

Entrega Individual 1

Daniel Rodriguez Orozco

Nombre de usuario: drodriguezo

BISOFT-12

Ejercicio 1: Total amount of points.

Problema a resolver:

Obtener el ganador de los partidos de una colección de datos que se nos muestra en un array de strings, calculando los puntos con base a los resultados que se nos da en los ejemplos.

Estrategia:

El kata nos dice las condiciones que se necesitan para poder recibir los puntos a calcular, 3 puntos si el equipo gana, 1 por empate, y 0 si se pierde, con estas condiciones es muy fácil deducir que solo se necesita un "if" que verifique los resultados que se nos da, probando si nuestro equipo tuvo mas puntos, menos o los mismo y dando el resultado a la suma de los puntos totales al final

Código aplicado:

```
no usages
public static int points(String[] games) {
    if (games.length != 10) {
        throw new IllegalArgumentException();
    }

    int TotalPoints = 0;

    for (String match : games) {
        String[] scores = match.split(regex: ":");
        int ourScore = Integer.parseInt(scores[0]);
        int opponentScore = Integer.parseInt(scores[1]);

        if (ourScore > opponentScore) {
            TotalPoints += 3;
        } else if (ourScore == opponentScore) {
            TotalPoints += 1;
        }
    }

    return TotalPoints;
}
```

Prueba del kata:

The screenshot shows a coding challenge interface. On the left, the 'Test Results' panel indicates that the solution passed all tests. The 'Solution' panel in the center contains the following Java code:

```
1 public class TotalPoints {
2
3     public static int points(String[] games) {
4         if (games.length != 10) {
5             throw new IllegalArgumentException();
6         }
7
8         int TotalPoints = 0;
9
10        for (String match : games) {
11            String[] scores = match.split(":");
12            int ourScore = Integer.parseInt(scores[0]);
13            int opponentScore = Integer.parseInt(scores[1]);
14
15            if (ourScore > opponentScore) {
16                TotalPoints += 3;
17            } else if (ourScore == opponentScore) {
18                TotalPoints += 1;
19            }
20        }
21    }
22 }
```

On the right, the 'Sample Tests' panel shows the following Java code:

```
1 import org.junit.Test;
2 import static org.junit.Assert.assertEquals;
3 import org.junit.runners.JUnit4;
4
5 public class SolutionTest {
6     @Test
7     public void basicTests() {
8         assertEquals(0, TotalPoints.points(new String[]
9             {"1:0", "2:0", "3:0", "4:0", "2:1", "3:1", "4:1", "3:2", "4:2", "4:3"}));
10
11         assertEquals(10, TotalPoints.points(new String[]
12             {"1:1", "2:2", "3:3", "4:4", "2:2", "3:3", "4:4", "3:3", "4:4", "4:4"}));
13
14         assertEquals(0, TotalPoints.points(new String[]
```

Conclusión:

Al ser uno de los kata con baja dificultad no hubo mayor problema a la hora de resolverlo, entonces solo con usar el array y los "if" se pudo lograr, aunque en las notas del kata decía que tenía que ser siempre 10 partidos, al principio estaba dudando mucho como implementar eso o si era muy importante, pero al final con un "throwexception" funcionó mejor de lo que esperaba.

Ejercicio 2: Follow that Spy

Problema a resolver:

Nuestra misión es descifrar las rutas que emprenderá Matthew Knight durante cada uno de sus viajes. Nos presentan strings de itinerarios de las rutas que tomo el espía y se nos pide ordenar las rutas a la posible ruta específica que tomo el espía.

Estrategia:

Para este kata, al principio no se entendía muy bien que se pedía o cómo hacerlo, después de un poco de análisis y pruebas, al final los arraylist fueron lo mas efectivo, primero el programa trate de encontrar el inicio de la secuencia buscando cual string solo aparece una vez en el array, si esto retorna falso se crea el inicio y se iguala a la primera pareja del array siendo esta 0. Si el inicio ya se encontró anteriormente, se añaden al array los puntos actuales al, luego se evalúa el tamaño del array temporal y con esos datos se añade la ruta al array principal.

