

## Main Code

```
3
4  /**
5   * StudentGradeProcessor class that processes student grades with multiple branches and loops
6   */
7  public class StudentGradeProcessor { 3 usages
8
9      /**
10       * Inner class to hold grade processing results
11       */
12      public static class GradeResult { 6 usages
13          private final int totalStudents; 5 usages
14          private final double average; 5 usages
15          private final int passCount; 5 usages
16          private final int aCount; 5 usages
17          private final int bCount; 5 usages
18          private final int cCount; 5 usages
19          private final int dCount; 5 usages
20          private final int fCount; 5 usages
21
22          @Contract(pure = true)
23          public GradeResult(int totalStudents, double average, int passCount, 1 usage
24                          int aCount, int bCount, int cCount, int dCount) {
25              this(totalStudents, average, passCount, aCount, bCount, cCount, dCount, fCount: 0);
26          }
27
28          @Contract(pure = true)
29          public GradeResult(int totalStudents, double average, int passCount, 2 usages
30                          int aCount, int bCount, int cCount, int dCount, int fCount) {
31              this.totalStudents = totalStudents;
32              this.average = average;
33              this.passCount = passCount;
34              this.aCount = aCount;
35              this.bCount = bCount;
36              this.cCount = cCount;
37              this.dCount = dCount;
38              this.fCount = fCount;
39          }
40
41          // Getters
42          public int getTotalStudents() { return totalStudents; } no usages
43          public double getAverage() { return average; } no usages
44          public int getPassCount() { return passCount; } 1 usage
45          public int getACount() { return aCount; } no usages
46          public int getBCount() { return bCount; } no usages
47          public int getCCount() { return cCount; } no usages
48          public int getDCount() { return dCount; } no usages
49          public int getFCount() { return fCount; } no usages
50      }
51  }
```

lyzer > src > StudentGradeProcessor > GradeResult > aCount 16:34 CRLF UTF-8 4 spaces

TextAnalyzer Version control Current File

Main.java StudentGradeProcessor.java .gitignore src\StudentGradeProcessorTest.java test\StudentGradeProcessorTest.java

```
7  public class StudentGradeProcessor { 3 usages
12  public static class GradeResult { 6 usages
26
27      @Contract(pure = true)
28      public GradeResult(int totalStudents, double average, int passCount, 2 usages
29                          int aCount, int bCount, int cCount, int dCount, int fCount) {
30          this.totalStudents = totalStudents;
31          this.average = average;
32          this.passCount = passCount;
33          this.aCount = aCount;
34          this.bCount = bCount;
35          this.cCount = cCount;
36          this.dCount = dCount;
37          this.fCount = fCount;
38      }
39
40      // Getters
41      public int getTotalStudents() { return totalStudents; } no usages
42      public double getAverage() { return average; } no usages
43      public int getPassCount() { return passCount; } 1 usage
44      public int getACount() { return aCount; } no usages
45      public int getBCount() { return bCount; } no usages
46      public int getCCount() { return cCount; } no usages
47      public int getDCount() { return dCount; } no usages
48      public int getFCount() { return fCount; } no usages
49  }
```

TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount 45:50 CRLF UTF-8 4 spaces

Windows taskbar: 27°C, 06:48, 2025/09/24

```
TextAnalyzer Version control Current File
Main.java StudentGradeProcessor.java .gitignore src\StudentGradeProcessorTest.java test\StudentGradeProcessorTest.java

7 public class StudentGradeProcessor { 3 usages
8     public GradeResult processGrades(List<Integer> scores, int bonusPoints, boolean strictMode) { 1 usage
9
10         totalScore += adjustedScore;
11
12         // Branch 4: Determine pass/fail threshold based on mode
13         int passThreshold = strictMode ? 70 : 60;
14         if (adjustedScore >= passThreshold) {
15             passCount++;
16         }
17
18         String letterGrade;
19         if (adjustedScore >= 90) {
20             letterGrade = "A";
21             aCount++;
22         } else if (adjustedScore >= 80) { // Branch 5
23             letterGrade = "B";
24             bCount++;
25         } else if (adjustedScore >= 70) { // Branch 6
26             letterGrade = "C";
27             cCount++;
28         } else if (adjustedScore >= 60) { // Branch 7
29             letterGrade = "D";
30             dCount++;
31         } else { // Branch 8
32             letterGrade = "F";
33             fCount++;
34         }
35     }
36 }
```

TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount 45:50 CRLF UTF-8 4 spaces 06:49 2025/09/24

```
TextAnalyzer Version control Current File
Main.java StudentGradeProcessor.java .gitignore src\StudentGradeProcessorTest.java test\StudentGradeProcessorTest.java

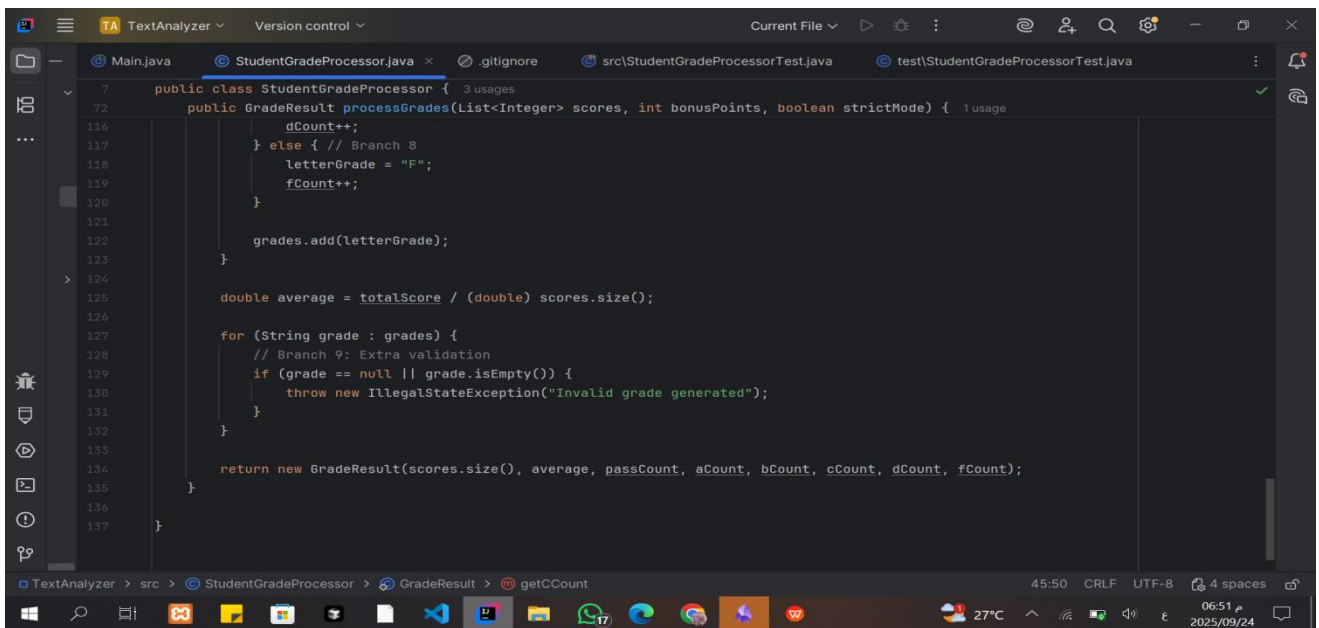
7 public class StudentGradeProcessor { 3 usages
8     public static class GradeResult { 6 usages
9
10         @Contract(value = "null -> false", pure = true)
11         @Override
12         public boolean equals(Object obj) {
13             if (this == obj) return true;
14             if (obj == null || getClass() != obj.getClass()) return false;
15             GradeResult that = (GradeResult) obj;
16             return totalStudents == that.totalStudents &&
17                 Double.compare(that.average, average) == 0 &&
18                 passCount == that.passCount &&
19                 aCount == that.aCount &&
20                 bCount == that.bCount &&
21                 cCount == that.cCount &&
22                 dCount == that.dCount &&
23                 fCount == that.fCount;
24         }
25
26         @Override
27         public String toString() {
28             return String.format("GradeResult{students=%d, avg=%.2f, pass=%d, A=%d, B=%d, C=%d, D=%d, F=%d}",
29                 totalStudents, average, passCount, aCount, bCount, cCount, dCount, fCount);
30         }
31     }
32 }
```

TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount 45:50 CRLF UTF-8 4 spaces 06:48 2025/09/24

```
TextAnalyzer Version control Current File
Main.java StudentGradeProcessor.java .gitignore src\StudentGradeProcessorTest.java test\StudentGradeProcessorTest.java

7 public class StudentGradeProcessor { 3 usages
8     public GradeResult processGrades(List<Integer> scores, int bonusPoints, boolean strictMode) { 1 usage
9
10         totalScore += adjustedScore;
11
12         // Branch 4: Determine pass/fail threshold based on mode
13         int passThreshold = strictMode ? 70 : 60;
14         if (adjustedScore >= passThreshold) {
15             passCount++;
16         }
17
18         String letterGrade;
19         if (adjustedScore >= 90) {
20             letterGrade = "A";
21             aCount++;
22         } else if (adjustedScore >= 80) { // Branch 5
23             letterGrade = "B";
24             bCount++;
25         } else if (adjustedScore >= 70) { // Branch 6
26             letterGrade = "C";
27             cCount++;
28         } else if (adjustedScore >= 60) { // Branch 7
29             letterGrade = "D";
30             dCount++;
31         } else { // Branch 8
32             letterGrade = "F";
33             fCount++;
34         }
35     }
36 }
```

TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount 45:50 CRLF UTF-8 4 spaces 06:51 2025/09/24



```
7 public class StudentGradeProcessor { 3 usages
72 public GradeResult processGrades(List<Integer> scores, int bonusPoints, boolean strictMode) { 1 usage
116     dCount++;
117 } else { // Branch 8
118     letterGrade = "F";
119     fCount++;
120 }
121
122     grades.add(letterGrade);
123 }
124
125     double average = totalScore / (double) scores.size();
126
127     for (String grade : grades) {
128         // Branch 9: Extra validation
129         if (grade == null || grade.isEmpty()) {
130             throw new IllegalStateException("Invalid grade generated");
131         }
132     }
133
134     return new GradeResult(scores.size(), average, passCount, aCount, bCount, cCount, dCount, fCount);
135 }
136
137 }
```

## Test class using JUnit framework

I created a unit test for the processGrades method .

Test Case 1 :

I provided a list of student scores and added bonus points to them. Then, I verified the results by checking the total number of students, the number of passing students, the average score, and the grade distribution (A, B, C, D, F). I used `assertEquals` to compare the expected values with the actual results returned by the method.

For the average score, I added a small delta (0.01) to handle floating-point precision.

```
@Test
void testGradeProcess() {
    List<Integer> scores = Arrays.asList(85, 57, 90);
    int bonusPoints = 2;
    boolean strictMode = false;

    StudentGradeProcessor p = new StudentGradeProcessor();
    StudentGradeProcessor.GradeResult result = p.processGrades(scores, bonusPoints, strictMode);

    assertEquals( expected: 3, result.getTotalStudents());
    assertEquals( expected: 2, result.getPassCount());

    double averageRounded = Math.round(result.getAverage() * 10) / 10.0;
    assertEquals( expected: 79.3, averageRounded, delta: 0.01);

    assertEquals( expected: 1, result.getACount());
    assertEquals( expected: 1, result.getBCount());
    assertEquals( expected: 0, result.getCCount());
    assertEquals( expected: 0, result.getDCount());
    assertEquals( expected: 1, result.getFCount());
}
```

