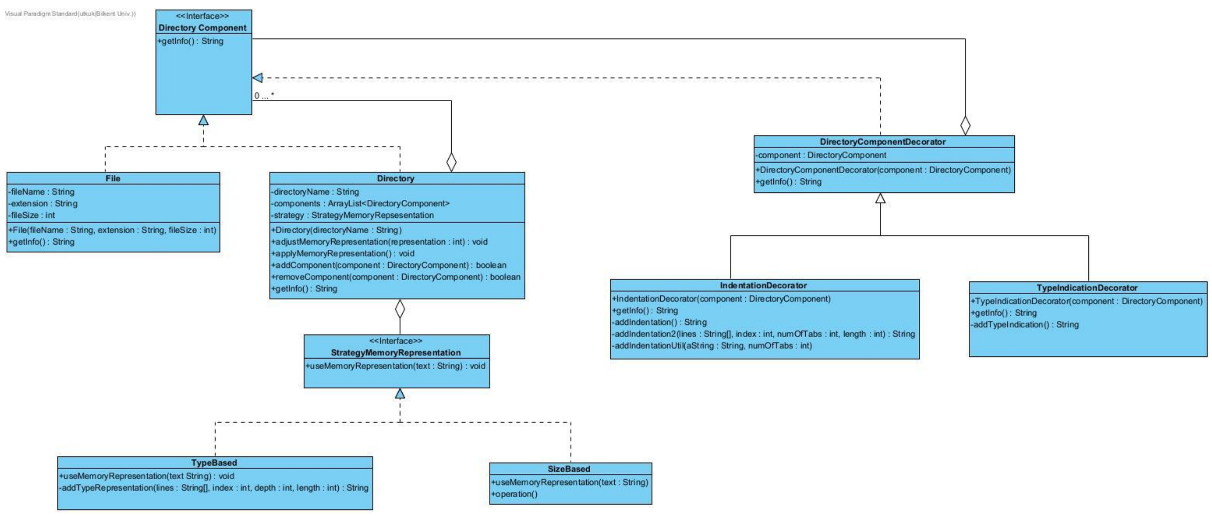
Bilkent University

CS 319 - Object-Oriented Software Engineering

Design Pattern Homework

Name: Utku Kurtulmuş
ID : 21903025

Class Diagram Of The Implementation


Visual Paradigm Standard(utkuk(Bilkent Univ.))
<<Interface>>
Directory Component
+getInfo() : String
0 ... *
File
-fileName : String
-extension : String
-fileSize : int
+File(fileName : String, extension : String, fileSize : int)
+getInfo() : String
Directory
-directoryName : String
-components : ArrayList<DirectoryComponent>
-strategy : StrategyMemoryRepsesentation
+Directory(directoryName : String)
+adjustMemoryRepresentation(representation : int) : void
+applyMemoryRepresentation() : void
+addComponent(component : DirectoryComponent) : boolean
+removeComponent(component : DirectoryComponent) : boolean
+getInfo() : String
<<Interface>>
StrategyMemoryRepresentation
+useMemoryRepresentation(text : String) : void
TypeBased
+useMemoryRepresentation(text String) : void
-addTypeRepresentation(lines : String[], index : int, depth : int, length : int) : String
SizeBased
+useMemoryRepresentation(text : String)
+operation()
DirectoryComponentDecorator
-component : DirectoryComponent
+DirectoryComponentDecorator(component : DirectoryComponent)
+getInfo() : String
IndentationDecorator
+IndentationDecorator(component : DirectoryComponent)
+getInfo() : String
-addIndentation() : String
-addIndentation2(lines : String[], index : int, numOfTabs : int, length : int) : String
-addIndentationUtil(aString : String, numOfTabs : int)
TypeIndicationDecorator
+TypeIndicationDecorator(component : DirectoryComponent)
+getInfo() : String
-addTypeIndication() : String

Design Patterns

Part 1

To solve the first part of the problem I have preferred to use the **composite** design pattern. The problem was very similar to the first example of the design pattern tutorial. The directory object needs to be able to store both "file" and "directory" objects. Thus, instead of checking the types of the contents of the directory by using conditional statements, I applied the composite design. This allows me to do my operations directly on the directory component object, and when I called "getInfo()" method it automatically decides if the object is a file or directory object.

Part 2

To solve the first part of the problem I have preferred to use the **decorator** design pattern. The problem was very similar to the second example of the design pattern tutorial. The decorator design patted allowed me to add new "indentation" and "type indication" features without changing the main structure of the directory object. Hence, I was able to use both decorator and composite design patterns at the same time. Moreover, we can combine different features easily with the help of a decorator design pattern. We don't need to write new methods to combine the features.

Part 3

To solve the first part of the problem I have preferred to use the **strategy** design pattern. Firstly, I thought that I could add these features as new decorator objects and use the decorator pattern that I have implemented for the part 2 of the problem. However, since the question says to use adjustMemoryRepresentation and applyMemoryRepresentation methods on the directory object, I preferred to use the strategy design pattern. The strategy design pattern also allowed me to add the new memory representation feature to the directory object without changing its structure. Furthermore, the strategy design pattern allowed me to select which memory representation I was going to use at runtime. I didn't need to use conditional statements in "applyMemoryRepresentation" method.