Miguel Andrés Sarasti – A00364978

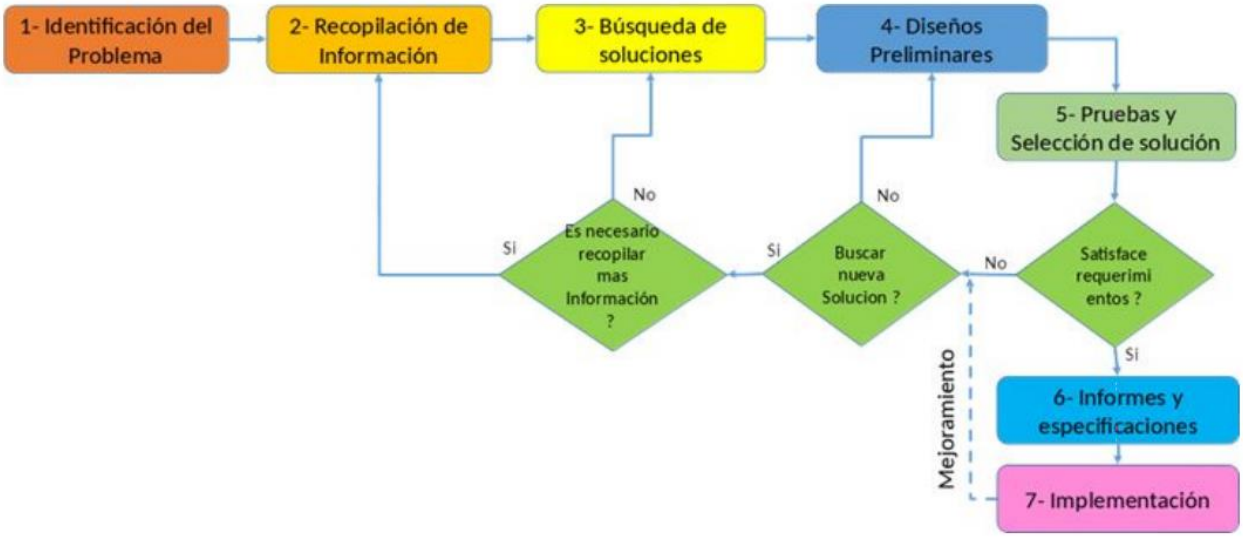
Método de la Ingeniería

Contexto problemático

El Equipo VIP de Simulación de la Universidad Icesi asignó un subproyecto que consiste en el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD (Crear, Leer, Modificar/Actualizar, Borrar) sobre una base de datos de personas del continente americano.

Desarrollo de la Solución

Seguiremos estos pasos mostrados en el siguiente diagrama de flujo para llegar al desarrollo de la solución:



Paso 1: Identificación del problema

Identificación de necesidades y síntomas

1. El proyecto necesita un software capaz de realizar las operaciones CRUD, con capacidad máxima de mil millones de personas.
2. Se busca que la generación de nombres completos sea por medio de dos dataset.
3. La fecha de nacimiento debe seguir una distribución de edad para toda América.
4. La estatura debe ser generada aleatoriamente en un intervalo lógico.
5. La nacionalidad debe mantener unos porcentajes relativos de la población de cada país respecto al continente.
6. El programa debe tener una barra de progreso e indicar cuanto tiempo se demoró la operación.
7. El programa debe guardar los datos, los datos deben ser persistentes.

Definición del Problema

El Equipo VIP requiere un programa capaz de realizar las operaciones CRUD sobre una base de datos de personas del continente americano, con una capacidad máxima de mil millones de personas.

