

## Exception Handling – Outputs and Explanation – Arthisree Saraswathi Rajamanickam-7216696

### 1. Energy Management Systems Main Class

This main class coordinates the various system functions, including log file management, metadata updates, data exchange, and exception handling. It showcases how different parts of the system interact.

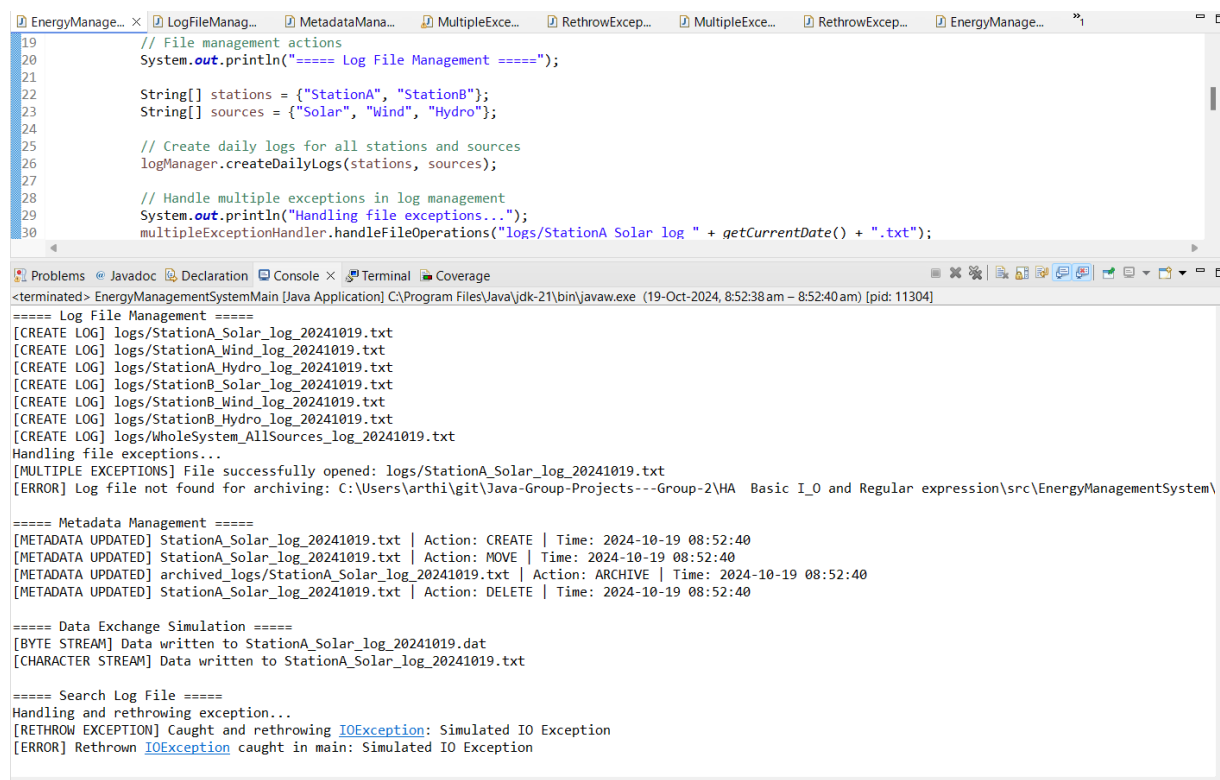
Create Daily Logs: `logManager.createDailyLogs(stations, sources);`

Handle File Operations: `multipleExceptionHandler.handleFileOperations("logs/StationA_Solar_log_" + getCurrentDate() + ".txt");`

Archive Log Files: `logManager.archiveLog("archived_logs/StationA_Solar_log_" + getCurrentDate() + ".txt");`

Explanation: Each of these lines manages different aspects, from creating logs to handling file exceptions and archiving them correctly.

