CodeCrunch

CS2030 Mock Practical Assessment #2: Test Analysis (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

CS2030 Practical Assessment #2

Problem Description

The instructor of a programming course decides to generate some reports from the recent practical test results of the students using two set sets of input:

- A master list of students in the course
- The marks for each student who attended the test

Each student is represented using a `Student` object, and each test mark using a `Mark` object. The corresponding classes have been implemented and made available for you. **You are not allowed to re-implement these classes.**

You are to complete the `Main` class with definitions of `static` methods for the different tasks. **No object instances of `Main` shall be created.**

This task is divided into several levels. Read through all the levels to see how the different levels are related. **Find a suitable level to start off.**

Just remember to:

- check for output format correctness using the `diff` utility (see specific level for usage details). Note that only **one** test case is provided for this.
- **as you will only be required to submit the `Main` class for every level, you only need to save a copy of the file as `Main1.java`, `Main2.java`, etc. using the `cp` command; make sure that the class remains as `Main`. No creation of sub-directories is necessary. Detailed instructions are given at each level below.**
- **only the classes `Student.class`, `Mark.class` and `Main.class` will be used for testing; ensure that your program compiles to these three classes and nothing else**
- only the `Main` class will be uploaded to CodeCrunch

- `checkstyle` and `javadoc` comments can be included after your programs have been up-loaded to CodeCrunch; you have till Friday midnight to do so.

Level 1

The `Main` class has been provided to you with a `static` instance of the `Scanner` class.

```
import java.util.Scanner;

class Main {
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {

    }
}
```

By making use of this `Scanner` object, read in the master list of students and output the list of students. Each line of input comprises a plab account, a student ID, and an integer tutorial group. You may assume that there is at least one student in the master list. In addition, input is terminated with `end`. **Do not create additional `Scanner` objects as it might cause problems in CodeCrunch.**

As an example, the following input

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
```

produces the output

```
plab1111,E0123456,9
plab2222,E0098765,1
plab0987,A0002468,9
```

The following is a sample run of the program. User input is underlined.

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab1111,E0123456,9
plab2222,E0098765,1
plab0987,A0002468,9
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test.in | diff - test1.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Main.java Main1.java
```

Level 2

As per the previous level, read in the master list of students, then output tutorial groups in ascending order.

As an example, the following input

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
```

produces the output

```
Groups(2):[1, 9]
plab1111,E0123456,9
plab2222,E0098765,1
plab0987,A0002468,9
```

The number within round brackets represent the number of groups, and output the list accordingly. In addition,

The following is a sample run of the program. User input is underlined.

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
Groups(2):[1, 9]
plab1111,E0123456,9
plab2222,E0098765,1
plab0987,A0002468,9
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test.in | diff - test2.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Main.java Main2.java
```

Level 3

In addition to the master list of students, you will now read the marks of the test. As an example, the input

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab2222 10
end
```

will produce the following output

```
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
```

Notice that each student record is now appended with a mark. If a student is absent, he will be given a `0` mark. You may assume that marks are non-negative integer values and that there is at least one mark entry, and every entry corresponds to a student in the master list.

The following is a sample run of the program. User input is underlined.

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test.in | diff - test3.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Main.java Main3.java
```

Level 4

Now include a list of absentees who did not attend the test. Absentees are output in the order of presentation in the master list.

The following is a sample run of the program. User input is underlined.

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
plab1111,E0123456,9
```

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab1111 0
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
None
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test.in | diff - test4.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Main.java Main4.java
```

Level 5

Now, include a mark frequency table.

Note that the frequency values span from the minimum mark to the maximum mark at one-mark intervals.

Absentees are not included in the frequency count, but those attended the test and got zero marks will be included.

The following is a sample run of the program. User input is underlined.

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
plab1111,E0123456,9
Mark frequency from 7 to 10
7 : 1
8 : 0
9 : 0
10 : 1
```

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab1111 0
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
None
Mark frequency from 0 to 10
0 : 1
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 1
8 : 0
9 : 0
10 : 1
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test.in | diff - test5.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Main.java Main5.java
```

Level 6

Other than the mark frequency of the entire student cohort, include the corresponding mark frequency tables for all tutorial groups in order of tutorial grouping.

Note that the frequency values still span from the minimum mark to the maximum mark at one-mark intervals.

Absentees are not included in the frequency count, but those attended the test and got zero marks will be included. As a consequence, tutorial groups with no students attending the tests will not have their frequency tables output.

The following is a sample run of the program. User input is underlined.

```
plab1111 E0123456 9
plab2222 E0098765 1
plab0987 A0002468 9
end
plab0987 7
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
plab1111,E0123456,9
Mark frequency from 7 to 10
7 : 1
8 : 0
9 : 0
10 : 1
Group #1...Mark frequency from 7 to 10
7 : 0
8 : 0
9 : 0
10 : 1
Group #9...Mark frequency from 7 to 10
7 : 1
8 : 0
9 : 0
10 : 0
```

```
plab1111 E0123456 9
plab2222 E0098765 1
```

```
plab0987 A0002468 9
end
plab0987 7
plab1111 0
plab2222 10
end
Groups(2):[1, 9]
plab1111,E0123456,9,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
None
Mark frequency from 0 to 10
0 : 1
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 1
8 : 0
9 : 0
10 : 1
Group #1...Mark frequency from 0 to 10
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 0
9 : 0
10 : 1
Group #9...Mark frequency from 0 to 10
0 : 1
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 1
8 : 0
9 : 0
10 : 0
```

```
plab1111 E0123456 8
plab2222 E0098765 1
plab0987 A0002468 9
end
```

```
plab0987 7
plab2222 10
end
Groups(3):[1, 8, 9]
plab1111,E0123456,8,0
plab2222,E0098765,1,10
plab0987,A0002468,9,7
List of absentees:
plab1111,E0123456,8
Mark frequency from 7 to 10
7 : 1
8 : 0
9 : 0
10 : 1
Group #1...Mark frequency from 7 to 10
7 : 0
8 : 0
9 : 0
10 : 1
Group #9...Mark frequency from 7 to 10
7 : 1
8 : 0
9 : 0
10 : 0
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test.in | diff - test6.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Main.java Main6.java
```