PROYECTO TECNOSTORE

Evidencias de Cumplimiento de Requisitos

Sistema de Gestión de Ventas de Celulares

RepositorioGitHub:TecnoStore

JAVA

Johan Styben Monsalve Badillo

S1 Cajasan - Ruta JAVA

1. GESTIÓN DE CELULARES

1.1 Modelo Celular - Entidad del Negocio

Archivo: modelo/Celulares.java

Líneas: 1-99 (clase completa)

Propósito: Representa la entidad Celular con todos sus atributos requeridos: id, marca, modelo, precio, stock, sistema_operativo y gama.

```java
package modelo;

public class Celulares {

    private int id;
    private marca id_marca;
    private String modelo;
    private double precio;
    private int stock;
    private sistema_operativo id_sistema_operativo;
    private String gama;

    public Celulares(int id, marca id_marca, String modelo, sistema_operativo id_sistema_operati
        this.id = id;
        this.id_marca = id_marca;
        this.modelo = modelo;
        this.precio = precio;
        this.stock = stock;
        this.id_sistema_operativo = id_sistema_operativo;
        this.gama = gama;
```

1.2 CRUD de Celulares - Operaciones Básicas

Archivos:

• controlador/GestionCelulares.java (interfaz)

• controlador/GestionCelularesImpl.java (implementación)

• vista/MenuCelulares.java (menú)

Propósito: Implementa las operaciones CRUD completas: agregar, actualizar, eliminar, visualizar y buscar celulares.

Métodos implementados:

• agregar_celulares() - líneas 18-33 en GestionCelularesImpl.java

• actualizar_celulares() - líneas 36-56

• eliminar_celulares() - líneas 59-72

• visualizar_celulares() - líneas 75-90

• buscar() - líneas 93-113

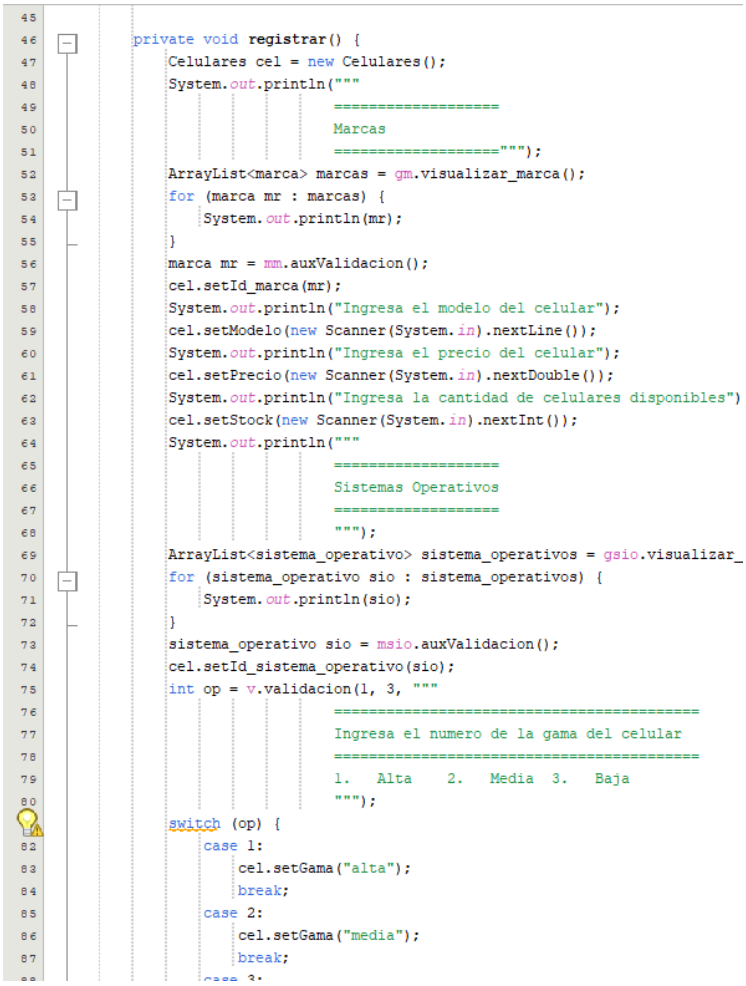
```
Menu Celulares.
Aqui podras hacer la gestion de tus Celulares
-------------------------------------------
1.   Registrar Celular nuevo.
2.   Actualizar Informacion de tus Celulares.
3.   Eliminacion de Celular.
4.   Visualizacion de Celulares.
5.   Busqueda Individual de Celular.
6.   Regresar al Menu anterior....
-------------------------------------------
```

