Comp 524 - Assignment 1:
Social Distancing: Java Static Methods

Date Assigned: August 28, 2021

Early Completion Date: Friday, September 3, 2021, 11:55pm (+5% extra credit)

Completion Date; Tuesday, September 7, 2021, 11:55pm

First Late Day: Tuesday, September 14, 2021, 11:55pm (-10%)

Second Late Day: Tuesday, September 21, 2021, 11:55pm (-25%)

You will implement a social distancing application in Java, involving the use of static Java methods.

The application involves writing several similar functions and procedures (methods that return and do not return values, respectively). It may seem a bit tedious to do so. There are two reasons for this apparent tedium:

1. To give you an opportunity to isolate similarities into reusable code.
2. To show the relative strengths of the different languages in which you will implement these functions and procedures. For instance, the printing functionality will be automatically performed by the Prolog interpreter. Similarly, functions taking varying number of parameters will be elegantly implemented using SML's curry operator, Thus, the whole point of introducing tedium here is to show how other languages remove it.

You will need an updatedComp524All.jar file from the Downloads folder. You will need a different checkstyle file though and need to call a different local checks suite based on the fact that the assignment number is 1 rather than 01. Thus, the suite is: gradingTools.comp524f21.assignment1.F21Assignment1Suite;

Recall that all checkstyle files are in: F21/CheckStyleFiles .

When an assignment is given, the local checks, checkstyle file, and gradescope and Sakai submissions may not be ready or finalized. They will be finalized by at least one day before the early submission date, otherwise, this date will be postponed. No date will be moved forward, but some dates may be moved back. We reserve the right to add checks to Gradescope and Sakai that were not part of local checks.

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 :Style Guide. It has some formatting rules that we will not check but do not hurt to obey. These rules do not contradict what you have learned from previous classes on how to write well documented programs.

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

An important rule for extra credit in this assignment is: If you have reused a method whose functionality is defined by some set of sentences in this document, then include these sentences as the JavaDoc for that method.

Social Distancing Table

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>: <parameter[1] type>*<parameter[2] 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. The class should be **public** and the required methods in it should all also be **public**; otherwise our grader cannot access them. The name of the class is up to you and specified in the registry.

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

Yes, this function, on its own, is extremely conservative. The isInterplatedSafe() and isDerivedSafe() functions will allow more liberal combinations of distance, duration, and exhalation level, that build on this rigid function.

Three-Parameter isInterpolatedSafe() static function

The various versions of isInterpolatedSafe() are being given to motivate SML curry later.

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.

Thus, the Duration values of 31 and 121 are interpolated high as 120 and Maximum Integer, respectively.

Conversely, the Distance values of 5 and 12 are interpolated low as 0 and 6, respectively.

More formally:

Consider a value V and a sorted sequence of values I^1, I^2, … I^N to which it must be interpolated.

Then its high interpolation, H, to this sequence of values is defined as follows:
H = Maximum Integer If $V > I^N$
H = I^1, if $V <= I^{1.}$
H = $I^{M,}$ if $V > I^{M-1}$, $V <= I^M$ for $1 < M <= N$

Its low interpolation, L, to this sequence of values is defined as follows:
L = I^N, if $V >= I^{N.}$
L = $I^{M-1,}$ if $V >= I^{M-1}$, $V < I^M$ for $1 < M <= N$
L = 0 If $V < I^1$

If this notation is obscure to you, here is a computing-based explanation at TA provided last year: M is an arbitrary index for the sequence I such that $1 < M <= N = |I|$. So I^M is like I[M] if I were an array, and $I^{(M-1)}$ is like I[M-1] if I were an array. (Since M == 1 implies M-1 == 0 is out of bounds of the sequence, we restrict 1 < M.)

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 I^1, I^2, … I^N are the ordered values to which you must interpolate, which will be low, medium and high values for the three safety parameters.

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 level to 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.

Three-Parameter isDerivedSafe() static function

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

isDerivedSafe: int*int*int-->boolean

Again, the three parameters represent distance, duration, and exhalation level.

It returns true if the combination of these three parameters is safer than **at least one** of the combinations in the table.

A combination (distance^1, duration^1, exhalation_level^1) is safer than combination (distance^2, duration^2, exhalation_level^2) if:

distance^1 >= distance^2 and duration^1 <= duration^2 and exhalation_level^1 <= exhalation_level^2.

Thus, the combination 28, 14, 29 is derived safe as 6, 15, 30 (Small, Small, Medium) is safe and 28 > 6, 14 < 15, 29 < 30. If we were to interopolate our three values, we would get 27, 15, 30 (Large, Small, Medium) which is not explicitly in the table.

Why have interpolation when we have derivation? We will see that Prolog cannot backtrack with derivation but it can do so with interpolation. Thus, this function highlights a limitation of Prolog.

printGeneratedCombinationDerivedSafety() static function

The various versions of the print are here to motivate later Prolog's backtracking and also to help generate data for the Weka assignment.

This procedure has the signature:
printGeneratedCombinationDerivedSafety: --> void.

