Informe de Laboratorio 14

Programación Orientada a Objetos (I) - Mecanismos de Agregación, Composición, He

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
14	Programación Orientada a Objetos (I) - Mecanismos de Agregación, Composición, Herencia	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 06 Noviembre 2023	Al 11 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
- Utilizando atributos que son otros objetos. Agregación y composición

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.

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 14 en el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO/FP2-LAB14`

4. EJERCICIO RESUELTO

EJERCICIO 1: Crear un POO donde se implementa la relación de Composición.

1.1 Creamos la Clase Dirección:

```java
public class Direccion{
    //ATRIBUTOS
    private String calle;
    private int numero;
    //CONSTRUCTORES
    public Direccion(){
    }
    public Direccion(String calle, int numero){
        this.calle = calle;
        this.numero = numero;
    }
    //METODOS GETTER Y SETTER
    public String getCalle(){
        return calle;
    }
    public void setCalle(String calle){
        this.calle = calle;
    }
    public int getNumero(){
        return numero;
    }
    public void setNumero(int numero){
        this.numero = numero;
    }
}
```

1.2 Creamos la Clase Persona:

```java
import java.util.ArrayList;
```

```
public class Persona{
    //ATRIBUTOS
    private String nombre;
    private int edad;
    private ArrayList <Direccion> direcciones;
    //CONSTRUCTORES
    public Persona(String nombre, int edad, String calle, int numero){
        this.nombre = nombre;
        this.edad = edad;
        direcciones = new ArrayList<Direccion>();
        Direccion direccion = new Direccion(calle,numero);
        //AGREGAMOS LA INSTANCIA DIRECCION
        direcciones.add(direccion);
    }
    //METODOS GETTER Y SETTER
    public String getNombre(){
        return nombre;
    }
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
    public int getEdad(){
        return edad;
    }
    public void setEdad(int edad){
        this.edad = edad;
    }

    public void agregarDireccion(String calle, int numero){
        //CREAMOS LA INSTANCIA DENTRO DE PERSONA - RELACION DE COMPOSICION
        Direccion direccion = new Direccion(calle,numero);
        direcciones.add(direccion);
    }
    public void imprimirDirecciones(){
        for(Direccion direccion : direcciones){
            System.out.println("Direccion: "+direccion.getCalle()+"
                "+direccion.getNumero());
        }
    }
}
```

1.3 Creamos la Clase Main:

```
public class Main{
    public static void main(String[] args){
        //CONSTRUCTOR CREA CENTRO LA INSTANCIA DE DIRECCION
        Persona personal = new Persona("Juan Perez", 20, "calle las Gardenias",
            458);
        personal.agregarDireccion("Calle Nueva", 907);

        //MOSTRAMOS LOS DATOS DE LA PERSONA, INCLUYENDO LAS DIRECCIONES
        System.out.println("Datos de la persona");
        System.out.println("Nombre: "+personal.getNombre());
        System.out.println("Edad: "+personal.getEdad());
        personal.imprimirDirecciones();
```

```
    }
}
```

Ejecucion:

```
Datos de la persona
Nombre: Juan Perez
Edad: 20
Direccion: calle las Gardenias 458
Direccion: Calle Nueva 907
```

EJERCICIO 2: Crear un POO donde se implementa la relación de Agregación.

2.1 Creamos la Clase Cliente:

```
public class Cliente{
    //ATRIBUTOS
    private String codigo;
    private String nombre;
    private String apellidos;
    //CONSTRUCTORES
    public Cliente(){
    }
    public Cliente(String codigo, String nombre, String apellidos){
        this.codigo = codigo;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }
    //METODOS GETTER Y SETTER
    public String getCodigo(){
        return codigo;
    }
    public void setCodigo(String codigo){
        this.codigo = codigo;
    }
    public String getNombre(){
        return nombre;
    }
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
    public String getApellidos(){
        return apellidos;
    }
    public void setApellidos(String apellidos){
        this.apellidos = apellidos;
    }
}
```

