DIT636 / DAT560 - Mutation Testing Exercise

In previous exercises, you wrote unit tests for a Meeting Planner system based on both the functionality and code structure. We will now return to the Meeting Planner one last time to assess the sensitivity of your test cases to seeded faults in the code.

1. Create at least four mutants for classes of your choice in the MeetingPlanner code.
    a. One mutant must be **invalid** (does not compile).
    b. One must be **equivalent** to the original code (you inserted a fault, but no test case can possibly yield a different solution to the original).
    c. One mutant must be **valid-but-not-useful** (all tests, or almost all tests, will expose this mutation).
    d. The last mutant must be **useful** (only a small number of specific tests will expose this mutation).
    e. Each mutant must be created by applying a different mutation operator, and you must use at least one mutation operator from each of the three categories in the attached handout (for more information, see Chapter 16 of Software Testing and Analysis).
    f. You do not have to use the same classes or methods for all mutant categories. Try mutating different parts of the code. You may use any class except Main or the exception.
2. Assess your test cases that you created in previous exercises, with respect to the set of mutants that you derived - Are you able to kill all of the non-equivalent mutants with your test suite? If not, write additional tests that can kill those non-equivalent mutants.
    a. Test cases that expose a mutant pass on the original code and fail on the mutated code.

If you finish early, try adding mutations to the CoffeeMaker classes from Homework Assignment 2. Do your unit tests detect those mutations? (This will be part of Assignment 3).