

Informe de Laboratorio 09

Tema: Definición de Clases de Usuario Clase Soldado

Nota

Estudiante	Escuela	Asignatura
Julio Rubén Chura Acabana jchuraaca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	F. de Programación 2 Semestre: I Código: 20230472

Laboratorio	Tema	Duración
09	Definición de Clases de Usuario Clase Soldado	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 6 Noviembre 2023	Al 13 Noviembre 2023

1. Tarea

- Crear 3 constructores sobrecargados.
- La actitud puede ser defensiva, ofensiva, fuga. Dicha actitud varía cuando el soldado defiende, ataca o huye respectivamente.
- Al atacar el soldado avanza, al avanzar aumenta su velocidad en 1. Al defender el soldado se para. Al huir aumenta su velocidad en 2. Al retroceder, si su velocidad es mayor que 0, entonces primero para y su actitud es defensiva, y si su velocidad es 0 entonces disminuirá a valores negativos. Al ser atacado su vida actual disminuye y puede llegar incluso a morir.
- Crear los atributos y métodos extra que considere necesarios.
- Tendrá 2 Ejércitos. Usar la estructura de datos que considere más adecuada. Inicializar el tablero con n soldados aleatorios entre 1 y 10 para cada Ejército. Cada soldado tendrá un nombre autogenerado: Soldado0X1, Soldado1X1, etc., un valor de puntos de vida autogenerado aleatoriamente [1..5], la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado). Nivel de ataque y de defensa son aleatorios [1..5]. Se debe mostrar el tablero con todos los soldados creados (usar caracteres) y distinguir los de un ejército de los del otro ejército. Además de los datos del Soldado con mayor vida de cada ejército, el promedio de puntos de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando 2 diferentes algoritmos de ordenamiento. Finalmente, que muestre qué ejército ganará la batalla (indicar la métrica usada para decidir al ganador de la batalla). Hacerlo un programa iterativo.

- Crear el diagrama de clases UML completo
- Usted debe realizar varios commits y al término de la actividad deberá realizar un informe.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows
- Vim 9.0
- OpenJDK 64-Bits 20.0.2
- Git 2.42.0
- Cuenta en GitHub con el correo institucional
- Definición de Clases de Usuario Clase Soldado

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/JulioChura/fp2-23b.git>
- URL para el laboratorio 01 en el Repositorio GitHub.
- <https://github.com/JulioChura/fp2-23b/tree/main/fase02/lab09>

4. Actividades con el repositorio GitHub

4.1. Inicialización del espacio de trabajo

Listing 1: Inicializando el espacio de trabajo

```
mkdir lab09
cd ..
cd lab07
Copy-Item "Soldier.java" -Destination "..\lab09"
Copy-Item "VideoJuego4.java" -Destination "..\lab09\VideoJuego6.java"
cd ..
cd lab09
vim VideoJuego6.java
```

Listing 2: Commit: 24b299f6d4b3ebd17971adadc7057ab72724b41c

```
git add VideoJuego6.java
git commit -m "Se copia VideoJuego4.java del lab07 al lab09"
git push -u origin main
```

Listing 3: Commit: de2a4e5f1787deaab8683e2f4ca01a0d3922cd9e

```
git add VideoJuego4.java
git commit -m "Se copia Soldier.java del lab07 al lab09"
git push -u origin main
```

aaaaaaaaaaaaaaaa

4.2. Funciones ya trabajadas en laboratorios pasados

```
64 public static ArrayList<ArrayList<Soldier>> generateArmy() {
65     ArrayList<ArrayList<Soldier>> army = new ArrayList<ArrayList<Soldier>>(10);
66     Random random = new Random();
67     int amount = random.nextInt(10) + 1;
68     int n = 0;
69     for (int i = 0; i < 10; i++) {
70         army.add(new ArrayList<>(Collections.nCopies(10, null)));
71     }
72     do {
73         int row = random.nextInt(10);
74         int column = random.nextInt(10);
75         if (army.get(row).get(column) == null) {
76             String name = "Soldier" + row + "x" + column;
77             int lifePoints = random.nextInt(5) + 1;
78             Soldier sol = new Soldier();
79             sol.setLifePoints(lifePoints);
80             sol.setName(name);
81             sol.setColumn(column);
82             sol.setRow(row);
83             army.get(row).set(column, sol);
84             n++;
85         }
86     } while (n < amount);
87     return army;
88 }
```

- Método generateArmy(): Crea un arreglo bidimensional de 10 filas y 10 columnas con el fin de cubrir la misma cantidad de casillas del tablero, por lo que habrán posiciones que quedarán vacías ya que la cantidad de elementos del arreglo será un número aleatoria que oscila entre 1 a 10. También se genera el nombre de cada Soldier y sus puntos de vida de forma aleatoria. Este arreglo nos servirá más que nada para imprimir el tablero de una forma más sencilla.
- Como sabemos, el ArrayList es una estructura de datos compacta, es decir puede recibir n cantidad de elementos y esos elementos se acomodarán, por lo que no hay posiciones con elementos vacíos, pero es posible inicializar algunas posiciones con ningún elemento, en nuestro caso null, y eso hace el primer for, se encarga de llenar el ArrayList con null.

```

90     public static ArrayList<ArrayList<Soldier>> generateArmyB(ArrayList<ArrayList<Soldier>> a) {
91         ArrayList<ArrayList<Soldier>> army = new ArrayList<ArrayList<Soldier>>(10);
92         Random random = new Random();
93         int amount = random.nextInt(10) + 1;
94         int n = 0;
95
96         for (int i = 0; i < 10; i++) {
97             army.add(new ArrayList<>(Collections.nCopies(10, null)));
98         }
99         do {
100             int row = random.nextInt(10);
101             int column = random.nextInt(10);
102             if (army.get(row).get(column) == null && a.get(row).get(column) == null) {
103                 String name = "Soldier" + row + "x" + column;
104                 int lifePoints = random.nextInt(5) + 1;
105
106                 Soldier sol = new Soldier();
107
108                 sol.setLifePoints(lifePoints);
109                 sol.setName(name);
110                 sol.setColumn(column);
111                 sol.setRow(row);
112                 army.get(row).set(column, sol);
113                 n++;
114             }
115         } while (n < amount);
116         return army;
117     }

