Informe de Laboratorio 13

Tema: Definición de Clases de Usuario Clase Soldado – Miembros de Clase

Nota

Estudiante	Escuela	Asignatura
Roni Companocca Checco rcompanocca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Semestre: II Código: 20210558

Laboratorio	Tema	Duración
13	Definición de Clases de Usuario Clase Soldado – Miembros de Clase	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 5 octubre 2023	Al 10 Noviembre 2023

1. TAREA

1.1. Objetivos:

- Que el alumno demuestre poder crear "clases definidas por el programador"
- Implementar métodos para las clases definidas por el programador
- Utilizar miembros de Clase

1.2. Competencias a alcanzar:

- Diseña, responsablemente, sistemas, componentes o procesos para satisfacer necesidades dentro de restricciones realistas: económicas, medio ambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad.
- Aplica de forma flexible, técnicas, métodos, principios, normas, estándares y herramientas de ingeniería necesarias para la construcción de software e implementación de sistemas de información.

1.3. Marco teorico:

- Clase: Una Fábrica de objetos. Una clase está conformada por dos partes: datos miembro (variables de instancia / atributos) y los métodos.
- Los métodos nos permiten acceder y/o modificar los datos miembros (atributos) de los objetos.
- Toda clase necesita al menos un Constructor. El constructor lleva el mismo nombre de la clase. El constructor es un método especial que siempre se llama con la palabra reservada new()

1.4. Indicaciones generales:

- Todos los ejercicios deberán ser guardados en el mismo Proyecto
- El Proyecto deberá tener el nombre del Laboratorio y el nombre del alumno, así por ejemplo: Laboratorio 1 – Juan Perez
- Cada Clase deberá tener el nombre del ejercicio, así por ejemplo: Ejercicio1
- Utilice nombres de variables significativos y todas las recomendaciones de estilo
- Especialmente, su código deberá estar correctamente indentado
- Deberá pasar TODOS los casos de prueba

2. EQUIPOS, MATERIALES Y TEMAS UTILIZADOS

- Sistema Operativo Windows
- OpenJDK 64-Bits 17.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.

3. URL DE REPOSITORIO GITHUB

- URL para el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO`
- URL para el laboratorio 13 en el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO/FP2-LAB-13`

4. EJERCICIO PROPUESTO

4.1. INTRODUCCION

4.1.1. Este laboratorio requiere que usted escriba un programa utilizando clases definidas por el programador. No deberá utilizar sintaxis o constructores que no han sido cubiertos durante las clases teóricas. Será penalizado por esta falta. A menos que una plantilla sea dada, deberá utilizar cada programa desde cero de manera que obtenga suficiente práctica en la escritura de programas en Java.

4.1.2. Un consejo: Programe incrementalmente. No trate de terminar todas las partes del programa y luego compilarlo. Escriba sus programas en partes y compílelo de forma frecuente. Trate de mantener un programa compilable aun cuando esté trabajando en él. Presentar un programa compilable que funcione parcialmente es mejor que presentar un programa no-compilable. EN SERIO, programe incrementalmente.

4.1.3. Los objetivos de este laboratorio son:

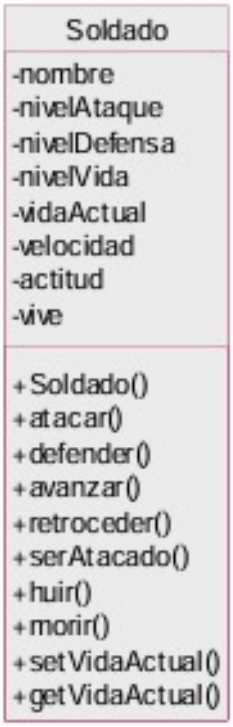
4.1.4. Deberá asumir que todos los datos de ingreso son correctos.

4.1.5. Deberá utilizar la clase Scanner en System.in para ingresos de datos y System.out para salida de datos en sus programas, a menos que se indique lo contrario.

4.1.6. Pruebe sus programas con sus propios datos de prueba antes de presentarlos.

4.1.7. Evitar duplicación de código.

4.1.8. Usar como base el diagrama de clases UML siguiente (puede aumentar atributos y métodos necesarios):



- .

4.1.9. Puede reutilizar todo el código del laboratorio 11, pero ahora el objetivo es gestionar los ejércitos autogenerados.

