SE 333 Software Testing

Assignment 5 Black Box Testing

Due date: October 26, 11:59 pm

Late penalties: 1% late penalty for each day of lateness, up to 7 days. No work later than 7 days will be accepted.

The Objective

The objective of this assignment is to design and implement test suites using Equivalence partitioning and parameterized tests.

Overview

The template for this assignment contains a jar file but no source code for the classes under test. Your assignment is to develop tests for this component, using the specifications as your only inputs as to how the applications work.

You must not:

1. Decompile the jar file
2. Re-implement the classes defined in the jar file

Problem specification

Speeding ticket calculator

The town of Plattville calculates speeding tickets using the following formula:

1 - 5 MPH over the limit :

If driver has had no tickets in the last 12 months, give a warning ($0.0 ticket)
Otherwise, $25

6 - 10 MPH over the limit : $25

11 20 MPH over the limit :

If the vehicle has 2 axles, $25 + $5 per each MPH over the limit
If the vehicle has more than 2 axles, $25 + $10 per each MPH over the limit

21 MPH and more : arrest

It is not valid to use the system with a speed that is not over the speed limit.

The test data contains the following drivers

DL12345 - Sam Phillips - no tickets in last 2 years
DL22222 - Latisha Jones - no tickets in last 12 months
DL12121 - Ahmed Ali - 1 ticket 5 months ago
DL54321 - Sarah Miller - 1 ticket 11 months ago

A driver with no record should be treated the same as a driver with no tickets in the last 12 months

An application has been implemented to help law enforcement calculate tickets
The class that implements these rules is TicketCalculator, which has 2 constructors:

```
// loads system with the drivers mentioned above.
public TicketCalculator()

// loads system with drivers you provide
public TicketCalculator(Map<String, LocalDate> drivers)
```

To provide your own driver data, create a Map that contains String keys that reference LocalDate values (date of last ticket).

A driver who never had a ticket will not be in the Map.

Note when creating your own driver data: a valid driver ID in this system has the form “DL” + 5 digits. Calling the TicketCalculator with ids that do not match this formula will result in an error.

This is the declaration of the cost function and the supporting Instruction class:

```
public Instruction computeCost(
   String driverId,
   int legalSpeed,
   int actualSpeed) throws IllegalArgumentException
```

```java
public class Instruction {
    enum Type {
        ARREST,
        FINE
    }
    private Type type;

    private double fine;

    public Instruction(Type type, double fine) {
        this.type = type;
        this.fine = fine;
    }

    public double fine() { return fine; }

    public Type type() { return type; }
}
```

Assignment

1. Identify the equivalence classes for each input variable, including both valid and invalid equivalence classes.

2. Using parameterized tests and other Junit techniques, design and implement a test suite with fewest test cases possible that meets the requirements of *weak normal* tests.

3. Extend the test suite in 2 with fewest additional test cases possible to satisfy the *weak robustness* criteria.

4. Design and implement a test suite with fewest test cases possible that meets the requirements of *strong normal* tests.

5. Extend the test suite in 4 with fewest additional test cases possible to satisfy the *strong robustness* criteria.

Tips

1. You do not need classes for the legal speed limit. You need classes of the difference between some legal limit and an actual speed. It would be perfectly acceptable to make the legal limit identical for all test cases unless you see something in the requirements that would make that unwise.

2. You may find it useful to work out the test cases in a spreadsheet or similar document. Remember that a Normal test case would include an expected numeric result from the test, so a spreadsheet would help you arrive at the right value. Turn in any such documents with your code.

3. The code you are testing contains defects. Do not write a test that lets the test pass, if you must violate the specifications to do so. When in doubt, assume the specifications are right, not the code.

4. The FunctionalTests examples from a previous week show how to use a CSV file as input to parameterized tests, as well as other examples that might be useful to you.

Submission

1. Zip up your solution using gradle in the usual way
2. Upload the zip as well as any spreadsheets or other supporting documents you created for the assignment. Supporting documents should be separate files, not added to the zip file.