```

- Método generateArmyB: La razón de la creación de este método se debe a que hay casos en los que la cantidad de fichas del ejército B que están ubicadas en el tablero no coincide en cantidad con lo que se genera. Básicamente este método es igual que generateArmy, solo que este recibe un parámetro de tipo ArrayList bidimensional- Soldier que se usará para verificar que se generen los soldados del ejército B sin que haya cruces con los del ejército A. Lo único que se añade al método generateArmy es a.get(row).get(column) == null

```

52     public static ArrayList<Soldier> arrayListUnidimensional(ArrayList<ArrayList<Soldier>> s) {
53         ArrayList<Soldier> armyUni = new ArrayList<Soldier>();
54         for (int i = 0; i < s.size(); i++) {
55             for (int j = 0; j < s.get(i).size(); j++) {
56                 if (s.get(i).get(j) != null) {
57                     armyUni.add(s.get(i).get(j));
58                 }
59             }
60         }
61         return armyUni;
62     }

```

- arrayListUnidimensional: A pesar de ya contar con dos ArrayList bidimensionales, se opta por transformarlos en ArrayList unidimensionales ya que nos facilitará trabajar con los demás métodos que la práctica de laboratorio solicita. En el main se hace la creación de dos ArrayList unidimensionales tanto para el ejército A y B.

```
1  InsertionSort(arr[], n)
2      for i = 1 to n-1
3          key = arr[i]
4          j = i-1
5          while j >= 0 and arr[j] > key
6              arr[j+1] = arr[j]
7              j = j-1
8          arr[j+1] = key |
```

```
153  public static Soldier longerLife(ArrayList<Soldier> s) {
154      int n = s.size();
155      for (int i = 1; i < n; i++) {
156          Soldier key = s.get(i);
157          int j = i - 1;
158          while (j >= 0 && s.get(j).getLifePoints() > key.getLifePoints()) {
159              s.set(j + 1, s.get(j));
160              j--;
161          }
162          s.set(j + 1, key);
163      }
164      return s.get(s.size() - 1);
165  }
```

- longerLife(): Se adapta el pseudocódigo de insertion y se le da la forma para que se aplique en un ArrayList. Lo que hace el método es ordenar de menor a mayor, por lo que último elemento es el mayor. El método devuelve al Soldier que se encuentra en esa posición.
- Se muestra el pseudocódigo, pero aplicado a un arreglo Estándar. A pesar de ser otra estructura, la idea es la misma

```
167  public static void showByCreation(ArrayList<Soldier> sol) {
168      for (Soldier n : sol) {
169          System.out.println(n);
170      }
171  }
```

- showByCreation(): Básicamente imprime los datos de un ArrayList unidimensional de tipo Soldier.

```
173  public static int totalLife(ArrayList<Soldier> sol) {
174      int addition = 0;
175      for (Soldier n : sol) {
176          addition = addition + n.getLifePoints();
177      }
178      return addition;
179  }
```

- `totalLife()`: El método devuelve los puntos de vida total del ejército. Se decide hacer el método con retorno para poder usar esos valores para determinar al ganador y calcular el promedio.

```

183     public static void orderByPower(ArrayList<Soldier> sol) {
184         boolean swapped;
185         Soldier temp;
186         for (int i = 0; i < sol.size() - 1; i++) {
187             swapped = false;
188             for (int j = 0; j < sol.size() - 1 - i; j++) {
189                 temp = sol.get(j);
190                 sol.set(j, sol.get(j + 1));
191                 sol.set(j + 1, temp);
192                 swapped = true;
193             }
194             if (swapped == false) {
195                 break;
196             }
197         }
198         for (Soldier n : sol) {
199             System.out.println(n);
200         }
201     }

```

Procedimiento `bubbleSort`(entero `arr[]`, entero `n`)

Inicio

 entero `i`, `j`

 booleano `swapped`

 Para `i = 0` Hasta `n - 1` Hacer

`swapped = falso`

 Para `j = 0` Hasta `n - i - 1` Hacer

 Si `arr[j] > arr[j + 1]` Entonces

 intercambiar(`arr[j]`, `arr[j + 1]`)

`swapped = verdadero`

 Fin Si

 Fin Para

 Si `swapped == falso` Entonces

 Romper

 Fin Si

 Fin Para

Fin

- `orderByPower()`: El método ordena el `ArrayList` tomando como criterio los puntos de Vida. No se retorna nada

- Se muestra el pseudocódigo, pero aplicado a un arreglo Estándar. A pesar de ser otra estructura, la idea es la misma.

```

203     public static void theWinner(int a, int b) {
204         if (a < b) {
205             System.out.println("El ganador es el equipo B");
206             System.out.println("Ventaja de " + (b - a) + " puntos de vida");
207         } else if (a > b) {
208             System.out.println("El ganador es el equipo A");
209             System.out.println("Ventaja de " + (a - b) + " puntos de vida");
210         } else {
211             System.out.println("Fue un empate");
212         }
213     }