Requerimientos funcionales

- R1: Generar personas, el usuario es capaz de elegir la cantidad de personas a generar. Se generará como máximo mil millones de personas, cuyos datos almacenados son: código (autogenerado), nombre, apellido, género, fecha de nacimiento, altura y nacionalidad.
- R2: Mostrar barra de progreso y tiempo. Se muestra una barra de progreso y tiempo empleado, vinculada al tiempo que se demora en generar las personas.
- R3: Agregar persona. Se agrega una nueva persona, manualmente, a la base de datos, los datos solicitados son: nombre, apellido, género, fecha de nacimiento, altura y nacionalidad.
- R4: Actualizar persona. Se puede actualizar cualquier dato, a excepción del código, de una persona existente en la base de datos.
- R5: Eliminar persona. Se puede eliminar a una persona existente de la base de datos.
- R6: Buscar persona. Se puede buscar una persona por medio del *nombre, apellido, nombre completo (nombre y apellido)* y *código*. Solo se puede realizar la búsqueda cuando escoge por que criterio buscar (un criterio al tiempo).
- R7: Guardar y cargar datos. Se pueden guardar y cargar los datos generados y/o agregados a la base de datos. Los datos son persistentes.
- R8: Mostrar lista emergente. Se muestra una lista emergente a medida que se van realizando las búsquedas de personas, el máximo de personas que se mostrarán son 100 y cuando queden 20 o menos personas luego de realizar la búsqueda, el programa mostrará un botón por cada persona para editarla (Actualizar datos o eliminar a la persona).

Paso 2: Recopilación de la información

Definiciones

CRUD: son las cuatro funciones básicas del almacenamiento persistente, el acrónimo hace referencia a las funciones *crear*, *leer*, *modificar/actualizar* y *eliminar/borrar*.

Autocompletar con Javafx: hay diferentes maneras de autocompletar información con Javafx, entre esas está el método Textfields.bindAutoCompletion(), el cual recibe el texto ingresado y una lista donde está la información de las palabras sugeribles.

Árbol AVL: es una estructura de datos que se basa en el árbol binario de búsqueda, pero todo nodo cumple con una propiedad de equilibrio AVL (Las alturas de los subárboles izquierdo y derecho no se diferencian en más de uno).

- Factor de balanceo: altura (subárbol derecho) – altura (subárbol izquierdo)
- El árbol AVL debe mantener el factor de balanceo entre -1 y 1.

TRIE: es una estructura de datos con forma similar a un árbol n-ario, en donde los nodos almacenan letras del alfabeto, que terminan formando una palabra al recorrer una rama por completo, y su raíz es un nodo vacío, que contiene la referencia a todos los nodos del trie.

Leer un textField mientras cambia su texto:

```
TextField.textProperty().addListener(new ChangeListener<String>()
   {
    @Override
    public void changed(ObservableValue<? extends String> observableValue, String oldValue, String newValue) {
        System.out.println(oldValue+“ changed to ”+newValue);
        }
});
```

Fuentes:

https://stackoverflow.com/questions/36861056/javafx-textfield-auto-suggestions

https://www.geeksforgeeks.org/trie-insert-and-search/

https://stackoverrun.com/es/q/12562762

https://gist.github.com/floralvikings/10290131

Paso 3: Búsqueda de soluciones creativas
Posibles ideas:

1. Utilizar un Trie para solucionar el autocompletar.
2. Utilizar un árbol AVL para solucionar el autocompletar.
3. Colocar la lista emergente en un ScrollPane.
4. Colocar un task para la barra de progreso y que actualice al mismo tiempo que la generación de los datos.
5. La generación de datos puede ser por medio de un archivo de texto con la cantidad máxima de nombres.

Paso 4: Transición de las Ideas a los Diseños Preliminares
Para la funcionalidad de autocompletar, se opta por utilizar la estructura Trie, ya que es más fácil y menos costoso encontrar todas las posibles combinaciones de palabras (nombre, apellido, nombre completo, código) de acuerdo con una cadena de texto inicial.

Complejidad temporal y espacial de los algoritmos y estructuras

Árbol AVL: en el peor de los casos su complejidad temporal es O(log n) y su complejidad espacial es O(n).

Trie: en el peor de los casos su complejidad temporal es O(n*m), siendo m la longitud de la palabra y n el número de nodos que contenga, y su complejidad espacial es O(n).

Paso 5: Evaluación y Selección de la Mejor Solución
Criterio A. Conocimiento. La alternativa ha sido estudiada:

- [2] Sí (se prefiere una solución exacta)
- [1] No

Criterio B. Eficiencia temporal. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

- [4] Constante.
- [3] Mayor a constante.
- [2] Logarítmica.
- [1] Lineal.

Criterio C. Eficiencia espacial. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

- [4] Constante
- [3] Mayor a constante.
- [2] Logarítmica.
- [1] Lineal.

