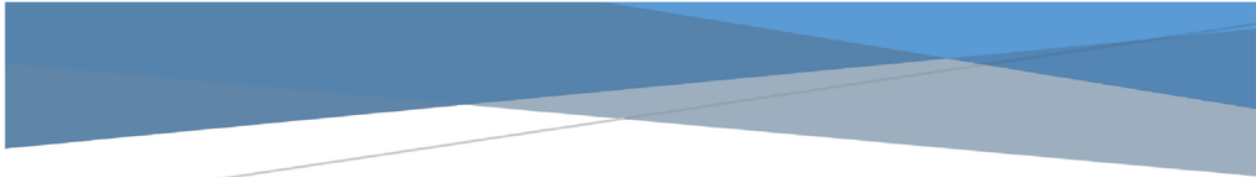


Final

by Anthony Njonge

FILE	12049_ANTHONY_NJONGE_FINAL_159704_1390105382.DOCX		
TIME SUBMITTED	11-JAN-2019 10:08PM (UTC+0000)	WORD COUNT	9827
SUBMISSION ID	98858422	CHARACTER COUNT	58770



Java Programming and Testing Part 1 and 2
ACCA7000

Anthony Njonge 1809703
This is a report for the Programming and Software Testing module assessment due
11/01/2019

Contents

Figures:	2
1. Part 1	3
1 Testing	3
1.1 Introduction	3
1.2 DVLA Testing	3
1.3 DVLA Tests Techniques:.....	7
2 Sets Code:	11
2.1 Junit Tests:	12
2.2 Sets Code Ideas to Development:	15
2. Part 2	17
2.1 Software Development – Cohesion, Coupling, Data Hiding, and Information Hiding ...	17
2.2 Connect 4 Code	20
2.2.1 Junit Tests:	20
2.2.2 Connect 4 Code Ideas to Development:.....	29
References	30
Appendix	32
Part 1 Code	32
Set Code JUnit Tests	39
Part 2 Code	41
Connect 4 Code JUnit Tests.....	51

Figures:

Figure 1 DVLA Testing Philosophy [8].....	5
Figure 2 DVLA Test Types Best Practise by vol.....	6
Figure 3: JUnit test initialisatiion	12
Figure 4:maxArray Negative Test.....	13
Figure 5 maxArray Positive Test	13
Figure 6 minArray Negative Test.....	14
Figure 7 minArray Positive Test	14
Figure 8: averageArray Negative Test.....	15
Figure 9 averageArray Positive Test	15
Figure 10: Connect 4 JUnit Initialisation	20
Figure 11: startBoard Negative Test.....	21
Figure 12:startBoard Positive Test	21
Figure 13 checkColThenAddPlay Negative Test.....	22
Figure 14 checkColThenAddPlay Positive Test	22
Figure 15 checkWin Negative Test	23
Figure 16 checkWin Positive Test	24
Figure 17 draw Negative Test	25
Figure 18 draw Positive Test.....	26
Figure 19 timeBomb Negative Test	27
Figure 20timeBomb Positive Test	29

1. Part 1

1.1 Testing

1.1.1 Introduction

The **defination** of testing is formally defined depending on a “view of the world”. One definition is that “Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Or testing is the process of executing a program or system with the intent of finding defects [1]”. This defines a view where the testing is done to show that the user requirements are met, resulting in a positive testing viewpoint. The latter definition infers to the actions during testing as a finding exercise of poking around identifying failures, errors, or bugs. The latter results in being a negative testing viewpoint.

Where a system is considered to be of high risk, the testing definition also evolves to capture the risk-based approach during testing. Risk-based testing ensures that the product risk is mitigated, managed and where necessary, a contingency employed.

Although the aim of testing is to find and eliminate bugs. McConnell [2], informs that finding defects is the easy part while correcting it is the hardest part. It is acknowledged that even with the best will in the world, research shows that even with test heavy removal techniques and methodology eliminates 90% of errors [3]. According to Sommerville [4], roughly 40% of all software development cost is testing costs. With such costs allocated to testing, the success of testing is crucial to software development regardless of whether it is being done in the public or private space.

1.1.2 DVLA Testing

DVLA undertakes software testing during the software development cycle in line with Agile methodologies. The aim is to ensure that delivered software for use by the public meets the threshold of value of money tests and meets the Government Digital Service requirements. To reduce testing cost, the DVLA has undertaken software development under an Agile methodology mainly using Scrum and Kanban systems of development with an Acceptance Test Driven Development and Behaviour Driven Development mentality. Recognition is given to the general principle of software quality that improving quality reduces development costs overall [2].

However, it is the limitations of developer testing that a quality assurance team role is embedded throughout the development process since as highlighted by McDonnell [2] and Beizer [5], developers:

- tests tend to be “clean” rather than test using “dirty” tests that break their code
- testing tends to be optimistic of the test coverage
- testing skips more sophisticated kinds of test coverage.

The development setup at the DVLA involves a squad setup, team, as follows:

1 Lead Software Engineer, 5 Software Engineers

2 Testers – quality assurance team, 2 BAs

And with the following roles covering two squads:

1 Solution architect, 1 Agile Delivery Manager, 1 Delivery Manager [6].

Testing is undertaken under the Quality Assurance(QA) viewpoint where the main reason for testing is to satisfy the strict legal undertakings of delivering government software. Functional and non-functional testing is undertaken as part of DVLA testing. The use of functional and non-functional tests meets the aim of an agile methodology approach that the DVLA IT Strategy [7] promotes. Documentation under this approach is dependent on the judgement of the product owners but as a minimum the detail of the test approach and application specific detail is dependent upon the test approach. The agreed 'best fit' approach is encouraged so that it does not constrain or hinder development in the team but rather provide necessary support as the default pragmatic approach. The teams in the DVLA ensure that any testing done delivers on the idea of “inspect and adapt”.

It is acknowledged that it is easier to meet a “happy path” testing criteria where tests are carried out to meet an “Acceptance Criteria” threshold but difficult to prove the non-existence of errors, bugs, and defects inherent in development. It is acknowledged that even after testing, residual bugs will exist but the aim of the QA team is to minimise, if not eliminate serious errors through the use of proven test suites, cases, and techniques. DVLA testing process can be summarised under the following headings:

- a) Test types
- b) Test designs
- c) Test automation
- d) Test deliverables
- e) Risk and mitigation
- f) Acceptable standards

The scope of testing activities is such that the QA team undertakes more than what is captured in the user stories or functional requirements including:

- Features and components design
- Documentation – technical, code and test evidence
- Development process – using agreed standards, pair testing, code reviews etc.

At all times through the testing process, the team follows a risk-based, consultative, collaborative and proactive approach. The test process follows a philosophy that the delivered functionality of the completed software adds value to the citizenry where the perspective of value is that it meets the government expectations and users' expectation in terms of quality, performance, and security. The test processes depend on individual projects but always will align with the development lifecycle as far attainable as possible. To avoid the 'digital hug' syndrome prevalent within a tool-centric approach [8], a closer inspection of an approach and employed methodology

is done in line with internal policy. The testing process is the “oil” that ensures the smooth running of the software development lifecycle process, the testing is always intended to align and fit in into the overall lifecycle process.

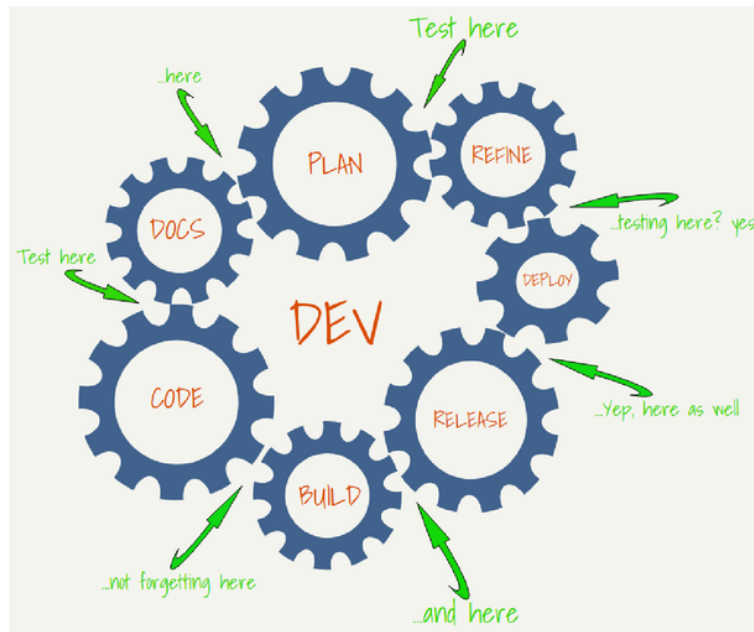


Figure 1 DVLA Testing Philosophy [8]

To ensure the success of testing, everyone in the squad is involved in line with the Agile methodology from the earliest opportunity in refinement sessions, three amigos [9] sessions, Sprints, and feedback and consultation meetings. The idea of such an approach is to encourage the identification of issues and get clarifications from “the horse’s mouth” on business expectation. This approach also lends itself to a behaviour driven development methodology that use tools such as Gherkin [10]. Example mapping is utilised where appropriate and the acceptance criteria [8] expressed in Gherkin where possible as part of making a user story ready. To ensure the participants understand the definitions used, a testing definitions library and glossary are maintained in Jira for ease of access to everyone. The definitions follow the BS7925-1 [11] and or ISO/IEC/IEEE 29119-1 [12]. The two most frequent referenced definitions relate to the following questions:

1. Is the DVLA building the product right?
2. Is it in line with GDS standards? [13] and is the DVLA building the right product?

The former relates to verification while the latter relates to validation.

