COMP3053.SQA Coursework 03
ART Test Harness; MT (30%)

Learning Aims:

This coursework is designed to give you experience writing high-quality object-oriented code to implement an Adaptive Random Testing algorithm; and to get you thinking about testing when faced with the Oracle Problem – using Metamorphic Testing (MT).

Background:

You have been working at the prestigious software development company TopQualitySoftware Inc. Since your arrival there, you have been promoted to "technical lead," and have been receiving increasing amounts of responsibility.

TopQualitySoftware's Chief Technology Officer (CTO), Ms Big, has spoken with you a number of times about the importance of software quality and appropriate documentation. As part of this, she has decided to give you a new project:

> "I want you to build me a test harness. The test harness will work from the command line. It will take a string of arguments, using flags if necessary, and then perform random testing or adaptive random testing on a specified program. For now, all the programs the harness will be used on will only have numeric parameters."

Your friend, Mr Nice, was very excited to hear about this new project. Because he is a QA expert, and an advocate of not only Random Testing, but also Adaptive Random Testing, he had a lot of very helpful advice for you:

> "This is a great idea. I think you should research a little about pseudo-random number generators (PRNGs), especially how to use them in Java, and the importance of the *seed*. It is probably a good idea to research how to scale the output of Java's PRNG to different ranges, without causing a negative impact on the quality of the pseudo-random number sequence. Think carefully about what you'll record in any data file that your harness will generate."

You are already familiar with the DART and RART (RRT) (and MART) versions of ART, and know the various control parameters that each would require. Mr Nice is very enthusiastic about this project. He tells you that it is really important that you get this project right. Perhaps this will be your ticket to becoming the vice-president! As an expert in ART he has some more very helpful advice for you:

> "In addition to all the things you need to do for an ordinary random testing harness, you should choose a version of ART that will work well. Personally, I like the Restriction-based ART implementation (RART/RRT), but you can decide which one to implement yourself. Remember that each ART implementation has different parameters that will need to be passed as arguments on the command line! You'll probably need to include some additional information in the data file, too."

You are very eager to make sure that this project is a success, and you have been studying Metamorphic Testing (MT) to see if there is any way to test the harness once completed.

Basic/Outline Harness Interaction:

After some careful consideration, you come up with an outline plan for how a simple random testing (not ART) test harness could work. From the command line, it would take a sequence of arguments introduced by flags. You have decided that the flags (and their arguments) needed are as follows:

-p <program under test (PUT)>
-o <oracle>
-s <seed>
-n <number of test cases to generate>
-a <number of arguments/parameters the PUT takes>
-r <series of lower and upper bounds for the arguments/parameters, in a [lwr, upr) range, meaning greater than or equal to lwr, but strictly less than upr>

Assuming the harness is called `TestHarness`, and that both the program under test (PUT) and the oracle are in the same folder (and called `PUT` and `Oracle`, respectively), then a typical call to the harness might be:

```
%TestHarness -p PUT -o Oracle -s 7 -n 10000 -a 2 -r 0.0 1.0 -1.0 0.0
```

Which would mean that TestHarness would generate 10000 test cases using the PRNG with the seed 7; each test case would be a 2-tuple with the first number in the range [0.0, 1.0), and the second number in the range [-1.0, 0.0); these test cases would be applied to the programs PUT and Oracle, with the outputs from both compared to see whether or not they are the same: If not the same, that would mean that the harness had found evidence of a failure/fault/bug.

After some further consideration, you have decided that, in addition to those already specified in the ordinary test harness, the following additional flags (and their arguments) would be needed if you were to implement ALL three of the ART algorithms that you know, in addition to RT:

-m [RT|DART|MART|RRT]
-k <Candidate Set size, as an integer: for DART>
-P <Partitioning schema, as a string: for MART>
-R <Target Exclusion Ratio, as a floating-point value: for RRT>

Of course, **you only need to implement one** version of ART!

Again assuming the harness is called `TestHarness`, and that both the PUT and the oracle are in the same folder (and called `PUT` and `Oracle`, respectively), then a call to the harness to run random testing (RT) might be:

```
%TestHarness -m RT -p PUT -o Oracle -s 7 -n 10000 -a 2 -r 0.0 1.0 -1.0 0.0
```

Which would mean that TestHarness would generate 10000 test cases using the PRNG with the seed 7; each test case would be a 2-tuple with the first number in the range [0.0, 1.0), and the second number in the range [-1.0, 0.0); these test cases would be applied to the programs PUT and Oracle, with the outputs from both compared to see if they are the same or not … if the output is not the same, that would mean that the harness had found a failure/fault/bug.

A similar call to the harness, but this time to use the DART implementation of ART, with a candidate set size of 10, might be:

```
%TestHarness -m DART -k 10 -p PUT -o Oracle -s 7 -n 10000 -a 2 -r 0.0 1.0
-1.0 0.0
```

You have also decided that when the test harness is run, the outputs will be put into a newly created file called “TestResults.dat”. You need to think very carefully about what should go into this file, and what should happen if the file already exists.

Finally, you should be careful to build the harness in such a way that it reacts appropriately to incorrect input. Error messages or exceptions may be a good idea.

Metamorphic Testing:

You should identify five (5) metamorphic relations (MRs) for the harness, and use them to conduct Metamorphic Testing (MT) of the harness.

Deliverables/Submissions:

- ### Java Code:

  All the Java code (including any additional libraries) should be put into a zip file (not a package). (Although you are not required to use TDD in this CW, you may choose to do so, even partially. In this case, you should include all your TDD-related test code.)

- ### (Brief) User Manual

  You should prepare a very brief User Manual, no more than about 250 words. This should explain clearly to any users how to set up, prepare, and use the test harness.

- ### Code Report

  You should write a report about the code used in the test harness. This is a document that would go to a maintenance team, so it should include details like algorithms used, external dependencies, known issues, etc. This should not be more than about 500 words.

- ### QA Report detailing testing

  You should write a report about the QA process followed to produce the final test harness. This should include all the steps taken to ensure the quality of the harness, including a listing of (at least) five MRs used for Metamorphic Testing. This should not be more than about 500 words. (Although you are not required to use TDD in this CW, you may choose to do so, even partially. In this case, you should describe your TDD in the QA report).

Submission:

All four deliverables should be put into a .zip file and submitted through Moodle, before the deadline.

Assessment:

Submissions will be manually marked and graded, with the weighting of the parts as follows.

- 60% for the Java code
- 10% for User Manual
- 15% for Code Report
- 15% for QA Report

Deadline:

The deadline will be **Wednesday, 4 pm, December 28th**, 2022.