2.2 Creamos la Clase Vehiculo:

```
public class Vehiculo {
    //ATRIBUTOS
```

```
    private String matricula;
    private String marca;
    private String modelo;
    private String color;
    private double tarifa;
    private boolean disponible;
    //CONSTRUCTORES
    public Vehiculo(){

    }
    public Vehiculo(String matricula, String marca, String modelo, String color,
         double tarifa, boolean disponible){
        this.matricula = matricula;
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
        this.tarifa = tarifa;
        this.disponible = disponible;
    }
    //METODOS GETTER Y SETTER
    public String getMatricula(){
        return matricula;
    }
    public void setMatricula(String matricula){
        this.matricula = matricula;
    }
    public String getMarca(){
        return marca;
    }
    public void setMarca(String marca){
        this.marca = marca;
    }
    public String getModelo(){
        return modelo;
    }
    public void setModelo(String modelo){
        this.modelo = modelo;
    }
    public String getColor(){
        return color;
    }
    public void setColor(String color){
        this.color = color;
    }
    public double getTarifa(){
        return tarifa;
    }
    public void setTarifa(double tarifa){
        this.tarifa = tarifa;
    }
    public boolean isDisonible(){
        return disponible;
    }
    public void setDisponible(boolean disponible){
        this.disponible = disponible;
    }
```

```
}
```

2.3 Creamos la Clase Alquiler:

```
public class Alquiler {
    //ATRIBUTOS
    private Cliente cliente;
    private Vehiculo vehiculo;
    private int diaAlquiler;
    private int mesAlquiler;
    private int aoAlquiler;
    private int totalDiasAlquiler;
    //CONSTRUCTORES
    public Alquiler(){

    }
    public Alquiler(Cliente cliente, Vehiculo vehiculo, int diaAlquiler, int
         mesAlquiler, int aoAlquiler, int totalDiasAlquiler){
        this.cliente = cliente;
        this.vehiculo = vehiculo;
        this.diaAlquiler = diaAlquiler;
        this.mesAlquiler = mesAlquiler;
        this.aoAlquiler = aoAlquiler;
        this.totalDiasAlquiler = totalDiasAlquiler;
    }
    //METODOS GETTER Y SETTER
    public Cliente getCliente(){
        return cliente;
    }
    public void setCliente(Cliente cliente){
        this.cliente = cliente;
    }
    public Vehiculo getVehiculo(){
        return vehiculo;
    }
    public void setVehiculo(Vehiculo vehiculo){
        this.vehiculo = vehiculo;
    }
    public int getDiaAlquiler(){
        return diaAlquiler;
    }
    public void setDiaAlquiler(int diaAlquiler){
        this.diaAlquiler = diaAlquiler;
    }
    public int getMesAlquiler(){
        return mesAlquiler;
    }
    public void setMesAlquiler(int mesAlquiler){
        this.mesAlquiler = mesAlquiler;
    }
    public int getAoAlquiler(){
        return aoAlquiler;
    }
    public void setAoAlquiler(int aoAlquiler){
        this.aoAlquiler = aoAlquiler;
```

```
    }
    public int getTotalDiasAlquiler(){
        return totalDiasAlquiler;
    }
    public void setTotalDiasAlquiler(int totalDiasAlquiler){
        this.totalDiasAlquiler = totalDiasAlquiler;
    }

}
```

2.4 Creamos la Clase Main, donde llamamos a nuestras clases y las operaciones implementadas:

```
import java.util.*;
public class Main {
    public static void main(String[] args){
        //COLECCION DE ALQUILERES
        ArrayList<Alquiler> alquileres = new ArrayList<Alquiler>();
        //CREAMOS OBJETO VEHICULO
        Vehiculo vehiculo1 = new Vehiculo("4050 ABJ", "VolsWagen", "GTI", "Blanco",
            100.0, true);
        //CREAMOS OBJETO VEHICULO
        Vehiculo vehiculo2 = new Vehiculo("2345 JVM", "SEAT", "Leon", "Negro",
            80.0, true);
        //CREAMOS OBJETO CLIENTE
        Cliente cliente1 = new Cliente("29681075", "Juan", "Perez");
        //CREAMOS RELACION DE AGREGACION
        Alquiler alquiler1 = new Alquiler(cliente1, vehiculo1, 11,0,2021,2);
        Alquiler alquiler2 = new Alquiler(cliente1, vehiculo2, 15,0,2021,3);
        //ADICIONAMOS LOS ALQUILERES A LA LISTA
        alquileres.add(alquiler1);
        alquileres.add(alquiler2);
        //MOSTRAMOS LOS OBJETOS COMPUESTOS
        for(Alquiler alquiler : alquileres){
            System.out.println("----------------------------");
            System.out.println("DATOS DEL ALQUILER");
            System.out.println("----------------------------");
            System.out.println("Fecha Alquiler:
                "+alquiler.getDiaAlquiler()+"/"+alquiler.getMesAlquiler()+"/"+alquiler.getAoAlquiler())
            System.out.println("Cantidad Dias: "+alquiler.getTotalDiasAlquiler());
            System.out.println("Cliente: "+alquiler.getCliente().getCodigo()+"
                "+alquiler.getCliente().getNombre()+"
                "+alquiler.getCliente().getApellidos());
            System.out.println("Vehiculo: "+alquiler.getVehiculo().getMatricula()+"
                "+alquiler.getVehiculo().getMarca()+"
                "+alquiler.getVehiculo().getModelo());
        }
    }
}
```

