CS 319 - 2022/2023 Fall
Design Patterns Homework
02 December 2022
Due: 23:55, December 9, 2022

Important Notes:

- In this lab, you are expected to use design patterns that you learned in class and in the tutorial.
- You will submit the following:
  - Source code which is written in Java.
    - ➢ For each different class, you should create a new java file.
    - ➢ There should be a main file where you test your implementation in detail. You must test each **part** of the question and separate them via comments.
  - A report in pdf named *report* which will contain:
    - ➢ **Class Diagram** of your implementation
    - ➢ Explanation of design patterns that you used. You should explicitly say **on which object(s)**, **how (e.g. which properties did you change),** and **why** you used that design pattern.
- While submitting your work, put your source code and report in a folder named **"<Student_ID>_<Name>_<Surname>_<Section_No>_DesignPattern"**. Finally, zip that file and upload it to moodle.
  (Example: 21011122_Metehan_Sacakci_02_DesignPattern.zip)
- The students who do not follow the submission guidelines will lose some points!
- **NO LATE SUBMISSIONS ALLOWED!**

Problem

Today, you have started a new position at Microsoft as a Software Engineer in the Operating Systems domain. In this domain, "managing file system" has importance for the company, and that is why you will be responsible for the implementation of that part.

The general idea about the file system is like the following:

- There are two important objects: *file* and *directory*
- Inside a *directory*, there might be another *file* and *directory* objects.

PART 1

As a first step, the boss wants you to implement a *File* class. This object will have three properties: *fileName* and *extension* with type String, *fileSize* with type integer representing the file size in kilobytes. Other than the constructor, *File* class will only have one method, which is *getInfo()*. When *getInfo()* function is called, it will return an informative message about that *file* object. The format of that message is important and should be like the following:

```
fileName.extension | File Size: fileSize kb
```

Example 1:

For a *File* object with:

fileName = "video"
extension = "mp4"
fileSize = 200

getInfo() will return:

```
video.mp4 | File Size: 200 kb
```

The boss also wants you to implement a *Directory* class. This object must have a *directoryName* property with the type String, however, you are free to add any other property as necessary. Similar to a *file* object, a *directory* object must have a *getInfo()* function, however, you are free to add any other new methodsas necessary. When you call *getInfo()* function, it will return an informative string about both the *directoryName* and the content of the directory. The format of that message is important and should be like the following:

```
- directoryName
<content_of_directory>
- DIRECTORY END | directoryName
```

IMPORTANT REMARK: The order of items in the directory is important and the earliest item added to the directory should be on the upper.

Example 2:

Assume there is an empty directory with the name "MyDocuments". That directory's *getInfo*() function will return the following string:

```
- MyDocuments
- DIRECTORY END | MyDocuments
```

Example 3:

Assume there is a directory with the name "FavoriteMusic". It only contains the following file:

fileName = "MetallicaTheUnforgiven"
extension = "mp3"
fileSize = 100

That directory's *getInfo*() function will return the following string:

```
- FavoriteMusic
MetallicaTheUnforgiven.mp3 | File Size: 100 kb
- DIRECTORY END | FavoriteMusic
```

Example 4:
Assume there is a directory with the name "BilkentStuff". In that directory, there are three files and the FavoriteMusic directory from Example 3. Those three files' properties can be found below:

fileName = "InternshipReport"
extension = "pdf"
fileSize = 10

fileName = "setup"
extension = "exe"
fileSize = 5

fileName = "ToDoList"
extension = "docx"
fileSize = 15

That directory's *getInfo*() function will return the following string:

```
- BilkentStuff
InternshipReport.pdf | File Size: 10 kb
setup.exe | File Size: 5 kb
ToDoList.docx | File Size: 15 kb
- FavoriteMusic
MetallicaTheUnforgiven.mp3 | File Size: 100 kb
- DIRECTORY END | FavoriteMusic
- DIRECTORY END | BilkentStuff
```

Example 5:
Assume there is a directory with the name "MyPC". It contains "MyDocuments" from Example 2 and "BilkentStuff" from Example 4. That directory's *getInfo*() function will return the following string:

```
- MyPC
- MyDocuments
- DIRECTORY END | MyDocuments
- BilkentStuff
InternshipReport.pdf | File Size: 10 kb
setup.exe | File Size: 5 kb
ToDoList.docx | File Size: 15 kb
- FavoriteMusic
MetallicaTheUnforgiven.mp3 | File Size: 100 kb
- DIRECTORY END | FavoriteMusic
- DIRECTORY END | BilkentStuff
- DIRECTORY END | MyPC
```

IMPORTANT REMARK: For simplicity, assume that any file object's name will not contain the character "-". In the output, you should see the "-" character only in the beginning and the ending locations of the directories.

PART 2

After the implementation that you did, people have started to face a problem. They are saying that whether they can understand the contents of the file system, they cannot easily detect file system elements' positions (which file belongs to which directory). In order to fix this issue, your boss wants you to implement two features. These features can be used alone, however, they should be combined and used together as well.

- **Feature 1: Indentation**

When this feature is applied, *getInfo()* function will indent the contents of directories according to their depth.

