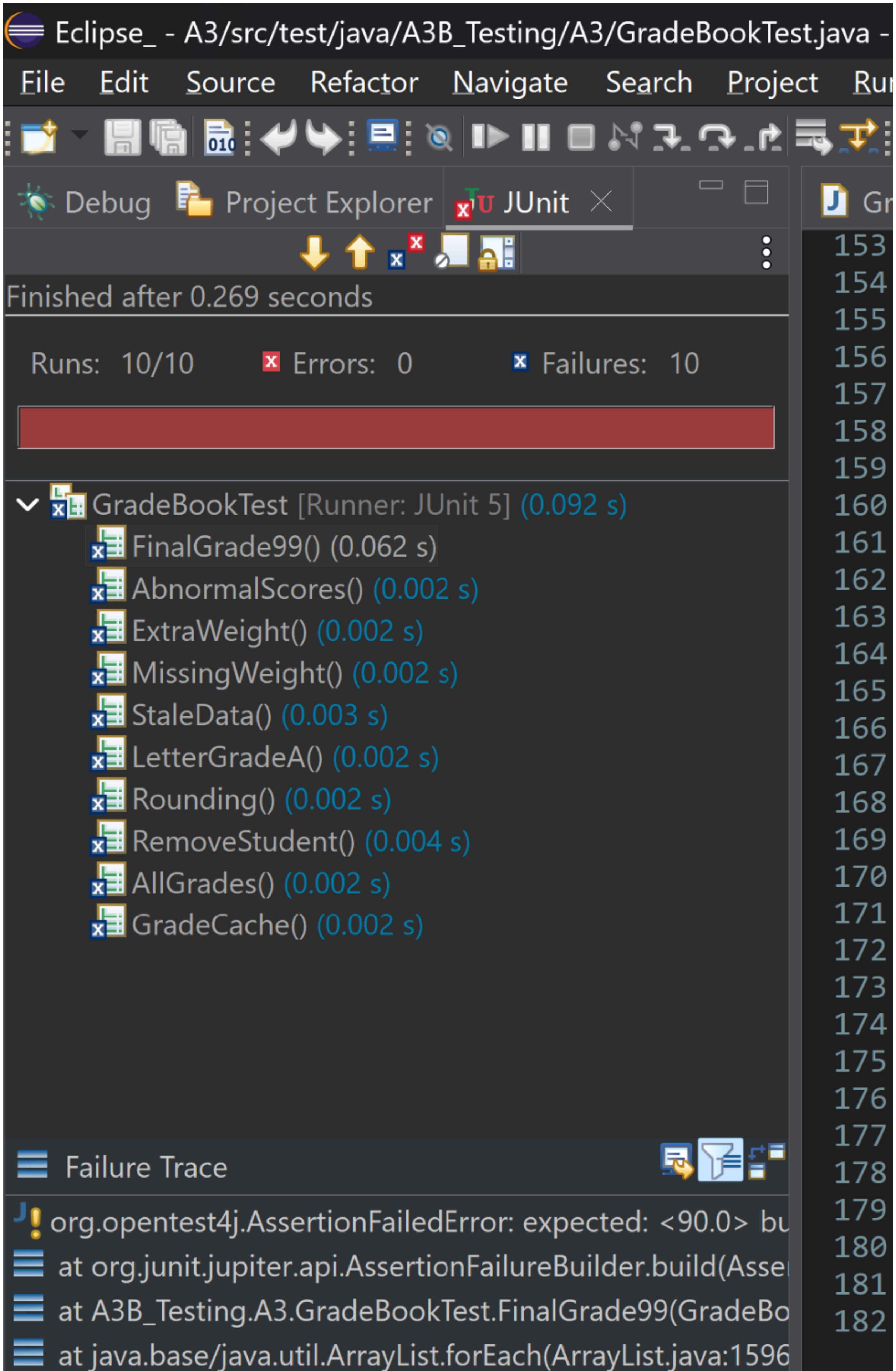
```java
package A3B_Testing.A3;

/**
 * Our gradebook system that calculates final grades and letter grades for students
 * CS 483 - Assignment 3A-3B (Bug Hunt)
 * @author Chase Garnett & Fabrizio Guzzo
 * @version 1.0 10/07/2025
 */

import java.util.*;

public class GradeBook {

    private Map<String, StudentRecord> allStudents;
    private Map<String, Double> assignmentWeights;
    private Map<String, Double> gradeCache;

    public GradeBook() {
        allStudents = new HashMap<>();
        assignmentWeights = new HashMap<>();
        gradeCache = new HashMap<>();
    }

    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
        allStudents.get(studentName).addAssignmentScore(assignment, score);
    }

    public void setAssignmentWeight(String assignment, double weight) {
        assignmentWeights.put(assignment, weight);
    }

    public double calculateFinalGrade(String studentName) {
        // 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
        }

        // BUG #6: Cached value not cleared when student removed
        if (gradeCache.containsKey(studentName)) {
            return gradeCache.get(studentName);
        }

        double totalPoints = 0.0;
        double weightTotal = 0.0;
        int assignmentCount = 0;

        for (Map.Entry<String, Double> weightEntry : assignmentWeights.entrySet()) {
            String assignmentName = weightEntry.getKey();
            double assignmentWeight = weightEntry.getValue();
            weightTotal += assignmentWeight;

            // BUG #1: Off-by-one – skip last assignment
