Ejercicio 28 - Lavadero

Kevin Manuel Gómez Rojas

Universidad de Cartagena, Campus Piedra Bolívar

Nota del autor

**Ingeniería de software semestre 2**

**Profesor**: Jhon Carlos Arrieta Arrieta

**Asignatura**: Programación Orientada a Objetos

**Segunda Fase**

**Fecha de entrega:** 23 de mayo del 2025

Resumen

Continuar el desarrollo del proyecto de la actividad 1, pero en este caso incorporar un package para crear nuestras clases CRUD.

* Crear clase CRUD
* Métodos incorporados en la clase CRUD:

Agregar (C objeto).

Buscar (String código).

Editar (C objeto).

Eliminar (C objeto).

Listar Todo ().

Contar ().

Palabras clave: Poo, clases, diagramas, UML, requerimientos, modelar, código, dominio, entidades, métodos, propiedades, constructor, relaciones, ejecución.

Tabla de contenido

[Resumen 2](#_Toc196248615)

[Introducción 4](#_Toc196248616)

[Objetivos 5](#_Toc196248617)

[Objetivos Generales 5](#_Toc196248618)

[Objetivos Específicos 5](#_Toc196248619)

[Objetivos transversales 6](#_Toc196248620)

[Justificación 7](#_Toc196248621)

[Desarrollo 8](#_Toc196248622)

Introducción

La Programación Orientada a Objetos (POO) es un paradigma fundamental en el desarrollo de software, basado en la organización del código mediante clases y objetos que interactúan entre sí. Esta actividad busca aplicar estos conceptos mediante el análisis, diseño e implementación de un programa en Java, partiendo de un enunciado específico. El proceso incluye la identificación de clases, sus propiedades y métodos, la modelación con diagramas UML y la codificación en un entorno estructurado (como NetBeans), respetando principios como encapsulamiento, herencia y relaciones entre objetos. Además, se enfatiza la originalidad del trabajo, evitando prácticas como el plagio o la entrega de soluciones idénticas entre estudiantes. El resultado final debe reflejar un entendimiento claro de la POO, demostrado tanto en la documentación técnica como en la funcionalidad del programa desarrollado.

CRUD es un acrónimo que representa las cuatro operaciones fundamentales en la gestión de bases de datos: Crear (Create), Leer (Read), Actualizar (Update) y Eliminar (Delete).

En esta fase del proyecto estaré implementando Estas operaciones para las clases que componen mi proyecto.

# Objetivos

## Objetivos Generales

1. **Aplicar los principios de la Programación Orientada a Objetos (POO)** en el desarrollo de un programa en Java, utilizando clases, objetos, herencia, encapsulamiento y relaciones entre entidades.
2. **Modelar un sistema real** mediante diagramas UML, identificando las clases necesarias, sus atributos, métodos y las relaciones entre ellas.
3. **Desarrollar habilidades de análisis y diseño** al descomponer un problema en componentes manejables y traducirlos a estructuras de código organizadas.
4. **Fomentar la originalidad y el pensamiento crítico**, evitando el plagio y garantizando que la solución sea producto del análisis y la lógica propia.
5. **Practicar la implementación de código limpio y estructurado**, siguiendo convenciones de nomenclatura, paquetes y jerarquías de clases.

## **Objetivos Específicos**

1. **Identificar requerimientos funcionales** a partir de un enunciado dado, extrayendo las clases y comportamientos clave del sistema.
2. **Definir propiedades y métodos** para cada clase, asegurando un correcto encapsulamiento de los datos.
3. **Establecer relaciones entre clases** (asociación, herencia, composición, etc.) y representarlas en un diagrama de clases UML.
4. **Implementar constructores** (por defecto y parametrizados) para la creación de objetos con estados iniciales definidos.
5. **Sobrescribir el método toString()** para mostrar de manera clara la información de los objetos.
6. **Utilizar métodos set y get** para modificar y acceder a los atributos de las clases, respetando el principio de encapsulamiento.
7. **Crear instancias de objetos** en el método main, demostrando el uso de constructores y la asignación de valores a sus propiedades.
8. **Validar el funcionamiento del programa** ejecutando pruebas que muestren la interacción entre objetos y la correcta salida de datos.
9. **Añadir Metodos CRUD** Creando clases CRUD las cuales contengan los métodos básicos de un CRUD.
10. **Documentar el proceso** en un informe técnico, justificando las decisiones de diseño y la estructura del código.
11. **Gestionar la entrega técnica**, siguiendo normas de formato, nomenclatura y organización de archivos (ZIP, paquetes, etc.).

