

Programming, Parts One and Two

by Kadmin Van vuuren

FILE	12020_KADMIN_VAN_VUUREN_PROGRAMMING__PARTS_ONE_AND_TW O_159704_1661616176.DOCX		
TIME SUBMITTED	10-JAN-2019 09:39AM (UTC+0000)	WORD COUNT	9574
SUBMISSION ID	98660594	CHARACTER COUNT	58493

2019

Programming & Software Testing

JAVA PROGRAMMING AND TESTING PARTS 1 & 2
KADMIN VAN VUUREN

DVLA UWTSD | 1809712

Table of Contents

TABLE OF CONTENTS	1
TABLE OF FIGURES	2
CHAPTER ONE: DVLA SOFTWARE TESTING TECHNIQUES	4
1. PART ONE: JUNIT TESTING	4
1.1 PART TWO: CUCUMBER & SELENIUM	5
1.2 PART THREE: GHERKIN	5
1.3 PART FOUR: DVLA CONCEPTUAL TESTING	6
CHAPTER TWO: PROGRAMMING EXERCISE TESTING	7
2. PART ONE: EXERCISE ONE AVERAGE TEST	7
2.1 PART TWO: EXERCISE ONE MIN TEST	8
2.2 PART THREE: EXERCISE ONE MAX TEST	9
2.3 PART FOUR: EXERCISE ONE PRINT USER ARRAY	10
2.4 PART FIVE: EXERCISE ONE INTERSECTION	11
CHAPTER THREE: PROGRAMMING EXERCISE & INVESTIGATION	14
3. PART ONE: HIGH COHESION	14
3.1 PART TWO: LOOSE COUPLING	15
3.2 PART THREE: DATA HIDING	16
3.3 PART FOUR: INFORMATION HIDING	17
3.4 PART FIVE: GIT REPO	18

3.5	PART SIX: EXERCISE TWO BUILDCONNECTBOARD	19
3.6	PART SEVEN: EXERCISE TWO VALUSERINPUTBOARDBOUNDS	21
3.7	PART EIGHT: EXERCISE TWO BLITZMOVE	22
3.8	PART NINE: EXERCISE TWO PLACETIMEBOMB	24
	CHAPTER ONE APPENDIX	29
	CHAPTER TWO APPENDIX	34

Table of Figures

FIGURE 1:	POM FILE XML	4
FIGURE 2:	GHERKIN	5
FIGURE 3:	DVLA CONCEPT TESTING ENVIRONMENT	6
FIGURE 4:	JUNIT TESTS AVERAGE	7
FIGURE 5:	JUNIT TESTS AVERAGE FAIL	8
FIGURE 6:	JUNIT TESTS AVERAGE PASS	8
FIGURE 7:	JUNIT TESTS MIN FAIL	8
FIGURE 8:	JUNIT TESTS MIN PASS	9
FIGURE 9:	JUNIT TESTS MAX FAIL	9
FIGURE 10:	JUNIT TESTS MAX PASS	9
FIGURE 11:	JUNIT TESTS PRINT USER ARRAY FAIL	10
FIGURE 12:	JUNIT TESTS PRINT USER ARRAY PASS	10
FIGURE 13:	JUNIT TESTS INTERSECTION FAIL	11
FIGURE 14:	JUNIT TESTS INTERSECTION REVISION	11
FIGURE 15:	JUNIT TESTS INTERSECTION PASS	12
FIGURE 16:	HIGH COHESION	14
FIGURE 17:	DATA HIDING	16
FIGURE 18:	GIT REPO	18
FIGURE 19:	GIT REPO	18
FIGURE 20:	JUNIT TESTS BUILDCONNECTBOARD FAIL	19
FIGURE 21:	JUNIT TESTS BUILDCONNECTBOARD FAIL	19
FIGURE 22:	JUNIT TESTS BUILDCONNECTBOARD PASS	20
FIGURE 23:	JUNIT TESTS VALUSERINPUTBOARDBOUNDS FAIL	21
FIGURE 24:	JUNIT TESTS VALUSERINPUTBOARDBOUNDS PASS	21
FIGURE 25:	JUNIT TESTS BLITZMOVE FAIL	22
FIGURE 26:	JUNIT TESTS BLITZMOVE PASS	22
FIGURE 27:	JUNIT TESTS BLITZMOVE FAIL	23
FIGURE 28:	JUNIT TESTS BLITZMOVE PASS	23
FIGURE 29:	JUNIT TESTS PLACETIMEBOMB FAIL	24
FIGURE 30:	JUNIT TESTS PLACETIMEBOMB PASS	24
FIGURE 31:	JUNIT TESTS PLACETIMEBOMB FAIL	25
FIGURE 32:	JUNIT TESTS PLACETIMEBOMB PASS	25

Introduction

This report will look to cover two main topics presented: Firstly, The Software Testing Processes and Techniques deployed at the DVLA and secondly look to present research and understanding into programming techniques such as High Cohesion, Loose Coupling, Data Hiding and Information Hiding.

Furthermore, this report will look to illustrate the unit testing performed on programming tasks covered throughout Part One and Two using techniques such Junit.

This Section Left Intentionally Blank

1

Chapter One: DVLA Software Testing Techniques

Throughout this chapter, a report will be presented on the topic of Software Testing Processes and Techniques deployed at the DVLA. As well as this, illustrations and discussion around Unit Testing will be presented to outline the testing processes performed throughout the programming exercise.

The DVLA provide a wide array of technical services to the public, all of which are created using many different techniques. This can be seen via their front-end Ruby Services, all of which interact with a backend primarily written in Java.

As such, there is a requirement for the utilisation of different testing techniques. The tools that we would use to test a front-end service may differ to that of a service which sits in the backend: This is important to outline as it sets the premise for the different tools and techniques used for testing at the DVLA.

1. Part One: JUnit Testing

When performing JUnit tests at the DVLA, Maven is used: Apache Maven is a software project management and comprehension tool [2]. By creating Maven based projects we can ensure that our project structure and dependencies are accurate. This is possible through its use of POM files, such as:

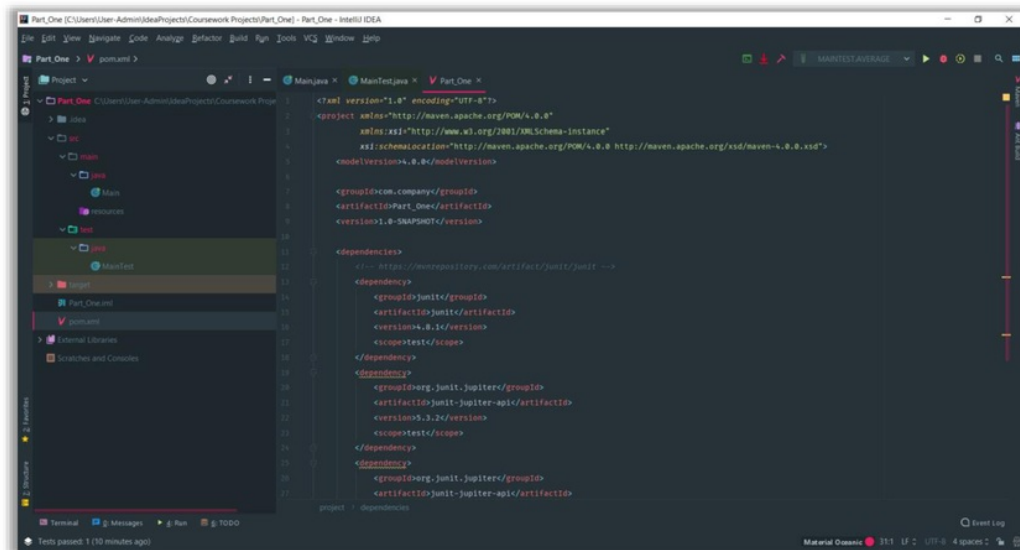


Figure 1: POM File | XML

Figure 1 illustrates the POM file which is created through the use of XML. This allows us to create a list of dependencies required for our application, such as dependencies for JUnit without the need to further manually update the projects structure.

Furthermore, at the DVLA developers will use JUnit for unit tests so to test features within an application such as methods that add new functionality. When performing unit tests for Ruby based application, RSpec is used.

1.1 Part Two: Cucumber & Selenium

The DVLA utilise features offered by that of Cucumber and Selenium for their web application testing. Cucumber is used to write integration tests and acceptance tests. There are a number of reasons to make use of testing when concerned with web application development, deployment and maintenance: Ensuring that a web application works and will continue to work across a set of supported browsers is an essential part of the web application development lifecycle [3].

There are a number of benefits to being able to automate a testing suite to run tests against your web services, such as detecting problems early and being able to obtain continuous feedback from tests.

1.2 Part Three: Gherkin

The DVLA also utilise Gherkin to write their ruby test cases. Below is an example of writing test cases during an exercise at the DVLA.

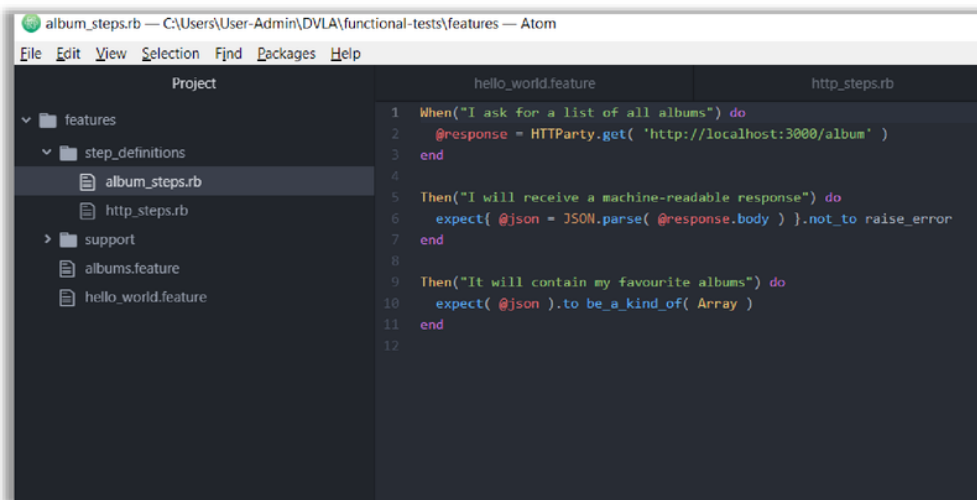


Figure 2: Gherkin

1.3 Part Four: DVLA Conceptual Testing

Below illustrates the process by which the DVLA conduct their testing

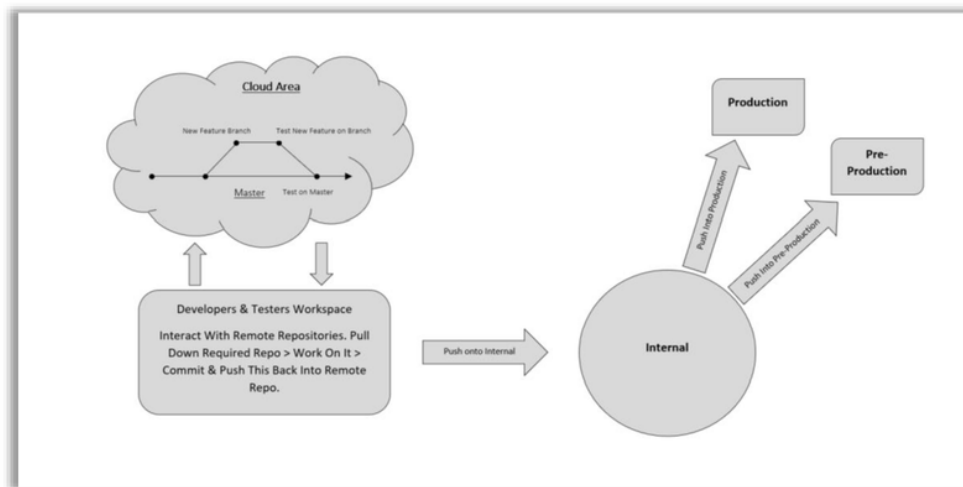


Figure 3: DVLA Concept Testing Environment

At the top of this illustration we can see their cloud area: It is here that any code repositories are stored. Staff can access this from their working areas to further create new feature testing branches while pulling the remote repo on their local workspace. Once these new features have been tested and are satisfactory, branches are then merged back into the Master.

Furthermore, once new features have been tested and implemented, they are pushed onto an internal repository. These are further pushed into pre-production or production depending on the nature of the application. Finally, at the DVLA we can see a multitude testing that is performed, such as:

Test Type	Description
Automation Tests	Automation Tests are heavily used at the DCLA to automate the process of testing. Manual testing is never done.
Static Tests	When Static Testing, the DVLA read documents from a different perspective to make sure it makes sense.
UI Tests	DVLA perform UI Testing to ensure that visual fields work as intended.
Acceptance Tests	During Acceptance Testing, the acceptance criteria is written first and is otherwise known as the three amigos, such as Developer, Tester and Business Analyst or BA.
Integration Tests	Such as JUnit are performed to test backend code such as the DVLA's Java built systems. These integration tests are performed to test the integration of new features onto core systems.
Load Tests	During load testing we prepare the system for expected loads in terms of the amount of requests that the system will be handling.
Stress Tests	When performing Stress Testing we deliberately push to fail.

2

Chapter Two: Programming Exercise Testing

This part of the report will look to illustrate the JUnit Testing performed throughout the first exercise. Importantly when performing unit testing we are primarily looking at creating the test before the code functionality. We want to test to fail at first, and then look to write the code to make the test pass.

2. Part One: Exercise One | Average Test

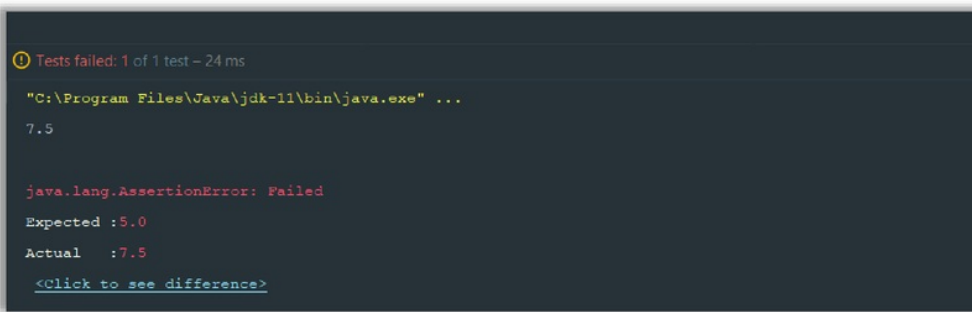
During the programming exercises there were a number of methods which performed separate functions. The average method allowed users to determine the average of both sets of numbers.



```
69      @Test
70      public void average()
71      {
72          double[] arr = new double[2];
73
74          double[] testSet = {5.0, 10.0};
75
76          double expected = 5.0;
77          double actual = Main.average(testSet);
78
79          assertEquals( message: "Failed", expected, actual, delta: 0);
80      }
81
```

Figure 4: JUnit Tests | Average

The above will first fail as the array only holds two numbers, added together we should have a value of 15, with the average being 7.5. However, we want our expected value to return an average of 10, as such once the test has failed we will ensure that the actual parameters return the expected value.



