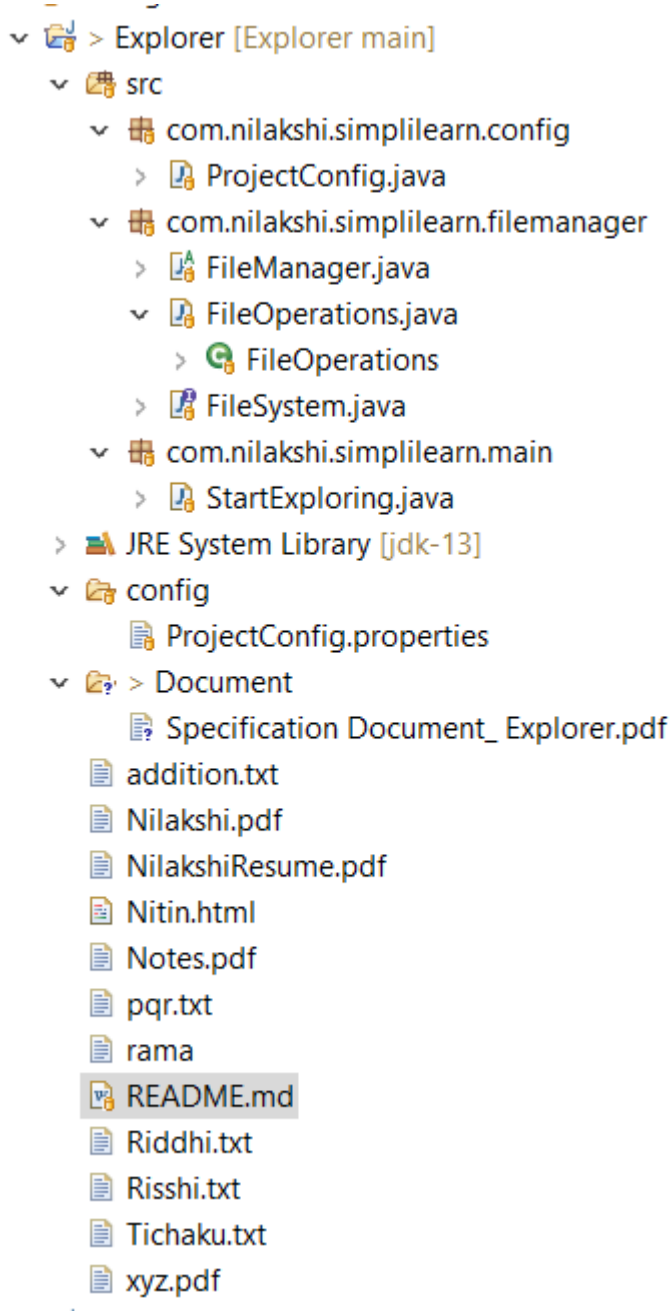
Phase1 Project Source Code

Explorer

By: Nilakshi Nitinkumar Patil
Date: 11/07/2021

Explorer

- **Project Structure:**



1

- **ProjectConfig.java:**
  It reads **./config/ProjectConfig.properties** file and reads the **RootDirectory property** value for performing operations.

```java
public class ProjectConfig {
    private Logger logger = Logger.getLogger(ProjectConfig.class.getName());
    private static ProjectConfig instance = null;
    private String configPath="./config/ProjectConfig.properties";
    private String rootDirectory="";
    private Properties prop;

    private ProjectConfig(){
        BufferedReader reader=null;
        prop = new Properties();
        try {
             reader = new BufferedReader(new FileReader(configPath));
             prop.load(reader);

             //logger.info("Started reading ProjectConfig file");

             if(prop.containsKey("RootDirectory")) {
                 rootDirectory = prop.getProperty("RootDirectory").trim();

                 /* If specified directory is not present, then current directory will be selected as root directory*/
                 if(!Files.exists(Paths.get(rootDirectory))) {
                     rootDirectory = System.getProperty("user.dir");
                 }
             }else {
                 rootDirectory = System.getProperty("user.dir");
             }

        }catch(Exception e) {
            logger.warning("Error while reading ProjectConfig file");
            //e.printStackTrace();
        }finally {
            prop.clear();
            if(reader!=null) {
                try {
                    reader.close();
                } catch (IOException e) {
                    logger.warning("Error caused while closing PropertyReader");
                }
            }
        }
    }
```

```java
    public static ProjectConfig getInstance() {
        if(instance==null)
            instance = new ProjectConfig();
        return instance;
    }

    public String getRootDirectory() {
        return rootDirectory;
    }

}
```

1

- **StartExploring.java:**

  This is the main class of the project. It displays the welcome message initially and then the main menu and file menu based on the user's choice.