### Test Case 2:

In this test case I check if processGrades method ignore null values in the scores list and ensure only valid inputs are counted when calculating total students, average score, pass count, and grade

```
8
9      @Test
10     public void testNullValues(){
11         List<Integer> scores = Arrays.asList(100, null, 95, null);
12         int bonusPoints = 2;
13         boolean strictMode = false;
14         StudentGradeProcessor p = new StudentGradeProcessor();
15         StudentGradeProcessor.GradeResult result = p.processGrades(scores, bonusPoints, strictMode);
16
17         assertEquals( expected: 4, result.getTotalStudents());
18         assertEquals( expected: 2, result.getPassCount());
19         assertEquals( expected: 49.25, result.getAverage(), delta: 0.01);
20         assertEquals( expected: 2, result.getACount());
21         assertEquals( expected: 0, result.getBCount());
22         assertEquals( expected: 0, result.getCCount());
23         assertEquals( expected: 0, result.getDCount());
24         assertEquals( expected: 0, result.getFCount());
25     }
```

### Test Case 3:

In this case I check if bounus all scores not cappted on 100

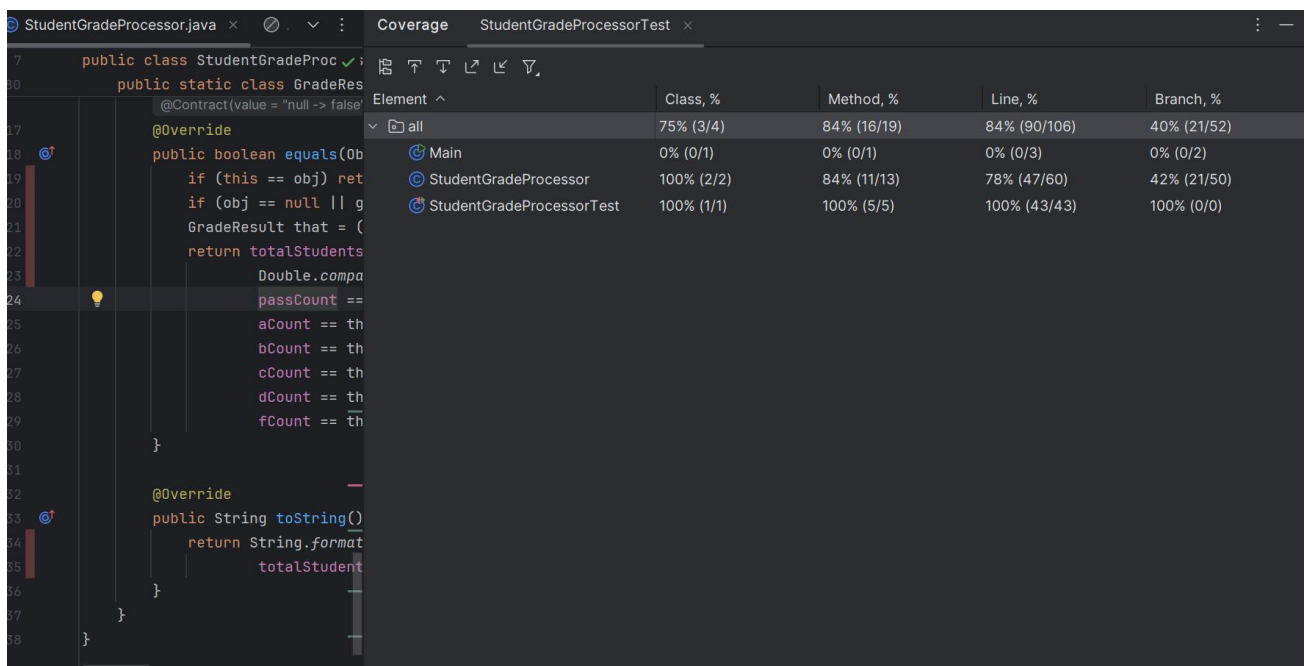
```
@Test
void testBonusCappingAt100() {
    List<Integer> scores = Arrays.asList(98, 100, 95);
    int bonusPoints = 5;
    boolean strictMode = false;
    StudentGradeProcessor p = new StudentGradeProcessor();
    StudentGradeProcessor.GradeResult result = p.processGrades(scores, bonusPoints, strictMode);
    // Assertions to check that bonus does not exceed 100
    assertEquals( expected: 3, result.getTotalStudents());
    assertEquals( expected: 3, result.getPassCount());
    assertEquals( expected: 100.0, result.getAverage(), delta: 0.01); // all scores capped at 100
    assertEquals( expected: 3, result.getACount());
    assertEquals( expected: 0, result.getBCount());
    assertEquals( expected: 0, result.getCCount());
    assertEquals( expected: 0, result.getDCount());
    assertEquals( expected: 0, result.getFCount());
}
```

