Jordan's Test Plan

numberOfPeople Test

Initial Conditions	A peopleList array has been created using a ParseFile object which contains a list of imported People and their biographical information
Input Test Data	A peopleList and empty Graph object is sent to the createPerson function
Expected Results	A graph is returned which is expected to contain an equal number of nodes as there are entries in peopleList
Test Log Status	Expected length: 22, real length: 22. Test successful

checkPersonData Test

Initial Conditions	A peopleList array has been created using a ParseFile object which contains a list of imported People and their biographical information
Input Test Data	A peopleList and empty Graph object is sent to the createPerson function
Expected Results	A Person node with ID P1 exists within the returned graph with first name Dick, last name Johnson, and suffix Jr.
Test Log Status	P1 person node contained the expected values. Test successful

numberOfRelationships Test

Initial Conditions	A relationshipList array has been created using a ParseFile object which contains the imported relationship biographical data. The graph already contains People nodes.
Input Test Data	A relationshipList and the graph object is sent to the createRelationships function
Expected Results	The returned graph contains an equal number of relationships as there are entries in the relationshipList array
Test Log Status	Expected length: 9, real length: 9. Test successful

checkRelationshipInfo Test

Initial Conditions	A relationshipList array has been created using a ParseFile object which contains the imported relationship biographical data. The graph already contains People nodes.
Input Test Data	A relationshipList and the graph object is sent to the

	createRelationships function
Expected Results	A RelationshipEdge object with its associated Relationship object ID R1 contains a start date of 6/7/1938 and no end date is contained within the return graph
Test Log Status	R1 relationship found and contained the expected values. Test successful

checkRelationshipEdges Test

Initial Conditions	A relationshipList array has been created using a ParseFile object which contains the imported relationship biographical data. The graph already contains People nodes.
Input Test Data	A relationshipList and the graph object is sent to the createRelationships function
Expected Results	Person node with ID P1 is successfully connected via Relationship R6 to the Unknown Person node
Test Log Status	Node with P1 contains R1 in its EdgeSet which is successfully connected to Unknown Person node. Test successful

numberOfParents Test

Initial Conditions	A parentList array has been created using a ParseFile object which contains all children and their parental relationship ID. The graph already contains both Person and Relationship data.
Input Test Data	A parentList and the graph object is sent to the createParents function
Expected Results	A graph is returned with a number of parent relationships equal to 2 * length of parentList (there's one parent relationship for each parent)
Test Log Status	Number of parent relationships: 38, number expected: 38. Test successful

checkParentEdge Test

Initial Conditions	A parentList array has been created using a ParseFile object which contains all the parent-child relationships. The graph already contains both Person and Relationship data.
Input Test Data	A parentList and the graph object is sent to the createParents function

Expected Results	A graph is returned with Person node P19 containing the relationship Child-P6 which connects to the P6 Person node.
Test Log Status	P19 contains Child-P6 in its edgesSet which is connected to P6. Test successful

Robert's Test Plan

onlyAddID()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of "P45" is entered as an input, with all other input fields left blank.
Expected Results	The iterator will stop when it finds "P45" in the graph and return true.
Test Log Status	Expected: P45 Actual: P45 Test passed successfully

onlyAddFullName()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of "P46", a first name of "Maximus", and a last name of "Decimus Meridius" are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds "P46" and check to see if both the first name and last name match (they should).
Test Log Status	Expected: Maximus Actual: Maximus Expected: Decimus Meridius Actual: Decimus Meridius Test passed successfully

onlyAddFirstName()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of “P47” and a first name of “Commodus” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P47” in the graph and check to see if the first name matches (it should).
Test Log Status	Expected: Commodus Actual: Commodus Test passed successfully

onlyAddLastName()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of “P48” and a last name of “Caesar” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P48” in the graph and check to see if the last name matches (it should).
Test Log Status	Expected: Caesar Actual: Caesar Test passed successfully

