ISE Final Assessment Report

By Jackson Mowatt Gok, 20568818

Practical Class: Thursdays/8am-10am/314.218

Note : Apologies for the stretched pages, this had to be the layout in order to fit the 'Test implementation and execution' table

Introduction

In total I have 4 necessary files for my code. One of these is "Scenario.java" which is the main code that has the method creations inside, another is the test case file "ScenarioTestCases.java" that utilises JUnit and my main code, another is "values.csv" which is the example file I use for reading a file, and the last is "OutputFile.csv" which is the example file I use for writing to a file. Within the Scenario.java file, I have 12 separate methods/functions. These are my "main" method which controls the menu system, the next 9 methods are the 'calculation/conversion methods' which address the problems addressed in "Category 1" and " Category 2 part a" in the assessment outline and my final two methods control reading and writing to a file. The code works so that there is a menu system which the user operates through by entering their choice into the system. They first, choose any of the operations specified in the "Categories" from the assessment outline, they then then enter an input or sometimes, depending on their choice, have the option to use a separate file's predetermined values and finally their input is then run through the operation of their choosing and the output is printed to the user and sometimes, depending on their choice, written to the "OutputFile.csv" file. Within each of these methods there is further functionality, which I'll discuss in later sections.

Module Descriptions

Original Ideas of modules:

module: converting string to upper or lower case

```
function name: convertString
takes two parameters: pInputString (input by user) and a boolean checker to swap between upper or lower case
returns pInputString in whatever chosen format determined by user (upper or lower case)
```

module: Identify whether numeric values are in a given string

```
function name: numericFilter
takes one parameter: pInputString (input by user)
turns the string into an array of characters
loop through character array and add +1 to a counter if a character is a numeric value
print out how many numeric values are in the array
```

module: Identify whether a given string is a valid number or not

```
function name: numberChecker
takes one parameter: pInputString (input by user)
parses an integer of the given string
if its not an integer itll spit out an error
if it is, itll print and tell you
```

module: remove any numeric values from a given array and then convert the string to upper or lower case

```
function name: removeNConvert
takes two parameters: pInputString (input by user) and a boolean checker
takes pInputString and turns it into a character array
loops through the array and whatever elements are not numeric values are added to a new array
then that new array is converted back to string and plugged into convertString along with the boolean checker
the newString is returned
```

module: converting meters to feet and vice versa

```
function name = metersNFeet
takes two parameters: pInputDoub and boolean checker
if the checker is true itll assume the double is in meters and will convert to feet (* 3.281)
if the checker is false itll assume the double is in feet and will convert to meters (/ 3.281)
return the result
```

module: converting centimeters to inches and vice versa

```
function name = centiNInches
takes two parameters: pInputDoub and boolean checker
if the checker is true itll assume the double is in centimeters and will convert to inches (/ 2.54)
if the checker is false itll assume the double is in inches and will convert to centimeters (* 2.54)
return the result
```

module: for reading files

```
function name = readFile
takes one parameter: pFileName
creates all necessary file reading variables: file stream, InputStreamReader, BufferedReader and something that reads the line of the BufferedReader
splits the line using regex (.split)
converts the strings from the line into doubles
returns the doubles
```

module: for writing to files

```
function name = writeOneRow
takes one parameter: pFileName, double array pValArray
creates all necessary file writing variables: file stream, print writer
for each value in the array, print that to the file, separated by a space
close the print writer
```

module: for all menu control

```
function name: main
outputs options of menu to user by printing to console
takes input from user and uses that to do one of multiple options using the above functions
prints the outcome of the users choice to the console
```

