

PRÁCTICA 1

Divide y vencerás

A continuación, se muestran una serie de capturas del código con una breve descripción de su funcionamiento:

main

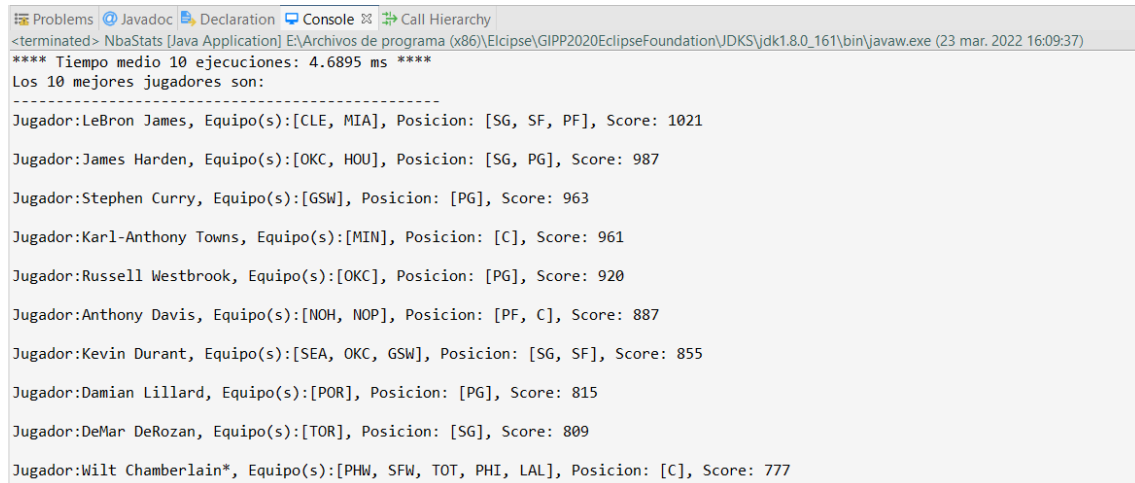
Carga del archivo NbaStats.csv según la ruta declarada en directorio, inicialización de tiempos a 0 y media de 10 tiempos de ejecución para los 10 mejores jugadores. Devuelve por pantalla dichos 10 mejores jugadores indicando el nombre, sus respectivos equipos, posiciones y score:

```
public static void main(String[] args) {
    NbaStats nba = new NbaStats();
    long Tinitial = 0, Tfinal = 0, Ttotal = 0;
    int topNjugadores = 10;
    cargarArchivo(directorio);

    for (int i = 0; i < 10; i++) {
        Tinitial = System.nanoTime();
        NbaStats.mergeSort(nba.getJugadores());
        Tfinal = System.nanoTime();
        Ttotal += Tfinal - Tinitial;
    }

    System.out.println("**** Tiempo medio 10 ejecuciones: " + (Ttotal / 10) / Math.pow(10, 6) + " ms ****");
    System.out.println("Los " + topNjugadores + " mejores jugadores son: ");
    System.out.println("-----");

    for (int i = nba.getJugadores().size() - 1; i >= nba.getJugadores().size() - topNjugadores; i--) {
        System.out.println(nba.getJugadores().get(i).toString() + "\n");
    }
}
```



```
<terminated> NbaStats [Java Application] E:\Archivos de programa (x86)\Eclipse\GIPP2020\EclipseFoundation\JDKS\jdk1.8.0_161\bin\javaw.exe (23 mar. 2022 16:09:37)
**** Tiempo medio 10 ejecuciones: 4.6895 ms ****
Los 10 mejores jugadores son:
-----
Jugador:LeBron James, Equipo(s):[CLE, MIA], Posicion: [SG, SF, PF], Score: 1021
Jugador:James Harden, Equipo(s):[OKC, HOU], Posicion: [SG, PG], Score: 987
Jugador:Stephen Curry, Equipo(s):[GSW], Posicion: [PG], Score: 963
Jugador:Karl-Anthony Towns, Equipo(s):[MIN], Posicion: [C], Score: 961
Jugador:Russell Westbrook, Equipo(s):[OKC], Posicion: [PG], Score: 920
Jugador:Anthony Davis, Equipo(s):[NOH, NOP], Posicion: [PF, C], Score: 887
Jugador:Kevin Durant, Equipo(s):[SEA, OKC, GSW], Posicion: [SG, SF], Score: 855
Jugador:Damian Lillard, Equipo(s):[POR], Posicion: [PG], Score: 815
Jugador:DeMar DeRozan, Equipo(s):[TOR], Posicion: [SG], Score: 809
Jugador:Wilt Chamberlain*, Equipo(s):[PHW, SFW, TOT, PHI, LAL], Posicion: [C], Score: 777
```

Esquema DyV con MergeSort

mergeSort se encarga de ordenar las mitades izquierda y derecha de *array* de manera recursiva, y pasa como parámetros a *merge* los *ArrayList izquierda*, *derecha* y *array*, que devolverá el *ArrayList* totalmente ordenado.

