**UNIVERSITY OF HERTFORDSHIRE**

**SCHOOL OF COMPUTER SCIENCE**

**BSC COMPUTER SCIENCE: 6COM1038**

**Module: Software Engineering Practise**

# Assignment 2 - Programming & Design

**Helitha Rupasinghe**

**Level 6**

**Academic year 2020-21**

# Table Of Contents

# Chapter 1.0 Refactoring & Design Process

The goal of this report is to act as a guide to identify solutions to different types of weaknesses that exist within the **Tinee Client Prototype**. The groundwork set in the critique of the Tinee Client has already identified issues by using static code analysers such as FindBugs and EasyPMD. The purpose of this report is to rework the client by refactoring code to make it more readable and maintainable without changing the function of the code with the support of redesign focused on optimizing classes and the way they communicate with each other.

## 1.2 Disclaimers

The information I present in this document is confidential and is provided by University of Hertfordshire. Code Reviews conducted are part of the analysis that target the weaknesses of the application under review which is reflected in this report. Additionally, new weaknesses can be discovered as new tests are run and so this report should be considered a guide and not a 100% representation of the weaknesses threatening your systems.

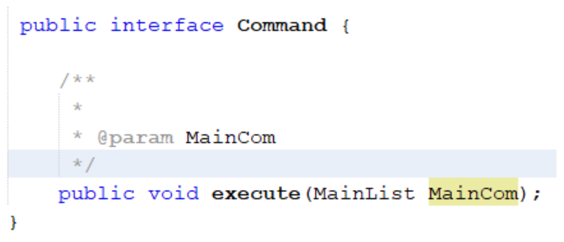
**Address questions** regarding legitimate use of this document to the postcode below:

University of Hertfordshire
Hatfield
Hertfordshire
AL10 9AB
UK

## 1.3 Command Pattern

The purpose of the command pattern is to handle dynamic requests (data drive Client's interactive user commands). In essence, the Command decouples the invoker from the receiver and the request carried out. A request is packaged under a command and passed to an invoker as an object. The invoker object stores the commands that gets passed to the corresponding object where it executes the command [1][2][3][4]. The implementation is outlined in the steps below.

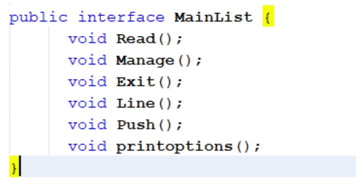
First step was to create the command interface.



```
public interface Command {

    /**
     *
     * @param MainCom
     */
    public void execute(MainList MainCom);
}
```

(Figure 1: Project view of Command Interface(Generated on Netbeans) . Date Accessed: 20/2/2021 )

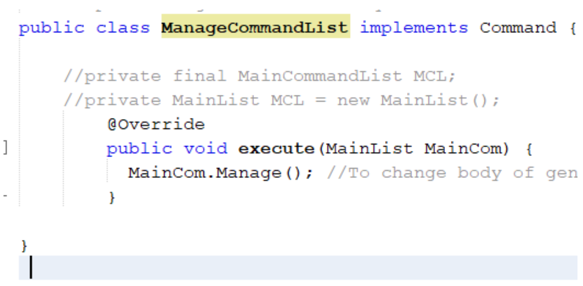
Second step, was to create a receiver called 'MainList' that implements commands.



```
public interface MainList {
        void Read();
        void Manage();
        void Exit();
        void Line();
        void Push();
        void printoptions();
}
```

(Figure 2: Project view of Receiver (Generated on Netbeans) . Date Accessed: 20/2/2021 )

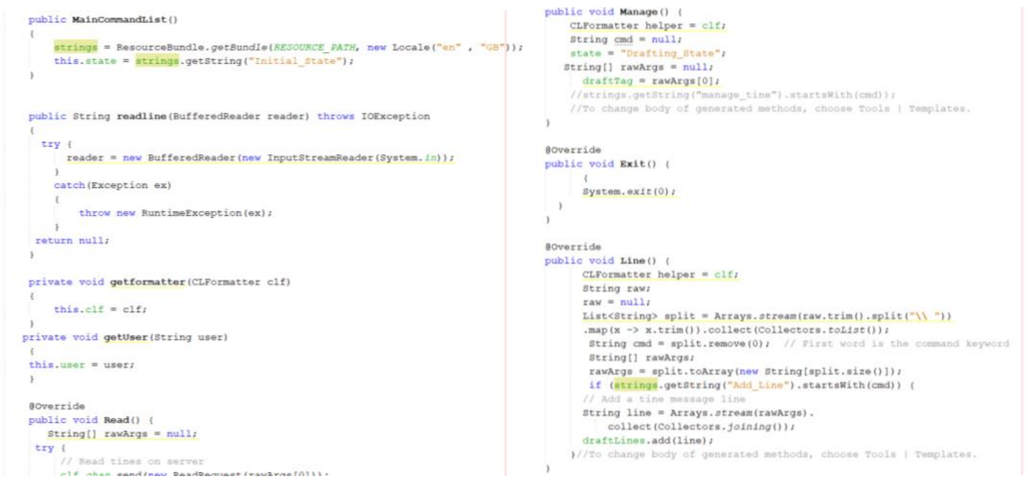
Third step, was to implement the command request for manage highlighted by figure 3. Further results can be accessed within the **Appendix Figures : 36/ 37/38/39**.



```
public class ManageCommandList implements Command {

    //private final MainCommandList MCL;
    //private MainList MCL = new MainList();
        @Override
        public void execute(MainList MainCom) {
          MainCom.Manage(); //To change body of gen
        }

}
```

(Figure 3: Project view of Concrete Commands (Generated on Netbeans). Date Accessed: 20/2/2021 )

Fourth step, involved implement the command definitions highlighted by figure 4. Further results can be accessed in **Appendix: Figures 44.**

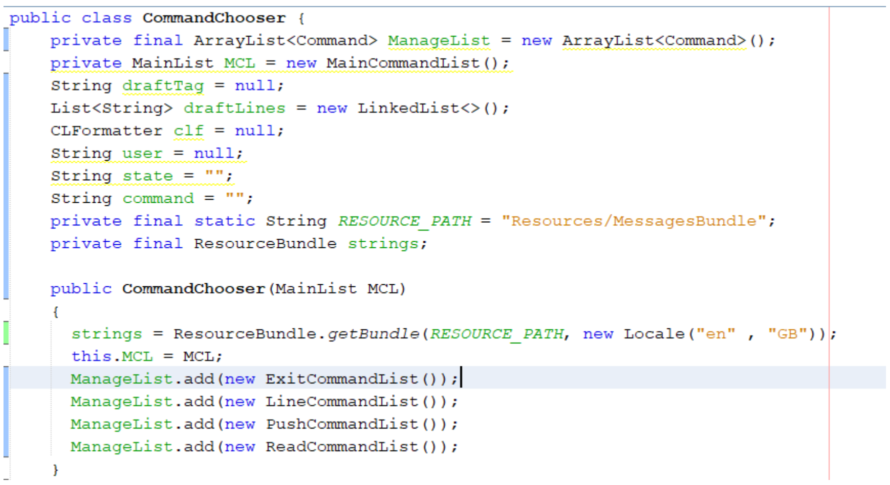


```
public MainCommandList()
{
    strings = ResourceBundle.getBundle(RESOURCE_PATH, new Locale("en" , "GB"));
    this.state = strings.getString("Initial_State");
}


public String readline(BufferedReader reader) throws IOException
{
  try {
      reader = new BufferedReader(new InputStreamReader(System.in));
    }
    catch(Exception ex)
    {
        throw new RuntimeException(ex);
    }
 return null;
}

private void getformatter(CLFormatter clf)
{
    this.clf = clf;
}
private void getUser(String user)
{
this.user = user;
}

@Override
public void Read() {
   String[] rawArgs = null;
 try {
     // Read tines on server
     clf.chan.send(new ReadRequest(rawArgs[0]));
```

```
public void Manage() {
    CLFormatter helper = clf;
    String cmd = null;
    state = "Drafting_State";
   String[] rawArgs = null;
      draftTag = rawArgs[0];
    //strings.getString("manage_tine").startsWith(cmd));
    //To change body of generated methods, choose Tools | Templates.
}

@Override
public void Exit() {
      {
      System.exit(0);
  }
}

@Override
public void Line() {
      CLFormatter helper = clf;
      String raw;
      raw = null;
      List<String> split = Arrays.stream(raw.trim().split("\\ "))
      .map(x -> x.trim()).collect(Collectors.toList());
       String cmd = split.remove(0);  // First word is the command keyword
       String[] rawArgs;
       rawArgs = split.toArray(new String[split.size()]);
       if (strings.getString("Add_Line").startsWith(cmd)) {
      // Add a tine message line
      String line = Arrays.stream(rawArgs).
          collect(Collectors.joining());
      draftLines.add(line);
    }//To change body of generated methods, choose Tools | Templates.
}
```

(Figure 4: Project view of Command Definitions (Generated on Netbeans) . Date Accessed: 20/2/2021 )

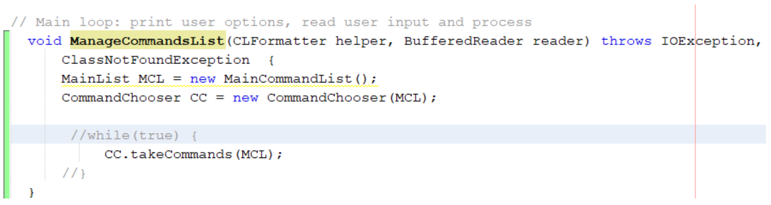
Fifth step, involved creating the invoker class highlighted by figure 5. Further results can be accessed in **Appendix Figures 40 /41/42/43**.



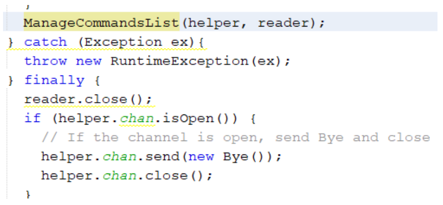
```
public class CommandChooser {
    private final ArrayList<Command> ManageList = new ArrayList<Command>();
    private MainList MCL = new MainCommandList();
    String draftTag = null;
    List<String> draftLines = new LinkedList<>();
    CLFormatter clf = null;
    String user = null;
    String state = "";
    String command = "";
    private final static String RESOURCE_PATH = "Resources/MessagesBundle";
    private final ResourceBundle strings;

    public CommandChooser(MainList MCL)
    {
      strings = ResourceBundle.getBundle(RESOURCE_PATH, new Locale("en" , "GB"));
      this.MCL = MCL;
      ManageList.add(new ExitCommandList());
      ManageList.add(new LineCommandList());
      ManageList.add(new PushCommandList());
      ManageList.add(new ReadCommandList());
    }
```

(Figure 5: Project view of Invoker Class (Generated on Netbeans) . Date Accessed: 20/2/2021 )

The sixth step involved calling the invoker class 'CommandChooser' and definer 'MainCommandList' highlighted by figure 6. Additional, refactoring of the run method was required to take ManageCommandsList instead of Loop tested by figure 45.



```
// Main loop: print user options, read user input and process
  void ManageCommandsList(CLFormatter helper, BufferedReader reader) throws IOException,
      ClassNotFoundException  {
      MainList MCL = new MainCommandList();
      CommandChooser CC = new CommandChooser(MCL);

       //while(true) {
          CC.takeCommands(MCL);
      //}
  }
```



```
    ;
    ManageCommandsList(helper, reader);
} catch (Exception ex){
    throw new RuntimeException(ex);
} finally {
    reader.close();
    if (helper.chan.isOpen()) {
      // If the channel is open, send Bye and close
      helper.chan.send(new Bye());
      helper.chan.close();
    }
```