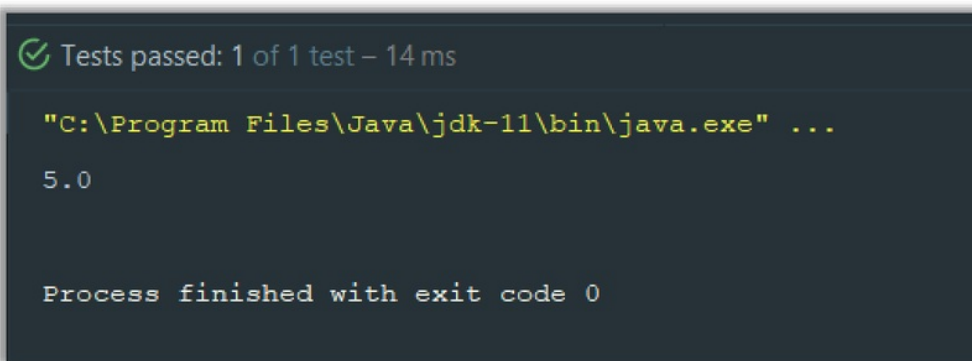
```
Tests failed: 1 of 1 test - 24 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...
7.5

java.lang.AssertionError: Failed
Expected :5.0
Actual :7.5
<Click to see difference>
```

Figure 5: JUnit Tests | Average Fail

Above we can see the test has expectedly failed. Let's now make it pass:



```
Tests passed: 1 of 1 test - 14 ms

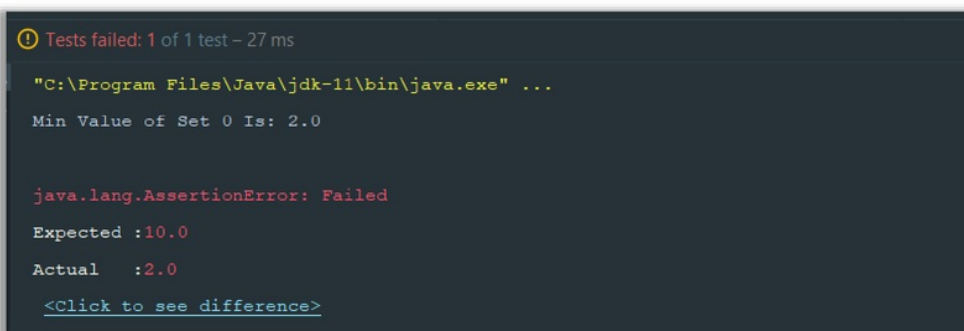
"C:\Program Files\Java\jdk-11\bin\java.exe" ...
5.0

Process finished with exit code 0
```

Figure 6: JUnit Tests | Average Pass

2.1 Part Two: Exercise One | Min Test

Method to determine the Minimum Value within a number set.



```
Tests failed: 1 of 1 test - 27 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...
Min Value of Set 0 Is: 2.0

java.lang.AssertionError: Failed
Expected :10.0
Actual :2.0
<Click to see difference>
```

Figure 7: JUnit Tests | Min Fail

```
✓ Tests passed: 1 of 1 test – 12 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

Min Value of Set 0 Is: 10.0

Process finished with exit code 0
```

Figure 8: JUnit Tests | Min Pass

2.2 Part Three: Exercise One | Max Test

```
"C:\Program Files\Java\jdk-11\bin\java.exe" ...

Max Value of Set 0 is: 25.0

java.lang.AssertionError: Failed
Expected :20.0
Actual   :25.0
<Click to see difference>
```

Figure 9: JUnit Tests | Max Fail

```
✓ Tests passed: 1 of 1 test – 16 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

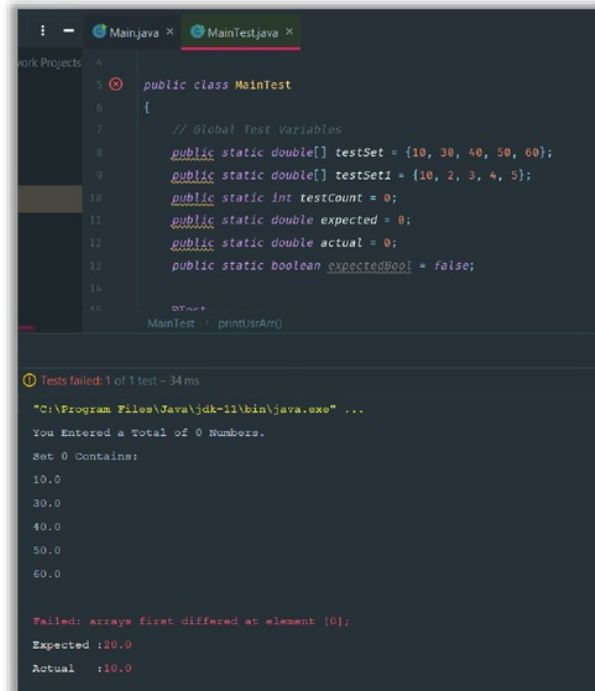
Max Value of Set 0 is: 60.0

Process finished with exit code 0
```

Figure 10: JUnit Tests | Max Pass

2.3 Part Four: Exercise One | Print User Array

Method to produce the user arrays.



```
4  
5 public class MainTest  
6 {  
7     // Global Test Variables  
8     public static double[] testSet = {10, 30, 40, 50, 60};  
9     public static double[] testSet1 = {10, 2, 3, 4, 5};  
10    public static int testCount = 0;  
11    public static double expected = 0;  
12    public static double actual = 0;  
13    public static boolean expectedBool = false;  
14  
15    @Test  
16    MainTest > printUserArr()  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

Tests failed: 1 of 1 test - 34 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

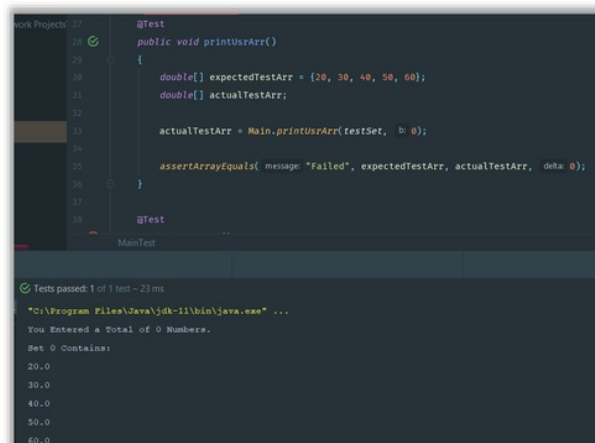
You Entered a Total of 0 Numbers.

Set 0 Contains:

10.0
30.0
40.0
50.0
60.0

Failed: arrays first differed at element [0]:
Expected :20.0
Actual :10.0

Figure 11: JUnit Tests | Print User Array Fail



```
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

Tests passed: 1 of 1 test - 23 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

You Entered a Total of 0 Numbers.

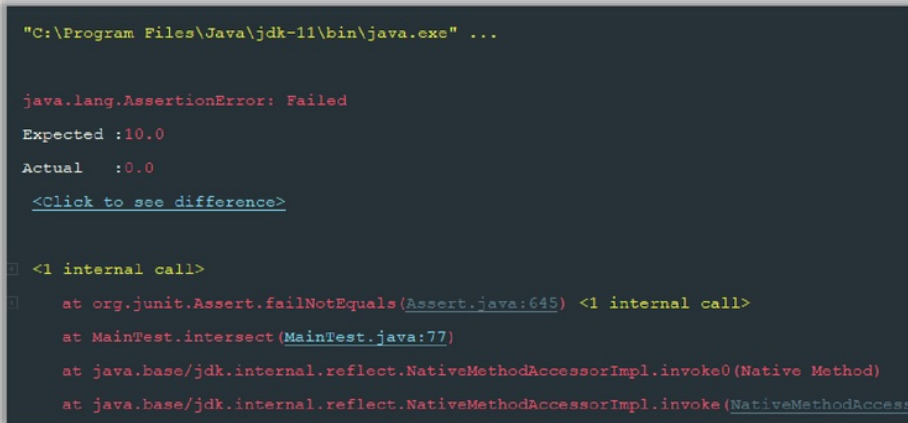
Set 0 Contains:

20.0
30.0
40.0
50.0
60.0

Figure 12: JUnit Tests | Print User Array Pass

2.4 Part Five: Exercise One | Intersection

Method to determine if there is an intersection that exists between sets.

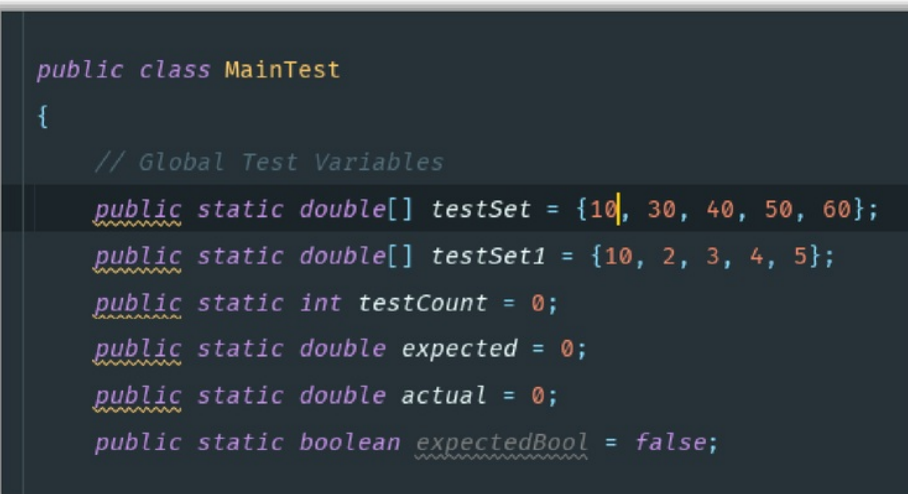


```
"C:\Program Files\Java\jdk-11\bin\java.exe" ...

java.lang.AssertionError: Failed
Expected :10.0
Actual   :0.0
<Click to see difference>

<1 internal call>
at org.junit.Assert.failNotEquals(Assert.java:645) <1 internal call>
at MainTest.intersect(MainTest.java:77)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccess
```

Figure 13: JUnit Tests | Intersection Fail



```
public class MainTest
{
    // Global Test Variables
    public static double[] testSet = {10, 30, 40, 50, 60};
    public static double[] testSet1 = {10, 2, 3, 4, 5};
    public static int testCount = 0;
    public static double expected = 0;
    public static double actual = 0;
    public static boolean expectedBool = false;
```

Figure 14: JUnit Tests | Intersection Revision

Revision made for passing.

```
✔ Tests passed: 1 of 1 test – 14 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

Matches Have Been Found

Intersection Value of 10.0 Has Been Found in Both Sets.

Process finished with exit code 0
```

Figure 15: JUnit Tests | Intersection Pass

This Page Left Intentionally Blank

3

Chapter Three: Programming Exercise & Investigation

Throughout this chapter research into High Cohesion, Loose Coupling, Data Hiding and Information Hiding will be conducted and reported upon. Furthermore, illustrations and discussion around Unit Testing will be presented to outline the testing processes performed throughout programming exercise.

3. Part One: High Cohesion

High Cohesion is a programming concept which suggests that a Class performs a “Well Defined Job”. In other words, we have a Class which performs that task that it sets out to achieve and only that [1].

So, we can say that an application that is well designed is one with that of High Cohesive Classes: Importantly, these Classes are designed towards specific and specialized task. This is also described as, Cohesion is used to indicate the degree to which a class has a single, well-focused purpose [9]. This can further be illustrated as a small one-part application that sees a Method of type Calculator return a subtraction of two numbers.

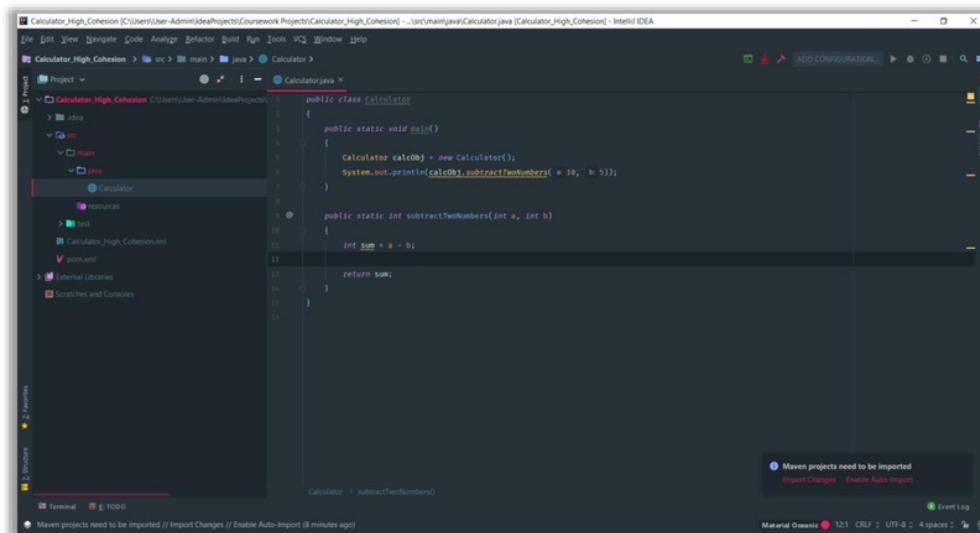


Figure 16: High Cohesion

Here we simply have an application which performs High Cohesion as its methods only perform one specific task: That of subtracting two numbers. Although, arbitrary this provides an insight into how we may ensure our code is of High Cohesion.

3.1 Part Two: Loose Coupling

Loose Coupling is a concept which is easy to understand, but difficult to code and detect. When two objects can interact with each other but have very limited knowledge about what each object can do [4], this is loose coupling. Importantly, we can see that Coupling is described as, Coupling is the degree to which one class knows about another class [9].

There are several advantages to enforcing loose coupling, such as:

- **Improved Testability:** As noted above, as your code is not dependent on other Objects, this makes unit testing easier to write.
- **Improved Readability:** When we are able to decouple our code and achieve loose coupling, we gain improved readability.

Importantly we can note that Loose Coupling is often achieved using Interfaces. Interestingly, after research into the subject matter, one can see that, the Java language and JVM are only very loosely coupled [8].

This Section Left Intentionally Blank

3.2 Part Three: Data Hiding

An important concept of Object-Oriented Programming is that of Data Hiding. Data Hiding is a way by which a developer can protect their application data from the “Outside World”, or rather protects its “State” from any interference.

Furthermore, data hiding is a process by which we hide the data within the class, making it available only through the methods [7].

