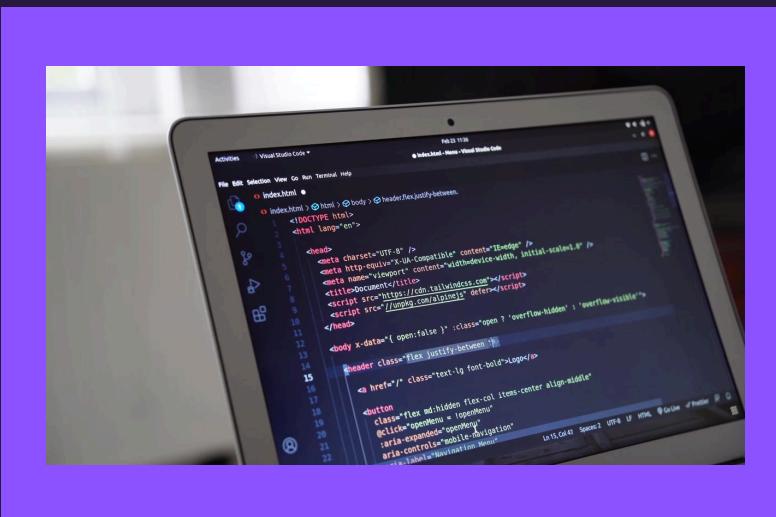


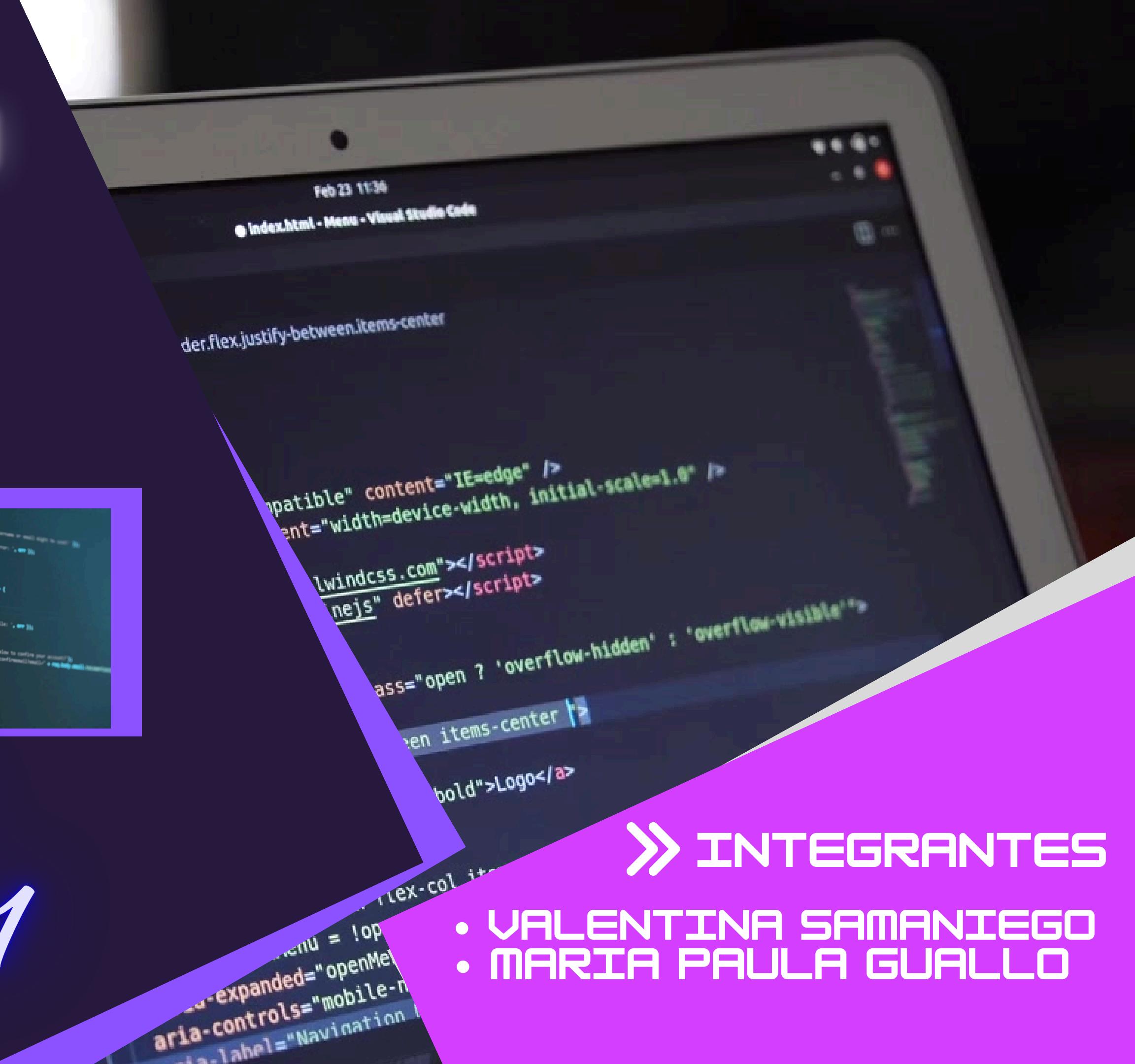
PROGRAMACIÓN ORIENTADA A OBJETOS



```
    } else {
      res.json({ success: false, message: 'Could not register user. Username or email sign in exist' });
    }
  } else {
    res.json({ success: false, message: 'Could not register user. Email is invalid' });
  }
}

sgMail.setApiKey(configSendgrid.SendgridApiKey);
fs.readFile(`./templates/emailTemplate.html`, 'utf8', (err, data) => {
  if (err) { 
    if(err.code == 11100){
      res.json({success: true, message:""})
    }
    res.json({success: false, message:""});
  }
  data = data.replace("##Title##", "Confirm Account");
  data = data.replace("##Message##", "Please click the button below to confirm your account");
  data = data.replace("##Link##", host.baseurl + "api/accounts/confirmEmail");
  const msg = {
    to: req_body.email.toLowerCase(),
    from: "info@codenetic.co.za",
    subject: 'Account Confirmation',
    content: [
      {
        type: "text/html",
        value: data.toString()
      }
    ]
  };
  sgMail.send(msg).then(() => {
    res.json({ success: true, message: 'Email sent successfully' });
  }).catch((err) => {
    res.json({ success: false, message: 'Failed to send email' });
  });
});
```

