CTTJFrame

Import Text File

Read text in

Pass to CTTLogic.AllPairs(String), receive back ArrayList<TestCaseObject> for test output

Print test output

Import Excel File

Read From Text Box

Radial Options

* All Pairs (set n to 2) – only option to start
* All N-th
  + Text Field for N value
* Complete
* Text output – only option to start
* Excel output

CTTLogic

Creation()

AllNth(String, Int nthValue)

Split String Input at “\n” – receive Array [Strings] (if array.length <= 1 return error to user)

Iterate over Array [Strings] filling Array[CTTVariableObject(String)] cttVariableObjectsArray

sortArrray(cttVariableObjectsArray)

ArrayList<TestCaseObject(cttVariableObjectsArray .length)> testCasesFullList

FillVariableNames(cttVariableObjectsArray, testCasesFullList);

fillNthColumns(nth, testCasesFullList, cttVariableObjectArray);

Int caseLocation, #MatchesRequired;

ArrayList<VariableValuesToMatch> matchesNeededList;

Int[] matchLocationsArray;

For(int variableLocation = nthValue; variableLocation < cttVariableObjectsArray.length; variableLocation ++)

{

For(int valueLocation = 0; valueLocation < cttVariableObjectsArray[variableLocation].GetNumberOfValues; valueLocation ++)

{

fillMatchesNeeded(matchesNeededList, cttVariableObjectsArray, variableLocation);

#matchesRequired = variableLocation;

caseLocatoin = 1;

matchLocationsArray = new Int[variableLocation];

While(!matchesNeededList.isEmpty())

If(caseLocation ==1)

While(answerFound(caseLocation, matchLocationsArray, matchesNeededList, #matchesRequired, testCasesFullList)!)

#MatchesRequired = MakeEasiser(#MatchesRequired, testCasesFullList, variableLocation);

Else if(answerFound(caseLocation, matchLocationsArray, matchesNeededList, #matchesRequired, testCasesFullList)!)

While(answerFound(caseLocation, matchLocationsArray, matchesNeededList, #matchesRequired, testCasesFullList)!)

#MatchesRequired = MakeEasiser(#MatchesRequired, testCasesFullList, variableLocation);

AddTestValue(matchesNeededList, caseLocation, testCasesFullList, matchLocationsArray, cttVariableObjectsArray[variableLocation].GetValue(valueLocation))

CleanMatchesNeeded(matchesNeededList);

}

}

sortArray(Array needSorting)

sort by GetNumberofValues – largest first

fillMatchesNeeded(matchesNeededList, cttVariableObjectsArray, int VariableLocation)

matchesNeededList = new ArrayList<>();

for(int x = 0; x < VariableLocation; x++)

{

matchesNeededList.add(VariableValuesToMatch());

For(int y = 0; y < cttVariableObjectsArray[x].GetNumberOfValues; y ++)

{

matchesNeededList[x].Add(cttVariableObjectsArray[x].GetValue(y);

}

}

fillVariableNames(cttVariableObjectsArray, testCasesFullList)

For(int x = 0; x < cttVariableObjects.length; x++)

{

testCasesFullList.get(0).SetValue(x , cttVariableObjects.get(x).GetName())

}

fillNthColumns(nth, testCasesFullList, cttVariableObjectsArray)

int casesNeeded = 1;

for(int x = 0; x < nth; x++)

casesNeeded = casesNeeded\*cttVariableObjectsArray[x].getNumberValues();

for(int x = 0; x < casesNeeded; x++)

testCasesFullList.add(TestCaseObject(cttVariableObjectsArray.length));

for(int x = 0; x < nth; x++)

for(int y = 0;

boolean AnswerFound(caseLocation (may need to be shared by whole class), matchLocationsArray, matchesNeededList, #MatchesRequired, testCasesFullList)

matchLocationsArray = Int[#MatchesRequired];

while(caseLocation<testCasesFullList.length && GetCount(matchLocationsArray) < #MatchesRequired)

if(testCasesFullList.get(caseLocation).getValue(matchesNeededList.length) == “ “)

if(matchesNeededList[matchesNeededList.length-1].get(0) == testCasesFullList(caseLocation).get(matchesNeededList.length-1) || “ “ == testCasesFullList(caseLocation).get(matchesNeededList.length-1))

for(int x = matchesNeededList.length – 2; x >= 0; x--)

if(matchesNeededList[x].contains(testCasesFullList(caseLocation).get(x)) || testCasesFullList(caseLocation).get(x) == “ “)

matchLocationsArray[x] = matchesNeededList[x].getLocation(testCasesFullList(caseLocation).get(x));

else

matchLocationsArray[x] = null;

return (!GetCount(matchLocationsArray)<#MatchesRequired)

int GetCount(matchLocationsArray)

int count = 0;

for(int x = 0; x <matchLocationsArray.length; x++)

if(matchlocationsArray[x] != null)

count +1

Return count;

MakeEasier(#MatchesRequired, testCasesFullList, maxMatchesRequired)

If(#MatchesRequired != 1)

#MatchesRequired -1

else

addRowToTestCases()

#MatchesRequired = maxMatchesRequired;

return #MatchesRequired;

CleanMatchesNeeded(matchesNeededList)

For(x = matchesNeededList.length – 1; x >= 0; x--)

If(matchesNeededList[x].isEmpty())

matchesNeededList.remove(x);

Else

Break;

AddTestValue(matchesNeededList, caseLocation, testCasesFullList, matchLocationsArray, currentVariableValue)

testCasesFullList[caseLocation].setValue(matchLocationsArray.length, currentVariableValue);

for(x = matchLocationsArray.length-1; x >=0; x--)

if(testCasesFullList[caseLocation].getValue(x) == “ “ && matchLocationsArray[x] != null)

testCasesFullList[caseLocation].setValue(x, matchesNeededList[x].getValue(matchLocationsArray[x]));

matchesNeededList[x].removeValue(matchLocationsArray[x]);

addRowToTestCases()

CTTVariableObject

Creation(String input)

Split input at “:” – receive array [Strings]

String variableName = array[0]

Split array[1] at “,” – receive array [strings] = Array variableValues

Iterate over variableValues to remove extra “ “s

GetName()

Return variableName

GetNumberOfValues()

Return variableValues.length()

GetValue(Int location)

Return varaibleValues[location]

TestCaseObject

Creation(Int length)

Fill String[length] testCase with “ “

SetValue(Int location, String value)

testCase[location] = value;

GetValue(Int location)

Return testCase [location]

VariableValuesToMatch

Creation()

Initialize List values

Add(String value)

Values.add(value);

Remove(int location)

Values.remove(location);

Boolean Contains(Sting value)

Return values.contains(value);

Int GetLocation(string value)

Return Values.location(value);