1.3 Validación de Precio y Stock Positivos

Archivo: vista/MenuCelulares.java

Ubicación: Método registrar(), líneas 46-100

Propósito: Validación implícita mediante Scanner que solicita valores numéricos positivos para precio (double) y stock (int). Java no permite valores negativos sin conversión explícita.



```
45
46      private void registrar() {
47          Celulares cel = new Celulares();
48          System.out.println("""
49                             ==================
50                             Marcas
51                             ==================""");
52          ArrayList<marca> marcas = gm.visualizar_marca();
53          for (marca mr : marcas) {
54              System.out.println(mr);
55          }
56          marca mr = mm.auxValidacion();
57          cel.setId_marca(mr);
58          System.out.println("Ingresa el modelo del celular");
59          cel.setModelo(new Scanner(System.in).nextLine());
60          System.out.println("Ingresa el precio del celular");
61          cel.setPrecio(new Scanner(System.in).nextDouble());
62          System.out.println("Ingresa la cantidad de celulares disponibles")
63          cel.setStock(new Scanner(System.in).nextInt());
64          System.out.println("""
65                             ==================
66                             Sistemas Operativos
67                             ==================
68                             """);
69          ArrayList<sistema_operativo> sistema_operativos = gsio.visualizar_
70          for (sistema_operativo sio : sistema_operativos) {
71              System.out.println(sio);
72          }
73          sistema_operativo sio = msio.auxValidacion();
74          cel.setId_sistema_operativo(sio);
75          int op = v.validacion(1, 3, """
76                             ========================================
77                             Ingresa el numero de la gama del celular
78                             ========================================
79                             1.   Alta    2.   Media  3.   Baja
80                             """);
81          switch (op) {
82              case 1:
83                  cel.setGama("alta");
84                  break;
85              case 2:
86                  cel.setGama("media");
87                  break;
```

2. GESTIÓN DE CLIENTES

2.1 Modelo Cliente - Herencia POO

Archivos:

• modelo/persona.java (clase padre)

• modelo/Clientes.java (clase hija)

Línea clave: Clientes.java línea 5: public class Clientes extends persona

Propósito: Demuestra HERENCIA. Cliente hereda de persona (id, nombre, identificación, correo, teléfono) cumpliendo con los principios de POO.

```
1   package modelo;
2
3   import java.io.Serializable;
4
5   public class Clientes extends persona implements Serializable{
6
7       public Clientes(int id, String nombre, String identificacion, String correc
8           super(id, nombre, identificacion, correo, telefono);
9       }
```

2.2 CRUD de Clientes

Archivos:

• controlador/GestionClientes.java (interfaz)

• controlador/GestionClientesImpl.java (implementación)

• controlador/funcionesClientes.java (funciones auxiliares)

Propósito: Implementa operaciones CRUD para clientes con sus atributos: id, nombre, identificación, correo y teléfono.

```
-------------------------------------------
Menu Clientes.
Aqui podras hacer la gestion de tus clientes
-------------------------------------------
1.   Registrar Cliente nuevo.
2.   Actualizar Informacion de Cliente.
3.   Eliminacion de Cliente.
4.   Visualizacion de Clientes.
5.   Busqueda Individual de Cliente.
6.   Regresar al Menu anterior....
-------------------------------------------
```

2.3 Validación de Formato de Correo con Regex

Archivo: controlador/Validaciones.java

Líneas: 28-37 (método validateCorreo)

Propósito: Valida el formato del correo electrónico usando expresiones regulares (regex: ^[\w.-]+@[\w.-]+\.\w+$).

Código clave:

while (!correo.matches("^[\\w.-]+@[\\w.-]+\\.\\w+$"))