Looking at the above, we can simply explain that when wanting to enforce Data Hiding, we are looking to “Hide” unimportant details from the user: User being that of a Software Developer, and only showing them relevant data. As such, an analogy of the brakes within a car: When we press our brakes, we know that they are there, and they should perform in a certain way, we don’t however know how this is working behind the scenes.

Now if we consider this example in programming, Data Hiding is performed with the help of Access Modifiers, such as the Public and Private Access Modifier. Here we can see Private Members being inaccessible from outside of the Class in which they were created: Furthermore, the Public Members are only accessible to the users, such as other developers.

Finally, the most important reason is to hide the internal implementation details of your class [7].

```
1  package com.company;
2
3  public class Data_Hiding
4  {
5      // Example of Violation of Data Hiding Rules
6      public int height;
7
8      // Here we can set a Private Member - This will not be accessible
9      // outside of Data_Hiding Class
10     private double weight;
11
12
13     // Here we can "Realise" Data Hiding Through Get and Set Methods
14     // And Gain Access to our Private weight Member
15     public double GetWeight()
16     {
17         return weight;
18     }
19
20     public double SetWeight(double weight)
21     {
22         if (weight >= 50)
23         {
24             this.weight = weight;
25         }
26         return this.weight;
27     }
28 }
```

Figure 17: Data Hiding

3.3 Part Four: Information Hiding

We can see the introduction of Information Hiding by that of David Parnas around 1972. He argued that the primary criteria for system modularization should concern the hiding of critical design decisions [10].

Similar to that of Data Hiding, importantly, Information Hiding can be used to help prevent the accidental or intentional change to parts of a program [5]. Importantly we can see Information Hiding as the process by which we can hide the critical design decisions within our code. Furthermore, modern Information Hiding together with subtyping creates situations where both an abstract variable and one or more of its dependencies are visible, but the associated representation is not visible [6].

It is important to understand that Information Hiding is not Encapsulation, and we should not confuse one for the other. Unlike Encapsulation, which is an Object Oriented Programming fundamental concept, Information Hiding is a mechanism by which we restrict the access to critical parts of an objects components.

We would want to use this as there are, as suggested, critical parts to a system that we do not want to provide access to, where this access could make changes to the way in which the object functions. One way in which we can do this is by ensuring that we make our objects components Private Members over Public Members.

3.4 Part Five: GIT Repo

Below illustrates the creating of the GIT repo for Part Two. This gives a brief overview of the process involved with working with a remote and local repository and using GIT adds, pushes, clone's commits and other commands such as status and init.

```
User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
$ git clone https://mightyboks@bitbucket.org/mightyboks/programming_part_two.git
Cloning into 'programming_part_two'...
warning: You appear to have cloned an empty repository.

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
$ ls
programming_part_two/

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
$ cd programming_part_two/

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ git init
Reinitialized existing Git repository in C:/Users/User-Admin/Documents/GITRepo/Programming_Software_Testing/Part_Two/programming_part_two/.git/

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ ls -a
./ ../ .git/

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ |
```

Figure 18: Git Repo

```
User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ git add Main.java MainTest.java

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   Main.java
        new file:   MainTest.java

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ git commit Main.java MainTest.java -m 'First commit for Part Two of the coursework. Contains skeleton base from which to work on.'
[master (root-commit) 18ba4f1] First commit for Part Two of the coursework. Contains skeleton base from which to work on.
 2 files changed, 29 insertions(+)
 create mode 100644 Main.java
 create mode 100644 MainTest.java

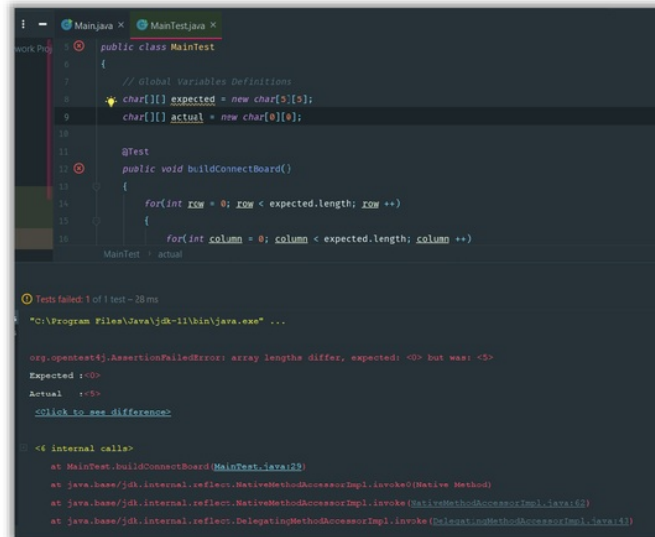
User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 548 bytes | 548.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://bitbucket.org/mightyboks/programming_part_two.git
 * [new branch]      master -> master

User-Admin@DESKTOP-L18PTMA MINGW64 ~/Documents/GITRepo/Programming_Software_Testing/Part_Two
/programming_part_two (master)
$ |
```

Figure 19: Git Repo

3.5 Part Six: Exercise Two | buildConnectBoard

The first test was to ensure that the board was being built strictly to the size parameters passed through the Global Board variable & the expected character placeholder found within the boards array elements.



```
1 public class MainTest
2 {
3     // Global variables Definitions
4     char[][] expected = new char[5][5];
5     char[][] actual = new char[0][0];
6
7     @Test
8     public void buildConnectBoard()
9     {
10         for(int row = 0; row < expected.length; row++)
11         {
12             for(int column = 0; column < expected.length; column++)
13             {
14                 MainTest actual
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Tests failed: 1 of 1 test - 28 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

org.opentest4j.AssertionFailedError: array lengths differ, expected: <0> but was: <5>

Expected :<0>

Actual :<5>

[Click to see difference](#)

<Internal call>

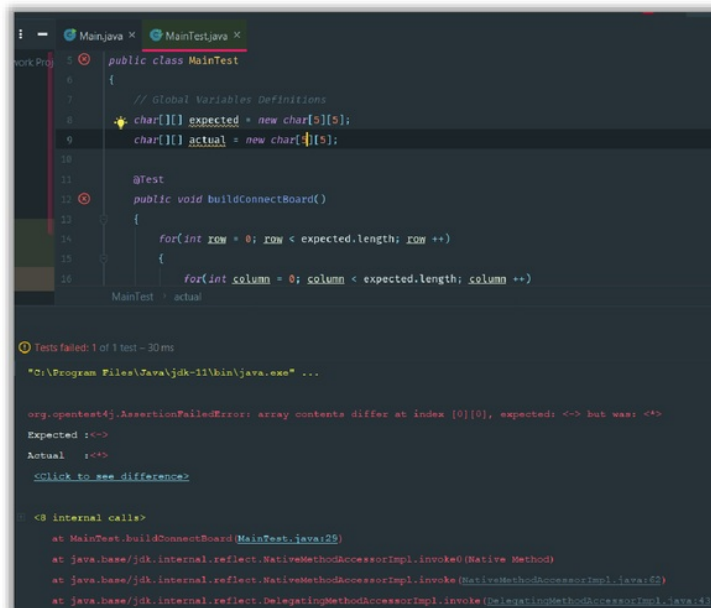
at MainTest.buildConnectBoard(MainTest.java:23)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

Figure 20: JUnit Tests | buildConnectBoard Fail



```
1 public class MainTest
2 {
3     // Global Variables Definitions
4     char[][] expected = new char[5][5];
5     char[][] actual = new char[4][5];
6
7     @Test
8     public void buildConnectBoard()
9     {
10         for(int row = 0; row < expected.length; row++)
11         {
12             for(int column = 0; column < expected.length; column++)
13             {
14                 MainTest actual
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Tests failed: 1 of 1 test - 30 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

org.opentest4j.AssertionFailedError: array contents differ at index [0][0], expected: <*> but was: <*>

Expected :<*>

Actual :<*>

[Click to see difference](#)

<Internal call>

at MainTest.buildConnectBoard(MainTest.java:23)

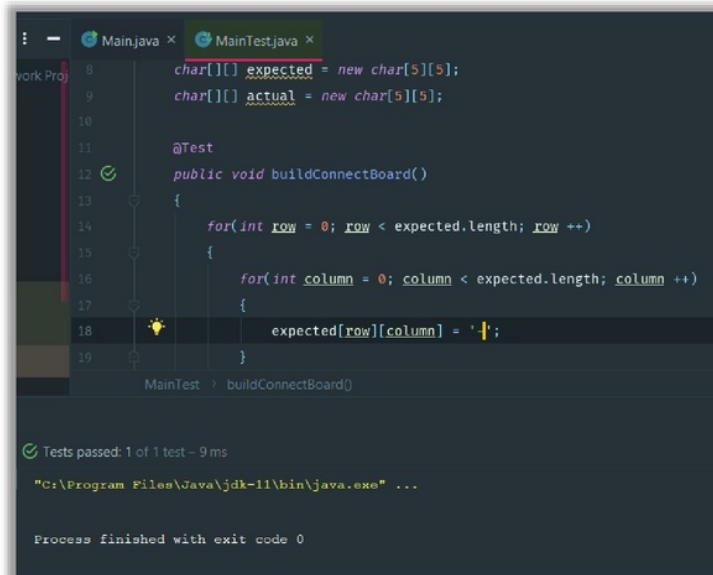
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)

at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

Figure 21: JUnit Tests | buildConnectBoard Fail

The second test was to ensure that the board was being built using the correct characters.

A screenshot of an IDE window showing a Java file named MainTest.java. The code defines a 5x5 character array 'expected' and 'actual', both initialized with spaces. A JUnit test method 'buildConnectBoard()' is shown, which iterates over the array and sets each element to a space character. The test is marked as passed with a green checkmark. The output console shows 'Tests passed: 1 of 1 test - 9 ms' and 'Process finished with exit code 0'.

```
8 char[][] expected = new char[5][5];
9 char[][] actual = new char[5][5];
10
11 @Test
12 public void buildConnectBoard()
13 {
14     for(int row = 0; row < expected.length; row++)
15     {
16         for(int column = 0; column < expected.length; column++)
17         {
18             expected[row][column] = ' ';
19         }
20     }
21 }
```

Tests passed: 1 of 1 test - 9 ms

"C:\Program Files\Java\jdk-11\bin\java.exe" ...

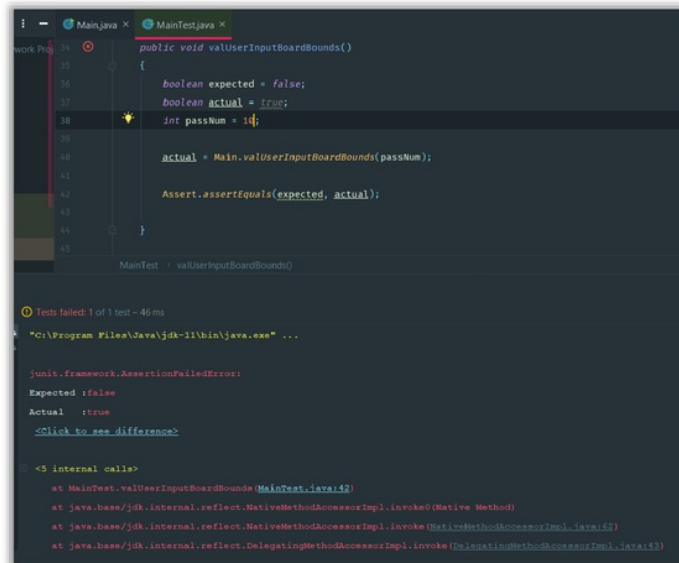
Process finished with exit code 0

Figure 22: JUnit Tests | buildConnectBoard Pass

Revision to test allows for pass: Ensuring that the actual char array size and placeholder value matches allows for pass.

3.6 Part Seven: Exercise Two | valUserInputBoardBounds

Below we can ensure the failing of the method by passing a higher bounds than what is expected by the application, further passed by ensuring the correct parameter.



The screenshot shows an IDE with two tabs: Main.java and MainTest.java. In MainTest.java, the following code is visible:

```
public void valUserInputBoardBounds()
{
    boolean expected = false;
    boolean actual = true;
    int passNum = 10;

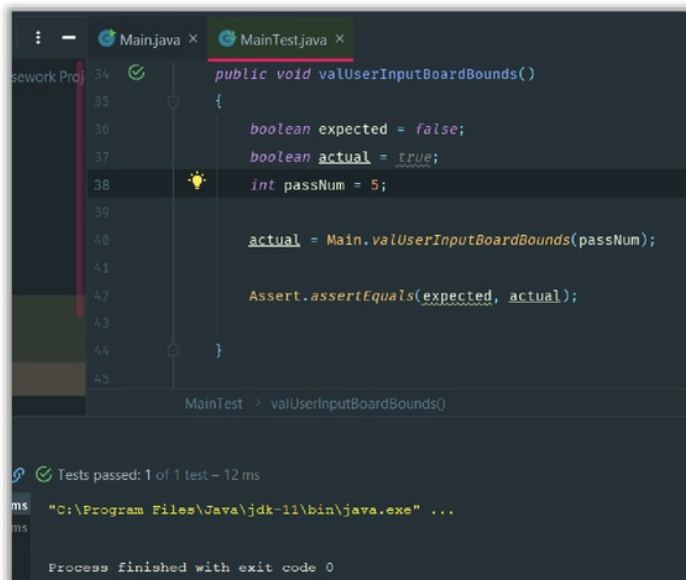
    actual = Main.valUserInputBoardBounds(passNum);

    Assert.assertEquals(expected, actual);
}
```

The test is failing, as indicated by a red circle icon. The output console shows the following error message:

```
Tests failed: 1 of 1 test - 46 ms
"C:\Program Files\Java\jdk-11\bin\java.exe" ...
java.lang.AssertionError:
Expected :false
Actual   :true
Click to see difference
<5 internal calls>
at MainTest.valUserInputBoardBounds(MainTest.java:42)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(MainTest.java:42)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
```

Figure 23: JUnit Tests valUserInputBoardBounds Fail

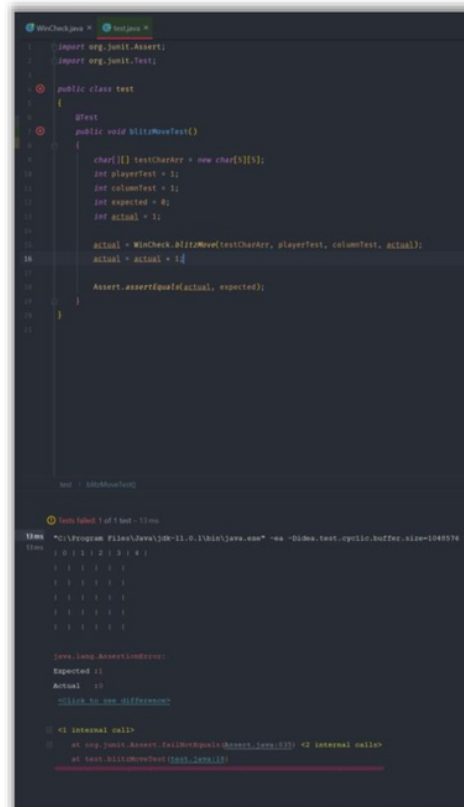


The screenshot shows the same IDE with the MainTest.java tab. The code is identical to the previous figure, but the test is now passing, as indicated by a green circle icon. The output console shows the following message:

```
Tests passed: 1 of 1 test - 12 ms
"C:\Program Files\Java\jdk-11\bin\java.exe" ...
Process finished with exit code 0
```