```

- the Winner(): Este método recibe como parámetros dos enteros los cuales representan los puntos de vida total de cada ejército. Con ayuda de una estructura condicional se imprime un mensaje indicando al ganador o si hubo empate.

aaaaaaaaaaaaaaaaaaaa

4.3. Desarrollo de nuevas funcionalidades

Listing 4: Se añaden atributos a la clase Soldier y se crean constructores

vim Soldier.java

```

1 public class Soldier {
2
3     private String name;
4     private int row;
5     private int column;
6     private int attackLevel;
7     private int defenseLevel;
8     private int lifePoints;
9     private int actualLife;
10    private int speed;
11    private String attitude;
12    private boolean current;
13
14    public Soldier(String name, int row, int column, int attackLevel, int defenseLevel, int lifePoints) {
15        this(name, row, column, attackLevel, defenseLevel, lifePoints, lifePoints, 0, "Reposo", true);
16    }
17
18    public Soldier(String name, int attackLevel, int defenseLevel, int lifePoints, int speed, String attitude,
19        boolean current) {
20        this(name, 0, 0, attackLevel, defenseLevel, lifePoints, lifePoints, speed, attitude, current);
21    }
22
23    public Soldier(String name, int row, int column, int attackLevel, int defenseLevel, int lifePoints, int actualLife,
24        int speed, String attitude, boolean current) {
25        this.name = name;
26        this.row = row;
27        this.column = column;
28        this.attackLevel = attackLevel;
29        this.defenseLevel = defenseLevel;
30        this.lifePoints = lifePoints;
31        this.actualLife = actualLife;
32        this.speed = speed;
33        this.attitude = attitude;
34        this.current = current;
35    }
36
37    ..

```

- Por especificaciones del problema se aumentan los atributos `attackLevel` (nivel de ataque), `defenseLevel` (nivel de defensa), `actualLife` (la vida actual después de un combate por ejemplo), `speed` (la velocidad), `attitude` (si está en modo de ataque, huida o defensa), `current` (si el soldado está vivo).
- En cuanto a los constructores. el de la línea 23, se añaden como parámetro todos los atributos. Otro constructor que lo usaremos en el main el cual contiene todo a excepción de `actualLife`, `speed`, `current`, `attitude`. Por último, se crea un constructor que no se usará mucho, sin embargo, por especificaciones del problema se crea.

Listing 5: Commit: 73ad0ec83d4c273a3169b0c15ec68c6e4dd9d2a2

```
git add Soldier.java
git commit -m "Se aumentan atributos nuevos a la clase y tambien los constructores"
git push -u origin main
```

Listing 6: Se crea un método que nos muestre los datos de un Soldier

vim Soldier.java

```
53 public String toString() {
54     return "Soldier [name=" + name + ", row=" + row + ", column=" + column + ", attackLevel=" + attackLevel
55         + ", defenseLevel=" + defenseLevel + ", lifePoints=" + lifePoints + ", actualLife=" + actualLife
56         + ", speed=" + speed + ", attitude=" + attitude + ", current=" + current + "]";
57 }
```

- El método `toString` muestra los datos del soldier.

Listing 7: Commit: 4ca2b91b8555fdaa32a7abd4c96c4803e64602dd7

```
git add Soldier.java
git commit -m "Se aumentan atributos a toString"
git push -u origin main
```

Listing 8: Se crea el metodo avanzar

vim Soldier.java

```
37 public void advance() {
38     speed++;
39 }
```

- Las instrucciones que ofrece la prácticas son muy breves, por lo que se limitó en seguir la instrucción que se nos da para la creación de este método. El método lo que hace es aumentar la velocidad en uno.

Listing 9: Commit: f024c49ff16d97ebf6611820a92ef6af6587f67c

```
git add Soldier.java
git commit -m "Se crea el metodo advance"
git push -u origin main
```


Listing 10: Se crea el metodo atacar

```
vim Soldier.java
```

```
41     public void attack() {  
42         attitude = "offensive";  
43         advance();  
44     }
```

- Según las instrucciones de la práctica, este método cambiará la actitud del Soldier a modo ofensivo, además de que la velocidad aumentará.

Listing 11: Commit: cf63fadd8443e7dcdd2cb62635b177373842e61d

```
git add Soldier.java  
git commit -m "Se cambian el nombre de la actitud del metodo attack"  
git push -u origin main
```

Listing 12: Se crea el metodo defender

```
vim Soldier.java
```

```
46     public void defend() {  
47         attitude = "defensive";  
48         speed = 0;  
49     }
```

- Según las instrucciones de la práctica, este método cambiará la actitud del Soldier a modo defensivo, además de que la velocidad será igual a 0.

Listing 13: Commit: 6b573ae3b1c0241a758325f55f311e19e76361f3

```
git add Soldier.java  
git commit -m "Se crea el metodo defend"  
git push -u origin main
```

Listing 14: Se crea el metodo retroceder

```
vim Soldier.java
```

```
51     public void back() {  
52         if (speed > 0) {  
53             speed = 0;  
54             attitude = "defensive";  
55         } else if (speed == 0) {  
56             speed--;  
57         }  
58     }
```

- Según las instrucciones de la práctica, este método cambiará la actitud del Soldier a modo defensivo. Si la velocidad es mayor que 0, el valor se restablece a 0, y si es igual a cero, la velocidad disminuye.

Listing 15: Commit: flde159aef9b2d9484f6fcfc3a72032764cc36ab

```
git add Soldier.java
git commit -m "Metodo back culminado"
git push -u origin main
```

Listing 16: Se crea el metodo morir

```
vim Soldier.java
```

```
60      public void die() {
61          if (actualLife < 0) {
62              current = false;
63          } else {
64              current = true;
65          }
66      }
```

- Según las instrucciones de la práctica, un soldado puede llegar a morir cuando su estado de vida actual tiene un número negativo.

