**qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm**

|  |
| --- |
| Project I  CSC1021  10/26/2013  Daniel Baranowski |

Contents

[How to use 2](#_Toc370564573)

[Testing input 3](#_Toc370564574)

[Normal Data 3](#_Toc370564575)

[Boundary Data 4](#_Toc370564576)

[Extreme Data 5](#_Toc370564577)

[Testing process and output 6](#_Toc370564578)

[Pass 6](#_Toc370564579)

[Compensable Fail 7](#_Toc370564580)

[Fail 8](#_Toc370564581)

[Stage: Pass 8](#_Toc370564582)

[Stage: Pass By Compensation 9](#_Toc370564583)

[Stage: Fail 10](#_Toc370564584)

[Code Listings 11](#_Toc370564585)

[Mark Calculator 11](#_Toc370564586)

[Summary 15](#_Toc370564587)

[StudentChart 18](#_Toc370564588)

# Bibliography

<http://bytes.com/topic/java/answers/541928-check-if-input-integer>

Marks Track Software for the School of Computer Science

The Marks Track is a simple java program designed to aid tracking marks for a single Stage 1 student. Marks Track allows the user to input student’s exam and coursework marks and will than compute module results and stage results according to the rules provided in the specification. Marks Track will also provide a graphical representation of the results in a form of coloured bar chart.

# How to use

This section of the documentation is meant to guide a potential user in using the software. It will also briefly explain the processes in the background. For further information please refer to the code listings section. The code is widely commented in order to aid future developers in the understanding.

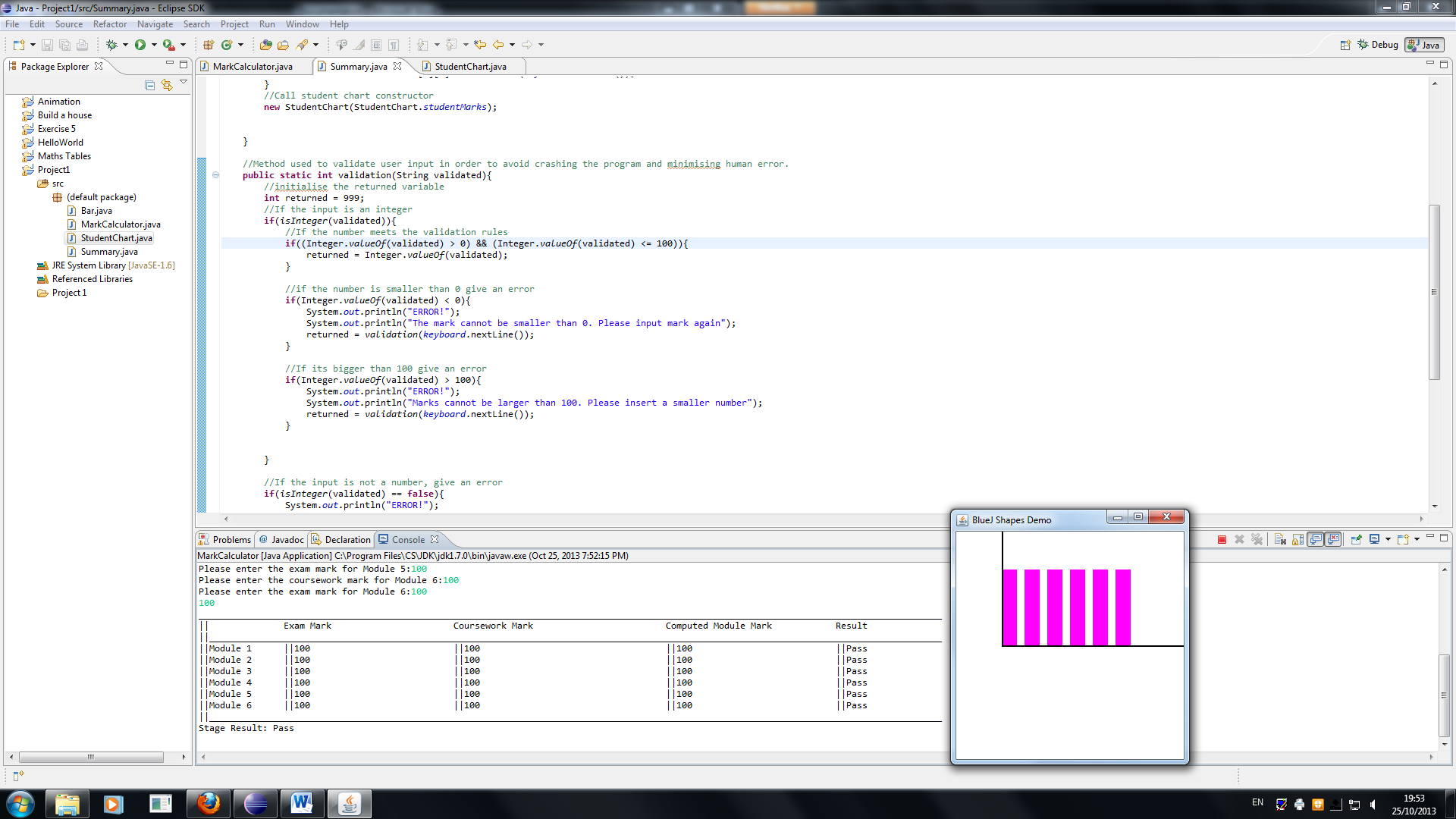
1. Input marks into the program – Once the program Is run It will ask the user to input marks for each module. The program will first ask for coursework mark and then for exam mark. In the table below you can see what number represents each module. The input is made by typing the number suing keyboard and accepted by hitting enter.

|  |  |
| --- | --- |
| Module 1 | Programming 1 |
| Module 2 | Programming 2 |
| Module 3 | The Software Engineering Professional |
| Module 4 | Computer Architecture |
| Module 5 | Mathematics for Computer Science |
| Module 6 | Website design and Construction |

In case of potential errors in the input the program will let the user know that there is a problem and ask the user to correct the error.

*Note – In order to make the program work with different modules, all that needs to be done is to change the values in the courseworkWeighting array inside MarkCalculator class.*

1. Processing – The program will than process the data. In order to get more information of how the data is processed please look into the specification or the code listings.
2. Output – Once the processing is done the program will than output the results in form of a table and a bar chart. The diagram on the next page will explain the output in more detail.



Results. The results computed by the program will be output in these columns. On the left the computed module mark for each module will be shown and on the right

Stage Result. Stage result is printed underneath the table.

Marks. The Exam marks and Coursework marks for each module will be shown in these two columns.

List of modules. Marks for each module are in the same row as the module name

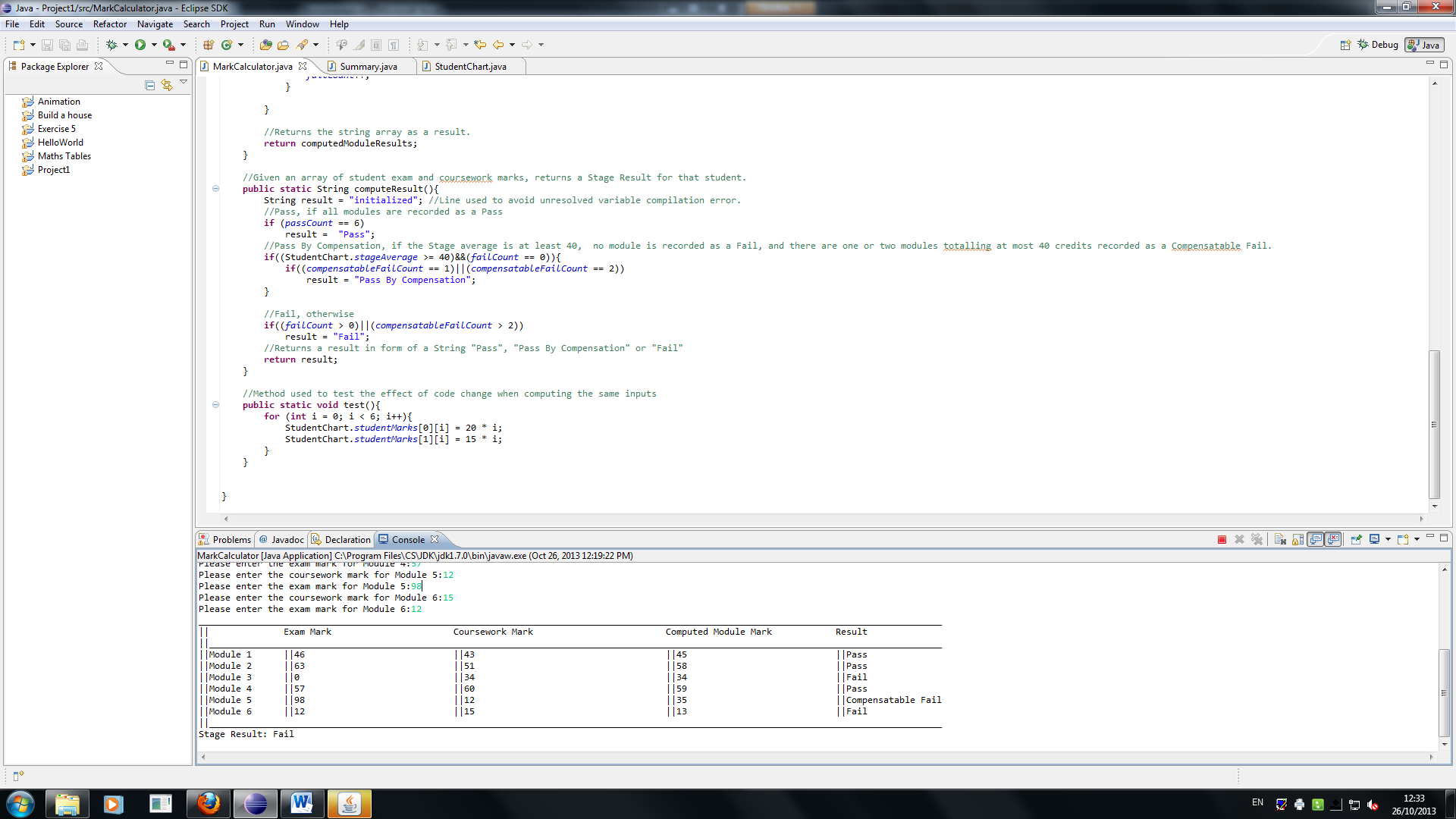
# Testing input

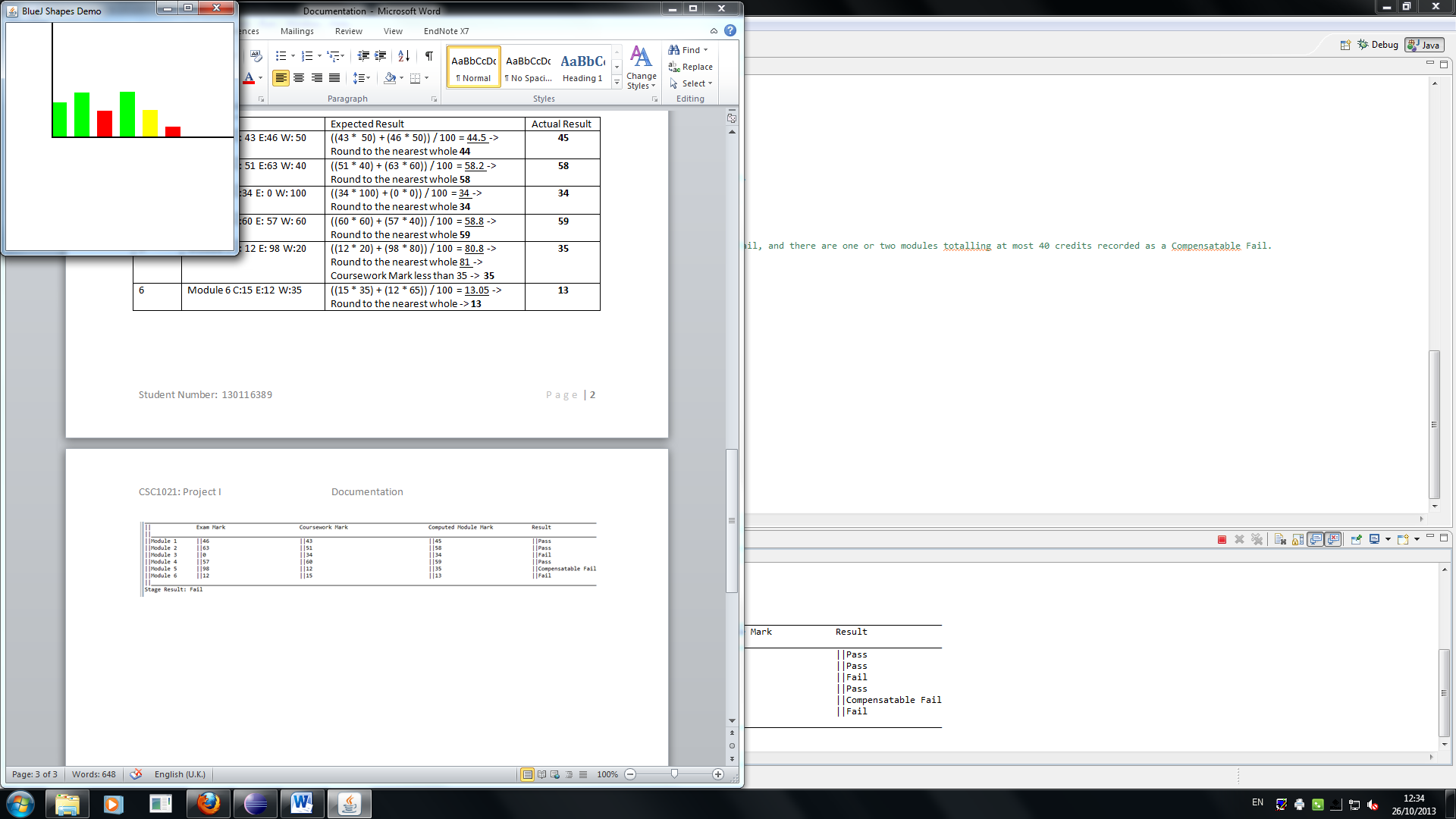
In this section of the document I will show how I tested the program and discuss why it works correctly.

## Normal Data

In this section I will test the program on normal data to see If the program computes expected results. “C” stands for coursework, “E” stands for exam marks and “W” stands for coursework weighting.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 1 | Module 1 C: 43 E:46 W: 50 | ((43 \* 50) + (46 \* 50)) / 100 = 44.5 -> Round to the nearest whole **44** | **45** |
| 2 | Module 2 C: 51 E:63 W: 40 | ((51 \* 40) + (63 \* 60)) / 100 = 58.2 -> Round to the nearest whole **58** | **58** |
| 3 | Module 3 C:34 E: 0 W: 100 | ((34 \* 100) + (0 \* 0)) / 100 = 34 ->  Round to the nearest whole **34** | **34** |
| 4 | Module 4 C:60 E: 57 W: 60 | ((60 \* 60) + (57 \* 40)) / 100 = 58.8 ->  Round to the nearest whole **59** | **59** |
| 5 | Module 5 C: 12 E: 98 W:20 | ((12 \* 20) + (98 \* 80)) / 100 = 80.8 ->  Round to the nearest whole 81 ->  Coursework Mark less than 35 -> **35** | **35** |
| 6 | Module 6 C:15 E:12 W:35 | ((15 \* 35) + (12 \* 65)) / 100 = 13.05 ->  Round to the nearest whole -> **13** | **13** |



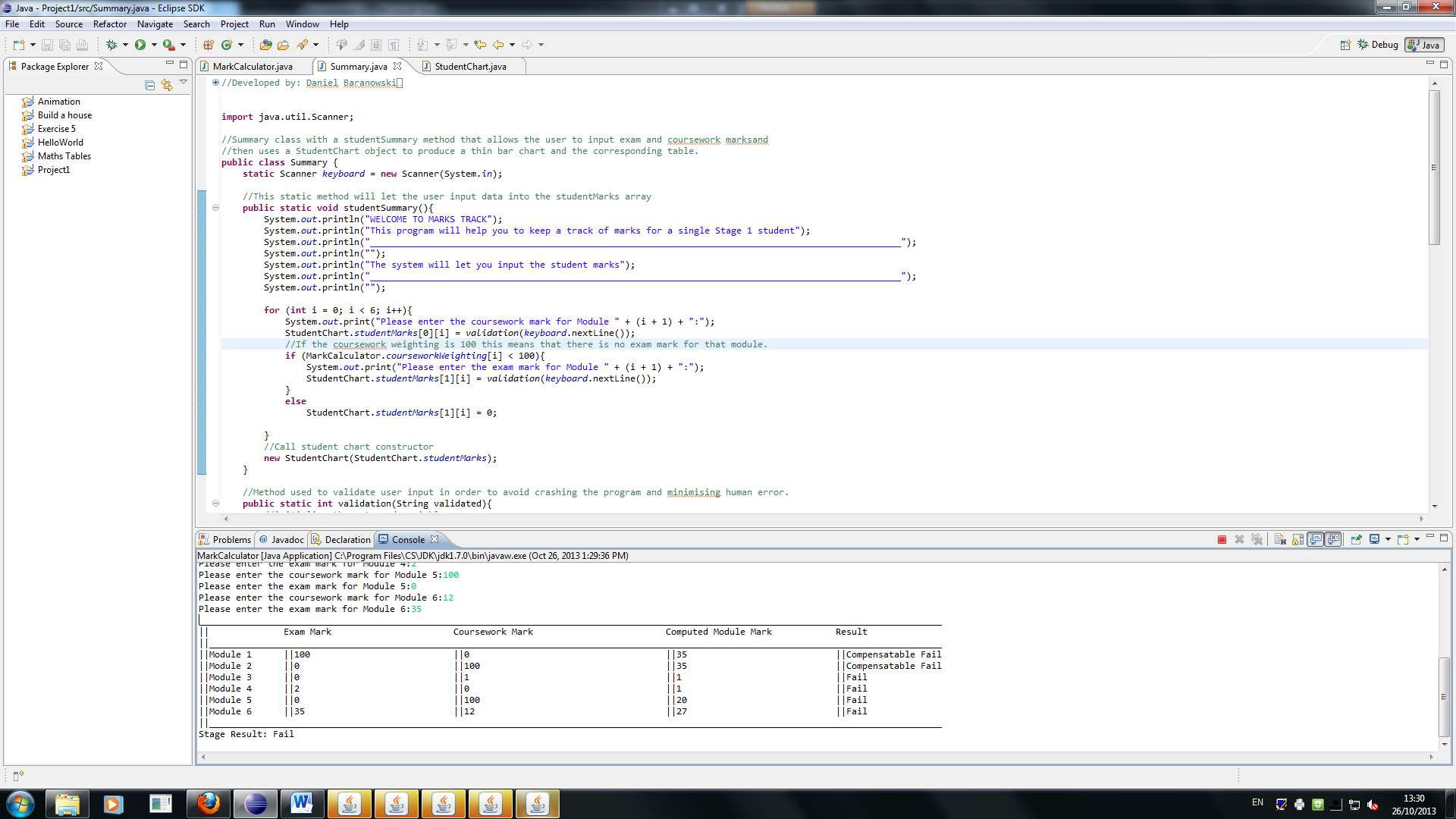


Testing on normal data was successful. The program didn’t cause any problems and returned expected results. I will continue testing of my program on some other inputs in order to see how it will react to possible human error.

