

Informe de Laboratorio 08

Tema: HashMap

Nota

Estudiante	Escuela	Asignatura
Ryan Fabian Valdivia Segovia rvaldiviase@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
08	HashMap	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 11 de Octubre 2023	Al 16 de Octubre 2023

1. Tarea

1.1. Videojuego

- Cree un Proyecto llamado Laboratorio8
- Usted deberá crear las dos clases Soldado.java y VideoJuego5.java. Puede reutilizar lo desarrollado en Laboratorios anteriores.
- Del Soldado nos importa el nombre, puntos de vida, fila y columna (posición en el tablero).
- El juego se desarrollará en el mismo tablero de los laboratorios anteriores. Para crear el tablero utilice la estructura de datos más adecuada.
- Tendrá 2 Ejércitos (usar HashMaps). 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). Se debe mostrar el tablero con todos los soldados creados (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 (indicar conclusiones respecto a este ordenamiento de HashMaps). Finalmente, que muestre qué ejército ganará la batalla (indicar la métrica usada para decidir al ganador de la batalla). Hacerlo como programa iterativo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home Single Language 64 bits 22H2.22H2
- VIM 9.0.
- Visual Studio Code 64 bits 1.82.2
- OpenJDK 64-Bits 11.0.16.1
- Git 2.41.0.windows.1
- Cuenta en GitHub con el correo institucional.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/RyanValdivia/fp2-23b.git>
- URL para el laboratorio 08 en el Repositorio GitHub.
- <https://github.com/RyanValdivia/fp2-23b/tree/main/fase02/lab08>

4. Actividades

4.1. Actividad 1

- En primer lugar, realicé un commit conteniendo el código de la clase Soldado.java, requerido para la clase principal

Listing 1: Obteniendo la clase Soldado

```
$ git log lab06
commit 23a84e524f4e434a9e2a93ac1faac82caf5c5298
Author: RYAN VALDIVIA <rvaldiviase@unsa.edu.pe>
Date: Sun Oct 22 14:22:03 2023 -0500
    Version casi final del código, puliendo la eficiencia y reciclando el código de
    laboratorios pasados, pero adaptandolos a mapas
```

- Conteniendo el siguiente código

Listing 2: Clase Soldado

```
6 public class Soldado {
7     private String nombre;
8     private int vida;
9     private int fila;
10    private int columna;
11    private String bandera;
12
13    public void setNombre(String s) {
14        this.nombre = s;
15    }
```

```
16
17 public void setVida(int n) {
18     this.vida = n;
19 }
20
21 public void setFila(int n) {
22     this.fila = n;
23 }
24
25 public void setColumna(int n) {
26     this.columna = n;
27 }
28
29 public void setBandera(String s) {
30     this.bandera = s;
31 }
32
33 public String getNombre() {
34     return nombre;
35 }
36
37 public int getVida() {
38     return vida;
39 }
40
41 public int getFila() {
42     return fila;
43 }
44
45 public int getColumna() {
46     return columna;
47 }
48
49 public String getBandera() {
50     return bandera;
51 }
52 }
```

- Para este problema, reutilicé el código del anterior laboratorio para contener el talero, ya que me daban libertad de elegir la estructura de datos que quiera para el tablero, así que usaré un arreglo bidimensional.

Listing 3: Código parcial reciclado

```
54 public static void main(String[] args) {
55     Scanner sc = new Scanner(System.in);
56     do {
57         Soldado[][] tablero = new Soldado[10][10];
58         int ej1 = (int) (Math.random() * 10 + 1);
59         int ej2 = (int) (Math.random() * 10 + 1);
60
61         HashMap<String, Soldado> ejercito1 = new HashMap<>();
62         HashMap<String, Soldado> ejercito2 = new HashMap<>();
63
64         int[] filas1 = numerosRandom(ej1);
```

```
65         int[] columnas1 = numerosRandom(ej1);
66         int[] filas2;
67         int[] columnas2;
68         do {
69             filas2 = numerosRandom(ej2);
70             columnas2 = numerosRandom(ej2);
71         } while (!diffCoordenadas(filas1, filas2, columnas1, columnas2));
72
73     public static int[] numerosRandom(int q) {
74         int[] nums = new int[q];
75         for (int i = 0; i < nums.length; i++) {
76             nums[i] = numerosRandom(q);
77         }
78         for (int i = 0; i < q; i++) {
79             int n;
80             do {
81                 n = (int) (Math.random() * 10);
82             } while (estaEnArreglo(nums, n, i));
83             nums[i] = n;
84         }
85         return nums;
86     }
87
88     public static boolean estaEnArreglo(int[] arreglo, int num, int indice) {
89         for (int i = 0; i < indice; i++) {
90             if (arreglo[i] == num) {
91                 return true;
92             }
93         }
94         return false;
95     }
96
97     public static boolean diffCoordenadas(int[] filas1, int[] filas2, int[] columnas1, int[]
98         columnas2) {
99         if (filas1.length > filas2.length) {
100             for (int i = 0; i < filas2.length; i++) {
101                 if (filas1[i] == filas2[i] && columnas1[i] == columnas2[i]) {
102                     return false;
103                 }
104             }
105         } else {
106             for (int i = 0; i < filas1.length; i++) {
107                 if (filas1[i] == filas2[i] && columnas1[i] == columnas2[i]) {
108                     return false;
109                 }
110             }
111         }
112         return true;
113     }
```

