Informe de Laboratorio 15

Tema: Mecanismos de Agregación, Composición, Herencia (II)

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
15	Mecanismos de Agregación, Composición, Herencia (II)	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 13 Noviembre 2023	Al 18 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, composición, herencia.

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 15 en el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO/FP2-LAB-15`

4. EJERCICIO RESUELTO

EJERCICIO 1: Crear un POO donde se implementa las propiedades de instancia y propiedades de clase.

1.1 Creamos la Clase Empleado:

```
public class Empleado{
    //ATRIBUTOS
    private String nombre, categoria;
    private int numeroHijos;
    private double sueldoBasico, bonificacionCategoria, bonificacionEscolaridad,
        sueldoNeto;
    //ATRIBUTOS STATIC
    public static double sumaSueldosNombrados = 0.00;
    public static double sumaSueldosContratados = 0.00;
    public static double sumaSueldosPorHoras = 0.00;

    public static int contaMenos3Hijos = 0;
    public static int contaMenos6Hijos = 0;
    public static int contaMas6Hijos = 0;
    //CONSTRUCTORES
    public Empleado(){
        this.nombre = "";
        this.categoria = "";
        this.numeroHijos = 0;
        this.sueldoBasico = 0;
        this.bonificacionCategoria = 0.00;
        this.bonificacionEscolaridad = 0.00;
        this.sueldoNeto = 0.00;
    }
    public Empleado(String nombre, String categoria, int numeroHijos, double
        sueldoBasico){
        this.nombre = nombre;
        this.categoria = categoria;
        this.numeroHijos = numeroHijos;
        this.sueldoBasico = sueldoBasico;
        this.bonificacionCategoria = 0.00;
        this.bonificacionEscolaridad = 0.00;
        this.sueldoNeto = 0.00;
    }
    //METODO GETTER Y SETTER
    public String getNombre(){
        return nombre;
    }
    public void setNombre(String nombre){
```

```
        this.nombre = nombre;
    }
    public String getCategoria(){
        return categoria;
    }
    public void setCategoria(String categoria){
        this.categoria = categoria;
    }
    public int getNumeroHijos(){
        return numeroHijos;
    }
    public void setNumeroHijos(int numeroHijos){
        this.numeroHijos = numeroHijos;
    }
    public double getSueldoBasico(){
        return sueldoBasico;
    }
    public void setSueldoBasico(double sueldoBasico){
        this.sueldoBasico = sueldoBasico;
    }
    public double getBonificacionCategoria(){
        return bonificacionCategoria;
    }
    public double getBonificacionEscolaridad(){
        return bonificacionEscolaridad;
    }
    public double getSueldoNeto(){
        return sueldoNeto;
    }
    //METODOS PRIVADOSO
    private void CalcularBonificacion(){
        switch (this.categoria.toLowerCase().trim()){
            case "nombrado":
                this.bonificacionCategoria = this.sueldoBasico * 0.12;
                break;
            case "contratado":
                this.bonificacionCategoria = this.sueldoBasico * 0.10;
                break;
            case "por horas":
                this.bonificacionCategoria = this.sueldoBasico * 0.08;
                break;
            default:
                this.bonificacionCategoria = this.sueldoBasico * 0.06;
        }
    }
    private void calcularEscolaridad(){
        if(numeroHijos>=1 && numeroHijos<3){
            bonificacionEscolaridad = numeroHijos*20;
            contaMenos3Hijos++;
        }else if (numeroHijos <= 6){
            bonificacionEscolaridad = numeroHijos*30;
            contaMenos6Hijos++;
        }else if(numeroHijos > 6){
            bonificacionEscolaridad = numeroHijos*40;
            contaMas6Hijos++;
        }
```

```
    }
    // METODOS UBLICOS
    public void calcularSueldoNeto(){
        this.CalcularBonificacion();
        this.calcularEscolaridad();
        this.sueldoNeto = sueldoBasico + bonificacionCategoria +
            bonificacionEscolaridad;
    }
}
```

