

Informe de Laboratorio 09

Tema: Laboratorio 09

Nota

Estudiante	Escuela	Asignatura
Victor Mamani Anahua vmamanian@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 20230489

Laboratorio	Tema	Duración
09	Laboratorio 09	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 25 Octubre 2023	Al 1 Noviembre 2023

1. Tarea

- Cree un Proyecto llamado Laboratorio9
- Crear 3 constructores sobrecargados.
- La actitud puede ser defensiva, ofensiva, fuga. Dicha actitud varía cuando el soldado defiende, ataca o huye respectivamente.
- Al atacar el soldado avanza, al avanzar aumenta su velocidad en 1. Al defender el soldado se para. Al huir aumenta su velocidad en 2. Al retroceder, si su velocidad es mayor que 0, entonces primero para y su actitud es defensiva, y si su velocidad es 0 entonces disminuirá a valores negativos. Al ser atacado su vida actual disminuye y puede llegar incluso a morir.
- Crear los atributos y métodos extra que considere necesarios.
- 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 el tablero utilizar la estructura de datos más adecuada.
- 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).

- 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).
- Crear el diagrama de clases UML.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Ubuntu GNU Linux 23 lunar 64 bits Kernel 6.2.v
- Visual Studio Code.
- VIM 9.0.
- OpenJDK 64-Bits 19.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Actividades del Laboratorio 08.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/VictorMA18/fp2-23b.git>
- URL para el laboratorio 09 en el Repositorio GitHub.
- <https://github.com/VictorMA18/fp2-23b/tree/main/Fase02/Lab09>

4. Actividades del Laboratorio 09

4.1. Ejercicio Soldado

- En el primer commit reutilizamos la clase Soldado de los anteriores laboratorios el cual a este le quitamos el atributo health y lo cambiamos con lifeActual y también añadimos diferentes atributos y diferentes métodos añadimos el 2do constructor el cual este me da un objeto nulo el cual lo podremos utilizar para poder reemplazar las casillas en blanco y también en el otro constructor le ponemos ya sus atributos correspondientes el cual este soldado va a tener.
- El código y el commit sería el siguiente:

Listing 1: Commit

```
$ git commit -m "Completando los atributos del constructor de la clase Soldado() el  
cual este va a ser para cuando el objeto sea nulo"
```

Listing 2: Las líneas de códigos del método creado:

```
// Laboratorio Nro 9 - Ejercicio Soldado
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
import java.util.*;
public class Soldado { //CREAMOS LA CLASE SOLDADO PARA PODER USAR UN ARREGLO
    BIDIMENSIONAL DONDE NECESITAMOS LA VIDA , EL NOMBRE DEL SOLDADO Y TAMBIEN SU
    POSICION COMO LA FILA Y LA COLUMNA

    private String name;
    private int health;
    private int row;
    private String column;
    private int attacklevel;
    private int defenselevel;
    private int lifelevel;
    private int lifeactual;
    private int speed;
    private String attitude;
    private boolean lives;

    Random rdm = new Random();

    //Anadiendo metodo que nos permita que un arreglo tenga datos nulos si este esta
    vacio
    public Soldado(){
        this.name = "";
        this.health = 0;
        this.row = 0;
        this.column = "";
        this.attacklevel = 0;
        this.defenselevel = 0;
        this.lifelevel = 0;
        this.lifeactual = 0;
        this.speed = 0;
        this.attitude = "";
        this.lives = false;
    }

    //Constructor
    public Soldado(String name, int health, int row, String column){
        this.name = name;
        this.health = health;
        this.lifeactual = health;
        this.row = row;
        this.column = column;
        this.lives = true;

        //YA QUE ESTOS DATOS SERIAN ALEATORIOS YA QUE SE ESTARIA CREANDO EL SOLDADO
        TENDRIAMOS DATOS QUE SERIAN COMO ATTACKLEVEL DEFENSELEVEL EL CUAL TENDRIAN
        QUE SER ALEATORIOS
        this.attacklevel = rdm.nextInt(5) + 1;
        this.defenselevel = rdm.nextInt(5) + 1;
    }
}
```

```
//Constructor para los diferentes niveles como de vida defensa ataque velocidad

// Metodos mutadores
public void setName(String n){
    name = n;
}
public void setHealth(int p){
    health = p;
}
public void setRow(int b){
    row = b;
}
public void setColumn(String c){
    column = c;
}

// Metodos accesoros
public String getName(){
    return name;
}
public int getHealth(){
    return health;
}

public int getRow(){
    return row;
}
public String getColumn(){
    return column;
}

// Completar con otros metodos necesarios
public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEL OBJETO
    String join = "\nNombre: " + getName() + "\nVida: " + getHealth() + "\nFila: " +
        getRow() + "\nColumna: " + getColumn(); //Agregamos un espaciador para poder
        separar
    return join;
}
}
```