## Boundary Data

In this section of the document I will show the testing of my program on boundary data.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 7 | Module 1 C: 0 E:100 W: 50 | ((0 \*50) + (100 \* 50)) / 100 = 50 ->  Coursework Mark less than 35 -> **35** | **35** |
| 8 | Module 2 C: 100 E:0 W: 40 | ((100 \* 40) + (0 \* 60)) / 100 = 40 ->  Exam Mark less than 35 -> **35** | **35** |
| 9 | Module 3 C:01 E: 0 W: 100 | ((1 \*100) + (0 \* 0)) / 100 = **1** | **1** |
| 10 | Module 4 C:00 E: 002 W: 60 | ((0 \* 60) + (2 \*40)) / 100 = -> 0.8 ->  Round to the nearest whole **1** | **1** |
| 11 | Module 5 C:0100 E:000 W:20 | ((100 \* 20) + (0 \* 80)) / 100 = 20 -> **20** | **20** |
| 12 | Module 6 C:0 E:12 W:35 | ((0 \* 35) + (12 \* 65)) / 100 = 7.8 ->  Round to the nearest whole -> 8 | **8** |

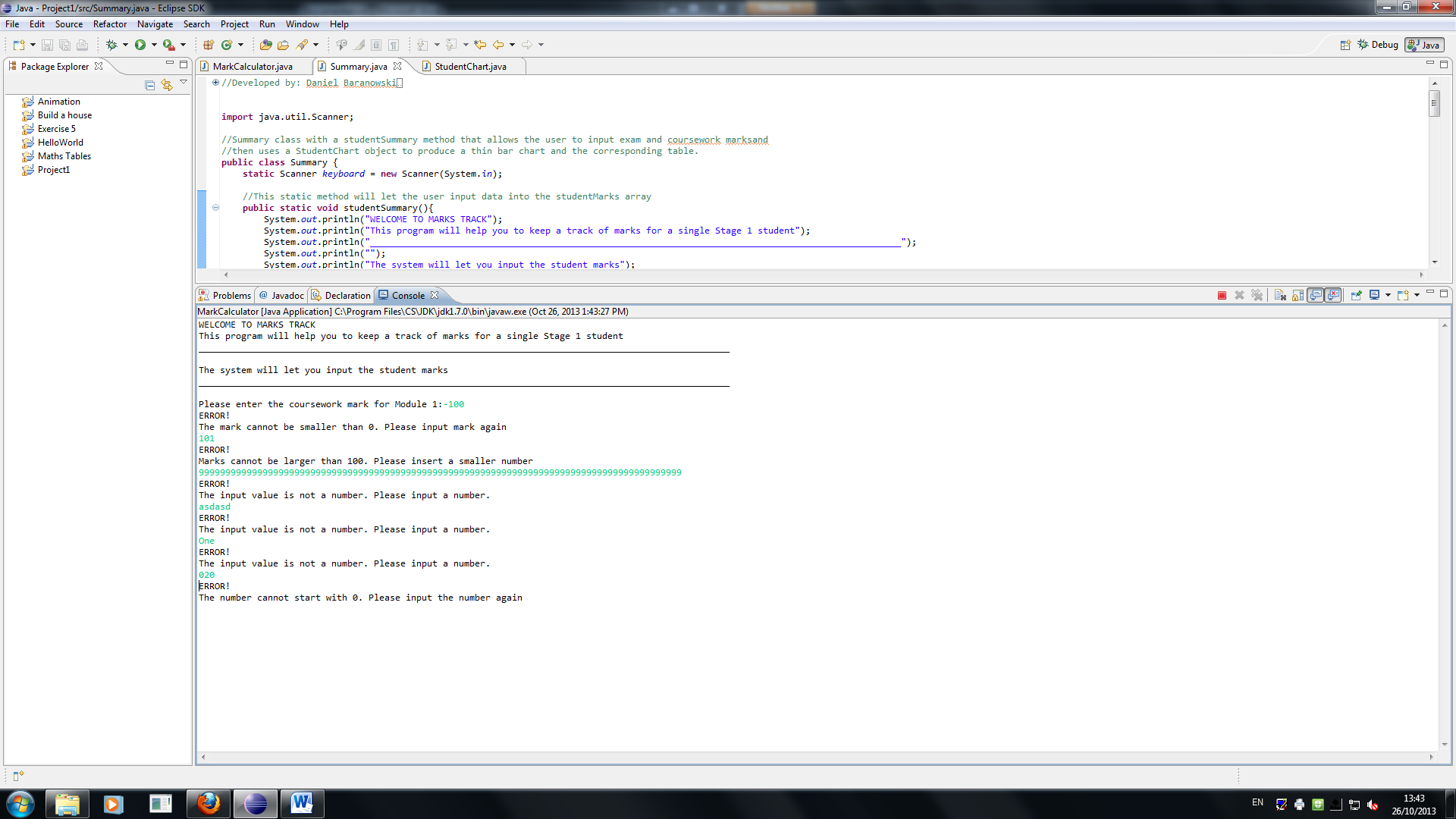


Testing on boundary data was successful the program ignored illogical inputs like “002” and converted them to normal numbers like “2”. This allowed the program to work correctly and calculate expected results. The only problem I can see this cause is for example when typing quickly a user might by mistake input for example “07” instead of “70” this might cause problems in the future. I will fix the problem and add an error that will force the user to retype the input if it begins with “0”.

## Extreme Data

In this section of the document I will show how I tested the program against extreme data.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 13 | -100 | Error | **Error** |
| 14 | 101 | Error | **Error** |
| 15 | 999999999999999999999999 | Error | **Error** |
| 16 | Asdasd | Error | **Error** |
| 17 | One | Error | **Error** |
| 18 | 020 | Error | **Error** |



The program didn’t allow the user to input inappropriate data. The program gave the user an error in order to inform him that the data he/she input is inappropriate and let the user try to input the data again. The process is repeated until user inputs appropriate data. This will help to stop users from entering wrong marks by mistake. Notice also that the numbers beginning with 0 are no longer available for input.

