

**Universidad Autonoma de Nuevo León.
Facultad de Ciencias Físico Matemáticas.**

Práctica #09 – Entrada/Salida en Java.

Laboratorio de Programación Orientada a Objetos.

Alumno: Jeremy Uriel Rossell Segura

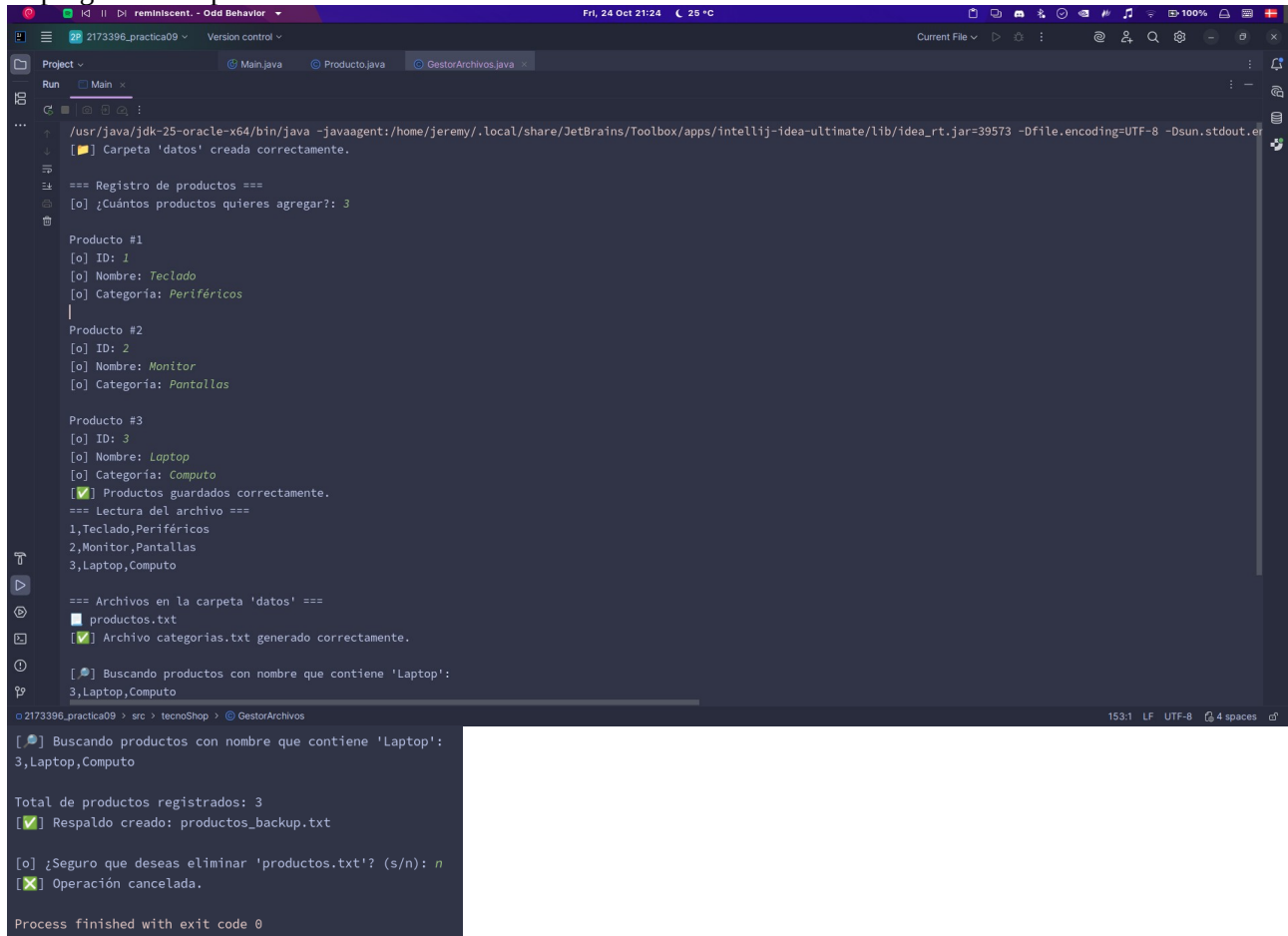
Matrícula: 2173396

Materia: Laboratorio de Programación Orientada a Objetos.

