Object-Oriented Software Engineering

CS 319

Fall 2022

Design Pattern Homework

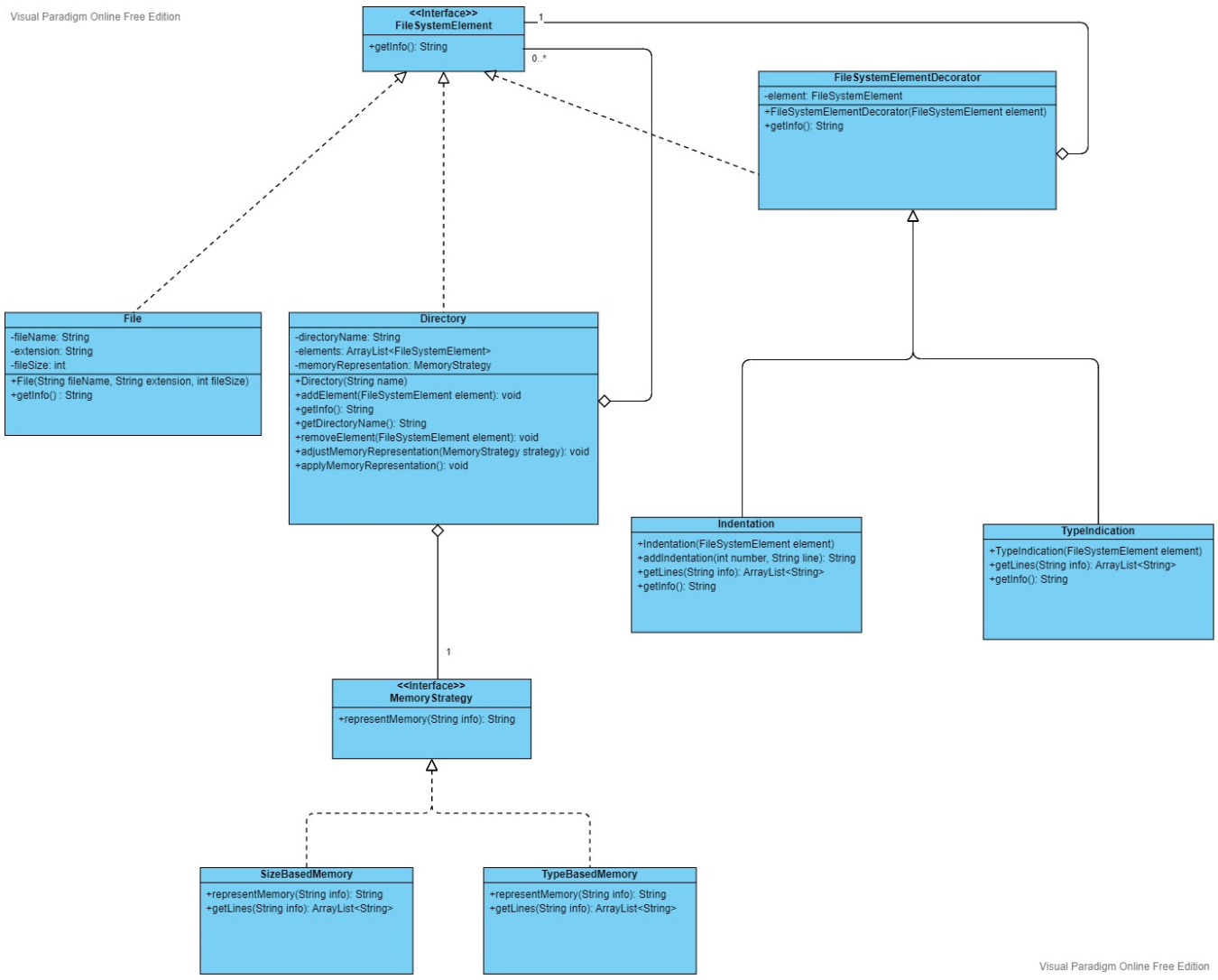
Instructor: Eray Tüzün

Name: Kutay Tire

ID: 22001787

Date: 09.12.2022

Class Diagram



Explanations

In this assignment, three design patterns were used for each specific functionality required. These design patterns were the **Composite Design Pattern** for the first part, **Decorator Design Pattern** for the second part and the **Strategy Design Pattern** for the third part.

Part 1

In the first part, the necessary methods and classes for displaying the content of directories and files were implemented. However, the key factor was that a Directory could include both a file and another directory with in self. So, this was achieved by adding an ArrayList<FileSystemElement> property inside Directory where FileSystemElement is the interface class that generalizes both Directory and File classes. This means that File and Directory classes implemented the FileSystemElement interface while Directory itself also held an array list of FileSystemElement objects. Naturally, composite design pattern was used in establishing this part-whole hierarchy between the classes. Furthermore, for this part, the FileSystemElement class was the component of the structure as it was the base interface for the objects involved in composition. The File class was the leaf with some default behavior whereas the Directory class was the composite that contained leaf elements.

Part 2

In the second part, two new ways of displaying information for directories and files were required. The first way was by indentation and the second way was by type indication. Moreover, combination of these could also be used by the customers. To implement these additional features, the decorator design pattern was used. This allowed adding new functionalities to an existing object without changing the original structure. For this, a decorator class which was FileSystemElementDecorator was added to act as a wrapper. This abstract decorator class also implemented the FileSystemElement interface. From this class, two concrete classes, Indentation and TypeIndication, were extended as these were the classes that would override the getInfo () method and provide the desired functionalities. For this to happen, the FileSystemElementDecorator class included a FileSystemElement object to call the getInfo () method which is modified in the subclasses. This design pattern was chosen for this part as it enables flexibility for extending functionalities. Moreover, covering all possible scenarios is much easier and faster especially when the number of additional features increases.

Part 3

In the third part, two memory representation methods that can be chosen by the users were implemented. The first method used size-based memory representation whereas the second method used type-based memory representation. To achieve this, strategy design pattern

was used. The idea was to help the users choose which algorithm to use for memory representation. The first step of implementation was to create a new interface called MemoryStrategy. The Directory class was also modified to hold an instance of this MemoryStrategy interface. Then, the concrete classes implementing this MemoryStrategy interface, TypeBasedMemory and SizeBasedMemory, implemented the representMemory () method individually to provide the desired memory representations. The final step was the implementation of adjustMemoryRepresentation () and applyMemoryRepresentation () methods of the Directory class. The first method sets the instance of MemoryStrategy in Directory to either TypeBasedMemory or SizeBasedMemory based on the choice of the user and the second method prints the memory representation by calling the representMemory () method of the concrete classes. Strategy design pattern was used because it was the most convenient for customers to use different variants of an algorithm within an object and switch from one algorithm to another during runtime without using unnecessary if-else statements.