

# Informe de Laboratorio 12

## Tema: Definición de Clases de Usuario

Nota

Estudiante	Escuela	Asignatura
Jhonatan David Arias Quispe jariasq@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
12	Definición de Clases de Usuario	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 4 Diciembre 2023	Al 11 Diciembre 2023

### 1. Actividades

- Al ejecutar el videojuego, el programa deberá dar las opciones:
- 1. Juego rápido (tal cual como en el laboratorio 11) Al acabar el juego mostrar las opciones de volver a jugar y de volver al menú principal. También se deberá tener la posibilidad de cancelar el juego actual en cualquier momento, permitiendo escoger entre empezar un juego totalmente nuevo o salir al menú principal.
- 2. Juego personalizado: permite gestionar ejércitos. Primero se generan los 2 ejércitos con sus respectivos soldados y se muestran sus datos. Luego se tendrá que escoger cuál de los 2 ejércitos se va a gestionar, después se mostrarán las siguientes opciones:
  - Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)
  - Eliminar Soldado (no debe permitir un ejército vacío)
  - Clonar Soldado (crea una copia exacta del soldado) y se añade al final del ejército (recordar que límite es de 10 soldados por ejército)
  - Modificar Soldado (con submenú para cambiar alguno de los atributos nivelAtaque, nivelDefensa, vidaActual)
  - Comparar Soldados (verifica si atributos: nombre, nivelAtaque, nivelDefensa, vidaActual y vive son iguales)
  - Intercambiar Soldados (intercambia 2 soldados en sus posiciones en la estructura de datos del ejército)

- Ver soldado (Búsqueda por nombre)
  - Ver ejército
  - Sumar niveles (usando Method-Call Chaining), calcular las sumatorias de nivelVida, nivelAtaque, nivelDefensa, velocidad de todos los soldados de un ejército
    - Por ejemplo, si ejército tendría 3 soldados:
    - `s=s1.sumar(s2).sumar(s3);`
    - `s` es un objeto Soldado nuevo que contendría las sumatorias de los 4 atributos indicados de los 3 soldados. Ningún soldado cambia sus valores
  - Jugar (se empezará el juego con los cambios realizados) y con las mismas opciones de la opción 1.
  - Volver (muestra el menú principal) Después de escoger alguna de las opciones 1) a 9) se podrá volver a elegir uno de los ejércitos y se mostrarán las opciones 1) a 11)
- 3. Salir

## 2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernell
- NeoVim
- OpenJDK 64-Bit 20.0.1
- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Creacion de programas con CLI
- Biblioteca Graphics (origen propio)

## 3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/JhonatanDczel/fp2-23b.git`
- URL para el laboratorio 12 en el Repositorio GitHub.
- `https://github.com/JhonatanDczel/fp2-23b/tree/main/fase02/lab12`
- El trabajo de este laboratorio es en su mayor parte lo mismo que en otros laboratorios, por lo que las variaciones serán mínimas

## 4. Proyecto lab12

- Creamos un directorio en fase02 que contenga los archivos del laboratorio y copiamos los archivos del anterior laboratorio.
- Para el tablero se usará un array bidimensional simple.
- La estructura del laboratorio presente es:

```
>>> lab12 git:(main) tree
.
├── latex
│   ├── acts
│   │   ├── a1.tex
│   │   ├── a2.tex
│   │   ├── a3.tex
│   │   ├── a4.tex
│   │   ├── a5.tex
│   │   ├── commits.tex
│   │   ├── exec.tex
│   │   └── main.tex
│   ├── foot
│   │   ├── rubricas.aux
│   │   └── rubricas.tex
│   ├── head
│   │   ├── codeFormat.tex
│   │   ├── dataTable.tex
│   │   ├── format.tex
│   │   ├── labData.tex
│   │   └── pkgs.tex
│   └── img
│       ├── exec.png
│       ├── logo_abet.png
│       ├── logo_episunsa.png
│       ├── logo_unsa.jpg
│       └── tree.jpg
├── informe-latex.aux
├── informe-latex.log
├── informe-latex.out
├── informe-latex.pdf
├── informe-latex.tex
├── Soldado.class
├── Soldado.java
├── VideoJuego.class
└── VideoJuego.java

6 directories, 29 files
```

