Software Testing and Quality Assurance Project

Automated Testing for Student Grade System

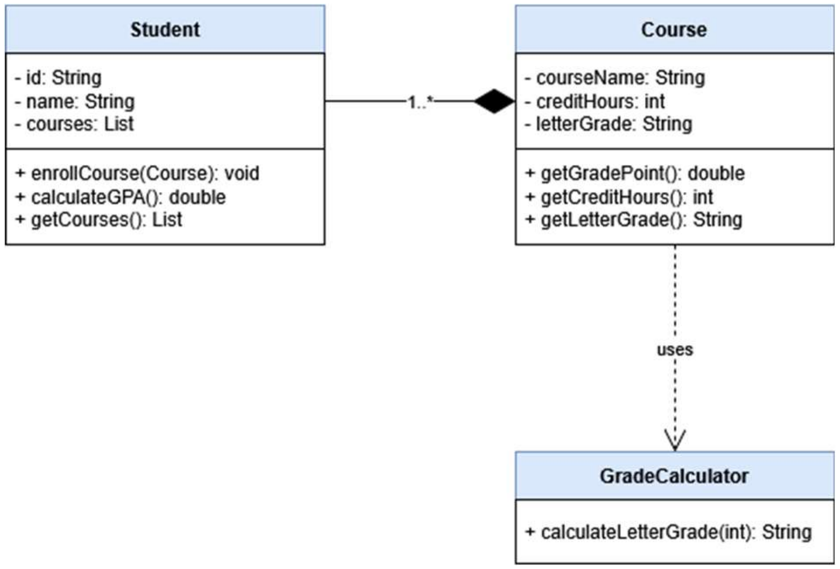
Course: Software Testing and Quality Assurance (Fall 2025)

Team Members:

- Ahmad Al-Dawood (202201115)
- Mohammed Al-Shammasi (202201348)
- Abdulhadi Al-Sultan (202201246)

Project Abstract & Objectives

- **Goal:** Validating the reliability of a Student Grade Calculation System (Classes: Student, Course, GradeCalculator).
- **Approach:** Implemented a comprehensive testing plan following IEEE 829 Standards.
- **Scope:** Functional correctness, boundary handling, and code coverage analysis.



System Environment & Configuration

- System Environment
- **Hardware:** Windows with 8+ GB of ram across all machines. AMD and Intel CPUs tested.
- **Language:** Java JDK 21
- **IDE:** Intellij
- **Build Tool:** Maven

Testing Tools Framework

- **JUnit 5:** Used for the core testing framework.
- **JaCoCo:** Used for code coverage analysis.
- **Surefire 3.1:** Executes the JUnit 5 test suites during the Maven build lifecycle. Also used to generate JaCoCo report.

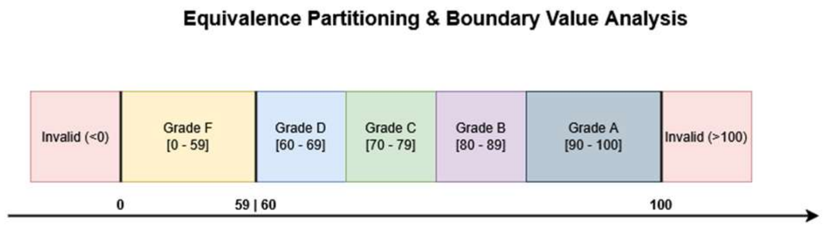
Test Case Design - Black-Box Testing

Equivalence Partitioning (EP):

- *Concept:* Grouping inputs where the system should behave similarly.
- *Example*GradeCalculator inputs 90-100 → "A", 80-89 → "B".

Boundary Value Analysis (BVA):

- *Concept:* Testing the edges of input ranges.
- Example: Testing input **89** (Boundary for B) vs. **90** (Boundary for A).



Test Case Design - White-Box Testing

Basis Path Testing:

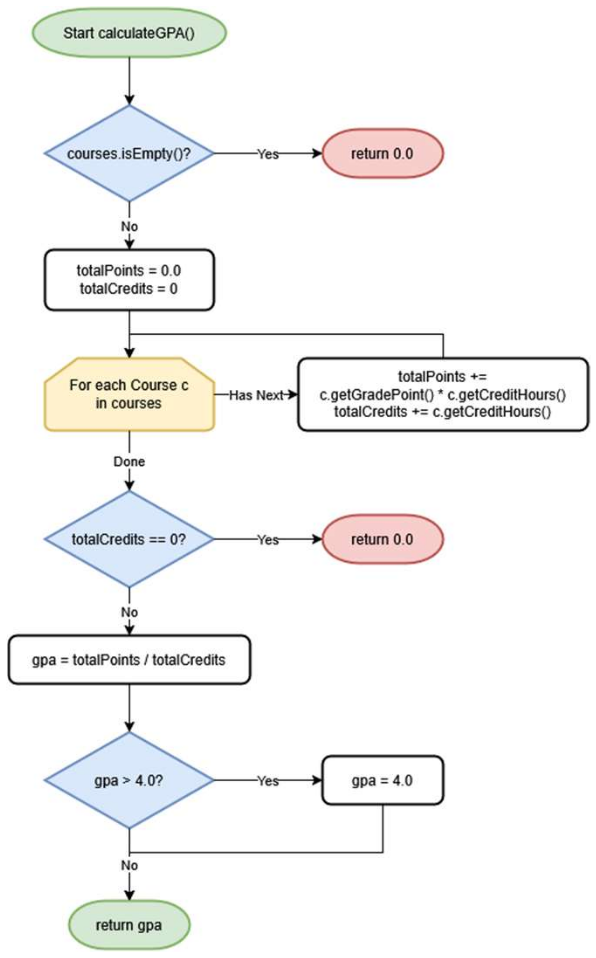
- Designed to execute every independent path through the calculateGPA method.