Código aplicado:

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  no usages
5  public class Routes {
6      no usages
7      @
8      public static String findRoutes(String[][] routes) {
9          List<String> trace = new ArrayList<>();
10
11          for (String[] currentPair : routes) {
12              boolean foundStart = false;
13              for (String[] point : routes) {
14                  if (point[1].equals(currentPair[0])) {
15                      foundStart = true;
16                      break;
17                  }
18              }
19
20              if (!foundStart) {
21                  trace.add(currentPair[0]);
22                  trace.add(currentPair[1]);
23              }
24
25              for (String[] route : routes) {
26                  for (String[] currentPair : routes) {
27                      if (currentPair[0].equals(trace.get(trace.size() - 1))) {
28                          trace.add(currentPair[1]);
29                      }
30                  }
31              }
32
33              return String.join(" ", trace);
34          }
35      }
36  }
```

Prueba del kata:

The screenshot shows the LeetCode interface for the 'Follow that Spy' kata. On the left, the 'Test Results' panel shows 'Time: 2095ms', 'Passed: 1', and 'Failed: 0'. Below this, a message states 'You have passed all of the tests! :)'.

The main area displays the 'Solution' code in Java. The code defines a `Routes` class with a `findRoutes` method that takes an array of strings and returns a list of strings. The method iterates through the input array, finding the start of a route and then following it until it reaches the end.

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class Routes {
5     public static String findRoutes(String[] routes) {
6         List<String> trace = new ArrayList<>();
7
8         for (String[] currentPair : routes) {
9
10            boolean foundStart = false;
11            for (String[] point : routes) {
12                if (point[0].equals(currentPair[0])) {
13                    foundStart = true;
14                    break;
15                }
16            }
17
18            if (!foundStart) {
19                trace.add(currentPair[0]);
20                trace.add(currentPair[1]);
21            }
22        }
23    }
24 }
```

Below the solution code, the 'Sample Tests' panel shows the test cases used to verify the solution. The tests are as follows:

```
1 public class FollowThatSpy {
2     @Test
3     public void spyRoutes() {
4         Routes routes = new Routes();
5         assertEquals("MNL", TAG, CEB, TAC, BOR);
6         routes.findRoutes(new String[]){("MNL", "TAG"), ("CEB", "TAC"), ("TAG", "CEB"), ("TAC", "BOR")});
7         assertEquals("Halifax, Montreal, Toronto, Chicago, Winnipeg, Seattle",
8             routes.findRoutes(new String[]){("Chicago", "Winnipeg"), ("Halifax", "Montreal"), ("Montreal", "Toronto"), ("Toronto", "Seattle")});
9     }
10 }
```

The interface also includes a 'Test' button and a 'Submit' button at the bottom right.

Conclusión:

Definitivamente fue mas confuso de lo que la dificultad decía, pero no fue difícil, solo a la hora de tratar de entender, al final usando ArrayLists fue mas sencillo, intente con linked lists pero eso tal vez me confundió más, también encontrar la ruta que solo aparece una vez y determinarla como el inicio fue algo que en retrospectiva parece tan obvio pero si me tomo tiempo pensarlo.

Ejercicio 4: Fun with trees: array to tree

Problema a resolver:

Con una serie de números enteros, implemente una función que cree un árbol binario completo. También se nos pide que todo el árbol este completo menos el nivel final.

Estrategia:

Debido a que el kata ya nos da una guía y base para probar código, ya se nos da un camino más fácil de seguir, primero se crean los arrays en un árbol de nodos y se crea un for loop, y crea un objeto del árbol y lo guarda en el array del árbol de nodos. Este loop pasa sobre la primera mitad de los nodos en la matriz de nodos y para cada nodo en el índice i , establece su hijo izquierdo en el nodo en el índice $2 * i + 1$, y si el índice hijo derecho $2 * i + 2$ está dentro de los límites de la matriz, también establece el hijo derecho.

Código aplicado:

```
no usages
1  public class Solution {
2
3      no usages
4      static TreeNode arrayToTree(int[] array) {
5          if (array == null || array.length == 0) return null;
6
7          TreeNode[] nodes = new TreeNode[array.length];
8
9          for (int i = 0; i < array.length; i++) {
10             nodes[i] = new TreeNode(array[i]);
11         }
12
13         for (int i = 0; i < array.length / 2; i++) {
14             nodes[i].left = nodes[2 * i + 1];
15             if (2 * i + 2 < array.length) {
16                 nodes[i].right = nodes[2 * i + 2];
17             }
18         }
19
20         return nodes[0];
21     }
22 }
```

Prueba del kata:

The screenshot shows a coding challenge interface for the 'array to tree' kata. The top bar indicates the challenge is 'Fun with trees: array to tree' with a 'Skip' button. It shows 246 attempts, 65 solved (94% of 410), and 536 of 2,999 users have solved it. The user 'janitormeir' is logged in. The interface is divided into three main sections: Instructions, Output, and Solution.

