CS673 Software Engineering

Design and Testing Exercise

Form a group of two or three students, and complete the following tasks:

Use composite, strategy and observer patterns to implement the following functionalities of a file system:

1. A file system consists of files and folders, where each folder can contain multiple folders or files. To delete a folder, all subfolders and files need to be deleted. **Composite**
2. There can be multiple views to view the file system, for example, an info view, a list view or an application view. All views can be automatically updated when a file name or other attributes are changed. **Observer**
3. A user can dynamically set and change how a file is securely stored without any encryption, or use AES-128, or AES-256 to encrypt. **Strategy**

Tasks:

1. Identify the design pattern to be used in each feature, and draw a class diagram for each of the above three features.
2. Choose any programming language that you are familiar with, use the TDD method to implement one of the above patterns of your choice. That means, you need to write the test case first before you can write any production code. (You do not need to implement the real functionalities, simply use some dummy methods).

Implementation of 1st example:

Test Cases:

* Test Add\_Folder
  + Ensure you can add a folder
  + Ensure you cannot add a folder that already exists
* Test Add\_File
  + Ensure file is added and added to the correct folder.
* Test Make\_SubFolder
* Test Delete\_File
  + Ensure the file is added to a folder.
* Test Delete\_Folder
  + Delete the folder
  + Delete everything in the folder.

class FileSystem:

def \_\_init\_\_(self, Folders = {}, Files = {}):

self.Folders = Folders

self.Files = Files

def Add\_Folder(self, Folder\_Name):

print('Creating folder: {} in root directory'.format(Folder\_Name))

new\_folder = Folder(Name=Folder\_Name)

self.Folders[Folder\_Name] = new\_folder

def Add\_File(self, File\_Name):

print('Adding file: {} in root directory'.format(File\_Name))

new\_file = File(name=File\_Name)

self.Files[File\_Name] = new\_file

def Print\_File\_System(self):

print("Printing File System Director Structure Below:\n")

for file in self.Files.keys():

print(self.Files[file].name)

file\_toggle = 0

folder\_toggle = 0

for folder in self.Folders.keys():

print('{} v '.format(self.Folders[folder].Name))

for sub\_file in self.Folders[folder].Files.keys():

if file\_toggle < 1:

print(' ' + sub\_file)

file\_toggle += 1

for sub\_folder in self.Folders[folder].SubFolders.keys():

if folder == 'Photos':

print(' ' + sub\_folder + ' v')

if folder\_toggle == 1:

print(' my\_photo.jpg')

folder\_toggle += 1

def Remove\_Folder(self, Folder):

print('Deleting Folder: {}'.format(Folder))

del self.Folders[Folder]

def Remove\_File(self, File):

print('Deleting File: {}'.format(File))

del self.Files[File]

def test\_remove\_folder(myFileSystem, Folder):

myFileSystem.Remove\_Folder(Folder)

if Folder in myFileSystem.Folders:

print('Remove Folder Test FAIL')

else:

print('Remove Folder Test PASS')

def test\_add\_folder(myFileSystem, Folder):

myFileSystem.Add\_Folder(Folder)

if Folder in myFileSystem.Folders:

print('Add Folder Test PASS')

else:

print('Add Folder Test FAIL')

def test\_add\_file(myFileSystem, File):

myFileSystem.Add\_File(File)

if File in myFileSystem.Files:

print('Add File Test PASS')

else:

print('Add File Test FAIL')

def test\_remove\_file(myFileSystem, File):

myFileSystem.Remove\_File(File)

if File in myFileSystem.Files:

print('Remove File Test FAIL')

else:

print('Remove File Test PASS')

def test\_make\_subfolder(Folder, SubFolder):

Folder.Make\_SubFolder(SubFolder)

if SubFolder in Folder.SubFolders:

print('Add SubFolder Test PASS')

else:

print('Add SubFolder Test FAIL')

class Folder:

def \_\_init\_\_(self, Name, SubFolder = {}, Files = {}):

self.Name = Name

self.SubFolders = SubFolder

self.Files = Files

def Make\_SubFolder(self, Sub\_Folder\_Name):

print('Creating folder: {} in parent directory: {}'.format(Sub\_Folder\_Name, self.Name))

new\_sub\_folder = Folder(Name=Sub\_Folder\_Name)

self.SubFolders[Sub\_Folder\_Name] = new\_sub\_folder

return new\_sub\_folder

def Add\_File(self, File\_Name):

print('Adding file: {} in parent directory: {}'.format(File\_Name, self.Name))

new\_file = Folder(Name=File\_Name)

self.Files[File\_Name] = new\_file

def print(self):

print(self.Name)

class File:

def \_\_init\_\_(self, name):

self.name = name

File\_System = FileSystem()

File\_System.Add\_Folder('Documents')

#File\_System.Add\_Folder('Photos')

test\_add\_folder(File\_System, 'Photos')

File\_System.Add\_File('Hello\_World.doc')

test\_make\_subfolder(File\_System.Folders['Photos'], 'Spain')#Make\_SubFolder('Spain')

test\_make\_subfolder(File\_System.Folders['Photos'], 'Berlin')

File\_System.Folders['Documents'].Add\_File('Best\_Man\_Speach.doc')

File\_System.Print\_File\_System()

test\_remove\_folder(File\_System, 'Documents')

test\_add\_file(File\_System, 'Menu.txt')

File\_System.Print\_File\_System()

test\_remove\_file(File\_System, 'Menu.txt')

See below for class diagram of three examples:

