Java File Comparison Documentation:

Written: May 14th 2013

Prepared by: Stephen Yee

Contents

Java File Comparison Documentation: 1

Goal of the Program: 2

Program Operation: 3

Program Output: 4

Directory View: 5

File View: 6

Assumptions: 8

String Comparison Algorithms: 10

SIMPLE 11

Contains: - set to a minimum of 3 consecutive characters 11

LCS 12

Simple vs. Linear vs. Exponential modifiers: 13

Setting Files: 14

Language File: 17

Code Specifications: 18

The File Comparer Object: 19

The Folder Comparer Object: 20

Unimplemented Features: 22

Known Bugs: 23

Goal of the Program:

The program attempts to determine the similarity of two text based files:

- It takes two files and designates one of them as a base file, and the other the reference. (Within the source code, the base is often referred to as left, or file A, while the reference is referred to as right, or file B

- Each line of the base is compared to the reference to look for a line with the best match, we will then make a connection between that line of the base towards the line of the reference. With this we can produce a mapping of each line of the base to the line of the reference.

- The program is designed with code and other new-line separated documents. The program separates blocks of codes to be compared via the new-line character, and also is far more effective at short blocks of texts, rather than large sentences/paragraphs. This leads it to be more designed at finding differences in two versions of codes, or log files.

- The program's output will be an easy to read HTML document.

- The program is capable of performing sub-folder comparisons.

Program Operation:

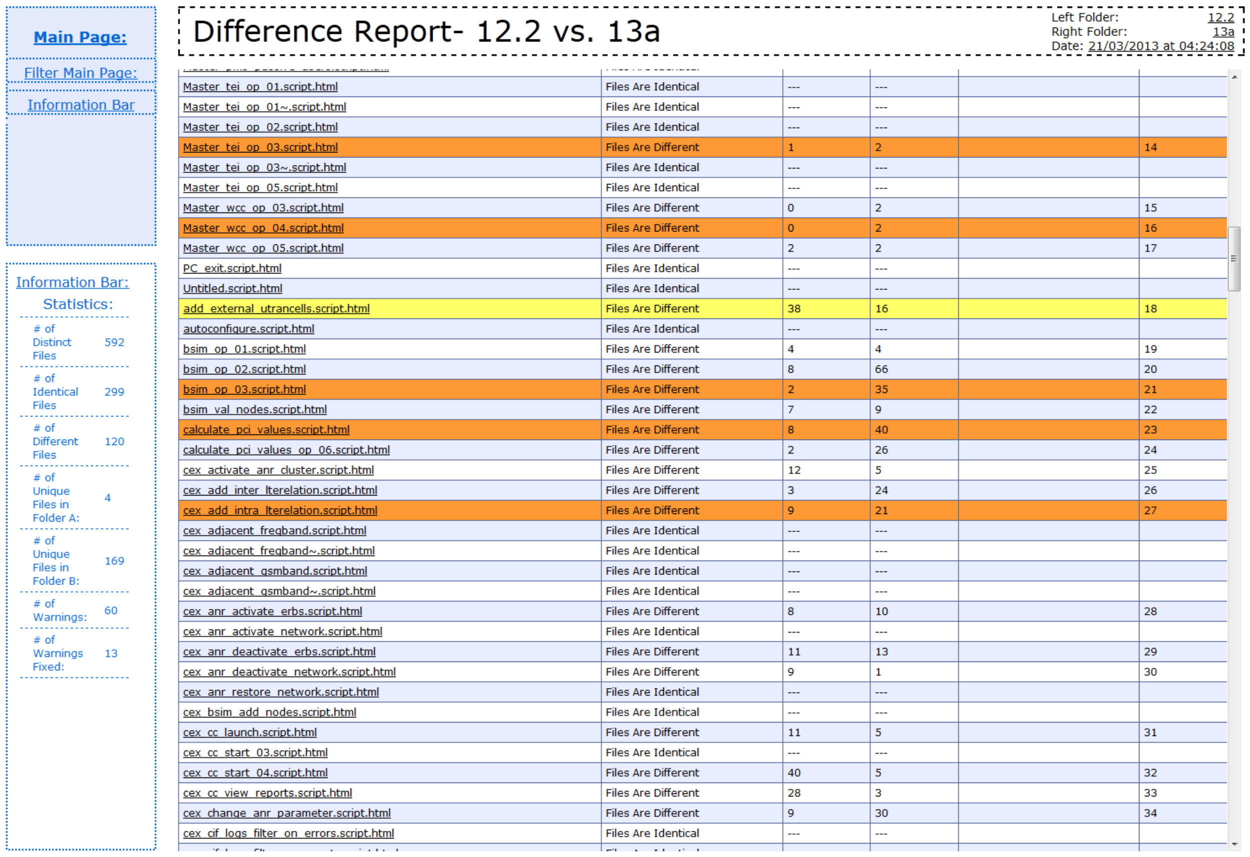
The program is a command line java based application. It is compiled in an executable JAR file, and requires a minimum of 3 arguments.

