DOCUMENTACIÓN

LUISA FERNANDA QUINTERO FERNÁNDEZ

JUAN DAVID PELÁEZ VALENCIA

ALGORITMOS Y ESTRUCTURAS DE DATOS

UNIVERSIDAD ICESI

1. Análisis

Requerimientos funcionales

R1: Generar personas, el usuario puede escoger la cantidad de registros generados. Se generará máximo mil millones de personas con sus respectivos datos, los cuales son: código, nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía. No deben de haber códigos repetidos por lo cual estos se generarán automáticamente.

R2: Mostrar barra de progreso, se muestra una barra de progreso cuando la generación de los datos tarda más de un segundo, además debe de indicar cuanto tiempo tardo el proceso en generarse

R3: Guardar datos generados, el programa debe de guardar los datos que se generan para después poderlos consultar y utilizar. Los datos deben de ser persistentes.

R4: Actualizar persona, el programa puede actualizar todos los datos de la persona a excepción del código debido a que este es autogenerado y único para cada uno.

R5: Agregar persona, se agrega una nueva persona a la lista de datos con todos sus campos requeridos a excepción del código debido a que este es autogenerado.

R6: Eliminar persona, borra a una persona seleccionada de la base de datos.

R7: Buscar persona, se puede buscar a una persona por cualquiera de los siguientes criterios: Nombre, apellido, nombre completo y código. No se puede buscar por varios criterios al mismo tiempo, es decir, se debe de escoger por cuál de los 4 mencionados se va a realizar la búsqueda. Además, al lado del campo de texto se debe de mostrar un número que indica la cantidad de elementos que coinciden hasta ahora.

R8: Mostrar lista emergente, el programa debe de mostrar una lista emergente a medida que se va realizando la búsqueda de la persona, donde se mostraran máximo 100 nombres que empiecen con las letras ya digitadas en la búsqueda. Cuando queden 20 o menos nombres que coincidan con la búsqueda deberá aparecer una lista con los registros que coinciden y un botón con la opción de editar, en donde se podrá actualizar o eliminar dicho registro.

2. Diseño

2.1 Diagrama de clases

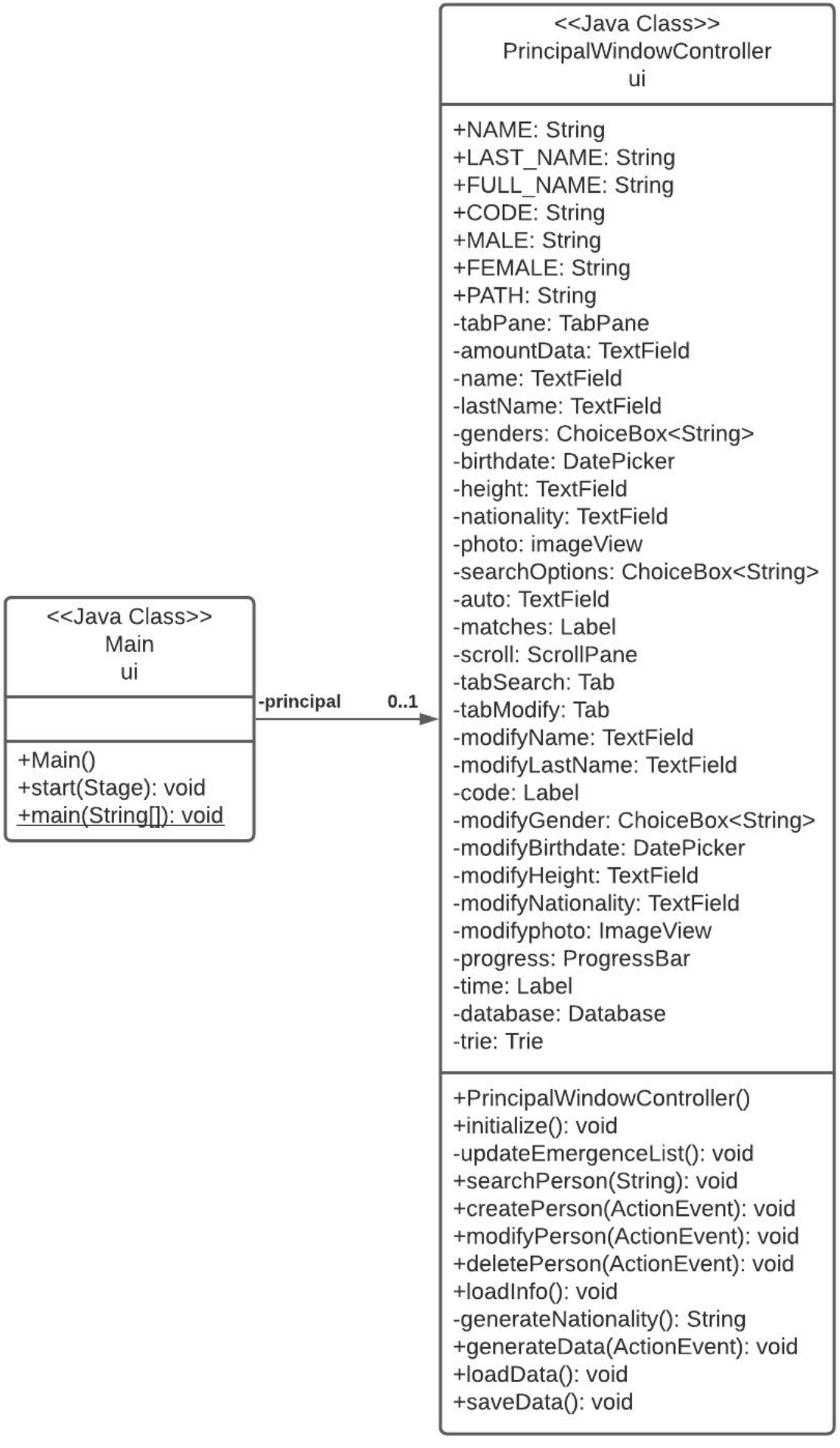
Modelo

Database
-nameAVLTree : AVLTree<String, Person> -lastNameAVLTree : AVLTree<String, Person> -fullNameRBTree : RedBlackBST<String, Person> -codeRBTree : RedBlackBST<String, Person>
+Database() +createPerson(code : String, name : String, lastName : String, gender : String, birthDate : LocalDate, height : double, nationality : . String) : void +createPerson(name : String, lastName : String, gender : String, birthDate : LocalDate, height : double, nationality : String) : void +deletePerson(name : String, lastName : String, code : String) : void +updatePerson(code : String, name : String, lastName : String, gender : String, birthDate : LocalDate, height : double, nationality : String) : void +searchByName(name : String) : Person +searchByLastName(lastName : String) : Person +searchByFullName(fullName : String) : Person +searchByCode(code : String) : Person +getPersonsByName() : List<Person> +getPersonsByLastName() : List<Person> +getPersonsbyFullName() : List<Person> +getPersonsbyCode() : List<Person>

