Programming Assignment 4

`DetailedSpeedSetter`

Overview

As you know, *DukeDash* has been selling a large number of `TripSetter` and `DetailedTripSetter` applications. What may surprise you, and has surprised *DukeDash* as well, is that they have also sold a large number of the original `SpeedSetter` applications. Though it has a much more limited feature set, the sales and marketing people at *DukeDash* think it is selling because it is relatively inexpensive.

Since they want to keep this portion of the market happy, they have decided to enhance its capabilities somewhat and sell the new version as the `DetailedSpeedSetter`.

As with `SpeedSetter`, `DetailedSpeedSetter` will be given the current speed in mi/hr and convert it to km/hr. However, it will also (optionally) be given the speed limit in km/hr and will indicate whether the user is speeding.

Note that, for sales and marketing reasons, `DetailedSpeedSetter` enhances `SpeedSetter`, not `TripSetter` or `DetailedTripSetter`. That is, it does not include the functionality that was added when you created `TripSetter` and/or `DetailedTripSetter`.

Specifications

The product must comply with the following specifications:

1. The main class must be named `DetailedSpeedSetter`.
2. In addition to the `main()` method, the main class must have the following method:

   ```
   public static boolean isSpeeding(double speed,
                                    double limit)
   ```

   which returns `true` if and only if the `speed` is strictly greater than the `limit`.

3. When executed, command-line argument 0 must contain a `String` representation of the current real-valued speed (in mi/hr). Argument 1 is optional. If present, it contains the speed limit (in km/hr because it is being provided by the local infrastructure).
4. The product must have a class named `Converter` that performs all of the necessary unit conversions. In particular, this class must have the following methods:

   ```
   public static double mphToKPH(double mph)
   ```

   It may have other methods as well. **Note: You should already have such a class.**
5. The product must display the current speed in km/hr.
6. If a speed limit is provided and is greater than or equal to the smallest allowable speed limit of 20 km/hr, the provided speed limit must be used as the current speed limit.
7. If a speed limit is provided and it is less than the smallest allowable speed limit, the smallest allowable speed limit (i.e., 20 km/hr) must be used as the current speed limit.
8. If no speed limit is provided, the default speed limit of 40 km/hr must be used as the current speed limit.
9. The `Dashboard` must be informed about whether or not the current speed exceeds the current speed limit. This can be accomplished by calling its `setSpeeding()` method (see below).

Existing Components

`Dashboard` Class

As you know from your earlier work on `SpeedSetter`, the `Dashboard` class contains the graphical user interface for an in-vehicle electronic dashboard. In addition to the methods you have used in the past, it has the following method:

```
public static void setSpeeding(boolean isSpeeding)
```

Will cause the speeding indicator (typically a change in color) to be displayed if the parameter `isSpeeding` contains the value `true`.

`Array` Class

Java has a “built in” `Array` class with a `getLength()` method that is passed an array and returns an `int` containing the number of elements in that array. To use this method, you must include the line:

```
import java.lang.reflect.Array;
```

at the top of any file that uses it. For example, you might do something like the following:

```
import java.lang.reflect.Array;

public class DetailedSpeedSetter {

   public static void main(String[] args) {

      int numberOfArguments;

      numberOfArguments = Array.getLength(args);

      // The rest of the main method
   }
}
```

Recommended Process

1. Read and understand the entire assignment.
2. Create a directory/folder (e.g., named `pa4`) that will hold all of the files for this assignment.
3. Copy `Text.class` and `Dashboard.class` into the directory you just created.
4. Copy an appropriate version of `Converter.java` class into the directory you just created. You may use either one of your implementations or one of the solutions that were provided to you.
5. Implement the `DetailedSpeedSetter` class.
6. Test the `DetailedSpeedSetter` class. Remember to include tests in which there is and isn't a speed limit, in which the given speed limit is less than the smallest allowable speed limit, and in which the current speed does and doesn't exceed the current speed limit.
7. Debug the `DetailedSpeedSetter` class as needed.
8. Submit your implementation of `Converter.java` and `DetailedSpeedSetter.java` in a file named `pa4.zip` using Autolab. Do not include any other files in the `.zip` file.

Reminders

1. `DetailedSpeedSetter` enhances `SpeedSetter`, not `TripSetter` or `DetailedTripSetter`. That is, `DetailedSpeedSetter` does not include any of the functionality that you added when creating `TripSetter` and/or `DetailedTripSetter`.
2. Remember that the current speed in `args[0]` is in mi/hr (because it comes from the vehicle that was brought over from the United States) and the speed limit in `args[1]` is in km/hr because it comes from the "local" road system (either from a map or from a road sign).

Grading

Your code will first be graded by Autolab and then by the Professor. The grade you receive from Autolab is the maximum grade that you can receive on the assignment.

Autolab Grading

Your code must compile (in Autolab, this will be indicated in the section on “Does your code compile?”) and all class names and method signatures comply with the specifications (in Autolab, this will be indicated in the section on “Do your class names, method signatures, etc. comply with the specifications?”) for you to receive any points on this assignment.

Autolab will then grade your submission as follows:

Conformance to the Course Style Guide:	**20 points** (Partial Credit Possible)
Correctness:	**80 points** (Partial Credit Possible)

Manual Grading

After the due date, the Professor may manually review your code. At this time, points may be deducted for inelegant code, inappropriate variable names, bad comments, etc.