Example 6:

If “indentation” is applied to MyPC directory from Example 5, the output will be the following:

```
- MyPC
     - MyDocuments
     - DIRECTORY END | MyDocuments
     - BilkentStuff
          InternshipReport.pdf | File Size: 10 kb
          setup.exe | File Size: 5 kb
          ToDoList.docx | File Size: 15 kb
          - FavoriteMusic
               MetallicaTheUnforgiven.mp3 | File Size: 100 kb
          - DIRECTORY END | FavoriteMusic
     - DIRECTORY END | BilkentStuff
- DIRECTORY END | MyPC
```

- **Feature 2: Type Indication**

When this feature is applied, each row of the output of the *getInfo()* function will be updated according to the following rules:

Rule 1: If the current row is the starting point of a *directory*, there will be “`(d)`” at the beginning of that row.

Rule 2: If the current row is the ending point of a *directory*, there will be “`(!)`” at the beginning of that row.

Rule 3: If the current row is information about a *file* object, there will be “`(f)`” at the beginning of that row.

Example 7:

If “type indication” is applied to MyPC directory from Example 5, the output will be the following:

```
(d) - MyPC
(d) - MyDocuments
(!) - DIRECTORY END | MyDocuments
(d) - BilkentStuff
(f) InternshipReport.pdf | File Size: 10 kb
(f) setup.exe | File Size: 5 kb
(f) ToDoList.docx | File Size: 15 kb
(d) - FavoriteMusic
(f) MetallicaTheUnforgiven.mp3 | File Size: 100 kb
(!) - DIRECTORY END | FavoriteMusic
(!) - DIRECTORY END | BilkentStuff
(!) - DIRECTORY END | MyPC
```

Example 8:

If “indentation” and “type indication” is applied together to MyPC directory from Example 5, the output will be the following:

```
(d) - MyPC
(d)  - MyDocuments
(!)  - DIRECTORY END | MyDocuments
(d)  - BilkentStuff
(f)       InternshipReport.pdf | File Size: 10 kb
(f)       setup.exe | File Size: 5 kb
(f)       ToDoList.docx | File Size: 15 kb
(d)       - FavoriteMusic
(f)            MetallicaTheUnforgiven.mp3 | File Size: 100 kb
(!)       - DIRECTORY END | FavoriteMusic
(!)  - DIRECTORY END | BilkentStuff
(!)  - DIRECTORY END | MyPC
```

Hint: Realize that if a row belongs to a beginning position of a directory, that row contains "`-`" character in any case.

Similarly, if a row belongs to an ending position of a directory, that row contains "`DIRECTORY END`" string.

Similarly, if a row belongs to a file, that row contains "`FileSize: `" string.

Those facts can be combined with Java's *indexOf()* and *substring()* functions.

PART 3

Soon after, the company decides to apply a "memory representation" tool for the file system. In this part, you will implement *applyMemoryRepresentation()*, *adjustMemoryRepresentation()* functions to the *directory* object.

The idea is there will be two memory representation methods and the user will be able to select either one of those methods by calling *adjustMemoryRepresentation()* function. When the user call *applyMemoryRepresentation()* function, the selected memory representation method will be applied. In the following, those methods will be explained:

For both memory representations: the beginning of a *directory* will be represented with "`[`" and the end of a *directory* will be represented with "`]`". Similarly, the beginning of a *file* will be represented with "`(`" and the end of a *file* will be represented with "`)`". Similarly, the beginning of the memory will be represented with "`{`" and the ending of the memory will be represented with "`}`".

- **Memory Representation 1: Size-Based Modeling**

In this model, file sizes will be given between parentheses that are opened for the *file* object.

Example 9:
If MyPC directory from Example 5 is represented with Size Based Modeling, the output will be the following:

Given:

```
- MyPC
- MyDocuments
- DIRECTORY END | MyDocuments
- BilkentStuff
InternshipReport.pdf | FileSize: 10 kb
setup.exe | File Size: 5 kb
ToDoList.docx | File Size: 15 kb
- FavoriteMusic
MetallicaTheUnforgiven.mp3 | File Size: 100 kb
- DIRECTORY END | FavoriteMusic
- DIRECTORY END | BilkentStuff
- DIRECTORY END | MyPC
```

Output:

```
{ [ [ ] [ (10) (5) (15) [ (100) ] ] ] }
```

- **Memory Representation 2: Type-Based Modeling**

In this modeling, there are two important rules:

Rule 1: Between the parentheses of the *file* object, there should "`F`" character.

Rule 2: After the opening parenthesis of the *directory* object, there should be a "`D<depth>`" structure where `<depth>` will be zero for the outside directory and incremented when entered into a new directory (see Example 10)).
Rule 3: After the closing parenthesis of the *directory* object, there should be an "`EOD<depth>`" structure where `<depth>` will be zero for the outside directory and incremented when entered into a new directory (see Example 10)). "EOD" corresponds to the "End of Directory".
Example 10:
If MyPC directory from Example 5 is represented with Type Based Modeling, the output will be the following:

Given:

```
- MyPC
- MyDocuments
- DIRECTORY END | MyDocuments
- BilkentStuff
InternshipReport.pdf | File Size: 10 kb
setup.exe | File Size: 5 kb
ToDoList.docx | File Size: 15 kb
- FavoriteMusic
MetallicaTheUnforgiven.mp3 | File Size: 100 kb
- DIRECTORY END | FavoriteMusic
- DIRECTORY END | BilkentStuff
- DIRECTORY END | MyPC
```

Output:

```
{ [D0 [D1 EOD1] [D1 (F) (F) (F) [D2 (F) EOD2] EOD1] EOD0] }
```

End of the Problem