REVISED MODULE DESCRIPTIONS:

module: converting string to upper or lower case **HAS NOW BEEN SPLIT INTO TWO METHODS**:

module: converting string to upper case

```
function name: stringUpper
takes one parameter: pInputString
returns pInputString in uppercase
```

module: converting string to lower case

```
function name: stringLower
takes one parameter: pInputString
returns pInputString in lowercase
```

module: Identify whether numeric values are in a given string. **See last line for revision**

```
function name: numericFilter
takes one parameter: pInputString (input by user)
turns the string into an array of characters
loop through character array and add +1 to a counter if a character is a numeric value
return the counter value (the revision)
```

module: Identify whether a given string is a valid number or not. **See 3rd and 4th line for revision**

```
function name: numberChecker
takes one parameter: pInputString (input by user)
parses an double (revision) of the given string
returns double (revision)
```

module: remove any numeric values from a given array and then convert the string to upper or lower case **See second last line for revision**

```
function name: removeNConvert
takes two parameters: pInputString (input by user) and a boolean checker
takes pInputString and turns it into a character array
loops through the array and whatever elements are not numeric values are added to a new array
then that new array is converted back to string and plugged into either stringUpper or stringLower depending on boolean (revision)
the newString is returned
```

module: converting meters to feet and vice versa **HAS NOW BEEN SPLIT INTO TWO METHODS**:

module: convert meters to feet

```
function name = metersToFeet
takes one parameter: pInputDoub
multiply pInputDoub by 3.281
return the result
```

module: convert feet to meters

```
function name = feetToMeters
takes one parameter: pInputDoub
divide pInputDoub by 3.281
return the result
```

module: converting centimeters to inches and vice versa **HAS NOW BEEN SPLIT INTO TWO METHODS**:

module: convert centimeters to inches

```
function name = centiToInches
takes one parameter: pInputDoub
divide pInputDoub by 2.54
return the result
```

module: convert inches to centimeters

```
function name = inchesToCenti
takes one parameter: pInputDoub
multiply pInputDoub by 2.54
return the result
```

Assumptions I made:

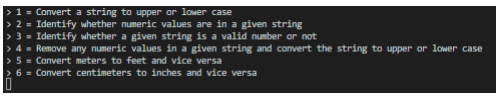
Whilst many of the methods have specific try catch statements, a lot of the time I assumed the user would be inputting the same variable type as they were asked for. For example, when asked to enter a double or a boolean to make their choices/inputs, I assumed the user would input a double or a boolean in their specific case. If they do not, the code will throw an error to the user.

Modularity

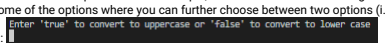
Description on how to run my code

First, enter the 'code' folder.

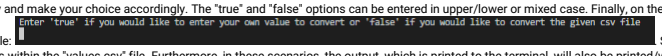
When you first run the main code, 'Scenario.java', you'll be prompted with the first menu:



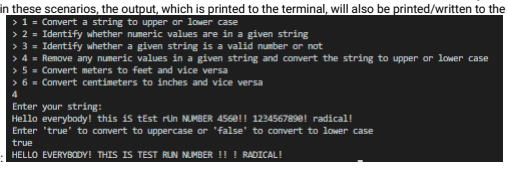
You are then prompted to enter your choice (as seen by the cursor underneath the options). To choose an option, enter the digit in front of the explanation. For example, if I wanted to convert a string to upper or lower case, I would enter 1. In the first four options, you'll be asked to enter a string or for the last two options, you'll be asked to enter a double. Enter normally using the terminal line. In some of the options where you can further choose between two options (i.e. convert a string to upper or lower case, convert centimeters to inches and vice versa), you will be prompted with another menu to ask what you want to be done with the value you have entered:



In these scenarios, read the prompt carefully and make your choice accordingly. The "true" and "false" options can be entered in upper/lower or mixed case. Finally, on the last two options on the first menu, you will be prompted with another menu asking if you want to enter your own values or use the given csv file:



Similarly, read the prompt carefully and make your choice accordingly. In these scenarios, the "given csv file" is referring to using the values within the "values.csv" file. Furthermore, in these scenarios, the output, which is printed to the terminal, will also be printed/written to the "OutputFile.csv" file. Which brings me to the final point; for all the options given within the menu, their final output/calculation will be printed to the terminal line. Full example:



Discussion on different modularity concepts applied

There are roughly 3 loose concepts I want to address in this topic; cohesion, couping and redundancy/repitition. I'll elaborate to discuss how I implemented or avoided these concepts within my code.

