|  |
| --- |
|  |
| CS4218 Milestone 1 |
|  |

|  |
| --- |
|  |

**Team 05**

KOH CHENG GEE A0126199W

MUHAMMAD RAZALI A0133267H

PHOON QIANONN A0126232U

TANG WEI REN A0125531R

Contents

[1. Implementation for Basic Functionality (BF) 3](#_Toc475310854)

[1.1 Implementation for Shell: Calling applications 3](#_Toc475310855)

[1.2 Implementation for Shell: Quoting 3](#_Toc475310856)

[1.3 Implementation for Shell: Semicolon Operator 3](#_Toc475310857)

[1.4 Implementation for Applications: cat 3](#_Toc475310858)

[1.5 Implementation for Applications: cd 3](#_Toc475310859)

[1.6 Implementation for Applications: pwd 3](#_Toc475310860)

[1.7 Implementation for Applications: echo 3](#_Toc475310861)

[1.8 Implementation for Applications: head 3](#_Toc475310862)

[1.9 Implementation for Applications: tail 3](#_Toc475310863)

[2. Implementation for Extended Functionality 1 (EF1) 3](#_Toc475310864)

[2.1 Implementation for Shell: Globbing 3](#_Toc475310865)

[2.2 Implementation for Shell: IO-redirection 3](#_Toc475310866)

[2.3 Implementation for Applications: cal 3](#_Toc475310867)

[2.4 Implementation for Applications: grep 4](#_Toc475310868)

[2.5 Implementation for Applications: sort 4](#_Toc475310869)

[3. Unit Testing for Basic Functionality (BF) 4](#_Toc475310870)

[3.1 Unit Testing for Shell: Calling applications 4](#_Toc475310871)

[3.2 Unit Testing for Shell: Quoting 4](#_Toc475310872)

[3.3 Unit Testing for Shell: Semicolon Operator 4](#_Toc475310873)

[3.4 Unit Testing for Applications: cat 4](#_Toc475310874)

[3.5 Unit Testing for Applications: cd 4](#_Toc475310875)

[3.6 Unit Testing for Applications: pwd 5](#_Toc475310876)

[3.7 Unit Testing for Applications: echo 5](#_Toc475310877)

[3.8 Unit Testing for Applications: head 5](#_Toc475310878)

[3.9 Unit Testing for Applications: tail 5](#_Toc475310879)

[4. Unit Testing of Extended Functionality 1 (EF1) 5](#_Toc475310880)

[4.1 Unit Testing for Shell: Globbing 5](#_Toc475310881)

[4.2 Unit Testing for Shell: IO-redirection 5](#_Toc475310882)

[4.3 Unit Testing for Applications: cal 5](#_Toc475310883)

[4.4 Unit Testing for Applications: grep 6](#_Toc475310884)

[4.5 Unit Testing for Applications: sort 6](#_Toc475310885)

[5. Test Cases for Extended Functionality 2 (EF2) 6](#_Toc475310886)

[5.1 Test Cases for Shell: Pipe Operator 6](#_Toc475310887)

[5.2 Test Cases for Shell: Command Substitution 6](#_Toc475310888)

[5.3 Test Cases for Applications: date 6](#_Toc475310889)

[5.4 Test Cases for Applications: sed 6](#_Toc475310890)

[5.5 Test Cases for Applications: wc 7](#_Toc475310891)

# Implementation for Basic Functionality (BF)

## Implementation for Shell: Calling applications

* Shell splits call command into arguments and redirection operators.
* Filenames are expanded.
* Application name is resolved.
* Specified application is executed.

## Implementation for Shell: Quoting

* Our shell supports three kinds of quotes:
  + Single quote (‘)
  + Double quotes (“)
  + Backquotes (`)
* The ‘ and “ are used to disable interpretation of all or some special characters
* The ` used to make command substitution
* Special characters are: \t (tab), \* (globbing), ‘ (single quote), “ (double quote), ` (backquote), | (pipe), < (input redirection), > (output redirection), ; (semicolon), space
* Using regex and pattern matching to search the quote
* If quote found then run the corresponding method

