

# Informe de Laboratorio 08

## Tema: HashMap

Nota

Estudiante	Escuela	Asignatura
Hernan Andy Choquehuanca Zapana hchoquehuanca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 20232191

Laboratorio	Tema	Duración
08	HashMap	02 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 18 Octubre 2023	Al 23 Octubre 2023

## 1. Tarea

- 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 Pro 22H2 64 bits.
- Visual Studio Code.
- Git 2.42.0.
- Cuenta en GitHub con el correo institucional.
- Editor LaTeX en línea Overleaf.
- Variables Simples
- Métodos.
- Métodos de Búsqueda y Ordenamiento.
- HashMap

## 3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/hernanchoquehuanca/fp2-23b.git`
- URL para el laboratorio 07 en el Repositorio GitHub.
- `https://github.com/hernanchoquehuanca/fp2-23b/tree/main/fase02/lab08`

## 4. Trabajo del Laboratorio 08

### 4.1. Actividad 01

#### 4.1.1. Clase Soldado.java

- Haciendo uso de la clase Soldado.java del laboratorio anterior (lab07), se adaptó al ejercicio actual.
  - Primero copiamos el código tal cual a la carpeta del laboratorio actual (lab08).

Listing 1: Commit 995eebf: En el primer commit se reutilizaba la clase Soldado.java del laboratorio anterior (lab07)

```
$ git add .  
$ git commit -m "Se reutilizo las clases del laboratorio anterior, para en este caso  
    realizarlo con hashmap"  
$ git push -u origin main
```

- Nuestra clase Soldado tendrá los siguientes atributos:

- name (Nombre).
- row (Fila).
- column (Columna).
- status (Estado).
- health (Vida).
- team (Equipo).

```
2 private String name;  
3 private int row;  
4 private char column;  
5 private boolean status;  
6 private int health;  
7 private char team;
```

- Además creamos el método constructor, teniendo en cuenta que tiene que tener el mismo nombre que la clase y considerando sus atributos como parámetros:

- Se utilizó el puntero `this` para evitar ambigüedades.

```
9 public Soldado(String name, int row, char column, boolean status, int health, char team) {  
10     this.name = name;  
11     this.row = row;  
12     this.column = column;  
13     this.status = status;  
14     this.health = health;  
15     this.team = team;  
16 }
```

- Luego de ello también se implementaron los getters y setters para cada atributo de nuestra clase Soldado.

- Setters:

```
18 public void setName(String n){  
19     name = n;  
20 }  
21 public void setRow(int r){  
22     row = r;  
23 }  
24 public void setColumn(char c){  
25     column = c;  
26 }  
27 public void setStatus(boolean s){  
28     status = s;  
29 }  
30 public void setHealth(int h){  
31     health = h;  
32 }  
33 public void setTeam(char t){  
34     team = t;  
35 }
```

- Getters:

```
37 public String getName(){
38     return name;
39 }
40 public int getRow(){
41     return row;
42 }
43 public char getColumn(){
44     return column;
45 }
46 public boolean getStatus(){
47     return status;
48 }
49 public int getHealth(){
50     return health;
51 }
52 public char getTeam(){
53     return team;
54 }
```

- Y finalizando con esta clase, se creo el método toString().
  - Se agregó la anotación `@Override` para indicar que se está reemplazando el método de su clase padre `Objetct`.
  - En este método se considera los atributos de la clase (name, row, column, status, health, team).
  - Se utilizaron saltos de con ayuda del `\n`.

```
55 @Override
56 public String toString() {
57     return "Data { " +
58         "\n Name: " + name +
59         "\n Row: " + row +
60         "\n Column: " + column +
61         "\n Status: " + status +
62         "\n Health: " + health +
63         "\n}\n";
64 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

```
EJERCITO "A"
Data {
  Name:  Soldier0XA
  Row:   9
  Column: I
  Status: true
  Health: 4
}

Data {
  Name:  Soldier1XA
  Row:   9
  Column: F
  Status: true
  Health: 2
}

Data {
  Name:  Soldier2XA
  Row:   5
  Column: J
  Status: true
  Health: 2
}
```

Figura 1

#### 4.1.2. Clase VideoJuego.java

- Primero se comenzó con la creación de la clase.
- Posterior a ello se crearon dos variables de clase:
  - Un HashMap bidimensional, el cual contendrá a los soldados del ejército.
  - Además de un HashMap unidimensional que nos servirá para contener a los soldados creados previamente.