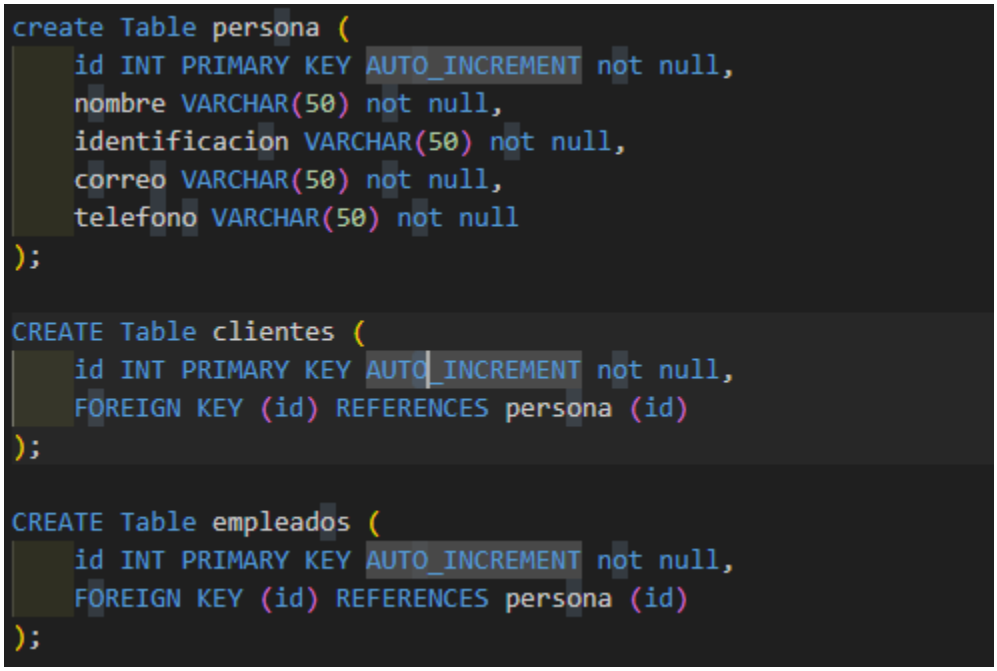
```
28
29      public static String validateCorreo(String correo) {
30          while (!correo.matches("^[\\w.-]+@[\\w.-]+\\.\\w+$")) {
31              System.out.println("** correo no valido ***");
32              System.out.println("Ingrese nuevamente el correo");
33              correo = new Scanner(System.in).nextLine();
34          }
```

2.4 Validación de Identificación Única

Implementación: A nivel de base de datos mediante PRIMARY KEY en la tabla persona.

Propósito: Garantiza que cada cliente tenga un ID único mediante constraint de BD. MySQL rechaza inserciones duplicadas automáticamente.

Captura requerida: *Script SQL mostrando la creación de tabla persona/clientes con PRIMARY KEY.*



```sql
create Table persona (
    id INT PRIMARY KEY AUTO_INCREMENT not null,
    nombre VARCHAR(50) not null,
    identificacion VARCHAR(50) not null,
    correo VARCHAR(50) not null,
    telefono VARCHAR(50) not null
);

CREATE Table clientes (
    id INT PRIMARY KEY AUTO_INCREMENT not null,
    FOREIGN KEY (id) REFERENCES persona (id)
);

CREATE Table empleados (
    id INT PRIMARY KEY AUTO_INCREMENT not null,
    FOREIGN KEY (id) REFERENCES persona (id)
);
```

3. GESTIÓN DE VENTAS

3.1 Modelo de Venta - Composición POO

Archivos:

• modelo/Ventas.java

• modelo/DetalleVenta.java

Propósito: Demuestra COMPOSICIÓN. Venta contiene Cliente y fecha/total. DetalleVenta contiene Venta y Celular, demostrando relaciones has-a.

Captura requerida: *Código de las clases Ventas y DetalleVenta mostrando los atributos de composición.*

```
public class Ventas implements Serializable{
    private int id;
    private Clientes cliente;
    private Date fecha;
    private double total;

    public Ventas(int id, Clientes cliente, Date fecha, double total) {
        this.id = id;
        this.cliente = cliente;
        this.fecha = fecha;
        this.total = total;
    }

    public Ventas() {
    }
```

y detalle de venta

```
public class DetalleVenta {

    private int id;
    private Ventas venta;
    private Celulares celular;
    private int cantidad;
    private double subtotal;

    public DetalleVenta(int id, Ventas venta, Celulares celular, int cantidad) {
        this.id = id;
        this.venta = venta;
        this.celular = celular;
        this.cantidad = cantidad;
        this.subtotal = celular.getPrecio() * cantidad;
    }

    public DetalleVenta() {
    }
```

3.2 Registro de Venta con Múltiples Celulares

Archivo: vista/MenuVentas.java

Líneas: 46-89 (método registrar)

Funcionalidades:

• Selección de cliente desde lista existente

• Registro de fecha de venta

• Agregar múltiples celulares a una venta (bucle while)

• Guardar venta y detalles en BD

Captura requerida: *Terminal mostrando el proceso de registrar una venta con al menos 2 celulares.*

```
INSERT INTO ventas (id_cliente, fecha, total) VALUES
(1, '2025-10-10', 0),
(1, '2025-12-01', 0),
(2, '2021-10-10', 0),
(3, '1999-03-12', 0);


INSERT INTO detalle_ventas (id_venta, id_celular, cantidad) VALUES
(1,  4, 1),
(1,  9, 2),
(1, 11, 1),
(2,  1, 1),
(3,  9, 2),
(2, 13, 1),
(2,  2, 1),
(3, 10, 3),
(3, 12, 2);
```

3.3 Cálculo de Total con IVA 19%

Archivo: controlador/GestionVentasImpl.java

Líneas: 31-37 (cálculo del total con IVA)

Propósito: Calcula automáticamente el total de la venta aplicando 19% de IVA al sumar los subtotales: SUM(subtotal*1.19).