onlyAddSuffix()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.Blah blah blah
Input Test Data	An ID of “P49” and a suffix of “Mrs.” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P49” in the graph and check to see if the suffix matches (it should).
Test Log Status	Expected: Mrs. Actual: Mrs. Test passed successfully

onlyAddDOB()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of “P50” and a date of birth of “07/12/100” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P50” in the graph and check to see if the birthdate matches (it should).
Test Log Status	Expected: 07/12/100 Actual: 07/12/100 Test passed successfully

onlyAddDOD()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of “P50” and a date of death of “03/15/44” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P50” in the graph and check to see if the date of death matches (it should).
Test Log Status	Expected: 03/15/44 Actual: 03/15/44 Test passed successfully

onlyAddBirthPlace()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of “P51” and a birthplace of “Rome” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P51” in the graph and check to see if the birthplace matches (it should).
Test Log Status	Expected: Rome Actual: Rome Test passed successfully

onlyAddDeathPlace()

Initial Conditions	A new FamilyGraph object and new graph object are instantiated and utilize a helper method called *addPerson* that adds a new person object to the graph.
Input Test Data	An ID of “P52” and a death place of “Gaul” are entered as inputs, with all other input fields left blank.
Expected Results	The iterator will stop when it finds “P52” in the graph and check to see if the death place matches (it should).
Test Log Status	Expected: Gaul Actual: Gaul Test passed successfully

Collin’s Test Plan

readFileLines()

Initial Conditions	A ParseFile object and list type strings is created in order to run getdata on the on the parsefile
Input Test Data	Input of “test” which is a file holding test data that would normally be parsed and turned into a family graph. It is parsed with ParseFile
Expected Results	Expected an int of 58 as this should be the size of the parsed data when put into the list
Test Log Status	Expected: 58 Actual: 58 Test Passed Successfully

readFile()

Initial Conditions	Tests to see if ParseFile can correctly read in the lines of a file. One file named test is used. First, a ParseFile object and string list are created which is filled by the object.getData(). Another list is created but is populated with files.readAlllines of the test file.
Input Test Data	Input of “test” which is a file holding test data that would normally be parsed and turned into a family graph. It is parsed with ParseFile
Expected Results	Two lists that are equal in everyway
Test Log Status	Expected: List

	Actual: List (These are equal, I just don't have the room here to print what they contain) Test Passed Successfully

parseFullPerson()

Initial Conditions	Arraylist holding a hashtable of a person with every field filled out. This will be compared with the first person in the data to be parsed and placed in the graph
Input Test Data	A person with these fields (P1,Johnson,Dick,Jr,9/1/1940,Flint,12/30/2020,Lansing,R1) and test file (same as the 2 tests above)
Expected Results	A person at peopleList(0) with the data P1,Johnson,Dick,Jr,9/1/1940,Flint,12/30/2020,Lansing,R1
Test Log Status	Expected: P1,Johnson,Dick,Jr,9/1/1940,Flint,12/30/2020,Lansing,R1 Actual: P1,Johnson,Dick,Jr,9/1/1940,Flint,12/30/2020,Lansing,R1 Test Passed Successfully

parsePartialPerson()

Initial Conditions	Same as test "parseFullPerson" but some of the fields of the person are missing
Input Test Data	Same as test "parseFullPerson" but some of the fields of the person are missing P2,Johnson,Jane Sarah,,6/15/1942,Saginaw,,,R1
Expected Results	A person at peopleList(1) with the data P2,Johnson,Jane Sarah,,6/15/1942,Saginaw,,,R1
Test Log Status	Expected: P2,Johnson,Jane Sarah,,6/15/1942,Saginaw,,,R1 Actual: P2,Johnson,Jane Sarah,,6/15/1942,Saginaw,,,R1 Test Passed Successfully

parseMoreEmptyPerson()