Figure 24: JUnit Tests valUserInputBoardBounds Pass

3.7 Part Eight: Exercise Two | blitzMove



```
1 import org.junit.Assert;
2 import org.junit.Test;
3
4 public class test
5 {
6     @Test
7     public void blitzMoveTest()
8     {
9         char[][] testCharter = new char[5][5];
10         int playerTest = 1;
11         int columnTest = 1;
12         int expected = 4;
13         int actual = 1;
14
15         actual = WinCheck.BlitzMove(testCharter, playerTest, columnTest, actual);
16         actual = actual + 1;
17
18         Assert.assertEquals(actual, expected);
19     }
20 }
```

Test failed: 1 of 1 test - 11ms

8ms "C:\Program Files\Java\jdk-11.0.1\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 -

8ms

```
1 0 1 1 2 3 4 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
```

java.lang.AssertionError:
Expected: 4
Actual: 14
[Click to see difference](#)

❌ Internal call:
at org.junit.Assert.failNotEquals(Assert.java:111) *2 Internal call:
at test.BlitzMoveTest.test_1000127

Figure 25:JUnit Tests | BlitzMove Fail



```
1 import org.junit.Assert;
2 import org.junit.Test;
3
4 public class test
5 {
6     @Test
7     public void blitzMoveTest()
8     {
9         char[][] testCharter = new char[5][5];
10         int playerTest = 1;
11         int columnTest = 1;
12         int expected = 4;
13         int actual = 1;
14
15         actual = WinCheck.BlitzMove(testCharter, playerTest, columnTest, actual);
16         actual = actual + 1;
17
18         Assert.assertEquals(actual, expected);
19     }
20 }
```

Tests passed: 1 of 1 test - 8 ms

8ms "C:\Program Files\Java\jdk-11.0.1\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 -

8ms

```
1 0 1 1 2 3 4 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
```

Process finished with exit code 0

Figure 26: JUnit Tests | BlitzMove Pass

The above is testing that the blitz has been set to 0 so that players cannot continue to use the move.

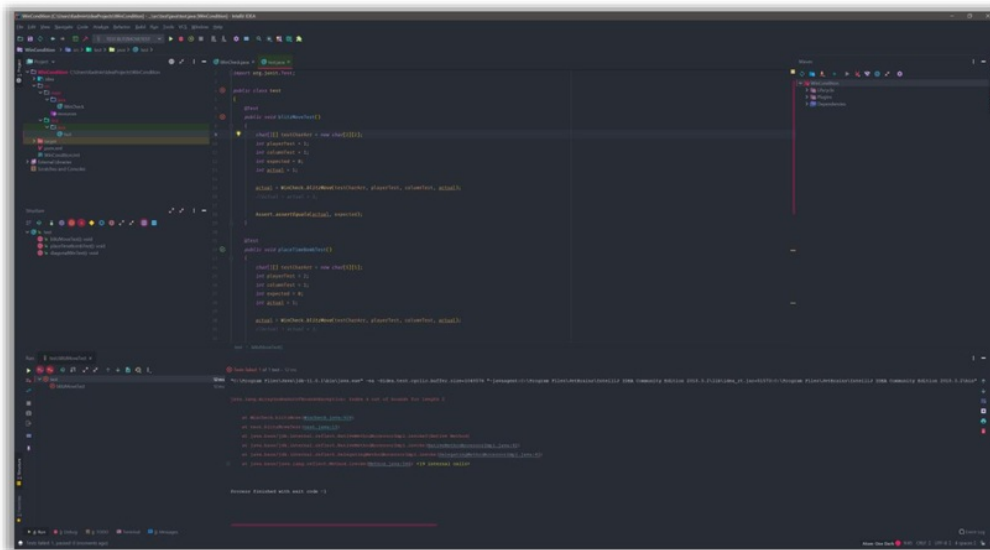


Figure 27: JUnit Tests | BlitzMove Fail

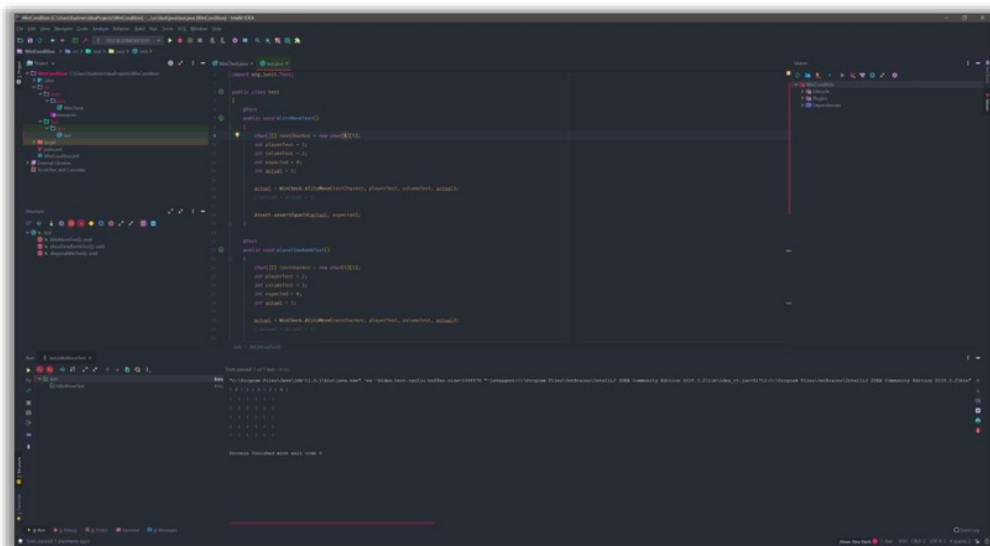


Figure 28: JUnit Tests | BlitzMove Pass

The above makes sure that the method only handles a 5x5 char array.

3.8 Part Nine: Exercise Two | placeTimeBomb

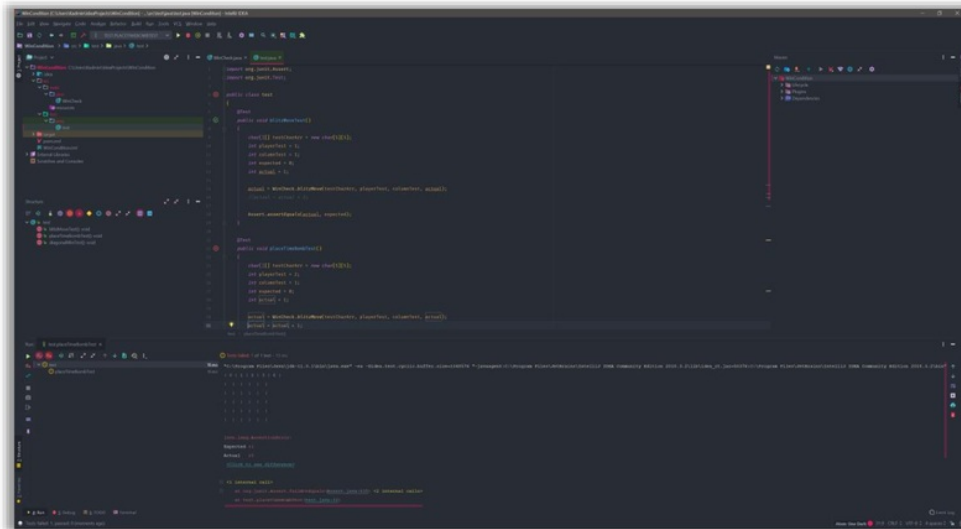


Figure 29: JUnit Tests | placeTimeBomb Fail

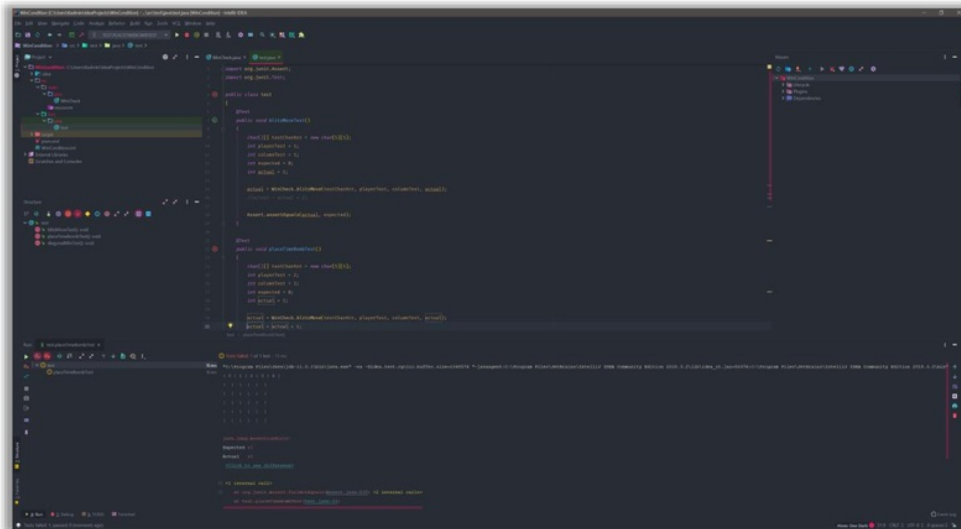


Figure 30: JUnit Tests | placeTimeBomb Pass

The above is testing that the bomb has been set to 0 so that players cannot continue to use the move.

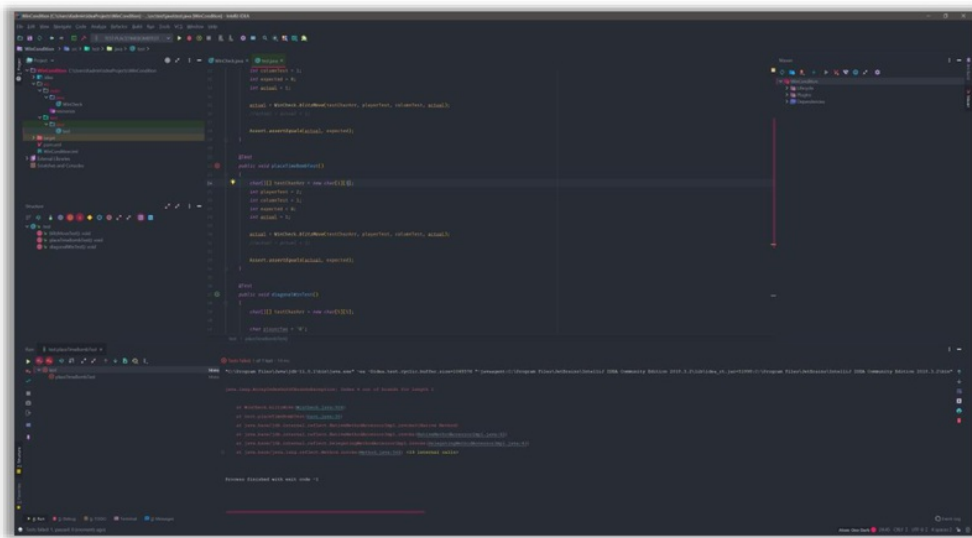


Figure 31: JUnit Tests | placeTimeBomb Fail

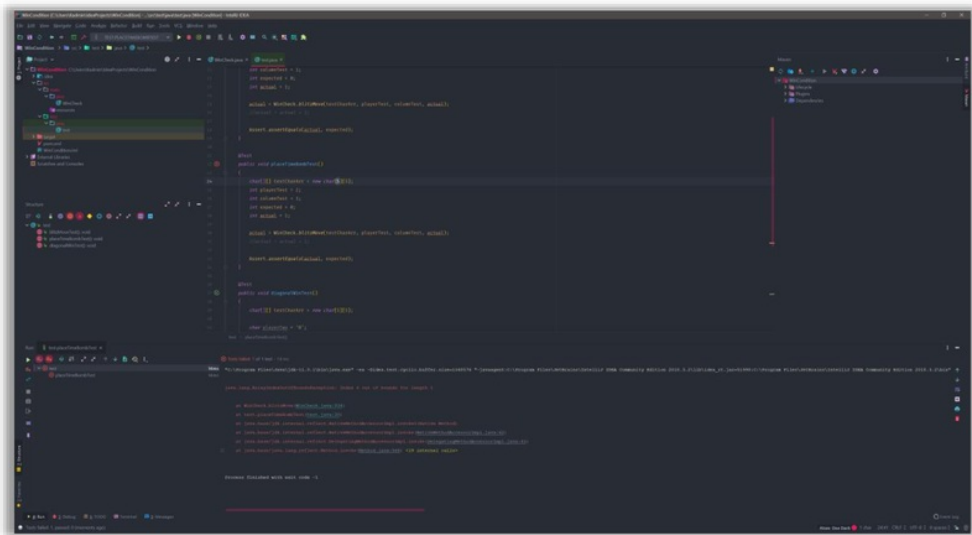


Figure 32: JUnit Tests | placeTimeBomb Pass

The above makes sure that the method only handles a 5x5 char array.

This Page Left Intentionally Blank

Reference List

- [1] Stack Overflow, "What is high cohesion and how to use it" *StackOverflow* 2018. [Online] Available: <https://stackoverflow.com/questions/10830135/what-is-high-cohesion-and-how-to-use-it-make-it>. [Accessed Dec. 19, 2018].
- [2] Maven, "What is Maven" *Maven* 2018. [Online] Available: <https://maven.apache.org/index.html> [Accessed Dec. 19, 2018].
- [3] IBM, "Automating RIA" *IBM* 2018. [Online] Available: <https://www.ibm.com/developerworks/library/a-automating-ria/index.html> [Accessed Dec. 20, 2018].
- [4] Danylkoweb, "What is Loose Coupling" *Danylkoweb* 2018. [Online] Available: <https://www.danylkoweb.com/Blog/what-is-loose-coupling-G6> [Accessed Dec. 20, 2018].
- [5] Webopedia, "Information Hiding" *Webopedia* 2018. [Online] Available: https://www.webopedia.com/TERM/I/information_hiding.html [Accessed Dec. 22, 2018].
- [6] Leino, K., & Nelson, G. (2002). Data abstraction and information hiding. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(5), 491-553.
- [7] Evans, B. and Flanagan, D. (2014). *JAVA IN A NUTSHELL*. 6th ed. Sebastopol: O'REILLY MEDIA, INC, USA, p.121.
- [8] Evans, B. and Flanagan, D. (2014). *JAVA IN A NUTSHELL*. 6th ed. Sebastopol: O'REILLY MEDIA, INC, USA, p.331.
- [9] Sanaulla, "Cohesion and Coupling" *Sanaulla* 2018. [Online] Available: <https://sanaulla.info/2008/06/26/cohesion-and-coupling-two-oo-design-principles/> [Accessed: Jan. 06, 2019].
- [10] P. Rogers, "Encapsulation is not information hiding" *The principles of information hiding go beyond the Java language facility for encapsulation*. [Online]. Available: <https://www.javaworld.com/article/2075271/core-java/encapsulation-is-not-information-hiding.html>. [Accessed: Jan. 07, 2019].

