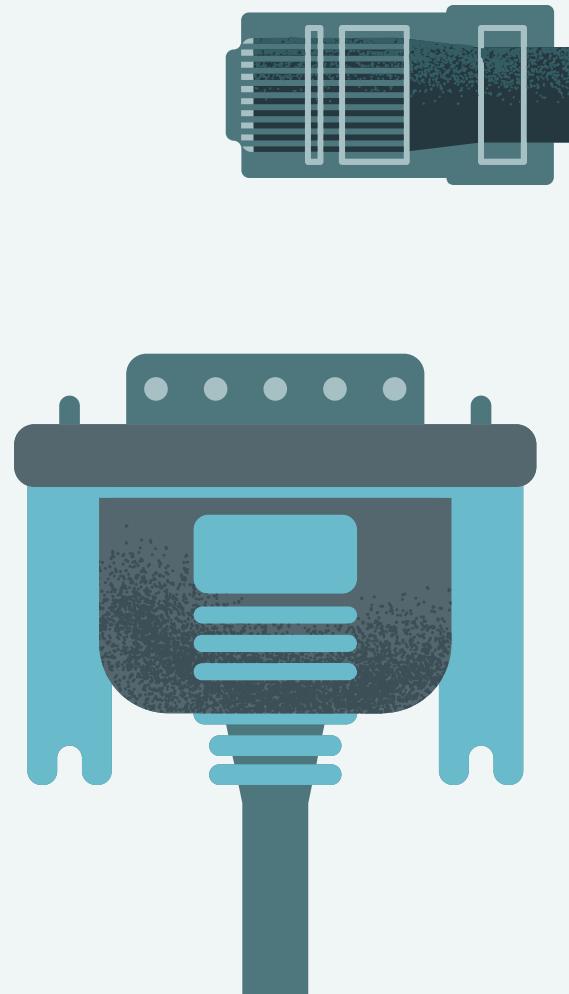
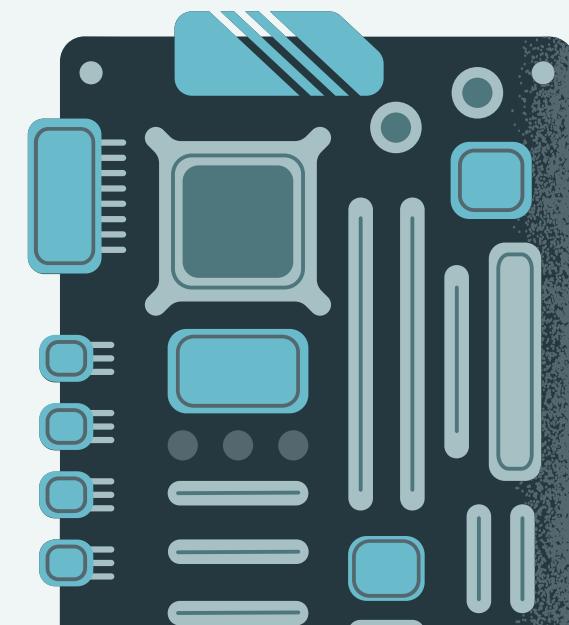
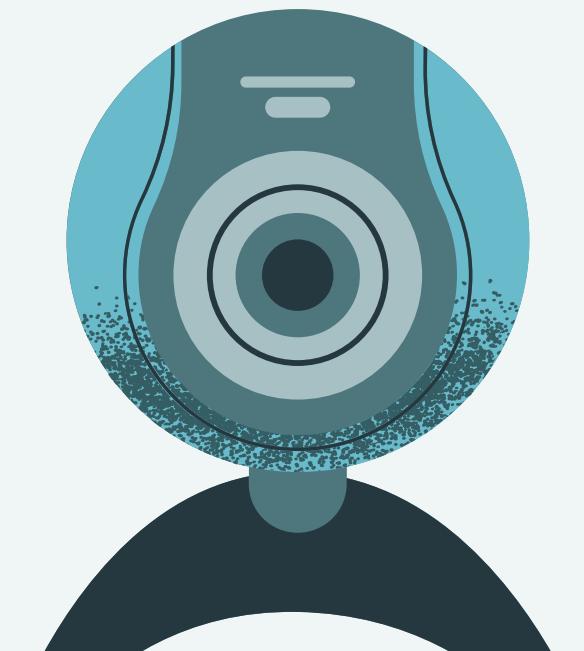
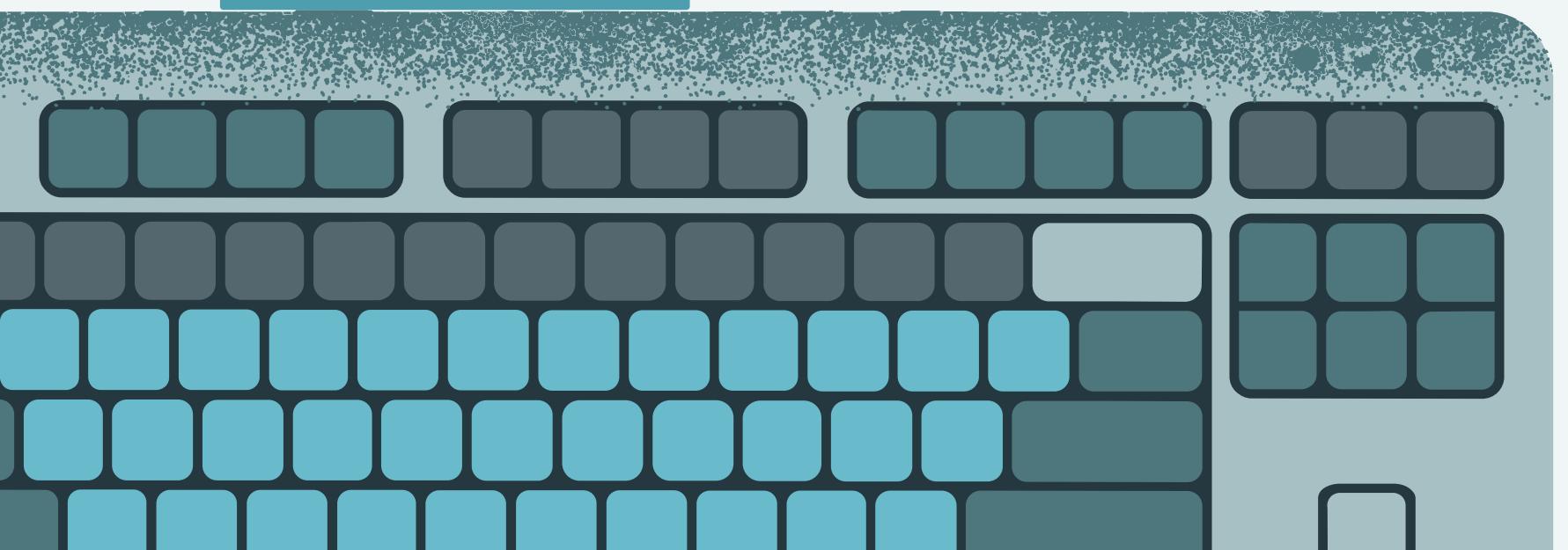


