Software Testing – CSE337s

Spring 2023

Level: senior-2

Group number: 8

- **Team members**

Name	ID
1- mohamed Saad Ahmed Saeed	1808369
2- Mohamed Sayed ibrahime ammar	1801149
3- Osama Muhammad Ramadan	1902255
4- Mohamed Nasser Ali Mohamed	1804727
5- Kyrillos Phelopos Sawiris	1804628

Participation percentage 20 % for each of us.

Application description:

The application input is a file. The application reads each line in this file as one string where each of its fields are separated by comma “,”.

The first line of the file contains the subject name, subject-code and the full mark of that subject where each of their fields are separated by comma “,”
Each of the following lines of that file (starting from line 2 to the end of file) should consists of the following items Student name, Student number, Student Activities mark, Oral/Practical mark, Midterm exam mark and Final exam mark the application result is to produce the GPA and Grade in this subject.

Our design:

The application consists of three classes (courseRecord, studentRecord, fileManager).
courseRecord: responsible for validating courses information and parsing the students’ records.
studentRecord: responsible for validating students’ information, calculating the full mark and setting the GPA and grade for the subject.
fileManager: to handing opening the input file, parsing it and write to the output file.

To test:

We created three classes to test courseRecord, studentRecord and fileManager.

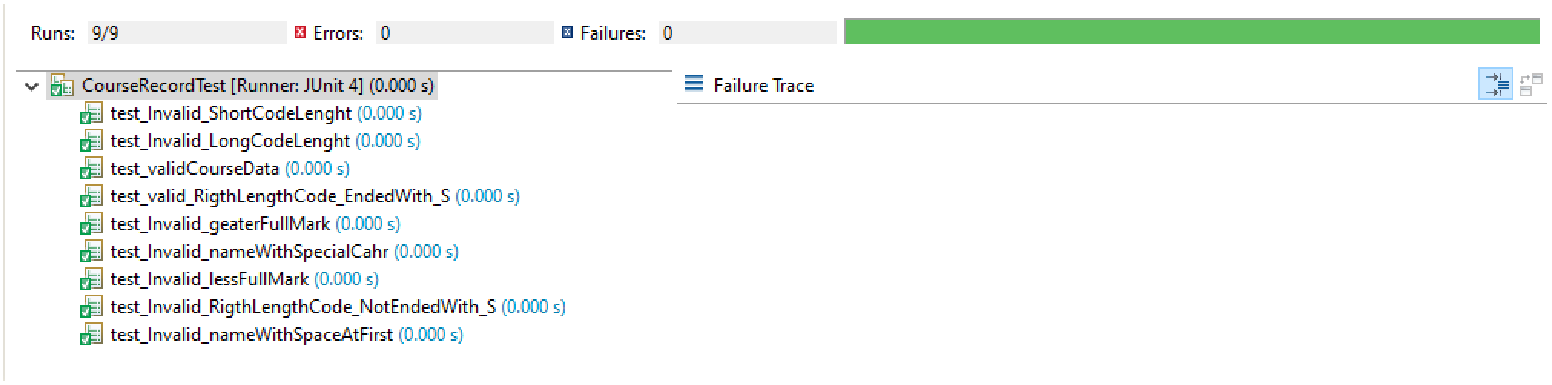
The following table shows the test cases.

Scen #	Scenario Description	Req #	Cond #	Test Data	Test Conditions/Steps	Expected Results/Comments	Post-Conditions	Actual Results	Pass/Fail (Y/N)
1	Test input course name			English,ENG101, 100\n …	1.write the input data in the input file 2.open the application and run it	English	-	English	Y
2	Test input course code			English,ENG101, 100\n …	1.write the input data in the input file 2.open the application and run it	ENG101	-	ENG101	Y
3	Test input course full mark			English,ENG101, 100\n …	1.write the input data in the input file 2.open the application and run it	100		100	Y
4	Test input student data			Input_file*	1.write the input data in the input file 2.open the application and run it	True		True	Y
5	Test the number of students in the input file			Input_file*	1.write the input data in the input file 2.open the application and run it	7		7	Y
6	Test the path of the input file			\\sample_input.txt	1.write the input data in the input file 2.open the application and run it	File data is read		File data is read successfully	Y
8	Test if the file doesn't exists			\\nonexistent.txt	1.write the input data in the input file 2.open the application and run it				Y
9	Test empty file			\\testEmptyFile.txt	1.write the input data in the input file 2.open the application and run it	File is read		Empty data is read	Y

Scen #	Scenario Description	Req #	Cond #	Test Data	Test Conditions/Steps	Expected Results/Comments	Post-Conditions	Actual Results	Pass/Fail (Y/N)
10	Test file with single line			\\singleLineFile.txt"	1.write the input data in the input file 2.open the application and run it	English,ENG101,100\n		English,ENG101,100\n	Y
11	Test single line with invalid data			\\invalidFirstLine.txt	1.write the input data in the input file 2.open the application and run it	Empty		Empty	Y

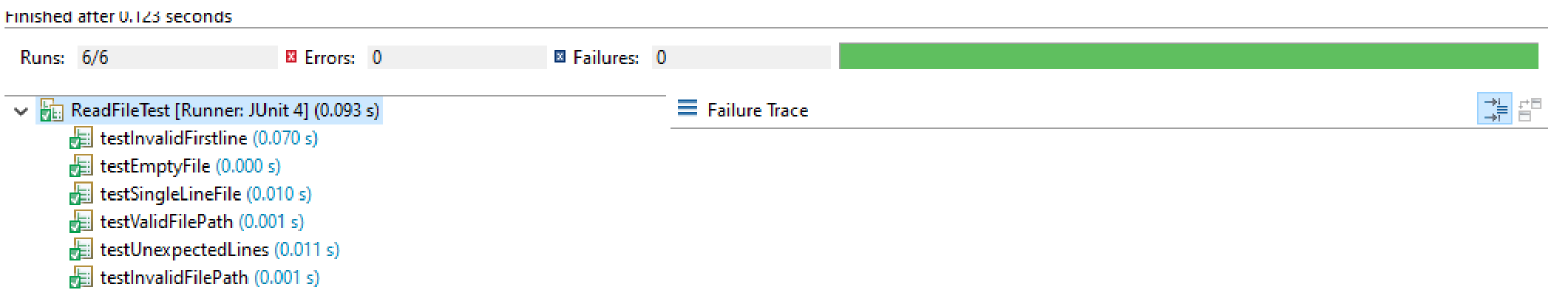
Screenshots for running test cases

- ### Course record


Runs: 9/9 Errors: 0 Failures: 0

CourseRecordTest [Runner: JUnit 4] (0.000 s)
- test_Invalid_ShortCodeLenght (0.000 s)
- test_Invalid_LongCodeLenght (0.000 s)
- test_validCourseData (0.000 s)
- test_valid_RigthLengthCode_EndedWith_S (0.000 s)
- test_Invalid_geaterFullMark (0.000 s)
- test_Invalid_nameWithSpecialCahr (0.000 s)
- test_Invalid_lessFullMark (0.000 s)
- test_Invalid_RigthLengthCode_NotEndedWith_S (0.000 s)
- test_Invalid_nameWithSpaceAtFirst (0.000 s)

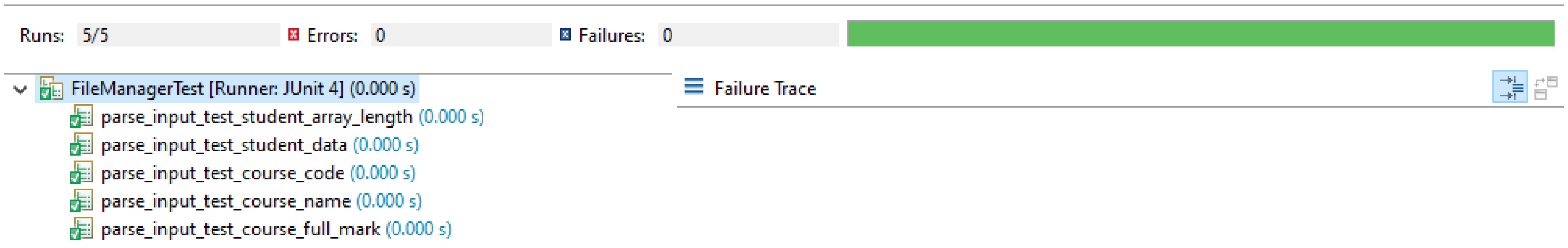
- ### Read file


Finished after 0.123 seconds

Runs: 6/6 Errors: 0 Failures: 0

ReadFileTest [Runner: JUnit 4] (0.093 s)
- testInvalidFirstline (0.070 s)
- testEmptyFile (0.000 s)
- testSingleLineFile (0.010 s)
- testValidFilePath (0.001 s)
- testUnexpectedLines (0.011 s)
- testInvalidFilePath (0.001 s)