Reading List

- Stack Overflow, "What is High Cohesion and How to Make Use of it" *Stackoverflow* 2018. [Online] Available: <https://stackoverflow.com/questions/10830135/what-is-high-cohesion-and-how-to-use-it-make-it> [Accessed Dec. 20, 2018].
- Stack Overflow, "Encapsulation Vs Data Hiding" *Stackoverflow* 2018. [Online] Available: <https://stackoverflow.com/questions/12013448/encapsulation-vs-data-hiding-java> [Accessed Dec. 20, 2018].
- Hib Pages, "Data Hiding in Java" *Hubpages* 2018. [Online] Available: <https://hubpages.com/technology/Data-Hiding-in-Java> [Accessed: Dec. 22, 2018].
- Software Engineer, "What is the Difference between Data Hiding and Encapsulation" *Softwareengineer* 2018. [Online] Available:

<https://softwareengineering.stackexchange.com/questions/173547/what-is-the-difference-between-data-hiding-and-encapsulation> [Accessed: Dec. 22, 2018].

<https://dzone.com/articles/the-secret-life-of-objects-information-hiding>

Chapter One Appendix

Chapter One Thought Process

The first feature I wanted to implement during the first exercise was the number arrays. Once I had this working I could move on to the other features of the application such as drawing out on paper how the set difference and intersection methods would need to work. Before this I also knew that I had to ensure that validation checking was being performed on users input as well as looping back through allowing users to attempt to input values until they pass error checking validation. After doing the assignment, I can see that the public global variables should really either not be used or at the very least set to private, so to perform good coding practice. As such, this is something that I would look to change in the future.

```
import java.util.*;

public class Main {
    // Global Variables
    public static double[] usrArr1;
    public static double[] usrArr2;
    public static Scanner scan = new Scanner(System.in);
    public static int count = 0;

    public static void main(String[] args) {
        methodCalls();
    }

    public static void methodCalls() {
        userMenu();
        userInput();
    }

    /**
     * Method To Provide User Menu
     */
    public static void userMenu() {
        System.out.println("Option 1: Display Numbers in Both Sets");
        System.out.println("Option 2: Maximum and Minimum Values in Both Sets");
        System.out.println("Option 3: Calculate Average of Numbers in Both Sets");
        System.out.println("Option 4: Determine if Sets A & B Overlap");
        System.out.println("Option 5: Determine Intersection of A & B");
        System.out.println("Option 6: Determine Set Difference of A & B");
        System.out.println("Option 7: Quit");
    }

    /**
     * Method to Provide Application Functionality to User
     * Such as Entering Number Sets and Providing Method Calls
     * To userMenuChoice
     */
    public static void userInput() {
        int usrMenuChoice = 0;

        System.out.println("How Many Numbers Would You Like to Enter?");

        // Exception Handler
        try {
            count = scan.nextInt();

            try {
                System.out.println("Please Enter Your First Set of Numbers: ");
                usrArr1 = buildUsrArr(count);
                System.out.println("Please Enter Your Second Set of Numbers: ");
                usrArr2 = buildUsrArr(count);

                System.out.println("Please Choose a Menu Option: ");
                userMenu();

                try {
                    usrMenuChoice = scan.nextInt();

                    if (usrMenuChoice <= 0 || usrMenuChoice > 7) {
                        boolean invalidInput = true;
                        while (invalidInput) {
                            System.out.println("Error: Please Enter From the Menu Options.");
                            usrMenuChoice = scan.nextInt();
                        }
                    }
                }
            }
        }
    }
}
```

```

        if (usrMenuChoice > 0 && usrMenuChoice < 8) {
            invalidInput = false;
            break;
        }
    }
    passMenuChoice(usrMenuChoice);
} catch (InputMismatchException a) {
    System.out.println("Error: Input Mismatch. Exiting...");
}
} catch (InputMismatchException a) {
    System.out.println("Error: Input Mismatch. Exiting...");
}
} catch (InputMismatchException a) {
    System.out.println("Error: Input Mismatch. Exiting...");
}
}

/**
 * @param a Takes count Parameter
 * @return tempArr to Be Allocated Into usrArr1/2
 */
public static double[] buildUsrArr(int a) {
    int countHolder = a;
    double[] tempArr = new double[countHolder];

    // Loop Through User Number Inputs
    for (int i = 0; i < tempArr.length; i++) {
        tempArr[i] = scan.nextDouble();

        // Secondly, Use Counting Variable j to Compare
        // Against i to Determine if User Has Entered Duplicate
        for (int j = 0; j < i; j++) {
            if (tempArr[i] == tempArr[j]) {
                System.out.println("Error: Please Don't Enter Duplicate Numbers. " +
                    "Enter Your Last Number Again.");
                // If Duplicate Found Decrement i
                i--;
            }
        }
    }
    return tempArr;
}

/**
 * Method to Take Input From userInput1/2
 * And Further Pass This on to Necessary Method
 */
/**
 * @param a Takes Users Option Choice
 * @return userChoice
 */
public static void passMenuChoice(int a) {
    int passChoice = a;

    switch (passChoice) {
        case 1:
            // Perform Method Call to Option One
            System.out.println("You Chose Option 1.");
            System.out.println("Preparing to Print Numbers...");
            double[] printTempArr1 = printUsrArr(usrArr1, 1);
            double[] printTempArr2 = printUsrArr(usrArr2, 2);
            break;
        case 2:
            // Perform Method Call to Option Two
            System.out.println("You Chose Option 2.");
            System.out.println("Preparing to Print Max & Min Values...");
            double max1 = max(usrArr1, 1);
            double min1 = min(usrArr1, 1);
            double max2 = max(usrArr2, 2);
            double min2 = min(usrArr2, 2);
            break;
        case 3:
            // Perform Method Call to Option Three
            System.out.println("You Chose Option 3.");
            System.out.println("Preparing to Print Averages...");
            System.out.println("Averages of Number Inputs 1 & 2 Are:");
            double temp1 = average(usrArr1);
            double temp2 = average(usrArr2);
            break;
        case 4:
            // Perform Method Call to Option Four
            System.out.println("You Chose Option 4.");
            System.out.println("Preparing to Determine Overlapping...");
            double[] determineOverlap1 = usrArr1;
            double[] determineOverlap2 = usrArr2;

```

```

        overlap(determineOverlap1, determineOverlap2);
        break;
    case 5:
        // Perform Method Call to Option Five
        System.out.println("You Chose Option 5.");
        System.out.println("Preparing to Find Intersection...");
        double intersect = intersect(usrArr1, usrArr2);
        break;
    case 6:
        // Perform Method Call to Option Six
        System.out.println("You Chose Option 6.");
        double[] determineSetDiff1 = usrArr1;
        double[] determineSetDiff2 = usrArr2;
        setDifference(determineSetDiff1, determineSetDiff2);
        break;
    case 7:
        // Perform Method Call to Option Seven
        System.out.println("You Chose Option 7.");
        System.out.println("Exiting Application...");
        break;
    }
}

/**
 * Method to Print Set Array
 */
/**
 * @param a Contains Set1
 * @param b Contains Set2
 * @return tempArr
 */
public static double[] printUsrArr(double[] a, int b) {
    double[] tempArr = a;
    int numSet = b;

    System.out.println("You Entered a Total of " + count + " Numbers.");
    System.out.println("Set " + numSet + " Contains:");

    for (int i = 0; i < tempArr.length; i++) {
        System.out.println(tempArr[i]);
    }

    return tempArr;
}

/**
 * Method to Calculate the Max Value Within Sets
 */
/**
 * @param a Contains Set1
 * @param b Contains Set2
 * @return max
 */
public static double max(double[] a, int b) {
    int localCount = b;
    double[] temp = a;

    // Set max & min to value of [0]
    // Position of userInput1 Array
    double max = temp[0];

    // Create a for Loop to Cycle Through userInput1 Array
    // Start From 1 As We Assign 0 to Max
    for (int i = 1; i < temp.length; i++) {
        // if Position [i] is Greater Than Position
        // Position [0] of userInput1 Allocate to Max
        if (temp[i] > max) {
            max = temp[i];
        }
    }
    System.out.println("Max Value of Set " + localCount + " is: " + max);

    return max;
}

/**
 * Method Will Determine the Minimum Value of Both Sets
 */
/**
 * @param a Contains Set1
 * @param b Contains Set2
 * @return
 */
public static double min(double[] a, int b) {
    int localCount = b;
    double[] temp = a;
    double min = temp[0];

    // Create a for loop to cycle through userInput1 Array

```

```

// Start From 1 As We Assign 0 to Min
for (int j = 1; j < temp.length; j++) {
    if (temp[j] < min) {
        // If Position [j] is Greater Than Position
        // Position [0] of userInput1 Allocate to Min
        min = temp[j];
    }
}
System.out.println("Min Value of Set " + localCount + " Is: " + min);

return min;
}

/**
 * @param a Contains Set1
 * @return av
 */
public static double average(double a[]) {
    double[] average = a;
    double sum = 0;
    double av;

    // https://docs.oracle.com/javase/7/html/jls-10.html#jls-10.4
    for (double i : average) {
        sum += i;
    }

    // Use average.length to Accurately Divide by
    // How Many Numbers Exist Within the Array
    av = sum / average.length;

    System.out.println(av);
    return av;
}

/**
 * Method Will Determine if Sets Have Overlapped in Any Way.
 * If They Have User Will be Prompted to See The Overlapped
 * Numbers
 */
* @param a Contains Set1
* @param b Contains Set2
*
public static void overlap(double[] a, double[] b) {
    // Store arrSet1 & 2
    boolean overlap = false;
    double[] tempSetArr1 = a;
    double[] tempSetArr2 = b;
    String userChoice;

    // Loop Through All Possible Index Positions of Array to Find Match
    for (int i = 0; i < tempSetArr1.length; i++) {
        for (int j = 0; j < tempSetArr2.length; j++) {
            if (tempSetArr1[i] == tempSetArr2[j]) {
                overlap = true;
                System.out.println("Overlap Has been Found...");
                System.out.println(tempSetArr1[i] + " | " + tempSetArr2[j]);
            }
        }
    }

    if (overlap == false) {
        System.out.println("No Overlap Found. Exiting...");
    }
}

/**
 * Method Will Determine the Intersection Values
 * Within Both Sets
 */
* @param a Contains Set1
* @param b Contains Set2
* @return intersect
*
public static double intersect(double[] a, double[] b) {
    // Store arrSet1 & 2
    double[] tempSetArr1 = a;
    double[] tempSetArr2 = b;
    double intersect = 0;

    // Loop Through All Possible Index Positions of Array to Find Match
    for (int i = 0; i < tempSetArr1.length; i++) {
        for (int j = 0; j < tempSetArr2.length; j++) {
            if (tempSetArr1[i] == tempSetArr2[j]) {

```

```

        intersect = tempSetArr1[i];
        System.out.println("Matches Have Been Found");
        System.out.println("Intersection Value of " + intersect + " Has Been Found in Both Sets.");
    }
}
return intersect;
}

/**
 * Method to Calculate the Set Difference
 */
/**
 * @param a Contains Set1
 * @param b Contains Set2
 */
public static void setDifference(double[] a, double[] b)
{
    ArrayList<Double> temp1 = new ArrayList<Double>();
    ArrayList<Double> temp2 = new ArrayList<Double>();

    double[] localTempSetArr1 = a;
    double[] localTempSetArr2 = b;

    // Store Array Contents in ArrayList so that we can use
    // Inbuilt Remove Method values from Array to Determine
    // Set Difference
    for (int i = 0; i < localTempSetArr1.length; i++)
    {
        temp1.add(localTempSetArr1[i]);
        for (int j = 0; j < localTempSetArr2.length; j++)
        {
            if (localTempSetArr1[i] == localTempSetArr2[j])
            {
                temp1.remove((localTempSetArr1[i]));
            }
        }
    }
    System.out.println("The Set Difference is: " + temp1);
}
}

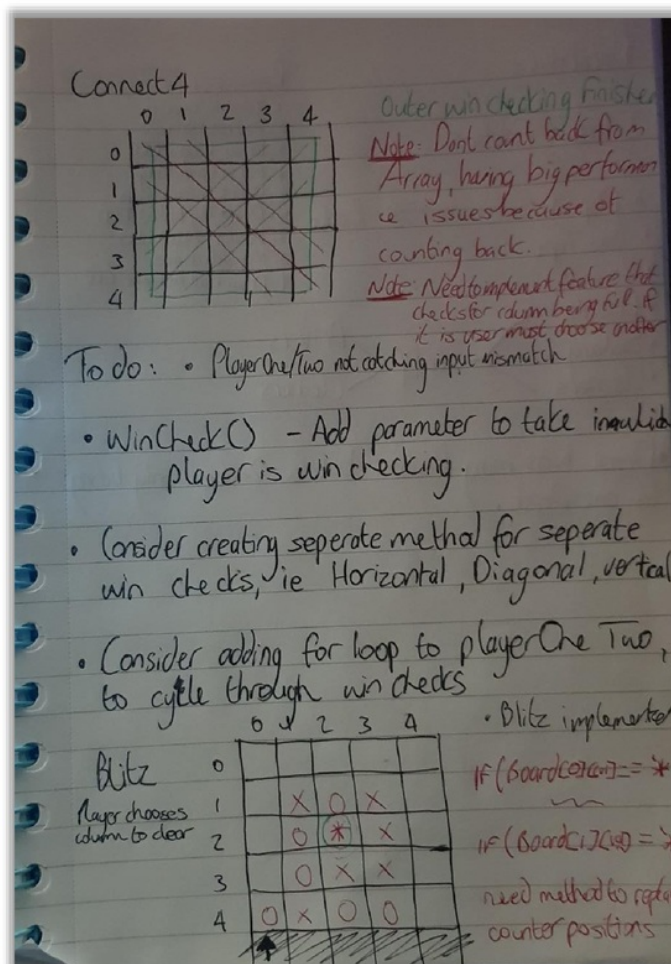
```

Chapter Two Appendix

Chapter Two Thought Process

My thought process going into the second exercise was to first build the board. Once the board was built and working I wanted to start creating some of the players gameplay functionality. I needed to do both of these first so that I can perform my own tests to ensure the game was bounds checking correctly.