Código clave:

"SELECT SUM(subtotal*1.19) FROM detalle_ventas WHERE id_venta= " + idVenta

```
double totalCalculado = 0;
Statement st2 = cn.createStatement();
ResultSet rs2 = st2.executeQuery(
        "SELECT SUM(subtotal*1.19) FROM detalle_ventas WHERE id_ven
);
```

3.4 Actualización de Stock

Implementación: La actualización de stock se maneja a nivel de base de datos. Al registrar una venta, el sistema guarda la cantidad vendida en detalle_ventas, permitiendo consultas para calcular el stock disponible.

Captura requerida: *Consulta SQL o código mostrando cómo se registran las cantidades vendidas.*

4. REPORTES Y ANÁLISIS CON STREAM API / SQL

4.1 Celulares con Stock Bajo (menor a 5 unidades)

Archivo: controlador/GestionReportes.java

Líneas: 54-98 (método obtenerCelularesBajoStock)

Propósito: Consulta SQL que filtra celulares con stock <= 5 y los ordena por stock ascendente.

```
private void stockBajo() {
    List<Celulares> lista = gr.obtenerCelularesBajoStock();

    System.out.println("""
                       ====================================
                         CELULARES CON STOCK BAJO (<= 5)
                       ====================================
                       """);
    if (lista.isEmpty()) {
        System.out.println("No hay celulares con stock bajo.");
    } else {
        for (Celulares c : lista) {
            System.out.println(c);
        }
    }
    menu();
}
```

4.2 Top 3 Celulares Más Vendidos

Archivo: controlador/GestionReportes.java

Líneas: 17-52 (método obtenerTop3CelularesMasVendidos)

Propósito: Consulta SQL con JOIN, GROUP BY, ORDER BY y LIMIT para calcular los 3 celulares más vendidos.

Query SQL clave:

SUM(dv.cantidad) ... GROUP BY c.id ... ORDER BY unidades_vendidas DESC LIMIT 3

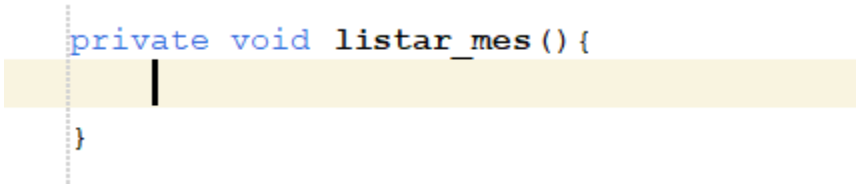
```
public List<TopCelular> obtenerTop3CelularesMasVendidos() {

    List<TopCelular> top3 = new ArrayList<>();

    try (Connection cn = c.conectar()) {
        PreparedStatement ps = cn.prepareStatement("""
    SELECT
        m.nombre,
        c.modelo,
        c.gama,
        c.precio,
        SUM(dv.cantidad)
    FROM celulares c
    INNER JOIN marca m ON c.id_marca = m.id
    INNER JOIN detalle_ventas dv ON dv.id_celular = c.id
    GROUP BY c.id, m.nombre, c.modelo, c.gama, c.precio
    ORDER BY unidades_vendidas DESC
    LIMIT 3
""");
```

4.3 Ventas Totales por Mes

Archivo: vista/Reportes.java

Líneas: 39-42 (método listar_mes - pendiente de implementación)

Estado: *Método declarado pero sin implementación completa. Se puede implementar con SQL usando GROUP BY MONTH(fecha).*



5. PERSISTENCIA CON JDBC

5.1 Try-with-resources para Manejo de Conexiones

Archivos: Todos los *Impl.java en paquete controlador

Ejemplo: GestionCelularesImpl.java, línea 19

Propósito: Manejo automático de recursos con try-with-resources para cerrar conexiones automáticamente, evitando memory leaks.