In the main class, the output related to exception handling provides insights into how exceptions are managed:



```
19 // File management actions
20 System.out.println("==== Log File Management =====");
21
22 String[] stations = {"StationA", "StationB"};
23 String[] sources = {"Solar", "Wind", "Hydro"};
24
25 // Create daily logs for all stations and sources
26 logManager.createDailyLogs(stations, sources);
27
28 // Handle multiple exceptions in log management
29 System.out.println("Handling file exceptions...");
30 multipleExceptionHandler.handleFileOperations("logs/StationA_Solar_log_" + getCurrentDate() + ".txt");
```

```
<terminated> EnergyManagementSystemMain [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe (19-Oct-2024, 8:52:38am - 8:52:40am) [pid: 11304]
==== Log File Management =====
[CREATE LOG] logs/StationA_Solar_log_20241019.txt
[CREATE LOG] logs/StationA_Wind_log_20241019.txt
[CREATE LOG] logs/StationA_Hydro_log_20241019.txt
[CREATE LOG] logs/StationB_Solar_log_20241019.txt
[CREATE LOG] logs/StationB_Wind_log_20241019.txt
[CREATE LOG] logs/StationB_Hydro_log_20241019.txt
[CREATE LOG] logs/WholeSystem_AllSources_log_20241019.txt
Handling file exceptions...
[MULTIPLE EXCEPTIONS] File successfully opened: logs/StationA_Solar_log_20241019.txt
[ERROR] Log file not found for archiving: C:\Users\arthis\git\Java-Group-Projects--Group-2\HA Basic I_O and Regular expression\src\EnergyManagementSystem\

==== Metadata Management =====
[METADATA UPDATED] StationA_Solar_log_20241019.txt | Action: CREATE | Time: 2024-10-19 08:52:40
[METADATA UPDATED] StationA_Solar_log_20241019.txt | Action: MOVE | Time: 2024-10-19 08:52:40
[METADATA UPDATED] archived_logs/StationA_Solar_log_20241019.txt | Action: ARCHIVE | Time: 2024-10-19 08:52:40
[METADATA UPDATED] StationA_Solar_log_20241019.txt | Action: DELETE | Time: 2024-10-19 08:52:40

==== Data Exchange Simulation =====
[BYTE STREAM] Data written to StationA_Solar_log_20241019.dat
[CHARACTER STREAM] Data written to StationA_Solar_log_20241019.txt

==== Search Log File =====
Handling and rethrowing exception...
[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception
[ERROR] Rethrown IOException caught in main: Simulated IO Exception
```

#### Multiple Exception Handling Output:

When trying to open files, if a `FileNotFoundException` occurs, the system catches and logs it:

[MULTIPLE EXCEPTIONS] Caught `FileNotFoundException` for file:  
`logs/StationA_Solar_log_20241019.txt`

[ERROR] File not found for archiving: `C:\Users\arthis\...`

This shows that the program identifies missing files and logs errors, ensuring the program can continue without crashing.

## Rethrow Exception Handling Output:

The main class handles exceptions that are rethrown by catching them at a higher level:

less[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception

[ERROR] Rethrown IOException caught in main: Simulated IO Exception

This output confirms that exceptions are managed properly, even when they propagate up the call stack, maintaining clear and informative error handling. The test class validates how well MultipleExceptionHandler manages exceptions by simulating scenarios and confirming expected results.

## 2. Handling Multiple exceptions:

The test class validates how well MultipleExceptionHandler manages exceptions by simulating scenarios and confirming expected results.

### MultipleExceptionHandlerTest Class

- testFileNotFoundException  
Verify that a FileNotFoundException is thrown when attempting to access a file that does not exist.
- testExceptionPropagation  
Ensure that exceptions are correctly propagated when accessing another missing file.
- testSimulatedFileAccessError  
Check if a FileNotFoundException is correctly handled when accessing a non-existent or restricted file.
- testDifferentFileFormats  
Confirm that the system throws a FileNotFoundException for invalid file formats.
- testIOExceptionHandling  
Simulate an IOException and verify if it is caught and handled correctly.
- testSecurityExceptionHandling  
Ensure that a SecurityException is thrown for restricted file access.