(Figure 6: Project view of Client (Generated on NetBeans). Date Accessed: 20/2/2021 )

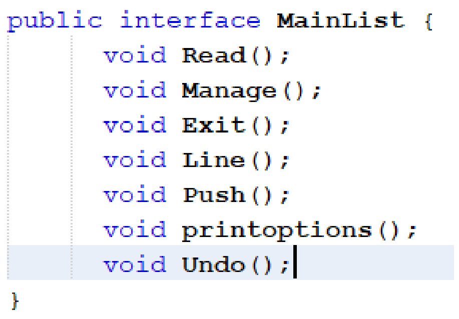
## 1.4 MVC

……………………………………………………

## 1.5 Extensions

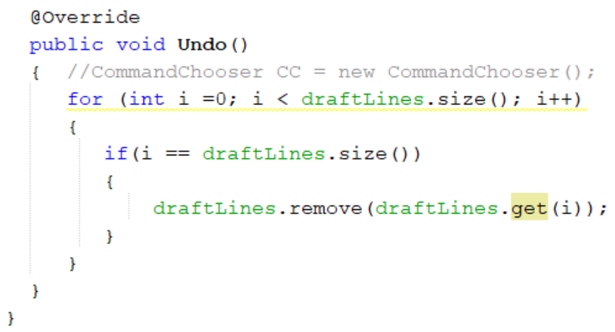
**Undo**

Figure 7 showcases refactoring to take undo which was implemented with its functionality.



```
public interface MainList {
        void Read();
        void Manage();
        void Exit();
        void Line();
        void Push();
        void printoptions();
        void Undo();
}
```

(Figure 7: Project view of Receiver(Generated on Netbeans) . Date Accessed: 20/2/2021 )



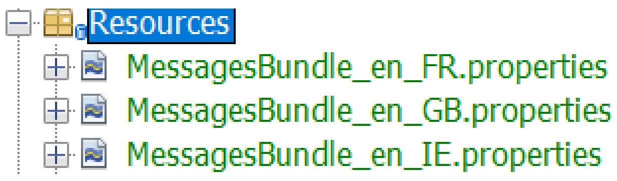
```
    @Override
    public void Undo()
    {   //CommandChooser CC = new CommandChooser();
        for (int i =0; i < draftLines.size(); i++)
        {
            if(i == draftLines.size())
            {
                draftLines.remove(draftLines.get(i));
            }
        }
    }
}
```

(Figure 8: Project view of Undo Functionality Implemented(Generated on Netbeans) . Date Accessed: 20/2/2021 )

## Internationalization

Internationalisation is an important feature for any application where Java already has built-in support for internationalisation that is suitable for the  Tinee-Client Application highlighted by figure 9 & 10 and 11.

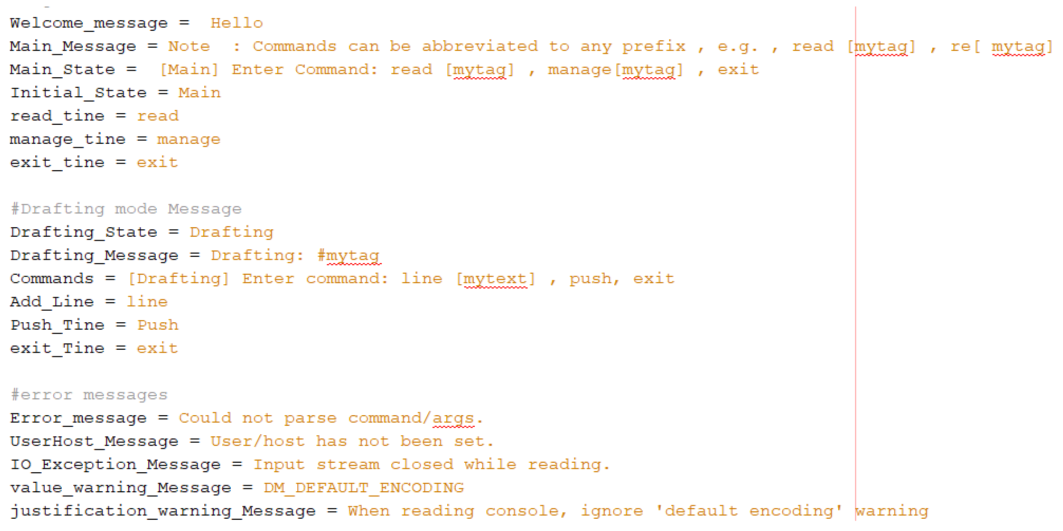
Figure 9 showcases the resource bundle created ‘Resources’ containing different locales en_Fr: en_GB and en_IE properties files.



```
Resources
  MessagesBundle_en_FR.properties
  MessagesBundle_en_GB.properties
  MessagesBundle_en_IE.properties
```

(Figure 9 : Project view of ResourceBundle (Generated on Netbeans) . Date Accessed: 20/2/2021 )

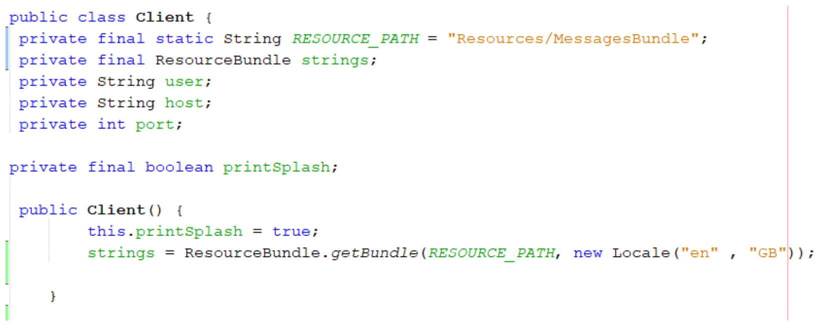
Its worth to note, that these might not be concrete translations but follow the format stated within the CLFormatter class.



```
Welcome_message =  Hello
Main_Message = Note  : Commands can be abbreviated to any prefix , e.g. , read [mytag] , re[ mytag]
Main_State =  [Main] Enter Command: read [mytag] , manage[mytag] , exit
Initial_State = Main
read_tine = read
manage_tine = manage
exit_tine = exit

#Drafting mode Message
Drafting_State = Drafting
Drafting_Message = Drafting: #mytag
Commands = [Drafting] Enter command: line [mytext] , push, exit
Add_Line = line
Push_Tine = Push
exit_Tine = exit

#error messages
Error_message = Could not parse command/args.
UserHost_Message = User/host has not been set.
IO_Exception_Message = Input stream closed while reading.
value_warning_Message = DM_DEFAULT_ENCODING
justification_warning_Message = When reading console, ignore 'default encoding' warning
```

 (Figure 10: Project view of EN_GB locale (Generated on Netbeans) . Date Accessed: 20/2/2021 )

Figure 11 calling the resource bundle & English locale.



```
public class Client {
 private final static String RESOURCE_PATH = "Resources/MessagesBundle";
 private final ResourceBundle strings;
 private String user;
 private String host;
 private int port;

private final boolean printSplash;

 public Client() {
        this.printSplash = true;
        strings = ResourceBundle.getBundle(RESOURCE_PATH, new Locale("en" , "GB"));

    }
```

(Figure 11: Project view of ResourcePath  (Generated on Netbeans) . Date Accessed: 20/2/2021 )

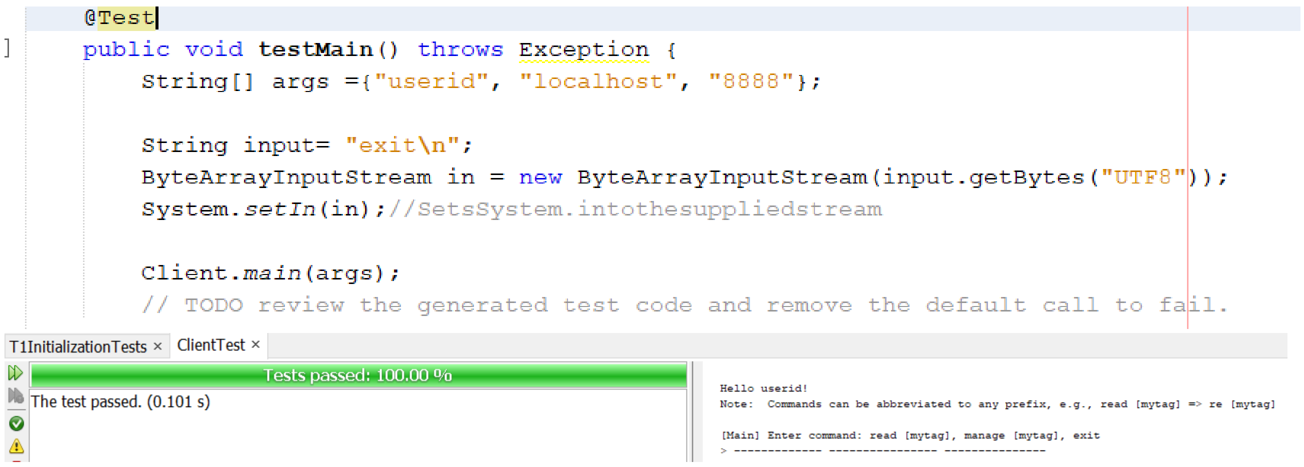
# Chapter 2.0 Version Control

Applying version control has been useful during this project using Git to track the version: compare files and merge prior changes all to one repository highlighted by Appendix _ 32. The two instances where I used a dedicated git branch called ‘Hotfix’ was for fixing issues identified using static analysis tool FindBugs on Client.java and on fixing translations that were incorrect for the client class showcased by Appendix _ 33.

## 2.1 Black Box Testing

The Tinee application Client reads user input using the standard input stream system.in which is used to manually feed in the desired input showcased in the figures below.

For this Application, the first test case was run against the main method of the Client class. The client uses the standard input stream as required by the specification which is checked within the testmain() method using the desired input. The results highlighted by figure 12 uses the input command ‘exit’ which correctly displays the front-end of the command line utility visible to the user[6].