            //if (assignmentCount == assignmentWeights.size() - 1) break; //Fix: Stop it from
            assignmentCount++;

            Double studentScore = student.getAssignmentScore(assignmentName);
            if (studentScore != null) {
                totalPoints += studentScore * assignmentWeight;
            } else {
             totalPoints += 0 * assignmentWeight; // fix: studentScore should count as zero, not
            }
            // BUG #5: Missing assignment ignored (should count as zero)
        }

        // 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
        } else {
            finalGrade = 0.0;
        }

        // BUG #2: Truncates instead of rounding
        finalGrade = Math.round(finalGrade * 10) / 10.0; //Fix: round to the nearest tenth's


        gradeCache.put(studentName, finalGrade);
        return finalGrade;
    }

    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";
    }

    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
                allStudents.remove(currentStudent);
                gradeCache.remove(currentStudent);  //Fix: removes student and cache
                break;
            }
        }
    }

    public Map<String, Double> generateAllFinalGrades() {
        Map<String, Double> finalGrades = new HashMap<>();
        for (String student : allStudents.keySet()) {
            finalGrades.put(student, calculateFinalGrade(student));
        }
        return finalGrades;
    }

    public boolean isStudentEnrolled(String studentName) {
        return allStudents.containsKey(studentName);
    }
}