4.1.10. 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:

2.1 Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)

2.2 Eliminar Soldado (no debe permitir un ejército vacío)

2.3 Clonar Soldado (crea una copia exacta del soldado) y se añade al final del ejército (recordar que límite es de 10 soldados por ejército)

2.4 Modificar Soldado (con submenú para cambiar alguno de los atributos nivelAtaque, nivelDefensa, vidaActual)

2.5 Comparar Soldados (verifica si atributos: nombre, nivelAtaque, nivelDefensa, vidaActual y vive son iguales)

2.6 Intercambiar Soldados (intercambia 2 soldados en sus posiciones en la estructura de datos del ejército)

2.7 Ver soldado (Búsqueda por nombre)

2.8 Ver ejército

2.9 Sumar niveles (usando Method-Call Chaining), calcular las sumatorias de nivelVida, nivelAtaque, nivelDefensa, velocidad de todos los soldados de un ejército

2.10 Jugar (se empezará el juego con los cambios realizados) y con las mismas opciones de la opción 1.

2.11 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

- la clase Main.java

```
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 13
// FUNDAMENTOS DE PROGRAMACION
import java.util.*;

public class Main{

   public static void main(String[] args) {

      //DECLARACION DE VARIABLES Y ARREGLOS NECESARIOS
      // Crea el tablero y los ejrcitos
        ArrayList<ArrayList<Soldado>> tablero = new ArrayList<>();
        ArrayList<Soldado> ejercito1 = new ArrayList<>();
        ArrayList<Soldado> ejercito2 = new ArrayList<>();

        inicializarTablero(tablero);
        inicializarEjercitos(ejercito1, ejercito2);

        // Muestra el tablero inicial
        imprimirTablero(tablero);

        // Ciclo del juego
        boolean juegoEnCurso = true;
        int jugadorActual = 1;

        while (juegoEnCurso) {
            // Turno del jugador
            ArrayList<Soldado> ejercitoActual = (jugadorActual == 1) ? ejercito1 :
                ejercito2;
            ArrayList<Soldado> ejercitoOponente = (jugadorActual == 1) ? ejercito2 :
                ejercito1;

            System.out.println("\nJugador " + jugadorActual + ", es tu turno:");

            // Solicita la coordenada del soldado a mover y la direccin
            int fila, columna, nuevaFila, nuevaColumna;
            Soldado soldado;

            do {
                Scanner scanner = new Scanner(System.in);
                System.out.print("Ingrese la fila del soldado a mover (1-10): ");
                fila = scanner.nextInt() - 1;
                System.out.print("Ingrese la columna del soldado a mover (A-J): ");
                scanner.nextLine(); // Consumir el salto de lnea anterior
                String columnaStr = scanner.nextLine();
                nuevaFila = fila;
                nuevaColumna = columnaStr.charAt(0) - 'A';
                soldado = tablero.get(fila).get(nuevaColumna);

                if (fila < 0 || fila >= 10 || nuevaColumna < 0 || nuevaColumna >= 10) {
                    System.out.println("Coordenadas fuera del rango. Intente de nuevo.");
                } else if (soldado == null || soldado.getVive() == false) {
```

```
                System.out.println("No hay un soldado en esa posicin o est muerto.
                    Intente de nuevo.");
            } else if (!esMovimientoValido(tablero, fila, nuevaColumna, nuevaFila,
                nuevaColumna)) {
                System.out.println("Movimiento no vlido. Intente de nuevo.");
            }
        } while (fila < 0 || fila >= 10 || nuevaColumna < 0 || nuevaColumna >= 10 ||
            soldado == null || soldado.getVive() == false ||
            !esMovimientoValido(tablero, fila, nuevaColumna, nuevaFila,
            nuevaColumna));

        // Realizar el movimiento
        Soldado soldadoOponente = tablero.get(nuevaFila).get(nuevaColumna);
        if (soldadoOponente != null && soldadoOponente.getVive()) {
            // Batalla
            System.out.println("Batalla!");
            double probabilidadGanarJugador =
                calcularProbabilidad(soldado.getPuntos(),
                soldadoOponente.getPuntos());

            Random random = new Random();
            double resultado = random.nextDouble();

            if (resultado <= probabilidadGanarJugador) {
                System.out.println("Gana el Jugador " + jugadorActual + "!");
                aumentarVida(soldado);
                eliminarSoldado(tablero, ejercitoOponente, nuevaFila, nuevaColumna);
            } else {
                System.out.println("Gana el Jugador " + (jugadorActual == 1 ? 2 : 1)
                    + "!");
                aumentarVida(soldadoOponente);
                eliminarSoldado(tablero, ejercitoActual, fila, nuevaColumna);
            }
        } else {
            // Movimiento normal