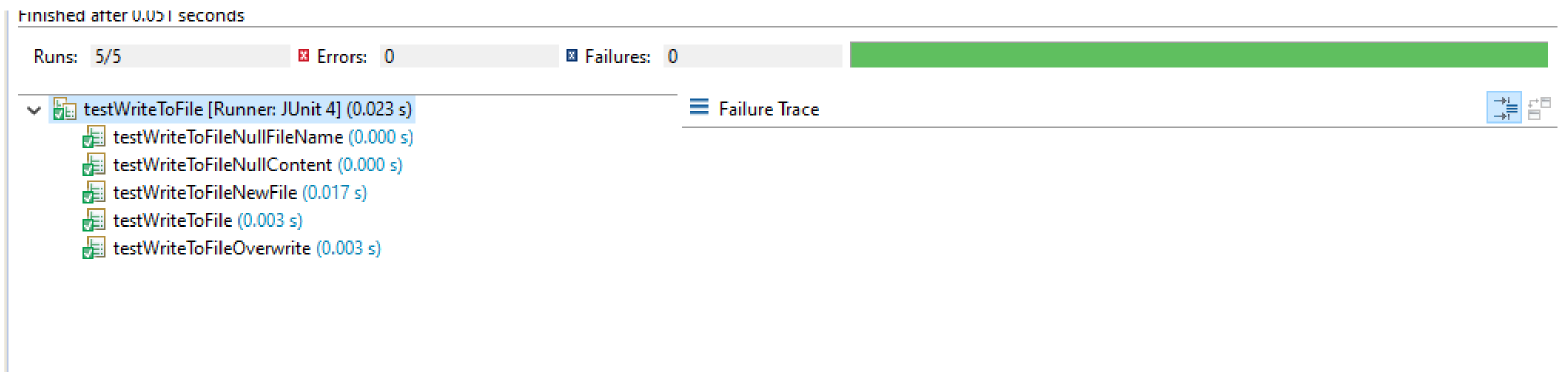
- ## Parse input data


Runs: 5/5 Errors: 0 Failures: 0

FileManagerTest [Runner: JUnit 4] (0.000 s)
- parse_input_test_student_array_length (0.000 s)
- parse_input_test_student_data (0.000 s)
- parse_input_test_course_code (0.000 s)
- parse_input_test_course_name (0.000 s)
- parse_input_test_course_full_mark (0.000 s)

Failure Trace

- ## Write to file

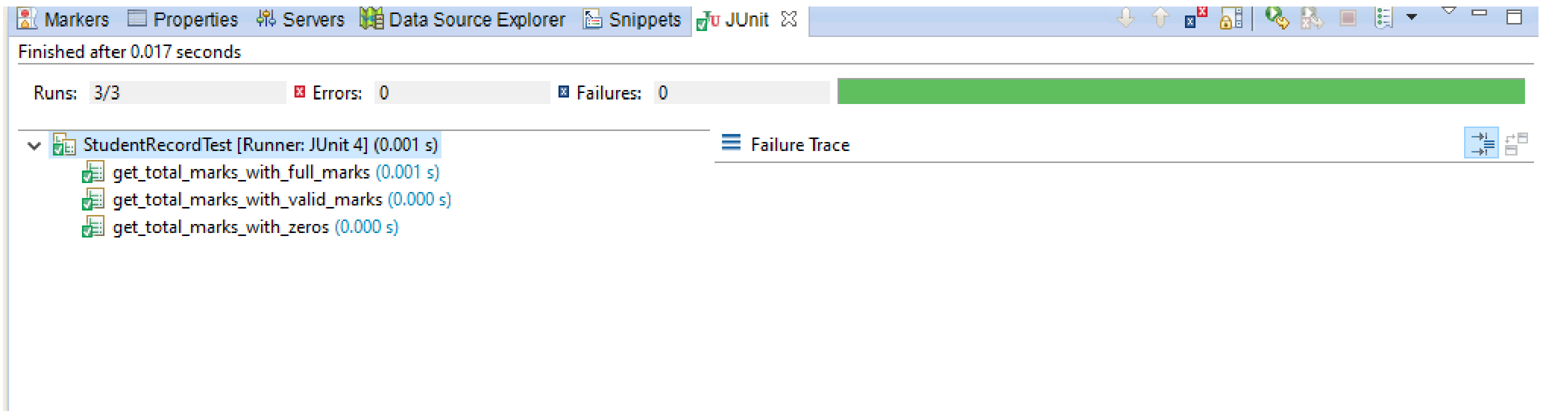

Finished after 0.051 seconds

Runs: 5/5 Errors: 0 Failures: 0

testWriteToFile [Runner: JUnit 4] (0.023 s)
- testWriteToFileNullFileName (0.000 s)
- testWriteToFileNullContent (0.000 s)
- testWriteToFileNewFile (0.017 s)
- testWriteToFile (0.003 s)
- testWriteToFileOverwrite (0.003 s)

Failure Trace

- ## stutent recorde - get total mark test cases



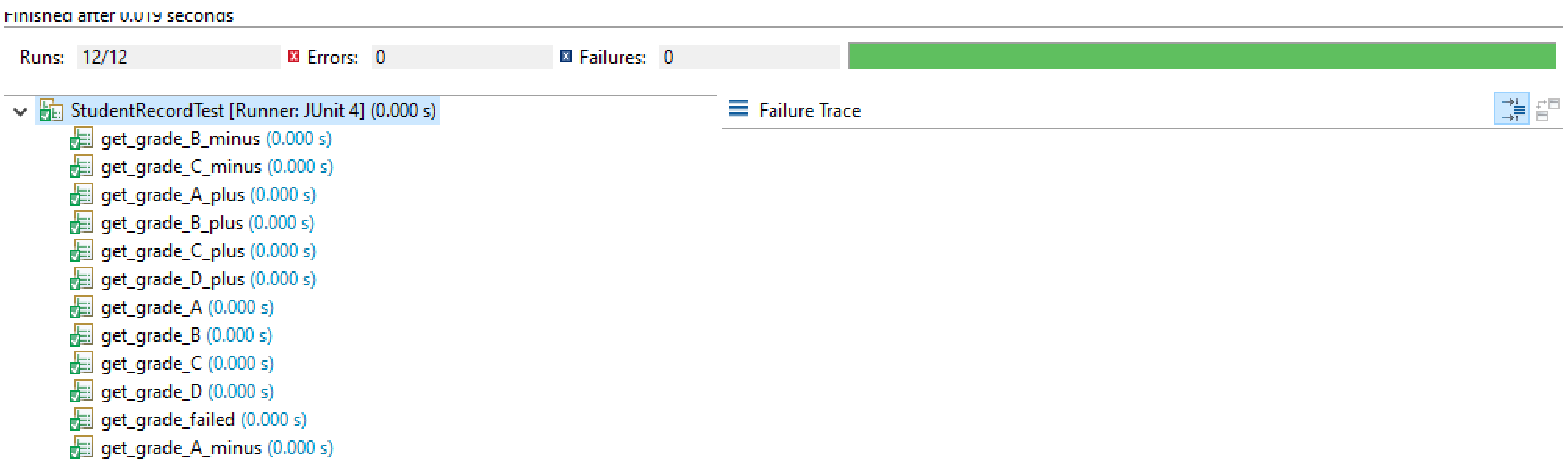
Finished after 0.017 seconds

Runs: 3/3 Errors: 0 Failures: 0

StudentRecordTest [Runner: JUnit 4] (0.001 s)
- get_total_marks_with_full_marks (0.001 s)
- get_total_marks_with_valid_marks (0.000 s)
- get_total_marks_with_zeros (0.000 s)