```

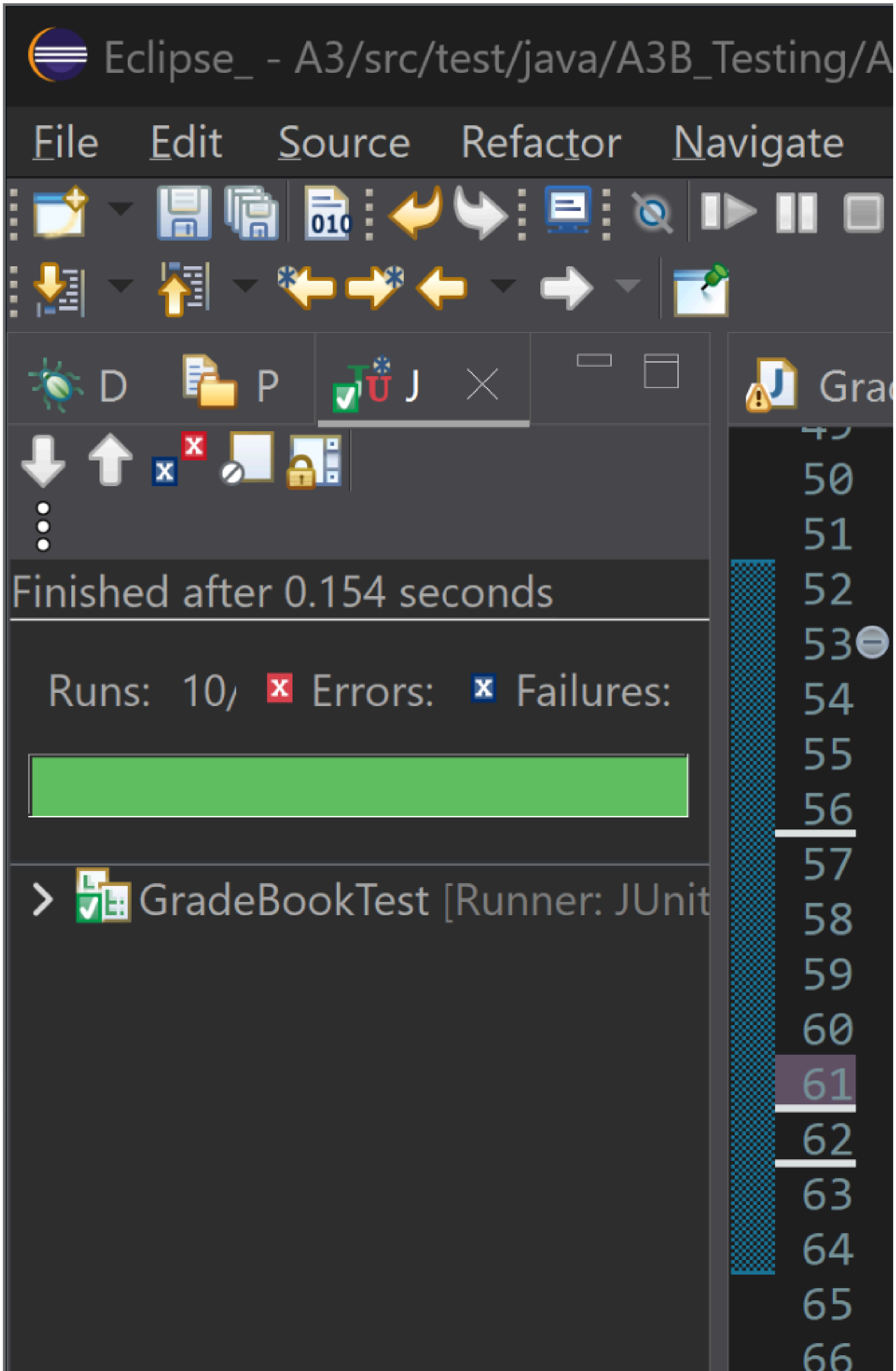
```java
package A3B_Testing.A3;

/**
 * Represents an individual student's set of grades.
 * CS 483 - Assignment 3A-3B (Bug Hunt)
 * @author Chase Garnett & Fabrizio Guzzo
 * @version 1.0 10/07/2025
 */

import java.util.*;

public class StudentRecord {

    private String studentName;
    private Map<String, Double> assignmentScores;

    public StudentRecord(String studentName) {
        this.studentName = studentName;
        this.assignmentScores = new HashMap<>();
    }

    public void addAssignmentScore(String assignmentName, double score) {
        assignmentScores.put(assignmentName, score);
    }

    public Double getAssignmentScore(String assignmentName) {
        return assignmentScores.get(assignmentName);
    }

    public Map<String, Double> getAllAssignmentScores() {
        return assignmentScores;
    }

    public String getStudentName() {
        return studentName;
    }

    // BUG #10: No reset or clear method — stale data across runs
    public void clearAssignments() {
        assignmentScores.clear(); //Fix Clears all scores and assignments
    }
}

