PROGRAMACIÓN II
Trabajo Práctico 3: Introducción a la Programación Orientada a Objetos

Alumno: Victor Barroeta

DNI: 95805903

Comisión: M2025 – 2

Matricula: 100393

Link de Github:
https://github.com/victormbar/UTN_Prog2_VictorBarroeta/tree/main/javaUTN_TPs/src/TPs_JAVA

OBJETIVO GENERAL

Comprender los fundamentos de la Programación Orientada a Objetos, incluyendo clases, objetos, atributos y métodos, para estructurar programas de manera modular y reutilizable en Java.

MARCO TEÓRICO

Concepto	Aplicación en el proyecto
Clases y Objetos	Modelado de entidades como Estudiante, Mascota, Libro, Gallina y NaveEspacial
Atributos y Métodos	Definición de propiedades y comportamientos para cada clase
Estado e Identidad	Cada objeto conserva su propio estado (edad, calificación, combustible, etc.)
Encapsulamiento	Uso de modificadores de acceso y getters/setters para proteger datos
Modificadores de acceso	Uso de private, public y protected para controlar visibilidad
Getters y Setters	Acceso controlado a atributos privados mediante métodos
Reutilización de código	Definición de clases reutilizables en múltiples contextos

Caso Práctico

Desarrollar en Java los siguientes ejercicios aplicando los conceptos de programación orientada a objetos:

1. **Registro de Estudiantes**
   a. Crear una clase Estudiante con los atributos: nombre, apellido, curso, calificación.

   Métodos requeridos: mostrarInfo(), subirCalificacion(puntos), bajarCalificacion(puntos).

   Tarea: Instanciar a un estudiante, mostrar su información, aumentar y disminuir calificaciones.

```java
public class Estudiante {
    private String nombre;
    private String apellido;
    private String curso;
    private double calificacion;

    // Constructor
    public Estudiante(String nombre, String apellido,String curso,double calificacion){
            this.nombre = nombre;
            this.apellido = apellido;
            this.curso = curso;
            this.calificacion = calificacion;
    }

   // Metodos

    public void mostrarInfo(){
        System.out.println("Nombre: " + this.nombre);
        System.out.println("Apellido: " + this.apellido);
        System.out.println("Curso: " + this.curso);
        System.out.println("Calificacion: " + this.calificacion);
    }

    public void subirCalificación(double puntos){
        this.calificacion = this.calificacion + puntos;
    }

    public void bajarCalificacion(double puntos){
        this.calificacion = this.calificacion - puntos;
```

```java
package TPs_JAVA_TP3;

import TPs_JAVA_TP3.Estudiante;

/**
 *
 * @author vcman
 */
public class TP3Main {

    public static void main(String[] args) {

        Estudiante p1 = new Estudiante("Victor", "Apellido", "Programacion", 8.5);
        p1.mostrarInfo();


        // Metodos

        p1.bajarCalificacion(1);
        p1.subirCalificación(2);



    }

}
```

```
run:
Nombre: Victor
Apellido: Apellido
Curso: Programacion
Calificacion: 8.5
Nombre: Victor
Apellido: Apellido
Curso: Programacion
Calificacion: 7.5
Nombre: Victor
Apellido: Apellido
Curso: Programacion
Calificacion: 10.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Registro de Mascotas
    a. Crear una clase Mascota con los atributos: nombre, especie, edad.

Métodos requeridos: mostrarInfo(), cumplirAnios().

Tarea: Crear una mascota, mostrar su información, simular el paso del tiempo y verificar los cambios.

```java
package TPs_JAVA_TP3;

public class Mascota{

    // Atributos
    private String nombre;
    private String especie;
    private int edad;

    //-----Constructores ----

    // constuctor vacío
    public Mascota(){
    }

    // Constructor sobrecargado
    public Mascota(String nombre, String especie,int edad){
        this.nombre = nombre;
        this.especie= especie;
        this.edad = edad;
    }

    // mostrar la informacion de mascotads
    public void mostrarInfo(){
        System.out.println("Nombre: " + this.nombre);
        System.out.println("Especie: " + this.especie);
        System.out.println("Edad: " + this.edad + " años");
    }
```

```java
    // Simula el cumpleaños de la mascota, incrementando su edad en 1.
    public void cumplirAnios() {
        this.edad ++; //Incrementa la edad en 1
        System.out.println("Ahora tu mascota tiene " + this.nombre + " años.");
    }

    // Getters y Setters

    public String getNombre(){
        return nombre;
    }

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

    public String getEspecie() {
        return especie;
    }

    public void setEspecie(String especie) {
        this.especie = especie;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
```

3. Encapsulamiento con la Clase Libro
    a. Crear una clase Libro con atributos privados: titulo, autor, añoPublicacion.

Métodos requeridos: Getters para todos los atributos. Setter con validación

para añoPublicacion.

Tarea: Crear un libro, intentar modificar el año con un valor inválido y luego con uno válido, mostrar la información final.

