CS102 – Algorithms and Programming II
Programming Assignment 6
Spring 2024

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn6_YourSurname_YourName.zip
- Replace the variables “Sec1”, “YourSurname” and “YourName” with your actual section, surname, and name.
- You may ask questions on Moodle and during your section’s lab.
- Upload the above zip file to Moodle by the deadline (if not, significant points will be taken off). You will get a chance to update and improve your solution by consulting with the TAs and tutors during your section’s lab.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Student Grading System

For this assignment, you will implement a console application for a student grading system. You should have a `Student` class that keeps the information about a student with the following variables:

- Name: String
- Surname: String
- School ID: String
- Age: int
- Grades: Grade[]

Include any necessary get and set methods, as well as a `String toString()` method to print the student in the following form: “School ID, Name Surname, Age”

Sample `toString()` result for a student: *9273636110*, *John Doe, 20*

`Grade` class will keep the grade of one exam of a student, you need to store the following information in this class:

- Exam Name: String
- Weight: float
- Points: float

Do not forget to include any necessary methods and the setters and getters. Each student can have any number of grades for different exams. To support this functionality, include a `Grade` array in the Student class. <u>For this assignment, you cannot use the Java collections</u>

framework, so you cannot use an `ArrayList`, and you have to implement this using a standard array (`Grade[] grades;`).

You should make sure that the grades array in the student class has no empty indexes. The constructor of the `Student` class should initialize an empty grades array at first. As we add new grades to that student, you should create a larger array (an array of size "`grades.length +1`", copy the existing grades to the new array, and finally add the new grade to the last position.

For example, for the same student, the grades array could look like the following:

After adding Calculus I - Midterm I:

```
Exam Name: Calculus I - Midterm I
Weight: 15
Grade: 60
```

After adding Calculus I - Midterm II:

```
Exam Name: Calculus I - Midterm I    | Exam Name: Calculus I - Midterm II
Weight: 15                           | Weight: 15
Grade: 60                            | Grade: 79
```

After adding CS 102 - Midterm:

```
Exam Name: Calculus I - Midterm I | Exam Name: Calculus I - Midterm II | Exam Name: CS 102 - Midterm
Weight: 15                        | Weight: 15                         | Weight: 20
Grade: 60                         | Grade: 79                          | Grade: 90
```

Notice how `grades` size changes based on the number of grades we add to this student.

Include a `void setGrade(String examName, float weight, float points)` method into the `Student` class to add a new grade for the student. This method should first look if there is already a grade in this student's grades array with the same exam name. If there is already an entry with the same exam name, this method updates that grade object with the given weight and points. If this exam name does not exist, you should create a new grade object and add it to this student's grades array.

For example, suppose a student's grades array looks like as follows:

```
Exam Name: Calculus I - Midterm I    | Exam Name: Calculus I - Midterm II
Weight: 15                           | Weight: 15
Grade: 60                            | Grade: 79
```

Calling the method `setGrade("Calculus I - Midterm II", 20, 90)` would change the grades as follows:

```
Exam Name: Calculus I - Midterm I    | Exam Name: Calculus I - Midterm II
Weight: 15                           | Weight: 20
Grade: 60                            | Grade: 90
```

In case the given exam name does not already exist for this student, it causes a new grade to be added to the grades.

The exam name of the grade must be longer than 3 characters; throw an exception with the message: examName + " must be longer than 3 characters!", if such an exam name is given to the `setGrade` method. Note that the methods calling `setGrade` would either require a “throws Exception” declaration or exception handling with a try-catch block.

Implement a `School` class to keep an array of students in it. Again, you should not use the Java collections framework; instead, you should use a standard array (Student[] students;). The constructor of the `School` class creates a `Student` array of size 10. Then, whenever the number of students in the school exceeds half of the current length of the students array, double the array size and copy the existing students to the new array. For example, the array size is initially 10, and we can add five students to the school. Then, when we try to add one more student, the array size should become 20. This requires you to keep the number of existing students in the school. You can add any new variables, methods, or classes to support the required functionality. You will keep the students in ascending order by their School ID. When adding a new student, you need to find the correct position to add this student by scanning the existing students; then, you will shift the rest of the students in the array.

Suppose we have the following students array:

25	34	37	55	66	-	-	-	-	-

Suppose we want to add a new student with ID = 35. Adding a new student will cause the student count to be 6, which is greater than 10 / 2 = 5, so we first need to double the array size and move the existing elements into the new array. This requires you to keep the old array in a temporary variable.