## 5. Clase Soldado

- Los atributos para la clase soldado son:

```
1 private String nombre;
2 private int fila;
3 private int nivelAtaque = random(5);
4 private int nivelDefensa = random(5);
5 private int columna;
6 private int nivelVida;
7 private int vidaActual;
8 private int velocidad;
9 private String actitud;
10 private boolean vive;
11 private String team;
```

- Cada atributo que lo requiere, tiene sus metodos setters y getters para encapsular la información.
- Se usan 3 constructores sobrecargados que son los siguientes:

```
1 public Soldado(String t) {
2     team = t;
3     velocidad = 0;
4     vive = true;
5     actitud = "ataque";
6
7 }
8 public Soldado(int v, String t) {
9     team = t;
10    velocidad = v;
11    vive = true;
12    actitud = "ataque";
13 }
14 public Soldado(int v, int nV, String t) {
15     team = t;
16     vive = true;
17     velocidad = v;
18     nivelVida = nV;
19     actitud = "ataque";
20 }
```

- Estos constructores nos servirán cuando vayamos a crear soldados con distintos datos base.
- Adicionalmente tenemos los metodos de accion del soldado:

```
1 public void atacar() {
2     actitud = "ofensiva";
3 }
4 public void defender() {
5     actitud = "defensiva";
6 }
7 public void huir() {
8     actitud = "fuga";
9     velocidad += 2;
10 }
11 public void avanzar() {
12     velocidad += 1;
13 }
14
15 public void serAtacado() {
16     vidaActual -= 1;
17     if(vidaActual == 0) morir();
18 }
19 public void morir() {
20     vive = false;
21 }
22
23 public void retroceder() {
24     if (velocidad > 0) {
25         velocidad = 0;
26         actitud = "defensiva";
27     } else if (velocidad == 0) {
28         velocidad = -1;
29     }
30 }
```

- Las acciones cambian los estados de los atributos del soldado, a los que accederemos despues con los metodos accesores:

```
1 public String getTeam() {
2     return team;
3 }
4
5 public void setNombre(String n) {
6     nombre = n;
7 }
8
9 public void setFila(int f) {
10    fila = f;
11 }
12
13 public void setColumna(int c) {
14    columna = c;
15 }
16
17 public void setNivelVida(int p) {
18    nivelVida = p;
19 }
20
21 public String getNombre() {
22    return nombre;
23 }
24
25 public int getFila() {
26    return fila;
27 }
28
29 public int getColumna() {
30    return columna;
31 }
32
33 public int getNivelVida() {
34    return nivelVida;
35 }
36
37 public int getNivelAtaque() {
38    return nivelAtaque;
39 }
40
41 public int getNivelDefensa() {
42    return nivelDefensa;
43 }
44
45 public void setNivelAtaque(int n) {
46    nivelAtaque = n;
47 }
48
49 public void setNivelDefensa(int n) {
50    nivelDefensa = n;
51 }
52
53 public boolean isLive() {
54    return vive;
55 }
```

- Estos metodos accesoros nos servirán para manejar la lógica interna del videojuego
- Adicionalmente tenemos algunos metodos auxiliares que usamos en la misma clase:

```
1 public String toString() {
2     return "Nombre: " + nombre +
```

```
3      " | Ubicacion: " + fila + ", " + column +  
4      " | nivelVida: " + nivelVida +  
5      " | Estado: " + (vive ? "Vivo" : "Muerto") +  
6      " | Actitud: "+ actitud +"\n" ;  
7  }  
8  private int random(int n) {  
9      return (int) (Math.random() * n + 1);  
10 }
```

## 6. Clase VideoJuego

La clase VideoJuego contiene la logica principal y es la clase que contiene el metodo main, por lo tanto, el que debería llamarse para ejecutar el juego. La estructura de la clase esta dada de la siguiente manera:

- Metodos:
- juegoRapido
- juegoPersonalizado
- crearSoldado
- eliminarSoldado
- clonarSoldado
- modificarSoldado
- compararSoldado
- intercambiarSoldado
- verSoldado
- verEjercito
- sumarNiveles
- jugar
- volver
- main
- fillTable
- addSoldado
- printTable
- printArr
- random
- printMayorNivel
- printPromedioPuntos
- printPuntosAll

- printSoladosOrdenados
- printRankingPointsBubble
- printRankingPointsSelect
- getWinner
- randomWinner
- mover
- atacar

A continuación veremos la explicación de los métodos principales para el funcionamiento.

