Tutor Marked Exercises 4 Part 2, Questions, Design Document, Testing Plan

TME 4: Test Plan and Design Document (Part 2: Concurrency)

Course: Computer Science 308 – Java for Programmers

Assignment: TME 4

Author: Boris Bojanov

Student ID: 3608550

Date: Jan 31, 2025

---

1. Design Overview

The GreenhouseControls GUI is designed to provide an interactive and user-friendly interface for managing the GreenhouseControls object. The GUI comprises multiple windows, each associated with a separate GreenhouseControls instance, allowing for concurrent monitoring and control. The interface includes a scrollable text area for logs, five control buttons, a pulldown menu with essential functions, and a popup menu for quick access to commands.

Run Instructions:

Compile the program:

`cd "/TME4 Part2/src" && javac GreenhouseGUI.java`

Run:

`java GreenhouseGUI`

Or make sure that you are in the `src` directory when you run the following:
`javac GreenhouseGUI.java`

GUI Components and Functionality

- **Scrollable Text Area:** Displays real-time logs and system messages.
- **Buttons:** Start, Restart, Terminate, Suspend, and Resume functionalities for controlling GreenhouseControls events.
- **Pulldown Menu:** Includes options to create new windows, close windows, open event files, restore from dump files, and exit the application.
- **Popup Menu:** Provides quick access to the same functionalities as the buttons.
- **Keyboard Shortcuts:** Each menu item is associated with a keyboard shortcut for ease of use.

2. Design Challenges

1. Event Synchronization and GUI Updates

One of the key challenges was ensuring proper synchronization between the event threads and the GUI components. Since Java Swing operates on a single-threaded event dispatch model, updating the GUI from event-driven threads required the use of `SwingUtilities.invokeLater()` to prevent concurrency issues.

2. Managing Multiple Windows

Each GUI window is associated with a separate GreenhouseControls instance. Proper handling of window closures and synchronization between multiple instances required careful design to prevent memory leaks and ensure that an application-wide exit does not leave orphaned objects.

3. File Handling for Events and Restoration

Ensuring proper error handling when opening event files and restoring from dump files required implementing exception handling to catch issues such as incorrect file formats, missing files, and file access errors. The GUI provides appropriate error messages when an invalid file is selected.

4. Button and Menu State Management

To prevent invalid operations, buttons and menu options dynamically enable or disable based on the current state of GreenhouseControls. For instance, the **Start** button is disabled once an event is running, and the **Restore** option is disabled when the GreenhouseControls object is active.

5. Implementing a User-Friendly Interface

Designing a clean and intuitive interface while meeting all functional requirements required iterative improvements. The addition of tooltips, appropriate spacing, and distinct menu icons improved usability.

3. Testing Plan

Test Environment

- **Operating System:** Windows 10 / Linux Ubuntu 20.04
- **JDK Version:** Standard Sun Java SDK for the course
- **Testing Tools:** JUnit for unit testing, manual UI interaction

Compile the test file:

```
javac -cp .:lib/junit-platform-console-standalone-1.X.X.jar *.java
```

Run the tests:

```
java -jar lib/junit-platform-console-standalone-1.X.X.jar --class-path . --select-class GreenhouseControlsTest
```

Error Handling Tests

- **Invalid file handling:** Attempting to load a corrupt or incorrectly formatted file should trigger an error message.

- **UI responsiveness:** Ensure the GUI remains responsive even when handling long-running tasks.
- **Concurrent execution:** Verify that multiple GreenhouseControls instances do not interfere with each other.

4. Conclusion

The GreenhouseControls GUI was successfully implemented, providing an intuitive and responsive user interface for managing greenhouse events. Key challenges such as event synchronization, file handling, and UI state management were addressed through careful design and rigorous testing. The GUI is designed to be robust, preventing invalid operations while ensuring a smooth user experience. The testing plan confirms the correctness and usability of the implemented functionality, ensuring compliance with course requirements.

End of Document

NOTE:

The following link is not working properly!!!

“Please see testplan.html for a sample test plan.” (https://scis.lms.athabascau.ca/file.php/422/tme_files/guidelines.htm)

Results in a black page. DegreeWorks does not present me with a way to find the template of a test plan. I have sent Emails requesting an example test plan.

```
Click here to access DegreeWorks using myAU login
```