Implementación del Patrón Adaptador para Integrar Librería de Analítica

Workshop / 2

Por:

Camilo Sierra

David Herrera

Angie Guevara

Jhonatan Avila

2024

Caso de Estudio: Monitoreo del Mercado Accionario con Integración de Librería de Analítica.

Introducción:

En el siguiente informe se plantea la siguiente situación *"Suponga que le asignan la tarea de diseñar una aplicación para monitorear el mercado accionario. La aplicación descarga los datos de las acciones desde múltiples fuentes en formato XML y luego presenta gráficos y diagramas para el usuario. Luego de diseñar la aplicación, le piden mejorarla integrando una librería inteligente de analítica de un tercero, sin que la aplicación quede acoplada a dicha librería para que en cualquier momento esta se pueda cambiar. La librería del tercero solo trabaja con datos en formato JSON"*, luego se analiza el diseño de una aplicación para monitorear el mercado accionario y la integración de una librería de analítica de terceros.

Diseño Inicial:

La aplicación al inicio recibe y descarga datos de las acciones en **DatosAcciones** en formato XML (gráficos y diagramas). Luego la aplicación cumple con su función principal de recopilar y presentar datos del mercado accionario en la clase **DatoAccionXML**, luego se busca la forma de pasar los datos por un patrón de diseño que nos ayude a la complejidad adaptador que realiza la conversión de XML a JSON y así finalizaría mostrando los cambios realizados

Mejora con Patrón Adaptador:

Para superar las limitaciones realizamos la búsqueda del patrón de diseño que más nos beneficie y luego implementarlo en el diseño inicial, se integrar una librería de analítica de terceros **“jackson-dataformat-xml”** como dependencia en POM.XML, utilizando el patrón de diseño Adaptador **AdaptadorJSON**. Este patrón adaptador aísla la información del código para pasar de formato de XML a JSON de la librería, permitiendo reemplazar en el futuro la información sin afectar el código principal.

Implementación del Adaptador:

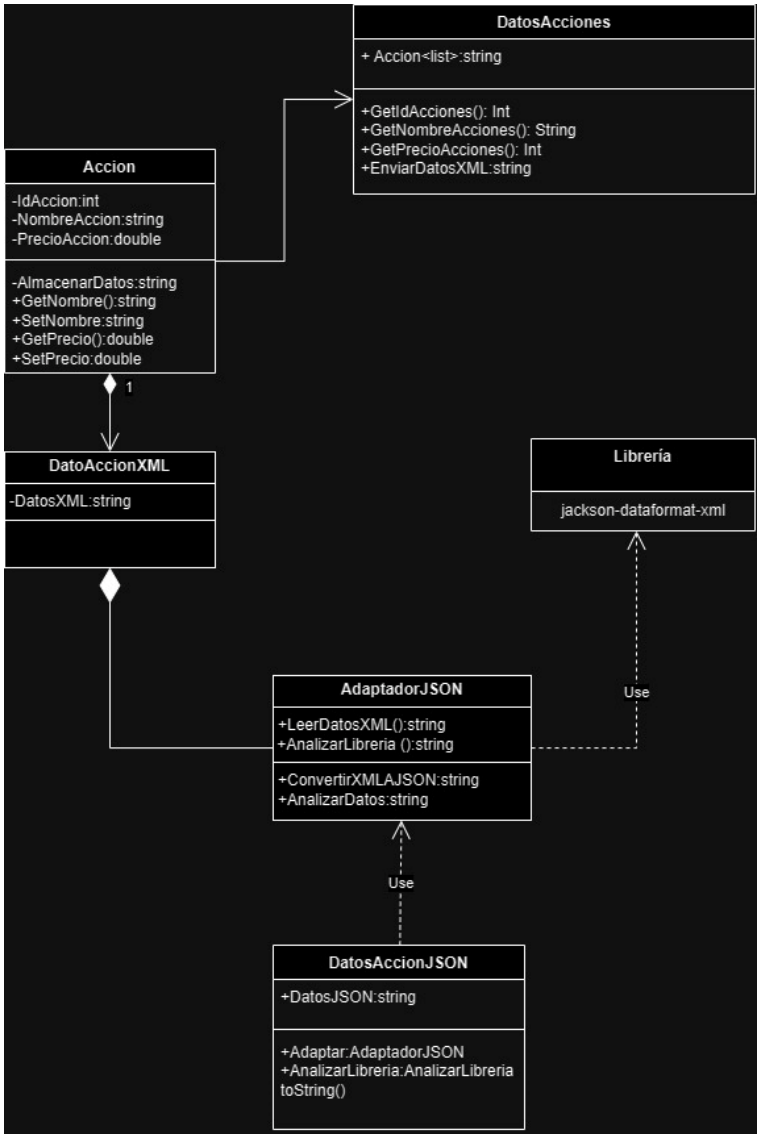
Para implementar el patrón Adaptador se:

1. Se crea la clase adaptadora **AdaptadorJSON** que traducen los datos XML a JSON y viceversa, y que encapsulan la información con la librería de analítica.
2. la aplicación utiliza la clase adaptador en lugar de interactuar directamente con los datos XML o la librería de analítica.
3. La librería de analítica queda aislada dentro de la clase adaptadora, lo que permite reemplazar sin modificar el código principal de la aplicación.
4. La clase adaptador analiza y convierte los datos, arrojando como resultado la conversión entre XML a JSON con su estructura correspondiente al formato

Beneficios del Patrón Adaptador:

El uso del patrón Adaptador en este caso nos permite:

- integrar diferentes librerías de analítica en el futuro sin necesidad de reescribir el código principal.
- Facilita la maleabilidad del código al aislar la integración de la librería.
- Las clases adaptadoras pueden ser reutilizadas para hacer el código más eficiente conciso por medio de librerías o similares para que así pueda convertir la interfaz de un objeto, de forma que otro objeto pueda comprenderlo.



Main.java

```java
package org.example;

import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try {
            AdaptadorJSON adapter = new AdaptadorJSON();
            String xmlFilePath = "C:\\Users\\camil\\IdeaProjects\\untitled\\src\\main\\java\\org\\example\\acciones.xml";

            Acciones poppy = adapter.fromXml(xmlFilePath, Acciones.class);
            String json = adapter.toJson(poppy);

            System.out.println(json);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

pom.xml (AplicacionMercadoAcciones)

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/xml-apis/xml-apis -->
        <dependency>
            <groupId>xml-apis</groupId>
            <artifactId>xml-apis</artifactId>
            <version>2.0.2</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.13.3</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
            <version>2.13.3</version>
        </dependency>
    </dependencies>
</project>
```

Accion.java

```java
package org.example;

import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;

public class Accion {  3 usages
    @JacksonXmlProperty(localName = "nombre")  2 usages
    private String nombre;

    @JacksonXmlProperty(localName = "precio")  2 usages
    private double precio;

    // Getters y setters
    public String getNombre() {  no usages
        return nombre;
    }

    public void setNombre(String nombre) {  no usages
        this.nombre = nombre;
    }

    public double getPrecio() {  no usages
        return precio;
    }

    public void setPrecio(double precio) {  no usages
        this.precio =precio;
    }
}

```

Acciones.java

```java
package org.example;

import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlElementWrapper;
import com.fasterxml.jackson.dataformat.xml.annotation.JacksonXmlProperty;

import java.util.List;

public class  2 usages
Acciones {
    @JacksonXmlElementWrapper(useWrapping = false)  2 usages
    @JacksonXmlProperty(localName = "Accion")
    private List<Accion> acciones;

    // Getters y setters
    public List<Accion> getAcciones() {  no usages
        return acciones;
    }

    public void setAcciones(List<Accion> acciones) {  no usages
        this.acciones = acciones;
    }
}
```

AdaptadorJSON.java

```java
package org.example;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.dataformat.xml.XmlMapper;

import java.io.File;
import java.io.IOException;

public class AdaptadorJSON {  2 usages
    private XmlMapper xmlMapper;  2 usages
    private ObjectMapper jsonMapper;  2 usages

    public AdaptadorJSON() {  1 usage
        xmlMapper = new XmlMapper();
        jsonMapper = new ObjectMapper();
    }

    public <T> T fromXml(String filePath, Class<T> valueType) throws IOException {  1 usage
        return xmlMapper.readValue(new File(filePath), valueType);
    }

    public String toJson(Object value) throws IOException {  1 usage
        return jsonMapper.writeValueAsString(value);
    }
}
```

```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
{"acciones":[{"nombre":"Apple Inc.","precio":150.0},{"nombre":"Google","precio":2800.0}]}

Process finished with exit code 0
```

Conclusión:

El patrón Adaptador ha demostrado ser una herramienta valiosa para integrar la librería de analítica de terceros en la aplicación de monitoreo del mercado accionario. La aplicación resultante es más flexible, escalable y mantenible, permitiendo una mejor experiencia para los usuarios y una mayor capacidad de adaptación a futuros cambios en las necesidades del negocio.