Informe de Laboratorio 06

Tema: ArrayList

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
06	ArrayList	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 Laboratorio6.
- Usted deberá crear las dos clases Soldado.java y VideoJuego3.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. Pero ahora el tablero debe ser un ArrayList bidimensional.
- Tendrá 2 Ejércitos. 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. Finalmente, que muestre qué ejército ganará la batalla (indicar la métrica usada para decidir al ganador de la batalla).

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home Single Language 64 bits 22621.2283
- 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 06 en el Repositorio GitHub.
- `https://github.com/RyanValdivia/fp2-23b/tree/main/fase02/lab06`

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 b13def7c362a6359bf680d2606fd7da6dfe93911
Author: RYAN VALDIVIA <rvaldiviase@unsa.edu.pe>
Date:  Mon Oct 16 09:29:53 2023 -0500
   Anadiendo la clase Soldado para poder crear la lista bidimensional
```

- 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
12     public void setNombre(String s) {
13         this.nombre = s;
14     }
15
16     public void setVida(int n) {
```

```
        this.vida = n;
    }

    public void setFila(int n) {
        this.fila = n;
    }

    public void setColumna(int n) {
        this.columna = n;
    }

    public String getNombre() {
        return nombre;
    }

    public int getVida() {
        return vida;
    }

    public int getFila() {
        return fila;
    }

    public int getColumna() {
        return columna;
    }
}
```

- Para este problema, reutilicé diferentes cosas que hice en el laboratorio anterior, como el sistema para crear las coordenadas de los soldados, usando arreglos de números aleatorios.

Listing 3: Números aleatorios

```
  public static int[] numerosRandom(int q) {
        int[] nums = new int[q];
        for (int i = 0; i < nums.length; i++) {
            nums[i] = nums.length;
        }
        for (int i = 0; i < q; i++) {
            int n;
            do {
                n = (int) (Math.random() * 10);
            } while (estaEnArreglo(nums, n, i));
            nums[i] = n;
        }
        return nums;
    }
    public static boolean estaEnArreglo(int[] arreglo, int num, int indice) {
        for (int i = 0; i < indice; i++) {
            if (arreglo[i] == num) {
                return true;
            }
        }
        return false;
    }
```

- Solo que, en este caso, son dos ejércitos y dos soldados de ambos ejércitos no pueden estar en la misma casilla, asi que adapté este código para que no se generen dos pares ordenados iguales.

Listing 4: Método main

```
66      Scanner sc = new Scanner(System.in);
67        ArrayList<ArrayList<Soldado>> tablero = new ArrayList<>(10);
68        int ej1 = (int) ((Math.random() * 10) + 1);
69        int ej2 = (int) ((Math.random() * 10) + 1);
70        int[] filas1 = numerosRandom(ej1);
71        int[] columnas1 = numerosRandom(ej1);
72        int[] filas2;
73        int[] columnas2;
74        do {
75            filas2 = numerosRandom(ej2);
76            columnas2 = numerosRandom(ej2);
77        } while (!diffCoordenadas(filas1, filas2, columnas1, columnas2));
```

- Aqui primero inicialicé los objetos necesarios, como el Scanner (que usaré después) y la Lista Bidimensional, además de conseguir los arreglos que formaran las coordenadas de los objetos de ambos ejércitos, asegurándome, con esa estructura do-while, que dos pares ordenados de ambos ejércitos jamás sean iguales, para que dos soldados nunca estén en una misma casilla, usando otro método.

Listing 5: Coordenadas diferentes

```
78  public static boolean diffCoordenadas(int[] filas1, int[] filas2, int[] columnas1, int[]
        columnas2) {
79      if (filas1.length > filas2.length) {
80          for (int i = 0; i < filas2.length; i++) {
81              if (filas1[i] == filas2[i] && columnas1[i] == columnas2[i]) {
82                  return false;
83              }
84          }
85      } else {
86          for (int i = 0; i < filas1.length; i++) {
87              if (filas1[i] == filas2[i] && columnas1[i] == columnas2[i]) {
88                  return false;
89              }
90          }
91      }
92      return true;
93  }
```

- Este método comprueba que, dados dos arreglos de coordenadas, no hayan pares ordenados iguales.
- Lo siguiente era inicializar el arreglo para empezar a añadir los soldados, ya que ya tenemos sus localizaciones. Para esto, creé un método específico para inicializar el ArrayList.

