Informe de Laboratorio 17
Tema: Programación Orientada a Objetos (III)

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
17	Programación Orientada a Objetos (III)	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 28 Noviembre 2023	Al 05 Diciembre 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 17 en el Repositorio GitHub.
- `https://github.com/RONI-COMPANOCCA-CHECCO/FP2-LAB-17`

4. EJERCICIO RESUELTO

EJERCICIO 1: Crear un POO donde se implementa la herencia y el polimorfismo con interfaces.

1.1 Creamos la Interface ArregloInterface dentro del paquete Interface:

```
package Interface;
public Interface ArregloInterface {
    public boolean estaVacio();
    public boolean estaLleno();
    public void insertar(int elemento, int posicion);
    public void agregar(int elemento);
    public int retirarUltimo();
    public void eliminar(int elemento);
    public int buscar(int elemento);
    public String imprimir();
}
```

1.2 Creamos la Clase ArregloImpl que implementa la interface ArregloInterface, dentro del paquete Impl:

```
package Impl;
import Interface.ArregloInterface;
public class ArregloImpl implements ArregloInterface{
    int Arreglo [];
    int puntero=0;

    //CONSTRUCTOR
    public ArregloImpl(int tamao){
        Arreglo = new int [tamao];
    }

    public boolean estaVacio(){
        return (puntero == 0 ? true : false);
    }

    public boolean estaLleno(){
        return (puntero >= Arreglo.length ? true : false);
    }

    public void insertar(int elemento, int posicion){
        int aux=0;
        if(!estaLleno()){
            for(int i=Arreglo.length-1; i>posicion; i--){
```

```
            Arreglo[i-1] = Arreglo[i];
        }
        Arreglo[posicion]=elemento;
        puntero++;
    }else{
        System.out.println("ARREGLO LLENO!!!");
    }
}

public void agregar(int elemento){
    if(!estaLleno()){
        Arreglo[puntero]=elemento;
        puntero++;
    }else{
        System.out.println("ARREGLO LLENO!!!");
    }
}

public int retirarUltimo(){
    int elemento = -1;
    if(!estaVacio()){
        elemento = Arreglo[puntero-1];
        Arreglo[puntero-1]=0;
        puntero--;
    }else
        System.out.println("ARREGLO VACIO!!!");
    return elemento;
}

public void eliminar(int posicion){
    int aux =0;
    if (!estaVacio()){
        for(int i=posicion; i<Arreglo.length-1; i++){
            Arreglo[i]=Arreglo[i+1];
        }
        puntero--;
    }else{
        System.out.println("ARREGLO VACIO!!!");
    }
}

public int buscar(int elemento){
    int arregloBusqueda[] = Arreglo.clone();
    Ordenamiento_SelectionSort(arregloBusqueda, 1);
    return metodoBinario(arregloBusqueda, elemento);
}

public String imprimir(){
    StringBuilder cadena = new
        StringBuilder("LISTA\n-------------------------\n");
    for(int x : Arreglo)
        cadena.append(x+" - ");
    return cadena.toString();
}

private int metodoBinario(int [] arreglo, int datoBuscado){
```

```
        int n = arreglo.length;
        int centro, inferior=0 , superior=n-1;
        while(inferior <= superior){
            centro = (superior+inferior)/2;
            if(arreglo[centro]==datoBuscado)
                return centro;
            else
                if (datoBuscado<arreglo[centro])
                    superior=centro-1;
                else
                    inferior = centro+1;

        }
        return -1;
    }

    private void Ordenamiento_SelectionSort(int [] arreglo, int tipo){
        for (int i=0; i<arreglo.length -1; i++){
            int min = i;
            for (int j=i+1; j<arreglo.length; j++){
                if(tipo==1){
                    if(arreglo[j]<arreglo[min]){
                        min = j;
                    }
                }else{
                    if(arreglo[j]>arreglo[min]){
                        min = j;
                    }
                }
            }
            if ((i!=min)) {
                int aux = arreglo[i];
                arreglo[i] = arreglo[min];
                arreglo[min] = aux;
            }
        }
    }
}
```