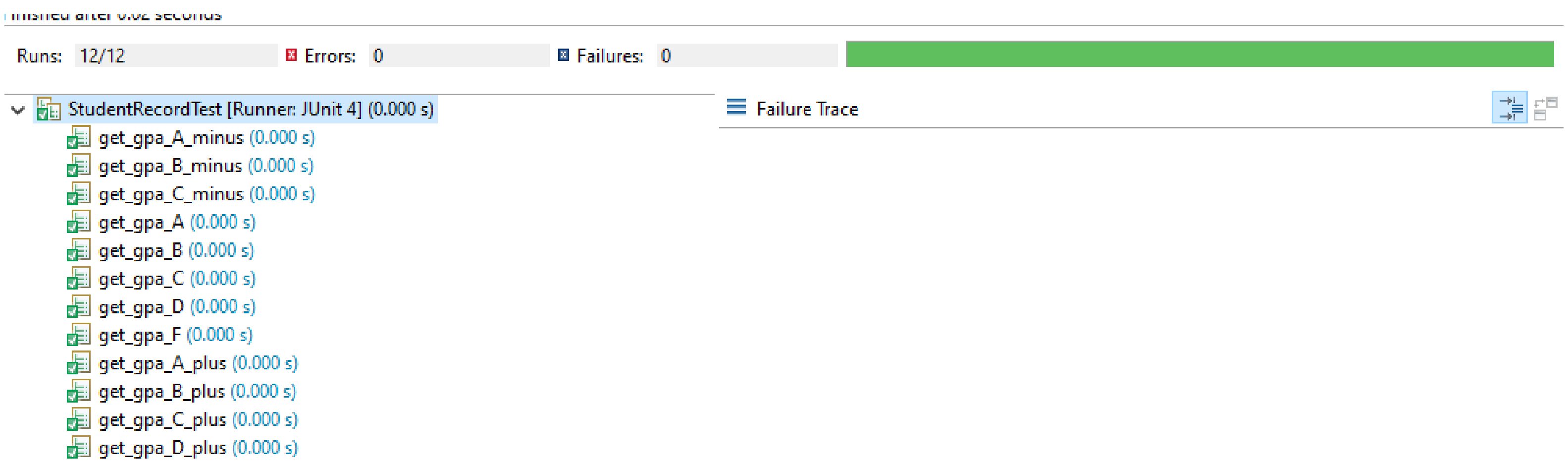
Failure Trace

- ## getGrade testcases



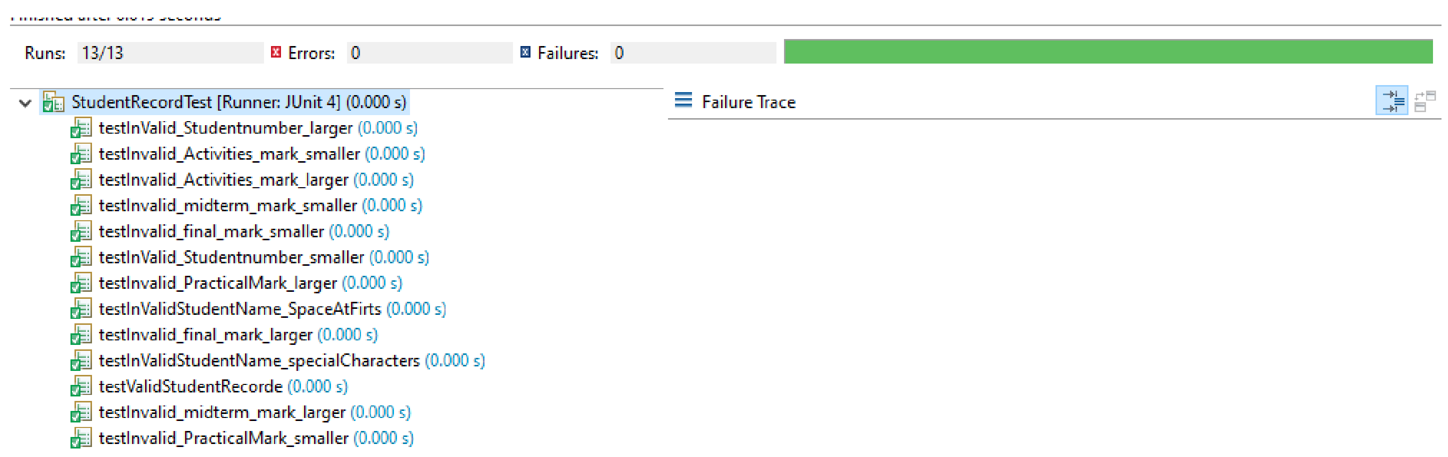
```
Finished after 0.019 seconds
Runs: 12/12    Errors: 0    Failures: 0

StudentRecordTest [Runner: JUnit 4] (0.000 s)        Failure Trace
    get_grade_B_minus (0.000 s)
    get_grade_C_minus (0.000 s)
    get_grade_A_plus (0.000 s)
    get_grade_B_plus (0.000 s)
    get_grade_C_plus (0.000 s)
    get_grade_D_plus (0.000 s)
    get_grade_A (0.000 s)
    get_grade_B (0.000 s)
    get_grade_C (0.000 s)
    get_grade_D (0.000 s)
    get_grade_failed (0.000 s)
    get_grade_A_minus (0.000 s)
```

- ## get GPA testcases



```
Finished after 0.02 seconds
Runs: 12/12    Errors: 0    Failures: 0

StudentRecordTest [Runner: JUnit 4] (0.000 s)        Failure Trace
    get_gpa_A_minus (0.000 s)
    get_gpa_B_minus (0.000 s)
    get_gpa_C_minus (0.000 s)
    get_gpa_A (0.000 s)
    get_gpa_B (0.000 s)
    get_gpa_C (0.000 s)
    get_gpa_D (0.000 s)
    get_gpa_F (0.000 s)
    get_gpa_A_plus (0.000 s)
    get_gpa_B_plus (0.000 s)
    get_gpa_C_plus (0.000 s)
    get_gpa_D_plus (0.000 s)
```

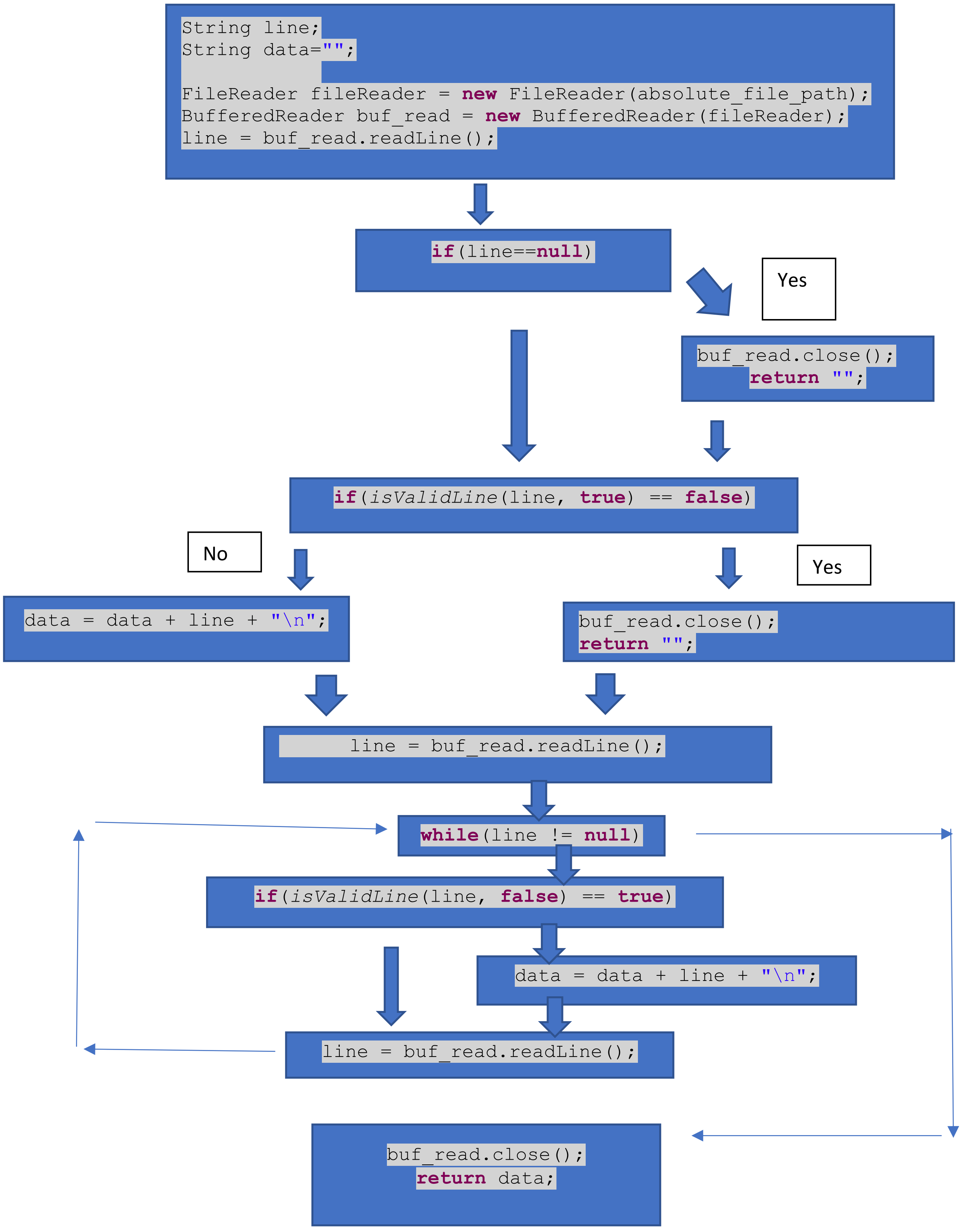
- ## student data



Runs: 13/13 Errors: 0 Failures: 0

```
StudentRecordTest [Runner: JUnit 4] (0.000 s)
    testInValid_Studentnumber_larger (0.000 s)
    testInvalid_Activities_mark_smaller (0.000 s)
    testInvalid_Activities_mark_larger (0.000 s)
    testInvalid_midterm_mark_smaller (0.000 s)
    testInvalid_final_mark_smaller (0.000 s)
    testInValid_Studentnumber_smaller (0.000 s)
    testInvalid_PracticalMark_larger (0.000 s)
    testInValidStudentName_SpaceAtFirts (0.000 s)
    testInvalid_final_mark_larger (0.000 s)
    testInValidStudentName_specialCharacters (0.000 s)
    testValidStudentRecorde (0.000 s)
    testInvalid_midterm_mark_larger (0.000 s)
    testInvalid_PracticalMark_smaller (0.000 s)
    /**
* A function to read a file & returns data as string
     * @throws FileNotFoundException
     */
```

- `public static String read_file(String absolute_file_path)`



```
String line;
String data="";

FileReader fileReader = new FileReader(absolute_file_path);
BufferedReader buf_read = new BufferedReader(fileReader);
line = buf_read.readLine();
```

```
if(line==null)
```

```
buf_read.close();
return "";
```

```
if(isValidLine(line, true) == false)
```

```
data = data + line + "\n";
```

```
buf_read.close();
return "";
```

```
line = buf_read.readLine();
```

```
while(line != null)
```

```
if(isValidLine(line, false) == true)
```