```

```java
package A3B_Testing.A3;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.Map;

import static org.junit.jupiter.api.Assertions.*;

class GradeBookTest {

    private GradeBook gradeBook;

    @BeforeEach
    void setUp() {
        gradeBook = new GradeBook();
    }


    /**
     * This Test uses the demo grades to make sure that the letter grade is an A
     * @author JonathanDargakis
     */
    @Test
    void LetterGradeA() {
      gradeBook.setAssignmentWeight("HW1", 0.2);
         gradeBook.setAssignmentWeight("HW2", 0.3);
         gradeBook.setAssignmentWeight("Exam", 0.5);

         gradeBook.addStudentGrade("Alice", "HW1", 99);
         gradeBook.addStudentGrade("Alice", "HW2", 99);
         gradeBook.addStudentGrade("Alice", "Exam", 99);

         String letterGrade = gradeBook.getLetterGrade("Alice");

         assertEquals("A", letterGrade);
    }


    /**
     * This Test ensures that the final calculation of the grade is 99.
     * Making sure that the logic is correct and not being cut off early (off by one)
     * @author JonathanDargakis
     */
    @Test
    void FinalGrade99() {
      gradeBook.setAssignmentWeight("HW1", 0.2);
         gradeBook.setAssignmentWeight("HW2", 0.3);
         gradeBook.setAssignmentWeight("Exam", 0.5);

         gradeBook.addStudentGrade("Alice1", "HW1", 99);
         gradeBook.addStudentGrade("Alice1", "HW2", 99);
         gradeBook.addStudentGrade("Alice1", "Exam", 99);

         //average: 90*0.2 + 90*0.3 + 90*.5 = 99
         //Missing Last Grade: 90*0.2 + 90*0.3 = 49
         double grade = gradeBook.calculateFinalGrade("Alice1");


         assertEquals(99, grade);
    }


    /**
     * This Tests makes sure that the normalization of the weight works properly with a lower
     * * Making sure that the weight is not assumed to be 1
     * @author JonathanDargakis
     */
    @Test
    void MissingWeight() {
      gradeBook.setAssignmentWeight("HW1", 0.1);
         gradeBook.setAssignmentWeight("HW2", 0.1);
      gradeBook.setAssignmentWeight("HW3", 0.3);

         gradeBook.addStudentGrade("Alice2", "HW1", 99);
         gradeBook.addStudentGrade("Alice2", "HW2", 99);
         gradeBook.addStudentGrade("Alice2", "HW3", 99);

         double grade = gradeBook.calculateFinalGrade("Alice2");
         assertEquals(99, grade);
    }


    /**
     * This test tests the normalization of the with with an excess amount - over 1
     * Making sure that the weight is not assumed to be 1
     * @author JonathanDargakis
     */
    @Test
    void ExtraWeight() {
     gradeBook.setAssignmentWeight("Hw1", 0.2);
     gradeBook.setAssignmentWeight("Exam", 0.9);

     gradeBook.addStudentGrade("Alice4", "Hw1", 99);
        gradeBook.addStudentGrade("Alice4", "Exam", 70);

        double grade = gradeBook.calculateFinalGrade("Alice4");
        assertEquals(75.3, grade);


    }


    /**
     * This test makes sure that when calling an older student from another test a grade is not
     * Ensuring there isnt any leftover stale data
     * @author JonathanDargakis
     */
    @Test
    void StaleDataThrowsException() {
        //calling a method that should throw IllegalArgumentException
        assertThrows(IllegalArgumentException.class, () -> {
            gradeBook.getLetterGrade("Alice");
        });
    }


    /**
     * This test Ensures that the grade bounds are working properly
     * As 90 should be an A
     * @author JonathanDargakis
     */
    @Test
    void Rounding() {
     gradeBook.setAssignmentWeight("Hw1", 0.2);
     gradeBook.setAssignmentWeight("Hw2", 0.3);
     gradeBook.setAssignmentWeight("Exam", 0.5);
     gradeBook.addStudentGrade("Alice5", "Hw1", 90);
     gradeBook.addStudentGrade("Alice5", "Hw2", 90);
     gradeBook.addStudentGrade("Alice5", "Exam", 90);

     String letterGrade = gradeBook.getLetterGrade("Alice5");
     assertEquals("A", letterGrade);
    }


    /**
     * This test makes sure that after removing a student that the student is not enrolled and
     * @author JonathanDargakis
     */
    @Test
    void GradeCache() {
     gradeBook.setAssignmentWeight("Hw1", 0.5);
     gradeBook.setAssignmentWeight("Hw2", 0.5);