## 7. Juego Rapido

El método `juegoRapido()` representa la implementación de un juego estratégico entre dos ejércitos de soldados dispuestos en una matriz bidimensional. A continuación, se proporciona una explicación detallada del código:

### 7.1. Configuración Inicial

La función comienza solicitando al usuario ingresar 'S' si desea salir de la partida. Si se ingresa 'S', la función termina inmediatamente. En caso contrario, se procede a la configuración inicial del juego.

```
1 Scanner sc = new Scanner(System.in);
2 System.out.println("Para salir de la partida ingresa 'S'");
3 String salir = sc.next();
4 if (salir.toLowerCase().charAt(0) == 's') return;
```

### 7.2. Inicialización de Variables

Se crea una matriz `table` de dimensiones 10x10 para representar el campo de batalla, y dos listas (`ejercito1` y `ejercito2`) que contendrán instancias de la clase `Soldado`. La función `fillTable()` se llama para inicializar la matriz y las listas.

```
1 Soldado[][] table = new Soldado[10][10];
2 ArrayList<Soldado> ejercito1 = new ArrayList<Soldado>();
3 ArrayList<Soldado> ejercito2 = new ArrayList<Soldado>();
4 fillTable(table, ejercito1, ejercito2);
```

### 7.3. Impresión de Información Inicial

Se imprime en consola el soldado con mayor nivel en cada ejército y, posteriormente, se muestra el promedio de puntos de ambos ejércitos.

```
1 printMayorNivel(table, ejercito1, ejercito2);
2 System.out.println("-----");
3 System.out.println("IMPRIMIR PROMEDIO DE PUNTOS");
4 printPromedioPuntos(table, ejercito1, ejercito2);
5 System.out.println("-----");
```

## 7.4. Ordenamiento y Ranking

Se imprime la lista de soldados ordenada por algún criterio no especificado y se muestra el ranking de puntos de ambos ejércitos, utilizando algoritmos de ordenamiento Bubble Sort y Select Sort, respectivamente.

```
1 System.out.println("-----");
2 System.out.println("SOLDADOS ORDENADOS");
3 printSoldadosOrdenados(table, ejercito1, ejercito2);
4 System.out.println("-----");
5 System.out.println("RANKING DE PUNTOS EJERCITO1 POR BUBBLE SORT");
6 printRankingPointsBubble(table, ejercito1);
7 System.out.println("-----");
8 System.out.println("RANKING DE PUNTOS EJERCITO2 POR SELECT SORT");
9 printRankingPointsSelect(table, ejercito2);
```

## 7.5. Desarrollo del Juego

Se inicia un bucle que representa los turnos del juego. En cada turno, se muestra el estado actual del campo de batalla (`printTable(table)`) y se realiza un movimiento de los soldados. Se alterna entre los dos ejércitos en cada turno.

```
1 int turno = 1;
2 while (true) {
3     printTable(table);
4     mover(table, ejercito1, ejercito2, turno);
5
6     turno = turno == 1 ? 2 : 1;
7
8     String winner = getWinner(ejercito1, ejercito2);
9
10    if (winner != null) {
11        System.out.println("#####");
12        System.out.println("GANA EL EQUIPO " + winner);
13        System.out.println("#####");
14        break;
15    }
16 }
```

## 7.6. Conclusión del Juego

Cuando se determina un ganador (mediante la función `getWinner()`), se imprime el equipo ganador y se finaliza el juego.

