ENGINEERING METHOD

PARTICIPANTS

CAMILO CAMPAZ JIMÉNEZ

DANIEL ESTEBAN JARABA GAVIRIA

JOHAN STIVEN RICARDO SIBAJA

TEACHER

URAM ANIBAL SOSA

INTEGRATIVE TASK #2

ALGORITHMS AND DATA STRUCTURES

2021-02

ENGINEERING METHOD

PROBLEM CONTEXT:

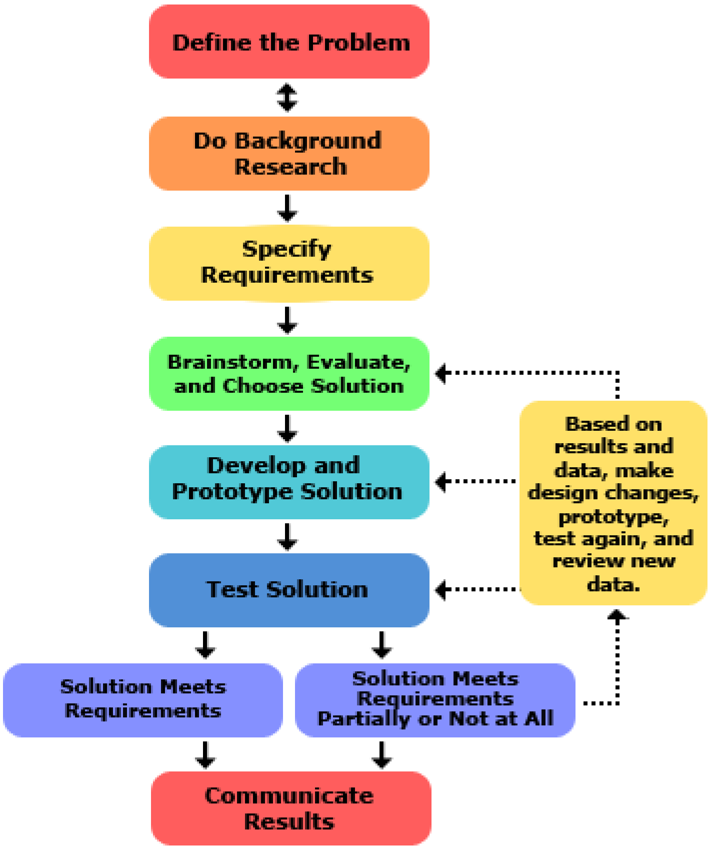
The FIBA (Federation Internationale de Basketball Amateur) is an important association directly related with basketball, as a wish to improve their development with the players, they want to analyze their statistics to improve their performance in the playing field. Due to the amount of players registered, they need a software application with a very efficient implementation that helps them with the analysis in a short amount of time.

SOLUTION DEVELOPMENT:

To solve the previous situation, the Engineering Method was chosen to develop the solution following a systematic approach and in accordance with the problematic situation posed.

Based on the description of the Engineering Method in the book "Introduction to Engineering" by Paul Wright.

The following flow chart was defined, the steps of which we will follow in the development of the solution.



Define the Problem
Do Background Research
Specify Requirements
Brainstorm, Evaluate, and Choose Solution
Develop and Prototype Solution
Test Solution
Based on results and data, make design changes, prototype, test again, and review new data.
Solution Meets Requirements
Solution Meets Requirements Partially or Not at All
Communicate Results

IDENTIFICATION OF THE PROBLEM:

In this section, it is important to have a good definition of what the solution to the problem that is being addressed is necessary to contain.

Identification of needs:

- The FIBA needs an application that gathers the information of all their players.
- The application must be able to analyze more than 200 thousands of statistics in a short amount of time.
- The information analyzed must be filtered based on a parameter of the players.
- The tool must have at least 4 parameters filtered in an amount of time equals to O(log n).
- The tool must use the following data structures: BST, ABT, AVL.

INFORMATION GATHERING:

In this section the concepts that are essential for understanding the solution of the specific problem are specified.

Algorithm:

An ordered set of systematic operations that allows making a calculation and finding the solution to a type of problem.

Sort algorithm:

In computation and mathematics, an ordering algorithm is an algorithm that puts elements of a list or a vector in a sequence given by an order relation, that is, the output result must be a permutation —or reordering— of the input that satisfies the given order relation.