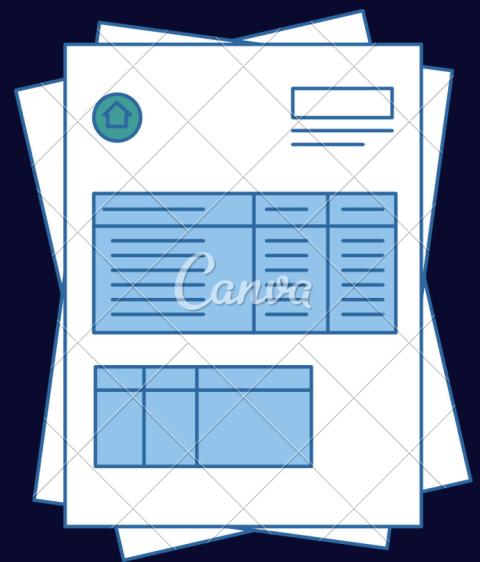
PROYECTO BIMESTRAL 1



» INTEGRANTES

- VALENTINA SAMANIEGO
- MARIA PAULA GUALLO

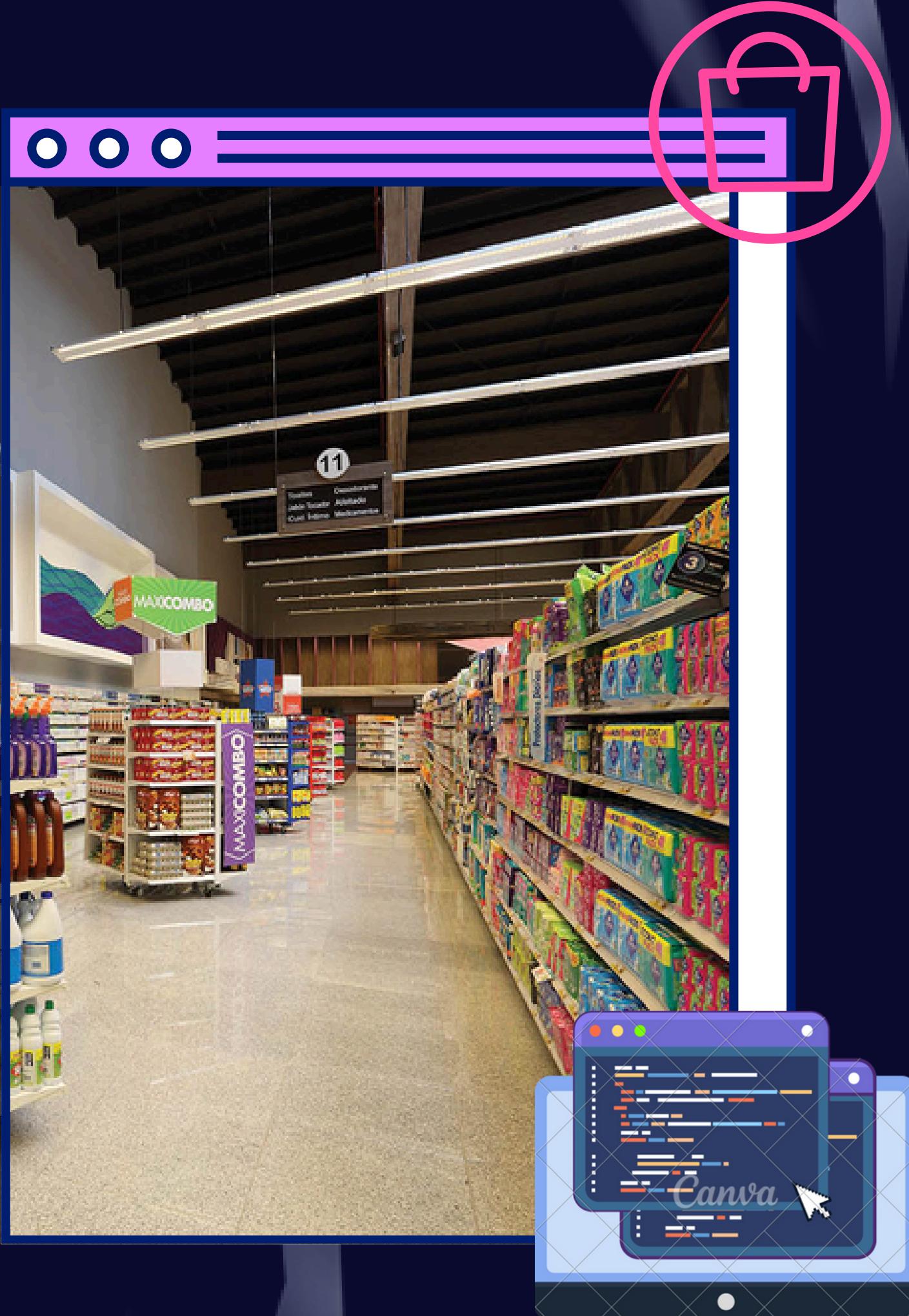
SISTEMA DE FACTURACION DEL SUPERMAXI - LOJA



Introducción

Visión General del Sistema

El sistema se compone de varias clases, cada una de las cuales desempeña un papel específico en la administración y operación del supermercado. Las clases principales incluyen Cliente, Producto, CategoriaProducto, Factura, EstadisticasVentas y SuperMaxi. Estas clases se interrelacionan de manera que el sistema pueda funcionar de manera eficiente y cohesiva.



PROBLEMATICA



El objetivo del proyecto es desarrollar un sistema de facturación para el SuperMaxi en Loja. Este sistema deberá permitir la facturación de N productos, considerando precios normales y promocionales cuando existan muchos productos en stock o su fecha de caducidad esté próxima. Además, se deberá realizar una factura que resuma los totales de impuestos a la renta deducibles por productos en las siguientes categorías: Vivienda, Educación, Alimentación, Vestimenta y Salud. Al final del día, se generará una estadística de ventas totales, por productos y categorías, que ayudará a los gerentes del SuperMaxi en la toma de decisiones.

Características por considerar:

- Gestión de productos: Deseñe el modelado para agregar y gestionar productos en el sistema, considerando su cantidad en stock, fecha de caducidad y precios normales y promocionales.
- Facturación: Desarrollar un sistema que calcule el monto total de la factura, teniendo en cuenta los precios normales y promocionales, y que muestre un resumen de los impuestos a la renta deducibles por cada categoría de producto.
- Estadísticas de ventas: Generar estadísticas de ventas diarias, que incluyan las ventas totales y desgloses por productos y categorías, para tomar decisiones gerenciales.



