: Vitruvio Edificios

Por favor, ingrese el número del trabajador al cual desea pagarle [1-3]:
>
```

Se pide el número del trabajador deseado y este es utilizado para seleccionar dicho trabajador:
Línea 91 **Persona trabajadorEscogido = listaTrabajadores.get(opcTrabajador - 1);**

Se sigue con el método **calcularPago** de la clase CuentaBancaria, el cual con el salario base (constante en la clase Persona) realiza los debidos cálculos según el trabajador y las veces que ha realizado pedidos, para devolver un entero que corresponde al pago que le enviará al trabajador.
Linea 95 **int total = CuentaBancaria.calcularPago(trabajadorEscogido);**

Dicho total es mostrado en la pantalla acompañado por la cantidad de veces que se trabajó para recibir ese pago. Se imprime además, la pregunta al administrador para decidir si desea o no analizar y bonificar al trabajador por el cumplimiento de sus metas:

```
Por favor, ingrese el número del trabajador al cual desea pagarle [1-3]:
> 3

Al trabajor Adriana se le pagará 6000 por trabajar 1 veces

¿Desea analizar y pagar al trabajador por sus metas cumplidas?
1.Si
2.No

Digite el número de la opción que desee
>
```

Si se digita 2 (no), se dirige directamente al pago que será mostrado después, si digita (1), se mostrará en pantalla las metas asociadas al tipo de trabajador, de la siguiente manera:

```
Digite el número de la opción que desee
> 1

El trabajador escogido tiene las siguientes metas

Indice: número de productos vendidos

Meta 1
Nivel de dificultad: Fácil
indice requerido: 3.0
Bonificación por cumplimiento: 10000.0

Meta 2
Nivel de dificultad: Dificil
indice requerido: 10.0
Bonificación por cumplimiento: 50000.0


Seleccione una de las metas
>
```

Al escoger una meta, se verifica que dicha meta no haya sido escogida y pagada antes, y se ejecutan los siguientes métodos:
verificadorMeta, el cúal recibe el índice que el trabajador lleva hasta el momento, este índice corresponde a productos vendidos para los vendedores y operario, y al peso de los productos transportados en el caso de los conductores. Se verifica si este número es igual o ha superado el asociado a la meta (si se ha cumplido la meta) y devuelve un booleano con esta información.
Línea 165 **boolean verificadorMeta = metaEscogida.cumpleMeta(indice);**
Luego, este mismo índice entra al método **estadisticasMeta,** el cual por medio de cálculos devuelve un String con información sobre el porcentaje de cumplimiento de la meta y si no se ha cumplido, el porcentaje que lleva cumplido y el índice faltante para completarla.
Línea 166 **String estadisticasMeta = metaEscogida.porcentajesCumplidos(indice);**

Si la meta se cumple, se imprimirá lo siguiente:

```
Seleccione una de las metas
> 1

La meta ha sido cumplida exitósamente
Sumaremos al pago la bonificación por esta meta
Porcentaje de la meta cumplido: 133.33333333333334%

¿Qué desea hacer?
 1.Revisar otra meta
 2.Proceder con el pago
>
```

Además, el pago por dicha meta se añade al total calculado anteriormente:
Línea 171 **valorPorMetas += metaEscogida.getPago();**

Por otro lado, si la meta no se cumple:

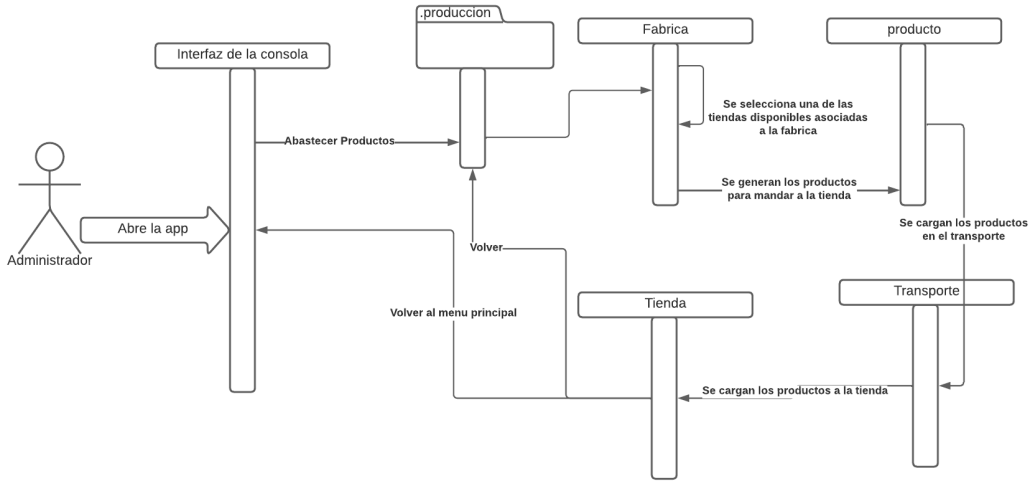
```
Seleccione una de las metas
> 2

La meta aún no ha sido cumplida
Porcentaje de la meta cumplido: 40.0%
Porcentaje faltante: 60.0%
Cantidad faltante del indice indicado: 6.0

¿Qué desea hacer?
 1.Revisar otra meta
 2.Proceder con el pago
>
```

Si se coloca 1, se pueden revisar alguna de las otras metas que aún no se hayan analizado, y si se coloca 2 al proceder con el pago, se ejecuta el método **recibirSueldo** de la clase Persona, que está sobreescrito en cada una de las clases que la heredan (Vendedores, Conductores y Operario), aquí se pasa un entero como argumento que hace referencia al pago que se hará, se descuenta cuenta bancaria de la Fábrica o la Tienda, y se suman a la cuenta del trabajador
Línea 211 **trabajadorEscogido.recibirSueldo(total + valorPorMetas);**

Al momento del pago se muestra en el pantalla el comprobante de pago, y por último una pregunta para que el administrador decida si quiere salir al menú principal o pagar a otro trabajador concluyendo así la funcionalidad:

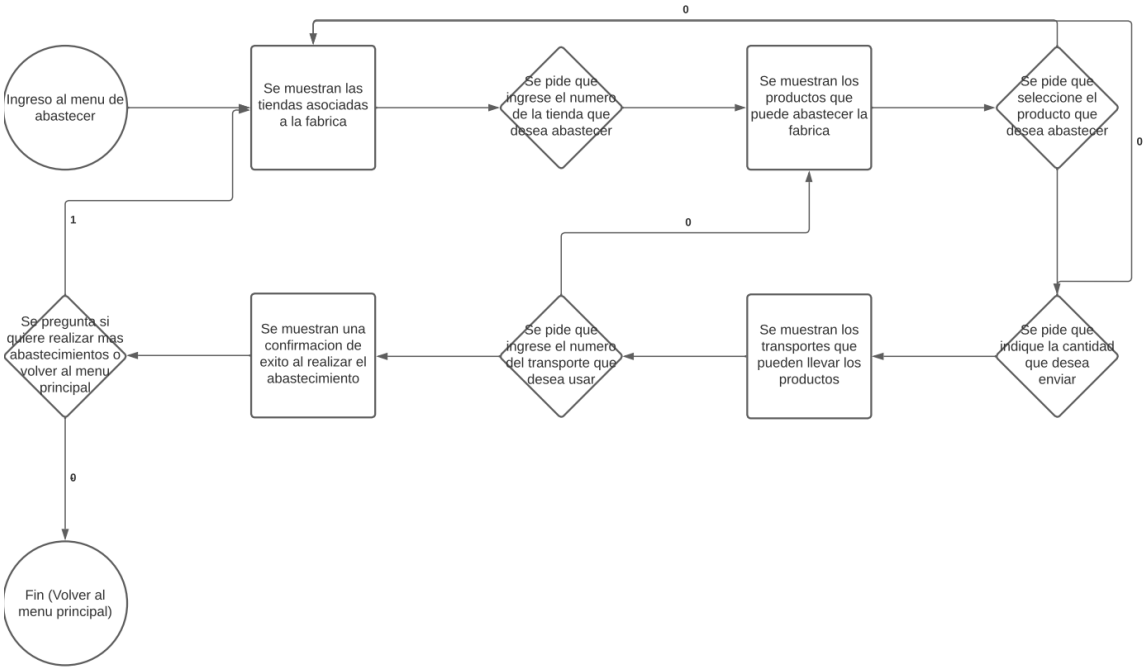
```
> 2
... Realizando pago ... Por favor espere ...