Data structures:

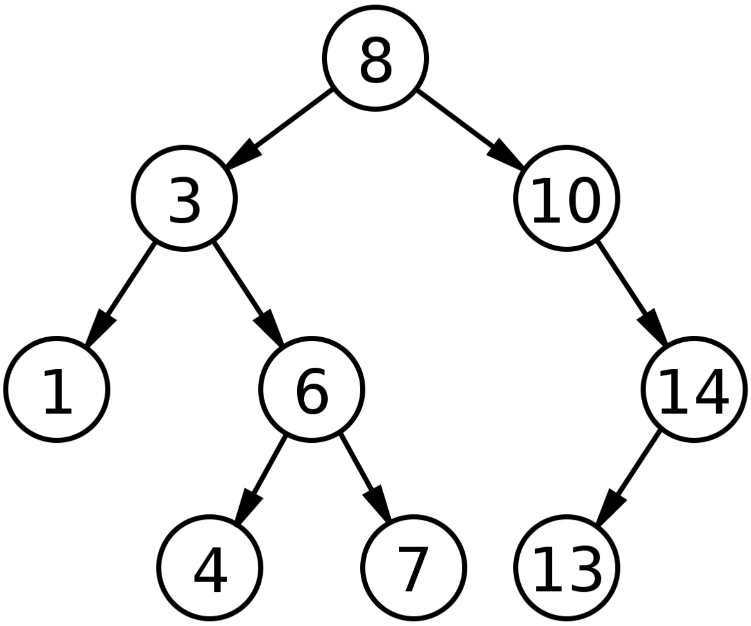
In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. Different types of data structures are suitable for different types of applications, and some are highly specialized for specific tasks.

Temporal complexity:

In computing, time complexity is computational complexity that describes the amount of time it takes to run an algorithm. ... Therefore, the amount of time required and the number of elementary operations performed by the algorithm differ by a constant factor at most.

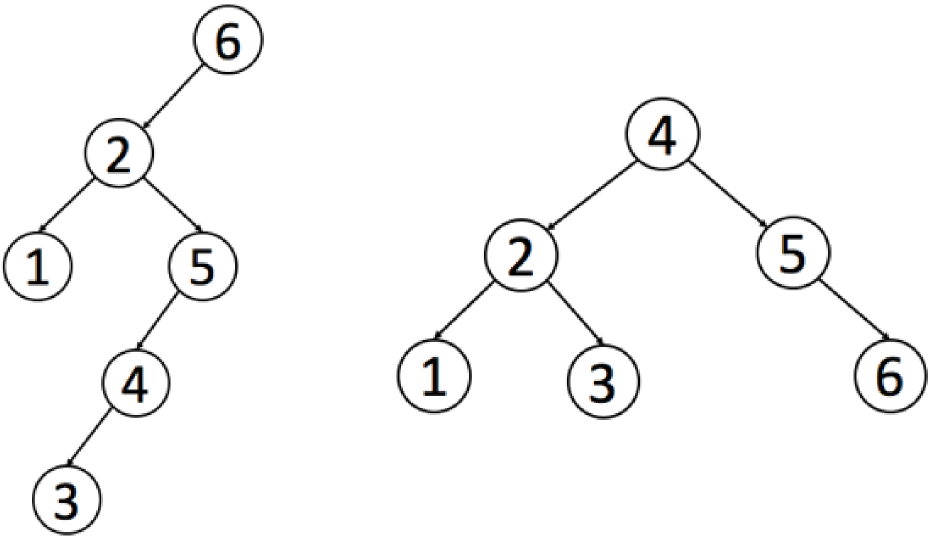
Binary search tree:

A binary search tree (BST), also called an ordered or sorted binary tree, is a data structure that visually looks like a tree, is conformed by nodes, where each node stores information and has a key. Each node also has a left and a right child. The left child value has to be less than the parent or failing empty, and the right child value has to be greater than or equal to the parent or failing empty.



Balanced binary tree:

A balanced binary tree is a data structure that follows the concept of binary tree, with the difference that the subtree left and the subtree right did not differ in height more than 1. In general, we can say that a balanced tree is a proportional structure where the left part has approximately the same number of nodes as the right part.



(a) binary search tree (b) balanced binary search tree

SEARCH FOR CREATIVE SOLUTIONS:

In this section is going to be introduced the ideas that can be optimal to solve the problem that is proposed.

The important aspect to analyze in this case, isn't the way we store the data, is the form we filter and analyze the statistics of the player, for that, we thought about 2 proposals:

Proposal 1:

For this proposal we will focus on the worst case possible for the solution. The objective of the model is to analyze the parameters of the data and filter it, for that, the easiest way that comes to our minds is a binary tree, which has the data organized by a key, from lowest to highest. It simplifies the way that we can filter the statistics, because we know that in the left of a node are values lower and in the right are values higher. Now, if we think in a huge dataset we can think that a certain amount of that data is repeated, for that we will store the repeated data in a list, and manage this list like a single node.