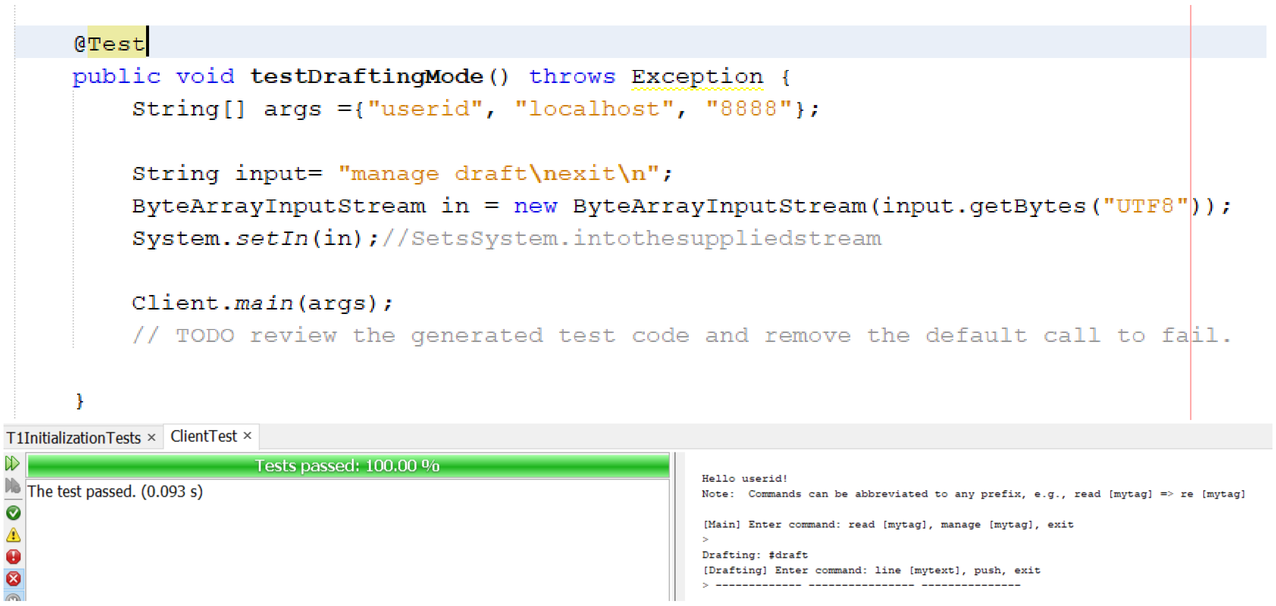
```
    @Test
    public void testMain() throws Exception {
        String[] args ={"userid", "localhost", "8888"};

        String input= "exit\n";
        ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
        System.setIn(in);//SetsSystem.intothesuppliedstream

        Client.main(args);
        // TODO review the generated test code and remove the default call to fail.
```

The test passed. (0.101 s)

```
Hello userid!
Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
> ------------- ---------------- ---------------
```

(Figure 12: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021 )

The second test case was conducted on the user entering drafting mode which is responsible for allowing the user to add text and push the drafted tines to the server. The results of the test were successful highlighted by figure 13 which correctly display the input ‘manage draft’ required by the specification to enter drafting mode [6].



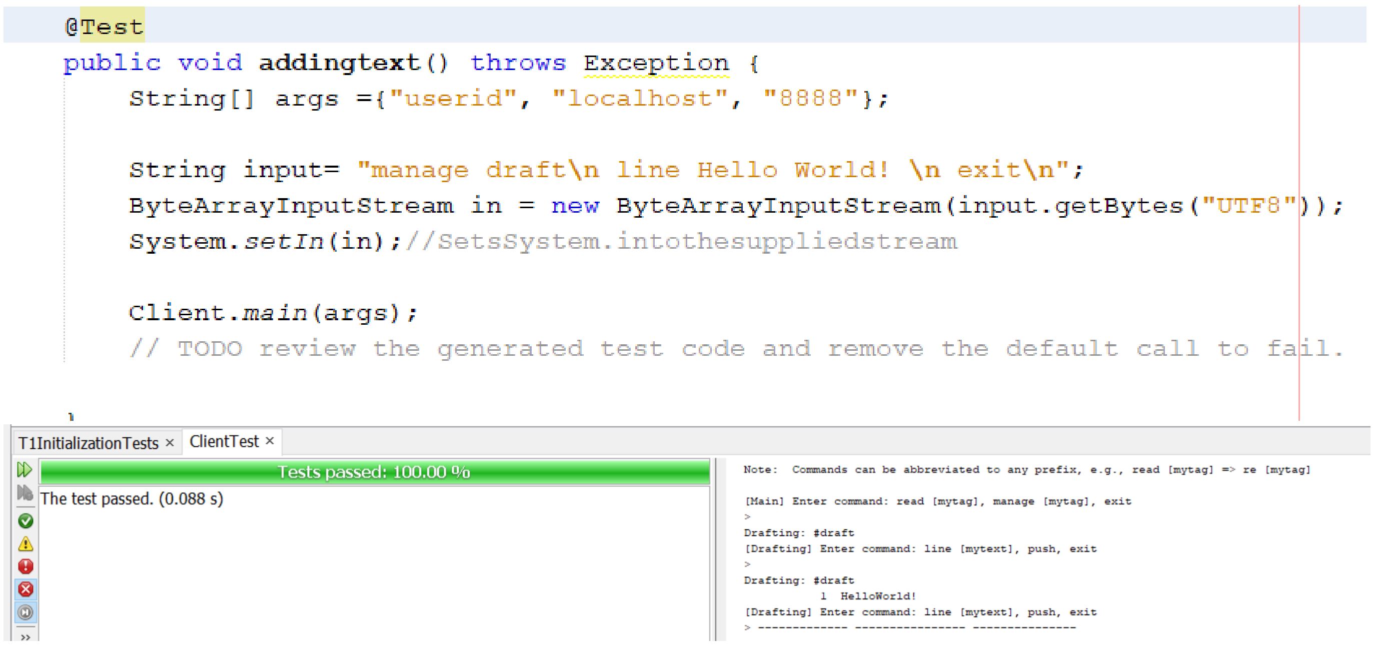
```
    @Test
    public void testDraftingMode() throws Exception {
        String[] args ={"userid", "localhost", "8888"};

        String input= "manage draft\nexit\n";
        ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
        System.setIn(in);//SetsSystem.intothesuppliedstream

        Client.main(args);
        // TODO review the generated test code and remove the default call to fail.

    }
```

```
The test passed. (0.093 s)
```

```
Hello userid!
Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
> ------------ --------------- ---------------
```

(Figure 13 : Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021 )

The third test case was conducted on the user adding text to the server as required by the specification. The results of the test were successful as highlighted by figure 14 which correctly displays the output text using the command ‘line Hello world’ in drafting mode [6].



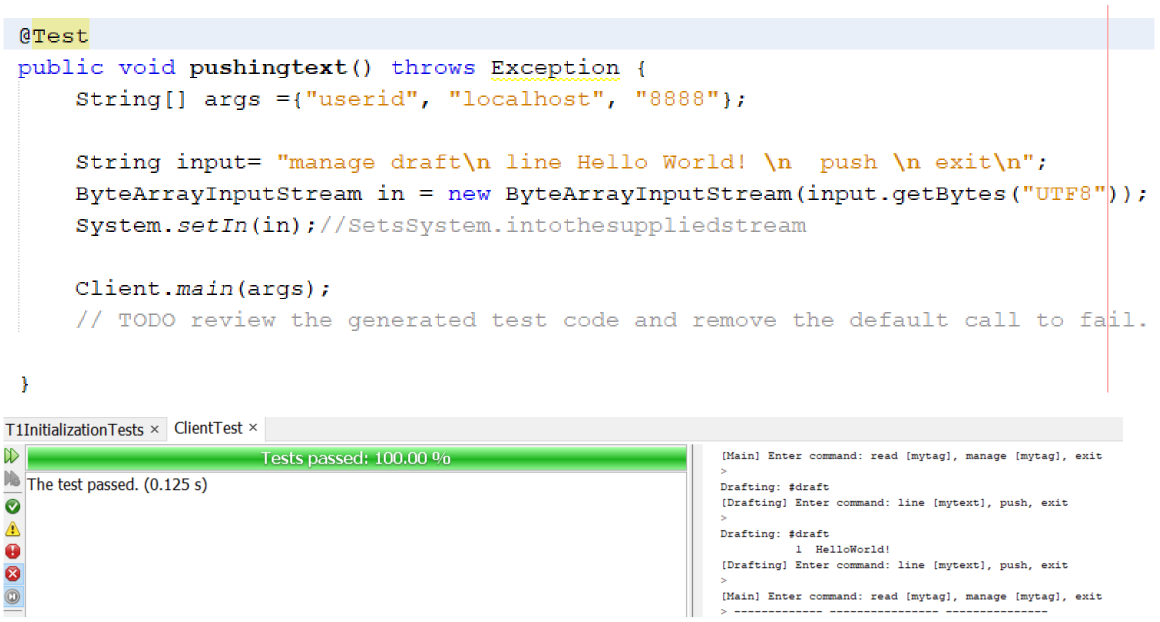
```
    @Test
    public void addingtext() throws Exception {
        String[] args ={"userid", "localhost", "8888"};

        String input= "manage draft\n line Hello World! \n exit\n";
        ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
        System.setIn(in);//SetsSystem.intothesuppliedstream

        Client.main(args);
        // TODO review the generated test code and remove the default call to fail.
```

```
The test passed. (0.088 s)
```

```
Note:  Commands can be abbreviated to any prefix, e.g., read [mytag] => re [mytag]

[Main] Enter command: read [mytag], manage [mytag], exit
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
>
Drafting: #draft
          1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
> ------------ --------------- ---------------
```

(Figure 14: Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021 )

The fourth test case was conducted on the user pushing text to the server as required by the specification. The results of the test were successful as highlighted by figure 15 which correctly simulates the output text ‘line Hello world’ in drafting mode that requires the ‘push’ command to send the drafted tine to the server [6].



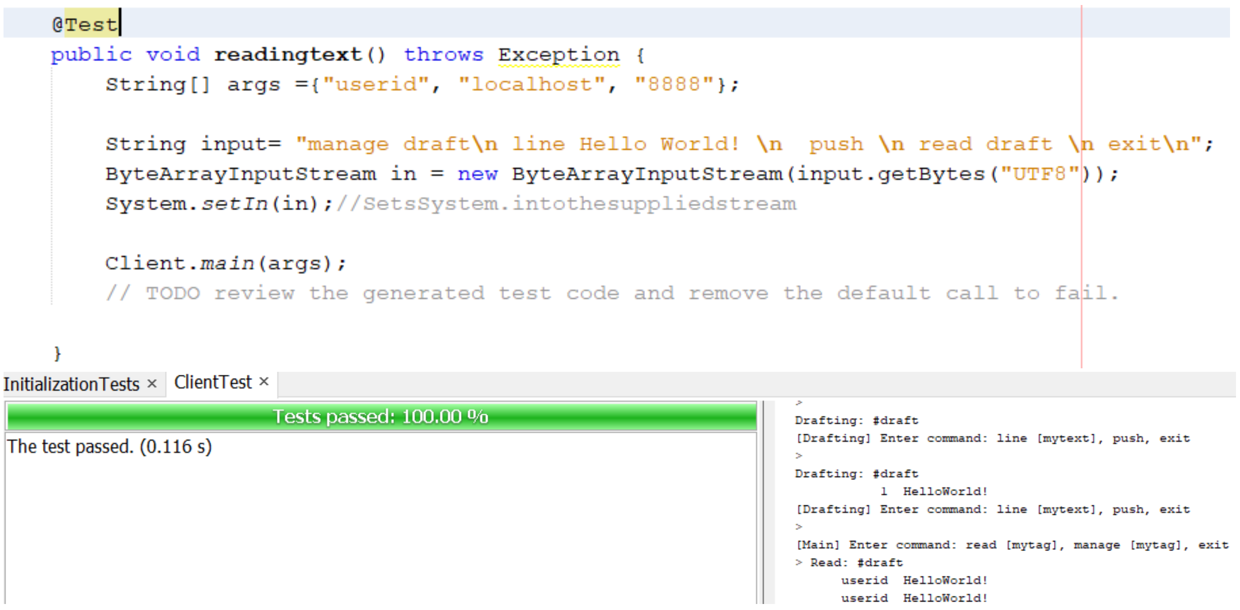
```java
@Test
public void pushingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n  push \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```