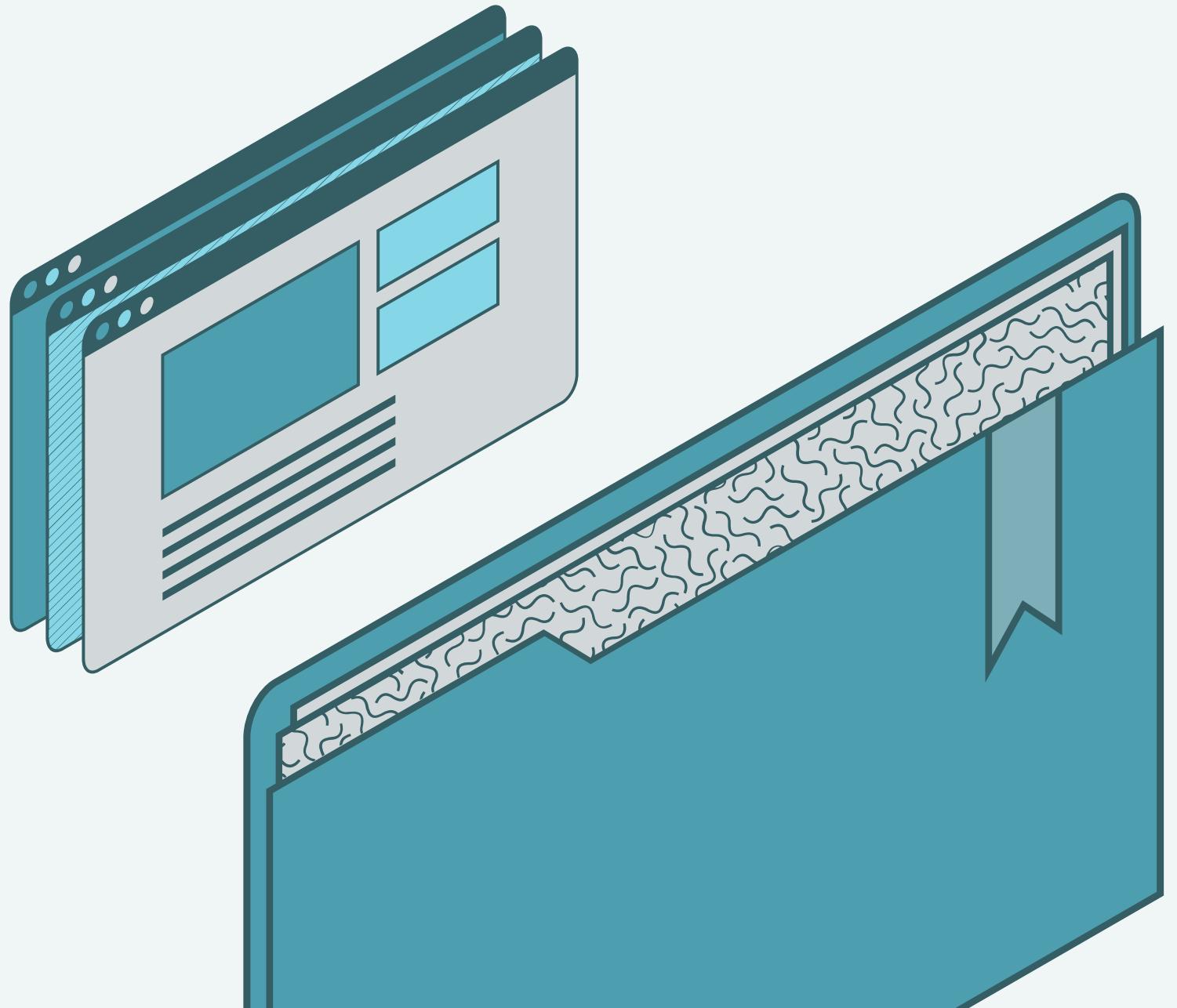


PROYECTO JAVA

Juan David Delgado Burbano
Nicolas Martinez
Camila Ariza



íNDICE



01. Introducción

02. predicción de precios c++

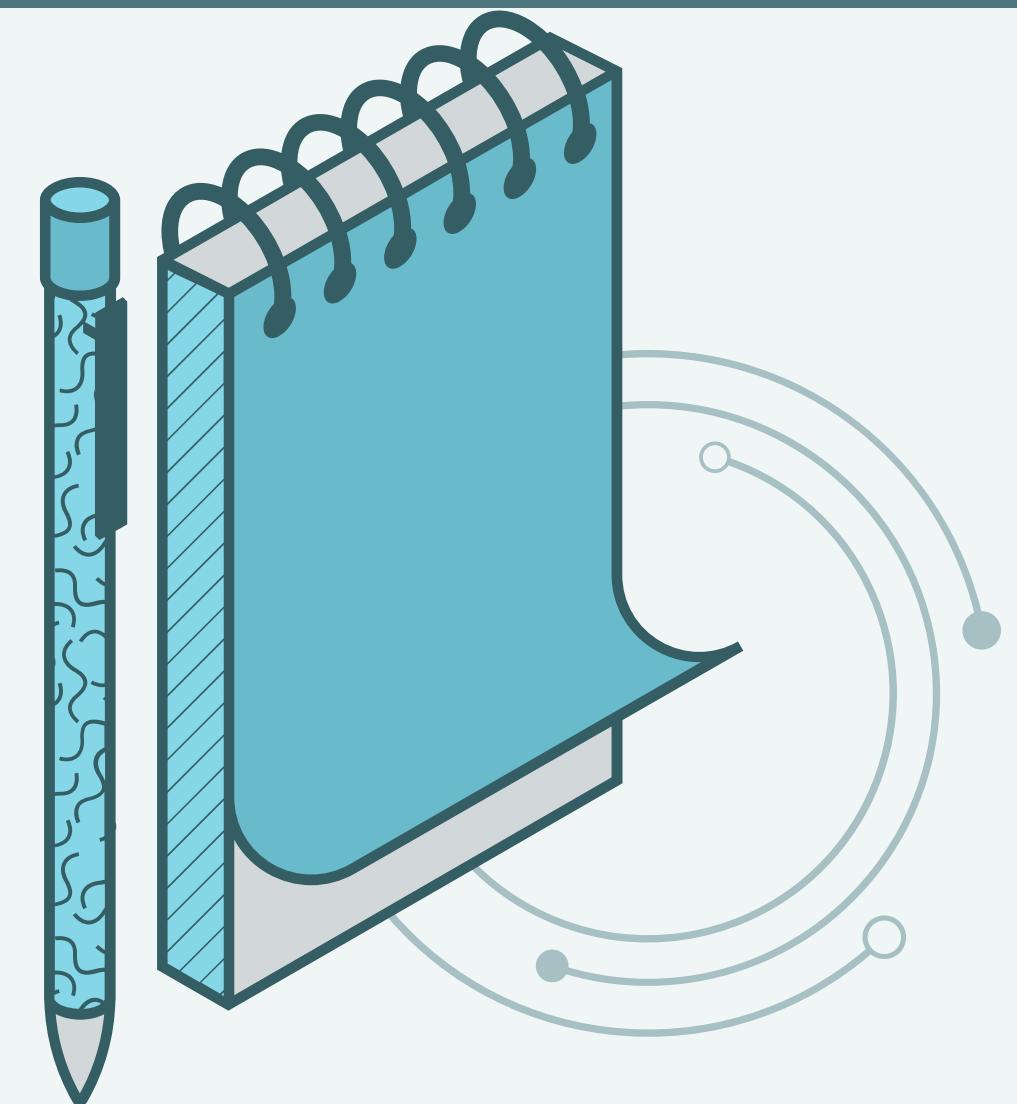
03. sistema de gestión de productos

04. Consola

05. conclusiones

INTRODUCCIÓN

- El proyecto de Java se trata de integrar la regresión lineal utilizada en proyecto anteriores, para una interfaz de gestión de productos con java, en donde se predijera el precio de productos en base a ventas anteriores, (cantidades / precio).



PREDICCION DE PRECIOS C++

2

PREDICCION:

- originalmente la regresión lineal en c++, se veía de la siguiente manera:

```
using namespace std;

void regresion(todo* arr, int n) {
    // Crear arreglos dinámicos para almacenar las habitaciones y precios
    float* habi = new float[n];
    float* preci = new float[n];

    for (int r = 0; r < n; r++) {
        habi[r] = arr[r].habitaciones;
        preci[r] = arr[r].precio;
    }

    // Normalizar los datos (entre 0 y 1)
    float max_habi = habi[0], max_preci = preci[0];
    float min_habi = habi[0], min_preci = preci[0];

    for (int i = 1; i < n; ++i) {
        if (habi[i] > max_habi) max_habi = habi[i];
        if (precii[i] > max_preci) max_preci = preci[i];
        if (habi[i] < min_habi) min_habi = habi[i];
        if (precii[i] < min_preci) min_preci = preci[i];
    }
}
```

```
for (int i = 0; i < n; ++i) {
    habi[i] = (habi[i] - min_habi) / (max_habi - min_habi); // Normalización
    preci[i] = (preci[i] - min_preci) / (max_preci - min_preci);
}

// Parámetros iniciales de la regresión
float m = 0.0; // Pendiente
float b = 0.0; // Intercepto
float alpha = 0.001; // Tasa de aprendizaje ajustada
float mse = 0.0; // Error cuadrático medio
float mse_objetivo = 0.001; // Umbral para detener el descenso de gradiente
int max_iteraciones = 20000; // Límite máximo de épocas
int iteraciones = 0;

// Descenso de gradiente con criterio de convergencia basado en MSE
do {
    float gradiente_m = 0.0;
    float gradiente_b = 0.0;
    mse = 0.0;

    for (int i = 0; i < n; ++i) {
        float prediccion = m * habi[i] + b;
        float error = preci[i] - prediccion;
        ---
```

- Donde se mandaba un arreglo de tipo todo, y después se creaban dos arreglos dinámicos en base al numero de datos.

PREDICCION:

- sin embargo esta regresión lineal era muy complicada, y realizaba demasiadas interacciones para una predicción de precios con pocas variables, por lo que se lo ajusto:

```
public class RegresionLineal {  