1.2 Creamos la Clase Persona:

```
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:

```java
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:

```
ingrese NOMBRES del Empleado
roni
Ingrese CATEGORIA del Empleado
Nombrado |  Contratado | Por Horas:
nombrado
Ingrese SUELDO BASICO del Empleado:
3455
Ingrese NUMERO DE HIJOS del Empleado:
2
Bonificacion Categoria: 414.59999999999997
Bonificacion Escolaridad: 40.0
Sueldo Neto: 3909.6
 Desea  seguir? (si/no)
no

--------------
RESULTADOS FINALES
------------------
Total Sueldos Nombrados: 3909.6
Total Sueldos Contratados: 0.0
Total Sueldos Por Horas: 0.0
Numero Empleados con MENOS de 3 Hijos: 1
Numero Empleados con MENOS de 6 Hijos: 0
Numero Empleados con MAS de 6 Hijos: 0
```

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

2.1 Creamos la Clase Persona:

```java
public class Persona {
    //ATRIBUTOS
    private String dni;
    private String nombre;
    private String direccion;
    //CONSTRUCTORES
```

```
    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;
    }
}
```

2.2 Creamos la Clase Padre Deposito:

```
public class Deposito {
    //ATRIBUTOS
    private Persona titular; // RELACION ASOCIACION
    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 GETTER Y SETTER
    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();
    }
}
```

2.3 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("\n Tipo Interes Variable: "+tipoInteresVariable);
        sbCadena.append("\n Capital Variable: "+ capitalVariable);
        sbCadena.append("Interes Variable: "+getInteresesVariables());
        return sbCadena.toString();
    }
}
```

2.4 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);
        return persona;
    }
    public Deposito crearDeposito(int tipo){
        System.out.println("----------------------\n NUEVO
            DEPOSITO\n--------------------------");
        System.out.println("DATOS DEL TITULAR\n ------------------");
        Persona persona = crearTitular();
        System.out.println("\nDATOS DEL 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 Tipo de Interes: ");
        double tipoInteres = consola.nextDouble();
        if(tipo == 1){
            Deposito objDeposito = new Deposito(persona, capital, plazo,
                tipoInteres);
            return objDeposito;
        }
        else{
            System.out.println("Ingrese el Interes Variable: ");
            double interesVariable = consola.nextDouble();
            System.out.println("Ingrese el Capital Variable: ");
            double capitalVariable = consola.nextDouble();
            DepositoEstructurado objDepositoEstructurado = new
                DepositoEstructurado(interesVariable, capitalVariable, persona,
                capital, plazo, tipoInteres);
            return objDepositoEstructurado;
        }
    }
    public static void main(String[] args){
        Main objPrincipal = new Main();
        System.out.println("1. Crear Deposito");
        System.out.println("2. Crear Deposito Estructurado");
        int opcion = consola.nextInt();
        Deposito deposito = objPrincipal.crearDeposito(opcion);
        System.out.println(deposito.toString());
    }
}
```

Ejecucion:

```
1. Crear Deposito
2. Crear Deposito Estructurado
1
-----------------------
 NUEVO DEPOSITO
-------------------------
DATOS DEL TITULAR
 ------------------
Ingrese el nombre:
roni
Ingrese el DNI:
72025070
Ingrese la Direccion:
calle nueva

DATOS DEL DEPOSITO
--------------------
Ingrese el Capital:
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextDouble(Scanner.java:2564)
        at Main.crearDeposito(Main.java:22)
        at Main.main(Main.java:45)
```

```
PS C:\Users\Roni\Desktop\FP2 LAB 15> c:; cd 'c:\Users\Roni\Desktop\FP2 LAB 15'; &
    'C:\Program Files\Java\jdk-16.0.1\bin\java.exe' '-XX:+Sho
wCodeDetailsInExceptionMessages' '-cp'
    'C:\Users\Roni\AppData\Roaming\Code\User\workspaceStorage\b61e33d765771421b91829ee7ca87311\redhat.
a\jdt_ws\FP2 LAB 15_b3203bda\bin' 'Main'
Error: Could not find or load main class Main
Caused by: java.lang.ClassNotFoundException: Main
PS C:\Users\Roni\Desktop\FP2 LAB 15> c:; cd 'c:\Users\Roni\Desktop\FP2 LAB 15'; &
    'C:\Program Files\Java\jdk-16.0.1\bin\java.exe' '-XX:+Sho
wCodeDetailsInExceptionMessages' '-cp'
    'C:\Users\Roni\AppData\Roaming\Code\User\workspaceStorage\b61e33d765771421b91829ee7ca87311\redhat.
a\jdt_ws\FP2 LAB 15_b3203bda\bin' 'Main'
1. Crear Deposito
2. Crear Deposito Estructurado
2
-----------------------
 NUEVO DEPOSITO
-------------------------
DATOS DEL TITULAR
 ------------------
Ingrese el nombre:
roni
Ingrese el DNI:
72025070
Ingrese la Direccion:
rio

DATOS DEL DEPOSITO
---------------------
Ingrese el Capital:
300
Ingrese el Plazo Dias:
56
Ingrese el Tipo de Interes:
45
Ingrese el Interes Variable:
5
Ingrese el Capital Variable:
344

 Tipo Interes Variable: 5.0
 Capital Variable: 344.0Interes Variable: 263.8904109589041
```