- Solo que, en este caso, los dos ejércitos serán almacenados en dos HashMap, por lo que hay que cambiar de estrategia al momento de inicializar los ejércitos, recorriendo los arreglos de coordenadas y asignando los valores al mapa, utilizando los nombres de los Soldados como claves y los objetos como tal como valores.

Listing 4: Inicializar mapa

```
113     inicializarEjercito(ejercito1, filas1, columnas1, 1);
114     inicializarEjercito(ejercito2, filas2, columnas2, 2);
115
116     public static void inicializarEjercito(HashMap<String, Soldado> army, int[] x, int[] y,
117         int ej) {
118         for (int i = 0; i < x.length; i++) {
119             String key = "Soldado" + i + "X" + ej;
120             int v = (int) (Math.random() * 5 + 1);
121             army.put(key, new Soldado());
122             army.get(key).setNombre(key);
123             army.get(key).setFila(x[i]);
124             army.get(key).setColumna(y[i]);
125             army.get(key).setVida(v);
126             if (ej == 1) {
127                 army.get(key).setBandera("#####");
128             } else {
129                 army.get(key).setBandera("*****");
130             }
131         }
132     }
```

- Una vez inicializados los mapas que contendrán a todos los soldados, debía desplegarlos en el tablero, para esto, primero debía inicializar el tablero, como en el laboratorio anterior, por lo que reciclé código.

Listing 5: Inicializar tablero

```
133     public static void inicializarTablero(Soldado[] [] tb) {
134         for (int i = 0; i < tb.length; i++) {
135             for (int j = 0; j < tb[i].length; j++) {
136                 tb[i][j] = new Soldado();
137                 tb[i][j].setNombre("");
138                 tb[i][j].setBandera("");
139             }
140         }
141     }
```

- Una vez inicializado el tablero, viene lo interesante, desplegar a los ejércitos, para ello, recorrí los mapas y asigné los Soldados a sus respectivas posiciones basándome en sus atributos de fila y columna, usando una función lambda con el método 'forEach()' de la clase HashMap.

Listing 6: Desplegando a los soldados

```
142     desplegarEjercito(tablero, ejercito1);
143     desplegarEjercito(tablero, ejercito2);
144     public static void desplegarEjercito(Soldado[] [] tb, HashMap<String, Soldado> army) {
145         army.forEach((key, value) -> {
146             tb[value.getFila()][value.getColumna()] = value;
147         });
148     }
```

- Una vez ya desplegados, quedaba mostrar el tablero, lo cual fue sencillo, ya que reutilicé el código del anterior laboratorio.

