

```

GradeBook.java
1 package A3B_Testing.A3;
2
3 /**
4  * Our gradebook system that calculates final grades and letter grades for students
5  * CS 483 - Assignment 3A-3B (Bug Hunt)
6  * @author Chase Garnett & Fabrizio Guzzo
7  * @version 1.0 10/07/2025
8  */
9
10 import java.util.*;
11
12 public class GradeBook {
13
14     private Map<String, StudentRecord> allStudents;
15     private Map<String, Double> assignmentWeights;
16     private Map<String, Double> gradeCache;
17
18     public GradeBook() {
19         allStudents = new HashMap<>();
20         assignmentWeights = new HashMap<>();
21         gradeCache = new HashMap<>();
22     }
23
24     public void addStudentGrade(String studentName, String assignment, double score) {
25         // BUG #8: Allows negative scores
26         if (!allStudents.containsKey(studentName)) {
27             allStudents.put(studentName, new StudentRecord(studentName));
28         }
29         if (score < 0 || score > 100) score = 0; //Fix: Set bounds for Score to not be over 100
30         allStudents.get(studentName).addAssignmentScore(assignment, score);
31     }
32
33     public void setAssignmentWeight(String assignment, double weight) {
34         assignmentWeights.put(assignment, weight);
35     }
36
37     public double calculateFinalGrade(String studentName) {
38         // BUG #4: Null handling – may crash if student has no record
39         StudentRecord student = allStudents.get(studentName);
40         if (student == null) {
41             throw new IllegalArgumentException("Student not found"); // Fix: if student is not
42         }
43
44         // BUG #6: Cached value not cleared when student removed
45         if (gradeCache.containsKey(studentName)) {
46             return gradeCache.get(studentName);
47         }
48
49         double totalPoints = 0.0;
50         double weightTotal = 0.0;
51         int assignmentCount = 0;
52
53         for (Map.Entry<String, Double> weightEntry : assignmentWeights.entrySet()) {
54             String assignmentName = weightEntry.getKey();
55             double assignmentWeight = weightEntry.getValue();
56             weightTotal += assignmentWeight;
57
58             // BUG #1: Off-by-one – skip last assignment
59             //if (assignmentCount == assignmentWeights.size() - 1) break; //Fix: Stop it from
60             assignmentCount++;