```
8 public class VideoJuego5 {
9     static HashMap <Integer, Soldado> army = new HashMap<>();
10    static HashMap <Integer, Soldado> army1DA = new HashMap<>();
11    static HashMap <Integer, Soldado> army1DB = new HashMap<>();
```

Listing 2: Commit 49a014b: En el commit se modificaba las variables de la clase VideoJuego5.java para utilizar HashMap

```
$ git add .
$ git commit -m "Redefiniendo los metodos createArmy y createArmyTeam para ahora
    trabajar con HashMap, ademas se cambiaron las variables de clase que almacenan
    el tablero y los ejércitos"
$ git push -u origin main
```

#### 4.1.3. Método para la creación de los ejército

- El método tiene como nombre `createArmy()`.
- Primero se define el número de soldados que contendrá cada ejército, haciendo uso de `Math.random`.
- Ahora se llama dos veces al método `createArmyTeam` para realizar la creación de los dos ejércitos a partir de los tamaños ya establecidos anteriormente y además el argumento tipo char para el identificador de cada equipo.

```
88 public static void createArmy(){
89     int numSoldiersA = (int) (Math.random() * 10) + 1;
90     int numSoldiersB = (int) (Math.random() * 10) + 1;
91
92     army1DA = createArmyTeam(numSoldiersA, army1DA, 'A');
93     army1DB = createArmyTeam(numSoldiersB, army1DB, 'B');
94 }
```

Listing 3: Commit 49a014b: En el commit se modificaba los métodos para trabajar con HashMap dentro del método `createArmy` eliminando un bucle que anteriormente trabajaba con HashMap

```
$ git add .
$ git commit -m "Redefiniendo los metodos createArmy y createArmyTeam para ahora
    trabajar con HashMap, ademas se cambiaron las variables de clase que almacenan
    el tablero y los ejercitos"
$ git push -u origin main
```

#### 4.1.4. Método para la creación de ejércitos

- El método tiene como nombre `createArmyTeam()`.
- Recibe como parámetros un entero que es el número de soldados del ejército, el HashMap de soldados a utilizar y un String que es el char que va al final del nombre de los soldados, esto último nos ayuda a identificarlos mejor.
- Utilizando un do while se crean posiciones aleatorias en el HashMap bidimensional, tomando como condición que dicha posición sea distinta de `null`.
- Finalmente se crea el soldado asignando su posición como clave `Integer` (el valor de la fila se guarda multiplicado por 10 sumándole el valor de la columna), el valor es el soldado y se almacena en cada HashMap y retorna el HashMap unidimensional con los soldados creados.

```
96 public static HashMap <Integer, Soldado> createArmyTeam(int numSoldiers, HashMap <Integer,
    Soldado> army1D, char t){
97     for (int i = 0; i < numSoldiers; i++){
98         int row, col;
99
100        do {
101            row = (int) (Math.random() * 9) + 1;
102            col = (int) (Math.random() * 9) + 1;
103        } while (army.containsKey(row * 10 + col));
104
105        Soldado s = new Soldado("Soldier" + i + "X" + t, row + 1, (char) (col + 'A'), true,
            (int) (Math.random() * 5) + 1, t);
106        army1D.put(i, s);
107        army.put(row * 10 + col, s);
108    }
109    return army1D;
110 }
```

Listing 4: Commit 49a014b: En el commit se modificaba los métodos para trabajar con HashMap dentro del método createArmyTeam

```
$ git add .
$ git commit -m "Redefiniendo los metodos createArmy y createArmyTeam para ahora
    trabajar con HashMap, ademas se cambiaron las variables de clase que almacenan
    el tablero y los ejércitos"
$ git push -u origin main
```

#### 4.1.5. Método para mostrar la tabla con el ejército

- El método tiene como nombre `showArmyTable()`.
- La funcionalidad es simple:
  - Se crea el String (`linesDown`), este será usado para la impresión de las líneas inferiores de cada fila de recuadros.
  - Primeramente se imprime la parte superior, donde se encuentran las letras que indica las columnas. Seguido de una línea que representa la parte superior de la tabla.
  - Se utilizó un bucle for con dos bucles en su interior, siendo su principal para las filas y los secundarios para las columnas.
  - En el primero, inicia imprimiendo los datos de los soldados si es que se encuentran, se evalúa si contiene un Soldado o `null`; en caso de ser así, imprimirá “ —”, en caso de contener a un soldado completará el cuadrado de la tabla incluyendo ejército (A o B), y su nombre simplificado (S + número generado).
  - Dentro del segundo for imprime el número de fila, se agregó una condicional que nos sirve para darle un toque de simetría a esta enumeración de fila, y así evitar que al momento de imprimir el 10 recorra un espacio. Además en caso de que en una posición se encuentre a un soldado, se colocará la vida de este.
  - Finalmente se imprime `linesDown` que completaría la línea inferior de cada fila.