The screenshot displays the Eclipse IDE interface. The main editor shows the `MultipleExceptionHandlerTest.java` file with the following code:

```
38  
39  
40 @Test  
41 public void testDifferentFileFormats() {  
42     MultipleExceptionHandler handler = new MultipleExceptionHandler();  
43     assertThrows(FileNotFoundException.class, () -> {  
44         handler.handleFileOperations("logs/invalid_format_file.jpg");  
45     });  
46  
47 @Test  
48 public void testIOExceptionHandling() {  
49     MultipleExceptionHandler handler = new MultipleExceptionHandler();  
50     try {  
51         handler.simulateIOException("logs/simulated_io_exception.txt");  
52     } catch (IOException e) {  
53         System.out.println("testIOExceptionHandling caught expected exception.");  
54         assert(e.getMessage().contains("Simulated IO Exception"));  
55     }  
56 }
```

The left sidebar shows the Package Explorer with the `MultipleExceptionHandlerTest` class selected. The bottom pane shows the Test Results for the `MultipleExceptionHandlerTest` class, indicating that all tests passed successfully.

Test Results:

- testIOExceptionHandling (0.000 s)
- testDifferentFileFormats (0.000 s)
- testExceptionPropagation (0.000 s)
- testSimulatedFileAccessError (0.000 s)
- testSecurityExceptionHandling (0.000 s)
- testFileNotFoundException (0.000 s)

The bottom pane also shows the console output, which includes the following messages:

```
<terminated> MultipleExceptionHandlerTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe (19-Oct-2024, 8:54:00 am - 8:54:02 am) [pid:  
testIOExceptionHandling caught expected exception.  
[MULTIPLE EXCEPTIONS] Caught FileNotFoundException for file: logs/invalid_format_file.jpg  
[MULTIPLE EXCEPTIONS] Caught FileNotFoundException for file: logs/another_missing_file.txt  
[MULTIPLE EXCEPTIONS] Caught FileNotFoundException for file: non_existent_or_restricted_file.txt  
[MULTIPLE EXCEPTIONS] Caught FileNotFoundException for file: logs/non_existent_file.txt
```

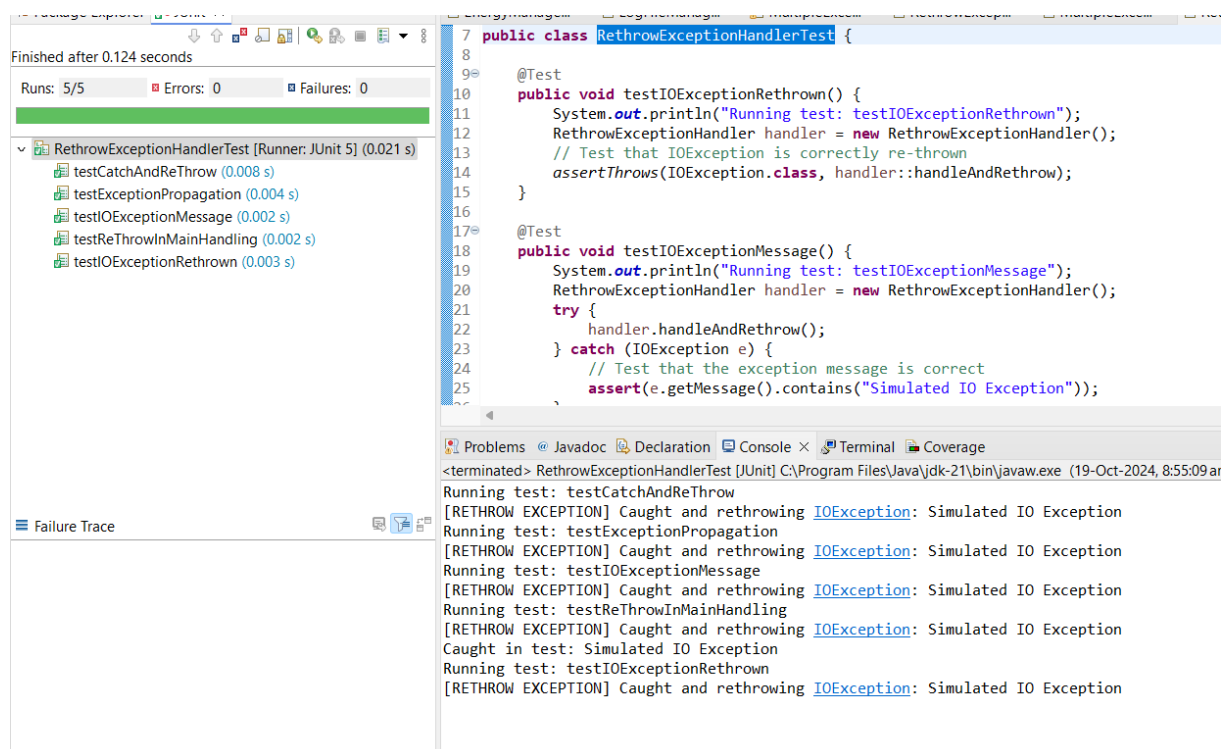
### 3. Rethrow Exception:

This class demonstrates how to handle exceptions and then rethrow them. It helps ensure exceptions propagate correctly when necessary. This test validates how `RethrowExceptionHandler` deals with rethrown exceptions.

#### `RethrowExceptionHandlerTest` Class

This test class confirms the behavior of rethrown exceptions, ensuring proper propagation and handling.

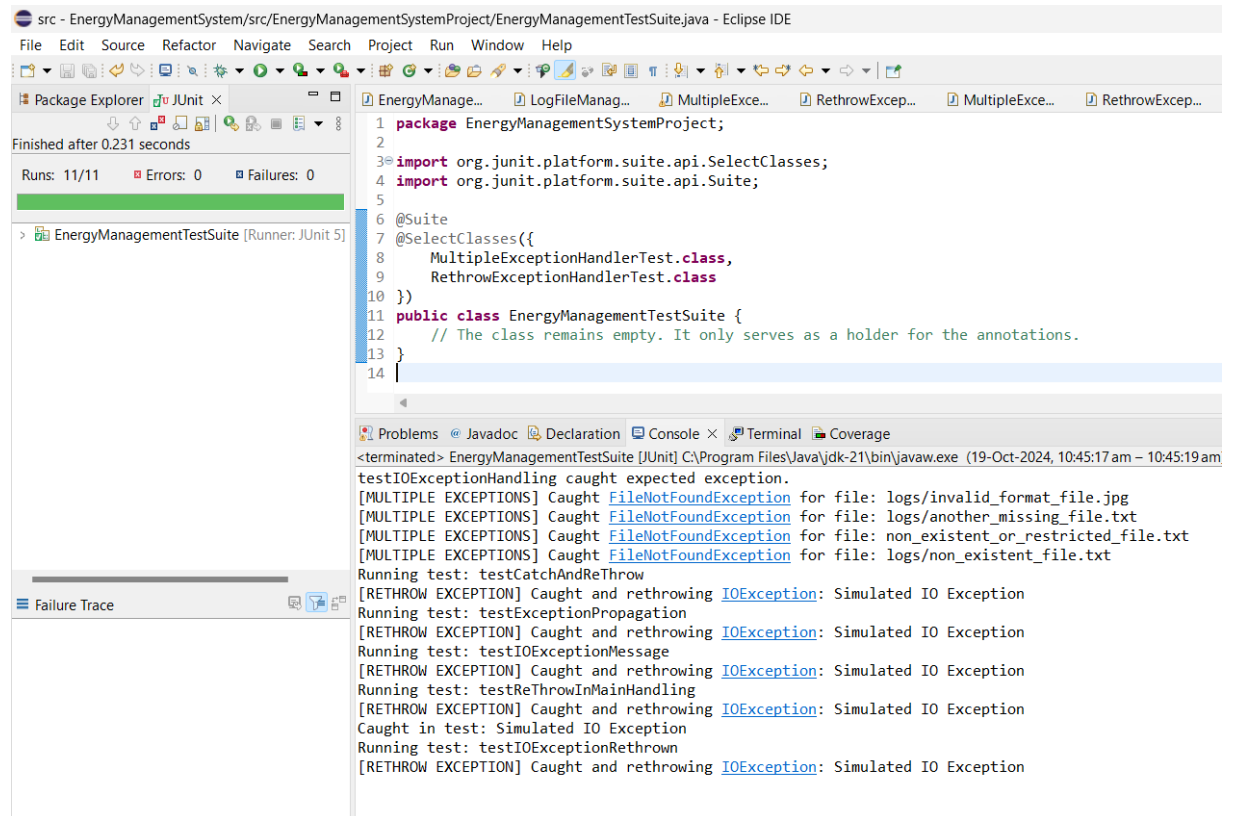
- `testIOExceptionRethrown`  
Purpose: Validate that an `IOException` is rethrown properly after being caught.
- `testIOExceptionMessage`  
Purpose: Check that the correct exception message is maintained when an `IOException` is rethrown.
- `testExceptionPropagation`  
Purpose: Ensure that exceptions continue to propagate correctly even after being rethrown.
- `testCatchAndReThrow`  
Purpose: Verify that exceptions can be caught, processed, and then rethrown effectively.
- `testReThrowInMainHandling`  
Purpose: Test how rethrown exceptions are handled at the main method level.



