KBTU KAZAKH-BRITISH TECHNICAL UNIVERSITY

Object-Oriented Programming & Design
Laboratory Work 1

5 points

Instructor: Izbassar Assylzhan
Referenced from: Pakita

Deadline: 5^{th} week, on your practice time. No late submissions will be accepted!
Please, read carefully class requirements. Remember about naming conventions. Keep your classes separately.
Lab requires much more effort & time, rather than practice. Plan your time carefully.
`If (you can't recreate "your" code) point = 0;`

Problem 1

Write a class `Data` that computes information about a set of data values. It should have:

- 3 private fields: 2 of type double and 1 of type int (you need to guess what to store in these fields)
- Constructor that constructs an empty data set.
- Method that adds a value to a data set.
- Method that returns average of the added data or 0 if no data has been added.
- Method that returns the largest of the added data.

Note: You are not allowed to store ALL values ! Do not use arrays or something like this!
Then write a class `Analyzer` that uses `Data` class described above to compute the average and maximum of a set of input values.

- Use `Scanner` to take data from user
- The process of taking inputs from user must continue until the user types "Q" instead of a number.

For example:
Enter number (Q to quit): 10
Enter number (Q to quit): 0
Enter number (Q to quit): -1
Enter number (Q to quit): Q
Average = 3.0
Maximum = 10.0

Note: Please, handle incorrect inputs gracefully by implementing error validations (e.g., incorrect values, invalid characters and so on).

Problem 2

Write any class at your choice, and show appropriate usage of: enums, `static` & `final` modifiers, `this` keyword (2 usage), read-only fields, methods overloading, initialization block. All these usages must be related to **one** context, and have a meaning. Be prepared to give examples of static non-final fields in your context as well.

Problem 3

You need to write a `Temperature` class that has two fields: a temperature value (a double number) and a character for the scale, either 'C' for Celsius or 'F' for Fahrenheit. Make sure that these two fields can ONLY be accessed through the accessor methods outside of the class.

Constructors:
The class should have four constructors:

- one for each instance field (assume zero degree if no value is specified and Celsius if no scale is specified)
- one with two parameters for the two instance variables
- default constructor (set to zero degrees Celsius).

Methods:
The class should have 2 types of methods

(1) Two methods to return the temperature: one to return the degrees in Celsius, the other to return the degrees in Fahrenheit. Use the following formulas for conversion:

degreesC = 5(degreesF – 32) / 9
degreesF = (9(degreesC/5) + 32

(2) Three methods to set the fields: one to set the value, one to set the scale ('F' or 'C'), and one to set both.

(3) Method to return scale.

Problem 4

Create a `GradeBook` Application. Use `Student` class from practice 2 (import the corresponding package). `GradeBook` object stores the `Course` object and a list of Students. `Course` object has a name, description, number of credits, and prerequisite(s).

`GradeBook` should have:

- `displayMessage()` - simply displays a welcome message to an instructor.
- `displayGradeReport()` - the average of the grades, best grade/lowest grade and its holder, and the number of students who received each letter grade (statistics based on frequency). Separate implementation of grade report parts, e.g. you should have methods `outputBarChart()` , `determineClassAverage()` , etc.

`GradeBook` should have a sufficient number of constructors.

Think, how will you store grades? This will be your design decision.

In addition, create a class `GradeBookTest` - application class in which the main method will use class `GradeBook`. In `GradeBookTest`, create 1 `Course` and several students. Grades must be inputted via console.

Students, Course and GradeBook MUST have an appropriate implementation of `toString()` method.

Output example:

```
Welcome to the grade book for CS101 Object-oriented Programming
and Design!

Please, input grades for students:

Student  A:  87
Student  B:  68
Student  C:  94
Student  D: 100
Student  E:  83
Student  F:  78

Student  G:  85
Student  H:  91
Student  I:  76
Student J:  87

Class average is 84.90. Lowest grade is 68 (Student B, id: 4).
Highest grade is 100 (Student D, id:87).

Grades distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *
```

Note: Please do not copy or move the `Student` class from its initial location. You have to import it! If you haven't created the `Student` class during the 'Practice 2', please, do it on separate project & package, and import to `GradeBook`.

Problem 5

There is a very scary dragon living in Almaty city near KBTU. Everyday he needs to eat several young students for a launch. He usually kidnaps them one by one in the morning and eats at a launch time, having put them in a line at a cell in his prison. But sometimes he has problems with his launch, because students vanish! He still doesn't know that *pair of boy and a girl (B-G) can disappear if they stand together exactly in this order (B-G)* due to the magic of love. After that line becomes smaller. So, there is a possibility that no one will be left for a dragon launch!

You need to model a dragon launch. You need to have:

- Enumeration `Gender` that is used to distinguish boys / girls.
- Class `Person` containing an instance variable of type Gender, method `toString()` and any fields you want.
- Main class – `DragonLaunch`, with methods `kidnap(Person p)`, `willDragonEatOrNot()`.
- All kidnapped people need to be stored in a `Vector`.

For example, for a line BBGG, there will be no launch, because firstly middle pair will vanish, after that two corner persons will become BG pair, and vanish in the same way. However, a line GBGB leaves 2 persons for a launch.

Note: `willDragonEatOrNot()` should not involve any additional data structures, like `Stack`, `Vector` and so on, and should run in linear time. For example, when you have a loop and one of the operations inside is removing some elements from `Vector`, it is not a linear time.