     gradeBook.setAssignmentWeight("Exam", 0.5); //random 3rb because off by one
     gradeBook.addStudentGrade("Alice6", "Hw1", 90);
     gradeBook.addStudentGrade("Alice6", "Hw2", 90);
     gradeBook.addStudentGrade("Alice6", "Exam", 90);


     gradeBook.removeStudentFromGradebook("Alice6");

     assertFalse(gradeBook.isStudentEnrolled("Alice6"));
     assertThrows(IllegalArgumentException.class, () -> {
            gradeBook.calculateFinalGrade("Alice6");
        });


    }

    /**
     * This test makes sure that it sets invalid scores to 0 and can handle doubles
     * @author JonathanDargakis
     */
    @Test
    void AbnormalScores() {
     gradeBook.setAssignmentWeight("Hw1", 0.2);
     gradeBook.setAssignmentWeight("Hw2", 0.3);
     gradeBook.setAssignmentWeight("Exam", 0.5);
     gradeBook.addStudentGrade("Alice7", "Hw1", -70);
     gradeBook.addStudentGrade("Alice7", "Hw2", 101);
     gradeBook.addStudentGrade("Alice7", "Exam", 75.5);

     double grade = gradeBook.calculateFinalGrade("Alice7");
     assertEquals(37.8, grade);

    }

    /**
     * This test makes sure they you can remove a student by using another string object
     * Ensuring that the comparison is between strings and not objects
     * @author JonathanDargakis
     */
    @Test
    void RemoveStudent() {
     gradeBook.setAssignmentWeight("Hw1", 1);
     gradeBook.addStudentGrade("Alice8", "Hw1", 75);

     String studentToRemove = new String("Alice8"); // another string object to compare
        gradeBook.removeStudentFromGradebook(studentToRemove);

     assertFalse(gradeBook.isStudentEnrolled("Alice8"));

    }


    /**
     * This tests makes sure that all of the grades bounds are working properly
     * @author JonathanDargakis
     */
    @Test
    void AllGrades() {
     gradeBook.setAssignmentWeight("Hw1", 1);
     gradeBook.addStudentGrade("Alice9", "Hw1", 90);

     String letterGradeA = gradeBook.getLetterGrade("Alice9");
     assertEquals("A", letterGradeA);


     gradeBook.addStudentGrade("Bob", "Hw1", 80);

     String letterGradeB = gradeBook.getLetterGrade("Bob");
     assertEquals("B", letterGradeB);

     gradeBook.addStudentGrade("Cat", "Hw1", 70);

     String letterGradeC = gradeBook.getLetterGrade("Cat");
     assertEquals("C", letterGradeC);

     gradeBook.addStudentGrade("Darek", "Hw1", 60);

     String letterGradeD = gradeBook.getLetterGrade("Darek");
     assertEquals("D", letterGradeD);

     gradeBook.addStudentGrade("Eric", "Hw1", 50);

     String letterGradeF = gradeBook.getLetterGrade("Eric");
     assertEquals("F", letterGradeF);

    }



    /**
     * This test ensures that you are able to generate all of the grades at once
     * Showing the number of all the grades and each student that is in the mapping.
     * @author JonathanDargakis
     */
    @Test
    void GenerateAllGrades() {
      // Step 1: Set assignment weights
        gradeBook.setAssignmentWeight("HW1", 0.5);
        gradeBook.setAssignmentWeight("HW2", 0.5);

        // Step 2: Add students and grades
        gradeBook.addStudentGrade("Alice", "HW1", 80.0);
        gradeBook.addStudentGrade("Alice", "HW2", 90.0);

        gradeBook.addStudentGrade("Bob", "HW1", 70.0);
        gradeBook.addStudentGrade("Bob", "HW2", 60.0);

        // Step 3: Generate all final grades
        Map<String, Double> allGrades = gradeBook.generateAllFinalGrades();

        // Step 4: Verify map contains all students
        assertEquals(2, allGrades.size());
        assertTrue(allGrades.containsKey("Alice"));
        assertTrue(allGrades.containsKey("Bob"));

    }



    /**
     * This ensures that when a grade is null it is being treated as an int adding 0 instead of
     * @author JonathanDargakis
     */
    @Test
    void NullScoreAsZero() {

        gradeBook.setAssignmentWeight("HW1", 0.5);
        gradeBook.setAssignmentWeight("HW2", 0.5);

        gradeBook.addStudentGrade("Alice10", "HW1", 80.0);
        //since HW2 is missing score = null = 0

        double finalGrade = gradeBook.calculateFinalGrade("Alice10");

        // = 80*0.5 + 0*0.5 = 40.0
        assertEquals(40.0, finalGrade);
    }

}