Argument 1: Folder/File A complete path
Argument 2: Folder/File B complete path
Argument 3: Output report destination complete path

Optional Arguments:
Optional arguments can be specified by stating first: -ARGUMENT <<MODIFIER>>

Name	Names the Output Report
settingFileComp	Points to a setting file or csv list of setting files
settingFileCompReference	Points to a text file containing a list of setting files

Program Output:



Main Page:

Filter Main Page:

Information Bar

Information Bar:

Statistics:

# of Distinct Files	592
# of Identical Files	299
# of Different Files	120
# of Unique Files in Folder A:	4
# of Unique Files in Folder B:	169
# of Warnings:	60
# of Warnings Fixed:	13

Difference Report- 12.2 vs. 13a

Left Folder: 12.2
Right Folder: 13a
Date: 21/03/2013 at 04:24:08

Master_tei_op_01.script.html	Files Are Identical	---	---		
Master_tei_op_01~.script.html	Files Are Identical	---	---		
Master_tei_op_02.script.html	Files Are Identical	---	---		
Master_tei_op_03.script.html	Files Are Different	1	2		14
Master_tei_op_03~.script.html	Files Are Identical	---	---		
Master_tei_op_05.script.html	Files Are Identical	---	---		
Master_wcc_op_03.script.html	Files Are Different	0	2		15
Master_wcc_op_04.script.html	Files Are Different	0	2		16
Master_wcc_op_05.script.html	Files Are Different	2	2		17
PC_exit.script.html	Files Are Identical	---	---		
Untitled.script.html	Files Are Identical	---	---		
add_external_utrancells.script.html	Files Are Different	38	16		18
autoconfigure.script.html	Files Are Identical	---	---		
bsim_op_01.script.html	Files Are Different	4	4		19
bsim_op_02.script.html	Files Are Different	8	66		20
bsim_op_03.script.html	Files Are Different	2	35		21
bsim_val_nodes.script.html	Files Are Different	7	9		22
calculate_pci_values.script.html	Files Are Different	8	40		23
calculate_pci_values_op_06.script.html	Files Are Different	2	26		24
cex_activate_anr_cluster.script.html	Files Are Different	12	5		25
cex_add_inter_lterelation.script.html	Files Are Different	3	24		26
cex_add_intra_lterelation.script.html	Files Are Different	9	21		27
cex_adjacent_freqband.script.html	Files Are Identical	---	---		
cex_adjacent_freqband~.script.html	Files Are Identical	---	---		
cex_adjacent_gsmband.script.html	Files Are Identical	---	---		
cex_adjacent_gsmband~.script.html	Files Are Identical	---	---		
cex_anr_activate_erbs.script.html	Files Are Different	8	10		28
cex_anr_activate_network.script.html	Files Are Identical	---	---		
cex_anr_deactivate_erbs.script.html	Files Are Different	11	13		29
cex_anr_deactivate_network.script.html	Files Are Different	9	1		30
cex_anr_restore_network.script.html	Files Are Identical	---	---		
cex_bsim_add_nodes.script.html	Files Are Identical	---	---		
cex_cc_launch.script.html	Files Are Different	11	5		31
cex_cc_start_03.script.html	Files Are Identical	---	---		
cex_cc_start_04.script.html	Files Are Different	40	5		32
cex_cc_view_reports.script.html	Files Are Different	28	3		33
cex_change_anr_parameter.script.html	Files Are Different	9	30		34
cex_cif_logs_filter_on_errors.script.html	Files Are Identical	---	---		

Pictured above is the Main Report output of the program as viewed in a HTML 5 Enabled browser.

There are 4 distinct regions, in this layout. This layout will be perserved for all HTML pages.

A.) The Top Bar: Displays the name of the current file, as well as some information about the report
B.) The Navigation Bar: Allows one to navigate to subfolders/pages, upper level folders or apply filters.
C.) The Information Bar: Displays some information in regards to the currently viewed file. The Information Bar can be changed by clicking on links in the navigation bar.
D.) The Context Box: Displays the contents of this file.

Within the context box, any row you highlight, will turn dark blue, as a visual indicator of what you currently are viewing.

Directory View:

Highlighted in the Main Report output image is a directory view of the highest level folder . Listed in the context box are the report files for every file that was analyzed. Files that the program believes may have some changes are listed in yellow, files that the program believes may have significant changes are listed in orange. All other files that contain no or minor changes have their backgrounds white or light blue.

