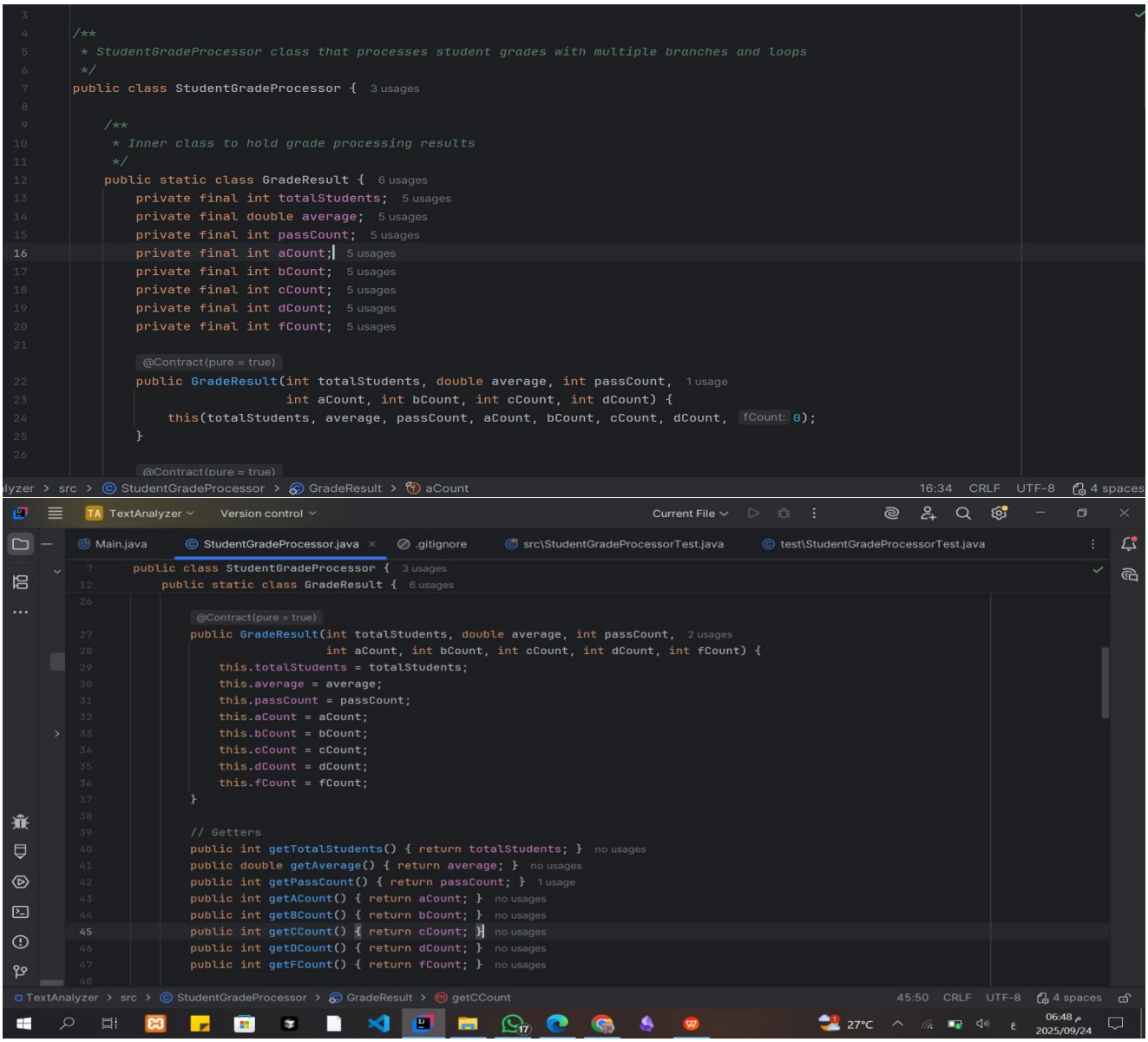
Main Code



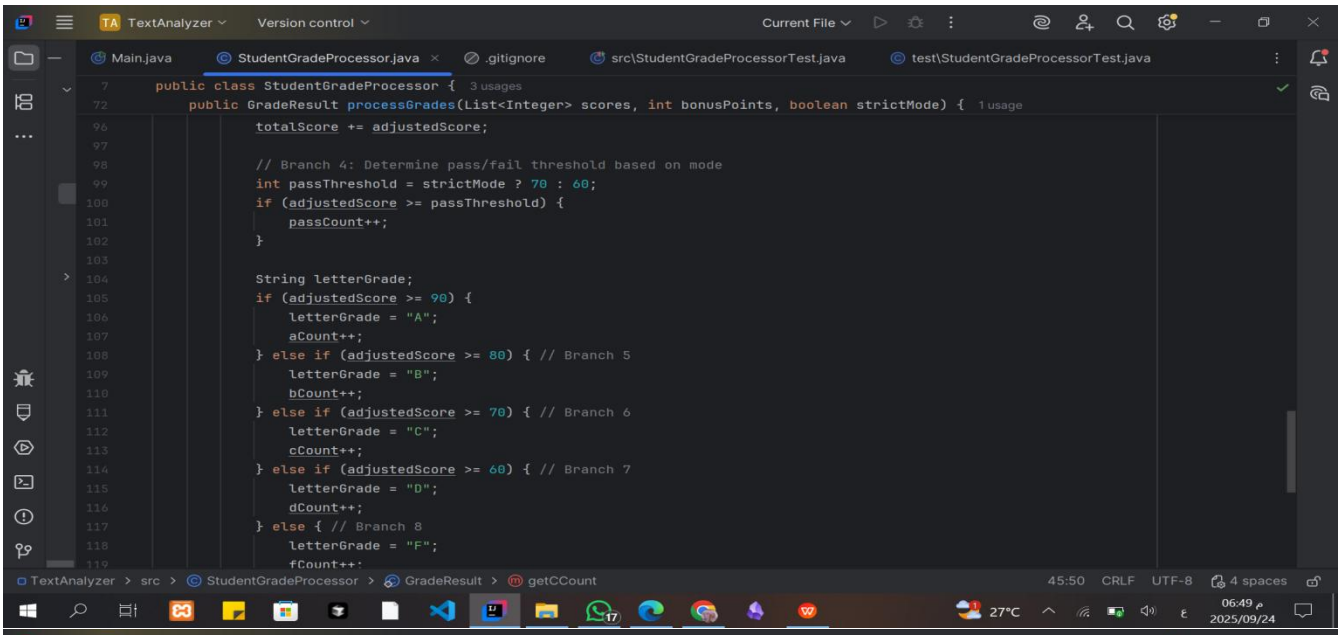
```java
/**
 * StudentGradeProcessor class that processes student grades with multiple branches and loops
 */
public class StudentGradeProcessor {  3 usages

    /**
     * Inner class to hold grade processing results
     */
    public static class GradeResult {  6 usages
        private final int totalStudents;  5 usages
        private final double average;  5 usages
        private final int passCount;  5 usages
        private final int aCount;  5 usages
        private final int bCount;  5 usages
        private final int cCount;  5 usages
        private final int dCount;  5 usages
        private final int fCount;  5 usages

        @Contract(pure = true)
        public GradeResult(int totalStudents, double average, int passCount,  1 usage
                           int aCount, int bCount, int cCount, int dCount) {
            this(totalStudents, average, passCount, aCount, bCount, cCount, dCount,  fCount: 0);
        }

        @Contract(pure = true)
```

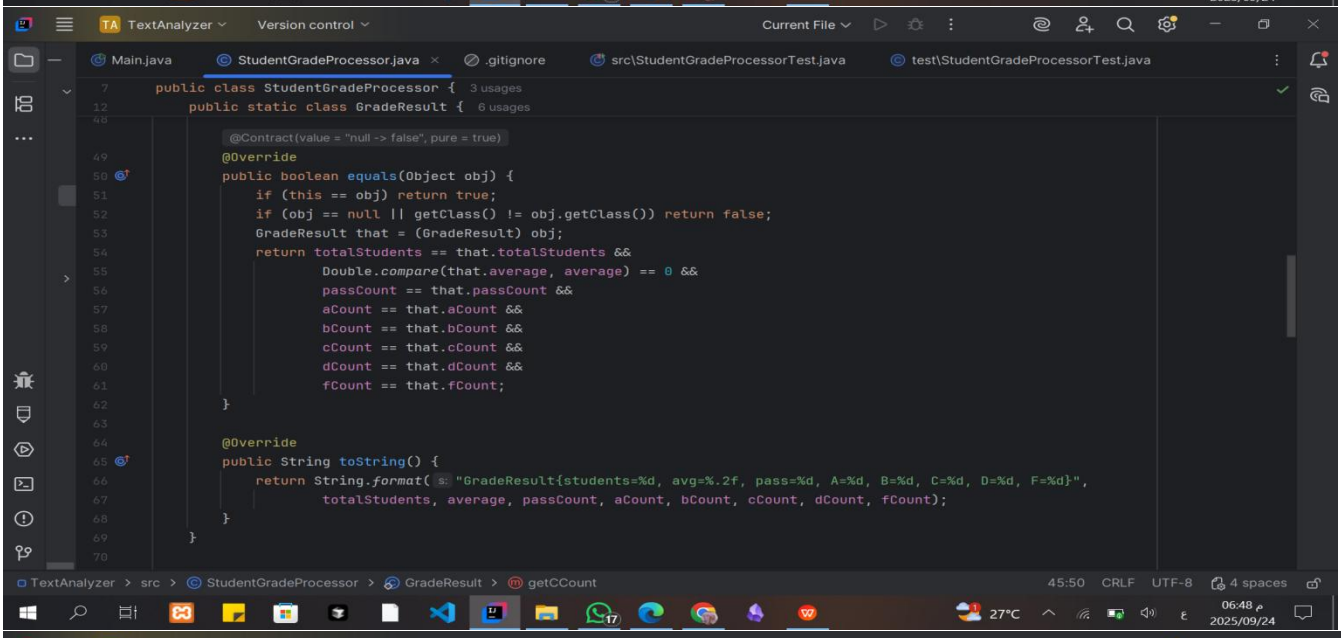
lyzer > src > StudentGradeProcessor > GradeResult > aCount 16:34 CRLF UTF-8 4 spaces

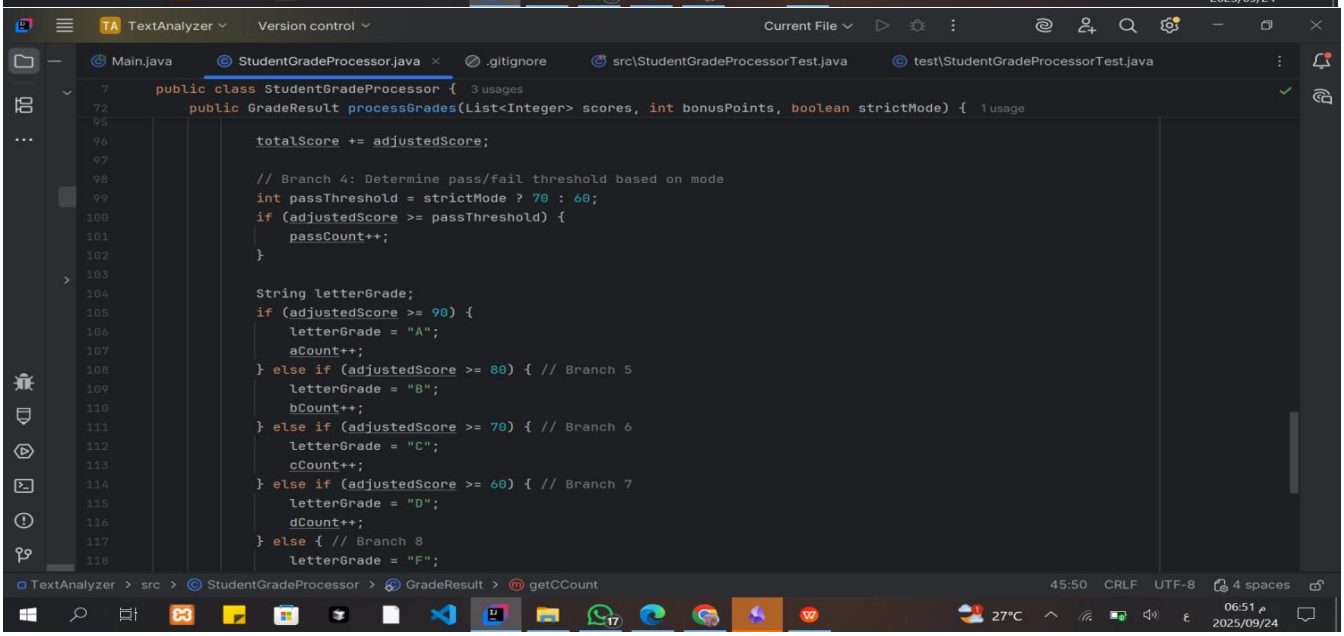