Criterio D. Facilidad en implementación algorítmica:

- [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno
- [1] No compatible completamente con las operaciones aritméticas básicas de un equipo de cómputo moderno.

Estructura	A	B	C	D	Total
Árbol AVL	Sí	Logarítmica	Lineal	Compatible	7
Trie	Sí	Lineal	Lineal	Compatible	6

Paso 6: Preparación de informes y especificaciones

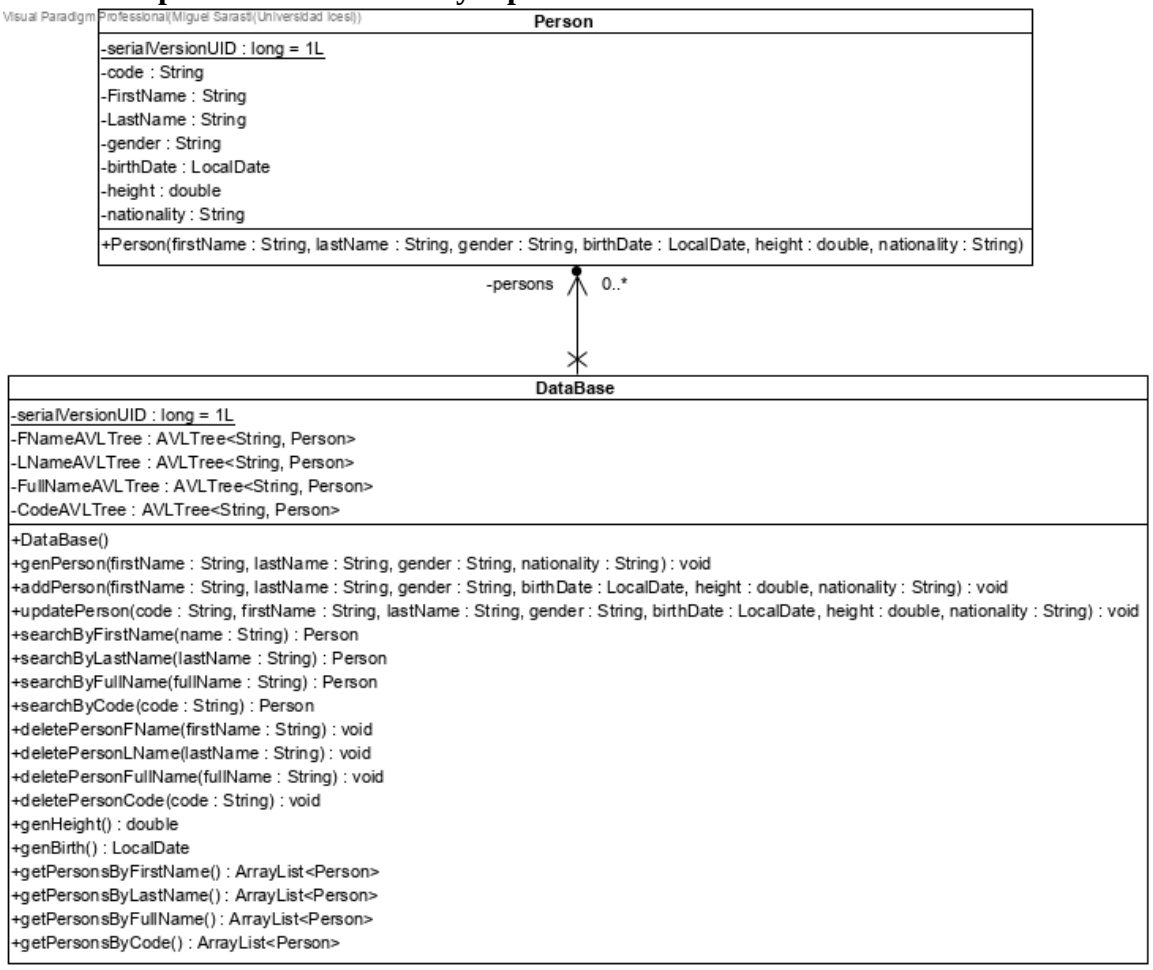


Ilustración 1 Diagrama de clases del modelo

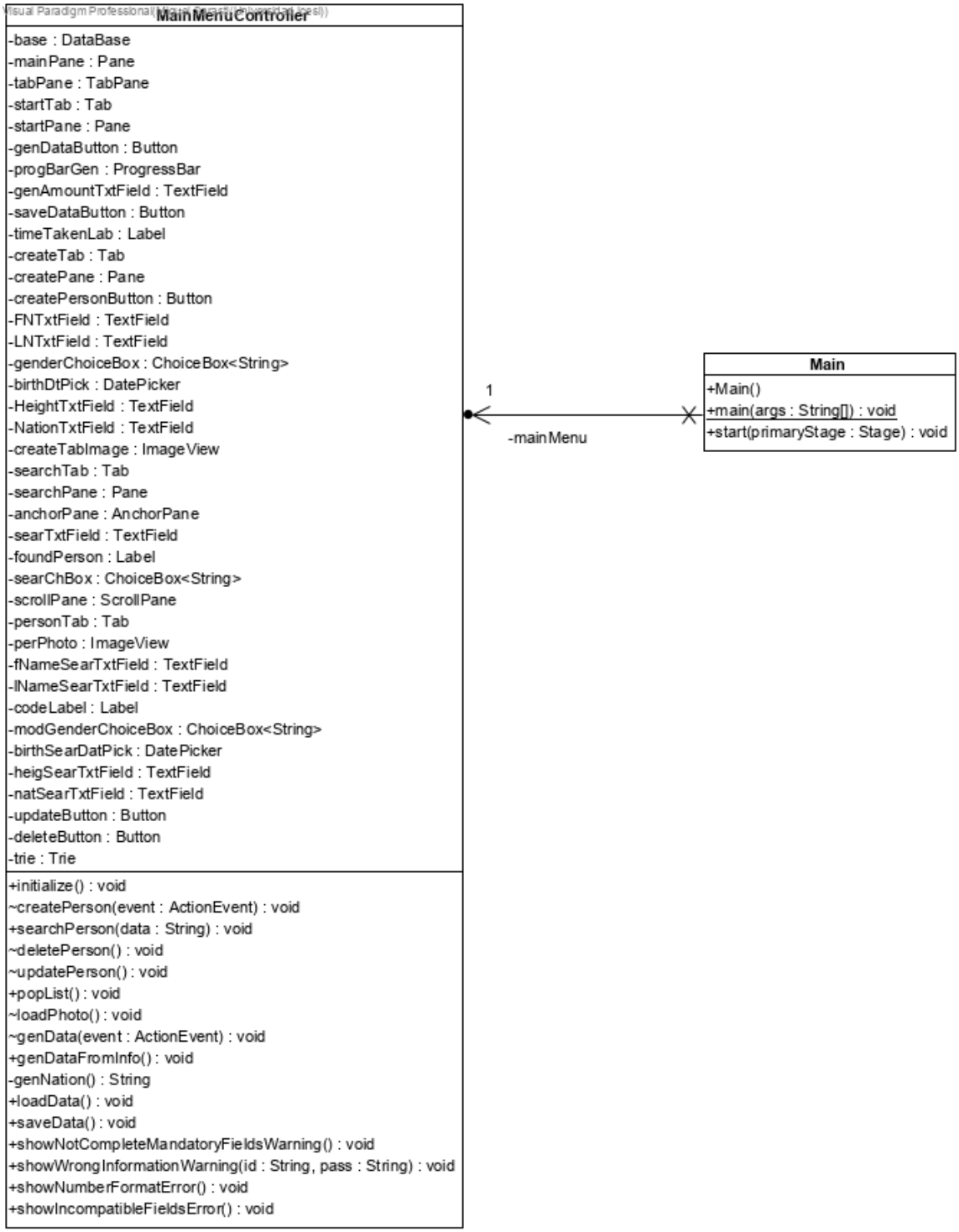


Ilustración 2 Diagrama de clases de la interfaz de usuario

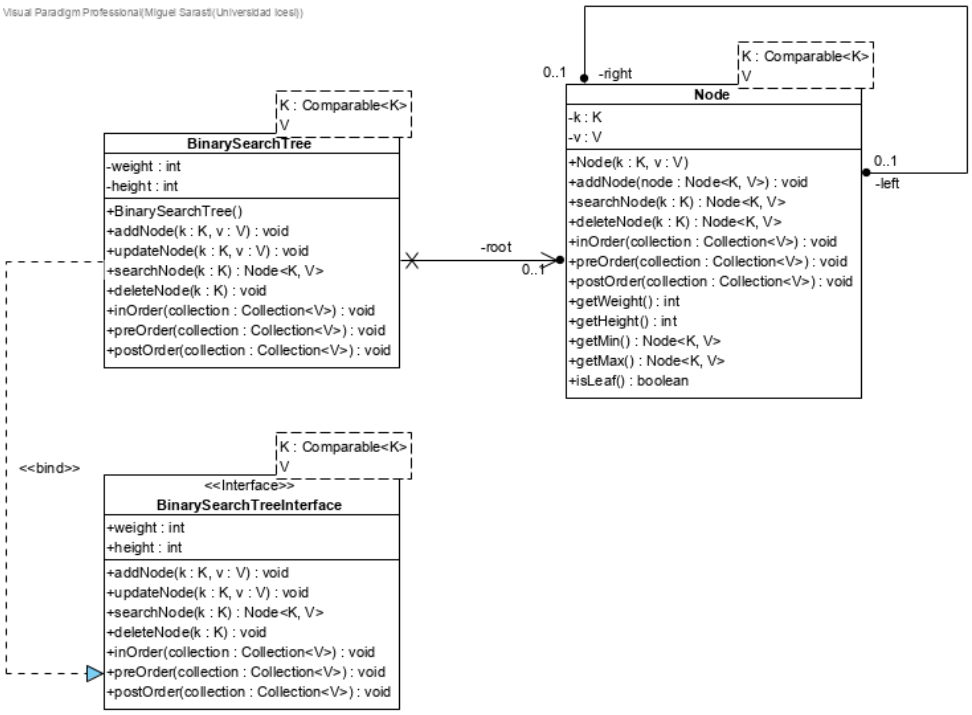


Ilustración 3 Diagrama de clases del ABB

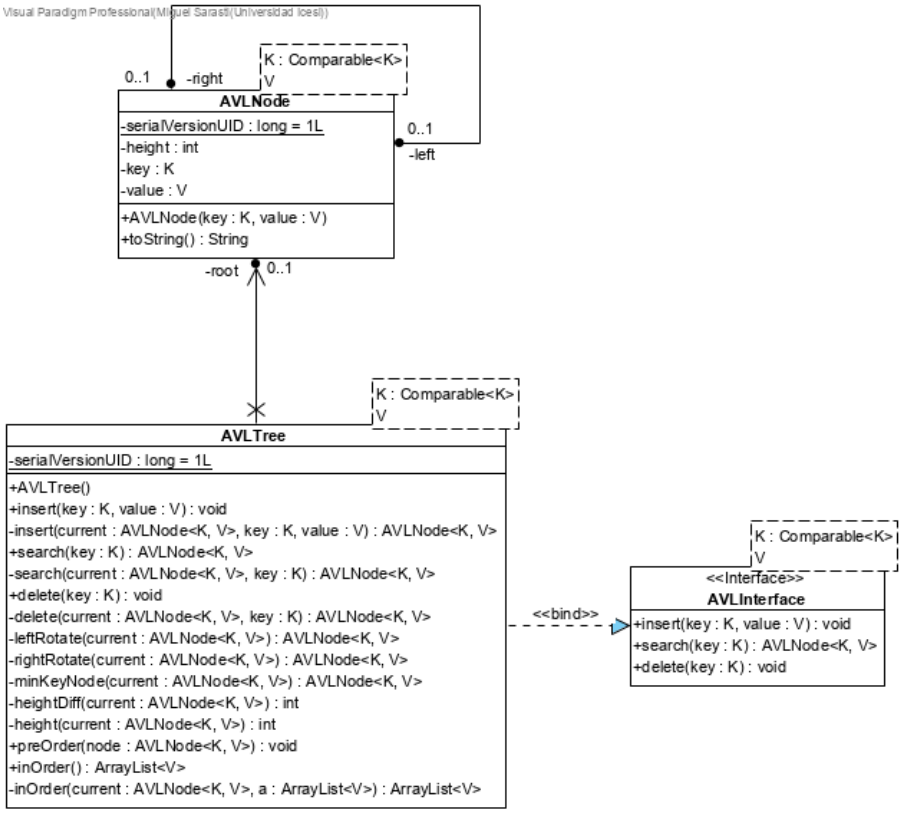