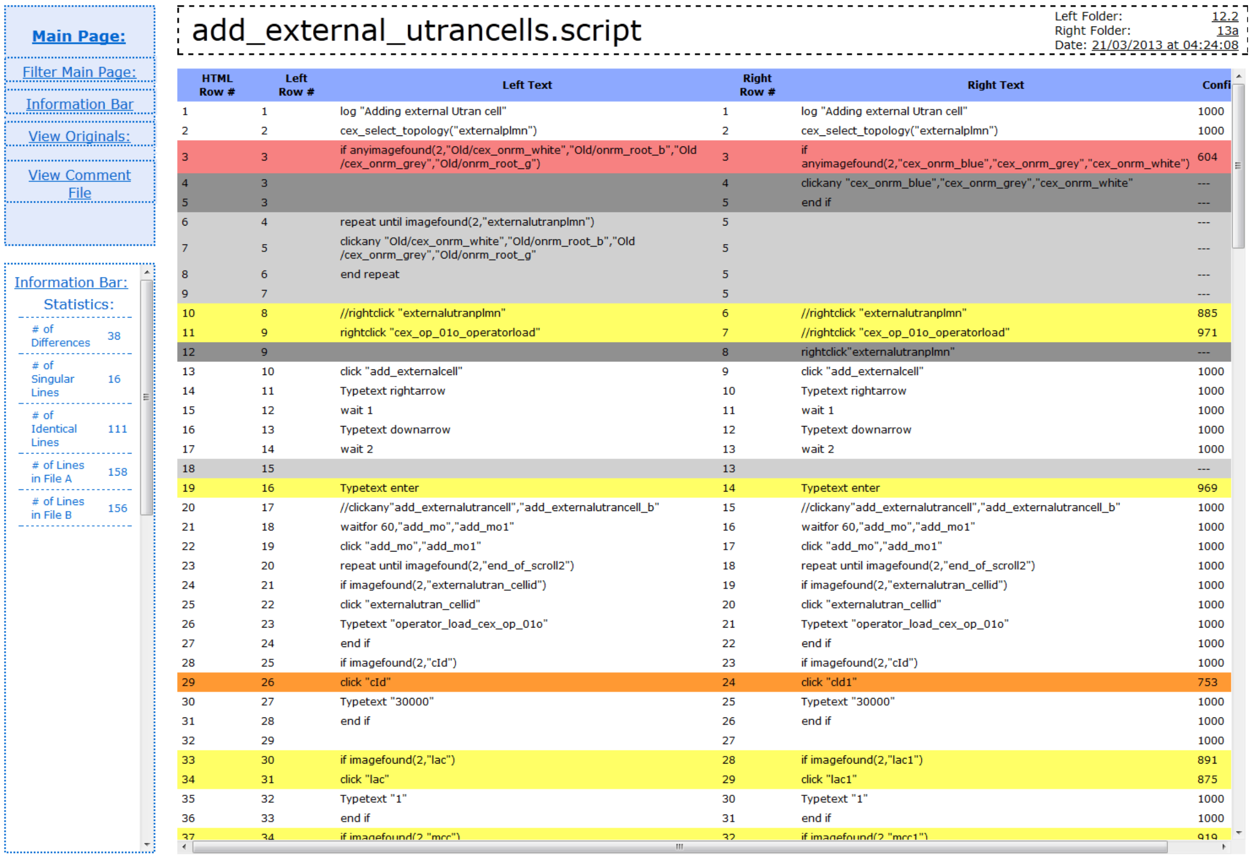
Note: This is the program evaluation, and is not necessarily a true reflection of whether or not there were signifcant changes, based on the settings the program was run, the scrutinization level may not be high enoguh to detect all the changes.

The program will list files as warnings or “fixed warnings”. Fixed warnings are files that initially produced a warning flag, however upon higher scrutinty manage to bring the file back below the warning threshold. However since the more lax settings still detected a warning, they are listed in yellow.

Information Bar:

- Statistics List
    - Contains information of numble of files, number of identical, unique, and differening files
    - Lists number of warnings, and the number of “fixed warnings”
- Legend:
    - Will rpvide an explanation of the row colourings in the context box

File View:



Main Page:

Filter Main Page:

Information Bar

View Originals:

View Comment File

Information Bar:

Statistics:

# of Differences	38
# of Singular Lines	16
# of Identical Lines	111
# of Lines in File A	158
# of Lines in File B	156

add_external_utrancells.script

Left Folder: 12.2
Right Folder: 13a
Date: 21/03/2013 at 04:24:08

HTML Row #	Left Row #	Left Text	Right Row #	Right Text	Confi
1	1	log "Adding external Utran cell"	1	log "Adding external Utran cell"	1000
2	2	cex_select_topology("externalplmn")	2	cex_select_topology("externalplmn")	1000
3	3	if anyimagefound(2,"Old/cex_onrm_white","Old/onrm_root_b","Old /cex_onrm_grey","Old/onrm_root_g")	3	if anyimagefound(2,"cex_onrm_blue","cex_onrm_grey","cex_onrm_white")	604
4	3		4	clickany "cex_onrm_blue","cex_onrm_grey","cex_onrm_white"	---
5	3		5	end if	---
6	4	repeat until imagefound(2,"externalutranplmn")	5		---
7	5	clickany "Old/cex_onrm_white","Old/onrm_root_b","Old /cex_onrm_grey","Old/onrm_root_g"	5		---
8	6	end repeat	5		---
9	7		5		---
10	8	//rightclick "externalutranplmn"	6	//rightclick "externalutranplmn"	885
11	9	rightclick "cex_op_01o_operatorload"	7	//rightclick "cex_op_01o_operatorload"	971
12	9		8	rightclick"externalutranplmn"	---
13	10	click "add_externalcell"	9	click "add_externalcell"	1000
14	11	Typetext rightarrow	10	Typetext rightarrow	1000
15	12	wait 1	11	wait 1	1000
16	13	Typetext downarrow	12	Typetext downarrow	1000
17	14	wait 2	13	wait 2	1000
18	15		13		---
19	16	Typetext enter	14	Typetext enter	969
20	17	//clickany"add_externalutrancell","add_externalutrancell_b"	15	//clickany"add_externalutrancell","add_externalutrancell_b"	1000
21	18	waitfor 60,"add_mo","add_mo1"	16	waitfor 60,"add_mo","add_mo1"	1000
22	19	click "add_mo","add_mo1"	17	click "add_mo","add_mo1"	1000
23	20	repeat until imagefound(2,"end_of_scroll2")	18	repeat until imagefound(2,"end_of_scroll2")	1000
24	21	if imagefound(2,"externalutran_cellid")	19	if imagefound(2,"externalutran_cellid")	1000
25	22	click "externalutran_cellid"	20	click "externalutran_cellid"	1000
26	23	Typetext "operator_load_cex_op_01o"	21	Typetext "operator_load_cex_op_01o"	1000
27	24	end if	22	end if	1000
28	25	if imagefound(2,"cId")	23	if imagefound(2,"cId")	1000
29	26	click "cId"	24	click "cld1"	753
30	27	Typetext "30000"	25	Typetext "30000"	1000
31	28	end if	26	end if	1000
32	29		27		1000
33	30	if imagefound(2,"lac")	28	if imagefound(2,"lac1")	891
34	31	click "lac"	29	click "lac1"	875
35	32	Typetext "1"	30	Typetext "1"	1000
36	33	end if	31	end if	1000
37	34	if imagefound(2,"mcc")	32	if imagefound(2,"mcc1")	919

The program, to the best of it's ability will attempt to line up corresponding lines in each file. It then will provide a score on whether or not it believes these two lines match up. In order to help the user immediately identify problems, lines are colour coated as follows:

White	Program is certain of the match
Yellow	Program is mildly concerned about the match
Orange	Program is moderately concerned about the match
Red	Program is quite concerned about the match
Dark Gray	Program believes that this line exists only in File B
Light Gray	Program believes that this line exists only in File A

Each of the background highlighting, is also a function of the confidence score seen at the very right hand side, these thresholds are set via the program settings files.

Note that confidence, is not purely a function of whether or not the lines contain the same text, but of various other factors as well. These highlighted lines are NOT to indicate that there is a problem, but that a user should draw his/her attention to this line, and determine whether or not there is actually enough cause for concern.

The Information bar contains statistics such as:

- # of differences
- # of singular Lines
- # of identical Lines
- Length of both File A and File B

The Navigation bar contains links to view the original files, as well as the comment file(not currently implemented)

Assumptions:

In order for the program to work a certain number of assumptions must be made.
Here are the assumptions that are made:

- Between two different versions of a program, blocks of code will not be re-arranged. This is because when trying to make a comparison between the base and the reference, we assume the progression is linear.
    - Example:

BLOCK A	BLOCK A
BLOCK B	BLOCK C
BLOCK C	BLOCK B
BLOCK D	BLOCK D

      The program will not correlate Block B and C being re-arranged
- There is a strong correlation to the accuracy of the comparison to the number of lines in the output.
    - Should the program mismatch a line the program will assume that the rest of the following code may be unique to the base or reference. Therefore it will produce 2x number of lines, where x is the number of lines skipped that shouldn't
    - Example:

File A	File B
If(x > 3)	If(x >1)
{	{
functionA1();	functionA1();
y = 3;	y = 3;
functionA2();	functionA2();
}	}
If(x>5)	If(x>5)
{	{
Function B1();	FunctionB1();
Y = 5;	Y= 5;
Functionb2();	FunctionB2();
}	}
REST OF CODE	If (x> 30)
	{
	FunctionC1();
	Y = 30;
	FunctionC2();
	}
	REST OF CODE

Say the comparison matches the if(x>3) in File A with if(x>30) in File B. We would have an output that would appear like:

File A	File B	
If(x > 3) { functionA1(); y = 3; functionA2(); }	If (x> 30) { FunctionC1(); Y = 30; FunctionC2(); }	Some Match
If(x>5) { Function B1(); Y = 5; Functionb2(); }		Unique to A
	If(x >1) { functionA1(); y = 3; functionA2(); } If(x>5) { FunctionB1(); Y= 5; FunctionB2(); }	Unique to B
REST OF CODE	REST OF CODE	Continuing match

Therefore we can detect the accuracy of a program based upon how many extra lines it contains compared to the original files. Note, that this causes a problem where files with significant additions will be viewed as more error prone.