## Objetivos transversales

* 1. Fortalecer el aprendizaje autónomo mediante la investigación de conceptos POO y herramientas como UML o NetBeans.
  2. Desarrollar competencias en la detección y corrección de errores durante la compilación y ejecución del código.
  3. Promover la ética académica, entregando un trabajo original y citando correctamente las fuentes de consulta.
  4. Prepararse para evaluaciones futuras, ya que la actividad integra conocimientos clave evaluables en el ámbito profesional.

# Justificación

La presente actividad tiene como propósito fundamental consolidar las competencias en Programación Orientada a Objetos (POO), un pilar indispensable en el desarrollo de software profesional. Su relevancia radica en que este paradigma es la base estructural de sistemas complejos, frameworks modernos y arquitecturas escalables utilizadas en la industria tecnológica actual.

Al implementar esta actividad, se busca:

* Internalizar buenas prácticas de desarrollo mediante el uso de principios POO como encapsulamiento, herencia y abstracción, que son estándares en entornos laborales reales.
* Desarrollar pensamiento algorítmico y de diseño al modelar soluciones mediante diagramas UML, traduciendo problemas conceptuales a estructuras de código organizadas y mantenibles.
* Fortalecer su capacidad de autogestión al investigar, analizar e implementar soluciones originales, simulando el flujo de trabajo de un proyecto de desarrollo real.

La exigencia de originalidad en el código no es solo formalismo académico, sino un reflejo de las necesidades del sector tecnológico, donde el plagio tiene consecuencias éticas, legales y profesionales. Esta actividad, al prohibir prácticas como el maquillaje de código ajeno, nos prepara para:

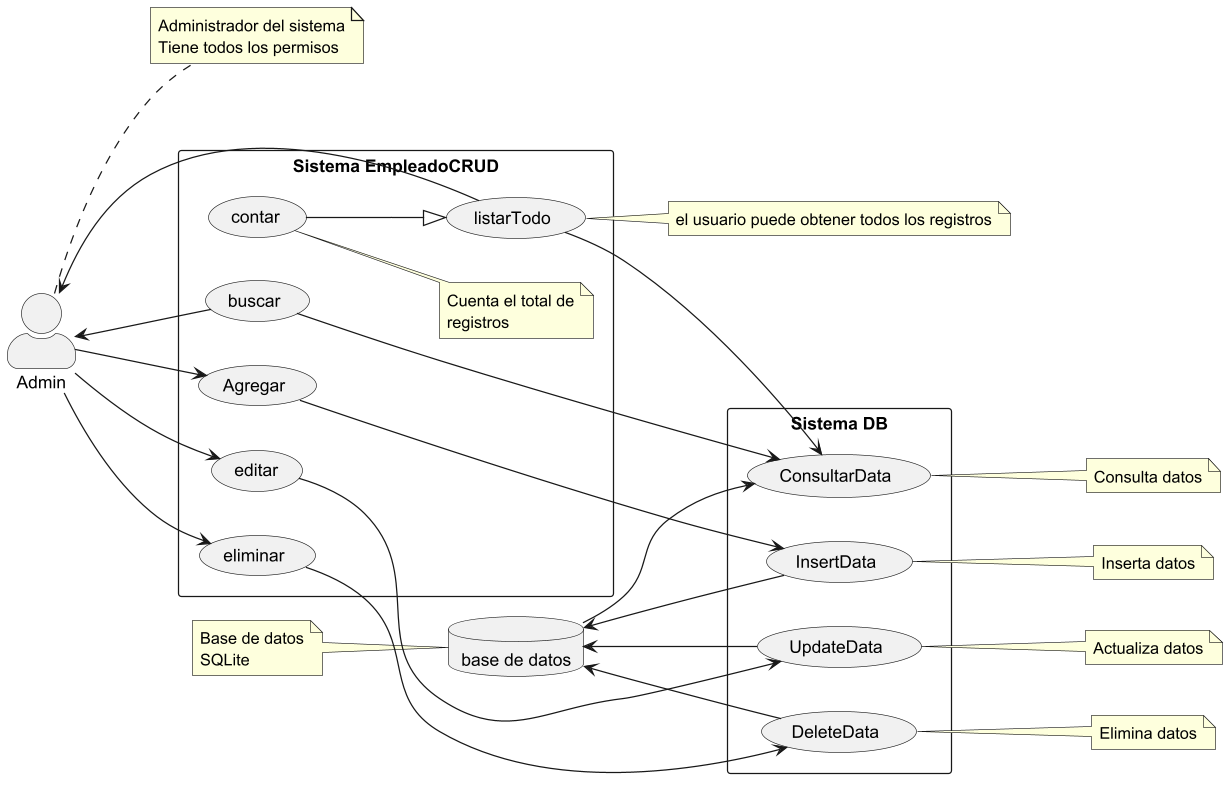
* Defender nuestras soluciones técnicamente
* Argumentar decisiones de diseño
* Desarrollar criterio propio como programado

