## Design

The Class Under Test cut contains various methods, which accept various Types of arguments and returns various Types of values. For simplicity, the arguments and return types are limited to void, int, float and string in cut.

The Test.java contains main method. It takes the name of the Class Under Test 'cut' in command line arguments and instantiates the class, create an object of that class and invokes all the methods of that class using Reflection.

The input values are read from a file 'input.txt' and passed as parameters to the methods. We want to test the robustness of the class. Hence we want to chain the methods. That means, a return value of a method (say m1) is given as argument to another method (say m2) if return type of m1 is same as type of input argument. In this Test Harness, a maximum of three level chaining has been implemented. That means, in some test cases a method say m1 is invoked, its return value is used and argument and invoked another method m2, and another method m3 is invoked using the return value of m2 as argument to m3.
Programmatically, three level chaining is m3 (m2 (m1)). But this is possible only when the return type of m1 is equal to argument type of m2 and return type of m2 is equal to argument type of m3.

As a baseline, first all the methods of the cut class are invoked by passing input values from text file. Then methods were chained at level two. And then methods were chained at a level 3. The same is implemented in blocks under comments "// Invoking all the methods of CUT." , "// two level chaining" and "// Three level chaining " respectively.

The crux of the Test.java lies in dynamically creating test cases and executing them. In our Test.java test cases are generated by

1) Invoking all the methods on cut
2) Chaining all possible methods at level 2. That means all possible M2(M1)
3) Chaining all possible methods at level 3. That means all possible M3(M2(M1))

The phrases "all possible methods at level 2" and "all possible methods at level 3" should be stressed because, if there are n methods in cut, there are $n^2$ combinations for M2(M1) but there will be only a few combinations are syntactically possible, because of the constraint 'Return Type of M1 should be same as argument type of M1' . Generating all such possible test cases involves looping over the available methods twice to check possible combinations. This is implemented in block under "// two level chaining" in Test.java. Similarly, for n methods in cut, there are $n^3$ combinations for M3(M2(M1)), but only a few of them are syntactically valid because of the constraint 'Return type of M1 is equal to argument type of M2 and return type of M2 is equal to argument type of M3'. This implemented in block under "Three level chaining" in Test.java

## Constraints

My main focus is to create as many test cases my chaining all possible combinations and exploring the limit of robustness of cut. Hence I have written a class cut so that there will be at least some valid combinations for chaining. For example, if I want to execute a test case by chaining 3 methods, M3(M2(M1)), there should at least 3 methods M1, M2, M3 such that return type of M1 is same as argument type of M2 and return type of M2 is same as argument type of M1. This is rational in taking my own class instead of a class from Open source. However, the Test.java can test any class. The constraints are

1) The Test.java invokes methods which take void, int, float and String as arguments.
2) Since a method can return only one value, we are chaining only those methods, which will accept one argument.

## Test Cases

The cut has 11 methods. So first 11 test cases are simply invoking each of these methods.
There are 20 valid combinations for two level chaining in cut. So next 20 text cases are these combinations
There are 33 valid combinations for three level chaining. So next 33 test cases are these combinations.
In total, we have 74 test cases.
All the 74 test cases are executed in 5 runs. Each run will read input values from file and randomly pass the arguments to invoked methods.

## Results

Each line of output of Test.java is
TestCase#, which is no of that test case, the method or chained methods invoked and the status of test case (Success or Failed).

**********RUN : 1 **********
Test Case 1 : takeVoidReturnVoid Success
Test Case 2 : takeVoidReturnInt Success
Test Case 3 : takeVoidReturnFloat Success
Test Case 4 : takeIntReturnVoid Success
Test Case 5 : takeIntReturnInt Success
Test Case 6 : takeIntReturnFloat Success
Test Case 7 : takeFloatReturnVoid Success
Test Case 8 : takeFloatReturnInt Success
Test Case 9 : takeFloatReturnFloat Success
Test Case 10 : takeStringReturnint Success
Test Case 11 : takeStringReturnString Success
Test Case 12 : takeIntReturnVoid( takeVoidReturnInt) Success