Listing 7: Mostrando el tablero

```
149 public static void mostrarTablero(Soldado[] [] tb) {
150     String vacio = "        ";
151     System.out.println(crearTecho());
152     for (int i = 0; i < tb.length; i++) {
153         System.out.println(separadorSup());
154         for (int j = 0; j < tb[i].length; j++) {
155             if (j == tb[i].length - 1) {
156                 System.out.print("| " + tb[i][j].getBandera() + " |\n");
157             } else {
158                 System.out.print("| " + tb[i][j].getBandera() + " ");
159             }
160         }
161         for (int j = 0; j < tb[i].length; j++) {
162             if (j == tb[i].length - 1) {
163                 System.out.print("| " + tb[i][j].getNombre() + " |\n");
164             } else {
165                 System.out.print("| " + tb[i][j].getNombre() + " ");
166             }
167         }
168         for (int j = 0; j < tb[i].length; j++) {
169             if (tb[i][j].getVida() != 0) {
170                 if (j == tb[i].length - 1) {
171                     System.out.print("| " + tb[i][j].getVida() + " HP" + " |\n");
172                 } else {
173                     System.out.print("| " + tb[i][j].getVida() + " HP" + " ");
174                 }
175             } else {
176                 if (j == tb[i].length - 1) {
177                     System.out.print("| " + vacio + " |\n");
178                 } else {
179                     System.out.print("| " + vacio + " ");
180                 }
181             }
182         }
183         System.out.println(separadorInf());
184     }
185     System.out.println();
186 }
```

- Imprimiendo esto al momento de ejecutar el código.

				***** Soldado0X2 2 HP					
							***** Soldado3X2 3 HP		
			***** Soldado6X2 1 HP						
						***** Soldado2X2 1 HP			
									***** Soldado9X2 3 HP
								***** Soldado4X2 2 HP	
					***** Soldado8X2 3 HP	##### Soldado0X1 4 HP			
			***** Soldado1X2 4 HP					##### Soldado1X1 5 HP	
		***** Soldado5X2 5 HP							
***** Soldado7X2 4 HP									

- Una vez terminado el tablero, pasé a trabajar el resto de requerimientos para el programa.
- Ahora debía mostrar el soldado con mayor nivel de vida, para esto, solo debía recorrer el mapa buscando el soldado con mayor vida, tomando el primer valor del mapa como pivote y comprar cada entrada para ver si es mayor o no, una vez encontrado el soldado con mayor vida, se muestra usando otro método reutilizado.

Listing 8: Soldado con mayor vida

```

187     soldadoMayorVida(ejercito1, 1);
188     soldadoMayorVida(ejercito2, 2);
189
190     public static void soldadoMayorVida(Map<String, Soldado> army, int ej) {
191         boolean pro = true;
192         String keyMx = "";
193         for (Map.Entry<String, Soldado> entry : army.entrySet()) {
194             if (pro) {
195                 keyMx = entry.getKey();
196                 pro = false;
197             }
198             if (army.get(keyMx).getVida() < entry.getValue().getVida()) {
199                 keyMx = entry.getKey();
200             }
201         }
202         System.out.println("El soldado con mayor vida del ejercito " + ej + " es: ");
203         mostrarSoldado(army.get(keyMx));
204         System.out.println();
205     }
206 }
```

```
207 public static void mostrarSoldado(Soldado s) {
208     String columna;
209     System.out.println("Nombre: " + s.getNombre());
210     System.out.println("Vida: " + s.getVida() + " HP");
211     switch (s.getColumna() + 1) {
212         case 1:
213             columna = "A";
214             break;
215         case 2:
216             columna = "B";
217             break;
218         case 3:
219             columna = "C";
220             break;
221         case 4:
222             columna = "D";
223             break;
224         case 5:
225             columna = "E";
226             break;
227         case 6:
228             columna = "F";
229             break;
230         case 7:
231             columna = "G";
232             break;
233         case 8:
234             columna = "H";
235             break;
236         case 9:
237             columna = "I";
238             break;
239         case 10:
240             columna = "J";
241             break;
242         default:
243             columna = "K";
244             break;
245     }
246     System.out.println("Posicion: " + (s.getFila() + 1) + "-" + columna);
247     System.out.println();
248 }
```

- Después, seguía el mostrar la vida total y promedio de cada ejército. Para ello, solo debía recorrer el mapa y sumar la vida de todos los valores.

Listing 9: Vida promedio y total

```
249 vidaPromedio(ejercito1, 1);
250 vidaPromedio(ejercito2, 2);
251
252 public static void vidaPromedio(HashMap<String, Soldado> army, int ej) {
253     int total = 0;
254     for (Map.Entry<String, Soldado> entry : army.entrySet()) {
255         total += entry.getValue().getVida();
256     }
```



```
256     }  
257     System.out.println("La vida total del ejercito " + ej + " es: " + total);  
258     System.out.println("La vida promedio del ejercito " + ej + " es: " + total / (1.0 *  
259         army.size()));  
259     System.out.println();  
260 }
```