```
The test passed. (0.125 s)
```

```
[Main] Enter command: read [mytag], manage [mytag], exit
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
>
Drafting: #draft
         1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
>
[Main] Enter command: read [mytag], manage [mytag], exit
> ------------ --------------- ---------------
```

(Figure 15 : Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021 )

The fifth test case was conducted on the user reading text to the server as required by the specification. The results of the test were successful as highlighted by figure 16 which correctly simulates the output text 'line Hello world' in drafting mode that uses the command 'read draft' to display text in the current draft.



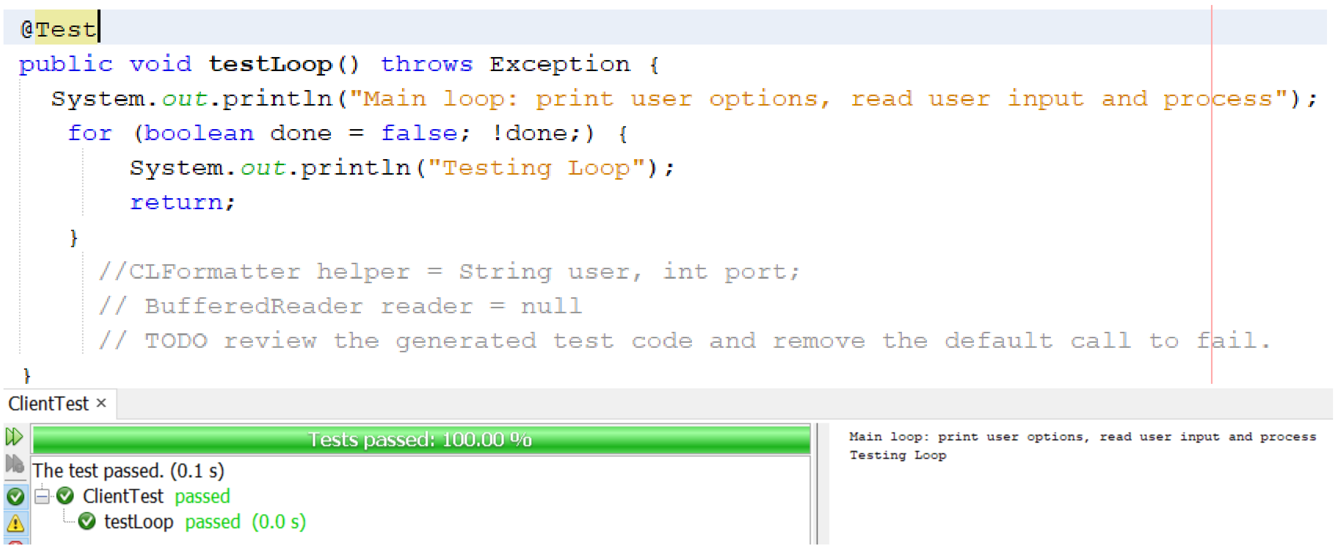
```java
@Test
public void readingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n  push \n read draft \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```

```
The test passed. (0.116 s)
```

```
>
Drafting: #draft
[Drafting] Enter command: line [mytext], push, exit
>
Drafting: #draft
         1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
>
[Main] Enter command: read [mytag], manage [mytag], exit
> Read: #draft
      userid  HelloWorld!
      userid  HelloWorld!
```

(Figure 16 : Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021 )

The sixth test case was conducted on the main loop that is used to handle user options , read user input and process the different types of inputs required by the specification. The importance of running this test is to understand that the for loop is responsible for reading and processing requests stated by the specification that demands constant user input and responses accordingly since it is a command line utility. The results of the test were successful as highlighted by figure 19 which correctly simulates the output text 'Testing loop' on an infinite loop that is exited using the return keyword to exit the program faster. The logic showcased in figure 17 is entry controlled by the use of the initialised boolean value 'done' that is set to false and then reversed to true which acts as a finite state machine that is not supposed to end until the user has reached the end state.



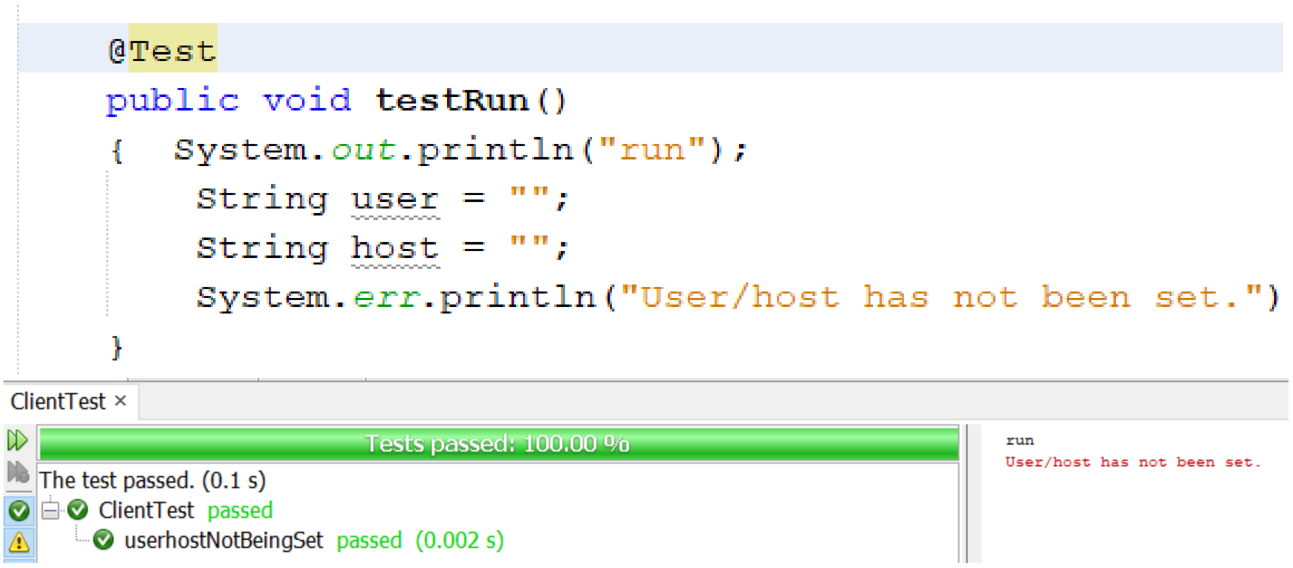
```
@Test
public void testLoop() throws Exception {
  System.out.println("Main loop: print user options, read user input and process");
   for (boolean done = false; !done;) {
       System.out.println("Testing Loop");
       return;
   }
     //CLFormatter helper = String user, int port;
     // BufferedReader reader = null
     // TODO review the generated test code and remove the default call to fail.
}
```

```
ClientTest ×
Tests passed: 100.00 %
The test passed. (0.1 s)
ClientTest passed
  testLoop passed (0.0 s)
```

```
Main loop: print user options, read user input and process
Testing Loop
```

(Figure 17 : Client.java class tests run (Generated on NetBeans) . Date Accessed: 22/2/2021 )

The seventh test case was conducted on the user leaving 'user' and 'host' empty (arguments) which is responsible for displaying error message to the server as required by the specification. The results of the test were successful as highlighted by figure 18 which correctly simulates the error message "user\host has not been set" that would be visible to the user within the output window.



```
@Test
public void testRun()
{   System.out.println("run");
     String user = "";
     String host = "";
     System.err.println("User/host has not been set.")
}
```

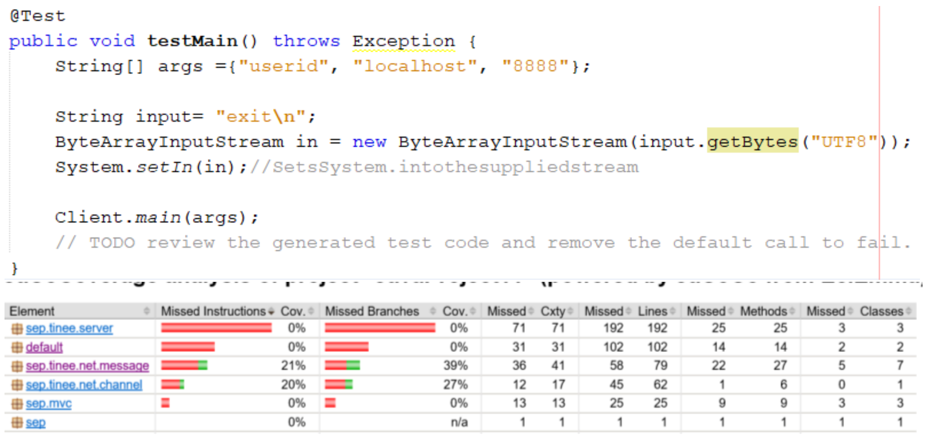
```
ClientTest ×
Tests passed: 100.00 %
The test passed. (0.1 s)
ClientTest passed
  userhostNotBeingSet passed (0.002 s)
```

```
run
User/host has not been set.
```

(Figure 18: Client.java class tests run (Generated on NetBeans). Date Accessed: 22/2/2021)

The JaCoCo plugin was installed to conduct code coverage testing on the source code for instruction and branch coverage. By conducting further testing, I found that the results that are shown here can fluctuate  when running the same tests with JaCoCo conducted for Client.java and therefore should only be seen as a guide and not a 100% representation of the branch coverage and instruction coverage as required by the specification.

JaCoCoverage was conducted on figure 19 where 41% was Instruction coverage against 66% branch coverage. Its worth to note that, figure 32  for Sep.Tinee.net.message covered 26% Instruction Coverage against 43% branch coverage whereas figure 33 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.

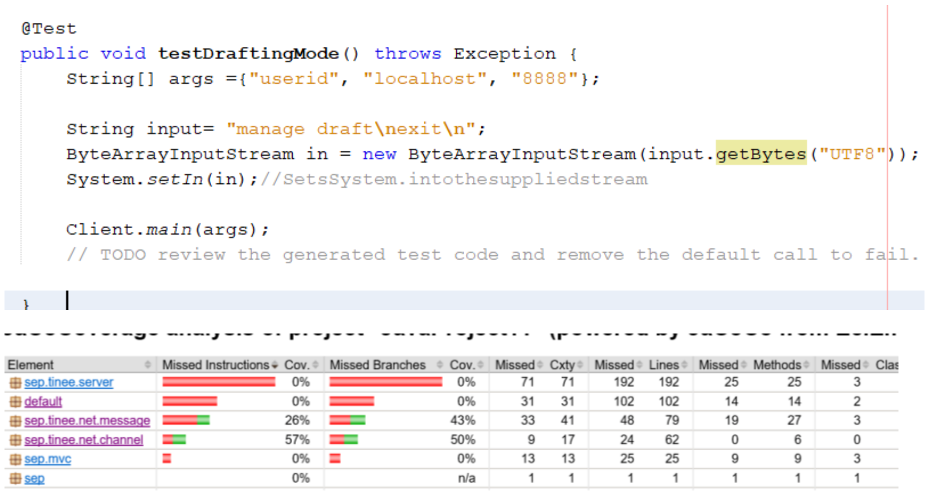


```
@Test
public void testMain() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.
}
```

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | | 0% | | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 | 3 |
| default | | 0% | | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 | 2 |
| sep.tinee.net.message | | 21% | | 39% | 36 | 41 | 58 | 79 | 22 | 27 | 5 | 7 |
| sep.tinee.net.channel | | 20% | | 27% | 12 | 17 | 45 | 62 | 1 | 6 | 0 | 1 |
| sep.mvc | | 0% | | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 | 3 |
| sep | | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(Figure 19 : JaCoCoverage Analysis of main method (Generated on NetBeans). Date Accessed: 22/04/2021)

JaCoCoverage was conducted on figure 20 where total of 10% was Instruction coverage against a total of 13% branch coverage. Its worth to note that, the results depict a change in code coverage from figure 19 to figure 20 for Sep.Tinee.net.message covered 26% Instruction Coverage against 43% branch coverage whereas Appendix A: figure _ 35 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.