Ejecucion:

```
----------------------------
DATOS DEL ALQUILER
```

```
----------------------------
Fecha Alquiler: 11/0/2021
Cantidad Dias: 2
Cliente: 29681075 Juan Perez
Vehiculo: 4050 ABJ VolsWagen GTI
----------------------------
DATOS DEL ALQUILER
----------------------------
Fecha Alquiler: 15/0/2021
Cantidad Dias: 3
Cliente: 29681075 Juan Perez
Vehiculo: 2345 JVM SEAT Leon
```

EJERCICIO 3: Crear un POO donde se implementa la relación de Herencia.

3.1 Creamos la Clase Persona:

```java
public class Persona {
    //ATRIBUTOS
    private String dni;
    private String nombre;
    private String direccion;
    // CONTRUCTORES
    public Persona(){
        this.dni = "";
        this.nombre = "";
        this.direccion = "";
    }
    public Persona(String dni, String nombre, String direccion){
        this.dni = dni;
        this.nombre = nombre;
        this.direccion = direccion;
    }
    //METODOS GETTER Y SETTER
    public String getDni(){
        return dni;
    }
    public void setDni(String dni){
        this.dni = dni;
    }
    public String getNombre(){
        return nombre;
    }
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
    public String getDireccion(){
        return direccion;
    }
    public void setDireccion(String direccion){
        this.direccion = direccion;
    }
}
```

3.2 Creamos la Clase Padre Deposito:

```
public class Deposito {
    //ATRIBUTOS
    private Persona titular;
    private double capital;
    private int plazoDias;
    private double tipoInteres;
    //CONSTRUCTORES
    public Deposito(){
        this.titular = null;
        this.capital = 0.00;
        this.plazoDias = 0;
        this.tipoInteres = 0.00;
    }
    public Deposito(Persona titular, double capital, int plazoDias, double
         tipoInteres){
        this.titular = titular;
        this.capital = capital;
        this.plazoDias = plazoDias;
        this.tipoInteres = tipoInteres;
    }
    // METODOS SETTER Y GETTTER
    public Persona getTitular(){
        return titular;
    }
    public void setTitular(Persona titular){
        this.titular = titular;
    }
    public double getCapital(){
        return capital;
    }
    public void setCapital(double capital){
        this.capital = capital;
    }
    public int getPlazoDias(){
        return plazoDias;
    }
    public void setPlazoDias(int plazoDias){
        this.plazoDias = plazoDias;
    }
    public double getTipoInteres(){
        return tipoInteres;
    }
    public void setTipoInteres(double tipoInteres){
        this.tipoInteres = tipoInteres;
    }
    //METODOS PUBLICOS
    public double getIntereses(){
        return (plazoDias*tipoInteres*capital)/365;
    }
    public double liquidar(){
        return getCapital()+getIntereses();
    }
    public String toString(){
        StringBuilder sbCadena = new StringBuilder();
```

```
        sbCadena.append("\n Titular: "+titular.getDni()+titular.getNombre());
        sbCadena.append("\n Capital: "+capital);
        sbCadena.append("\n Plazo Dias: "+plazoDias);
        sbCadena.append("\n Tipo Interes: "+tipoInteres);
        sbCadena.append("\n TOTAL: "+liquidar());
        return sbCadena.toString();
    }
}
```

