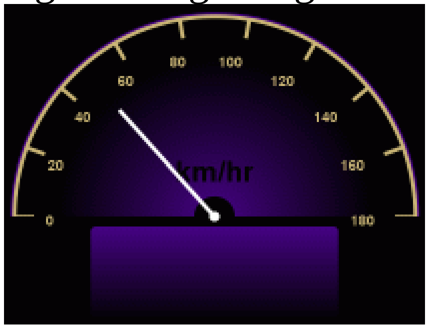
Programming Assignment 1



`SpeedSetter`

Overview

A former JMU Computer Science professor has started a software company called *DukeDash* that is creating electronic dashboards of various kinds (for both vehicles and other systems).

She has asked you to create a prototype of a product called `SpeedSetter`. The market for `SpeedSetter` is U.S. citizens who have moved to a country that uses the international system of units (i.e., SI units) and have brought their car/motorcycle with them. The difficulty that such people have is that the car measures and reports the speed in miles per hour (mi/hr) but the speed limit signs on the side of the road are in kilometers per hour (km/hr). `SpeedSetter` is an electronic dashboard that replaces the built-in dashboard that came with the car/motorcycle. It is given the current speed in mi/hr, converts it to km/hr, and displays the speed in km/hr.

Specifications

The product must comply with the following specifications:

1. The main class must be named `SpeedSetter`.
2. The main class must have a double-valued “class constant” named `MILES_PER_KILOMETER` that is assigned the vale 0.621371.
3. When executed, command-line argument 0 must contain a `String` representation of a speed (in mi/hr).
4. The product must convert the given speed from mi/hr to km/hr using the constant `MILES_PER_KILOMETER`.
5. The product must display the converted speed (in km/hr) on a `Dashboard` (see below).

Existing Components

Other programmers at *DukeDask* have created two classes that you must use.

`Dashboard`

The `Dashboard` class contains the graphical user interface for an in-vehicle electronic dashboard. It has the following methods:

```
public static void setSpeed(double kph)
```
Displays the speed (in km/hr) on the speedometer.

`Text`

The `Text` class contains methods that can be used to convert `String` representations of numerical values to the corresponding numerical value. It has the following methods:

```
public static double toNonnegativeDouble(String s)
```
Converts the `String` representation of a non-negative `double` contained in the parameter named `s` to a non-negative `double` value and returns it. In the event that `s` does not represent a non-negative `double` it will return the value `-1.0` instead.

```
public static double toNonnegativeInt(String s)
```
Converts the `String` representation of a non-negative `int` contained in the parameter named `s` to a non-negative `int` value and returns it. In the event that `s` does not represent a non-negative `int` it will return the value `-1` instead.

You may also use the `JMUConsole` class from lecture/lab to help with debugging.

Recommended Process

1. Read and understand the entire assignment.
2. By hand, convert the following values from mi/hr to km/hr:

   0.0,   7.0,   10.0,   14.8,   55.0,   75.0,   100.0
3. Create a directory/folder (e.g., named `pa1`) that will hold all of the files for this assignment.
4. Download the .zip file containing the `Dashboard`, `Text`, and `JMUConsole` classes and **unzip it into the directory you created for this assignment**.
5. Implement the `SpeedSetter` class.
6. Test your implementation using all of the values from step 2.
7. Debug your implementation, as necessary.

8. Submit your implementation of `SpeedSetter.java` in a file named `pa1.zip` using Autolab. Do not include any other files (for example, the `Text` class or the `Dashboard` class) in the `.zip` file.

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:	**50 points** (Partial Credit Possible)
Correctness:	**50 points** (All or Nothing)

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.