File Fixer Documentation

- Introduction: Team Members, Project scope and purpose

- ### Project description

  FileFixer is an object oriented JAVA solution that allows a lecturer to rename a collection of assignment files, according to a particular convention, before uploading them onto the myElearning platform. The core functionality of the application is file renaming. When the main class is executed, the application would search a particular folder, called filesToRename, for a CSV file and the assignment files aforementioned. When these files are found, the application would proceed to rename all assignment files and then place them in a nested folder, called renamedFiles. The application would also generate a list of missing submission files and export that list to a text file.

- ### Team Members
  - Dexter Cain: 816021817
  - Chelsea Joyeau: 816020515
  - Videsh Jagai: 816014860
  - Satash Rampersad: 816020134

- ### Objectives
  - The java application will be able to accept a CSV folder named filesToRename.
  - The java application will be able to read files within the csv folder and determine the various attributes of the file name as they are separated by an underscore.
  - The java application will be able to determine whether a file name conforms to a particular naming convention and if it is valid.
  - The application will be designed to conform to the SOLID design principles.

- ### Constrains
  - The program will be limited to the classes and packages available within the Java Environment.
  - The program can only read files of the .pdf extension that is located in the primary csv folder to be read.

- ### Risks
  - Submission files may become damaged during processing.
  - If the lecturer has multiple feedback files for each student, grades allocated initially may be overwritten by subsequent allocation(s).
  - Corrupted assignment files in the filesToRename folder.

- Assumptions
    - There may be cases where the marked up assignment files, to be renamed, follows and violates the required naming convention of files.
    - All data contained in the CSV file is valid.

- Analysis: Major requirements and use cases, Target students

- Major Requirements
    - Locates and processes zero or more PDF files to be renamed in a particular directory.
    - Locates and extracts relevant data from a csv file.
    - Renames one or more PDF files according to convention (2) on project specification document.
    - Produces a list of missing submission files based on csv student list.
    - Original and renamed files are matched with a student correctly.

- Use Cases
    - Assignment submission files with only the student's name in the filename.
    - Assignment submission files with only the student's ID in the filename.
    - Students who made no file submissions.

- Target Students
    - The FileFixer application is earmarked for students at the tertiary level.

- Design: Design patterns used, Conformance to SOLID, Class Diagram

- Design Patterns Used
    - The design patterns used in the FileFixer application are Chain of Responsibility and Composite. Chain of Responsibility is used in conjunction with Composite in that when a leaf component (such as the StudentIDHandler) gets a request, it may pass it through the chain to another component (such as the StudentNameHandler) if it is unable to process a student's data.

    - How we implemented the Chain of Responsibility Design Pattern

- ❖ 1. We declared a handler interface (Handler) and described the signature of a method (findStudentData(File)) for handling requests. Initially, we
- ❖ 2. We created concrete handler subclasses and implemented their handling methods. The concrete handlers created were StudentIDHandler.java and StudentNameHandler.java. The handlers were tasked with process requests to find a student's data (i.e. their name or ID) in the filename of the file to be renamed. Each handler makes two decisions when receiving said request:
  1. Whether it'll process the request.
  2. Whether it'll pass the request along the chain.

● Conformance to SOLID

In regards to the Single Responsibility Principle, it is noted that a class should have only a single responsibility and we only have one reason to change or modify it's class. Moreover, this was executed by creating separate concrete handler classes namely, StudentNameHandler and StudentIDHandler where each concrete handler class was used for processing student search requests as it pertains to their name and identification number respectively.

In regards to the Open/Closed Principle, it is noted that various classes and methods should be open for extension, but closed for modification. And so, with this principle in mind, with the code development sought to it that the entities that we created were extendable if need be.

In regards to the Interface Segregation Principle, it is said that "Many client-specific interfaces are better than one general-purpose interface." Where "big" interface are to be further split into smaller interfaces until the client of the interface will only know about the methods that are related to them.However, while bearing that in mind, the final development stages sought that there only need be one interface, that is, the "Handler".

In regards to the Dependency Inversion Principle, it is noted that the code development employed solutions to reduce tight coupling in our whereby depending on various level abstractions in completion of this application.