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

It determines whether this combination is safe based on the isDerivedSafe() function described above.

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

Like any method, this method can invoke helper methods implementing its functionality.

The next method will call the code implementing this method's functionality 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 next 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 in the next assignment, it might be useful to make the maximum value in this range to, **say, 1.2 times the maximum value** for the parameter in the table. Play around to get good results in the next assignment. **For this assignment, the range does not matter.**

printGivenAndGeneratedCombinationsDerivedSafety() static function

This procedure has the signature:
printGivenAndGeneratedCombinationsDerivedSafety: --> 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 the code implementing printGeneratedCombinationDerivedSafety() 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
----------------
7,15,9,true
11,11,2,true
12,14,20,true
10,8,9,true
3,2,23,false
11,28,16,false
0,13,11,false
12,5,3,true
1,20,10,false
8,21,28,false
```

Please check all outputs and examples with all constraints imposed – there are likely to be mistakes.

generateSafeDistancesAndDurations() static function

This procedure has the signature:
generateSafeDistancesAndDurations: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.

Thus, for exhalation level, 11, it should return [{13,30}{27,120}{6,15}]
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:
printSafeDistancesAndDurations: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 the 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.

To get extra credit, if an optional method you write is related to some requirement sentences in this document, then, as in A1, put *all* of the relevant sentences, verbatim, as a Javadoc comment before the header of the method. Such a comment has the format:

/** comment */

For instance, if isGivenSafe() method has the JavaDoc comment:
/**
If the combination of the method parameters is safe, based on the given data, the function returns true. Otherwise, it returns false.
*/
and if it was optional, which it is not, you would get extra credit points. Put this comment in front of your implementation to see if our checks find it (and give you 0 points for it).

Our checks remove punctuation, line break and * occurencesbefore checking.

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
*/

Thus, for extra credit, we will try to match the JavaDoc of the **non-required** methods you write to various parts of the requirements. The more such 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.

Utility Class Tester

Create also a main class, which we will refer to as the **social distance utility tester main**, that calls the following utility class method at the start:

```
printGivenAndGeneratedCombinationsDerivedSafety();
```

In addition, it calls printSafeDistancesAndDurations three times, with medium exhalation level, medium exhalation level -1 and medium exhalation level + 2, respectively. This class should be separate from the social distance utility class.

Class Registry

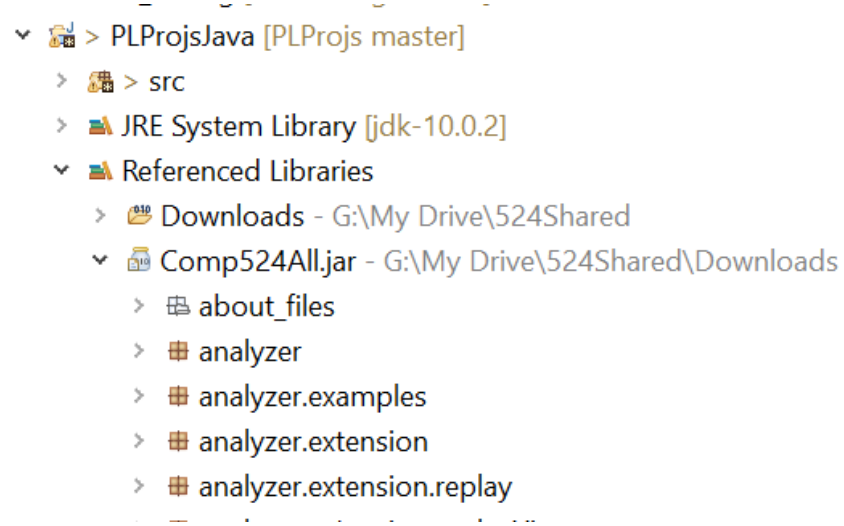
Implement the interface
gradingTools.comp524f21.assignment1.F21A1SocialDistanceClassRegistry.

Updating the Tools

Every Java assignment will involve a new checkstyle file and a new version of the jar file. Sometimes, the new checkstyle file will require an update also to the plugin. Each time such an update occurs, the following Piazza message will be updated:
https://piazza.com/class/ksf2o98md07hl

The jat file update can be tricky. It is sometimes not sufficient to simply replace the previous jar with a new one.

As the figure below shows, each project has a Referenced Libraries subfolder. Select it and use the right click menu to refresh it.



As the figure also shows, you can use a tree view to look at all the packages and classes in a referenced jar. If Java tells you some class is not accessible, check it is in the version of the jar you have.

Screenshots

Create screenshots (not text files) of running the (a) utility class tester, and (b) the test results of localchecks, if you have run them. You should submit two screenshots, thus,

Gradescope does not seem to handle image files with names whose suffix is capitals (that is, .PNG). So if the tool you use creates such a name, convert the capitals to lower case (e.g. .png).

Submission

You will submit again **both to Sakai and Gradescope**. The directory structure of the submission is the same as in A01.

Issues Faced by Students and Resolutions