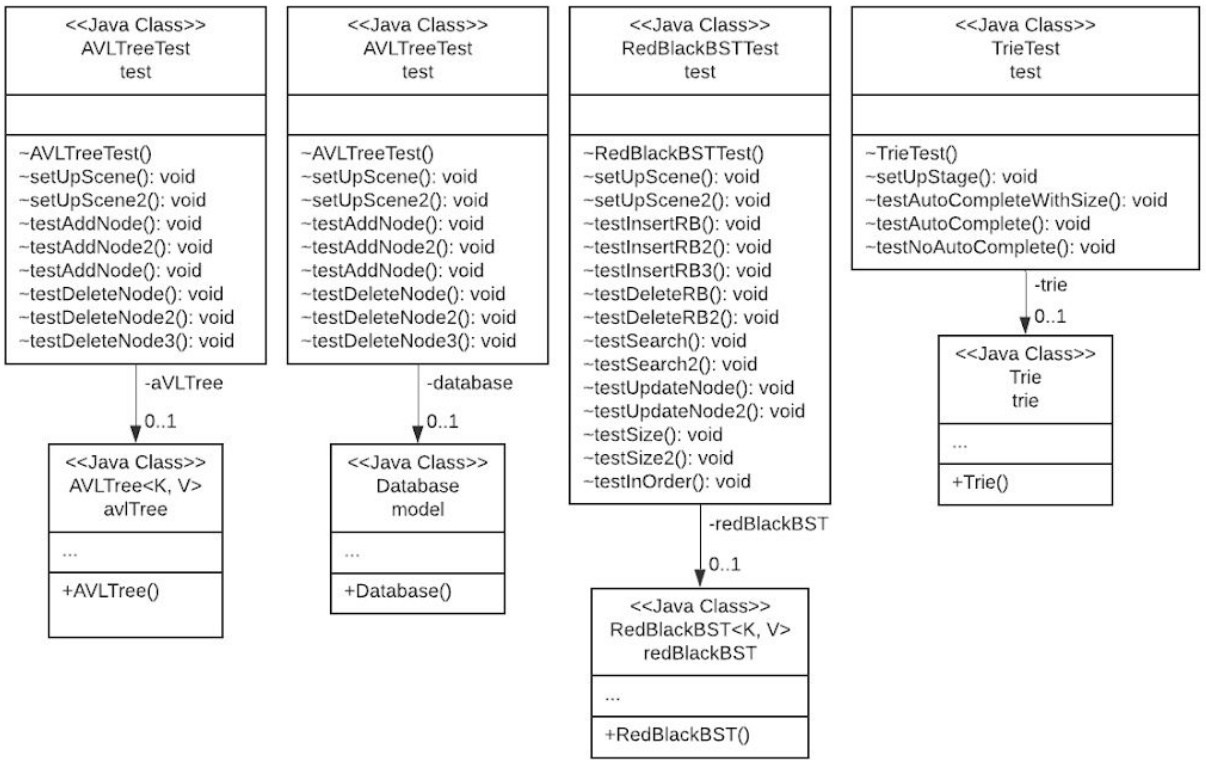
0..* -persons

Person
-code : String -name : String -lastName : String -gender : String -birthDate : LocalDate -height : Double -nationality : String
+Person(code : String, name : String, lastName, gender : String, birthDate : String, height : double, nationality : String) +getCode() : String +setCode(code : String) : void +getName() : String +setName(name : String) : void +getLastName() : String +setLastName(lastName : String) : void +getGender() : String +setGender(gender : String) : void +getBirthDate() : LocalDate +setBirthDate(birthDate : LocalDate) : void +getHeight() : Double +setHeight(height : Double) : void +getNationality() : String +setNationality(nationality : String) : void