● Class Diagram

- ○ FileFixer/Documentation/Screenshots/Class Diagram FileFixer Application.png
  - ■ Direct LINK: https://github.com/DexterUWI/FileFixer/blob/main/Documentation/Screenshots/Class%20Diagram%20FileFixer%20Application.png

- Implementation: How to run, Setup requirements

- Step Up Requirements
  - Device: Desktop or a Laptop
  - Operating System: Windows OS or Mac OS
  - Visual Studio Code
  - Java SE Development Kit (JDK)

- Source files
  - Click on the “Code” button above and download as a zip file.
  - Choose a destination to save to the zip file to or move it to your preferred location if it is downloaded automatically.
  - Extract the contents of the zip file.
  - Open Visual Studio Code and go to File -> Open Folder. Choose the location where you have the project folder saved and click “Open”
  - Follow the on-screen instructions that follow and if any assistance is needed, go to the “Help” tab in visual studio

- How to run
  - Due to the nature of the application, which relies heavily on the correctness of the various folder paths for the data files, we will give you a thorough walkthrough (inclusive of screenshots) as to how to move from downloading the project folder from this repository to successfully executing the FileFixer.java (main class) on Visual Studio Code *(recommended environment).*

    - Windows OS
      - In Visual Studio Code, go to Settings and search “Copy Relative Path Separator” and ensure that path separation character is a “\” and not a “/”.
      - Press the run code button and observe the output.
    - Mac OS
      - Press the run code button and observe the output.

- Testing and Evaluation: Test Cases and Suites, Demo video link

- Test Suite 1.0 :: ProcessFilesTest
  - Test Case 1.1 public void testgetFoldersExistance()

```
 @Test
    public void testgetFoldersExistance(){
         System.out.println("Testing getFoldersExistance()
method in the ProcessFiles.java class...");
         assertTrue(fileProcessor.getFoldersExistance());
    }
```

- Test Case 1.2 public void testGetCSVFeedbackFile()

```
@Test
    public void testGetCSVFeedbackFile(){
        System.out.println("Testing getCSVFeedbackFile()
method in the ProcessFiles.java class...");
        assertNotNull(fileProcessor.getCSVFeedbackFile());
    }
```

- Test Case 1.3 public void testGetRenamedFiles()

```
@Test
    public void testGetRenamedFiles(){
        System.out.println("Testing getRenamedFiles() method
in the ProcessFiles.java class...");
        assertNotNull(fileProcessor.getRenamedFiles());
    }
```

- Test Case 1.4 public void testGetFilesToRename()

```
@Test
    public void testGetFilesToRename(){
        System.out.println("Testing getFilesToRename() method
in the ProcessFiles.java class...");
        assertNotNull(fileProcessor.getFilesToRename());
    }
```

- Test Case 1.5 public void testRenameFile()

```
 @Test
    public void testRenameFile(){
        System.out.println("Testing renameFile(File, Student)
method in the ProcessFiles.java class...");
        fileProcessor.extractDataFromCSV();
        fileProcessor.appendDataFilesList();

        File f = new File("testing", ".pdf");
        Student student = new Student("Participant 112233",
"Tom Hanks", "816001122");
```

```
        String expResult = "Tom
Hanks_112233_assignsubmission_file_" + f.getName();;
        String actualResult = fileProcessor.renameFile(f,
student);
        assertEquals(expResult, actualResult);
    }
```

- Test Case 1.6 public void testGetFileExtension()

```
@Test
    public void testGetFileExtension(){
        System.out.println("Testing getFileExtension(File)
method in the ProcessFiles.java class...");
        File f = new File("testing", ".pdf");
        String expResult = "pdf";
        String actualResult =
fileProcessor.getFileExtension(f);
        assertEquals(expResult, actualResult);
    }
```

- Test Suite 2.0 :: StudentTest
  - Test Case 1.1 public void testGetFullName()

```
 @Test
    public void testGetFullName(){
        System.out.println("Testing getFullName() method in
the Student class...");
        String expResult = "Tom Hanks";
        String actualResult = student.getFullName();
        assertEquals(expResult, actualResult);
    }
```

  - Test Case 1.2 public void testGetIDNumber()

```
    @Test
    public void testGetIDNumber(){
        System.out.println("Testing getIDNumber() method in
the Student class...");
        String expResult = "816001122";
        String actualResult = student.getIDNumber();
        assertEquals(expResult, actualResult);
    }
```

- Test Case 1.3 public void testGetIdentifier()

```
@Test
    public void testGetIdentfier(){
        System.out.println("Testing getIdentfier() method in
the Student class...");
        String expResult = "112233";
        String actualResult = student.getIdentfier();
        assertEquals(expResult, actualResult);
    }
```

- Test Case 1.4 public void testSetMissing()

```
 @Test
    public void testSetMissing(){
        System.out.println("Testing setMissing(boolean) method
in the Student class...");
        boolean expResult = false;
        student.setMissing(false);
        boolean actualResult = student.getMissing();
        assertEquals(expResult, actualResult);
    }
```

- Demo Video Link:: https://youtu.be/HJzn2jUnDK8