El pago fue realizado con éxito

*************************
Comprobante
Pago asociado a los envios realizados 6000
Pago asociado al cumpliento de metas 10000
Total = 16000
*************************

¿Qué desea hacer?
1. Pagar a otro trabajador
0.Volver al menu principal
>
```

Funcionalidad 3: Abastecimiento de tiendas



[Link Funcionalidad Abastecimiento]

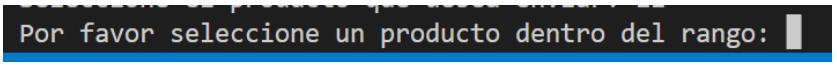
Diagrama de secuencia



[Link Diagrama Funcionalidad Abastecer]

La funcionalidad abastecimiento de tiendas en Java consiste en menús desplegables que permiten al administrador seleccionar una tienda a la cual enviarle una cantidad N de un producto desde la fábrica mientras este no supere la cantidad de productos que la tienda puede tener por categoría.

Cada parte del menú tiene una restricción en caso de que se salga de las posibilidades establecidas por este, un ejemplo es este mensaje:



En el cual deberá introducir un valor que esté dentro del rango de los posibles productos y de manera similar para los otros menús.
El menú está hecho a base de un switch y un ciclo condicional el cual nos permitirá entrar en el apartado adecuado del menú de manera secuencial para evitar problemas.

Apartados del switch:

Case 1:
En este caso lo que hacemos es seleccionar la tienda la cual se va a abastecer, para ello se muestra una lista la cual tiene todas las tiendas disponibles con cada tienda acompañada con los productos que tiene dentro de sí en ese momento esto se hace mediante un método de la clase fábrica en la línea 82 llamado **mostrarTiendas()**(funciona iterando sobre una lista que tiene las tiendas y guardando los nombres en un string el cual se imprime desde el main) y se le solicita al administrador que seleccione la tienda la cual quiere abastecer esto se hace llamando la lista de tiendas almacenada en la fabrica **fabrica.getListaTienda().size()** esto determinara el rango de las tiendas y se seleccionará con **fabrica.getListaTienda().get(x - 1)** que quedará guardado en el objeto **tiendaSeleccionada** y aquí ya pasa al caso 2.
Ejemplo:

```
Abastecer tiendas - Apartado de tiendas

0. Volver al menu anterior

1. Hefesto Construcciones-PRODUCTOS:
     Ladrillo: 1
     LimpiaMax: 1
     Sandwich de pollo: 1

2. Vitruvio Edificios-PRODUCTOS:
     Hojas de Te: 1
     Adhesivo: 1
     Lavadora: 1

3. Zaha Hadid Arquitectura -PRODUCTOS:
     Marmol: 1
     Carne de res: 1
     Jabon: 1

Seleccione la tienda a la que desea enviar: 2
```

Case 2:
En este caso se selecciona el producto que se va a mandar a la tienda esto se hace de manera similar al anterior se traen todos productos que pueden ser producidos en la fabrica a partir de una lista que tiene la cual se le pasa en el constructor de la fábrica la cual contiene todos los productos que puede abastecer la lista se muestra mediante el método **mostrarProductos()**(funciona iterando sobre una lista que tiene los productos y guardando los nombres en un string el cual se imprime desde el main) de la clase fábrica en la línea 57 además de que también se muestran los límites de categoría que tiene cada tienda esto se hace con el método de la clase tienda en la línea 129 **productosPorCategoria()**(funciona iterando sobre la lista de productos que hay en la tienda con los cuales se va contando la cantidad de productos que hay por categoría mediante un diccionario) y ya continua en el siguiente case.

```
Abastecer tiendas - Apartado de productos

La capacidad de productos para esta tienda es la siguiente:
Aseo 1/15  Consumible 1/18  Construccion 1/29

0. Regresar al menu anterior

INDICE-PRODUCTO-PESO-PRECIO-CATEGORIA
1. LimpiaMax - 50.0 - 30.0 - aseo
2. Sandwich de pollo - 1.0 - 1000.0 - consumible
3. Ladrillo - 2.5 - 0.25 - construccion
4. Lavadora - 675.0 - 20.0 - aseo
5. Hojas de Te - 60.0 - 15.0 - consumible
6. Adhesivo - 0.3 - 5.0 - construccion
7. Jabon - 271.0 - 10.0 - aseo
8. Carne de res - 265.0 - 1000.0 - consumible
9. Marmol - 3000000.0 - 1000.0 - construccion
10. Granito - 27800.0 - 750.0 - construccion
```

Case 3:
Aquí se selecciona la cantidad de productos que se van a enviar a la tienda, mediante el diccionario que cuenta la cantidad que hay por categoría y haciendo una resta podemos evitar que se envíen productos en una cantidad mayor a lo que soporta la tienda por categoría (máxima - cantidad en la tienda) permite saber cuántos productos puede tener la tienda en esa categoría se usando estas dos cantidades **tiendaSeleccionada.getProductosPorCategoria().get(productoSeleccionado.getCategoria())** y **tiendaSeleccionada.getCantidadPorCategoria().get(productoSeleccionado.getCategoria())**.

```
Escriba la cantidad de productos que desea abastecer: 50
Por favor seleccione una cantidad en el limite de la tienda por categoria:
```

Case 4:
Aquí se realiza la selección del transporte para abastecer los productos en las tiendas funciona de igual manera a como lo hace la en el case 4 en la funcionalidad enviar pedido filtrando los medio de transporte posibles y presentando la lista de transportes disponibles en base al peso de los productos que vamos a llevar la función se realiza con los mismos métodos por lo que no se pondran aqui ya se pasa al case 5.

```
Seleccione en que medio de transporte quiere enviar este producto

Advertencia: Los tipos de transporte han sido filtrados de manera que solo puede seleccionar los que puedan soportar el peso de su producto.
0. Regresar al menu principal
1. Camion
2. Avion
3. Automovil
4. Camioneta
5. Barco
6. Helicoptero
7. Tren

Seleccione el número del tipo de transporte:
>
```

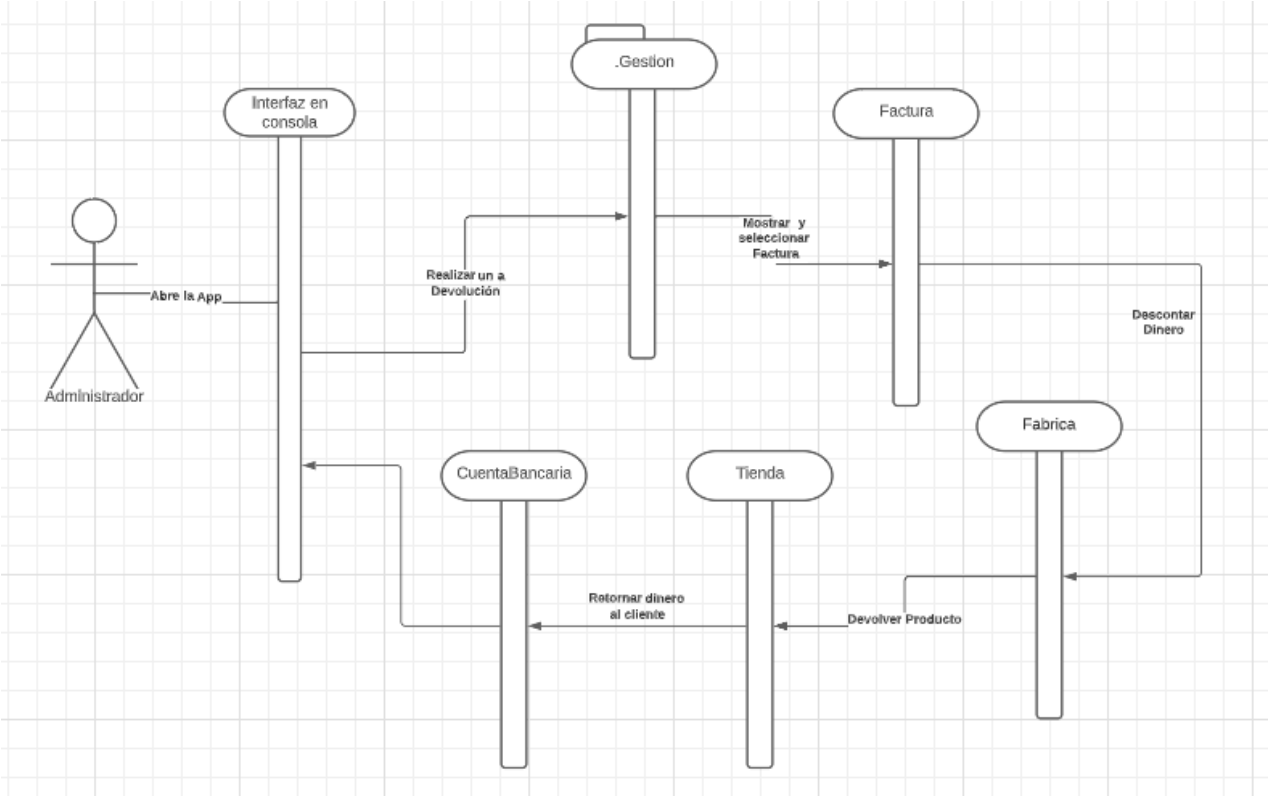
Case 5:
Esta es la parte más importante de la funcionalidad ya que aquí es donde se generan los productos para ser cargados en el transporte estos se meten en una lista todo esto con el metodo **cantidadProductos()**(funciona mediante un ciclo el cual va agregando productos a la lista) de la clase tienda en la línea 93 entonces se cargan los productos en el transporte seleccionado esto se hace con el método **abastecerProducto(Tienda tiendaSeleccionada, ArrayList<Producto> productoSeleccionado)**(Este método carga los productos en el transporte y asigna a la tienda a la que va a mandar los productos) de la clase transporte en la línea 56 ya luego se aplica el método **descargarProducto(Transporte transporteSeleccionado)**(Este método envía los productos del transporte a la tienda) de la clase tienda en la línea 335 el cual saca los productos de la lista del transporte y los mete a la tienda.

