Using Built-in Packages and Java API Classes

Project Abstract

The purpose of this project is to demonstrate how to write to a file in Java using the `FileWriter` class. This task helps understand the process of creating a file if it doesn't exist, writing text to a file, and handling errors related to file writing.

This project focuses on:
1. Understanding how to create and write to files in Java using the `FileWriter` class.
2. Checking if the file already exists before writing to it.
3. Handling file-writing errors using `try-catch` blocks.
4. Demonstrating the usage of the `write()` method to write data to the file.

Tasks Overview

Task 1: Create a File and Write to It Using the `FileWriter` Class

ObjectiveObjective: Create a file if it doesn't exist and write some text to it using the `FileWriter` class in Java.

Detailed Description: In this task, you will use the `File` class to create a file named "writeFile.txt". If the file doesn't already exist, it will be created. Then, you will use the `FileWriter` class to write a line of text to the file. After writing the content, you will close the file writer and print a success message.

Steps:
1. **Create a File Instance:**
   - Declare a `File` object named `file` using the constructor `new File("writeFile.txt")`. This will represent the file "writeFile.txt", which you will attempt to write to.

2. **Check if the File Exists:**
   - Use the `exists()` method to check if the file already exists. If it does not exist, call the `createNewFile()` method to create the file. If the file is created, print the message "File created: writeFile.txt".

3. **Write to the File:**
   - Use the `FileWriter` class to open the file for writing. Declare a `FileWriter` object named

`writer` and use the constructor `new FileWriter(file)`.
   - Use the `write()` method of the `FileWriter` class to write a line of text to the file: "This is a sample text written to the file."

4. **Close the File Writer:**
   - Once writing is complete, call the `close()` method on the `FileWriter` object to save and close the file.
   - Print the message "Successfully wrote to the file." to confirm the writing operation was successful.

5. **Error Handling:**
   - Wrap the file creation and writing process in a `try-catch` block to handle potential `IOException`. - If an error occurs while writing to the file, catch the `IOException` and print the message "An error occurred while writing to the file." followed by the exception stack trace using `e.printStackTrace()`.

Execution Steps to Follow:

1. **All actions like build, compile, running application, running test cases will be through Command Terminal.**
2. **To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) Terminal New Terminal.**
3. **This editor Auto Saves the code.**
4. **If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.**
5. **These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.**
6. **To run your project use command:**
   mvn compile exec:java -Dexec.mainClass="com.yaksha.assignment.FileWriting"
7. **To test your project test cases, use the command**
   mvn test

8. **You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.**