BASKETBALL MANAGER

DIEGO FERNANDO HIDALGO LÓPEZ
LAURA DANIELA MARTÍNEZ ORTIZ

1 NOVEMBER 2021

ALGORITHMS AND DATA STRUCTURES

ICESI UNIVERSITY

Engineering method

Problem context

The enormous amount of data generated by basketball makes necessary the development of a tool that facilitates the management and consultation of the information. For this purpose, it's required an application that allows to register and consult the statistics about the activities of this sport.

Solution Development

In order to solve the problem previously presented, it will be used the steps indicated in the Method of the Engineering to develop a solution following a systematic approach and taking into account the proposed problematic situation.

Based on the description of the Engineering Method in the book "Introduction to Engineering" by Paul Wright, the steps to be followed in the development of the solution are presented as follows:

1. Problem identification
2. Compilation of necessary information
3. Search of creative solutions
4. Transition from the formulation of ideas to preliminary designs
5. Evaluation and selection of the best solution
6. Preparation of reports and specifications
7. Design implementation

1. Problem identification

At this stage, the needs of the problematic situation are recognized, as well as its symptoms and conditions under which it must be resolved.

Identification of needs and conditions

- FIBA wants to consolidate, in an application, the most relevant data of each of the professional players on the planet.
- The objective is to be able to make different consults that allow the analysis of this data, to know patterns about the development of the sport, the criteria that are gaining strength or, in general, where the sport is heading nowadays. The solution to the problem must ensure that:
    - It includes data per player for the following items: name, age, team and 5 statistics (e.g. points per game, rebounds per game, assists per game, steals per game, blocks per game).
    - It has a quick access to the data (efficiency in the searches).
    - It shows the search time to justify using balanced trees.
    - The search to filter players is not linear.
    - It works with at least 200,000 valid data.

Problem definition

FIBA requires the development of a program that allows you to register and consult the statistics of each basketball player in an efficient and faster way.

2. Compilation of necessary information

In order to have complete clarity in the concepts involved, a search is made for the definitions of the terms most closely related to the problem presented.

BST Tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtree each must also be a binary search tree.

Source: GeeksforGeeks. (2021). Binary Search Tree. Recovered from: https://www.geeksforgeeks.org/binary-search-tree-data-structure/

AVL Tree

AVL tree is a Binary Search Tree (BST) that is height balanced, where the difference between heights of left and right subtrees cannot be more than one for all nodes.

Each node, in addition to the information to be stored, must have the two pointers to the right and left trees, just like the binary search trees, and also the data that controls the balancing factor.

The balancing factor is the difference between the heights of the right and left trees:

By definition, for an AVL tree, this value must be -1, 0 or 1.

If the balance factor of a node is:

- 0 = the node is balanced and its subtrees have exactly the same height.
- 1 = the node is balanced and its right subtree is one level higher.
- - 1 = the node is balanced and its left subtree is one level higher.

If the balancing factor ≥ 2 it is necessary to rebalance.

Source: GeeksforGeeks. (2021). AVL Tree. Recovered from: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

Generics

Generics mean parameterized types. The idea is to allow type (Integer, String, … etc, and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is called a generic entity.

Source: Geeks for Geeks. (2021). Sorting Algorithms. Geeks for Geeks. Recovered from: https://www.geeksforgeeks.org/generics-in-java/

Abstract Data Type (ADT)

The abstract data type is a special kind of data type, whose behavior is defined by a set of values and set of operations. The keyword "Abstract" is used as we can use these datatypes, we can perform different operations. But how those operations are working is totally hidden from the user. The ADT is made of primitive data types, but operation logics are hidden.

Source: Chakraborty, A. (2019). Abstract Data Type in Data Structures. Tutorials Point. Recovered from: https://www.tutorialspoint.com/abstract-data-type-in-data-structures

3. Search of creative solution

At the end of the brainstorming, the following solutions are proposed:

- *Alternative 1:* Implement an application that saves the player's information through an arraylist.
- *Alternative 2:* Implement an application that stores the player's information in the ABB trees and AVL trees, using threads and serialization.
- *Alternative 3:* Implement an application that stores the player's information in the ABB trees and AVL trees, using serialization but all through the console.

4. Transition from the formulation of ideas to preliminary designs

First, ideas that are not feasible are discarded. We decided to discard alternative 1 because does not fulfill the requested data structures and search times.