```java
public class StudentGradeProcessor {  3 usages
    public static class GradeResult {  6 usages

        @Contract(pure = true)
        public GradeResult(int totalStudents, double average, int passCount,  2 usages
                           int aCount, int bCount, int cCount, int dCount, int fCount) {
            this.totalStudents = totalStudents;
            this.average = average;
            this.passCount = passCount;
            this.aCount = aCount;
            this.bCount = bCount;
            this.cCount = cCount;
            this.dCount = dCount;
            this.fCount = fCount;
        }

        // Getters
        public int getTotalStudents() { return totalStudents; }  no usages
        public double getAverage() { return average; }  no usages
        public int getPassCount() { return passCount; }  1 usage
        public int getACount() { return aCount; }  no usages
        public int getBCount() { return bCount; }  no usages
        public int getCCount() { return cCount; }  no usages
        public int getDCount() { return dCount; }  no usages
        public int getFCount() { return fCount; }  no usages
```

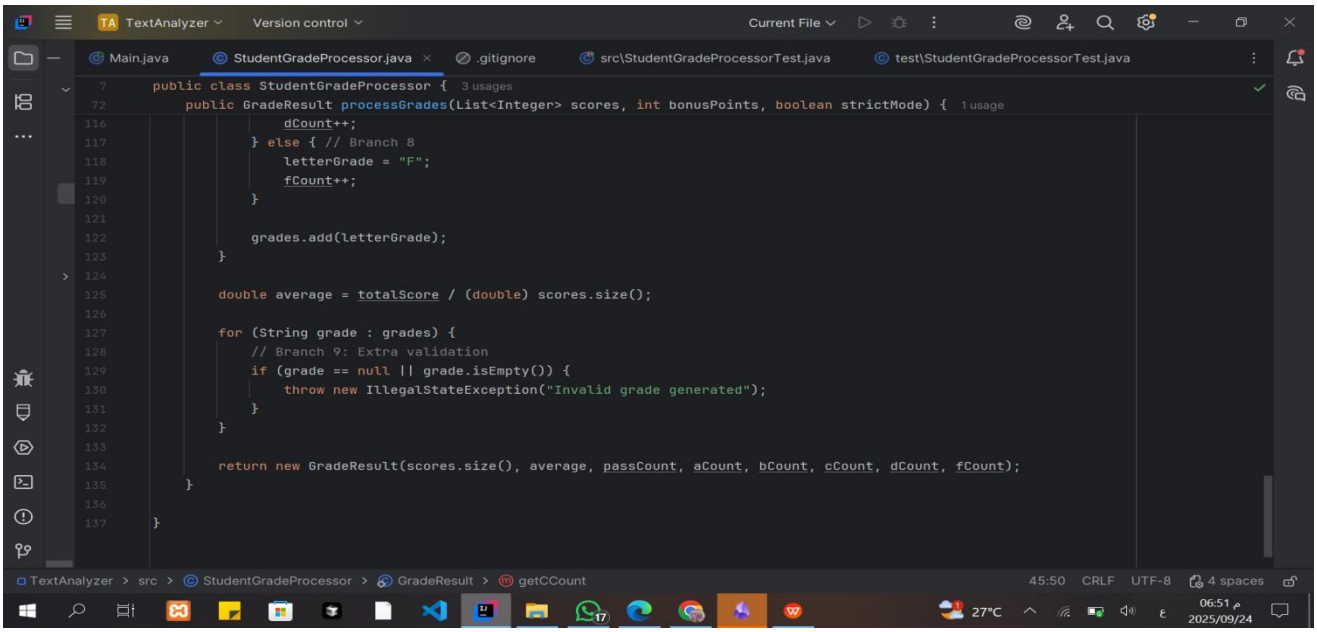
TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount 45:50 CRLF UTF-8 4 spaces

27°C 06:48 م 2025/09/24


TextAnalyzer
Version control
Current File
Main.java
StudentGradeProcessor.java
.gitignore
src\StudentGradeProcessorTest.java
test\StudentGradeProcessorTest.java
public class StudentGradeProcessor { 3 usages
public GradeResult processGrades(List<Integer> scores, int bonusPoints, boolean strictMode) { 1 usage
totalScore += adjustedScore;
// Branch 4: Determine pass/fail threshold based on mode
int passThreshold = strictMode ? 70 : 60;
if (adjustedScore >= passThreshold) {
passCount++;
}
String letterGrade;
if (adjustedScore >= 90) {
letterGrade = "A";
aCount++;
} else if (adjustedScore >= 80) { // Branch 5
letterGrade = "B";
bCount++;
} else if (adjustedScore >= 70) { // Branch 6
letterGrade = "C";
cCount++;
} else if (adjustedScore >= 60) { // Branch 7
letterGrade = "D";
dCount++;
} else { // Branch 8
letterGrade = "F";
fCount++;
TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount
45:50 CRLF UTF-8 4 spaces
27°C
06:49
2025/09/24


TextAnalyzer
Version control
Current File
Main.java
StudentGradeProcessor.java
.gitignore
src\StudentGradeProcessorTest.java
test\StudentGradeProcessorTest.java