```

```java
package A3B_Testing.A3;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.Map;

class StudentRecordTest {

    private StudentRecord student;

    @BeforeEach
    void setUp() {
        student = new StudentRecord("Jonathan");
    }


    /**
     * THis test makes sure that the student name is Jonathan
     * @author JonathanDargakis
     */
    @Test
    void GetStudentName() {
        assertEquals("Jonathan", student.getStudentName());
    }


    /**
     * This test ensures that you are able to add assignments and later call them
     * @author JonathanDargakis
     */
    @Test
    void AddAndGetAssignmentScore() {
        student.addAssignmentScore("HW1", 80);
        student.addAssignmentScore("Exam", 76);

        assertEquals(80, student.getAssignmentScore("HW1"));
        assertEquals(76, student.getAssignmentScore("Exam"));

        // Test getting score for assignment not added
        assertNull(student.getAssignmentScore("HW2")); // Score should be null for missing
    }


    /**
     * This test ensures that a missing assignment score is Null
     * later turned into 0
     * @author JonathanDargakis
     */
    @Test
    void MissingAssignmentScore() {
     student.addAssignmentScore("HW1", 80);
        student.addAssignmentScore("Exam", 76);

        assertNull(student.getAssignmentScore("HW2"));
    }


    /**
     * This test makes sure that you are able to get all of the assignment scores at once
     * verifying the size and contents
     * @author JonathanDargakis
     */
    @Test
    void GetAllAssignmentScores() {
        student.addAssignmentScore("HW1", 56);
        student.addAssignmentScore("HW2", 79);

        Map<String, Double> allScores = student.getAllAssignmentScores();
        assertEquals(2, allScores.size());
        assertEquals(56, allScores.get("HW1"));
        assertEquals(79, allScores.get("HW2"));
    }


    /**
     * This test makes sure that all of the Assignments are removed and cannot be called later
     * @author JonathanDargakis
     */
    @Test
    void ClearAssignments() {
        student.addAssignmentScore("HW1", 89);
        student.addAssignmentScore("Exam", 82);

        assertEquals(2, student.getAllAssignmentScores().size());

        student.clearAssignments();

        assertEquals(0, student.getAllAssignmentScores().size());
        assertNull(student.getAssignmentScore("HW1"));
        assertNull(student.getAssignmentScore("Exam"));
    }


}
```