Docente: Jorge Alberto Islas Pineda

Compilación

El programa compiló adecuadamente.



```
... /usr/java/jdk-25-oracle-x64/bin/java -javaagent:/home/jeremy/.local/share/JetBrains/Toolbox/apps/intellij-idea-ultimate/lib/idea_rt.jar=39573 -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
[ ] Carpeta 'datos' creada correctamente.

=== Registro de productos ===
[o] ¿Cuántos productos quieres agregar?: 3

Producto #1
[o] ID: 1
[o] Nombre: Teclado
[o] Categoría: Periféricos
|
Producto #2
[o] ID: 2
[o] Nombre: Monitor
[o] Categoría: Pantallas
|
Producto #3
[o] ID: 3
[o] Nombre: Laptop
[o] Categoría: Computo
[✓] Productos guardados correctamente.
=== Lectura del archivo ===
1,Teclado,Periféricos
2,Monitor,Pantallas
3,Laptop,Computo

=== Archivos en la carpeta 'datos' ===
[ ] productos.txt
[✓] Archivo categorias.txt generado correctamente.

[ ] Buscando productos con nombre que contiene 'Laptop':
3,Laptop,Computo

[ ] Buscando productos con nombre que contiene 'Laptop':
3,Laptop,Computo

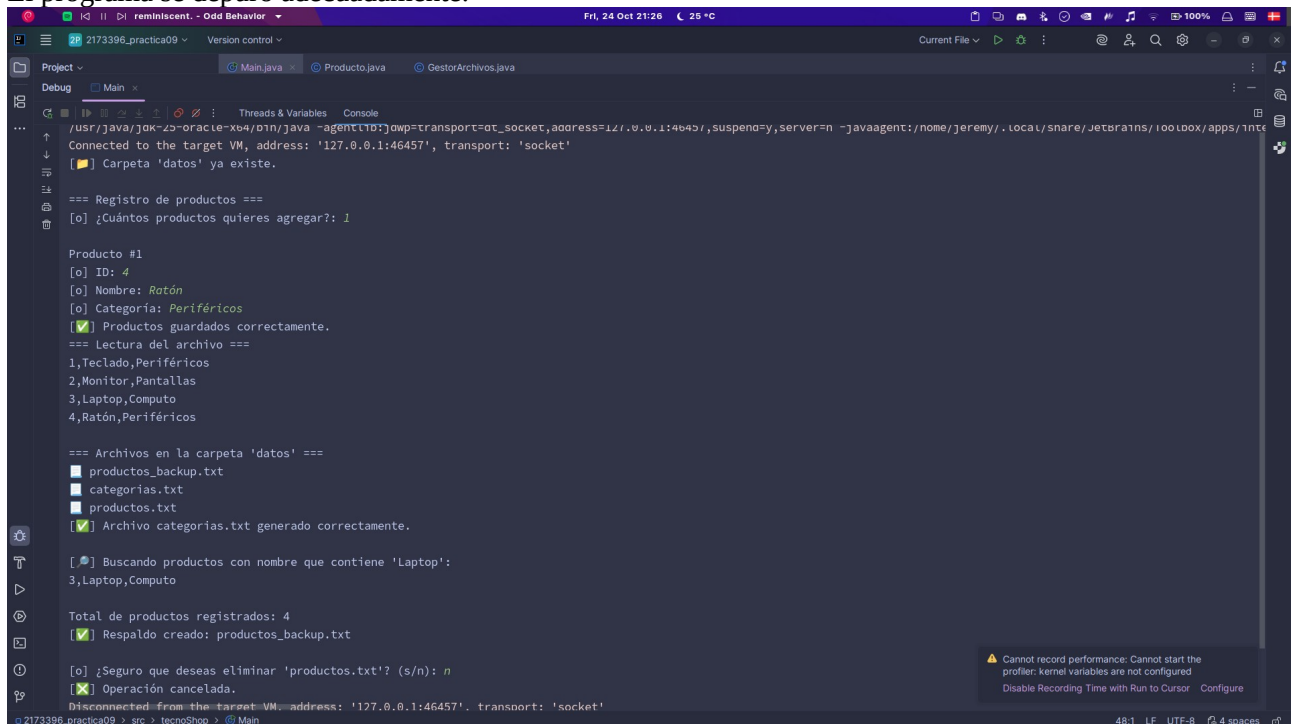
Total de productos registrados: 3
[✓] Respaldo creado: productos_backup.txt

[o] ¿Seguro que deseas eliminar 'productos.txt'? (s/n): n
[✗] Operación cancelada.

Process finished with exit code 0
```