public class StudentGradeProcessor { 3 usages
public static class GradeResult { 6 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 &&
passCount == that.passCount &&
aCount == that.aCount &&
bCount == that.bCount &&
cCount == that.cCount &&
dCount == that.dCount &&
fCount == that.fCount;
}
@Override
public String toString() {
return String.format(s: "GradeResult{students=%d, avg=%.2f, pass=%d, A=%d, B=%d, C=%d, D=%d, F=%d}",
totalStudents, average, passCount, aCount, bCount, cCount, dCount, fCount);
}
}
TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount
45:50 CRLF UTF-8 4 spaces
27°C
06:48
2025/09/24


TextAnalyzer
Version control
Current File
Main.java
StudentGradeProcessor.java
.gitignore
src\StudentGradeProcessorTest.java
test\StudentGradeProcessorTest.java
public class StudentGradeProcessor { 3 usages
public GradeResult processGrades(List<Integer> scores, int bonusPoints, boolean strictMode) { 1 usage
totalScore += adjustedScore;
// Branch 4: Determine pass/fail threshold based on mode
int passThreshold = strictMode ? 70 : 60;
if (adjustedScore >= passThreshold) {
passCount++;
}
String letterGrade;
if (adjustedScore >= 90) {
letterGrade = "A";
aCount++;
} else if (adjustedScore >= 80) { // Branch 5