```
Ha seleccionado el transporte #3
La tienda se abastecera por: Automovil
El producto fue enviado con exito ahora la tienda tiene
PRODUCTOS:
     LimpiaMax: 2
     Adhesivo: 1
     Hojas de Te: 1
     Lavadora: 1

0.Volver al menu principal
1. Realizar más abastecimientos
>
```

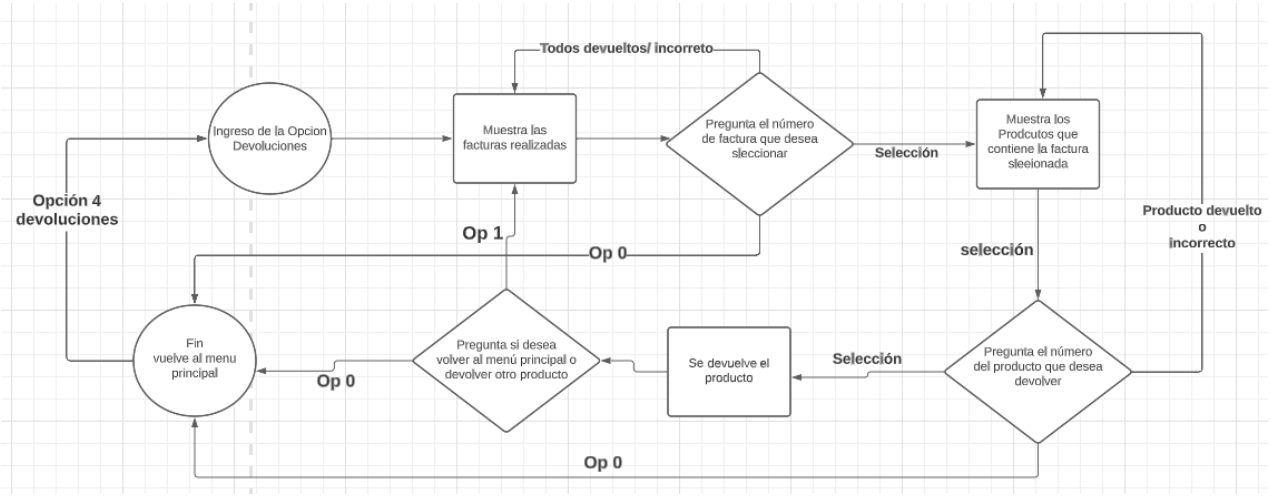
En el mismo case 5 se decide si se vuelve a realizar otro abastecimiento o si se desea volver al menú principal.

Funcionalidad 4: Devoluciones de pedidos

Diagrama de secuencia



Link Funcionalidad Devoluciones



Link Diagrama Funcionalidad Devoluciones

Esta funcionalidad consiste en realizar la devolución de uno o varios productos (1 a la vez) del que se haya realizado algún envío erróneo, a medida de que se van pidiendo las opciones se validan de que estén dentro del rango disponible, en cualquier momento puede volver al menú principal.

luego de escoger la opción 4 en el menú principal vemos el siguiente listado que son las facturas disponibles

```
Por favor seleccione el número que le corresponda
a la factura para realizar la devolucion

0. Volver al menu principal
1. ID: 1 Cliente: Luis Gómez
2. ID: 2 Cliente: María García
3. ID: 3 Cliente: Juan Pérez
4. ID: 4 Cliente: Luis Gómez
5. ID: 5 Cliente: Pedro Gómez
6. ID: 6 Cliente: Josesito Gómez
7. ID: 7 Cliente: Juan Pérez
8. ID: 8 Cliente: Julián Araña Alvarez
9. ID: 9 Cliente: Emiliano Dibu Martinez
10. ID: 10 Cliente: Diego Armando Maradona

Digite su opcion:
> 
```

:

este menú se despliega gracias al método **mostrarFacturas()**(clase Factura línea 368) que devuelve una cadena de texto que contiene el ID de la factura junto con el nombre del cliente que luego se imprime en pantalla, seguido de esto un escáner lee la opción que el Administrador digité por teclado.

se le muestra la opción que digitó al administrador en pantalla y este entero se le pasa como parámetro al método **seleccionarFactura(int opcion)** (clase Factura linea 390) el método retorna un objeto de tipo factura que es la seleccionada por el administrador.
ese objeto de tipo factura se accede y se tiene la lista de productos que contiene, y esa lista de Objetos de tipo producto se le pasa como parametro al metodo **mostrarProductosFacturas(ArrayList<Producto>)**(clase Factura linea 408) que devuelve un string con todos los productos que tiene la factura escogida y si algún producto ya fué devuelto se hace la aclaración a un lado, acompañado de un índice para escoger la opción necesaria, esto es impreso en pantalla y es lo que observa el administrador:

```
Seleccionó la factura con la opcion número: 3

Por favor seleccione el número que le corresponda
a el producto para realizar la devolucion

0. Volver al menu principal
1. Producto: LimpiaMax
2. Producto: Sandwich de pollo
3. Producto: Ladrillo
4. Producto: Lavadora
5. Producto: Hojas de Te (Devuelto)
6. Producto: Adhesivo
7. Producto: Jabon
8. Producto: Carne de res
9. Producto: Marmol
10. Producto: Granito
11. Producto: Jamon
```

de nuevo un escáner pide la opción del producto que desea devolver de esa factura, y se le pasa como parametro al metodo **seleccionarProductoDevolver(int opcion)**(clase Factura linea 438) este método

devuelve un objeto de tipo Producto y se muestra la opción del producto que eligió devolver el administrador.

el objeto de tipo Producto devuelto en el método anterior se le pasa como parámetro al método **descontarDineroCuentaAdmin(Producto producto)**(clase Fábrica linea 133) este método devuelve un double que es el valor total del producto devuelto.

del objeto Factura que seleccionó el administrador se ve la tienda que contiene y allí se aplica el método **devolverProducto(Factura facturaSeleccionada, Producto Seleccionado)**(clase Tienda linea 253) recibe 2 parámetros, primero la factura seleccionada por el administrador y segundo el producto seleccionado por el administrador de esa factura, regresa el producto a una lista de la tienda donde se almacenan los productos devueltos, por último este método devuelve un objeto cliente que se extrae de la factura pasada como parámetro.

del cliente que retorna el método anterior se extrae la cuenta bancaria para hacerle la devolución se le pasa como parámetro el double total que es el valor del producto y el mismo objeto cliente, **devolverDinero()**(clase CuentaBancaria línea 58) no devuelve nada (void)

y por último se remueve el producto devuelto del atributo lista de productos del cliente.

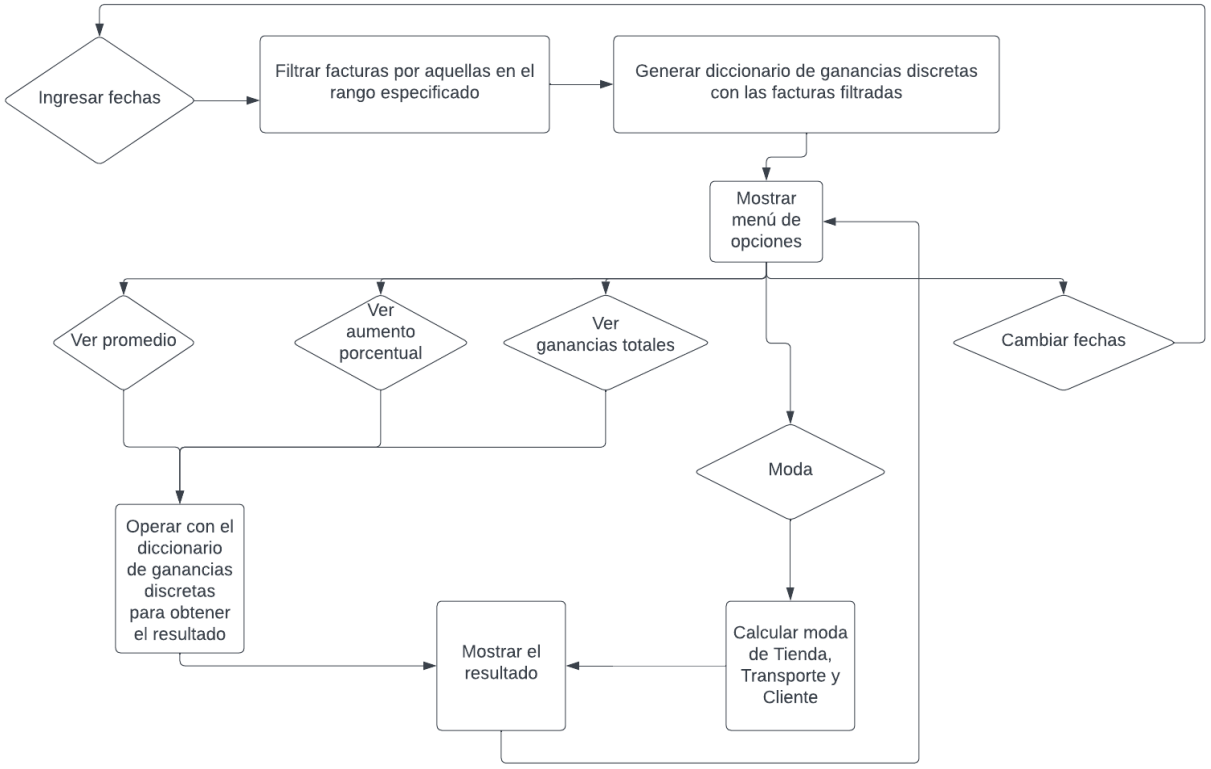
```
Digite su opcion:
> 7