```

```
61
62     Double studentScore = student.getAssignmentScore(assignmentName);
63     if (studentScore != null) {
64         totalPoints += studentScore * assignmentWeight;
65     } else {
66         totalPoints += 0 * assignmentWeight; // fix: studentScore should count as zero, not
67     }
68     // BUG #5: Missing assignment ignored (should count as zero)
69 }
70
71 // BUG #7: Fails to normalize weight sum (assumes it's 1)
72 double finalGrade;
73 if (weightTotal > 0) {
74     finalGrade = totalPoints / weightTotal; //Fix instead of directly assuming the
75 } else {
76     finalGrade = 0.0;
77 }
78
79 // BUG #2: Truncates instead of rounding
80 finalGrade = Math.round(finalGrade * 10) / 10.0; //Fix: round to the nearest tenth's
81
82
83 gradeCache.put(studentName, finalGrade);
84 return finalGrade;
85 }
86
87 public String getLetterGrade(String studentName) {
88     double numericGrade = calculateFinalGrade(studentName);
89     // BUG #3: Cutoff uses '>' not '>='
90     if (numericGrade >= 90) return "A";
91     if (numericGrade >= 80) return "B";
92     if (numericGrade >= 70) return "C";
93     if (numericGrade >= 60) return "D";
94     return "F";
95 }
96
97 public void removeStudentFromGradebook(String studentName) {
98     // BUG #9: Uses == for String comparison (fails on new String)
99     for (String currentStudent : allStudents.keySet()) {
100         if (currentStudent.equals(studentName)) { // Should be equals() Fix: instead of
101             allStudents.remove(currentStudent);
102             gradeCache.remove(currentStudent); //Fix: removes student and cache
103             break;
104         }
105     }
106 }
107
108 public Map<String, Double> generateAllFinalGrades() {
109     Map<String, Double> finalGrades = new HashMap<>();
110     for (String student : allStudents.keySet()) {
111         finalGrades.put(student, calculateFinalGrade(student));
112     }
113     return finalGrades;
114 }
115
116 public boolean isStudentEnrolled(String studentName) {
117     return allStudents.containsKey(studentName);
118 }
119
120
```



```
StudentRecord.java
1 package A3B_Testing.A3;
2
3 /**
4  * Represents an individual student's set of grades.
5  * CS 483 - Assignment 3A-3B (Bug Hunt)
6  * @author Chase Garnett & Fabrizio Guzzo
7  * @version 1.0 10/07/2025
8  */
9
10 import java.util.*;
11
12 public class StudentRecord {
13
14     private String studentName;
15     private Map<String, Double> assignmentScores;
16
17     public StudentRecord(String studentName) {
18         this.studentName = studentName;
19         this.assignmentScores = new HashMap<>();
20     }
21
22     public void addAssignmentScore(String assignmentName, double score) {
23         assignmentScores.put(assignmentName, score);
24     }
25
26     public Double getAssignmentScore(String assignmentName) {
27         return assignmentScores.get(assignmentName);
28     }
29
30     public Map<String, Double> getAllAssignmentScores() {
31         return assignmentScores;
32     }
33
34     public String getStudentName() {
35         return studentName;
36     }
37
38     // BUG #10: No reset or clear method – stale data across runs
39     public void clearAssignments() {
40         assignmentScores.clear(); //Fix Clears all scores and assignments
41     }
42 }
43
```

```

GradeBookTest.java
1 package A3B_Testing.A3;
2
3 import org.junit.jupiter.api.BeforeEach;
4 import org.junit.jupiter.api.Test;
5 import java.util.Map;
6
7 import static org.junit.jupiter.api.Assertions.*;
8
9 class GradeBookTest {
10
11     private GradeBook gradeBook;
12
13     @BeforeEach
14     void setUp() {
15         gradeBook = new GradeBook();
16     }
17
18
19     /**
20      * This Test uses the demo grades to make sure that the letter grade is an A
21      * @author JonathanDargakis
22      */
23     @Test
24     void LetterGradeA() {
25         gradeBook.setAssignmentWeight("HW1", 0.2);
26         gradeBook.setAssignmentWeight("HW2", 0.3);
27         gradeBook.setAssignmentWeight("Exam", 0.5);
28
29         gradeBook.addStudentGrade("Alice", "HW1", 99);
30         gradeBook.addStudentGrade("Alice", "HW2", 99);
31         gradeBook.addStudentGrade("Alice", "Exam", 99);
32
33         String letterGrade = gradeBook.getLetterGrade("Alice");
34
35         assertEquals("A", letterGrade);
36     }
37
38
39     /**
40      * This Test ensures that the final calculation of the grade is 99.
41      * Making sure that the logic is correct and not being cut off early (off by one)
42      * @author JonathanDargakis
43      */
44     @Test
45     void FinalGrade99() {
46         gradeBook.setAssignmentWeight("HW1", 0.2);
47         gradeBook.setAssignmentWeight("HW2", 0.3);
48         gradeBook.setAssignmentWeight("Exam", 0.5);
49
50         gradeBook.addStudentGrade("Alice1", "HW1", 99);
51         gradeBook.addStudentGrade("Alice1", "HW2", 99);
52         gradeBook.addStudentGrade("Alice1", "Exam", 99);
53
54         //average: 90*0.2 + 90*0.3 + 90*.5 = 99
55         //Missing Last Grade: 90*0.2 + 90*0.3 = 49
56         double grade = gradeBook.calculateFinalGrade("Alice1");
57
58
59         assertEquals(99, grade);
60