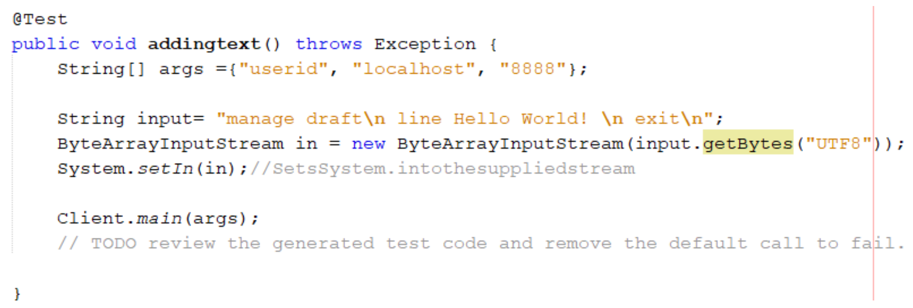


```
@Test
public void testDraftingMode() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\nexit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```

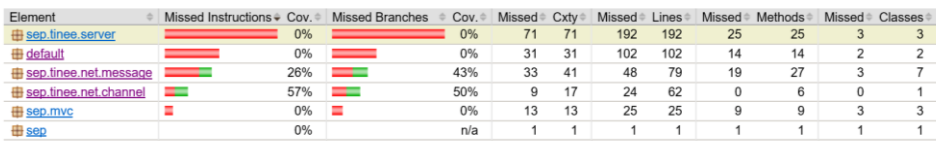
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Clas |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | | 0% | | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 | |
| default | | 0% | | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 | |
| sep.tinee.net.message | | 26% | | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 | |
| sep.tinee.net.channel | | 57% | | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | |
| sep.mvc | | 0% | | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 | |
| sep | | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 | |

(Figure 20: JaCoCoverage Analysis of Drafting method (Generated on NetBeans). Date Accessed: 22/04/2021)

JaCoCoverage was conducted on figure 21 where total of 10% was Instruction coverage against a total of 13% branch coverage. Its worth to note that, Appendix: figure _ 34 for Sep.Tinee.net.message covered 26% Instruction Coverage against 43% branch coverage whereas Appendix A:figure_35 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.



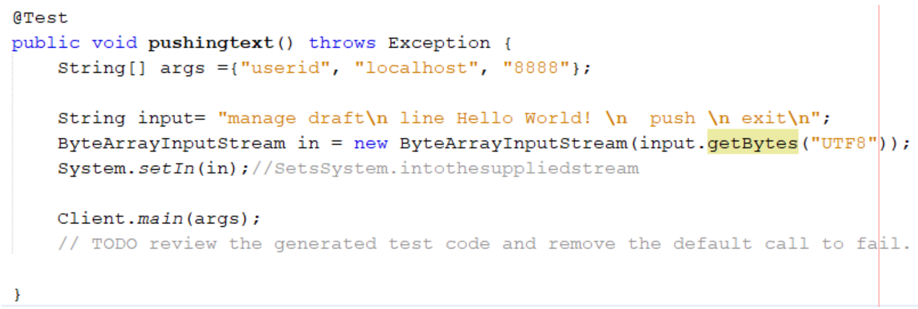
```
@Test
public void addingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```



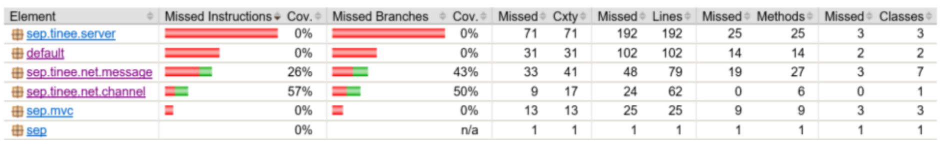
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | | 0% | | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 | 3 |
| default | | 0% | | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 | 2 |
| sep.tinee.net.message | | 26% | | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 | 7 |
| sep.tinee.net.channel | | 57% | | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | 1 |
| sep.mvc | | 0% | | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 | 3 |
| sep | | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(Figure 21: JaCoCoverage Analysis of Adding Text method (Generated on NetBeans). Date Accessed: 22/04/2021)

JaCoCoverage was conducted on figure 22 where total of 10% was Instruction coverage against a total of 13% branch coverage. Its worth to note that, Appendix A: figure _34 for Sep.Tinee.net.message covered 26% Instruction Coverage against 43% branch coverage whereas Appendix A: figure_35 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.



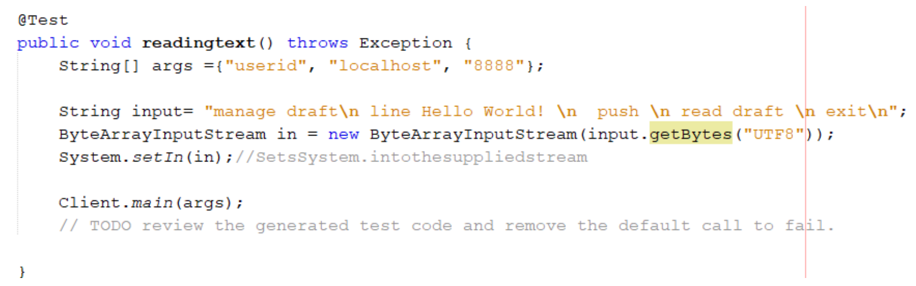
```
@Test
public void pushingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n  push \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```



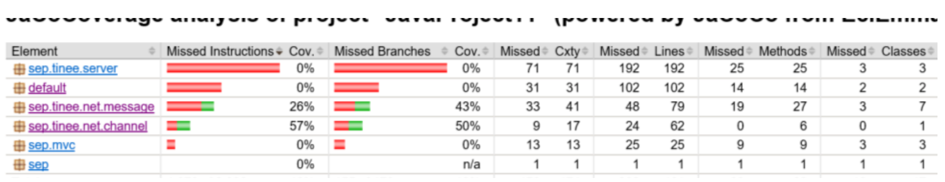
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | | 0% | | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 | 3 |
| default | | 0% | | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 | 2 |
| sep.tinee.net.message | | 26% | | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 | 7 |
| sep.tinee.net.channel | | 57% | | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | 1 |
| sep.mvc | | 0% | | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 | 3 |
| sep | | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(Figure 22: JaCoCoverage Analysis of Pushingtext method (Generated on NetBeans). Date Accessed: 22/04/2021)

JaCoCoverage was conducted on figure 23 where total of 10% was Instruction coverage against a total of 13% branch coverage. Its worth to note that, Appendix A: figure _34 for Sep.Tinee.net.message covered 26% Instruction Coverage against 43% branch coverage whereas Appendix A: figure_35 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.



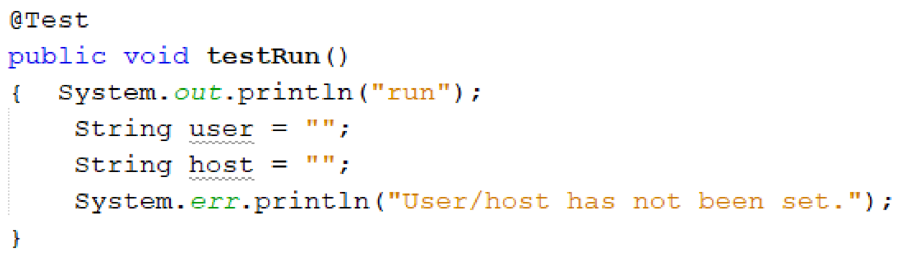
```
@Test
public void readingtext() throws Exception {
    String[] args ={"userid", "localhost", "8888"};

    String input= "manage draft\n line Hello World! \n  push \n read draft \n exit\n";
    ByteArrayInputStream in = new ByteArrayInputStream(input.getBytes("UTF8"));
    System.setIn(in);//SetsSystem.intothesuppliedstream

    Client.main(args);
    // TODO review the generated test code and remove the default call to fail.

}
```



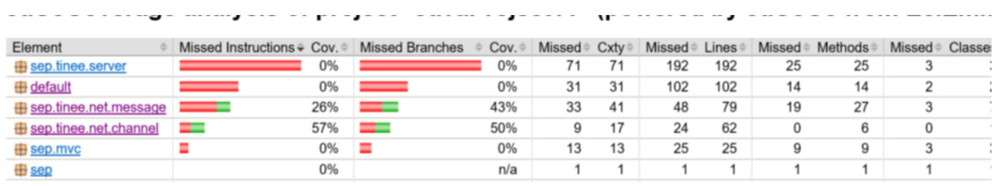
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | 🟥 | 0% | 🟥 | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 | 3 |
| default | 🟥 | 0% | 🟥 | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 | 2 |
| sep.tinee.net.message | 🟥🟩 | 26% | 🟥🟩 | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 | 7 |
| sep.tinee.net.channel | 🟥🟩 | 57% | 🟥🟩 | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | 1 |
| sep.mvc | 🟥 | 0% | 🟥 | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 | 3 |
| sep |  | 0% |  | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(Figure 23: JaCoCoverage Analysis of Readtext method (Generated on NetBeans). Date Accessed: 22/04/2021)

JaCoCoverage was conducted on figure 24 where total of 10%  was Instruction coverage against  a total of 13% branch coverage. Its worth to note that,  Appendix A: figure 34  for Sep.Tinee.net.message covered  26% Instruction Coverage against 43% branch coverage whereas Appendix A: figure 35 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.



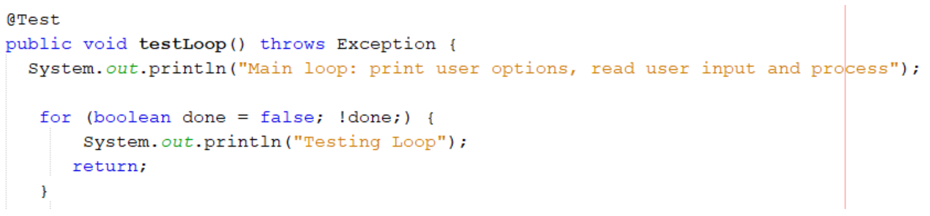
```
@Test
public void testRun()
{   System.out.println("run");
     String user = "";
     String host = "";
     System.err.println("User/host has not been set.");
}
```



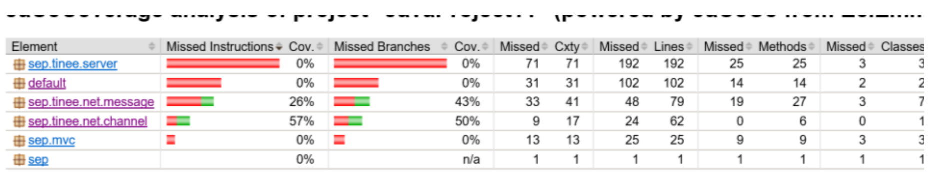
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed |
|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | 🟥 | 0% | 🟥 | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 |
| default | 🟥 | 0% | 🟥 | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 |
| sep.tinee.net.message | 🟥🟩 | 26% | 🟥🟩 | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 |
| sep.tinee.net.channel | 🟥🟩 | 57% | 🟥🟩 | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 |
| sep.mvc | 🟥 | 0% | 🟥 | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 |
| sep |  | 0% |  | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(Figure 24: JaCoCoverage Analysis of testRun method (Generated on NetBeans). Date Accessed: 22/04/2021)

JaCoCoverage was conducted on figure 25 where total of 10%  was Instruction coverage against  a total of 13% branch coverage. Its worth to note that, Appendix A: figure 34  for Sep.Tinee.net.message covered  26% Instruction Coverage against 43% branch coverage whereas Appendix: figure 35 for Sep.Tinee.net.channel covered 57% Instruction coverage against 50% branch coverage.