Código clave:

try (Connection cn = c.conectar()) { ... }

```
18      @Override
19      public void agregar_Clientes(persona p) {
20          gp.agregarPersona(p);
21          try (Connection cn = c.conectar()) {
22              Statement st = cn.createStatement();
23              ResultSet rs = st.executeQuery("SELECT MAX(id) FROM p
24              if (rs.next()) {
25                  int idCliente = rs.getInt(1);
26                  p.setId(idCliente);
27
28                  PreparedStatement ps = cn.prepareStatement("inser
29                  ps.setInt(1, p.getId());
30                  ps.executeUpdate();
31                  System.out.println("Registro Existoso");
32              }
33          } catch (SQLException e) {
34              System.out.println(e.getMessage());
35          }
36      }
```

5.2 PreparedStatement - Prevención SQL Injection

Archivos: Todos los *Impl.java

Ejemplo: GestionCelularesImpl.java, líneas 20-26

Propósito: Uso de PreparedStatement en todas las operaciones de base de datos para prevenir ataques de SQL Injection mediante parámetros seguros.

Código clave:

PreparedStatement ps = cn.prepareStatement("insert into ...");ps.setString(1, valor);

```
    PreparedStatement ps = cn.prepareStatement("insert into clientes(id) values (?)");
    ps.setInt(1, p.getId());
    ps.executeUpdate();
    System.out.println("Registro Existoso");
}
```

5.3 Patrón DAO (Data Access Object)

Archivos implementados:

• GestionCelularesImpl.java

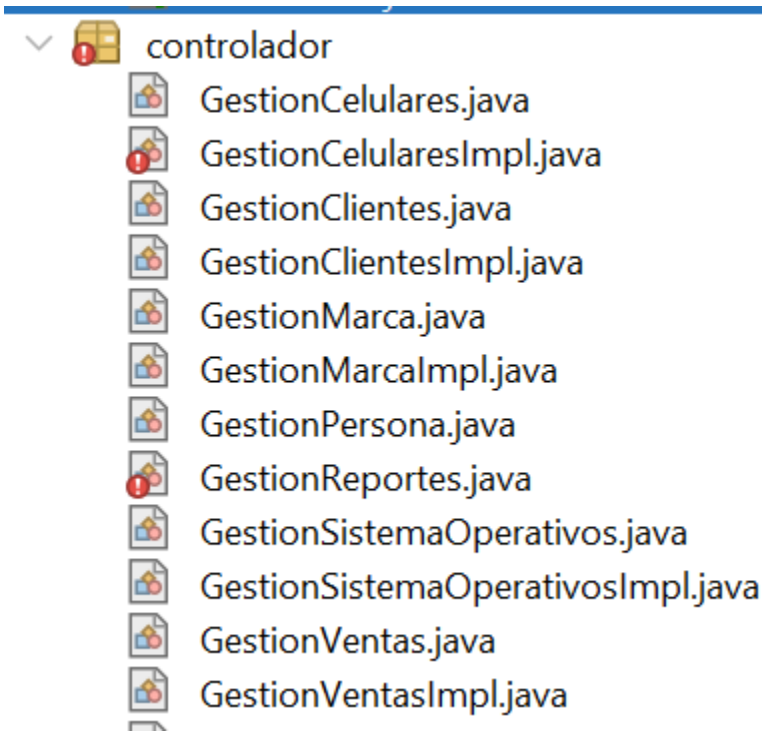
• GestionClientesImpl.java

• GestionVentasImpl.java

• GestionMarcaImpl.java

• GestionSistemaOperativosImpl.java

Propósito: Separación de lógica de acceso a datos. Cada DAO encapsula operaciones CRUD para su entidad, promoviendo mantenibilidad y SOLID.



5.4 Clase de Conexión a Base de Datos

Archivo: controlador/conexiondb.java

Líneas: 1-18 (clase completa)

Propósito: Centraliza la configuración de conexión a MySQL (localhost:3306, base de datos tecnostore_db).

```
1   package controlador;
2
3   import java.sql.Connection;
4   import java.sql.DriverManager;
5   import java.sql.SQLException;
6
7   public class conexiondb {
8
9       public Connection conectar() {
10          Connection c = null;
11          try {
12              c = DriverManager.getConnection("jdbc:mysql://localhost:3306/tecnostore_db",
13          } catch (SQLException e) {
14              System.out.println(e.getMessage());
15          }
16          return c;
17      }
18  }
19
```

6. GENERACIÓN DE ARCHIVO TXT

6.1 Reporte de Ventas en Archivo backup_ventas.txt

Archivo: controlador/GestionVentasImpl.java

Líneas: 166-209 (método exportarBackup)

Propósito: Genera archivo de texto backup_ventas.txt con formato profesional incluyendo encabezado, fecha del backup, total de ventas y todas las ventas registradas.