Ilustración 4 Diagrama de clases del árbol AVL

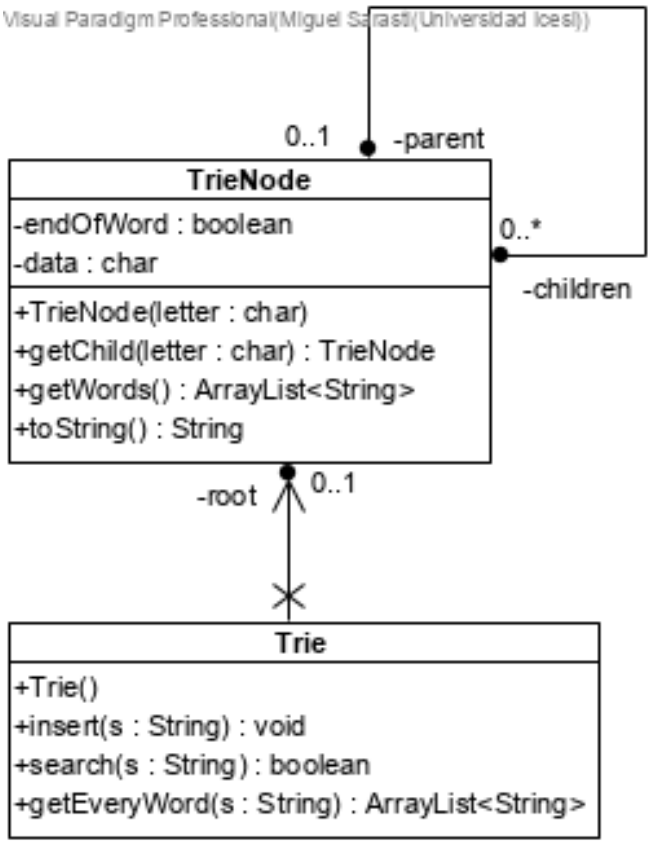


Ilustración 5 Diagrama de clases del Trie

Paso 7: Implementación del diseño
Implementando en el lenguaje de programación Java.

Tareas por implementar:

1. Generar personas.
2. Mostrar barra de progreso y tiempo.
3. Agregar persona.
4. Actualizar persona.
5. Eliminar persona.
6. Buscar persona.
7. Guardar y Cargar datos.
8. Mostrar lista emergente.

Pruebas Unitarias

Árbol binario de búsqueda

Configuración de los Escenarios de BinarySearchTree:

Nombre	Clase	Escenario
setUpStage1	BSTTest	b = new BinarySearchTree<Integer, String>()

Nombre	Clase	Escenario