# Desarrollo

Al agregar la metodología CRUD crearemos las clases CRUD en el package documentación.diagramas. para crear el modelado de los casos de uso de dichas clases.

Diagrama de una de las clases:

Clase: ProductoCRUD



En este caso este Diagrama de clase de uso representa la estructura de como va a funcionar o los casos de uso de dicha clase.

Lo único que distingue a estos diagramas entre si es la misma diferencia que tienen las clases entre sí, y esta diferencia es la instancia con la que se trabaja, En este caso trabajamos con una instancia del tipo **Producto**; Es decir, Vamos a agregar Productos a nuestra base de datos, buscar, editar y eliminar registros de productos.  
Teniendo en cuenta la anterior información, podemos deducir que solo necesitamos saber la lógica de como funcionan esos casos (Agregar, Buscar, Editar, Eliminar, Listar todo, Contar) para comprender como funcionan todas las clases CRUD.

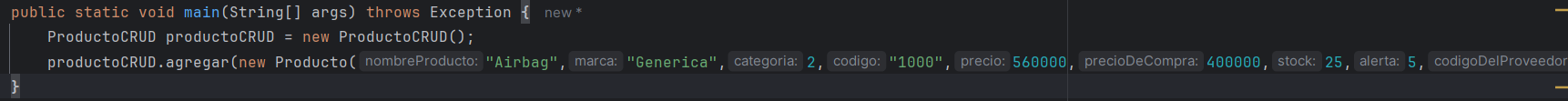
# Explicación general y resumida

## Agregar (C Object)

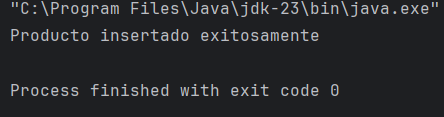
* **C** => Representa una Clase y **Object** el callback del parámetro (Instancia solicitada).
* Este método lo primero que hace es evaluar si dicha instancia se encuentra dentro de nuestra base de datos (o medio de persistencia en donde será almacenada).
* Sí este Objecto ya se encuentra guardado, retornará un error indicando la existencia de este.
* Sí el objeto no se encuentra guardado, entonces ejecutara un InserData (es una función creada para insertar datos en la base de datos) para guardar e insertar un nuevo registro.

Ejemplo:

Usando Agregar



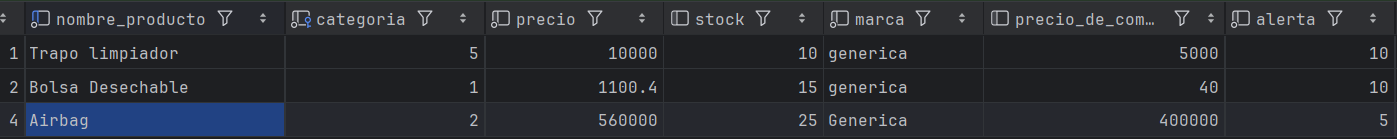
Resultado:



Al ejecutar la siguiente sentencia SQL

“SELECT \* FROM Productos;”

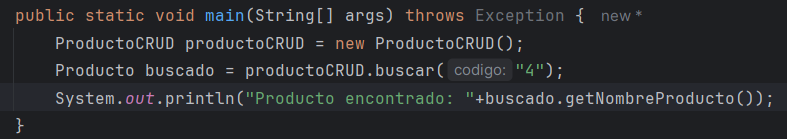
Podemos ver en nuestra Base de Datos que el producto ha sido agregado



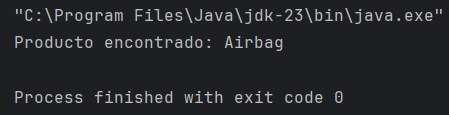
## Buscar (String codigo)

* Codigo Representa el valor identificador del objeto que vamos a buscar.
* Recorremos toda la lista de **listarTodo()** para navegar por todos los objetos guardados.
* Sí un objeto coincide en identificador con el identificador **codigo** entonces lo retornamos.
* Sí el identificador nunca coincide le retornamos una Excepción informándole de que no se encontró dicho objeto.

