

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 1

INFORME DE LABORATORIO

(formato estudiante)

INFORMACIÓN BÁSICA					
ASIGNATURA:	<i>Fundamentos de la Programación 2</i>				
TÍTULO DE LA PRÁCTICA:	<i>HashMap</i>				
NÚMERO DE PRÁCTICA:	<i>08</i>	AÑO LECTIVO:	<i>2024 B</i>	NRO. SEMESTRE:	<i>//</i>
FECHA DE PRESENTACIÓN	<i>29/11/2024</i>	HORA DE PRESENTACIÓN	<i>18:30:00</i>		
INTEGRANTE (s) <i>Layme Salas Rodrigo Fabricio</i>				NOTA (0-20)	
DOCENTE(s): <i>Ing. Lino Jose Pinto Oppe</i>					

RESULTADOS Y PRUEBAS
<p>I. EJERCICIOS RESUELTOS:</p> <pre> 1 /*Propósito: Simular el tablero de ejércitos usando HashMap*/ 2 import java.util.*; 3 4 public class VideoJuego8 { 5 public static int vidaTotalAzul = 0; 6 public static int vidaTotalRojo = 0; 7 public static Soldado mayorVidaAzul = new Soldado(nombre:null, vida:0, filaYcolumna:null, equipo:null); 8 public static Soldado mayorVidaRojo = new Soldado(nombre:null, vida:0, filaYcolumna:null, equipo:null); 9 public static Soldado[] soldadosUniDimensionalAzul = new Soldado[10]; 10 public static Soldado[] soldadosUniDimensionalRojo = new Soldado[10]; </pre> <p><i>Esta es la cabecera de mi programa, usé variables globales porque son varios los métodos que lo utilizan y varias veces. Hay objetos de Soldado que me ayudarán para hallar los mejores y peores soldados, también para los ordenamientos. Además, usé arreglos estándar para los ordenamientos, los HashMap complican el ordenamiento debido a la key que es inmutable.</i></p>

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 2

```

12 public static void main(String[] args) {
13     Scanner scan = new Scanner(System.in);
14     boolean seguir = true;
15
16     HashMap<String, Soldado> tablero = new HashMap<>();
17     while (seguir) {
18         System.out.println(x:"¿Desea ejecutar el programa? (s/n)");
19         String rpt = scan.next();
20         if (rpt.equals(anObject:"s")) {
21             iniciarPrograma(tablero);
22             tablero.clear(); /*Nuevo comando, clear() para reiniciar los HashMaps que utilizo */
23             vidaTotalAzul = 0;
24             vidaTotalRojo = 0;
25             mayorVidaAzul = new Soldado(nombre:null, vida:0, filaYcolumna:null, equipo:null); /*Nueva estructura de constructor*/
26             mayorVidaRojo = new Soldado(nombre:null, vida:0, filaYcolumna:null, equipo:null); /*Por el uso de HashMap */
27             soldadosUniDimensionalAzul = new Soldado[10];
28             soldadosUniDimensionalRojo = new Soldado[10];
29         } else if (rpt.equals(anObject:"n"))
30             seguir = false;
31         else
32             System.out.println(x:"Esa no es una opción válida");
33     }
34 }

```

Este es el main, lo hace iterativo con el bucle while, modifiqué la estructura del constructor que utilizaba por otro que, con ayuda de la key, almacena fila y columna. Aquí también preparo los arreglos con datos null para los ordenamientos.

```
36 ✓ public static void iniciarPrograma(HashMap<String, Soldado> tablero) {
37     int cantidad = (int) (Math.random()*10+1);
38     int cantidadEnemiga = (int) (Math.random()*10+1);
39
40     inicializarEjercito(tablero, cantidad, color:"\u001B[44mSoldado", equipo:"azul");
41     inicializarEjercito(tablero, cantidadEnemiga, color:"\u001B[41mSoldado", equipo:"rojo");
42
43     mostrarTabla(tablero);
44     hallarSoldadoMayorVida(tablero);
45
46     System.out.println("El soldado con mayor vida del ejército azul es: " + mayorVidaAzul);
47     System.out.println("El soldado con mayor vida del ejército rojo es: " + mayorVidaRojo);
48     System.out.println("El promedio del ejército azul es: " + vidaTotalAzul/cantidad);
49     System.out.println("El promedio del ejército rojo es: " + vidaTotalRojo/cantidadEnemiga);
50
51     System.out.println(x:"DATOS DEL EJÉRCITO AZUL POR ORDEN DE INGRESO:");
52     imprimirInformacion(cantidad, soldadosUniDimensionalAzul);
53     System.out.println(x:"DATOS DEL EJÉRCITO ROJO POR ORDEN DE INGRESO:");
54     imprimirInformacion(cantidadEnemiga, soldadosUniDimensionalRojo);
55
56     rankingDePoder(cantidad, soldadosUniDimensionalAzul);
57     ordenarSeleccion(cantidadEnemiga, soldadosUniDimensionalRojo);
58
59     System.out.println(x:"DATOS DEL EJÉRCITO AZUL ORDENADO POR NIVEL DE VIDA:");
60     imprimirInformacion(cantidad, soldadosUniDimensionalAzul);
61     System.out.println(x:"DATOS DEL EJÉRCITO ROJO ORDENADO POR NIVEL DE VIDA:");
62     imprimirInformacion(cantidadEnemiga, soldadosUniDimensionalRojo);
63
64     mostrarGanador();
65 }
```