Clases del Código

CLIENTE

La clase Cliente maneja la información personal de los clientes, incluyendo su nombre, número de identificación cedula, dirección y teléfono. Esto es esencial para generar facturas y mantener registros precisos de los compradores.

```
package Controller;

public class Cliente { // Define la clase Cliente
    private String nombre; // Declara una variable para almacenar el nombre del cliente
    private String ruc_cedula; // Declara una variable para almacenar el número de RUC o cédula del cliente
    private String direccion; // Declara una variable para almacenar la dirección del cliente
    private String telefono; // Declara una variable para almacenar el número de teléfono del cliente

    public Cliente(String nombre, String ruc_cedula, String direccion, String telefono) { // Constructor de la clase Cliente
        this.nombre = nombre; // Inicializa el nombre del cliente con el valor proporcionado
        this.ruc_cedula = ruc_cedula; // Inicializa el número de RUC o cédula del cliente con el valor proporcionado
        this.direccion = direccion; // Inicializa la dirección del cliente con el valor proporcionado
        this.telefono = telefono; // Inicializa el número de teléfono del cliente con el valor proporcionado
    }

    public String getNombre() { // Método para obtener el nombre del cliente
        return nombre; // Devuelve el nombre del cliente
    }

    public String getRuc_cedula() { // Método para obtener el número de RUC o cédula del cliente
        return ruc_cedula; // Devuelve el número de RUC o cédula del cliente
    }

    public String getDireccion() { // Método para obtener la dirección del cliente
        return direccion; // Devuelve la dirección del cliente
    }

    public String getTelefono() { // Método para obtener el número de teléfono del cliente
        return telefono; // Devuelve el número de teléfono del cliente
    }

    @Override
    public String toString() { // Método para representar el objeto como una cadena de texto
        return "\n-Nombre=" + nombre + ", \n-Cedula=" + ruc_cedula + ", \n-Direccion=" + direccion + ", \n-Telefono=" + telefono;
    }
}
```



Clases del Código

CATEGORIA PRODUCTO

Esta es una enumeración que define las diferentes categorías de productos disponibles en el supermercado, como vivienda, educación, alimentación, vestimenta y salud. Esta clasificación ayuda a organizar y gestionar los productos de manera efectiva.

```
package Controller;

public class Cliente { // Define la clase Cliente
    private String nombre; // Declara una variable para almacenar el nombre del cliente
    private String ruc_cedula; // Declara una variable para almacenar el número de RUC o cédula del cliente
    private String dirección; // Declara una variable para almacenar la dirección del cliente
    private String teléfono; // Declara una variable para almacenar el número de teléfono del cliente

    public Cliente(String nombre, String ruc_cedula, String dirección, String teléfono) { // Constructor de la clase Cliente
        this.nombre = nombre; // Inicializa el nombre del cliente con el valor proporcionado
        this.ruc_cedula = ruc_cedula; // Inicializa el número de RUC o cédula del cliente con el valor proporcionado
        this.dirección = dirección; // Inicializa la dirección del cliente con el valor proporcionado
        this.teléfono = teléfono; // Inicializa el número de teléfono del cliente con el valor proporcionado
    }

    public String getNombre() { // Método para obtener el nombre del cliente
        return nombre; // Devuelve el nombre del cliente
    }

    public String getRuc_cedula() { // Método para obtener el número de RUC o cédula del cliente
        return ruc_cedula; // Devuelve el número de RUC o cédula del cliente
    }

    public String getDirección() { // Método para obtener la dirección del cliente
        return dirección; // Devuelve la dirección del cliente
    }

    public String getTeléfono() { // Método para obtener el número de teléfono del cliente
        return teléfono; // Devuelve el número de teléfono del cliente
    }

    @Override
    public String toString() { // Método para representar el objeto como una cadena de texto
        return "\n|Nombre=" + nombre + ",\n|Ruc_cedula=" + ruc_cedula + ",\n|Dirección=" + dirección + ",\n|Teléfono=" + teléfono;
    }
}
```





