Integrative task 2

Juan Diego Bustamante Obando
Sebastián Jaramillo Torres

ICESI University
Software Systems Engineering
Algorithms and Data Structures
2020

1) Functional Requirements.

The program should:

- **Generate** automatically a given number of persons
- **Show** a progress bar in case the generation of persons takes more than 1 second
- **Save** the generated data to load it when the user opens the program again
- **Create** a person
- **Delete** a person
- **Search** a person by:
    - Name
    - Surname
    - Full name
    - Code
- **Show** a suggestion list when the users search by name, surname or full name

2) Design of test cases

2.1 AVL tree test.

Stage

Name	**Class**	**Stage**
setupEmpty	AVLTreeTest	An empty AVL tree
setupPopulatedTree	AVLTreeTest	An AVL tree with 7 nodes: 5, “brown”; 1,”blue”; 2,”black”; 3,”red”; 4,”beige”; 6,”pink”; 7,”white”

Test cases

Goal of the test: Verify that the add method adds a node successfully into the AVL tree				
Class	**Method**	**Stage**	**In's**	**Out's**
AVLTree	add	setupEmpty	3, "red"	red
AVLTree	add getLeft getRight		1, "blue" 5, "black"	getLeft == "blue" getRight == "black"
AVLTree	add		1, "white"	1
AVLTree	add getBalanceFactor		4, "beige" 6, "pink" 7, "white"	getRoot.getBalanceFactor<2 == true

Goal of the test: Verify that the delete method deletes successfully a node of the AVL tree				
Class	**Method**	**Stage**	**In's**	**Out's**
AVLTree	delete search	setupPopulatedTree	id =5	search (id) == "brown" delete(id) == "brown"

AVLTree	delete search		id = 5	search(id) == null delete(id) == null
AVLTree	delete getBalanceFactor		5 7 6	root.getBalanceFactor<2

Goal of the test: Verify that if there are two nodes with same key, the delete with the parameters (id, value) deletes the node with the given value				
Class	**Method**	**Stage**	**In's**	**Out's**
AVLTree	search	setupPopulatedTree	id=1	search(id) == "blue"
AVLTree	add delete		1, "purple"	delete(1, "purple") search(1).size ==1

2.2 Binary Search Tree test.

Stage

Name	**Class**	**Stage**
setupEmpty	BinarySearchTreeTest	An empty binary search tree

setupPopulatedTree	BinarySearchTreeTest	A binary search tree with 7 nodes: 5, “brown”; 1,”blue”; 2,”black”; 3,”red”; 4,”beige”; 6,”pink”; 7,”white”

Goal of the test: Verify that the add method adds a node successfully into the AVL tree				
Class	**Method**	**Stage**	**In’s**	**Out’s**
BinarySearch Tree	add	setupEmpty	3, “red”	red
BinarySearch Tree	add getLeft getRight		1, “blue” 5, “black”	getLeft == “blue” getRight == “black”
BinarySearch Tree	add		1, “white”	1
BinarySearch Tree	search		1	search(1).size == 2

Goal of the test: Verify that the search method finds successfully the nodes to search				
Class	**Method**	**Stage**	**In's**	**Out's**
BinarySearch Tree	search	setupPopulatedTree	id =5	search(id) == "brown"
BinarySearch Tree	delete search	delete(id)	id	search(id) == null

Goal of the test: Verify that the delete method deletes successfully a node of the binary search tree				
Class	**Method**	**Stage**	**In's**	**Out's**
BinarySearch Tree	delete search	setupPopulatedTree	id =5	search (id) == "brown" delete(id) == "brown"
BinarySearch Tree	delete search		id = 5	search(id) == null delete(id) == null
BinarySearch Tree	add search		id = 1 "purple"	search(id) != null

BinarySearch Tree	delete		1	delete(1) should delete the node with value “blue”
BinarySearch Tree	search size		1	search(1) == “purple” search(1).size == 1

Goal of the test: Verify that if there are two nodes with same key, the delete with the parameters (id, value) deletes the node with the given value				
Class	**Method**	**Stage**	**In’s**	**Out’s**
BinarySearch Tree	search	setupPopulatedTree	id=1	search(id) == “blue”
BinarySearch Tree	add delete		1, “purple”	delete(1, “purple”) search(1).size ==1