```java
public class Libro {

    // Atributos
    private String titulo;
    private String autor;
    private int anioPublicacion;

    // Constructores

    public Libro (){
    }


    public Libro (String titulo, String autor, int anioPublicacion){
         this.titulo = titulo;
         this.autor= autor;
         this.setAnioPublicacion(anioPublicacion);
    }

    // Metodos
        public void mostrarInfo(){
         System.out.println("Título: " + this.titulo);
         System.out.println("Autor: " + this.autor);
         System.out.println("Año: " + this.anioPublicacion);
     }
```

```java
29      // Getters y Setters
30
31      public String getTitulo (){
32          return titulo;
33      }
34
35      public String getAutor() {
36       return autor;
37      }
38
39      public int getAnioPublicacion() {
40          return anioPublicacion;
41      }
42
43      public void setTitulo(String titulo){
44          this.titulo = titulo;
45      }
46
47      public void setAutor(String autor){
48          this.autor= autor;
49      }
50
51       // Setter con validacion
52
53      public void setAnioPublicacion(int anio){
54          if (anio > 0 && anio <= 2025){
55              this.anioPublicacion = anio;
56              System.out.println("Año de publicación actualizado a " + anio);
57          } else{
58              System.out.println("Error: El año '" + anio + "' no es valido. No se realizó el cambi
59          }
```

```
==========================================
=== 3. PRUEBA DE ENCAPSULAMIENTO (CLASE LIBRO) ===
==========================================
T□tulo del libro: Principito
Autor: Antoine de Saint-Exupéry

A□o de publicaci□n (v□lido, ej: 1995): Error: Ingrese un n□mero entero v□lido.
A□o de publicaci□n (v□lido, ej: 1995): 1943
A□o de publicaci□n actualizado a 1943
T□tulo: Principito
Autor: Antoine de Saint-Exup□ry
A□o: 1943

--- Prueba de Setter con Validaci□n ---
Intentando establecer a□o inv□lido: -500
Error: El a□o '-500' no es valido. No se realiz□ el cambio
Ahora ingrese un a□o v□lido (ej: 2020): 1943
A□o de publicaci□n actualizado a 1943

--- Informaci□n Final del Libro ---
T□tulo: Principito
Autor: Antoine de Saint-Exup□ry
A□o: 1943
```

4. Gestión de Gallinas en Granja Digital
    a. Crear una clase Gallina con los atributos: idGallina, edad, huevosPuestos.

Métodos requeridos: **ponerHuevo(), envejecer(), mostrarEstado()**.

Tarea: Crear dos gallinas, simular sus acciones (envejecer y poner huevos), y mostrar su estado.

```java
public class Gallina {
    // --- Atributos ---
    private int idGallina;
    private int edad;
    private int huevosPuestos;

    // --- Constructores ---

    //Constructor vacío.

    public Gallina() {
    }

    //Constructor sobrecargado.

    public Gallina(int idGallina) {
        this.idGallina = idGallina;
        this.edad = 0; // Las gallinas nacen con edad 0
        this.huevosPuestos = 0; // Y 0 huevos puestos
    }

    // --- Métodos Requeridos ---

    //Simula que la gallina pone un huevo, incrementando el contador.
    public void ponerHuevo() {
        this.huevosPuestos++;
        System.out.println("¡Gallina " + this.idGallina + " ha puesto un huevo!");
    }
```

```
    //Simula el paso de un año para la gallina.

    public void envejecer() {
        this.edad++;
        System.out.println("Gallina " + this.idGallina + " ahora tiene " + this.edad + " años.
    }


    //Muestra el estado actual de la gallina.

    public void mostrarEstado() {
        System.out.println("Estado Gallina ID: " + this.idGallina);
        System.out.println("Edad: " + this.edad + " años");
        System.out.println("Huevos Puestos: " + this.huevosPuestos);

    }

    // --- Getters y Setters ---

    public int getIdGallina() {
        return idGallina;
    }

    public void setIdGallina(int idGallina) {
        this.idGallina = idGallina;
    }

    public int getEdad() {
        return edad;
```

```
    // --- Getters y Setters ---

    public int getIdGallina() {
        return idGallina;
    }

    public void setIdGallina(int idGallina) {
        this.idGallina = idGallina;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public int getHuevosPuestos() {
        return huevosPuestos;
    }

    public void setHuevosPuestos(int huevosPuestos) {
        this.huevosPuestos = huevosPuestos;
    }

}
```

```
=== 4. PRUEBA DE GESTI□N DE GALLINAS ===
========================================
Creando Gallina #1 y Gallina #2 (ID 1 y 2)
Estado Gallina ID: 1
Edad: 0 a□os
Huevos Puestos: 0
Estado Gallina ID: 2
Edad: 0 a□os
Huevos Puestos: 0

--- Simulando Acciones ---
Gallina 1 ahora tiene 1 a□os.
□Gallina 1 ha puesto un huevo!
□Gallina 1 ha puesto un huevo!
Gallina 2 ahora tiene 1 a□os.
Gallina 2 ahora tiene 2 a□os.
□Gallina 2 ha puesto un huevo!