Seleccionó el producto con la opcion número: 7
... Realizando devolución ... Por favor espere ...
¡¡ El producto ha sido devuelto exitosamente !!
```

Una vez termina este ciclo se vuelve a preguntar si desea devolver otro producto o volver al menú principal en caso de que vaya a devolver otro producto se repite todo el proceso las veces necesarias.

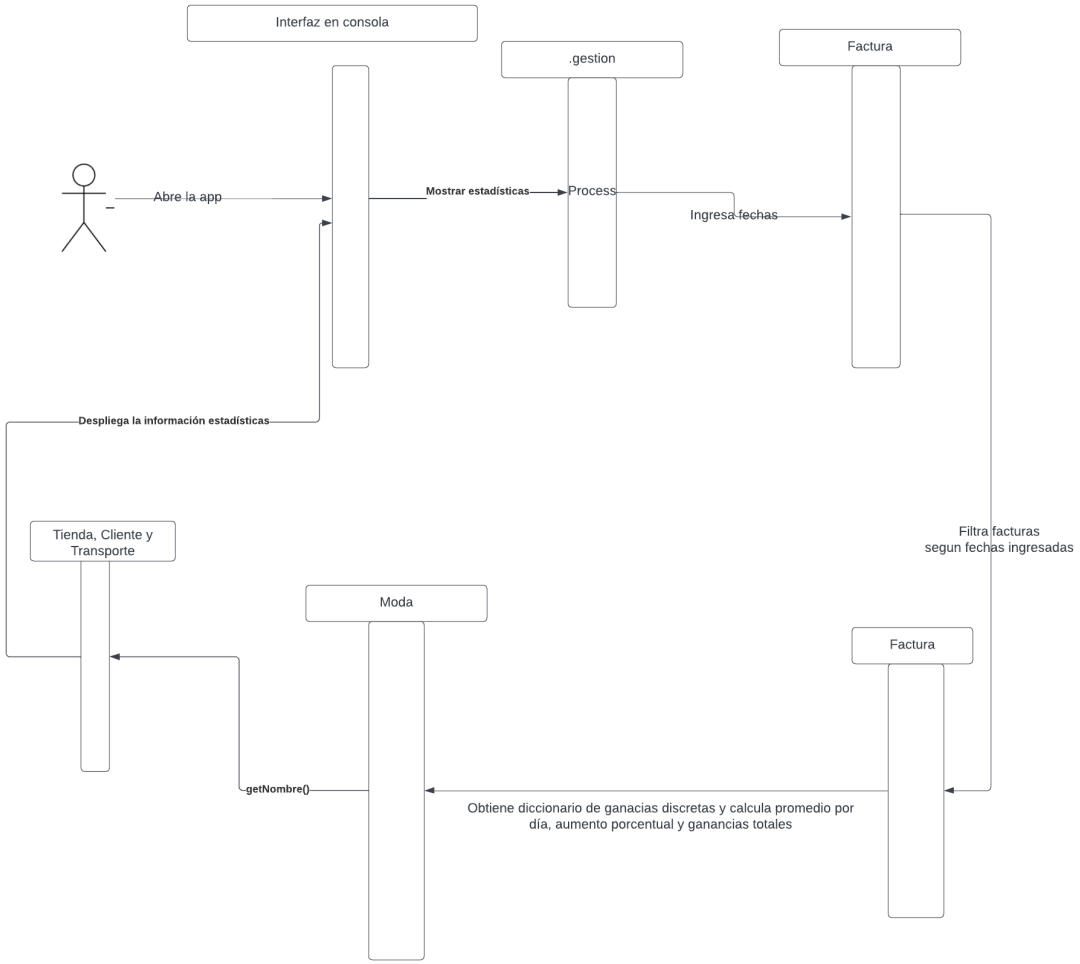
```
¿Desea hacer otra devolucion o volver al menu principal?

0. Volver al menu principal
1. Realizar otra devolución
>
```

Funcionalidad 5: Muestra de estadísticas

Diagrama de secuencia
[Link diagrama de secuencia]



[Link funcionalidad estadística]



La funcionalidad muestra de estadísticas consiste en el tratamiento de la información que se va almacenando a medida que se hacen pedidos. En cada paso se valida que los números ingresados si son opciones válidas. Cada menú se invoca con objetos de la clase Menu, la cual “activa” el proceso del menú mediante el método mostrar(). La funcionalidad consta de dos partes:

La primera parte consiste en la información numérica sobre las ventas. La clase factura guarda el histórico de todas las facturas que se han creado con la funcionalidad enviar pedido en un arreglo estático.

```
private static ArrayList<Factura> listaFacturas
```

Luego, toda la información de las ventas de la empresa se encuentra representada en este arreglo. Además, como cada factura tiene un atributo fecha, esto permite inducir un orden sobre este histórico. Lo primero que se hace es especificar el rango de fechas sobre el cual se quiere analizar la información, en la clase UiEstadistica, mediante el método ingresar fechas.

```java
public static int[] ingresarFechas(){

    Scanner sc = new Scanner(System.in);

    int opcion = new Menu(enunciado:"REPORTE", new String[]{"Analizar toda la información",
                                                            "Ingresar fechas específicas"}).mostrar();

    int fechaMin = Factura.getFechaMin();
    int fechaMax = Factura.getFechaMax();

    while(opcion != 0){

    switch(opcion){

        case 1:

            return new int[]{fechaMin, fechaMax};

        case 2:

            System.out.println("\nLa fecha mínima es " + fechaMin + " y la fecha máxima es " + fechaMax);

            System.out.println(x:"Ingrese fecha de inicio: ");
            int fecha1 = Menu.ingresarConLimites(fechaMin, fechaMax);

            System.out.println(x:"Ingrese fecha final: ");
            int fecha2 = Menu.ingresarConLimites(fecha1, fechaMax);
```

Este método retorna las fechas en las cuales se va a analizar la información, y se usa en el método estadística:

```java
public static void estadistica(){

    int[] fechas = ingresarFechas();

    analisis(fechas[0], fechas[1]);


}
```

Es en el llamado a el método análisis que ocurre el “grueso” de la funcionalidad. Con las fechas ingresadas, se obtienen las ganancias discretas de la siguiente manera.

```java
HashMap<Integer, Double> disc =  Factura.gananciasDiscretas(fecha1, fecha2);
```

Lo que se hace después es crear un diccionario cuyas keys son las fechas existentes. Luego, se recorren las fechas y se suman las ganancias de cada factura a la fecha correspondiente en el diccionario. Por último este diccionario es retornado, y es precisamente este el que tiene las ganancias discretas. En código:

```java
public static HashMap<Integer, Double> gananciasDiscretas(int fecha1, int fecha2){

    ArrayList<Integer> fechas = getListaFechas(fecha1, fecha2);

    ArrayList<Factura> facturas = getFacturasEntreFechas(fecha1, fecha2);

    HashMap<Integer, Double> dictGananciasDiscretas = new HashMap<Integer, Double>();

    for(int fecha: fechas)
        dictGananciasDiscretas.put(fecha, value:0.0);

    for(int fecha: dictGananciasDiscretas.keySet()){

        for(Factura factura: facturas){

            if(factura.fecha == fecha){

                double valorAnterior = dictGananciasDiscretas.get(fecha);

                dictGananciasDiscretas.put(fecha, valorAnterior + factura.getTotal());
            }
        }
    }

    return dictGananciasDiscretas;
}
```

Mediante este dato se alimenta a los demás métodos, los cuales procesan la información en el diccionario de ganancias discretas y la transforman para retornar la demás información estadística, así:

```java
case "GANANCIAS DISCRETAS":

        desplegarInfo(disc, strData:"GANANCIA", posfijo:"");

    break;
case "GANANCIAS TOTALES":

        System.out.println("\nLas ganancias totales entre las fechas ingresadas han sido: " + Factura.gananciasTotales(disc));

    break;
case "PROMEDIO POR DIA":

        System.out.println("\nEl promedio por día es: " + Factura.promedioPorDia(disc));

    break;
case "AUMENTO PORCENTUAL":

        desplegarInfo(Factura.aumentoPorcentual(disc), strData:"AUMENTO", posfijo:"%");

    break;
```

El método desplegar info permite desplegar la tabla de valores.

La segunda parte de la funcionalidad consiste en la obtención de las modas estadísticas de las tiendas, clientes y transportes, i.e qué tienda ha vendido más (en términos de ventas realizadas, no de ganancias), a cual cliente se le ha vendido más y cuál ha sido el transporte más usado. Para esto se tiene la interfaz

Moda. Esta interfaz sólo tiene un método, getNombre(). El propósito de esta interfaz es la de marcar objetos que tienen la propiedad de que se les puede contar y asignar una moda. Luego, se utiliza el hashmap de la clase factura:

```
private static HashMap<String, Moda> infoAtributos = new HashMap<String,Moda>();
```

El contenido de este HashMap son tuplas de dos. La primera entrada tiene el nombre del atributo y en la segunda un objeto que implemente la interfaz moda. Cada vez que sea crea una nueva factura, se le ingresan datos a este HashMap así:

```
infoAtributos.put(key:"tienda", tienda);
infoAtributos.put(key:"transporte", transporte);
infoAtributos.put(key:"cliente", cliente);
```

Finalmente, lo que se hace es que para obtener la moda de estos atributos, se usa el método masComun de la clase Factura, en el retorno del siguiente método:

```
public static Moda moda(int fecha1, int fecha2, String atributo){

    ArrayList<Factura> facturas = Factura.getFacturasEntreFechas(fecha1, fecha2);

    ArrayList<Moda> objetos = new ArrayList<Moda>();

    for(Factura factura: facturas){

        objetos.add(factura.getAtributos().get(atributo));
    }

    return Factura.masComun(objetos);
}
```

El cual es el que precisamente se llama en la clase de la funcionalidad:

```
case 1:

    System.out.println("La tienda más usada ha sido "
    + Factura.moda(fecha1, fecha2, atributo:"tienda").getNombre());

    break;

case 2:

    System.out.println("El transporte más usado ha sido "
    + Factura.moda(fecha1, fecha2, atributo:"transporte").getNombre());

    break;

case 3:

    System.out.println("El cliente al que más se le ha vendido ha sido "
    + Factura.moda(fecha1, fecha2, atributo:"cliente").getNombre());
```

V. MANUAL DE USUARIO

Objetos de prueba
Se tienen los siguiente objetos serializados:

- **Fábrica**
  - fabrica
    - Catálogo: Lista de productos (producto1,producto2,...,producto22)
    - Tiendas: Lista de tiendas (tienda1,tienda2,tienda3)
    - CuentaBancaria: cuentaEmpresa
    - Operario: operario1

- **Tiendas**
  - tienda1
    - Nombre: "Hefesto Construcciones"
    - Vendedor: vendedor1
    - CuentaBancaria: cuentaEmpresa

  - tienda2
    - Nombre: "Consumibles de la Abuela Tata"
    - Vendedor: vendedor2
    - CuentaBancaria: cuentaEmpresa

  - tienda3
    - Nombre: "Miss Músculo Aseo"
    - Vendedor: vendedor3
    - CuentaBancaria: cuentaEmpresa

- **Productos**
  - producto3
    - Nombre: "Adhesivo"
    - Descripción: "Adhesivo multiusos"
    - Valor: 20
    - Peso: 0.3
    - Tamaño: 20000
    - Costo de Producción: 5
    - Categoria: "construccion”

Se tienen 22 productos diferentes, creados de la misma manera que el anterior:

```
//Producto(String nombre, String descripcion, Double valor, Double peso, Double tamano, Double costoDeProduccion);
Producto producto1 = new Producto(nombre:"Ladrillo",descripcion:" Bloque utilizado en paredes y estructuras", valor:1, peso:2.5, tamano:10374,costoDeProduccion:0.25,categ
Producto producto2 = new Producto(nombre:"Lámpara LED", descripcion:"Bombilla LED de bajo consumo energético", valor:80, peso:0.2, tamano:500, costoDeProduccion:50, categ
Producto producto3 = new Producto(nombre:"Adhesivo",descripcion:"Adhesivo multiusos", valor:20, peso:0.3, tamano:20000,costoDeProduccion:5,categoria:"construccion");
Producto producto4 = new Producto(nombre:"Cemento", descripcion:"Material utilizado para la construcción", valor:200, peso:40, tamano:8000, costoDeProduccion:10, categori
Producto producto5 = new Producto(nombre:"Marmol",descripcion:"Roca metamórfica utilizada para la decoración y acabado de interiores", valor:20345, peso:3000000, tamano:2
Producto producto6 = new Producto(nombre:"Pintura blanca", descripcion:"Pintura acrílica de color blanco", valor:50, peso:1.5, tamano:10000, costoDeProduccion:30, categor
Producto producto7 = new Producto(nombre:"Granito",descripcion:"Roca ígnea utilizada en encimeras, pisos y paredes", valor:10000, peso:27800, tamano:20,costoDeProduccion:

Producto producto8 = new Producto(nombre:"Sandwich de pollo",descripcion:"Sandwich de pollo", valor:8, peso:0.2, tamano:1,costoDeProduccion:3,categoria:"consumible");
Producto producto9 = new Producto(nombre:"Hojas de Te",descripcion:"hojas de té verde", valor:6, peso:0.02, tamano:1,costoDeProduccion:15,categoria:"consumible");
Producto producto10 = new Producto(nombre:"Carne de res",descripcion:"Carne de res de vacas del llano", valor:15, peso:1, tamano:1,costoDeProduccion:7,categoria:"consumib
Producto producto11 = new Producto(nombre:"Jamon",descripcion:"Jamón de cerdo", valor:9, peso:0.03, tamano:1,costoDeProduccion:4,categoria:"consumible");
Producto producto12 = new Producto(nombre:"Carne de cerdo",descripcion:"Cerdo de criadero", valor:12, peso:1, tamano:1,costoDeProduccion:3,categoria:"consumible");
Producto producto13 = new Producto(nombre:"Galletitas",descripcion:"Galletitas hechas con amor", valor:5, peso:0.006, tamano:1,costoDeProduccion:2,categoria:"consumible")
Producto producto14 = new Producto(nombre:"Pastel de cumpleaños",descripcion:"Pastel de cumpleaños de vainilla", valor:20, peso:2, tamano:2,costoDeProduccion:7,categoria:

Producto producto15 = new Producto(nombre:"LimpiaMax",descripcion:"Limpiador multiusos", valor:8, peso:1, tamano:1,costoDeProduccion:2,categoria:"aseo");
Producto producto16 = new Producto(nombre:"Lavadora",descripcion:"Máquina para lavar ropa", valor:800, peso:70, tamano:1,costoDeProduccion:200,categoria:"aseo");
Producto producto17 = new Producto(nombre:"Jabon",descripcion:"Producto de limpieza", valor:5, peso:1, tamano:1,costoDeProduccion:1,categoria:"aseo");
Producto producto18 = new Producto(nombre:"Papel higiénico", descripcion:"Rollos de papel suave y absorbente", valor:8, peso:1, tamano:1, costoDeProduccion:2, categoria:"
Producto producto19 = new Producto(nombre:"Detergente", descripcion:"Detergente líquido para lavadora", valor:15, peso:1.2, tamano:1, costoDeProduccion:2, categoria:"aseo
Producto producto20 = new Producto(nombre:"Cepillo de dientes", descripcion:"Cepillo de cerdas suaves para la higiene bucal", valor:4, peso:1, tamano:1, costoDeProduccion
Producto producto21 = new Producto(nombre:"Bolsas de basura", descripcion:"Bolsas resistentes para desechar la basura", valor:3, peso:0.5, tamano:1, costoDeProduccion:1,
Producto producto22 = new Producto(nombre:"Jabón líquido", descripcion:"Jabón líquido para manos", valor:5, peso:3, tamano:1, costoDeProduccion:1, categoria:"aseo");
```

- **Cuentas Bancarias**
    - cuentaEmpresa
        - Número Cuenta: 9999999
        - Saldo: 1000000000
    - cuentaOperario1
        - Número Cuenta: 55555
        - Saldo: 100000
    - cuentaVendedor1
        - Número Cuenta: 56932
        - Saldo: 100
    - cuentaVendedor2
        - Número Cuenta: 45728
        - Saldo: 200
    - cuentaVendedor3
        - Número Cuenta: 95687
        - Saldo: 200
    - cuentaCliente1
        - Número Cuenta: 11111
        - Saldo: 5000000
    - cuentaCliente2
        - Número Cuenta: 22222
        - Saldo: 200
    - cuentaCliente3
        - Número Cuenta: 33333
        - Saldo: 100
    - cuentaCliente4
        - Número Cuenta: 44444
        - Saldo: 20000
    - cuentaCliente5

        - Número Cuenta: 55555
        - Saldo: 20000

    - cuentaCliente6
        - Número Cuenta: 66666
        - Saldo: 30000

    - cuentaCliente7
        - Número Cuenta: 77777
        - Saldo: 70000

    - cuentaCliente8
        - Número Cuenta: 88888
        - Saldo: 211000

    - cuentaConductor1
        - Número Cuenta: 646541231
        - Saldo: 100

    - cuentaConductor2
        - Número Cuenta: 13213544
        - Saldo: 100

    - cuentaConductor3
        - Número Cuenta: 1321354
        - Saldo: 100

- **Clientes**
    - cliente1
        - Nombre: "Juan Pérez"
        - Dirección: "Calle 123"
        - CuentaBancaria: cuentaCliente1

    - cliente2
        - Nombre: "María García"
        - Dirección: "Avenida 456"
        - CuentaBancaria: cuentaCliente2

    - cliente3
        - Nombre: "Pedro Gómez"
        - Dirección: "Carrera 789"
        - CuentaBancaria: cuentaCliente3

    - cliente4
        - Nombre: "Luis Gómez"
        - Dirección: "Carrera 789"
        - CuentaBancaria: cuentaCliente4

    - cliente5
        - Nombre: "Josesito Gómez"
        - Dirección: "Carrera 789"
        - CuentaBancaria: cuentaCliente5

- cliente6
  - Nombre: "Julián Araña Alvarez"
  - Dirección: "Carrera 101"
  - CuentaBancaria: cuentaCliente6

- cliente7
  - Nombre: "Emiliano Dibu Martinez"
  - Dirección: "Carrera 23"
  - CuentaBancaria: cuentaCliente7

- cliente8
  - Nombre: "Diego Armando Maradona"
  - Dirección: "Carrera 10"
  - CuentaBancaria: cuentaCliente8

- **Operario**
  - operario1
    - Nombre: "Jaime"
    - Edad: 20
    - Cédula: 97890
    - CuentaBancaria: cuentaOperario1
    - Fabrica: fabrica

- **Vendedores**
  - vendedor1
    - Nombre: "Maria Beatriz"
    - Edad: 20
    - Cédula: 57793
    - CuentaBancaria: cuentaVendedor1
    - Tienda: tienda1

  - vendedor2
    - Nombre: "Adriana Alexia Putellas"
    - Edad: 21
    - Cédula: 89235
    - CuentaBancaria: cuentaVendedor2
    - Tienda: tienda2

  - vendedor3
    - Nombre: "Lionel Andres Messi"
    - Edad: 22
    - Cédula: 14720
    - CuentaBancaria: cuentaVendedor3
    - Tienda: tienda3

- **Conductores**

  - conductor1
    - Nombre: "Pablo Ramirez"
    - Edad: 45
    - Cédula: 544764513
    - CuentaBancaria: cuentaConductor1

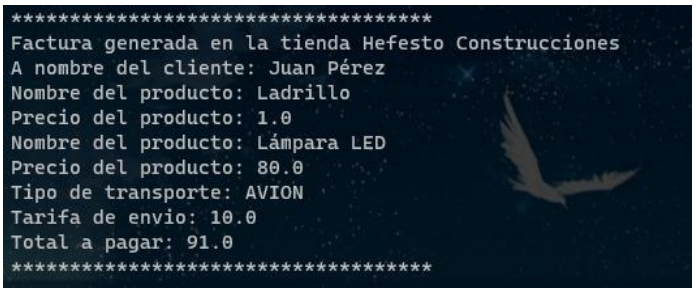
- conductor2
  - Nombre: "Hernando Cruz"
  - Edad: 50
  - Cédula: 645541321
  - CuentaBancaria: cuentaConductor2

- conducto3
  - Nombre: "Mario Casas"
  - Edad: 25
  - Cédula: 13216531
  - CuentaBancaria: cuentaConductor3

- **Facturas (De los envíos realizados)**

Se realizaron varios envíos para la prueba de las funcionalidades, algunos de ellos son:

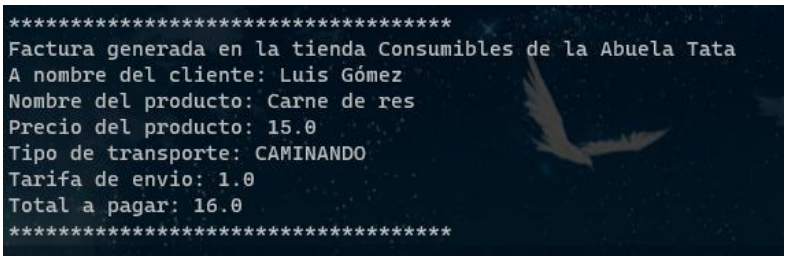
- Primer envio
  - Tienda: tienda1
  - Cliente: cliente1
  - Transporte: Avion
  - Lista de productos: [producto1,producto2]



- Segundo envío
  - Tienda: tienda2
  - Cliente: cliente2
  - Transporte: Automovil
  - Lista de productos:[producto8,producto9]

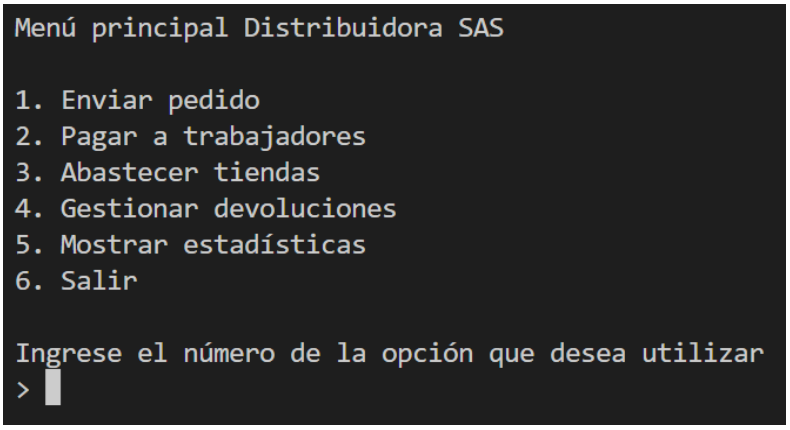
- Tercer envío
  - Tienda: tienda3
  - Cliente: cliente 3
  - Transporte: Barco
  - Lista de productos:[producto15,producto16,producto17,producto18]

- Cuarto envío
  - Tienda: tienda2
  - Cliente: cliente4
  - Transporte: Caminando
  - Lista de productos:[producto10]



Vista de funcionalidades

En el menú principal se despliega el siguiente mensaje:



Escoja una de las opciones disponibles para acceder a cada una de las funcionalidades.

- Funcionalidad 1: Envío de pedidos

Al seleccionar la opción 1 en el menú principal se despliega una lista con los clientes disponibles y sus direcciones.