Listing 17: Commit: 878a442ed1b50d5bf9c8da86d2f096c1128784fd

```
git add Soldier.java
git commit -m "Se termina el metodo die"
git push -u origin main
```

Listing 18: Se crea el metodo ser atacado

```
vim Soldier.java
```

```
68      public void beAttacked(int damage) {
69          this.actualLife = this.actualLife - damage;
70          die();
71      }
```

- Según las instrucciones de la práctica, un soldado cuando es atacado, su vida actual disminuye y puede llegar a morir.

Listing 19: Commit: 592d758a6eabb49b6bc92d420a6c112081203c4d

```
git add Soldier.java
git commit -m "Se culmina el metodo beAtacked"
git push -u origin main
```

```
vim Soldier.java
```

```
vim Soldier.java
```

```
vim VideoJuego4.java
```

```

116 public static void myBoard(ArrayList<ArrayList<Soldier>> a, ArrayList<ArrayList<Soldier>> b) {
117     String[][] tablero = new String[10][10];
118     for (int i = 0; i < tablero.length; i++) {
119         for (int j = 0; j < tablero[i].length; j++) {
120             tablero[i][j] = "|_|";
121         }
122     }
123     for (int i = 0; i < a.size(); i++) {
124         for (int j = 0; j < a.get(i).size(); j++) {
125             if (a.get(i).get(j) != null) {
126                 String strA = "|_" + "a" + a.get(i).get(j).getLifePoints() + "_|";
127                 tablero[i][j] = strA;
128             }
129         }
130     }
131     for (int i = 0; i < b.size(); i++) {
132         for (int j = 0; j < b.get(i).size(); j++) {
133             if (b.get(i).get(j) != null && tablero[i][j] != "s") {
134                 String strB = "|_" + "b" + b.get(i).get(j).getLifePoints() + "_|";
135                 tablero[i][j] = strB;
136             }
137         }
138     }
139     System.out.print("  A    B    C    D    E    F    G    H    I    J \n");
140     for (int i = 0; i < tablero.length; i++) {
141         System.out.printf("%2d", (i + 1));
142         for (int j = 0; j < tablero[i].length; j++) {
143             System.out.print(tablero[i][j]);
144         }
145         System.out.println();
146     }
147 }

```

- Se aumenta un subguión y la barra vertical tanto en la línea 126 y 134 para que se impriman las casillas donde se ubican los Soldier de forma armónica.

Listing 25: Compilando y probando el tablero

```

javac VideoJuego4.java
java VideoJuego4
oooooooooooooooooooo FASE 1 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando estadísticas de cada ejercito

Mostrando soldados por orden de creacion
DATOS DEL DEL EJERCITO A
Soldier [name=Soldier1x5, lifePoints=1, row=2, column=6]
Soldier [name=Soldier3x3, lifePoints=5, row=4, column=4]
Soldier [name=Soldier5x1, lifePoints=3, row=6, column=2]
Soldier [name=Soldier6x8, lifePoints=5, row=7, column=9]
Soldier [name=Soldier7x8, lifePoints=1, row=8, column=9]
Soldier [name=Soldier9x1, lifePoints=1, row=10, column=2]
Mayor vida en A: Soldier [name=Soldier6x8, lifePoints=5, row=7, column=9]
El total de vida del ejercito A es: 16
El promedio de vida del ejercito A es: 2.6666666666666665
Mostrando soldados por ranking de poder de A
Soldier [name=Soldier6x8, lifePoints=5, row=7, column=9]
Soldier [name=Soldier3x3, lifePoints=5, row=4, column=4]
Soldier [name=Soldier5x1, lifePoints=3, row=6, column=2]
Soldier [name=Soldier9x1, lifePoints=1, row=10, column=2]
Soldier [name=Soldier7x8, lifePoints=1, row=8, column=9]
Soldier [name=Soldier1x5, lifePoints=1, row=2, column=6]

DATOS DEL EJERCITO B

```

```
Soldier [name=Soldier3x6, lifePoints=1, row=4, column=7]
Mayor vida en B: Soldier [name=Soldier3x6, lifePoints=1, row=4, column=7]
El total de vida del ejercito B es: 1
El promedio de vida del ejercito B es: 1.0
Mostrando soldados por ranking de poder de B
Soldier [name=Soldier3x6, lifePoints=1, row=4, column=7]
```

```
oooooooooooooooooooo FASE 2 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando el tablero de juego
  A   B   C   D   E   F   G   H   I   J
1 |___| |___| |___| |___| |___| |___| |___| |___| |___|
2 |___| |___| |___| |___| |___| |a1_| |___| |___| |___|
3 |___| |___| |___| |___| |___| |___| |___| |___| |___|
4 |___| |___| |___| |a5_| |___| |___| |b1_| |___| |___|
5 |___| |___| |___| |___| |___| |___| |___| |___| |___|
6 |___| |a3_| |___| |___| |___| |___| |___| |___| |___|
7 |___| |___| |___| |___| |___| |___| |___| |___| |a5_|
8 |___| |___| |___| |___| |___| |___| |___| |___| |a1_|
9 |___| |___| |___| |___| |___| |___| |___| |___| |___|
10|___| |a1_| |___| |___| |___| |___| |___| |___| |___|
```

```
+++++ FASE 3 DE LA CONTIENDA +++++
El ganador se determina en base a los puntos de vida total
Enfrentamiento
El ganador es el equipo A
Ventaja de 15 puntos de vida
```

Listing 26: Commit: 8a5f07948da6ece26389a11919f8118d45bcea51

```
git add VideoJuego4.java
git commit -m "El tablero fue mejorado"
git push -u origin main
```