1.3 Creamos la Clase Main, donde instanciamos a nuestras clases y llamamos a las operaciones implementadas en la clase, haciendo uso del polimorfismo:

```
package tadarreglo.tad_arreglo;
import java.util.Scanner;
import Impl.ArregloImpl;
import Interface.ArregloInterface;
public class Main {
    static Scanner consola = new Scanner(System.in);
    private int menu(){
        int opcion;
        System.out.println(
            "-------------------------------------------\n"+
            "1. Agregar en TAD Arreglo\n"+
            "2. Retirar de TAD Arreglo\n"+
            "3. Insertar en TAD Arreglo\n"+
```

```
            "4. Buscar en TAD Arreglo\n"+
            "5. Eliminar del TAD Arreglo\n"+
            "6. Imprimir TAD Arreglo\n"+
            "0. FIN\n"+
            "-------------------------------------------\n"+
            "Ingrese la ocion [1-7]");
        opcion = consola.nextInt();
        return opcion;
    }

    public static void main(String[] args){
        int num = 1;
        System.out.println("Ingrese la Cantidad de Elementos: ");
        int cantidad = consola.nextInt();
        ArregloImpl TadArreglo = new ArregloImpl(cantidad);

        int opcion;
        Main objPrincipal = new Main();
        do{
            opcion = objPrincipal.menu();
            switch (opcion) {
                case 1:
                    System.out.println("Ingrese un Numero: ");
                    num = consola.nextInt();
                    TadArreglo.agregar(num);
                    break;

                case 2:
                    num = TadArreglo.retirarUltimo();
                    System.out.println("Eliminado: "+num);
                    System.out.println(TadArreglo.imprimir());
                    break;

                case 3:
                    System.out.println("Ingrese un numero: ");
                    num = consola.nextInt();
                    System.out.println("Ingrese la posicion: ");
                    int pos = consola.nextInt();
                    TadArreglo.insertar(num, pos);
                    break;

                case 4:
                    System.out.println("Ingrese un Numero a Buscar: ");
                    int numero = consola.nextInt();
                    int posicion = TadArreglo.buscar(numero);
                    if(posicion != -1)
                        System.out.println("Numero Encontrado");
                    else
                        System.out.println("Numero no ENcontrado! ");
                    break;

                case 5:
                    System.out.println("Ingrese la Posicion a Eliminar: ");
                    num = consola.nextInt();
                    TadArreglo.eliminar(num);
                    break;
```

```
                case 6:
                    System.out.println(TadArreglo.imprimir());
                    break;

                case 0:
                    System.exit(0);
                    break;
            }
        }while(true);
    }
}
```

Ejecucion:

```
Ingrese la Cantidad de Elementos:
2
-------------------------------------------
1. Agregar en TAD Arreglo
2. Retirar de TAD Arreglo
3. Insertar en TAD Arreglo
4. Buscar en TAD Arreglo
5. Eliminar del TAD Arreglo
6. Imprimir TAD Arreglo
0. FIN
-------------------------------------------
Ingrese la ocion [1-7]
1
Ingrese un Numero:
4
-------------------------------------------
1. Agregar en TAD Arreglo
2. Retirar de TAD Arreglo
3. Insertar en TAD Arreglo
4. Buscar en TAD Arreglo
5. Eliminar del TAD Arreglo
6. Imprimir TAD Arreglo
0. FIN
-------------------------------------------
Ingrese la ocion [1-7]
6
LISTA
-------------------------
4 - 0 -
-------------------------------------------
1. Agregar en TAD Arreglo
2. Retirar de TAD Arreglo
3. Insertar en TAD Arreglo
4. Buscar en TAD Arreglo
5. Eliminar del TAD Arreglo
6. Imprimir TAD Arreglo
0. FIN
-------------------------------------------
Ingrese la ocion [1-7]
6
```

```
LISTA
--------------------------
4 - 0 -
-------------------------------------------
1. Agregar en TAD Arreglo
2. Retirar de TAD Arreglo
3. Insertar en TAD Arreglo
4. Buscar en TAD Arreglo
5. Eliminar del TAD Arreglo
6. Imprimir TAD Arreglo
0. FIN
-------------------------------------------
Ingrese la ocion [1-7]
0
```

5. EJERCICIO PROPUESTO

PROBLEMA 01

Crear diagrama de clases UML y programa.

Crear los miembros de cada clase de la forma más adecuada: como miembros de clase o de instancia.

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

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

El mapa tiene un solo tipo de territorio.

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, campo abierto. En dichos casos, se aumenta el nivel de vida en 1 a todos los soldados del reino beneficiado.

En la historia, los ejércitos estaban conformados por diferentes tipos de soldados, que tenían similitudes, pero también particularidades.