**Ejemplo:**



**Resultado:**

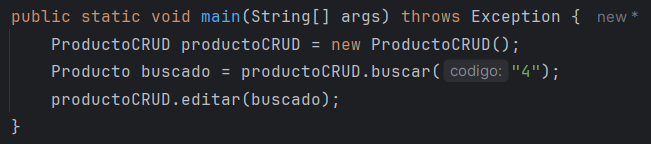
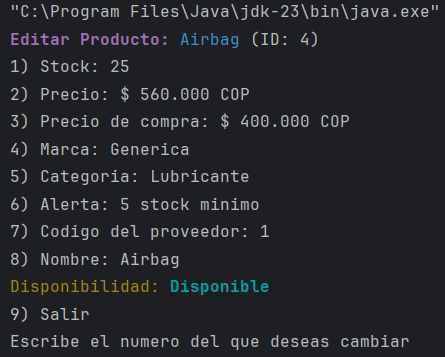
****

Recordemos que el código del producto que registramos era **4** entonces buscamos ese mismo producto por su código y como podemos ver, nos retorna exactamente ese producto.

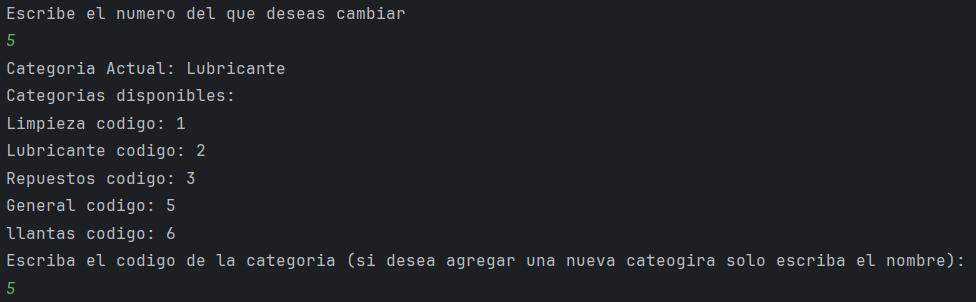
## Editar (C Object)

* C 🡪 Representa una Clase y Object representa.
* Para Editar un objeto primero validamos su existencia, Sí este existe será editado, de lo contrario enviará una excepción.
* La validación Se hace directamente en la Clase **InsertData** donde por las excepciones enviamos un mensaje “Error al insertar Producto (basados en este ejemplo): + Error recibido”
* Podemos estar seguros de que solo será editado si existe gracias a que estamos trabajando con base de dato, ella se encarga de validar si el registro existe o no.

**Ejemplo:**

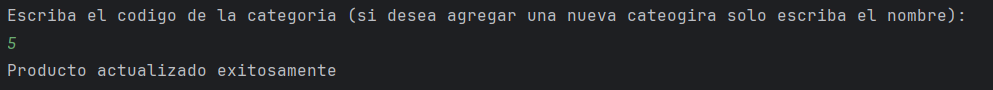
****

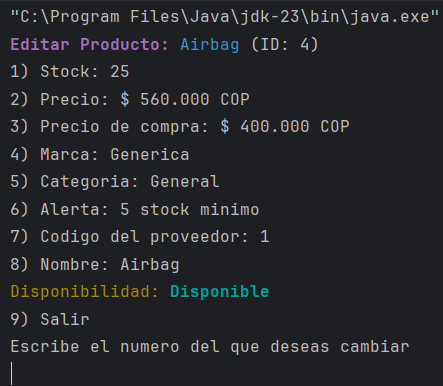
Podemos notar que la categoría no corresponde al Producto, por ende, vamos a cambiarla.