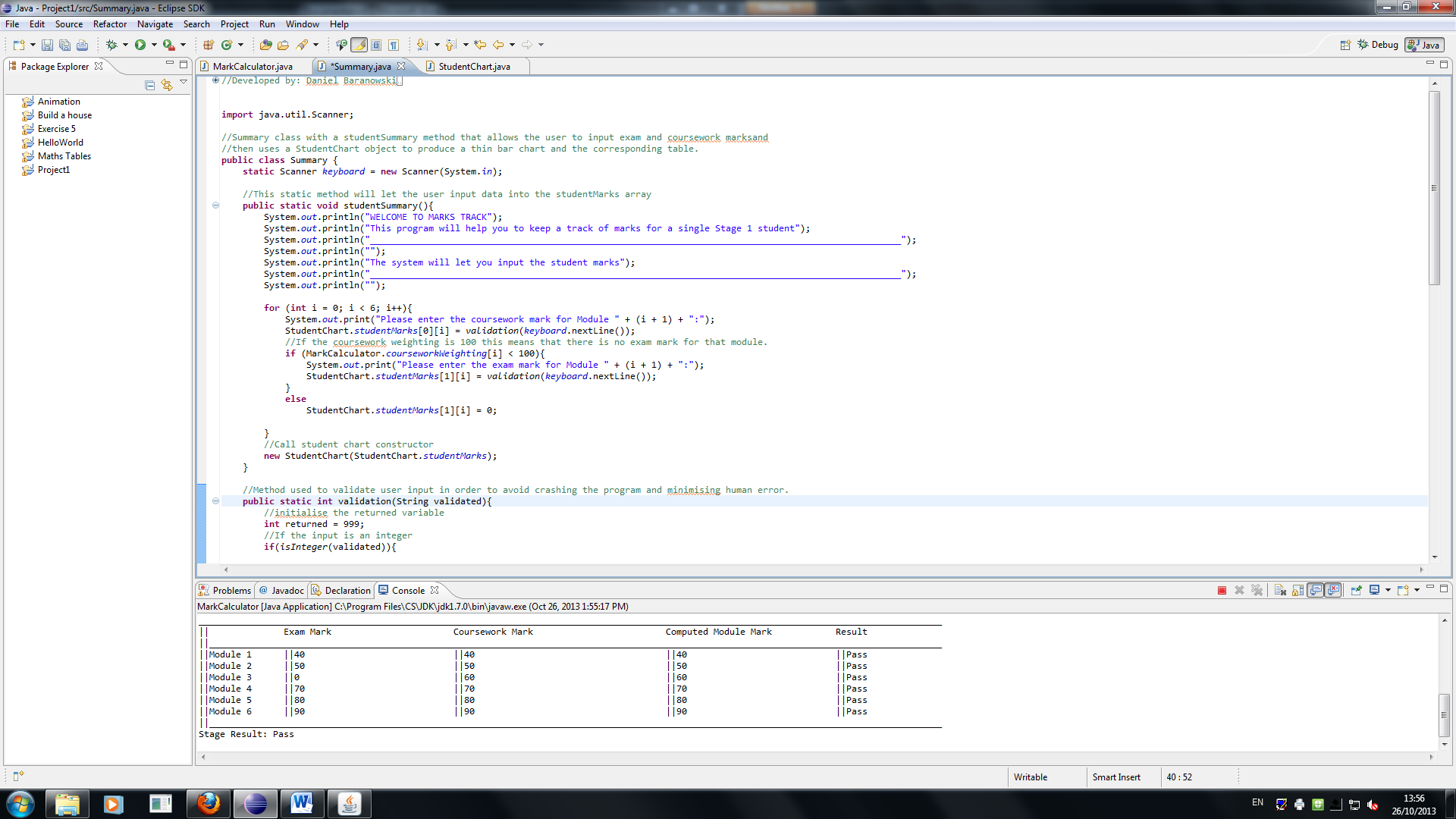
# Testing process and output

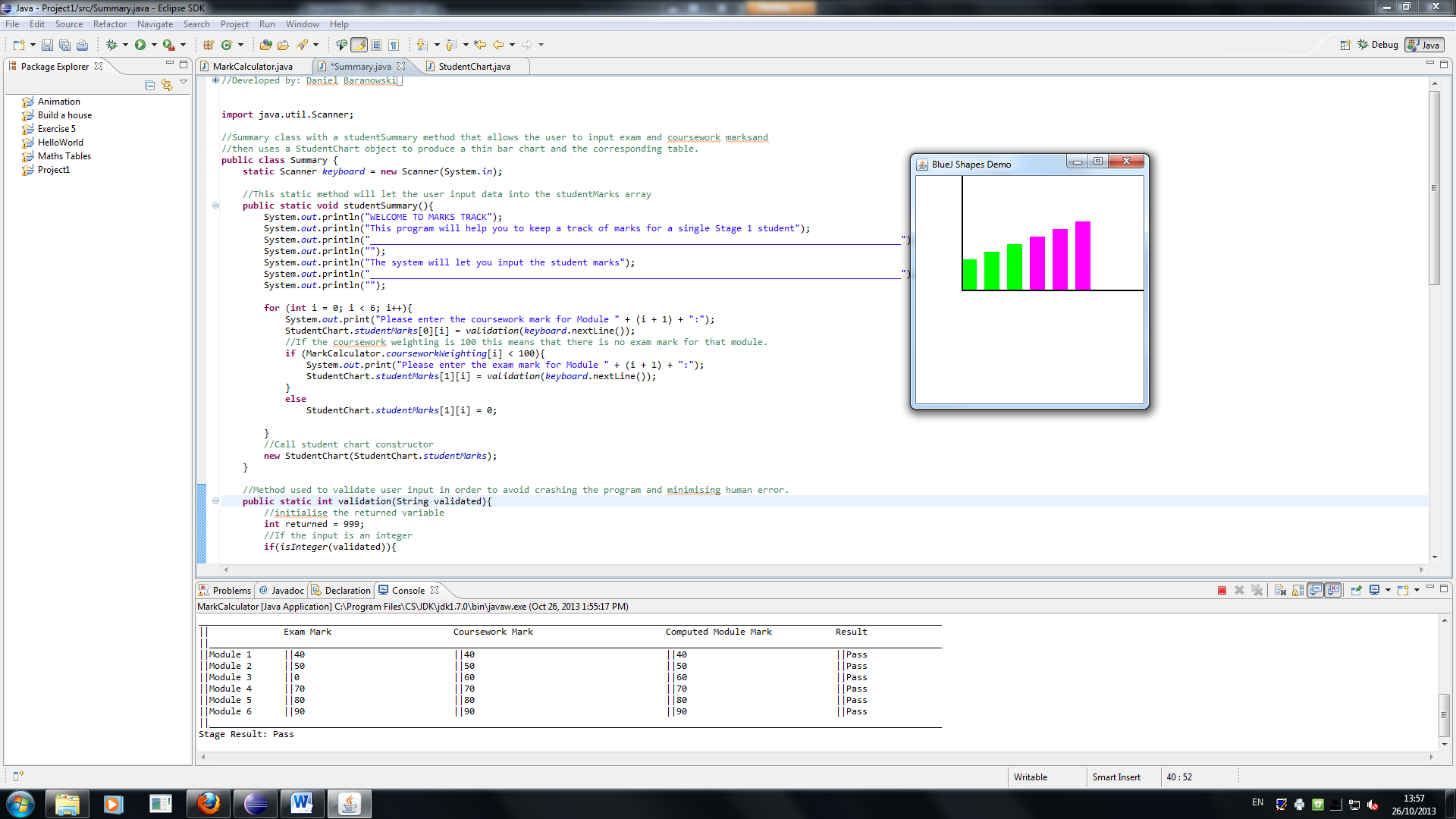
In the previous section I have tested the input process of the program in order to see if the program takes appropriate values for calculation and validates the user input. The testing has also shown that the calculated module marks are accurate and meet the requirements of the specification. In this section I will test the program in order to check if it computes the module results and stage results as required.

## Pass

I will perform some tests in order to see if my program can successfully compute pass module results.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 19 | Module 1 C: 40 E:40 W: 50 | **Pass – Green Bar** | **Pass – Green Bar** |
| 20 | Module 2 C: 50 E:50 W: 40 | **Pass – Green Bar** | **Pass – Green Bar** |
| 21 | Module 3 C:60 E:0 W: 100 | **Pass – Green Bar** | **Pass – Green Bar** |
| 22 | Module 4 C:70 E:70 W: 60 | **Pass – Magenta Bar** | **Pass – Magenta Bar** |
| 23 | Module 5 C:80 E:80 W:20 | **Pass – Magenta Bar** | **Pass – Magenta Bar** |
| 24 | Module 6 C:90 E:90 W:35 | **Pass – Magenta Bar** | **Pass – Magenta Bar** |





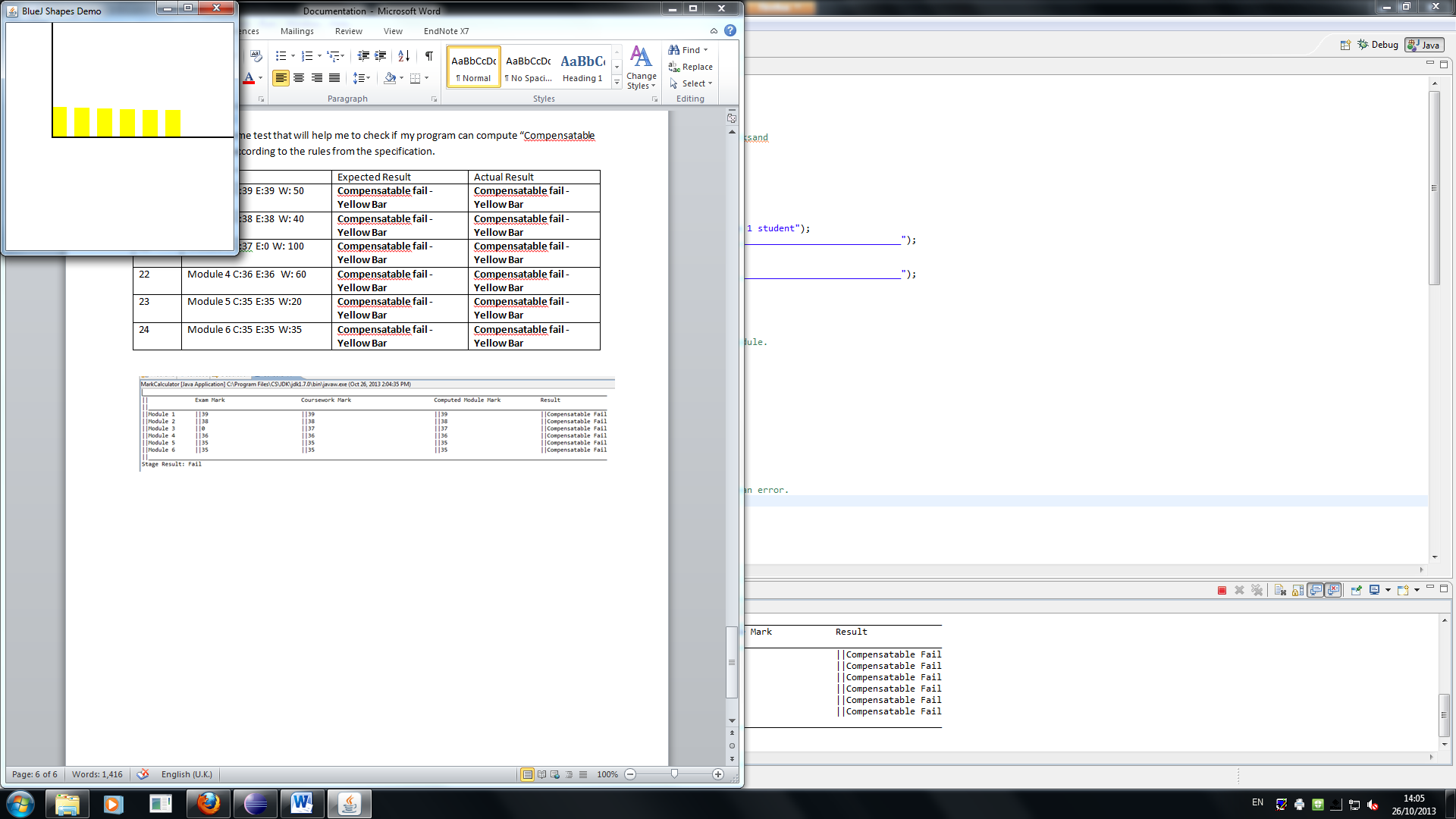
The results were as expected. According to these tests I know that my program can compute pass module results according to the rules from the specification.

