Advanced Programming

Lab Sheet 8 – Test-Driven Development

Introduction

Test-Driven Development (TDD) is a practice by which unit tests are written *before* the software being created. Proponents assert that this allows a greater number of errors to be caught during development, reducing the number of defects present in the version of the software that ships. In this lab exercise, you will gain some practical experience of TDD by developing some classes for which I have already developed a suite of test cases.

The exercises involve writing code that calculates estimates based on the battery status of an Electric Vehicle (EV). Your code will handle two types of estimate: range estimates and charging time estimates. Each of the calculations will depend on a number of variables, which your code will need to validate the values of when set.

Importing the Code

A skeleton of the classes you are going to develop, along with the suite of JUnit tests that your code needs to pass, is available on Moodle as an eclipse project, exported as a zip file. You can import it into your eclipse workspace by selecting File → Import, and then selecting "Existing Projects Into Workspace". You will see a dialog similar to that depicted in Fig. 1, below.

On this dialog, you should make sure "Select archive file" is selected, then browse to the zip you downloaded from Moodle. Make sure that the "TDDLab" project is ticked, before clicking the "Finish" button at the bottom of the dialog. You will see the project in the left-hand panel in eclipse, along with your previous projects.

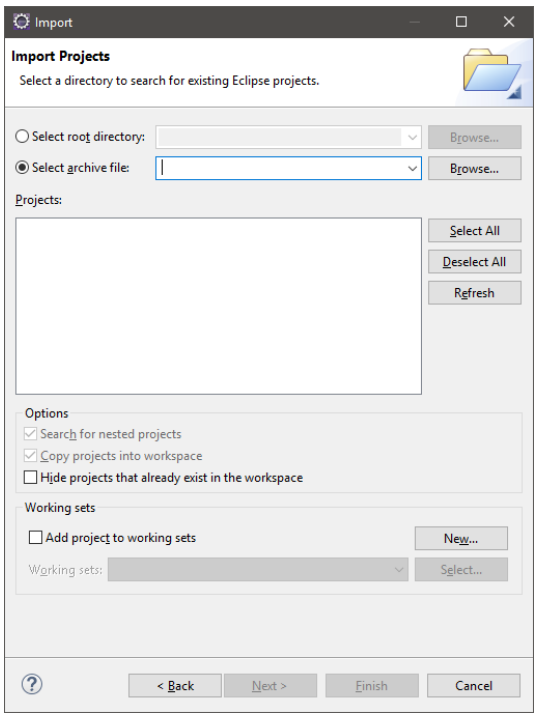

Figure 1: Eclipse Import Projects Dialog

The TDDLab project contains two packages, one for the code and one for the tests. To check how you're doing, you can open the EVBatteryManagerTest class, and run it: eclipse should run the JUnit tests within and tell you how many are passing and how many are failing.

Task 1: Parameters & Validation

The EVBatteryManager class has a number of methods that other systems (or perhaps even users) would use to supply us with parameters to be used in the calculation of our estimates. Unfortunately, other programmers, and particularly users, are not to be trusted! They might give us a percentage as 0.5 instead of 50, or might give us a date with the day & month the wrong way round! Thus, a significant portion of code and tests in industry is dedicated to ensuring that the data supplied is valid.

You will need to work on the EVBatteryManager class to complete all of the various getters and setters. The method signatures are already in place, you simply need to assign the right values to the right variables. For validation, make sure that all percentage values are between 0-100, that the charger power is positive, etc. If any of the values are invalid, your code should throw an IllegalArgumentException. This is a runtime exception, so will not force those using your code to write a try/catch. The supplied tests do, however, use try/catch in places where they are expecting an exception – this is a standard unit testing pattern for making sure code that should throw an exception actually does.

Task 2: Range Estimates

EV owners often keep quite a close eye on their remaining range, concerned at the prospect of getting caught out running out of charge while far away from a charging point. There's even a name for this habitual gauge-watching: *range anxiety*. Fortunately, the calculation of remaining range is quite simple. You will need to implement the getEstimatedRemainingRange() method such that it calculates:

- How many miles per unit-of-charge in the battery has been consumed on the journey so far
- How much battery life is remaining, excluding the reserve charge
- If the current rate of usage is maintained, how much further could be travelled?

The method will return the range estimate in miles, rounded to the nearest 0.1 mile.

Task 3: Charging Time Estimates

Charging time is also a careful consideration for EV owners. If forced to perform a long charge at a motorway service station, for example, it might be difficult to find something to do for 4-5 hours! You are going to implement the getEstimatedChargingTime() method such that it calculates:

- How any kWh will be needed to get from the current charge level to the desired charge level
- How the efficiency of the charger will increase that figure
- How long that will take at the charger's rated power level

The method will return the charging time estimate in minutes, rounded to the nearest minute.

Extension Task: More Comprehensive Tests

The tests I included for the EVBatteryManager methods are quite simplistic, ensuring that you have performed some basic validation, and checking that the calculated estimates are correct in some simple scenarios. A more comprehensive suite of tests would check the code's behaviour when handling range estimates at low charge levels, and charging time estimates on very low and very high powered chargers Design some further tests (remember: *tests first!*) to handle some of the edge cases you have identified, and then re-work your EVBatteryManager implementation so that it passes your new tests.