Cohesion:

To allow my code to be cohesive, every simple calculation/conversion is separated into its own individual method. All of the code required to transform a given variable is contained entirely within that method. These methods are all labelled in a way that is easy to understand for any reader (i.e. stringUpper, numericFilter, removeNConvert, metersToFeet). This also applies for any variable created (i.e. pInputString, belowZeroChecker, counter, valArray). At one point in my code I did use a control flag, which is discouraged in most cases, however I believe it did not impact my cohesion. The only control flag used within my code was for removeNConvert to choose between upper or lower case. I believe this level of control flag use is justified in being viable due to reducing repetition in code whilst also barely impacting the level of cohesion within my code. I will further elaborate on this in the 'Coupling' section. On a separate note, all that is within these methods is purely conversion, there is no print statements or type casting done, all that is handled within the 'main' method and the rest are left to do pure calculation. This reduces the amount of unrelated tasks within a method. Similarly, there is a maximum of two variables imported as parameters per method, this reduces the amount of 'different data' handled per method.

Coupling

As previously mentioned, to reduce coupling there is a maximum of two parameters per method. This reduces the amount of calls per method. On that point, there is one section within the code where a calculation method calls upon another calculation method to perform its overall task, which is usually discouraged. This is within the "removeNConvert" method where "stringUpper" or "stringLower" is called at the end. Whilst this is usually discouraged, I believe this one case is justified. As mentioned within the overall unit, there

is a constant balancing act between 'maximising reuse' and 'minimising coupling'. In order to reduce redundancy and increase reuse within my code, I decided to utilise these pre-existing methods in 'removeNConvert' so that there was no unneccessary duplication. Similarly, this is also the same reasoning I used for allowing the control flag within removeNConvert, I briefly touched on this prior but my reasoning for the control flag within removeNConvert was based on the principle of balancing 'reuse/minimising redundancy' with 'minimising coupling/maximising cohesion'. If I was to not use a control flag, and was trying to minimise coupling / maximise cohesion, I would've had to create an almost duplicate method with just the last line changed. In order to avoid this redundancy, I chose to implement a minor control flag. To decrease coupling, there was no global variables used throughout the entire code.

Review Checklist and its results

Checklist contained: - No global variables - Low number of calls per method - Reduce control flags as much as possible - One main purpose per method (no 'different data')

This checklist impacted the way I went about writing my code and even forced me to go back and refactor a lot of it. Having the idea of the checklist in mind, I refrained from global variables and kept calls low. However I did not notice the amount of control flags I was putting in my code. After creating the checklist and comparing it with my code, I decided to refactor it in accordance with my control flag rule. This can be seen in the git log (git hash 96f2f9a). Of course, I left one minor control flag in the code but this has been justified previously in this report.

Black Box testing

All my black box test cases can be seen within my ScenarioTestCases.java file where I use JUnit.

My equivalence partioning test cases go as follows:

- Testing stringUpper
  - pInputString is 'my full name as appears on student ID' in upper+lower case = 'Jackson Mowatt Gok'
  - pInputString is 'my last name' in upper case = 'MOWATT GOK'
  - pInputString is 'name of a movie I wish to watch' in lower case = 'interstellar'
  - pInputString is 'last four digits of my student ID' in numbers = '8818'
- Testing stringLower
  - pInputString is 'my full name as appears on student ID' in upper+lower case = 'Jackson Mowatt Gok'
  - pInputString is 'my last name' in upper case = 'MOWATT GOK'
  - pInputString is 'name of a movie I wish to watch' in lower case = 'interstellar'
  - pInputString is 'last four digits of my student ID' in numbers = '8818'
- Testing numericFilter
  - pInputString with no numeric values = 'Jackson Mowatt Gok'
  - pInputString with some numeric values = 'Jackson Mowatt Gok 2002'
  - pInputString with all numeric values = '24052022'
- Testing numberChecker
  - pInputString is all numbers = '24052022'
  - pInputString is numbers mixed with characters '24/05/2022'. THIS CATCHES AN EXCEPTION WHICH IS CALCUALTED FOR IN THE JUNIT
- Testing removeNConvert
  - Choosing to convert to upper case. pInputString is all lower case = 'jackson mowatt gok'
  - Choosing to convert to upper case. pInputString is all lower case and numbers = 'jackson mowatt gok 19'
  - Choosing to convert to upper case. pInputString is all numbers = '19'
  - Choosing to convert to upper case. pInputString is all upper case = 'JACKSON MOWATT GOK'
  - Choosing to convert to upper case. pInputString is all upper case and numbers = 'JACKSON MOWATT GOK 19'
  - Choosing to convert to upper case. pInputString is all upper+lower case (mixed) = 'jAcKson MoWaTt Gok'
  - Choosing to convert to upper case. pInputString is all upper+lower case (mixed) and numbers = 'jAcKson MoWaTt Gok 19'
  - Choosing to convert to lower case. pInputString is all lower case = 'jackson mowatt gok'
  - Choosing to convert to lower case. pInputString is all lower case and numbers = 'jackson mowatt gok 19'
  - Choosing to convert to lower case. pInputString is all numbers = '19'
  - Choosing to convert to lower case. pInputString is all upper case = 'JACKSON MOWATT GOK'
  - Choosing to convert to lower case. pInputString is all upper case and numbers = 'JACKSON MOWATT GOK 19'
  - Choosing to convert to lower case. pInputString is all upper+lower case (mixed) = 'jAcKson MoWaTt Gok'
  - Choosing to convert to lower case. pInputString is all upper+lower case (mixed) and numbers = 'jAcKson MoWaTt Gok 19'

My boundary analysis test cases go as follows:

- Testing metersToFeet
  - testing above zero. pInputDoub = 0.1
  - testing below zero. pInputDoub = -0.1
- Testing feetToMeters
  - testing above zero. pInputDoub = 0.1
  - testing below zero. pInputDoub = -0.1
- Testing centiToInches
  - testing above zero. pInputDoub = 0.1
  - testing below zero. pInputDoub = -0.1
- Testing inchesToCenti
  - testing above zero. pInputDoub = 0.1
  - testing below zero. pInputDoub = -0.1

As seen through my 'ScenarioTestCases.java' JUnit file, all these test cases went through perfectly. I created my test cases this way so that every possible significant input is tested on each method. It's important to mention that for numericChecker there is two separate testing methods, one of which detects and catches the exception from the method.

White Box testing

My two methods used for white box testing are my 'readFile' and 'writeOneRow' methods.

For 'readFile' the test case was inputting an invalid file name. This was done manually in the code and not through JUnit. As can be seen here: `valArray = readFile(pFileName: "vales.csv");`

"values.csv" is spelt incorrectly and so the code won't recognise it. Then, as expected in this line of code within readFile:

```
}catch(IOException errorDetails){
    if(fileStream != null){
        try{
            fileStream.close();
        }catch(IOException ex2){

        }
    }
    System.out.println("An error! " + errorDetails.getMessage());
```

the incorrect file name will supposedly trigger this catch statement. As expected, this was the outcome:

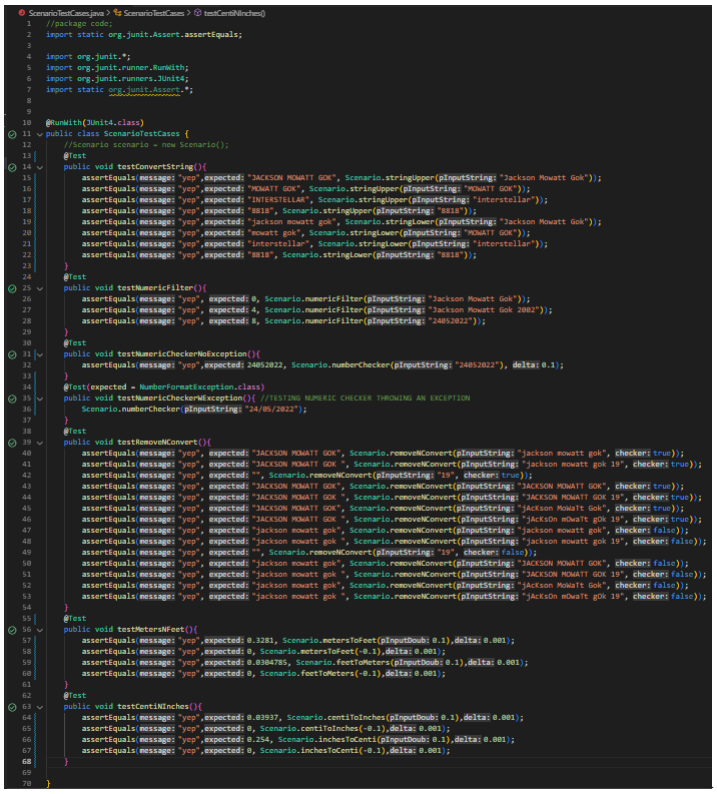
```
PS C:\Users\jacks\Downloads\Mowatt Gok_Jackson28568818_ISErepo\code>  & 'C:\Users\jacks\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.2.8-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '@C:\Users\jacks\AppData\Local\Temp\cp_e16s771iwx2lhuvbjyglp539f.argfile' 'Scenario'
> 1 = Convert a string to upper or lower case
> 2 = Identify whether numeric values are in a given string
> 3 = Identify whether a given string is a valid number or not
> 4 = Remove any numeric values in a given string and convert the string to upper or lower case
> 5 = Convert meters to feet and vice versa
> 6 = Convert centimeters to inches and vice versa
5
Enter 'true' if you would like to enter your own value to convert or 'false' if you would like to convert the given csv file
false
An error! vales.csv (The system cannot find the file specified)
```

As you can see in the last line, the catch statement executed as expected.

Similarly, for 'writeOneRow' the test case was inputting an invalid file name. This was also done manually. As can be seen here: `writeOneRow(pFilename: "OutputFil.csv", valArray);` "OutputFile.csv" is spelt incorrectly and so the code will misconstrue it for a new different file. In this case, a new separate file will be made accidentally and the values will be printed into it. Implementing these test cases leads exactly to that result, executing as expected.

Test implementation and execution

To run all my black box testing, all that is needed is to run the JUnit tests within the 'ScenarioTestCases.java' file. This file includes every black box test case discussed. How to run this file will depend on what interface you use. For me, on Visual Studio Code, I specifically open the 'code' folder and click the small green tick next to line 10 'public class ScenarioTestCases' which will automatically run all the JUnit tests using Visual Studio Code's in built functions. This layout can be seen in the following screenshot:



The green ticks are on the left of the line numbers. For someone on a Unix terminal system, you would need to run the necessary JUnit compilers and runners to run this code. This would look something like:

$ javac -cp junit-path ScenarioTestCases.java

$ java -cp junit-path org.junit.runner.JUnitCore ScenarioTestCases

Where junit-path is replaced with the path of the file in your computers directories.

To run all my white box testing, all that is needed is to edit two lines. This have been demonstrated above but I'll elaborate further. To white box test the 'readFile' file, go to line 53 'valArray = readFile("values.csv");' and change values.csv to vales.csv. After this, run the code and enter '5' on the first menu. On the second input, enter 'false' and then you will get the desired test case as the error statement prints. Similarly, to white box test the 'writeOneRow' file, go to line 120 'writeOneRow("OutputFile.csv",valArray);' and change "OutputFile.csv" to "OutputFil.csv". After this, run the code, enter '5' on the first menu and then 'false' on the second input and then 'false' on the third input. After this, you will get the desired test case as the new 'OutputFil.csv' file is created.

Success and failures of test implementation

On the first test implementation, every test case went through perfectly except for 'numberChecker' 's test case. In this example, I was going to enter an invalid number ("24/05/2022") and expect the test case outcome. Of course, this would've caused an error. In the main code however, this error is handled within 'main' where the input for this method is entered and the 'numberChecker' is left to do nothing but calculate. So unfortunately this would cause issues for this JUnit test case as I couldn't call upon 'main' but had to implement something directly within 'numberChecker' or JUnit. Originally I tried something like a catch statement within numberChecker, something that returned -1 if the catch statement was triggered. However this would mean that the main method would have to recognise that -1 means an error from numberChecker, meaning that no one could ever actually put the string "-1" through to see if it is a number. So after some tinkering and google search, I found that you can actually expect an error through JUnit and have it be a valid test case. And so 'testNumberCheckerWException' was created with this ability.