## Compensable Fail

Now I will perform some test that will help me to check if my program can compute “Compensatable Fail” module results according to the rules from the specification.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 25 | Module 1 C:39 E:39 W: 50 | **Compensatable fail - Yellow Bar** | **Compensatable fail - Yellow Bar** |
| 26 | Module 2 C:38 E:38 W: 40 | **Compensatable fail - Yellow Bar** | **Compensatable fail - Yellow Bar** |
| 27 | Module 3 C:37 E:0 W: 100 | **Compensatable fail - Yellow Bar** | **Compensatable fail - Yellow Bar** |
| 28 | Module 4 C:36 E:36 W: 60 | **Compensatable fail - Yellow Bar** | **Compensatable fail - Yellow Bar** |
| 29 | Module 5 C:35 E:35 W:20 | **Compensatable fail - Yellow Bar** | **Compensatable fail - Yellow Bar** |
| 30 | Module 6 C:35 E:35 W:35 | **Compensatable fail - Yellow Bar** | **Compensatable fail - Yellow Bar** |



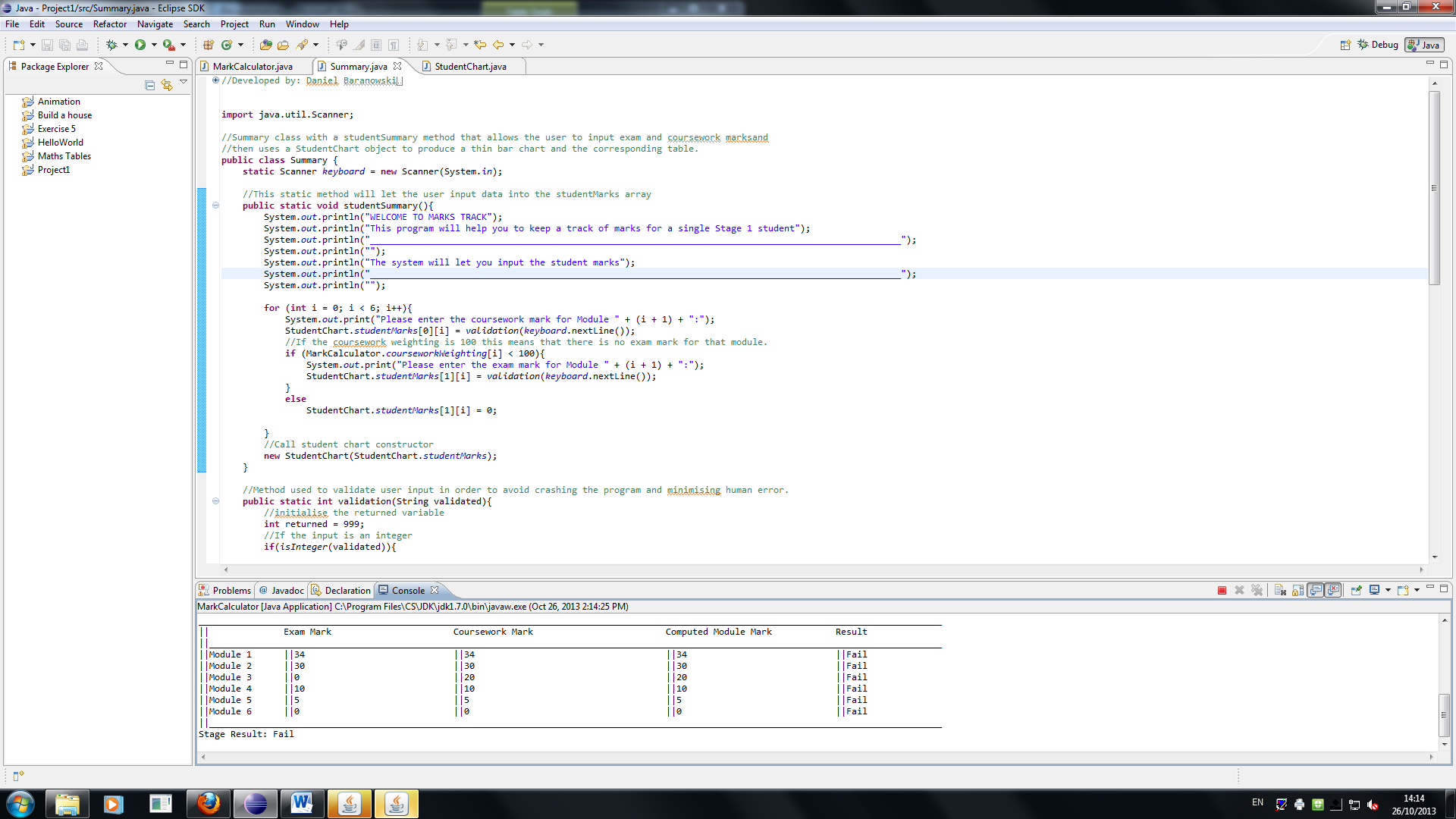


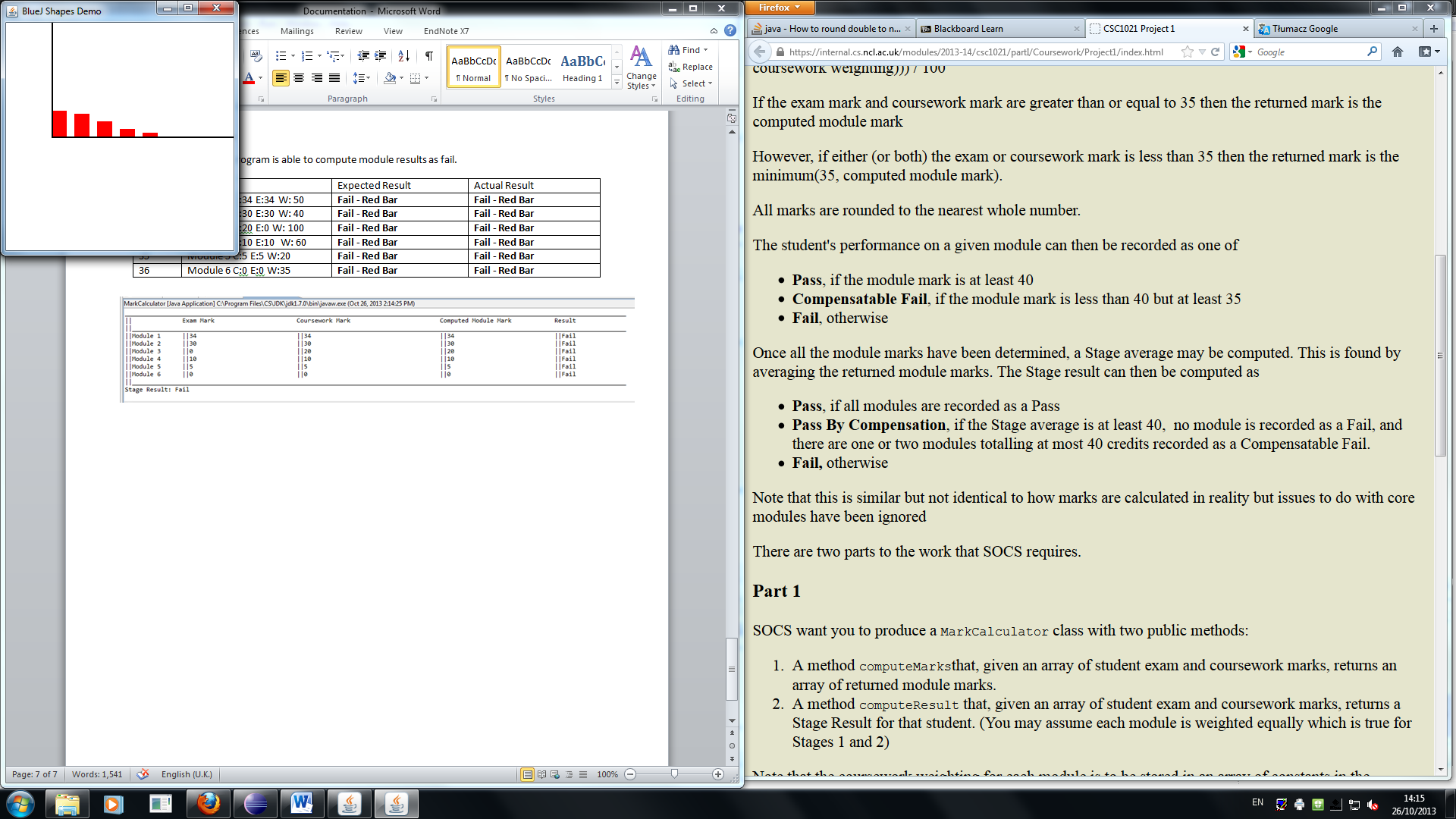
The program can successfully compute “Compensatable fails” according to the rules from the specification.