```

112 public static void showArmyTable(HashMap <Integer, Soldado> army){
113     System.out.println("\n          TABLA CON LAS UBICACIONES DE LOS SOLDADOS CREADOS:
114         \n");
115     String linesDown = "
116         |-----|-----|-----|-----|-----|-----|-----|-----|-----|";
117     System.out.println("          A          B          C          D          E          F          G          H          I
118         J\n"
119         + "
120         -----");
121     for (int r = 0; r < 10; r++){
122         System.out.print(" |");
123         for (int c = 0; c < 10; c++){
124             System.out.print(" " + (army.get(r*10+c) != null ? ("\'" + army.get(r*10+c).getTeam()
125                 + "\'")
126                 + "S" + army.get(r*10+c).getName().charAt(7) + " |") : " |"));
127         System.out.print("\n" + (r+1) + ((r != 9) ? " |" : " |"));
128         for (int c = 0; c < 10; c++){
129             System.out.print(" " + (army.get(r*10+c) != null ? "HP: " +
130                 army.get(r*10+c).getHealth() : " ") + " |");
131         System.out.println("\n" + linesDown );
132     }
133     System.out.println();
134 }

```

- Un ejemplo de como se muestra en la siguiente imagen:

TABLA CON LAS UBICACIONES DE LOS SOLDADOS CREADOS:										
	A	B	C	D	E	F	G	H	I	J
1										
2				'A'S3 HP: 3						
3					'A'S0 HP: 2		'B'S1 HP: 1		'A'S1 HP: 3	
4							'B'S0 HP: 2			
5										
6									'B'S4 HP: 5	
7					'A'S2 HP: 5					
8					'A'S4 HP: 4		'B'S2 HP: 4			
9										
10					'B'S3 HP: 1					

Figura 2



Listing 5: Commit 7bfe7cc: Se adaptó e implementó el método para mostrar la tabla con los soldados de ambos ejércitos incluyendo su vida

```
$ git add .  
$ git commit -m "Redefiniendo el metodo showArmyTable para ahora trabajar con  
    HashMap, simplemente se adaptaron los metodos para acceder a los soldados y los  
    limites de interacciones en cada for"  
$ git push -u origin main
```

#### 4.1.6. Método para mostrar los datos de los soldados de un ejército

- El método tiene como nombre `showArmyData()`.
- Este usa un `for each` para recorrer el `HashMap` unidimensional de `Soldado`, y luego mostrar sus datos con el `System.out.println`, que a su vez este sigue el formato que se estableció en el método `toString()` de la clase `Soldado.java`.
- La impresión en consola será la misma que la figura 1.

```
133 public static void showArmyData(HashMap <Integer, Soldado> army1D, char t){  
134     System.out.println("EJERCITO \" + t + "\"");  
135     for (Soldado s : army1D.values())  
136         System.out.println(s);  
137 }
```

Listing 6: Commit c610a49: Se adaptó e implementó el método para mostrar los datos de los soldados de un ejército

```
$ git add .  
$ git commit -m "Redefiniendo el metodo showArmyData para ahora trabajar con  
    HashMap, se cambio el metodo para acceder a los valores del HashMap en el for  
    each"  
$ git push -u origin main
```

#### 4.1.7. Método para mostrar aquellos soldados con más vida de un ejército

- El método tiene como nombre `moreHealt()`.
- Este recibe el `HashMap` con los soldados de un ejército, además de un `char` que indica el nombre del equipo.
- Primero recorre el `HashMap` unidimensional de soldados haciendo uso de un bucle `for`, de esta manera obtendrá el máximo de vida del ejército, el cual será almacenado en un entero `maxHealth`.
- Finalmente imprimirá aquellos soldados que tengan la vida igual a `maxHealth`, con un `for` y un `if` que controlará aquello.
- En la impresión se incluye el nombre del ejército antes de mostrar a los soldados.