    private double[] x; // Cantidad vendidas  
    private double[] y; // Precios de productos vendidos  
    private double m; // Pendiente  
    private double b; // Intercepto  
  
    public RegresionLineal(double[] x, double[] y) {  
        this.x = x;  
        this.y = y;  
        calcularParametros();  
    }  
  
    private void calcularParametros() {  
        int n = x.length;  
        double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;  
  
        for (int i = 0; i < n; i++) {  
            sumX += x[i];  
            sumY += y[i];  
            sumXY += x[i] * y[i];  
            sumX2 += x[i] * x[i];  
        }  
    }  
}
```

```
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36
```

```
    // Calculando la pendiente (m) y el intercepto (b)  
    m = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);  
    b = (sumY - m * sumX) / n;  
}  
  
public double predecir(double nuevaCantidad) {  
    return m * nuevaCantidad + b;  
}  
  
public void mostrarParametros() {  
    System.out.println("Pendiente (m): " + m + ", Intercepto  
(b): " + b);  
}  
}
```

PREDICCION:

- Atributos de la clase
- double[] x: Es un arreglo que contiene los valores de la variable independiente (en este caso, las cantidades vendidas).
- double[] y: Es un arreglo que contiene los valores de la variable dependiente (los precios de los productos vendidos).
- double m: La pendiente de la recta, que determina cuánto cambia el precio por cada cambio en la cantidad.
- double b: El intercepto de la recta, que es el valor de y (precio) cuando x (cantidad) es 0.

PREDICCION:

Constructor: public RegresionLineal(double[] x, double[] y)

Este constructor recibe dos arreglos como parámetros: uno para las cantidades (x) y otro para los precios (y). Luego, llama al método calcular Parámetros para calcular la pendiente m y el intercepto b.

Método calcular Parámetros()

Este método es el que realiza el cálculo de la pendiente y el intercepto de la línea de mejor ajuste, utilizando la fórmula de mínimos cuadrados. Aquí te explico cada paso:

1. Sumar todos los valores:

- **sumX**: Suma todos los valores del arreglo x (cantidades vendidas).
- **sumY**: Suma todos los valores del arreglo y (precios de los productos).
- **sumXY**: Suma los productos de cada valor de x con su correspondiente valor de y (es decir, para cada punto $(x[i], y[i])$, calcula $x[i] * y[i]$ y suma esos valores).
- **sumX2**: Suma los cuadrados de cada valor en x (es decir, para cada $x[i]$, calcula $x[i] * x[i]$ y suma esos valores).