4.2. Ejercicio Soldado

- En el segundo commit creamos el 3er constructor el cual tendremos los niveles la velocidad y su estado tambien añadimos sus getters y setters , tambien modificamos el metodo toString() el cual nos dara toda la informacion del soldado
- El codigo y el commit seria el siguiente:

Listing 3: Commit

```
$ git commit -m "Creamos el 3er constructor el cual tendremos los niveles la velocidad
y su estado tambien anadimos sus getters y setters , tambien modificamos el metodo
toString() el cual nos dara toda la informacion del soldado"
```

Listing 4: Las líneas de códigos del método creado:

```
public Soldado(String name , int attacklevel, int defenselevel, int lifelevel, int
    speed, boolean lives, int row, String column) {
    this.name = name;
    this.attacklevel = attacklevel;
    this.defenselevel = defenselevel;
    this.lifelevel = lifelevel;
    this.speed = speed;
    this.lives = lives;
    this.row = row;
    this.column = column;
}
public void setAttackLevel(int attacklevel) {
    this.attacklevel = attacklevel;
}
public void setDefenseLevel(int defenselevel) {
    this.defenselevel = defenselevel;
}
public void setLifeLevel(int lifelevel){
    this.lifelevel = lifelevel;
}
public void setSpeed(int speed) {
    this.speed = speed;
}
public void setAttitude(String attitude) {
    this.attitude = attitude;
}
public void setLives(boolean lives) {
    this.lives = lives;
}

public int getAttackLevel() {
    return attacklevel;
}
public int getDefenseLevel() {
    return defenselevel;
}
public int getLifeLevel(){
    return lifelevel;
}
public int getSpeed() {
    return speed;
}
public String getAttitude() {
    return attitude;
}
public boolean getLives() {
    return lives;
}
public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEL OBJETO
    String join = "\nNombre: " + getName() + "\nVida: " + getLifeActual() + "\nFila: " +
        getRow() + "\nColumna: " + getColumn() + "\nNivel de ataque: " +
        getAttackLevel() + "\nNivel de Defensa: " + getDefenseLevel() + "\nNivel de
        vida: " + getLifeLevel() + "\nVelocidad: " + getSpeed() + "\nActitud: " +
        getAttitude() + "\nEstado: " + getLives(); //Agregamos un espaciador para poder
        separar
```

```
    return join;
}
```

4.3. Ejercicio Soldado

- En el tercer commit añadimos los metodos restantes a los cuales se les podra identificar debido al mensaje que estos daran y saber la accion que estan ejecutando por ejemplo como advance() defense() flee() back() attaack() metodos pedidos que se hagan para los soldados
- El codigo y el commit seria el siguiente:

Listing 5: Commit

```
$ git commit -m "Anadiendo los metodos restantes a los cuales se les podra identificar
debido al mensaje que estos daran y saber la accion que estan ejecutando por
ejemplo como advance() defense() flee() back() attaack() metodos pedidos que se
hagan para los soldados"
```

Listing 6: Las lineas de codigos del metodo creado:

```
public void defense(){
    this.speed = 0;
    this.attitude = "DEFENSIVA";
    System.out.println("El soldado " + this.name + "esta defendiendo");
}
public void flee(){
    this.speed = getSpeed() + 2;
    this.attitude = "HUYE";
    System.out.println("El soldado " + this.name + "esta huyendo");
}
public void back(){
    System.out.println("El soldado " + this.name + "esta retrocediendo");
    if(this.speed == 0){
        this.speed = rdm.nextInt(5) - 5;
    }else{
        if(this.speed > 0){
            this.speed = 0;
            this.attitude = "DEFENSIVA";
        }
    }
}
public void attaack(Soldado soldier){
    if(this.getLifeActual() > soldier.getLifeActual()){
        int life = this.getLifeActual() - soldier.getLifeActual();
        this.setLifeActual(life);
        this.setLifeLevel(life);
        soldier.lives = false;
        System.out.println(this.name + " asesino al soldado " + soldier.name);
    }else if(soldier.getLifeActual() > this.getLifeActual()){
        int life = soldier.getLifeActual() - this.getLifeActual();
        this.lives = false;
        soldier.setLifeActual(life);
        soldier.setLifeLevel(life);
        System.out.println(soldier.name + " asesino al soldado " + this.name);
    }
}
```

```
}else{
    this.lives = false;
    soldier.lives = false;
    System.out.println("los 2 soldados se asesinaron");
}
}
```