```

139 public static void moreHealth(HashMap <Integer, Soldado> army1MH, char t){
140     int maxHealth = -1;
141     for(Soldado s : army1MH.values())
142         if (s.getHealth() > maxHealth)
143             maxHealth = s.getHealth();
144
145     System.out.println("Soldado(s) con mayor vida del Ejercito " + t + ": ");
146     for (Soldado s : army1MH.values())
147         if (s.getHealth() == maxHealth)
148             System.out.println("Nombre: " + s.getName() + " Vida: " + s.getHealth());
149     System.out.println();
150 }

```

- Un ejemplo de como se muestra en la siguiente imagen:

```

Soldado(s) con mayor vida del Ejercito A:
Nombre: Soldier4XA  Vida: 5
Nombre: Soldier6XA  Vida: 5

Soldado(s) con mayor vida del Ejercito B:
Nombre: Soldier2XB  Vida: 5
Nombre: Soldier3XB  Vida: 5

```

Figura 3

Listing 7: Commit 9c29bbd: Se agregó el método moreHealth() que imprimirá aquellos soldados que tengan la mayor vida dentro de su ejército

```

$ git add .
$ git commit -m "Redefiniendo el metodo moreHealth para ahora trabajar con HashMap,
    se cambio el metodo para acceder a los valores del HashMap en el for each"
$ git push -u origin main

```

#### 4.1.8. Método para hallar la suma de vida en un ejército

- El método tiene como nombre `sumHealth()`.
- Se utiliza un entero inicializado en 0 para mientras que se recorre el HashMap unidimensional de Soldado con un bucle for each, este entero (sum) va almacenando la vida de todos los soldados.
- Finalmente se retorna el entero `sum`.

```

156 public static int sumHealth(HashMap <Integer, Soldado> army1DSH){
157     int sum = 0;
158     for (Soldado s : army1DSH.values())
159         sum += s.getHealth();
160     return sum;
161 }

```

- Un ejemplo de como se muestra en la siguiente imagen:

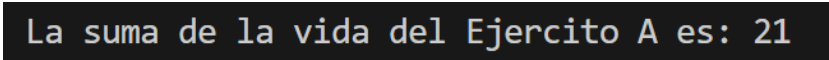


Figura 4

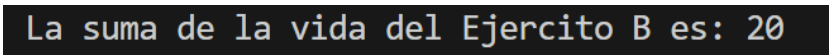


Figura 5

#### 4.1.9. Método para hallar el promedio de vida en un ejército

- El método tiene como nombre `averageHealth()`.
- De manera breve como el método, este retorna una división entre la suma de la vida del ejército, haciendo uso del método `sumHealth()` y dividiendo entre el tamaño del ejército.

```
152 public static double averageHealth(HashMap <Integer, Soldado> army1DAH){  
153     return sumHealth(army1DAH) / army1DAH.size();  
154 }
```

- Un ejemplo de como se muestra en la siguiente imagen:




Figura 6




Figura 7

Listing 8: Commit 56d8575 - 4cec82f: Se agregó el método `averageHealth` y `sumHealth`, siendo el segundo de utilidad para el primero

```
$ git add .  
$ git commit -m "Redefiniendo el metodo averageHealth para ahora trabajar con  
HashMap, cambiando solamente el parametro del mismo"  
$ git push -u origin main
```

#### 4.1.10. Método de ordenamiento BubbleSort

- El método tiene como nombre `bubbleSort()`.
- El método tiene como finalidad ordenar el `HashMap` unidimensional de `Soldado` haciendo uso del algoritmo `BubbleSort`, tomando en cuenta la vida de los soldados que contiene cada soldado de dicho `HashMap`. Este algoritmo se extrajo de [Geeksforgeeks](https://www.geeksforgeeks.org/bubble-sort/) y fue adaptado a este proyecto.
- <https://www.geeksforgeeks.org/bubble-sort/>
- CONCLUSIONES:
  - El algoritmo itera a través de los elementos del `HashMap`, comparando la salud de los soldados y realizando intercambios si es necesario.
  - El proceso se ejecuta hasta que no se realicen más intercambios, lo que indica que los elementos están ordenados.
  - El algoritmo de ordenamiento burbuja es simple, pero no eficiente, ya que su complejidad es de  $O(n^2)$  y si se trabajase con más datos la ejecución tardaría demasiado.

```
163 public static HashMap <Integer, Soldado> bubbleSort(HashMap <Integer, Soldado> army1DBS){
164     HashMap<Integer, Soldado> army1DCopyBubble = new HashMap<>(army1DBS);
165     int n = army1DCopyBubble.size();
166     boolean swapped;
167     for (int i = 0; i < n - 1; i++) {
168         swapped = false;
169         for (int j = 0; j < n - i - 1; j++)
170             if (army1DCopyBubble.get(j).getHealth() < army1DCopyBubble.get(j+1).getHealth()) {
171                 Soldado temp = army1DCopyBubble.get(j);
172                 army1DCopyBubble.put(j, army1DCopyBubble.get(j+1));
173                 army1DCopyBubble.put(j+1, temp);
174                 swapped = true;
175             }
176         if (!swapped)
177             break;
178     }
179     return army1DCopyBubble;
180 }
```

Listing 9: Commit 4310e77: Se agregó el método que contiene el algoritmo de ordenamiento `bubbleSort`, reutilizándola del laboratorio anterior y adaptándola a los nuevos ejércitos en `HashMap`