Depuración

El programa se depuró adecuadamente.



```
... /usr/java/jdk-25-oracle-x64/bin/java -agentlib:jdwp=transport=dt_socket,address=127.0.0.1:46457,suspend=y,server=n -javaagent:/home/jeremy/.local/share/JetBrains/Toolbox/apps/intellij-idea-ultimate/lib/idea_rt.jar=48117/127.0.0.1 -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
Connected to the target VM, address: '127.0.0.1:46457', transport: 'socket'
[ ] Carpeta 'datos' ya existe.

=== Registro de productos ===
[o] ¿Cuántos productos quieres agregar?: 1

Producto #1
[o] ID: 4
[o] Nombre: Ratón
[o] Categoría: Periféricos
[✓] Productos guardados correctamente.
=== Lectura del archivo ===
1,Teclado,Periféricos
2,Monitor,Pantallas
3,Laptop,Computo
4,Ratón,Periféricos

=== Archivos en la carpeta 'datos' ===
[ ] productos_backup.txt
[ ] categorias.txt
[ ] productos.txt
[✓] Archivo categorias.txt generado correctamente.

[ ] Buscando productos con nombre que contiene 'Laptop':
3,Laptop,Computo

Total de productos registrados: 4
[✓] Respaldo creado: productos_backup.txt

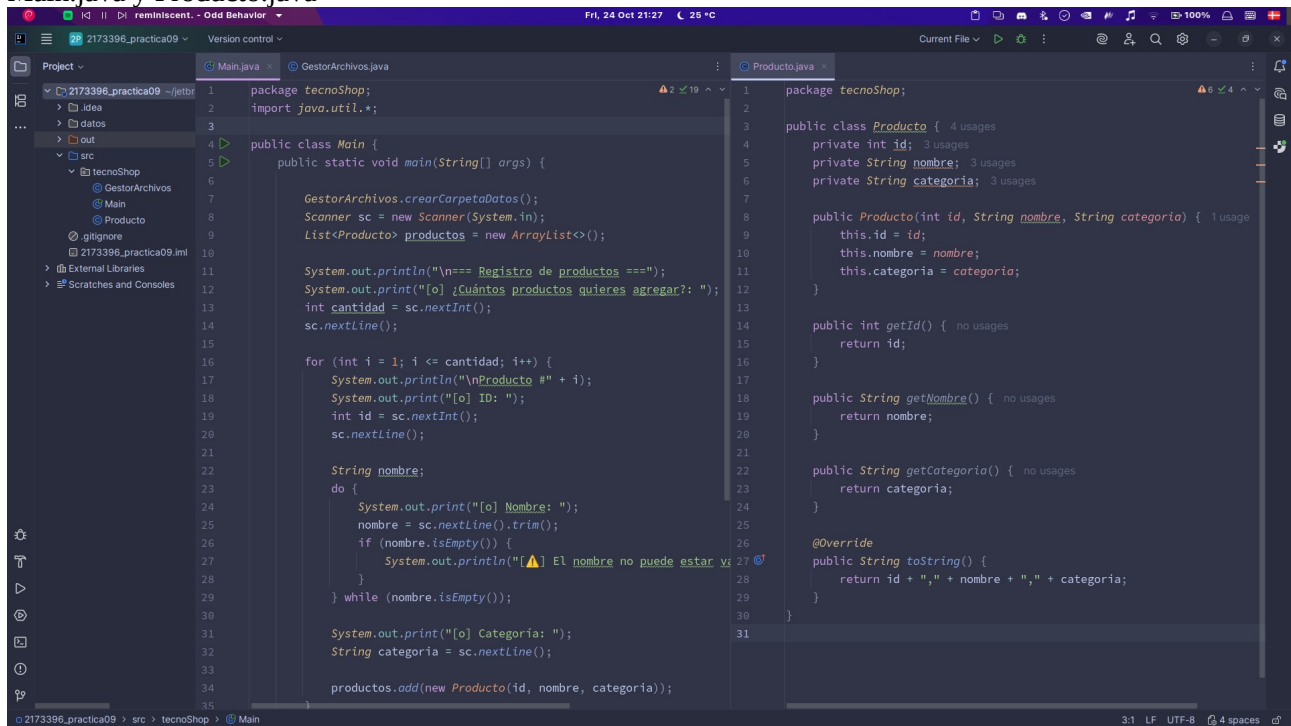
[o] ¿Seguro que deseas eliminar 'productos.txt'? (s/n): n
[✗] Operación cancelada.

Disconnected from the target VM, address: '127.0.0.1:46457', transport: 'socket'

Cannot record performance: Cannot start the profiler: kernel variables are not configured
Disable Recording Time with Run to Cursor Configure
```