## Fail

Now I will test if my program is able to compute module results as fail.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 31 | Module 1 C:34 E:34 W: 50 | **Fail - Red Bar** | **Fail - Red Bar** |
| 32 | Module 2 C:30 E:30 W: 40 | **Fail - Red Bar** | **Fail - Red Bar** |
| 33 | Module 3 C:20 E:0 W: 100 | **Fail - Red Bar** | **Fail - Red Bar** |
| 34 | Module 4 C:10 E:10 W: 60 | **Fail - Red Bar** | **Fail - Red Bar** |
| 35 | Module 5 C:5 E:5 W:20 | **Fail - Red Bar** | **Fail - Red Bar** |
| 36 | Module 6 C:0 E:0 W:35 | **Fail - Red Bar** | **Fail - Red Bar** |



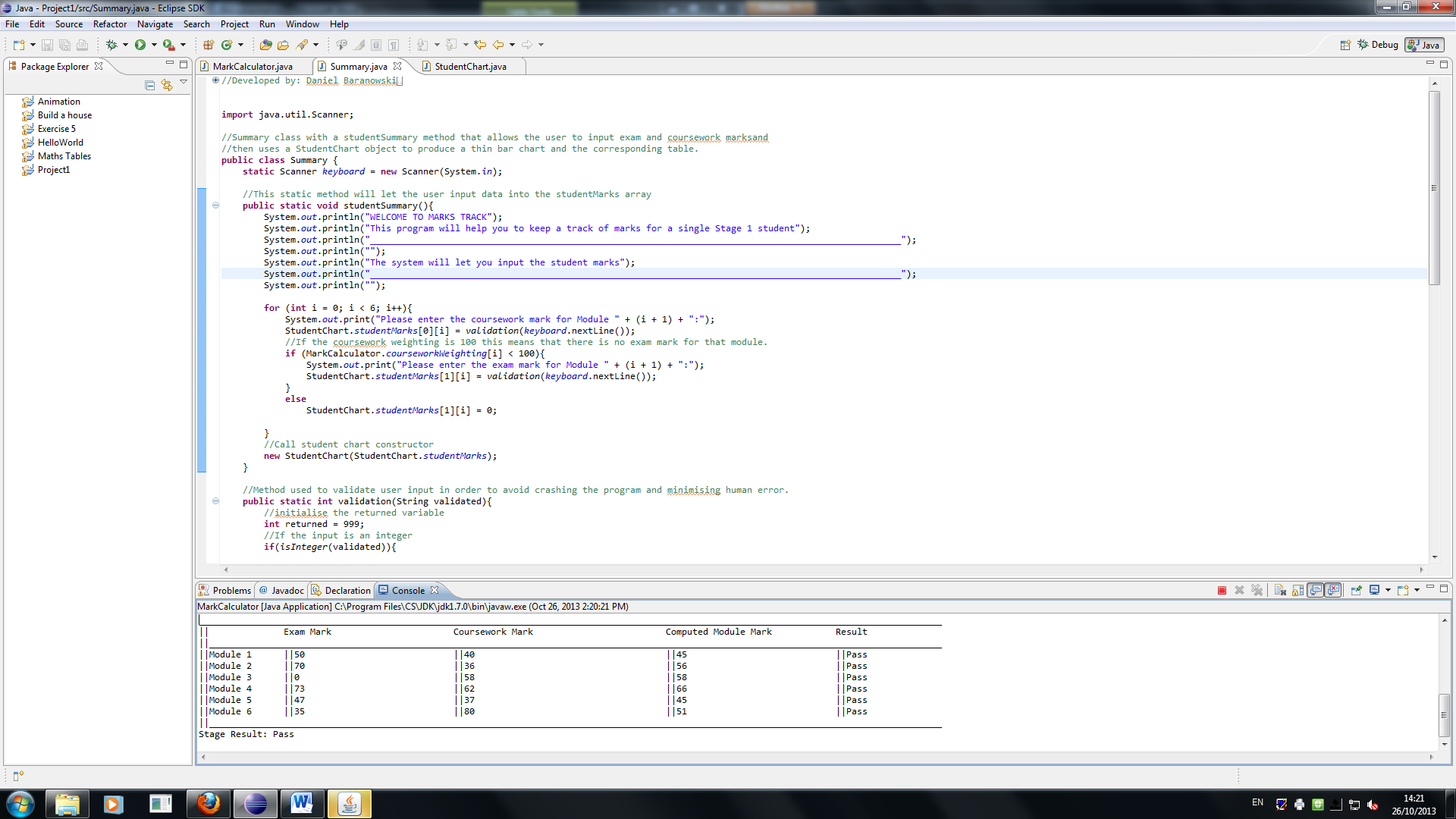


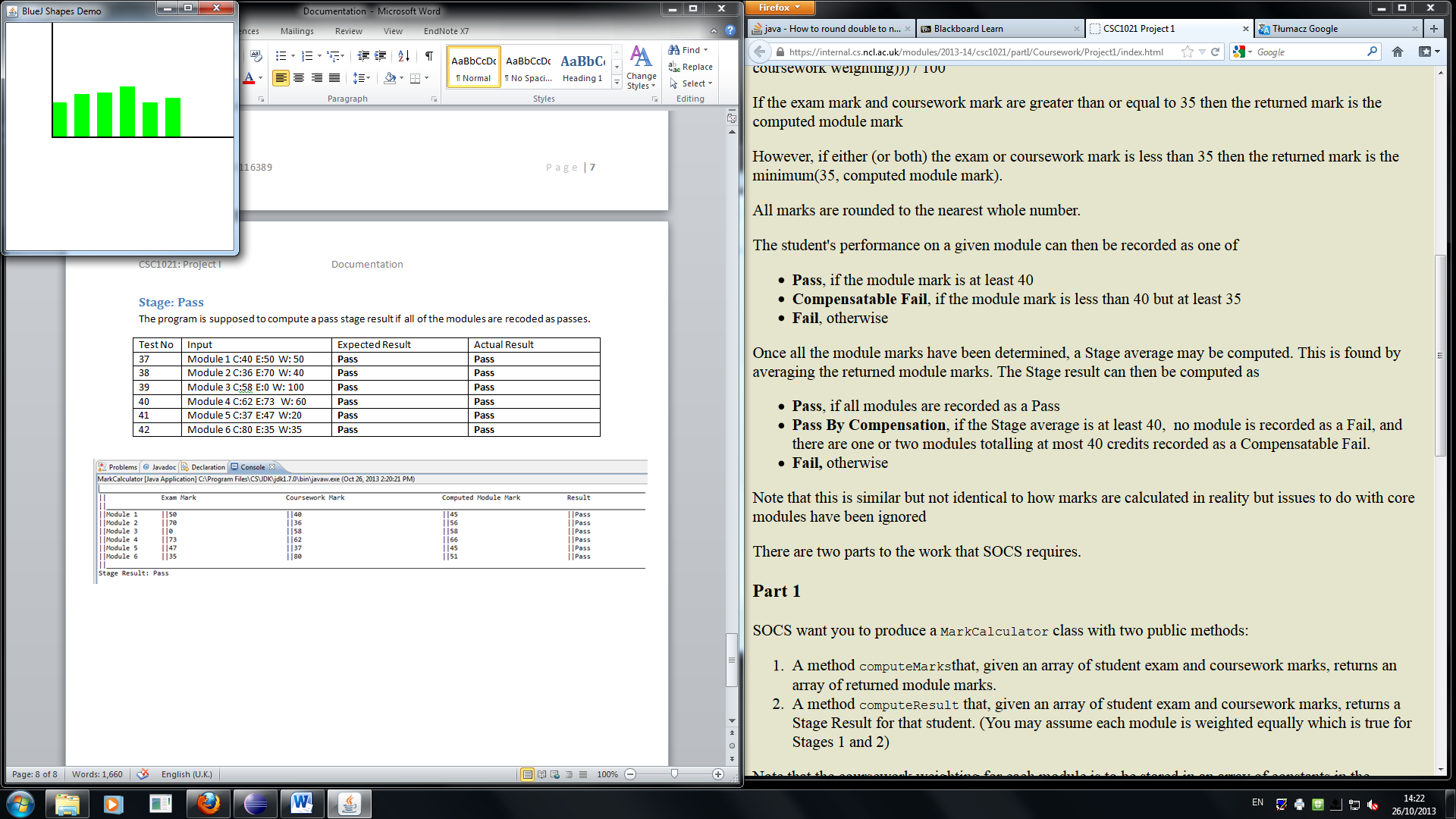
The program computed expected results which prove that it can successfully compute “fail” module results according to the rules from the specification. Next I will test if the program computes stage result according to the rules from the specification.