5. EJERCICIO PROPUESTO

PROBLEMA 01

Basándose en la clase Soldado de las prácticas anteriores, crear una nueva clase Ejército (clase contenedora) que tendrá como parte a un conjunto de Soldados (cada soldado es exclusivo de un ejército). Cada ejército debe pertenecer a algún reino específico (Inglaterra, Francia, Castilla-Aragón, Sacro Imperio Romano-Germánico, Moros, etc.).

Considere constructores sobrecargados sets y gets

toString (devuelve datos generales del ejército y de todos sus soldados)

Que Ejército tenga un atributo misSoldados (puede ser un arreglo estándar, ArrayList o HashMap de Soldado, ud. debe elegir la mejor opción). Considerar un máximo 10 soldados por ejército

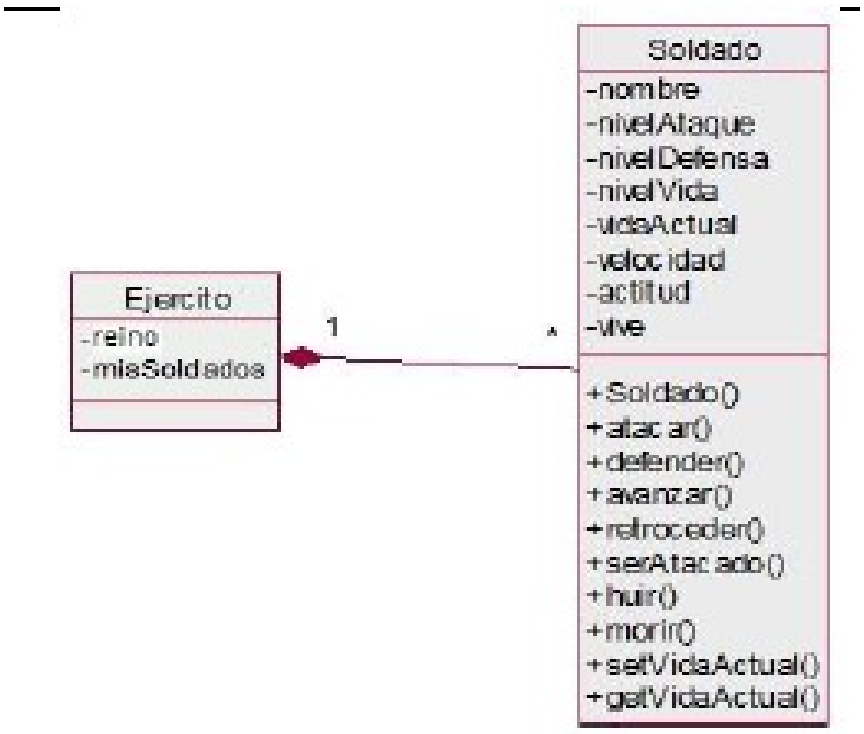
Ejército debe tener métodos para ingresar los datos de todos sus soldados (2 opciones: manualmente y autogenerado), modificarlos y eliminarlos (crear algún menú pensando en el usuario). Considerar 1..10 soldados por cada ejército.

