Adventuine

Please also note the information at the bottom.

Adventuin party

We are already in the middle of the pre-Christmas period, and at this time the PGdP penguins are also preparing for Christmas. We therefore call such a penguin Adventuin.

The Adventuin

An Adventuin has a name (a character string), a height in centimeters (integer), a color (we can use the colors implemented in W07H01 for this; these are already in the project), a (Christmas) hat and the language he usually uses speaks. For hat and language we still need our own classes, we will use enums for this:

HatType exists No results

Create an Enum HatType with the values SANTA_CLAUS, REINDEER, FISHY_HAT and NO_HAT.
Language is available No results
Create an enum language with the values ENGLISH and GERMAN. The class should have a getLocalizedChristmasGreeting (String greeterName) method that can be called on a Language value and returns a Christmas greeting in the respective language (as a string):
At GERMAN: Merry Christmas wishes you and directly behind it greeterName with one! at the very end
At ENGLISH: the greeterName, followed directly by wishes you a Merry Christmas!
If you speak even more languages, you are welcome to add them (this is optional and will not be tested).
Adventuin is available with all methods No results
Then we can create the Adventuin class, which contains the named attributes. Set the public getters getName, getHeight, getColor, getHatType and getLanguage. You also need a public constructor that accepts name, size, color, hat type and language in exactly this order. For your own testing it is advisable to overwrite toString (). A validity check (e.g. to zero) is not tested, but you are welcome to add it.

The test_Adventuin_createExample test does the following:
new Adventuin ("AdvenTux", 123, new RgbColor (3, 2, 7, 0), HatType.FISHY_HAT, Language.ENGLISH);

The party

Now that the basics are done, we want to invite the Adventuins to a party and

do some interesting operations on them.
All methods of this subtask are created in AdventuinParty and should be static. Furthermore, each method receives a java.util.List with adventuins as a parameter, which contains all participating adventuins. This list is always free of duplicates and zero. Do not return null in the methods either directly or in any collections / maps.

Group participants by hats No results

Implement the groupByHatType method. Return a map <HatType, List <Adventuin>> in which the submitted participants are grouped by hats. The map should only contain the hat types that are worn by at least one participant. (This is also the case with the maps in the following subtasks)

Christmas greetings No results

Implement the printLocalizedChristmasGreetings method. Nothing is to be returned, but all Adventuins in the list are to be given their personal Christmas greetings on the console. The string that the respective Language object returns as an argument using getLocalizedChristmasGreeting and the name of the Adventuin should be used. The order should be determined by the size of the adventuins, the little ones can greet them first, so sort the adventuins in ascending order according to their size.
You can use System.out.println (String) to output text on the console.
Example:

Fröhliche Weihnachten wünscht dir Christian!

Adventuins with the longest names in their hat grouping No results

Implement the getAdventuinsWithLongestNamesByHatType method. Return a map <HatType, List <Adventuin>> in which the participants are grouped by hats whose name length is equal to the maximum name length (number of characters) in the group of Adventuins with the same hat type. Since several Adventuins can have names of the same length, this must also be a list.

Average color brightness by size No results

Implement the getAverageColorBrightnessByHeight method. Return a map <Integer, Double> that contains the average brightness of the colors of the Adventuine. Grouped according to size, rounded in 10 cm steps, that means, for example, from 95 to 104, all are grouped under integer 100, and 105 to 114 under 110, etc. For the average brightness, the colors must be transformed to their brightness we first convert them to 8-bit colors (value range 0 to 255) using the toRgbColor8Bit method and then use the formula $(0.2126*R+0.7152*G+0.0722*B)/255$. R, G and B are red, green and blue, the class RgbColor has suitable getters.

Advent wreath circle dance No results

At the end of the party, our Adventuins want to perform a dance on an advent wreath in a large circle. It is particularly important to them that the size difference to the neighbors is as big as possible under Adventuins with the same hat. In order to be able to measure this, the Adventuins now want to calculate the absolute difference between the difference averages "larger than predecessor" and "smaller than predecessor" and thus receive a measurement for the mixing of the sizes in the dance circle. However, the Adventuins still want to keep the freedom to dance side by side in groups of the same size without worsening the statistics.
Implement the getDiffOfAvgHeightDiffsToPredecessorByHatType method.
Return a map <HatType, Double> that:

Grouped by HatType and then within each group ...
From the Adventuin size differences to the respective predecessor (difference successor-predecessor)
... grouped by type of difference result (sign: -1, 0 or 1)
... the absolute difference between
… The average of all negative height differences (or 0.0 if not available) and
… The average of all positive height differences (or 0.0 if not available)
calculated. It should be noted that the adventuins are actually in a circle, and the predecessor (of course with the same hat type) of the first adventuin of a hat type group is the last in this group. If a penguin is alone with its hat type, it is its own predecessor, but this is not a problem for the calculation (this then falls into the difference category 0), and the absolute difference between the difference averages is therefore also 0 in its group.