            tablero.get(nuevaFila).set(nuevaColumna, soldado);
            tablero.get(fila).set(nuevaColumna, null);
            soldado.setColumna(nuevaColumna);
            soldado.setFila(nuevaFila);
        }

        // Mostrar el tablero actualizado
        imprimirTablero(tablero);

        // Verificar si hay un ganador
        if (ejercito1.isEmpty()) {
            System.out.println("Jugador 2 gana!");
            juegoEnCurso = false;
        } else if (ejercito2.isEmpty()) {
            System.out.println("Jugador 1 gana!");
            juegoEnCurso = false;
        }

        // Cambiar al siguiente jugador
        jugadorActual = (jugadorActual == 1) ? 2 : 1;
    }
```

```
}

// METODO PARA CREAR NUMEROS ALEATORIOS EN UN RANGO
public static int aleatorio(int min, int max) {
   return(int)(Math.random()*(max-min+1)+min);
}

// METODO PARA INICIAR UN ARRAYLIST
public static void inicializarArreglo (ArrayList<Soldado> soldadito, int num) {
   for (int i=0; i<num; i++) {
      soldadito.add(new Soldado());
   }
}

// METODO PARA GENERAR DATOS DEL OBJETO SOLDADO
public static Soldado generarDatos() {
   Soldado soldadito = new Soldado();
   soldadito.setPuntos(aleatorio(1,5));
   soldadito.setColumna(aleatorio(1,10));
   soldadito.setFila(aleatorio(1,10));
   return soldadito;
}

// METODOS PARA GENERAR LOS EJERCITOS DE MANERA ALEATORIA
public static void generarEjercitos(ArrayList<Soldado>B1, ArrayList<Soldado>B2) {
   ArrayList<Soldado>Soldados = new ArrayList();
   Soldados.add(generarDatos());
   for (int i=1; i<(B1.size()+B2.size()); i++) {
      Soldados.add(generarDatos());
      for (int j=0; j<i; j++) {
         if(Soldados.get(i).getFila()==Soldados.get(j).getFila()) {
            if(Soldados.get(i).getColumna()==Soldados.get(j).getColumna()){
               Soldados.remove(i);
               i--;
            }
         }
      }
   }
   for (int i=0; i<B1.size(); i++) {
      B1.add(i, Soldados.get(i));
      B1.get(i).setNombre("Soldado"+i+"x1");
      B1.get(i).setColumn(B1.get(i).getPuntos()+"[E1]");
      B1.remove(i+1);
   }
   for (int i=0; i<B2.size(); i++) {
      B2.add(i, Soldados.get(i+B1.size()));
      B2.get(i).setNombre("Soldado"+i+"x2");
      B2.remove(i+1);
      B2.get(i).setColumn(B2.get(i).getPuntos()+"[E2]");
   }
}

// METODO PARA AADIR LOS EJERCITOS AL TABLERO
public static void aadirTablero(ArrayList<Soldado>soldadito,
    ArrayList<ArrayList<Soldado>>table) {
   for (int i=0; i<soldadito.size(); i++) {
```

```
      table.get(soldadito.get(i).getColumna()-1).add(soldadito.get(i).getFila()-1,soldadito.get(i));
      table.get(soldadito.get(i).getColumna()-1).remove(soldadito.get(i).getFila());
   }
}

// METODO PARA IMPRIMIR EL TABLERO EN LA CUAL SE DESARROLLA EL JUEGO
public static void imprimirTablero(ArrayList<ArrayList<Soldado>> table) {
   System.out.println("\tA\tB\tC\tD\tF\tG\tH\tI\tJ");
   for(int i=0; i<table.size(); i++) {
      System.out.print(i+1);
      for(int j=0; j<table.get(i).size();j++) {
         System.out.print("\t"+table.get(i).get(j).getColumn());
      }
      System.out.println("\n");
   }
}

//METODO PARA IMPRIMIR LOS SOLDADOS DE MAYOR VIDA
public static void SoldadoConMayorVida (ArrayList<Soldado>soldadito) {
   Soldado mayor = new Soldado();
   mayor.setPuntos(0);
   for(int i=0; i<soldadito.size();i++) {
      if (mayor.getPuntos()<soldadito.get(i).getPuntos()) {
         mayor = soldadito.get(i);
      }
   }
   imprimir(mayor);
}

// METODO PARA IMPRIMIR EL NOMBRE, LA POSICION Y NIVEL DE VIDA DEL SOLDADO
public static void imprimir(Soldado soldadito) {
   System.out.println("Nombre: "+soldadito.getNombre()+"\nPosicion:
       "+soldadito.getColumna()+"X"+soldadito.getFila()+"\tVida:
       "+soldadito.getPuntos());
}

// METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA,
    USUANDO UN ALGORITMO DE ORDENAMIENTO DE BURBUJA
public static void ordenarPorVidaMetodoA(ArrayList<Soldado>soldadito) {
   Soldado aux = new Soldado();
   for(int i=0; i<soldadito.size()-1; i++) {
      for(int j=0; j<soldadito.size()-i-1; j++) {
         if(soldadito.get(j).getPuntos()<soldadito.get(j+1).getPuntos()) {
            aux = soldadito.get(j);
            soldadito.set(j,soldadito.get(j+1));
            soldadito.set(j+1,aux);
         }
      }
   }
}

 // METODO QUE NOS AYUDA A ORDENAR LOS SOLDADOS DE ACUERDO A SU NIVEL DE VIDA, EN
     ESTA OCACION DIFERENTE A LA ANTERIOR QUE ERA ALGORITMO DE BURBUJA
 public static void ordenarPorVidaMetodoB(ArrayList<Soldado> soldadito) {
    Collections.sort(soldadito, new Comparator<Soldado>() {
       public int compare(Soldado s1, Soldado s2) {
```

```
                // Orden descendente por puntos de vida
                return Integer.compare(s2.getPuntos(), s1.getPuntos());
            }
        });
    }
}
```