Basándose en la clase Soldado crear las clases Espadachín, Arquero, Caballero y Lancero. Las cuatro clases heredan de la superclase Soldado pero aumentan atributos y métodos, o sobrescriben métodos heredados.

Los espadachines tienen como atributo particular "longitud de espadaz como acción çrear un muro de escudos"que es un tipo de defensa en particular.

Los caballeros pueden alternar sus armas entre espada y lanza, además de desmontar (sólo se realiza cuando está montando e implica defender y cambiar de arma a espada), montar (sólo se realiza cuando está desmontado e implica montar, cambiar de arma a lanza y

envestir). El caballero también puede envestir, ya sea montando o desmontando, cuando es desmontado equivale a atacar 2 veces pero cuando está montando implica a atacar 3 veces.

Los arqueros tienen un número de flechas disponibles las cuales pueden dispararse y se gastan cuando se hace eso.

Los lanceros tienen como atributo particular, "longitud de lanzaz como acción "schiltrom"(como una falange que es un tipo de defensa en particular y que aumenta su nivel de defensa en 1).

Tendrá 2 Ejércitos que pueden ser constituidos sólo por espadachines, caballeros, arqueros y lanceros. No se acepta guerra civil. Crear una estructura de datos conveniente para el tablero. Los soldados del primer ejército se almacenarán en un arreglo estándar y los soldados del segundo ejército se almacenarán en un ArrayList. Cada soldado tendrá un nombre autogenerado: Espadachin0X1, Arquero1X1, Caballero2X2, etc., un valor de nivel de vida autogenerado aleatoriamente, la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado) y valores autogenerados para el resto de atributos.

Todos los caballeros tendrán los siguientes valores: ataque 13, defensa 7, nivel de vida [10..12] (el nivel de vida actual empieza con el valor del nivel de vida).

Todos los arqueros tendrán los siguientes valores: ataque 7, defensa 3, nivel de vida [3..5] (el nivel de vida actual empieza con el valor del nivel de vida).

Todos los espadachines tendrán los siguientes valores: ataque 10, defensa 8, nivel de vida [8..10] (el nivel de vida actual empieza con el valor del nivel de vida).

Todos los lanceros tendrán los siguientes valores: ataque 5, defensa 10, nivel de vida [5..8] (el nivel de vida actual empieza con el valor del nivel de vida).

Mostrar el tablero, distinguiendo los ejércitos y los tipos de soldados creados. Además, se debe mostrar todos los datos de todos los soldados creados para ambos ejércitos. Además de los datos del soldado con mayor vida de cada ejército, el promedio de nivel de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando algún algoritmo de ordenamiento.

Finalmente, que muestre el resumen los 2 ejércitos, indicando el reino, cantidad de unidades, distribución del ejército según las unidades, nivel de vida total del ejército y qué ejército ganó la batalla (usar la métrica de suma de niveles de vida y porcentajes de probabilidad de victoria basado en ella). Este porcentaje también debe mostrarse.

Hacerlo programa iterativo.

la clase main.java

```java
// RONI COMPANOCCA CHECCO
// CUI: 20210558
// LABORATORIO 17
// FUNDAMENTOS DE PROGRAMACION
public class Main {
    public static void main(String[] args){
        int cant;
```

```
        String cultura[] = {"Romanos: ","Francos: ","Sajones: ","Visigodos:
            ","Vandalos: "};
        cant = aleatorio(1,10);
        Ejercito e1 = new Ejercito(cultura[aleatorio(0,4)], cant);
        mostrar(e1);
        cant = aleatorio(1,10);
        Ejercito e2 = new Ejercito(cultura[aleatorio(0,4)], cant);
        mostrar(e2);
        determinarGanador(e1,e2);
    }

    public static int aleatorio(int a, int b){
        return (int)(Math.random()*(b-a+1))+a;
    }

    public static void mostrar(Ejercito e){
        System.out.print(e);
        System.out.println("\nCantidad total de Soldados:
            "+Soldado.cuantos()+"\n"+"Espadachines:
            "+Espadachin.cuantos()+"\n"+"Arqueros:
            "+Arquero.cuantos()+"\n"+"Caballeros: "+Caballero.cuantos()+"\n");
        Ejercito.resetearCantidad();
    }

    public static void determinarGanador(Ejercito e1, Ejercito e2){
        System.out.println("Ejercito 1: "+e1.getCultura()+""+e1.poder());
        System.out.println("Ejercito 2: "+e2.getCultura()+""+e2.poder());
        if (e1.poder()>e2.poder()){
            System.out.println("El ganador es ejercito 1 de : "+e1.getCultura());
        } else if (e1.poder()<e2.poder()){
            System.out.println("El ganador es ejercito 2 de : "+e2.getCultura());
        } else{
            System.out.println("Sin ganador");
        }
    }
}
```