El orden de complejidad de este algoritmo es logarítmico: $O(n \log n)$.

```
//Esquema DyV
public static ArrayList<Player> mergeSort(ArrayList<Player> array) {
    ArrayList<Player> izquierda = new ArrayList<Player>();
    ArrayList<Player> derecha = new ArrayList<Player>();
    int centro;
    //caso base
    if (array.size() == 1) {
        return array;
    } else {
        centro = array.size() / 2;
        for (int i = 0; i < centro; i++) {
            izquierda.add(array.get(i));
        }
        for (int i = centro; i < array.size(); i++) {
            derecha.add(array.get(i));
        }
        izquierda = mergeSort(izquierda);
        derecha = mergeSort(derecha);

        merge(izquierda, derecha, array);
    }
    return array;
}
```

merge

Recibe como parámetros los *ArrayList izquierda*, *derecha* y *array*, y se encarga de ordenar de menor a mayor según el score. Resto se encargará de añadir, en caso necesario, los elementos que hayan quedado sin procesar de una de las listas.

```
private static void merge(ArrayList<Player> izquierda, ArrayList<Player> derecha, ArrayList<Player> array) {
    int indiceIzquierda = 0;
    int indiceDerecha = 0;
    int indiceOrigen = 0;

    while (indiceIzquierda < izquierda.size() && indiceDerecha < derecha.size()) {
        if ((izquierda.get(indiceIzquierda).compareTo(derecha.get(indiceDerecha))) < 0) {
            array.set(indiceOrigen, izquierda.get(indiceIzquierda));
            indiceIzquierda++;
        } else {
            array.set(indiceOrigen, derecha.get(indiceDerecha));
            indiceDerecha++;
        }
        indiceOrigen++;
    }

    ArrayList<Player> resto;
    int indiceResto;
    if (indiceIzquierda >= izquierda.size()) {
        resto = derecha;
        indiceResto = indiceDerecha;
    } else {
        resto = izquierda;
        indiceResto = indiceIzquierda;
    }
    for (int i = indiceResto; i < resto.size(); i++) {
        array.set(indiceOrigen, resto.get(i));
        indiceOrigen++;
    }
}
```

cargarArchivo

En este método:

Se define el sistema de separación para adaptar el CSV a un ArrayList y poder manejar los diferentes datos.

Se comprueba si *jugadores* está vacío. Si lo está, añade los datos del jugador (nombre, equipo, posición y los respectivos cálculos para obtener el score).

Si el jugador está ya añadido en la estructura de datos, añadiremos el equipo en la variable *teams*, la posición en la variable *positions* y haremos la media del *score* que ya haya y el *score* correspondiente en el nuevo año. Si no está añadido, lo añade a *jugadores*, como se ha indicado en el apartado anterior.

Finalmente, se devuelve un ArrayList con los nombres de los jugadores con sus respectivos equipos, posiciones y scores.

```
public static void cargarArchivo(String archivo) {
    try {
        Scanner scan = new Scanner(new File(archivo));
        String line;
        String[] items;

        // [2]PlayerName, [4]Pos, [6]Tm, [7]FG%, [8]PTS |

        while (scan.hasNextLine()) {
            line = scan.nextLine().trim();
            if (line.isEmpty() || line.startsWith("#") || line.contains("%"))
                continue;
            items = line.split(",");

            if (jugadores.size() == 0) {
                jugadores.add(new Player(items[2], items[6], items[4],
                    (int)Double.parseDouble(items[7].isEmpty() ? "0.0" : items[7].replace(".", ""))*Integer.parseInt(items[8].isEmpty() ? "0" : items[8])/100));
                continue;
            }

            if (jugadores.get(jugadores.size()-1).getPlayerName().equals(items[2])){
                Player ultimoJug = jugadores.get(jugadores.size()-1);

                if(!ultimoJug.getTeams().contains(items[6])) ultimoJug.getTeams().add(items[6]);
                if(!ultimoJug.getPositions().contains(items[4])) ultimoJug.getPositions().add(items[4]);
                int score = (int) (Double.parseDouble(items[7].isEmpty() ? "0.0" : items[7].replace(".", "")) * (Integer.parseInt(items[8].isEmpty() ? "0" : items[8]) / 100);

                ultimoJug.setScore((ultimoJug.getScore()+score)/2);
            } else{
                jugadores.add(new Player(items[2], items[6], items[4],
                    (int)Double.parseDouble(items[7].isEmpty() ? "0.0" : items[7].replace(".", ""))*Integer.parseInt(items[8].isEmpty() ? "0" : items[8])/100));
            }
        }
        scan.close();
    } catch (IOException e) {
        System.out.println(e.getMessage());
        System.exit(-1);
    }
}
```