Comp 524 - Assignment 1:

Social Distancing: Java, MVC, and Weka

**Date Assigned: August 30, 2020**

**Early Completion Date: Friday, September 11, 2020 (+5% extra credit)**

**Completion Date: Tuesday, September 15, 2020**

You will implement three versions of a social distancing application in Java using (a) static Java methods, (b) the MVC pattern, and (c) the Weka decision tree classifier.

Parts of the assignment will be added as we cover relevant aspects in class and our tests get written. The first version can be written with just a knowledge of Java.

You will be graded for both correctness and style. There will be extra credit for meeting both kinds of requirements.

The general style guide for all Java programs is [here](https://docs.google.com/document/d/17SQfQ_bVNvBPIfv9GiOdNHttH3h18JuG/edit?pli=1). It has some formatting rules that we will remove and do not hurt to obey. These rules do not contradict what you have learned from previous classes on how to write well documented programs. The modified general style guide should be complete by Monday morning, September 7th.

Additional, assignment specific style rules are given in this document.

## Social Distancing

Our social distancing application assumes some subject has had a safe encounter with a guest present in the same room. It considers three input integer parameters:

1. Distance to the guest.
2. Duration the guest was in the room.
3. The exhalation level of the guest, which would vary based on whether, for instance, the guest was quiet, talking or coughing/sneezing.

For each parameter, three points are defined: small, medium, and large. The following table gives the current values of these points.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Small | Medium | Large |
| Distance | 6 | 13 | 27 |
| Duration | 15 | 30 | 120 |
| Exhalation Level | 10 | 30 | 50 |

**Table 1 Small, Medium and Large Values**

**Assume** the following table contains the safe combinations of these points.

|  |  |  |
| --- | --- | --- |
| **Distance** | **Duration** | **Exhalation Level** |
| Medium | Medium | Medium |
| Small | Medium | Small |
| Large | Medium | Large |
| Medium | Small | Large |
| Medium | Large | Small |
| Large | Large | Medium |
| Small | Small | Medium |

**Table 2 Assumed Given Safe Combinations**

Some of these assumed combinations are based on results from web searches that indicate that < 15 minute interaction is always safe, a person sitting two rows ahead or behind a person in an airplane (which is about 6 feet) was infected, and a person’s breath travels 13 feet when talking and 27 feet when sneezing/coughing.

We will call these **assumed** combinations as **data given** to the application.

## isGivenSafe() static function

Write a **static** function called **isGivenSafe** that returns a **boolean**, and takes three **int** parameters representing a distance, duration, and exhalation level. It thus, has the signature:

isGivenSafe: int\*int\*int🡪boolean

In general, the syntax of a signature is:

<function name>: <parameter1 type>\*<parameter2 type>\*<parameter type>🡪<return type>

If the combination of the method parameters is safe, based on the given data, the function returns true. Otherwise, it returns false. In other words, only if the combination is an *exact match* in Table 2, it returns true. The next function considers ranges.

Let us call the class in which this method is implemented as the **social distancing utility class**. All static methods required in this assignment should be in this class.

In general, a function or procedure you are asked to implement can, and probably should, call other methods in the same or different class.

## Three-Parameter isInterpolatedSafe() static function

Write another static function in the social distancing utility class with the signature:

isInterpolatedSafe: int\*int\*int🡪boolean

Again, the three parameters represent distance, duration, and exhalation level. The function interpolates each of the parameters to a value in Table 2, and determines if the interpolated values are safe based on whether they occur in Table 2.

The interpolation is conservative. If higher values of the parameter are safer, then it is interpolated low . If lower values of the parameter are safer, then it is interpolated high.

Consider a value V and a sorted sequence of values I1, I2, … IN to which it must be interpolated.

Then its high interpolation, H, to this sequence of values is defined as follows:

H = Maximum Integer If V > IN

H = I1, if V <= I1.

H = IM, if V > IM-1, V <= IM

Its low interpolation, L, to this sequence of values is defined as follows:

L = IN, if V >= IN.

L = IM-1, if V >= IM-1, V < IM

L = 0 If V < I1

Thus, low interpolation to a sequence of values is either an element of the sequence or zero .

Similarly, high interpolation to a sequence of values is either an element of the sequence or max integer..

Again, V is the value you have, and I1, I2, … IN are the ordered values to which you must interpolate.

As higher distances are safer, the distance parameter is interpolated low.

This means the distance parameter is interpolated to small, medium, large distances or the value 0.

As lower duration and exhalation levels are safer, duration and exhalation levels are interpolated high.

Thus, the duration parameter is interpolated to small, medium large durations or the maximum integer.

Similarly, the exhalation level parameter is interpolated to small, medium large exhalation levels or the maximum integer.

In Java, the maximum integer is Integer.MAX\_VALUE.

## Two-Parameter isInterpolatedSafe() static function

Write a static function in the social distancing utility class with the signature:

isInterpolatedSafe: int\*int🡪boolean

The two parameters represent distance and duration. It fixes the value of exhalation leve the medium exhalation. It determines if the combination of interpolated distance, interpolated duration, and medium exhalation level is safe.