Entonces, el método `juegoRapido()` encapsula la lógica de un juego estratégico entre dos ejércitos de soldados, proporcionando información inicial, estadísticas y una interfaz para que los usuarios participen en turnos alternativos hasta que se determine un ganador.

## 8. Juego Personalizado

El método `juegoPersonalizado` proporciona una interfaz interactiva para que el usuario personalice y gestione los soldados de dos equipos en un juego de estrategia. Cada opción del menú está asociada a una función específica que realiza acciones como crear, eliminar, clonar o modificar soldados, así como visualizar información sobre los mismos y el estado general de los ejércitos. el método también incluye opciones para avanzar al juego o volver al menú principal. A continuación se detalla su estructura interna:



## 8.1. Inicialización de Variables

Se utiliza un objeto `Scanner` para la entrada de datos desde la consola. Se crea una matriz llamada `table` de dimensiones  $10 \times 10$  para representar el campo de juego. Se declaran dos `ArrayList` llamados `ejercito1` y `ejercito2` para almacenar los soldados de cada equipo.

```
1 Scanner sc = new Scanner(System.in);
2 Soldado[][] table = new Soldado[10][10];
3 ArrayList<Soldado> ejercito1 = new ArrayList<Soldado>();
4 ArrayList<Soldado> ejercito2 = new ArrayList<Soldado>();
5 fillTable(table, ejercito1, ejercito2);
```

## 8.2. Rellenar la Matriz y Listas de Soldados

Se invoca el método `fillTable` para inicializar la matriz y las listas de soldados de ambos equipos.

```
1 fillTable(table, ejercito1, ejercito2);
```

## 8.3. Impresión de Soldados Ordenados

Se imprime en la consola la información de los soldados ordenados en el campo de juego.

```
1 System.out.println("SOLDADOS ORDENADOS");
2 printSoldadosOrdenados(table, ejercito1, ejercito2);
3 System.out.println("#####");
```

## 8.4. Selección del Equipo a Personalizar

Se solicita al usuario que ingrese el símbolo del equipo ("\*" o "#") que desea personalizar. Se determina el `ArrayList` del equipo seleccionado mediante un operador ternario.

```
1 System.out.print("Escoge al equipo a personalizar: ");
2 String team = sc.next();
3 ArrayList<Soldado> currentE = team.equals("*") ? ejercito1 : ejercito2;
```

## 8.5. Menú de Opciones

Se presenta un menú con varias opciones numeradas que permiten al usuario realizar acciones específicas en relación con los soldados y equipos.

```
1 System.out.println("Escoge una opcion: ");
2 // ... (Listado de opciones)
3 int input = sc.nextInt();
```

## 8.6. Ejecución de Acciones Según la Opción Seleccionada

Se utiliza una serie de condicionales para determinar la acción a realizar según la opción ingresada por el usuario.

```
1 if (input == 1) crearSoldado(table, currentE, team);
2 if (input == 2) eliminarSoldado(table, currentE, team);
3 // ... (Resto de opciones)
4 if (input == 10) jugar();
```

```
5 if (input == 11) volver();
```

## 9. Métodos de gestión de Soldados

Estas funciones proporcionan al usuario la capacidad de crear, eliminar, clonar y modificar soldados en su ejército, contribuyendo así a la personalización y estrategia dentro del juego. Cada función interactúa directamente con la matriz que representa el campo de juego y la lista de soldados del equipo correspondiente.

### 9.1. Función crearSoldado

**Propósito:** Permite al usuario crear un nuevo soldado para su ejército.

**Parámetros:**

- `Soldado[] [] t`: La matriz que representa el campo de juego.
- `ArrayList<Soldado>e`: Lista de soldados del equipo actual.
- `String team`: Símbolo del equipo actual.