Listing 6: Inicializar la lista

```
94    public static void inicializarLista(ArrayList<ArrayList<Soldado>> army) {
```

```
95          for (int i = 0; i < 10; i++) {
96              army.add(new ArrayList<>());
97          }
98          for (int i = 0; i < 10; i++) {
99              for (int j = 0; j < 10; j++) {
100                 army.get(i).add(new Soldado());
101                 army.get(i).get(j).setNombre("      ");
102             }
103         }
104
105     }
```

- Este método inicia instanciando las Listas (Para utilizar una lista bidimensional) y posteriormente, instancia los objetos en cada lista con el atributo de nombre como un String vacío, esto es para comodidad al momento de imprimir todo el tablero.
- Una vez ya inicializado e instanciado, toca añadir a todos los soldados, dados las coordenadas ya generadas de forma aleatoria, para esto, creé otro método.

Listing 7: Desplegando nuestras tropas

```
106     public static void desplegarEjercito(ArrayList<ArrayList<Soldado>> army, int[] filas,
            int[] columnas, int ej) {
107         for (int i = 0; i < filas.length; i++) {
108             int v = (int) ((Math.random() * 5) + 1);
109             army.get(filas[i]).get(columnas[i]).setNombre("Soldado" + i + "X" + ej);
110             army.get(filas[i]).get(columnas[i]).setVida(v);
111             army.get(filas[i]).get(columnas[i]).setFila(filas[i]);
112             army.get(filas[i]).get(columnas[i]).setColumna(columnas[i]);
113         }
114     }
```

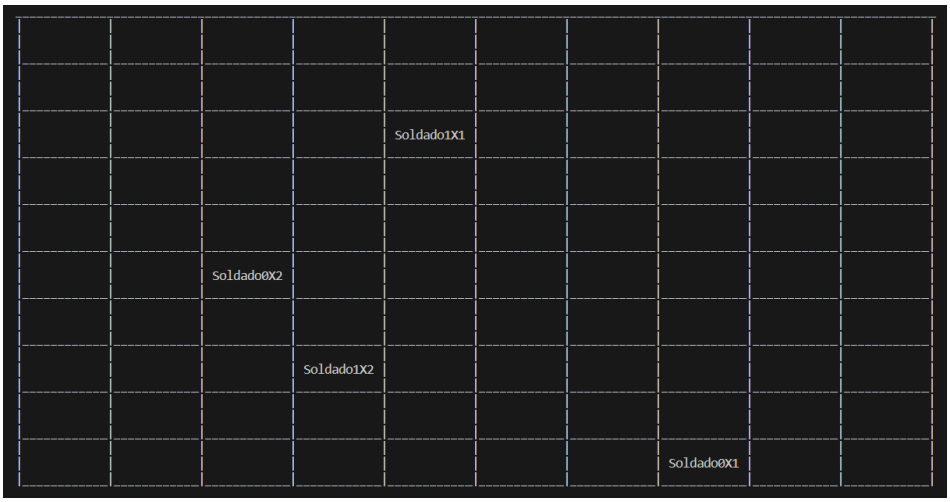
- Una vez ya desplegados los dos ejércitos, viene la parte algo complicada, mostrar el tablero, para lo cual reciclaré el código del anterior laboratorio, modificandolo para que funcione con las nuevas medidas.
- Comenzaré con el método para mostrar el tablero, imprimiendo los atributos 'Nombre' de todos los objetos de la lista (Por eso fue que inicialicé todos los objetos con un String vacío por defecto).

Listing 8: Mostrar el tablero

```
115     public static void mostrarTablero(ArrayList<ArrayList<Soldado>> army) {
116         System.out.println(crearTecho());
117         for (int i = 0; i < 10; i++) {
118             System.out.println(separadorSup());
119             for (int j = 0; j < 10; j++) {
120                 if (j == 10 - 1) {
121                     System.out.print("| " + army.get(i).get(j).getNombre() + " |");
122                 } else {
123                     System.out.print("| " + army.get(i).get(j).getNombre() + " ");
124                 }
125             }
126             System.out.println();
127             System.out.println(separadorInf());
```

```
128         }
129     }
130     public static String crearTecho() {
131         String franky = "";
132         for (int i = 0; i < 131; i++) {
133             franky += "_";
134         }
135         return franky;
136     }
137 
138     public static String separadorInf() {
139         String franky = "";
140         for (int i = 0; i < 131; i++) {
141             if (i % 13 == 0) {
142                 System.out.print("|");
143             } else {
144                 System.out.print("_");
145             }
146         }
147         return franky;
148     }
149 
150     public static String separadorSup() {
151         String franky = "";
152         for (int i = 0; i < 131; i++) {
153             if (i % 13 == 0) {
154                 System.out.print("|");
155             } else {
156                 System.out.print(" ");
157             }
158         }
159         return franky;
160     }
```

- Imprimiendo esto al momento de ejecutar el código.