Instructions: The 'Test Results' section shows 'Time: 2428ms', 'Passed: 5', and 'Failed: 0'. A 'SolutionTest' section lists several test cases: 'randomBigArray', 'emptyArray', 'randomSmallArray', 'arrayWithSingleElement', and 'arrayWithMultipleElements'. A green message states 'You have passed all of the tests! :)'.

Output: This section is currently empty.

Solution: The 'Solution' section displays the following Java code:

```
1 public class Solution {
2
3     static TreeNode arrayToTree(int[] array) {
4         if (array == null || array.length == 0) return null;
5
6         TreeNode[] nodes = new TreeNode[array.length];
7
8         for (int i = 0; i < array.length; i++) {
9             nodes[i] = new TreeNode(array[i]);
10        }
11
12        for (int i = 0; i < array.length / 2; i++) {
13            nodes[i].left = nodes[2 * i + 1];
14            nodes[i].right = nodes[2 * i + 2];
15        }
16
17        return nodes[0];
18    }
19 }
```

Below the solution code, a message says 'Excellent! You may take your time to refactor/comment your solution. Submit when ready.'

Sample Tests: The 'Sample Tests' section shows the following Java code:

```
1 import org.junit.Test;
2 import static org.hamcrest.CoreMatchers.*;
3 import static org.junit.Assert.assertThat;
4
5 public class SolutionTest {
6
7     @Test
8     public void emptyArray() {
9         TreeNode expected = null;
10        assertThat(Solution.arrayToTree(arrayFrom()), is(expected));
11    }
12
13    @Test
14    public void arrayWithMultipleElements() {
15    }
```

At the bottom of the interface, there are buttons for 'SKIP', 'UNLOCK SOLUTIONS', 'DISCUSS (26)', 'RESET', 'TEST', and 'SUBMIT'.

Conclusión:

Como el kata ya nos dio un código para basarnos y hacer las pruebas, fue más fácil de lo que parecía y al final fue usar los for loops para añadir los nodos. También pensé que había que poner todo el código con el que se nos da en la solución, y siempre me daba error entonces si me preocupé al principio antes de llegar a la conclusión de que solo era la parte que se añadía

Ejercicio 4: Integers: Recreation One

Problema a resolver:

Encuentre todos los números enteros entre m y n (m y n enteros con $1 \leq m \leq n$) tales que la suma de sus divisores al cuadrado sea en sí misma un cuadrado.

Estrategia:

Para este kata, el código recorre cada número en el rango especificado, calcula la suma de los divisores al cuadrado para cada número y verifica si la suma es un cuadrado perfecto. Luego construye una cadena que contiene pares de números [número, suma] donde la suma es un cuadrado perfecto y devuelve esta cadena para el resultado

Código aplicado:

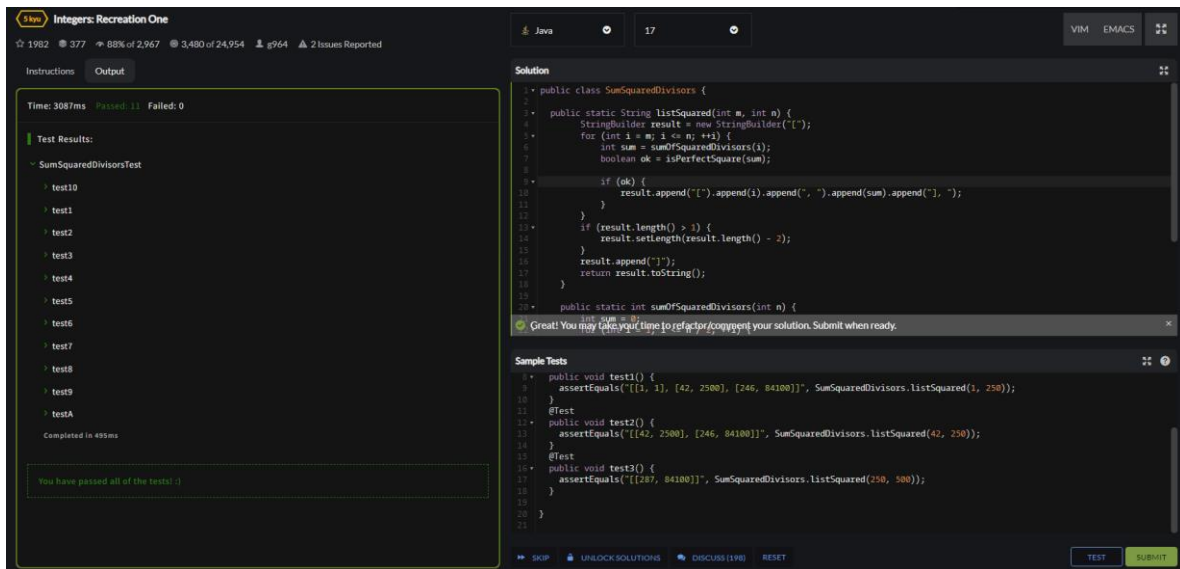
```
no usages
public static String listSquared(int m, int n) {
    StringBuilder result = new StringBuilder("");
    for (int i = m; i <= n; ++i) {
        int sum = sumOfSquaredDivisors(i);
        boolean ok = isPerfectSquare(sum);

        if (ok) {
            result.append("[").append(i).append(", ").append(sum).append("], ");
        }
    }
    if (result.length() > 1) {
        result.setLength(result.length() - 2);
    }
    result.append("]");
    return result.toString();
}

1 usage
public static int sumOfSquaredDivisors(int n) {
    int sum = 0;
    for (int i = 1; i <= n / 2; ++i) {
        if (n % i == 0) {
            sum += i * i;
        }
    }
    return sum + n * n;
}

1 usage
public static boolean isPerfectSquare(int num) {
    int sqrt = (int) Math.sqrt(num);
    return sqrt * sqrt == num;
}
}
```


Prueba Kata:



The screenshot shows a Kata exercise interface for "Integers: Recreation One". The left panel displays test results for the `SumSquaredDivisorsTest` class, listing tests from `test10` to `testA`. A green message at the bottom states "You have passed all of the tests!". The right panel shows the Java solution code for the `SumSquaredDivisors` class, which includes methods `listSquared` and `sumOfSquares`. A notification at the bottom of the code editor says "Great! You may take your time to refactor/cleanup your solution. Submit when ready." The bottom of the interface has buttons for "SKIP", "UNLOCK SOLUTIONS", "DISCUSS (190)", "RESET", "TEST", and "SUBMIT".

```
1 public class SumSquaredDivisors {
2     public static String listSquared(int m, int n) {
3         StringBuilder result = new StringBuilder("");
4         for (int i = m; i <= n; ++i) {
5             int sum = sumOfSquares(i);
6             boolean ok = isPerfectSquare(sum);
7             if (ok) {
8                 result.append(i).append(" ").append(sum).append(" ");
9             }
10            if (result.length() > 1) {
11                result.setLength(result.length() - 2);
12            }
13            result.append("\n");
14        }
15        return result.toString();
16    }
17
18    public static int sumOfSquares(int n) {
19        int sum = 0;
20        for (int i = 1; i <= n; ++i) {
21            sum += i * i;
22        }
23        return sum;
24    }
25}
```

Conclusión:

Para este kata al principio la descripción de lo que se pedía los ejemplos no daban mucho sentido pero con los ejemplos que nos dio se pudieron hacer las pruebas y mejorar el código, usando varios métodos siento que el código puede ser mas optimizado a usar solo uno o dos métodos.

Ejercicio 5: Line Safari - Is that a line?

Problema a resolver:

Se nos da un grid que siempre va a tener dos puntos x, se nos pide determinar si los mapas que unen estos dos puntos x son validos dependiendo de la línea que las conecta.

Estrategia:

Este problema resulto ser muy complicado, así que tuve que recurrir a el foro mismo de codewars y documentación aparte, la solución encontrada básicamente en un mapa se usan las coordenadas U R D L donde el código va recorriendo los pasos que se nos da en las pruebas hasta llegar al final, usando cases y returns para completar lo dicho, y se da un false cuando se llega un camino sin salida.

