Tecnológico Nacional De México

Instituto Tecnológico de Oaxaca

Ingeniería En Sistemas Computacionales

Diseño e Implementación de Software con Patrones.

7 am – 8 am

Unidad 2

“Patrón de Diseño Adaptador”

Presenta:

Copy Max

Nombres	Numero de Control
Bautista Fabian Max	**C19160532**
Celis Delgado Jorge Eduardo	**21160599**
Flores Guzmán Alan Ismael	**20161193**
García Osorio Bolívar	**20161819**
Pérez Barrios Diego	**21160750**
Perez Martínez Edith Esmeralda	**21160752**
Sixto Morales Ángel	**21160797**

Periodo Escolar:

Febrero – Julio

Grupo:

7SB

Maestro:

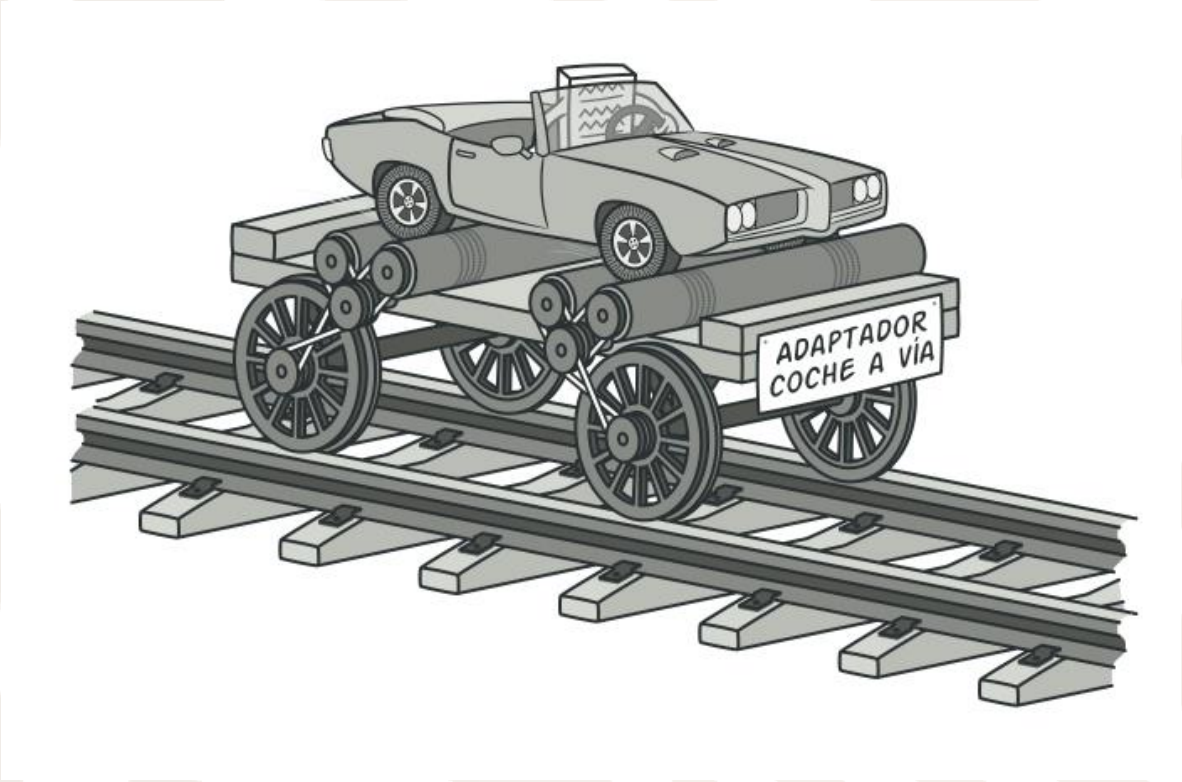
Espinoza Pérez Jacob

Oaxaca de Juárez, Oaxaca **Marzo, 2025**

INDICE

Patrón de Diseño Adapter

Adapter es un patrón de diseño estructural que permite la colaboración entre objetos con interfaces incompatibles.