Now, following our first idea, that our objective is to focus on the worst case possible, the best data structure to manage the data is the balanced binary tree, works like a BST, with the difference that the worst case wouldn't be as bad as with the BST.

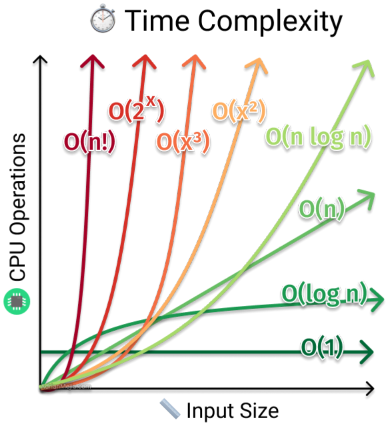
Proposal 2:

Our second proposal will focus on the simplicity of the solution. For this, we think about a simple binary tree that will store the data for the analysis and the filter. This structure is optimal to our objective, because it organizes the data from lower to higher but in a simple way. That means, we don't need intermediary structures for its management. For the filter, we know that placing bounds mean nodes higher than and less than another.

TRANSITION FROM FORMULATION OF IDEAS TO PRELIMINARY DESIGNS

In this case it is considered that the best option is the proposal 1, however it is important to see why the other proposal is not that efficient.

As we mentioned before, the best proposal for our tool will be the one that manages and filters the information in the lowest time possible. Due to this, the first proposal does the principal operations in a lower time than the second one, this because the second doesn't group up the repeated information. Also, because the worst case for the BST depends on how much unbalanced the tree is, that means, the worst case of all the tree will be the node with more height, on the other hand, the worst situation for the balanced binary tree will be the same for all its terminal nodes or leafs.



EVALUATION AND SELECTION OF THE BEST SOLUTION

Is time to select the final choice, so it is important to define evaluation criteria that let us prove that the solution we select is optimal (for us) for this problem, in particular thinking about it generally.

Criteria A: Solve the problem correctly

- [2] Yes
- [1] No

Criteria B: The data structures that use let storage data naturally (it means to don't need to create another structure to save in)

- [3] Yes
- [2] Sometimes
- [1] No

Criteria C: The data structures that use the most are optimal (It means that for example the most are O(1))

- [3] Yes
- [2] More or less
- [1] no

	Criteria A	***Criteria B***	***Criteria C***	***Total***
Proposal 1	2	2	3	7
Proposal 2	2	3	1	6

REPORT PREPARATION AND SPECIFICATIONS

Don't forget the limitations of the solution, so it is important to explain which are the limitations of the solution or considerations for the implementation.

In this case is needed show how will be the solution that is proposed

Considerations:

1. For the optimal work in the data structures it might be all the data in the correct format
2. All the data structures are might be defined properly
3. To create properly the objects all the data that is given might be correct
4. All the data is complete

DESIGN IMPLEMENTATION

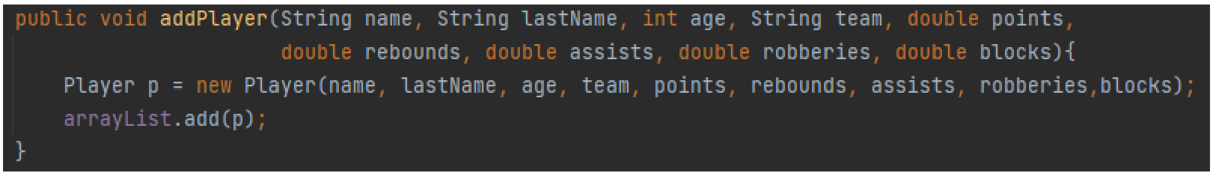
List of tasks to implement:

a. Add player
b. Import data
c. Filter by assists
d. Filter by blocks
e. Filter by points
f. Filter by rebounds
g. Filter by robberies

Specifications and subroutines:

A:

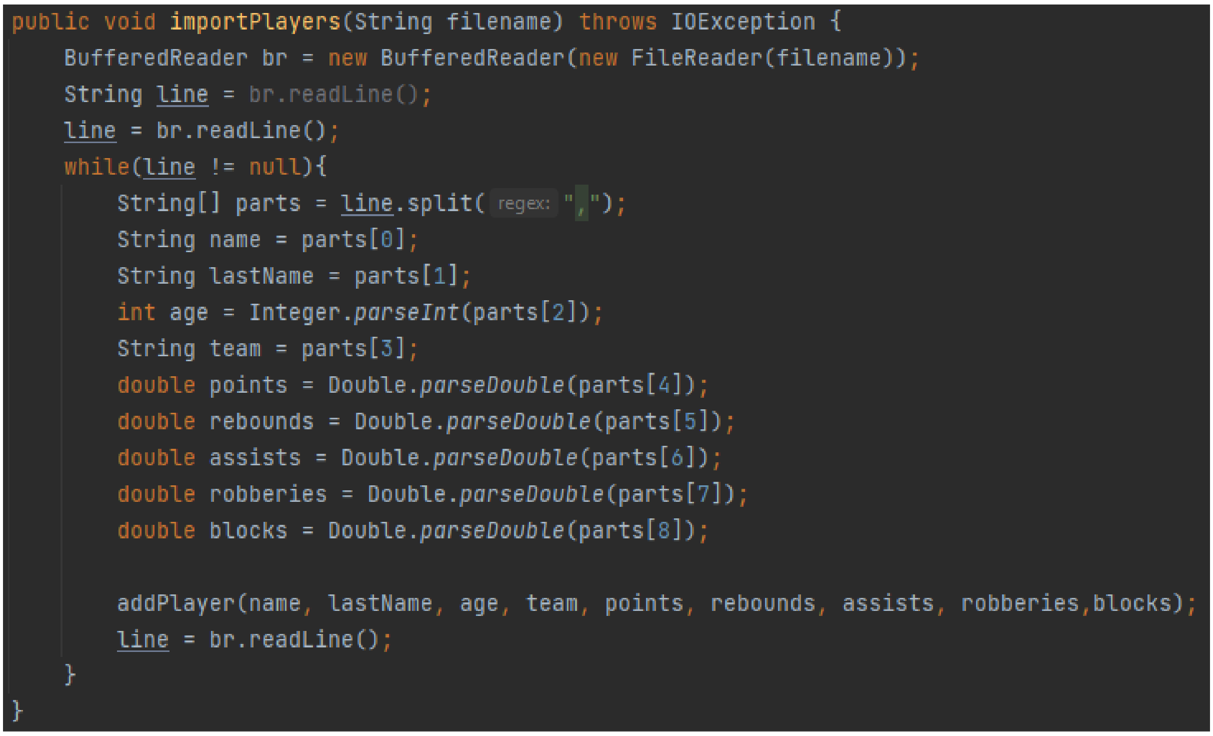
Name:	addPlayer
Description:	Adds a new player
Input:	- name: First name of player - lastName: Last name of player - age: Age of player - team: Team of the player - points: Points of the player - rebounds: Rebounds of the player - assists: Assists of the player - robberies: Robberies of the player - blocks: Blocks of the player
Output:	



```
public void addPlayer(String name, String lastName, int age, String team, double points,
                      double rebounds, double assists, double robberies, double blocks){
    Player p = new Player(name, lastName, age, team, points, rebounds, assists, robberies,blocks);
    arrayList.add(p);
}
```

B:

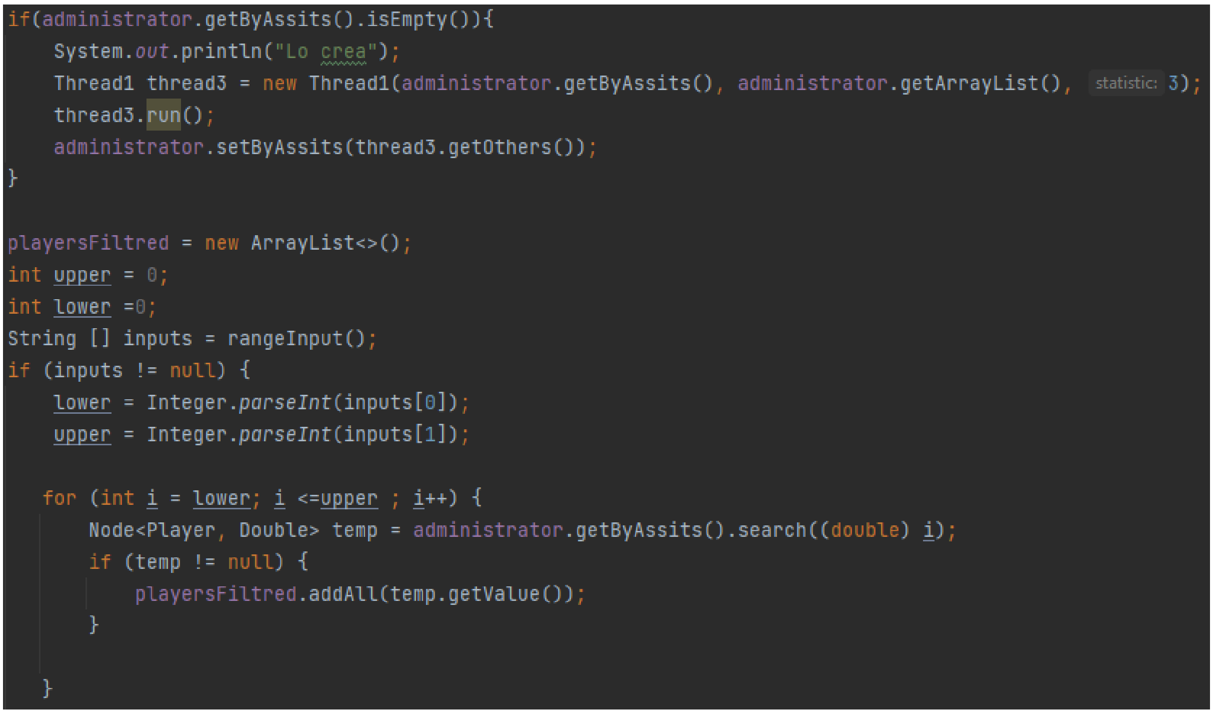
Name:	importPlayers
Description:	Import data from a file
Input:	- filename: Route of the file
Output:	



```
public void importPlayers(String filename) throws IOException {
    BufferedReader br = new BufferedReader(new FileReader(filename));
    String line = br.readLine();
    line = br.readLine();
    while(line != null){
        String[] parts = line.split( regex: ",");
        String name = parts[0];
        String lastName = parts[1];
        int age = Integer.parseInt(parts[2]);
        String team = parts[3];
        double points = Double.parseDouble(parts[4]);
        double rebounds = Double.parseDouble(parts[5]);
        double assists = Double.parseDouble(parts[6]);
        double robberies = Double.parseDouble(parts[7]);
        double blocks = Double.parseDouble(parts[8]);

        addPlayer(name, lastName, age, team, points, rebounds, assists, robberies,blocks);
        line = br.readLine();
    }
}
```

C:

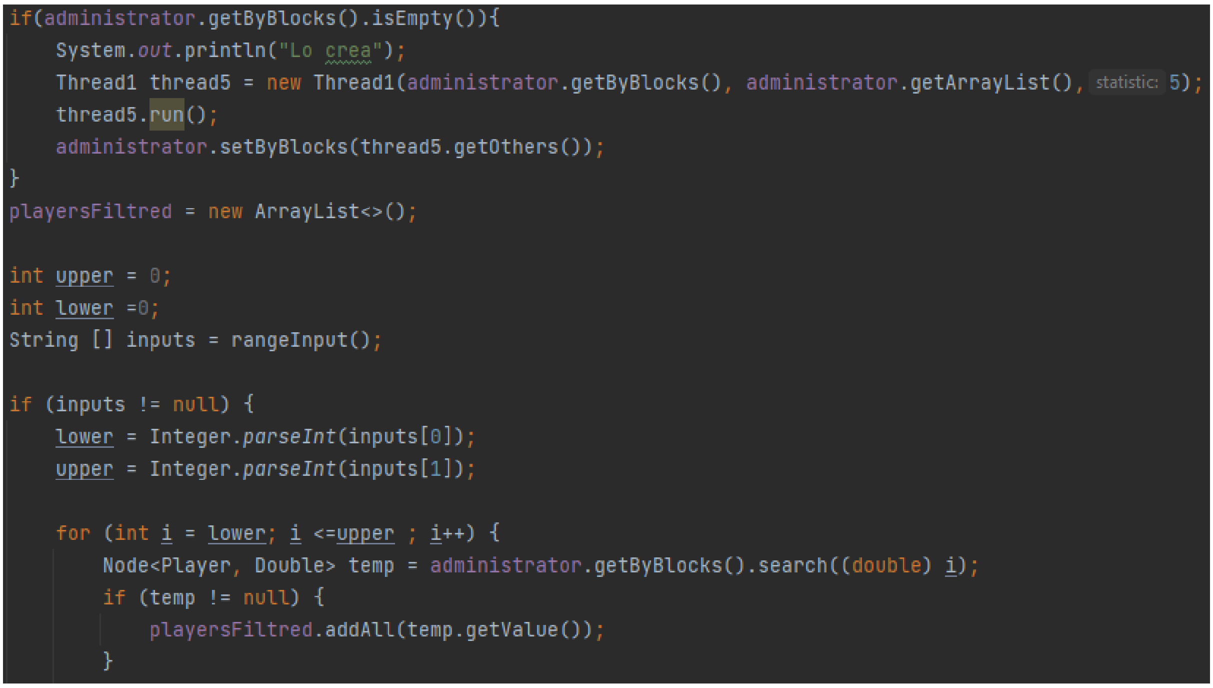
Name:	filterByAssists
Description:	Filter information of the player by assists
Input:	- dataset: Dataset of the players
Output:	- filterDataset: Filtered dataset



```
if(administrator.getByAssits().isEmpty()){
    System.out.println("Lo crea");
    Thread1 thread3 = new Thread1(administrator.getByAssits(), administrator.getArrayList(),  statistic: 3);
    thread3.run();
    administrator.setByAssits(thread3.getOthers());
}

playersFiltred = new ArrayList<>();
int upper = 0;
int lower =0;
String [] inputs = rangeInput();
if (inputs != null) {
    lower = Integer.parseInt(inputs[0]);
    upper = Integer.parseInt(inputs[1]);

   for (int i = lower; i <=upper ; i++) {
       Node<Player, Double> temp = administrator.getByAssits().search((double) i);
       if (temp != null) {
           playersFiltred.addAll(temp.getValue());
       }

   }
```