Listing 27: Se crea un método que valida la respuesta del jugador

```
vim VideoJuego4.java
```

```

212     public static boolean validation() {
213         Scanner sc = new Scanner(System.in);
214         do {
215             System.out.println("¿Desea jugar una ronda?(si/no)");
216             String answer = sc.next();
217             if (answer.equalsIgnoreCase("Si")) {
218                 return true;
219             } else if (answer.equalsIgnoreCase("No")) {
220                 return false;
221             } else {
222                 System.out.println("Respuesta no admisible");
223             }
224         } while (true);
225     }
226 }

```

- Se crea este método para poder implementar más adelante un ciclo while en el main el cual permitirá que se juegue la cantidad de veces que el usuario deseé.
- En el método se usa el loop do while y una estructura condicional la que retornará un boolean de acuerdo a la respuesta del usuario y para que no hayan errores en las respuestas se usa el método equalsIgnoreCase(Str str) la cual ignora mayúsculas o minúsculas. Además de que en caso se ingrese una respuesta que no sea "si." o "no" se imprimirá el mensaje de Respuesta no admisible y se repetirá el ciclo hasta que se ingresen los valores correctos.

Listing 28: Commit: 47116e7456c1488e24e11bcd3518f2be3ec47bed

```

git add VideoJuego4.java
git commit -m "Se mejora el metodo que valida la respuesta del jugador"
git push -u origin main

```

Listing 29: Se implementa el loop While en el main para poder repetir una cantidad n de veces

```
vim VideoJuego4.java
```

```

8 public class VideoJuego4 {
9     public static void main(String[] args) {
10
11         while (validation()) {
12             ArrayList<ArrayList<Soldier>> armyA = generateArmy();
13             ArrayList<ArrayList<Soldier>> armyB = generateArmyB(armyA);
14             ArrayList<Soldier> armyAU = arrayListUnidimensional(armyA);
15             ArrayList<Soldier> armyBU = arrayListUnidimensional(armyB);

```

- No hay grandes cambios en el main porque solo se mete todo lo que ya estaba en el loop (ver línea 11 de la imagen)
- En la condición del while se usa el método validation() para que el loop pueda funcionar de forma correcta sin causar un bucle infinito

Listing 30: Compilando y probando

```
javac VideoJuego4.java
java VideoJuego4
Desea jugar una ronda?(si/no)
si
oooooooooooooooooooo FASE 1 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando estadísticas de cada ejercito

Mostrando soldados por orden de creacion
DATOS DEL DEL EJERCITO A
Soldier [name=Soldier0x4, lifePoints=2, row=1, column=5]
Soldier [name=Soldier0x8, lifePoints=1, row=1, column=9]
Soldier [name=Soldier0x9, lifePoints=4, row=1, column=10]
Soldier [name=Soldier1x7, lifePoints=2, row=2, column=8]
Soldier [name=Soldier2x1, lifePoints=3, row=3, column=2]
Soldier [name=Soldier2x7, lifePoints=4, row=3, column=8]
Soldier [name=Soldier3x9, lifePoints=5, row=4, column=10]
Soldier [name=Soldier5x8, lifePoints=1, row=6, column=9]
Soldier [name=Soldier6x3, lifePoints=3, row=7, column=4]
Soldier [name=Soldier8x9, lifePoints=4, row=9, column=10]
Mayor vida en A: Soldier [name=Soldier3x9, lifePoints=5, row=4, column=10]
El total de vida del ejercito A es: 29
El promedio de vida del ejercito A es: 2.9
Mostrando soldados por ranking de poder de A
Soldier [name=Soldier3x9, lifePoints=5, row=4, column=10]
Soldier [name=Soldier8x9, lifePoints=4, row=9, column=10]
Soldier [name=Soldier2x7, lifePoints=4, row=3, column=8]
Soldier [name=Soldier0x9, lifePoints=4, row=1, column=10]
Soldier [name=Soldier6x3, lifePoints=3, row=7, column=4]
Soldier [name=Soldier2x1, lifePoints=3, row=3, column=2]
Soldier [name=Soldier1x7, lifePoints=2, row=2, column=8]
Soldier [name=Soldier0x4, lifePoints=2, row=1, column=5]
Soldier [name=Soldier5x8, lifePoints=1, row=6, column=9]
Soldier [name=Soldier0x8, lifePoints=1, row=1, column=9]

DATOS DEL EJRCITO B
Soldier [name=Soldier0x5, lifePoints=2, row=1, column=6]
Soldier [name=Soldier1x3, lifePoints=5, row=2, column=4]
Soldier [name=Soldier3x8, lifePoints=1, row=4, column=9]
Mayor vida en B: Soldier [name=Soldier1x3, lifePoints=5, row=2, column=4]
El total de vida del ejercito B es: 8
El promedio de vida del ejercito B es: 2.6666666666666665
Mostrando soldados por ranking de poder de B
Soldier [name=Soldier1x3, lifePoints=5, row=2, column=4]
Soldier [name=Soldier0x5, lifePoints=2, row=1, column=6]
Soldier [name=Soldier3x8, lifePoints=1, row=4, column=9]

oooooooooooooooooooo FASE 2 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando el tablero de juego
  A    B    C    D    E    F    G    H    I    J
1 |___||___||___||___||_a2_|_b2_|_||_|_a1_|_a4_|
2 |___||___||___||_b5_|_||_|_||_|_a2_|_||_|_
3 |___||_a3_|_||_|_||_|_||_|_||_|_a4_|_||_|_
4 |___||___||___||___||___||___||___||___||_b1_|_a5_|
5 |___||___||___||___||___||___||___||___||___||___|
6 |___||___||___||___||___||___||___||___||_a1_|_||_|_
```



```

7 |_____|_____|_____|_a3_|_____|_____|_____|_____|_____|
8 |_____|_____|_____|_____|_____|_____|_____|_____|_____|
9 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_a4_|
10|_____|_____|_____|_____|_____|_____|_____|_____|_____|