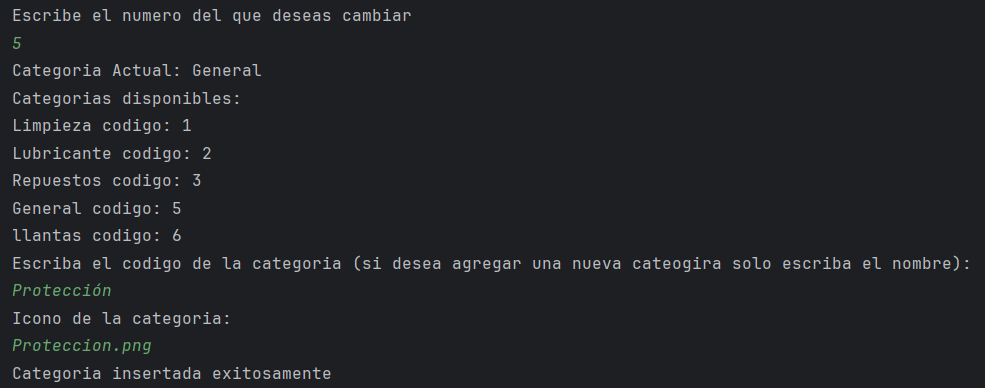
Elegimos General, porque aún no registramos una categoría específica para las características.

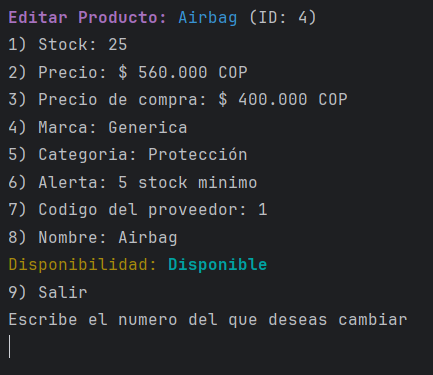
**Resultado:**

****

****

Como agregamos la categoría general, agreguemos una especifica para elementos de protección, usando el mismo editar.





Podemos ver que ahora la categoría ha cambiado a la nueva categoría agregada.

## Eliminar (C Object)

* Igual que en todos los casos, C 🡪 representa una Clase y Object el callback de dicha instancia.
* Para eliminar Usamos a la clase DeleteData, ya creada también para manejar solicitudes DELETE a nuestra base de datos, usando la siguiente sentencia **SQL**

“DELETE FROM “ + table + “ WHERE “ + **where** + “ = ?”

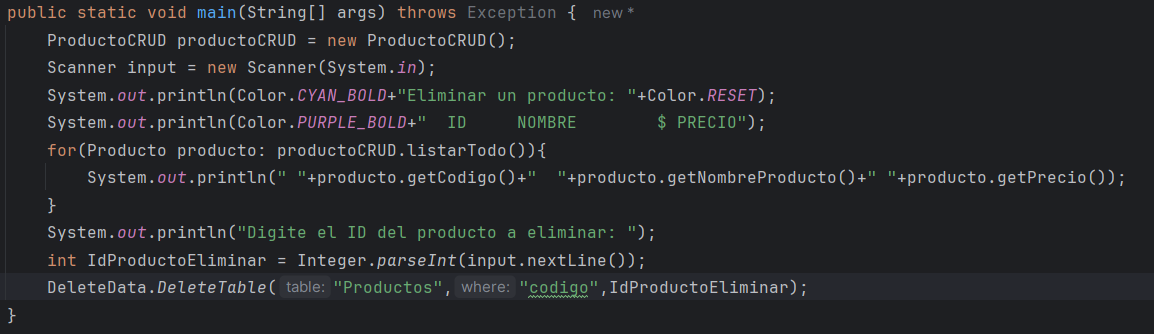
Donde **table** y **where** son pasados como parámetros a nuestra función:

**DeleteData.DeleteTable(String table, String where, int codigo)**

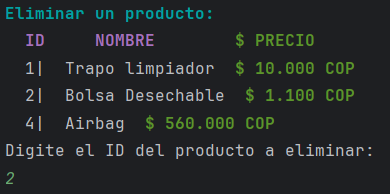
Y **?** es un argumento sql que será parseado antes de insertarlo para evitar inyecciones sql.

**Ejemplo:**

En este caso crearemos un menú para seleccionar un producto y eliminarlo.