letterGrade = "B";
bCount++;
} else if (adjustedScore >= 70) { // Branch 6
letterGrade = "C";
cCount++;
} else if (adjustedScore >= 60) { // Branch 7
letterGrade = "D";
dCount++;
} else { // Branch 8
letterGrade = "F";
TextAnalyzer > src > StudentGradeProcessor > GradeResult > getCCount
45:50 CRLF UTF-8 4 spaces
27°C
06:51
2025/09/24



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

```
    @Test
public void testNullValues(){
        List<Integer> scores = Arrays.asList(100, null, 95, null);
        int bonusPoints = 2;
        boolean strictMode = false;
        StudentGradeProcessor p = new StudentGradeProcessor();
        StudentGradeProcessor.GradeResult result = p.processGrades(scores, bonusPoints, strictMode);

        assertEquals( expected: 4,result.getTotalStudents());
        assertEquals( expected: 2,result.getPassCount());
        assertEquals( expected: 49.25, result.getAverage(),  delta: 0.01);
        assertEquals( expected: 2,result.getACount());
        assertEquals( expected: 0,result.getBCount());
        assertEquals( expected: 0,result.getCCount());
        assertEquals( expected: 0,result.getDCount());
        assertEquals( expected: 0,result.getFCount());
    }
```

Test Case 3:
In this case I check if bounus all scores not cappted on 100

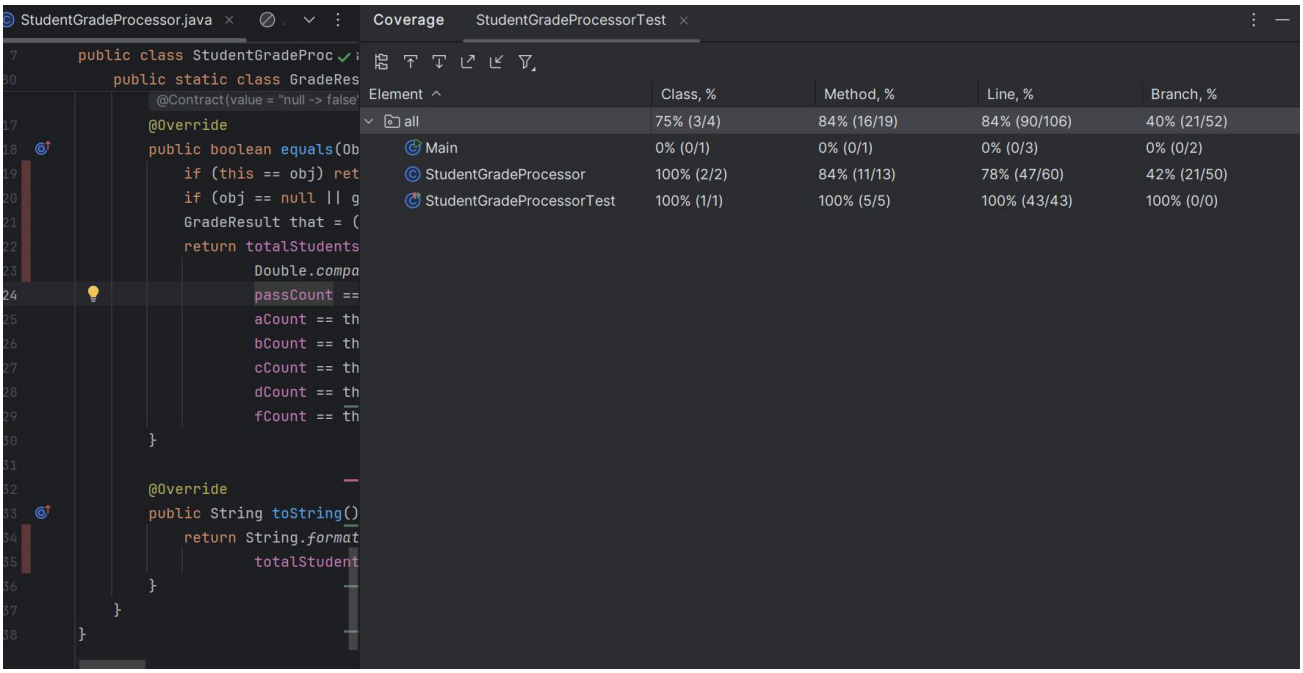
```
@Test
void testBonusCappingAt100() {
    List<Integer> scores = Arrays.asList(98, 100, 95);
    int bonusPoints = 5;
    boolean strictMode = false;
    StudentGradeProcessor p = new StudentGradeProcessor();
    StudentGradeProcessor.GradeResult result = p.processGrades(scores, bonusPoints, strictMode);
    // 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



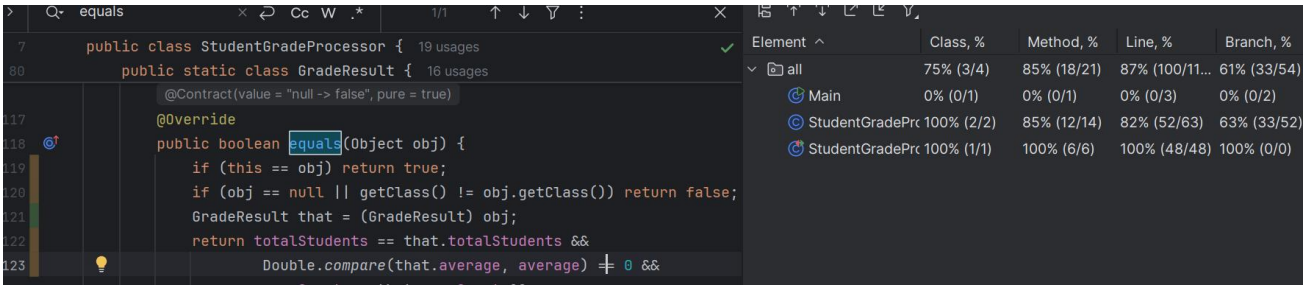
After add more test cases :
1: Make sure the equals method works correctly in all scenarios.

```
@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());
}
```



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	
StudentGradeProcessor.java	95%	37/39	72%	21/29	75%	21/28

Report generated by PIT 1.20.0

Mutations

11	1. negated conditional → KILLED 2. negated conditional → KILLED
12	1. replaced return value with null for StudentGradeProcessor::processGrades → NO_COVERAGE
23	1. negated conditional → KILLED
27	1. Replaced integer addition with subtraction → KILLED
30	1. changed conditional boundary → SURVIVED Covering tests 2. negated conditional → KILLED
34	1. Replaced integer addition with subtraction → KILLED
37	1. negated conditional → SURVIVED Covering tests
38	1. changed conditional boundary → SURVIVED Covering tests 2. negated conditional → KILLED
39	1. Changed increment from 1 to -1 → KILLED
44	1. negated conditional → KILLED 2. changed conditional boundary → SURVIVED Covering tests
46	1. Changed increment from 1 to -1 → KILLED
47	1. changed conditional boundary → SURVIVED Covering tests 2. negated conditional → KILLED
49	1. Changed increment from 1 to -1 → KILLED
50	1. negated conditional → KILLED 2. changed conditional boundary → SURVIVED Covering tests
52	1. Changed increment from 1 to -1 → KILLED
53	1. changed conditional boundary → SURVIVED Covering tests 2. negated conditional → KILLED
55	1. Changed increment from 1 to -1 → KILLED
58	1. Changed increment from 1 to -1 → KILLED
64	1. Replaced double division with multiplication → KILLED
69	1. negated conditional → KILLED 2. negated conditional → KILLED
74	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%	37/39	93%	27/29	96%	27/28

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
StudentGradeProcessor.java	95%	37/39	93%	27/29	96%	27/28

Report generated by PIT 1.20.0

Mutations

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