También se debe poder consultar el Soldado con mayor nivel de ataque, ver ranking de poder considerando el nivel de vida (descendente) y ver todos los datos del ejército y de los soldados que lo conforman

Un main que permita crear un Ejército y gestionarlo por medio del menú

Crear los atributos y métodos extras que considere necesarios

Basarse en el siguiente diagrama de clases de UML y crear un diagrama completo que considere inclusive la clase principal



la clase Main.java

```java
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 15
// 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 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;
    }
}
```

- la clase Ejercito.java

```
import java.util.ArrayList;

public class Ejercito {
    private String reino;
    private ArrayList<Soldado> misSoldados;

    // Constructor
    public Ejercito(String reino) {
        this.reino = reino;
        this.misSoldados = new ArrayList<>();
    }

    // Getters y Setters
    public String getReino() {
        return reino;
    }

    public void setReino(String reino) {
        this.reino = reino;
    }

    // Mtodo para agregar un soldado al ejrcito
    public void agregarSoldado(Soldado soldado) {
        if (misSoldados.size() < 10) {
            misSoldados.add(soldado);
        } else {
            System.out.println("El ejrcito ya tiene el mximo de soldados permitidos
```

```
            (10).");
    }
}

// Mtodo para eliminar un soldado por nombre
public void eliminarSoldado(String nombreSoldado) {
    for (Soldado soldado : misSoldados) {
        if (soldado.getNombre().equals(nombreSoldado)) {
            misSoldados.remove(soldado);
            System.out.println("Soldado eliminado exitosamente: " +
                nombreSoldado);
            return;
        }
    }
    System.out.println("Soldado no encontrado: " + nombreSoldado);
}

// Mtodo para modificar un atributo de un soldado por nombre
public void modificarSoldado(String nombreSoldado, String atributo, Object
    valor) {
    for (Soldado soldado : misSoldados) {
        if (soldado.getNombre().equals(nombreSoldado)) {
            switch (atributo) {
                case "nombre":
                    soldado.setNombre((String) valor);
                    break;
                // Agregar ms casos segn los atributos que se puedan modificar
                default:
                    System.out.println("Atributo no vlido: " + atributo);
                    return;
            }
            System.out.println("Soldado modificado exitosamente: " +
                nombreSoldado);
            return;
        }
    }
    System.out.println("Soldado no encontrado: " + nombreSoldado);
}

// Mtodo para consultar el soldado con mayor nivel de ataque
public Soldado consultarSoldadoMasPoderoso() {
    Soldado masPoderoso = null;
    int maxNivelAtaque = Integer.MIN_VALUE;

    for (Soldado soldado : misSoldados) {
        if (soldado.getNivelAtaque() > maxNivelAtaque) {
            maxNivelAtaque = soldado.getNivelAtaque();
            masPoderoso = soldado;
        }
    }

    return masPoderoso;
}

// Mtodo para ver el ranking de poder considerando el nivel de vida
    (descendente)
```

```
    public ArrayList<Soldado> consultarRankingPoder() {
        ArrayList<Soldado> ranking = new ArrayList<>(misSoldados);
        ranking.sort((s1, s2) -> Integer.compare(s2.getVidaActual(),
            s1.getVidaActual()));
        return ranking;
    }

    // Mtodo para ver todos los datos del ejrcito y de los soldados que lo
        conforman
    @Override
    public String toString() {
        StringBuilder result = new StringBuilder("Ejrcito de " + reino + ":\n");
        for (Soldado soldado : misSoldados) {
            result.append("\t").append(soldado.toString()).append("\n");
        }
        return result.toString();
    }
}
```