The DVLA acceptable minimum standards for each delivered product should meet:

- a) Documentation:
For any software development done, relevant documentation must be available for audit purposes. It should provide enough detail for the business

to understand the developed software. As a minimum, the code should be commented with sufficient detail and at a minimum, a readme is included in the repository [8]. All test activity must include an audit trail that traces its origin from a user story. Automated tests should contain the minimum readme that documents the overview of functionality, list of tests, setup and running instruction and any relevant related information. Manual tests are also performed and documented with scenarios tested. Exploratory testing done informs on the mind maps, debrief sessions, observations and meeting summary. A test plan is maintained to show the detail of tester independence, the environment of testing, test case design, rationale, and results.

b) Test Designs:

This provides at the start the manner the tests are carried out. The designs include a summary and a clear statement on any dependencies. A mind map that details the idea of the test areas is captured. Any feedback and conversations within the team on the exploratory sessions are also captured. A spreadsheet is maintained for audit purposes that shows the inputs and outputs. Gherkin statements capture the idea of the BDD and the acceptance criteria. This design is done within the team to allow for peer review coverage. Any re-reviews or corrective actions are undertaken by experienced QA team members [8].

c) Test Types:

The best practice model of the test types undertaken by volume is represented by Figure 2 below.

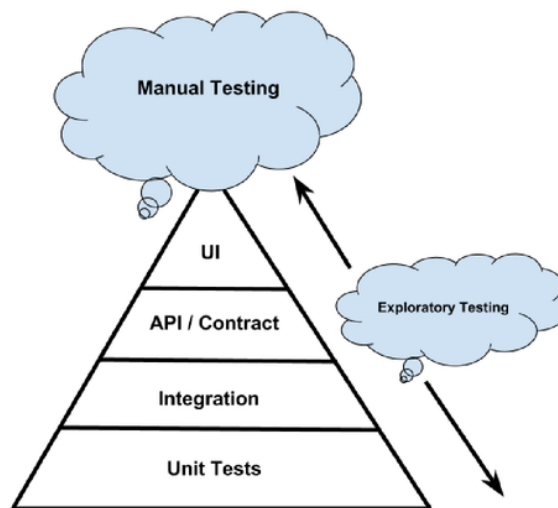


Figure 2 Best Practise Tests Pyramid [8]

d) Test Automation:

Test automation is not in on itself implicit of a mark of quality software and no defects [8]. Automated tests are undertaken to support the software development lifecycle adopted by the DVLA. The test code standard is held to the highest quality scrutiny as if the code were at the production level. All the

tests are required to be independent to allow for audit trail investigations and future-proofing to provide a reference for maintenance with an artifact and test data available. All assertions are commented clearly to define how they meet the functional requirements. The acceptance criteria which drive most test automation is documented in a readme file.

e) Test deliverables:

The testing is to be approached within each of the identified test phases and can take the form of a plan or an informal document (A plan on a page) shared with other members of the squad. Details of the test design, test cases, and test scripts are expected to be part of the deliverables. The following items form part of the deliverables:

- a) test procedure(s) – which defines what and how the tests have run.
- b) test schedule: details of what and when tests have been run. This includes details of the outputs of continuous integration (unit and integration tests) and continuous delivery (pipelines used).
- c) test logs: although sometimes considered optional, when produced they include detail of the test results outcome i.e. was it successful, or did it fail.
- d) test completion report: this is produced at the end of each test phase and details results and conclusions drawn on the completed test run. This document acts as a summary to any stakeholders to get a view of the reports. The report is written to offer an opinion that can be translated to show the business value delivered.
- e) test data: Data that is used to deliver the software is kept to be reused for assurance. The data is also used to evidence testing has been done for auditing purposes and later used for future regression testing.
- f) debriefs: This is either a formal meeting or informal depending on the type of development.

1.1.2 DVLA Tests Techniques:

The DVLA undertakes multiple tests covering both functional and non-functional tests. This report provides the findings using the 5Ws and H method of what, why, who, where, when and how.

1.1.2.1 General Testing

Positive and Negative

What: The testing involves verification with known positive and negative bias.

Why: The purpose of running the negative and positive testing is to ensure that the working of the developed software meets the needs as requested by the business (product owner) through the user stories.

Who: Positive and Negative tests are done by the developers in the first instance as they develop the software, using their informed skilled knowledge ensuring the coverage exceeds the happy path picture only coverage.

Where: In the squads during the sprints. The developers ensure that for the handover of a completed user story to the QA for testing, they have done tests.

When: Before moving a user story to the test section of the storyboard.

How: Using testing frameworks at code development

Unit testing

What: During development, the code is tested by the developers.

Why: The purpose of running unit testing is to ensure that the errors, bug or defects are captured very early in code lifecycle.

Who: Developers make use of testing frameworks to undertake unit testing.

Where: The developers ensure that all completed user stories have unit tests done and QA are obliged to decline further testing unless unit testing is done.

When: Code development stage.

How: Use of unit frameworks helps make it run easier and faster.

Automated Software Testing

What: This is undertaken through the use of BDD tools incorporated in the CI/CD platform

Why: This allows the test to undertake acceptance testing and integration testing.

Who: The QA team only works on user stories that have been delivered in the tests workflow. The QA team undertake unit tests for specific test cases according to a feature.

Where: This is done before the peer-review to move the user story to the Definition of Done section on the Kanban board.

How: Using RSpec enterprise framework, Gherkin, Cucumber and use of Jenkins.

1.1.2.2 Functional Testing:

Functional testing techniques ensure that the software (application under tests) meets the functional requirements as set out in the user stories.

Boundary Analysis Testing

Boundary Analysis as defined [11] [14], tests that inputs and outputs at identified boundaries where each boundary is defined undergoes a test task.

What: Tests the interaction at the edge of the system using input and outputs.

Why: These tests are used especially where mathematical operations are being done to system input and outputs.

Who: Everyone is involved in doing this. It is done with other tests such as unit testing, manual test, and automated tests.

Where: Boundary testing at all levels during development until deployment is done.

When: This is done all through development but special attention is given during acceptance testing phases.

How: The use of available test cases is used for "interesting" boundary values.

Integration Testing

What: The tests are done using black box techniques.

Why: The tester performs integration testing against the functional requirements of the software.

Who: The testing team does these tests since they are independent in the actual coding.

Where: During the code review stage before delivery.

When: The testing is done to complement the software design in the acceptance criteria of the user story.

How: The test case design that is used with integration testing should give confidence that there exist correct interfacing and interaction between modules and part of the system. This test should not be a duplication of any unit test done previously.

Regression Testing

What: The tests check on the whole system working together.

Why: Regression testing is used to ensure that when the developed system is being integrated to the whole, the whole system works as expected but with the new functionality.

Who: The testing team does these tests as automatic tests with test cases using previously known peculiarities.

Where: This is done before exploratory testing.

When: The testing is done to give assurance that any dependencies and the functionality of the new system work.

How: The test case for regression is done automatically where the tests cover the whole system in total including the new functionality. Ideally, regression testing is done whenever any new system is enjoined with the full system. A review should be done at the end of the testing phase to capture cases that could be added to the test cases.

1.1.2.3 Non- functional Testing

Non-functional testing techniques ensure that software (application under tests) meet other "reality" requirements, for example, the architecture, technology, load requirements, security, and other non-functional requirements.

System Operation Testing

What: This is done to ensure that the software will work under pre-production environment conditions.

Why: This ensures that the system, when deployed to production, will work as expected.

Who: The test analyst with the help of the business will run test cases to simulate operation.

Where: This is done under a pre-production environment condition.

When: This is done after user acceptance testing has been done but before “live” delivery.

How: This involves the use of “close to live” data as possible.

Usability Testing including A/B Testing

What: Software is presented to the public for use and the usability recorded and feedback captured on which presentation they preferred.

Why: Get public feedback on the usability of delivered software.

Who: This is undertaken by a selected group of users, including members of the public.

Where: This testing is undertaken in a purpose-built lab that members of the public are invited in to test software.

When: Members of the public are invited to test a product after acceptance by the product owner and before the roll-out to the wider public. The public could be testing the software at alpha or beta stages.

How: The process is managed by the Customer Insight and Diversity Team [15].

1.2 Sets Code:

The code for this part, including a relevant JAVADOC folder, has been shared through the repository

<https://mwirikia@bitbucket.org/mwirikia/programmingacca7000.git>.

A test is a check of whether the code works as expected or not, and to be able to identify what, why and where the issue has occurred and allowing capturing of any deviations from an expected outcome. An error in the code is known as a bug. The ultimate aim for any testing is to find and identify, resolve and eliminate bugs. In software, the tests will involve both functional and non-functional requirements and include a wider range of tests and types of tests.

The most basic testing that can be done is unit testing. Unit testing involves testing the code, usually the methods, in isolation [16]. Integrating testing into software development allows for the easy adoption of agile methodologies which allow for iterative development. Test driven development(TDD) in agile gives confidence that the developed code meets the goal of development and is of high quality through the elimination of bugs. The basic tenet of TDD can be summarised as red, green then refactoring. Where the red signifies test failing, and green signifies passing of the tests and refactoring involves cleaning up of the code and design and test and restarting the cycle again. The outcome of this cycle is that the unit tests drive the direction of the design of the code [17].

For developers can choose to run the tests manually or automated. For automation, this can be done through code-driven unit testing frameworks that can be easily integrated into the working environments.

For this report, all the tests have been done using JUnit. JUnit is a regression testing framework that is used to undertake unit testing in Java [18], allowing for a quicker programming experience and an improvement in the quality of code. The tests below have been developed to include a negative and positive case test.