String Comparison Algorithms:

A String comparison Algorithm produces a score on how similar two strings are. The program contains 3 types of algorithms and 3 modifiers. Note that the starting point of an algorithm may affect the result. Both the match and contains algorithm may produce differing results depending whether they start at the front or end of a String. In order to solve, the program by default compares them both ways and takes the highest score of the two.

- Match:
  Simple algorithms score when the character at index X of both String A and String B are the same.

- Contains:
  The contains algorithm looks for substrings of at least minimum length. It then proceeded in a linear fashion attempting to increase the length of the substring. Should the substring be broken up, the program will, remove all the parts of the string it has already evaluated, and perform the operation again on the remaining String.

- LCS:
  The LCS uses the longest common subsequence algorithm, with a dynamic programming approach. The LCS is the algorithm used by the diff command in Unix/Linux based OSs, and searches for the longest substring of a contained in b. Note that this will also contain single characters, and therefore does not as accurately reflect the similarity of words as contains does.

  Examples:
  We will use the two strings:
    - String A: the cat saw us by the beach

- String B: she sat all of us by the beach

SIMPLE

t	h	e	c	a	t	s	a	w	u	s	b	y	t	h	e	b	e	a	c	h		
s	h	e	s	a	t	a	l	l	o	f	u	s	b	y	t	h	e	b	e	a	c	h
	x	x		x	x																	

The only characters that occupy the same index at the same time are the 4 in the beginning, therefore this will produce a score of: 4/23

If we were to evaluate this instead, from the back:

h	c	a	e	b	e	h	t	y	b	s	u	w	a	s	t	a	c	e	h	t		
h	c	a	e	b	e	h	t	y	b	s	u	f	o	l	l	a	t	a	s	e	h	s
x	x	x	x	x	x	x	x	x	x	x	x					x						

This will produce a score of: 13/23

Contains: - set to a minimum of 3 consecutive characters

Forward Algorithm:

Iteration 1: searching for “the” in string B

Matched:

String A: thecatsawusbythebeach

String B: shesatallofusbythebeach

Found: “the”

The program will take “the” in String A and search for the first case of “the” in String B. Therefore the program will discount everything in String B before “the”.

Iteration 2:

Matched: the

String A: catsawusbythebeach

String B: beach

The program will then continually remove characters from String A until it produces a consecutive set of characters at least 3 letters long that remain in string B.

Iteration 3: Searching for “cat”

Iteration 4: searching for “ats”

Iteration 5: Searching for "tsa"
…
Iteration 16: Searching for "bea"
Found corresponding string in String B:

Matched Characters: "beach"

Final result: Matched: thebeach --- length: 8
Longest String is: 23

Score of 8/23

Reverse Algorithm:
Iteration 1: searching for "hca" in String B
Matched:
String A: hcaebehtybsuwastaceht
String B: hcaebehttasufollatasehs

Found: hcaebeht
Iteration 2: searching for "ybs"
Matched: hcaebeht
String A: ybsuwastaceht
String B: tasufollatasehs
Iteration 3: searching for "ybs"
Iteration 4: searching for "bsu"
Iteration 5: searching for "suw"
…
No more matches, as there are no more common 3 letter substrings

Final result: Matched: the beach --- length: 8
Longest String is: 23
Score of 8/23

*note that if the substring match was instead 2, we would have had additional matches in the reverse algorithm

LCS

LCS: Working out the LCS algorithm, for a long string such as the one above using the tabular method, will take a lot of space. As it is a fairly established-algorithm, Wikipedia has a very good description of how the LCS algorithm works: http://en.wikipedia.org/wiki/Longest_common_subsequence_problem

Simple vs. Linear vs. Exponential modifiers:

There are 3 evaluations that can be made, that will cause the program to evaluate words or substrings at the beginning or end of a function more than the other characters when attempting to determine a match.

- The simple function, weights the score of each character equally, therefore characters at the beginning and end have no more importance than one another.

- The Linear function weighs the first character in a string the highest and the last character the least. With each successive character after the first worth one point less than the previous character.

- The Exponential function weights the first character in a string the highest and the last character the least. With each successive character after the first worth, a factor less than the previous character. This factor is usually a number greater than 1 and less than 2, in order to not have that large of an exponential swing.

Setting Files:

In order to decide how the program will perform its comparisons, setting files are used to specify the scrutiny the program and iteration level the program is allowed to perform.

