```java
package use_case_controller;

import java.io.BufferedReader;

/**
 * Class FileIO responsible for reading/writing files.
 * @author Daria Vekic (Student ID: 586661)
 *
 */
public class FileIO {

    /**
     * Method to check if the required Paths and Files exist in user's storage.
     * @param dirPath the path of the directory structure required to maintain data
     * @param filePath the path to the files required to maintain data
     * @return absPath the path to the files
     */
    public Path checkFiles(Path dirPath, Path filePath) {
        Path absPath = dirPath.resolve(filePath);
        try {
            if(Files.notExists(dirPath))
                Files.createDirectories(dirPath); //if path doesn't exist create the directory structure
            //endif
            if(Files.notExists(absPath))
                Files.createFile(absPath); //if the file doesn't exist create the file
        }//end try
        catch (IOException x) {
            System.err.println(x);
            return null;
        }//end catch
        return absPath;
    } //end method checkFiles

    /**
     * Method to control routine of creating file paths if they don't exist.
     * Invoked when program is launched to allow for data persistence.
     */
    public void createPaths() {
        Path p = Paths.get("C:/McRae&DickCaseManagementSystem"); //the directory
        Path usersfilePath = Paths.get("Users.txt"); //the text file for login data
        Path users;
        users = checkFiles(p, usersfilePath);
    } //end method createPaths

    /**
     * Method to read in login credentials from text file.
     * @return map the HashMap containing login credentials.
     */
    public HashMap<String, String> readFromFile() {
        HashMap<String, String> map = new HashMap<String, String>();
        String line = ""; //stores each line read from file
        //Path object to hold absolute file path returned from static method get() from Path class
        //This static call returns an instance of an absolute file path
        Path path = Paths.get("C:/McRae&DickCaseManagementSystem/Users.txt");
```

```
67          try {
68              //Use Files class to return a BufferedReader object from path value supplied
69              //Character set is defined
70              BufferedReader fileInput = Files.newBufferedReader(path, Charset.forName("ISO-8859-1"));
71              line = fileInput.readLine();
72              while(line != null) {
73                  //split the line by the comma
74                  String[] data = line.split(", ");
75                  //Key is username
76                  String username = data[0].trim();
77                  String pw = data[1].trim();
78                  //put Key and Value into the map
79                  if(!username.equals("") && !pw.equals(""))
80                      map.put(username, pw);
81                      line = fileInput.readLine();
82              } //end while
83              fileInput.close();
84          } catch (FileNotFoundException e) {
85              System.out.println("File not found.");
86          } //end catch
87          catch(EOFException eofe) {
88              System.out.println("No more lines to read.");
89          } //end catch
90          catch(IOException ioe) {
91              System.out.println("Error reading file.");
92          } //end catch
93          return map;
94      } //end method readFromFile
95
96      /**
97       * Method to write updated login credentials to file.
98       * @param logins the Map to be written to file.
99       */
100     public void writeToFile(HashMap<String, String> logins) {
101         File file = new File("C:/McRae&DickCaseManagementSystem/Users.txt");
102         BufferedWriter bf = null;
103         try {
104             bf = new BufferedWriter(new FileWriter(file));
105             for(Map.Entry<String, String> entry : logins.entrySet()) {
106                 bf.write(entry.getKey() + ", " + entry.getValue()); //write the Key and Value to file separated with comma
107                 bf.newLine(); //take a new line
108             } //endfor
109             bf.flush();
110         } //end try
111         catch(IOException e) {
112             e.printStackTrace();
113         } //end catch
114         finally {
115             try {
116                 bf.close();
117             } //end try
118             catch(Exception e) {
119                 System.out.println("Something went wrong.");
120             } //end catch
```

```
121         } //end finally
122     } //end method writeToFile
123 } //end class FileIO
```