- Mostrando lo siguiente al momento de ejecutar el código (Siguiendo con la ejecución de la anterior captura).

```
El soldado con mayor vida del ejercito 1 es:  
Nombre: Soldado1X1  
Vida: 5 HP  
Posición: 8-I  
  
El soldado con mayor vida del ejercito 2 es:  
Nombre: Soldado5X2  
Vida: 5 HP  
Posición: 9-B  
  
La vida total del ejercito 1 es: 9  
La vida promedio del ejercito 1 es: 4.5  
  
La vida total del ejercito 2 es: 28  
La vida promedio del ejercito 2 es: 2.8
```

- Ahora seguía algo más complicado, mostrar el ejército según el orden de creación de los soldados, para esto, no me servía simplemente recorrer el mapa, ya que en HashMap, no se sigue un orden según la inserción, por lo tanto, debía guiarme de otra cosa, como las keys de los objetos (Por eso le puse como claves a los objetos, sus respectivos nombres) dando lugar al siguiente método.

Listing 10: Mostrando el ejército

```
261 public static void mostrarEjercito(Map<String, Soldado> army, int ej) {  
262     System.out.println("Ejercito " + ej);  
263     for (int i = 0; i < army.size(); i++) {  
264         String key = "Soldado" + i + "X" + ej;  
265         mostrarSoldado(army.get(key));  
266     }  
267     System.out.println();  
268 }
```

- Mostrando lo siguiente al momento de ejecutar:

```

Ejercito 1
Nombre: Soldado0X1
Vida: 4 HP
Posición: 7-G

Nombre: Soldado1X1
Vida: 5 HP
Posición: 8-I

Nombre: Soldado2X2
Vida: 1 HP
Posición: 4-G

Nombre: Soldado3X2
Vida: 3 HP
Posición: 2-H

Nombre: Soldado4X2
Vida: 2 HP
Posición: 6-I

Nombre: Soldado5X2
Vida: 5 HP
Posición: 9-B

Nombre: Soldado6X2
Vida: 1 HP
Posición: 3-D

Nombre: Soldado7X2
Vida: 4 HP
Posición: 10-A

Nombre: Soldado8X2
Vida: 3 HP
Posición: 7-F

Nombre: Soldado9X2
Vida: 3 HP
Posición: 5-J

```

- Ahora sigue algo mucho más difícil, ordenar los mapas y mostrar el Ranking de soldados de mayor a menor vida.
- El principal problema es que los HashMap no siguen un orden por inserción, por lo tanto, sería imposible ordenar uno, es por ello que tuve que buscar una alternativa.
- Lo primero que hice, fue copiar todos los valores (objetos / Soldados) del mapa, a una lista, para posteriormente ordenarla según el algoritmo deseado. Una vez terminado esto, copiaba de nuevo los objetos ya ordenados a un nuevo mapa, pero ya no un HashMap, sino, un LinkedHashMap, ya que este, a diferencia del normal, si sigue un orden según la inserción.

Listing 11: Algoritmos de ordenamiento

269

```

public static LinkedHashMap<String, Soldado> bubbleSort(HashMap<String, Soldado> map, int
ej) {

```

```
270     ArrayList<Soldado> army = getListas(map);
271     boolean sorted = false;
272     while (!sorted) {
273         sorted = true;
274         for (int i = 0; i < army.size() - 1; i++) {
275             if (army.get(i).getVida() < army.get(i + 1).getVida()) {
276                 Soldado temp = army.get(i);
277                 army.set(i, army.get(i + 1));
278                 army.set(i + 1, temp);
279                 sorted = false;
280             }
281         }
282     }
283     LinkedHashMap<String, Soldado> sortedMap = new LinkedHashMap<>();
284     for (int i = 0; i < army.size(); i++) {
285         String key = army.get(i).getNombre();
286         sortedMap.put(key, army.get(i));
287     }
288     return sortedMap;
289 }
290
291 public static LinkedHashMap<String, Soldado> insertionSort(HashMap<String, Soldado> map,
292     int ej) {
293     ArrayList<Soldado> army = getListas(map);
294     int i, j;
295     for (i = 1; i < army.size(); i++) {
296         Soldado tmp = army.get(i);
297         j = i;
298         while ((j > 0) && (army.get(j - 1).getVida() < tmp.getVida())) {
299             army.set(j, army.get(j - 1));
300             j--;
301         }
302         army.set(j, tmp);
303     }
304     LinkedHashMap<String, Soldado> sortedMap = new LinkedHashMap<>();
305     for (int k = 0; k < army.size(); k++) {
306         String key = army.get(k).getNombre();
307         sortedMap.put(key, army.get(k));
308     }
309     return sortedMap;
310 }
311
312 public static ArrayList<Soldado> getListas(HashMap<String, Soldado> map) {
313     ArrayList<Soldado> list = new ArrayList<Soldado>();
314     list.addAll(map.values());
315     return list;
316 }
```