**Descripción:**

```
1 public static void crearSoldado(Soldado[] [] t, ArrayList<Soldado> e, String team) {
2     Scanner sc = new Scanner(System.in);
3     Soldado s = new Soldado(team);
4
5     if(e.size() >= 10) {
6         System.out.println("El ejercito tiene el maximo de soldados: ");
7         return;
8     }
9
10    System.out.print("Ingresa el nivel del soldado: ");
11    s.setNivelVida(sc.nextInt());
12    System.out.print("Ingresa el nivel de ataque: ");
13    s.setNivelAtaque(sc.nextInt());
14    System.out.print("Ingresa el nivel de defensa: ");
15    s.setNivelDefensa(sc.nextInt());
16    System.out.print("Ingresa la columna: ");
17    s.setColumna(sc.nextInt());
18    System.out.print("Ingresa la fila: ");
19    s.setFila(sc.nextInt());
20    s.setNombre("Soldado " + (e.size() + 1));
21
22    e.add(s);
23    t[s.getColumna()][s.getFila()] = s;
24
25 }
```

### 9.2. Función eliminarSoldado

**Propósito:** Permite al usuario eliminar un soldado de su ejército.

**Parámetros:**

- `Soldado[] [] t`: La matriz que representa el campo de juego.
- `ArrayList<Soldado>e`: Lista de soldados del equipo actual.
- `String team`: Símbolo del equipo actual.

**Descripción:**

```
1 public static void eliminarSoldado(Soldado[][] t, ArrayList<Soldado> e, String team) {
2     Scanner sc = new Scanner(System.in);
3
4     if(e.size() <= 1) {
5         System.out.println("El ejercito nesecita al menos 1 soldado");
6         return;
7     }
8
9     System.out.print("Ingresa la columna: ");
10    int col = sc.nextInt();
11    System.out.print("Ingresa la fila: ");
12    int fila = sc.nextInt();
13
14    for(int i = 0; i < e.size(); i += 1) {
15        if(e.get(i).getColumna() == col && e.get(i).getFila() == fila){
16            e.remove(i);
17            break;
18        }
19    }
20
21    t[col][fila] = null;
22 }
23 }
```

**9.3. Función clonarSoldado**

**Propósito:** Permite al usuario clonar un soldado existente en una nueva posición.

**Parámetros:**

- Soldado[][] t: La matriz que representa el campo de juego.
- ArrayList<Soldado>e: Lista de soldados del equipo actual.
- String team: Símbolo del equipo actual.

**Descripción:**

```
1 public static void clonarSoldado(Soldado[][] t, ArrayList<Soldado> e, String team) {
2     Scanner sc = new Scanner(System.in);
3
4     System.out.print("Ingresa la columna: ");
5     int col = sc.nextInt();
6     System.out.print("Ingresa la fila: ");
7     int fila = sc.nextInt();
8
9     Soldado soldadoOriginal = t[col][fila];
10
11     Soldado soldadoNuevo = new Soldado(soldadoOriginal.getTeam());
12     soldadoNuevo.setNivelAtaque(soldadoOriginal.getNivelAtaque());
13     soldadoNuevo.setNivelDefensa(soldadoOriginal.getNivelDefensa());
14     soldadoNuevo.setNivelVida(soldadoOriginal.getNivelVida());
15     soldadoNuevo.setNombre("Soldado " + (e.size() + 1));
16
17     System.out.print("Ingresa la columna nueva: ");
18     int newCol = sc.nextInt();
19     System.out.print("Ingresa la fila nueva: ");
20     int newFila = sc.nextInt();
21
22     e.add(soldadoNuevo);
23     t[newCol][newFila] = soldadoNuevo;
24 }
```

## 9.4. Función modificarSoldado

**Propósito:** Permite al usuario modificar las características y posición de un soldado existente.

**Parámetros:**

- Soldado[] [] t: La matriz que representa el campo de juego.
- ArrayList<Soldado>e: Lista de soldados del equipo actual.
- String team: Símbolo del equipo actual.