#### Test Case 4 :

The test with -5 for totalStudents passed because the current constructor does not include **validation**. It simply stores the values as they are. That means negative or out-of-range inputs are accepted without errors.

```
@Test
void testNegativeStudentNumber(){
    StudentGradeProcessor.GradeResult SG=new StudentGradeProcessor.GradeResult( totalStudents: -5, average: 70.0, passCount: 3, aCount: 1,
    assertEquals( expected: -5,SG.getTotalStudents());
}
```

Applying **test coverage** made me realize that some test cases were not covered



Element	Class, %	Method, %	Line, %	Branch, %
all	75% (3/4)	84% (16/19)	84% (90/106)	40% (21/52)
Main	0% (0/1)	0% (0/1)	0% (0/3)	0% (0/2)
StudentGradeProcessor	100% (2/2)	84% (11/13)	78% (47/60)	42% (21/50)
StudentGradeProcessorTest	100% (1/1)	100% (5/5)	100% (43/43)	100% (0/0)

After add more test cases :

1: Make sure the equals method works correctly in all scenarios.

```
@Test
void Testequals(){
    StudentGradeProcessor.GradeResult result1=new StudentGradeProcessor.GradeResult( totalStudents: 20, average: 70.0, passCount: 5, aCount: 5
    StudentGradeProcessor.GradeResult result2=new StudentGradeProcessor.GradeResult( totalStudents: 20, average: 70.0, passCount: 5, aCount: 5
    StudentGradeProcessor.GradeResult result3=new StudentGradeProcessor.GradeResult( totalStudents: 20, average: 90, passCount: 7, aCount: 3,

    assertTrue(result1.equals(result2));
    assertFalse(result2.equals(result3));
    // assertEquals(result2,result1);
}
```



2 :

Test case 5: test to string method

```
@Test
void testToString(){
    StudentGradeProcessor.GradeResult result =
        new StudentGradeProcessor.GradeResult( totalStudents: 5, average: 75.50, passCount: 3, aCount: 1, bCount: 1, cCount: 1, dCount: 1, fCount: 1);
    String expected = "GradeResult{students=5, avg=75.50, pass=3, A=1, B=1, C=1, D=1, F=3}";
    assertEquals(expected, result.toString());
}
```

Q equals

public class StudentGradeProcessor { 19 usages  
public static class GradeResult { 16 usages  
@Contract(value = "null -> false", pure = true)  
@Override  
public boolean equals(Object obj) {  
if (this == obj) return true;  
if (obj == null || getClass() != obj.getClass()) return false;  
GradeResult that = (GradeResult) obj;  
return totalStudents == that.totalStudents &&  
Double.compare(that.average, average) != 0 &&