		Requirement
Iteration Count	Describes how many lines ahead the program will look for matches. *Should the program compare line 1 in File A, it will look through lines 1 to 100 in file B, looking for matches*	Type: Integer Range: Value > 0
Line decrement Factor	Describes the penalty a match score will be multiplied by for each line it is off set of	Type: Decimal Range: 0 < Value < 1.0
Line Decrement Step	For each iteration of the program, describes the decrement that the program will modify the Line Decrement factor	Type: Decimal Range: > 0
Line Decrement Max Steps	Describes the maximum number of iterations in which we'll decrement the line decrement steps.	Type: Integer Range: Value > 0
Warning Factor	Describes the percentage of singular lines in reference to the longest file that the output must be composed of to produce a warning flag for the file. *Should File A have 40 liens and File B has 60 lines, the reference point will be 60 lines. If the Warning flag is set to 0.25, then if the output contains 15 singular lines, it will raise the warning flag for this iteration*	Type: Decimal Range: 0 < Value < 1.0
Minimum Confidence Factor	Describes the minimum confidence level a match must make for it to be considered a match. Should the match be below this value, the line is considered singular	Type: Decimal Range: 0 < Value < 1.0
Minimum Confidence Step	Describes the increment value that the program will modify the minimum confidence, for each iteration of a file comparison object that produces a warning flag.	Type: Decimal Range: > 0
Minimum Confidence Max Step	Describes the maximum number of times we'll increment the Minimum confidence Step	Type: Integer range: Value > 0
Iteration Depth Level	**Deprecated**	
Iteration Process	**Deprecated**	
Always Compare Both Ways	The comparison operation does not satisfy the commutative property. This means File A being compared to File B does not necessarily produce the same result as File B being compared to File A. By default: File A is compared to File B, by stating TRUE to the always compare both ways. The program will also compare File B to File A	Boolean: Range: TRUE or FALSE
Compare Both Ways	**If Always Compare Both Ways is False:**	Boolean:

On warning	The program will compare both ways if the warning flag is raised, to see if that resolves the warning.	Range: TRUE or FALSE
Always Compare From Bottom of Page	The program may produce different results if it starts its comparison operations from the start of the file or the bottom of the file. By Default: The program starts from the top of the files. If this is set to TRUE the program will also repeat the operation and compare from the end of the file, to see if it achieves a better result.	Boolean: Range: TRUE or FALSE
Compare From Bottom On Warning	**If Always Compare Both Ways is False:** The program will compare from the bottom of the file if the warning flag is raised, to see if that resolves the warning.	Boolean: Range: TRUE or FALSE
Scrub Comments	**Deprecated, replaced by Ignore Comments**	
Ignore Case	If set to TRUE : The program will make all character comparisons to be CASE insensitive *Therefore if true: a = A evaluates to TRUE*	Boolean: Range: TRUE or FALSE
Ignore Whitespace	If set to TRUE: The program will ignore all whitespace characters.	Boolean: Range: TRUE or FALSE
Ignore Comments	If set to TRUE: The program will ignore all comments as specified by the corresponding Language File.	Boolean: Range: TRUE or FALSE
Ignore Single Line Comments	**If Ignore Comments is set to FALSE:** The program will ignore all single line comments as specified by the corresponding Language File:	Boolean: Range: TRUE or FALSE
Ignore Multi Line Comments	**If Ignore Comments is set to FALSE:** The program will ignore all multi-line comments as specified by the corresponding Language File:	Boolean: Range: TRUE or FALSE
Ignore Numbers	Specifies to the program, to treat all numerical characters as the same. *The two lines:* - *If(A == 10)* - *If(A == 22)* *will evaluate to a comparison score of 1* *However the two liens:* - *If(A == 100)* - *If(A == 3)* *Will NOT because of the differing number of digits of the numbers.*	Boolean: Range: TRUE or FALSE
Minimum # of	Specifies the minimum number of consecutive characters	Type: Integer

Consecutive Characters	required to satisfy the CONTAINS comparison algorithm	Range: > 1
Validate Lines containing only keywords	Should a line contain only keywords, the program is to wait for further validation, before confirming that it's comparison is correct. *Because some statements might occur frequently, as they are keywords in programming such as: long bracketing, or while(true) statements etc, the program will verify using additional comparison operations to determine if it's assumption is true*	Boolean: Range: TRUE or FALSE
Language Type	Specifies which language files the program should use	Type: String, must match a language file
String Algorithms to use	Specifies, which Algorithms to use, and what their weighting factor should be. Algorithms should be separated by ampersands (&), and the algorithm name should be listed followed by a colon (:) then the weighting factor. The weighting factors must add up to 1. *Example:* *LinearMatch:0.5 & ExponentialMatch:0.5*	Type: Custom ALGORITHM1:WEIGHT1 & ALGORITHM2:WEIGHT2 … The sum of the weights must equal 1
HTML Output Properties:		
Highlight Keywords	Specifies whether in the output file the keywords in the language file be highlighted	Boolean: Range: TRUE or FALSE
Thresholds	Specifies the minimum comparison values needed to have the cell highlighted a certain colour.	Value: Integers Range: Three Integers of decreasing value < 1000

Language File:

A language file contains specifications on how to evaluate the strings of a particular file.

It contains the following fields:

- Single Line Comment Indicator: A String that indicates all characters on this line passed this string are comments
- Multi line comment start Indicator: A String that indicates all the lines and characters after this character are inadmissible until we find the multi line comment end indicator
- Multi line comment end indicator: See Above
- List of Strings to validate:
    - A list of strings that may occur multiple times, and would require validation
- List of Keywords A:
    - A list of keywords that could be highlighted by the HTML Writer Object, in a specific format
- List of Keywords B:
    - A list of keywords that could be highlighted by the HTML Writer Object, in a specific format
- List of Keywords C
    - A list of keywords that could be highlighted by the HTML Writer Object, in a specific format

  Note that A,B,C will all be highlighted differently.