Table

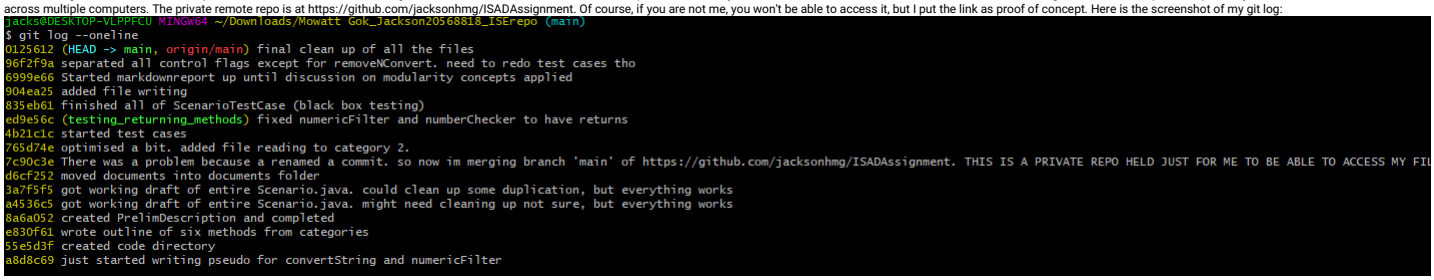
For this table, I will not copy verbatim what is in my "ScenarioTestCases.java" JUnit file for the sake of redundancy. I will, however, copy the main points across, which can be cross referenced with a JUnit file if necessary.

Module name	BB test design (EP)	BB test design (BVA)	WB test design	EP test code (implemented/run)	BVA test code (implemented/run)	White-Box testing (implemented/run)
stringUpper	done	not done	not done	assertEquals("yep","JACKSON MOWATT GOK", Scenario.stringUpper("Jackson Mowatt Gok")); assertEquals("yep","MOWATT GOK", Scenario.stringUpper("MOWATT GOK")); assertEquals("yep","INTERSTELLAR", Scenario.stringUpper("interstellar")); assertEquals("yep","8818", Scenario.stringUpper("8818"));		
stringLower	done	not done	not done	assertEquals("yep","jackson mowatt gok", Scenario.stringLower("Jackson Mowatt Gok")); assertEquals("yep","mowatt gok", Scenario.stringLower("MOWATT GOK")); assertEquals("yep","interstellar", Scenario.stringLower("interstellar")); assertEquals("yep","8818", Scenario.stringLower("8818"));		
numericFilter	done	not done	not done	assertEquals("yep", 0, Scenario.numericFilter("Jackson Mowatt Gok")); assertEquals("yep", 4, Scenario.numericFilter("Jackson Mowatt Gok 2002")); assertEquals("yep", 8, Scenario.numericFilter("24052022"));		
numberChecker	done	not done	not done	assertEquals("yep",24052022, Scenario.numberChecker("24052022"), 0.1);		
numberChecker again (calculating exception so in a different test method)	done	not done	not done	@Test(expected = NumberFormatException.class) public void testNumberCheckerWException(){ Scenario.numberChecker("24/05/2022"); }		
removeNConvert	done	not done	not done	assertEquals("yep", "JACKSON MOWATT GOK", Scenario.removeNConvert("jackson mowatt gok", true)); assertEquals("yep", "JACKSON MOWATT GOK ", Scenario.removeNConvert("jackson mowatt gok 19", true)); assertEquals("yep", "", Scenario.removeNConvert("19", true)); assertEquals("yep", "JACKSON MOWATT GOK", Scenario.removeNConvert("JACKSON MOWATT GOK", true)); assertEquals("yep", "JACKSON MOWATT GOK ", Scenario.removeNConvert("JACKSON MOWATT GOK 19", true)); assertEquals("yep", "JACKSON MOWATT GOK", Scenario.removeNConvert("jAcKson MoWaTt Gok", true)); assertEquals("yep", "JACKSON MOWATT GOK ", Scenario.removeNConvert("jAcKsOn mOwaTt gOk 19", true)); assertEquals("yep", "jackson mowatt gok", Scenario.removeNConvert("jackson mowatt gok", false)); assertEquals("yep", "jackson mowatt gok ", Scenario.removeNConvert("jackson mowatt gok 19", false)); assertEquals("yep", "", Scenario.removeNConvert("19", false)); assertEquals("yep", "jackson mowatt gok", Scenario.removeNConvert("JACKSON MOWATT GOK", false)); assertEquals("yep", "jackson mowatt gok ", Scenario.removeNConvert("JACKSON MOWATT GOK 19", false)); assertEquals("yep", "jackson mowatt gok", Scenario.removeNConvert("jAcKson MoWaTt Gok", false)); assertEquals("yep", "jackson mowatt gok ", Scenario.removeNConvert("jAcKsOn mOwaTt gOk 19", false));		
metersToFeet	not done	done	not done		assertEquals("yep",0.3281, Scenario.metersToFeet(0.1),0.001); assertEquals("yep",0, Scenario.metersToFeet(-0.1),0.001);	