```

```

+++++ FASE 3 DE LA CONTIENDA +++++
El ganador se determina en base a los puntos de vida total
Enfrentamiento
El ganador es el equipo A
Ventaja de 21 puntos de vida
Desea jugar una ronda?(si/no)
7
Respuesta no admisible
Desea jugar una ronda?(si/no)
no

```

Listing 31: Commit: 41ffd9ca2d4934c86227d650940d5a93865bc037

```

git add VideoJuego3.java
git commit -m "Se crea el loop que permite jugar n cantidad de veces"
git push -u origin main

```

Listing 32: Se implementa un método que realiza una búsqueda Binaria según el nombre

```
vim VideoJuego4.java
```

```

1 function busqueda_binaria (datos, valor_busqueda):
2     inicio = 0
3     fin = longitud (datos) - 1
4     mientras inicio <= fin:
5         medio = (inicio + fin) // 2
6         si datos [medio] == valor_busqueda:
7             devolver medio
8         sino si datos [medio] < valor_busqueda:
9             inicio = medio + 1
10        sino:
11            fin = medio - 1
12    devolver -1

```

- Este es el pseudocódigo general de la búsqueda binaria.

```

237     public static void binarySearchByName(ArrayList<Soldier> armyA, String name) {
238         Collections.sort(armyA, new Comparator<Soldier>() {
239             public int compare(Soldier soldier1, Soldier soldier2) {
240                 return soldier1.getName().compareTo(soldier2.getName());
241             }
242         });
243         int high = armyA.size() - 1;
244         int low = 0;
245         while (low <= high) {
246             int mid = (high + low) / 2;
247             Soldier soldier = armyA.get(mid);
248             if (name.equalsIgnoreCase(soldier.getName())) {
249                 System.out.println("Se ha encontrado: " + soldier);
250                 return;
251             } else if (name.compareTo(soldier.getName()) < 0) {
252                 high = mid - 1;
253             } else {
254                 low = mid + 1;
255             }
256         }
257         System.out.println("No fue encontrado");
258     }

```

- Para una mejor presentación del código, se decide por solo limitarse a mostrar mensajes al usuario y colocar los métodos en el main, esa es la razón por la que se decide hacer este método sin retorno.
- Pese a contar con un método de ordenamiento, ese es de tipo void, así que para ordenar este arreglo y hacer la búsqueda binaria, se tendrá que hacerlo en el mismo método, por lo que se hará uso del método Collections.sort del cual se hará una personalización en el parámetro de Comparator haciendo que el criterio sean los nombres de cada Soldier.

Listing 33: Compilando y probando

```

javac VideoJuego4.java
java VideoJuego4
Desea jugar una ronda?(si/no)
si
oooooooooooooooooooo FASE 1 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando estadísticas de cada ejercito

Mostrando soldados por orden de creacion
DATOS DEL DEL EJERCITO A
Soldier [name=Soldier0x1, lifePoints=3, row=1, column=2]
Soldier [name=Soldier1x4, lifePoints=4, row=2, column=5]
Soldier [name=Soldier2x9, lifePoints=1, row=3, column=10]
Soldier [name=Soldier5x6, lifePoints=3, row=6, column=7]
Soldier [name=Soldier7x4, lifePoints=5, row=8, column=5]
Soldier [name=Soldier9x0, lifePoints=5, row=10, column=1]
Soldier [name=Soldier9x4, lifePoints=5, row=10, column=5]
Mayor vida en A: Soldier [name=Soldier9x4, lifePoints=5, row=10, column=5]
El total de vida del ejercito A es: 26
El promedio de vida del ejercito A es: 3.7142857142857144
Mostrando soldados por ranking de poder de A
Soldier [name=Soldier9x4, lifePoints=5, row=10, column=5]
Soldier [name=Soldier9x0, lifePoints=5, row=10, column=1]
Soldier [name=Soldier7x4, lifePoints=5, row=8, column=5]

```

```
Soldier [name=Soldier1x4, lifePoints=4, row=2, column=5]
Soldier [name=Soldier5x6, lifePoints=3, row=6, column=7]
Soldier [name=Soldier0x1, lifePoints=3, row=1, column=2]
Soldier [name=Soldier2x9, lifePoints=1, row=3, column=10]
Ingrese el nombre del Soldier que desea buscar
Soldier0x1
Se ha encontrado: Soldier [name=Soldier0x1, lifePoints=3, row=1, column=2]

DATOS DEL EJRCITO B
Soldier [name=Soldier6x3, lifePoints=5, row=7, column=4]
Mayor vida en B: Soldier [name=Soldier6x3, lifePoints=5, row=7, column=4]
El total de vida del ejercito B es: 5
El promedio de vida del ejercito B es: 5.0
Mostrando soldados por ranking de poder de B
Soldier [name=Soldier6x3, lifePoints=5, row=7, column=4]
Ingrese el nombre del Soldier que desea buscar
oooooooooooooooooooo FASE 2 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando el tablero de juego
  A    B    C    D    E    F    G    H    I    J
1 |____|_a3_|____|____|____|____|____|____|____|
2 |____|____|____|____|_a4_|____|____|____|____|____|
3 |____|____|____|____|____|____|____|____|____|_a1_|
4 |____|____|____|____|____|____|____|____|____|____|
5 |____|____|____|____|____|____|____|____|____|____|
6 |____|____|____|____|____|____|_a3_|____|____|____|
7 |____|____|____|_b5_|____|____|____|____|____|____|
8 |____|____|____|_a5_|____|____|____|____|____|____|
9 |____|____|____|____|____|____|____|____|____|____|
10|_a5_|____|____|____|_a5_|____|____|____|____|____|

+++++ FASE 3 DE LA CONTIENDA +++++
El ganador se determina en base a los puntos de vida total
Enfrentamiento
El ganador es el equipo A
Ventaja de 21 puntos de vida
Desea jugar una ronda?(si/no)
NO
```