setUpStage2	BSTTest	b = new BinarySearchTree<Integer, String>() b.addNode(5, "Michael") b.addNode(10, "Leonard") b.addNode(7, "Raphael") b.addNode(2, "Donatello") b.addNode(1, "Splinter")

Diseño de Casos de Prueba:

Objetivo de la Prueba: Comprobar que el programa agregue nuevos elementos al BST				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.addNode()	setUpStage1	b.addNode(2, “Kirito”)	Kirito

Objetivo de la Prueba: Comprobar que el programa no agregue elementos al BST con una llave ya existente				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.addNode()	setUpStage1	b.addNode(2, "Kirito") b.addNode(1, "Asuna") b.addNode(6, "Sinon") b.addNode(1, “Alice”)	Exception

Objetivo de la Prueba: Comprobar que el programa permita modificar elementos existentes del BST				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.updateNode()	setUpStage2	b.updateNode(1, “Kirito”)	Kirito

Objetivo de la Prueba: Comprobar que el programa busque un elemento por medio de una llave en el BST.				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.searchNode()	setUpStage2		Michael

Objetivo de la Prueba: Comprobar que el programa elimine elementos correctamente del BST				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.deleteNode()	setUpStage2	b.deleteNode(2)	True

Objetivo de la Prueba: Comprobar que el programa calcule el peso del BST				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.getWeight()	setUpStage2		5