## One-Parameter isInterpolatedSafe() static function

Write a static function in the social distancing utility class with the signature:

isInterpolatedSafe: int🡪boolean

The single parameter represents a distance. It fixes the value of duration and exhalation level to medium duration and medium exhalation level respectively. It determines if the combination of interpolated distance, medium duration, and medium exhalation level is safe.

## printGeneratedCombination() static function

This procedure has the signature:

printGeneratedCombination:🡪 void.

It uses Math.random() function to generate a distance, duration, and exhalation level combination.

It determines whether the interpolation of the values in this combination is safe.

It prints the (distance, duration, exhalation level, and safety) tuple, using a comma to separate the elements of the 4-tuple.

The next method will call this method repeatedly to generate these 4-tuples. These tuples can be examples for (a) testing your solutions, (b) demonstrating to us that your code works, and (c) data you give in the last part of this assignment to the Weka classifier. To help train it, it would be useful to choose values near the edge cases in Table 2. You are free to choose the range of values for each parameter from which you pick randomly. For training the classifier, it might be useful to make the maximum value in this range to **say twice the maximum value** for the parameter in the table. Thus, the maximum value for distance **might** be 54. Play around to get good results in version 3 of this program, it is not clear twice will give you the desired values. **For this version, it does not matter what the range is.**

## printGivenAndGeneratedCombinations() static function

This procedure has the signature:

printGivenAndGeneratedCombinations:🡪 void.

It prints the following header:

Distance,Duration,Exhalation,IsSafe

For each 3-tuple combination in Table 2, it adds the Boolean true to create a 4-tuple combination and prints the 4-tuple, again using a comma to separate the elements of the tuple.

It prints a separator line with one or more hyphens.

It then calls printGeneratedCombinations() ten times.

Thus, its output should look like this:

Distance,Duration,Exhalation,IsSafe

13,30,30,true

6,30,10,true

27,30,50,true

13,15,50,true

13,120,10,true

27,120,30,true

6,15,30,true

----------------

5,214,77,false

48,24,85,false

35,88,49,false

53,95,97,false

31,1,74,false

24,138,6,false

19,44,24,false

20,94,4,true

4,64,78,false

8,35,41,false

**Please check all outputs with all constraints imposed.**

## generateSafeDistancesAndDurations() static function

This procedure has the signature:

generateSafeDistancesAndDutrations:int🡪 List<Integer[]>.

The parameter represents an exhalation level.

The procedure computes a (possibly empty) list of given safe distance and duration pairs that are associated with an interpolation of the exhalation level in Table 2.

Each pair is returned by a two-element array whose first element is the distance and second element is the duration.

## printSafeDistancesAndDurations() static function

Signature:

printSafeDistancesAndDutrations:int🡪 void

The parameter is an exhalation-level. It uses the method above to determine the list of safe distances and durations for the passed exhalation-level and prints the passed exhalation level together with the returned list using the format below:

<exhalation level>, [{<destination,duration1}, … {<destination>, <duration>}]

Thus, the following three calls:

printGivenSafeDistancesAndDuration(9);

printGivenSafeDistancesAndDuration(11);

printGivenSafeDistancesAndDuration(51);

output the following three lines:

9,[{6,30}{13,120}]

11,[{13,30}{27,120}{6,15}]

51,[]

## Style Constraints: Regular and Extra Credit

Several of the general style constraints are applicable there such as use of mnemonic names, named constants, final parameters, block count and levels.

In addition, in the implementation of the required static methods, you must pay particular emphasis on the related metrics of modularity, extensibility and reusability. Doing so may earn you extra credit based on whether our automatic and manual grading objectively recognizes optional decisions that improve the solution in these three dimensions.

You may also earn extra credit points for elegance of solution measured by the number of relational and boolean operators, and conditional and looping constructs your solution uses. Less use of them should also lead to improvement in other metrics.

As mentioned above, to improve these metrics, you are allowed to and, in fact, encouraged to write additional optional methods called by the methods required in the social distancing utility class. These methods can be in the same or different classes and packages. The style credit you get will not consider the class or package of these additional methods.

The description of the problem is modular to encourage a modular solution.

If an optional or required method you write is related to some requirement sentences in this document, then put those sentences as a Javadoc comment before the header of the method. Such a comment has the format:

/\*\* comment \*/

For instance, the isGivenSafe() method should have the comment:

/\*\*

If the combination of the method parameters is safe, based on the given data, the function returns true. Otherwise, it returns false.

\*/

You do **not** have to but can put obvious text such as the following describing method headers:

/\*\*

Write a **static** function called **isGivenSafe** that returns a **boolean**, and takes three **int** parameters representing a distance, duration, and exhalation level. It thus, has the signature:

isGivenSafe: int\*int\*int🡪boolean

\*/

We will try to match the JavaDoc of methods you write to various parts of the requirements. The more methods are matched to the requirements, the more modular your solution is.

The requirements of some of the more complex functions such as isInterpolatedSafe() have multiple sentences describing different sub-requirements. It is possible to implement a monolithic method that supports all of these sub-requirements or write different methods for different sub-requirements. The latter approach will result in more matched methods and also, of course, a more modular solution.