Test Case 13 : takeIntReturnInt( takeVoidReturnInt) Success
Test Case 14 : takeIntReturnFloat( takeVoidReturnInt) Success
Test Case 15 : takeFloatReturnVoid( takeVoidReturnFloat) Success
Test Case 16 : takeFloatReturnInt( takeVoidReturnFloat) Success
Test Case 17 : takeFloatReturnFloat( takeVoidReturnFloat) Success
Test Case 18 : takeIntReturnVoid( takeIntReturnInt) Success
Test Case 19 : takeIntReturnInt( takeIntReturnInt) Success
Test Case 20 : takeIntReturnFloat( takeIntReturnInt) Success
Test Case 21 : takeFloatReturnVoid( takeIntReturnFloat) Failed
Test Case 22 : takeFloatReturnInt( takeIntReturnFloat) Success
Test Case 23 : takeFloatReturnFloat( takeIntReturnFloat) Success
Test Case 24 : takeIntReturnVoid( takeFloatReturnInt) Success
Test Case 25 : takeIntReturnInt( takeFloatReturnInt) Success
Test Case 26 : takeIntReturnFloat( takeFloatReturnInt) Success
Test Case 27 : takeFloatReturnVoid( takeFloatReturnFloat) Success
Test Case 28 : takeFloatReturnInt( takeFloatReturnFloat) Success
Test Case 29 : takeFloatReturnFloat( takeFloatReturnFloat) Success
Test Case 30 : takeIntReturnVoid( takeStringReturnint) Success
Test Case 31 : takeIntReturnInt( takeStringReturnint) Success
Test Case 32 : takeIntReturnFloat( takeStringReturnint) Success
Test Case 33 : takeIntReturnVoid( takeIntReturnInt ( takeVoidReturnInt ) ) Success
Test Case 34 : takeIntReturnInt( takeIntReturnInt ( takeVoidReturnInt ) ) Success
Test Case 35 : takeIntReturnFloat( takeIntReturnInt ( takeVoidReturnInt ) ) Success
Test Case 36 : takeFloatReturnVoid( takeIntReturnFloat ( takeVoidReturnInt ) ) Success
Test Case 37 : takeFloatReturnInt( takeIntReturnFloat ( takeVoidReturnInt ) ) Success
Test Case 38 : takeFloatReturnFloat( takeIntReturnFloat ( takeVoidReturnInt ) ) Success
Test Case 39 : takeIntReturnVoid( takeFloatReturnInt ( takeVoidReturnFloat ) ) Success
Test Case 40 : takeIntReturnInt( takeFloatReturnInt ( takeVoidReturnFloat ) ) Success
Test Case 41 : takeIntReturnFloat( takeFloatReturnInt ( takeVoidReturnFloat ) ) Success
Test Case 42 : takeFloatReturnVoid( takeFloatReturnFloat ( takeVoidReturnFloat ) ) Success
Test Case 43 : takeFloatReturnInt( takeFloatReturnFloat ( takeVoidReturnFloat ) ) Success
Test Case 44 : takeFloatReturnFloat( takeFloatReturnFloat ( takeVoidReturnFloat ) ) Success
Test Case 45 : takeIntReturnVoid( takeIntReturnInt ( takeIntReturnInt ) ) Success
Test Case 46 : takeIntReturnInt( takeIntReturnInt ( takeIntReturnInt ) ) Success
Test Case 47 : takeIntReturnFloat( takeIntReturnInt ( takeIntReturnInt ) ) Success
Test Case 48 : takeFloatReturnVoid( takeIntReturnFloat ( takeIntReturnInt ) ) Success
Test Case 49 : takeFloatReturnInt( takeIntReturnFloat ( takeIntReturnInt ) ) Success
Test Case 50 : takeFloatReturnFloat( takeIntReturnFloat ( takeIntReturnInt ) ) Success
Test Case 51 : takeIntReturnVoid( takeFloatReturnInt ( takeIntReturnFloat ) ) Success
Test Case 52 : takeIntReturnInt( takeFloatReturnInt ( takeIntReturnFloat ) ) Success
Test Case 53 : takeIntReturnFloat( takeFloatReturnInt ( takeIntReturnFloat ) ) Failed

Test Case 54 : takeFloatReturnVoid( takeFloatReturnFloat ( takeIntReturnFloat ) ) Failed
Test Case 55 : takeFloatReturnInt( takeFloatReturnFloat ( takeIntReturnFloat ) ) Success
Test Case 56 : takeFloatReturnFloat( takeFloatReturnFloat ( takeIntReturnFloat ) ) Failed
Test Case 57 : takeIntReturnVoid( takeIntReturnInt ( takeFloatReturnInt ) ) Success
Test Case 58 : takeIntReturnInt( takeIntReturnInt ( takeFloatReturnInt ) ) Success
Test Case 59 : takeIntReturnFloat( takeIntReturnInt ( takeFloatReturnInt ) ) Success
Test Case 60 : takeFloatReturnVoid( takeIntReturnFloat ( takeFloatReturnInt ) ) Success
Test Case 61 : takeFloatReturnInt( takeIntReturnFloat ( takeFloatReturnInt ) ) Success
Test Case 62 : takeFloatReturnFloat( takeIntReturnFloat ( takeFloatReturnInt ) ) Success
Test Case 63 : takeIntReturnVoid( takeFloatReturnInt ( takeFloatReturnFloat ) ) Success
Test Case 64 : takeIntReturnInt( takeFloatReturnInt ( takeFloatReturnFloat ) ) Success
Test Case 65 : takeIntReturnFloat( takeFloatReturnInt ( takeFloatReturnFloat ) ) Success
Test Case 66 : takeFloatReturnVoid( takeFloatReturnFloat ( takeFloatReturnFloat ) ) Success
Test Case 67 : takeFloatReturnInt( takeFloatReturnFloat ( takeFloatReturnFloat ) ) Success
Test Case 68 : takeFloatReturnFloat( takeFloatReturnFloat ( takeFloatReturnFloat ) ) Success
Test Case 69 : takeIntReturnVoid( takeIntReturnInt ( takeStringReturnint ) ) Success
Test Case 70 : takeIntReturnInt( takeIntReturnInt ( takeStringReturnint ) ) Success
Test Case 71 : takeIntReturnFloat( takeIntReturnInt ( takeStringReturnint ) ) Success
Test Case 72 : takeFloatReturnVoid( takeIntReturnFloat ( takeStringReturnint ) ) Success
Test Case 73 : takeFloatReturnInt( takeIntReturnFloat ( takeStringReturnint ) ) Success
Test Case 74 : takeFloatReturnFloat( takeIntReturnFloat ( takeStringReturnint ) ) Success

Explanation :

In “Test Case 74 : takeFloatReturnFloat( takeIntReturnFloat ( takeStringReturnint ) ) Success” Test Case 74 denotes that it is 74$^{th}$ test case. The test case executed is takeFloatReturnFloat( takeIntReturnFloat ( takeStringReturnint ) ), which a three level chained methods. “Success” denotes that the test case is passed.

The Results of only First Run were shown here. Results for all runs are shown in output.txt