General Structure and Design of In Memory File System

The whole design of the file system has been divided into 6 chief classes

- File Attribute
- Text
- Zip
- Drive
- Folder
- File System

File Attribute – It is the main class that holds size, name, type and path properties and respective methods that add the data into them.

Text – It is a derived method that inherits the properties from File Attribute and holds a new property called content which bears the data.

Zip – It inherits from File Attribute and reduces the size of folder and file to half of it.

Folder – It holds hashsets of zip, text and folder and inherits properties from File Attribute

Drive – It holds folder, zip and text and has no parent. It also acquires properties from File Attribute.

File System – It is the driver class that performs the requisite file operations viz, create, delete, move and write to file.

Methods in File System

- fileOperation () – This method allows user to select any of the file operation to be performed.
- fileType () – This method allows user to select the particular file type.
- Create () – This method internally calls isValidPath () method and takes Type, Name, Parent Path and Scanner as arguments.
- isValidPath () – The method takes parent path, type, name and scanner. The method throws exceptions such as IllegalFileOperation, FileAlreadyExistsException (String path) and FileNotFoundException ().

  IllegalFileOperation – In case a driver is added is about to be added to either folder/text/zip.

  FileAlreadyExistsException – If the same driver already exists in the Hash Set of Drivers.

  FileNotFoundException – Executed in pathTraversal() method.

- pathTraversal() – The method takes type, content, hash set of drivers, hash set of folders, array of concatenated path, drive object, zip object, folder object, text object, name and type of the last object.

  If there is no driver, FileNotFoundException is thrown. For text and zip files driver is added into the driveHolder hash set whereas folders are inserted into folderHolder hash set. Each concatenated path index is checked with the subsequent hash sets of the respective type objects. In case of any discrepancy FileNotFoundException is thrown.

  The file or folder is added into the parent folder and updateSize() function is called which uses driveHolder, folderHolder and size to update all the preceding folders and driver of the file/folder mentioned in the path and name.

- checkPath() – It returns an operationFolder class which holds the operation related data. The method is used by delete, write to file and move operations. The method traverses through the given path and checks if the given path is valid by validating each driver, folder and file mentioned in it.

  For delete operation the file is removed from the hash set and the parent folders and drive are updated with new size by using updateSize() method as discussed above.

  For write to file operation the text file is updated and then the size difference is updated across the parent folders and drive using updateSize().

  Move operation only returns the operationFolder class.

- operateMove() – The method uses operationFolders as arguments and throws IllegalFileOperation. The method is elaborated in detail as below.

- operationUpdate() – The method is used by delete and write to file operations and uses checkPath() method.

- moveOperation() – The method uses checkPath() method to validate if the two paths were valid. Once validated it uses operateMove() to perform the move operation.

File Path Format

The file path format is given as under:

/Drive<>/Folder<>/Text<>, or, /Drive<>/Folder<>/Zip<>, or, /Drive<>, or, /Drive<>/Folder<>/Folder<>

<u>Move Operation</u>

Move operation starts with validating the source and destination paths. Move Operation only allows folder/file to move to drive/folder. In case of file source path and file destination path, an exception is thrown. The file replacement and rename operations are not supported by move operation. The move operation is operated in the following way:

1. It is checked if the source path is driver or not, if found to be driver an exception is thrown.
2. If the destination folder path suggests the file is text or zip then again an exception is thrown.
3. If the source path is folder's then check if the parent is folder or drive. Remove the folder from folder hash map of drive/folder. Also the size of the folder is removed from all the ancestral folders and drive. Also it is checked if the destination is folder/drive. The folder map of destination folder/drive is updated with the folder entry. The size of this folder is added to the parent folders and drive.
4. Similar operation is performed in case of source path is of file or zip and the respective hash maps and size are updated.
5. If in case the source folder is actually an ancestor of the destination folder then an exception is thrown once again.