The hardest part was the logic for detonating the bombs, more so, the clearing of the counters. Although I did it, I found it frustrating as I know this can be done within a for loop instead of hard coded. So that I could write code for the blitzing and bomb, I drew on paper and made notes so that I could tackle the problem. This can be further illustrated below:



Above gives an example of how I approached tasks such as win checking and the clearing of counters when the time bomb detonated.

```

import java.util.InputMismatchException;
import java.util.Scanner;

public class Main {
    // Global Variables
    public static char[][] board = new char[5][5];
    public static Scanner scan = new Scanner(System.in);
    public static int playerChoice = 0;
    public static char playerOne = 'X';
    public static char playerTwo = 'O';
    public static boolean gameOver = false;
    public static boolean verticalWin = false;
    public static boolean horizontalWin = false;
    public static boolean diagonalWin = false;
    public static boolean P1BombTick = false;
    public static boolean P2BombTick = false;
    public static int P1BlitzCount = 1;
    public static int P2BlitzCount = 1;
    public static int P1BombCol = 0;
    public static int P2BombCol = 0;
    public static int P1TimeBomb = 1;
    public static int P2TimeBomb = 1;
    public static int P1BombCounter = 3;
    public static int P2BombCounter = 3;

    public static void main(String[] args) {
        methodCalls();
    }

    /**
     * Performs Method Calls on the Applications Methods
     */
    public static void methodCalls() {
        buildConnectBoard();
        game();

        if (!gameOver)
        {
            printConnectBoard();
        }
    }

    /**
     * Create Connect 4 Board
     *
     * @return Game Board
     */
    public static char[][] buildConnectBoard()
    {
        for (int row = 0; row < board.length; row++)
        {
            for (int column = 0; column < board.length; column++)
            {
                board[row][column] = '-';
            }
        }
        return board;
    }

    /**
     * Print the Connect 4 Board
     */
    public static void printConnectBoard() {
        // Print Visual Column Que to Help Players Choose
        // Playable Column by Calling printColNum()
        printColNum();
        System.out.println();
        for (int row = 0; row < board.length; row++)
        {
            System.out.print(" | ");
            for (int column = 0; column < board.length; column++)
            {
                System.out.print(board[row][column] + " | ");
            }
            System.out.println();
        }
    }

    /**
     * Provides Visual Que to Players by Printing Column Numbers
     * To Console

```

```

*/
public static void printColNum()
{
    int[] colNum = new int[5];

    // Populate the colNum Array
    for (int count = 0; count < colNum.length; count++)
    {
        colNum[count] = count;
    }
    // Now Print colNum Array
    for (int i = 0; i < colNum.length; i++)
    {
        System.out.print(" | " + colNum[i] + " ");
    }
    System.out.print("\n");
}

/**
 * Creates and Prints Game Rules to Console
 * To Provide Visual Cue to Player
*/
public static void gameRules()
{
    System.out.println("GAME RULES");
    System.out.println("\t[1] Players 1 & 2 on Turn, Choose Column to Drop " +
        "Your Counter Down.");
    System.out.println("\t[2] Game is Won by Player Obtaining Horizontal, " +
        "Vertical or Diagonal Line of Four Matching Counters");
    System.out.println("\t[3] BLITZ: Activates by Pressing B. Clears a Column, " +
        "Each Player has One BLITZ Play.");
    System.out.println("\t[4] TIME BOMB: Activates by Pressing T. Player Needs " +
        "to Select Column to Place Bomb. ");
    System.out.println("[Note] TIME BOMB Will Clear Counters Surrounding Bomb Position.");
}

/**
 * Performs Required Game Method Calls
*/
public static void game()
{
    gameRules();
    System.out.println("Welcome to CONNECT 4...");
    while (!gameOver) {
        playerOne();
        playerTwo();
    }

    if (isBoardFull()) {
        System.out.println("Game Board is Full. No Winner.");
    }
}

/**
 * Creates Player One Game Functionality
*/
public static void playerOne()
{
    String specialMove;
    int colToClear = 0;
    int player = 1;
    boolean specialMoveTrigger = false;
    boolean valInp = false;
    boolean valColumn;

    // Continue whilst valInput is False: This allows game to
    // continue if user enters invalid out of bounds positions
    while (!valInp)
    {
        // Try Catch
        try
        {
            printConnectBoard();
            System.out.println("Player 1: Please Choose a Column: ");
            playerChoice = Integer.parseInt(scan.next());
        }
        catch (Exception e)
        {
            // If input is out of bounds, return error message
            if (valUserInputBoardBounds(playerChoice))
            {
                System.out.println("ERROR: Position Chosen is out of Bounds...");
            }
        }
    }
}

```

```

    }

    // Validate Column is Not Full
    valColumn = valColumnAvailable(playerChoice);

    if(!valColumn)
    {
        System.out.println("Column Unavailable");
        System.out.println("Please Choose Another Column");
    }

    // Otherwise continue with the game
    else if (!valUserInputBoardBounds(playerChoice))
    {
        valInp = true;
        placePlayerCounters(player, playerChoice);
        printConnectBoard();
        horizontalWin = horizontalWinCheck(board, player);
        verticalWin = verticalWinCheck(board, player);
        diagonalWin = diagonalWinCheck(board, player);
        if (horizontalWin) {
            System.out.println("WINNER \t Player " + playerOne + " Wins By Horizontal Line.");
            gameOver = true;
            endGame();
        } else if (verticalWin) {
            System.out.println("WINNER \t Player " + playerOne + " Wins by Vertical Line.");
            gameOver = true;
            endGame();
        } else if (diagonalWin)
        {
            System.out.println("WINNER \t Player " + playerOne + " Wins by Diagonal Line.");
            gameOver = true;
            endGame();
        }
    }
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("ERROR. " + e.getMessage());
}
catch (NumberFormatException e)
{
    System.out.println("ERROR. " + e.getMessage());
}
}
while ((!specialMoveTrigger && P1BlitzCount > 0)
{
    try
    {
        System.out.println("Would you like to Blitz?");
        specialMove = scan.next();

        if (specialMove.matches("B") || specialMove.matches("b"))
        {
            specialMoveTrigger = true;

            System.out.println("Please Enter a Column to Blitz: ");
            colToClear = scan.nextInt();

            P1BlitzCount = blitzMove(board, player, colToClear, P1BlitzCount);
        }
        else if (specialMove.matches("No") || specialMove.matches("no"))
        {
            break;
        }
        else if (!specialMove.matches(".*[a-zA-Z].*"))
        {
            System.out.println("ERROR: Please Enter a Valid String.");
            System.out.println("Please Enter B to Blitz. Otherwise, No to skip...");
        }
    }
    catch (InputMismatchException e)
    {
        System.out.println("ERROR: " + e.getMessage());
    }
    catch (Exception e)
    {
        System.out.println("Exception Occurred...");
    }
}
}

```

```

while ((!specialMoveTrigger) && P1TimeBomb > 0)
{
    try
    {
        System.out.println("Would you like to place a Bomb?");
        specialMove = scan.next();

        if (specialMove.matches("T") || specialMove.matches("t"))
        {
            specialMoveTrigger = true;
            P1BombTick = true;
            System.out.println("Please Enter a Column to place a Bomb: ");
            P1BombCol = scan.nextInt();

            P1TimeBomb = placeTimeBombMove(board, player, P1BombCol, P1TimeBomb);
        }
        else if (specialMove.matches("No") || specialMove.matches("no"))
        {
            break;
        }
        else if (!specialMove.matches("[a-zA-Z]+.*"))
        {
            System.out.println("ERROR: Please Enter a Valid String.");
            System.out.println("Please Enter T to Place Bomb. Otherwise, No to skip...");
        }
    } catch (InputMismatchException e) {
        System.out.println("ERROR: " + e.getMessage());
    } catch (Exception e)
    {
        System.out.println("Exception Occurred...");
    }
}

// Allow the Bomb to Tick for 2 Rounds
while (P1BombTick)
{
    System.out.println("Detonating Player " + player + "'s" + " Bomb in... " + P1BombCounter);
    P1BombCounter--;
    if (P1BombCounter == 0) {
        P1BombTick = false;
        P1TimeBomb = 0;
        // Method call to bomb clear method
        System.out.println("Detonating Player " + player + "'s" + " Bomb...");
        detonateBomb(board, player, P1BombCol);
        break;
    }
    else if (P1BombCounter > 0)
    {
        break;
    }
}
}

/**
 * Creates Player Two Game Functionality
 */
public static void playerTwo()
{
    String specialMove;
    int colToClear = 0;
    int player = 2;
    boolean specialMoveTrigger = false;
    boolean valinp = false;
    boolean valColumn;

    // Continue whilst valInput is False: This allows game to
    // continue if user enters invalid out of bounds positions
    while ((!valinp) && !gameOver)
    {
        // Try Catch
        try
        {
            printConnectBoard();
            System.out.println("Player 2: Please Choose a Column: ");
            playerChoice = Integer.parseInt(scan.next());
        }
        catch (Exception e)
        {
            // If input is out of bounds, return error message
            if (valUserInputBoardBounds(playerChoice))
            {
                System.out.println("ERROR: Position Chosen is out of Bounds...");
            }
        }
    }
}

```

```

// Validate Column is Not Full
valColumn = valColumnAvailable(playerChoice);

if(!valColumn)
{
    System.out.println("Column Unavailable");
    System.out.println("Please Choose Another Column");
}

// Otherwise continue with the game
else if (!valUserInputBoardBounds(playerChoice))
{
    valInp = true;
    placePlayerCounters(player, playerChoice);
    printConnectBoard();
    horizontalWin = horizontalWinCheck(board, player);
    verticalWin = verticalWinCheck(board, player);
    diagonalWin = diagonalWinCheck(board, player);
    if (horizontalWin) {
        System.out.println("WINNER \t Player " + playerTwo + " Wins by Horizontal Line.");
        gameOver = true;
        endGame();
    } else if (verticalWin) {
        System.out.println("WINNER \t Player " + playerTwo + " Wins by Vertical Line.");
        gameOver = true;
        endGame();
    } else if (diagonalWin)
    {
        System.out.println("WINNER \t Player " + playerTwo + " Wins by Diagonal Line.");
        gameOver = true;
        endGame();
    }
}
}
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("ERROR: " + e.getMessage());
}
}
catch (NumberFormatException e)
{
    System.out.println("ERROR: " + e.getMessage());
}
}
}
while (!specialMoveTrigger && P2BlitzCount > 0)
{
    try
    {
        System.out.println("Would you like to Blitz?");
        specialMove = scan.next();

        if (specialMove.matches("B") || specialMove.matches("b"))
        {
            specialMoveTrigger = true;

            System.out.println("Please Enter a Column to Blitz: ");
            colToClear = scan.nextInt();

            P2BlitzCount = blitzMove(board, player, colToClear, P2BlitzCount);
        }
        else if (specialMove.matches("No") || specialMove.matches("no"))
        {
            break;
        }
        else if (!specialMove.matches("[a-zA-Z]+"))
        {
            System.out.println("ERROR: Please Enter a Valid String.");
            System.out.println("Please Enter B to Blitz. Otherwise, No to skip...");
        }
    }
    catch (InputMismatchException e)
    {
        System.out.println("ERROR: " + e.getMessage());
    }
    catch (Exception e)
    {
        System.out.println("Exception Occurred...");
    }
}
while (!specialMoveTrigger && P2TimeBomb > 0)
{

```

```

try
{
    System.out.println("Would you like to place a Bomb?");
    specialMove = scan.next();

    if (specialMove.matches("T") || specialMove.matches("t"))
    {
        specialMoveTrigger = true;
        P2BombTick = true;
        System.out.println("Please Enter a Column to place a Bomb: ");
        P2BombCol = scan.nextInt();

        P2TimeBomb = placeTimeBombMove(board, player, P2BombCol, P2TimeBomb);
    }
    else if (specialMove.matches("No") || specialMove.matches("no"))
    {
        break;
    }
    else if (!specialMove.matches("[a-zA-Z]+.*"))
    {
        System.out.println("ERROR: Please Enter a Valid String.");
        System.out.println("Please Enter T to Place Bomb. Otherwise, No to skip...");
    }
} catch (InputMismatchException e) {
    System.out.println("ERROR: " + e.getMessage());
} catch (Exception e)
{
    System.out.println("Exception Occurred...");
}
}

// Allow the Bomb to Tick for 2 Rounds
while (P2BombTick)
{
    System.out.println("Detonating Player " + player + "'s" + " Bomb in... " + P2BombCounter);
    P2BombCounter--;
    if (P2BombCounter == 0) {
        P2BombTick = false;
        P2TimeBomb = 0;
        // Method call to bomb clear method
        System.out.println("Detonating Player " + player + "'s" + " Bomb...");
        detonateBomb(board, player, P2BombCol);
        break;
    }
    else if (P2BombCounter > 0)
    {break;}
    break;
}
}

/**
 * Validates Users Board Position
 */
* @param column Passes Players Column Choice
* @return boolean Returns True - False
*/
public static boolean valUserInputBoardBounds(int column) {
    return column > board.length;
}

/**
 *
 * @param column Takes the Column Player has Chosen
 * @return True if Available else False
 */
public static boolean valColumnAvailable(int column)
{
    for (int i = 0; i <= 4; i++)
    {
        if (board[0][column] != '.'
            && board[1][column] != '.'
            && board[2][column] != '.'
            && board[3][column] != '.'
            && board[4][column] != '.')

        {
            return false;
        }
    }

    return true;
}

```

```

/**
 * Method Will Perform The Placing of Counters For Players
 */
*
* @param player Takes in The Current Player
* @param column Takes in The Column Chosen by User
*/
public static void placePlayerCounters(int player, int column) {
    if (player == 1) {
        // Loop Through Board, Bottom to Top
        for (int i = 4; i >= 0; i--) {
            // If Position Available
            if (board[i][column] == '.') {
                // Insert Counter and Stop looping
                board[i][column] = playerOne;
                break;
            }
        }
    } else if (player == 2) {
        for (int i = 4; i >= 0; i--) {
            if (board[i][column] == '.') {
                board[i][column] = playerTwo;
                break;
            }
        }
    }
}

/**
 * Method to Perform Checks on Board
 * Space Availability
 */
* @return True - False
*/
public static boolean isBoardFull() {
    for (int row = 0; row < board.length; row++) {
        for (int column = 0; column < board.length; column++) {
            if (board[row][column] == '.') {
                return false;
            }
        }
    }
    return true;
}

/**
 * Method to Terminate the Application Upon Finding Winner
 */
public static void endGame()
{
    System.out.println("Exiting Game...");
    System.exit(0);
}

/**
 * Method to Perform Horizontal Win Checking
 */
*
* @param board is Parameters Taken from Board
* @param player is Parameter Taken from PlayerOne/Two()
* @return false if No Win Condition Found or True if Found
*/
public static boolean horizontalWinCheck(char[][] board, int player) {
    int localPlayer = player;

    if (localPlayer == 1) {
        // Check Horizontal Possibilities
        for (int row = 0; row <= 5; row++) {
            if (board[row][0] == playerOne
                && board[row][1] == playerOne
                && board[row][2] == playerOne
                && board[row][3] == playerOne) {
                return true;
            }
            if (board[row][1] == playerOne
                && board[row][2] == playerOne
                && board[row][3] == playerOne
                && board[row][4] == playerOne) {
                return true;
            }
            if (board[3][0] == playerOne