Código aplicado:

```
1 function line(grid) {
2   let is = (a, b) => b.includes(a),
3       moves = [
4     [-1, 0],
5     [0, 1],
6     [1, 0],
7     [0, -1],
8   ],
9   g;
10
11   let trav = ([r, c], dir) => {
12     g[r][c] = " ";
13     let X = moves.map(([R, C]) => [R + r, C + c]);
14     let P = X.map(([R, C]) => (g[R] && g[R][C]) || " ").map((d, i) =>
15       d != "-" || !["i"] ? d : " "
16     ),
17     [U, R, D, L] = P; // Path pieces
18     let go = (d) => trav(X["URDL".indexOf(d)], d);
19
20     switch (grid[r][c]) {
21       case "X":
22         return (
23           (dir && g.every((w) => w.every((q) => q == " "))) ||
24           (P.join``.match(/S/g) || []).length == 1 &&
25           go("URDL"[P.findIndex((l) => l != " ")])
26         );
27       case "|":
28         return is(dir, "UD") && go(dir);
29       case "-":
30         return is(dir, "RL") && go(dir);
31       case "+":
32         return is(dir, "RL")
33           ? (is(U, "+X") && is(D, "- ") && go("U")) ||
34             (is(D, "+X") && is(U, "- ") && go("D"))
35           : (is(R, "-X") && is(L, "| ") && go("R")) ||
36             (is(L, "-X") && is(R, "| ") && go("L"));
37     }
38     return false;
39   };
40   return grid.some((w, r) =>
41     [...w].some(
42       (s, c) => s == "X" && ((g = grid.map((y) => [...y])), trav([r, c]))
43     )
44   );
45 }
```

Prueba kata:

The screenshot shows a coding challenge interface for "Line Safari - Is that a line?". The interface is divided into several sections:

- Header:** Shows the challenge title "Line Safari - Is that a line?", a progress bar (393/73, 86% of 228, 182 of 745), and the user "dinglemouse".
- Instructions:** A section on the left with tabs for "Instructions", "Output", and "Past Solutions".
- Test Results:** A section on the left showing "Time: 666ms", "Passed: 8/7", and "Failed: 0". It also lists "Solution Tests" with "Good", "Bad", "More tests", and "Random tests". A green box at the bottom of this section says "You have passed all of the tests!".
- Solution:** A section on the right showing the JavaScript code for the solution. The code is as follows:

```
1 function line(grid) {
2   let is = (a, b) => b.includes(a),
3     moves = [
4       [-1, 0],
5       [0, 1],
6       [1, 0],
7       [0, -1],
8     ],
9     is;
10
11   let trav = ((r, c), dir) => {
12     let i = 0;
13     let X = moves.map(([[R, C]] => [R + r, C + c]));
14     let P = X.map(([[R, C]] => {g[R] && g[R][C] ? " " : ""})).map((d, i) =>
15       d += "-|-|"[i] ? d : " "
16     );
17     [0, R, 0, L] = P; // Path pieces
18     let go = (d) => trav(X[0].indexOf(d), d);
19
20     switch (grid[r][c]) {
21       case "X":
22         return go(is);
23       case " ":
24         return go(is);
25     }
26   };
27
28   return is;
29 }
```
- Sample Tests:** A section on the right showing the test cases. The code is as follows:

```
1 describe("Example tests", function() {
2
3   // "Good" examples from the Kata description.
4   // -----
5
6   describe("Good", function() {
7
8     it("ex good #1", function(){
9       var grid = preloaded.makedGrid([
10         "X-----X",
11         " ",
12         " "
13       ]);
14     });
15   });
16 });
```
- Footer:** A section at the bottom with buttons for "SKIP", "VIEW SOLUTIONS", "DISCUSS (33)", "RESET", "TEST", and "SUBMIT".

Conclusión:

Primero que nada, al intentar completarlo en java, siempre daba todo en falso, no se si fue un error mío o del kata pero habían varias personas en el foro del kata que tuvieron el problema, entonces como se puede observar intente hacerlo en javascript, hasta tuve que abrir el visual studio code que no usaba desde fundamentos, pero al final traduciendo el codito de java a js pudo funcionar, aunque este kata si fue uno de los mas difíciles que he intentado.