**Descripción:**

```
1 public static void modificarSoldado(Soldado[] [] t, ArrayList<Soldado> e, String team) {
2     Scanner sc = new Scanner(System.in);
3
4     System.out.print("Ingresa la columna: ");
5     int col = sc.nextInt();
6     System.out.print("Ingresa la fila: ");
7     int fila = sc.nextInt();
8
9     Soldado s = t[col][fila];
10    System.out.println("Ingresa nivel de vida nuevo: ");
11    s.setNivelVida(sc.nextInt());
12    System.out.println("Ingresa nivel de ataque nuevo: ");
13    s.setNivelAtaque(sc.nextInt());
14    System.out.println("Ingresa nivel de defensa nuevo: ");
15    s.setNivelDefensa(sc.nextInt());
16    System.out.println("Ingresa la columna nueva: ");
17    s.setColumna(sc.nextInt());
18    System.out.println("Ingresa la fila nueva: ");
19    s.setFila(sc.nextInt());
20 }
```

## 10. Ejecucion grafica y por consola

- A continuacion se vera la Ejecucion tanto grafica como por consola

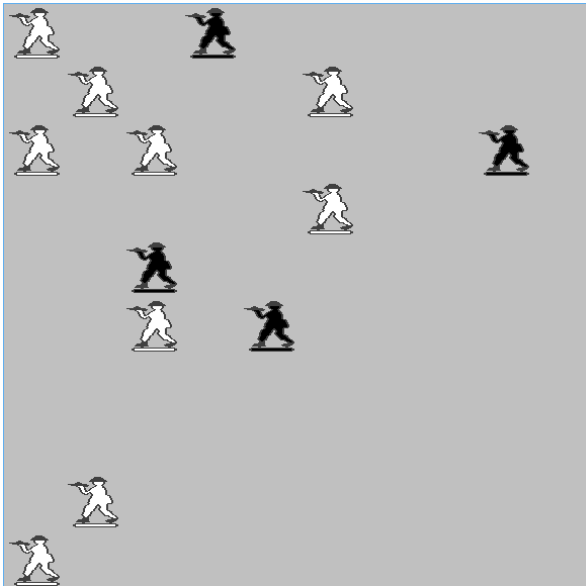
### 10.1. Ejecucion por consola

```
1 ===== Ejercito 1 =====
2 Soldado 0x8:
3   Nivel de vida: 3
4   Fila: 9
5   Columna: 2
6
7 Soldado 0x6:
8   Nivel de vida: 3
9   Fila: 3
10  Columna: 1
11
12 Soldado 0x7:
13  Nivel de vida: 5
14  Fila: 1
15  Columna: 1
16
17 Soldado 0x4:
18  Nivel de vida: 4
19  Fila: 2
20  Columna: 2
```

```
21
22 Soldado 0x5:
23   Nivel de vida: 1
24   Fila: 6
25   Columna: 3
26
27 Soldado 0x2:
28   Nivel de vida: 4
29   Fila: 4
30   Columna: 6
31
32 Soldado 0x3:
33   Nivel de vida: 3
34   Fila: 10
35   Columna: 1
36
37 Soldado 0x0:
38   Nivel de vida: 4
39   Fila: 2
40   Columna: 6
41
42 Soldado 0x1:
43   Nivel de vida: 2
44   Fila: 3
45   Columna: 3
46
47
48 ===== Ejercito 2 =====
49 Soldado 1x3:
50   Nivel de vida: 1
51   Fila: 1
52   Columna: 4
53
54 Soldado 1x1:
55   Nivel de vida: 3
56   Fila: 5
57   Columna: 3
58
59 Soldado 1x2:
60   Nivel de vida: 4
61   Fila: 3
62   Columna: 9
63
64 Soldado 1x0:
65   Nivel de vida: 1
66   Fila: 6
67   Columna: 5
68
69 Soldado con maxima vida:
70 Soldado 0x7:
71   Nivel de vida: 5
72   Fila: 1
73   Columna: 1
74
75
76 ===== Ranking de soldados: =====
77 Soldado 0x7:
78   Nivel de vida: 5
79   Fila: 1
80   Columna: 1
81
82 Soldado 0x2:
83   Nivel de vida: 4
84   Fila: 4
85   Columna: 6
```

```
86
87 Soldado 1x2:
88 Nivel de vida: 4
89 Fila: 3
90 Columna: 9
91
92 Soldado 0x0:
93 Nivel de vida: 4
94 Fila: 2
95 Columna: 6
96
97 Soldado 0x4:
98 Nivel de vida: 4
99 Fila: 2
100 Columna: 2
101
102 Soldado 1x1:
103 Nivel de vida: 3
104 Fila: 5
105 Columna: 3
106
107 Soldado 0x8:
108 Nivel de vida: 3
109 Fila: 9
110 Columna: 2
111
112 Soldado 0x6:
113 Nivel de vida: 3
114 Fila: 3
115 Columna: 1
116
117 Soldado 0x3:
118 Nivel de vida: 3
119 Fila: 10
120 Columna: 1
121
122 Soldado 0x1:
123 Nivel de vida: 2
124 Fila: 3
125 Columna: 3
126
127 Soldado 1x3:
128 Nivel de vida: 1
129 Fila: 1
130 Columna: 4
131
132 Soldado 0x5:
133 Nivel de vida: 1
134 Fila: 6
135 Columna: 3
136
137 Soldado 1x0:
138 Nivel de vida: 1
139 Fila: 6
140 Columna: 5
```