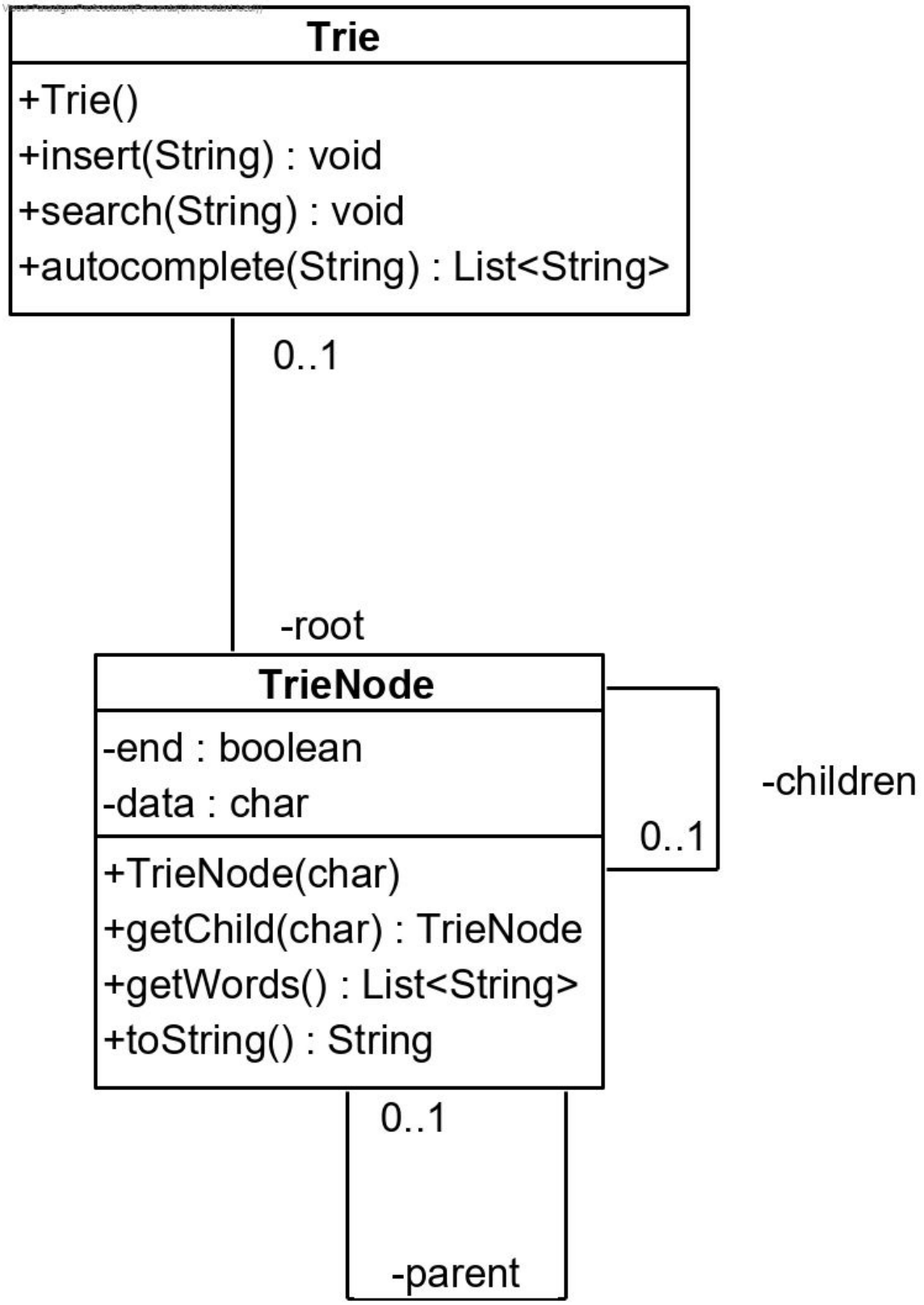
Interfaz


<<Java Class>>
PrincipalWindowController
ui
+NAME: String
+LAST_NAME: String
+FULL_NAME: String
+CODE: String
+MALE: String
+FEMALE: String
+PATH: String
-tabPane: TabPane
-amountData: TextField
-name: TextField
-lastName: TextField
-genders: ChoiceBox<String>
-birthdate: DatePicker
-height: TextField
-nationality: TextField
-photo: imageView
-searchOptions: ChoiceBox<String>
-auto: TextField
-matches: Label
-scroll: ScrollPane
-tabSearch: Tab
-tabModify: Tab
-modifyName: TextField
-modifyLastName: TextField
-code: Label
-modifyGender: ChoiceBox<String>
-modifyBirthdate: DatePicker
-modifyHeight: TextField
-modifyNationality: TextField
-modifyphoto: ImageView
-progress: ProgressBar
-time: Label
-database: Database
-trie: Trie
+PrincipalWindowController()
+initialize(): void
-updateEmergenceList(): void
+searchPerson(String): void
+createPerson(ActionEvent): void
+modifyPerson(ActionEvent): void
+deletePerson(ActionEvent): void
+loadInfo(): void
-generateNationality(): String
+generateData(ActionEvent): void
+loadData(): void
+saveData(): void
-principal
0..1
<<Java Class>>
Main
ui
+Main()
+start(Stage): void
+main(String[]): void