- la clase Soldado.java

```
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 13
// FUNDAMENTOS DE PROGRAMACION - LABORATORIO
public class Soldado {
   private String nombre;
    private int fila;
    private int columna;
    private String column;
    private int nivelAtaque;
    private int nivelDefensa;
    private int puntos;
    private int vidaActual;
    private int velocidad;
    private String actitud;
    private boolean vive;

   public Soldado() {
      nombre = "";
        fila = 0;
        columna = 0;
        column = "";
        nivelAtaque = 0;
        nivelDefensa = 0;
        puntos = 0;
        vidaActual = 0;
        velocidad = 0;
        actitud = "";
        vive = true;
   }

   // METODOS MUTADORES

   public void setNombre(String nombre) {
      this.nombre = nombre;
   }

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

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

   public void setColumn(String column) {
      this.column = column;
```

```
}

public void setNivelAtaque(int nivelAtaque) {
    this.nivelAtaque = nivelAtaque;
 }

 public void setNivelDefensa(int nivelDefensa) {
    this.nivelDefensa = nivelDefensa;
 }

 public void setPuntos(int puntos) {
    this.puntos = puntos;
 }

 public void setVidaActual(int vidaActual) {
    this.vidaActual = vidaActual;
 }

 public void setVelocidad(int velocidad) {
    this.velocidad = velocidad;
 }

 public void setActitud(String actitud) {
    this.actitud = actitud;
 }

 public void setVive(boolean vive) {
    this.vive = vive;
 }

// METODOS ACCESORES
public String getNombre() {
   return nombre;
}

public int getFila() {
   return fila;
}

public int getColumna() {
   return columna;
}

public String getColumn() {
   return column;
}

public int getNivelAtaque() {
    return nivelAtaque;
 }

 public int getNivelDefensa() {
    return nivelDefensa;
 }

 public int getPuntos() {
```

```
        return puntos;
    }

    public int getVidaActual() {
        return vidaActual;
    }

    public int getVelocidad() {
        return velocidad;
    }

    public String getActitud() {
        return actitud;
    }

    public boolean getVive() {
        return vive;
    }

   // Otros mtodos

    public void serAtacado() {
    }

    // Mtodo para atacar: avanza y aumenta la velocidad en 1
    public void atacar() {
        avanzar();
        this.velocidad++;
        this.actitud = "ofensiva";
    }

    // Mtodo para avanzar: aumenta la velocidad en 1
    public void avanzar() {
        this.velocidad++;
    }

    // Mtodo para defender: el soldado se detiene
    public void defender() {
        this.actitud = "defensiva";
        this.velocidad = 0;
    }

    // Mtodo para huir: aumenta la velocidad en 2
    public void huir() {
        this.actitud = "fuga";
        this.velocidad += 2;
    }

    // Mtodo para retroceder: disminuye la velocidad o se para
    public void retroceder() {
        if (this.velocidad > 0) {
            defender();
        } else {
            this.velocidad--; // Disminuye a valores negativos si la velocidad es 0
        }
    }
```

```
    // Mtodo para ser atacado: disminuye el nivel de vida
    public void serAtacado(int danio) {
        this.vidaActual -= danio;
        if (this.vidaActual <= 0) {
            morir();
        }
    }

    // Mtodo para morir: cambia el estado del soldado a "no vive"
    public void morir() {
        this.vive = false;
    }

    public void aumentarVida() {
        this.vidaActual++;
    }

}
```