```
$ git add .
$ git commit -m "Adaptando el metodo bubbleSort para ahora trabajar con HashMap,
    cambiando el parametro del mismo y adaptando el metodo para colocar valores
    dentro del HashMap"
$ git push -u origin main
```

#### 4.1.11. Método de ordenamiento InsertionSort

- El método tiene como nombre `insertionSort()`.
- El método tiene como finalidad ordenar el HashMap unidimensional de Soldado haciendo uso del algoritmo InsertionSort, tomando en cuenta la vida de los soldados que contiene cada soldado de dicho HashMap. Este algoritmo se extrajo de Geeksforgeeks y fue adaptado a este proyecto.
- <https://www.geeksforgeeks.org/insertion-sort/>
- CONCLUSIONES:
  - El algoritmo recorre el HashMap desde el segundo soldado hasta el último, considerando cada vez al soldado como una "llave" compara su salud con la de los soldados anteriores.
  - Los soldados son recorridos hacia la derecha en el HashMap hasta que se encuentren en la posición correcta según la vida del soldado "llave".<sup>actual</sup>.
  - El algoritmo de ordenamiento por inserción es más eficiente que el burbuja, pero a pesar de ello su complejidad sigue siendo de  $O(n^2)$ .

```
182 public static HashMap <Integer, Soldado> insertionSort(HashMap <Integer, Soldado> army1DIS)
183 {
184     int n = army1DIS.size();
185     HashMap<Integer, Soldado> army1DCopyInsertion = new HashMap<>(army1DIS);
186
187     for (int i = 1; i < n; i++) {
188         Soldado key = army1DCopyInsertion.get(i);
189         int j = i - 1;
190
191         while (j >= 0 && army1DCopyInsertion.get(j).getHealth() < key.getHealth()) {
192             army1DCopyInsertion.put(j+1, army1DCopyInsertion.get(j));
193             j = j - 1;
194         }
195         army1DCopyInsertion.put(j+1, key);
196     }
197     return army1DCopyInsertion;
198 }
```

Listing 10: Commit e569a5f: Se agregó el método que contiene el algoritmo de ordenamiento insertion-Sort, reutilizándola del laboratorio anterior y adaptándola a los nuevos ejércitos en HashMap

```
$ git add .
$ git commit -m "Adaptando el metodo insertionSort para ahora trabajar con HashMap,
    cambiando el parametro del mismo y adaptando el metodo para colocar valores
    dentro del HashMap"
$ git push -u origin main
```

#### 4.1.12. Método de ordenamiento SelectionSort

- El método tiene como nombre `selectionSort()`.
- El método tiene como finalidad ordenar el HashMap unidimensional de Soldado haciendo uso del algoritmo SelectionSort, tomando en cuenta la vida de los soldados que contiene cada soldado de dicho HashMap. Este algoritmo se extrajo de Geeksforgeeks y fue adaptado a este proyecto.
- <https://www.geeksforgeeks.org/selection-sort/>
- CONCLUSIONES:
  - El algoritmo recorre el HashMap, dividiendo el ejército en dos partes (la ordenada y la no ordenada).
  - En cada iteración, se busca el soldado con la salud máxima en la parte no ordenada y se intercambia con el soldado en la parte ordenada.
  - El algoritmo de ordenamiento por selección tiene una complejidad de tiempo  $O(n^2)$ , pero puede ser más rápido que el burbuja o la inserción en algunos casos debido al número de soldados dentro de un ejército.

```
199 public static HashMap <Integer, Soldado> selectionSort(HashMap <Integer, Soldado> army1DSS)
    {
200     int n = army1DSS.size();
201     HashMap<Integer, Soldado> army1DCopySelection = new HashMap<>(army1DSS);
202
203     for (int i = 0; i < n - 1; i++) {
204         int min_idx = i;
205
206         for (int j = i + 1; j < n; j++)
207             if (army1DCopySelection.get(j).getHealth() >
                army1DCopySelection.get(min_idx).getHealth())
208                 min_idx = j;
209
210         Soldado temp = army1DCopySelection.get(min_idx);
211         army1DCopySelection.put(min_idx, army1DCopySelection.get(i));
212         army1DCopySelection.put(i, temp);
213     }
214     return army1DCopySelection;
215 }
```

Listing 11: Commit 4fdc5df: Se agregó el método que contiene el algoritmo de ordenamiento selection-Sort, reutilizándola del laboratorio anterior y adaptándola a los nuevos ejércitos en HashMap