Código

Main.java y Producto.java

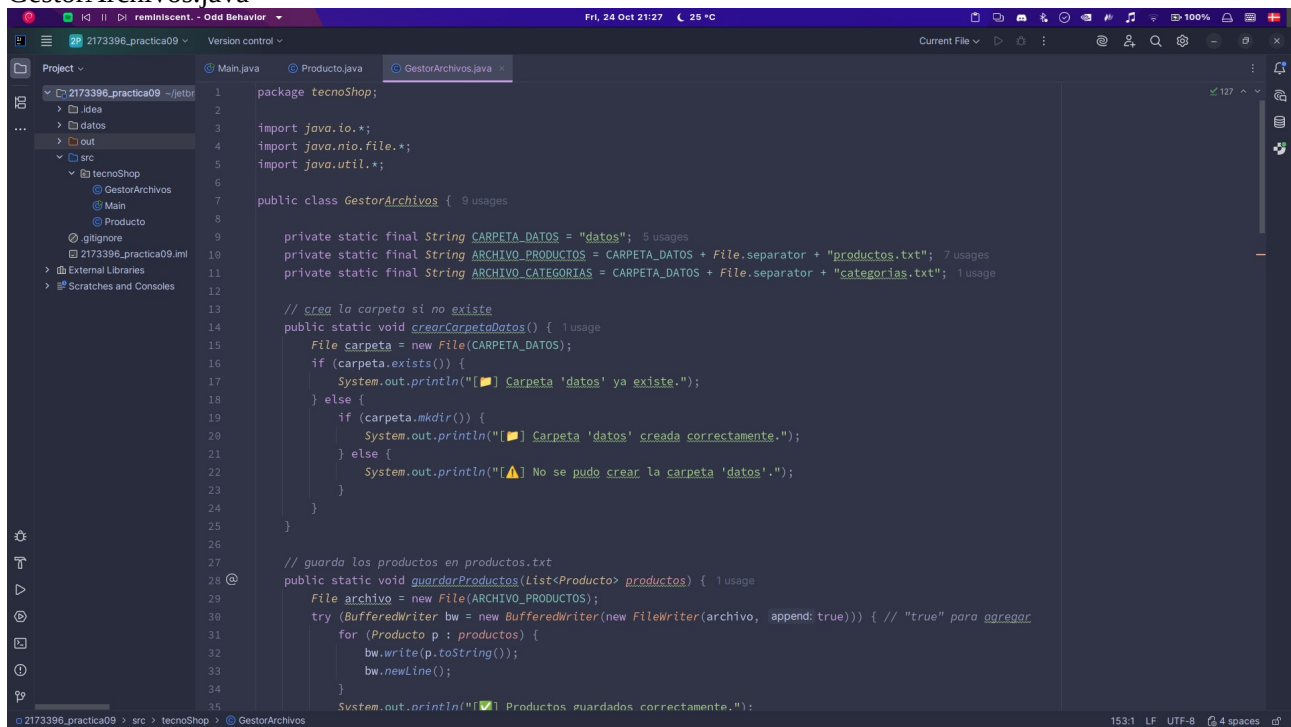


The screenshot shows an IDE with two files open: Main.java and Producto.java. The left sidebar shows a project structure with folders like 'idea', 'datos', 'out', 'src', and 'tecnosShop'. The 'src' folder contains 'GestorArchivos', 'Main', and 'Producto'.

```
1 package tecnoShop;
2 import java.util.*;
3
4 public class Main {
5     public static void main(String[] args) {
6
7         GestorArchivos.crearCarpetaDatos();
8         Scanner sc = new Scanner(System.in);
9         List<Producto> productos = new ArrayList<>();
10
11         System.out.println("\n== Registro de productos ==");
12         System.out.print("[o] ¿Cuántos productos quieres agregar?: ");
13         int cantidad = sc.nextInt();
14         sc.nextLine();
15
16         for (int i = 1; i <= cantidad; i++) {
17             System.out.println("\nProducto #" + i);
18             System.out.print("[o] ID: ");
19             int id = sc.nextInt();
20             sc.nextLine();
21
22             String nombre;
23             do {
24                 System.out.print("[o] Nombre: ");
25                 nombre = sc.nextLine().trim();
26                 if (nombre.isEmpty()) {
27                     System.out.println("[!⚠] El nombre no puede estar vacío");
28                 }
29             } while (nombre.isEmpty());
30
31             System.out.print("[o] Categoría: ");
32             String categoria = sc.nextLine();
33
34             productos.add(new Producto(id, nombre, categoria));
35         }
36     }
37 }
```

```
1 package tecnoShop;
2
3 public class Producto {
4     private int id;
5     private String nombre;
6     private String categoria;
7
8     public Producto(int id, String nombre, String categoria) {
9         this.id = id;
10        this.nombre = nombre;
11        this.categoria = categoria;
12    }
13
14    public int getId() {
15        return id;
16    }
17
18    public String getNombre() {
19        return nombre;
20    }
21
22    public String getCategoria() {
23        return categoria;
24    }
25
26    @Override
27    public String toString() {
28        return id + "," + nombre + "," + categoria;
29    }
30 }
```