Goal of the test: Verify that the method count counts the number of nodes correctly				
Class	**Method**	**Stage**	**In’s**	**Out’s**
BinarySearch Tree	count	setupEmpty		0

BinarySearchTree	count	setupPopulatedTree	5, “brown”; 1,”blue”; 2,”black”; 3,”red”; 4,”beige”; 6,”pink”; 7,”white”	14

2.3 Trie test

Stage:

Name	**Class**	**Stage**
setupEmpty	TrieTest	Empty trie
setupPopulatedTrie	TrieTest	An trie populated with 9 words: “black”, “block”, “blonde”, “wine”, “win”, “war”, “dislike”, “disliking”, “bourbon”

Test cases:

Goal of the test: Verify that the method adds words properly and with the correct multiplicities.				
Class	**Method**	**Stage**	**In’s**	**Out’s**
Trie	add	setupEmpty	“black”, “blonde”, “bourbon”	trie.getNode(“black”) is not null trie.getNode(“black”) is word trie.getNode(“bourbon”) multiplicities are 1, 2, 3

				after adding it 3 times (and testing after each time)

Goal of the test: Verify that the method removes words properly and subtract to the right multiplicities.				
Class	**Method**	**Stage**	**In's**	**Out's**
Trie	remove	setupPopulatedTrie	"disliking", "dislike", "bourbon"	trie.getNode("disliking") is null trie.getNode("dislike") is null size of trie.root. is 2 trie.getNode("bourbon") multiplicity is 0 after adding it 2 times and deleting it twice too.

Goal of the test: Verify that the method removes words properly and subtract to the right multiplicities.				
Class	**Method**	**Stage**	**In's**	**Out's**
Trie	getNode	setupPopulatedTrie		"black", "wine", "win" is word, is true "wi", "blo" is word is false

Goal of the test: Verify that the method removes words properly and subtract to the right multiplicities.				
Class	**Method**	**Stage**	**In's**	**Out's**
Trie	getSuggestionsTest	setupPopulatedTrie	{"black", "block", "blonde"} {"black", "block", "blonde", "bourbon"} {"wine"} {"war", "win", "wine"} {"dislike", "disliking"}	size(suggestions("bl")-list1) is zero size(suggestions("b")-list2) is zero size(suggestions("win")-list3) is zero size(suggestions("w")-list4) is zero size(suggestions("dis")-list5) is zero

Goal of the test: Verify that the method removes words properly and subtract to the right multiplicities.				
Class	**Method**	**Stage**	**In's**	**Out's**
Trie	countSuggestionsTest	setupPopulatedTrie	prefixes: "bl", "b", "win", "w", "dis", "z" "blanco", "bronce", "winnie", "zombie"	before adding, the suggestion count for the prefixes is, in order: 3, 4, 1, 3, 2, 0 after adding: 4, 6, 2, 4, 2, 1

Goal of the test: Verify that the method removes words properly and subtract to the right multiplicities.				
Class	**Method**	**Stage**	**In's**	**Out's**
Trie	countTotalSuggestionsTest	setupPopulatedTrie	"blanco", "bronce", "winnie", "zombie"	before adding, the total suggestion count is 9 after adding, 15