By default: if no language file is specified it will default to basic template.

Code Specifications:

The program has 3 main objects that are created numerous times for the purpose of performing the task.

- Folder Comparer Object
  - Constructor: Takes two file objects that specifies the directories to the two directories that are to be compared
  - The object is initiated using the start() method.
  - Main Method can then extract three ArrayList<String> from the folder comparer object that contains the paths for files
    - Contained uniquely in A
    - Contained uniquely in B
    - Contained in both A and B
- File Comparer Object
  - Constructor: Takes
    - two file objects that specifies the files that are to be compared
    - A FileComparer Settings Object
    - An ExecutableMain Object, in order to call terminal errors and create top level log errors, rather than defining and throwing errors
  - The object is initiated using the start() method.
  - Main method will then perform the comparison , he output of the program is a String[][] that is accessed by getOutput(); This is the String array representation of the the HTML Table.
- HTML Writer Object
  - Constructor: Takes a String representing the destination path of the file
    - An enum that specifies what kind of HTML document this is
    - A String representing the directory this file belongs in (for internal navigation purposes)
    - A String representing the Main report directory (for internal navigation purposes)
    - An Executable Main Object, in order to call terminal errors, and execute top level code, rather than defining and throwing errors
  -

The File Comparer Object:

Main Logic:

- Converts both file's contents into StringArrays indexed by the new line character
  - Determines if the file comparison should be by inversion and therefore flip the array.
- Generates a FileIndexMatch[]
  - A File Index Match object corresponds to a line in File A. It contains an index that corresponds to the appropriate line in File B, and a score based on the algorithm(s) in question.
  - Therefore a FileIndexMatch[] corresponds to all the lines in the String[] representation of File A
- Generates a String[][] representation of the HTML table.
- Performs any necessary inversions
- Determines if by criteria this comparison has produced a warning

The FileIndexMatch Creation Algorithm:

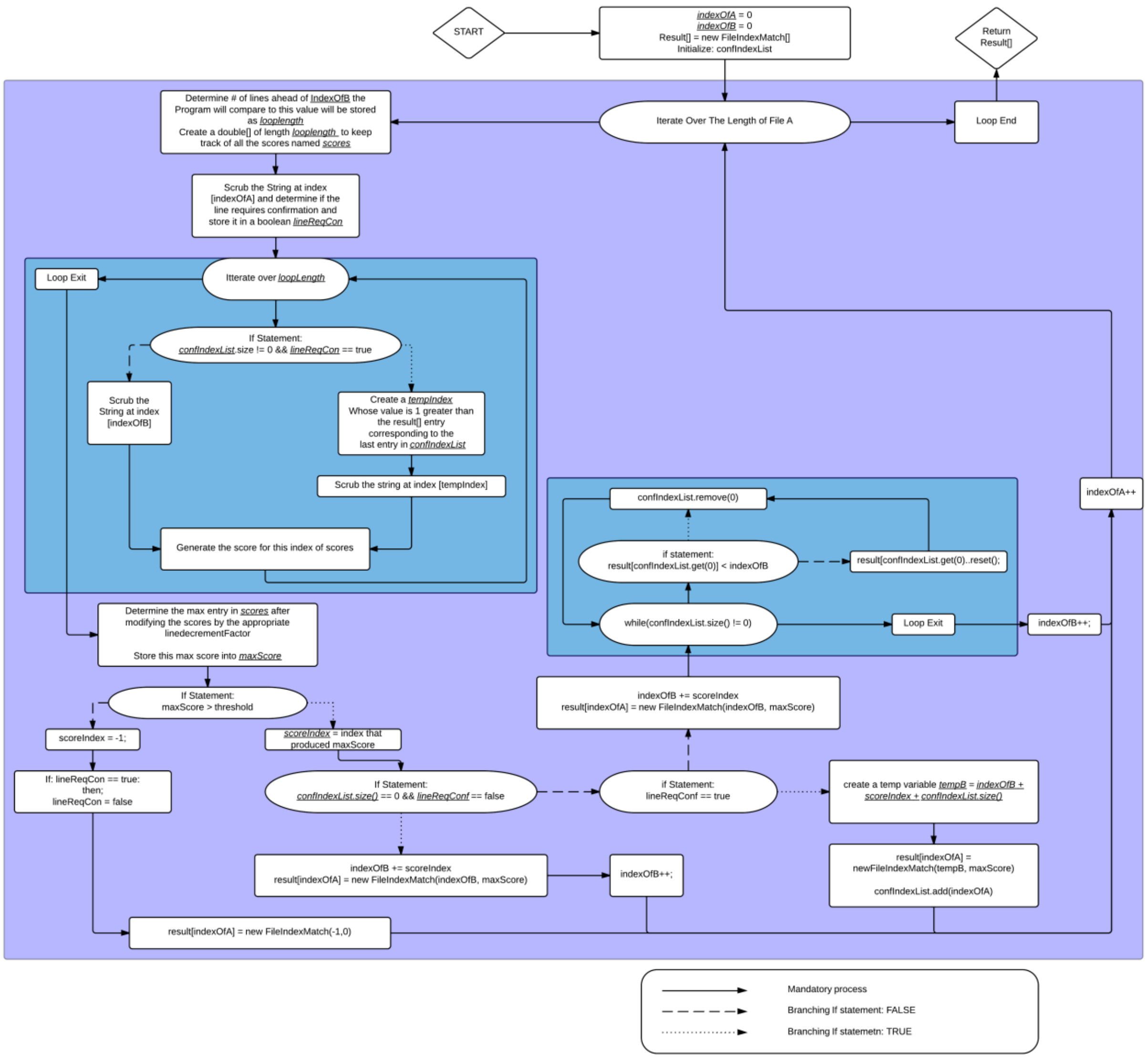
- Variables of note:
  - A variable named **indexOfB,** is used to keep track of the last line of B that we have successfully matched. We increment this line upon successful match as a starting point for all further comparisons. When this variable is > the length of the array, the main loop has been completed
  - ConfirmationIndexList:
    - Some lines may produce multiple matches, due to how common they are, blank lines, statements including while(true), else {, … etc. Therefore we will confirm that these assumptions are correct, by using the line below it to validate. Lines that require validation are stored in this list.