Puedes crear un adaptador. Se trata de un objeto especial que convierte la interfaz de un objeto, de forma que otro objeto pueda comprenderla.

Un adaptador envuelve uno de los objetos para esconder la complejidad de la conversión que tiene lugar tras bambalinas. El objeto envuelto ni siquiera es consciente de la existencia del adaptador.

Implementación en el 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.

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);
    }
}
```

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.

Clase Producto()

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.

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:

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

Exportacion a un archivo JSON

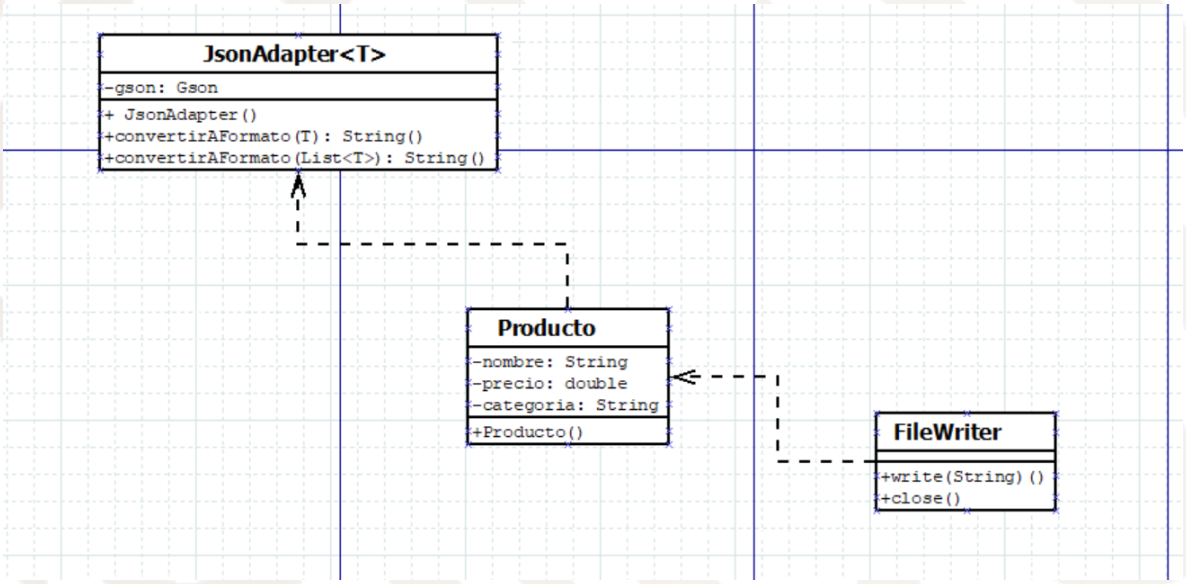
Para guardar los productos en un archivo JSON, puedes usar el siguiente código:

```
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.

Estructura UML



Ventajas y Desventajas

Ventajas	Desventajas
Principio de responsabilidad única. Puedes separar la interfaz o el código de conversión de datos de la lógica de negocio primaria del programa.	La complejidad general del código aumenta, ya que debes introducir un grupo de nuevas interfaces y clases. En ocasiones resulta más sencillo cambiar la clase de servicio de modo que coincida con el resto de tu código.
Principio de abierto/cerrado. Puedes introducir nuevos tipos de adaptadores al programa sin descomponer el código cliente existente, siempre y cuando trabajen con los adaptadores a través de la interfaz con el cliente.	

Conclusión

El patrón Adaptador es una solución estructural que permite que dos interfaces incompatibles trabajen juntas sin modificar su código fuente. Es útil cuando se necesita integrar sistemas heredados o reutilizar clases existentes con una nueva interfaz. Aunque mejora la reutilización de código y la compatibilidad, un uso excesivo puede generar una estructura de código difícil de manejar.