Module name	BB test design (EP)	BB test design (BVA)	WB test design	EP test code (implemented/run)	BVA test code (implemented/run)	White-Box testing (implemented/run)
feetToMeters	not done	done	not done		assertEquals("yep",0.0304785, Scenario.feetToMeters(0.1),0.001); assertEquals("yep",0, Scenario.feetToMeters(-0.1),0.001);	
centiToInches	not done	done	not done		assertEquals("yep",0.03937, Scenario.centiToInches(0.1),0.001); assertEquals("yep",0, Scenario.centiToInches(-0.1),0.001);	
inchesToCenti	not done	done	not done		assertEquals("yep",0.254, Scenario.inchesToCenti(0.1),0.001); assertEquals("yep",0, Scenario.inchesToCenti(-0.1),0.001);	
readFile	not done	not done	done			change line 53 to "valArray = readFile("vales.csv");" purposefully mispelling values.csv. On the menu system, enter '5' and then 'false'.
writeOneRow	not done	not done	done			change line 120 to "writeOneRow("OutputFil.csv",valArray);" purposefully mispelling OutputFile.csv. On the menu system, enter '5', then 'false' and then 'false'.

Version Control

For this assignment, I used Git. It's important to mention that in one of the git commits, there is reference to a remote repo. This is already addressed within the commit 7c90c3e but I will address it again here, that is a private repo held just for me to access these files across multiple computers. The private remote repo is at https://github.com/jacksonhmg/ISADAssignment. Of course, if you are not me, you won't be able to access it, but I put the link as proof of concept. Here is the screenshot of my git log:



And here is the text version of the git log:

commit 012561256fdd7d7114eda8c6b5daeb761fac7a6d Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Sun May 29 11:40:32 2022 +0800

```
final clean up of all the files
```

commit 96f2f9afb5d043f74f4ae0ef423d2a8338bed33b Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Fri May 27 19:52:48 2022 +0800

```
separated all control flags except for removeNConvert. need to redo test cases tho
```

commit 6999e66870afb6a4161cc1b0cd4e9527a76f6041 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Wed May 25 00:06:50 2022 +0800

```
Started markdownreport up until discussion on modularity concepts applied
```

commit 904ea25f24604f3d8ce999e212d9b583b3e2659c Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 24 13:40:41 2022 +0800

```
added file writing
```

commit 835eb61ea6484b8261e59b3c4365a1bf9361e4bb Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 24 12:19:51 2022 +0800

```
finished all of ScenarioTestCase (black box testing)
```

commit ed9e56c59305ffb6bdc5d8cc062a43745a914e48 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 24 11:18:57 2022 +0800

```
fixed numericFilter and numberChecker to have returns
```

commit 4b21c1c761b9da0752443147953233c7a232de3c Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 24 11:13:28 2022 +0800

```
started test cases
```

commit 765d74e8380a7fda24c006fb18dfe1fa04c0b871 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Sun May 22 11:08:21 2022 +0800

```
optimised a bit. added file reading to category 2.
```