```

```

        && board[3][1] == playerOne
        && board[3][2] == playerOne
        && board[3][3] == playerOne) {
            return true;
        }
        if (board[3][1] == playerOne
            && board[3][2] == playerOne
            && board[3][3] == playerOne
            && board[3][4] == playerOne) {
            return true;
        }
        if (board[2][0] == playerOne
            && board[2][1] == playerOne
            && board[2][2] == playerOne
            && board[2][3] == playerOne) {
            return true;
        }
        if (board[2][1] == playerOne
            && board[2][2] == playerOne
            && board[2][3] == playerOne
            && board[2][4] == playerOne) {
            return true;
        }
        if (board[1][0] == playerOne
            && board[1][1] == playerOne
            && board[1][2] == playerOne
            && board[1][3] == playerOne) {
            return true;
        }
        if (board[1][1] == playerOne
            && board[1][2] == playerOne
            && board[1][3] == playerOne
            && board[1][4] == playerOne) {
            return true;
        }
        if (board[0][0] == playerOne
            && board[0][1] == playerOne
            && board[0][2] == playerOne
            && board[0][3] == playerOne) {
            return true;
        }
        if (board[0][1] == playerOne
            && board[0][2] == playerOne
            && board[0][3] == playerOne
            && board[0][4] == playerOne) {
            return true;
        }
    }
} else if (localPlayer == 2) {
    // Check Horizontal Possibilities
    for (int row = 0; row <= 5; row++) {
        if (board[4][0] == playerTwo
            && board[4][1] == playerTwo
            && board[4][2] == playerTwo
            && board[4][3] == playerTwo) {
            return true;
        }
        if (board[4][1] == playerTwo
            && board[4][2] == playerTwo
            && board[4][3] == playerTwo
            && board[4][4] == playerTwo) {
            return true;
        }
        if (board[3][0] == playerTwo
            && board[3][1] == playerTwo
            && board[3][2] == playerTwo
            && board[3][3] == playerTwo) {
            return true;
        }
        if (board[3][1] == playerTwo
            && board[3][2] == playerTwo
            && board[3][3] == playerTwo
            && board[3][4] == playerTwo) {
            return true;
        }
        if (board[2][0] == playerTwo
            && board[2][1] == playerTwo
            && board[2][2] == playerTwo
            && board[2][3] == playerTwo) {
            return true;
        }
    }
}

```

```

        if (board[2][1] == playerTwo
            && board[2][2] == playerTwo
            && board[2][3] == playerTwo
            && board[2][4] == playerTwo) {
            return true;
        }
        if (board[1][0] == playerTwo
            && board[1][1] == playerTwo
            && board[1][2] == playerTwo
            && board[1][3] == playerTwo) {
            return true;
        }
        if (board[1][1] == playerTwo
            && board[1][2] == playerTwo
            && board[1][3] == playerTwo
            && board[1][4] == playerTwo) {
            return true;
        }
        if (board[0][0] == playerTwo
            && board[0][1] == playerTwo
            && board[0][2] == playerTwo
            && board[0][3] == playerTwo) {
            return true;
        }
        if (board[0][1] == playerTwo
            && board[0][2] == playerTwo
            && board[0][3] == playerTwo
            && board[0][4] == playerTwo) {
            return true;
        }
    }
    return false;
}

/**
 * Method to Perform Vertical Win Checking
 */
* @param board is Parameters Taken from Board
* @param player is Parameter Taken from PlayerOne/Two()
* @return false if No Win Condition Found or True if Found
*/
public static boolean verticalWinCheck(char[][] board, int player) {
    int localPlayer = player;

    if (localPlayer == 1) {
        for (int row = 0; row <= 5; row++) {
            if (board[0][0] == playerOne
                && board[1][0] == playerOne
                && board[2][0] == playerOne
                && board[3][0] == playerOne) {
                return true;
            }
            if (board[1][0] == playerOne
                && board[2][0] == playerOne
                && board[3][0] == playerOne
                && board[4][0] == playerOne) {
                return true;
            }
            if (board[0][1] == playerOne
                && board[1][1] == playerOne
                && board[2][1] == playerOne
                && board[3][1] == playerOne) {
                return true;
            }
            if (board[1][1] == playerOne
                && board[2][1] == playerOne
                && board[3][1] == playerOne
                && board[4][1] == playerOne) {
                return true;
            }
            if (board[0][2] == playerOne
                && board[1][2] == playerOne
                && board[2][2] == playerOne
                && board[3][2] == playerOne) {
                return true;
            }
            if (board[1][2] == playerOne
                && board[2][2] == playerOne
                && board[3][2] == playerOne
                && board[4][2] == playerOne) {

```

```

        return true;
    }
    if (board[0][3] == playerOne
        && board[1][3] == playerOne
        && board[2][3] == playerOne
        && board[3][3] == playerOne) {
        return true;
    }
    if (board[1][3] == playerOne
        && board[2][3] == playerOne
        && board[3][3] == playerOne
        && board[4][3] == playerOne) {
        return true;
    }
    if (board[0][4] == playerOne
        && board[1][4] == playerOne
        && board[2][4] == playerOne
        && board[3][4] == playerOne) {
        return true;
    }
    if (board[1][4] == playerOne
        && board[2][4] == playerOne
        && board[3][4] == playerOne
        && board[4][4] == playerOne) {
        return true;
    }
}
} else if (localPlayer == 2) {
    for (int row = 0; row <= 5; row++) {
        if (board[0][0] == playerTwo
            && board[1][0] == playerTwo
            && board[2][0] == playerTwo
            && board[3][0] == playerTwo) {
            return true;
        }
        if (board[1][0] == playerTwo
            && board[2][0] == playerTwo
            && board[3][0] == playerTwo
            && board[4][0] == playerTwo) {
            return true;
        }
        if (board[0][1] == playerTwo
            && board[1][1] == playerTwo
            && board[2][1] == playerTwo
            && board[3][1] == playerTwo) {
            return true;
        }
        if (board[1][1] == playerTwo
            && board[2][1] == playerTwo
            && board[3][1] == playerTwo
            && board[4][1] == playerTwo) {
            return true;
        }
        if (board[0][2] == playerTwo
            && board[1][2] == playerTwo
            && board[2][2] == playerTwo
            && board[3][2] == playerTwo) {
            return true;
        }
        if (board[1][2] == playerTwo
            && board[2][2] == playerTwo
            && board[3][2] == playerTwo
            && board[4][2] == playerTwo) {
            return true;
        }
        if (board[0][3] == playerTwo
            && board[1][3] == playerTwo
            && board[2][3] == playerTwo
            && board[3][3] == playerTwo) {
            return true;
        }
        if (board[1][3] == playerTwo
            && board[2][3] == playerTwo
            && board[3][3] == playerTwo
            && board[4][3] == playerTwo) {
            return true;
        }
        if (board[0][4] == playerTwo
            && board[1][4] == playerTwo
            && board[2][4] == playerTwo
            && board[3][4] == playerTwo) {

```

```

        return true;
    }
    if (board[1][4] == playerTwo
        && board[2][4] == playerTwo
        && board[3][4] == playerTwo
        && board[4][4] == playerTwo) {
        return true;
    }
}
return false;
}

/**
 * Method to Check Diagonal Win
 */
* @param board is Parameters Taken from Board
* @param player is Parameter Taken from PlayerOne/Two()
* @return false if No Win Condition Found or True if Found
*/
public static boolean diagonalWinCheck(char[][] board, int player) {
    int localPlayer = player;

    if (localPlayer == 1) {
        for (int row = 0; row <= 5; row++) {
            if (board[4][0] == playerOne
                && board[3][1] == playerOne
                && board[2][2] == playerOne
                && board[1][3] == playerOne) {
                return true;
            }
            if (board[3][1] == playerOne
                && board[2][2] == playerOne
                && board[1][3] == playerOne
                && board[0][4] == playerOne) {
                return true;
            }
            if (board[3][0] == playerOne
                && board[2][1] == playerOne
                && board[1][2] == playerOne
                && board[0][3] == playerOne) {
                return true;
            }
            if (board[4][1] == playerOne
                && board[3][2] == playerOne
                && board[2][3] == playerOne
                && board[1][4] == playerOne) {
                return true;
            }
            if (board[4][4] == playerOne
                && board[3][3] == playerOne
                && board[2][2] == playerOne
                && board[1][1] == playerOne) {
                return true;
            }
            if (board[3][3] == playerOne
                && board[2][2] == playerOne
                && board[1][1] == playerOne
                && board[0][0] == playerOne) {
                return true;
            }
            if (board[4][3] == playerOne
                && board[3][2] == playerOne
                && board[2][1] == playerOne
                && board[1][0] == playerOne) {
                return true;
            }
            if (board[3][4] == playerOne
                && board[2][3] == playerOne
                && board[1][2] == playerOne
                && board[0][1] == playerOne) {
                return true;
            }
        }
    } else if (localPlayer == 2) {
        for (int row = 0; row <= 5; row++) {
            if (board[4][0] == playerTwo
                && board[3][1] == playerTwo
                && board[2][2] == playerTwo
                && board[1][3] == playerTwo) {
                return true;
            }
        }
    }
}

```

```

    }
    if (board[3][1] == playerTwo
        && board[2][2] == playerTwo
        && board[1][3] == playerTwo
        && board[0][4] == playerTwo) {
        return true;
    }
    if (board[3][0] == playerTwo
        && board[2][1] == playerTwo
        && board[1][2] == playerTwo
        && board[0][3] == playerTwo) {
        return true;
    }
    if (board[4][1] == playerTwo
        && board[3][2] == playerTwo
        && board[2][3] == playerTwo
        && board[1][4] == playerTwo) {
        return true;
    }
    if (board[4][4] == playerTwo
        && board[3][3] == playerTwo
        && board[2][2] == playerTwo
        && board[1][1] == playerTwo) {
        return true;
    }
    if (board[3][3] == playerTwo
        && board[2][2] == playerTwo
        && board[1][1] == playerTwo
        && board[0][0] == playerTwo) {
        return true;
    }
    if (board[4][3] == playerTwo
        && board[3][2] == playerTwo
        && board[2][1] == playerTwo
        && board[1][0] == playerTwo) {
        return true;
    }
    if (board[3][4] == playerTwo
        && board[2][3] == playerTwo
        && board[1][2] == playerTwo
        && board[0][1] == playerTwo) {
        return true;
    }
    }
    return false;
}

/**
 * Method to Allow Player to Perform the Blitz Move
 */
* @param board is Parameters Taken from Board
* @param player is Parameter Taken from PlayerOne/Two()
* @param column Parameter to Take in Players Column Choice
* @param blitzCount Parameter to Take in Amount of Blitz Turns Remaining
* @return Value to Pass Back New Blitz Turns Remaining
*/
public static int blitzMove(char[][] board, int player, int column, int blitzCount) {
    int localPlayer = player;
    int columnToClear = column;
    int localBlitzCount = blitzCount;

    if (localPlayer == 1) {
        for (int i = 4; i >= 0; i--) {
            board[i][columnToClear] = '1';
        }
        localBlitzCount = 0;
    } else if (localPlayer == 2) {
        for (int i = 4; i >= 0; i--) {
            board[i][columnToClear] = '2';
        }
        localBlitzCount = 0;
    }
    printConnectBoard();
    System.out.println();

    return localBlitzCount;
}

/**
 * Method to Place the Time Bomb for Player
 */

```

```

*
* @param board is Parameters Taken from Board
* @param player is Parameter Taken from PlayerOne/Two()
* @param column is the Players Chosen Column
* @param timeBombCount is the Bomb Timer
* @return Returns a new Counter Value
*/
public static int placeTimeBombMove(char[][] board, int player, int column, int timeBombCount)
{
    int localPlayer = player;
    int columnToPlaceBomb = column;
    int localTimeBombCount = timeBombCount;
    char bomb = '*';

    if (localPlayer == 1) {
        for (int i = 4; i >= 0; i--) {
            // If Position Available
            if (board[i][columnToPlaceBomb] == '.') {
                // Insert Bomb and Stop looping
                board[i][columnToPlaceBomb] = bomb;
                break;
            }
        }
        localTimeBombCount = 0;
    }
    else if (localPlayer == 2)
    {
        for (int i = 4; i >= 0; i--) {
            // If Position Available
            if (board[i][columnToPlaceBomb] == '.') {
                // Insert Bomb and Stop looping
                board[i][columnToPlaceBomb] = bomb;
                break;
            }
        }
        localTimeBombCount = 0;
    }
    printConnectBoard();

    return localTimeBombCount;
}

/**
 * Method will Detonate Bomb Once Counter
 * has Reached a Certain Point.
 */
* @param board is Parameters Taken from Board
* @param player is Parameter Taken from PlayerOne/Two()
* @param columnToPlaceBomb the Column the
  * Player has Chosen to Place Bomb
*/
public static void detonateBomb(char[][] board, int player, int columnToPlaceBomb)
{
    if (player == 1) {
        if (columnToPlaceBomb == 0) {
            for (int row = 0; row < 5; row++) {
                if (board[row][columnToPlaceBomb] == '*') {
                    if (board[0][columnToPlaceBomb] == '*') {
                        board[0][0] = '.';
                        board[0][1] = '.';
                        board[1][0] = '.';
                        board[1][1] = '.';
                    }
                    else if (board[1][columnToPlaceBomb] == '*') {
                        board[0][0] = '.';
                        board[0][1] = '.';
                        board[1][0] = '.';
                        board[1][1] = '.';
                        board[2][0] = '.';
                        board[2][1] = '.';
                    }
                    else if (board[2][columnToPlaceBomb] == '*') {
                        board[1][0] = '.';
                        board[1][1] = '.';
                        board[2][0] = '.';
                        board[2][1] = '.';
                        board[3][0] = '.';
                        board[3][1] = '.';
                    }
                    else if (board[3][columnToPlaceBomb] == '*') {
                        board[2][0] = '.';
                        board[2][1] = '.';
                        board[3][0] = '.';
                        board[3][1] = '.';
                    }
                }
            }
        }
    }
}