- 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, decidí crear un arreglo simple a partir de la lista bidimensional que ya tenía, para que sea mucho más sencillo de trabajar luego.

Listing 9: De Lista a Arreglo

```
161     public static ArrayList<Soldado> crearLista(ArrayList<ArrayList<Soldado>> army, int[]
            filas, int[] columnas) {
162         ArrayList<Soldado> nuevo = new ArrayList<>();
163         for (int i = 0; i < filas.length; i++) {
164             nuevo.add(army.get(filas[i]).get(columnas[i]));
165         }
166         return nuevo;
167     }
168     public static Soldado[] convertirArray(ArrayList<Soldado> army) {
169         Soldado[] nuevo = new Soldado[army.size()];
170         for (int i = 0; i < nuevo.length; i++) {
171             nuevo[i] = army.get(i);
172         }
173         return nuevo;
174     }
```

- Con estos métodos, convierto mi lista bidimensional en una lista normal, y luego en un arreglo simple.
- Además, reusé el método para mostrar un soldado y mostrar el ejército a partir de un arreglo.

Listing 10: Mostrar soldado y ejército

```
175     public static void mostrarSoldado(Soldado[] army, int i) {
176         String columna;
177         System.out.println("Nombre: " + army[i].getNombre());
178         System.out.println("Vida: " + army[i].getVida() + " HP");
179         switch (army[i].getColumna() + 1) {
180             case 1:
181                 columna = "A";
182                 break;
183             case 2:
184                 columna = "B";
185                 break;
186             case 3:
187                 columna = "C";
188                 break;
189             case 4:
190                 columna = "D";
191                 break;
192             case 5:
193                 columna = "E";
194                 break;
195             case 6:
196                 columna = "F";
197                 break;
198             case 7:
199                 columna = "G";
200                 break;
201             case 8:
```

```
202                 columna = "H";
203                 break;
204             case 9:
205                 columna = "I";
206                 break;
207             case 10:
208                 columna = "J";
209                 break;
210             default:
211                 columna = "K";
212                 break;
213         }
214         System.out.println("Posicion: " + (army[i].getFila() + 1) + "-" + columna);
215     }
216 
217     public static void mostrarEjercito(Soldado[] army, int ej) {
218         System.out.println("Ejercito " + ej);
219         for (int i = 0; i < army.length; i++) {
220             mostrarSoldado(army, i);
221         }
222         System.out.println();
223     }
```

- Ahora si, muestro el soldado con mayor nivel de vida de cada ejército, además del nivel total y promedio de vida de cada ejército.

Listing 11: Mayor nivel de vida

```
224     public static void soldadoMayorVida(Soldado[] army, int ej) {
225         int max = 0;
226         for (int i = 0; i < army.length; i++) {
227             if (army[i].getVida() > army[max].getVida()) {
228                 max = i;
229             }
230         }
231         System.out.println("El soldado con mayor vida del ejercito " + ej + " es: ");
232         mostrarSoldado(army, max);
233         System.out.println();
234     }
235      public static void mostrarTotalYPromedio(Soldado[] army, int ej) {
236         int total = 0;
237         for (int i = 0; i < army.length; i++) {
238             total += army[i].getVida();
239         }
240         System.out.println("El nivel de vida total del ejercito " + ej + " es: " + total);
241         System.out.println("El nivel de vida promedio del ejercito " + ej + " es: " + total *
                1.0 / army.length);
242         System.out.println();
243     }
```

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

```
El soldado con mayor vida del ejercito 1 es:
Nombre: Soldado0X1
Vida: 3 HP
Posición: 10-H

El soldado con mayor vida del ejercito 2 es:
Nombre: Soldado0X2
Vida: 3 HP
Posición: 6-C

El nivel de vida total del ejercito 1 es: 5
El nivel de vida promedio del ejercito 1 es: 2.5

El nivel de vida total del ejercito 2 es: 5
El nivel de vida promedio del ejercito 2 es: 2.5
```

- También realicé el método para mostrar todo el ejército según el orden de creación de los objetos.