3.3 Creamos la Clase Hija Deposito Estructurado:

```
public class DepositoEstructurado extends Deposito {
    //ATRIBUTOS
    private double tipoInteresVariable;
    private double capitalVariable;
    //CONSTRUCTORES
    public DepositoEstructurado(){

    }
    public DepositoEstructurado(double tipoInteresVariable, double capitalVariable,
         Persona titular, double capital, int plazoDias, double tipoInteres){
        super(titular, capital, plazoDias, tipoInteres);
        this.tipoInteresVariable = tipoInteresVariable;
        this.capitalVariable = capitalVariable;
    }
    //METODOS GETTER Y SETTER
    public double getTipoInteresVariable(){
        return tipoInteresVariable;
    }
    public void setTipoInteresVariable(double tipoInteresVariable){
        this.tipoInteresVariable = tipoInteresVariable;
    }
    public double getCapitalVariable(){
        return capitalVariable;
    }
    public void setCapitalVariable(double capitalVariable){
        this.capitalVariable = capitalVariable;
    }
    //METODOS PUBLICOS
    public double getInteresesVariables(){
        return (getPlazoDias()*tipoInteresVariable*capitalVariable)/365;
    }
    public String toString(){
        StringBuilder sbCadena = new StringBuilder();
        sbCadena.append(super.toString());
        sbCadena.append("\n Tio Interes Variable: "+tipoInteresVariable);
        sbCadena.append("\n Capital Variable: "+capitalVariable);
        sbCadena.append("\n Interes Variable: "+getInteresesVariables());
        return sbCadena.toString();
    }
}
```

3.4 Creamos la Clase Main, donde llamamos a nuestras clases y las operaciones implementadas:

```
import java.util.Scanner;

public class Main {
    static Scanner consola = new Scanner(System.in);
    public Persona crearTitular(){
        Persona persona;
        System.out.println("Ingrese el nombre: ");
        String nombre = consola.next();
        System.out.println("Ingrese el DNI: ");
        String dni = consola.next();
        System.out.println("Ingrese la Direccion: ");
        String direccion = consola.next();
        persona = new Persona(dni, nombre, direccion);//CREAMOS INSTANCIA
        return persona; // DEVOLVEMOS LA INSTANCIA
    }
    public Deposito crearDeposito(int tipo){
        System.out.println("-------\nNUEVO DEPOSITO----------");
        System.out.println("DATOS DEL TITULAR\n-----");
        Persona persona = crearTitular();
        System.out.println("DATOS DE DEPOSITO\n------");
        System.out.println("Ingrese el Capital: ");
        double capital = consola.nextDouble();
        System.out.println("Ingrese el plazo Dias: ");
        int plazo = consola.nextInt();
        System.out.println("Ingrese el tio de interes: ");
    }
}
```

5. EJERCICIO PROPUESTO

5.1. INTRODUCCION

5.1.1. 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.

5.1.2. 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.

5.1.3. Los objetivos de este laboratorio son:

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

5.1.5. 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.