## Implementation for Shell: Semicolon Operator

* The semicolon operator split a command into 2 command.
* Run the first command; when the first command terminates, run the second command.
* If an exception is thrown during the execution of the first command, the execution of the whole command must be terminated.
* Using regex and split to split the command

## Implementation for Applications: cat

|  |  |
| --- | --- |
| Command | Description |
| cat | Outputs error message |
| Cat [filepath]+ | Checks validity of file path. If file could not be found or read, an empty string is sent as an output.  Outputs the contents of the file. |

## Implementation for Applications: cd

The cd command changes the current working directory

|  |  |
| --- | --- |
| Command | Description |
| cd | change to home directory |
| cd ~ | Change to home directory |
| cd / | Change to the root directory  For Mac or Linux   * Linux uses “/” for its “file separator” * However, for Linux, if a user entered “\” as the “file separator” instead of “/” there will be no auto correction as “\” is a valid folder & file name in the Linux. An appropriate error message will be display if the “path” entered does not exist in the shell system. |
| cd / | Change to the root directory  For Windows   * Windows uses “\” for its “file separator”. * If “/” is mistakenly used as the “file separator” in the “path” entered, the shell system will auto correct “/” to “\”. * The following observation was made in the “command prompt” on Windows, hence this correction was implemented to help ease the user as their intention is clear. * As “/” is an invalid folder & file name in Windows, the auto-correction will not cause any issues |
| cd \ | For Windows:   * Windows uses “\” for its “file separator”. |
| cd . | No change in directory. Remain at current working directory |
| cd .. | Change to parent directory |
| cd [PATH] | Change the directory to the “PATH”. If the “PATH” does not exist it will return an error message   * Windows uses “\” for its “file separator”. * If the shell system is run on Windows, auto correction of the “file separator” from “/” to “\” will take if “/” is mistakenly used as the “file separator” in the “PATH” entered. * The following observation was made in the “command prompt” on Windows, hence this correction was implemented to help ease the user as their intention is clear. * As “/” is an invalid folder & file name in Windows, the auto-correction will not cause any issues * “PATH” is a “relative path directory” e.g. “folder1/folder2” * “PATH” as specific in the project document cannot be an “absolute path directory” e.g. /user/home/directory”. A “/” (for mac & Linux) and “\” (for Windows) indicates that the “PATH” is an “absolute path directory” hence the “shell system” will return an error message if “/” or “\” is detected as the first char of the “PATH” |

## Implementation for Applications: pwd

The pwd command prints the current working directory followed by a newline

|  |  |
| --- | --- |
| Command | Description |
| pwd | Display the current working directory followed by a newline   * Use “*Environment.currentDirectory*” to get the current working directory * Add “*System.lineSeparator*” to insert a newline after displaying the results |

## Implementation for Applications: echo

|  |  |
| --- | --- |
| Command | Description |
| echo | Prints new line |
| Echo [args]\* | Prints arguments |

## Implementation for Applications: head

**Command: head [OPTION] [PATH]**

|  |
| --- |
| Description |
| The “head” commands prints first N lines of a specific file or input |
| “head” with no PATH and OPTION will print the first 10 lines of the InputStream |
| OPTION is in the format: “-n 3”, where “-n” is the command and “3” is the number of lines |
| PATH is a “relative path directory” e.g. “folder1/folder2/123.txt” |
| “head” command accepts the rightmost OPTION if more than 1 OPTION exists. e.g. “head –n 3 –n 5 –n 2 123.txt”. It accepts –n 2 as the OPTION |
| “head” command accepts multiple PATH in the command. e.g. “head –n 6 123.txt 246.txt”. It prints out the first 6 lines of 123.txt and 246.txt files |
| OPTION can input be behind the PATH. e.g. “head 123.txt –n 6” |
| Other than the OPTION, any command that starts with “-” is invalid |
| Any input other than the OPTION will be regarded as PATH e.g. “head 1 5.txt 7”. The PATH of the inputs are: 1, 5.txt and 7 |