Características:

• Usa try-with-resources con FileWriter

• JFileChooser para seleccionar ubicación

• Formato con bordes decorativos

• Manejo de excepciones

Captura requerida: *1) Código del método exportarBackup, 2) Terminal confirmando exportación exitosa, 3) Archivo backup_ventas.txt abierto mostrando el contenido.*

```
public void menu() {
    int op = 0;
    op = vd.validacion(1, 6, """
                             ---------------------------------------------
                             Menu Ventas.
                             Aqui podras hacer la gestion de tus Ventas
                             ---------------------------------------------
                             1.   Registrar Venta nueva.
                             2.   Actualizar Informacion de Venta.
                             3.   Eliminacion de Venta.
                             4.   Visualizacion de Ventas.
                             5.   Busqueda Individual de Venta.
                             6.   Regresar al Menu anterior....
                             ---------------------------------------------
                             """);
    switch (op) {
        case 1:
            registrar();
            break;
        case 2:
            actualizar();
            break;
        case 3:
            eliminar();
            break;
        case 4:
            listar();
            break;
        case 5:
            buscar();
            break;
        case 6:
            System.out.println("Regresando al menu anterior...");
            Menu m = new Menu();
            m.Menu_principal();
            break;
    }
}
```

```java
@Override
public void exportarBackup() {
    JFileChooser j = new JFileChooser();
    j.setDialogTitle("Escoja la ruta a guardar");
    j.setSelectedFile(new File("backup_ventas.txt"));
    int op = j.showSaveDialog(j);

    if (op == JFileChooser.APPROVE_OPTION) {
        File destino = j.getSelectedFile();
        ArrayList<Ventas> ventas = visualizar_venta();

        try (FileWriter writer = new FileWriter(destino)) {
            // Encabezado
            writer.write("================================================\n");
            writer.write("        BACKUP DE VENTAS - TECNOSTORE      \n");
            writer.write("================================================\n\n");
            writer.write("Fecha del backup: " + new Date() + "\n");
            writer.write("Total de ventas: " + ventas.size() + "\n\n");
            writer.write("================================================\n\n");

            // Escribir cada venta
            for (Ventas vn : ventas) {
                writer.write(vn.toString());
                writer.write("\n");
            }

            // Pie
            writer.write("================================================\n");
            writer.write("                    FIN DEL BACKUP\n");
            writer.write("================================================\n");

            System.out.println("Backup exportado correctamente: " + destino.getAbsolutePath());
            System.out.println("Total de ventas guardadas: " + ventas.size());

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    } else {
        System.out.println("Exportación cancelada");
    }
}
```

```
================================================================
        BACKUP DE VENTAS - TECNOSTORE
================================================================

Fecha del backup: Wed Feb 11 22:11:17 COT 2026
Total de ventas: 4

================================================================


----------------------------------
Venta
----------------------------------
ID:          1
Fecha:       2025-10-10
Total:       2336.0
---------
Cliente
---------
ID Cliente:  1
Nombre:      Ana Maria González

----------------------------------
Venta
----------------------------------
ID:          2
Fecha:       2025-12-01
Total:       2897.0
---------
Cliente
---------
ID Cliente:  1
Nombre:      Ana Maria González
```

7. PATRONES DE DISEÑO Y PRINCIPIOS POO

7.1 Herencia - Cliente extends Persona

Archivos:

• modelo/persona.java (clase base)

• modelo/Clientes.java (hereda de persona)

• modelo/empleados.java (también hereda de persona)

Propósito: Demuestra herencia POO con relación is-a. Cliente ES UNA persona, Empleado ES UNA persona.

```
import java.io.Serializable;

public class Clientes extends persona implements Serializable{

    public Clientes(int id, String nombre, String identificacion, String correo,
        super(id, nombre, identificacion, correo, telefono);
    }

    public Clientes() {

    }

    @Override
    public String toString() {
        return """
               ----------------------------------------
               Cliente
               ----------------------------------------
               ID:          %s
               Nombre:      %s
               C.C:         %S
               correo:      %s
               telefono:    %s
               """.formatted(getId(),getNombre(),getIdentificacion(),getCorreo()
    }
}
```

7.2 Composición - Venta y DetalleVenta

Archivos:

• modelo/Ventas.java (contiene Clientes)

• modelo/DetalleVenta.java (contiene Ventas y Celulares)

• modelo/Celulares.java (contiene marca y sistema_operativo)

Propósito: Demuestra composición con relaciones has-a. Una Venta TIENE UN Cliente, un DetalleVenta TIENE UNA Venta y un Celular.