PRÉDICCION FINAL:

```
public double predecir(double nuevaCantidad) {  
    return m * nuevaCantidad + b;  
}
```

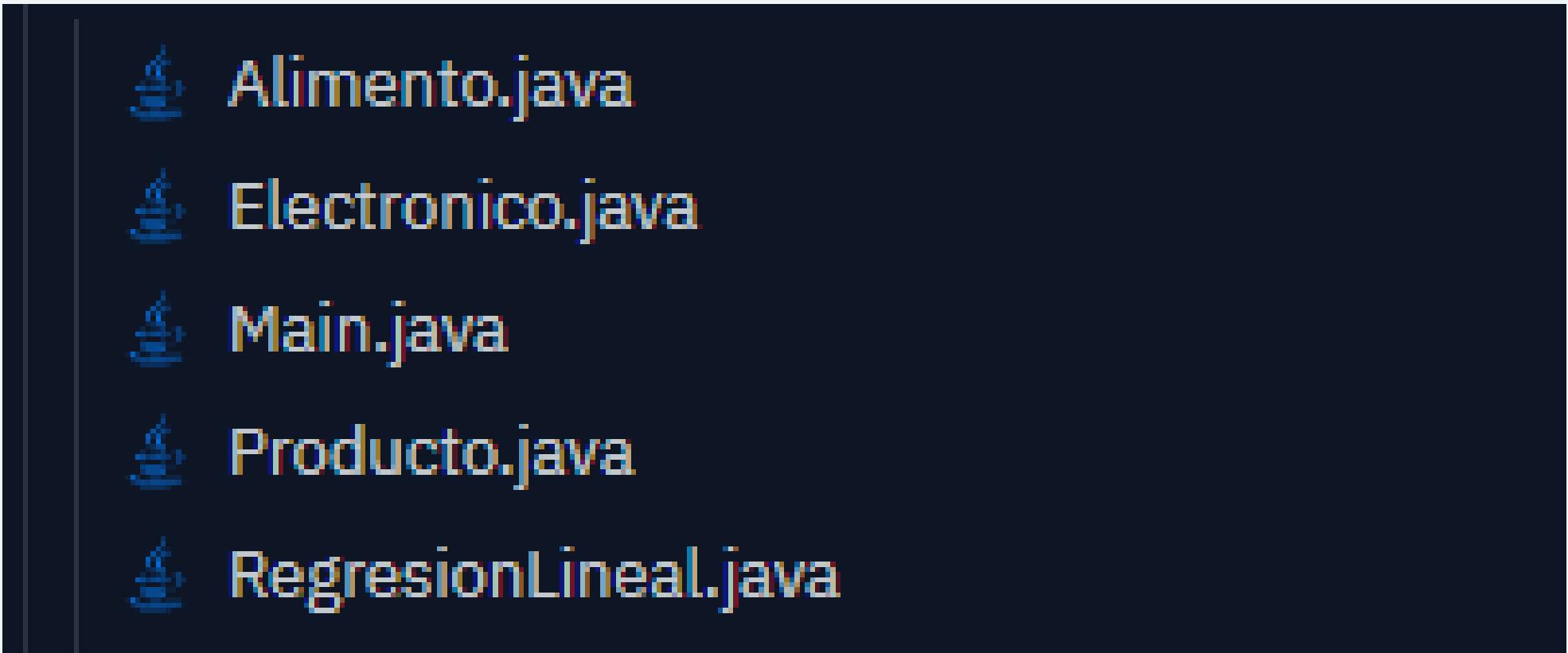
Con la ecuación de la pendiente, y la cantidad a predecir, obtenemos el precio.

SISTEMA DE GESTIÓN DE PRODUCTOS

3



FORMATO



- Alimento.java
- Electronico.java
- Main.java
- Producto.java
- RegresionLineal.java

PRODUCTOS:

la clase producto, tiene atributos como nombre, cantidad, precio

ES una Clase abstracta (abstract class):

La palabra clave abstract indica que la clase no se puede instanciar directamente, La clase sirve como base para otras clases que heredarán de ella. (alimento / electronico)

```
1  public abstract class Producto {  
2      protected String nombre;  
3      protected int cantidad;  
4      protected double precio;  
5  
6      public Producto(String nombre, int cantidad, double precio) {  
7          this.nombre = nombre;  
8          this.cantidad = cantidad;  
9          this.precio = precio;  
10     }  
11  
12     public String getNombre() {  
13         return nombre;  
14     }  
15  
16     public int getCantidad() {  
17         return cantidad;  
18     }  
19  
20     public void setCantidad(int cantidad) {  
21         this.cantidad = cantidad;  
22     }  
23  
24     public double getPrecio() {  
25         return precio;  
26     }  
27  
28     public void setPrecio(double precio) {  
29         this.precio = precio;  
30     }  
31  
32     public abstract void mostrarDetalles();  
33 }
```

SUB CLASE:

ALIMENTO:

Este código define una clase llamada Alimento que hereda de la clase abstracta Producto. lo que significa que hereda los atributos y métodos de la clase Producto.

- **mesesCaducidad:** Este atributo específico ,almacena los meses restantes antes de que el producto caduque.
- **Estadovendido:** (otro atributo específico)

```
public class Alimento extends Producto {  
    private int mesesCaducidad;  
    private boolean estaVendido;  
  
    public Alimento(String nombre, int cantidad, double precio, int  
mesesCaducidad, boolean estaVendido) {  
        super(nombre, cantidad, precio);  
        this.mesesCaducidad = mesesCaducidad;  
        this.estadovendido = estaVendido;  
    }  
  
    public int getMesesCaducidad() {  
        return mesesCaducidad;  
    }  
  
    public void setMesesCaducidad(int mesesCaducidad) {  
        this.mesesCaducidad = mesesCaducidad;  
    }  
  
    public boolean isEstadovendido() {  
        return estadovendido;  
    }  
  
    public void setEstadovendido(boolean estaVendido) {  
        this.estadovendido = estaVendido;  
    }  
}
```

SUB CLASE:

ALIMENTO:

El **constructor** de Alimento recibe varios parámetros: nombre, cantidad, precio, mesesCaducidad y estado.

Utiliza la palabra clave **super** para llamar al constructor de la clase padre (Producto), inicializando los atributos heredados (nombre, cantidad, precio).

Luego, inicializa los atributos específicos de la clase Alimento, como mesesCaducidad.

Métodos get y set para mesesCaducidad y tambien para estado, Estos métodos permiten acceder y modificar los valores de los atributos específicos de Alimento.

```
public Alimento(String nombre, int cantidad, double precio, int  
mesesCaducidad, boolean estaVendido) {  
    super(nombre, cantidad, precio);  
    this.mesesCaducidad = mesesCaducidad;  
    this.estaVendido = estaVendido;  
}
```

```
public int getMesesCaducidad() {  
    return mesesCaducidad;  
}  
  
public void setMesesCaducidad(int mesesCaducidad) {  
    this.mesesCaducidad = mesesCaducidad;  
}
```

SUB CLASE:

ELECTRONICO:

Este código define una clase llamado electronico que hereda de la clase abstracta Producto. lo que significa que hereda los atributos y métodos de la clase Producto. (nombre, cantidad, precio)

Atributos específicos:

- estado: es un atributo específico.

```
1 public class Electronico extends Producto {  
2     private String estado;  
3  
4     public Electronico(String nombre, int cantidad, double precio,  
5                         String estado) {  
6         super(nombre, cantidad, precio);  
7         this.estado = estado;  
8     }  
9  
10    public String getEstado() {  
11        return estado;  
12    }  
13  
14    public void setEstado(String estado) {  
15        this.estado = estado;  
16    }  
17  
18    @Override  
19    public void mostrarDetalles() {  
20        System.out.println("Electrónico: " + nombre + ", Cantidad: " +  
21                           cantidad + ", Precio: " + precio +  
22                           ", Estado: " + estado);  
23    }  
24}
```

SUB CLASE: ELÉCTRÓNICO:

El **constructor** de `electrónico` recibe varios parámetros: nombre, cantidad, precio, estado.