|  |
| --- |
| Methods & Description |
| public void run(String args[], InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method process the arguments and print out the details in console |
| private void checkArgumentLengthAndRun(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method check the details within args and run methods according to the args length |
| private void readEveryFilePathWithLineNumbers(OutputStream stdout, int lineNumber, ArrayList<String> listOfArgs)   * stdout takes in outputStream * lineNumber is the number of lines to be display on console * listOfArgs is the list that stores the PATH of files * this method reads the first lineNumber of lines in each file of listOfArgs and display in console through stdout |
| private void readEveryFilePath(String[] args, OutputStream stdout, ArrayList<String> listOfArgs)   * args[] is an empty array * stdout takes in outputStream * listOfArgs is the list that stores the PATH of files * this method lets args get each individual PATH for listOfArgs, reads the PATH from args and display on the console through stdout |
| private int checkDuplicateLineNumbers(String[] args1, ArrayList<String> listOfArgs)   * args[] takes in an array of String which consists of OPTION and PATH * listOfArgs is an empty ArrayList * this method checks the validity in args1, adding the PATHs to listOfArgs and return the rightmost OPTION |
| private void readWithLinesAndDirectory(String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdout takes in outputStream * this method reads the file of PATH and display first N lines given in args and display through stdout |
| private void readWithLines(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display first N lines given in args and display through stdout |
| private void readFromPath(String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of PATH * stdout takes in outputStream * this method reads the file from PATH and display through stdout |
| private void readFromStdin(InputStream stdin, OutputStream stdout)   * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display the first 10 lines through stdout |
| private boolean checkNullInput(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method checks whether all args, stdin and stdout and return true if all 3 are null |

## 

## Implementation for Applications: tail

**Command: tail [OPTION] [PATH]**

|  |
| --- |
| Description |
| The “tail” commands prints last N lines of a specific file or input |
| “tail” with no PATH and OPTION will print the first 10 lines of the InputStream |
| OPTION is in the format: “-n 3”, where “-n” is the command and “3” is the number of lines |
| PATH is a “relative path directory” e.g. “folder1/folder2/123.txt” |
| “tail” command accepts the rightmost OPTION if more than 1 OPTION exists. e.g. “tail –n 3 –n 5 –n 2 123.txt”. It accepts –n 2 as the OPTION |
| “tail” command accepts multiple PATH in the command. e.g. “tail –n 6 123.txt 246.txt”. It prints out the first 6 lines of 123.txt and 246.txt files |
| OPTION can be input behind the PATH. e.g. “tail 123.txt –n 6” |
| Other than the OPTION, any command that starts with “-” is invalid |
| Any input other than the OPTIONT will be regarded as PATH e.g. “tail 1 5.txt 7”. The PATH of the inputs are: 1, 5.txt and 7 |

|  |
| --- |
| Methods & Description |
| public void run(String args[], InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method process the arguments and print out the details in console |
| private void checkArgumentLengthAndRun(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method check the details within args and run methods according to the args length |
| private int checkDuplicateLineNumbers(String[] args1, ArrayList<String> listOfArgs)   * args[] takes in an array of String which consists of OPTION and PATH * listOfArgs is an empty ArrayList * this method checks the validity in args1, adding the PATHs to listOfArgs and return the rightmost OPTION |
| private void readWithLinesAndDirectory(String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdout takes in outputStream * this method reads the file of PATH and display first N lines given in args and display through stdout |
| private void readFromPath (String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display first N lines given in args and display through stdout |
| private void readWithDirectory (String[] args, OutputStream stdout)   * args[] takes in an array of String which consists of PATH * stdout takes in outputStream * this method reads the file from PATH and display through stdout |
| private void readFromStdin(InputStream stdin, OutputStream stdout)   * stdin takes in inputStream * stdout takes in outputStream * this method reads the file from stdin and display the first 10 lines through stdout |
| private boolean checkNullInput(String[] args, InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which consists of OPTION and PATH * stdin takes in inputStream * stdout takes in outputStream * this method checks whether all args, stdin and stdout and return true if all 3 are null |

# Implementation for Extended Functionality 1 (EF1)

## Implementation for Shell: Globbing

* For each argument ARG in a shell command check if it contains unquoted \* (asterisk)
* Find the deepest path which does not contain \*
* Create path regex to search correct file
* Find all the file from that path by using pattern matching
* Store all the correct file into arraylist and pass it to next function as argument

## Implementation for Shell: IO-redirection

* Open InputStream from the file for input redirection (the one following “<”symbol).
* Open the OutputStream to the file for output redirection (the one following “>” symbol).
* If several files are specified for input redirection or output redirection, throw an exception.
* If no files are given, use the NULL value.
* If the file specified for input redirection does not exist, throw an exception.
* If the file specified for output redirection does not exist, create it.