Eclipse_ - A3/src/test/java/A3B_Testing/A3/GradeBookTest.java -
File Edit Source Refactor Navigate Search Project Ru
Debug
Project Explorer
JUnit
Gr

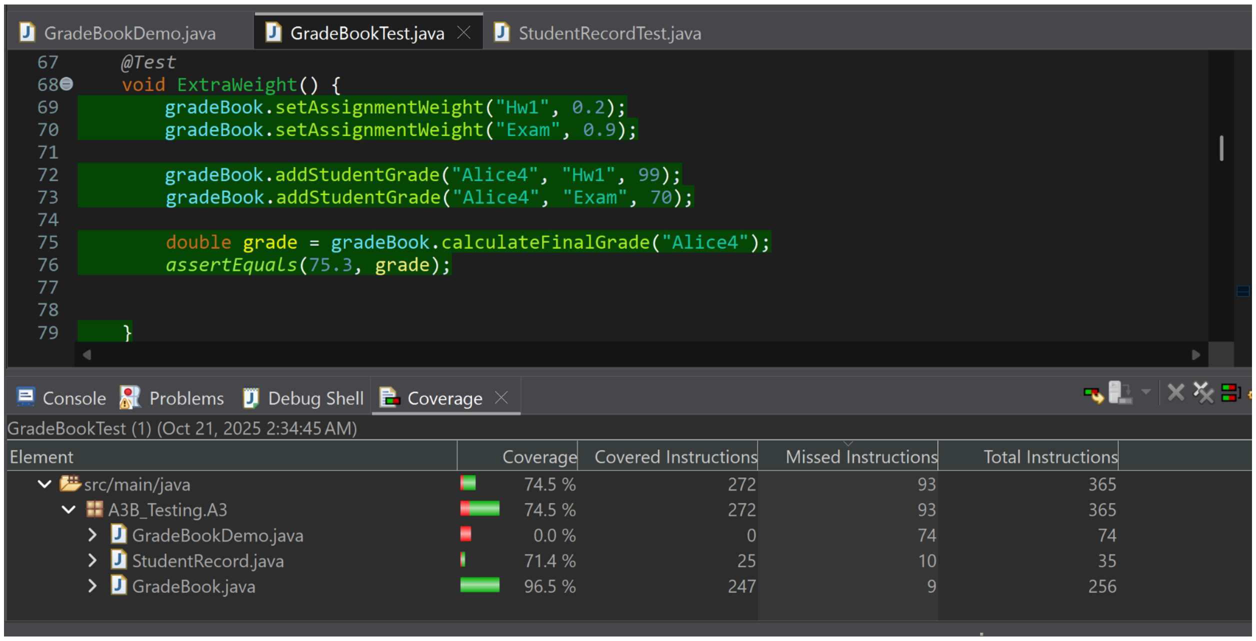
```
Finished after 0.269 seconds

Runs:  10/10        Errors:  0        Failures:  10

GradeBookTest [Runner: JUnit 5] (0.092 s)
    FinalGrade99() (0.062 s)
    AbnormalScores() (0.002 s)
    ExtraWeight() (0.002 s)
    MissingWeight() (0.002 s)
    StaleData() (0.003 s)
    LetterGradeA() (0.002 s)
    Rounding() (0.002 s)
    RemoveStudent() (0.004 s)
    AllGrades() (0.002 s)
    GradeCache() (0.002 s)

Failure Trace
org.opentest4j.AssertionFailedError: expected: <90.0> bu
at org.junit.jupiter.api.AssertionFailureBuilder.build(Asse
at A3B_Testing.A3.GradeBookTest.FinalGrade99(GradeBo
at java.base/java.util.ArrayList.forEach(ArrayList.java:1596
```



Finished after 0.154 seconds

Runs: 10/ Errors: Failures:

GradeBookTest [Runner: JUnit

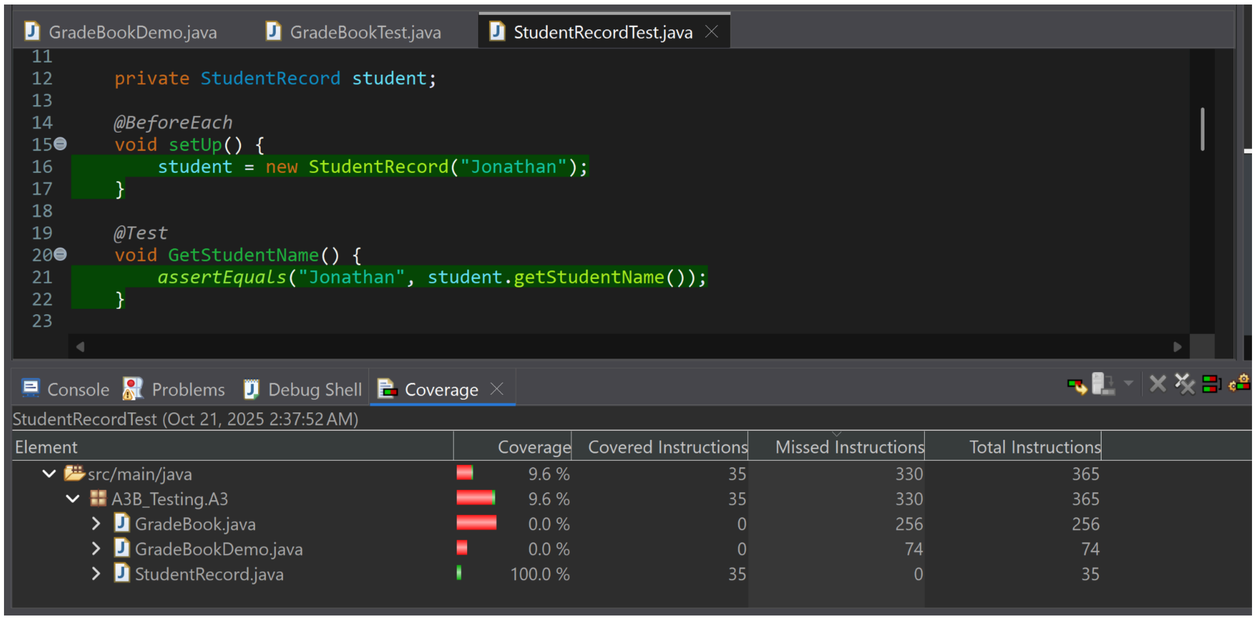

GradeBookDemo.java | GradeBookTest.java | StudentRecordTest.java

```java
    @Test
    void ExtraWeight() {
        gradeBook.setAssignmentWeight("Hw1", 0.2);
        gradeBook.setAssignmentWeight("Exam", 0.9);

        gradeBook.addStudentGrade("Alice4", "Hw1", 99);
        gradeBook.addStudentGrade("Alice4", "Exam", 70);

        double grade = gradeBook.calculateFinalGrade("Alice4");
        assertEquals(75.3, grade);


    }
```

Console | Problems | Debug Shell | Coverage

GradeBookTest (1) (Oct 21, 2025 2:34:45 AM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
src/main/java	74.5 %	272	93	365
A3B_Testing.A3	74.5 %	272	93	365
GradeBookDemo.java	0.0 %	0	74	74
StudentRecord.java	71.4 %	25	10	35
GradeBook.java	96.5 %	247	9	256


GradeBookDemo.java | GradeBookTest.java | StudentRecordTest.java

```java

    private StudentRecord student;

    @BeforeEach
    void setUp() {
        student = new StudentRecord("Jonathan");
    }

    @Test
    void GetStudentName() {
        assertEquals("Jonathan", student.getStudentName());
    }

```

Console | Problems | Debug Shell | Coverage

StudentRecordTest (Oct 21, 2025 2:37:52 AM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
src/main/java	9.6 %	35	330	365
A3B_Testing.A3	9.6 %	35	330	365
GradeBook.java	0.0 %	0	256	256
GradeBookDemo.java	0.0 %	0	74	74
StudentRecord.java	100.0 %	35	0	35