Objetivo de la Prueba: Comprobar que el programa calcule la altura del BST				
Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTree	b.getHeight()	setUpStage2		3

Árbol AVL

Configuración de los Escenarios de AVL Tree:

Nombre	Clase	Escenario
setUpStage1	AVLTTest	tree = new AVLTree<Integer, String>()

Nombre	Clase	Escenario
setUpStage2	AVLTTest	tree = new AVLTree<Integer, String>() tree.insert(10, "Hi") tree.insert(20, "Hello") tree.insert(30, "Nein") tree.insert(40, "Luck") tree.insert(50, "Be") tree.insert(25, "Die")

Diseño de Casos de Prueba:

Objetivo de la Prueba: Comprobar que el programa agregue nuevos elementos al árbol AVL

Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.insert()	setUpStage1	tree.insert(2, "Kirito")	Kirito

Objetivo de la Prueba: Comprobar que el programa agregue elementos a un árbol AVL existente				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.insert()	setUpStage2	tree.insert (2, "Kirito") tree.insert (1, "Asuna") tree.insert (6, "Sinon")	Nein

Objetivo de la Prueba: Comprobar que el programa busque un elemento por medio de una llave en un árbol AVL existente				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.search()	setUpStage2		Hello

Objetivo de la Prueba: Comprobar que el programa busque un elemento agregado por medio de una llave en el árbol AVL.				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.insert() y tree.search()	setUpStage2	tree.insert(15, “Kirito”)	Kirito

Objetivo de la Prueba: Comprobar que el programa elimine elementos correctamente del árbol AVL				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.delete()	setUpStage2	tree.delete(25)	True

Objetivo de la Prueba: Comprobar que el programa elimine elementos correctamente del árbol AVL				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.delete()	setUpStage2	tree.delete(50)	True

Objetivo de la Prueba: Comprobar que el programa autobalancee correctamente al árbol AVL				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.insert() y tree.getRoot()	setUpStage1	tree.insert(1, "Kirito") tree.insert(2, "Asuna") tree.insert(3, "Sinon") tree.insert(10, "Eugeo");	Asuna

Objetivo de la Prueba: Comprobar que el programa autobalancee correctamente al árbol AVL				
Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTree	tree.insert() y tree.getRoot()	setUpStage2	tree.insert(1, "Kirito") tree.insert(2, "Asuna") tree.insert(3, "Sinon") tree.insert(15, "Eugeo");	Nein

Trie

Configuración de los Escenarios del Trie:

Nombre	Clase	Escenario
setUpStage1	TrieTest	t = new Trie()

Nombre	Clase	Escenario
setUpStage2	TrieTest	t = new Trie() t.insert("hola") t.insert("hello") t.insert("game") t.insert("goma") t.insert("kirito") t.insert("kisune")

Diseño de Casos de Prueba:

Objetivo de la Prueba: Comprobar que el programa agregue nuevos elementos al Trie

Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.insert()	setUpStage1	t.insert(“ginger”)	True

Objetivo de la Prueba: Comprobar que el programa agregue elementos a un Trie existente				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.insert()	setUpStage2	t.insert(“ginger”)	True True

Objetivo de la Prueba: Comprobar que el programa agregue elementos a un Trie existente y verificar la cantidad de palabras con una letra específica				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.insert() t.getEveryWord()	setUpStage2	t.insert(“ginger”) t.getEveryWord(“g”)	3

Objetivo de la Prueba: Comprobar que el programa busque un elemento en el Trie				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.search()	setUpStage1	t.search(“hola”)	False

Objetivo de la Prueba: Comprobar que el programa busque un elemento existente en el Trie				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.search()	setUpStage2	t.search(“hola”)	True

Objetivo de la Prueba: Comprobar que el programa busque un elemento agregado en el Trie				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.insert() y t.search()	setUpStage2	t.insert(“Sho”) t.search(“Sho”)	True

Objetivo de la Prueba: Comprobar que el programa retorne todas las posibles palabras formadas por una cadena de texto inicial				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.getEveryWord()	setUpStage1	t.getEveryWord(“a”)	0

Objetivo de la Prueba: Comprobar que el programa retorne todas las posibles palabras formadas por una cadena de texto inicial				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.getEveryWord()	setUpStage2	t.getEveryWord(“g”)	2