Este bloque es el motor del programa, están los métodos que explicaré más adelante. Aquí se inicializan ambos ejércitos y se les diferencia por colores, se muestra la tabla inicial y con los soldados inicializados en el HashMap y en el arreglo estándar se hacen búsquedas y ordenamientos. Lo demás es la impresión de los datos y salida del ganador, gana el ejército con mayor vida total, ese es el criterio.

```

67 public static void inicializarEjercito(HashMap<String, Soldado> tablero, int cantidad, String color, String equipo) {
68     int contadorIndiceSoldadoAzul = 0;
69     int contadorIndiceSoldadoRojo = 0;
70
71     while (cantidad > 0) {
72         String key = (int) (Math.random()*10) + "x" + (int) (Math.random()*10); /*Ahora uso keys para la posición y nombre */
73
74         if (!tablero.containsKey(key)) {
75             Soldado soldado = new Soldado(color + key + "\u001B[0m", (int) (Math.random()*5+1), key, equipo);
76             tablero.put(key, soldado); /*Cambio al método put() para insertar los Soldados */
77             cantidad--;
78
79             if (equipo.equals(anObject:"azul")) {
80                 vidaTotalAzul += soldado.getVida();
81                 soldadosUniDimensionalAzul[contadorIndiceSoldadoAzul] = soldado;
82                 contadorIndiceSoldadoAzul++;
83             } else {
84                 vidaTotalRojo += soldado.getVida();
85                 soldadosUniDimensionalRojo[contadorIndiceSoldadoRojo] = soldado;
86                 contadorIndiceSoldadoRojo++;
87             }
88         }
89     }
90 }

```

En este bloque de código se inicializan los objetos Soldado con su key que guarda la posición en la simulación de tablero. Con la cantidad generada se insertan los soldados al HashMap con posiciones aleatorias con put(), el nuevo método de HashMaps. Mientras se insertan Soldados también se modifican las variables globales, vida total de cada ejército y arreglos.

```

105 public static void hallarSoldadoMayorVida(HashMap<String, Soldado> tablero) {
106     for (Soldado soldado : tablero.values()) { /*Itera solo en los Soldados del HashMap */
107         if (soldado.getEquipo().equals(anObject:"azul") && soldado.getVida() > mayorVidaAzul.getVida())
108             mayorVidaAzul = soldado;
109         if (soldado.getEquipo().equals(anObject:"rojo") && soldado.getVida() > mayorVidaRojo.getVida())
110             mayorVidaRojo = soldado;
111     }
112 }

```

Aquí se halla el Soldado con mayor nivel de vida, se usa for each sólo sobre los valores del HashMap y no sobre las keys. Estos reemplazos se hacen en la variables globales declaradas.

```

114 public static void rankingDePoder(int cantidad, Soldado[] soldadosUniDimensional) {
115     boolean intercambio = true;
116     while (intercambio) {
117         intercambio = false;
118         for (int i = 0; i < cantidad - 1; i++)
119             if (soldadosUniDimensional[i].getVida() < soldadosUniDimensional[i + 1].getVida()) {
120                 intercambio = true;
121                 Soldado temp = new Soldado(nombre:null, vida:0, filaYcolumna:null, equipo:null);
122                 temp = soldadosUniDimensional[i + 1];
123                 soldadosUniDimensional[i + 1] = soldadosUniDimensional[i];
124                 soldadosUniDimensional[i] = temp;
125             }
126     }
127 }