- De este modo, obtengo un mapa ya ordenado, donde solo me queda recorrerlo imprimiendo los valores de cada soldado, haciendo un pequeño menú para seleccionar el algoritmo de ordenamiento preferido.

Listing 12: Mostrar mapa ya ordenado

```

317 System.out.println("Bajo que algoritmo de ordenamiento le gustaria ordenar su ejercito?");
318 System.out.println("1. Ordenamiento por burbuja");
319 System.out.println("2. Ordenamiento por insercion\n");
320 switch (sc.nextInt()) {
321     case 1:
322         System.out.println("Ranking por Vida: \n");
323         mostrarOrdenado(bubbleSort(ejercito1, 1), 1);
324         mostrarOrdenado(bubbleSort(ejercito2, 2), 2);
325         break;
326     case 2:
327         System.out.println("Ranking por Vida: \n");
328         mostrarOrdenado(insertionSort(ejercito1, 1), 1);
329         mostrarOrdenado(insertionSort(ejercito2, 2), 2);
330         break;
331     default:
332 }
333 public static void mostrarOrdenado(Map<String, Soldado> map, int ej) {
334     System.out.println("Ejercito " + ej);
335     System.out.println(map.get("Soldado0X" + ej).getBandera() + "\n");
336     for (Map.Entry<String, Soldado> entry : map.entrySet()) {
337         mostrarSoldado(entry.getValue());
338     }
339 }

```

- Mostrando lo siguiente:

```

Bajo que algoritmo de ordenamiento le gustaria ordenar su ejercito?  Ejercito 2
1. Ordenamiento por burbuja                                         *****
2. Ordenamiento por insercion

1
Ranking por Vida:
Ejercito 1
#####
Nombre: Soldado1X1
Vida: 5 HP
Posición: 8-I
Nombre: Soldado0X1
Vida: 4 HP
Posición: 7-G
Nombre: Soldado5X2
Vida: 5 HP
Posición: 9-B
Nombre: Soldado1X2
Vida: 4 HP
Posición: 8-C
Nombre: Soldado7X2
Vida: 4 HP
Posición: 10-A
Nombre: Soldado3X2
Vida: 3 HP
Posición: 2-H
Nombre: Soldado8X2
Vida: 3 HP
Posición: 7-F
Nombre: Soldado9X2
Vida: 3 HP
Posición: 5-J
Nombre: Soldado4X2
Vida: 2 HP
Posición: 6-I
Nombre: Soldado0X2
Vida: 2 HP
Posición: 1-E
Nombre: Soldado2X2
Vida: 1 HP
Posición: 4-G
Nombre: Soldado6X2
Vida: 1 HP
Posición: 3-D

```

- Una vez culminado esto, solo quedaba algo mucho más simple, determinar cuál ejército gana, para lo cual solo necesitaba comparar los niveles de vida totales de ambos mapas.

Listing 13: Ejército ganador

```

340 public static void ejercitoGanador(Map<String, Soldado> map1, Map<String, Soldado> map2) {
341     int total1 = 0, total2 = 0;
342     for (Map.Entry<String, Soldado> entry : map1.entrySet()) {

```

```

343         total1 += entry.getValue().getVida();
344     }
345     for (Map.Entry<String, Soldado> entry : map2.entrySet()) {
346         total2 += entry.getValue().getVida();
347     }
348     if (total1 > total2) {
349         System.out.println("El ejercito 1 es ganador!");
350     } else if (total1 == total2) {
351         System.out.println("Hay empate!");
352     } else {
353         System.out.println("El ejercito 2 es ganador!");
354     }
355     System.out.println("Bajo la metrica de que ejercito tiene mas vida");
356 }

```

```

Bajo que criterio le gustaria ordenar los ejercitos?
1. Burbuja
2. Insercion
1
Ranking segun vida (Del mayor al menor):
Ejercito 1
Nombre: Soldado1X1
Vida: 4 HP
Posición: 4-B
Nombre: Soldado0X1
Vida: 3 HP
Posición: 10-A

Ejercito 2
Nombre: Soldado0X2
Vida: 5 HP
Posición: 5-C
Nombre: Soldado1X2
Vida: 4 HP
Posición: 9-B
Nombre: Soldado3X2
Vida: 4 HP
Posición: 8-D
Nombre: Soldado2X2
Vida: 2 HP
Posición: 10-G

```

- Ya para terminar, solo faltaría saber qué ejército ganó, para esto, usé el criterio de qué ejército tiene más nivel de vida total.