- la clase Soldado.java

```
public class Soldado {
   private String nombre;
    private int nivelAtaque;
    private int nivelDefensa;
    private int nivelVida;
    private int vidaActual;
    private int velocidad = 0;
    private String actitud = "defensa";
    private boolean vive = true;
    private static int num = 0;

    //COMSTRUCTORES
   public Soldado(String nomb, int ataque, int defensa, int vida) {
      nombre = nomb;
       nivelAtaque = ataque;
       nivelDefensa = defensa;
       nivelVida = vida;
```

```
        vidaActual = vida;
        num++;
    }

    // Otros mtodos
     public void atacar() {
        actitud = "ataque";
        avanzar();
     }

     public void defender() {
        actitud = "defensa";
        velocidad = 0;
     }

     public void avanzar() {
        velocidad++;
     }

     public void retroceder() {
        velocidad--;
     }

     public void serAtacado() {
        vidaActual--;
        if (vidaActual == 0)
            morir();
     }

     public void huir(){
        actitud = "fuga";
        velocidad++;
     }

     public void morir() {
        vive = false;
     }

     public String toString() {
        return nombre+ " " +nivelAtaque+ " "+nivelDefensa+ " " +nivelVida+ " "
            +vidaActual+ " " +velocidad+ " " +actitud+ " " +vive;
     }

     public static int cuantos(){
        return num;
     }

     public static void resetearCantidad(){
        num=0;
     }

     public int getVida(){
        return nivelVida;
     }
}
```

- la clase Ejercito.java

```
import java.util.*;
public class Ejercito {
    ArrayList<Soldado> misSoldados = new ArrayList<Soldado>();
    String cultura;

    public Ejercito(String cult, int cantidad){
        cultura = cult;
        int tipo;
        for(int i=0; i<cantidad; i++){
            tipo = (int)(Math.random()*3)+1;
            switch (tipo) {
                case 1: misSoldados.add(new Espadachin("\nE: "+i, 10, 8, 10, 40));
                    break;

                case 2: misSoldados.add(new Caballero("\nC: "+i, 13, 7, 12));
                    break;

                case 3: misSoldados.add(new Arquero("\nA: "+i, 7, 3, 7, 20));
                    break;
            }
        }
    }

    public String toString(){
        String todos ="";
        for(int i=0; i<misSoldados.size(); i++){
            todos += misSoldados.get(i)+"";
        }
        return cultura+" "+misSoldados.size()+" "+todos;
    }

    public int poder(){
        int poder = 0;
        for(int i=0; i<misSoldados.size(); i++){
            poder += misSoldados.get(i).getVida();
        }
        return poder;
    }

    public String getCultura(){
        return cultura;
    }

    public static void resetearCantidad(){
        Soldado.resetearCantidad();
        Arquero.resetearCantidad();
        Caballero.resetearCantidad();
        Espadachin.resetearCantidad();
    }
}
```

- la clase Caballero.java

```
public class Caballero extends Soldado{
```

```
    private String armaActual = "lanza";
    private boolean montando = true;
    private static int num = 0;

    public Caballero(String s, int ata, int def, int vid){
        super(s,ata,def,vid);
        num++;
    }

    public void envestir(){
        if(montando==true){
            for(int i=0; i<=2; i++){
                super.atacar();
            }
        }else{
            super.atacar();
        }
    }

    public void desmontar(){
        if(montando==true){
            montando = false;
            super.defender();
            cambiaArma();
        }
    }

    public void cambiaArma(){
        if(armaActual=="lanza"){
            armaActual = "Espada";
        }else{
            armaActual = "lanza";
        }
    }

    public void montar(){
        if(montando==false){
            montando = true;
            super.atacar();
            cambiaArma();
        }
    }

    public static int cuantos(){
        return num;
    }

    public static void resetearCantidad(){
        num=0;
    }

    public String toString(){
        return super.toString()+" "+armaActual+" "+montando;
    }
}
```