Objetivo de la Prueba: Comprobar que el programa retorne todas las posibles palabras formadas por una cadena de texto inicial				
Clase	Método	Escenario	Valores de Entrada	Resultado
Trie	t.insert() y t.getEveryWord()	setUpStage2	t.insert(“kermit”) t.getEveryWord(“ki”)	2

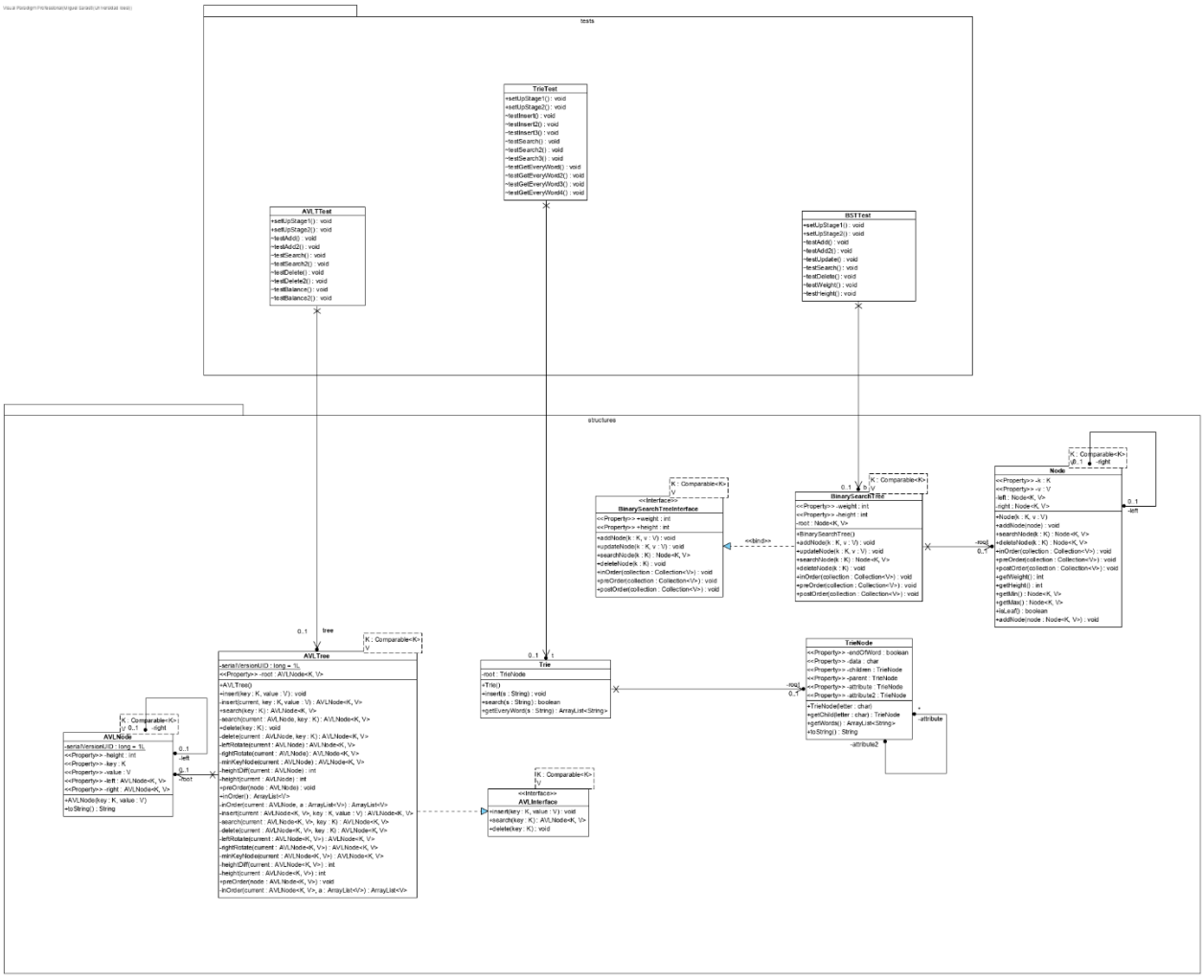


Ilustración 6 Diagrama de clases de las pruebas unitarias.