D:

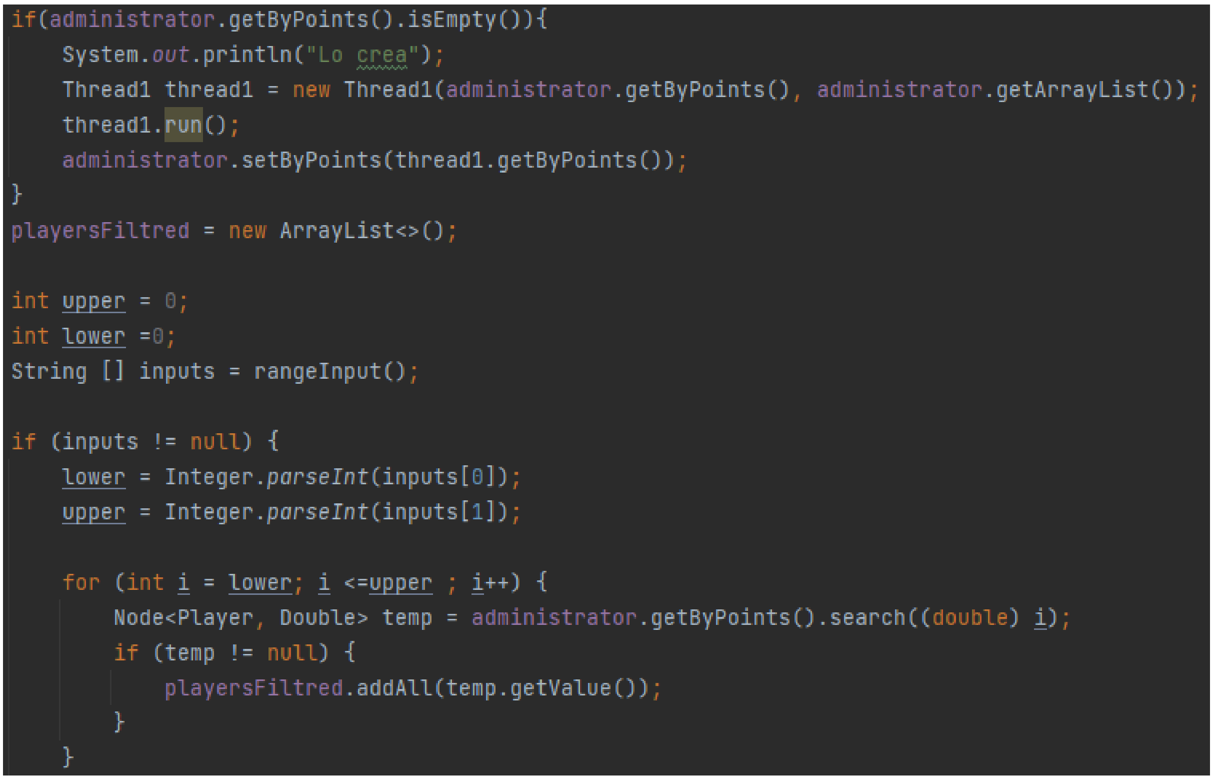
Name:	filterByBlocks
Description:	Filter information of the player by blocks
Input:	- dataset: Dataset of the players
Output:	- filterDataset: Filtered dataset



```
if(administrator.getByBlocks().isEmpty()){
    System.out.println("Lo crea");
    Thread1 thread5 = new Thread1(administrator.getByBlocks(), administrator.getArrayList(), statistic: 5);
    thread5.run();
    administrator.setByBlocks(thread5.getOthers());
}
playersFiltred = new ArrayList<>();

int upper = 0;
int lower =0;
String [] inputs = rangeInput();

if (inputs != null) {
    lower = Integer.parseInt(inputs[0]);
    upper = Integer.parseInt(inputs[1]);

    for (int i = lower; i <=upper ; i++) {
        Node<Player, Double> temp = administrator.getByBlocks().search((double) i);
        if (temp != null) {
            playersFiltred.addAll(temp.getValue());
        }
```