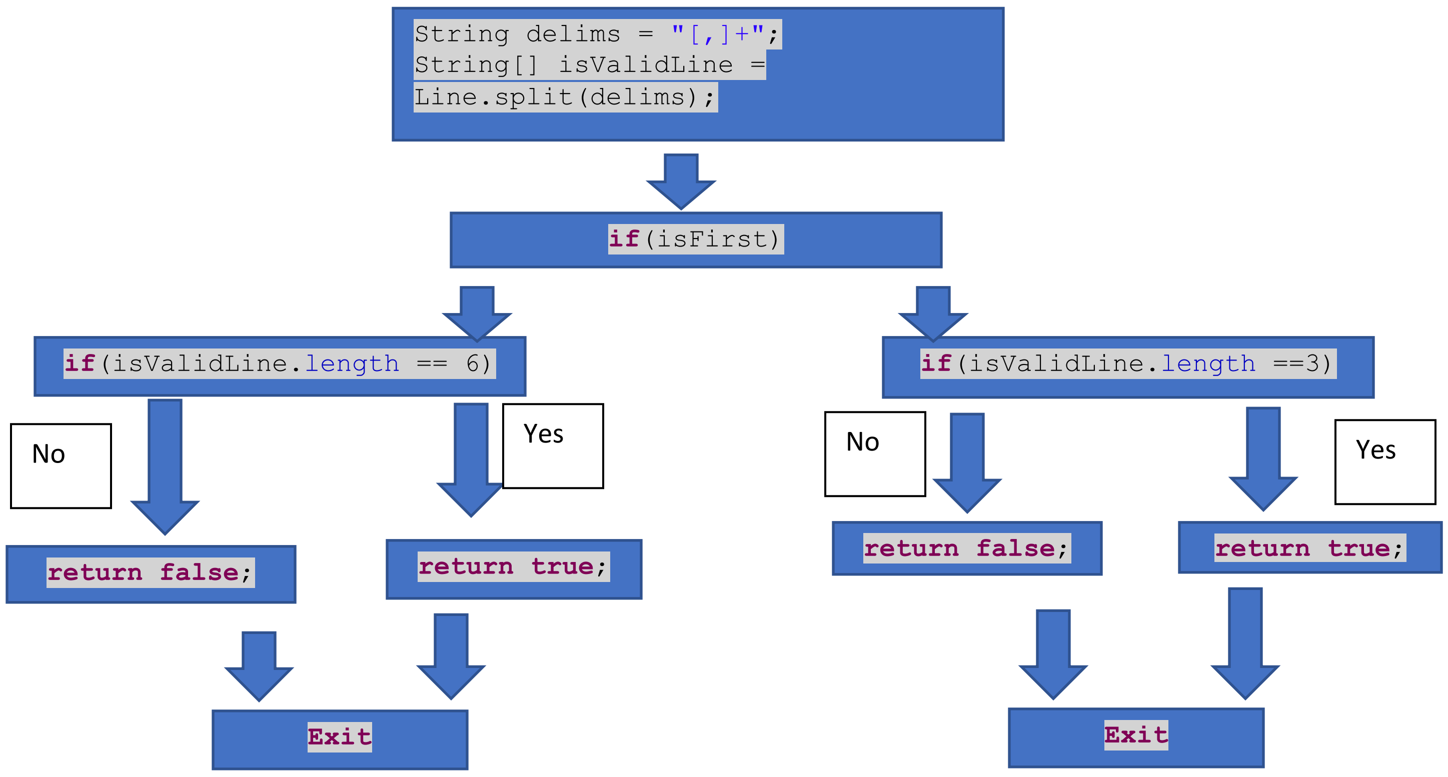
```
data = data + line + "\n";
```

```
line = buf_read.readLine();
```

```
buf_read.close();
return data;
```

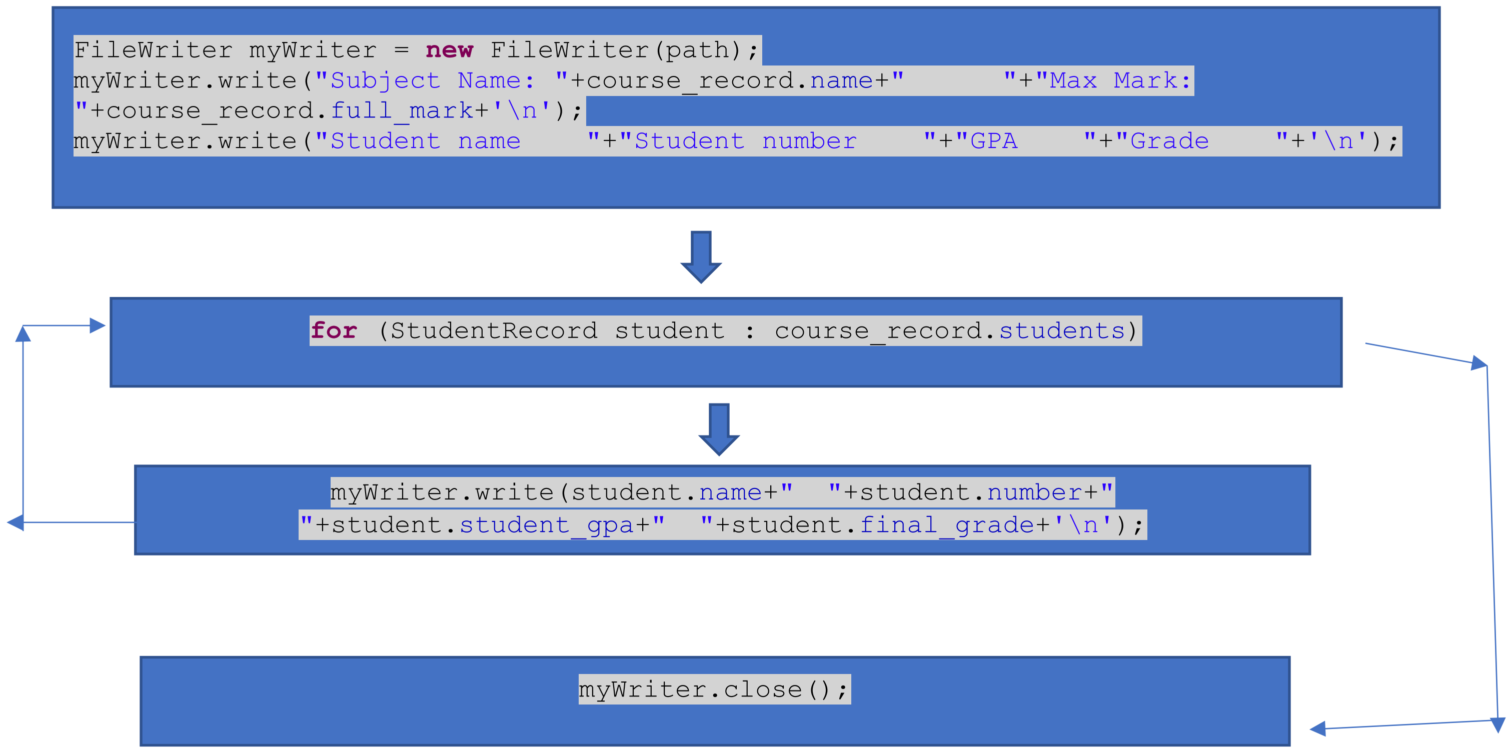
```
/**
       * A function to check if the lines in the file are in the expected     format or
not
       * inputs 1- line to check its format
       *        2- boolean if its the first line in the file or not
       *
       */
```

```
public static boolean isValidLine(String Line,boolean isFirst)
```



```
/*
      * A function to write a CourseRecord into a file
      * */
```