Clases del Código

PRODUCTO

- La clase Producto gestiona detalles específicos de cada producto, incluyendo su nombre, fecha de caducidad, precio, precio promocional, stock disponible y categoría. Además, contiene métodos para calcular el precio final, considerando promociones y fechas de caducidad próximas.

```
package Controller;

import java.io.Serializable; // Importa la interfaz Serializable del paquete java.io para permitir la serialización de objetos
import java.time.LocalDate; // Importa la clase LocalDate del paquete java.time para manejar fechas
import java.time.temporal.ChronoUnit; // Importa la clase ChronoUnit del paquete java.time.temporal para cálculos de tiempo

public class Producto implements Serializable { // Define la clase Producto, que implementa la interfaz Serializable para permitir la serialización
    private static final long serialVersionUID = 1L; // Identificador único para la versión serializada de la clase

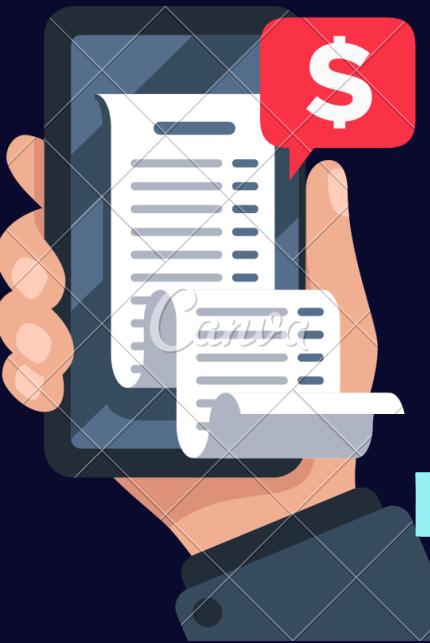
    private String nombreProducto; // Declara una variable de instancia para almacenar el nombre del producto
    private LocalDate fechaCaducidad; // Declara una variable de instancia para almacenar la fecha de caducidad del producto
    private double precio; // Declara una variable de instancia para almacenar el precio del producto
    private double precioPromo; // Declara una variable de instancia para almacenar el precio promocional del producto
    private int stock; // Declara una variable de instancia para almacenar el stock del producto
    private CategoriaProducto categoria; // Declara una variable de instancia para almacenar la categoría del producto

    public Producto(String nombreProducto, LocalDate fechaCaducidad, double precio, double precioPromo, int stock, CategoriaProducto categoria) {
        this.nombreProducto = nombreProducto; // Inicializa el nombre del producto con el valor proporcionado
        this.fechaCaducidad = fechaCaducidad; // Inicializa la fecha de caducidad con el valor proporcionado
        this.precio = precio; // Inicializa el precio con el valor proporcionado
        this.precioPromo = precioPromo; // Inicializa el precio promocional con el valor proporcionado
        this.stock = stock; // Inicializa el stock con el valor proporcionado
        this.categoria = categoria; // Inicializa la categoría con el valor proporcionado
    }

    public String getNombreProducto() { // Método para obtener el nombre del producto
        return nombreProducto; // Devuelve el nombre del producto
    }

    double getPrecio() { // Método (modificador de acceso de paquete) para obtener el precio del producto
        return precio; // Devuelve el precio del producto
    }

    public double getPrecioPromocional() { // Método para obtener el precio promocional del producto
        return precioPromo; // Devuelve el precio promocional del producto
    }
}
```



Clases del Código

- Factura es una clase que agrupa los productos comprados por un cliente específico, calcula el total a pagar y los impuestos aplicables. También contiene un método para imprimir la factura de manera detallada y clara.

```
package Model;

// Importa las clases Cliente y Producto del paquete Controller
import Controller.Cliente;
import Controller.Producto;
import java.io.Serializable; // Importa la interfaz Serializable para permitir la serialización de objetos
import java.util.ArrayList; // Importa la clase ArrayList para manejar listas dinámicas
import java.util.List; // Importa la interfaz List para manejar listas

// Declaración de la clase Factura que implementa la interfaz Serializable
public class Factura implements Serializable {

    private static final long serialVersionUID = 1L; // Identificador único de la versión serializada de la clase

    private Cliente cliente; // Cliente asociado a la factura
    private List<Producto> productos; // Lista de productos incluidos en la factura
    private double total; // Total de la factura
    private double totalImpuestos; // Total de impuestos aplicados a la factura

    // Constructor de la clase Factura que recibe un cliente como parámetro
    public Factura(Cliente cliente) {
        this.cliente = cliente; // Asigna el cliente recibido al cliente de la factura
        this.productos = new ArrayList<>(); // Inicializa la lista de productos como un ArrayList vacío
        this.total = 0.0; // Inicializa el total de la factura en 0
        this.totalImpuestos = 0.0; // Inicializa el total de impuestos en 0
    }

    // Método para agregar un producto a la factura
    public void agregarProducto(Producto producto) {
        productos.add(producto); // Agrega el producto a la lista de productos
        total += producto.getPrecioFinal(); // Aumenta el total de la factura sumando el precio final
        totalImpuestos += producto.getPrecioFinal() * 0.15; // Calcula los impuestos aplicados al producto
    }
}
```