1.2.1 Junit Tests:

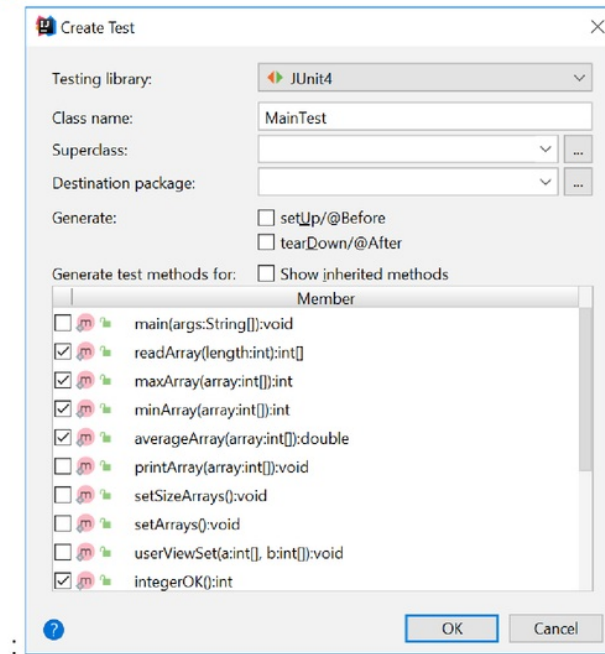


Figure 3: JUnit test initialisation

For the void methods, no JUnit Tests were done. These methods were the readArray, printArray, setArrays, userViewSet, intersection, setDiff, entry, userViewSet. Also not having JUnit test included methods that include user input, such as the setSizeArrays, userEntry, readArray, and integerOk method.

maxArray Test:

Negative Case:

@Test

```
public void maxArray() {  
    int length = 3;  
    int[] array = new int[length];  
    array[0] = 1;  
    array[1] = 3;  
    array[2] = 7;  
  
    int check;  
    check = Main.maxArray(array);  
  
    assertEquals(2, check);  
}
```

Output:

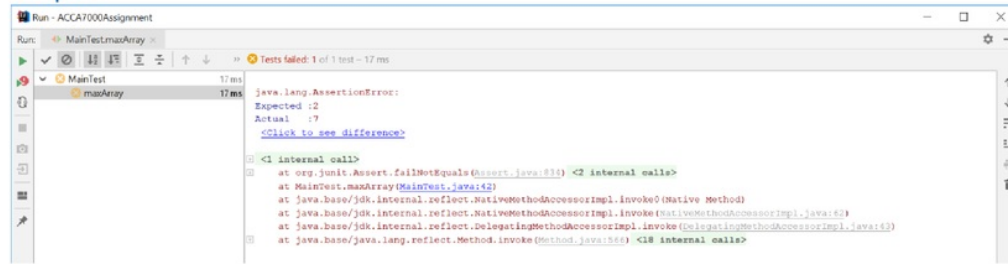


Figure 4: maxArray Negative Test

Positive Test Case:

```
public void maxArray() {  
    int length = 3;  
    int[] array = new int[length];  
    array[0] = 1;  
    array[1] = 3;  
    array[2] = 7;  
  
    int check;  
    check = Main.maxArray(array);  
    assertEquals(7, check);  
}
```

Output



Figure 5 maxArray Positive Test

minArray Test

Negative Case:

@Test

```
public void minArray() {  
    int length = 3;  
    int[] array = new int[length];  
    array[0] = 1;  
    array[1] = 3;  
    array[2] = 7;  
    int check;  
    check = Main.minArray(array);  
}
```

```

    assertEquals(7,check);
}

```

Output:

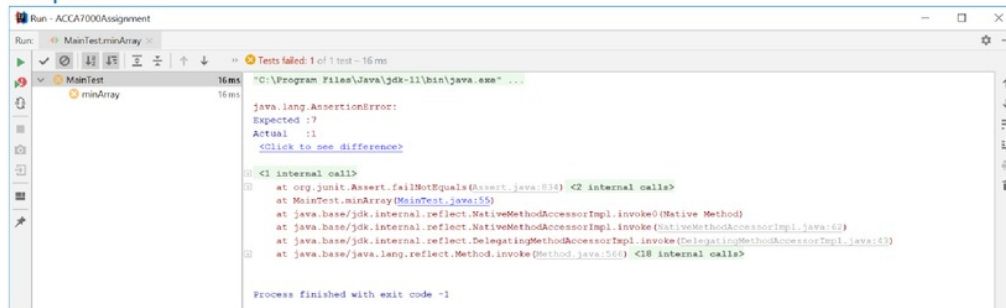


Figure 6 minArray Negative Test

Positive Case:

```

@Test
public void minArray() {
    int length = 3;
    int[] array = new int[length];
    array[0]= 1;
    array[1] = 3;
    array[2]= 7;
    int check;
    check = Main.minArray(array);
    assertEquals(1,check);
}

```

Output:



Figure 7 minArray Positive Test

averageArray Test

Negative Case:

```

@Test
public void averageArray() {
    int length = 3;
    int[] array = new int[length];
    array[0]= 1;
    array[1] = 3;
    array[2]= 7;
    double check;
    check = Main.averageArray(array);
}

```

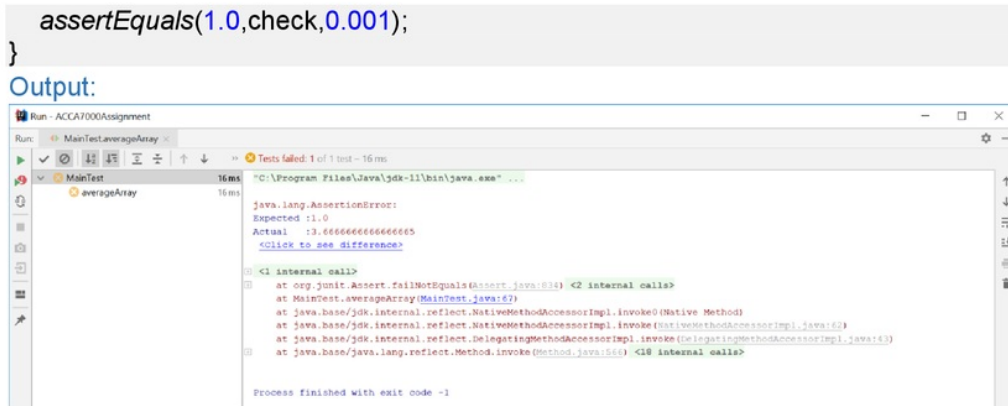


Figure 8: averageArray Negative Test

Positive Case

```

@Test
public void averageArray() {
    int length = 3;
    int[] array = new int[length];
    array[0]= 1;
    array[1] = 3;
    array[2]= 7;
    double check;
    check = Main.averageArray(array);
    assertEquals(3.667,check,0.001);
}

```



Figure 9 averageArray Positive Test

1.2.2 Sets Code Ideas to Development:

The ideas to the development process involved the following steps:

1. Thinking about how to get user input – this was decided to include a Scanner outside the main method.
2. Capture user input – to be able to capture the user input, it was important to think about the data type being entered by the user. This required finding a way of how to accept only integers, validation.

3. Use the user input to declare the sets and input values into the sets- having captured the user input, it was possible to use the input to initialise the set size and accept the set values.
4. Declaring the sets – as the sets contained integer it was important to declare int arrays.
5. Print the contents of the sets – to print the sets it involved reading into the set arrays rather than usual System.out statement.
6. Calculate the min, max, average – once it was able to loop through the sets, next was to calculate the min and max and average by simple mathematical calculations.
7. Work out the intersection – to calculate the intersections involved using the knowledge garnered from the previous exercise and using the min and max values to decide the intersections of the sets.
8. Work out the set difference – this was the most challenging part, but figured out that use of a non-matching counter to determine if the counter matched the count of the second set and increment the counter. If the counter was greater than 0 then a match had been found and only the non-matched values were printed out as the set loop went round and reset at the end of set 2.

2. Part 2

2.1 Cohesion, Coupling, Data Hiding, and Information Hiding

Software has developed over the years to become increasingly and inherently complex. With every technological improvement in terms of memory capacity and processing power, software and its related systems have increased exponentially too [19, 20]. These systems reflect things or processes in the real world, and to better understand them, it is helpful to break them down into a group of smaller parts. These smaller parts would represent objects that work together to achieve the aims and goals of the system.

With the division of systems into small parts, it helps that those small parts are defined to have a clear unique job. These parts are also known as components. Some attributes of components are that they should be isolated, reusable and customizable. Any component that can be defined at a programming level is known as a module.

To achieve a working system, good software design is required. Qualities that are needed to be met include usability, efficiency, reusability, maintainability, and reliability. Software design in simplicity could be broken down as either through traditional structured design or through object-oriented analysis and design (OOAD). OOAD is a current method used through the use of models in developing modern software.

This model is collectively known as the object model. The object model has defining principles that include concurrency, persistence, typing, modularity, hierarchy, abstraction and encapsulation. These can further be divided into major and minor parts of the model. The major principles being abstraction, encapsulation, modularity, and hierarchy and minor principles being typing, concurrency, and persistence.

The successive development of programming languages has mirrored the development of complex systems. Over time a class of programming languages that complemented the object model developed. Some of these programming languages included Java and C++. Use of these languages has allowed the development of systems into classes and where those classes are broken down into methods.

Abstraction:

Abstraction is a simplification of an idea by ignoring different characteristics and specific details while putting the emphasis on only common basic essential characteristics and detail. For an abstraction to be successful, the rule of least astonishment is followed which states that essential attributes and behaviours should be fully captured to ensure no surprise and no definitions fall outside the scope of the system [21].

Cohesion:

Cohesion is a measure of how well a component works as part of the system, its connectivity. Cohesion is, therefore, an attribute of a component. There are different types of cohesion. These are functional, informational, object, communicational,

sequential, procedural, logical association and coincidental. Functional and informational and object cohesion are considered as high, strong and desired while logical association and coincidental are considered low, weak and undesirable in a system design. A module that does only one task only is considered to have high cohesion [22] [21] and should be the aim of design. Each module should have a specific set of closely related tasks that are well defined [23]. The least desired cohesion is coincidental cohesion where unrelated abstraction exists [20].

Coupling:

Good software design should allow for modules in the system to work well separate and in harmony with other modules when the interaction occurs. Coupling is concerned about the complexity of the modules in relation to other modules within the system. Coupling can, therefore, be defined as tight(strong, bad) or loose(good). To evaluate the coupling of a module, consideration is given to the degree, ease, and flexibility. The degree defines the number of connection to other modules in the system. The ease is how connections to other modules are evident. The flexibility is defined as the level of how other modules in the system could be used in place of the current module and also how those modules could be replaceable with others.

A tightly coupled module defines a module that is tied to the working of another module. The best example to explain it is content coupling which is when one module references another. This is considered bad design. Loose coupling is defined as a module that works separate from other modules in the system and if it requires interaction with others, it does so through a well-defined interface. The best example of loose coupling would be data coupling. If data coupling exists then drawbacks of the other kinds of coupling i.e. content, common, stamp and control are non-existent. Loose coupling is highly desired as tight coupling makes the overall system difficult to understand, maintain, update or change [22] [21]. Coupling is, therefore, a measure of accessibility which is a quality of good software design [20] with loose coupling preferred.