4.4. Ejercicio Videojuego

- En el cuarto commit creamos el metodo fillRegister() el cual nos ayudara a poder registrar a soldados los cuales van a ser creados el cual tambien en este comprobamos que 2 soldados del mismo ejercito no se repitan para esto creamos una condicional ala vez hicimos pruebas para poder saber si lo puesto como condiciones exista cuando lo imprimamos esto de final de este metodo nos retornara a los soldados creados de cada ejercito por orden de creacion el cual va a ser sus datos
- El codigo , el commit y la ejecución seria el siguiente:

Listing 7: Commit

```
$ git commit -m "Creando el metodo fillRegister() el cual nos ayudara a poder registrar
a soldados los cuales van a ser creados el cual tambien en este comprobamos que 2
soldados del mismo ejercito no se repitan para esto creamos una condicional ala vez
hicimos pruebas para poder saber si lo puesto como condiciones exista cuando lo
imprimamos esto de final de este metodo nos retornara a los soldados creados de
cada ejercito por orden de creacion el cual va a ser sus datos"
```

Listing 8: Las lineas de codigos del metodo creado:

```
import java.util.*;
class Videojuego {
    public static ArrayList<ArrayList<Soldado>> fillRegister(int num){
        Random rdm = new Random();
        ArrayList<ArrayList<Soldado>> army = new ArrayList<ArrayList<Soldado>>();
        int numbersoldiers = rdm.nextInt(10) + 1; //NUMERO DE SOLDADOS ALEATORIOS ENTRE 1
        A 10 SOLDADOS
        for(int i = 0; i < 10; i++){ //ITERACION
            army.add(new ArrayList<Soldado>()); //LLENAMOS NUESTROS ARRAYLIST BIDIMENSIONAL
            CON CADA FILA PARA QUE CUMPLAN CON ESTRUCTURA DEL TABLERO
            for(int j = 0; j < 10 ; j++){//ITERACION
                army.get(i).add(null); // LLENAMOS CADA FILA DEL ARRAYLIST CON UN OBJETO
                SOLDADO CON TAL QUE ESTE SEA NULL PARA QUE SEPA QUE ESTE TIENE UNA
                CASILLA PERO NO HAY NADIE TODAVIA SE PUEDE LLENAR
            }
        }
        System.out.println("El Ejercito " + num + " tiene " + numbersoldiers + " soldados
        : " );
        System.out.println("");
        for(int i = 0; i < numbersoldiers; i++){ //LLENAMOS CASILLAS CON CADA SOLDADO
            CREADO ALEATORIAMENTE
            String name = "Soldado" + i + "X" + num;
            //System.out.println(name); PRUEBA QUE SE HIZO PARA VER LOS NOMBRES
            int health = rdm.nextInt(5) + 1;
            int row = rdm.nextInt(10) + 1;
```

```
int speed = rdm.nextInt(5) + 1;
String column = String.valueOf((char)(rdm.nextInt(10) + 65)); //REUTILIZAMOS
CODIGO DEL ANTERIOR ARCHIVO VIDEOJUEGO2.JAVA YA QUE TENDRIAN LA MISMA
FUNCIONALIDAD
//System.out.println(array.get(row - 1).get((int)column.charAt(0) - 65)); PRUEBA
QUE SE HIZO PARA COMPROBAR SI EL OBJETO SE ESTABA DANDO O NO CAPAZ NI
EXISTIA
if(array.get(row - 1).get((int)column.charAt(0) - 65) == null){
    System.out.print("Registrando al " + (i + 1) + " soldado del Ejercito " +
        num + " ");
    array.get(row - 1).set((int)column.charAt(0) - 65, new Soldado(name, health,
        row, column));
    array.get(row - 1).get((int)column.charAt(0) - 65).setSpeed(speed);
    System.out.println(array.get(row - 1).get((int)column.charAt(0) -
        65).toString());
    System.out.println("-----");
}else{
    i -= 1; //NOS AYUDARIA CON LOS SOLDADOS QUE SE REPITEN EN EL MISMO CASILLERO
    CON TAL QUE NO DEBERIA CONTAR
}
}
System.out.println("*****");
return array;
}
public static void main(String args[]){
    ArrayList<ArrayList<Soldado>> army1 = fillRegister(1);
    ArrayList<ArrayList<Soldado>> army2 = fillRegister(2);
}
}
```

