Exportación de Productos en Formato JSON Usando el Patrón Adapter

Objetivo:

El objetivo de este proyecto es implementar un sistema que permita exportar objetos (productos) a un archivo JSON, utilizando el patrón de diseño **Adapter** para desacoplar la conversión de los objetos del sistema de la lógica de exportación de datos.

1. Introducción al Patrón Adapter

El **patrón Adapter** es un patrón estructural de diseño que actúa como un intermediario entre dos interfaces incompatibles, permitiendo que trabajen juntas. En este caso, el patrón Adapter se utiliza para convertir los objetos de la aplicación a un formato de salida específico (JSON) sin modificar la estructura interna de los objetos.

2. Estructura del Proyecto

La estructura del proyecto se compone principalmente de las siguientes clases:

- **JsonAdapter.java**: Esta clase es el **Adapter** que maneja la conversión de los objetos (productos) a formato JSON.
- **Producto.java** (u otra clase de dominio): Esta clase contiene la estructura de los objetos que se desean exportar (en este caso, productos).
- **Controlador de Exportación**: La clase que invoca el adapter para convertir los objetos a JSON y guardarlos en un archivo.

3. Implementación del Adapter

3.1. Clase JsonAdapter.java

Esta clase es responsable de adaptar los objetos de tipo T a un formato JSON. Utiliza la librería **Gson** para realizar la conversión.

```
package Adapter;

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import java.util.List;


/**

 * Clase que adapta los objetos para convertirlos a formato JSON.

 *
```

```
 * @param <T> Tipo de objeto que se convertirá a JSON.
 */
public class JsonAdapter<T> {
   private final Gson gson;

   // Constructor que inicializa el Gson con formato bonito
   public JsonAdapter() {
      this.gson = new GsonBuilder().setPrettyPrinting().create();
   }
   /**
    * Convierte un objeto de tipo T a formato JSON.
    *
    * @param objeto El objeto de tipo T que se convertirá.
    * @return El objeto convertido a formato JSON.
    */
   public String convertirAFormato(T objeto) {
      return gson.toJson(objeto);
   }
   /**
    * Convierte una lista de objetos de tipo T a formato JSON.
    *
    * @param lista La lista de objetos de tipo T que se convertirá.
    * @return La lista convertida a formato JSON.
    */
   public String convertirAFormato(List<T> lista) {
      return gson.toJson(lista);
   }
}
```

3.2. Explicación del Código:

- **Gson**: Se utiliza para la conversión de objetos a JSON. Se crea un objeto Gson configurado para producir un JSON "bonito" (con saltos de línea y sangrías).
- **convertirAFormato(T objeto)**: Convierte un solo objeto de tipo T a JSON.
- **convertirAFormato(List lista)**: Convierte una lista de objetos de tipo T a JSON.

4. Clase de Producto (Ejemplo)

La clase Producto representa los objetos que se van a exportar a JSON.

```
public class Producto {
  private String nombre;
  private double precio;
  private String categoria;

  public Producto(String nombre, double precio, String categoria) {
    this.nombre = nombre;
    this.precio = precio;
    this.categoria = categoria;
  }

  // Getters y setters
}
```

Esta clase contiene la estructura de los productos, con atributos como nombre, precio y categoria.

5. Uso del Adapter

Para utilizar el patrón Adapter, primero se crea un objeto JsonAdapter y luego se utiliza para convertir los productos o listas de productos a formato JSON.

```
public class Exportador {

  public static void main(String[] args) {
    // Crear productos de ejemplo
```

```
        Producto p1 = new Producto("Producto 1", 19.99, "Categoría 1");
        Producto p2 = new Producto("Producto 2", 29.99, "Categoría 2");

        // Lista de productos
        List<Producto> productos = Arrays.asList(p1, p2);

        // Crear el adaptador para exportar a JSON
        JsonAdapter<Producto> adaptador = new JsonAdapter<>();

        // Convertir la lista de productos a JSON
        String jsonProductos = adaptador.convertirAFormato(productos);

        // Guardar el JSON en un archivo (este paso se puede hacer con FileWriter o similar)
        System.out.println(jsonProductos);
    }
}
```

Explicación:

1. Se crean algunos productos de ejemplo.
2. Se agrupan en una lista.
3. Se crea una instancia de JsonAdapter<Producto>.
4. Se convierte la lista de productos a formato JSON.
5. Se imprime el JSON generado.