```
Ingrese el número de la opción que desea utilizar
> 1

0. Volver al menu anterior

Seleccione el cliente al que desea enviar:
1. Juan Pérez Dir. Calle 123
2. María García Dir. Avenida 456
3. Pedro Gómez Dir. Carrera 789
4. Luis Gómez Dir. Carrera 789
5. Josesito Gómez Dir. Carrera 789
6. Julián Araña Alvarez Dir. Carrera 101
7. Emiliano Dibu Martinez Dir. Carrera 23
8. Diego Armando Maradona Dir. Carrera 10

>
```

Seleccione el índice del cliente que desee, para este ejemplo seleccionaremos al número 1.
Para seleccionar a este escriba en la terminal “1” y presione enter

```
> 8
Has seleccionado al cliente #8
El cliente es: Diego Armando Maradona
```

La siguiente lista desplegable que vemos en la terminal contiene tiendas y sus productos disponibles.

```
Su pedido se enviará desde alguna de estas tiendas, por favor seleccione una:
0. Volver al menu principal
1. Hefesto Construcciones-PRODUCTOS:
     Adhesivo: 1
     Pintura blanca: 1
     Granito: 1
     Lámpara LED: 1
     Cemento: 1
     Marmol: 1
     Ladrillo: 1

2. Consumibles de la Abuela Tata-PRODUCTOS:
     Sandwich de pollo: 1
     Hojas de Te: 1
     Jamon: 1
     Carne de cerdo: 1
     Carne de res: 1
     Galletitas: 1
     Pastel de cumpleaños: 1

3. Miss Músculo Aseo-PRODUCTOS:
     Jabon: 1
     Jabón líquido: 1
     LimpiaMax: 1
     Lavadora: 1
     Papel higiénico: 1
     Detergente: 1
     Bolsas de basura: 1
     Cepillo de dientes: 1

Seleccione la tienda desde la que desea enviar:
>
```

Escriba el índice de la tienda que desea seleccionar y presione enter, para este ejemplo seleccionamos la tienda #2 con un int 2.

*Los nombres de las tiendas son ficticios y están sujetos a cambios

Ahora, seleccione cuantos productos de esta tienda quiere enviar, el máximo son 5 productos.
Para este ejemplo pediremos 2 productos. Presionamos 3 y enter.

```
Has seleccionado la tienda: 2
¿Cuantos productos deseas comprar de esta tienda?
 Máximo 5 productos por cliente
2
```

Ahora aparecerán los productos disponibles de esta tienda. Puede seleccionar de a un producto escribiendo el índice del producto y enter, se le repetirá la pregunta una cantidad de veces igual al número de productos que desee comprar. La cantidad de productos disponibles se actualizará automáticamente con cada selección.

En este ejemplo, la pregunta se repetirá dos veces, la primera iteración se ve así:

```
Seleccione el producto que desea enviarle al cliente
0. Regresar al menu principal

1. Sandwich de pollo: 1
2. Hojas de Te: 1
3. Carne de res: 1
4. Jamon: 1
5. Carne de cerdo: 1
6. Galletitas: 1
7. Pastel de cumpleaños: 1
Seleccione el producto que desea enviar:
>
```

Seleccione el índice del producto que desee, para este ejemplo seleccionaremos el número 2.

Aparecerá una línea de texto con el índice y nombre del producto. Como queda una iteración, se volverá a desplegar la lista y pedir una opción.

Tenga en cuenta que las cantidades de productos disponibles cambian con cada selección, y los índices pueden cambiar.

```
Seleccione el producto que desea enviar:
> 2
Ha seleccionado el producto # 2 Nombre del producto: Hojas de Te
Seleccione el producto que desea enviarle al cliente
0. Regresar al menu principal

1. Sandwich de pollo: 1
2. Carne de res: 1
3. Jamon: 1
4. Carne de cerdo: 1
5. Galletitas: 1
6. Pastel de cumpleaños: 1
Seleccione el producto que desea enviar:
>
```

En este ejemplo, como no quedan más hojas de té, los índices de la lista se organizaron para no mostrar las hojas de té.

Para esta iteración de ejemplo elijamos el índice 6: pastel de cumpleaños. Presionamos 6 y enter.

```
Ha seleccionado el producto # 6 Nombre del producto: Pastel de cumpleaños

Seleccione en que medio de transporte quiere enviar este producto
Junto a cada tipo de transporte se encuentra su precio.

Advertencia: Los tipos de transporte han sido filtrados de manera que solo puede seleccionar los que puedan soportar el peso de su producto.
Su pedido pesa 2 kilogramos
0. Regresar al menu principal
1. Camion $5.0
2. Avion $10.0
3. Automovil $3.0
4. Camioneta $4.0
5. Bicicleta $1.0
6. Patines $1.0
7. Barco $15.0
8. Helicoptero $22.0
9. Tren $33.0
10. Caminando $1.0

Seleccione el número del tipo de transporte:
>
```

Ahora se desplegará una lista de transportes, esta lista ha sido filtrada para que todos los transportes que salgan tengan la capacidad de carga para llevar sus productos.

Para esta iteración de ejemplo elijamos el índice 5: bicicleta. Presionamos 5 y enter.

La siguiente pregunta será si desea aplicarle envío gratis al pedido.

```
Ha seleccionado el transporte #5
El pedido se enviará por Bicicleta
Desea aplicar envío gratis?
 1. Si   2. No
>
```

Para este ejemplo, digamos que queremos enviarlo sin costo, se presiona 1 y enter.

```
Su tarifa de envío ha bajado de: $1.0 a $0

Digite el día del mes entre 1 y 30:
>
```

Por temas de contabilidad, se pide el día del mes. Si estamos en el día 23 del mes, presionamos 23 y enter.

A continuación verá una factura en consola. Esta factura contiene el detalle de los productos comprados y el transporte.

La factura para el ejemplo que hemos realizado es:

```
************************************
Factura generada en la tienda Consumibles de la Abuela Tata
A nombre del cliente: Diego Armando Maradona
Nombre del producto: Hojas de Te
Precio del producto: 6.0
Nombre del producto: Pastel de cumpleaños
Precio del producto: 20.0
Tipo de transporte: BICICLETA
Tarifa de envio: 0.0
Total a pagar: 26.0
************************************
```

Por último, le saldrá esta pregunta:

```
¿Desea hacer otro envio o volver al menu principal?
0. Volver al menu principal
1. Realizar otro  envio
```

Presione 0, enter para volver al menú principal de las funciones y 1, enter para realizar otro envío.

- Funcionalidad 2: Pago a trabajadores

Al seleccionar la opción 2 en el menú principal se le pide al administrador que ingrese el número asociado al tipo de trabajador que desea pagarle o que digite 0 si desea volver al menú principal:

```
Ingrese el número de la opción que desea utilizar
> 2

Has escogido la opción pagar a trabajadores

¿A qué tipo de trabajador desea pagarle?