```

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 5</p>

Este es el primer ordenamiento, burbuja, funciona sobre el arreglo estándar para ahorrar memoria. Usando técnica de variables temporales se hace el intercambio sin problema, hacerlo en HashMap hubiera ocasionado bastante conflicto por la naturaleza de la key al momento de intercambiar Soldados.

```

129  public static void ordenarSeleccion(int cantidad, Soldado[] soldadosUniDimensional) {
130      for (int i = 0; i < cantidad-1; i++) {
131          int menor = i;
132          for (int j = i + 1; j < cantidad; j++)
133              if (soldadosUniDimensional[j].getVida() > soldadosUniDimensional[menor].getVida())
134                  menor = j;
135          Soldado temp = soldadosUniDimensional[menor];
136          soldadosUniDimensional[menor] = soldadosUniDimensional[i];
137          soldadosUniDimensional[i] = temp;
138      }
139  }

```

Este es el segundo ordenamiento, por selección, es también usa variables temporales pero buscará siempre el menor de cada iteración, también funciona a la perfección.

```

141  public static void imprimirInformacion(int cantidad, Soldado[] soldadosUniDimensional) {
142      for (int i = 0; i < cantidad; i++) {
143          System.out.println("SOLDADO " + i + ":");
144          System.out.println(soldadosUniDimensional[i].toString());
145      }
146  }

```

Este es un método simple, solo imprime la información usando el arreglo estándar, también para consumir menos memoria.

```

148  public static void mostrarGanador() {
149      if (vidaTotalAzul > vidaTotalRojo)
150          System.out.println("¡El ejercito azul gana por mayor nivel de vida! " + "\nAzul " + vidaTotalAzul + ":" + vidaTotalRojo + " Rojo");
151      else if (vidaTotalAzul < vidaTotalRojo)
152          System.out.println("¡El ejercito rojo gana por mayor nivel de vida! " + "\nAzul " + vidaTotalAzul + ":" + vidaTotalRojo + " Rojo");
153      else
154          System.out.println("¡Ha ocurrido un empate! " + "\nAzul " + vidaTotalAzul + ":" + vidaTotalRojo + " Rojo");
155  }
156  }

```

Con este bloque el programa elige al ganador en base a qué ejército tiene más vida en total, si hay empate, se imprime este mensaje. Luego se regresa al main, preguntando si se volverá a iniciar la batalla.

EJECUTANDO EL PROGRAMA:

```

¿Desea ejecutar el programa? (s/n)
s

```

									Soldado1X9
							Soldado5X7	Soldado5X8	
					Soldado6X5				
				Soldado8X4					
						Soldado9X6			

```

El soldado con mayor vida del ejército azul es: Nombre: Soldado6X5 | Vida: 5 | Fila y Columna: 6X5
El soldado con mayor vida del ejército rojo es: Nombre: Soldado1X9 | Vida: 5 | Fila y Columna: 1X9
El promedio del ejército azul es: 3
El promedio del ejército rojo es: 3

```

Se muestra el tablero con los soldados diferenciados con colores y los datos generales.

```

DATOS DEL EJÉRCITO AZUL POR ORDEN DE INGRESO:
SOLDADO 0:
Nombre: Soldado6X5 | Vida: 5 | Fila y Columna: 6X5
SOLDADO 1:
Nombre: Soldado9X6 | Vida: 1 | Fila y Columna: 9X6
SOLDADO 2:
Nombre: Soldado5X7 | Vida: 5 | Fila y Columna: 5X7
DATOS DEL EJÉRCITO ROJO POR ORDEN DE INGRESO:
SOLDADO 0:
Nombre: Soldado8X4 | Vida: 2 | Fila y Columna: 8X4
SOLDADO 1:
Nombre: Soldado5X8 | Vida: 2 | Fila y Columna: 5X8
SOLDADO 2:
Nombre: Soldado1X9 | Vida: 5 | Fila y Columna: 1X9



```

Se muestran los ejércitos por orden de ingreso y sus datos.

```

DATOS DEL EJÉRCITO AZUL ORDENADO POR NIVEL DE VIDA:
SOLDADO 0:
Nombre: Soldado6X5 | Vida: 5 | Fila y Columna: 6X5
SOLDADO 1:
Nombre: Soldado5X7 | Vida: 5 | Fila y Columna: 5X7
SOLDADO 2:
Nombre: Soldado9X6 | Vida: 1 | Fila y Columna: 9X6

```

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 7</p>

Acá se usa el ordenamiento burbuja para ordenar del que tiene más nivel de vida al que tiene menos.

```

DATOS DEL EJÉRCITO ROJO ORDENADO POR NIVEL DE VIDA:
SOLDADO 0:
Nombre: Soldado1X9 | Vida: 5 | Fila y Columna: 1X9
SOLDADO 1:
Nombre: Soldado5X8 | Vida: 2 | Fila y Columna: 5X8
SOLDADO 2:
Nombre: Soldado8X4 | Vida: 2 | Fila y Columna: 8X4

```

Se hace lo mismo con el ejército rojo pero con el ordenamiento por selección.

```

¡El ejercito azul gana por mayor nivel de vida!
Azul 11:9 Rojo
¿Desea ejecutar el programa? (s/n)

```

Finalmente, se muestra el ganador y se pregunta si se quiere volver a ejecutar

II. PRUEBAS

¿Con qué valores comprobaste que tu práctica estuviera correcta?