GestorArchivos.java



The screenshot shows an IDE with the file GestorArchivos.java open. The left sidebar shows the same project structure as the previous screenshot. The 'src' folder contains 'GestorArchivos', 'Main', and 'Producto'.

```
1 package tecnoShop;
2
3 import java.io.*;
4 import java.nio.file.*;
5 import java.util.*;
6
7 public class GestorArchivos {
8
9     private static final String CARPETA_DATOS = "datos";
10    private static final String ARCHIVO_PRODUCTOS = CARPETA_DATOS + File.separator + "productos.txt";
11    private static final String ARCHIVO_CATEGORIAS = CARPETA_DATOS + File.separator + "categorias.txt";
12
13    // crea la carpeta si no existe
14    public static void crearCarpetaDatos() {
15        File carpeta = new File(CARPETA_DATOS);
16        if (carpeta.exists()) {
17            System.out.println("[!⚠] Carpeta 'datos' ya existe.");
18        } else {
19            if (carpeta.mkdir()) {
20                System.out.println("[!⚠] Carpeta 'datos' creada correctamente.");
21            } else {
22                System.out.println("[!⚠] No se pudo crear la carpeta 'datos'.");
23            }
24        }
25    }
26
27    // guarda los productos en productos.txt
28    public static void guardarProductos(List<Producto> productos) {
29        File archivo = new File(ARCHIVO_PRODUCTOS);
30        try (BufferedWriter bw = new BufferedWriter(new FileWriter(archivo, append: true))) {
31            for (Producto p : productos) {
32                bw.write(p.toString());
33                bw.newLine();
34            }
35            System.out.println("[!⚠] Productos guardados correctamente.");
36        } catch (IOException e) {
37            e.printStackTrace();
38        }
39    }
40 }
```

```

public class GestorArchivos { // 9 usages
    public static void guardarProductos(List<Producto> productos) { // 1 usage
    }
    System.out.println("✅ Productos guardados correctamente.");
    } catch (IOException e) {
        System.out.println("❌ Error al guardar productos: " + e.getMessage());
    }
}

// lee e imprime todo el contenido del archivo
public static void leerProductos() { // 1 usage
    System.out.println("=== Lectura del archivo ===");
    try (BufferedReader br = new BufferedReader(new FileReader(ARCHIVO_PRODUCTOS))) {
        String linea;
        while ((linea = br.readLine()) != null) {
            System.out.println(linea);
        }
    } catch (IOException e) {
        System.out.println("❌ Error al leer archivo: " + e.getMessage());
    }
}

// muestra todos los archivos dentro de la carpeta datos
public static void listarArchivos() { // 1 usage
    System.out.println("\n=== Archivos en la carpeta 'datos' ===");
    File carpeta = new File(CARPETA_DATOS);
    File[] archivos = carpeta.listFiles();
    if (archivos != null) {
        for (File f : archivos) {
            System.out.println("■ " + f.getName());
        }