1. Operarios
2. Conductores
3. Vendedores
0. Volver al menú principal
>
```

Tanto esta como las demás opciones de enunciados siguientes tienen validaciones, por lo que si se digita un número incorrecto saldrá un mensaje como el que se muestra a continuación:

```
> 4
El valor ingresado no es válido. Ingreselo nuevamente por favor.
>
```

Al seleccionar uno de los tipos, se mostrará una lista del tipo de trabajadores que se ha escogido que se encuentran en las facturas y no se les ha pagado por el trabajo realizado. Si no hay pagos pendientes o aún no se han realizado envíos se mostrará lo siguiente, para que el administrador decida salir o volver a iniciar el proceso de pago y buscar otro tipo de trabajador:

```
Lo sentimos, ningun de trabajador de este tipo ha trabajado por el momento

¿Qué desea hacer?
1. Pagar a otro trabajador
0.Volver al menu principal
>
```

En el caso de que si hayan trabajadores para pagarles, se desplegará lo siguiente, en este ejemplo seleccionamos el tipo número 3 (vendedores):

```
Esta es la lista de Vendedores que han trabajado

Trabajador 1
Nombre: Lionel Andres
Edad: 22
Cedula: 14720
Tienda: Zaha Hadid Arquitectura

Trabajador 2
Nombre: Maria Beatriz
Edad: 20
Cedula: 57793
Tienda: Hefesto Construcciones

Trabajador 3
Nombre: Adriana
Edad: 21
Cedula: 89235
Tienda: Vitruvio Edificios

Por favor, ingrese el número del trabajador al cual desea pagarle [1-3]:
>
```

Luego de elegir el trabajador, que en este caso es un vendedor, se mostrará en la pantalla la cantidad de dinero que le pagará por el número de veces que ha trabajado, luego se mostrará al administrador si desea revisar la información sobre las metas del vendedor o si no continuar con el pago normalmente:

```
Por favor, ingrese el número del trabajador al cual desea pagarle [1-3]:
> 3

Al trabajor Adriana se le pagará 6000 por trabajar 1 veces

¿Desea analizar y pagar al trabajador por sus metas cumplidas?
1.Si
2.No

Digite el número de la opción que desee
>
```

Caso 1: Elige continuar con el pago sin analizar las metas

Se mostrará la secuencia de pago y su comprobante asociado, para después preguntar al administrador si desea realizar otro pago y repetir el proceso explicado o volver al menú principal:

```
Digite el número de la opción que desee
> 2
... Realizando pago ... Por favor espere ...

El pago fue realizado con éxito

**************************
Comprobante
Pago asociado a los envios realizados 6000
Pago asociado al cumpliento de metas 0
Total = 6000
**************************

¿Qué desea hacer?
1. Pagar a otro trabajador
0.Volver al menu principal
>
```

Caso 2: Elige analizar las metas

Al seleccionar de opción si (1) , se desplegará la lista de metas del tipo de trabajador escogido con sus especificaciones, para que se escoja cual se desea analizar y observar estadísticas:

```
Digite el número de la opción que desee
> 1

El trabajador escogido tiene las siguientes metas

Indice: número de productos vendidos

Meta 1
Nivel de dificultad: Fácil
indice requerido: 3.0
Bonificación por cumplimiento: 10000.0

Meta 2
Nivel de dificultad: Dificil
indice requerido: 10.0
Bonificación por cumplimiento: 50000.0


Seleccione una de las metas
>
```

Se selecciona la meta por la cual se quiere bonificar al trabajador y dependiendo de si se cumple a no se muestran los siguientes mensajes.

Si se cumple, por ejemplo con la meta 1:

```
Seleccione una de las metas
> 1

La meta ha sido cumplida exitósamente
Sumaremos al pago la bonificación por esta meta
Porcentaje de la meta cumplido: 133.33333333333334%

¿Qué desea hacer?
 1.Revisar otra meta
 2.Proceder con el pago
>
```

Si no se cumple, por ejemplo con la meta 2:

```
Seleccione una de las metas
> 2

La meta aún no ha sido cumplida
Porcentaje de la meta cumplido: 40.0%
Porcentaje faltante: 60.0%
Cantidad faltante del indice indicado: 6.0

¿Qué desea hacer?
 1.Revisar otra meta
 2.Proceder con el pago
>
```

Luego de mirar alguna meta, salen las opciones de revisar otra meta, si la meta que se escoge ya fue revisada manda el mensaje:

```
Seleccione una de las metas
> 1
Esta opción ya fue mostrada

¿Qué desea hacer?
 1.Revisar otra meta
 2.Proceder con el pago
>
```

Cuando sean revisadas las metas deseadas el administrador seleccionará la opción 2 para prodecer con el pago, es aquí donde finaliza la funcionalidad y se vuelve a preguntar si se desea realizar el pago a otro trabajador o volver al menú principal:

```
> 2
... Realizando pago ... Por favor espere ...

El pago fue realizado con éxito

**************************
Comprobante
Pago asociado a los envios realizados 6000
Pago asociado al cumpliento de metas 10000
Total = 16000
**************************

¿Qué desea hacer?
1. Pagar a otro trabajador
0.Volver al menu principal
>
```

- Funcionalidad 3: Abastecimiento de tiendas

Todas las entradas deben estar dentro del rango adecuado y solo reciben números enteros, en caso tal de que no saldrá una advertencia.
Al ingresar en la opción 3 del menú principal se le da al administrador una lista de todas las tiendas junto con los productos que tiene dentro de la fábrica con el fin de decidir a cual se le desea distribuir productos.

```
Abastecer tiendas - Apartado de tiendas

0. Volver al menu anterior

1. Hefesto Construcciones-PRODUCTOS:
     LimpiaMax: 1
     Sandwich de pollo: 1
     Ladrillo: 1

2. Vitruvio Edificios-PRODUCTOS:
     Lavadora: 1
     Hojas de Te: 1
     Adhesivo: 1

3. Zaha Hadid Arquitectura -PRODUCTOS:
     Marmol: 1
     Jabon: 1
     Carne de res: 1

Seleccione la tienda a la que desea enviar: 2
```

Para este ejemplo seleccionaremos la tienda 2 después de hacer eso se despliega el siguiente menú.

```
Abastecer tiendas - Apartado de productos

La capacidad de productos para esta tienda es la siguiente:
Aseo 1/20  Consumible 1/15  Construccion 1/12

0. Regresar al menu anterior

INDICE-PRODUCTO-PESO-PRECIO-CATEGORIA
1. LimpiaMax - 50.0 - 30.0 - aseo
2. Sandwich de pollo - 0.7 - 1000.0 - consumible
3. Ladrillo - 2.5 - 0.25 - construccion
4. Lavadora - 675.0 - 20.0 - aseo
5. Hojas de Te - 60.0 - 15.0 - consumible
6. Adhesivo - 0.3 - 5.0 - construccion
7. Jabon - 271.0 - 10.0 - aseo
8. Carne de res - 265.0 - 1000.0 - consumible
9. Marmol - 3000000.0 - 1000.0 - construccion
10. Granito - 27800.0 - 750.0 - construccion
11. Jamon - 27800.0 - 750.0 - consumible
12. Carne de cerdo - 27800.0 - 750.0 - consumible
13. Papel higiénico - 1.0 - 2.0 - aseo
14. Pintura blanca - 1.5 - 30.0 - construccion
15. Detergente - 1.2 - 20.0 - aseo
16. Cemento - 40.0 - 10.0 - construccion
17. Cepillo de dientes - 0.1 - 5.0 - aseo
18. Bolsas de basura - 0.5 - 3.0 - aseo
19. Lámpara LED - 0.2 - 50.0 - construccion
20. Jabón líquido - 0.5 - 8.0 - aseo

Seleccione el producto que desea enviar: 3
```

Este menú contiene el límite de productos por categoría que tiene la tienda y la cantidad de productos que ya tiene por categoría, además de los productos que pueden ser abastecidos a la tienda para este ejemplo seleccionaremos el producto 3.

Se despliega el siguiente menú el cual nos pide que ingresemos la cantidad de productos que queremos abastecer se debe tener en cuenta que la cantidad de productos no debe ser mayor a la siguiente resta Límite por categoría - productos existentes en categoría.

```
Escriba la cantidad de productos que desea abastecer: 10
```

Después de este se despliega el menú del transporte por el que quieres hacer el envío:

```
Seleccione en que medio de transporte quiere enviar este producto

Advertencia: Los tipos de transporte han sido filtrados de manera que solo puede seleccionar los que puedan soportar el peso de su producto.
0. Regresar al menu principal
1. Camion
2. Avion
3. Automovil
4. Camioneta
5. Bicicleta
6. Barco
7. Helicoptero
8. Tren

Seleccione el número del tipo de transporte:
> 5
```

En este menú se debe especificar porque medio de transporte se quiere llevar el abastecimiento a la tienda.
Ya finalmente solo queda el menú de si se quiere realizar otro abastecimiento o volver al menú principal.