E:

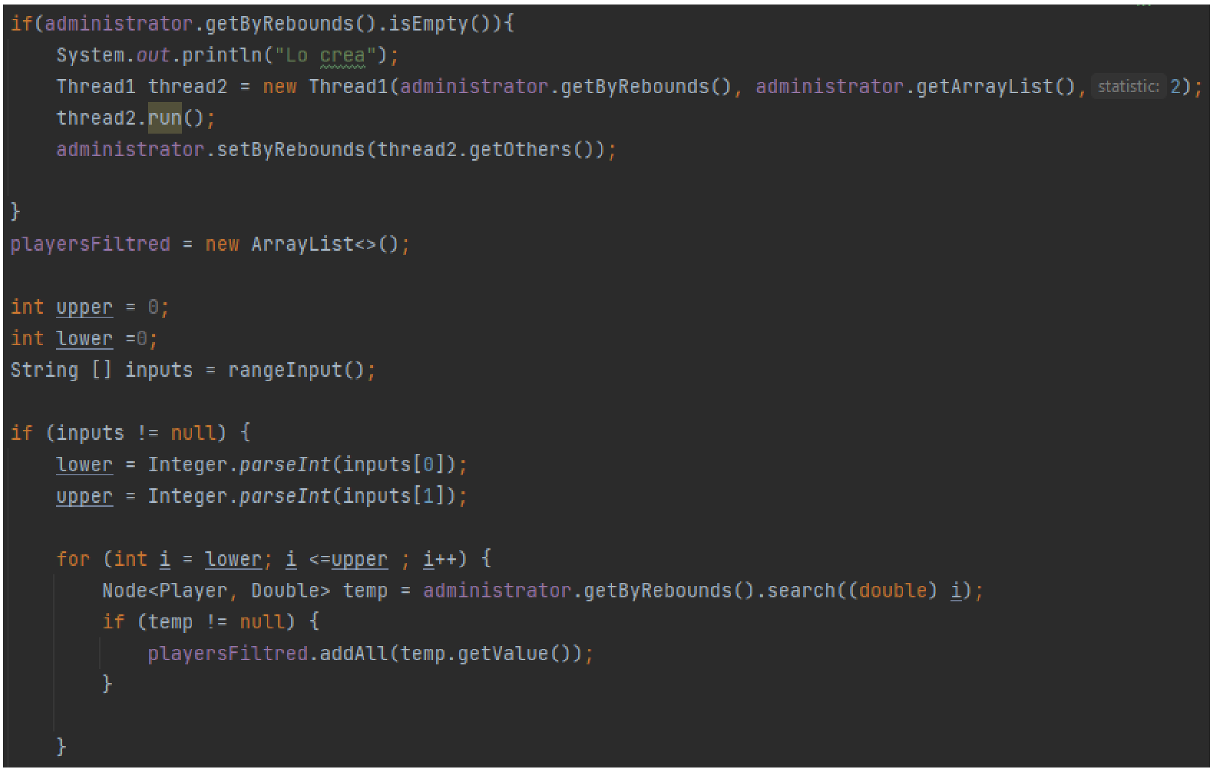
Name:	filterByPoints
Description:	Filter information of the player by points
Input:	- dataset: Dataset of the players
Output:	- filterDataset: Filtered dataset



```
if(administrator.getByPoints().isEmpty()){
    System.out.println("Lo crea");
    Thread1 thread1 = new Thread1(administrator.getByPoints(), administrator.getArrayList());
    thread1.run();
    administrator.setByPoints(thread1.getByPoints());
}
playersFiltred = new ArrayList<>();

int upper = 0;
int lower =0;
String [] inputs = rangeInput();

if (inputs != null) {
    lower = Integer.parseInt(inputs[0]);
    upper = Integer.parseInt(inputs[1]);

    for (int i = lower; i <=upper ; i++) {
        Node<Player, Double> temp = administrator.getByPoints().search((double) i);
        if (temp != null) {
            playersFiltred.addAll(temp.getValue());
        }
    }
```

F:

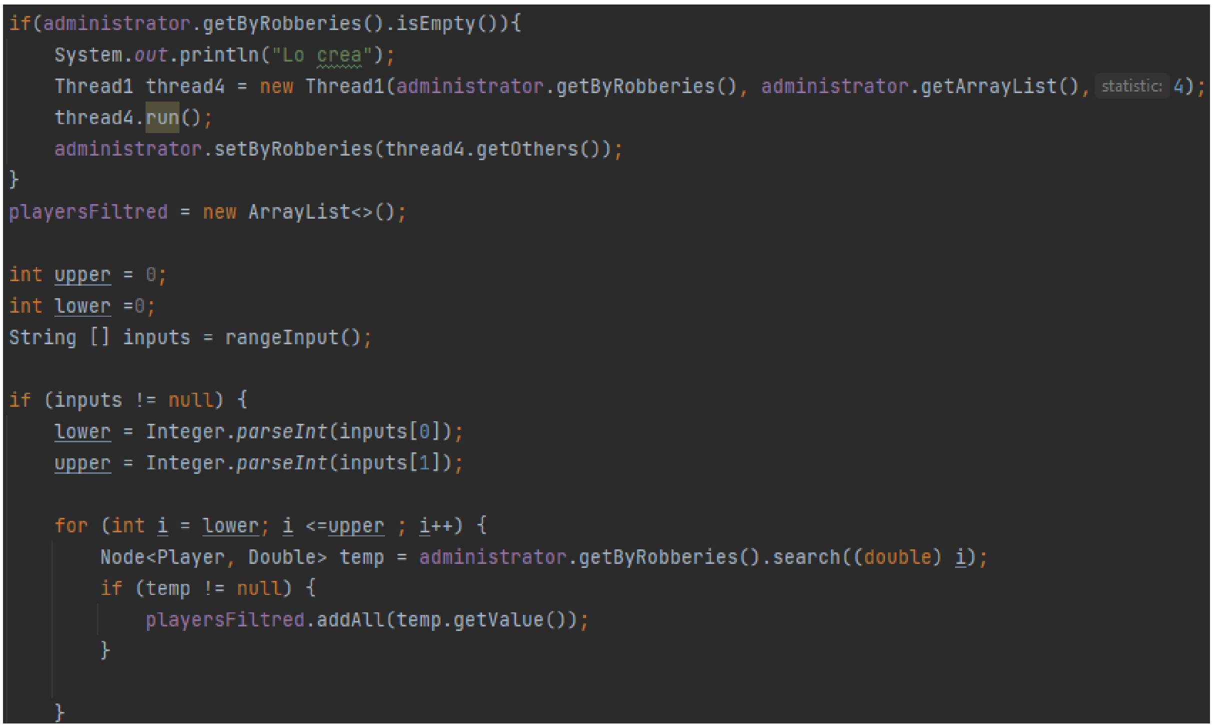
Name:	filterByRebounds
Description:	Filter information of the player by rebounds
Input:	- dataset: Dataset of the players
Output:	- filterDataset: Filtered dataset



```
if(administrator.getByRebounds().isEmpty()){
    System.out.println("Lo crea");
    Thread1 thread2 = new Thread1(administrator.getByRebounds(), administrator.getArrayList(), statistic: 2);
    thread2.run();
    administrator.setByRebounds(thread2.getOthers());

}
playersFiltred = new ArrayList<>();

int upper = 0;
int lower =0;
String [] inputs = rangeInput();

if (inputs != null) {
    lower = Integer.parseInt(inputs[0]);
    upper = Integer.parseInt(inputs[1]);

    for (int i = lower; i <=upper ; i++) {
        Node<Player, Double> temp = administrator.getByRebounds().search((double) i);
        if (temp != null) {
            playersFiltred.addAll(temp.getValue());
        }

    }
```

G:

Name:	filterByRobberies
Description:	Filter information of the player by robberies
Input:	- dataset: Dataset of the players
Output:	- filterDataset: Filtered dataset



```
if(administrator.getByRobberies().isEmpty()){
    System.out.println("Lo crea");
    Thread1 thread4 = new Thread1(administrator.getByRobberies(), administrator.getArrayList(), statistic: 4);
    thread4.run();
    administrator.setByRobberies(thread4.getOthers());
}
playersFiltred = new ArrayList<>();

int upper = 0;
int lower =0;
String [] inputs = rangeInput();

if (inputs != null) {
    lower = Integer.parseInt(inputs[0]);
    upper = Integer.parseInt(inputs[1]);

    for (int i = lower; i <=upper ; i++) {
        Node<Player, Double> temp = administrator.getByRobberies().search((double) i);
        if (temp != null) {
            playersFiltred.addAll(temp.getValue());
        }

    }
```