2.4 Model Test

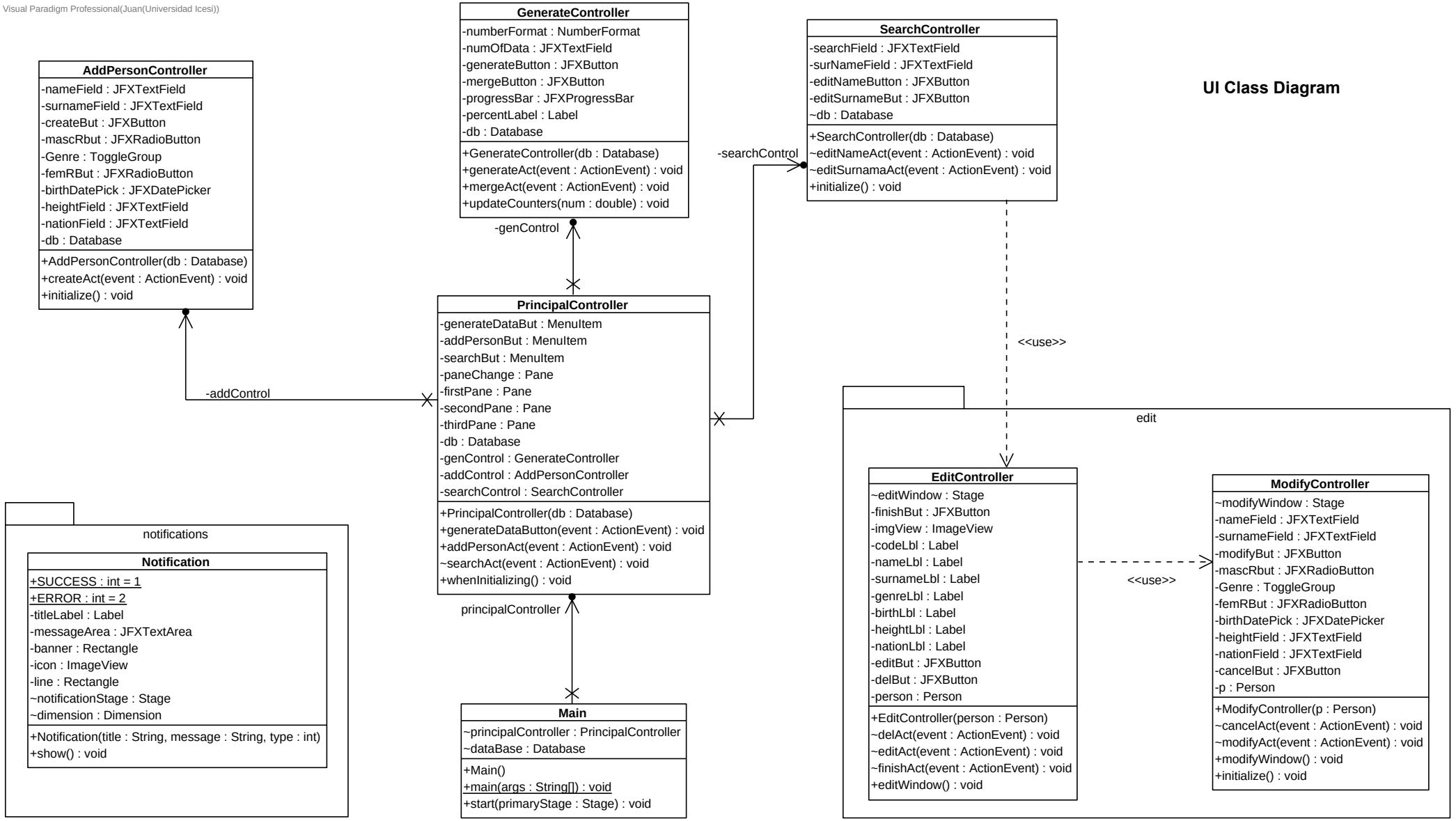
Name	**Class**	**Stage**
setupEmpty	DatabaseTest	An empty database
setupOnePerson	DatabaseTest	A database with one person: 1, "Test", "McTester", "Male", "2/02/20", 1.4, "Colombia"

Goal of the test: Test that the generate and the merge methods are working correctly				
Class	**Method**	**Stage**	**In's**	**Out's**
Database	generateTempList	setupEmpty	amount = 200	
Database	mergeTempList		amount	true
Database	count		amount	The counting of getting people by name, surname, code and the counting of the name and surname suggestions should equals amount