Initial Conditions	Same as test "parsePartialPerson" but more of the fields of the person are missing

Input Test Data	Same as test “parsePartialPerson” but more of the fields of the person are missing P16,Smith,John J,,,,,,R7
Expected Results	A person at peopleList(10) with the data P16,Smith,John J,,,,,,R7
Test Log Status	Expected: P16,Smith,John J,,,,,,R7 Actual: P16,Smith,John J,,,,,,R7 Test Passed Successfully

parseRelationship()

Initial Conditions	An arraylist that holds a hashtable of a full relationship, Another arraylist that holds hashtables populated by a parser object .getrelationship method. Parser was made with the test file used in previous tests
Input Test Data	Test file called “test.txt” that has relationships in it
Expected Results	A relationship at relationshiplist(0) with data R1,P7,P6,6/7/1938,,St Matthew's Flint Michigan,,,
Test Log Status	Expected: R1,P7,P6,6/7/1938,,St Matthew's Flint Michigan,,, Actual: R1,P7,P6,6/7/1938,,St Matthew's Flint Michigan,,, Test Passed Successfully

parsePartialRelationship()

Initial Conditions	Same as test “parseRelationship” but more of the fields of the relationship are missing
Input Test Data	Test file called “test.txt” that has relationships in it
Expected Results	A relationship at relationshiplist(3) with data R6,,P1,,,,,,
Test Log Status	Expected: R6,,P1,,,,,, Actual: R6,,P1,,,,,,

	Test Passed Successfully

parseChild()

Initial Conditions	An arraylist that holds a hashtable of a full child, Another arraylist that holds hashtables populated by a parser object .getParsedChild method. Parser was made with the test file used in previous tests
Input Test Data	Test file called “test.txt” that has children in it
Expected Results	A child at childList(0) with data R1,P1,,,,,,,
Test Log Status	Expected: R1,P1,,,,,,, Actual: R1,P1,,,,,,, Test Passed Successfully

Michael’s Test Plan

TestGrandparents Test

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	The Relationship array is searched through to find any grandparents and counted.
Expected Results	There to be a equal amount of grandparents to persons labeled as grandparents.
Test Log Status	expected : 26 actual 26 TEST SUCCESSFUL

TestGrandparents19 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p1 , and check the relationships connected too.

Expected Results	That a person node will connect to another person node based on the relationship of Grandparent.
Test Log Status	expected : p19 actual p19 TEST SUCCESSFUL

TestGrand2 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p19 , and check the relationships connected too.
Expected Results	That a person node will connect to another person node based on the relationship of Grandparent
Test Log Status	expected : p1 actual p1 TEST SUCCESSFUL

TestGrand20 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p20, and check the relationships connected too.
Expected Results	That a person node will connect to another person node based on the relationship of Grandparent
Test Log Status	expected : p1 actual p1 TEST SUCCESSFUL

TestGrand10 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p10 , and check the relationships connected too.
Expected Results	That a person node will connect to another person node based on

	the relationship of Grandparent
Test Log Status	expected : p1 actual p19 TEST SUCCESSFUL

TestGrand8 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p8, and check the relationships connected too.
Expected Results	That a person node will connect to another person node based on the relationship of Grandparent
Test Log Status	expected : p1 actual p1 TEST SUCCESSFUL

TestGrand7 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.
Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p7 , and check the relationships connected too.
Expected Results	That a person node will connect to another person node based on the relationship of Grandparent
Test Log Status	expected : p9 actual p9 TEST SUCCESSFUL

TestGrand6 TEST

Initial Conditions	Person,Relationship, Parent arrays are created using the parsefile which has 3 list of data to imported from.

Input Test Data	A iterator is created using the person,Relationship List. Then we search for a individual person p6 , and check the relationships connected too.
Expected Results	That a person node will connect to another person node based on the relationship of Grandparent
Test Log Status	expected : p9 actual p9 TEST SUCCESSFUL