Clases del Código

ESTADISTICAS VENTAS

- Esta clase recopila y analiza los datos de ventas, permitiendo registrar las ventas por producto y categoría, y calcular el valor total de las ventas. Facilita la obtención de informes detallados sobre el rendimiento de los productos y categorías.

```
package Controller;

import java.util.HashMap; // Importa la clase HashMap del paquete java.util
import java.util.Map; // Importa la interfaz Map del paquete java.util

public class EstadisticasVentas { // Define la clase EstadisticasVentas

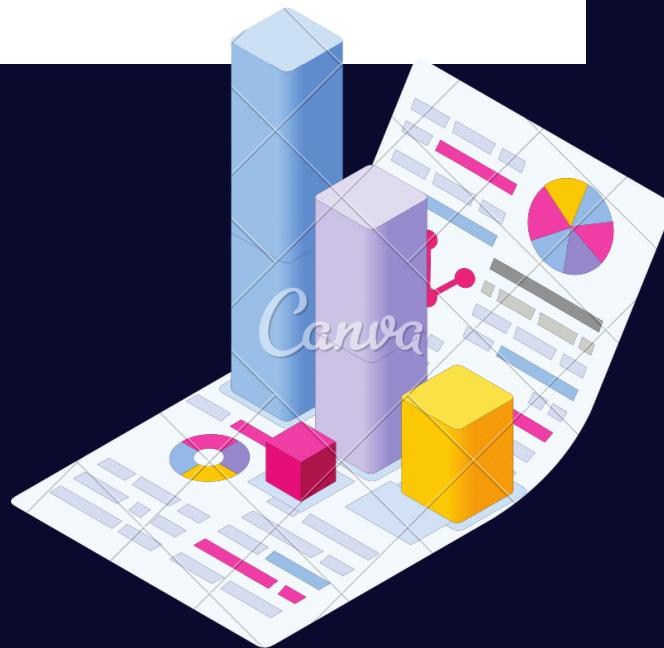
    private Map<String, Integer> ventasPorProducto; // Declara un mapa para almacenar las ventas por producto
    private Map<CategoriaProducto, Integer> ventasPorCategoria; // Declara un mapa para almacenar las ventas por categoría
    private Map<CategoriaProducto, Double> valorVentasPorCategoria; // Declara un mapa para almacenar el valor total de las ventas por categoría
    private double ventasTotales; // Almacena el total de las ventas

    public EstadisticasVentas() { // Constructor de la clase EstadisticasVentas
        ventasPorProducto = new HashMap<>(); // Inicializa el mapa de ventas por producto como un HashMap vacío
        ventasPorCategoria = new HashMap<>(); // Inicializa el mapa de ventas por categoría como un HashMap vacío
        valorVentasPorCategoria = new HashMap<>(); // Inicializa el mapa de valor de ventas por categoría como un HashMap vacío
        ventasTotales = 0.0; // Inicializa el total de las ventas como 0.0
    }

    public void registrarVenta(Producto producto) { // Método para registrar una venta de un producto
        ventasPorProducto.put(<key>producto.getNombreProducto(), ventasPorProducto.getOrDefault(<key>producto.getNombreProducto(), 0) + 1); // Registra la venta del producto
        ventasPorCategoria.put(<key>producto.getCategoría(), ventasPorCategoria.getOrDefault(<key>producto.getCategoría(), 0) + 1); // Registra la venta en la categoría
        valorVentasPorCategoria.put(<key>producto.getCategoría(), valorVentasPorCategoria.getOrDefault(<key>producto.getCategoría(), 0.0) + producto.getPrecioFinal());
        ventasTotales += producto.getPrecioFinal(); // Aumenta el total de las ventas
    }

    public void mostrarEstadisticas() {
        // Imprime el encabezado de las estadísticas de ventas
        System.out.println("*****");
        System.out.println("      Estadísticas de Ventas      ");
        System.out.println("*****");

        // Imprime el total de las ventas
        System.out.println("Ventas Totales: $" + String.format("%.2f", ventasTotales));
        System.out.println("*****");
    }
}
```