The review of the other 2 alternatives leads us to the following:

- *Alternative 2.*
  - The player is searched for and when it is found the statistical information is displayed graphically.
  - The search time condition is fulfilled.
  - Allows adding, deleting and searching for players through the graphical interface.
  - Can add data through a file.
  - The statistical attributes are stored in separate trees and then the index is associated with the corresponding player.

- *Alternative 3.*
  - The player is searched and when found the statistical information is displayed by console.
  - It will take time to add and save a large amount of data because it does not handle threads.
  - You can add data through a file, but you will have to put the path manually.

5. Evaluation and selection of the best solution

As the engineering design process evolves, the engineer can evaluate alternative ways to solve the problem in question. Commonly, the engineer abandons the design possibilities that are not promising, thus obtaining a set progressively smaller of options. Feedback, modification and evaluation can occur repeatedly as the device or system evolves from concept to final design.

<u>Criteria</u>

The criteria we chose in this case are those listed below. Next to each criteria, a numerical value has been established with the aim of establishing a weight that indicates which of the possible values of each criteria have more weight and, therefore, are more desirable.

- *Criteria A.* Specified complexity.
  - [2] Satisfied.
  - [1] Not satisfied.
- *Criteria B.* Results.
  - [3] Satisfied.
  - [2] Partially satisfied.
  - [1] Not satisfied.

- *Criteria C.* Waiting time.
  - [3] Little time.
  - [2] Normal time.
  - [1] Long time.
- *Criteria D.*Usability.
  - [3]Easy.
  - [2] Normal.
  - [1] Difficult.

Evaluation

Evaluating the above criteria in the alternatives that are maintained, we obtain the following table:

	Criteria A	**Criteria B**	**Criteria C**	**Criteria D**	**Total**
Alternative 2	2	3	3	3	11
Alternative 3	1	3	1	2	7

Selection

According to the previous evaluation, the Alternative 2 must be selected because it obtained the highest score according to the defined criteria.

6. Preparation of reports and specifications

Review the functional and non-functional requirements section and the class diagram.

7. Design implementation

Review the implemented project.

Requirement specification

Functional requirements

In order to correctly meet with the needs and functionalities required for this project, the system to be developed must be able to:

FR1: **Manage the information of the players** with a name, age, team and 5 statistics (to define)

- **FR1.1 -** It must be possible to register a new player.
- **FR1.2 -** It must be possible to modify the information of a specific player.
- **FR1.3 -** It must be possible to delete a specific player.

FR2: **Import information from a plain text file** with information of the players.

FR3: **Consult information of the players**

- **FR3.1 -** It must be possible to consult the general information of a specific player.
- **FR3.2 -** It must be possible to consult information using a search criteria.

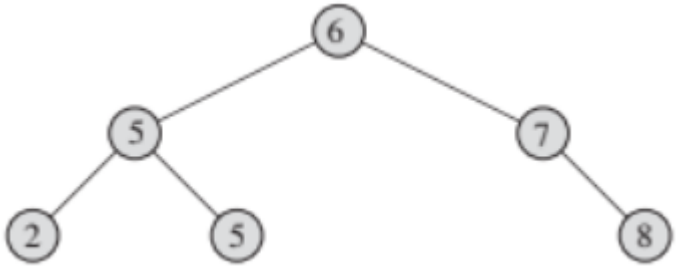
Non-functional requirements

In order to guarantee the correct operation of the system and assure the quality of the software, the system must have the next validations:

NFR1: **Tests.** Implement tests to make sure that the methods work correctly.
NFR2: **Generics.**

Abstract Data Type (ADT) Design

Binary search tree

ADT Binary search tree
BinarySearchTree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>} 
{inv: Let x be a node in a binary search tree. If y is a node in the left subtree of x, then y.key ≤ x.key. If y is a node in the right subtree of x, then y.key ≥ x.key. If a child or the parent is missing, the appropriate attribute contains the value NULL. The root node is the only node in the tree whose parent is NULL.}
Primitive operations: BinarySearchTree: →BinarySearchTree IsEmpty: BST → Boolean Height: BST → Integer Insert: BST x Key x Value →BinarySearchTree ContainsKey: BST x Key → Boolean ContainsValue: BST x Value → Boolean Remove: BST x Key →BinarySearchTree Minimum: BST →Value Maximum: BST →Value PreOrder: BST →BinarySearchTree InOrder: BST →BinarySearchTree PosOrder: BST →BinarySearchTree