--- Reporte Final de la Granja ---
Estado Gallina ID: 1
Edad: 1 a□os
Huevos Puestos: 2
Estado Gallina ID: 2
Edad: 2 a□os
Huevos Puestos: 1

Presione [Enter] para continuar...
```

5. Simulación de Nave Espacial

Crear una clase NaveEspacial con los atributos: nombre, combustible.

Métodos requeridos: despegar(), avanzar(distancia), recargarCombustible(cantidad), mostrarEstado().

Reglas: Validar que haya suficiente combustible antes de avanzar y evitar que se supere el límite al recargar.

Tarea: Crear una nave con 50 unidades de combustible, intentar avanzar sin recargar, luego recargar y avanzar correctamente. Mostrar el estado al final.

```java
package TPs_JAVA_TP3;

public class NaveEspacial {
    // --- Atributos ---
    private String nombre;
    private double combustible;
    // Usamos una constante para el límite de combustible
    private static final double MAX_COMBUSTIBLE = 100.0;
    // Definimos una tasa de consumo (ej: 10 unidades de distancia por 1 de combustible)
    private static final double TASA_CONSUMO = 10.0;

    // --- Constructores ---

    // Constructor vacío.

    public NaveEspacial() {
        this.combustible = 0;
    }

    // Constructor sobrecargado.

    public NaveEspacial(String nombre, double combustibleInicial) {
        this.nombre = nombre;
        // Usamos el método recargarCombustible para la carga inicial,
        // así nos aseguramos de no superar el máximo.
        this.combustible = 0; // Empieza en 0 antes de recargar
        this.recargarCombustible(combustibleInicial);
    }
```

```
// --- Métodos Requeridos ---

public void despegar() {
    System.out.println("Nave " + this.nombre + ": ¡Iniciando secuencia de despegue!");
}

// Intenta avanzar una distancia, validando el combustible.

public void avanzar(double distancia) {
    double combustibleNecesario = distancia / TASA_CONSUMO;

    System.out.println("Intentando avanzar " + distancia + " unidades (requiere " + combus

    if (this.combustible >= combustibleNecesario) {
        this.combustible -= combustibleNecesario;
        System.out.println("¡Avance exitoso! Combustible restante: " + this.combustible);
    } else {
        System.out.println("¡FALLO! Combustible insuficiente. Solo se puede avanzar " + (t
    }
}

//Recarga combustible, validando no superar el máximo.

public void recargarCombustible(double cantidad) {
    if (cantidad <= 0) {
        System.out.println("La cantidad a recargar debe ser positiva.");
        return;
    }
```

```
//Recarga combustible, validando no superar el máximo.

public void recargarCombustible(double cantidad) {
    if (cantidad <= 0) {
        System.out.println("La cantidad a recargar debe ser positiva.");
        return;
    }

    this.combustible += cantidad;

    if (this.combustible > MAX_COMBUSTIBLE) {
        this.combustible = MAX_COMBUSTIBLE;
        System.out.println("Tanque lleno. Combustible al máximo (" + MAX_COMBUSTIBLE + ").
    } else {
        System.out.println(cantidad + " de combustible recargado. Nivel actual: " + this.c
    }
}


//Muestra el estado actual de la nave.

public void mostrarEstado() {
    System.out.println("--- Estado Nave: " + this.nombre + " ---");
    System.out.println("Combustible: " + this.combustible + " / " + MAX_COMBUSTIBLE);

}

// --- Getters y Setters ---
```

```java
    // --- Getters y Setters ---

    public String getNombre() {
        return nombre;
    }

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

    public double getCombustible() {
        return combustible;
    }

}
```

```
=== 5. PRUEBA DE SIMULACI□N NAVE ESPACIAL ===
========================================
Nombre de la nave: Space
Combustible inicial (Max 100): 50
50.0 de combustible recargado. Nivel actual: 50.0
Nave Space: □Iniciando secuencia de despegue!

--- Prueba de Avance (Fallo) ---
Distancia larga a viajar (ej: 800): 100
Intentando avanzar 100.0 unidades (requiere 10.0 de combustible)...
□Avance exitoso! Combustible restante: 40.0
--- Estado Nave: Space ---
Combustible: 40.0 / 100.0

--- Prueba de Recarga (Llenado) ---
Cantidad a recargar (ej: 70): 50
50.0 de combustible recargado. Nivel actual: 90.0

--- Prueba de Avance (□xito) ---
Distancia corta a viajar (ej: 300): 500
Intentando avanzar 500.0 unidades (requiere 50.0 de combustible)...
□Avance exitoso! Combustible restante: 40.0

--- Fin de la Simulaci□n ---
--- Estado Nave: Space ---
Combustible: 40.0 / 100.0
BUILD SUCCESSFUL (total time: 3 minutes 49 seconds)
```

CONCLUSIONES ESPERADAS

- Comprender la diferencia entre clases y objetos.
- Aplicar principios de encapsulamiento para proteger los datos.
- Usar getters y setters para gestionar atributos privados.
- Implementar métodos que definen comportamientos de los objetos.
- Manejar el estado y la identidad de los objetos correctamente.

- Aplicar buenas prácticas en la estructuración del código orientado a objetos.
- Reforzar el pensamiento modular y la reutilización del código en Java.