Utiliza la palabra clave **super** para llamar al constructor de la clase padre (`Producto`), inicializando los atributos heredados (nombre, cantidad, precio).

Luego, inicializa los atributos específicos de la clase `electrónico`.

Métodos get y set para Estado:

Estos métodos permiten acceder y modificar los valores de los atributos específicos de `electrónico`.

```
1 public class Electronico extends Producto {  
2     private String estado;  
3  
4     public Electronico(String nombre, int cantidad, double precio,  
5                         String estado) {  
6         super(nombre, cantidad, precio);  
7         this.estado = estado;  
8     }  
9  
10    public String getEstado() {  
11        return estado;  
12    }  
13  
14    public void setEstado(String estado) {  
15        this.estado = estado;  
16    }  
17  
18    @Override  
19    public void mostrarDetalles() {  
20        System.out.println("Electrónico: " + nombre + ", Cantidad: " +  
21                           cantidad + ", Precio: " + precio +  
22                           ", Estado: " + estado);  
23    }  
24}
```

MENÚ:

Sistema de Gestión de Productos

1. Agregar Producto
2. Vender Producto
3. Mostrar Productos
4. Mostrar Productos Vendidos
5. Realizar Predicción de Precio (Regresión Lineal)
6. Salir

Seleccione una opción:

- Desde el menú de la consola se da las opciones para realizar cada operación.

MÉNU AGREGAR PRODUCTO:

Seleccione una opción: 1

Ingresar el tipo de producto (1: Alimento, 2: Electrónico):

1

Ingresar el nombre del producto: pera

Ingresar la cantidad: 2

Ingresar el precio: 200

Ingresar los meses de caducidad: 4

- se puede llenar cada atributo de la clase producto, según que subclase se escoja (alimento/electrónico).
- vende el producto seleccionado, eliminándolo de la lista de productos.

Seleccione una opción: 2

Ingresar el nombre del producto que desea vender: ps2

Producto vendido: ps2

MÉNU AGREGAR PRODUCTO:

Seleccione una opción: 3

Productos disponibles:

Alimento: pera, Cantidad: 2, Precio: 200.0, Meses de Caducidad: 4, Vendido: No

- muestra la lista de productos, en consola:

Seleccione una opción: 4

Productos vendidos:

Electrónico: ps2, Cantidad: 1, Precio: 2.0, Estado: nuevo

- muestra la lista de productos vendidos, en consola:

Sistema de Gestión de Productos

MAIN.CPP

```
1 import java.util.ArrayList;
2 import java.util.Scanner;
3
4 public class Main {
5     // Listas para almacenar productos y productos vendidos
6     private static ArrayList<Producto> productos = new ArrayList<>();
7     private static ArrayList<Producto> productosVendidos = new
ArrayList<>();
8
9     public static void main(String[] args) {
10         Scanner scanner = new Scanner(System.in);
11
12         int opcion;
13         do {
14             System.out.println("\nSistema de Gestión de Productos");
15             System.out.println("1. Agregar Producto");
16             System.out.println("2. Vender Producto");
17             System.out.println("3. Mostrar Productos");
18             System.out.println("4. Mostrar Productos Vendidos");
19             System.out.println("5. Realizar Predicción de Precio
(Regresión Lineal)");
20             System.out.println("6. Salir");
21             System.out.print("Seleccione una opción: ");
22
23         } while (opcion != 6);
24
25         switch (opcion) {
26             case 1:
27                 agregarProducto(scanner);
28                 break;
29             case 2:
30                 venderProducto(scanner);
31                 break;
32             case 3:
33                 mostrarProductos();
34                 break;
35             case 4:
36                 mostrarProductosVendidos();
37                 break;
38             case 5:
39                 realizarRegresionLineal();
40                 break;
41         }
42     }
43 }
```

```
        System.out.println("5. Realizar Predicción de Precio
(Regresión Lineal)");
        System.out.println("6. Salir");
        System.out.print("Seleccione una opción: ");
        opcion = scanner.nextInt();

        switch (opcion) {
            case 1:
                agregarProducto(scanner);
                break;
            case 2:
                venderProducto(scanner);
                break;
            case 3:
                mostrarProductos();
                break;
            case 4:
                mostrarProductosVendidos();
                break;
            case 5:
                realizarRegresionLineal();
                break;
            case 6:
                System.out.println("Sistema de Gestión de Productos Finalizado.");
                break;
        }
    }
}
```

se hace uso de **arraylist** para arreglo “dinamico”

ArrayList<Producto> productos: almacena los productos disponibles.

ArrayList<Producto> productosVendidos: almacena los productos que ya han sido vendidos.

MAIN.CPP

```
// Método para agregar productos
private static void agregarProducto(Scanner scanner) {
    System.out.println("Ingrese el tipo de producto (1: Alimento,
2: Electrónico): ");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpiar buffer

    System.out.print("Ingrese el nombre del producto: ");
    String nombre = scanner.nextLine();

    System.out.print("Ingrese la cantidad: ");
    int cantidad = scanner.nextInt();

    System.out.print("Ingrese el precio: ");
    double precio = scanner.nextDouble();

    if (tipo == 1) {
        System.out.print("Ingrese los meses de caducidad: ");
        int mesesCaducidad = scanner.nextInt();

        productos.add(new Alimento(nombre, cantidad, precio,
mesesCaducidad, false));
    } else if (tipo == 2) {
```

Método agregarProducto

Este método permite agregar productos de dos tipos: Alimentos o Electrónicos.

Pide al usuario que seleccione el tipo de producto, el nombre, la cantidad, el precio y otros detalles específicos para cada tipo de producto (meses de caducidad para alimentos, estado para electrónicos).

Luego, el nuevo producto se agrega a la lista productos.