```

```
board[4][0] = '';
board[4][1] = '';
    }
}
}
if (columnToPlaceBomb == 1) {
for (int row = 0; row < 5; row++) {
    if (board[0][columnToPlaceBomb] == '*') {
        board[0][0] = '';
        board[0][1] = '';
        board[0][2] = '';
        board[1][0] = '';
        board[1][1] = '';
        board[1][2] = '';
    } else if (board[1][columnToPlaceBomb] == '*') {
        board[0][0] = '';
        board[0][1] = '';
        board[0][2] = '';
        board[1][0] = '';
        board[1][1] = '';
        board[1][2] = '';
        board[2][0] = '';
        board[2][1] = '';
        board[2][2] = '';
    } else if (board[2][columnToPlaceBomb] == '*') {
        board[1][0] = '';
        board[1][1] = '';
        board[1][2] = '';
        board[2][0] = '';
        board[2][1] = '';
        board[2][2] = '';
        board[3][0] = '';
        board[3][1] = '';
        board[3][2] = '';
    } else if (board[3][columnToPlaceBomb] == '*') {
        board[2][0] = '';
        board[2][1] = '';
        board[2][2] = '';
        board[3][0] = '';
        board[3][1] = '';
        board[3][2] = '';
        board[4][0] = '';
        board[4][1] = '';
        board[4][2] = '';
    } else if (board[4][columnToPlaceBomb] == '*') {
        board[3][0] = '';
        board[3][1] = '';
        board[3][2] = '';
        board[4][0] = '';
        board[4][1] = '';
        board[4][2] = '';
    }
}
}
if (columnToPlaceBomb == 2) {
for (int row = 0; row < 5; row++) {
    if (board[0][columnToPlaceBomb] == '*') {
        board[0][0] = '';
        board[0][2] = '';
        board[0][3] = '';
        board[1][1] = '';
        board[1][2] = '';
        board[1][3] = '';
    } else if (board[1][columnToPlaceBomb] == '*') {
        board[0][1] = '';
        board[0][2] = '';
        board[0][3] = '';
        board[1][1] = '';
        board[1][2] = '';
        board[1][3] = '';
        board[2][1] = '';
        board[2][2] = '';
        board[2][3] = '';
    } else if (board[2][columnToPlaceBomb] == '*') {
        board[1][1] = '';
        board[1][2] = '';
        board[1][3] = '';
        board[2][1] = '';
        board[2][2] = '';
        board[2][3] = '';
    }
}
```

```

        board[3][1] = ' ';
        board[3][2] = ' ';
        board[3][3] = ' ';
    } else if (board[3][columnToPlaceBomb] == '*') {
        board[2][1] = ' ';
        board[2][2] = ' ';
        board[2][3] = ' ';
        board[3][1] = ' ';
        board[3][2] = ' ';
        board[3][3] = ' ';
        board[4][1] = ' ';
        board[4][2] = ' ';
        board[4][3] = ' ';
    } else if (board[4][columnToPlaceBomb] == '*') {
        board[3][1] = ' ';
        board[3][2] = ' ';
        board[3][3] = ' ';
        board[4][1] = ' ';
        board[4][2] = ' ';
        board[4][3] = ' ';
    }
}
}
if (columnToPlaceBomb == 3) {
    for (int row = 0; row < 5; row++) {
        if (board[0][columnToPlaceBomb] == '*') {
            board[0][2] = ' ';
            board[0][3] = ' ';
            board[0][4] = ' ';
            board[1][2] = ' ';
            board[1][3] = ' ';
            board[1][4] = ' ';
        } else if (board[1][columnToPlaceBomb] == '*') {
            board[0][2] = ' ';
            board[0][3] = ' ';
            board[0][4] = ' ';
            board[1][2] = ' ';
            board[1][3] = ' ';
            board[1][4] = ' ';
            board[2][2] = ' ';
            board[2][3] = ' ';
            board[2][4] = ' ';
        } else if (board[2][columnToPlaceBomb] == '*') {
            board[1][2] = ' ';
            board[1][3] = ' ';
            board[1][4] = ' ';
            board[2][2] = ' ';
            board[2][3] = ' ';
            board[2][4] = ' ';
            board[3][2] = ' ';
            board[3][3] = ' ';
            board[3][4] = ' ';
        } else if (board[3][columnToPlaceBomb] == '*') {
            board[2][2] = ' ';
            board[2][3] = ' ';
            board[2][4] = ' ';
            board[3][2] = ' ';
            board[3][3] = ' ';
            board[3][4] = ' ';
            board[4][2] = ' ';
            board[4][3] = ' ';
            board[4][4] = ' ';
        } else if (board[4][columnToPlaceBomb] == '*') {
            board[3][2] = ' ';
            board[3][3] = ' ';
            board[3][4] = ' ';
            board[4][2] = ' ';
            board[4][3] = ' ';
            board[4][4] = ' ';
        }
    }
}
if (columnToPlaceBomb == 4) {
    for (int row = 0; row < 5; row++) {
        if (board[0][columnToPlaceBomb] == '*') {
            board[0][3] = ' ';
            board[0][4] = ' ';
            board[1][3] = ' ';
            board[1][4] = ' ';
        } else if (board[1][columnToPlaceBomb] == '*') {
            board[0][3] = ' ';

```

```

board[0][4] = '-';
board[1][3] = '-';
board[1][4] = '-';
board[2][3] = '-';
board[4][4] = '-';
} else if (board[2][columnToPlaceBomb] == '*') {
board[1][3] = '-';
board[1][4] = '-';
board[2][3] = '-';
board[2][4] = '-';
board[3][3] = '-';
board[3][4] = '-';
} else if (board[3][columnToPlaceBomb] == '*') {
board[2][3] = '-';
board[2][4] = '-';
board[3][3] = '-';
board[3][4] = '-';
board[4][3] = '-';
board[4][4] = '-';
} else if (board[4][columnToPlaceBomb] == '*') {
board[3][3] = '-';
board[3][4] = '-';
board[4][3] = '-';
board[4][4] = '-';
}
}
}
}
}

```

```

if (player == 2) {
if (columnToPlaceBomb == 0) {
for (int row = 0; row < 5; row++) {
if (board[row][columnToPlaceBomb] == '*') {
if (board[0][columnToPlaceBomb] == '*') {
board[0][0] = '-';
board[0][1] = '-';
board[1][0] = '-';
board[1][1] = '-';
} else if (board[1][columnToPlaceBomb] == '*') {
board[0][0] = '-';
board[0][1] = '-';
board[1][0] = '-';
board[1][1] = '-';
board[2][0] = '-';
board[2][1] = '-';
} else if (board[2][columnToPlaceBomb] == '*') {
board[1][0] = '-';
board[1][1] = '-';
board[2][0] = '-';
board[2][1] = '-';
board[3][0] = '-';
board[3][1] = '-';
} else if (board[3][columnToPlaceBomb] == '*') {
board[2][0] = '-';
board[2][1] = '-';
board[3][0] = '-';
board[3][1] = '-';
board[4][0] = '-';
board[4][1] = '-';
}
}
}
}
}
}
}

```

```

if (columnToPlaceBomb == 1) {
for (int row = 0; row < 5; row++) {
if (board[0][columnToPlaceBomb] == '*') {
board[0][0] = '-';
board[0][1] = '-';
board[0][2] = '-';
board[1][0] = '-';
board[1][1] = '-';
board[1][2] = '-';
} else if (board[1][columnToPlaceBomb] == '*') {
board[0][0] = '-';
board[0][1] = '-';
board[0][2] = '-';
board[1][0] = '-';
board[1][1] = '-';
board[1][2] = '-';
board[2][0] = '-';
board[2][1] = '-';
board[2][2] = '-';
}
}
}
}
}

```

```

    } else if (board[2][columnToPlaceBomb] == '*') {
        board[1][0] = '*';
        board[1][1] = '*';
        board[1][2] = '*';
        board[2][0] = '*';
        board[2][1] = '*';
        board[2][2] = '*';
        board[3][0] = '*';
        board[3][1] = '*';
        board[3][2] = '*';
    } else if (board[3][columnToPlaceBomb] == '*') {
        board[2][0] = '*';
        board[2][1] = '*';
        board[2][2] = '*';
        board[3][0] = '*';
        board[3][1] = '*';
        board[3][2] = '*';
        board[4][0] = '*';
        board[4][1] = '*';
        board[4][2] = '*';
    } else if (board[4][columnToPlaceBomb] == '*') {
        board[3][0] = '*';
        board[3][1] = '*';
        board[3][2] = '*';
        board[4][0] = '*';
        board[4][1] = '*';
        board[4][2] = '*';
    }
}
}
if (columnToPlaceBomb == 2) {
    for (int row = 0; row < 5; row++) {
        if (board[0][columnToPlaceBomb] == '*') {
            board[0][1] = '*';
            board[0][2] = '*';
            board[0][3] = '*';
            board[1][1] = '*';
            board[1][2] = '*';
            board[1][3] = '*';
        } else if (board[1][columnToPlaceBomb] == '*') {
            board[0][1] = '*';
            board[0][2] = '*';
            board[0][3] = '*';
            board[1][1] = '*';
            board[1][2] = '*';
            board[1][3] = '*';
            board[2][1] = '*';
            board[2][2] = '*';
            board[2][3] = '*';
        } else if (board[2][columnToPlaceBomb] == '*') {
            board[1][1] = '*';
            board[1][2] = '*';
            board[1][3] = '*';
            board[2][1] = '*';
            board[2][2] = '*';
            board[2][3] = '*';
            board[3][1] = '*';
            board[3][2] = '*';
            board[3][3] = '*';
        } else if (board[3][columnToPlaceBomb] == '*') {
            board[2][1] = '*';
            board[2][2] = '*';
            board[2][3] = '*';
            board[3][1] = '*';
            board[3][2] = '*';
            board[3][3] = '*';
            board[4][1] = '*';
            board[4][2] = '*';
            board[4][3] = '*';
        } else if (board[4][columnToPlaceBomb] == '*') {
            board[3][1] = '*';
            board[3][2] = '*';
            board[3][3] = '*';
            board[4][1] = '*';
            board[4][2] = '*';
            board[4][3] = '*';
        }
    }
}
}
if (columnToPlaceBomb == 3) {
    for (int row = 0; row < 5; row++) {

```

```

        if (board[0][columnToPlaceBomb] == '*') {
            board[0][2] = '*';
            board[0][3] = '*';
            board[0][4] = '*';
            board[1][2] = '*';
            board[1][3] = '*';
            board[1][4] = '*';
        } else if (board[1][columnToPlaceBomb] == '*') {
            board[0][2] = '*';
            board[0][3] = '*';
            board[0][4] = '*';
            board[1][2] = '*';
            board[1][3] = '*';
            board[1][4] = '*';
            board[2][2] = '*';
            board[2][3] = '*';
            board[2][4] = '*';
        } else if (board[2][columnToPlaceBomb] == '*') {
            board[1][2] = '*';
            board[1][3] = '*';
            board[1][4] = '*';
            board[2][2] = '*';
            board[2][3] = '*';
            board[2][4] = '*';
            board[3][2] = '*';
            board[3][3] = '*';
            board[3][4] = '*';
        } else if (board[3][columnToPlaceBomb] == '*') {
            board[2][2] = '*';
            board[2][3] = '*';
            board[2][4] = '*';
            board[3][2] = '*';
            board[3][3] = '*';
            board[3][4] = '*';
            board[4][2] = '*';
            board[4][3] = '*';
            board[4][4] = '*';
        } else if (board[4][columnToPlaceBomb] == '*') {
            board[3][2] = '*';
            board[3][3] = '*';
            board[3][4] = '*';
            board[4][2] = '*';
            board[4][3] = '*';
            board[4][4] = '*';
        }
    }
}

if (columnToPlaceBomb == 4) {
    System.out.println("column 4");
    for (int row = 0; row < 5; row++) {
        if (board[0][columnToPlaceBomb] == '*') {
            board[0][3] = '*';
            board[0][4] = '*';
            board[1][3] = '*';
            board[1][4] = '*';
        } else if (board[1][columnToPlaceBomb] == '*') {
            board[0][3] = '*';
            board[0][4] = '*';
            board[1][3] = '*';
            board[1][4] = '*';
            board[2][3] = '*';
            board[2][4] = '*';
        } else if (board[2][columnToPlaceBomb] == '*') {
            board[1][3] = '*';
            board[1][4] = '*';
            board[2][3] = '*';
            board[2][4] = '*';
            board[3][3] = '*';
            board[3][4] = '*';
        } else if (board[3][columnToPlaceBomb] == '*') {
            board[2][3] = '*';
            board[2][4] = '*';
            board[3][3] = '*';
            board[3][4] = '*';
            board[4][3] = '*';
            board[4][4] = '*';
        } else if (board[4][columnToPlaceBomb] == '*') {
            board[3][3] = '*';
            board[3][4] = '*';
            board[4][3] = '*';
            board[4][4] = '*';
        }
    }
}

```

```
    }  
    }  
    }  
    }  
    printConnectBoard();  
    System.out.println();  
    }  
}
```

Programming, Parts One and Two

GRADEMARK REPORT

FINAL GRADE

72/100

GENERAL COMMENTS

Instructor

PART ONE

=====

Engagement with Literature Skills (25) **17**

Knowledge and Understanding Skills (10) **7**

Cognitive and Intellectual Skills (10) **7**

Transferable Skills (5) **4**

Practical Application Skills (50) **37**

Part 1 Software Testing

Report started off very promisingly but then tailed off and finished rather abruptly. I did like the discussion of the technologies e.g. Gherkin, etc., but the overall testing process used in the DVLA is not very clear from this report.

No references provided.

Code Part1 Number Sets

a. Display numbers in sets COMPLETE

b. Determine max and min COMPLETE

c: Calculate average COMPLETE

d: Determine overlap COMPLETE

e: Intersection COMPLETE

f: Set difference INCORRECT

Use of JUnit YES, SIMPLE FUNCTIONS HAVE BEEN TESTING

Use of Git YES

Use of Java Doc YES

Code quits after you choose any menu option, it should only quit if you choose to do so. Set difference functions incorrectly. More comments required in code, especially in more complicated functions.

PART TWO

=====

Engagement with Literature Skills (25) **8**

Knowledge and Understanding Skills (10) **8**

Cognitive and Intellectual Skills (10) **7**

Transferable Skills (5) **4**

Practical Application Skills (50) **37**

Part 2 Design Principles

Your references are far too web-based and lack academic credibility as a result.

The discussion in your report is good.

You tread the line uncomfortably close to unfair practice in a few places - there are two chunks on page 17 in particular where material clearly should've been quoted.

The report is beautifully-presented, though there

are some minor issues with phrasing in places.

Code Part 2 Connect 4

Display board COMPLETE

Enter column COMPLETE

Check column position is empty COMPLETE

Drop counter to position COMPLETE

Check win/lose COMPLETE

Blitz feature COMPLETE

Time bomb feature COMPLETE, WITH BUG

Use of JUnit YES, A BIT BRIEF

Use of Git YES

Use of Java Doc YES

Usability of game could have been better, when do you press 'b'?? There is far too much "manual code" checking that should have been performed in a loop.

Overall Comments

A number of Git commits made with branching. The time bomb leaves a floating counter because it doesn't move any counter above those that have been destroyed.

Marks Part One: **72**

Marks Part Two: **64**

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50