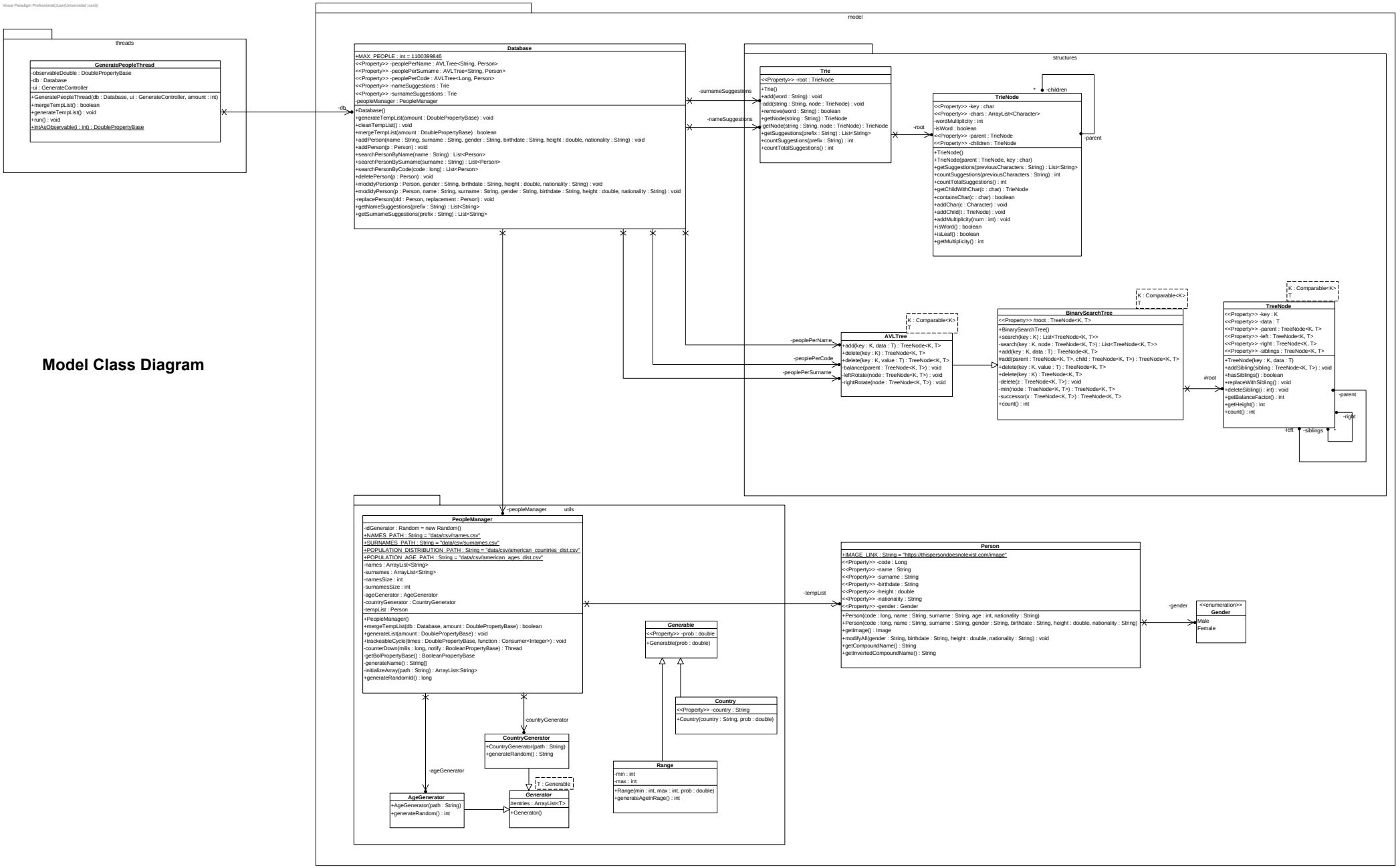
Goal of the test: Verify that the add method in database is working correctly				
Class	**Method**	**Stage**	**In's**	**Out's**
Database	add	setupEmpty	person : 1, "Test", "McTester", "Male", "2/02/20", 1.4, "Colombia"	When we search the person by the name, surname and code the result should be the person added

Goal of the test: Verify that the search method and the delete method are working correctly				
Class	**Method**	**Stage**	**In's**	**Out's**
Database	search	setupOnePerson	code =1 compound =”Test McTester” inverseCompound= “McTester Test”	searchByCode.size==1 searchPersonByCode.name== “Test” searchPersonByCode.surname== “McTester”
Database	delete		p(The person created in the stage)	searchByCode.size==0 searchPersonByCode== null