```

```
61
62
63 /**
64  * This Tests makes sure that the normalization of the weight works properly with a lower
65  * * Making sure that the weight is not assumed to be 1
66  * @author JonathanDargakis
67  */
68 @Test
69 void MissingWeight() {
70     gradeBook.setAssignmentWeight("HW1", 0.1);
71     gradeBook.setAssignmentWeight("HW2", 0.1);
72     gradeBook.setAssignmentWeight("HW3", 0.3);
73
74     gradeBook.addStudentGrade("Alice2", "HW1", 99);
75     gradeBook.addStudentGrade("Alice2", "HW2", 99);
76     gradeBook.addStudentGrade("Alice2", "HW3", 99);
77
78     double grade = gradeBook.calculateFinalGrade("Alice2");
79     assertEquals(99, grade);
80 }
81
82
83 /**
84  * This test tests the normalization of the with with an excess amount - over 1
85  * Making sure that the weight is not assumed to be 1
86  * @author JonathanDargakis
87  */
88 @Test
89 void ExtraWeight() {
90     gradeBook.setAssignmentWeight("Hw1", 0.2);
91     gradeBook.setAssignmentWeight("Exam", 0.9);
92
93     gradeBook.addStudentGrade("Alice4", "Hw1", 99);
94     gradeBook.addStudentGrade("Alice4", "Exam", 70);
95
96     double grade = gradeBook.calculateFinalGrade("Alice4");
97     assertEquals(75.3, grade);
98
99 }
100
101
102
103 /**
104  * This test makes sure that when calling an older student from another test a grade is not
105  * Ensuring there isnt any leftover stale data
106  * @author JonathanDargakis
107  */
108 @Test
109 void StaleDataThrowsException() {
110     //calling a method that should throw IllegalArgumentException
111     assertThrows(IllegalArgumentException.class, () -> {
112         gradeBook.getLetterGrade("Alice");
113     });
114 }
115
116
117 /**
118  * This test Ensures that the grade bounds are working properly
119  * As 90 should be an A
120  * @author JonathanDargakis
```

```
121 */
122 @Test
123 void Rounding() {
124     gradeBook.setAssignmentWeight("Hw1", 0.2);
125     gradeBook.setAssignmentWeight("Hw2", 0.3);
126     gradeBook.setAssignmentWeight("Exam", 0.5);
127     gradeBook.addStudentGrade("Alice5", "Hw1", 90);
128     gradeBook.addStudentGrade("Alice5", "Hw2", 90);
129     gradeBook.addStudentGrade("Alice5", "Exam", 90);
130
131     String letterGrade = gradeBook.getLetterGrade("Alice5");
132     assertEquals("A", letterGrade);
133 }
134
135
136 /**
137  * This test makes sure that after removing a student that the student is not enrolled and
138  * @author JonathanDargakis
139  */
140 @Test
141 void GradeCache() {
142     gradeBook.setAssignmentWeight("Hw1", 0.5);
143     gradeBook.setAssignmentWeight("Hw2", 0.5);
144     gradeBook.setAssignmentWeight("Exam", 0.5); //random 3rb because off by one
145     gradeBook.addStudentGrade("Alice6", "Hw1", 90);
146     gradeBook.addStudentGrade("Alice6", "Hw2", 90);
147     gradeBook.addStudentGrade("Alice6", "Exam", 90);
148
149
150     gradeBook.removeStudentFromGradebook("Alice6");
151
152     assertFalse(gradeBook.isStudentEnrolled("Alice6"));
153     assertThrows(IllegalArgumentException.class, () -> {
154         gradeBook.calculateFinalGrade("Alice6");
155     });
156
157 }
158
159 /**
160  * This test makes sure that it sets invalid scores to 0 and can handle doubles
161  * @author JonathanDargakis
162  */
163 @Test
164 void AbnormalScores() {
165     gradeBook.setAssignmentWeight("Hw1", 0.2);
166     gradeBook.setAssignmentWeight("Hw2", 0.3);
167     gradeBook.setAssignmentWeight("Exam", 0.5);
168     gradeBook.addStudentGrade("Alice7", "Hw1", -70);
169     gradeBook.addStudentGrade("Alice7", "Hw2", 101);
170     gradeBook.addStudentGrade("Alice7", "Exam", 75.5);
171
172     double grade = gradeBook.calculateFinalGrade("Alice7");
173     assertEquals(37.8, grade);
174
175 }
176
177 /**
178  * This test makes sure they you can remove a student by using another string object
179  * Ensuring that the comparison is between strings and not objects
180  * @author JonathanDargakis
```

```
181  */
182  @Test
183  void RemoveStudent() {
184      gradeBook.setAssignmentWeight("Hw1", 1);
185      gradeBook.addStudentGrade("Alice8", "Hw1", 75);
186
187      String studentToRemove = new String("Alice8"); // another string object to compare
188      gradeBook.removeStudentFromGradebook(studentToRemove);
189
190      assertFalse(gradeBook.isStudentEnrolled("Alice8"));
191  }
192
193
194