## Implementation for Applications: cal

|  |  |
| --- | --- |
| Command | Description |
| cal | Prints current month, Sunday first |
| cal [year] | Prints all 12 months of year specified, Sunday first.  Prints error if year is invalid format |
| cal [month] [year] | Prints the calendar of the month and year specified. If any of it is invalid, error is sent. |

## Implementation for Applications: grep

|  |  |
| --- | --- |
| Command | Description |
| Grep | Prints new line |
| Grep [pattern] | Prints new line |
| Grep [pattern] [file]+ | Searches for matching strings in file and prints it. |

## Implementation for Applications: sort

* The “sort” commands sort a specific file or input in a sorted order
* The sorted order is defined in the order of “special character”, “numbers”, “capital letters” and “simple letters”
* If “-n” is specific after the “sort” command (e.g. sort [-n] [FILE]), the first word of a line will be treated as a number. The first word will be treated as number only if it contains all numeric characters (e.g 8 days later, so “8” is the first word). If the first word will be treated as a normal word if it contains other types of characters(e.g. 8days later, “8days” is the first word)
* If 2 or more “FILES” are provided, the lines in the “FILES” will be combined and sorted together

|  |
| --- |
| Methods & Description |
| public void run(String args[], InputStream stdin, OutputStream stdout)   * args[] takes in an array of String which contains either “-n” or “FILE” * stdin takes in inputStream * stdout takes in outputStream |
| public String sortStringSimple(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with simple letter |
| public String sortStringCapital(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with capital letter |
| public String sortNumbers(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers |
| public String sortSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars |
| public String sortSimpleCapital(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with capital letter and simple letter. In the order of capital letter followed by simple letter |
| public String sortSimpleNumbers(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers and simple letter. In the order of numbers followed by simple letter |
| public String sortSimpleSpecialChars (String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with capital letter and simple letter. In the order of capital letter followed by simple letter |
| public String sortCapitalNumbers(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers and capital letter. In the order of numbers followed by captial letter |
| public String sortCaptialSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars and capital letter. In the order of special chars followed by capital letter. |
| public String sortNumbersSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars and numbers. In the order of special chars and numbers. |
| public String sortSimpleCapitalNumber(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with numbers, capital letter and simple letter. In the order of numbers, capital letter followed by simple letters. |
| public String sortSimpleCapitalSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, capital letter and simple letter. In the order of special chars, capital letter followed by simple letters. |
| public String sortSimpleNumbersSpecialChars (String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, numbers and simple letter. In the order of special chars, numbers followed by simple letters. |
| public String sortCaptialNumbersSpecialChars(String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, numbers and capital letter. In the order of special chars, numbers followed by capital letters. |
| public String sortAll (String toSort)   * toSort takes in String. * Data from InputStream will have to be converted to “String” type first The conversion is done inside public void run(String args[], InputStream stdin, OutputStream stdout). * Sort and return a string of lines starting with special chars, numbers, capital letter, simple letters. In the order of special chars, numbers, capital letters followed by simple letters. |

# Unit Testing for Basic Functionality (BF)

## Unit Testing for Shell: Calling applications

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Cat App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 2 | Cd App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 3 | Pwd App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 4 | Echo App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 5 | Head App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |
| 6 | Tail App Called | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Argument of application, Application name |

## Unit Testing for Shell: Quoting

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Echo No Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 2 | Echo Single Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 3 | Echo Double Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 4 | Echo Back Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | ShellException |
| 5 | Echo Double With Back Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 6 | Echo Single With Back Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |

## Unit Testing for Shell: Semicolon Operator

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Echo Semicolon | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |
| 2 | Echo Semicolon With Exception | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | ShellException |
| 3 | Echo Semicolon With Quotes | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of input with a New Line:‘\n’ at the end |