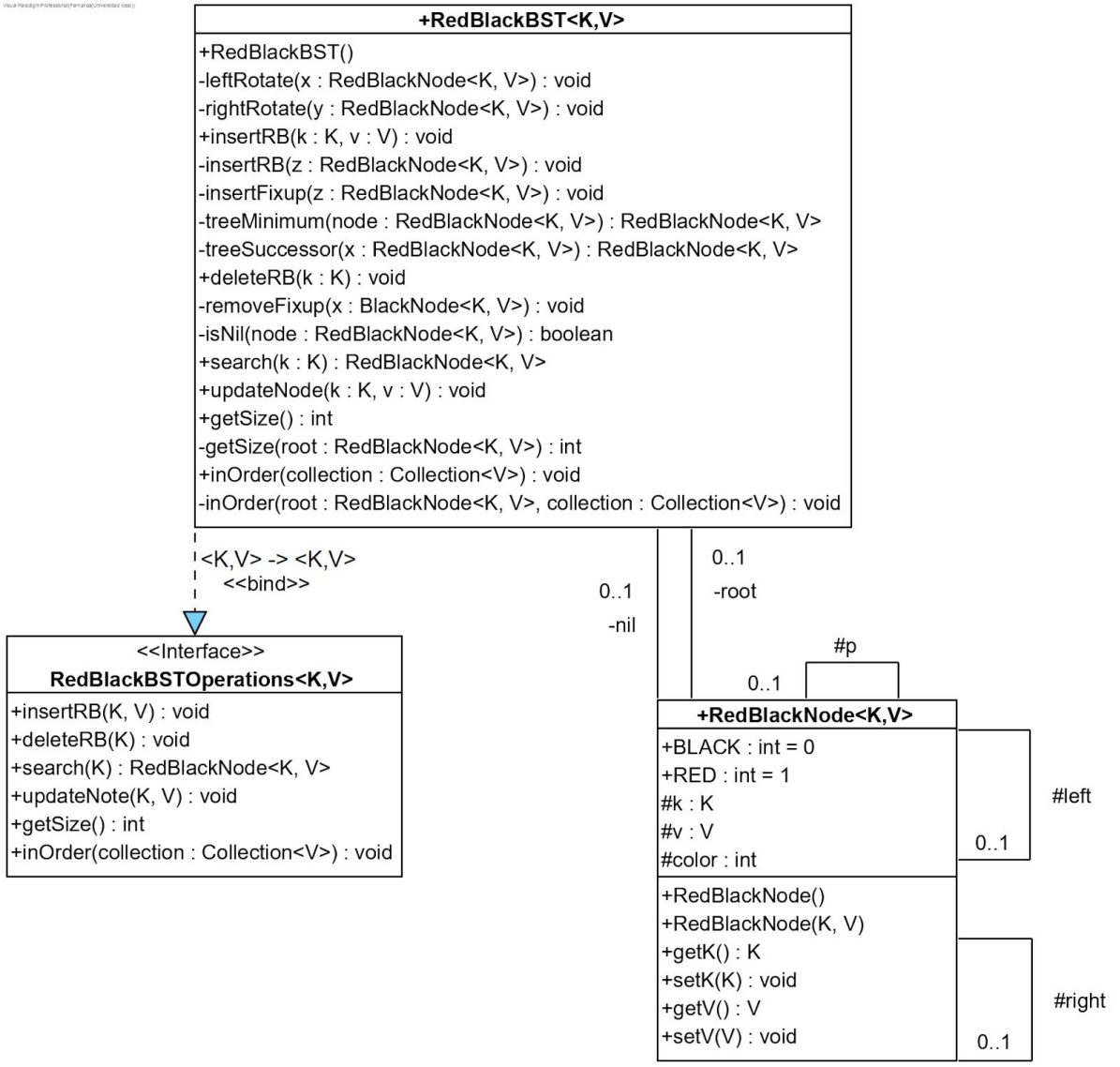
Casos de prueba



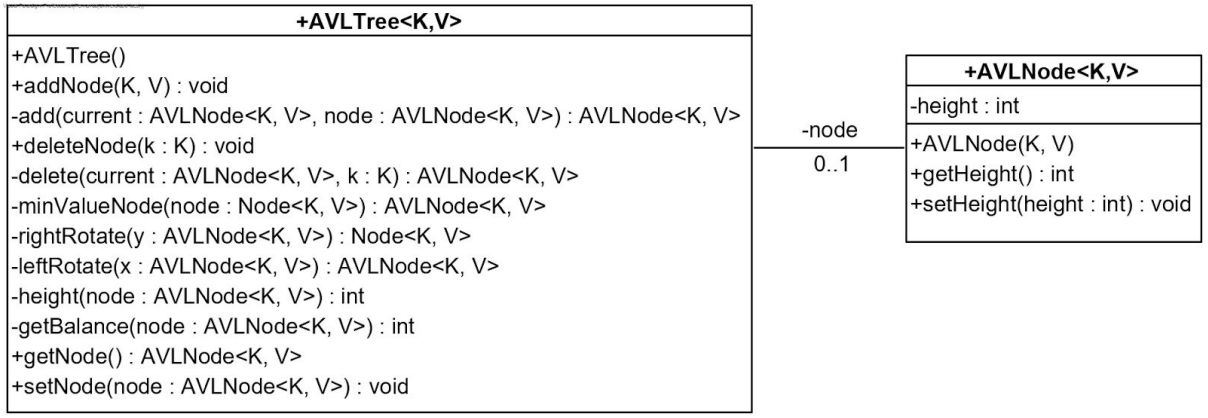
Trie



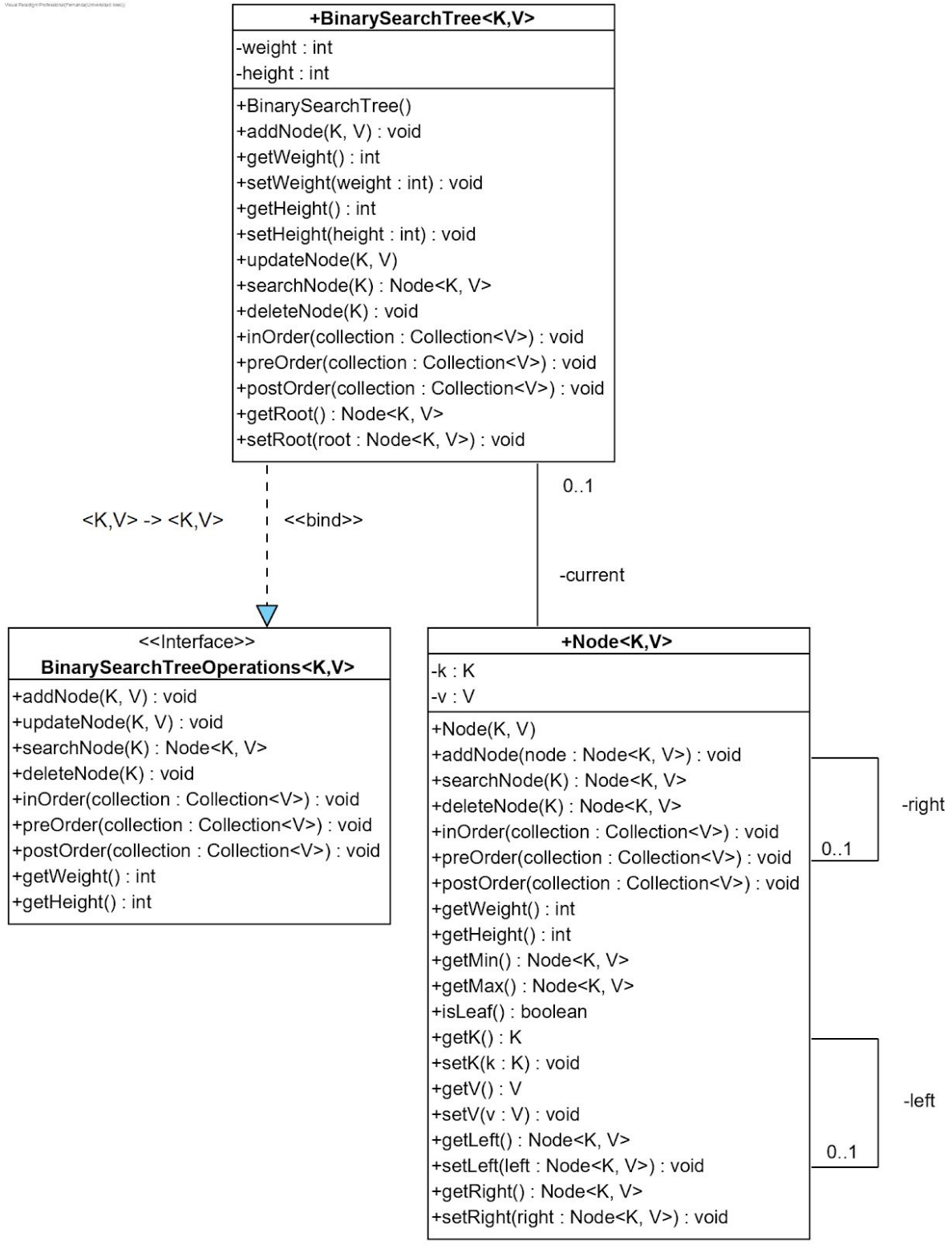
Árbol Rojinegro



Árbol AVL



Árbol binario de búsqueda



2.2 Diseño de casos de pruebas Unitarias

Configuración de los Escenarios de RedBlackTree:

Nombre	Clase	Escenario

setUpScene	RedBlack BSTTest	Se instancia únicamente el árbol rojinegro

Nombre	Clase	Escenario
setUpScene2	RedBlack BSTTest	redBlackBST = new RedBlackBST<Integer, String>() redBlackBST.insertRB(2, "a") redBlackBST.insertRB(3, "b") redBlackBST.insertRB(1, "c") redBlackBST.insertRB(5, "d") redBlackBST.insertRB(4, "e") redBlackBST.insertRB(6, "f")

Diseño de Casos de Prueba:

Objetivo de la Prueba:. El programa es capaz de agregrar un elemento cuando el arbol rojinegro esta vacio				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	insertRB	setUpScene	redBlackBST.insertRB(1, "3")	1

Objetivo de la Prueba:. El programa es capaz de agregrar un elemento cuando el arbol rojinegro no esta vacio				
Clase	Método	Escenario	Valores de Entrada	Resultado

RedBlackBST	insertRB	setUpScene2	redBlackBST.insertRB(7, "g");	7

Objetivo de la Prueba:. El programa es capaz de agregrar un elemento cuando el arbol rojinegro no esta vacio o de atrapar cualquier excepción que pueda producirse				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	insertRB	setUpScene2	redBlackBST.insertRB(3, "d");	IllegalArgumentException

Objetivo de la Prueba:. El programa es capaz de eliminar un elemento dentro del arbol rojinegro				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	deleteRB	setUpScene2	redBlackBST.deleteRB(3);	null

Objetivo de la Prueba:. El programa es capaz de eliminar un elemento dentro del arbol rojinegro y si no lo encuentra lanza una excepción				
Clase	Método	Escenario	Valores de Entrada	Resultado

RedBlackBST	deleteRB	setUpScene	redBlackBST.deleteRB(4);	NullPointerException

Objetivo de la Prueba:. El programa es capaz de buscar un elemento dentro del árbol rojinegro y mostrarlo				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	search	setUpScene2	redBlackBST.search(1).getK()	1

Objetivo de la Prueba:. El programa es capaz de actualizar un elemento dentro del árbol rojinegro				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	updateNode	setUpScene2	redBlackBST.updateNode(1, "a");	null

Objetivo de la Prueba:. El programa es capaz de actualizar un elemento dentro del árbol rojinegro y si no hay nadie para editar o no existe la persona para modificar, retorna nulo				
Clase	Método	Escenario	Valores de Entrada	Resultado

RedBlackBST	updateNode	setUpScene	redBlackBST.updateNode(1, "a");	NullPointerException

Objetivo de la Prueba:. El programa es capaz de dar el tamaño del arbol rojinegro cuando tiene elementos				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	getSize	setUpScene2	redBlackBST.getSize()	6

Objetivo de la Prueba:. El programa es capaz de dar el tamaño del arbol rojinegro cuando no tiene elementos				
Clase	Método	Escenario	Valores de Entrada	Resultado
RedBlackBST	getSize	setUpScene	redBlackBST.getSize()	1

Objetivo de la Prueba:. El programa es capaz de ordenar en inorder el arbol rojinegro				
Clase	Método	Escenario	Valores de Entrada	Resultado

RedBlackBST	inOrder	setUpScene2	ArrayList<String> list = new ArrayList<>(); redBlackBST.inOrder(list);	String[] expected = {"c", "a", "b", "e", "d", "f"};

Configuración de los Escenarios de TRIE:

Nombre	Clase	Escenario

setUpStage1	TrieTest	trie = new Trie();
		trie.insert("Edgar Allan Poe");
		trie.insert("James Barrie");
		trie.insert("Emily Bronte");
		trie.insert("Euripides");
		trie.insert("Ernest Hemingway");
		trie.insert("Arthur Conan Doyle");
		trie.insert("Lewis Carroll");
		trie.insert("JRR Tolkien");
		trie.insert("Elvira Sastre");
		trie.insert("Alejandra Pizarnik");