commit 7c90c3ec3e49431d1aaccd85781ae69ab629b076 Merge: d6cf252 a4536c5 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Fri May 20 19:20:56 2022 +0800

```
There was a problem because a renamed a commit. so now im merging branch 'main' of https://github.com/jacksonhmg/ISADAssignment. THIS IS A PRIVATE REPO HELD JUST FOR ME TO BE ABLE TO ACCESS MY FILES FROM ANY COMPU
```

commit d6cf2527d2adabb0c62e6d7e8aee269f5f7a4697 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Fri May 20 19:20:13 2022 +0800

```
moved documents into documents folder
```

commit 3a7f5f5f97990c157117476ea944fd6e4fc6aa4f Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Fri May 20 19:15:29 2022 +0800

```
got working draft of entire Scenario.java. could clean up some duplication, but everything works
```

commit a4536c5d8e1b48a05a4165f9118ab1d68c8eab29 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Fri May 20 19:15:29 2022 +0800

```
got working draft of entire Scenario.java. might need cleaning up not sure, but everything works
```

commit 8a6a0529706995d6f6f028d73ad5fa8a4632dc9b Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Fri May 20 18:14:39 2022 +0800

```
created PrelimDescription and completed
```

commit e830f61795b70e27c8e1772aa05bac0ffea215ea Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 17 13:55:09 2022 +0800

```
wrote outline of six methods from categories
```

commit 55e5d3f7391fb1c04f726d7e129f49d978ee9741 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 17 12:56:15 2022 +0800

```
created code directory
```

commit a8d8c6964518a2ea91f9267b44a52d9b752f0768 Author: Jackson Mowatt Gok jacksonmowattgok@gmail.com Date: Tue May 17 12:51:57 2022 +0800

```
just started writing pseudo for convertString and numericFilter
```

Ethics and Professionalism

How lack of ethics and professionalism can result in harmful effects using my code

After researching through the unit's content, I have concluded that there are two main ways in which harmful effects can result from my code. These can conveniently be splitted into lack of ethics and lack of professionalism.

For lack of ethics, a software developer having a lack of ethics can range from being complacent to being involved in unethical companies. Both of these, of course. can be extremely harmful for the public. For example, considering most of the code has to do with filtering and editing strings, my code could be used for something unethical like scraping through exploited/hacked emails. This creates a harmful result as I would then be assisting in a crime and would be harming the public potentially financially psychologically.

Another example, more on the side of complacence side, could relate to my distance conversion methods (metersToFeet, feetToMeters, centiToInches, inchesToCenti). If I was to be unethical in a way more subtle, not accurately pay attention to the methods, not do any testing and ship the code without knowing 100% that it works 100% of the time. That could have major repurcusssions. If that faulty distance converter was then used for a self-driving car that needed pin-point accurate measurements, that could cause the car to crash, steer off track or worse. This would result in physical, financial and psychological harm for a user.

For lack of professionalism, this is more subtle. A lack of professionalism relates more to a work-place environment and the harm that can be caused there. An example could be failing to communicate the purpose of your code well, such as not labelling variables well or not leaving descriptive comments. This would lead to psychological harm to your fellow team members as they struggle to understand. It could potentially lead to harm to outside users if team members misunderstand and try to use your code in a place that doesn't make sense.

Two suggestions to avoid ethical and professional issues in my software proposed in this assignment

Following the IEEE CS ethical guidelines I would give these two suggestions: - Follow the PRODUCT rule. This means ensuring that all products are up the highest professional standards. This ensures that there could be little to no room for error on the coding side and that you have held your ethical integrity. - Follow the COLLEAGUES rule. This means to be fair and helpful to any colleagues you may have. An example would be to be communicative about what you're working on, how it works and how to incorporate it into their work.

Discussion

On a final note, I belive my code is up to par for the requirements. In the future, with more time available, I potentially could've added more flexibility within the code. Something akin to catching every input error in a way that doesn't affect the menu system. Overall though, I am proud of the work and ingenuity that went into this assignment. Thank you for your time.