Welcome message:

```java
public class StartExploring {

    private static Scanner sc = null;
    public static void main(String[] args) {

        /* Welcome message and developer details */
        System.out.println("*****************************************************************");
        System.out.println("******************** Welcome to EXPLORER ********************");
        System.out.println("*********************** Version: 1.0 ************************");
        System.out.println("***************** Created By: NILAKSHI PATIL *****************");
        System.out.println("*****************************************************************");
        System.out.println();
```

Main menu:

```java
        main_menu:
            while(true) {
                sc = new Scanner(System.in);
                System.out.println("SELECT from following options:");
                System.out.println("[1]\t List files");
                System.out.println("[2]\t File Menu");
                System.out.println("[3]\t Close the Application");
                int choice=0;
                try {
                    choice = sc.nextInt();
                }catch (Exception e) {
                    /* Handled the exception whenever input is other than number */
                    System.err.println("Select Correct option from menu");
                    sc.reset();
                }

                switch(choice) {
                    case 0: continue main_menu;

                    case 1: System.out.println("Listing files:");
                            FileManager.getFileOperation().list().forEach(System.out::println);
                            break;

                    case 2: //File Menu
                            showFileMenu();
                            break;

                    case 3: System.out.println("Are you sure you want to close the application [yes/no]:");
                            String option = sc.next();

                            if(option.equals("yes")) {
                                System.out.println("Application Closed");
                                System.exit(0);
                            }
                            break;

                    /* If you input other than number options provided */
                    default: System.err.println("Invalid option selected! Please, reselect correct option:");
                             break;
```

File Menu:

```
76                System.out.println("FILE MENU Options:");
77                System.out.println("[1]\t Add file");
78                System.out.println("[2]\t Delete file");
79                System.out.println("[3]\t Search file");
80                System.out.println("[4]\t Main Menu");
81                int menu = sc.nextInt();
82
83                String file;
84                switch(menu) {
85                case 1: /* Create a new file */
86                    System.out.println("Enter file name to be added");
87                    file=sc.next();
88                    boolean result=FileManager.getFileOperation().addFile(file);
89                    if(result) {
90                        System.out.println("File successfully created.");
91                    }
92                    break;
93
94                case 2: /* Delete a file */
95                    System.out.println("Enter file name to be deleted");
96                    file=sc.next();
97                    boolean result1 = FileManager.getFileOperation().deleteFile(file);
98
99                    if(result1) {
100                       System.out.println("File successfully deleted.");
101                   }
102
103                   break;
104
105               case 3: /* Search a file */
106                   System.out.println("Enter file name to be searched");
107                   file=sc.next();
108                   boolean status=FileManager.getFileOperation().searchFile(file);
109                   if(status) {
110                       System.out.println("File is present");
111                   }else {
112                       System.out.println("File not found");
113                   }
114                   break;
115
116               case 4: /* Back to main menu */
117                   return;
```

- **FileSystem.java:**

  Interface with file operation abstract methods.

```
5
6  public interface FileSystem {
7
8      public List<String> list();
9
10     public boolean addFile(String file);
11
12     public boolean deleteFile(String file) throws FileNotFoundException;
13
14     public boolean searchFile(String file);
15 }
16
```

- **FileOperations.java:**
  FileOperations implements the FileSystem interface and provides implementation for it's abstract methods.

  list() and addFile() implementation:

```java
public class FileOperations implements FileSystem{

    private File currentDir = null;

    protected FileOperations(String filePath) {
        this.currentDir = new File(filePath);
    }

    @Override
    public List<String> list() {
        List<String> currentDirFiles = Stream.of(currentDir.listFiles())
                .filter(f -> f.isFile())
                    .sorted()
                    .map(f->f.getName())
                    .collect(Collectors.toList());
        return currentDirFiles;
    }

    @Override
    public boolean addFile(String fileName) {
        File file = null;
        try {
            file = new File(fileName);
            if(!file.exists())
                return file.createNewFile();
            else
                throw new FileAlreadyExistsException("File Already exists");
        }catch(IOException e) {
            System.err.println(e.getMessage());
            return false;
        }
    }
```

deleteFile() implementation

```
@Override
public boolean deleteFile(String fileName) throws FileNotFoundException{
    File file = null;
    try {
        file = new File(fileName);
        if(Files.exists(Paths.get(fileName))) {
            return file.delete();
        }else
            throw new FileNotFoundException("File not found");
    }catch(SecurityException | IOException e) {
        System.err.println(e.getMessage());
    }
    return false;
}
```

searchFile() implementation using Binary search algorithm

```
@Override
public boolean searchFile(String fileName){
    List<String> sortedFiles = list();

    return binarySearch(sortedFiles, fileName);
}

private boolean binarySearch(List<String> fileList, String fileName) {
    int start=0, end=fileList.size()-1;

    // base case 1
    if(fileList.isEmpty())
        return false;

    // base case 2
    if(start==end) {
        return fileList.get(start).equals(fileName);
    }

    while(start<=end) {
        int mid = (start+end)/2;

        if(fileList.get(mid).equals(fileName)) {
            return true;
        }else if(fileList.get(mid).equalsIgnoreCase(fileName)) {
            return false;
        }

        if(fileList.get(mid).toLowerCase().compareTo(fileName.toLowerCase())<0) {
            start = mid + 1;
        }

        if(fileList.get(mid).toLowerCase().compareTo(fileName.toLowerCase())>0) {
            end = mid -1;
        }
    }

    return false;
}
```

- **FileManager.java:**

  To achieve abstraction and dynamic binding

```
5  public abstract class FileManager{
6
7      private static String filename = ProjectConfig.getInstance().getRootDirectory();
8      private static FileSystem fileSystem = null;
9
10     public static FileSystem getFileOperation() {
11         if(fileSystem == null)
12             fileSystem = new FileOperations(filename);
13         return fileSystem;
14     }
15 }
```