Listing 34: Commit: dde0af374ba825d7f45cb32605658c8624920f51

```
git add VideoJuego4.java
git commit -m "Se culmina el metodo binarySearchByName"
git push -u origin main
```

Listing 35: Se implementa un método que realiza una búsqueda secuencial según el nombre

```
vim VideoJuego4.java
```

```
260 public static void sequenceSearchByName(ArrayList<Soldier> armyB, String name) {
261     for (int i = 0; i < armyB.size(); i++) {
262         if (name.equalsIgnoreCase(armyB.get(i).getName())) {
263             System.out.println("Se ha encontrado: " + armyB.get(i));
264             return;
265         }
266     }
267     System.out.println("No se ha encontrado coincidencias");
268 }
```

- Este método busca a lo largo del ArrayList elemento por elemento. La búsqueda secuencial no es una buena opción debido a que hace recorridos demás. Se lo emplea más que nada porque los objetivos de la práctica lo exigen.

Listing 36: Compilando y probando el código terminado

```
javac VideoJuego4.java
java VideoJuego4
Desea jugar una ronda?(si/no)
SI
oooooooooooooooooooo FASE 1 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando estadísticas de cada ejercito

Mostrando soldados por orden de creacion
DATOS DEL EJERCITO A
Soldier [name=Soldier5x2, lifePoints=2, row=6, column=3]
Soldier [name=Soldier9x8, lifePoints=1, row=10, column=9]
Mayor vida en A: Soldier [name=Soldier5x2, lifePoints=2, row=6, column=3]
El total de vida del ejercito A es: 3
El promedio de vida del ejercito A es: 1.5
Mostrando soldados por ranking de poder de A
Soldier [name=Soldier5x2, lifePoints=2, row=6, column=3]
Soldier [name=Soldier9x8, lifePoints=1, row=10, column=9]
Ingrese el nombre del Soldier que desea buscar
soldier5x2
Se ha encontrado: Soldier [name=Soldier5x2, lifePoints=2, row=6, column=3]

DATOS DEL EJERCITO B
Soldier [name=Soldier2x9, lifePoints=5, row=3, column=10]
Soldier [name=Soldier3x1, lifePoints=3, row=4, column=2]
Soldier [name=Soldier3x5, lifePoints=2, row=4, column=6]
Soldier [name=Soldier3x7, lifePoints=3, row=4, column=8]
Soldier [name=Soldier4x2, lifePoints=1, row=5, column=3]
Soldier [name=Soldier5x0, lifePoints=1, row=6, column=1]
Soldier [name=Soldier5x8, lifePoints=1, row=6, column=9]
Soldier [name=Soldier7x4, lifePoints=1, row=8, column=5]
Mayor vida en B: Soldier [name=Soldier2x9, lifePoints=5, row=3, column=10]
El total de vida del ejercito B es: 17
El promedio de vida del ejercito B es: 2.125
Mostrando soldados por ranking de poder de B
Soldier [name=Soldier2x9, lifePoints=5, row=3, column=10]
Soldier [name=Soldier3x7, lifePoints=3, row=4, column=8]
Soldier [name=Soldier3x1, lifePoints=3, row=4, column=2]
Soldier [name=Soldier3x5, lifePoints=2, row=4, column=6]
Soldier [name=Soldier7x4, lifePoints=1, row=8, column=5]
Soldier [name=Soldier5x8, lifePoints=1, row=6, column=9]
Soldier [name=Soldier5x0, lifePoints=1, row=6, column=1]
```

```
Soldier [name=Soldier4x2, lifePoints=1, row=5, column=3]
Ingrese el nombre del Soldier que desea buscar
soldier4x2
Se ha encontrado: Soldier [name=Soldier4x2, lifePoints=1, row=5, column=3]

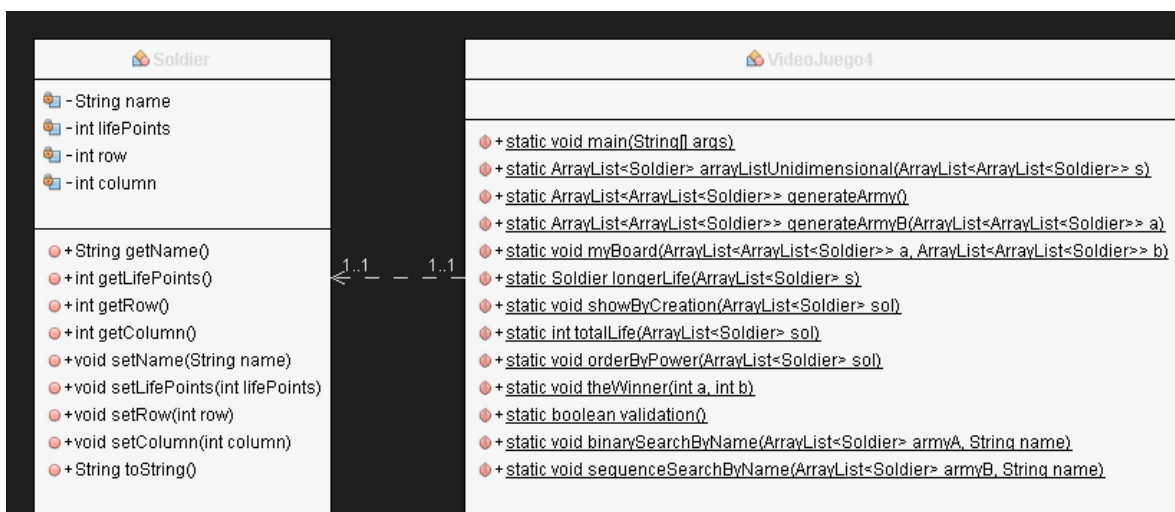
oooooooooooooooooooo FASE 2 DE LA CONTIENDA ooooooooooooooooooooo
Mostrando el tablero de juego
      A      B      C      D      E      F      G      H      I      J
1 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
2 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
3 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_b5_|
4 |_____|_b3_|_____|_____|_____|_____|_b2_|_____|_b3_|_____|_____|
5 |_____|_____|_b1_|_____|_____|_____|_____|_____|_____|_____|
6 |_b1_|_____|_a2_|_____|_____|_____|_____|_____|_____|_b1_|_____|
7 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
8 |_____|_____|_____|_____|_b1_|_____|_____|_____|_____|_____|
9 |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|
10|_____|_____|_____|_____|_____|_____|_____|_____|_____|_a1_|_____|

+++++ FASE 3 DE LA CONTIENDA +++++
El ganador se determina en base a los puntos de vida total
Enfrentamiento
El ganador es el equipo B
Ventaja de 14 puntos de vida
Desea jugar una ronda?(si/no)
no
```

