# Discovered Bugs

**Gradebook.java**

**Bug #1: Off-by-one — skip last assignment**

```
//if (assignmentCount == assignmentWeights.size() - 1) break; //Fix: Stop it from
breaking the loop on the last index
assignmentCount++;
```

- In this code the if statement was breaking the loop on the last index instead of allowing it to process the last assignment. So, I had to remove/comment out this line to allow for every assignment to be processed correctly.

**Bug #2: Truncates instead of rounding**

```
finalGrade = Math.round(finalGrade * 10) / 10.0; //Fix: round to the nearest
tenth's place
```

In the original code it would convert the final grade to an int rounding it to the nearest whole number. But we all know that every points counts in grades so I changed it to round to the nearest tenth's place by multiplying by 10 before rounding and then dividing by 10.0 to get a rounded double result.

**Bug #3: Cutoff uses '>' not '>='**

```
    public String getLetterGrade(String studentName) {
        double numericGrade = calculateFinalGrade(studentName);
        // BUG #3: Cutoff uses '>' not '>='
        if (numericGrade >= 90) return "A";
        if (numericGrade >= 80) return "B";
        if (numericGrade >= 70) return "C";
        if (numericGrade >= 60) return "D";
        return "F";
    }
```

In this code the it only used > instead of >= making it so that it the final grade was 90 exactly it would return a B instead of an A. Now using >= so that the upper and lower bounds of each letter grade are inclusive.

**Bug #4: Null handling — may crash if student has no record**

```
 StudentRecord student = allStudents.get(studentName);
        if (student == null) {
            throw new IllegalArgumentException("Student not found"); // Fix: if
```

```
student is not throw and error
        }
```

In this code there was no null handling for if a student did not exist in the records. So I added a check to see if the student was null and if so throw an IllegalArgumentException with a message saying the student was not found.

**Bug #5: Missing assignment ignored (should count as zero)**

```
  Double studentScore = student.getAssignmentScore(assignmentName);
            if (studentScore != null) {
                totalPoints += studentScore * assignmentWeight;
            } else {
                totalPoints += 0 * assignmentWeight; // fix: studentScore should
count as zero, not nothing when null
            }
```

In this code it would only check if the student score did not equal null and if it was null it would just keep the null adding nothing to the total points. But instead of adding null I made it so that the student score would count as zero when it was null so that missing assignments would be factored into the final grade as a zero.

**Bug #6: Cached value not cleared when student removed**Z

**BUG #9: Uses == for String comparison (fails on new String)**

```
    public void removeStudentFromGradebook(String studentName) {
        // BUG #9: Uses == for String comparison (fails on new String)
        for (String currentStudent : allStudents.keySet()) {
            if (currentStudent.equals(studentName)) { // Should be equals() Fix:
instead of comparing objects compare strings using .equals()
                allStudents.remove(currentStudent);
                gradeCache.remove(currentStudent);  //Fix: removes student and
cache
                break;
            }
        }
    }
```

In this code it would only remove the students name from the cache leaving its cached grade where you would still call it after removing a studedent. And in this code it used == to compare strings now treating them as objects where they could have the same name but be different objects. So I changed it to use .equals() to compare the string values instead of the objects and also remove the grache cache when removing a student.

**Bug #7: Fails to normalize weight sum (assumes it's 1)**

```
       double finalGrade;
        if (weightTotal > 0) {
            finalGrade = totalPoints / weightTotal;  //Fix instead of directly
assuming the weighted sum to be 1, normalize weight to be over/under 1
        } else {
            finalGrade = 0.0;
        }
```

This code would assume that the total weight was always equal to 1. So you have to normalize the weight total to be over or under 1 when calculating the final grade. So if the wight was over 0 it divides the total points by the weight total and if the total wight is invalid -negative or zero, the final grade was set to 0.0.

**Bug #8: Allows negative scores**

```
    public void addStudentGrade(String studentName, String assignment, double
score) {
        // BUG #8: Allows negative scores
        if (!allStudents.containsKey(studentName)) {
            allStudents.put(studentName, new StudentRecord(studentName));
        }
        if (score < 0 || score > 100) score = 0; //Fix: Set bounds for Score to
not be over 100 or negative
        allStudents.get(studentName).addAssignmentScore(assignment, score);
    }
```

This code would allow the input of negative scores which are impossible to achieve. So i added an if statemtn that checks the bounds of the score to make sure it was valid and it was not in the bounds it set the score to 0.

**StudentRecord.java**

**Bug #10: No reset or clear method — stale data across runs**

```
    public void clearAssignments() {
        assignmentScores.clear(); //Fix Clears all scores and assignments
    }
```

In this Java class there was no method to clear the assignments and scores of a student record. So I added a method that clears all the scores and assignments from the assignmentScores map to prevent stale data across runs.

# My Experience

- A write up of your overall experience. Discuss the code coverage your tests have been able to achieve. If you took any support from any LLM, give credit clearly to the LLM you used and to what degree.

My Overall expiernce with this assignment was kind of interesting as I spent allot of time trying to figure out how to work eclipse with Git. As i would pull the repo and would not be able to run anything no matter what i did like messing with the settings and coverting it to a maven project and adding a package route to run the tests but nothing would work. And plus I though it was interesting how in the demo it called the wrong names of the function and did not give any warnings, so i was like why isnt this demo working lol. So it shows you that you cannot take anything for granted and have to test everything. And I enjoyed this assignmen as this project was well written and had some very well thought out bugs in it making it enjoyable to debug and fix.

For the Code coverage I though that it was interesting how it highlighted the lines that were being covered or not. Making it very clear which lines you needed to write tests for. Where for the initial bug hunting tests I would only test the one grade letter, but for the coverage it forced me to check every letter grade and think about the upper and lower bounds of these tests to make sure that they were all working properly.

Credit: For this assignment I used ChatGPT to help me understand some of the bugs and how to fix them like how to round to the nearest tenth's place and how to normalize weights when calculating the final grade allowing me to better understand the formula and what that truely meant. As i had a test case that did not set an assignment weight but added it to the student grade to only realize that the wight was being normalized when calculating the final grade. Also it helped me understand how to throw an exception when a student was not found in the records. and search for that excention when creating tests to make sure it was being thrown properly.