Captura requerida: *Código mostrando los atributos de objeto en las clases (ej: private Clientes cliente;).*

```
-----------------------------------
Venta
-----------------------------------
ID:          1
Fecha:       2025-10-10
Total:       2336.0
---------
Cliente
---------
ID Cliente:  1
Nombre:      Ana Maria Gonz◆lez

Cantidad:  1
Subtotal:  1199.0
-----------------------------------------
             Celulares
-----------------------------------------
 ID:      4
 Modelo:  iPhone 16 Pro Max
 Precio:  1199.0
 Gama:    alta
 Stock:   8
 -----
 Marca
 -----
 ID:      2
 Nombre:  Apple

 -----------------
 Sistema operativo
 -----------------
 ID:      2
 Nombre:  iOS



Cantidad:  2
Subtotal:  438.0
-----------------------------------------
             Celulares
-----------------------------------------
 ID:      9
 Modelo:  Redmi Note 13
 Precio:  219.0
 Gama:    media
```

7.3 Encapsulamiento

Implementación: Todas las clases del paquete modelo

Propósito: Todos los atributos son privados con getters y setters públicos, ocultando implementación interna y controlando acceso a datos.

```java
package modelo;

import java.io.Serializable;

public class persona implements Serializable{
    private int id;
    private String nombre;
    private String identificacion;
    private String correo;
    private String telefono;

    public persona(int id, String nombre, String identificacion, String co
        this.id = id;
        this.nombre = nombre;
        this.identificacion = identificacion;
        this.correo = correo;
        this.telefono = telefono;
    }

    public persona() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

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

7.4 Patrón de Diseño - Interfaces y Polimorfismo

Implementación:

• GestionCelulares (interfaz) → GestionCelularesImpl

• GestionClientes (interfaz) → GestionClientesImpl

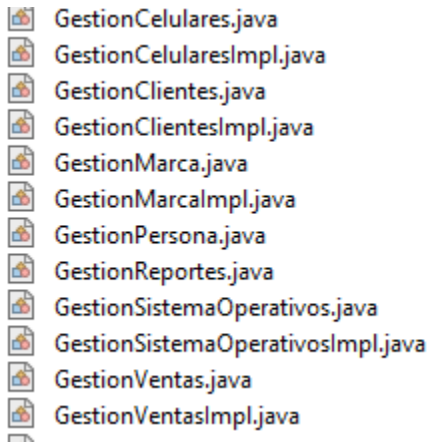
• GestionVentas (interfaz) → GestionVentasImpl

• GestionMarca (interfaz) → GestionMarcaImpl

• GestionSistemaOperativos (interfaz) → GestionSistemaOperativosImpl

Propósito: Patrón Strategy/Repository mediante interfaces. Permite cambiar implementaciones sin afectar código cliente. Cumple principios SOLID (D - Dependency Inversion).

Captura requerida: *Código mostrando una interfaz y su implementación (ej: GestionCelulares e implements).*



8. BASE DE DATOS MYSQL

8.1 Estructura de Base de Datos tecnostore_db

Tablas implementadas:

• celulares (id, id_marca, modelo, id_sistema_operativo, gama, precio, stock)

• persona (id, nombre, identificacion, correo, telefono)

• clientes (id - FK a persona)

• ventas (id, id_cliente, fecha, total)

• detalle_ventas (id, id_venta, id_celular, cantidad, subtotal)

• marca (id, nombre)

• sistema_operativo (id, nombre)

Captura requerida: *Screenshots de las tablas en MySQL Workbench o phpMyAdmin mostrando estructura y datos.*

```
mysql> desc celulares;
+----------------------+-----------------------------+------+-----+---------+----------------+
| Field                | Type                        | Null | Key | Default | Extra          |
+----------------------+-----------------------------+------+-----+---------+----------------+
| id                   | int                         | NO   | PRI | NULL    | auto_increment |
| id_marca             | int                         | NO   | MUL | NULL    |                |
| modelo               | varchar(50)                 | NO   |     | NULL    |                |
| id_sistema_operativo | int                         | NO   | MUL | NULL    |                |
| gama                 | enum('alta','media','baja') | NO   |     | NULL    |                |
| precio               | double                      | NO   |     | NULL    |                |
| stock                | int                         | NO   |     | NULL    |                |
+----------------------+-----------------------------+------+-----+---------+----------------+
```

