CSD 3464 – ASSIGNMENT 02 (Question 06)

Overview:

The following question is an extension to *Absolute Java* (6th Ed.)'s Chapter 04 Programming Project Q6 (pg. 254). Please follow the instructions included in **this** document and implement the following Java files:

⇒ StudentRecordDemo.java *(Contains a `main()` method)*
⇒ StudentRecord.java

The above classes/files should both be inside a package called *q6*.

Instructions:

Create a `StudentRecord` class with `private` fields that hold the positive integers: `quiz01Points, quiz02Points, quiz03Points, midtermPoints,` and `examPoints`. In addition, the class should have constant integers: `MAX_QUIZ_POINTS = 10`, `MAX_MIDTERM_POINTS = 100`, and `MAX_EXAM_POINTS = 100`.

Additionally, a `private char` field named `letterGrade` is required to hold the character 'A', 'B', 'C', 'D' or 'F' corresponding to the letter grade earned. Also, a `private double` field named `numericGrade` will hold a value between 0-100 representing the percentage grade earned.

The `StudentRecord` class should have two constructors:

1. No-parameter constructor:
   - Should have the method signature: `public StudentRecord()`
   - Set all five non-constant `Points` fields to a default value of 0
   - Set `letterGrade` to 'F'
   - Set `numericGrade` to 0
2. 5-parameter constructor:
   - Should have method signature: `public StudentRecord(int quiz01Points, quiz02Points, quiz03Points, midtermPoints, examPoints)`
   - In the case of **invalid** values:
     - Values less than zero → Set corresponding field to 0
     - Values greater than `MAX_` → Set corresponding field to allowed `MAX_` points
   - Finally, call the `private` method `calculate()` at the end of the constructor to update the `letterGrade` and `numericGrade` fields

In terms of methods, your class requires accessor and mutator methods for **ALL** the non-constant `Points` fields. The `setQuizPoints` mutator takes two parameters `int quizNum` and

`int quizPoints` and updates the corresponding quiz (`quizNum`), if it exists, to the `quizPoints`. Similarly, the accessor `getQuizPoints` takes a single parameter `int quizNum` and returns the points of the appropriate quiz based on `quizNum`; if an invalid `quizNum` is provided return the value -1 from `getQuizPoints`.

The `letterGrade` and `numericGrade` fields **DO NOT** require accessors or mutators. When updating **ALL** `Points` fields via mutators ensure valid inputs are provided and remember to call `calculate()` to update the `letterGrade` and `numericGrade` fields. In the case of **invalid** values provided to the mutators do the following:

- Values less than zero → Set corresponding field to 0
- Values greater than `MAX_` → Set corresponding field to allowed `MAX_` points

Furthermore, the `StudentRecord` class should have a `private` method `calculate()` that calculates and assigns values to the student's `letterGrade` and `numericGrade` fields.

- `numericGrade` is the grade between 0-100 the student receives. It is calculated based on the following weightings:
  - Quizzes are worth 25% of `numericGrade`
  - Midterm is worth 35%
  - Exam is worth 40%
- Once the `numericGrade` is correctly calculated find the appropriate `letterGrade`. `letterGrades` are assigned based on the below table.

Range	Letter Grade
90% or higher	A
80% or higher	B
70% or higher	C
60% or higher	D
Below 60%	F

- The `calculate()` method in terms of lines of code may be on the longer side and this is fine for the purposes of this assignment.

Another method that is to be included is an overridden `public` method for `toString()` which returns a `String` that represents the student's record. The `String` returned should be in the format (replace the `#` and `c` with the appropriate fields, use \n to create newlines) below:

Quiz 01: # points
Quiz 02: # points
Quiz 03: # points
Midterm: # points
Exam: # points
Numeric Grade: # %

Letter Grade: `c`

- **IMPORTANT**: `numericGrade` in `toString()` should be formatted to include up to 2 decimal places, if they exist. Use the `DecimalFormat` class to do this (Chapter 02, pg. 72-76).

Your class must also include the method `public equals()` which compares two `StudentRecord` objects and returns the `boolean` value `true` if and only if both **ALL** the non-constant `Points` fields of the two `StudentRecord` objects are the same.

You are also required to write a `static` class called `StudentRecordDemo` in another file that contains a `main()`. The purpose of this class is to test the functionality of your `StudentRecordDemo` class. It is up to you to decided what to include in the `main()` to test the class; however, all `public` facing methods and constructors need to be tested for correctness. Additionally, you should test to ensure invalid entries (`Points`) passed to the constructors and mutator methods are correctly handled.

The basic structure of your `StudentRecord` class is required to look like the following:

```
public class StudentRecord
{

     public static final int MAX_QUIZ_POINTS = 10;
     public static final int MAX_MIDTERM_POINTS = 100;
     public static final int MAX_EXAM_POINTS = 100;

     private int quiz01Points;
     private int quiz02Points;
     private int quiz03Points;
     private int midtermPoints;
     private int examPoints;

     private char letterGrade;
     private double numericGrade;


     // No Parameter Constructor
     public StudentRecord()
     {
        // insert code here
     }

     // Five Parameter Constructor
     public StudentRecord(int quiz01Points, int quiz02Points, int
quiz03Points, int midtermPoints, int examPoints)
     {
         // insert code here
     }
```

```
// Mutator – QuizPoints
public void setQuizPoints(int quizNum, int quizPoints)
{
    // insert code here
}

// Accessor - QuizPoints
public int getQuizPoints (int quizNum)
{
    // insert code here
}

// Mutator - MidtermPoints
public void setMidtermPoints(int midtermPoints)
{
   // insert code here
}


// Accessor - MidtermPoints
public int getMidtermPoints()
{
   // insert code here
}


// Mutator - ExamPoints
public void setExamPoints(int examPoints)
{
   // insert code here
}


// Accessor - ExamPoints
public int getExamPoints()
{
   // insert code here
}

//calculate method
private void calculate()
{
   // insert code here
}

// toString method
@Override
public String toString()
{
   // insert code here
}
```

```
    // equals method
    @Override
    public boolean equals(StudentRecord stdRcd2)
    {
       // insert code here
    }

}
```

UML Diagram:

The below UML diagram is provided for your convivence. Ensure all shown fields, constructors, and methods are included in your Fraction class prior to your final submission.