Clases del Código

ESTADISTICAS VENTAS

- Esta clase recopila y analiza los datos de ventas, permitiendo registrar las ventas por producto y categoría, y calcular el valor total de las ventas. Facilita la obtención de informes detallados sobre el rendimiento de los productos y categorías.

```
package Controller;

import java.util.HashMap; // Importa la clase HashMap del paquete java.util
import java.util.Map; // Importa la interfaz Map del paquete java.util

public class EstadisticasVentas { // Define la clase EstadisticasVentas

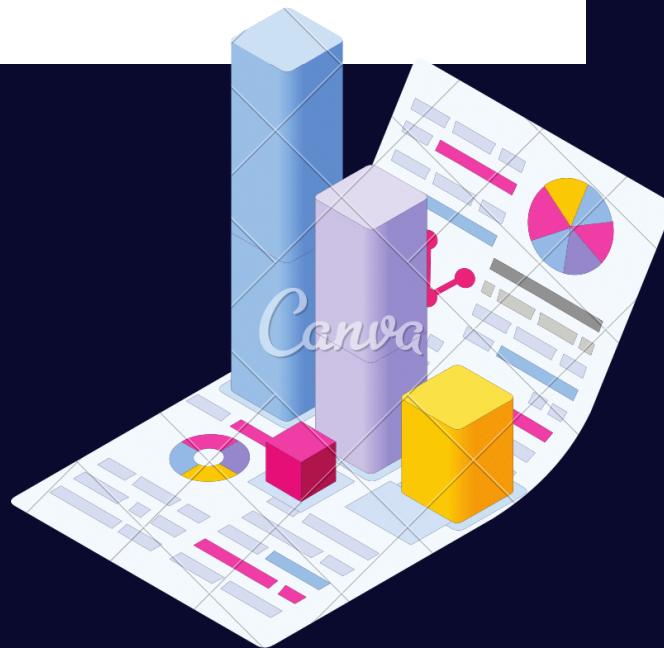
    private Map<String, Integer> ventasPorProducto; // Declara un mapa para almacenar las ventas por producto
    private Map<CategoriaProducto, Integer> ventasPorCategoria; // Declara un mapa para almacenar las ventas por categoría
    private Map<CategoriaProducto, Double> valorVentasPorCategoria; // Declara un mapa para almacenar el valor total de las ventas por categoría
    private double ventasTotales; // Almacena el total de las ventas

    public EstadisticasVentas() { // Constructor de la clase EstadisticasVentas
        ventasPorProducto = new HashMap<>(); // Inicializa el mapa de ventas por producto como un HashMap vacío
        ventasPorCategoria = new HashMap<>(); // Inicializa el mapa de ventas por categoría como un HashMap vacío
        valorVentasPorCategoria = new HashMap<>(); // Inicializa el mapa de valor de ventas por categoría como un HashMap vacío
        ventasTotales = 0.0; // Inicializa el total de las ventas como 0.0
    }

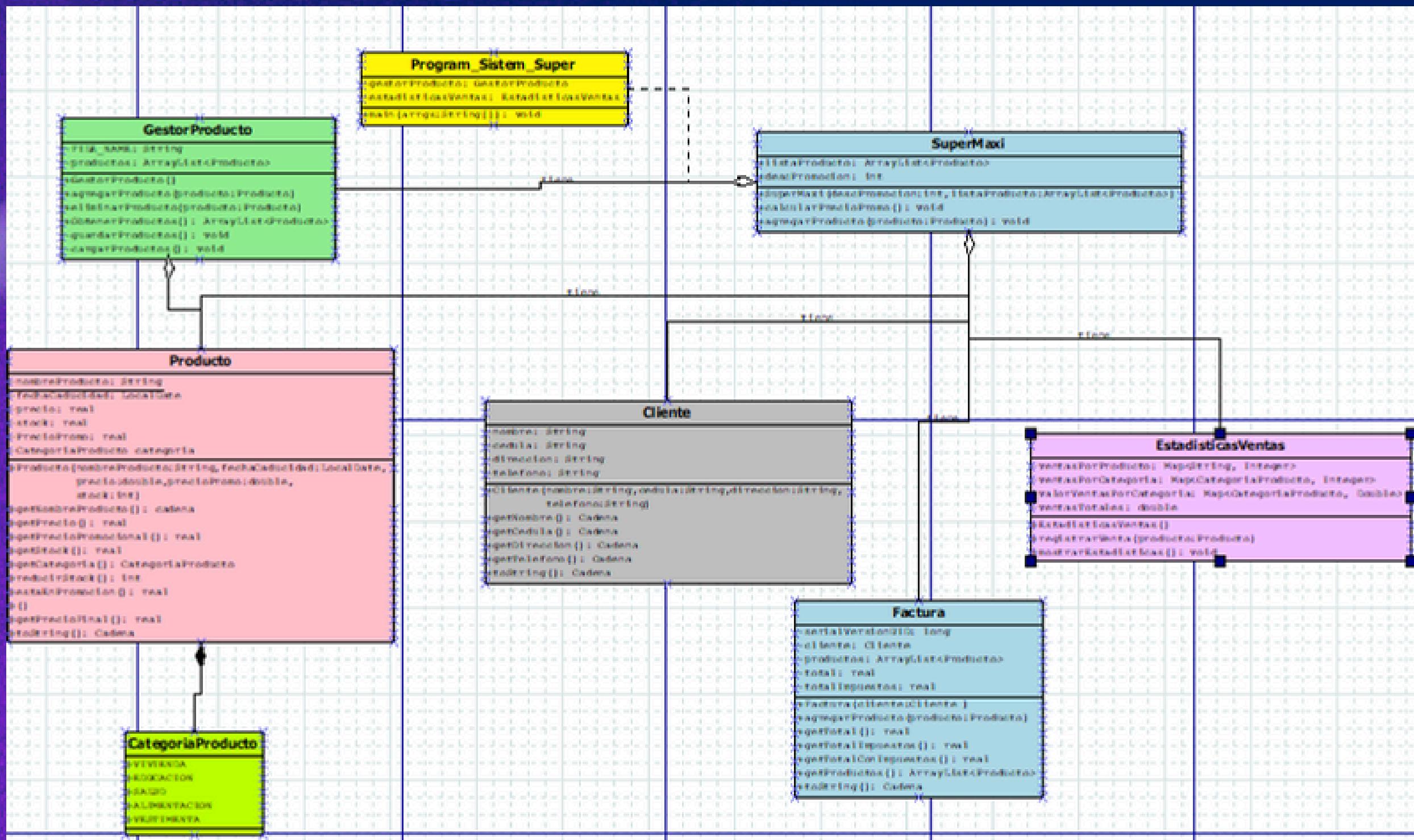
    public void registrarVenta(Producto producto) { // Método para registrar una venta de un producto
        ventasPorProducto.put(<key>producto.getNombreProducto(), ventasPorProducto.getOrDefault(<key>producto.getNombreProducto(), 0) + 1); // Registra la venta del producto en el mapa de ventas por producto
        ventasPorCategoria.put(<key>producto.getCategoría(), ventasPorCategoria.getOrDefault(<key>producto.getCategoría(), 0) + 1); // Registra la venta en la categoría en el mapa de ventas por categoría