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

5.1.7. Evitar duplicación de código.

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

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

5.1.10. 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. . Salirla clase Main.java

-

```java
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 12
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class Main {

    public static void main(String[] args) {
```

```
//DECLARACION DE VARIABLES Y ARREGLOS NECESARIOS
ArrayList<Soldado> ejercito1 = new ArrayList();
ArrayList<Soldado> ejercito2 = new ArrayList();
ArrayList<ArrayList<Soldado>> tablero = new ArrayList();
int batallon1, batallon2;
int vidatotal1=0, vidatotal2=0;
double promedioVida1=0, promedioVida2=0;

// BUCLE PARA DESIGNAR LA CANTIDAD DE FILAS Y COLUMNAS DEL TABLERO
for(int i=0; i<10; i++) {
   tablero.add(new ArrayList<Soldado>());
   for(int j=0; j<10; j++) {
      tablero.get(i).add(new Soldado());
   }
}

// CREACION DEL NUMERO DE POSICIONES DE CADA EJERCITO
batallon1 = aleatorio(1,10);
batallon2 = aleatorio(1,10);

// INICIALIZAR ARREGLOS
inicializarArreglo(ejercito1, batallon1);
inicializarArreglo(ejercito2, batallon2);

// GENERAR EJERCITOS VALIDOS
generarEjercitos(ejercito1, ejercito2);

//  AADIR  LOS EJERCITOS AL TABLERO
aadirTablero(ejercito1, tablero);
aadirTablero(ejercito2, tablero);

//IMPRIMIR EL TABLERO
imprimirTablero(tablero);

//IMPRIMIR LOS SOLDADOS DE MAYOR VIDA DE CADA EJERCITO
System.out.println("Soldado de mayor vida del ejercito 1");
SoldadoConMayorVida(ejercito1);
System.out.println("soldado de mayor vida del ejercito 2");
SoldadoConMayorVida(ejercito2);

//IMPRIMIR LA VIDA TOTAL Y EL PROMEDIO DEL EJERCITO 1
System.out.println("\nEJERCITO 1: ");
for (int i=0; i<ejercito1.size(); i++) {
   vidatotal1+=ejercito1.get(i).getPuntos();
   promedioVida1 = vidatotal1/(ejercito1.size()*1.0);
}
System.out.println("Vida total: "+vidatotal1);
System.out.println("Promedio de vida: "+promedioVida1);

//IMPRIMIR LA VIDA TOTAL Y EL PROMEDIO DEL EJERCITO 2
System.out.println("\nEJERCITO 2: ");
for (int i=0; i<ejercito2.size(); i++) {
   vidatotal2+=ejercito2.get(i).getPuntos();
   promedioVida2 = vidatotal2/(ejercito2.size()*1.0);
}
System.out.println("Vida total: "+vidatotal2);
```

```
    System.out.println("Promedio de vida: "+promedioVida2);

    //IMPRIMIR LOS SOLDADOS CREADOS EN EL ORDEN POR DEFECTO
    System.out.println("\nLista ejercito 1:");
    for(int i=0; i<ejercito1.size(); i++) {
       imprimir(ejercito1.get(i));
    }
    System.out.println("\nLista ejercito 2:");
    for(int i=0; i<ejercito2.size(); i++) {
       imprimir(ejercito2.get(i));
    }

    // IMPRIMIR LOS DATOS DE LOS SOLDADOS ORDENADOS DE MAYOR A MENOR DEPENDIENDO
        DE SU NIVEL DE VIDA USANDO DOS TIPOS DE ALGORITMO
    ordenarPorVidaMetodoA(ejercito1);
    ordenarPorVidaMetodoB(ejercito2);
    System.out.println("\nEjercito 1 Ordenados por nivel de vida");
    for(int i=0; i<ejercito1.size(); i++) {
       imprimir(ejercito1.get(i));
    }
    System.out.println("\nEjercito 2 Ordenados por nivel de vida");
    for(int i=0; i<ejercito2.size(); i++) {
       imprimir(ejercito2.get(i));
    }

    // MOSTRAR EJERCITO GANADOR LA METRICA USADA ARA DESIGNAR AL GANADOR ES POR
        EL NIVEL DEL PROMEDIO DE VIDA DE CADA EJERCITO
    if(promedioVida1>promedioVida2) {
       System.out.println("\nGANADOR ***EJERCITO 1***");
    }else if (promedioVida1<promedioVida2) {
       System.out.println("\nGANADOR ***EJERCITO 2***");
       }
    else {
       System.out.print("\n***ES UN EMPATE***");
    }
}

 // 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.setNivelVida(aleatorio(1, 5));
     soldadito.setFila(aleatorio(1, 10));
     soldadito.setColumna(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).setActitud(B1.get(i).getNivelVida() + "[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).setActitud(B2.get(i).getNivelVida() + "[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).getActitud());
        }
        System.out.println("\n");
    }
}
```

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

    // 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).getNivelVida() < soldadito.get(j +
                    1).getNivelVida()) {
                    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 nivel de vida
                return Integer.compare(s2.getNivelVida(), s1.getNivelVida());
            }
        });
    }
}
    // 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).getActitud());
            }
            System.out.println("\n");
        }
```

```
    }

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

    // 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).getNivelVida() < soldadito.get(j +
                    1).getNivelVida()) {
                    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 nivel de vida
                return Integer.compare(s2.getNivelVida(), s1.getNivelVida());
            }
        });
    }
}
```

- la clase Soldado.java

```
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 12
// FUNDAMENTOS DE PROGRAMACION
// CLASE SOLDADO PARA LOS METODOS SETTER Y GETTER
```