```
$ git add .
$ git commit -m "Adaptando el metodo selectionSort para ahora trabajar con HashMap,
    cambiando el parametro del mismo y adaptando el metodo para colocar valores
    dentro del HashMap"
$ git push -u origin main
```

#### 4.1.13. Método para imprimir los soldados de un ejército, ordenados según su vida

- El método tiene como nombre `printArmyHealth()`.
- Este método recibirá un `HashMap` unidimensional de `Soldado` (previamente ordenado), lo recorrerá usando un bucle `for` y mostrará los soldados, teniendo en cuenta que primero se mostrarán los de mayor vida hasta los de menor.

```
217 public static void printArmyHealth(HashMap <Integer, Soldado> armyPrint, char t){
218     System.out.println("EJERCITO " + t + " : ");
219     for(int i = 0; i < armyPrint.size(); i++)
220         System.out.println((i + 1) + ". " + armyPrint.get(i).getName() + " Vida: " +
221             armyPrint.get(i).getHealth());
222 }
```

Listing 12: Commit a08c087: Se implementará el método `printArmyHealth`

```
$ git add .
$ git commit -m "Redefiniendo el metodo printArmyHealth para ahora trabajar con
    HashMap, cambiando solamente el parametro del mismo)"
$ git push -u origin main
```

- Un ejemplo de como se muestra utilizando los 3 algoritmos de ordenamiento en la siguiente imagen:

```
Ejercitos ordenados (bubbleSort) segun la vida:
EJERCITO A:
1. Soldier0XA Vida: 4
2. Soldier1XA Vida: 1
EJERCITO B:
1. Soldier0XB Vida: 4
2. Soldier3XB Vida: 4
3. Soldier2XB Vida: 3
4. Soldier1XB Vida: 1

Ejercitos ordenados (insertionSort) segun la vida:
EJERCITO A:
1. Soldier0XA Vida: 4
2. Soldier1XA Vida: 1
EJERCITO B:
1. Soldier0XB Vida: 4
2. Soldier3XB Vida: 4
3. Soldier2XB Vida: 3
4. Soldier1XB Vida: 1

Ejercitos ordenados (selectionSort) segun la vida:
EJERCITO A:
1. Soldier0XA Vida: 4
2. Soldier1XA Vida: 1
EJERCITO B:
1. Soldier0XB Vida: 4
2. Soldier3XB Vida: 4
3. Soldier2XB Vida: 3
4. Soldier1XB Vida: 1
```

Figura 8

#### 4.1.14. Método para mostrar al ejército ganador según la vida de sus soldados

- El método tiene como nombre `armyWinnerHealth()`.
- Este método utilizará la suma de vida de cada ejército, hará comparaciones y si uno de los dos es superior en número de vida, se imprimirá que ganó, en caso de ser iguales mostrará aquello. **Extraído tal cual del laboratorio anterior (lab07) ya que no requiere modificaciones, por ende no hay un commit en el laboratorio 08 en esta sección**