## Unit Testing for Applications: cat

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Cat without Output Stream | **Arguments**: Valid  **InputStream**: Valid  **OutputStream**: Null | **CatException**, **Message**: cat: OutputStream not provided |
| 2 | Cat with one valid file | **Arguments**: Valid  **InputStream**: Valid  **OutputStream**: Valid | Contents of file with a **New Line:** ‘\n’ at the end |
| 3 | Cat with multiple valid files | **Arguments**: Valid  **InputStream**: Valid  **OutputStream**: Valid | Contents of all fileswith a **New Line:** ‘\n’ at the end |
| 4 | Cat with non-existent file | **Arguments**: Invalid  **InputStream**: Valid  **OutputStream**: Valid | **New Line:** ‘\n’ |
| 5 | Cat with no Arguments and No input stream | **Arguments**: Null  **InputStream**: Null  **OutputStream**: Valid | **CatException**, **Message**: cat: InputStream not provided |
| 6 | Cat with no Arguments, Valid Input Stream Exists | **Arguments**: Null  **InputStream**: Valid  **OutputStream**: Valid | Contents of InputStream |

## Unit Testing for Applications: cd

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | public void testEmptyArgs() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to user home directory |
| 2 | public void testRootDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to root directory |
| 3 | public void testHomeDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to user home directory |
| 4 | public void testCurrentDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Remain in the same directory |
| 5 | public void testParentDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | Return to parent directory |
| 6 | public void testCdToInvalidDirectory() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Null | **CdException, Message:** Cd: The directory does not exist |
| 7 | public void testCdToValidDirectory() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null |  |

## Unit Testing for Applications: pwd

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | public void testPwdWithNoArgs() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | The current working directory is displayed |
| 2 | public void testPwdWithArgs() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Valid | **PwdException, Message:** Pwd: Invalid Arguments |

## Unit Testing for Applications: echo

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Echo without Output Stream | **Arguments:** Valid  **InputStream**: Valid  **OutputStream**: Null | **EchoException:** OutputStream not provided |
| 2 | Echo without arguments | **Arguments:** Null  **InputStream**: Valid  **OutputStream**: Valid | **EchoException:** Null Arguments |
| 3 | Echo empty string | **Arguments:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Empty String** |
| 4 | Echo with arguments | **Arguments:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Arguments** |

## Unit Testing for Applications: head

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methos | Input | Expected Results |
| 1 | testAllNullArgument | **Arguments**: N  **InputStream**: N  **OutputStream**: N | **HeadException**, **Message**: Head: args, stdin, stdout are null |
| 2 | testNullInputStream | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **HeadException**,  **Message**:Head: stdin is null |
| 3 | testNullOutputStream | **Arguments**: V  **InputStream**: V  **OutputStream**: N | **HeadException**,  **Message**:Head: stdout is null |
| 4 | testNoArgument | **Arguments**: N  **InputStream**: V  **OutputStream**: V | **InputStream Message:** “test\nstring” |
| 5 | testOneArgument | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **InputStream Message: “**31423\n 115ewafg\n gaqwtq345 \ntqtqt \nc592859v \ngasgsad" |
| 6 | testTwoArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |
| 7 | testThreeArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |

## Unit Testing for Applications: tail

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | testAllNullArgument | **Arguments**: N  **InputStream**: N  **OutputStream**: N | **TailException**, **Message**: Tail: args, stdin, stdout are null |
| 2 | testNullInputStream | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **TailException**,  **Message**:Tail: stdin is null |
| 3 | testNullOutputStream | **Arguments**: V  **InputStream**: V  **OutputStream**: N | **TailException**,  **Message**:Tail: stdout is null |
| 4 | testNoArgument | **Arguments**: N  **InputStream**: V  **OutputStream**: V | **InputStream Message:** “test\nstring” |
| 5 | testOneArgument | **Arguments**: V  **InputStream**: N  **OutputStream**: V | **InputStream Message: “**31423\n 115ewafg\n gaqwtq345 \ntqtqt \nc592859v \ngasgsad" |
| 6 | testTwoArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |
| 7 | testThreeArgument | **Arguments**: V  **InputStream**: V  **OutputStream**: V | **InputStream Message:**  "31423\n1" |