195  /**
196   * This tests makes sure that all of the grades bounds are working properly
197   * @author JonathanDargakis
198   */
199  @Test
200  void AllGrades() {
201      gradeBook.setAssignmentWeight("Hw1", 1);
202      gradeBook.addStudentGrade("Alice9", "Hw1", 90);
203
204      String letterGradeA = gradeBook.getLetterGrade("Alice9");
205      assertEquals("A", letterGradeA);
206
207
208      gradeBook.addStudentGrade("Bob", "Hw1", 80);
209
210      String letterGradeB = gradeBook.getLetterGrade("Bob");
211      assertEquals("B", letterGradeB);
212
213      gradeBook.addStudentGrade("Cat", "Hw1", 70);
214
215      String letterGradeC = gradeBook.getLetterGrade("Cat");
216      assertEquals("C", letterGradeC);
217
218      gradeBook.addStudentGrade("Darek", "Hw1", 60);
219
220      String letterGradeD = gradeBook.getLetterGrade("Darek");
221      assertEquals("D", letterGradeD);
222
223      gradeBook.addStudentGrade("Eric", "Hw1", 50);
224
225      String letterGradeF = gradeBook.getLetterGrade("Eric");
226      assertEquals("F", letterGradeF);
227
228  }
229
230
231  /**
232   * This test ensures that you are able to generate all of the grades at once
233   * Showing the number of all the grades and each student that is in the mapping.
234   * @author JonathanDargakis
235   */
236  @Test
237  void GenerateAllGrades() {
238      // Step 1: Set assignment weights
239      gradeBook.setAssignmentWeight("HW1", 0.5);
240      gradeBook.setAssignmentWeight("HW2", 0.5);
```



```
241
242     // Step 2: Add students and grades
243     gradeBook.addStudentGrade("Alice", "HW1", 80.0);
244     gradeBook.addStudentGrade("Alice", "HW2", 90.0);
245
246     gradeBook.addStudentGrade("Bob", "HW1", 70.0);
247     gradeBook.addStudentGrade("Bob", "HW2", 60.0);
248
249     // Step 3: Generate all final grades
250     Map<String, Double> allGrades = gradeBook.generateAllFinalGrades();
251
252     // Step 4: Verify map contains all students
253     assertEquals(2, allGrades.size());
254     assertTrue(allGrades.containsKey("Alice"));
255     assertTrue(allGrades.containsKey("Bob"));
256
257 }
258
259
260 /**
261  * This ensures that when a grade is null it is being treated as an int adding 0 instead of
262  * @author JonathanDargakis
263  */
264 @Test
265 void NullScoreAsZero() {
266
267     gradeBook.setAssignmentWeight("HW1", 0.5);
268     gradeBook.setAssignmentWeight("HW2", 0.5);
269
270     gradeBook.addStudentGrade("Alice10", "HW1", 80.0);
271     //since HW2 is missing score = null = 0
272
273     double finalGrade = gradeBook.calculateFinalGrade("Alice10");
274
275     // = 80*0.5 + 0*0.5 = 40.0
276     assertEquals(40.0, finalGrade);
277 }
278
279 }
280
```

```

StudentRecordTest.java
1 package A3B_Testing.A3;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 import org.junit.jupiter.api.BeforeEach;
6 import org.junit.jupiter.api.Test;
7
8 import java.util.Map;
9
10 class StudentRecordTest {
11
12     private StudentRecord student;
13
14     @BeforeEach
15     void setUp() {
16         student = new StudentRecord("Jonathan");
17     }
18
19
20     /**
21      * This test makes sure that the student name is Jonathan
22      * @author JonathanDargakis
23      */
24     @Test
25     void GetStudentName() {
26         assertEquals("Jonathan", student.getStudentName());
27     }
28
29
30     /**
31      * This test ensures that you are able to add assignments and later call them
32      * @author JonathanDargakis
33      */
34     @Test
35     void AddAndGetAssignmentScore() {
36         student.addAssignmentScore("HW1", 80);
37         student.addAssignmentScore("Exam", 76);
38
39         assertEquals(80, student.getAssignmentScore("HW1"));
40         assertEquals(76, student.getAssignmentScore("Exam"));
41
42         // Test getting score for assignment not added
43         assertNull(student.getAssignmentScore("HW2")); // Score should be null for missing
44     }
45
46
47     /**
48      * This test ensures that a missing assignment score is Null
49      * later turned into 0
50      * @author JonathanDargakis
51      */
52     @Test
53     void MissingAssignmentScore() {
54         student.addAssignmentScore("HW1", 80);
55         student.addAssignmentScore("Exam", 76);
56
57         assertNull(student.getAssignmentScore("HW2"));
58     }
59
60