- la clase Espadachin.java

```
public class Espadachin extends Soldado{
    private int longitudEspada;
    private boolean muroEscudos = false;
    private static int num = 0;

    public Espadachin(String s, int ata, int def, int vid, int lon){
        super(s,ata,def,vid);
        longitudEspada = lon;
        num++;
    }

    public void muroEscudos(){
        if(muroEscudos == true ){
            muroEscudos = false;
        }else{
            muroEscudos = true;
        }
    }

    public static int cuantos(){
        return num;
    }

    public static void resetearCantidad(){
        num =0;
    }

    public String toString(){
        return super.toString()+" "+longitudEspada+" "+muroEscudos;
    }
}
```

- la clase Arquero.java

```
public class Arquero extends Soldado {
    private int numeroFlechas;
    private static int num = 0;

    public Arquero(String s, int ata, int def, int vid, int cant){
        super(s,ata,def,vid);
        numeroFlechas = cant;
        num++;
    }

    public void disparar(){
        if(numeroFlechas>0){
            numeroFlechas--;
        }
    }

    public static int cuantos(){
        return num;
    }
```

```
    public static void resetearCantidad(){
        num = 0;
    }

    public String toString(){
        return super.toString()+" "+numeroFlechas;
    }
}
```

- la clase Mapa.java

```
import java.util.Random;

public class Mapa {
    private String tipoTerritorio;
    private Soldado[][] tablero;

    // Constructor
    public Mapa(String tipoTerritorio) {
        this.tipoTerritorio = tipoTerritorio;
        this.tablero = new Soldado[10][10]; // Tamao del tablero (puedes ajustarlo
            segn tus necesidades)
        posicionarSoldadosAleatorios();
    }

    // Mtodo para posicionar soldados aleatorios en el tablero
    private void posicionarSoldadosAleatorios() {
        Random rand = new Random();
        for (int i = 0; i < 10; i++) { // Nmero arbitrario de soldados por ejrcito
            int fila = rand.nextInt(10);
            int columna = rand.nextInt(10);

            // Verificar si la posicin est ocupada
            while (tablero[fila][columna] != null) {
                fila = rand.nextInt(10);
                columna = rand.nextInt(10);
            }

            // Crear un soldado aleatorio (puedes ajustar estos valores segn tus
                necesidades)
            Soldado soldado = new Soldado("Soldado" + i, 8, 5, rand.nextInt(5) + 5);
            tablero[fila][columna] = soldado;
        }
    }

    // Mtodo para mostrar el tablero
    public void mostrarTablero() {
        System.out.println("Mapa - Tipo de Territorio: " + tipoTerritorio);
        for (Soldado[] fila : tablero) {
            for (Soldado soldado : fila) {
                if (soldado != null) {
                    System.out.print(soldado.getNombre() + " ");
                } else {
                    System.out.print("Vaco ");
                }
            }
```

```
            System.out.println();
        }
    }
}
```

- Ejecucion

```
Romanos: 9
E: 0 10 8 10 10 0 defensa true 40 false
C: 1 13 7 12 12 0 defensa true lanza true
E: 2 10 8 10 10 0 defensa true 40 false
A: 3 7 3 7 7 0 defensa true 20
A: 4 7 3 7 7 0 defensa true 20
E: 5 10 8 10 10 0 defensa true 40 false
E: 6 10 8 10 10 0 defensa true 40 false
E: 7 10 8 10 10 0 defensa true 40 false
A: 8 7 3 7 7 0 defensa true 20
Cantidad total de Soldados: 9
Espadachines: 5
Arqueros: 3
Caballeros: 1

Ejercito 1: Sajones: 69
Ejercito 2: Romanos: 83
El ganador es ejercito 2 de : Romanos:
```

6. CUESTIONARIO

- ¿Para que usamos las interfaces?
  Las interfaces se utilizan para facilitar la interacción entre humanos y sistemas, dispositivos o programas. Sirven como un medio de comunicación que permite a las personas enviar y recibir información de manera eficiente y comprensible.
- ¿Qué ventajas tiene el uso de las interfaces?
  Las interfaces permiten a los usuarios interactuar con sistemas, dispositivos o programas de manera más fácil y comprensible. Proporcionan un medio para que las personas se comuniquen con la tecnología de una manera intuitiva.
- ¿Cómo se utiliza el polimorfismo con las interfaces?
  En programación orientada a objetos, el polimorfismo es la capacidad de un objeto de tomar múltiples formas. Se puede implementar a través de interfaces en muchos lenguajes de programación

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.