Listing 37: Commit: 5575a0d5eead5b5dd4dca7425a5b9f37229e25fa

```
git add VideoJuego4.java
git commit -m "Se culima el metood sequenceSearchByName"
git push -u origin main
```

4.4. Diagrama UML y el main



```

8 public class VideoJuego4 {
9     public static void main(String[] args) {
10         Scanner sc = new Scanner(System.in);
11         while (validation()) {
12             ArrayList<ArrayList<Soldier>> armyA = generateArmy();
13             ArrayList<ArrayList<Soldier>> armyB = generateArmyB(armyA);
14             ArrayList<Soldier> armyAU = arrayListUnidimensional(armyA);
15             ArrayList<Soldier> armyBU = arrayListUnidimensional(armyB);
16             int a = totalLife(armyAU);
17             int b = totalLife(armyBU);
18
19             System.out.println("oooooooooooooooooooo FASE 1 DE LA CONTIENDA ooooooooooooooooooooo");
20             System.out.println("Mostrando estadísticas de cada ejercito" + "\n");
21             System.out.println("Mostrando soldados por orden de creacion");
22             System.out.println("DATOS DEL EJERCITO A");
23             showByCreation(armyAU);
24             System.out.println("Mayor vida en A: " + longerLife(armyAU));
25             System.out.println("El total de vida del ejercito A es: " + totalLife(armyAU));
26             System.out.println("El promedio de vida del ejercito A es: " + (double) a / armyAU.size());
27             System.out.println("Mostrando soldados por ranking de poder de A");
28             orderByPower(armyAU);
29             System.out.println("Ingrese el nombre del Soldier que desea buscar");
30             String nameA = sc.next();
31             binarySearchByName(armyAU, nameA);
32             System.out.println();
33             System.out.println("DATOS DEL EJERCITO B");
34             showByCreation(armyBU);
35             System.out.println("Mayor vida en B: " + longerLife(armyBU));
36             System.out.println("El total de vida del ejercito B es: " + totalLife(armyBU));
37             System.out.println("El promedio de vida del ejercito B es: " + (double) b / armyBU.size());
38             System.out.println("Mostrando soldados por ranking de poder de B");
39             orderByPower(armyBU);
40             System.out.println("Ingrese el nombre del Soldier que desea buscar");
41             String nameB = sc.next();
42             sequenceSearchByName(armyBU, nameB);
43             System.out.println();
44
45             System.out.println("oooooooooooooooooooo FASE 2 DE LA CONTIENDA ooooooooooooooooooooo");
46             System.out.println("Mostrando el tablero de juego");
47             myBoard(armyA, armyB);
48             System.out.println();
49
50             System.out.println("+++++ FASE 3 DE LA CONTIENDA +++++");
51             System.out.println("El ganador se determina en base a los puntos de vida total");
52             System.out.println("Enfrentamiento");
53             theWinner(a, b);
54         }
55     }
56 }

```

4.5. Estructura de laboratorio 07

- El contenido que se entrega en este laboratorio es el siguiente:

```

lab08
|   Soldier.java
|
|   VideoJuego4.java
|
----latex
|   programacion_lab07_rescobedoq_v1.0.pdf
|   programacion_lab07_rescobedoq_v1.0.tex
|
|----img
|   arrayListUni.jpg
|   binarySearch.jpg
|   board.jpg
|   boardComplete.jpg
|   burbuja.jpg

```

```
| generateArmy.jpg
| generateArmyB.jpg
| insertion.jpg
| logo_abet.png
| logo_episunsa.png
| logo_unsa.jpg
| longerLife.png
| loop.jpg
| main.jpg
| orderByPower.png
| sequence.jpg
| showByCreation.png
| theWinner.jpg
| totalLife.png
| uml.png
| validation.jpg
|
|-----src
| binary.java
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	1	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	3	
Total		20		18	

6. Referencias

- <https://www.geeksforgeeks.org/bubble-sort/>
- <https://www.geeksforgeeks.org/insertion-sort/>