For example, if we have two Adventuins with hat type SANTA_CLAUS, which are sizes 100 and 120, the first difference is 100-120 = -20, and the second difference 120-100 = 20; the absolute difference between them is 40.
We have three Adventuins with hat type SANTA_CLAUS, which have the sizes 100 and 120 and 100, so the first difference is 100-100 = 0, the second difference 120-100 = 20 and the third difference 100-120 = -20; the absolute difference between them is still 40. Here the case that two penguins of exactly the same size follow each other in the hat group, the statistics should not deteriorate.
But we have four Adventuins with hat type SANTA_CLAUS, which have the sizes 100 and 120, 100 and 110, so the first difference is 100-110 = -10, the second difference 120-100 = 20, the third difference 100-120 = 20 and the fourth difference is 110-100 = 10. The averages are -15 for the negative and 15 for the positive differences, so the absolute difference is only 30.

Bonus: Kranzuin (5 bonus points)

The Kanzuin is a very special penguin, whose name is naturally derived from the Advent wreath. Especially at Christmas time, he is increasingly asked questions about how many exercise sheets there will be at PGdP. However, since their number is not known before Christmas, the Kranzuin answers inaccurately, so that it does not promise too much or commit itself too much in advance.

The process is as follows:

First the questions about the number of sheets are collected. (Is not part of the task)
Based on their content, the Kranzuin decides how to answer the questions (he answers all at once).
In response, he returns a number that limits the number of PGdP task sheets.
His answer to the questions is based on these criteria:

Questions are only relevant if they contain sheets, tasks or number in the text.
All relevant questions are filtered by contained numbers (they are only positive integers that fit int). You can assume that there is at least one space between the numbers in a question. If there are several numbers in a question, only the mean between the minimum and maximum number is taken (rounding off at .5).
If one of the contained numbers is less than 15, 1783 is returned. (So you're always on the safe side)
If all numbers are greater than or equal to 15, the smallest of these numbers is returned. (anything over 14 is safe)
If there are no numbers in the relevant questions, we return a random number greater than or equal to 15. It should be the first random number that is greater than or equal to 15 and that is divisible by the length of at least one relevant question string. For this you can get an IntStream from ThreadLocalRandom.current (). Ints (int, int) with endless random numbers in the specified range.
If there is not a single relevant question, 15 plus the maximum of the lengths of all question strings is returned.
If there are no questions in the list, the Kranzuin is sad and returns Integer.MAX_VALUE.
A single example of such a question (the "" are not part of the question, is expected in 1783):

"Hallo, letztes Jahr gab es insgesamt 14 Blätter, wird es dieses Jahr vielleicht nur 13 Blätter geben oder sind es wieder 14?"

A test call that tests with a list that contains only this one question No results

Create the Kranzuin class, which is public and final, but whose only constructor is private. It is therefore no longer possible to create objects of the class from outside. Create a single static Kranzuin object in the class that is named JULIAN. It should be public static final.

Create the non-static answer answer (List <String>) method that returns an int and implements the behavior described above. As usual, keep the value range of int in mind.

You can assume that the questions only consist of the characters a to z, A to Z, 0 to 9, äöüÄÖÜß,.?! and normal spaces. You can use any of the methods of the Character, String and Integer classes (e.g. Integer.parseInt) for the task. Links to all kinds of methods: character, string, integer. There are many completely different solutions for extracting the numbers, here you can be creative. Unless the list <String> of questions is non-zero, their implementation should not throw any errors. Questions cannot be null either, but they can be empty.

Performance: Please pay attention to an acceptable performance during the task, the tests should not run endlessly (no tutor wants to test that either). Therefore, their implementation should pass the following performance tests. The tests themselves ignore any errors, functionality is not checked. The time limit for the performance tests is 20 seconds.

Acceptable performance level No results

General information about the tasks:
Do not use any loops in these tasks (for, while, ...).
Recursion is also problematic, tests with larger amounts of data are not excluded.
Solve the task only by using streams.
Mark all attributes with final.
Mark all classes newly created in this task with final.
Your methods should also deal with empty lists / streams / other collections.
zero need not be treated separately.
Hide implementation details of the classes.
Therefore mark all attributes with private.
If you write auxiliary methods, also mark them with private or package private.
Be careful when implementing, the public test cases only test for the existence of classes, methods, etc. and whether something is returned. What exactly is tested is included in the test case or Artemis task.
Test your implementation thoroughly yourself.
Beyond that, pay attention to proper and formatted source code, see Style Guide (P-Task, Moodle).
If you change anything in pgdp.color, you should be aware that tests may not work and you may lose many points.