Utku Boran Torun / Section-01 / 21901898

PART 1

In Part 1, I used a composite design pattern. I implemented this design pattern because I needed to create a tree structure that one of the leaves uses the parent class (FileComponent interface). In this homework, our leaves were File and Directory, and the interface these classes use is FileComponent. Since File and Directory share same method (getInfo()), I put that function in the interface. The implementation of getInfo() changes between File and Directory, so I did the implementation in their own class. I put the array list of FileComponent instance in the Directory class, and when I did that, my composite design pattern was done. By putting the FileComponent instance in the Directory class, I could put both File and Directory instances in a Directory object.

PART 2

In Part 2 I used a decorator design pattern. I used this design pattern because I need to add new behaviors to objects at runtime without breaking the code that uses those objects. In this homework, these objects were File and Directory. I created a class called FileDecorator which implements the FileComponent interface and created an instance of FileComponent in this class. Then I created two classes which are FileIndentationDecorator and FileTypeIndicateDecorator. These classes inherit the FileDecorator class. These two classes do not have any properties, but they override the getInfo() method in the FileDecorator class. When I've done with this, the decorator design pattern was finished and implemented.

PART 3

In Part 3 I used a strategy design pattern. I used this design pattern because I needed to switch one algorithm to other algorithm during the runtime (adjustMemoryRepresentation()). I created an interface called FileStrategy that has one function (logDirectory(String directoryInfo))). Then I created two new classes called MemoryBasedStrategy and SizeBasedStrategy. These two classes implement the FileStrategy interface and do not have any properties or methods other than logDirectory(). After this, I put the instance of FileStrategy in the Directory class. Then I implemented adjustMemoryRepresentation() and applyMemoryRepresentation() functions. After I had done with all these, the strategy design pattern was done.

Class Diagram