MÓDULO IV: PATRONES DE ESTRUCTURA DE SOFTWARE | ACTVIDAD 1

Integrantes:
Mario Augusto Lúe Morales
Wilber Denilson Lopez Perez
Daniel Ernesto Mira Santos

Indicaciones:

- Explica de forma escrita la solución para cada caso
- Realiza el diagrama de flujo resultante
- Dibuja un diseño de clases

Caso 1: Sistema de Clasificación de Archivos (Patrón Strategy)

Una aplicación de gestión de archivos necesita clasificar archivos en diferentes carpetas según criterios específicos, como la extensión del archivo, el tamaño del archivo y la fecha de creación. El objetivo es permitir que los usuarios seleccionen diferentes estrategias de clasificación sin cambiar el código principal de la aplicación.

1.1. Problema

La aplicación de gestión de archivos necesita clasificar archivos en diferentes carpetas según criterios específicos, como la extensión del archivo, el tamaño del archivo y la fecha de creación. Sin embargo, el requisito clave es permitir que los usuarios seleccionen diferentes estrategias de clasificación sin cambiar el código principal de la aplicación.

1.2. Solución: Patrón Strategy

El patrón Strategy se utiliza para definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. En este caso, podemos aplicar este patrón para resolver el problema de clasificación de archivos.

Componentes

1. **Contexto (ArchivoClassifier):** Representa la aplicación de gestión de archivos que necesita clasificar archivos. Este componente es responsable de delegar la tarea de clasificación a una estrategia específica.
2. **Estrategias (FileExtensionStrategy, FileSizeStrategy, FileDateStrategy):** Son las diferentes estrategias de clasificación que se pueden utilizar. Cada estrategia implementa la lógica específica para clasificar archivos según un criterio determinado (extensión del archivo, tamaño del archivo, fecha de creación, etc.).
3. **Interfaz de estrategia (FileClassificationStrategy):** Define la interfaz común que deben implementar todas las estrategias de clasificación. Esta interfaz declara el método *classify File()* que se encarga de clasificar un archivo según el criterio específico de la estrategia.

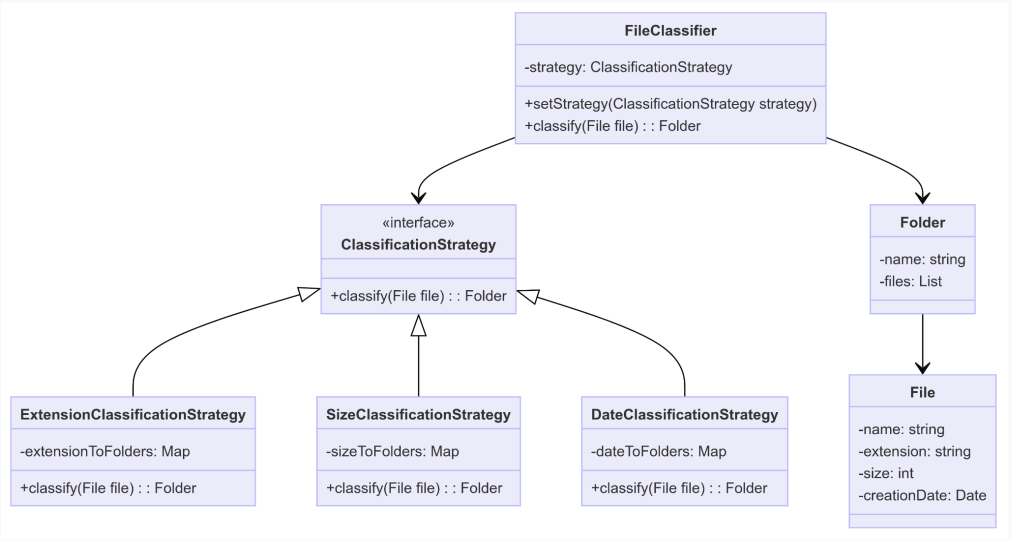
Flujo de trabajo

1. El usuario selecciona una estrategia de clasificación específica (por ejemplo, clasificar por extensión del archivo).
2. La aplicación de gestión de archivos (Contexto) crea una instancia de la estrategia seleccionada (FileExtensionStrategy).
3. El Contexto llama al método *classify File()* de la estrategia, pasando el archivo que se va a clasificar como parámetro.
4. La estrategia implementa la lógica específica para clasificar el archivo según su criterio (en este caso, la extensión del archivo).
5. La estrategia devuelve la carpeta correspondiente donde se debe clasificar el archivo.
6. El Contexto utiliza la carpeta de vuelta para clasificar el archivo.