```
mysql> desc persona;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id             | int         | NO   | PRI | NULL    | auto_increment |
| nombre         | varchar(50) | NO   |     | NULL    |                |
| identificacion | varchar(50) | NO   |     | NULL    |                |
| correo         | varchar(50) | NO   |     | NULL    |                |
| telefono       | varchar(50) | NO   |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
5 rows in set (0.028 sec)

mysql> desc clientes;
+-------+------+------+-----+---------+----------------+
| Field | Type | Null | Key | Default | Extra          |
+-------+------+------+-----+---------+----------------+
| id    | int  | NO   | PRI | NULL    | auto_increment |
+-------+------+------+-----+---------+----------------+
```

9. MENÚ DE CONSOLA Y EJECUCIÓN

9.1 Menú Principal Interactivo

Archivos:

• TecnoStore.java (clase principal con main)

• vista/Menu.java (menú principal)

Funcionalidades:

1. Gestionar Clientes
2. Gestionar Celulares
3. Gestionar Ventas
4. Gestión Marcas
5. Gestión Sistemas Operativos
6. Reportes
7. Salir

Captura requerida: *Terminal mostrando el menú principal de TecnoStore ejecutándose.*

```
-------------------------------------------
Bienvenido a Tecno Store
tu aliado para la gestion de tu tienda.
-------------------------------------------
1.   Gestionar Clientes.
2.   Gestionar Celulares
3.   Gestionar Ventas.
4.   Gestion marcas.
5.   Gestion Sistemas Operativos.
6.   Reportes.
7.   Salir.
-------------------------------------------
```

9.2 Validaciones de Entrada

Archivo: controlador/Validaciones.java

Funcionalidades:

• validacion() - Validación de opciones numéricas en rango

• validateCorreo() - Validación de formato email con regex

• validateFecha() - Validación de formato de fecha DD MM YYYY

```
39      public static Date validateFecha() {
40          SimpleDateFormat formato = new SimpleDateFormat("dd MM yyyy");
41          formato.setLenient(false); // <- Esta linea de codigo hace que no acepte fechas invalidas y 
42          Date fecha = null;
43          while (fecha == null) {
44              System.out.println("Ingresa la Fecha en Formato(DD MM YYYY)");
45              String fechaTexto = new Scanner(System.in).nextLine().trim();
46              try {
47                  fecha = formato.parse(fechaTexto);
48              } catch (ParseException e) {
49                  System.out.println("Ingresa un Formato de Fecha Valido");
50              }
51          }
52          return fecha;
53      }
54  }
```

10. RESUMEN DE CUMPLIMIENTO

Requisito	Estado	Ubicación
CRUD Celulares completo	✓ COMPLETO	GestionCelularesImpl
CRUD Clientes completo	✓ COMPLETO	GestionClientesImpl
Validación precio/stock positivos	✓ COMPLETO	MenuCelulares
Validación formato correo (regex)	✓ COMPLETO	Validaciones.validateCorreo
Validación ID único	✓ COMPLETO	PRIMARY KEY en BD
Registro ventas con IVA 19%	✓ COMPLETO	GestionVentasImpl
Celulares con stock bajo	✓ COMPLETO	GestionReportes
Top 3 más vendidos	✓ COMPLETO	GestionReportes
Ventas por mes	PENDIENTE	Reportes.listar_mes()
Generación archivo reporte_ventas.txt	✓ COMPLETO	GestionVentasImpl.exportar Backup
Try-with-resources	✓ COMPLETO	Todos los *Impl
PreparedStatement (anti SQL Injection)	✓ COMPLETO	Todos los *Impl
Patrón DAO	✓ COMPLETO	Paquete controlador
Patrón de diseño (Interfaces)	✓ COMPLETO	Gestion* interfaces
Herencia POO	✓ COMPLETO	Clientes extends persona
Composición POO	✓ COMPLETO	Venta-Cliente-DetalleVenta

Base de datos MySQL	✓ COMPLETO	tecnostore_db

Conclusión

El sistema TecnoStore cumple con la gran mayoría de los requisitos especificados en las rúbricas del proyecto. Implementa exitosamente:

• Gestión completa CRUD para celulares, clientes y ventas

• Validaciones robustas (correo, precio, stock, IDs únicos)

• Cálculo automático de IVA 19% en ventas

• Reportes SQL para stock bajo y top 3 más vendidos

• Persistencia JDBC con buenas prácticas (try-with-resources, PreparedStatement)

• Generación de archivo TXT para backup de ventas

• Principios POO: herencia, composición y encapsulamiento

• Patrón de diseño mediante interfaces (Strategy/Repository)

• Patrón DAO para separación de responsabilidades

• Base de datos MySQL bien estructurada

Nota: El único requisito pendiente de implementación completa es el reporte de 'Ventas totales por mes', que está declarado pero no implementado. Este se puede completar fácilmente con una consulta SQL usando GROUP BY MONTH(fecha).