```
@Test
public void testLoop() throws Exception {
  System.out.println("Main loop: print user options, read user input and process");

   for (boolean done = false; !done;) {
      System.out.println("Testing Loop");
     return;
   }
```



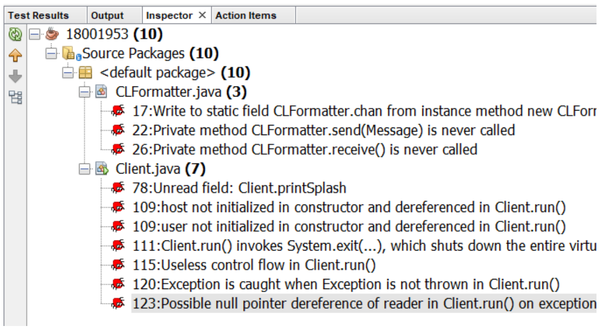
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| sep.tinee.server | | 0% | | 0% | 71 | 71 | 192 | 192 | 25 | 25 | 3 | 3 |
| default | | 0% | | 0% | 31 | 31 | 102 | 102 | 14 | 14 | 2 | 2 |
| sep.tinee.net.message | | 26% | | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 | 7 |
| sep.tinee.net.channel | | 57% | | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | 1 |
| sep.mvc | | 0% | | 0% | 13 | 13 | 25 | 25 | 9 | 9 | 3 | 3 |
| sep | | 0% | | n/a | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

(Figure 25: JaCoCoverage Analysis of testLoop method (Generated on NetBeans). Date Accessed: 22/04/2021)

## 2.2 Static Code Analysis

The results using Find Bugs involved manual testing of the Java source code for the prototype application. The tool identified 10 issues referenced to two classes 'Client.java' and 'CLFormatter.java' as showcased in figure 1 [5].

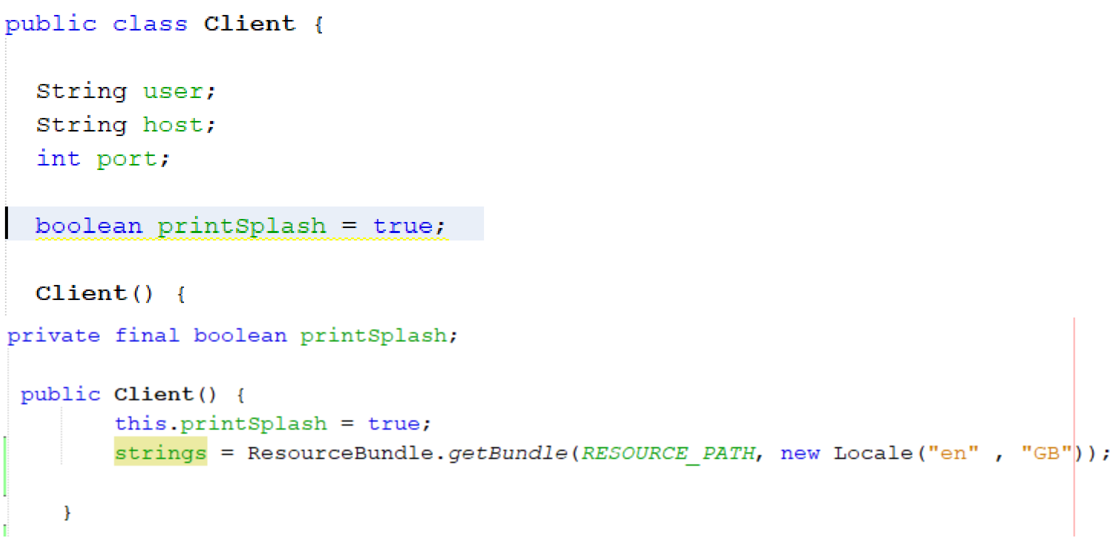
There were seven issues in total highlighted with 'Client.java' class focused on uninitialized fields (line "109"): unread fields (line "78") , useless control flow (line "115") in client.run() and further issues were found with Client.run() invoking system.exit() which shutdown the entire virtual machine(line "111"). On the whole, bugs were found at (line "120") within the run method that contained a try-catch block that catches exceptions objects, but the exception caught is not being thrown in Client.Run(). A variant to the bug found at (line "120") would be to explicitly catch RunTime Exceptions, rethrow it and then catch all non-runtime exceptions showcased in figure 2.  Its worth to note, that these checks found issues with the 'CLFormatter.java' that follow bad coding practise. Overall, the findings discovered will be remediated to secure the application and prevent vulnerabilities [5].



```
18001953 (10)
  Source Packages (10)
    <default package> (10)
      CLFormatter.java (3)
        17:Write to static field CLFormatter.chan from instance method new CLForr
        22:Private method CLFormatter.send(Message) is never called
        26:Private method CLFormatter.receive() is never called
      Client.java (7)
        78:Unread field: Client.printSplash
        109:host not initialized in constructor and dereferenced in Client.run()
        109:user not initialized in constructor and dereferenced in Client.run()
        111:Client.run() invokes System.exit(...), which shuts down the entire virtu
        115:Useless control flow in Client.run()
        120:Exception is caught when Exception is not thrown in Client.run()
        123:Possible null pointer dereference of reader in Client.run() on exception
```

(Figure 26: Project view identified issues with two classes (Generated on Netbeans) . Date Accessed: 20/2/2021 )

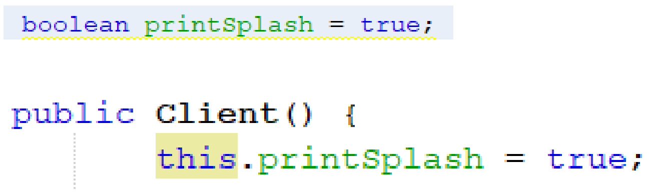
With the implementation of the command pattern, majority of the issues identified got fixed. The ones that were chosen were referenced to concrete code using findbugs with  regards to unread fields at line 78 which was solved by reading the field from within a constructor highlighted by figure 27 .



```
public class Client {

  String user;
  String host;
  int port;

  boolean printSplash = true;

  Client() {
private final boolean printSplash;

 public Client() {
        this.printSplash = true;
        strings = ResourceBundle.getBundle(RESOURCE_PATH, new Locale("en" , "GB"));

    }
```

(Figure 27: Project view Findbugs solutions Before || After (Generated on Netbeans) . Date Accessed: 20/2/2021 )

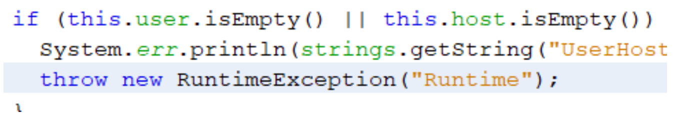
The second error was at line 109 with regards to a field not getting initialised within any constructor which was solved by using the constructor provided in the source code highlighted by figure 28.



```
 boolean printSplash = true;

public Client() {
       this.printSplash = true;
```

(Figure 28: Project view Findbugs solutions Before || After (Generated on Netbeans) . Date Accessed: 20/2/2021 )

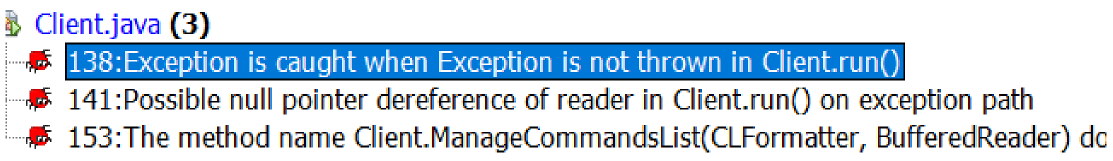
The Third error was at line 111 with regards to method invoking system.exit which shutsdown the entire virtual machine that was solved by removing ‘System.exit(1)’ showcased by figure 29.



```
 if (this.user.isEmpty() || this.host.isEmpty())
   System.err.println(strings.getString("UserHost
   throw new RuntimeException("Runtime");
```

(Figure 29: Project view Findbugs solutions Before || After (Generated on Netbeans) . Date Accessed: 20/2/2021 )

On the whole, rerunning findbugs identified 3 issues shown in figure 30 where I’ve chosen to not treat these results due to lack of testing time.

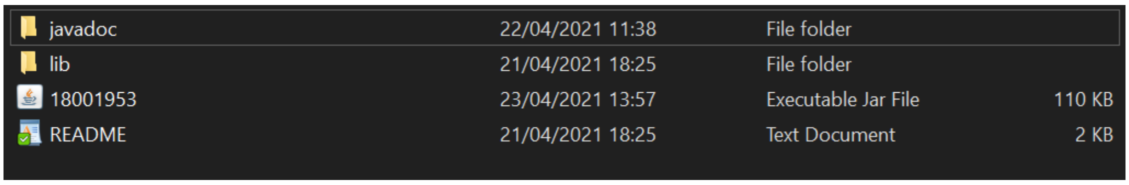


```
Client.java (3)
  138:Exception is caught when Exception is not thrown in Client.run()
  141:Possible null pointer dereference of reader in Client.run() on exception path
  153:The method name Client.ManageCommandsList(CLFormatter, BufferedReader) dc
```

(Figure 30: Project view Client Class Solutions(Generated on Netbeans) . Date Accessed: 20/2/2021 )

## 2.3 Documentation

The benefit of generating the javadoc and readme is to output the build description: specify packages and source files.