Element	Class, %	Method, %	Line, %	Branch, %
all	75% (3/4)	85% (18/21)	87% (100/11...)	61% (33/54)
Main	0% (0/1)	0% (0/1)	0% (0/3)	0% (0/2)
StudentGradePrc	100% (2/2)	85% (12/14)	82% (52/63)	63% (33/52)
StudentGradePrc	100% (1/1)	100% (6/6)	100% (48/48)	100% (0/0)

Use PIT for mutation test

Before :

## Pit Test Coverage Report

### Package Summary

default

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	95% 37/39	72% 21/29	75% 21/28

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">StudentGradeProcessor.java</a>	95% 37/39	72% 21/29	75% 21/28

Report generated by [PIT](#) 1.20.0

## Mutations

<a href="#">11</a>	1. negated conditional → KILLED
	2. negated conditional → KILLED
<a href="#">12</a>	1. replaced return value with null for StudentGradeProcessor::processGrades → NO_COVERAGE
<a href="#">23</a>	1. negated conditional → KILLED
<a href="#">27</a>	1. Replaced integer addition with subtraction → KILLED
<a href="#">30</a>	1. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
	2. negated conditional → KILLED
<a href="#">34</a>	1. Replaced integer addition with subtraction → KILLED
<a href="#">37</a>	1. negated conditional → SURVIVED <a href="#">Covering tests</a>
<a href="#">38</a>	1. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
	2. negated conditional → KILLED
<a href="#">39</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">44</a>	1. negated conditional → KILLED
	2. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
<a href="#">46</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">47</a>	1. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
	2. negated conditional → KILLED
<a href="#">49</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">50</a>	1. negated conditional → KILLED
	2. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
<a href="#">52</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">53</a>	1. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
	2. negated conditional → KILLED
<a href="#">55</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">58</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">64</a>	1. Replaced double division with multiplication → KILLED
<a href="#">69</a>	1. negated conditional → KILLED
	2. negated conditional → KILLED
<a href="#">74</a>	1. replaced return value with null for StudentGradeProcessor::processGrades → KILLED

After :

# Pit Test Coverage Report

## Package Summary

default

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	95% <div><div>37/39</div></div>	93% <div><div>27/29</div></div>	96% <div><div>27/28</div></div>

## Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">StudentGradeProcessor.java</a>	95% <div><div>37/39</div></div>	93% <div><div>27/29</div></div>	96% <div><div>27/28</div></div>

Report generated by [PIT](#) 1.20.0

## Mutations

<a href="#">11</a>	1. negated conditional → KILLED
	2. negated conditional → KILLED
<a href="#">12</a>	1. replaced return value with null for StudentGradeProcessor::processGrades → NO_COVERAGE
<a href="#">23</a>	1. negated conditional → KILLED
<a href="#">27</a>	1. Replaced integer addition with subtraction → KILLED
<a href="#">30</a>	1. changed conditional boundary → SURVIVED <a href="#">Covering tests</a>
	2. negated conditional → KILLED
<a href="#">34</a>	1. Replaced integer addition with subtraction → KILLED
<a href="#">37</a>	1. negated conditional → KILLED
<a href="#">38</a>	1. changed conditional boundary → KILLED
	2. negated conditional → KILLED
<a href="#">39</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">44</a>	1. negated conditional → KILLED
	2. changed conditional boundary → KILLED
<a href="#">46</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">47</a>	1. changed conditional boundary → KILLED
	2. negated conditional → KILLED
<a href="#">49</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">50</a>	1. negated conditional → KILLED
	2. changed conditional boundary → KILLED
<a href="#">52</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">53</a>	1. changed conditional boundary → KILLED
	2. negated conditional → KILLED
<a href="#">55</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">58</a>	1. Changed increment from 1 to -1 → KILLED
<a href="#">64</a>	1. Replaced double division with multiplication → KILLED
<a href="#">69</a>	1. negated conditional → KILLED
	2. negated conditional → KILLED
<a href="#">74</a>	1. replaced return value with null for StudentGradeProcessor::processGrades → KILLED