- Como vemos, la estructura del metodo main esta representada en la salida por consola
- A continuacion se vera la salida grafica:



## 11. Commits mas importantes

- A continuacion se muestran los commits mas importantes
- Los commits se hicieron siguiendo la convencion para commits de git, y siguiendo las recomendaciones practicas para hacer mensajes de commits
- Cada mensaje de commit esta estructurado por un titulo y una descripcion separados por una linea en blanco

Listing 1: commits mas importantes

```

1  commit c1082e4d2b725108cdbf1f4acb639a89da4174d8
2  Author: JhonatanDczel <jariasq@unsa.edu.pe>
3  Date:   Mon Oct 23 14:02:50 2023 -0500
4
5      Creando el proyecto lab08
6
7      Se copiaron los archivos soldado, videojuego y graphics del laboratorio
8      anterior
9
10
11  commit a362ecf01678492c2ba977fd8c3d4f866ccc0313
12  Author: JhonatanDczel <jariasq@unsa.edu.pe>
13  Date:   Mon Oct 23 14:12:17 2023 -0500
14
15      Hace que el metodo inicialice army devuelva hashmap
16
17  commit 40035dcc9534d58e9b577105339203b1dd0aa839
18  Author: JhonatanDczel <jariasq@unsa.edu.pe>
19  Date:   Mon Oct 23 14:08:29 2023 -0500
20
21      Creacion del metodo inicialiceArmyHashMap
22
23  commit c78508b8e703041b1d11d8410744bed4c9870f31
24  Author: JhonatanDczel <jariasq@unsa.edu.pe>
25  Date:   Mon Oct 23 14:25:05 2023 -0500
26

```

```
27     Se cambio la forma de establecer el atributo negro
28
29     Se cambio la forma en que se establece que un ejercito sea de color
30     negro, antes del cambio, se usaba un condicional para evaluar el valor
31     booleano negro, ahora, se ingresa directamente este valor como atributo
32     del soldado
33
34     commit 6e7b92190399f9550e32bb58243d0b07ba26d918
35     Author: JhonatanDczel <jariasq@unsa.edu.pe>
36     Date:   Mon Oct 23 15:06:45 2023 -0500
37
38     Se crea metodo para ordenar el ejercito
39
40     Como no se puede ordenar directamente un hashmap, ya que no mantiene un
41     oprden especifico, se copia el contenido a un array antes de mandarlo al
42     metodo de ordenamiento de soldados por vida
43
44     commit edfb9c396ca95be5c3038fb18c4bb282894758da
45     Author: JhonatanDczel <jariasq@unsa.edu.pe>
46     Date:   Mon Oct 23 15:48:12 2023 -0500
47
48     Adaptando el codigo para hacer el ordenamiento
49
50     Se adapto el codigo para crear un array grande que una a los dos
51     ejercitos, luego se creo el metodo ranking que agarra estos dos
52     ejercitos, los junta y crea uno mas grande que ordena con el algoritmo
53     bubble sort, finalmente el resultado se muestra en pantalla
```



## 12. Rúbricas

### 12.1. 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.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: 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 2: 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	1.5	
<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	1.5	
<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	4	
<b>Total</b>		20		19	