BinarySearchTree() → Constructor
"Creates an empty binary search tree to add new nodes"
{pre: TRUE}
{post: Binary search tree initialized}

IsEmpty()→ Analyzer
"Checks if the binary search tree is empty "

{pre: Binary search tree initialized}
{post: TRUE if the root == null FALSE if the queue isn't empty}

Height() → Analyzer
"Returns the largest number of the edges from the root to the most distant leaf node"
{pre: BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}}
{post: <Height>}

Insert(K key, V value) → Modifier
"Inserts a new node in the binary search tree"
{pre: BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Added the element and increase the depth of the branch}

ContainsKey(K key)→ Analyzer
"Checks if a specific key is in the binary tree"
{pre: BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: TRUE if the key is found FALSE if the key doesn't exist}

ContainsValue(V value)→ Analyzer
"Checks if a specific value is in the binary tree"
{pre: BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: TRUE if the value is found FALSE if the value doesn't exist}

Remove(K key)→ Modifier
"Removes a key in the binary search tree"
{pre:BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Deletes the element and decrease the depth of the branch}

Minimum()→ Analyzer
“Search for the minimum element in the binary tree”
{pre:BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Value, returns the minimum value}

Maximum()→ Analyzer
“Search for the maximum element in the binary tree”
{pre:BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Value, returns the maximum value}

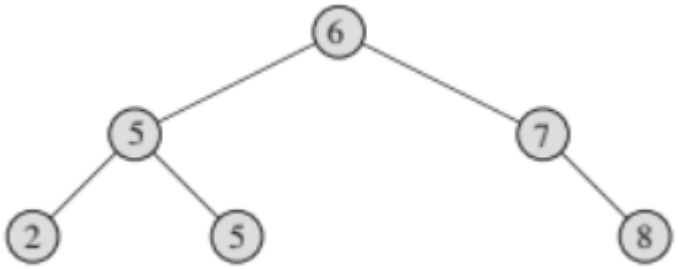
Preorder()→ Analyzer
“Shows the binary tree in preorder traversal ”
{pre:BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Preorder (Root, Left, Right)}

InOrder()→ Analyzer
“Shows the binary tree in inorder traversal ”
{pre:BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Inorder (Left, Root, Right)}

posOrder()→ Analyzer
“Shows the binary tree in posorder traversal ”
{pre:BinarySearchTree tree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Postorder (Left, Right, Root)}

AVL Tree (Balanced Binary Tree)

ADT AVL Tree
AVLtree = {{<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>} }



{inv:Let x be a node in a binary search tree. If y is a node in the left subtree of x, then y.key ≤ x.key. If y is a node in the right subtree of x, then y.key ≥ x.key.
It is height balanced, for each node x, the heights of the left and right subtrees of x differ by at most 1.}

Primitive operations:

BalancedBinaryTree:		→ BalancedBinaryTree
Insert:	AVL x Key x Value	→ BalancedBinaryTree
Remove:	AVL x Key	→ BalancedBinaryTree
BalanceFactor:	AVL x Node	→ BalancedBinaryTree
LeftRotate:	AVL x Node	→ BalancedBinaryTree
RightRotate:	AVL x Node	→ BalancedBinaryTree
Rebalance:	AVL x Node	→ BalancedBinaryTree

BalancedBinaryTree() → Constructor
"Creates an empty AVL tree to add new nodes"
{pre: TRUE}
{post: AVL tree initialized}

Insert(K key, V value) → Modifier
"Inserts a new node in the AVL tree"
{pre: AVLtree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Added the element and increase the depth of the branch}

Remove(K key) → Modifier
"Removes a key in the AVL tree"
{pre: AVLtree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: Deletes the element and decrease the depth of the branch}

BalanceFactor(N node) → Modifier
“Recalculate the balance factor”
{pre: AVLtree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: AVL tree with the balance factor calculated}

LeftRotate(N node) → Modifier
“This rotation is performed when a new node is inserted at the right child of the right subtree.”
{pre: AVLtree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: AVL tree modified with a left rotation}

RightRotate(N node) → Modifier
“This rotation is performed when a new node is inserted at the left child of the left subtree.”
{pre: AVLtree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>}}
{post: AVL tree modified with a right rotation}