- Ejecucion

```
          A      B      C      D      F      G      H      I      J
1                                          2[E1]
2                                                        5[E1]
3
4
5
6                                                                     2[E2]
7
8                            2[E1]  3[E2]
9      3[E2]                                                          2[E1]
10                                         2[E1]         3[E2]

Soldado de mayor vida del ejercito 1
Nombre: Soldado4x1
Posicion: 2X8 Vida: 5
soldado de mayor vida del ejercito 2
Nombre: Soldado0x2
Posicion: 10X8 Vida: 3

EJERCITO 1:
Vida total: 13
Promedio de vida: 2.6

EJERCITO 2:
Vida total: 11
```

```
Promedio de vida: 2.75

Lista ejercito 1:
Nombre: Soldado0x1
Posicion: 1X6 Vida: 2
Nombre: Soldado1x1
Posicion: 9X9 Vida: 2
Nombre: Soldado2x1
Posicion: 8X4 Vida: 2
Nombre: Soldado3x1
Posicion: 10X6 Vida: 2
Nombre: Soldado4x1
Posicion: 2X8 Vida: 5

Lista ejercito 2:
Nombre: Soldado0x2
Posicion: 10X8 Vida: 3
Nombre: Soldado1x2
Posicion: 8X5 Vida: 3
Nombre: Soldado2x2
Posicion: 6X9 Vida: 2
Nombre: Soldado3x2
Posicion: 9X1 Vida: 3

Ejercito 1 Ordenados por nivel de vida
Nombre: Soldado4x1
Posicion: 2X8 Vida: 5
Nombre: Soldado0x1
Posicion: 1X6 Vida: 2
Nombre: Soldado1x1
Posicion: 9X9 Vida: 2
Nombre: Soldado2x1
Posicion: 8X4 Vida: 2
Nombre: Soldado3x1
Posicion: 10X6 Vida: 2

Ejercito 2 Ordenados por nivel de vida
Nombre: Soldado0x2
Posicion: 10X8 Vida: 3
Nombre: Soldado1x2
Posicion: 8X5 Vida: 3
Nombre: Soldado3x2
Posicion: 9X1 Vida: 3
Nombre: Soldado2x2
Posicion: 6X9 Vida: 2

GANADOR ***EJERCITO 2***
```

5. REFERENCIAS

- M. Aedo, “Fundamentos de Programación 2 - Tópicos de Programación Orientada a Objetos”, Primera Edición, 2021, Editorial UNSA.
- `https://github.com/rescobedoq/programacion.git`

- J. Dean, Ïntroduction to programming with Java: A Problem Solving Approach", Third Edition, 2021, McGraw-Hill.
- C. T. Wu, .ªn Introduction to Object-Oriented Programming with Java", Fifth Edition, 2010, McGraw-Hill.
- P. Deitel, "Java How to Program", Eleventh Edition, 2017, Prentice Hall.