The screenshot displays an IDE interface with two main panels. The left panel shows the test results for `RethrowExceptionHandlerTest` [Runner: JUnit 5] (0.021 s). The tests listed are `testCatchAndReThrow` (0.008 s), `testExceptionPropagation` (0.004 s), `testIOExceptionMessage` (0.002 s), `testReThrowInMainHandling` (0.002 s), and `testIOExceptionRethrown` (0.003 s). The right panel shows the source code for `RethrowExceptionHandlerTest`. The code includes two test methods: `testIOExceptionRethrown` and `testIOExceptionMessage`. The `testIOExceptionRethrown` method uses `assertThrows(IOException.class, handler::handleAndRethrow)` to verify that an `IOException` is rethrown. The `testIOExceptionMessage` method uses `assert(e.getMessage().contains("Simulated IO Exception"))` to verify the exception message. The bottom panel shows the console output, which includes the following text:   
<terminated> RethrowExceptionHandlerTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe (19-Oct-2024, 8:55:09 ar  
Running test: testCatchAndReThrow  
[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception  
Running test: testExceptionPropagation  
[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception  
Running test: testIOExceptionMessage  
[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception  
Running test: testReThrowInMainHandling  
[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception  
Caught in test: Simulated IO Exception  
Running test: testIOExceptionRethrown  
[RETHROW EXCEPTION] Caught and rethrowing IOException: Simulated IO Exception

## 4. Test suite class to merge them

I also created a test suite class here and added my part. The `EnergyManagementTestSuite` class is used to run multiple test classes together, like `MultipleExceptionHandlerTest` and `RethrowExceptionHandlerTest`, all at once. This makes testing more efficient because you don't have to run each test separately. It's also useful for automated testing, as it ensures all exception-related tests are checked in one go.



The screenshot shows the Eclipse IDE interface. The top toolbar includes menus like File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. Below the toolbar is the Package Explorer on the left, showing the project structure. The main editor displays the `EnergyManagementTestSuite.java` file. The code is as follows:

```
1 package EnergyManagementSystemProject;
2
3 import org.junit.platform.suite.api.SelectClasses;
4 import org.junit.platform.suite.api.Suite;
5
6 @Suite
7 @SelectClasses({
8     MultipleExceptionHandlerTest.class,
9     RethrowExceptionHandlerTest.class
10 })
11 public class EnergyManagementTestSuite {
12     // The class remains empty. It only serves as a holder for the annotations.
13 }
14
```

Below the code editor, the Console tab shows the execution output. The output indicates that the tests passed successfully, with no errors or failures. The output also shows the execution of various tests, including `testIOExceptionHandling`, `testCatchAndReThrow`, `testExceptionPropagation`, `testIOExceptionMessage`, `testReThrowInMainHandling`, `testIOExceptionRethrown`, and `testReThrowInMainHandling`.

Failure Trace