There are two class diagrams EntityClassDiagram and ExceptionsClassDiagram in the zip file. I have explained the move operation in points no 4 and 5. The code is a full implementation.

1. Underlying Logic

The whole file system is a K-ary tree. At the root there's an object of FileSystem class. The FileSystem can have only the Drive's as its children. Hence at the first level we have Drive's as the nodes in the tree. The further lower levels can have Folder's, ZipFile's or TextFile's as nodes.

2. General structure of the code (please refer EntityClassDiagram while reading this section)

- The abstract class **Entity** represents any entity in the file system.
- The abstract class **Container** extends Entity and it serves to represent any entity that can contain other entities.
- The class **TextFile** represents a text file. It extends Entity. I could have created an abstract class to represent all entities that can't contain entities but since in our case there was only one such entity i.e. TextFile, I decided not to.
- Classes **Drive**, **Folder** and **ZipFile** all extend Container. The behavior of the three classes is not very different from each other. The size calculation behavior is different in the case of ZipFiles. I have kept them as three different classes as their behavior is different from the File System's point of view. For example a Drive can contain a ZipFile but a ZipFile cannot contain a Drive.
- The class **FileSystem** extends from Container. I have defined FileSystem as the parent of all the drives. The FileSystem can contain only drives. Only the constructor of FileSystem is public the constructors of other entities such as Drive, Folder, TextFile are not public.
- The class **InMemoryFileSystem** creates an instance of FileSystem class. In order to create, delete entities, it iterates through the entities contained in the FileSystem instance and then calls appropriate methods on them. This class exclusively deals with the class Entity and FileSystem. It doesn't deal with Drive, Folder etc.
- Enum **EntityType** is used for defining all possible values for the type property in Entity class.
- The class **Rules** contains all the rules that the file system is supposed to obey. Whenever a rule is broken an **IllegalFileSystemOperationException** is thrown. The rule broken is passed as a message to the thrown exception object.
- I used the **Driver** class to test the code. I have tested size functionality in it.

3. Exception Handling

Please refer the ExceptionsClassDiagram in order to understand the structure of classes related to exception handling.

4. Assumptions taken

I took a few assumptions while coding the move functionality. I will mention them below. I have also mentioned them in the class InMemoryFileSystem where I took them.

- There's no renaming involved. Hence a drive cannot be moved because that would be essentially a rename operation.
  In other words the following operation is not allowed
  sourcePath   C/pqr/xyz
  destinationPath D/pqr/abc/lmn

  Allowed operation
  sourcePath   C/pqr/xyz
  destinationPath D/pqr/abc/xyz
- Suppose a file xyz has to be moved then the sample sourcePath and destinationPath will be as following
  sourcePath   C/pqr/xyz
  destinationPath D/pqr/abc/xyz
  destinationPath will contain the name of the file being moved, which over here is xyz.

Changing the move code to allow renaming will not take much effort.

5. <u>Functionality</u>

InMemoryFileSystem creates an instance of the class FileSystem. In order to create, delete entities it iterates through the entities contained in the FileSystem instance and then calls appropriate methods on them. This class exclusively deals with the class Entity and FileSystem. It doesn't deal with Drive, Folder etc.
I have described below how the four rules have been enforced.

- Rule A:- Drive, Folder and ZipFile classes extend Container class. Hence they can contain other entities.
- Rule B:- TextFile class does not extend Container. Further methods for adding and deleting a child in the class Textfile straight away throw an IllegalFileSystemOperationException. The exception object contains the message that Rule B has been violated.
- Rule C:- The method create throws an IllegalFileSystemOperationException exception if we try to put a Drive entity into any other entity except the FIleSystem entity. Furthermore the constructor of Drive is not public. Hence the only way the user can create a Drive is by calling the create method.
- Rule D:- The method create throws an IllegalFileSystemOperationException exception if we try to put a non-Drive entity into the FIleSystem entity. Furthermore the constructor of Folder, ZipFile and TextFile are not public. Hence the only way the user can create them is by calling the create method.

In order to **move** an entity I take the entity object which is to be moved and then add it to the entity whose child it should be after the move operation completes (basically copying it to the other location). Once the entity has been successfully added to the new location it is deleted from its earlier location. After this I iterate recursively through the entity and all of its children and correct the value of the path property inside them. The move operation is **atomic**.