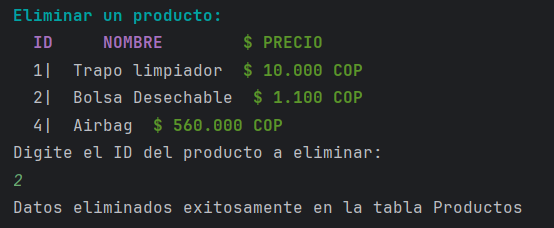


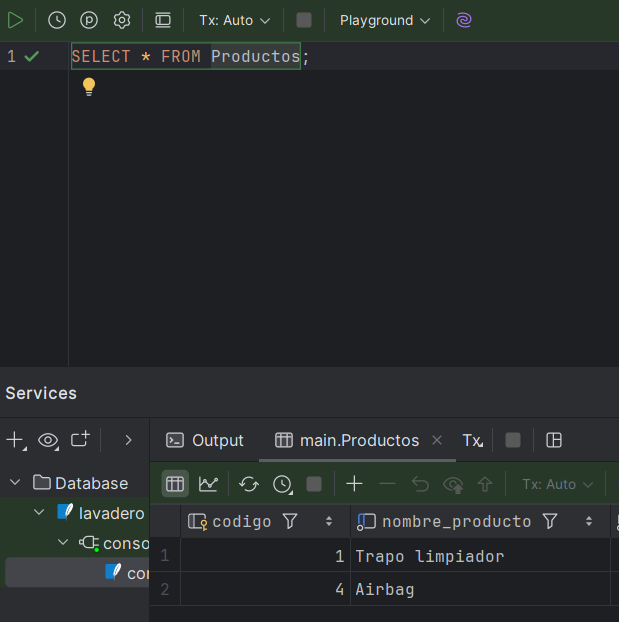
El **for** se encargará de mostrar todos los elementos de la lista, **IdProductoEliminar** se encargará de almacenar el id del producto seleccionado y el **DeleteData.DeleteTable** se encargará de eliminar el registro de ese producto en especifico.



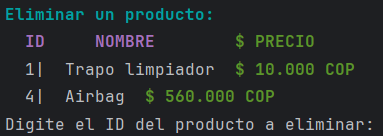
Eliminaremos la Bolsa Desechable solo para probar el correcto funcionamiento del eliminar.

**Resultado:**

****

****

Podemos ver en la base de datos que ya no existe el producto, pero al recargar el mismo codigo de antes ahora también solo nos muestra los siguientes productos:



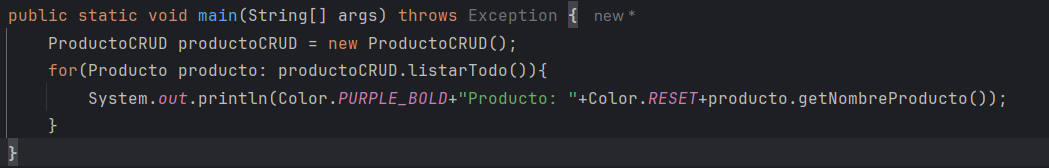
## listarTodo()

* + Listar todo nos retorna todos los registros existentes en la base de datos (en un ArrayList<Producto> en este ejemplo). El ArrayList contiene C que seria la clase empleada en el CRUD.

Podemos ver el ejemplo anterior donde lo usamos para crear el menú de eliminar.

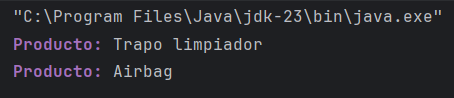
Sin embargo, ahora lo usaremos para solo listar los productos existentes:

**Ejemplo:**



Empleamos el método listarTodo para recorrer la lista que nos retorna e imprimimos en pantalla el nombre del producto.

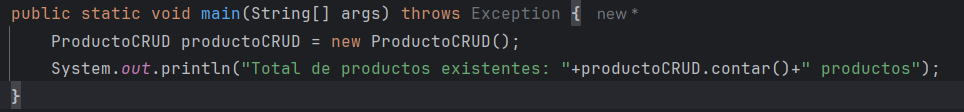
**Resultado:**



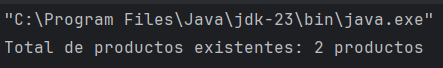
## Contar()

* + Este método emplea internamente al metodo listar y retorna el total del tamaño de la lista.