Information hiding:

A program class is a collection of data and methods.

Information hiding is the principle that information is accessed only by modules that require that information [24]. Modules should be able to access this information if required through interfaces. A general goal of good design is that information that is sensitive is hidden. Booch et.al [20], defines information hiding as the process of hiding all the secrets of an object that are non-essential. The reason for hiding is to prevent unintended changes. An example of achieving information hiding in Java is through the use of access modifiers. These allow how information can be accessed by other parts of the system. Each level of access is different and unique. The levels are public, protected, default and private.

Information hiding is important because it restricts the data flow across classes, ensuring that data is hidden from an external view of a class. The idea as initially presented by Parnas [24] and summarised by Schach [25], information hiding is geared towards future maintenance.

Data hiding:

Access to data in the class is only available to methods within the class. Data hiding is, therefore, the binding of data within a class where access to the data is only through the methods [26]. Having the data within a class hidden allows for methods to only define operations that could only be done by what they can access. This gives the advantage of providing confidence that it works correctly every time. Another advantage of data hiding is when implementation of the class changes the interface does not require to change as the users of the method are not concerned about the detail of within the method but rather if used the method delivers the expected response.

In conclusion the preferred software development output should have high cohesion, loose coupling and implement data hiding and information hiding.

2.2 Connect 4 Code

The code including a relevant JAVADOC folder has been shared through the repository <https://mwirikia@bitbucket.org/mwirikia/programmingacca7000.git>.

2.2.1 Junit Tests:

Initialisation step for the JUnit tests in the IDE is captured in the next figure: Connect 4 JUnit initialisation.

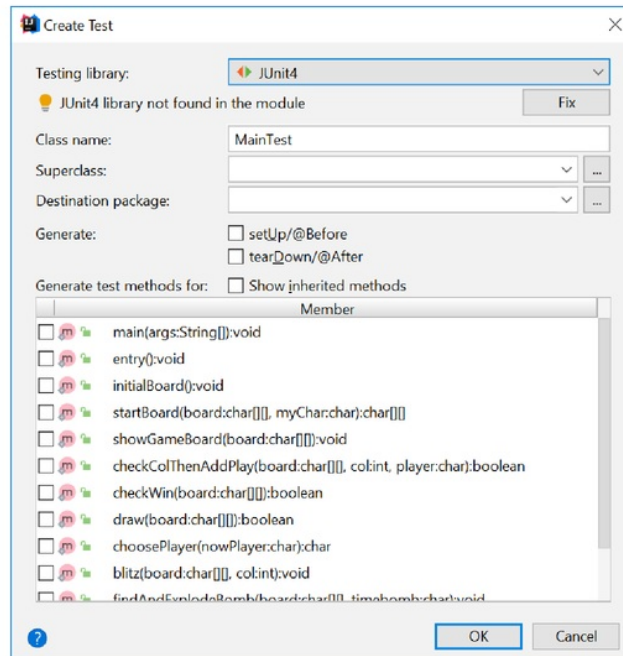


Figure 10: Connect 4 JUnit Initialisation

startBoard() Test

Negative Case:

`@org.junit.Test`

```
public void startBoard() {
    char[][] board = new char[6][6];
    char initial = '-';

    for(int i=0; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=initial;
        }
    }
    assertEquals(board[0][0], '*');
}
```



Figure 11: startBoard Negative Test

Positive Case:

```
@org.junit.Test
public void startBoard() {
    char[][] board = new char[6][6];
    char initial = '-';

    for(int i=0; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=initial;
        }
    }

    assertEquals(board[0][0],initial);
}
```

Output:



Figure 12: startBoard Positive Test

checkColThenAddPlay() Test

Negative Case:

```
@org.junit.Test
public void checkColThenAddPlay() {
    char[][] board = new char[6][6];
    char initial = '-';
    for(int i=0; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=initial;
        }
    }
    boolean play;
    if(board[0][0]=='X'){
```

```

        play=true;
    }else{
        play = false;
    }
    //user wants to play col 0
    assertTrue(play);
}

```

Output:



Figure 13 checkColThenAddPlay Negative Test

Positive Case:

```

@org.junit.Test
public void checkColThenAddPlay() {
    char[][] board = new char[6][6];
    char initial = '-';
    for(int i=0; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=initial;
        }
    }
    boolean play;
    if(board[0][0]=='-'){
        play=true;
    }else{
        play = false;
    }
    //user wants to play col 0
    assertTrue(play);
}

```

Output:



Figure 14 checkColThenAddPlay Positive Test

checkWin() Test

The test undertaken for a check for a win will need to be tested for horizontal, twice diagonal and vertical; four tests in total. The tests below will only check for a horizontal win only.

Negative Case:

```
public void checkWin() {
    char[][] board = new char[6][6];
    char initial = '-';

    for(int i=0; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=initial;
        }
    }
    board[5][0]='X';
    board[5][1]='X';
    board[5][2]='X';
    board[5][3]='O';

    boolean win;
    //horizontal
    if((board[5][0]=='X' && board[5][1]=='X' &&board[5][2]=='X' &&board[5][3]=='X' )
        ||(board[5][0]=='O' &&board[5][0]=='O' &&board[5][0]=='O'
        &&board[5][0]=='O')){
        win=true;
    }else{
        win = false;
    }

    assertTrue(win);
}
```

Output:

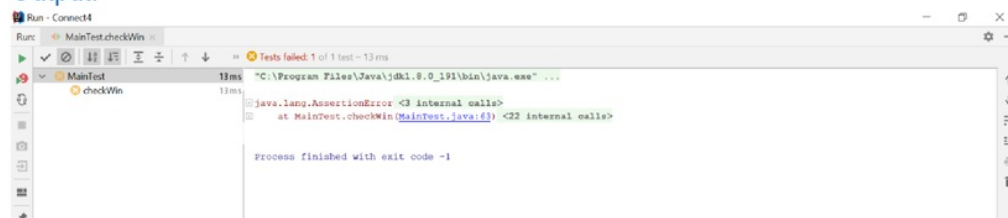


Figure 15 checkWin Negative Test

Positive Case:

```
public void checkWin() {
    char[][] board = new char[6][6];
    char initial = '-';

    for(int i=0; i<6;i++){
        for (int j=0; j<6;j++){
```

```

        board[i][j]=initial;
    }
}
board[5][0]='X';
board[5][1]='X';
board[5][2]='X';
board[5][3]='X';

boolean win;
//horizontal
if((board[5][0]=='X' && board[5][1]=='X' &&board[5][2]=='X' &&board[5][3]=='X' )
    ||(board[5][0]=='O' &&board[5][0]=='O' &&board[5][0]=='O'
&&board[5][0]=='O')){
    win=true;
}else{
    win = false;
}

assertTrue(win);
}

```

Output:



Figure 16 checkWin Positive Test

draw() Test

Negative Case:

```

@org.junit.Test
public void draw() {
    char[][] board = new char[6][6];
    char player1 = 'X';
    char player2 = 'O';
    int count1 = 0;
    int count2 = 0;

    for(int i=0; i<4;i++){
        for (int j=0; j<6;j++){
            board[i][j]=player1;
        }
    }
    for(int i=4; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=player2;
        }
    }
}

```

```

    }

    boolean draw;
    for(int i=0;i<6;i++){
        for(int j=0; j<6;j++){
            if(board[i][j]=='X'){
                count1++;
            } else{
                count2++;
            }
        }
    }

    if(count1==count2){
        draw=true;
    }else{
        draw = false;
    }
    assertTrue(draw);
}

```

Output

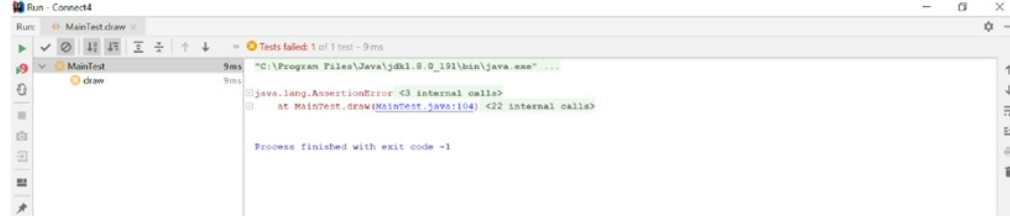


Figure 17 draw Negative Test

Positive Case:

```

@org.junit.Test
public void draw() {
    char[][] board = new char[6][6];
    char player1 = 'X';
    char player2 = 'O';
    int count1 = 0;
    int count2 = 0;

    for(int i=0; i<3;i++){
        for (int j=0; j<6;j++){
            board[i][j]=player1;
        }
    }
    for(int i=3; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=player2;
        }
    }
}

```

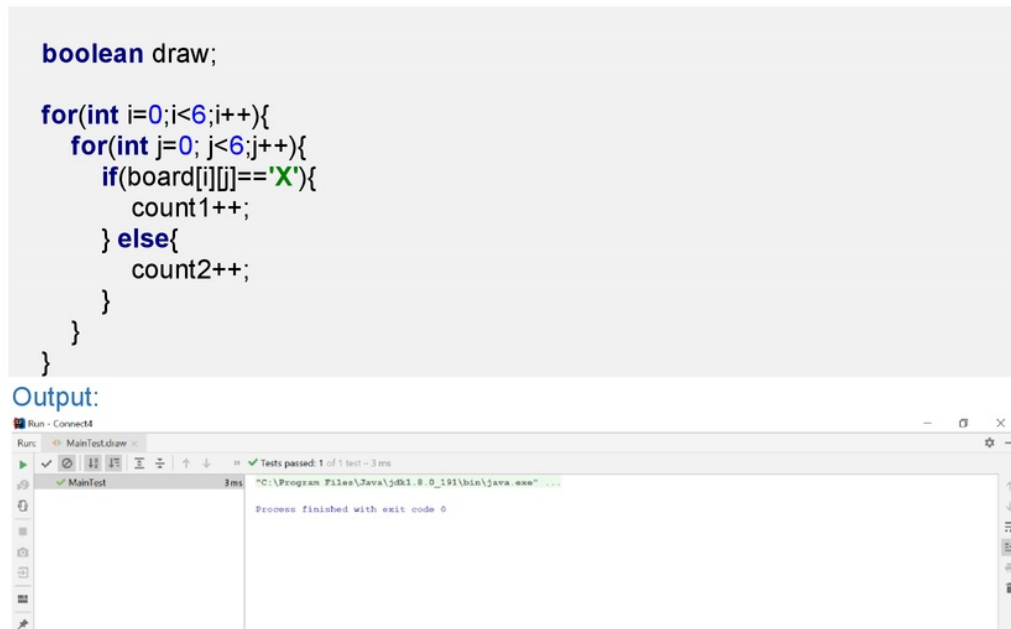


Figure 18 draw Positive Test

timeBomb() Test:

Negative Case:

@org.junit.Test

```
public void timeBomb() {
    char[][] board = new char[6][6];
    char timeBomb = '*';
    char initial = '-';
    char player1 = 'X';
    char player2 = 'O';
    int countinitial=0;
```

//initialise board as draw

```
for(int i=0; i<6;i++){
    for (int j=0; j<3;j++){
        board[i][j]=player1;
    }
}
```

```
for(int i=0; i<6;i++){
    for (int j=3; j<6;j++){
        board[i][j]=player2;
    }
}
```

//initialise a timebomb

```
board[2][3] = timeBomb;
```

//reset values around timebomb

```

for(int i=0;i<6;i++){
    for (int j = 0; j < 6; j++) {
        if (board[i][j] == timeBomb) {
            //board[i][j] = '-'; the timebomb still is unchanged
            board[i - 1][j - 1] = '-';
            board[i][j - 1] = '-';
            board[i + 1][j - 1] = '-';
            board[i - 1][j] = '-';
            board[i + 1][j] = '-';
            board[i - 1][j + 1] = '-';
            board[i][j + 1] = '-';
            board[i + 1][j + 1] = '-';
        }
    }
}

for(int i=0;i<6;i++){
    for(int j=0; j<6;j++){
        if(board[i][j]==initial){
            countinitial++;
        }
    }
}

boolean timeBombCount;

if(countinitial==9){
    timeBombCount=true;
}else{
    timeBombCount = false;
}

//this should pass for Positive Case
assertTrue(timeBombCount);
}

```

Output:



Figure 19 timeBomb Negative Test

Positive Case:

```

@org.junit.Test
public void timeBomb() {
    char[][] board = new char[6][6];
}

```

```

char timeBomb = '*';
char initial = '-';
char player1 = 'X';
char player2 = 'O';
int countinitial=0;

//initialise board as draw
for(int i=0; i<6;i++){
    for (int j=0; j<3;j++){
        board[i][j]=player1;
    }
}
for(int i=0; i<6;i++){
    for (int j=3; j<6;j++){
        board[i][j]=player2;
    }
}

//initialise a timebomb
board[2][3] = timeBomb;

//reset values around timebomb
for(int i=0;i<6;i++) {
    for (int j = 0; j < 6; j++) {
        if (board[i][j] == timeBomb) {
            board[i][j] = '-';
            board[i - 1][j - 1] = '-';
            board[i][j - 1] = '-';
            board[i + 1][j - 1] = '-';
            board[i - 1][j] = '-';
            board[i + 1][j] = '-';
            board[i - 1][j + 1] = '-';
            board[i][j + 1] = '-';
            board[i + 1][j + 1] = '-';
        }
    }
}

for(int i=0;i<6;i++){
    for(int j=0; j<6;j++){
        if(board[i][j]==initial){
            countinitial++;
        }
    }
}

boolean timeBombCount;

if(countinitial==9){
    timeBombCount=true;
}

```

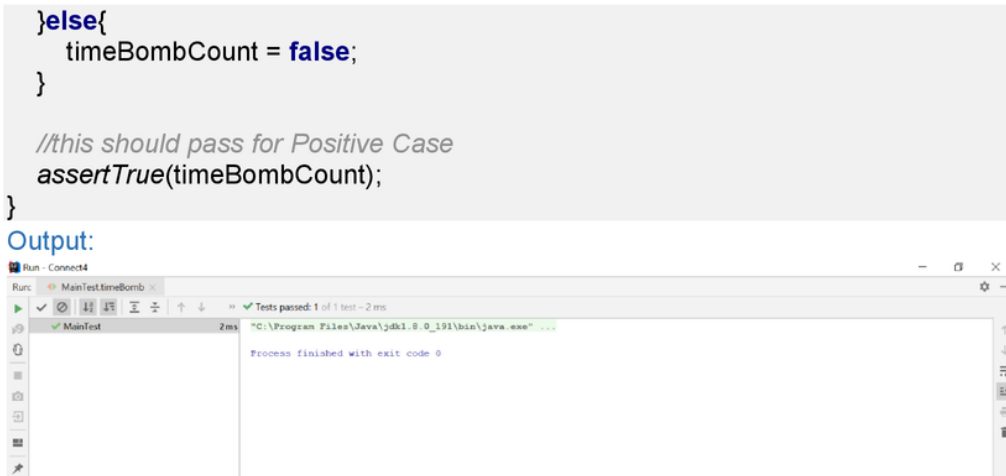


Figure 20timeBomb Positive Test

2.2.2 Connect 4 Code Ideas to Development:

The ideas to develop Connect 4 Code involved the following steps:

1. Think about how to get user input – this was decided to include a Scanner at outside the main method.
2. Initialising the board involved initialising a char board of set width and height.
3. Capture user input – to be able to capture the user input, it was important to think about the data types being entered by the user and to allow both characters 'b', 'B', 't', 'T' and the integers 1-6.
4. Use the user input to determine what action to take whether to blitz for b, t for time bomb or choose column number to play turn. User input will determine what part of the switch case to play.
5. Dropping the counters to the end of the array depended on figure how to place the counters at the last row first of the board.
6. Next, a method was required for figuring out whether the move is legitimate. Was a column full? The check involved how to read the char at a position in the char[][] board.
7. Print the board after every move would involve a print char loop.
8. The prompt for the blitz would involve capturing a user choice of either char 'b' or 'B'. Next, it should involve having a counter that will be used to check if the player has the one blitz move available. Next, a check play method will be checked to confirm move is legitimate.
9. The prompt for the time bomb would involve capturing either 't' or 'T' choices from user to play the time bomb. Like the blitz method, a counter will have to be checked to confirm that the play still has one move for a time bomb. Then allow the player to play the desired column then check if the move is legitimate.
10. The time bomb should then explode after a few moves.

References

- [1] J. Watkins, *Testing IT - An Off the Shelf Software Testing Process*, Cambridge: Cambridge University Press, 2004.
- [2] S. C. McConnell, *Code Complete 2*, Redmond: Microsoft Press, 2004.
- [3] C. Jones, *Software Engineering Best Practices*, New York: The McGraw-Hill Companies, 2010.
- [4] I. Sommerville, *Software Engineering - Tenth Edition (Global Edition)*, Harlow: Pearson Education Limited, 2016.
- [5] B. Beizer, *Software Testing Techniques (Second Ed.)*, New York: Van Nostrand Reinhold Co., 1990.
- [6] DVLA, "Digital Marketplace," [Online]. Available: <https://www.digitalmarketplace.service.gov.uk/digital-outcomes-and-specialists/opportunities/7645>. [Accessed 30 12 2018].
- [7] DVLA, "DVLA IT Strategy 2017 to 2020," 27 06 2017. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/621747/dvla-it-strategy-2017-20.pdf. [Accessed 01 01 2019].
- [8] D. I. Anonymous, Interviewee, *Testing Practices within the DVLA*. [Interview]. 19 12 2018.
- [9] Agile Alliance, "Agile Alliance," Agile Alliance, 2019. [Online]. Available: <https://www.agilealliance.org/glossary/three-amigos>. [Accessed 02 01 2019].
- [10] Behat.org, "Behat Gherkin," Konstantin Kudryashov, 2016. [Online]. Available: <http://docs.behat.org/en/v2.5/guides/1.gherkin.html>. [Accessed 31 12 2018].
- [11] BCS Specialist Interest Group, "BS7925-1," Testing Standards, [Online]. Available: http://www.testingstandards.co.uk/bs_7925-1_online.htm. [Accessed 02 01 2019].
- [12] ISO, "ISO 29119-1:2013," ISO, [Online]. Available: <https://www.iso.org/standard/45142.html>. [Accessed 02 01 2019].
- [13] GDS, "Digital Service Standard," GDS, [Online]. Available: <https://www.gov.uk/service-manual/service-standard>. [Accessed 02 01 2019].
- [14] T. B. T. M. T. a. C. S. Glenford Myers, *The Art of Software Testing*, Hoboken: John Wiley & Sons, 2004.
- [15] DVLA, "GDS blog at GOV.UK," GDS, 03 05 2013. [Online]. Available: <https://gds.blog.gov.uk/2013/05/03/user-testing-at-dvla/>. [Accessed 23 12 2018].
- [16] D. Sale, *Testing Python : Applying Unit Testing, TDD, BDD and Acceptance Testing*, New York: John Wiley & Sons, Incorporated, 2014.
- [17] S. Acharya, *Mastering Unit Testing Using Mockito and JUnit*, Birmingham: Packt Publishing, 2014.
- [18] K. Beck, *JUnit Pocket Guide*, Sebastopol: O'Reilly Media, 2009.
- [19] P. Fitsilis, "Measuring the Complexity of Software Projects," *WRI World Congress on Computer Science and Information Engineering*, pp. 644 - 648, 2009.
- [20] G. e. Booch, *Object-Oriented Analysis and Design with Applications*, Boston: Pearson Education, Inc, 2007.
- [21] U. o. A. -. F. o. Science, "GITHUB," [Online]. Available: https://github.com/akueisara/Object-Oriented-Design/blob/master/Object-Oriented%20Design_Course%20Notes.pdf. [Accessed 30 12 2018].
- [22] K. a. S. J. Barclay, *Object-Oriented Design with UML and Java*, Oxford: Elsevier Butterworth-Heinemann, 2004.

