Sandeep Kumar Anil Kumar

Method Chaining Project Report

- I will start with the description of the Class Under Test (CUT) namely **Project.java**. This java file consists of a class called **Project**, which has six methods, having different parameter and return types. There is a static nested class called **second** which extends the Project class.
- The Project class is a **generic** class and has a generic method called "**numberGeneric()**".
- Comments are provided inside the java file Project.java for easy understanding of the code.
- Now, we move to the next java file called **ReflectionObjects.java**, this is the **Test HarnEss (THE)**. The THE will be used to reflect on Project.java at runtime and create instances of the class and invoke its methods with different input parameters and chain them with other methods. The exact process will be explained in the following steps.
- This java file starts with a try block, so that if there are any exceptions during the execution of the program, they will be caught and throw the user the type of the exception and a stack trace of it.
- The THE takes as input the name of the CUT. If the code is run from **Eclipse**, be sure to include the arguments in the **Run configurations > arguments** tab. And specify the value as **project.srcCode.Project**.
- **project.srcCode** is the package of both the THE and CUT.
- The THE takes the name of the CUT and creates a Class object for it. Which will be used to get the classes, methods, constructors and other fields inside the CUT.
- THE is designed in such a way that all the classes inside CUT will be stored in a List and the user will be asked as to which class needs to be tested. Based on the user input the appropriate class constructor will be called and a new instance of that class will be created with **newInstance()** method in the **java.lang.reflect.Constructor**.
- The code is well commented and is self-explanatory if gone through. We first create a Map of all the methods inside the class which is to be tested and the List of all the parameters of the method as value in the map. We also create a Map for all the methods inside the class to be tested as key to the map and return parameters of the methods as the 'value' to the map.
- A list is maintained which consists of all the methods of CUT, at first the method from the Map of method name and parameters is picked up and invoked and all the other methods inside the total method list is iterated and chaining of methods is done. Two approaches are used for this process as described below.
- I have used two different approaches as to how implement method chaining for the CUT.
    - **Approach 1: -** A method from the CUT is invoked and we later iterate through the next method in the CUT and chain it with the return value of the first method irrespective of the return value of the first method nor the parameter type of the next. And continue in this way and chain the next method from the CUT and so on. In this way every method is chained with every other method in the CUT. Appropriate type castings are done in the java file to facilitate this. This approach is depicted in the java file **ReflectionObjects.java**

- **Approach 2**: - A method from the CUT is invoked and based on the return value of the first method, we invoke the next method, which accepts the parameter of the type of the return value of the first method. In Simple terms if we first invoked a method which returns a value of type 'int', we later iterate through all the other methods of CUT and see if there is any method that has a parameter of type 'int' in it's method declaration. If there is a method which accepts 'int' as parameter it'll be chained and invoked, later we will check the return type of this method and continue invoking other methods in similar fashion. This approach is depicted in the file **ReflectionObjects1.java**

- Appropriate statements are output on console during the entire execution of the code, to help the user to understand the flow of control.
- Based on the approach required select the java file and run it as described in the **ReadMe.txt** file.
- Now regarding the input values to the parameters of methods, this is facilitated through the use of an external text file called **MyData.txt.**
- The program is designed in such a way that even if there are no values in the MyData.txt, the THE won't halt or crash. The default values in such a case are given to the input parameters in the THE. But there needs to be at least one float, int and string value so that the methods can be tested properly.
- There is a method which returns void when called, as such during the iteration of methods (chaining), if the return value of the previous method is null, it will be overwritten with 1, so that THE won't fail. And chaining continues as described in the approaches.
- The CUT consists of a static nested class called 'second', it extends the class Project and has a default constructor which has a function call. As such during the creation of the static nested class's object, the parent constructor will be invoked, which also has a method call inside the constructor, which will call the **second's** method with value as 0. Much like the discussion in the class on static nested class (TestClass and B) and also in Quiz 2.
- I have tested the THE harness for different input values and different test cases are as below: -
  - **TestCase 1** : - Values in MyData.txt : - **1.2 abc 1**
    - The classes (Project and second) both ran successfully. There was no exception or error encountered during the program execution.
  - **TestCase 2** : - Values in MyData.txt : - **xyz**
    - The classes (Project and second) both ran successfully. There was no exception or error encountered during the program execution.
  - **TestCase 3** : - Values in MyData.txt : - **100**
    - The classes (Project and second) both ran successfully. There was no exception or error encountered during the program execution.
  - **TestCase 4**: - No Values in MyData.txt
    - The classes (Project and second) both ran successfully. There was no exception or error encountered during the program execution.

- Although the program runs successfully without requiring input values for methods from the external file, if a user needs that a input value be given every time during program execution, the user can throw appropriate message, saying "**No Input values to methods provided**", and make sure the input values are given. I just wanted the program to run successfully for the above test cases as such employed the above implementation. i.e. if the values are present in the external file they will be used, but if they are not, then methods will run taking default values set in the THE.
- Also the location of the MyData.txt file is very important. If the directory structure is maintained as it is (As is have provided), then there is no need to worry. But if in case the program is not able to read the MyData.txt file, the user can print the following command in the java code and can see which location the code is referring to, and can place the MyData.txt file in that location. The command is :- **System.out.println(System.getProperty("user.dir"));**