```
Ha seleccionado el transporte #5
La tienda se abastecera por: Bicicleta
El producto fue enviado con exito ahora la tienda tiene
PRODUCTOS:
     Lavadora: 1
     Hojas de Te: 1
     Ladrillo: 10
     Adhesivo: 1

0.Volver al menu principal
1. Realizar más abastecimientos
> 0
```

- Funcionalidad 4: Devoluciones de pedidos

NOTA: el programa explota, si se digita una letra, espacio, o cualquier carácter que no sea un número entero positivo menor a 2.147.483.647 el programa no va a correr

Luego de seleccionar la opción 4 en el menú principal

```
4. Gestionar devoluciones
```

se desplegará el siguiente menú con las listas de las facturas disponibles:

```
Por favor seleccione el número que le corresponda
a la factura para realizar la devolucion

0. Volver al menu principal
1. ID: 1 Cliente: Luis Gómez
2. ID: 2 Cliente: María García
3. ID: 3 Cliente: Juan Pérez
4. ID: 4 Cliente: Luis Gómez
5. ID: 5 Cliente: Pedro Gómez
6. ID: 6 Cliente: Josesito Gómez
7. ID: 7 Cliente: Juan Pérez
8. ID: 8 Cliente: Julián Araña Alvarez
9. ID: 9 Cliente: Emiliano Dibu Martinez
10. ID: 10 Cliente: Diego Armando Maradona

Digite su opcion:
>
```

}

En la consola se pedirá que digite el número del índice (número que está más a la izquierda seguido de un punto) de alguna de las facturas que aparece en pantalla, según lo que se le pase a la consola pueden pasar 4 cosas:

1. **Volver al menú principal:** si digita la opción 0 lo llevará de nuevo al menú principal de la aplicación:

```
Menú principal Distribuidora SAS

1. Enviar pedido
2. Pagar a trabajadores
3. Abastecer tiendas
4. Gestionar devoluciones
5. Mostrar estadísticas
6. Salir

Ingrese el número de la opción que desea utilizar
>
```

2. Vuelve a mostrar las facturas: Si escribe un número fuera del rango del índice mostrado, se le indicará que cometió un error y se procede a volver a mostrar las facturas:

```
Digite su opcion:
> 12

¡ATENCION! La opcion que digitó es incorrecta

Por favor seleccione el número que le corresponda
a la factura para realizar la devolucion

0. Volver al menu principal
1. ID: 1 Cliente: Luis Gómez
2. ID: 2 Cliente: María García
3. ID: 3 Cliente: Juan Pérez
4. ID: 4 Cliente: Luis Gómez
5. ID: 5 Cliente: Pedro Gómez
6. ID: 6 Cliente: Josesito Gómez
7. ID: 7 Cliente: Juan Pérez
8. ID: 8 Cliente: Julián Araña Alvarez
9. ID: 9 Cliente: Emiliano Dibu Martinez
10. ID: 10 Cliente: Diego Armando Maradona

Digite su opcion:
>
```

3. Volver a mostras las facturas (2): en caso de que la factura que intentó seleccionar ya se han devuelto todos los producto de la misma saldrá otro mensaje de advertencia indicandolo, y se procede a volver a mostrar las facturas para escoger otra opción

```
Seleccionó la factura con la opcion número: 10
En la factura que seleccionó ya se han devuelto todos los productos
Por favor digite el Número de otra factura

Por favor seleccione el número que le corresponda
a la factura para realizar la devolucion

0. Volver al menu principal
1. ID: 1 Cliente: Luis Gómez
2. ID: 2 Cliente: María García
3. ID: 3 Cliente: Juan Pérez
4. ID: 4 Cliente: Luis Gómez
5. ID: 5 Cliente: Pedro Gómez
6. ID: 6 Cliente: Josesito Gómez
7. ID: 7 Cliente: Juan Pérez
8. ID: 8 Cliente: Julián Araña Alvarez
9. ID: 9 Cliente: Emiliano Dibu Martinez
10. ID: 10 Cliente: Diego Armando Maradona

Digite su opcion:
>
```

4. Siguiente menú: Si el administrador selecciona un entero dentro del rango se procederá a mostrar en pantalla un listado con los productos que contiene esa factura (productos enviados al cliente)

```
Digite su opcion:
> 9

Seleccionó la factura con la opcion número: 9

Por favor seleccione el número que le corresponda
a el producto para realizar la devolucion

0. Volver al menu principal
1. Producto: Ladrillo (Devuelto)
2. Producto: Lámpara LED
3. Producto: Adhesivo
4. Producto: Cemento (Devuelto)
5. Producto: Marmol

Digite su opcion:
>
```

como se puede observar en la lista de productos hay algunos que van acompañados de la palabra devueltos, esto indica que en devoluciones anteriores se devolvió ese producto y no puede devolverse 2 veces, a partir de este menú puede pasar lo siguiente:

1. Menú principal, si se seleccioná la opción 0 se mostrará en pantalla de nuevo el menú principal de la aplicación.

2. Volver a mostrar productos: Si se digita un numero fuera del rango del índice (número más a la izquierda seguido de un punto) ó se selecciona un producto que ya ha sido devuelto (producto acompañado de la palabra devuelto) se mostrará una advertencia y se procede a enseñar en pantalla de nuevo la lista de productos que contiene la factura seleccionada

```
Por favor seleccione el número que le corresponda
a el producto para realizar la devolucion

0. Volver al menu principal
1. Producto: Ladrillo (Devuelto)
2. Producto: Lámpara LED
3. Producto: Adhesivo
4. Producto: Cemento (Devuelto)
5. Producto: Marmol

Digite su opcion:
> 1

¡ATENCION! La opcion que digitó es incorrecta o el producto ya ha sido devuelto

Por favor seleccione el número que le corresponda
a el producto para realizar la devolucion

0. Volver al menu principal
1. Producto: Ladrillo (Devuelto)
2. Producto: Lámpara LED
3. Producto: Adhesivo
4. Producto: Cemento (Devuelto)
5. Producto: Marmol

Digite su opcion:
>
```

3. Eliminar producto: si la opción escrita por el admin está dentro del rango y el producto no ha sido devuelto se procede a hacer la devolución del producto que demora aproximadamente 2 segundos y saldrá una notificación en la consola indicando que el producto ha sido devuelto

```
Por favor seleccione el número que le corresponda
a el producto para realizar la devolucion

0. Volver al menu principal
1. Producto: Ladrillo (Devuelto)
2. Producto: Lámpara LED
3. Producto: Adhesivo
4. Producto: Cemento (Devuelto)
5. Producto: Marmol

Digite su opcion:
> 3

Seleccionó el producto con la opcion número: 3
... Realizando devolución ... Por favor espere ...
¡¡ El producto ha sido devuelto exitosamente !!

¿Desea hacer otra devolucion o volver al menu principal?

0. Volver al menu principal
1. Realizar otra devolución
>
```

Como se ve en la imagen una vez realizada la devolución se pregunta por consola si desea realizar otra devolución o volver al menú principal.

1. si selecciona un número diferente a 0 o 1 saldrá una advertencia

```
¿Desea hacer otra devolucion o volver al menu principal?

0. Volver al menu principal
1. Realizar otra devolución
> 2
¡ATENCION! La opcion que digitó es incorrecta
¿Desea hacer otra devolucion o volver al menu principal?

0. Volver al menu principal
1. Realizar otra devolución
>
```

2. si seleccioná 0 procederá a volver a mostrar el menú principal

3. si seleccioná 1 se muestran de nuevo el listado de facturas que corresponden a los envíos hechos (vuelve a comenzar la funcionalidad)

```
¿Desea hacer otra devolucion o volver al menu principal?

0. Volver al menu principal
1. Realizar otra devolución
> 1

Por favor seleccione el número que le corresponda
a la factura para realizar la devolucion

0. Volver al menu principal
1. ID: 1 Cliente: Luis Gómez
2. ID: 2 Cliente: María García
3. ID: 3 Cliente: Juan Pérez
4. ID: 4 Cliente: Luis Gómez
5. ID: 5 Cliente: Pedro Gómez
6. ID: 6 Cliente: Josesito Gómez
7. ID: 7 Cliente: Juan Pérez
8. ID: 8 Cliente: Julián Araña Alvarez
9. ID: 9 Cliente: Emiliano Dibu Martinez
10. ID: 10 Cliente: Diego Armando Maradona

Digite su opcion:
>
```

- Funcionalidad 5: Muestra de estadísticas

Al ingresar la opción 5, se le da la opción al administrador de considerar un rango de fechas en específico (especificadas con dos números enteros), o analizar todo el histórico, es decir, desde la primera fecha hasta la última.

```
REPORTE

1. Analizar toda la información
2. Ingresar fechas específicas
```

Si se digita la opción 2, se le pide al administrador que ingrese dos fechas entre el rango permitido:

```
La fecha mínima es 1 y la fecha máxima es 5
Ingrese fecha de inicio:

> 2
Ingrese fecha final:

> 5
```

Ya sea que se hayan ingresado fechas específicas o se haya decidido analizar toda la información, se despliega el siguiente menú.. En él se ofrecen varios tipos de información de interés para el administrador:

```
Ingrese información a obtener

1. Ganancias Discretas
2. Ganancias Totales
3. Promedio por dia
4. Aumento porcentual
5. Modas estadisticas
6. Cambiar fechas
0. Cancelar
```

Veamos qué hace cada una de estás opciones:

(1) **Ganancias discretas:** Muestra una tabla con la información de las ganancias obtenidas en cada una de las fechas del rango establecido. Solo se muestran los días en los que se obtuvieron ganancias

```
DIA       GANANCIA
2         927.0
5         30.0
```

(2) **Ganancias Totales**: Muestra la suma de todas las ganancias en el rango de fechas establecido.

```
Las ganancias totales entre las fechas ingresadas han sido: 957.0
```

(3) **Promedio por día**: Muestra el promedio de ganancias por día, respecto al rango de fechas establecido.

```
El promedio por día es: 478.5
```

(4) **Aumento porcentual:** Muestra una tabla, la cual refleja el porcentaje de aumento en las ganancias de cada fecha respecto al día anterior. Por lo tanto, no hay aumento para el primer día. Números negativos reflejan que se ganó menos que el día anterior.

```
DIA       AUMENTO
5         -96.76375404530745%
```

(5) **Modas estadísticas:** Este menú permite conocer cuál ha sido la tienda más usada, el transporte más usado, y el cliente al que más se le ha vendido.

```
Seleccione moda:

1. Tienda más usada
2. Transporte más usado
3. Cliente al que más se le ha vendido
```

Si se selecciona la opción 1:

```
La tienda más usada ha sido Consumibles de la Abuela Tata
```

Si se selecciona la opción 2:

```
El transporte más usado ha sido Caminando
```

Si se selecciona la opción 3:

```
El cliente al que más se le ha vendido ha sido Luis Gómez
```