6. Exportación a un Archivo JSON

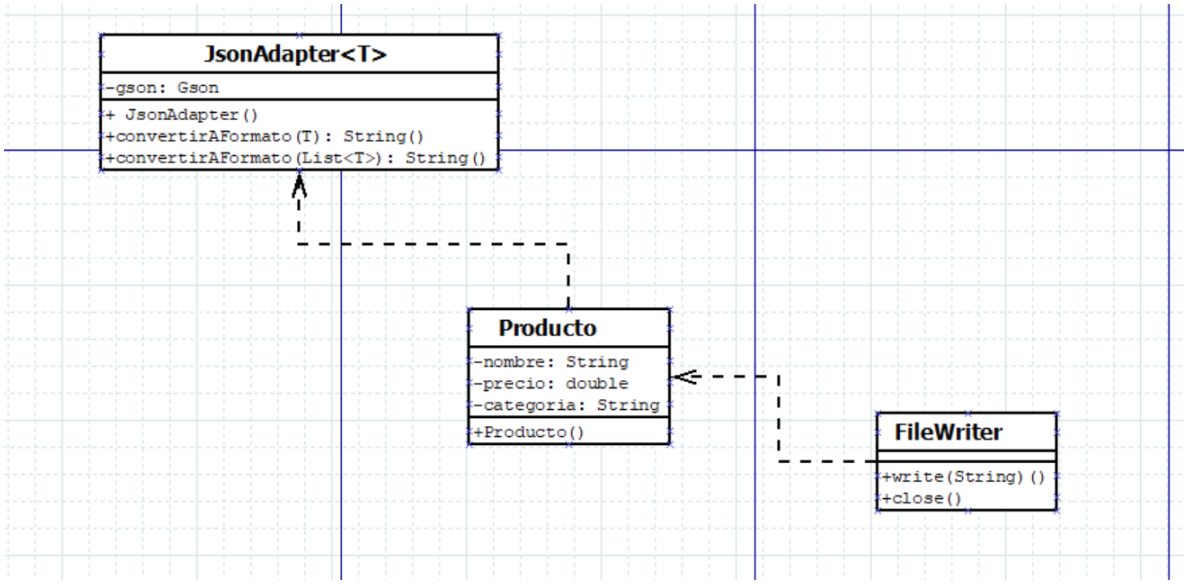
Para guardar los productos en un archivo JSON, puedes usar el siguiente código:

```
import java.io.FileWriter;
import java.io.IOException;

public class Exportador {
```

```
    public static void main(String[] args) {
        // Crear productos de ejemplo
        Producto p1 = new Producto("Producto 1", 19.99, "Categoría 1");
        Producto p2 = new Producto("Producto 2", 29.99, "Categoría 2");

        // Lista de productos
        List<Producto> productos = Arrays.asList(p1, p2);

        // Crear el adaptador para exportar a JSON
        JsonAdapter<Producto> adaptador = new JsonAdapter<>();

        // Convertir la lista de productos a JSON
        String jsonProductos = adaptador.convertirAFormato(productos);

        // Guardar el JSON en un archivo
        try (FileWriter writer = new FileWriter("productos.json")) {
            writer.write(jsonProductos);
            System.out.println("Productos exportados correctamente a productos.json");
        } catch (IOException e) {
            System.err.println("Error al guardar el archivo: " + e.getMessage());
        }
    }
}
```

Este código guarda la cadena JSON en un archivo llamado productos.json en la misma carpeta del proyecto.

7. UMl



8. Beneficios del Patrón Adapter en Este Contexto

- **Desacoplamiento**: La conversión de objetos a JSON está separada de la lógica del negocio. Esto significa que puedes cambiar la forma en que los productos se convierten o exportan sin afectar el resto de la aplicación.
- **Facilidad para Extender el Código**: Si necesitas exportar otros tipos de datos en el futuro (por ejemplo, Clientes, Órdenes), puedes reutilizar el patrón Adapter sin tener que reescribir el código.
- **Manejo de Diferentes Formatos**: El patrón Adapter puede ser extendido para soportar otros formatos de exportación, como XML o CSV, simplemente agregando más adaptadores o métodos.

Conclusión

El patrón Adapter ha sido implementado con éxito para transformar objetos Producto en formato JSON y exportarlos a un archivo. Esta implementación permite mantener el código flexible y escalable para futuras necesidades de exportación o adaptación a otros formatos.