The new array will be as follows after we double its size:

25	34	37	55	66	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Then, we locate the index to add the new student whose ID = 35, which is index 2. We need to shift the students after index 2 by 1 to the right and finally add the new student:

25	34	**35**	37	55	66	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Include a `void addStudent(String schoolID, String name, String surname, int age)` method to the `School` class to create and add a new student to the `students` array. School IDs are required to be unique; therefore, this method should first check if the given `schoolID` already belongs to an existing student; if so, you should throw an exception with the message "Duplicate ID: " + schoolID. If the given `schoolID` is not already given to a different student, this method should create a new student object and add it to the correct position in the array, keeping the students in ascending order by Student ID.

Include a `Student getStudent(String schoolID)` method in the `School` class to return the `Student` object that has the given Student ID. <u>This method should use binary search to find the desired student.</u> If the student with the given schoolID exists in the array, this method will return it; otherwise, if there is no student with the given schoolID, you should trigger an exception with the message: “No such student with the id ” + schoolID + “!”.

The `School` class should also include a `Student[] getStudentsByNameOrder()` method to retrieve a sorted copy of the `students` array in ascending name order. <u>You should use Quick Sort to sort the students in this new array.</u> You should not change the order of the students in the original `students` array; this method should return a new array. You may utilize the code from the course slides to implement the sort algorithm. If two students' names are identical, you should compare their surnames.

The method above will be used by a `void printStudentsByNameOrder()` method to print the students' information in name order. This method should use `toString` method of the `Student` class to print each student’s information. Also, include a `void printStudents()` method that prints all the students as they exist in the original `students` array, which is ordered by their School IDs.

The `School` class should include a `float getGradeAverage(Student s)` method to calculate the average grade of a given student as (∑ (weight * points)) / ∑ (weight). In other words, you will sum up the product of each grade’s weight and points and divide it by the total weight of that student's grades to return the average grade. There is no distinction between the student's courses when calculating the average.

Implement a `void printStudentGradeAverages()` method to print each student’s information (using its `toString` method) and grade average together, ordered by the average grade. The students with the same grade average will be ordered by their Student IDs. This method would require you to calculate the average grade of each student and sort them in descending order (You can use any sorting algorithm you want). Exclude the students who have no grade. A sample console output for this method will look as follows:

```
7363361110, Jane Doe, 21 - Average: 100.0
9273636110, John Doe, 20 - Average: 100.0
6694636110, Thomas Anderson, 19 - Average: 97.4
```

Include a method to print out the grades of a student whose Student ID is given to the method `void printGradesOf(String schoolID)` in the `School` class. This method should first find the student using the `getStudent` method to print the student’s information and all the grades this student has. A sample output for this method is as follows:

```
Student: 9273636110, John Doe, 20
Grades:
   Calculus I - Midterm I (Weight: 15.0) 60.0
   Calculus I - Midterm II (Weight: 15.0) 85.0
   History I - Final (Weight: 30.0) 79.0
   Calculus I - Final (Weight: 25.0) 95.0
```

Grades are displayed as they occur in the `grades` array, which is in the order they are added to the array. We display the name of the exam, the weight in parentheses, and the grade.

The `School` class should include a `void processTextFile(String filename)` to import a text file that contains the information to operate the system. You will read this text file line by line and call the corresponding methods based the line's content. A sample input text file (input.txt) you can use is included with the assignment. The console output for this sample input is given in the "output.txt" file. Note that you do not need to output to a file, you just need to print the results on console. The input text can include the following types of lines:

- **Create student line:** These lines should be used with the `addStudent` method; they are in the following form:
  - `Student:StudentName StudentSurname, Age, StudentID`
  - Sample Line: `Student:John Doe, 20, 9273636110`

- **Grade student line:** These lines include a Student ID and the grade information to be set for that student. In case the given Student ID does not belong to an existing student, this would cause getStudent method to trigger an exception. You need to handle this exception with a try-catch block so that even if certain lines cause an exception, the `processTextFile` method continues to operate on the remaining lines. Grade lines are in the following form:
  - `Grade:StudentID, ExamName, Weight, Grade`
  - Sample Line: `Grade:9273636110, Calculus I - Midterm II, 15, 79`

- **Print student line:** These lines should call printGradesOf method to print the grades of the student whose School ID is given. These lines are in the following form:
  - `GradesOf:StudentID`
  - Sample Line: `GradesOf:9273636110`

- **Print all line:** These lines either call `printStudentsByNameOrder` or `printStudents` or `printStudentGradeAverages` to print all the students with the desired order. This input can be one of the following:
  - `PrintByNameOrder`
  - `PrintByGradeAverages`
  - `PrintStudents`

In order to get the part you need out of the text line, you may use `split` and `trim` methods of the `String` class, as well as `Float.parseFloat` and `Integer.parseInt` methods for conversion.

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least have the following parts completed:

- Grade and Student classes should be completed.
- The methods addStudent, getStudent, and getStudentsByNameOrder in the School class should be completed.

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end.

Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in a 50% reduction of this assignment's final grade.