- [23] G. P. a. D. W. Brett D. MacLaughlin, Head First Object-Oriented Analysis and Design, Sebastopol: O'Reilly Media, Inc, 2007.
- [24] D. Parnas, "On the Criteria to Be Used in Decomposing Systems Into Modules," 1972. [Online]. Available: https://www.win.tue.nl/~wstomv/edu/2ip30/references/criteria_for_modularization.pdf. [Accessed 05 01 2019].
- [25] S. R. Schach, Software Engineering with Java, New York: Irwin/McGraw-Hill, 1997.
- [26] D. Flanagan, Java in a Nutshell, Sebastopol: O'Reilly Media, Inc, 2005.
- [27] G. J. S. w. J. S. Harold Abelson, Structure and Interpretation of Computer Programs, London and Montreal: The MIT Press & McGraw-Hill Book Company, 1996.

Appendix

Part 1 Code

```
/*
Part 1
Write the code to allow the user to enter two sets of numbers A and B,
but preventing the user from entering duplicate values.
Allow the user to choose functions to be performed on these sets including:

1. Display the numbers in both sets
2. Determining the maximum and the minimum values of both sets
3. Calculate the average of the numbers in each set
4. Determine if sets A and B overlap
5. If possible, determine the intersection of the A and B (manually coded)
6. If possible, determine the set difference A – B (manually coded)
7. Quit

These functions should be called from a menu that allows the user to select the
appropriate operation.

*/

/**
 * @author Anthony Njonge 1809703
 * ACCA7000
 */

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

public class Main {

    private static Scanner scan = new Scanner(System.in);
    private static int[] arrayA ;
    private static int[] arrayB ;
    private static int userSizeArrayA;
    private static int userSizeArrayB;

    public static void main(String[] args)
    {
        //Welcome message and let users enter sets
        //after entering the arrays users can select the menu choices progressively
        try {
            setSizeArrays(); // user to enter the size of sets
            setArrays(); // users enters the integers into the sets
            boolean exit = false;
            while (!exit) // loop through the menu asking for user choice
            {
                entry(); // menu list
            }
        }
    }
}
```

```

int choice = integerOK();
boolean menuChoiceEntered = false;
switch (choice) {

    case 1:
        if (!menuChoiceEntered) {
            userViewSet(arrayA, arrayB);
        }
        break;
    case 2:
        if (!menuChoiceEntered) {
            //max and min values in the sets
            printMinMax();
        }
        break;
    case 3:
        if (!menuChoiceEntered) {
            printAverage();
        }
        break;
    case 4:
        if (!menuChoiceEntered) {
            overlapSets(arrayA, arrayB);
        }
        break;

    case 5:
        if (!menuChoiceEntered) {
            intersection(arrayA, arrayB, userSizeArrayA, userSizeArrayB);
        }
        break;

    case 6:
        if (!menuChoiceEntered) {
            setDiff(arrayA, arrayB, userSizeArrayA, userSizeArrayB);
        }
        break;

    case 7:
        System.exit(0); //exit choice

    default:
        System.out.println("Please enter a number between 1 and 7!");
        System.out.println("");
        break;
}
}
} catch (Exception e){//catch all exceptions
    System.out.println("Apologies. An error has occurred.");
}
}

```

```

}

/**
 * This method is to read the values into the Set arrays
 * @param length
 * @return
 */
public static int[] readArray(int length)
{
    int[] array = new int[length];
    int checkValue;
    for(int i=0; i<length; i++)
    {
        System.out.println("Enter an integer:");
        checkValue = integerOK();

        //check if integer exists in the set
        for (int j = 0; j < length; j++) {
            if (checkValue==(array[j]) && i != j) {
                System.out.println("Integer exists.");
                i--;
            }
        }

        array[i] = checkValue;
    }
    return array;
}

/**
 * pass in an array and return the maximum value of the array
 * @param array
 * @return
 */
public static int maxArray(int[] array)
{
    int temp,max;
    max= array[0]; //assume the first value is the maximum value
    for(int i=0; i<array.length;i++) {
        temp = array[i];
        if (temp >= max) {
            max = temp;
        }
    }
    return max;
}

/**

```

```

* pass in an array and return the minimum value of the array
* @param array
* @return
*/
public static int minArray(int[] array)
{
    int temp;
    int min;
    min = array[0]; // assume the first value as the minimum value
    for(int i=0; i<array.length;i++){
        temp = array[i];
        if(temp < min){
            min=temp;
        }
    }
    return min;
}

/**
* pass in an array and return the average
* @param array
* @return
*/
public static double averageArray(int[] array)
{

    double sum, average;
    sum =0;
    for(int i=0; i<array.length;i++) {
        sum += array[i];
    }
    average=sum/array.length;

    return average;
}

/**
* This method is to print the Set Arrays
* @param array
*/
public static void printArray(int[] array)
{
    System.out.println(Arrays.toString(array));
}

/**
* Start of the set question. Initialise the size of sets and value in the arrays
*/
public static void setSizeArrays()
{

```

```

System.out.println("Do you want to find out about your sets?");
System.out.println("Next, please choose your sets.");
System.out.println("Please enter the size of Set A.");
userSizeArrayA = integerOK();
System.out.println("Please enter the size of Set B.");
userSizeArrayB = integerOK();

}

public static void setArrays()
{

    System.out.println("Enter integers to go into Set A.");
    arrayA = readArray(userSizeArrayA);

    System.out.println("Enter integers to go into Set B.");
    arrayB = readArray(userSizeArrayB);
}

/**
 * pass in the two sets of array and print the sets for the user to view
 * @param a
 * @param b
 */
public static void userViewSet(int[]a, int[]b)
{
    System.out.println("Set A is: ");
    printArray(a);

    System.out.println("Set B is: ");
    printArray(b);
}

/**
 * check that the entered integer is valid
 * @return
 */
public static int integerOK()
{
    boolean checkInteger = false;
    int val = 0;
    while (!checkInteger) {
        try {
            val = userEntry();
            if(val <= 0)
            {
                System.out.print("Please enter a positive integer only!");
                checkInteger = false;
            }
        }
        else
        {

```

```

        checkInteger = true;
    }
} catch (NumberFormatException e) {
    System.out.println("Please ensure you enter a whole number!");
    checkInteger = false;
} catch (InputMismatchException e){
    System.out.println("Please ensure you enter an integer!");
    break;
}
}
return val;
}

/**
 * user input point
 * @return
 */
public static int userEntry()
{
    try {
        return Integer.parseInt(scan.next());
    } catch (InputMismatchException e){
        System.out.println("Please only enter integers.");
        return Integer.parseInt(scan.next());
    }
}

/**
 * The menu of the set questions
 */
public static void entry()
{
    System.out.println("Please choose your selection.\n" +
        "1. Print out the sets.\n" +
        "2. Determine the maximum and minimum of the set values.\n" +
        "3. Calculate the average of the sets.\n" +
        "4. Determine if set A and B overlap.\n" +
        "5. Check set A, B intersection.\n" +
        "6. Determine set difference A - B\n" +
        "7. Quit.");

    return;
}

/**
 * method to print the minimum and maximum values of the set values
 */
public static void printMinMax()
{
    System.out.printf("The max of Set A is: %d",maxArray(arrayA));
}

```

```

        System.out.println();
        System.out.printf("The max of Set B is: %d",maxArray(arrayB));
        System.out.println();
        System.out.printf("The min of Set A is: %d",minArray(arrayA));
        System.out.println();
        System.out.printf("The min of Set B is: %d",minArray(arrayB));
        System.out.println();
    }

    /**
     * method to print the average of the set values for the different sets
     */
    public static void printAverage(){
        System.out.printf("The average of Set A is: %.2f",averageArray(arrayA));
        System.out.println();
        System.out.printf("The average of Set B is: %.2f",averageArray(arrayB));
        System.out.println();
    }

    /**
     * pass in the arrays and determine if they overlap
     * @param a
     * @param b
     */
    public static void overlapSets(int[]a, int[]b){
        if(minArray(b) <= maxArray(a) || maxArray(a) >= minArray(b))
        {
            System.out.println("The sets overlap");
        } else if(minArray(b) > maxArray(a) || maxArray(a) < minArray(b)){
            System.out.println("The sets do not overlap");
        }
    }

    /**
     * pass in the two arrays and get the intersection
     * @param a arrayA
     * @param b arrayB
     * @param c userSizeArrayA
     * @param d userSizeArrayB
     */
    public static void intersection(int[]a, int[]b, int c, int d){

        System.out.println("The values of intersection are:");

        for(int j=0;j<c;j++) {
            for (int i = 0; i < d; i++) {
                if (b[i] == a[j]) {
                    System.out.println(b[i]);
                }
            }
        }
    }

```

```

    }

    System.out.println("Next choice."); // give a prompt that intersection values
    have been printed and user can move on.
}

public static void setDiff(int[]a, int[]b, int c, int d){ // mathematical set difference
    int countSame =0;
    for (int i= 0; i<c; i++) {
        for(int j=0;j<d;j++){
            if (a[i] != b[j]) {
                countSame++;
            }
        }
    }

    if(countSame==d){
        System.out.print(a[i]+" ");
        countSame=0;
    }else{
        countSame=0;
    }
}

System.out.println("");
if(countSame==0){
    System.out.println("There is no set difference."); //inform user that there is
    no set difference.
}
System.out.println("Next choice."); // give a prompt that intersection values
have been printed and user can move on.
System.out.println("");
}
}

```

Set Code JUnit Tests

```

import org.junit.Test;
import static org.junit.Assert.*;
public class MainTest {

    @Test
    public void main() {
    }

    @Test
    public void readArray() {
        int length = 2;
        int[] array = new int[length];
    }
}