# Unit Testing of Extended Functionality 1 (EF1)

## Unit Testing for Shell: Globbing

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Case(s) | Input | Expected Results |
| 1 | Cat Glob No Path | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Empty |
| 2 | Cat Glob One File | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |
| 3 | Cat Glob Directories | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |
| 4 | Cat Glob Files Directories | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |
| 5 | Cat Glob Multiple Files Directories | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Valid | Contents of file with a New Line:‘\n’ at the end |

## Unit Testing for Shell: IO-redirection

* Summary of Test cases provided
* What have you covered during testing
* Did you have any plans for generation tests

## Unit Testing for Applications: cal

* Summary of Test cases provided
* What have you covered during testing
* Did you have any plans for generation tests

## Unit Testing for Applications: grep

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | Grep without Output Stream | **Pattern**: Valid  **Files:** Valid  **InputStream**: Valid  **OutputStream**: Null | **GrepException**, **Message**: grep: OutputStream not provided |
| 2 | Grep without pattern and files | **Pattern**: Null  **Files:** Null  **InputStream**: Valid  **OutputStream**: Valid | **New Line:** ‘\n’ |
| 3 | Grep with pattern, without files, without input stream | **Pattern**: Valid  **Files:** Null  **InputStream**: Null  **OutputStream**: Valid | **GrepException**, **Message**: grep: InputStream not provided |
| 4 | Grep with pattern, without files, with valid input stream | **Pattern**: Valid  **Files:** Null  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the input stream containing the matching pattern** |
| 5 | Grep with one valid file | **Pattern**: Valid  **Files:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the file containing the matching pattern** |
| 6 | Grep multiple valid files | **Pattern**: Valid  **Files:** Valid  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the files containing the matching pattern** |
| 7 | Grep with some valid files | **Pattern**: Valid  **Files:** Valid & Invalid  **InputStream**: Valid  **OutputStream**: Valid | **Line(s) from the valid files containing the matching pattern** |
| 8 | Grep with non-existent file | **Pattern**: Valid  **Files:** Invalid  **InputStream**: Valid  **OutputStream**: Valid | **New Line:** ‘\n’ |

## Unit Testing for Applications: sort

Unit Test for interface **public void run(String args[], InputStream stdin, OutputStream stdout)**

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Test Methods | Input | Expected Results |
| 1 | public void testEmptyInputStreamArgs() | **Arguments**: Valid  **InputStream**: Null  **OutputStream**: Null | **SortException, Message:** Input stream empty |
| 2 | public void testInvalidFileName() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Null | **SortException, Message:** error reading file |
| 3 | public void testInvalidArgs() | **Arguments**: Invalid  **InputStream**: Null  **OutputStream**: Null | **SortException, Message:** Sort: error reading file |

Unit Test for Sort Interfaces

The following methods test the interfaces as defined in Sort.java. The test checks if the interfaces returns the correct strings

|  |  |  |
| --- | --- | --- |
| ID | Test Methods | Expected Results |
| 1 | public void testSortStringsSimple() | True |
| 2 | public void testSortStringsCaptial () | True |
| 3 | public void testSortNumbers () | True |
| 4 | public void testSortNumbersWithN | True |
| 5 | public void testSortSpecialChars | True |
| 6 | public void testSortSimpleCaptial | True |
| 7 | public void testSortSimpleNumbers | True |
| 8 | public void testSortSimpleNumbersWithN | True |
| 9 | public void testSortSimpleSpecialChars | True |
| 10 | public void testSortCapitalNumbers | True |
| 11 | public void testSortCapitalNumbersWithN | True |
| 12 | public void testSortCaptialSpecialChars | True |
| 13 | public void testSortNumbersSpecialChars | True |
| 14 | public void testSortNumbersSpecialCharsWithN | True |
| 15 | public void testSortSimpleCapitalNumbers | True |
| 16 | public void testSortSimpleCapitalNumbersWithN | True |
| 17 | public void testSortSimpleCapitalSpecialChars | True |
| 18 | public void testSortSimpleNumbersSpecialChars | True |
| 19 | public void testSortSimpleNumbersSpecialCharsWithN | True |
| 20 | public void testSortCaptialNumbersSpecialChars | True |
| 21 | public void testSortCaptialNumbersSpecialCharsWithN | True |
| 22 | public void testSortAll | True |
| 23 | public void testSortAllWithN | True |