**Ejemplo:**



Siguiendo con el ejemplo de Producto, empleamos su clase crud para contar el total de productos existentes y el código anterior nos retorna lo siguiente:



Lo cual sería correcto, recordemos que eliminamos a bolsa desechable y solo quedaron 2 productos (Trapo limpiador y airbag)

## Código de los métodos:

Agregar

**public** **void** agregar(Producto p) **throws** Exception {  
 InsertData.Producto(p.getNombreProducto(),p.getMarca(),p.getCategoria(),p.getPrecio(),p.getPrecioDeCompra(),p.getStock(),p.getAlerta(),p.getCodigoDelProveedor());

}

Buscar:

**public** Producto buscar(String codigo) {  
 **return** ConsultarData.Producto(codigo);

}

Editar:

**public** **void** editar(Producto p) **throws** FileNotFoundException {  
  
 Scanner input = **new** Scanner(System.**in**);  
 CategoriaCRUD categoriaCRUD = **new** CategoriaCRUD();  
 Categoria categoria = categoriaCRUD.buscar(p.getCategoria());  
 Producto producto = ConsultarData.Producto(p.getCodigo());  
 System.**out**.println(Color.PURPLE\_BOLD+"Editar Producto: "+Color.AZUL+producto.getNombreProducto()+Color.RESET+" (ID: "+producto.getCodigo()+")");  
 System.**out**.println("1) Stock: " + producto.getStock());  
 System.**out**.println("2) Precio: " + convertirDivisa(producto.getPrecio()));  
 System.**out**.println("3) Precio de compra: "+convertirDivisa(producto.getPrecioDeCompra()));  
 System.**out**.println("4) Marca: " + producto.getMarca());  
 System.**out**.println("5) Categoria: " + categoria.getNombre());  
 System.**out**.println("6) Alerta: "+producto.getAlerta()+" stock minimo");  
 System.**out**.println("7) Codigo del proveedor: "+producto.getCodigoDelProveedor());  
 System.**out**.println("8) Nombre: " + producto.getNombreProducto());  
 System.**out**.**print**(Color.AMARILLO\_BLINK+"Disponibilidad: "+Color.CYAN\_BOLD);  
 System.**out**.**print**(producto.getDisponibilidad()?"Disponible":"No disponible");  
 System.**out**.println(Color.RESET+"\n9) Salir");  
  