Rebalance(N node) → Modifier
“Rebalance the AVL tree when a node is inserted or removed”
{pre: AVLtree= {<x_1, x_2, x_3, … ,x_n>, <k_1, k_2, k_3, … ,k_n>} and the tree is unbalanced}
{post: According to the balanced factor, the method will determined which rotation is necessary}

Unit test design
Configuration of the scenes

Name	Class	Scenes
setupScenary1	AVLTest	3 Players have been added: name = “Samuel” age = 33 team = “Col” points = 100 rebounds = 300 assists = 200 steals = 90 blocks = 123 name = “Charles” age = 29 team = “Arg” points = 107 rebounds = 500 assists = 254 steals = 70 blocks = 213 name = “Aiden” age = 36 team = “Col” points = 205 rebounds = 330 assists = 120 steals = 88 blocks = 103

Name	Class	Scenes
setupScenary1	PlayerTest	A Player has been added: name = “Lanita” age = 25 team = “Guam” points = 213 rebounds = 847 assists = 100 steals = 78 blocks = 352

Name	Class	Scenes
setupScenary1	FIBATest	5 Players have been added: name = “Lanita” age = 25 team = “Guam”

		points = 213 rebounds = 847 assists = 100 steals = 78 blocks = 352 name = “Samuel” age = 33 team = “Col” points = 100 rebounds = 300 assists = 200 steals = 90 blocks = 123 name = “Charles” age = 29 team = “Arg” points = 207 rebounds = 500 assists = 254 steals = 70 blocks = 213 name = “Aiden” age = 36 team = “Col” points = 105 rebounds = 330 assists = 120 steals = 88 blocks = 103 name = “Cobe” age = 28 team = “Mex” points = 208 rebounds = 200 assists = 204 steals = 85 blocks = 216

Design of test cases for the model

Player methods

Purpose of the test: Verify that the getters work.				
Class	**Method**	**Scenes**	**Inputs**	**Results**
Player	getName	setupScenary1		Lanita It is correct according to the player´s info

				given in the scenary.
Player	getAge	setupScenary1		25 It is correct according to the player´s info given in the scenary.
Player	getTeam	setupScenary1		Guam It is correct according to the player´s info given in the scenary.
Player	getPoints	setupScenary1		213 It is correct according to the player´s info given in the scenary.
Player	getRebounds	setupScenary1		847 It is correct according to the player´s info given in the scenary.
Player	getAssists	setupScenary1		100 It is correct according to the player´s info given in the scenary.
Player	getSteals	setupScenary1		78 It is correct according to the player´s info given in the scenary.
Player	getBlocks	setupScenary1		352 It is correct according to the player´s info given in the scenary.

Purpose of the test: Verify that the setters work.

Class	Method	Scenes	Inputs	Results
Player	setName	setupScenary1	newName = “Frank”	Frank The modification of the name is done correctly.
Player	setAge	setupScenary1	newAge = 26	26 The modification of the age is done correctly.
Player	setTeam	setupScenary1	newTeam = "Arg"	Arg The modification of the team is done correctly.
Player	setPoints	setupScenary1	newPoints = 111.0	111 The modification of the points is done correctly.
Player	setRebounds	setupScenary1	newRebounds = 234.0	234 The modification of the rebounds is done correctly.
Player	setAssists	setupScenary1	newAssists = 294.0	294 The modification of the assists is done correctly.
Player	setSteals	setupScenary1	newSteals = 114.0	114 The modification of the steals is done correctly.
Player	setBlocks	setupScenary1	newBlocks = 14.0	14 The modification of the blocks is done correctly.

FIBAManager Methods

Purpose of the test: Verify that the methods of the FIBAManager works.				
Class	**Method**	**Scenes**	**Inputs**	**Results**
FIBAManager	addPlayer	setupScenary1	player = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	True The new player has been added.

FIBAManager	searchPlayerByName	setupScenary1	player = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	The player is added and found when it is searched by name.
FIBAManager	searchPlayerByValue	setupScenary1	player = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	The player is added and found when it is searched by a value.
FIBAManager	changePlayer	setupScenary1		The info of one of the players has been changed correctly.
FIBAManager	deletePlayer	setupScenary1		One of the players is deleted correctly.