```

```

    array[0] = 1;
    array[1] = 1;
    int[] testarray;
    boolean expected;
    for(int i=0;i<length-1;i++){
        if(array[i]==array[i+1]){
            expected = true;
        }
    }
    testarray = Main.readArray(2);
    assertArrayEquals(testarray,array);
}

@Test
public void maxArray() {
    int length = 3;
    int[] array = new int[length];
    array[0] = 1;
    array[1] = 3;
    array[2] = 7;
    int check;
    check = Main.maxArray(array);
    assertEquals(7,check);
}

@Test
public void minArray() {
    int length = 3;
    int[] array = new int[length];
    array[0] = 1;
    array[1] = 3;
    array[2] = 7;
    int check;
    check = Main.minArray(array);
    assertEquals(1,check);
}

@Test
public void averageArray() {
    int length = 3;
    int[] array = new int[length];
    array[0] = 1;
    array[1] = 3;
    array[2] = 7;
    double check;
    check = Main.averageArray(array);
    assertEquals(3.667,check,0.001);
}
}

```

Part 2 Code

```
/*  
Build a Connect 4 style Console game which allows the players to select the column  
they wish to drop their counter  
(O or X) down. The program should be able to detect a winning horizontal, vertical or  
diagonal line of 4 matching  
counters, then stop the game and declare the winner.
```

```
However in our version of the game, we will allow each player  
two optional special moves that they can each play once during the game. These  
are:
```

1. BLITZ activated by pressing the B key and selecting a column to clear.
2. TIME BOMB activated by pressing the T key and selecting a column. This places
a special symbol '*' in the
selected column as normal. However it clears the counters surrounding it
horizontally, vertically and diagonally
after the next player has made 2 more moves.

```
*/
```

```
import java.util.InputMismatchException;  
import java.util.Scanner;  
import static java.util.Arrays.*;
```

```
/**
```

```
 * @author Anthony Njonge
```

```
 * This program is designed to appear as shown in the assignment question: a 6X6  
grid
```

```
*/
```

```
public class Main {
```

```
    private static int boardWidth=6;  
    private static int boardHeight=6;  
    public static char[][] board = new char[boardHeight][boardWidth];  
    public static char player = 'X';  
    public static char timeBomb1 = '*';  
    public static char timeBomb2 = 'b';  
    private static Scanner stdin = new Scanner(System.in);
```

```
    //private static BufferedInputStream stdin = new BufferedInputStream(System.in);
```

```
    public static void main(String[] args) {
```

```
        //splash start
```

```
        entry();
```

```
        board = startBoard(board, '-');
```

```
        int blitzPlayerOne = 0;
```

```

int blitzPlayerTwo = 0;
int timerBombPlayerOne=0;
int timerBombPlayerTwo=0;
int timerLoopX =0;
int timerLoopY =0;

while (true){
    try {

        // Show the state of the board
        showGameBoard(board);
        // Ask player for move place
        System.out.println("Player " + player + ",please choose your column:");
        String s = stdin.next().toLowerCase();
        if ((s.contentEquals("b"))) { // blitz moves
            if (player == 'X' && blitzPlayerOne == 0) {
                System.out.println("Please choose the column to blitz.");
                int blitzCol = stdin.nextInt();
                blitz(board, blitzCol - 1);
                blitzPlayerOne = 1;
                player = choosePlayer(player);
            } else if (player == 'O' && blitzPlayerTwo == 0) {
                System.out.println("Please choose the column to blitz.");
                int blitzCol = stdin.nextInt();
                blitz(board, blitzCol - 1);
                blitzPlayerTwo = 1;
                player = choosePlayer(player);
            } else {
                System.out.println("You've exhausted the blitz moves");
            }
        } else if (s.contentEquals("t")) { //timer bombs moves
            if (player == 'X' && timerBombPlayerOne == 0) {
                System.out.println("Please choose the column to add the timer.");
                int timerCol = stdin.nextInt();
                timeBomb(board, timerCol - 1, timeBomb1);
                //timerBombPlayerOne =1; // counter to show player X has played the
timerbomb
                timerLoopX = 1; //timer loop for Player 1 moves
                player = choosePlayer(player);
            } else if (player == 'O' && timerBombPlayerTwo == 0) {
                System.out.println("Please choose the column to add the timer.");
                int timerCol = stdin.nextInt();
                timeBomb(board, timerCol - 1, timeBomb2);
                //timerBombPlayerTwo =1; // counter to show player O has played the
timerbomb
                timerLoopY = 1; //timer loop for Player 2 moves
                player = choosePlayer(player);
            } else {
                System.out.println("You've exhausted the time bomb moves");
            }
        }
    }
}

```

```

    } else { // all other normal moves
        int col = Integer.parseInt(s);
        if (checkColThenAddPlay(board, col - 1, player)) { // Try move
            if (draw(board)) { //check for draw!
                System.out.println("It is a draw!");
                showGameBoard(board);
                return;
            }
            if (checkWin(board)) { // Check for winner
                System.out.println("Player " + player + " wins!");
                showGameBoard(board);
                return; // End game
            }
            player = choosePlayer(player); // Switch players
        }

        if (timerLoopX == 1) // this to be actioned when timebomb has been
initiated.
        {
            if (timerBombPlayerOne >= 0 && timerBombPlayerOne < 3)
{ //increment timeBombPlayerOne
                //player == 'X' && timerBombPlayerOne >= 0 &&
timerBombPlayerOne < 3
                if (timerBombPlayerOne == 2) { //find and explode the bomb after
looping twice for player X
                    findAndExplodeBomb(board, timeBomb1);
                    fallSuspendedPlays(board);
                    timerLoopX = 0;
                }
                timerBombPlayerOne++; //add counter of time bomb counts
            }
        }
        if (timerLoopY == 1) // this to be actioned when timebomb has been
initiated.
        {
            if (timerBombPlayerTwo >= 0 && timerBombPlayerTwo < 3)
{ //increment timeBombPlayerTwo
                //player == 'O' && timerBombPlayerTwo >= 0 &&
timerBombPlayerTwo < 3
                if (timerBombPlayerTwo == 2) { //find and explode the bomb after
looping twice for player O
                    findAndExplodeBomb(board, timeBomb2);
                    fallSuspendedPlays(board);
                    timerLoopY = 0;
                }
                timerBombPlayerTwo++; // add counter of time bomb counts
            }
        }
    }
} catch (IndexOutOfBoundsException e) { //catch an out of bounds error

```

```

        System.out.println("Please choose a column within 1 - 6.");
    } catch (InputMismatchException e) { //catch an input mismatch error
        System.out.println("Please enter only numbers 1-6 and character t or
b.");
    } catch (NumberFormatException e) { //catch a number format error
        System.out.println("Please enter only numbers 1-6 or characters t or b.
");
    }
}
}

/**
 * this the start intro to the game
 */
public static void entry(){
    System.out.println("Welcome to a Connect 4 game for two players!");
    System.out.println();
    System.out.println("Player 1 will be player X.");
    System.out.println("Player 2 will be player O.");
    System.out.println();
    System.out.println("This game includes two extra moves on the classic
game:");
    System.out.println("1. A BLITZ activated by pressing the B key and
selecting a column to clear.");
    System.out.println("2. TIME BOMB activated by pressing the T key and
selecting a column. A '*' or '#' will appear where the bomb is set.");
    System.out.println();
}

/**
 * initialise the board with column title and '-' character.
 */
public static void initialBoard(){
    //print board
    System.out.println(" 1  2  3  4  5  6");
    for(int i=0; i<boardWidth; i++){
        for(int j=0; j<boardHeight; j++){
            System.out.print(board[i][j] + " - ");
        }
        System.out.println();
    }
}

/**
 * set the board with set character myChar
 * @param board
 * @param myChar '-'
 * @return
 */
public static char[][] startBoard(char[][] board, char myChar){

```

```

        for (int i = 0; i < board.length; i++)
        {
            fill(board[i], 0, board[i].length, myChar); //use fill method in Arrays to fill the
2D array
        }
        return board;
    }

    /**
     * Show the current status of the board.
     * @param board
     */
    public static void showGameBoard(char[][] board){
        System.out.println(" 1  2  3  4  5  6");
        for (int i = 0; i < board.length; i++)
        {
            System.out.print("|");
            for (int j = 0; j < board[i].length; j++)
            {
                System.out.print(" " + board[i][j] + " |");
            }
            System.out.println();
        }
    }

    /**
     * determine whether the column choosen is playable
     * @param board
     * @param col
     * @param player
     * @return
     */
    public static boolean checkColThenAddPlay(char[][] board, int col, char player){
        boolean answer = false;
        //check col
        if( board[0][col] != '-'){
            System.out.println("Column chosen is full");
            return false;
        }
        //accept play
        for(int i=board.length-1; i>=0;i--){
            if(board[i][col]=='-'){
                board[i][col]= player;
                return answer = true;
            }
        }
        return answer;
    }

    /**

```

```

* check if a winner exists
* @param board
* @return
*/
public static boolean checkWin(char[][] board){

    boolean answer = false;

    //check horizontal
    for (int i = 0; i < board.length; i++)
    {
        for (int j = 0; j < board[i].length - 3; j++)
        {
            if (board[i][j] != '.'
                && board[i][j] == board[i][j+1]
                && board[i][j] == board[i][j+2]
                && board[i][j] == board[i][j+3])
            {
                return true;
            }
        }
    }

    //check vertical
    for (int i = 0; i < board[0].length; i++)
    {
        for (int j = 0; j < board.length - 3; j++)
        {
            if (board[j][i] != '.'
                && board[j][i] == board[j+1][i]
                && board[j][i] == board[j+2][i]
                && board[j][i] == board[j+3][i])
            {
                return true;
            }
        }
    }

    // check diagonally top left to right bottom
    for (int i = 0; i < board.length - 3; i++)
    {
        for (int j = 0; j < board[i].length - 3; j++)
        {
            if (board[i][j] != '.'
                && board[i][j] == board[i+1][j+1]
                && board[i][j] == board[i+2][j+2]
                && board[i][j] == board[i+3][j+3])
            {
                return true;
            }
        }
    }
}