```
223 public static void armyWinnerHealth(){
224     System.out.print("\nSegun la suma de vida de los soldados, ");
225     if (sumHealth(army1DA) > sumHealth(army1DB))
226         System.out.println("el ejercito ganador es el: \'A\'");
227     else if (sumHealth(army1DB) > sumHealth(army1DA))
228         System.out.println("el ejercito ganador es el: \'B\'");
229     else
230         System.out.println("la batalla quedo en empate");
231     System.out.println();
232 }
```

#### 4.1.15. Método para mostrar la interfaz y volver el programa iterativo

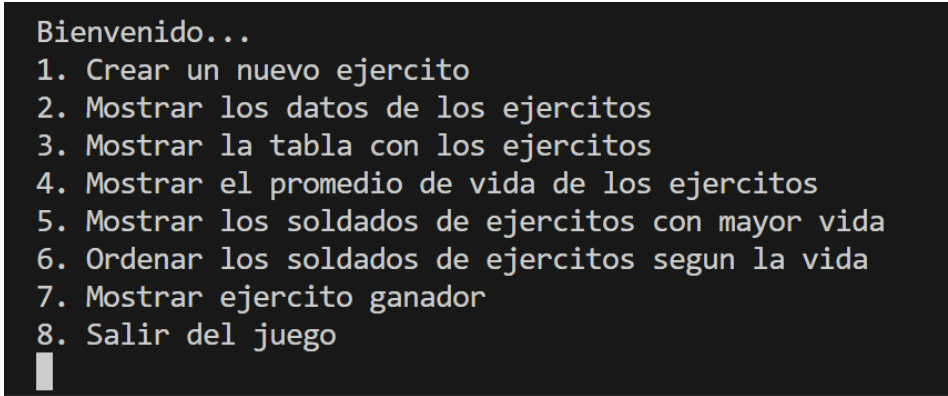
- El método tiene como nombre `mainInterfaz()`.
- Al iniciar esta pequeña interfaz, se muestra las acciones que se pueden realizar, las cuales se eligen escribiendo un número del 1 - 8 según se desee.
- Al terminar cada caso desde el 1 al 7, se vuelve a llamar al mismo método, esto lo vuelve iterativo.
- En el caso 1, se hace el llamado al método `createArmy()`, esto para crear un nuevo ejército eliminando primero todos los ejércitos previamente creado (incluyendo el tablero).
- En el caso 2, se hace el llamado al método `showArmyData()`, esto mostrará los soldados de ambos ejércitos (A - B) siguiendo el orden de su creación.
- En el caso 3, se hace el llamado al método `showArmyTable()`, entonces mostrará el tablero con los soldados de ambos ejércitos, incluye su team, nombre y vida.
- En el caso 4, se hace el llamado al método `averageHealth()`, será realizado para ambos ejércitos, mostrando así el promedio de vida en cada uno.
- En el caso 5, se hace el llamado al método `moreHealth()`, aplicándolo a cada ejército mostrando así sus soldados con mayor cantidad de vida dentro de cada uno.
- En el caso 6, se hace el llamado al método `printArmyHealth()`, dentro de este se llama a 3 métodos por cada ejército los cuales son algoritmos de ordenamiento que ordenarán sus soldados de mayor a menor vida, para luego mostrarlos en consola.
- En el caso 7, se hace el llamado al método `armyWinnerHealth()`, mostrando así un ejército ganador en caso sus vidas no sean iguales, y en caso de que sí, mostrará aquello.
- En el caso 8, se muestra el mensaje **Fin** ya que esta opción termina el programa.
- Y por último en caso de no ser ningún número anteriormente mencionado, se muestra un mensaje indicando que se debe seleccionar una opción válida, y se hace un llamado nuevamente a la función.



```
16 public static void mainInterfaz(){
17     Scanner sc = new Scanner(System.in);
18     System.out.println("Bienvenido...");
19
20     System.out.println("1. Crear un nuevo ejercito"
21         + "\n2. Mostrar los datos de los ejercitos"
22         + "\n3. Mostrar la tabla con los ejercitos"
23         + "\n4. Mostrar el promedio de vida de los ejercitos"
24         + "\n5. Mostrar los soldados de ejercitos con mayor vida"
25         + "\n6. Ordenar los soldados de ejercitos segun la vida"
26         + "\n7. Mostrar ejercito ganador"
27         + "\n8. Salir del juego");
28     int action = sc.nextInt();
29
30     switch (action){
31         case 1 -> { // Crear un nuevo ejercito
32             army.clear();
33             army1DA.clear();
34             army1DB.clear();
35             createArmy();
36             mainInterfaz();
37         }
38         case 2 -> { // Mostrar los datos de los ejercitos
39             System.out.println("DATOS DE LOS SOLDADOS CREADOS:\n");
40             showArmyData(army1DA, 'A');
41             showArmyData(army1DB, 'B');
42             mainInterfaz();
43         }
44         case 3 -> { // Mostrar la tabla con los ejercitos
45             showArmyTable(army);
46             mainInterfaz();
47         }
48         case 4 -> { // Mostrar el promedio de vida de los ejercitos
49             System.out.println("El promedio de vida del Ejercito A es: " +
50                 averageHealth(army1DA));
51             System.out.println("El promedio de vida del Ejercito B es: " +
52                 averageHealth(army1DB));
53             mainInterfaz();
54         }
55         case 5 -> { // Mostrar los soldados de ejercitos con mayor vida
56             moreHelath(army1DA, 'A');
57             moreHelath(army1DB, 'B');
58             mainInterfaz();
59         }
60         case 6 -> { // Ordenar los soldados de ejercitos segun la vida
61             System.out.println("\nEjercitos ordenados (bubbleSort) segun la vida: ");
62             printArmyHealth(bubbleSort(army1DA), 'A');
63             printArmyHealth(bubbleSort(army1DB), 'B');
64
65             System.out.println("\nEjercitos ordenados (insertionSort) segun la vida: ");
66             printArmyHealth(insertionSort(army1DA), 'A');
67             printArmyHealth(insertionSort(army1DB), 'B');
68
69             System.out.println("\nEjercitos ordenados (selectionSort) segun la vida: ");
70             printArmyHealth(selectionSort(army1DA), 'A');
71             printArmyHealth(selectionSort(army1DB), 'B');
```