```
javadoc      22/04/2021 11:38   File folder
lib          21/04/2021 18:25   File folder
18001953     23/04/2021 13:57   Executable Jar File   110 KB
README       21/04/2021 18:25   Text Document           2 KB
```

(Figure 31: Project view Client Class Solutions(Generated on Netbeans) . Date Accessed: 20/2/2021 )

# Chapter 3.0 Conclusions

This section of the report will assess my own performance of the SEP conducted in this report that covers a broad range of topics that required project management skills: knowledge of suitable tools where the implementation of the specification reflects the objectives which have been met!

The groundwork laid within the Code review assessed the security weaknesses which does not guarantee the possibility of new weaknesses being found. Its worth to note, that the code review did have carryover to this report as shown in **Chapter 1.0 – 3.0**.

The solutions presented in this report required a high-level overview of the command pattern: MVC, blackbox-testing, Static code analysis and data classification which was implemented to some degree by adjusting the source code. To finalise the report, on '3/05/2021 by Helitha' the solutions presented are viable to change and be improved upon to support future functionality.

# Chapter 4.0 References

[1] Tut. 2020. Tutorialspoint.com. https://www.tutorialspoint.com/design_pattern/command_pattern.htm . Date Accessed: 30/4/2021

[2] Tut. 2020. Tutorialspoint.com. https://www.tutorialspoint.com/design_pattern/index.htm . Date Accessed: 30/4/2021

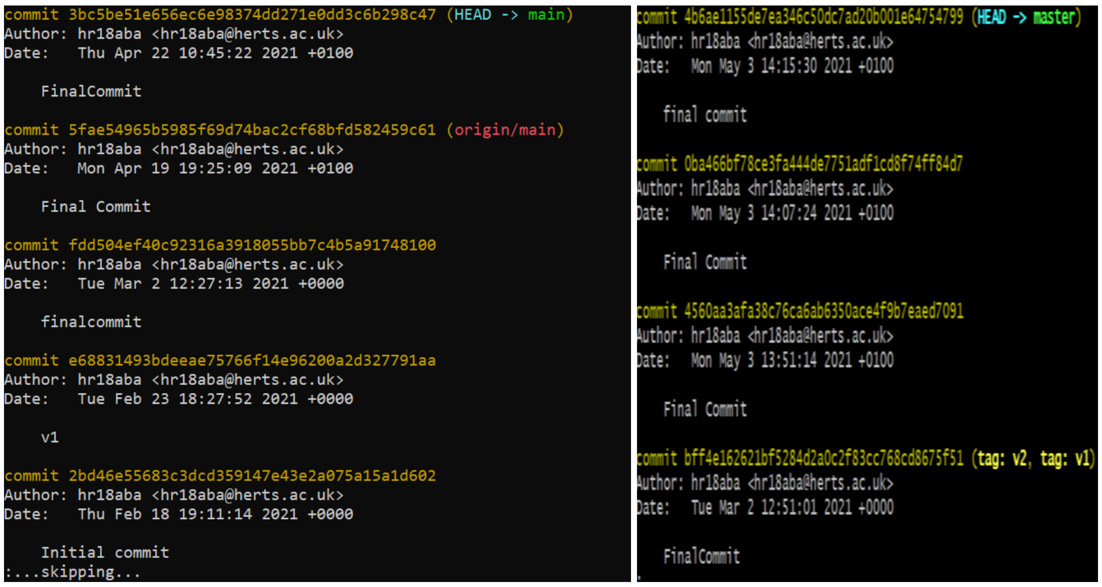
[3] Reft. 2020. Refactoring.com. https://refactoring.guru/design-patterns/command Date Accessed:30/4/2021

[4] Herts. 2020. Herts.instructure.com. https://herts.instructure.com/courses/77532/pages/link-to-recording-of-l22-design-patterns-2?module_item_id=1410832 Date Accessed: 30/4/2021

[5] FindBugs. 2020. Findbugs.sourceforge.net. http://findbugs.sourceforge.net/ Date Accessed: 20/02/2021

[6] Acceptance testing. 2020. Tutorialspoint.com. https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm#:~:text=Acceptance%20testing%2C%20a%20testing%20technique,for%20delivery%20to%20end%20users. Date Accessed: 20/02/2021

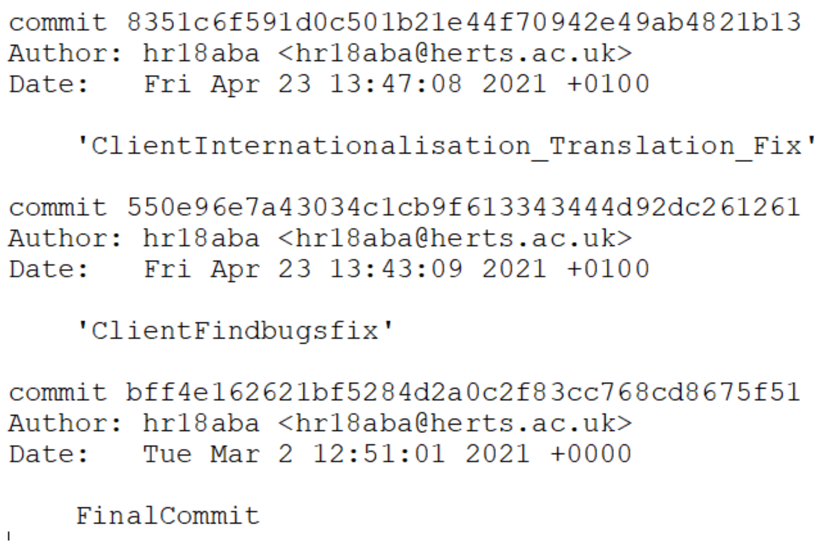
# Chapter 5.0 Appendix



```
commit 3bc5be51e656ec6e98374dd271e0dd3c6b298c47 (HEAD -> main)
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Thu Apr 22 10:45:22 2021 +0100

    FinalCommit

commit 5fae54965b5985f69d74bac2cf68bfd582459c61 (origin/main)
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Mon Apr 19 19:25:09 2021 +0100

    Final Commit

commit fdd504ef40c92316a3918055bb7c4b5a91748100
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Tue Mar 2 12:27:13 2021 +0000

    finalcommit

commit e68831493bdeeae75766f14e96200a2d327791aa
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Tue Feb 23 18:27:52 2021 +0000

    v1

commit 2bd46e55683c3dcd359147e43e2a075a15a1d602
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Thu Feb 18 19:11:14 2021 +0000

    Initial commit
:...skipping...
```

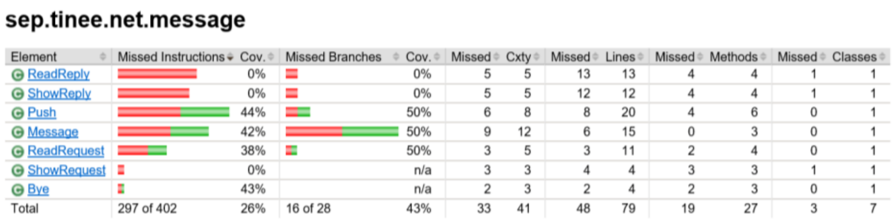
```
commit 4b6ae1155de7ea346c50dc7ad20b001e64754799 (HEAD -> master)
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Mon May 3 14:15:30 2021 +0100

    final commit

commit 0ba466bf78ce3fa444de7751adf1cd8f74ff84d7
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Mon May 3 14:07:24 2021 +0100

    Final Commit

commit 4560aa3afa38c76ca6ab6350ace4f9b7eaed7091
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Mon May 3 13:51:14 2021 +0100

    Final Commit

commit bff4e162621bf5284d2a0c2f83cc768cd8675f51 (tag: v2, tag: v1)
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Tue Mar 2 12:51:01 2021 +0000

    FinalCommit
```

(Figure 32: Project results for log (Generated on Netbeans) . Date Accessed: 20/2/2021 )



```
commit 8351c6f591d0c501b21e44f70942e49ab4821b13
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Fri Apr 23 13:47:08 2021 +0100

    'ClientInternationalisation_Translation_Fix'

commit 550e96e7a43034c1cb9f613343444d92dc261261
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Fri Apr 23 13:43:09 2021 +0100

    'ClientFindbugsfix'

commit bff4e162621bf5284d2a0c2f83cc768cd8675f51
Author: hr18aba <hr18aba@herts.ac.uk>
Date:   Tue Mar 2 12:51:01 2021 +0000

    FinalCommit
```

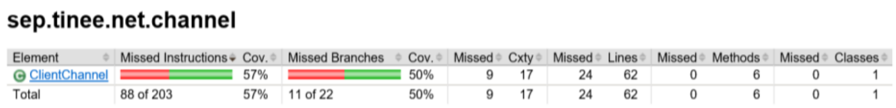
(Figure 33: Project view Change.LogFiles (Generated on Netbeans) . Date Accessed: 20/2/2021 )



**sep.tinee.net.message**

| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ReadReply | | 0% | | 0% | 5 | 5 | 13 | 13 | 4 | 4 | 1 | 1 |
| ShowReply | | 0% | | 0% | 5 | 5 | 12 | 12 | 4 | 4 | 1 | 1 |
| Push | | 44% | | 50% | 6 | 8 | 8 | 20 | 4 | 6 | 0 | 1 |
| Message | | 42% | | 50% | 9 | 12 | 6 | 15 | 0 | 3 | 0 | 1 |
| ReadRequest | | 38% | | 50% | 3 | 5 | 3 | 11 | 2 | 4 | 0 | 1 |
| ShowRequest | | 0% | | n/a | 3 | 3 | 4 | 4 | 3 | 3 | 1 | 1 |
| Bye | | 43% | | n/a | 2 | 3 | 2 | 4 | 2 | 3 | 0 | 1 |
| Total | 297 of 402 | 26% | 16 of 28 | 43% | 33 | 41 | 48 | 79 | 19 | 27 | 3 | 7 |

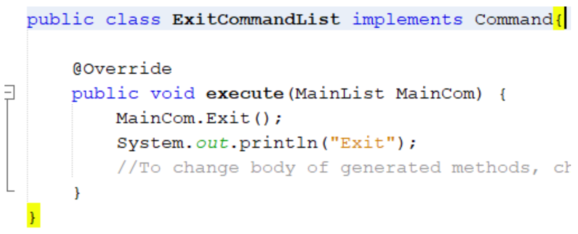
(Figure 34: JaCoCoverage Analysis of net.message (Generated on Netbeans) . Date Accessed: 20/2/2021 )



**sep.tinee.net.channel**

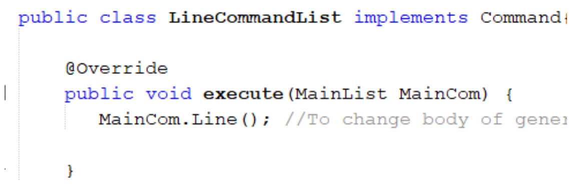
| Element | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods | Missed | Classes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ClientChannel | | 57% | | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | 1 |
| Total | 88 of 203 | 57% | 11 of 22 | 50% | 9 | 17 | 24 | 62 | 0 | 6 | 0 | 1 |

(Figure 35 : JaCoCoverage Analysis of net.Channel (Generated on Netbeans) . Date Accessed: 20/2/2021 )



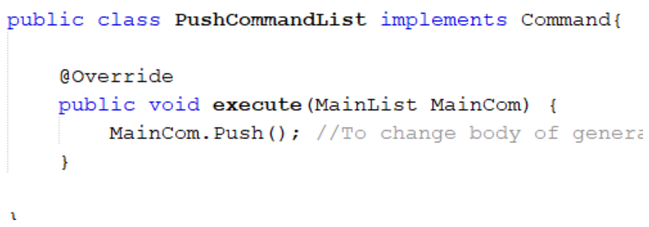
```
public class ExitCommandList implements Command{

    @Override
    public void execute(MainList MainCom) {
        MainCom.Exit();
        System.out.println("Exit");
        //To change body of generated methods, ch
    }
}
```

(Figure 36: Project view of Concrete Commands (Generated on Netbeans) . Date Accessed: 20/2/2021 )



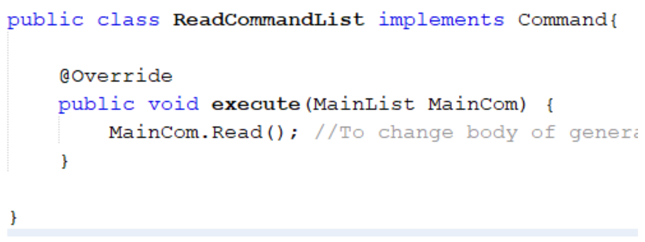
```
public class LineCommandList implements Command{

    @Override
    public void execute(MainList MainCom) {
       MainCom.Line(); //To change body of gene