```

```

    }
}

// check diagonally right bottom to left top
for (int i = 0; i < board.length - 3; i++)
{
    for (int j = 3; j < board[i].length; j++)
    {
        if (board[i][j] != '-'
            && board[i][j] == board[i+1][j-1]
            && board[i][j] == board[i+2][j-2]
            && board[i][j] == board[i+3][j-3])
        {
            return true;
        }
    }
}

return answer;
}

/**
 * check the status of the board for a draw
 * @param board
 * @return
 */
public static boolean draw(char[][] board){///check for draw
    int spare = 0;
    for(int i=0;i<board.length;i++)
    {
        for(int j=0;j<board.length;j++)
        {
            if(board[i][j] != '-')
            {
                spare++;
            }
        }
    }

    if(spare==36){
        return true;
    }
    return false;
}

/**
 * choose the next player
 * @param nowPlayer
 * @return

```

```

*/
public static char choosePlayer(char nowPlayer){
    if (nowPlayer == 'X')
    {
        return 'O';
    } else
    {
        return 'X';
    }
}

/**
 * blitz move
 * @param board
 * @param col
 */
public static void blitz(char[][] board, int col){
    for (int i = 0; i < board.length; i++)
    {
        for(int j=0;j<board.length;j++)
        {
            board[i][col] = '-'; //use fill method in Arrays to fill the 2D array
        }
    }
}

/**
 * finds and explodes the timer bomb
 * @param board
 */
public static void findAndExplodeBomb (char[][] board,char timebomb){//find
and explode the bomb

    for(int i=0;i<board.length;i++){
        for(int j=0;j<board.length;j++){
            if(board[i][j]==timebomb)
            {
                if(i>=1 && i<=4){//anywhere inside the inner board
                    if(j>=1 && j<=4)
                    {
                        board[i][j]='-';
                        board[i-1][j-1] = '-';
                        board[i][j-1]='-';
                        board[i+1][j-1]='-';
                        board[i-1][j]='-';
                        board[i+1][j]='-';
                        board[i-1][j+1]='-';
                        board[i][j+1]='-';
                        board[i+1][j+1]='-';
                    }else if(j==0){//left line

```

```

        //j==0&&i>0&&i<5
        board[i][j]='-';
        board[i+1][j]='-';
        board[i][j+1]='-';
        board[i+1][j+1]='-';
        board[i-1][j]='-';
        board[i-1][j+1]='-';
    }else if(j==5){//right line &&i<5&&i>0
        board[i][j]='-';
        board[i-1][j]='-';
        board[i+1][j]='-';
        board[i-1][j-1]='-';
        board[i][j-1]='-';
        board[i+1][j-1]='-';
    }
} else //if (i==0||i==5||j==0||j==5)
{
    if(i==0&&j==0){//top left
        board[i][j]='-';
        board[i+1][j]='-';
        board[i+1][j+1]='-';
        board[i][j+1]='-';
    } else if(i==5&&j==0){//top right
        board[i][j]='-';
        board[i-1][j]='-';
        board[i-1][j+1]='-';
        board[i][j+1]='-';
    } else if(i==5&&j==5){//bottom right
        board[i][j]='-';
        board[i-1][j]='-';
        board[i-1][j-1]='-';
        board[i][j-1]='-';
    } else if(i==0&&j==5){//bottom left
        board[i][j]='-';
        board[i+1][j]='-';
        board[i+1][j-1]='-';
        board[i][j-1]='-';
    } else if(i==0){//top line &&j<5&&j>0
        board[i][j]='-';
        board[i][j-1]='-';
        board[i][j+1]='-';
        board[i+1][j+1]='-';
        board[i+1][j]='-';
        board[i+1][j-1]='-';
    } else if(i==5){//bottom line &&j>0&&j<5
        board[i][j]='-';
        board[i-1][j]='-';
        board[i-1][j+1]='-';
        board[i][j+1]='-';
        board[i-1][j-1]='-';
    }
}

```

```

        board[i][j-1]='-';
    }
}
}
}
}

/**
 * check where the timebomb star to play the time bomb move play
 * @param board
 * @param col
 * @param star
 * @return
 */
public static boolean timeBomb (char[][] board, int col, char star){
    boolean answer = false;
    //char star = '*'; // add the star bomb
    //check col
    if( board[0][col] != '-') {
        System.out.println("Column chosen is full");
        return false;
    }
    //accept play
    for(int i=board.length-1; i>=0; i--){
        {
            if(board[i][col]=='-')
            {
                board[i][col]= star;
                return answer = true;
            }
        }
    }
    return answer;
}

/**
 * check for suspended plays and drop them accordingly
 * @param board
 */
public static void fallSuspendedPlays(char[][] board){
    for(int k=0; k<5; k++) { // loop through the whole board
        for (int i = 0; i < board.length - 1; i++) { //check floating played characters for
            anything but bottom line
                for (int j = 0; j < board.length - 1; j++) {
                    if (board[i][j] != '-' && board[i + 1][j] == '-') {
                        board[i + 1][j] = board[i][j];
                        board[i][j] = '-';
                    }
                }
            }
        }
    }
}

```

```

    }
}
}

```

Connect 4 Code JUnit Tests

```

import org.junit.Test;

import static org.junit.Assert.*;

public class MainTest {

    @org.junit.Test
    public void startBoard() {
        char[][] board = new char[6][6];
        char initial = '-';

        for(int i=0; i<6;i++){
            for (int j=0; j<6;j++){
                board[i][j]=initial;
            }
        }
        assertEquals(board[0][0], '-');
    }

    @org.junit.Test
    public void checkColThenAddPlay() {
        char[][] board = new char[6][6];
        char initial = '-';
        for(int i=0; i<6;i++){
            for (int j=0; j<6;j++){
                board[i][j]=initial;
            }
        }
        boolean play;
        if(board[0][0]=='-'){
            play=true;
        }else{
            play = false;
        }
        //user wants to play col 0
        assertTrue(play);
    }

    @Test
    public void checkWin() {
        char[][] board = new char[6][6];
        char initial = '-';

        for(int i=0; i<6;i++){
            for (int j=0; j<6;j++){

```

```

        board[i][j]=initial;
    }
}
board[5][0]='X';
board[5][1]='X';
board[5][2]='X';
board[5][3]='X';

boolean win;
//horizontal
if((board[5][0]=='X' && board[5][1]=='X' &&board[5][2]=='X' &&board[5][3]=='X' )
    ||(board[5][0]=='O' &&board[5][0]=='O' &&board[5][0]=='O'
&&board[5][0]=='O')){
    win=true;
}else{
    win = false;
}

assertTrue(win);
}

@org.junit.Test
public void draw() {
    char[][] board = new char[6][6];
    char player1 ='X';
    char player2='O';
    int count1 =0;
    int count2 =0;

    for(int i=0; i<3;i++){
        for (int j=0; j<6;j++){
            board[i][j]=player1;
        }
    }
    for(int i=3; i<6;i++){
        for (int j=0; j<6;j++){
            board[i][j]=player2;
        }
    }

    boolean draw;

    for(int i=0;i<6;i++){
        for(int j=0; j<6;j++){
            if(board[i][j]=='X'){
                count1++;
            } else{
                count2++;
            }
        }
    }
}

```

```

    }

    if(count1==count2){
        draw=true;
    }else{
        draw = false;
    }
    assertTrue(draw);
}

@org.junit.Test
public void timeBomb() {
    char[][] board = new char[6][6];
    char timeBomb = '*';
    char initial = '-';
    char player1 = 'X';
    char player2 = 'O';
    int countinitial=0;

    //initialise board as draw
    for(int i=0; i<6;i++){
        for (int j=0; j<3;j++){
            board[i][j]=player1;
        }
    }
    for(int i=0; i<6;i++){
        for (int j=3; j<6;j++){
            board[i][j]=player2;
        }
    }

    //initialise a timebomb
    board[2][3] = timeBomb;

    //reset values around timebomb
    for(int i=0;i<6;i++) {
        for (int j = 0; j < 6; j++) {
            if (board[i][j] == timeBomb) {
                board[i][j] = '-';
                board[i - 1][j - 1] = '-';
                board[i][j - 1] = '-';
                board[i + 1][j - 1] = '-';
                board[i - 1][j] = '-';
                board[i + 1][j] = '-';
                board[i - 1][j + 1] = '-';
                board[i][j + 1] = '-';
                board[i + 1][j + 1] = '-';
            }
        }
    }
}

```

```
}

for(int i=0;i<6;i++){
    for(int j=0; j<6;j++){
        if(board[i][j]==initial){
            countinitial++;
        }
    }
}

boolean timeBombCount;

if(countinitial==9){
    timeBombCount=true;
}else{
    timeBombCount = false;
}

//this should pass for Positive Case
assertTrue(timeBombCount);
}
```

Final

GRADEMARK REPORT

FINAL GRADE

83/100

GENERAL COMMENTS

Instructor

PART ONE

=====

Engagement with Literature Skills (25) **23**

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

Cognitive and Intellectual Skills (10) **8**

Transferable Skills (5) **4**

Practical Application Skills (50) **40**

Part 1 Software Testing

Very detailed report, although I would like to have seen more analysis of the testing tools used given the importance of the tool chain. Good.

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 COMPLETE

Use of JUnit LIMITED TESTS

Use of Git VERY LIMITED

Use of Java Doc LIMITED

Code executes correctly. Well written and structured code. In your comments you have said what the arguments and return types are but comments should be used to describe the functionality being performed.

PART TWO

=====

Engagement with Literature Skills (25) **15**

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

Cognitive and Intellectual Skills (10) **8**

Transferable Skills (5) **3**

Practical Application Skills (50) **40**

Part 2 Design Principles

A modest number of sources of reasonable quality.

A good report in terms of its technical detail. As a minor point, the presentation could use some work.

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

Use of JUnit YES

Use of Git VERY LIMITED

Use of Java Doc YES

Code executes correctly, nice display provided. The main() function needed to be subdivided, how many problems are being solved in it? More comments required. Time bomb handling could be more elegant.

Overall Comments

Git usage is a bit limited, there are only 5 commits across both projects.

Marks Part One: **83**

Marks Part Two: **74**

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1

definition



Comment 2

Not sure this makes any sense??



Comment 3

Sophisticated tests??

PAGE 5



Comment 4

Define acronym before using it.



Comment 5

Interesting, I would like to find out more about this.

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

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55