Listing 14: Determinando el ganador

```

357 public static void ejercitoGanador(Soldado[] army1, Soldado[] army2) {
358     int total1 = 0;
359     int total2 = 0;
360     for (int i = 0; i < army1.length; i++) {
361         total1 += army1[i].getVida();
362     }
363     for (int i = 0; i < army2.length; i++) {
364         total2 += army2[i].getVida();
365     }
366     if (total1 > total2) {
367         System.out.println("El ejercito 1 es ganador!");

```

```
368     } else if (total1 == total2) {  
369         System.out.println("Empate");  
370     } else {  
371         System.out.println("El ejercito 2 es ganador");  
372     }  
373     System.out.println("Bajo el criterio de que ejercito tiene mas vida total");  
374  
375 }
```

- Imprimiendo lo siguiente:

```
El ejercito 2 es ganador!  
Bajo la metrica de que ejercito tiene mas vida
```

- Y para hacer el programa iterativo, solo coloqué un do-while en el método main, para que se pueda repetir el programa.

Listing 15: Método main final

```
376 public static void main(String[] args) {  
377     Scanner sc = new Scanner(System.in);  
378     do {  
379         Soldado[][] tablero = new Soldado[10][10];  
380         int ej1 = (int) (Math.random() * 10 + 1);  
381         int ej2 = (int) (Math.random() * 10 + 1);  
382  
383         HashMap<String, Soldado> ejercito1 = new HashMap<>();  
384         HashMap<String, Soldado> ejercito2 = new HashMap<>();  
385  
386         int[] filas1 = numerosRandom(ej1);  
387         int[] columnas1 = numerosRandom(ej1);  
388         int[] filas2;  
389         int[] columnas2;  
390         do {  
391             filas2 = numerosRandom(ej2);  
392             columnas2 = numerosRandom(ej2);  
393         } while (!diffCoordenadas(filas1, filas2, columnas1, columnas2));  
394  
395         inicializarEjercito(ejercito1, filas1, columnas1, 1);  
396         inicializarEjercito(ejercito2, filas2, columnas2, 2);  
397         inicializarTablero(tablero);  
398         desplegarEjercito(tablero, ejercito1);  
399         desplegarEjercito(tablero, ejercito2);  
400         mostrarTablero(tablero);  
401         soldadoMayorVida(ejercito1, 1);  
402         soldadoMayorVida(ejercito2, 2);  
403         vidaPromedio(ejercito1, 1);  
404         vidaPromedio(ejercito2, 2);  
405         mostrarEjercito(ejercito1, 1);  
406         mostrarEjercito(ejercito2, 2);  
407  
408         System.out.println("Bajo que algoritmo de ordenamiento le gustaria ordenar su  
ejercito?");
```

```
409     System.out.println("1. Ordenamiento por burbuja");
410     System.out.println("2. Ordenamiento por insercion\n");
411     switch (sc.nextInt()) {
412         case 1:
413             System.out.println("Ranking por Vida: \n");
414             mostrarOrdenado(bubbleSort(ejercito1, 1), 1);
415             mostrarOrdenado(bubbleSort(ejercito2, 2), 2);
416             break;
417         case 2:
418             System.out.println("Ranking por Vida: \n");
419             mostrarOrdenado(insertionSort(ejercito1, 1), 1);
420             mostrarOrdenado(insertionSort(ejercito2, 2), 2);
421             break;
422         default:
423     }
424     System.out.println("Presione q para salir, o cualquier otra tecla para volver a
425         jugar");
426     ejercitoGanador(ejercito1, ejercito2);
427 } while (!sc.next().equals("q"));
428 }
```

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	2	
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	2	
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	4	
Total		20		18	

6. Referencias

- Fundamentos de la programación 2 - Tópicos de la programación Orientada a Objetos (Marco Aedo)