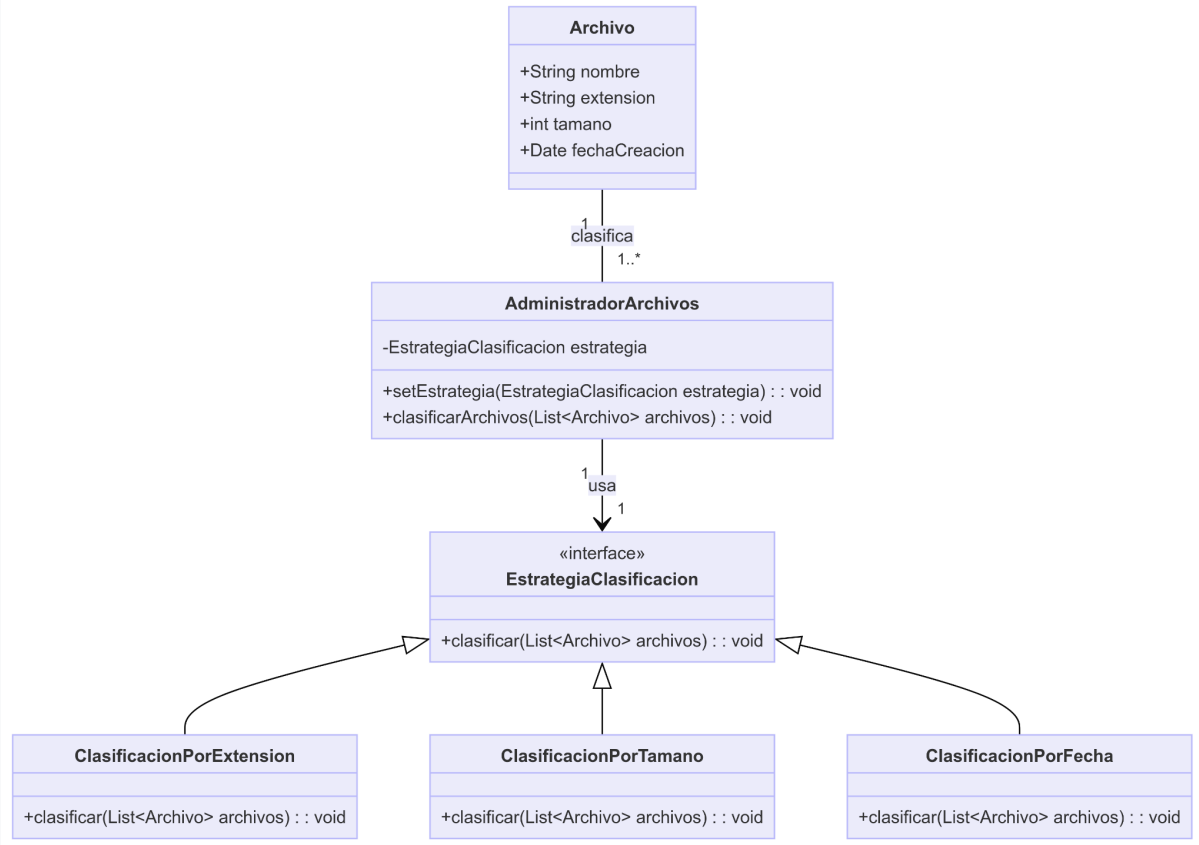
Ventajas

1. Flexibilidad: El patrón Strategy permite cambiar la estrategia de clasificación sin modificar el código principal de la aplicación.
2. Reutilización: Las estrategias de clasificación se pueden reutilizar en diferentes contextos.
3. Escalabilidad: Es fácil agregar nuevas estrategias de clasificación sin afectar el código existente.

1.3. Diseño de clases



1.4. diagrama de flujo resultante



Caso 2: Generación de Reportes (Patrón Template Method)

Una empresa necesita generar diferentes tipos de reportes (por ejemplo, informes financieros y de ventas). Cada reporte sigue una estructura general común, pero los detalles específicos, como el contenido y el formato, varían entre los tipos de reportes. El objetivo es permitir la reutilización de la estructura común del reporte mientras se permite la personalización de los detalles específicos.

2.1. Definición del patrón

El patrón Template Method se basa en la creación de una clase abstracta que define la estructura común del reporte, y varias clases concretas que heredan de la clase abstracta y proporcionan la implementación específica para cada tipo de reporte.

2.2 Estructura del patrón

La estructura del patrón Template Method se compone de los siguientes elementos:

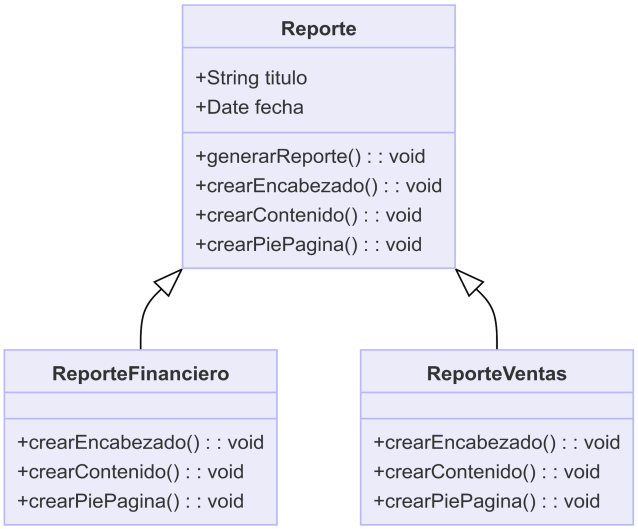
- Clase abstracta Report: Define la estructura común del reporte, incluyendo los métodos que se encargan de generar el reporte. Algunos de estos métodos serán abstractos, lo que significa que no tendrán implementación en la clase abstracta.
- Clases concretas FinancialReport, SalesReport, etc.: Heredan de la clase abstracta Report y proporcionan la implementación específica para cada tipo de reporte.

Ventajas

El patrón Template Method ofrece varias ventajas:

- Reutilización de código: La estructura común del reporte se define solo una vez en la clase abstracta, lo que reduce la duplicación de código.
- Personalización: Cada clase concreta puede personalizar los detalles específicos del reporte sin afectar la estructura común.
- Flexibilidad: Es fácil agregar nuevos tipos de reportes sin modificar la clase abstracta.

2.3. Diseño de clases



2.4. diagrama de flujo resultante