 System.**out**.println("Escribe el numero del que deseas cambiar");  
 String answer = input.nextLine();  
 Gson gson = **new** Gson();  
 String json;  
 **switch** (answer){  
 **case** "1":  
 Principal.limpiarPantalla();  
 System.**out**.println("Stock Actual: "+producto.getStock());  
 System.**out**.**print**("Escriba el nuevo Stock: ");  
 **int** stock = Integer.parseInt(input.nextLine());  
 UpdateData.Bodega("stock",String.valueOf(producto.getCodigo()),String.valueOf(stock));  
 **break**;  
 **case** "2":  
 Principal.limpiarPantalla();  
 System.**out**.println("Precio Actual: "+convertirDivisa(producto.getPrecio()));  
 System.**out**.**print**("Escriba el nuevo Precio: $");  
 **double** precio = Double.parseDouble(input.nextLine());  
 UpdateData.Producto("precio",String.valueOf(precio),Integer.parseInt(producto.getCodigo()));  
 **break**;  
 **case** "3":  
 Principal.limpiarPantalla();  
 System.**out**.println("Precio de compra actual: "+convertirDivisa(producto.getPrecioDeCompra()));  
 System.**out**.**print**("Escriba el nuevo Precio de compra: $");  
 **double** precioCompra = Double.parseDouble(input.nextLine());  
 UpdateData.Producto("precio\_de\_compra",String.valueOf(precioCompra),Integer.parseInt(producto.getCodigo()));  
 **break**;  
 **case** "4":  
 Principal.limpiarPantalla();  
 System.**out**.println("Marca Actual: "+producto.getMarca());  
 System.**out**.**print**("Escriba la nueva Marca: ");  
 String marca = input.nextLine();  
 UpdateData.Producto("marca",marca,Integer.parseInt(producto.getCodigo()));  
 **break**;  
 **case** "5":  
 Principal.limpiarPantalla();  
 System.**out**.println("Categoria Actual: "+categoria.getNombre());  
 System.**out**.println("Categorias disponibles: ");  
 **for**(Categoria cat: ConsultarData.Categorias()){  
 System.**out**.println(cat.getNombre()+" codigo: "+cat.getCodigo() );  
 }  
 System.**out**.println("Escriba el codigo de la categoria (si desea agregar una nueva cateogira solo escriba el nombre): ");  
 String nuevaCategoria = input.nextLine();  
 **boolean** esNum = nuevaCategoria.matches("-?\\d+(\\.\\d+)?");  
 **if**(!esNum){  
 System.**out**.println("Icono de la categoria: ");  
 String icono = input.nextLine();  
 InsertData.Categoria(nuevaCategoria,icono);  
 **int** idCatagoriaNueva = Integer.parseInt(categoriaCRUD.listarTodo().getLast().getCodigo());  
 UpdateData.Producto("categoria",String.valueOf(idCatagoriaNueva),Integer.parseInt(producto.getCodigo()));  
 System.**out**.println("Categoria agregada con exito");  
 }**else**{  
 UpdateData.Producto("categoria",nuevaCategoria,Integer.parseInt(producto.getCodigo()));  
 }  
 **break**;  
 **case** "6":  
 Principal.limpiarPantalla();  
 System.**out**.println("Alerta Actual: "+producto.getAlerta()+" stock minimo");  
 System.**out**.**print**("Escriba la nueva Alerta: ");  
 **int** alerta = Integer.parseInt(input.nextLine());  
 UpdateData.Producto("alerta",String.valueOf(alerta),Integer.parseInt(producto.getCodigo()));  
 **break**;  
 **case** "7":  
 Principal.limpiarPantalla();  
 System.**out**.println("Codigo Actual del proveedor: "+producto.getCodigoDelProveedor());  
 System.**out**.**print**("Escriba el nuevo codigo del proveedor: ");  
 String codigoProveedor = input.nextLine();  
 UpdateData.Producto("codigo\_provedor",codigoProveedor,Integer.parseInt(producto.getCodigo()));  
 **break**;  
 **case** "8":  
 Principal.limpiarPantalla();  
 System.**out**.println("Nombre Actual: "+producto.getNombreProducto());  
 System.**out**.**print**("Escriba el nuevo nombre: ");  
 String nombre = input.nextLine();  
 System.**out**.println("Nombre actualizado: "+nombre + "en id: "+producto.getCodigo());  
 UpdateData.Producto("nombre\_producto",nombre,Integer.parseInt(producto.getCodigo()));  
 **break**;  
 **case** "9":  
 Principal.limpiarPantalla();  
 System.**out**.println("Saliendo");  
 System.**exit**(0);  
 **break**;  
 }  
  
 input.close();  
}

Eliminar:

**public** **void** eliminar(String codigo) {  
 DeleteData.DeleteTable("Productos","codigo",Integer.parseInt(codigo));  
}

listar todo:

**public** List<Producto> listarTodo() {  
 **return** ConsultarData.Productos();  
}

contar:

**public** Integer contar() {  
 **return** ConsultarData.Productos().size();  
}

Metodo exclusivo de ProductoCRUD:

**public** String convertirDivisa(**double** monto){  
 DecimalFormatSymbols dfs = **new** DecimalFormatSymbols(Locale.forLanguageTag("es-CO"));  
 dfs.setCurrencySymbol(Currency.getInstance("COP").getSymbol());  
 DecimalFormat df = **new** DecimalFormat("¤ #,##0", dfs);  
 **return** df.format(monto)+" "+dfs.getCurrency().getCurrencyCode();  
}

Cree este método para formatear valores y retornarlos con el formato de peso.

## Información Adicional u detallada

Los ejemplos mostrados en este documento representan el funcionamiento, la lógica y el paso a paso que llevan los datos dentro de los métodos solicitados en las clases Crud, para mayor información se continuará la documentación por gusto personal en archivos Markdown en el repositorio del proyecto.  
Puede encontrar los links en el siguiente párrafo.

Enlaces:

Video: [ver](https://www.youtube.com/watch?v=wEfcJSNVAC8)

Repositorio: [ver](https://github.com/GomezRKevinM/Lavadero)

Documentación General: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#fase-2)

Documentación sobre Agregar: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#agregar-)

Documentación sobre Buscar: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#buscar-)

Documentación sobre Editar: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#editar-%EF%B8%8F)

Documentación sobre Eliminar: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#eliminar-%EF%B8%8F)

Documentación sobre Listar todo: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#listar-todo-%EF%B8%8F)

Documentación sobre Contar: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#contar-)

Documentación sobre DB: [ver](https://github.com/GomezRKevinM/Lavadero?tab=readme-ov-file#metodos-db-%EF%B8%8F)