File Comparison Loop Logic:

Note:

Dashed lines represent If evaluated to FALSE

Dotted liens represent if evaluated to TRUE


START
indexOfA = 0
indexOfB = 0
Result[] = new FileIndexMatch[]
Initialize: confIndexList
Return Result[]
Iterate Over The Length of File A
Loop End
Determine # of lines ahead of IndexOfB the Program will compare to this value will be stored as looplength
Create a double[] of length looplength to keep track of all the scores named scores
Scrub the String at index [indexOfA] and determine if the line requires confirmation and store it in a boolean lineReqCon
Loop Exit
Itterate over loopLength
If Statement:
confIndexList.size != 0 && lineReqCon == true
Scrub the String at index [indexOfB]
Create a tempIndex Whose value is 1 greater than the result[] entry corresponding to the last entry in confIndexList
Scrub the string at index [tempIndex]
Generate the score for this index of scores
Determine the max entry in scores after modifying the scores by the appropriate linedecrementFactor
Store this max score into maxScore
confIndexList.remove(0)
if statement:
result[confIndexList.get(0)] < indexOfB
result[confIndexList.get(0)..reset();
while(confIndexList.size() != 0)
Loop Exit
indexOfB++;
indexOfA++
If Statement:
maxScore > threshold
scoreIndex = -1;
scoreIndex = index that produced maxScore
If: lineReqCon == true:
then;
lineReqCon = false
indexOfB += scoreIndex
result[indexOfA] = new FileIndexMatch(indexOfB, maxScore)
If Statement:
confIndexList.size() == 0 && lineReqConf == false
if Statement:
lineReqConf == true
create a temp variable tempB = indexOfB + scoreIndex + confIndexList.size()
indexOfB += scoreIndex
result[indexOfA] = new FileIndexMatch(indexOfB, maxScore)
indexOfB++;
result[indexOfA] =
newFileIndexMatch(tempB, maxScore)
confIndexList.add(indexOfA)
result[indexOfA] = new FileIndexMatch(-1,0)
Mandatory process
Branching If statement: FALSE
Branching If statemetn: TRUE

The Folder Comparer Object:

The Folder comparer object is provided to File objects representing the directories **A and B**.

Comparison Logic:

- Produce File[] arrays indicating the contents of directory A and B
- Create String[] arrays containing the names of the files listed in directory A and directory B
- Create ArrayList<String> objects that represent the files contained **uniquelyInA** and **uniquelyInB** and move the contents of the String[]arrays into their respective ArrayList objects.
    - We assume that all files are unique to their respective directories until proven otherwise
- Sort both lists, for faster comparision
- For each entry in list **uniquelyInA** we perform a binarysearch for it's name in **uniquelyInB**, if we determine that there is a matching index (indicated by >= 0) we will then remove that index in A and B and add it to **containedInBoth**

Unimplemented Features:

- Design a GUI so that clumsy setting files do not have to be used. The GUI could create the setting files on the go, so that we do not have to implement new code to be able to read the settings.
- The comment section of the files currently does not work, this is because it would require the program to be able to modify files on the local drive via an HTML browser, and this inherently has a security flaw:
    - Potential Solution: Use DJSwingBrowser as a Java Program that will view the HTML documents, this allows the javascript to throw a command up to the Java Program and then modify the local saved files.
        - Disadvantage JSwing, requires packages to compile that are unique to each OS. Therefore the executive files of this program are OS dependent.

Known Bugs:

1.) The program unnecessarily penalizes lines that are used to validate lines that require validation by the appropriate line-decrement factors.

    Solution: An exception clause should be coded to restore the line decrement factor if the size of the validation ArrayList > 1.

2.)