    }
```

(Figure 37: Project view of Concrete Commands (Generated on Netbeans) . Date Accessed: 20/2/2021 )



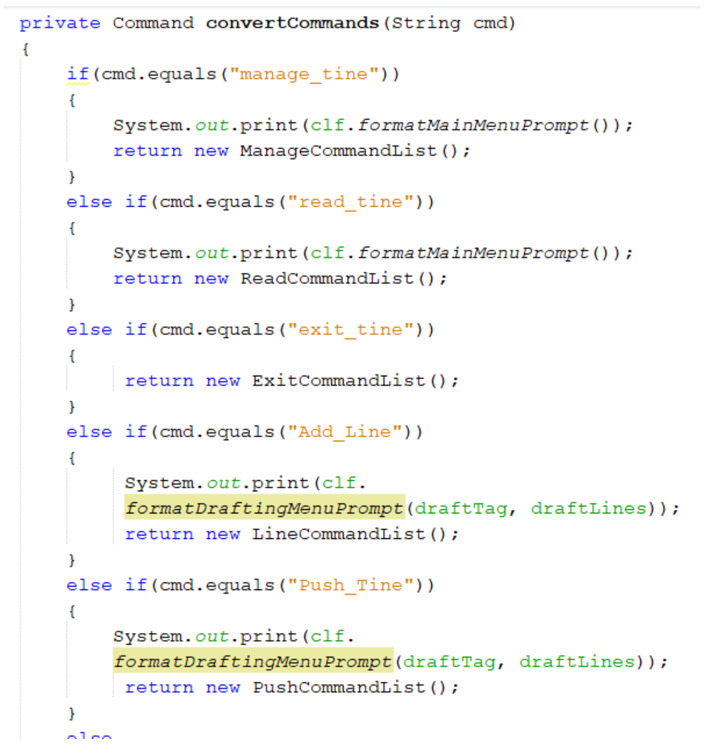
```
public class PushCommandList implements Command{

    @Override
    public void execute(MainList MainCom) {
        MainCom.Push(); //To change body of genera
    }
```

(Figure 38: Project view of Concrete Commands (Generated on Netbeans) . Date Accessed: 20/2/2021 )



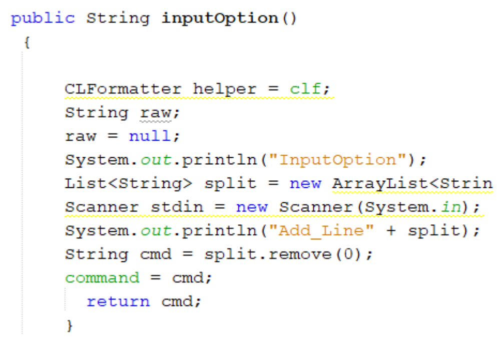
```
public class ReadCommandList implements Command{

    @Override
    public void execute(MainList MainCom) {
        MainCom.Read(); //To change body of genera
    }

}
```

(Figure 39: Project view of Concrete Commands (Generated on Netbeans) . Date Accessed: 20/2/2021 )



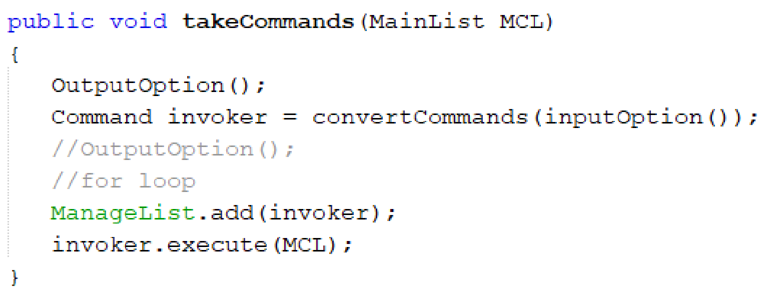
```
private Command convertCommands(String cmd)
{
    if(cmd.equals("manage_tine"))
    {
        System.out.print(clf.formatMainMenuPrompt());
        return new ManageCommandList();
    }
    else if(cmd.equals("read_tine"))
    {
        System.out.print(clf.formatMainMenuPrompt());
        return new ReadCommandList();
    }
    else if(cmd.equals("exit_tine"))
    {
         return new ExitCommandList();
    }
    else if(cmd.equals("Add_Line"))
    {
         System.out.print(clf.
         formatDraftingMenuPrompt(draftTag, draftLines));
         return new LineCommandList();
    }
    else if(cmd.equals("Push_Tine"))
    {
        System.out.print(clf.
        formatDraftingMenuPrompt(draftTag, draftLines));
         return new PushCommandList();
    }
```

(Figure 40: Project view of Invoker Class (Generated on Netbeans) . Date Accessed: 20/2/2021 )



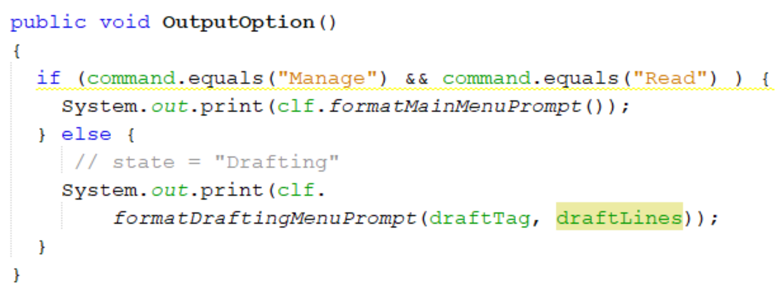
```
public String inputOption()
 {

    CLFormatter helper = clf;
    String raw;
    raw = null;
    System.out.println("InputOption");
    List<String> split = new ArrayList<Strin
    Scanner stdin = new Scanner(System.in);
    System.out.println("Add_Line" + split);
    String cmd = split.remove(0);
    command = cmd;
      return cmd;
    }
```

(Figure 41: Project view of Invoker Class (Generated on Netbeans) . Date Accessed: 20/2/2021 )



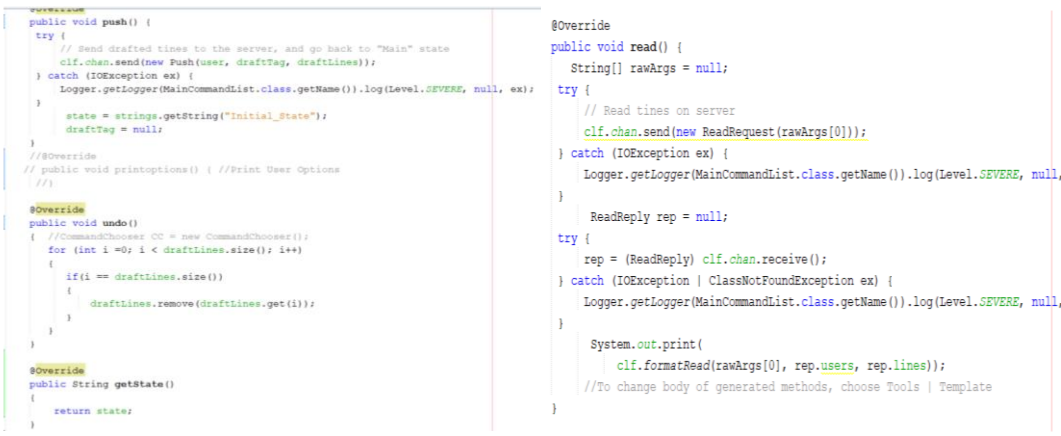
```
public void takeCommands(MainList MCL)
{
    OutputOption();
    Command invoker = convertCommands(inputOption());
    //OutputOption();
    //for loop
    ManageList.add(invoker);
    invoker.execute(MCL);
}
```

(Figure 42: Project view of Invoker Class (Generated on Netbeans) . Date Accessed: 20/2/2021 )



```java
public void OutputOption()
{
  if (command.equals("Manage") && command.equals("Read") ) {
    System.out.print(clf.formatMainMenuPrompt());
  } else {
     // state = "Drafting"
    System.out.print(clf.
        formatDraftingMenuPrompt(draftTag, draftLines));
  }
}
```

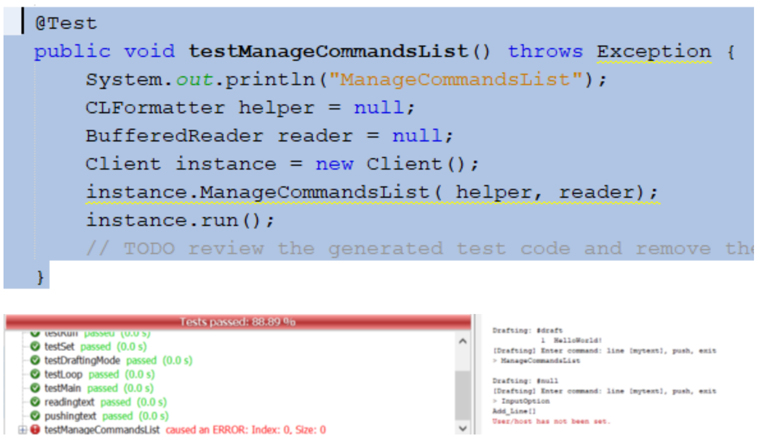
(Figure 43: Project view of Invoker Class (Generated on Netbeans) . Date Accessed: 20/2/2021 )



```java
public void push() {
 try {
     // Send drafted tines to the server, and go back to "Main" state
     clf.chan.send(new Push(user, draftTag, draftLines));
 } catch (IOException ex) {
     Logger.getLogger(MainCommandList.class.getName()).log(Level.SEVERE, null, ex);
 }
      state = strings.getString("Initial_State");
      draftTag = null;
}
//@Override
// public void printoptions() { //Print User Options
 //}

@Override
public void undo()
{  //CommandChooser CC = new CommandChooser();
   for (int i =0; i < draftLines.size(); i++)
   {
      if(i == draftLines.size())
      {
         draftLines.remove(draftLines.get(i));
      }
   }
}

@Override
public String getState()
{
    return state;
}
```

```java
@Override
public void read() {
   String[] rawArgs = null;
 try {
     // Read tines on server
     clf.chan.send(new ReadRequest(rawArgs[0]));
 } catch (IOException ex) {
     Logger.getLogger(MainCommandList.class.getName()).log(Level.SEVERE, null, ex);
 }
      ReadReply rep = null;
 try {
     rep = (ReadReply) clf.chan.receive();
 } catch (IOException | ClassNotFoundException ex) {
     Logger.getLogger(MainCommandList.class.getName()).log(Level.SEVERE, null, ex);
 }
      System.out.print(
          clf.formatRead(rawArgs[0], rep.users, rep.lines));
     //To change body of generated methods, choose Tools | Template
}
```

(Figure 44: Project view of Definer class (Generated on Netbeans) . Date Accessed: 20/2/2021 )



```java
@Test
public void testManageCommandsList() throws Exception {
    System.out.println("ManageCommandsList");
    CLFormatter helper = null;
    BufferedReader reader = null;
    Client instance = new Client();
    instance.ManageCommandsList( helper, reader);
    instance.run();
    // TODO review the generated test code and remove th
}
```

```
Tests passed: 88.89 %
  testRun passed (0.0 s)
  testSet passed (0.0 s)
  testDraftingMode passed (0.0 s)
  testLoop passed (0.0 s)
  testMain passed (0.0 s)
  readingtext passed (0.0 s)
  pushingtext passed (0.0 s)
  testManageCommandsList caused an ERROR: Index: 0, Size: 0
```

```
Drafting: #draft
        1  HelloWorld!
[Drafting] Enter command: line [mytext], push, exit
> ManageCommandsList

Drafting: #null
[Drafting] Enter command: line [mytext], push, exit
> InputOption
Add_Line[]
User/host has not been set.
```

(Figure 45: Project view of Command Pattern Test (Generated on Netbeans) . Date Accessed: 20/2/2021 )