- 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]
```

PROBLEMA 02

Basándose en los laboratorios anteriores

Crear la clase Mapa, que esté constituida por el tablero antes visto, pero que en lugar de soldados, posicione ejércitos en ciertas posiciones aleatorias (entre 1 y 10 ejércitos por cada reino). Se deben generar ejércitos de 2 reinos, entre 1 y 10 soldados por ejército. No se admite guerra civil. El Mapa tiene como atributo el tipo de territorio que puede ser (bosque, campo abierto, montaña, desierto, playa). La cantidad de ejércitos y cantidad de soldados por ejército, así como todos sus atributos se deben generar aleatoriamente

Dibujar el Mapa con las restricciones que sólo 1 ejercito como máximo en cada cuadrado

El mapa tiene un solo tipo de territorio autogenerado

Considerar que el territorio influye en los resultados de las batallas, así cada reino tiene bonus según el territorio: Inglaterra-¿bosque, Francia-¿campo abierto, Castilla-Aragón-¿montaña, Moros-¿desierto, Sacro Imperio RomanoGermánico-¿bosque, playa y campo abierto. En dichos casos, se aumenta el nivel de vida en 1 a todos los soldados del reino beneficiado

Realizar el diagrama de clases de UML completo

Indicar quién ganaría la guerra y justificar por qué. Describir 3 métricas usadas

Hacerlo iterativo

Ejecucion

```
|       Ejrcito       |
+---------------------+
| - reino: String     |
| - misSoldados: ArrayList<Soldado> |
+---------------------+
| + Ejrcito(reino: String) |
| + setReino(reino: String) |
| + getReino(): String    |
| + agregarSoldado(soldado: Soldado) |
| + eliminarSoldado(nombre: String) |
| + modificarSoldado(nombre: String, atributo: String, valor: Object) |
| + consultarSoldadoMasPoderoso(): Soldado |
| + consultarRankingPoder(): ArrayList<Soldado> |
| + toString(): String     |
+---------------------+
```

```
+----------------------+
|     Soldado          |
+----------------------+
| - nombre: String     |
| - nivelAtaque: int   |
| - nivelDefensa: int  |
| - puntos: int        |
| - vidaActual: int    |
| - velocidad: int     |
| - actitud: String    |
| - vive: boolean      |
+----------------------+
| + Soldado(nombre: String, nivelAtaque: int, nivelDefensa: int, puntos: int,
    vidaActual: int, velocidad: int, actitud: String) |
| + setNombre(nombre: String) |
| + setNivelAtaque(nivelAtaque: int) |
| + setNivelDefensa(nivelDefensa: int) |
| + setPuntos(puntos: int) |
| + setVidaActual(vidaActual: int) |
| + setVelocidad(velocidad: int) |
| + setActitud(actitud: String) |
| + setVive(vive: boolean) |
| + serAtacado(danio: int) |
| + atacar() |
| + avanzar() |
| + defender() |
| + huir() |
| + retroceder() |
| + morir() |
| + aumentarVida() |
| + toString(): String |
+----------------------+
        Ejercito
|- reino: String
|- misSoldados: ArrayList<Soldado>

|+ Ejercito(reino: String)
|+ getReino(): String
|+ setReino(reino: String): void
|+ agregarSoldado(soldado: Soldado): void
|+ eliminarSoldado(nombreSoldado: String): void
|+ modificarSoldado(nombreSoldado: String, atributo: String, valor: Object): void
|+ consultarSoldadoMasPoderoso(): Soldado
|+ consultarRankingPoder(): ArrayList<Soldado>
|+ toString(): String
```

6. CUESTIONARIO

- ¿Qué son la propiedades y métodos de instancia, qué diferencia tienen con las propiedades y métodos de clase?
  En la programación orientada a objetos, tanto en lenguajes como Python, Java, C++, entre otros, se utilizan conceptos como propiedades y métodos de instancia, así como propiedades y métodos de clase. Estos conceptos son fundamentales para entender la estructura y el

comportamiento de las clases.

- ¿Qué ventajas tiene la relación de herencia?
  La herencia es uno de los conceptos fundamentales de la programación orientada a objetos (POO) y proporciona varias ventajas en el diseño y desarrollo de software.
- ¿Qué es la sobreescritura y la sobrecarga de métodos?
  La sobreescritura de métodos ocurre cuando una clase hija proporciona una implementación específica para un método que ya está definido en su clase base (o en una de sus clases ancestrales). La firma del método (nombre y parámetros) en la clase hija es la misma que la firma en la clase base. La sobreescritura permite a las clases hijas proporcionar su propia implementación del método, reemplazando así la implementación de la clase base.

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.