Listing 9: Ejecucion:

```
El Ejercito 1 tiene 9 soldados :

Registrando al 1 soldado del Ejercito 1
Nombre: Soldado0X1
Vida: 1
Fila: 5
Columna: C
Nivel de ataque: 2
Nivel de Defensa: 1
Nivel de vida: 1
Velocidad: 1
Actitud: null
Estado: true
-----
Registrando al 2 soldado del Ejercito 1
Nombre: Soldado1X1
Vida: 2
Fila: 6
Columna: B
Nivel de ataque: 4
Nivel de Defensa: 4
Nivel de vida: 2
Velocidad: 2
Actitud: null
```



```
Estado: true
-----
Registrando al 3 soldado del Ejercito 1
Nombre: Soldado2X1
Vida: 5
Fila: 3
Columna: J
Nivel de ataque: 1
Nivel de Defensa: 1
Nivel de vida: 5
Velocidad: 5
Actitud: null
Estado: true
-----
Registrando al 4 soldado del Ejercito 1
Nombre: Soldado3X1
Vida: 2
Fila: 3
Columna: B
Nivel de ataque: 3
Nivel de Defensa: 4
Nivel de vida: 2
Velocidad: 2
Actitud: null
Estado: true
-----
Registrando al 5 soldado del Ejercito 1
Nombre: Soldado4X1
Vida: 3
Fila: 9
Columna: G
Nivel de ataque: 2
Nivel de Defensa: 4
Nivel de vida: 3
Velocidad: 3
Actitud: null
Estado: true
-----
Registrando al 6 soldado del Ejercito 1
Nombre: Soldado5X1
Vida: 1
Fila: 9
Columna: J
Nivel de ataque: 5
Nivel de Defensa: 4
Nivel de vida: 1
Velocidad: 3
Actitud: null
Estado: true
-----
Registrando al 7 soldado del Ejercito 1
Nombre: Soldado6X1
Vida: 5
Fila: 4
Columna: F
Nivel de ataque: 3
```

```
Nivel de Defensa: 4
Nivel de vida: 5
Velocidad: 4
Actitud: null
Estado: true
-----
Registrando al 8 soldado del Ejercito 1
Nombre: Soldado7X1
Vida: 4
Fila: 4
Columna: H
Nivel de ataque: 3
Nivel de Defensa: 3
Nivel de vida: 4
Velocidad: 1
Actitud: null
Estado: true
-----
Registrando al 9 soldado del Ejercito 1
Nombre: Soldado8X1
Vida: 5
Fila: 3
Columna: H
Nivel de ataque: 3
Nivel de Defensa: 4
Nivel de vida: 5
Velocidad: 5
Actitud: null
Estado: true
-----
*****
El Ejercito 2 tiene 3 soldados :

Registrando al 1 soldado del Ejercito 2
Nombre: Soldado0X2
Vida: 3
Fila: 6
Columna: G
Nivel de ataque: 2
Nivel de Defensa: 2
Nivel de vida: 3
Velocidad: 2
Actitud: null
Estado: true
-----
Registrando al 2 soldado del Ejercito 2
Nombre: Soldado1X2
Vida: 5
Fila: 3
Columna: E
Nivel de ataque: 5
Nivel de Defensa: 3
Nivel de vida: 5
Velocidad: 4
Actitud: null
Estado: true
```

```
-----  
Registrando al 3 soldado del Ejercito 2  
Nombre: Soldado2X2  
Vida: 5  
Fila: 1  
Columna: B  
Nivel de ataque: 5  
Nivel de Defensa: 2  
Nivel de vida: 5  
Velocidad: 4  
Actitud: null  
Estado: true  
-----  
*****
```

4.5. Estructura de laboratorio 09

- El contenido que se entrega en este laboratorio09 es el siguiente:

```
/Lab09  
"Poner RAMA"
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	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	
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	2	
Total		20		18	

6. Referencias

- https://drive.google.com/file/d/1TbYqdt7cGTuw_P_ZnkiAXBmPI8YhDMb/view