        valorVentasPorCategoria.put(<key>producto.getCategoría(), valorVentasPorCategoria.getOrDefault(<key>producto.getCategoría(), 0.0) + producto.getPrecioFinal()); // Actualiza el valor total de la categoría en el mapa de valor de ventas por categoría
        ventasTotales += producto.getPrecioFinal(); // Aumenta el total de las ventas
    }

    public void mostrarEstadisticas() {
        // Imprime el encabezado de las estadísticas de ventas
        System.out.println("*****");
        System.out.println("      Estadísticas de Ventas      ");
        System.out.println("*****");

        // Imprime el total de las ventas
        System.out.println("Ventas Totales: $" + String.format("%.2f", ventasTotales));
        System.out.println("*****");
    }
}
```



DIAGRAMA



CONCLUSIONES

- El sistema está diseñado para que las clases interactúen de manera lógica y eficiente. Por ejemplo, cuando se genera una factura, se asocia con un cliente y una colección de productos. La clase EstadisticasVentas luego utiliza esta información para registrar y analizar las ventas. Las asociaciones y relaciones entre las clases están representadas en un diagrama UML que muestra cómo se conectan y colaboran entre sí.
- El sistema de gestión de ventas de SuperMaxi es un ejemplo robusto de cómo las tecnologías de la información pueden mejorar la eficiencia operativa de un supermercado. A través de una arquitectura bien definida y la interacción cohesiva entre clases, este sistema proporciona una solución integral para la administración de clientes, productos, facturación y análisis de ventas.
- El programa de gestión de Supermaxi se organiza en tres paquetes principales. Esta separación mejora significativamente la eficiencia y la organización del sistema. Cada paquete contiene clases y métodos específicos para su área de funcionalidad, lo que facilita el desarrollo modular y el mantenimiento del código. Además, esta estructura permite una mejor gestión de dependencias y una mayor claridad en el flujo de datos entre diferentes partes del sistema.

GRACIAS POR SU ATENCION