Decision Coverage:

- Ensuring logical branches (True/False) are executed.

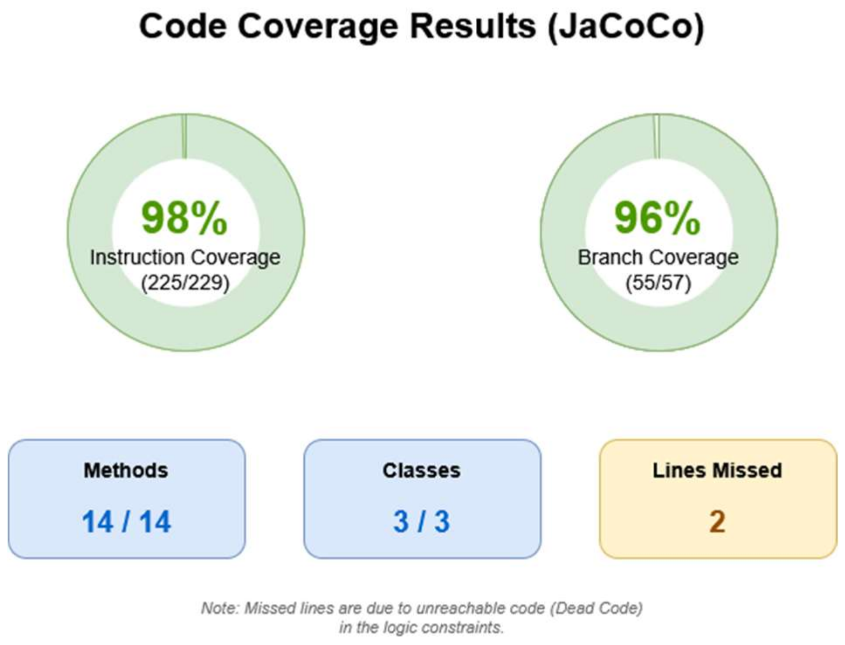
Example from Code:

- Scenario: testCalculateGPACapped (Ensures GPA never exceeds 4.0).
- Scenario: testCalculateGPAInvalidGrade (Ensures invalid grades default to F/0.0).



Code Coverage Analysis (JaCoCo Results)

- **Metrics Evaluated:**
- Instruction Coverage (Lines of code executed).
- Branch Coverage (If/Else paths taken).
- **Target:** Aimed for high percentage coverage

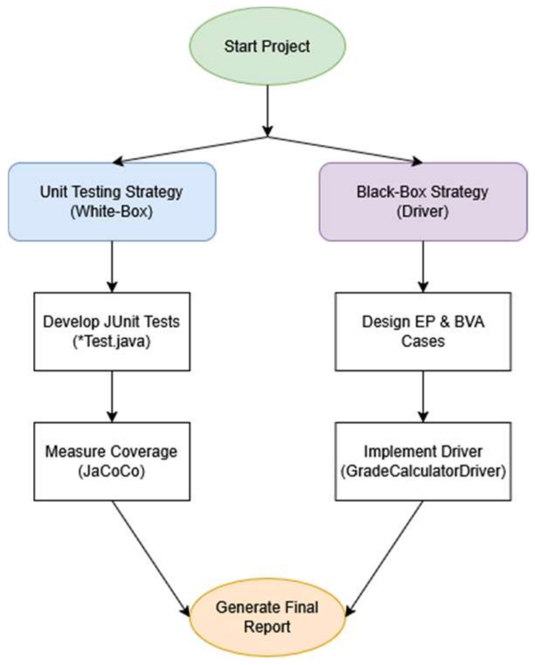


Note: Missed lines are due to unreachable code (Dead Code) in the logic constraints.

Integration Testing

Approach: Bottom-Up Integration.

- **Interaction:** Tested the interaction between Student class and Course class.
- **Test Case:** testCalculateGPAMultipleCourses
- Input: Student enrolls in Course A (3 credits) and Course B (4 credits).
- Verification: Calculated GPA matches the weighted average formula.


Start Project
Unit Testing Strategy (White-Box)
Black-Box Strategy (Driver)
Develop JUnit Tests (*Test.java)
Design EP & BVA Cases
Measure Coverage (JaCoCo)
Implement Driver (GradeCalculatorDriver)
Generate Final Report

Challenges & Solutions

Environment Setup:

- *Challenge:* Configuring JaCoCo in the IDE to generate reports correctly.
- *Solution:* Adjusted Maven build configuration to include the JaCoCo agent.

Test case development:

- *Challenge:* Handling "Dead Code" (Unreachable Logic).
- *Solution:* Analyzed upstream constraints (Constructor sanitization) and documented why 100% coverage was theoretically impossible (e.g., GPA Cap check).
- *Challenge:* Defining effective Black-Box inputs.
- *Solution:* Utilized Boundary Value Analysis (BVA) to specifically target edge cases (e.g., 59/60, 89/90) rather than random sampling.

Conclusion

- Successfully verified the Grade Calculation System.
- Demonstrated robust handling of edge cases (Invalid grades, empty course lists).
- Achieved high code coverage using industry-standard tools (JaCoCo).