    } else {
        System.out.println("[!] No hay archivos en la carpeta.");
    }
}

// busca productos por nombre dentro del archivo
public static void buscarProductoPorNombre(String nombre) { // 1 usage
    System.out.println("\n🔍 Buscando productos con nombre que contiene '" + nombre + "':");
    try (BufferedReader br = new BufferedReader(new FileReader(ARCHIVO_PRODUCTOS))) {
        String linea;
        boolean encontrado = false;
        while ((linea = br.readLine()) != null) {
            if (linea.toLowerCase().contains(nombre.toLowerCase())) {
                System.out.println(linea);
                encontrado = true;
            }
        }
        if (!encontrado) System.out.println("[!] No se encontró ningún producto con ese nombre.");
    } catch (IOException e) {
        System.out.println("❌ Error al buscar producto: " + e.getMessage());
    }
}

// crea un archivo categorias.txt con categorías únicas
public static void guardarCategoriasUnicas() { // 1 usage
    HashSet<String> categorias = new HashSet<>();

    try (BufferedReader br = new BufferedReader(new FileReader(ARCHIVO_PRODUCTOS))) {
        String linea;
        while ((linea = br.readLine()) != null) {
            String[] partes = linea.split(":", 2);
            if (partes.length == 2) {
                categorias.add(partes[1].trim());
            }
        }
    } catch (IOException e) {
        System.out.println("❌ Error al leer productos para categorías: " + e.getMessage());
    }
}

public class GestorArchivos { // 9 usages
    public static void guardarCategoriasUnicas() { // 1 usage

        try (BufferedWriter bw = new BufferedWriter(new FileWriter(ARCHIVO_CATEGORIAS))) {
            for (String cat : categorias) {
                bw.write(cat);
                bw.newLine();
            }
            System.out.println("✅ Archivo categorias.txt generado correctamente.");
        } catch (IOException e) {
            System.out.println("❌ Error al escribir categorías: " + e.getMessage());
        }
    }
}

// borra el archivo productos.txt tras confirmación
public static void borrarArchivo() { // 1 usage
    Scanner sc = new Scanner(System.in);
    System.out.print("\n¿Seguro que deseas eliminar 'productos.txt'? (s/n): ");
    String respuesta = sc.nextLine().trim().toLowerCase();
    if (respuesta.equals("s")) {
        File archivo = new File(ARCHIVO_PRODUCTOS);
        if (archivo.exists() && archivo.delete()) {
            System.out.println("🗑️ Archivo productos.txt eliminado correctamente.");
        } else {
            System.out.println("⚠️ No se pudo eliminar el archivo o no existe.");
        }
    } else {
        System.out.println("❌ Operación cancelada.");
    }
}

// cuenta cuántas líneas tiene productos.txt
public static int contarProductos() { // 1 usage
    int contador = 0;
    try (BufferedReader br = new BufferedReader(new FileReader(ARCHIVO_PRODUCTOS))) {