- **public static void write_file(CourseRecord course_record , String path)**



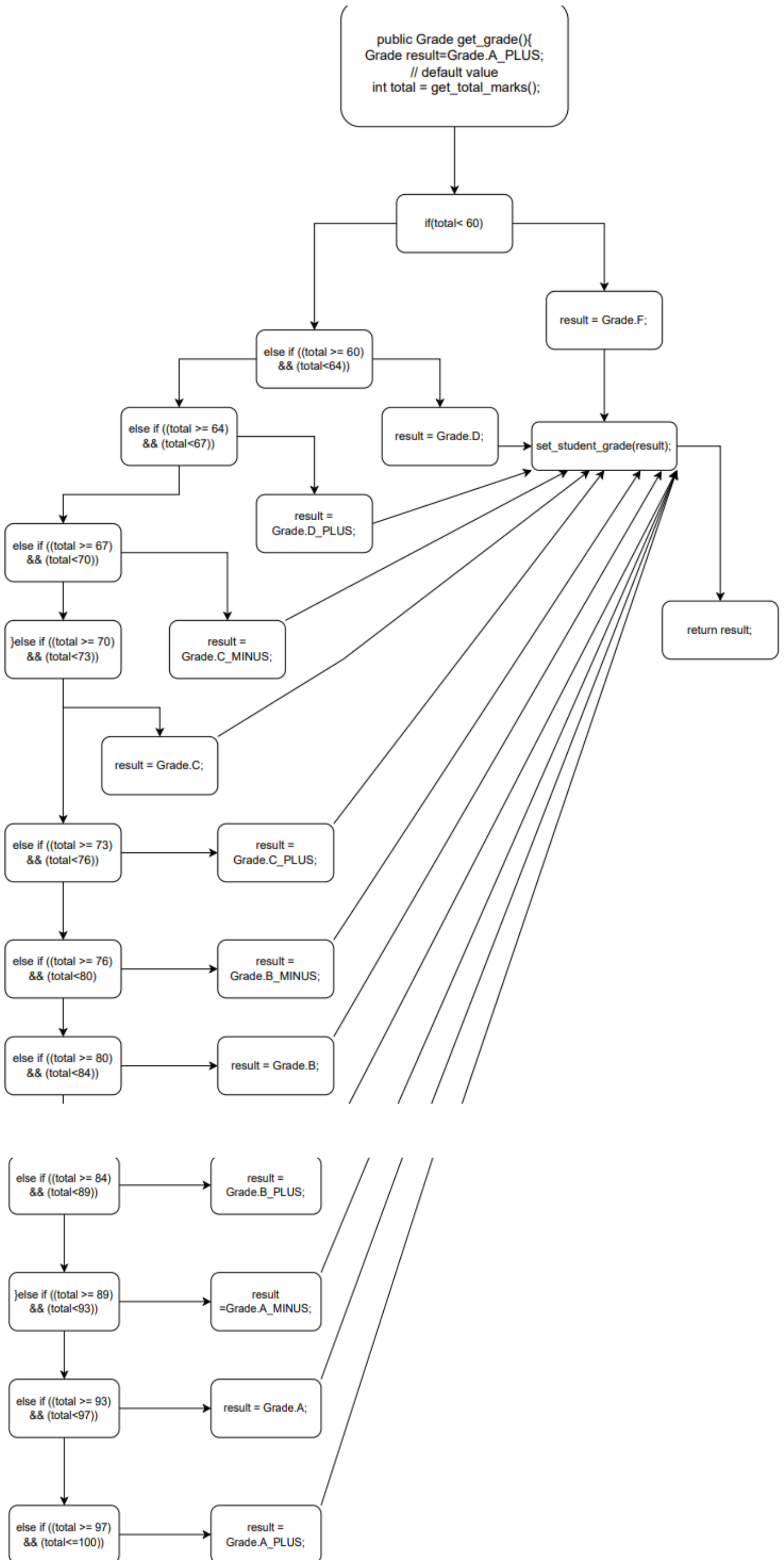
```
FileWriter myWriter = new FileWriter(path);
myWriter.write("Subject Name: "+course_record.name+"       "+"Max Mark: "+course_record.full_mark+'\n');
myWriter.write("Student name    "+"Student number    "+"GPA    "+"Grade    "+'\n');
```

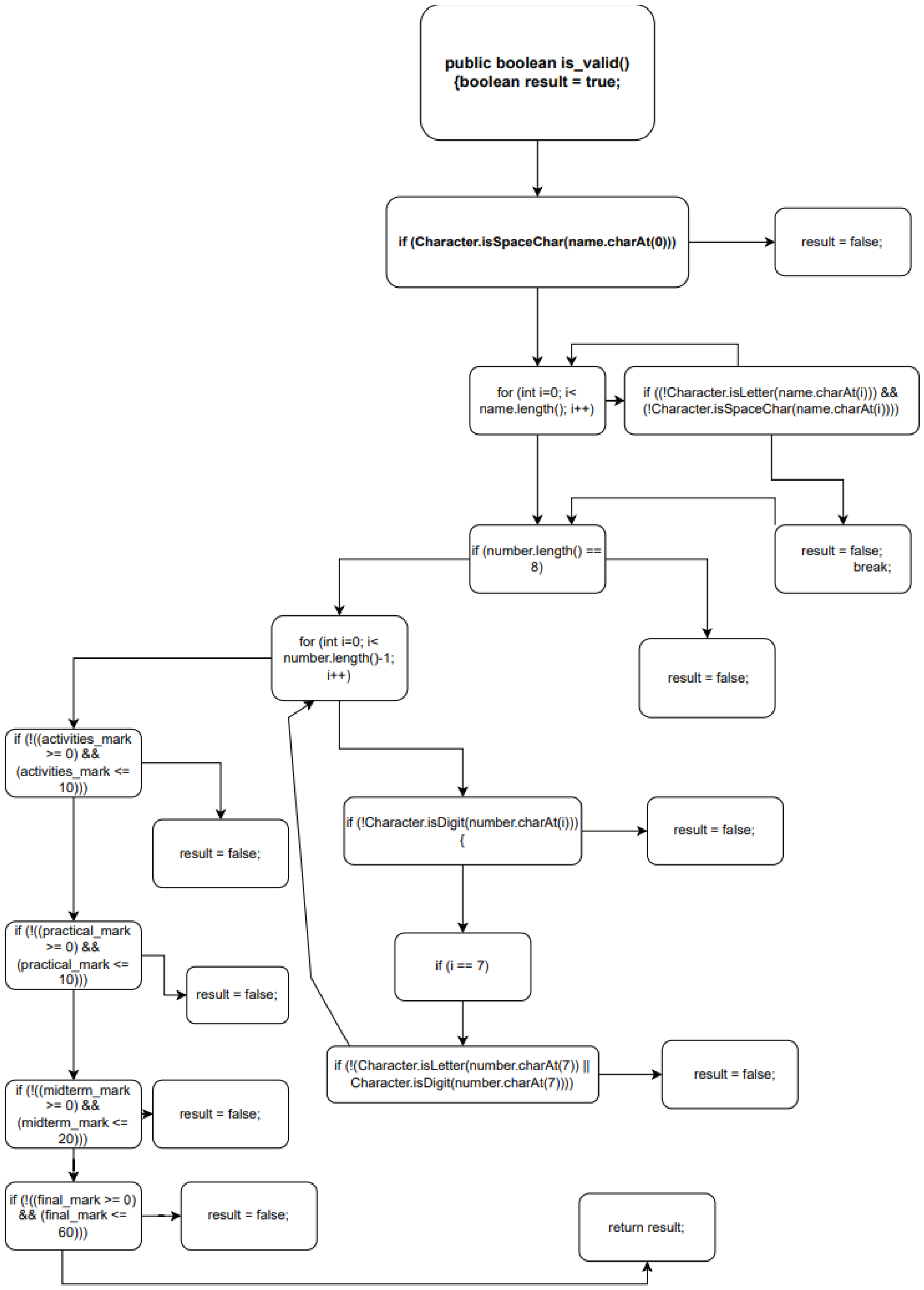
```
for (StudentRecord student : course_record.students)
```

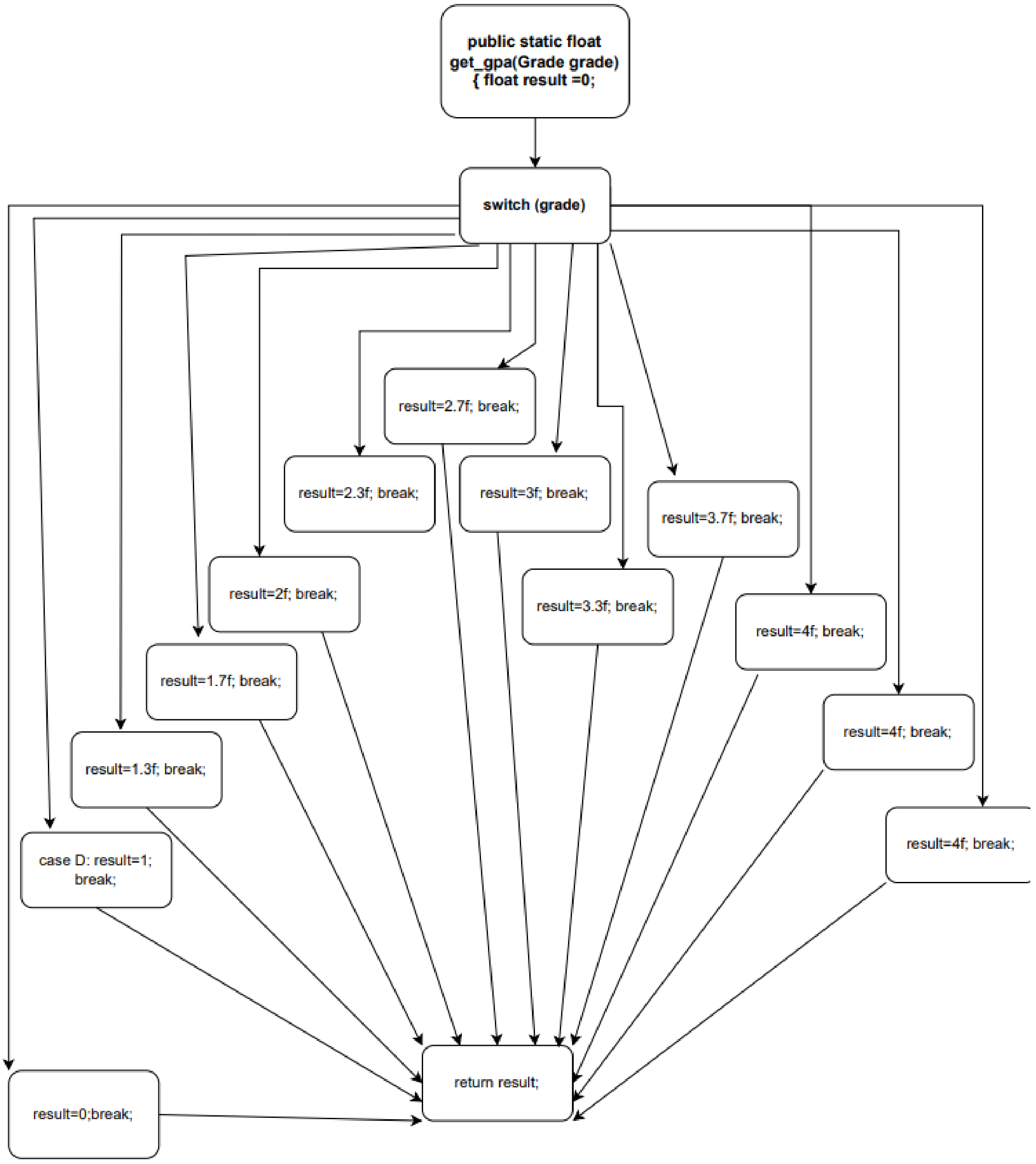
```
myWriter.write(student.name+"  "+student.number+"  "+student.student_gpa+"  "+student.final_grade+'\n');
```

```
myWriter.close();
```