Design of test cases for each data structure

BST Methods

Purpose of the test: Verify that the methods in the BST works correctly.				
Class	**Method**	**Scenes**	**Inputs**	**Results**
BinarySearchTree	Insert		player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	Correctly adds the 4 players taking into account their points.
BinarySearchTree	ContainsKey		player1 = new Player("Lanita",	TRUE The keys are in

			25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	the tree.
BinarySearchTree	Remove		player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	Four players are added. The last one is removed. To verify that, it is used the method Height.
BinarySearchTree	Minimum		player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new	Four players are added. Player2 is the minimum (according to the points (203))

			Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	
BinarySearchTree	Maximum		player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	Four players are added. Player4 is the maximum (according to the points (313))
BinarySearchTree	PreOrder		player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	4 players added. Printed in preorder traversal: player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)
BinarySearchTree	InOrder		player1 = new	4 players added.

			Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	Printed in inorder traversal: player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)
BinarySearchTree	PosOrder		player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162)	4 players added. Printed in posorder traversal: player2 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302) player4 = new Player("Link", 37, "Mex", 313, 507, 230, 88, 162) player3 = new Player("Juan", 35, "Arg", 217, 247, 150, 58, 152) player1 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78,

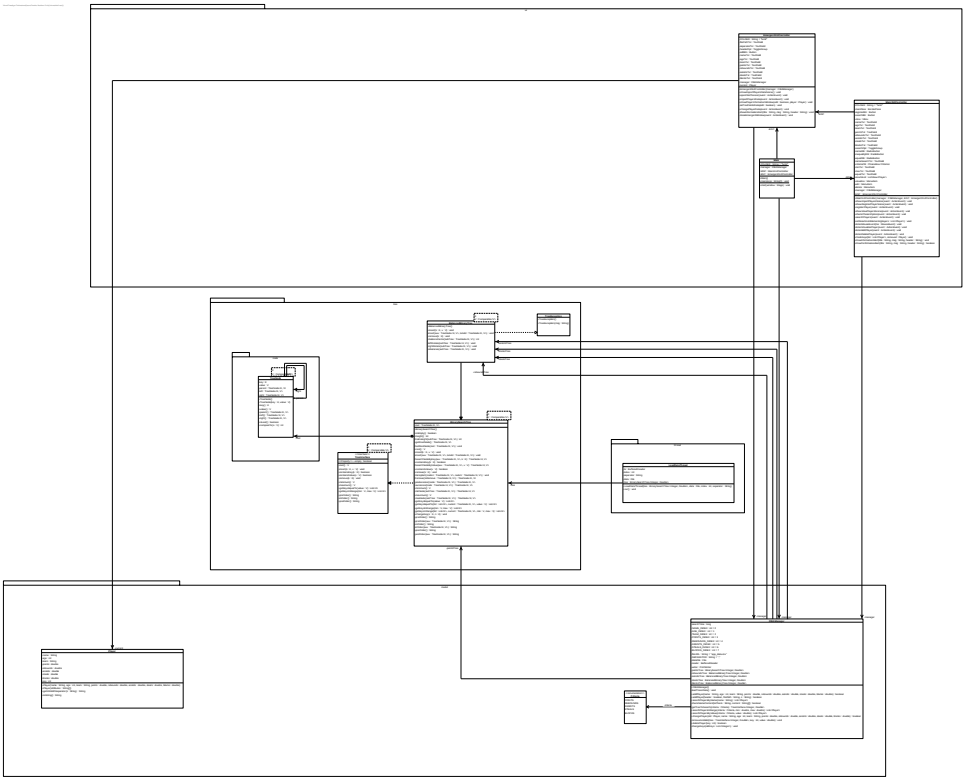
				352)

AVL Methods

The rest of the methods in the AVL tree are auxiliaries, meaning that they are part of the following methods and are tested in them.

Purpose of the test: Verify that the method insert in the AVL works.				
Class	**Method**	**Scenes**	**Inputs**	**Results**
BalancedBinaryTree	Insert	setupScenary1	player4 = new Player("Lanita", 25, "Guam", 213, 847, 100, 78, 352) player5 = new Player("Carlos", 27, "Guam", 203, 842, 111, 76, 302	Correctly adds the 2 players. There are 5 players in the tree. It is checked the balance factor.

Purpose of the test: Verify that the method remove in the AVL works.				
Class	**Method**	**Scenes**	**Inputs**	**Results**
BalancedBinaryTree	Remove	setupScenary1	player4 = new Player("Lanita", 25, "Guam", 219, 847, 100, 78, 352)	Player4 is added and then it is correctly removed. It is verified by calculating the height.



Visual Paradigm Professional(Laura Daniela Martínez Ortiz(Universidad Icesi))