```

```

// cuenta cuántas líneas tiene productos.txt
public static int contarProductos() { 1 usage
    int contador = 0;
    try (BufferedReader br = new BufferedReader(new FileReader(ARCHIVO_PRODUCTOS))) {
        while (br.readLine() != null) contador++;
    } catch (IOException e) {
        System.out.println("[X] Error al contar productos: " + e.getMessage());
    }
    return contador;
}

// realiza una copia de respaldo del archivo productos.txt
public static void crearRespaldo() { 1 usage
    try {
        Path origen = Paths.get(ARCHIVO_PRODUCTOS);
        Path destino = Paths.get( first: CARPETA_DATOS + File.separator + "productos_backup.txt");
        Files.copy(origen, destino, StandardCopyOption.REPLACE_EXISTING);
        System.out.println("[✓] Respaldo creado: productos_backup.txt");
    } catch (IOException e) {
        System.out.println("[X] Error al crear respaldo: " + e.getMessage());
    }
}
}

```

Preguntas

1. ¿Qué diferencia hay entre File y Buffered Writer?

File representa una ruta o archivo en el sistema pero no escribe ni lee directamente, mientras que BufferedWriter escribe texto de manera eficiente en un archivo, usando un búfer para mejorar el rendimiento.

2. ¿Qué sucede si ejecutas el programa dos veces? ¿Se sobrescriben los datos o se agregan?

Los datos se agregan al final porque se usa new FileWriter(archivo, true) (modo append).

3. ¿Qué pasa si cambias new FileWriter(archivo, true) por new FileWriter(archivo)?

El archivo se sobrescribe completamente en cada ejecución, eliminando los registros anteriores.

4. ¿Por qué se usa try-with-resources en este programa?

Para cerrar automáticamente los recursos (FileWriter, BufferedReader, etc.) y evitar memory-leaks (fugas de memoria) o bloqueos de archivo.

5. ¿Qué función cumple new Line() dentro del Buffered Writer?

Inserta un salto de línea apropiado según el sistema operativo, separando cada producto en una línea distinta.

6. ¿Cómo podrías modificar el programa para leer solo productos de una categoría específica?

Leeyendo línea por línea y usar:

```

```java
if (linea.contains("CategoríaBuscada")) {
 System.out.println(linea);
}
```

```

O dividir cada línea con split(",") y comparar

```

```java
partes[2].trim().equalsIgnoreCase("CategoríaBuscada").
```

```

7. ¿Qué ocurriría si intentas leer un archivo que no existe?

Se lanza una excepción `FileNotFoundException`, que en mi programa se maneja dentro del `catch` mostrando un mensaje de error.

8. ¿Cómo podrías crear un método `contarProductos()` que cuente cuántas líneas tiene el archivo?

Leer el archivo con un `BufferedReader` y aumentar un contador en cada línea (ya implementado).

9. ¿Cuál es la ventaja de usar `BufferedReader` en lugar de `FileReader` directamente?

`BufferedReader` es más eficiente porque lee datos en bloques grandes en memoria, reduciendo el acceso directo al disco.

10. ¿Cómo podrías hacer un respaldo del archivo `productos.txt` dentro de la misma carpeta?

Usando:

```
```java
```

```
Files.copy(origen, destino, StandardCopyOption.REPLACE_EXISTING);
```

```
```
```

Como en `crearRespaldo()`, copiando `productos.txt` a `productos_backup.txt`.