MAIN.CPP

```
// Método para agregar productos
private static void agregarProducto(Scanner scanner) {
    System.out.println("Ingrese el tipo de producto (1: Alimento,
2: Electrónico): ");
    int tipo = scanner.nextInt();
    scanner.nextLine(); // Limpiar buffer

    System.out.print("Ingrese el nombre del producto: ");
    String nombre = scanner.nextLine();

    System.out.print("Ingrese la cantidad: ");
    int cantidad = scanner.nextInt();

    System.out.print("Ingrese el precio: ");
    double precio = scanner.nextDouble();

    if (tipo == 1) {
        System.out.print("Ingrese los meses de caducidad: ");
        int mesesCaducidad = scanner.nextInt();

        productos.add(new Alimento(nombre, cantidad, precio,
mesesCaducidad, false));
    } else if (tipo == 2) {
        scanner.nextLine(); // Limpiar buffer
        System.out.print("Ingrese el estado del electrónico: ");
        String estado = scanner.nextLine();

        productos.add(new Electronico(nombre, cantidad, precio,
estado));
    } else {
        System.out.println("Tipo de producto inválido.");
    }
}
```

```
3
4     if (tipo == 1) {
5         System.out.print("Ingrese los meses de caducidad: ");
6         int mesesCaducidad = scanner.nextInt();
7
8             productos.add(new Alimento(nombre, cantidad, precio,
mesesCaducidad, false));
9     } else if (tipo == 2) {
10        scanner.nextLine(); // Limpiar buffer
11        System.out.print("Ingrese el estado del electrónico: ");
12        String estado = scanner.nextLine();
13
14            productos.add(new Electronico(nombre, cantidad, precio,
estado));
15     } else {
16         System.out.println("Tipo de producto inválido.");
17     }
18 }
```

Método agregarProducto

Este método permite agregar productos de dos tipos: Alimentos o Electrónicos.

Pide al usuario que seleccione el tipo de producto, el nombre, la cantidad, el precio y otros detalles específicos para cada tipo de producto (meses de caducidad para alimentos, estado para electrónicos).

Luego, el nuevo producto se agrega a la lista productos.

MAIN.CPP

```
// Método para vender productos
private static void venderProducto(Scanner scanner) {
    scanner.nextLine(); // Limpiar buffer
    System.out.print("Ingrese el nombre del producto que desea
vender: ");
    String nombre = scanner.nextLine();

    boolean encontrado = false;
    for (Producto producto : productos) {
        if (producto.getNombre().equalsIgnoreCase(nombre)) {
            if (producto instanceof Alimento) {
                ((Alimento) producto).setEstaVendido(true);
            }
            productosVendidos.add(producto);
            productos.remove(producto);
            System.out.println("Producto vendido: " + nombre);
            encontrado = true;
            break;
        }
    }
}
```

Método venderProducto

Este método se encarga de vender un producto.

Solicita al usuario que ingrese el nombre del producto que quiere vender.

Busca el producto en la lista de productos disponibles (`productos`), y si lo encuentra:
Si es un alimento, marca el producto como "vendido".

El producto se agrega a la lista **productosVendidos** y se elimina de la lista de productos disponibles.

Si el producto no se encuentra, se muestra un mensaje de error.

MAIN.CPP

```
// Método para vender productos
private static void venderProducto(Scanner scanner) {
    scanner.nextLine(); // Limpiar buffer
    System.out.print("Ingrese el nombre del producto que desea
vender: ");
    String nombre = scanner.nextLine();

    boolean encontrado = false;
    for (Producto producto : productos) {
        if (producto.getNombre().equalsIgnoreCase(nombre)) {
            if (producto instanceof Alimento) {
                ((Alimento) producto).setEstaVendido(true);
            }
            productosVendidos.add(producto);
            productos.remove(producto);
            System.out.println("Producto vendido: " + nombre);
            encontrado = true;
            break;
        }
    }
}
```

Método venderProducto

Este método se encarga de vender un producto.

Solicita al usuario que ingrese el nombre del producto que quiere vender.

Busca el producto en la lista de productos disponibles (`productos`), y si lo encuentra:
Si es un alimento, marca el producto como "vendido".

El producto se agrega a la lista **productosVendidos** y se elimina de la lista de productos disponibles.

Si el producto no se encuentra, se muestra un mensaje de error.

MAIN.CPP

```
// Método para mostrar productos disponibles
private static void mostrarProductos() {
    System.out.println("\nProductos disponibles:");
    for (Producto producto : productos) {
        producto.mostrarDetalles();
    }
}

// Método para mostrar productos vendidos
private static void mostrarProductosVendidos() {
    System.out.println("\nProductos vendidos:");
    for (Producto producto : productosVendidos) {
        producto.mostrarDetalles();
    }
}
```

Metodos de impresion de listas:

MAIN.CPP

```
// Método para mostrar productos disponibles
private static void mostrarProductos() {
    System.out.println("\nProductos disponibles:");
    for (Producto producto : productos) {
        producto.mostrarDetalles();
    }
}

// Método para mostrar productos vendidos
private static void mostrarProductosVendidos() {
    System.out.println("\nProductos vendidos:");
    for (Producto producto : productosVendidos) {
        producto.mostrarDetalles();
    }
}
```

Metodos de impresion de listas:

MAIN.CPP

```
121     // Método para realizar regresión lineal usando datos de
122     // productos vendidos
123     private static void realizarRegresionLineal() {
124         if (productosVendidos.size() < 2) {
125             System.out.println("Se necesitan al menos 2 productos
126 vendidos para realizar la regresión lineal.");
127         }
128         int n = productosVendidos.size();
129         double[] cantidades = new double[n];
130         double[] precios = new double[n];
131
132         for (int i = 0; i < n; i++) {
133             Producto producto = productosVendidos.get(i);
134             cantidades[i] = producto.getCantidad();
135             precios[i] = producto.getPrecio();
136         }
137
138         RegresionLineal regresion = new RegresionLineal(cantidades,
139         precios);
140         System.out.println("Ingrese la cantidad de un nuevo producto
141 para predecir su precio:");
142         Scanner scanner = new Scanner(System.in);
143         double nuevaCantidad = scanner.nextDouble();
```

