Github Link:

https://github.com/Alex44711/SOEN_6011

ETERNITY: FUNCTIONS-Problem 4

According to the requirements that proposed in problem 2, the function is implemented by Java from "scratch" in this section. The seven requirements are all implements in this application. All the code follow the Google Java program style and under checkStyle verification.

Effort Made

In order to build the correct, efficient, maintainable, robust, and usable program. I made several effort to achieve these attributes.
For **usable and correct** attributes, I wrote the unit test for each function and executed the acceptance test to keep the correct and usable of the program. According to the results of tests, the program can be proven usable and correct. For the **efficient** attribute, the most time-consuming function in this program is the square method. So, I used the Newton iteration method to get the square result of the double number, which is more efficient and accurate. To achieve the **maintainable** attribute, I seperated the program into several methods, each one corresponding to a specific requirement, and there are only the launch methods exist in the main method. which is easy to modify the function of the program or add new needs. To make the program **robust**, I added input validation check in every input statement. If the input is invalid, the console will show the corresponding reminder and request for the new input. It avoid the exception happen and improve the robust of the program.

Debugger Description

In order to solve the problems I encountered when programming, I used the eclipse build-in debugger, which is easy to use and convenient. For debug a java application program, we need to enter the eclipse debug view first, just click the button that look like a green bug in the right-top of the window. Then, we can see the debug view, which contains lots of related views.

View	Description
Debug view	Mainly show the current method invoke stack and the line number of code
Variables view	Show the current mehod local fields, non-static method
Breakpoints View	BreakPoint list window
Expressions View	Show the current implemented expression
Display View	Log and related contents output area

From the related view, we can know the states of specific variables. If we want to know the state of one specific in somewhere of this program, we can set the breakpoint in this line. The approach to set the breakpoint:

Key	Descritpion
ctrl+shift+b	Set/cancel breakpoint in the cursor location
ctrl+alt+b	Ignore all the breakpoint
Alt+shift+q, b	Active all the breakpoint view

Then running the program and open the debug view, implement the debug process.

Function	Key	Description
Step Info	F5	single step enter the invoke method
Step Over	F6	Direct implement over the current code and jump to the next line
Step Return	F7	Single return the place of current method was invoked
Resume	F8	Resume the normal implementation until encounter the next breakpoint

In order to find the problem, the data inspection is necessary. So, we need to check the execute result of the expression. The related key is described in the following.

Function	Key	Description
Inspect	ctrl+shift+i	Check the result of variable or expression or execution.
Display	ctrl+shift+d	Show the result of selected variable, expression or execution.
Execute	ctrl+u	Implement the selected expression.
Run to Line	ctrl+r	Exexute to the current line.
All Instances	ctrl+shift+n	Check all the objects in current class.

Using the build-in debugger appropriately can solve most of problem we encountered in the development process.All above are about the **advantages** of the Eclipse build-in debugger. However, the **disadvantages** of debugger is it only can show the statues of variables and expressions. These statues help programmer judge the location of problem happened, it cannot find the logical error of the program automatically.

Code Style Check

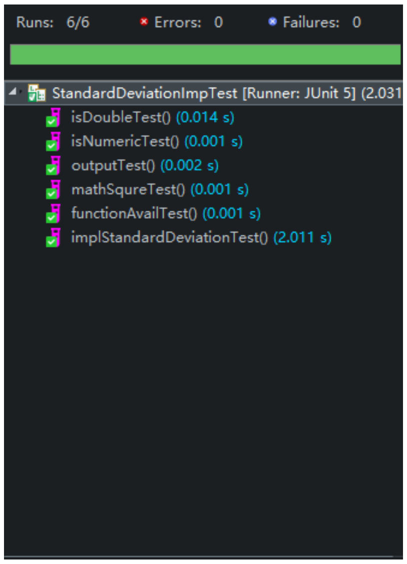
In order to check the quality of the source code, i choose the **Checkstyle** as the code quality validation tool. CheckStyle is a project under SourceForge that provides a tool to help Java developers adhere to certain coding conventions. It automates the code specification checking process, freeing developers from this important but boring task[1]. It can check the code according to the set encoding rules[2]. For example, variable naming in accordance with the specification, maximum number of rows in the method body, duplicate code checking, and so on. I used the Eclipse as the IDE, so i had to install the Checkstyle plugin into Eclipse. I chose the online installation, click the help− >install new software then input the source link and click the next follow the installation interface. Restart eclipse after the installation process completed. Then, i configured the the checkstyle as google, which is the code sepcification we used[3].
By default, it supports the java style guide for google and sun. And it's highly configurable, allowing for custom coding specifications and support for various IDEs (eclipse, Intellij) and build tools (maven, gradle). After the configuration, user can right click on the code editor, select the checkstyle− >checkcode with checkstyle, then you can see the the code highlight where the code is incompatible with the google java specification. It can help programmer dind out the details that programmers can easily ignore. The typical wrong is like the following:

Reminder	Solution
Type is missing a javadoc commentClass	add javadoc comment
"{" should be on the previous line	put "{" on previous line
Methos is missing a javadoc comment	add javadoc cpmment
"=" is not followed with whitespace	add a space after "="
Line has trailing spaces	remove the extra space

There are many other reminders, just correct it as the reminder description. After the correct, clike the check code with checkstyle again to check if there are other lines are marked as incompatible. If there is left lines marked as incompatible, repeat the previous process again. My code does not have any incompatible with the google java specification.
Except the **advantages** of Checkstyle mentioned above. The mainly **disadvantage** of Checkstyle is the Checkstyle only can find the problem where it is and show the reminder, it cannot correct it automatically. So, the programmer have to read the reminder then correct it one by one. It is not a problem in a small-scale application. But if there is application with more than 100M lines of code. It is impossible to correct the incompatible location manually. The other problem is Checkstyle only can help people find the problem related with code style. It cannot analysis the code metrics like complexity, coupling, lines of code, declaration of method and so on.

ETERNITY: FUNCTIONS-Problem 6

In order to ensure the robust, correct of this program. The Junit 5 is selected as the unit test framework. I wrote 6 test to keep the corresponding methods are correct under unit test. Since this program rely on the Scanner class, so the test will implement until the user input the parameter. As we can see following, all the 6 methods passed the unit test and ran correctly.



```
Runs: 6/6    Errors: 0    Failures: 0

StandardDeviationImpTest [Runner: JUnit 5] (2.031
    isDoubleTest() (0.014 s)
    isNumericTest() (0.001 s)
    outputTest() (0.002 s)
    mathSqureTest() (0.001 s)
    functionAvailTest() (0.001 s)
    implStandardDeviationTest() (2.011 s)
```

For the **traceability** with the requirement in problem 2:

Requirement	correspondingUnitTest
1. Available Function display	functionAvailTest()
2. Functions selection	implStandardDeviationTest()
3. Parameter input	implStandardDeviationTest()
4. Get the result	outputTest() & mathSqureTest()
5. Whether start another calculate	outputTest()
6. Input parameter validation check	isDoubleTest() & isNumericTest()

The corresponding relationship between unit test and requirement in problem 2 are presented in the table above. As we can see, each requirement are included in the methods of the program and there are tested by Junit. It can ensure all the requirements are satisfied by this program.

Reference

[1] https://www.cnblogs.com/woshimrf/p/using-checkstyle.html
[2] https://blog.csdn.net/sunjavaduke/article/details/4708924
[3] https://blog.csdn.net/qq_36871364/article/details/72472059