Listing 12: Mostrando el ejército

```
244  public static void mostrarEjercito(Soldado[] army, int ej) {
245      System.out.println("Ejercito " + ej);
246      for (int i = 0; i < army.length; i++) {
247          mostrarSoldado(army, i);
248      }
249      System.out.println();
250  }
```

- Mostrando lo siguiente al momento de ejecutar.

```
Ejercito 1
Nombre: Soldado0X1
Vida: 3 HP
Posición: 10-H
Nombre: Soldado1X1
Vida: 2 HP
Posición: 3-E

Ejercito 2
Nombre: Soldado0X2
Vida: 3 HP
Posición: 6-C
Nombre: Soldado1X2
Vida: 2 HP
Posición: 8-D
```

- A continuación falta lo último, que es ordenar los soldados, mostrarlos ya ordenados y determinar cuál ejército gana.
- Entonces, reciclé los algoritmos de ordenamiento del laboratorio anterior.

Listing 13: Algoritmos de ordenamiento

```
251  public static void ordenamientoInsercion(Soldado[] army) {
252      for (int i = 1; i < army.length; i++) {
253          Soldado valor = army[i];
254          int j = i;
255          for (j = i; 0 < j && army[j - 1].getVida() < valor.getVida(); j--) {
256              army[j] = army[j - 1];
257          }
258          army[j] = valor;
259      }
260  }
261
262  public static void ordenamientoBurbuja(Soldado[] army) {
263      for (int i = 0; i < army.length; i++) {
264          for (int j = 0; j < army.length - 1; j++) {
265              if (army[j].getVida() < army[j + 1].getVida()) {
266                  intercambiar(army, j, j + 1);
267              }
268          }
269      }
270  }
271
272  public static void intercambiar(Soldado[] flota, int i, int j) {
273      Soldado temp;
274      temp = flota[i];
275      flota[i] = flota[j];
276      flota[j] = temp;
277  }
```

- Para esto, una vez ya implementado, hicé un menú para el usuario como en el anterior trabajo, en el método main, además de armar todos los métodos realizados.

Listing 14: Método main

```
278     Soldado[] ejercito1 = convertirArray(crearLista(tablero, filas1, columnas1));
279       Soldado[] ejercito2 = convertirArray(crearLista(tablero, filas2, columnas2));
280       System.out.println();
281       soldadoMayorVida(ejercito1, 1);
282       soldadoMayorVida(ejercito2, 2);
283       mostrarTotalYPromedio(ejercito1, 1);
284       mostrarTotalYPromedio(ejercito2, 2);
285       mostrarEjercito(ejercito1, 1);
286       mostrarEjercito(ejercito2, 2);
287
288       System.out.println("Bajo que criterio le gustaria ordenar los ejercitos?");
289       System.out.println("1. Burbuja");
290       System.out.println("2. Insercion");
291       switch (sc.nextInt()) {
292           case 1:
293               ordenamientoBurbuja(ejercito1);
294               ordenamientoBurbuja(ejercito2);
295               break;
296           case 2:
297               ordenamientoInsercion(ejercito1);
298               ordenamientoBurbuja(ejercito2);
299               break;
300           default:
301       }
302       System.out.println("Ranking segun vida (Del mayor al menor): ");
303       mostrarEjercito(ejercito1, 1);
304       mostrarEjercito(ejercito2, 2);
```

- Para probar esta parte, hice otra ejecución del programa, ya que los ejércitos ya estaban ordenados.
- Siendo el ejército ordenado por orden de creación el siguiente:

```
Ejercito 1
Nombre: Soldado0X1
Vida: 3 HP
Posición: 10-A
Nombre: Soldado1X1
Vida: 4 HP
Posición: 4-B

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

- Quedando así una vez ya ordenados:

```
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 15: Determinando el ganador

```
305    public static void ejercitoGanador(Soldado[] army1, Soldado[] army2) {
```

```
306         int total1 = 0;
307         int total2 = 0;
308         for (int i = 0; i < army1.length; i++) {
309             total1 += army1[i].getVida();
310         }
311         for (int i = 0; i < army2.length; i++) {
312             total2 += army2[i].getVida();
313         }
314         if (total1 > total2) {
315             System.out.println("El ejercito 1 es ganador!");
316         } else if (total1 == total2) {
317             System.out.println("Empate");
318         } else {
319             System.out.println("El ejercito 2 es ganador");
320         }
321         System.out.println("Bajo el criterio de que ejercito tiene mas vida total");
322 
323     }
```

- Imprimiendo lo siguiente:

```
El ejercito 2 es ganador
Bajo el criterio de que ejercito tiene mas vida total
```

- Ya que el ejército 2 tiene 15 de vida, mientras que el 1 tiene 7.

```
El nivel de vida total del ejercito 1 es: 7
El nivel de vida promedio del ejercito 1 es: 3.5

El nivel de vida total del ejercito 2 es: 15
El nivel de vida promedio del ejercito 2 es: 3.75
```

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 cumplio 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 lo items.
- 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 estan 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)