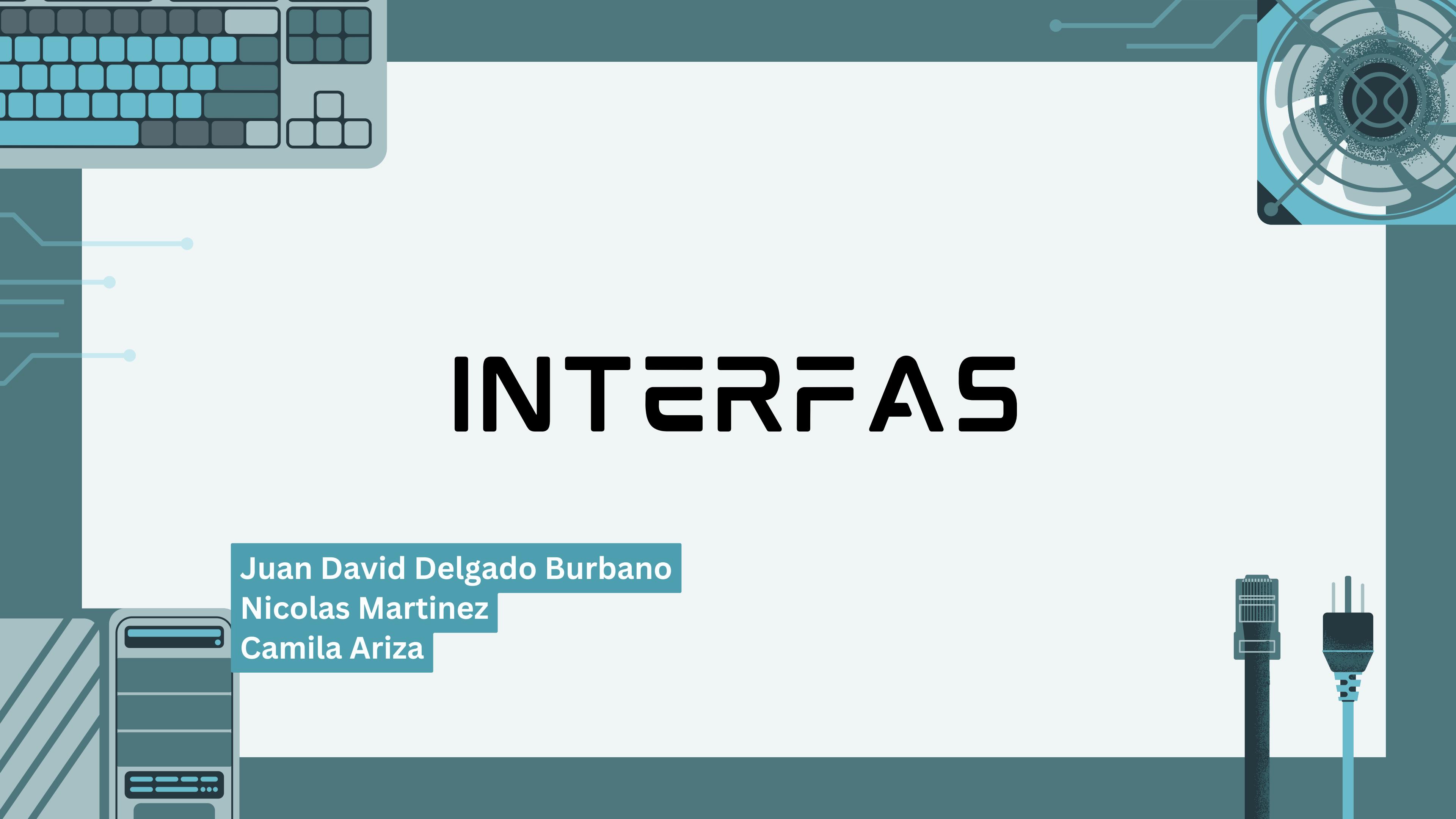
Metodos de Regrecion Lineal:

IMPRÉCION FINAL

--- Menú ---

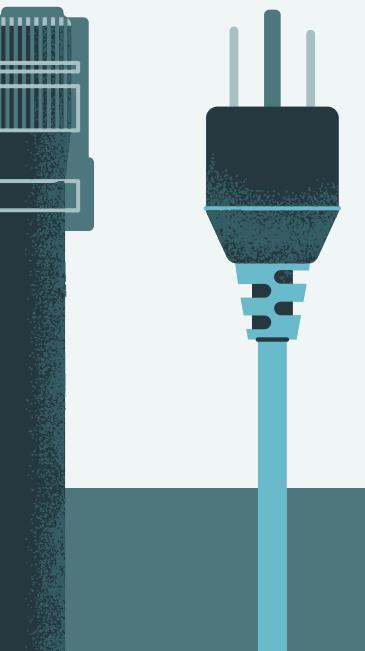
1. Agregar Producto
2. Vender Producto
3. Mostrar Productos
4. Mostrar Productos Vendidos
5. Regresión Lineal
6. Salir

Selecciona una opción:



INTERFAS

Juan David Delgado Burbano
Nicolas Martinez
Camila Ariza



INT̄RFAZ:

Main:

utilizando la biblioteca integrada de java (swing), creamos una interfaz grafica (panatalla) en donde ahora el menu se ve asi:



INTERFAZ:

Main:

en esta version todas las clases son las mismas, lo unico ue cambia es la presentacion del menu en el main en donde, se crean bottones.

```
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
       .setLayout(new GridLayout(6, 1));

        JButton btnAgregar = new JButton("Agregar
Producto");
        JButton btnVender = new JButton("Vender
Producto");
        JButton btnMostrar = new JButton("Mostrar
Productos");
        JButton btnMostrarVendidos = new
JButton("Mostrar Productos Vendidos");
        JButton btnPredecirPrecio = new
JButton("Realizar Predicción de Precio");
        JButton btnSalir = new JButton("Salir");

        btnAgregar.addActionListener(new
ActionListener() {
            public void
actionPerformed(ActionEvent e) {
```

INTERFAZ:

Main:

Y se crean acciones al
precionar estos botones que
llaman a las funciones ya
explicadas.
todos los demás archivos son
iguales.