```

```
61  /**
62   * This test makes sure that you are able to get all of the assignment scores at once
63   * verifying the size and contents
64   * @author JonathanDargakis
65   */
66  @Test
67  void GetAllAssignmentScores() {
68      student.addAssignmentScore("HW1", 56);
69      student.addAssignmentScore("HW2", 79);
70
71      Map<String, Double> allScores = student.getAllAssignmentScores();
72      assertEquals(2, allScores.size());
73      assertEquals(56, allScores.get("HW1"));
74      assertEquals(79, allScores.get("HW2"));
75  }
76
77
78  /**
79   * This test makes sure that all of the Assignments are removed and cannot be called later
80   * @author JonathanDargakis
81   */
82  @Test
83  void ClearAssignments() {
84      student.addAssignmentScore("HW1", 89);
85      student.addAssignmentScore("Exam", 82);
86
87      assertEquals(2, student.getAllAssignmentScores().size());
88
89      student.clearAssignments();
90
91      assertEquals(0, student.getAllAssignmentScores().size());
92      assertNull(student.getAssignmentScore("HW1"));
93      assertNull(student.getAssignmentScore("Exam"));
94  }
95
96
97
98
```

Eclipse_ - A3/src/test/java/A3B_Testing/A3/GradeBookTest.java -

File Edit Source Refactor Navigate Search Project Run



Debug Project Explorer JUnit



Finished after 0.269 seconds

Runs: 10/10 Errors: 0 Failures: 10

GradeBookTest [Runner: JUnit 5] (0.092 s)

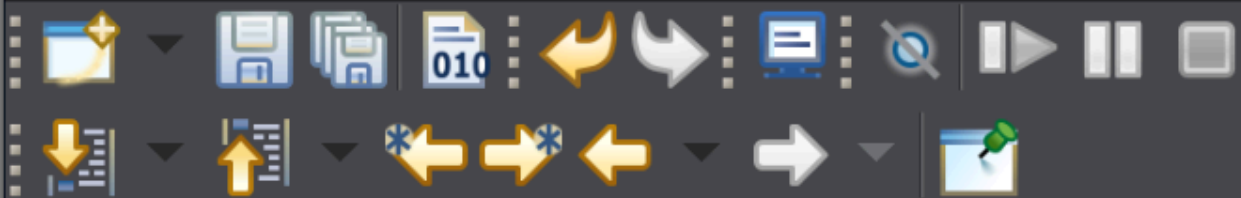
- FinalGrade99() (0.062 s)
- AbnormalScores() (0.002 s)
- ExtraWeight() (0.002 s)
- MissingWeight() (0.002 s)
- StaleData() (0.003 s)
- LetterGradeA() (0.002 s)
- Rounding() (0.002 s)
- RemoveStudent() (0.004 s)
- AllGrades() (0.002 s)
- GradeCache() (0.002 s)

Failure Trace

org.opentest4j.AssertionFailedError: expected: <90.0> but
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

Eclipse_ - A3/src/test/java/A3B_Testing/A

File Edit Source Refactor Navigate

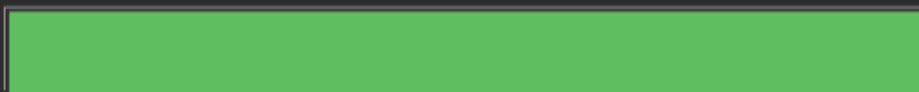


D P J ×



Finished after 0.154 seconds

Runs: 10/ Errors: Failures:



> GradeBookTest [Runner: JUnit

45

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

GradeBookDemo.java GradeBookTest.java StudentRecordTest.java

```
67 @Test
68 void ExtraWeight() {
69     gradeBook.setAssignmentWeight("Hw1", 0.2);
70     gradeBook.setAssignmentWeight("Exam", 0.9);
71
72     gradeBook.addStudentGrade("Alice4", "Hw1", 99);
73     gradeBook.addStudentGrade("Alice4", "Exam", 70);
74
75     double grade = gradeBook.calculateFinalGrade("Alice4");
76     assertEquals(75.3, grade);
77
78
79 }
```

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	<div><div></div></div> 74.5 %	272	93	365
A3B_Testing.A3	<div><div></div></div> 74.5 %	272	93	365
GradeBookDemo.java	<div><div></div></div> 0.0 %	0	74	74
StudentRecord.java	<div><div></div></div> 71.4 %	25	10	35
GradeBook.java	<div><div></div></div> 96.5 %	247	9	256

GradeBookDemo.java GradeBookTest.java StudentRecordTest.java

```
11
12 private StudentRecord student;
13
14 @BeforeEach
15 void setUp() {
16     student = new StudentRecord("Jonathan");
17 }
18
19 @Test
20 void GetStudentName() {
21     assertEquals("Jonathan", student.getStudentName());
22 }
23
```

Console Problems Debug Shell Coverage

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

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