- **Student record class methods:**


public Grade get_grade(){
Grade result=Grade.A_PLUS;
// default value
int total = get_total_marks();
if(total< 60)
result = Grade.F;
else if ((total >= 60) && (total<64))
result = Grade.D;
set_student_grade(result);
return result;
else if ((total >= 64) && (total<67))
result = Grade.D_PLUS;
else if ((total >= 67) && (total<70))
result = Grade.C_MINUS;
}else if ((total >= 70) && (total<73))
result = Grade.C;
else if ((total >= 73) && (total<76))
result = Grade.C_PLUS;
else if ((total >= 76) && (total<80)
result = Grade.B_MINUS;
else if ((total >= 80) && (total<84))
result = Grade.B;
else if ((total >= 84) && (total<89))
result = Grade.B_PLUS;
}else if ((total >= 89) && (total<93))
result =Grade.A_MINUS;
else if ((total >= 93) && (total<97))
result = Grade.A;
else if ((total >= 97) && (total<=100))
result = Grade.A_PLUS;


public boolean is_valid() {boolean result = true;
if (Character.isSpaceChar(name.charAt(0)))
result = false;
for (int i=0; i< name.length(); i++)
if ((!Character.isLetter(name.charAt(i))) && (!Character.isSpaceChar(name.charAt(i))))
result = false; break;
if (number.length() == 8)
result = false;
for (int i=0; i< number.length()-1; i++)
if (!Character.isDigit(number.charAt(i))) {
result = false;
if (i == 7)
if (!(Character.isLetter(number.charAt(7)) || Character.isDigit(number.charAt(7))))
result = false;
if (!((activities_mark >= 0) && (activities_mark <= 10)))
result = false;
if (!((practical_mark >= 0) && (practical_mark <= 10)))
result = false;
if (!((midterm_mark >= 0) && (midterm_mark <= 20)))
result = false;
if (!((final_mark >= 0) && (final_mark <= 60)))
result = false;
return result;

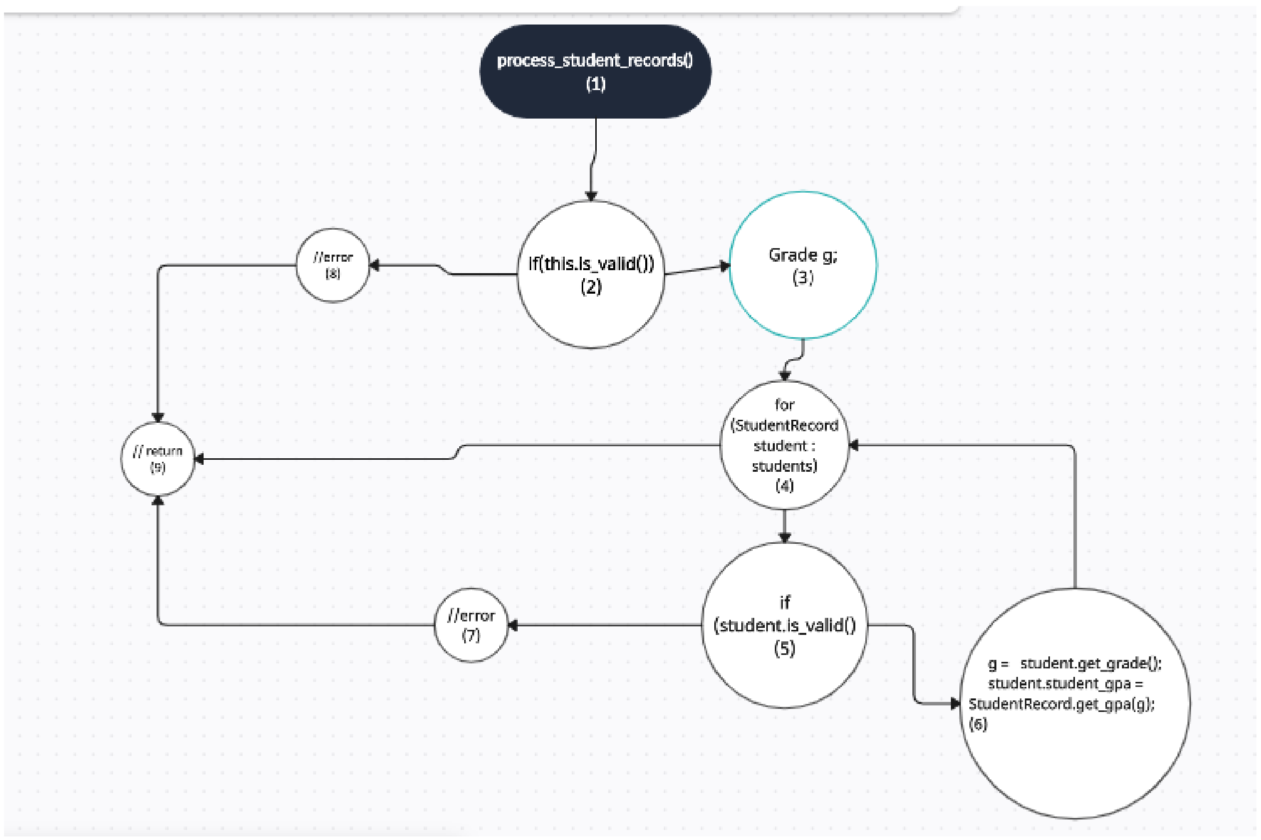

public static float get_gpa(Grade grade) { float result =0;
switch (grade)
result=2.7f; break;
result=2.3f; break;
result=3f; break;
result=3.7f; break;
result=2f; break;
result=3.3f; break;
result=4f; break;
result=1.7f; break;
result=4f; break;
result=1.3f; break;
result=4f; break;
case D: result=1; break;
return result;
result=0;break;

- **Course record class have two methods:**

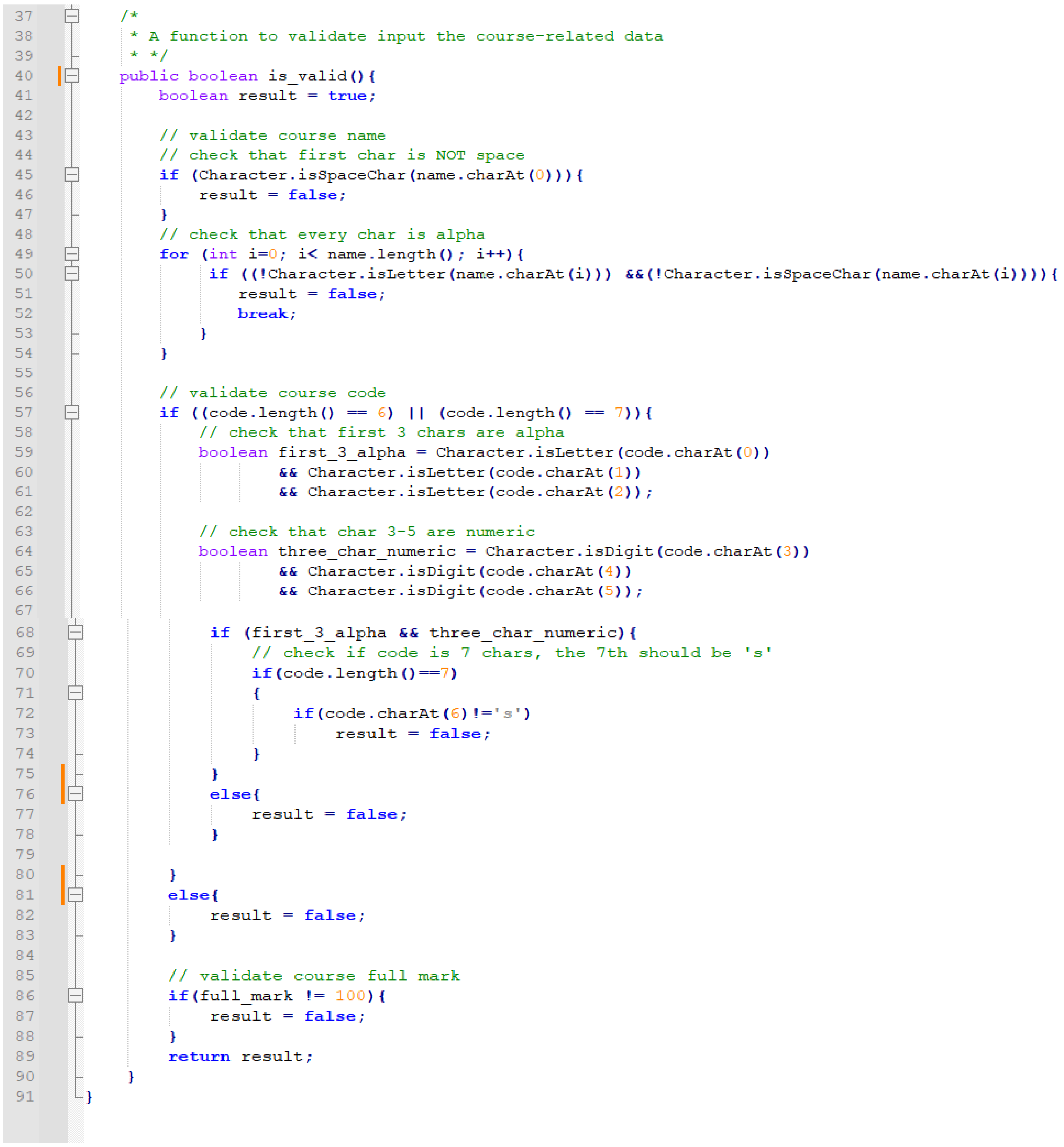
- **1-:**

```java
/*
 * A function to iterate on students & set their grades & GPAs
 * */
public void process_student_records(){

    if(this.is_valid())//check if the course data is valid
    {
        Grade g;
        for (StudentRecord student : students) {

            if (student.is_valid()){
                // calculate the student grade
                g =   student.get_grade();
                student.student_gpa = StudentRecord.get_gpa(g); // set the GPA based on the grade
            }else{
                // error handling
            }
        }
    }
    else
    {
        //error handling
    }

}
```