```
public class Soldado {
    private String nombre;
    private int nivelAtaque;
    private int nivelDefensa;
    private int nivelVida;
    private int vidaActual;
    private int velocidad;
    private String actitud;
    private boolean vive;

    public Soldado() {
        nombre = "";
        nivelAtaque = 0;
        nivelDefensa = 0;
        nivelVida = 0;
        vidaActual = nivelVida;
        velocidad = 0;
        actitud = "";
        vive = true;
    }

    public void atacar(Soldado enemigo) {
        int danio = this.nivelAtaque - enemigo.nivelDefensa;
        if (danio > 0) {
            enemigo.serAtacado(danio);
            System.out.println(this.nombre + " atac a " + enemigo.nombre + " y le
                caus " + danio + " de dao.");
        } else {
            System.out.println(this.nombre + " atac a " + enemigo.nombre + " pero no
                le caus dao.");
        }
    }

    public void defender() {
        // Lgica para la defensa
        // Puede disminuir el dao recibido en un futuro ataque, por ejemplo
        this.nivelDefensa += 10; // Aumentar la defensa por ejemplo
        System.out.println(this.nombre + " se est defendiendo.");
    }

    public void avanzar() {
        // Lgica para avanzar en el juego
        // Podra mover al soldado a una nueva posicin en el tablero, por ejemplo
        if (this.velocidad > 0) {
            // Mover al soldado en la direccin correspondiente
            System.out.println(this.nombre + " est avanzando.");
        } else {
            System.out.println(this.nombre + " no puede avanzar, la velocidad es
                0.");
        }
    }

    public void retroceder() {
        // Lgica para retroceder en el juego
        // Podra mover al soldado hacia atrs en el tablero, por ejemplo
        System.out.println(this.nombre + " est retrocediendo.");
```

```
}

public void serAtacado(int danioRecibido) {
    this.vidaActual -= danioRecibido;
    if (this.vidaActual <= 0) {
        this.morir();
    }
}

public void huir() {
    // Lgica  para huir del combate
    // Podra  cambiar la posicin del soldado a una zona segura, por ejemplo
    if (this.nivelVida < 10) {
        // Huir solo si la vida es baja
        System.out.println(this.nombre + " est huyendo del combate.");
    } else {
        System.out.println(this.nombre + " no puede huir, su vida est alta.");
    }
}

public void morir() {
    this.vidaActual = 0;
    this.vive = false;
}

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

public int getVidaActual() {
    return vidaActual;
}

// Getters y Setters

public String getNombre() {
    return nombre;
}

public String getNivelVida() {
    return nivelVida;
}

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

public int getNivelAtaque() {
    return nivelAtaque;
}

public int setNivelVida() {
    this.nivelVida = nivelVida;
}

public void setNivelAtaque(int nivelAtaque) {
```

```
        this.nivelAtaque = nivelAtaque;
    }

    public int getNivelDefensa() {
        return nivelDefensa;
    }

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

    public boolean isVive() {
        return vive;
    }

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

- 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***
```

6. CUESTINARIO

- ¿Qué es una relación de agregación entre clases? La relación de agregación es un concepto en la programación orientada a objetos (POO) que describe una asociación entre clases en la que una clase es parte de otra clase. En términos más sencillos, la relación de agregación indica que una clase "tiene.º .está compuesta por.ºtras clases. Esta relación implica que una clase contiene instancias de otra clase como parte de su estructura.
- ¿Qué diferencias tienen las relaciones de agregación y composición? as relaciones de agregación y composición son dos tipos de asociaciones entre clases en programación orientada a objetos (POO), y aunque comparten similitudes, hay diferencias clave entre ellas. Aquí se presentan las principales diferencias:
- ¿Cómo se implementa en Java la relación de agregación? En Java, la implementación de la relación de agregación implica crear una instancia de la clase compuesta y pasarla como parámetro o asignarla como un atributo de la clase contenedora. Aquí hay un ejemplo simple para ilustrar cómo se podría implementar la relación de agregación en Java:
- ¿Cómo se implementa en Java la relación de composición? En Java, la implementación de la relación de composición implica que los objetos de una clase están fuertemente vinculados a los objetos de otra clase, y los objetos compuestos generalmente se crean y destruyen junto con la clase contenedora. Aquí hay un ejemplo simple que ilustra cómo se podría implementar la relación de composición:
- ¿Qué ventajas tiene la relación de herencia? La relación de herencia es un concepto fundamental en la programación orientada a objetos (POO) que permite la creación de una nueva clase basada en una clase existente, denominada clase base o superclase. La herencia ofrece varias ventajas que pueden facilitar el diseño y la implementación de software. Aquí hay algunas de las ventajas clave de la relación de herencia:

7. 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.