## Stage: Pass

The program is supposed to compute a pass stage result if all of the modules are recoded as passes.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 37 | Module 1 C:40 E:50 W: 50 | **Pass** | **Pass** |
| 38 | Module 2 C:36 E:70 W: 40 | **Pass** | **Pass** |
| 39 | Module 3 C:58 E:0 W: 100 | **Pass** | **Pass** |
| 40 | Module 4 C:62 E:73 W: 60 | **Pass** | **Pass** |
| 41 | Module 5 C:37 E:47 W:20 | **Pass** | **Pass** |
| 42 | Module 6 C:80 E:35 W:35 | **Pass** | **Pass** |





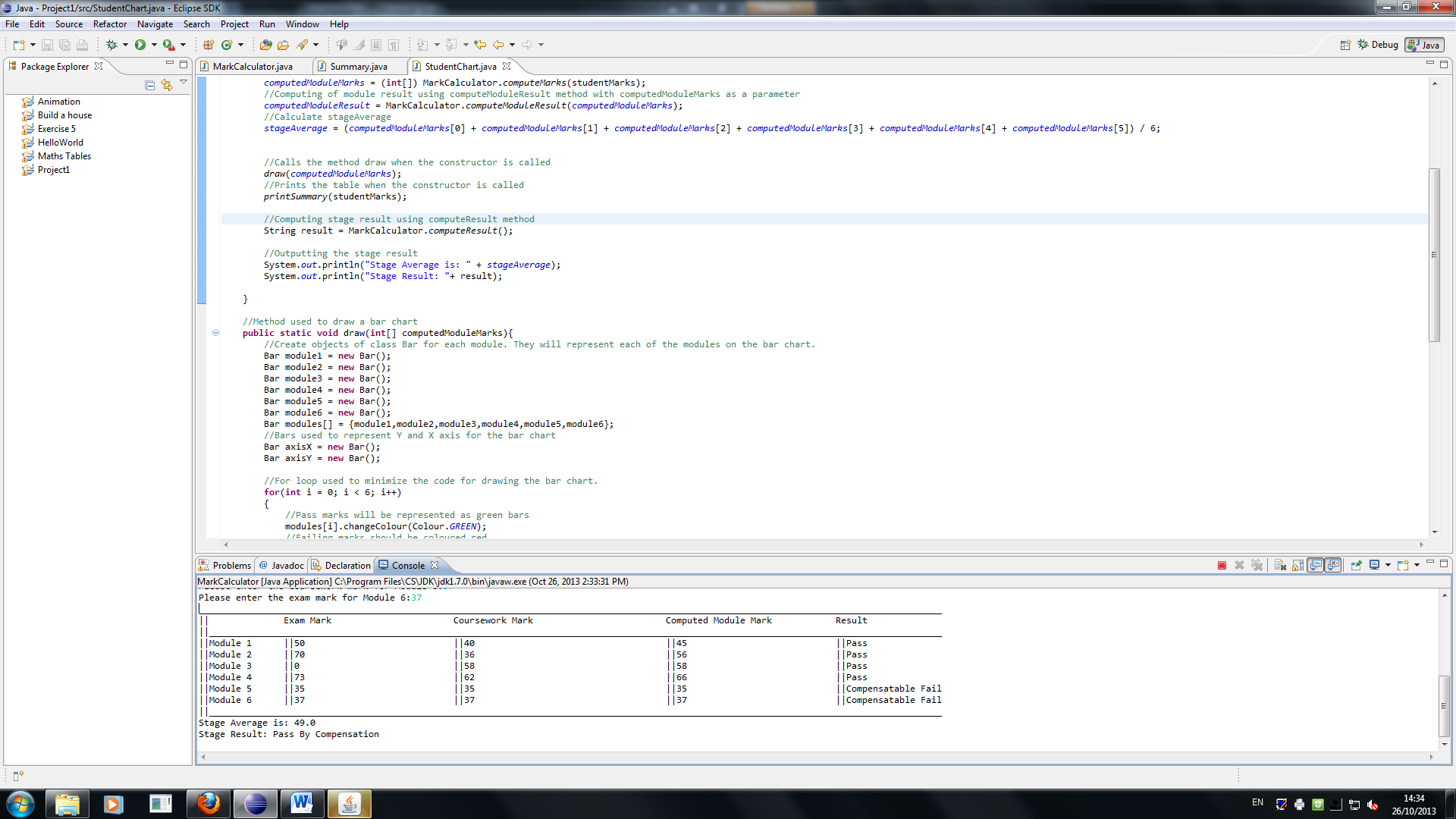
All of the module results have been recoded as Pass. This means that the program is expected to compute a Pass as a stage result. The program provides expected results which means that it is able to successfully compute pass stage result according to the rules from the specification.

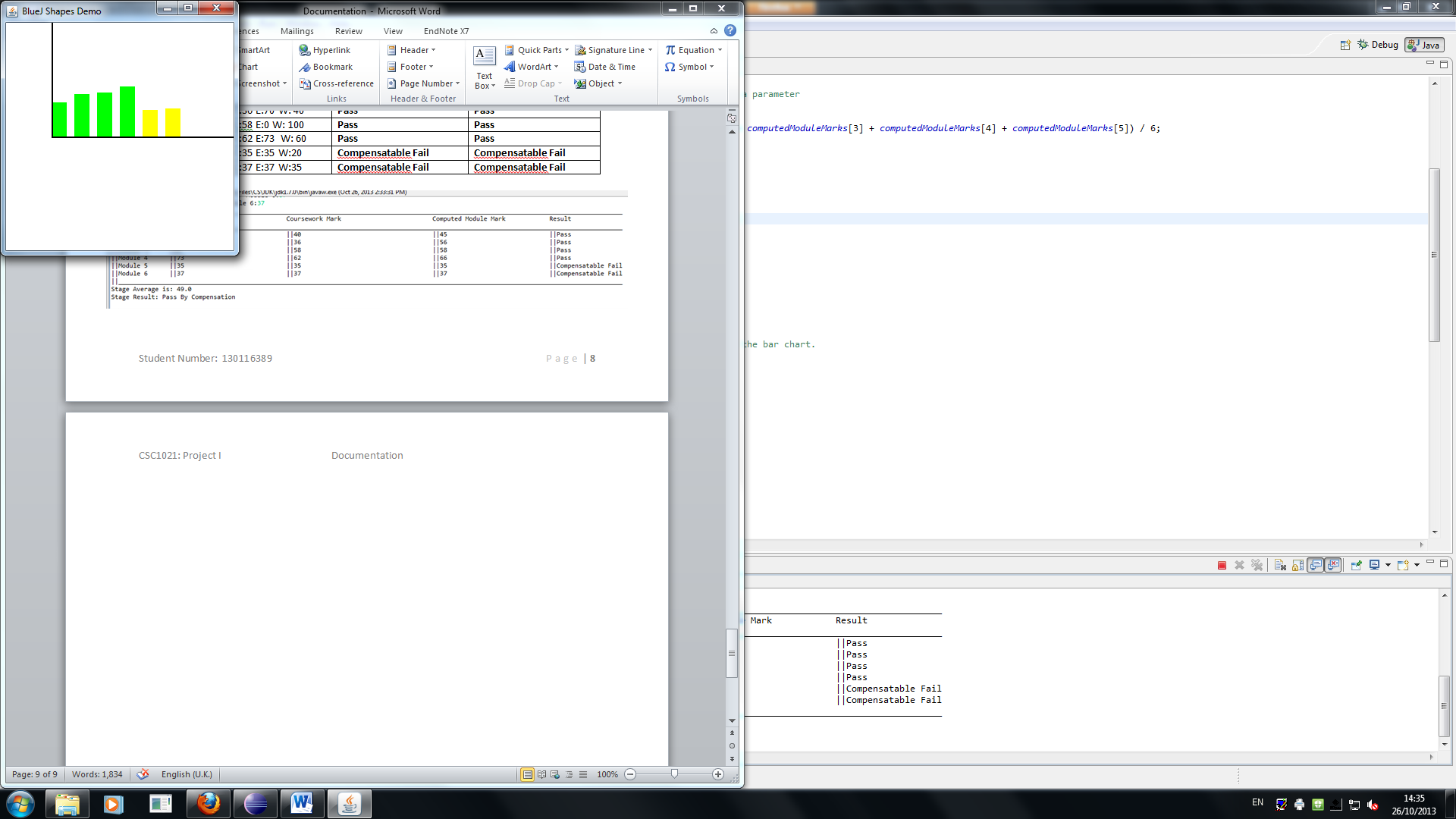
## Stage: Pass By Compensation

Next I will test if the program can successfully compute the stage result as “Pass by Compensation” according to the rules set by the specification.

