1. System Overview

The Assignment Checker is a desktop JavaFX application designed to automate the grading of student programming assignments. It allows the user to define test suites (inputs/expected outputs), compile student code found in a directory structure, execute tests, and compare results between different runs.

2. Actors

- **User:** A grader, instructor, or teaching assistant responsible for evaluating Java assignments.
- **System:** The Assignment Checker application.

3. Pre-conditions

- **Student Submissions:** A root folder exists containing subfolders (one per student). Inside each subfolder is the student's .java source code.
- **Java Environment:** The system running the tool has the JDK installed and javac/java are accessible via command line, as the Coordinator uses ProcessBuilder to invoke them.

4. Operational Scenarios

Scenario A: Suite Management (Setup)

Goal: Create a test suite to grade a specific assignment (e.g., "Assignment 3 - Sum Calculator").

1. **Launch Application:** The User starts the application. The UI loads with three main sections: "Test Suite Management", "Execution & Results", and "Result Comparison".
2. **Create Suite:**
    a. User clicks **"Create Suite"**.
    b. System prompts for a name.
    c. User enters "Assignment3".

d. System creates the suite and adds it to the dropdown list.

3. **Add Test Cases:**
    a. User clicks **"Add Test Case"**.
    b. System opens a dialog requesting "Input" and "Expected".
    c. User enters:
        i. **Input:** 1 2 3 (Matches the format required by the uploaded sum3.java example).
        ii. **Expected:** 6.
    d. User clicks OK. System saves the test case to memory.
4. **Manage/Edit Cases:**
    a. User realizes they made a typo and clicks **"Manage Test Cases"**.
    b. System displays a list of current cases.
    c. User selects the case and clicks **"Edit Selected"**, modifies the values, or clicks **"Delete Selected"** to remove it.

Scenario B: Execution (Grading)

Goal: Run the created test suite against a batch of student submissions.

1. **Select Context:**
    a. In the "Execution & Results" pane, User selects "Assignment3" from the **"Select Suite"** dropdown.
2. **Select Root Folder:**
    a. User clicks **"Select Root Folder"**.
    b. User browses to the directory containing student folders (e.g., benlingfalcon06/.../Submissions).
    c. System validates the directory.
3. **Run Tests:**
    a. User clicks the green **"RUN TEST SUITE"** button.
    b. System Internal Process:
        i. Scans the root folder for subdirectories.
        ii. Identifies the .java file containing main for each student.
        iii. **Compiles** the code. If compilation fails, it records "Compilation Failed".
        iv. **Executes** successful builds against the test cases, injecting the input and capturing Standard Output.
        v. Compares Actual Output vs. Expected Output (normalizing newlines).

    c. **Feedback:** The Log Area updates in real-time showing pass/fail percentages for each student.
4. **Output Generation:**
    a. System automatically saves a timestamped text report (e.g., Assignment3_20231209_120000.txt) into the test_results folder.

Scenario C: Reviewing Specific Results

Goal: View the detailed text report generated by a run.

1. User clicks **"View/Reload Specific Result File"**.
2. System opens a file chooser in the test_results directory.
3. User selects a file.
4. System loads the full text content into the main Log Area, showing detailed breakdown of which specific test cases failed for which student.

Scenario D: Result Comparison (Version 2 Feature)

Goal: Compare a previous grading run against a new one (e.g., after students submitted corrections).

1. User expands the **"Result Comparison"** pane.
2. **Select File 1:** User clicks "Select Result File 1" and chooses the report from the initial submission.
3. **Select File 2:** User clicks "Select Result File 2" and chooses the report from the second submission (or regrade).
4. User clicks **"COMPARE RESULTS"**.
5. System parses both text files and generates a side-by-side table in the log area.
    a. **Format:** Student Name | Run 1 Rate | Run 2 Rate | Change
    b. **Example Output:** StudentA | 50.0% | 100.0% | +50.0%.

5. Exception & Edge Case Handling

- **Compilation Failure:** If a student's code (e.g., sum3.java) contains syntax errors, the system captures the compiler error from stderr and marks the student as "Compilation Failed" rather than crashing.

- **Infinite Loops:** *Note: The current Coordinator.java implementation uses p.waitFor() without a timeout. In a real-world scenario, if a student has an infinite loop, the UI will freeze.*
- **Missing Main Method:** If a student folder does not contain a file with public static void main, the log will report "WARNING: No file with 'public static void main' found".
- **Empty Suites:** If the user tries to run a suite with no test cases, the system returns "Test suite is empty" and aborts execution.