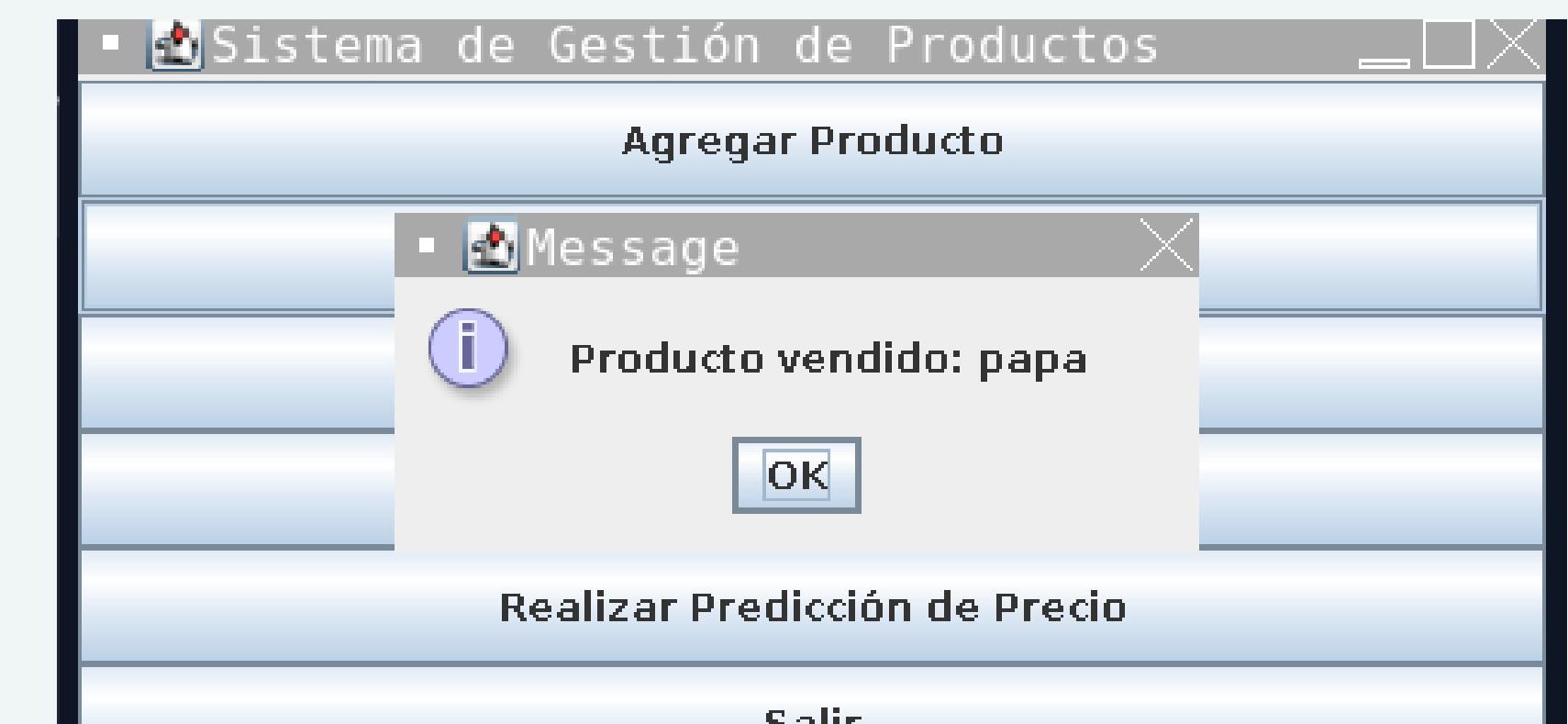
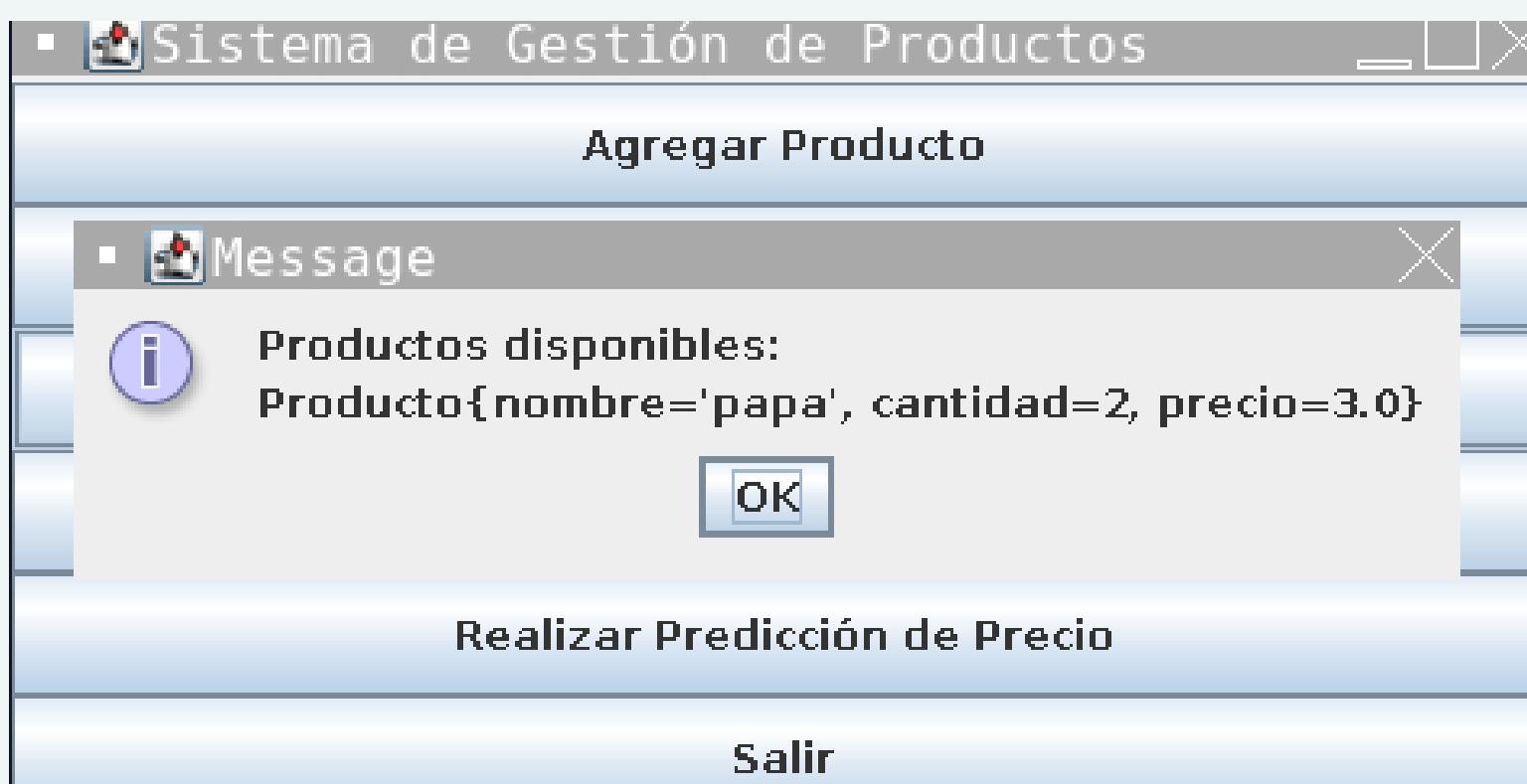
```
6             public void
7                 actionPerformed(ActionEvent e) {
8                     agregarProducto();
9                 }
0             });
1
1         btnVender.addActionListener(new
2             ActionListener() {
3                 public void
4                     actionPerformed(ActionEvent e) {
5                         venderProducto();
6                     }
7                 });
8
7         btnMostrar.addActionListener(new
8             ActionListener() {
9                 public void
10                actionPerformed(ActionEvent e) {
11                    mostrarProductos();
12                }
13            });

```

INTERFAZ:



INTERFAZ:



CONCLUSIONES

5

CONCLUSIONES

- El modelo puede mejorarse, puesto que la regresión de venta, no es tan precisa, debido a que esta solo utiliza datos de productos vendidos, entre mas productos vendidos, mayor será la precisión de la regresión.
- El proyecto logró integrar con éxito la regresión lineal en C++ para la predicción de precios y un sistema de gestión de productos en Java, aplicando principios de Programación Orientada a Objetos. A través de esta implementación, se consolidaron habilidades en múltiples lenguajes de programación y control de versiones, permitiendo un sistema funcional y adaptable tanto para la consola como para una interfaz gráfica.