```
70     mainInterfaz();
71 }
72 case 7 -> { // Mostrar ejercito ganador
73     armyWinnerHealth();
74     mainInterfaz();
75 }
76 case 8 -> { // Salir del juego
77     System.out.println("Fin.");
78 }
79 default -> {
80     System.out.println("Selecciona una opcion valida");
81     mainInterfaz();
82 }
83 }
84
85 sc.close();
86 }
```

- Un ejemplo de como se muestra en la siguiente imagen:



```
Bienvenido...
1. Crear un nuevo ejercito
2. Mostrar los datos de los ejercitos
3. Mostrar la tabla con los ejercitos
4. Mostrar el promedio de vida de los ejercitos
5. Mostrar los soldados de ejercitos con mayor vida
6. Ordenar los soldados de ejercitos segun la vida
7. Mostrar ejercito ganador
8. Salir del juego
█
```

Figura 9

- Extraído tal cual del laboratorio anterior (lab07) ya que no requiere modificaciones, por ende no hay un commit en el laboratorio 08 en esta sección
- Commit del lab07

Listing 13: Commit 170f535: Se concluyó el switch dentro del método mainInterfaz tomando en cuenta todos los casos (ÚLTIMO COMMIT)

```
$ git add .
$ git commit -m "Implementando el caso 7 donde se muestra un ejercito ganador o
    empate, ya teniendo en cuenta la suma de vida de los soldados, esot dentro de la
    funcion mainInterfaz"
$ git push -u origin main
```

#### 4.1.16. Método main, utilización de los métodos creados

- En el método principal (main) se utilizará simplemente dos métodos.
- Primero llamaremos al método `createArmy()` que inicializará los ejércitos, quiere decir sus Hash-Map y el tablero.
- Finalmente haciendo uso del método `mainInterfaz()` se inicia el programa, ya que llamará a este y se ejecutarán los métodos según el usuario solicite y acabará hasta que este lo decida.
- Extraído tal cual del laboratorio anterior (lab07) ya que no requiere modificaciones, por ende no hay un commit en el laboratorio 08 en esta sección

```
12 public static void main(String [] args){  
13     createArmy();  
14     mainInterfaz();  
15 }
```

- Commit del lab07

Listing 14: Commit 5410f63: Último commit, donde se implementó el método de la interfaz, sobre el cual está puesto los métodos del programa

```
$ git add .  
$ git commit -m "Terminando el código, se adaptó el interfaz al método main y se  
concluyó el método del mismo"
```

## 4.2. Diagrama de clase UML



Figura 10

### 4.3. Estructura de laboratorio 08

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

```
lab07
| Soldado.java
| VideoJuego5.java
|
|-----latex
| Informe_Lab08.pdf
| Informe_Lab08.tex
|
|-----img
| averageHealth.png
| averageHealth2.png
| diagrama08.png
| logo_abet.png
| logo_episunsa.png
| logo_unsa.jpg
| mainInterfaz.png
| moreHealth.png
| printArmyHealth.png
| showArmyTable.png
| sumHealth.png
| sumHealth2.png
| toString.png
|
|-----src
| Soldado.java
| VideoJuego5.java
```

## 5. Rúbricas

### 5.1. Entregable Informe

Tabla 1: Tipo de Informe

<b>Informe</b>	
<b>Latex</b>	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y fácil 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 cumplió 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 auto calificarse 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
<b>1. GitHub</b>	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
<b>2. Commits</b>	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
<b>4. Ejecución</b>	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
<b>5. Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
<b>6. Fechas</b>	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
<b>7. Ortografía</b>	El documento no muestra errores ortográficos.	2	X	2	
<b>8. Madurez</b>	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	
<b>Total</b>		20		19	

## 6. Referencias

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html>
- <https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html>
- <https://docs.oracle.com/javase/tutorial/java/java00/methods.html>
- <https://www.geeksforgeeks.org/selection-sort/>
- <https://www.geeksforgeeks.org/bubble-sort/>
- <https://www.geeksforgeeks.org/insertion-sort/>
- <https://es.stackoverflow.com/questions/108171/>