UI Class Diagram

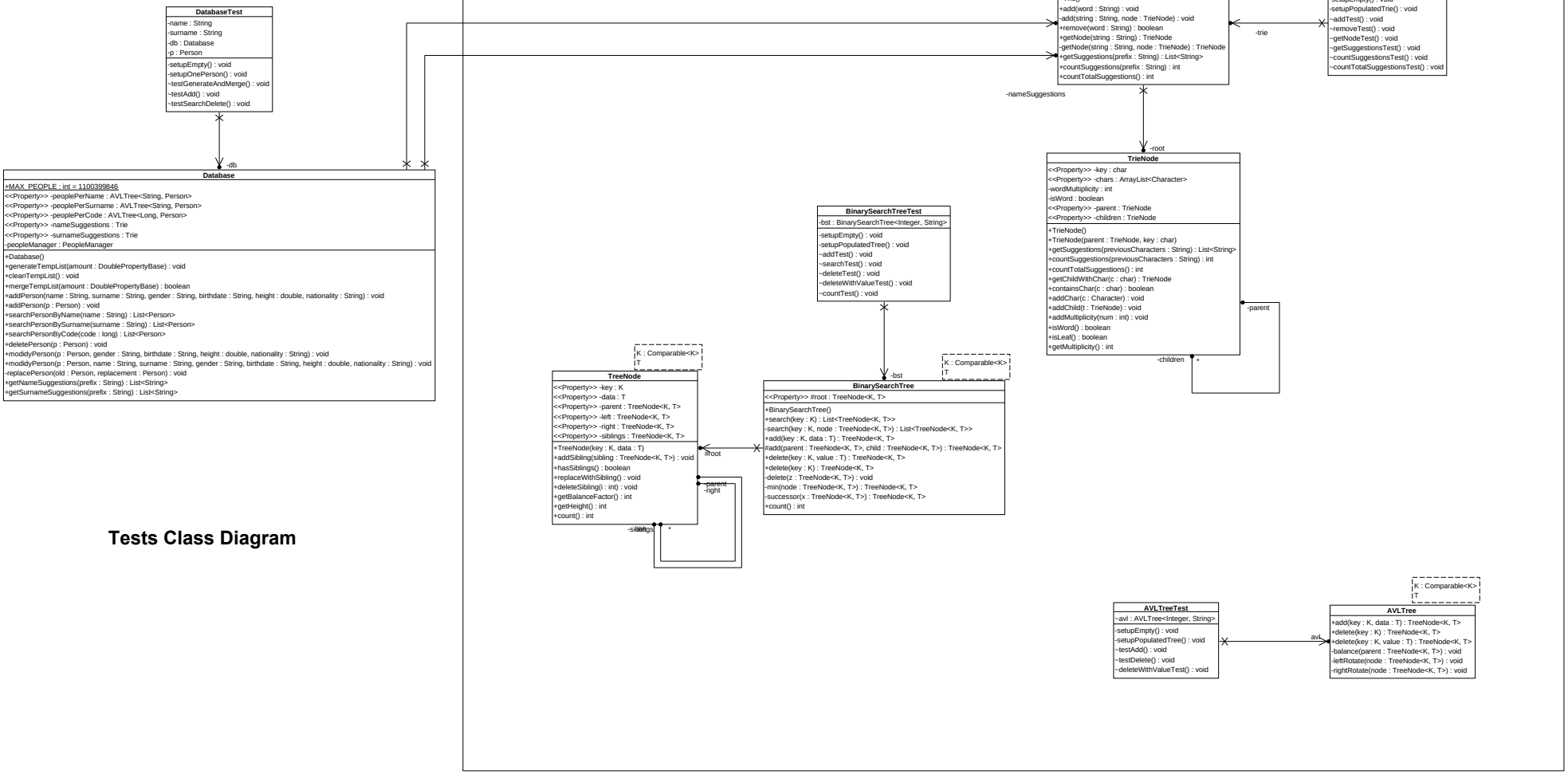


Model Class Diagram



model

structures



Tests Class Diagram