Diseño de Casos de Prueba:

Objetivo de la Prueba:. El programa es capaz de autocompletar con la letra ingresada, y mostrar la cantidad y/o tamaño de las sugerencias posibles				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	size	setUpStage1	List<String> data= trie.autocomplete("E");	5

Objetivo de la Prueba:. El programa es capaz de autocompletar y mostrar las sugerencias disponibles de acuerdo a la letra ingresada				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	autocomplete	setUpStage1	List<String> data= trie.autocomplete("E");	Edgar Allan Poe Emily Bronte Euripides Ernest Hemingway Elvira Sastre

Objetivo de la Prueba:. El programa es capaz de autocompletar y no mostrar ninguna de no haber ninguna letra que comience con la ingresada				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	size	setUpStage1	List<String> data= trie.autocomplete("Y");	0

Configuración de los Escenarios de AVLTree:

Nombre	Clase	Escenario
setUpScene	AVLTreeTest	Se instancia únicamente el árbol AVL

Nombre	Clase	Escenario

setUpScene2	AVLTreeTest	aVLTree = new AVLTree<Integer, Integer>();
		aVLTree.addNode(9, 1);
		aVLTree.addNode(5, 2);
		aVLTree.addNode(10, 3);
		aVLTree.addNode(0, 4);
		aVLTree.addNode(6, 5);
		aVLTree.addNode(11, 6);
		aVLTree.addNode(-1, 7);
		aVLTree.addNode(1, 8);
		aVLTree.addNode(2, 9);

Diseño de Casos de Prueba:

Objetivo de la Prueba: El programa es capaz de agregar un elemento cuando el arbol AVL está vacío				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	addNode	setUpScene	aVLTree.addNode (1, 1);	1

Objetivo de la Prueba:. El programa es capaz de agregar un elemento cuando el árbol AVLno está vacío				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	addNode	setUpScene2	aVLTree.addNode (7, 10);	7

Objetivo de la Prueba:. El programa es capaz de agregar un elemento cuando el árbol AVL contiene elementos previamente o, atrapar cualquier excepción que pueda producirse				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	addNode	aVLTree.addNode(3, 10);	redBlackBST.insertRB(3, "d");	IllegalArgumentException

Objetivo de la Prueba:. El programa es capaz de eliminar un elemento dentro del árbol AVL				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	deleteNode	setUpScene2	aVLTree.deleteNode(3);	null

Objetivo de la Prueba:. El programa es capaz de eliminar un elemento dentro del árbol AVL y balancearse				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	deleteNode	setUpScene2	aVLTree.deleteNode(10);	1

Objetivo de la Prueba:. El programa es capaz de eliminar un elemento dentro del árbol AVL y si no lo encuentra lanza una excepción

Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	deleteNode	setUpScene	aVLTree.deleteNode(4);	NullPointerException

Configuración de los Escenarios de Database:

Nombre	Clase	Escenario
setUpStage1	DatabaseTest	database= new Database(); database.createPerson("Fernanda", "Rojas", "Colombian");

Nombre	Clase	Escenario
setUpStage2	DatabaseTest	database= new Database(); database.createPerson("Amanda", "Rojas", "Female", LocalDate.now(), 1.62, "Colombian");

Nombre	Clase	Escenario
setUpStage3	DatabaseTest	database= new Database();

Diseño de Casos de Prueba:

Objetivo de la Prueba:. El programa es capaz de buscar por nombre completo en la base de datos				
Clase	Método	Escenario	Valores de Entrada	Resultado
Database	searchByFullName	setUpStage1		“Fernanda”

Objetivo de la Prueba:. El programa es capaz de buscar por el nombre en la base de datos				
Clase	Método	Escenario	Valores de Entrada	Resultado
Database	searchByName	setUpStage1		“Fernanda”

Objetivo de la Prueba:. El programa es capaz de buscar por el código en la base de datos				
Clase	Método	Escenario	Valores de Entrada	Resultado
Database	searchByName	setUpStage2		“Rojas”

Objetivo de la Prueba:. El programa es capaz de eliminar a una persona dentro de la base de datos

Clase	Método	Escenario	Valores de Entrada	Resultado
Database	deletePerson	setUpStage2	database.deletePerson("Carlos", "Perez", "abcde");	null