- **Control flow graph-:**

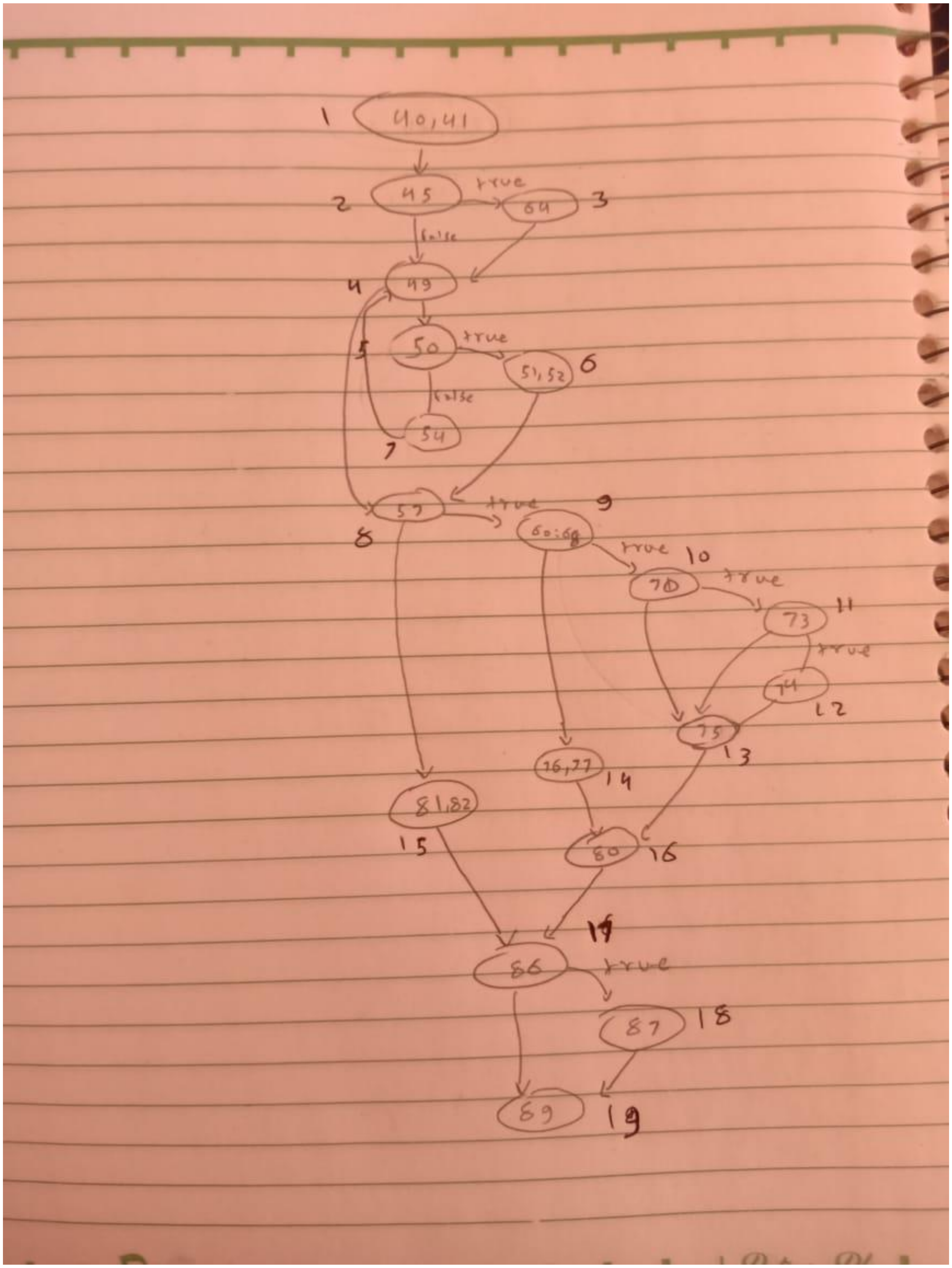

process_student_records()
(1)
If(this.is_valid())
(2)
Grade g;
(3)
//error
(8)
for (StudentRecord student : students)
(4)
// return
(9)
if (student.is_valid()
(5)
//error
(7)
g = student.get_grade(); student.student_gpa = StudentRecord.get_gpa(g);
(6)

2-:



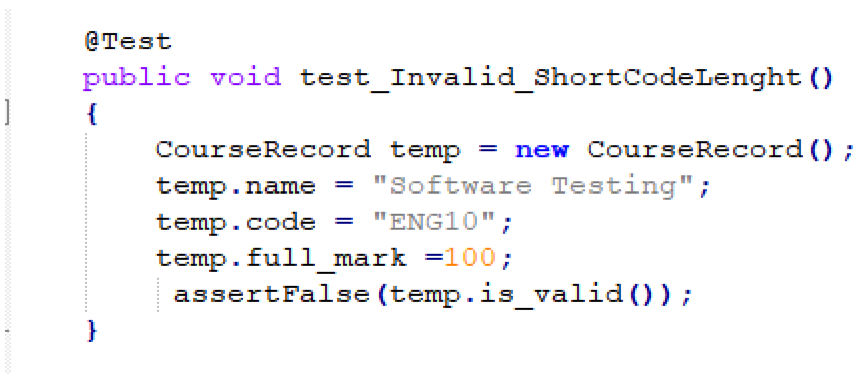
```
   /*
    * A function to validate input the course-related data
    * */
   public boolean is_valid(){
       boolean result = true;

       // validate course name
       // check that first char is NOT space
       if (Character.isSpaceChar(name.charAt(0))){
           result = false;
       }
       // check that every char is alpha
       for (int i=0; i< name.length(); i++){
            if ((!Character.isLetter(name.charAt(i))) &&(!Character.isSpaceChar(name.charAt(i)))){
               result = false;
               break;
           }
       }

       // validate course code
       if ((code.length() == 6) || (code.length() == 7)){
           // check that first 3 chars are alpha
           boolean first_3_alpha = Character.isLetter(code.charAt(0))
                   && Character.isLetter(code.charAt(1))
                   && Character.isLetter(code.charAt(2));

           // check that char 3-5 are numeric
           boolean three_char_numeric = Character.isDigit(code.charAt(3))
                   && Character.isDigit(code.charAt(4))
                   && Character.isDigit(code.charAt(5));

            if (first_3_alpha && three_char_numeric){
                // check if code is 7 chars, the 7th should be 's'
                if(code.length()==7)
                {
                    if(code.charAt(6)!='s')
                        result = false;
                }
            }
            else{
                result = false;
            }

        }
        else{
            result = false;
        }

        // validate course full mark
        if(full_mark != 100){
            result = false;
        }
        return result;
    }
}
```

Control flow graph-:



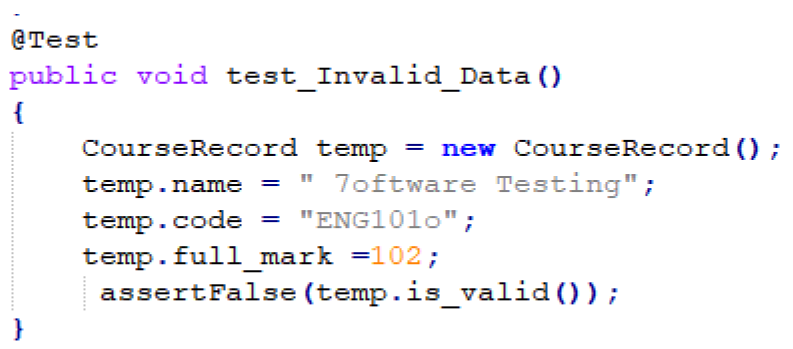
Applying White Box Testing techniques

1- Statement Coverage



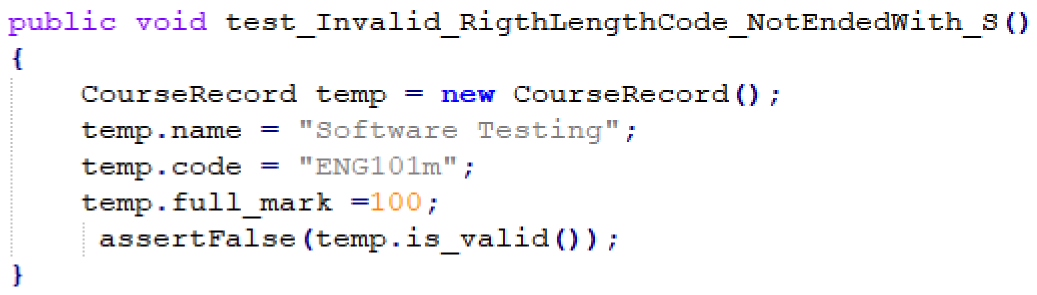
```
@Test
public void test_Invalid_ShortCodeLenght()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG10";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}
```

Cover nodes: 1,2,4,5,7,8,9,10,13,16,17,19



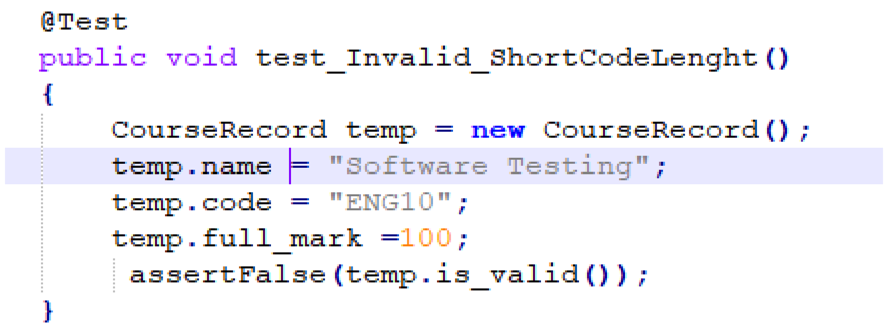
```
@Test
public void test_Invalid_Data()
{
    CourseRecord temp = new CourseRecord();
    temp.name = " 7oftware Testing";
    temp.code = "ENG101o";
    temp.full_mark =102;
     assertFalse(temp.is_valid());
}
```

1,2,3,4,5,6,7,8,9,10,11,12,13,16,17,18,19



```
public void test_Invalid_RigthLengthCode_NotEndedWith_S()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG101m";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}
```

Will hit node 14.

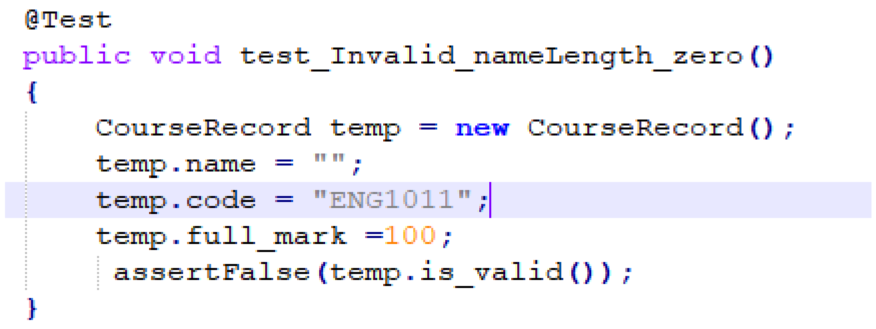


```
@Test
public void test_Invalid_ShortCodeLenght()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG10";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}
```

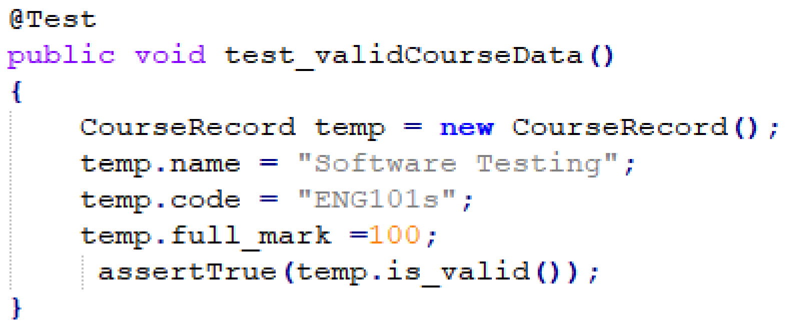
will hit node 15.

2- Branch Coverage

To achieve branch coverage use tests in statement coverage and



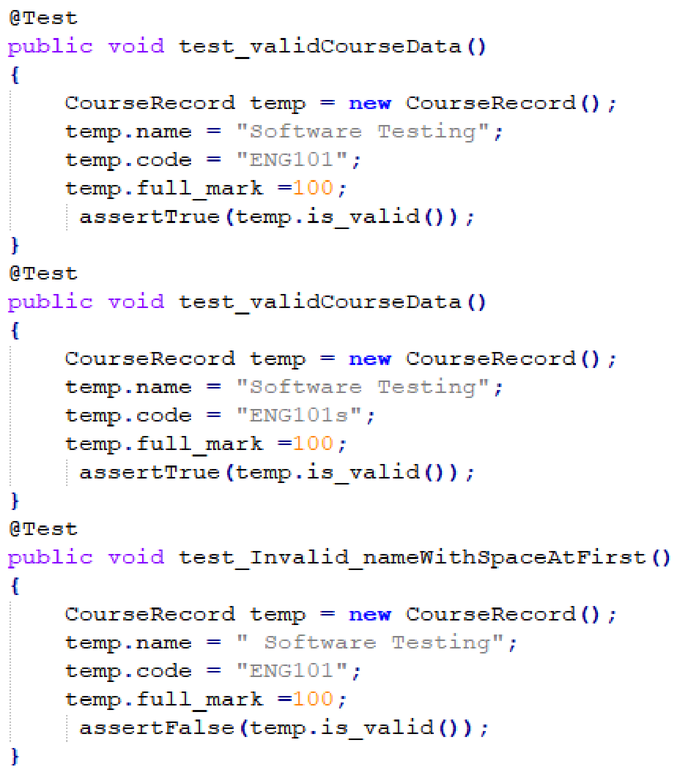
```java
@Test
public void test_Invalid_nameLength_zero()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "";
    temp.code = "ENG1011";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}
```



```java
@Test
public void test_validCourseData()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG101s";
    temp.full_mark =100;
     assertTrue(temp.is_valid());
}
```

3- Path Coverage

We have 8 bounded areas, so we need to 9 tests at least to achieve 100% basis coverage.



```java
@Test
public void test_validCourseData()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG101";
    temp.full_mark =100;
     assertTrue(temp.is_valid());
}
@Test
public void test_validCourseData()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG101s";
    temp.full_mark =100;
     assertTrue(temp.is_valid());
}
@Test
public void test_Invalid_nameWithSpaceAtFirst()
{
    CourseRecord temp = new CourseRecord();
    temp.name = " Software Testing";
    temp.code = "ENG101";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}
```

```
@Test
public void test_Invalid_Data()
{
    CourseRecord temp = new CourseRecord();
    temp.name = " 7oftware Testing";
    temp.code = "ENG101o";
    temp.full_mark =102;
     assertFalse(temp.is_valid());
}

@Test
public void test_Invalid_nameWithSpecialCahr()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing##";
    temp.code = "ENG101";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}

@Test
public void test_Invalid_LongCodeLenght()
{
    CourseRecord temp = new CourseRecord();
    temp.name = "Software Testing";
    temp.code = "ENG1011";
    temp.full_mark =100;
     assertFalse(temp.is_valid());
}
 @Test
 public void test_Invalid_nameLength_zero()
 {
     CourseRecord temp = new CourseRecord();
     temp.name = "";
     temp.code = "ENG1011";
     temp.full_mark =100;
      assertFalse(temp.is_valid());
 }

 @Test
 public void test_Invalid_ShortCodeLenght()
 {
     CourseRecord temp = new CourseRecord();
     temp.name = "Software Testing";
     temp.code = "ENG10";
     temp.full_mark =100;
      assertFalse(temp.is_valid());
 }

 @Test
 public void test_Invalid_RigthLengthCode_NotEndedWith_S()
 {
     CourseRecord temp = new CourseRecord();
     temp.name = "Software Testing";
     temp.code = "ENG101m";
     temp.full_mark =100;
      assertFalse(temp.is_valid());
 }
```

GitHub link

https://github.com/osamamuhammad3623/sw_testing_project