*"If the stage average is at least 40, no module is recoded as a Fail, and there are one of two modules totalling at most 40 credits recoded as a Compensatable Fail. “*

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 43 | Module 1 C:40 E:50 W: 50 | **Pass** | **Pass** |
| 44 | Module 2 C:36 E:70 W: 40 | **Pass** | **Pass** |
| 45 | Module 3 C:58 E:0 W: 100 | **Pass** | **Pass** |
| 46 | Module 4 C:62 E:73 W: 60 | **Pass** | **Pass** |
| 47 | Module 5 C:35 E:35 W:20 | **Compensatable Fail** | **Compensatable Fail** |
| 48 | Module 6 C:37 E:37 W:35 | **Compensatable Fail** | **Compensatable Fail** |



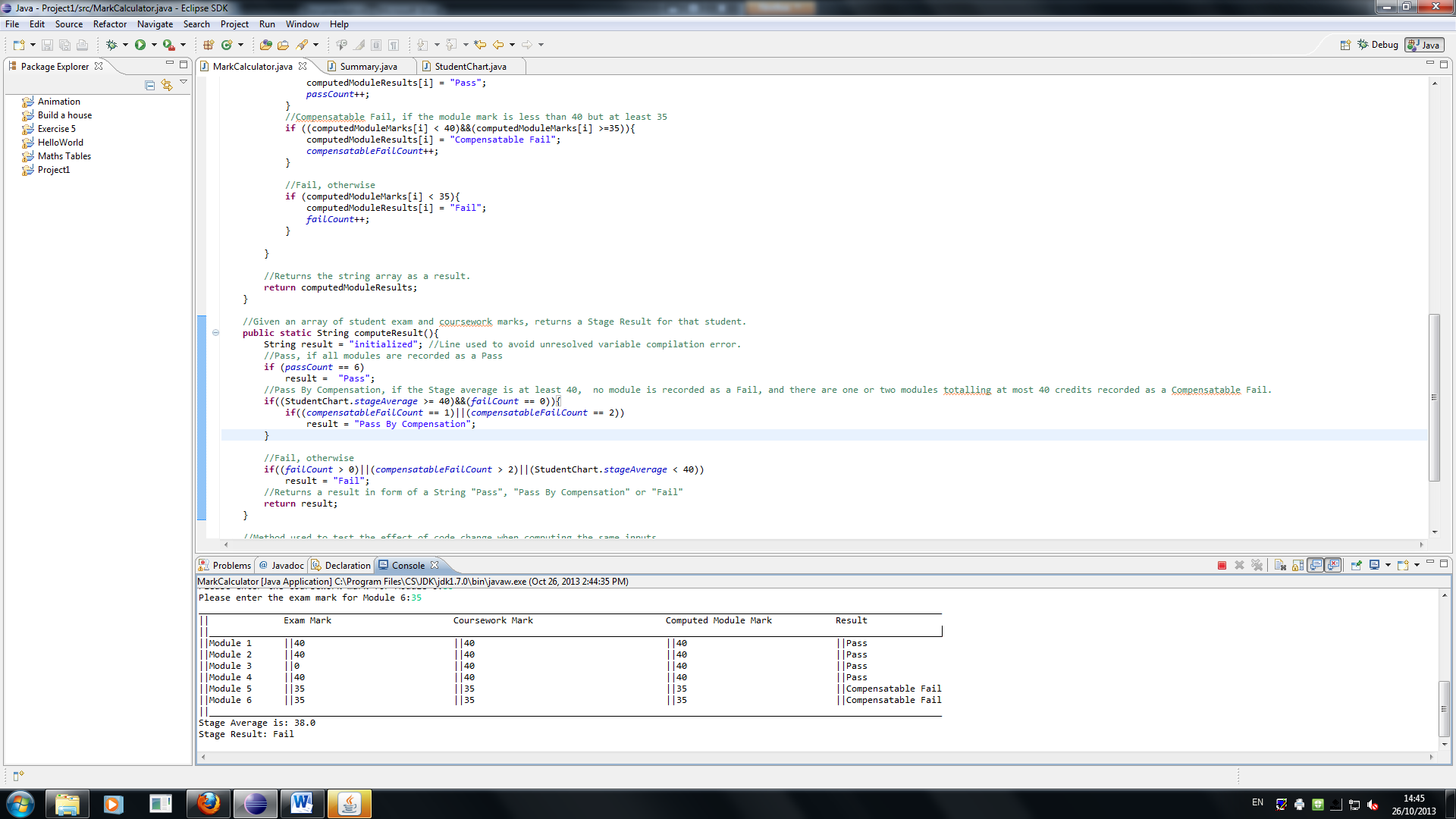


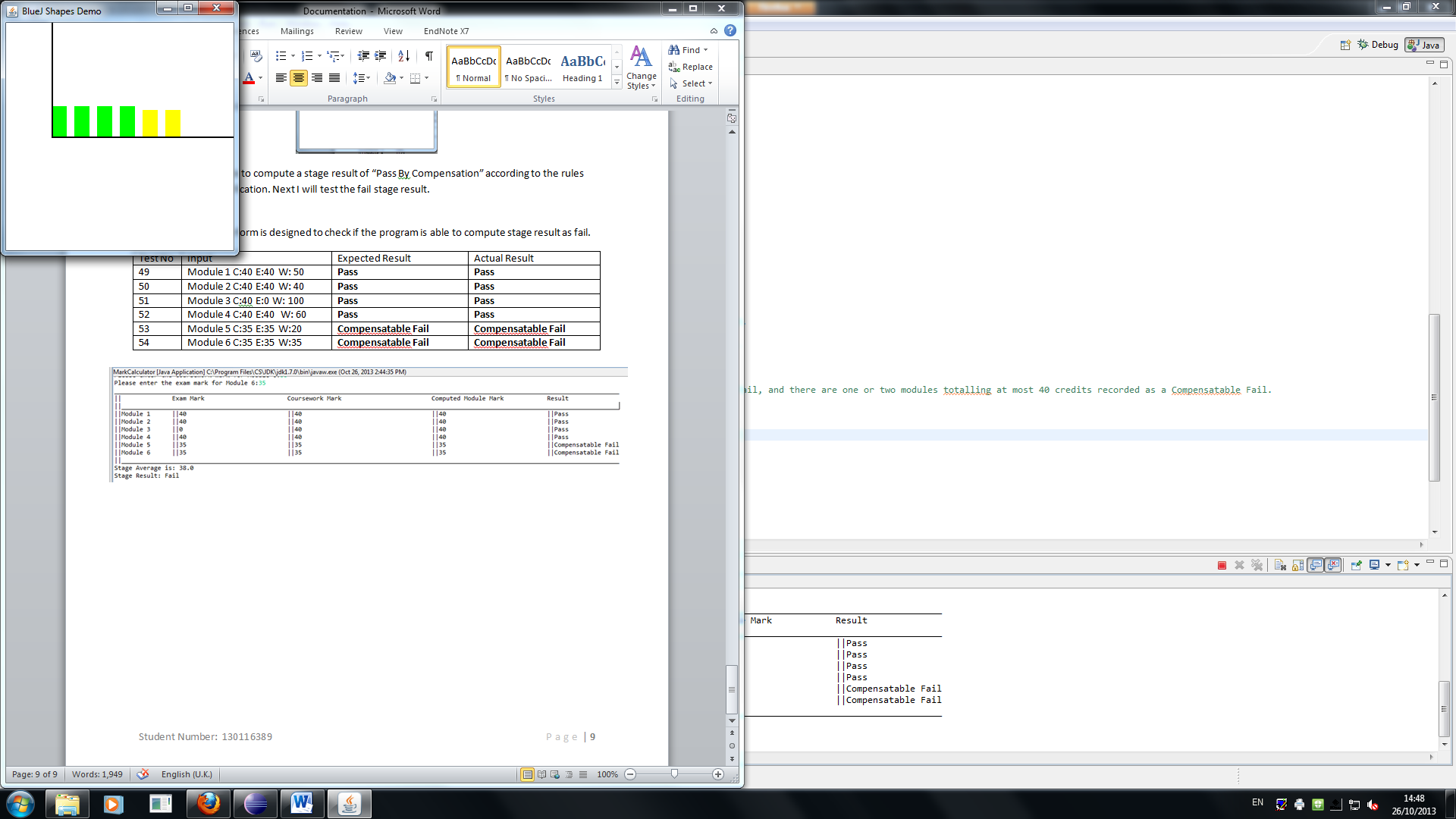
The program was able to compute a stage result of “Pass By Compensation” according to the rules provided in the specification. Next I will test the fail stage result.

## Stage: Fail

The last test I will perform is designed to check if the program is able to compute stage result as fail.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No | Input | Expected Result | Actual Result |
| 49 | Module 1 C:40 E:40 W: 50 | **Pass** | **Pass** |
| 50 | Module 2 C:40 E:40 W: 40 | **Pass** | **Pass** |
| 51 | Module 3 C:40 E:0 W: 100 | **Pass** | **Pass** |
| 52 | Module 4 C:40 E:40 W: 60 | **Pass** | **Pass** |
| 53 | Module 5 C:35 E:35 W:20 | **Compensatable Fail** | **Compensatable Fail** |
| 54 | Module 6 C:35 E:35 W:35 | **Compensatable Fail** | **Compensatable Fail** |





The program calculated the Stage Result as fail because the stage average was less than 40. Now I know that the program is able to compute stage fail results according to the rules from the specification. The next section of the document is code listings.