Con valores int, String, boolean y con pruebas en la parte de key del HashMap para encontrar el mejor tipo de dato, usé String.

¿Qué resultado esperabas obtener para cada valor de entrada?

Esperaba que el HashMap recibiera un String para la key y el objeto Soldado para el valor, eso funcionó pero también había pensado en usar un HashMap bidimensional, pero hubiera generado una key innecesaria al crear el tablero, por eso opte por una simulación de tablero con un HashMap normal.

¿Qué valor o comportamiento obtuviste para cada valor de entrada?

Obtuve, al final, el correcto, una secuencia limpia de los métodos usados en el main y sin ningún error.

III. CUESTIONARIO:

PRUEBAS DE COMMIT HECHO EN GIT BASH:

Primer commit:

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 8</p>

```
ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LAYME_SALAS_LABORATORIO_07/LAYME_SALAS_LABORATORIO_07.pdf
    LAYME_SALAS_LABORATORIO_08/

nothing added to commit but untracked files present (use "git add" to track)

ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git add .

Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (7/7), 670.00 KiB | 18.61 MiB/s, done.
Total 7 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/F4brici0L4yme/PF2.git
   ad0fca7..ba46f16  main -> main


ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
```

Segundo commit:

```
ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git log
commit d310762fc0438c4948f29d03d756ef8f6db5f09c (HEAD -> main, origin/main, origin/HEAD)
Author: F4brici0L4yme <rlaymes@unsa.edu.pe>
Date:   Tue Nov 26 10:53:24 2024 -0500

    Implementé HashMap en el tablero y eliminé 2 atributos
```

Tercer commit:

	<p style="text-align: center;">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p style="text-align: center;">Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 9</p>

```

ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LAYME_SALAS_LABORATORIO_08/Soldado.java
        modified:   LAYME_SALAS_LABORATORIO_08/VideoJuego8.java

no changes added to commit (use "git add" and/or "git commit -a")

ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git add .

ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   LAYME_SALAS_LABORATORIO_08/Soldado.java
        modified:   LAYME_SALAS_LABORATORIO_08/VideoJuego8.java

ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git commit -m "Programa ahora con comentarios"
[main b28a1fa] Programa ahora con comentarios
 2 files changed, 15 insertions(+), 20 deletions(-)

ASUS@DESKTOP-J2KJOAM MINGW64 /d/FP2 LABORATORIO/PF2 (main)
$ git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 883 bytes | 883.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/F4brici0L4yme/PF2.git
 d310762..b28a1fa  main -> main

```

LINK A MI REPOSITORIO DE GIT HUB: <https://github.com/F4brici0L4yme/PF2.git>

CONCLUSIONES

Sobre el ordenamiento con HashMap, consumirá más memoria y línea al iterar en el tablero 10x10. Además, el que la key sea inmutable, hace dificultoso el intercambio que hasta pediría un HashMap temporal para realizar el intercambio. Por eso elegí un arreglo estándar para hacer los ordenamientos con solo la cantidad de soldados que se generó y no los 100 espacios de HashMap. A pesar de todo, HashMap es increíble para mantener un orden con keys para múltiples valores, me ayudó a reducir línea de código y atributos.

	<p>UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
<p>Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación</p>		
<p>Aprobación: 2022/03/01</p>	<p>Código: GUIA-PRLE-001</p>	<p>Página: 10</p>

METODOLOGÍA DE TRABAJO

Usé las mismas que fui usando durante estos laboratorios, comentar bloques de código para poder concentrarme en una parte y revisando problemas pasados y similares que ya resolví para tener una idea y construir un nuevo método.

REFERENCIAS Y BIBLIOGRAFÍA

E. G. Castro Gutiérrez y M. W. Aedo López, Fundamentos de programación 2: tópicos de programación orientada a objetos, 1st ed. Arequipa, Perú: Universidad Nacional de San Agustín, 2021, pp. 170, ISBN 978-612-5035-20-2.

RÚBRICA DE CALIFICACIÓN DE LABORATORIO

(EN LA SIGUIENTE PÁGINA)

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	2	

	<p align="center">UNIVERSIDAD NACIONAL DE SAN AGUSTIN FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMA</p>	
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		
Aprobación: 2022/03/01	Código: GUIA-PRLE-001	Página: 12

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		19	