Programación II – Trabajo Práctico: **Introducción a la POO**

Alumno: *Fermin Alliot*
Carrera: Tecnicatura Universitaria en Programación
Cátedra: Programación II
Comisión: *“3”*
GitHub: *https://github.com/Falliot00/UTN-TUPaD-P2/tree/main/3.%20Introduccion%20a%20POO*

Ejercicio 1 — Registro de Estudiantes

Crear una clase `Estudiante` con atributos `nombre`, `apellido`, `curso`, `calificacion`. Implementar `mostrarInfo()`, `subirCalificacion(puntos)` y `bajarCalificacion(puntos)`. Instanciar y probar.

Diseño

- Atributos **privados** para proteger el estado.
- Rango de calificación: **0 a 10** (validación en los métodos de modificación).
- Método `mostrarInfo()` imprime un resumen legible.

Código

```
public class Estudiante {
    private String nombre;
    private String apellido;
    private String curso;
    private double calificacion; // 0.0 a 10.0

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

    private double ajustarARango(double valor) {
        if (valor < 0) return 0;
        if (valor > 10) return 10;
        return valor;
    }

    public void subirCalificacion(double puntos) {
        if (puntos < 0) return; // no sube con valores negativos
```

```
        this.calificacion = ajustarARango(this.calificacion + puntos);
    }

    public void bajarCalificacion(double puntos) {
        if (puntos < 0) return; // no baja con valores negativos
        this.calificacion = ajustarARango(this.calificacion - puntos);
    }

    public void mostrarInfo() {
        System.out.println("Estudiante: " + nombre + " " + apellido);
        System.out.println("Curso: " + curso);
        System.out.println("Calificación: " + calificacion);
        System.out.println("----------------------------");
    }
}
```

Prueba

```
// En el main
Estudiante e1 = new Estudiante("Ana", "Pérez", "Programación II", 7.5);
e1.mostrarInfo();

e1.subirCalificacion(1.2);   // 7.5 -> 8.7
e1.bajarCalificacion(3.5);   // 8.7 -> 5.2

// Intentos fuera de rango
e1.subirCalificacion(10);    // tope en 10

e1.mostrarInfo();
```

Ejercicio 2 — Registro de Mascotas

Clase `Mascota` con `nombre`, `especie`, `edad`, métodos `mostrarInfo()` y `cumplirAnios()`.

Código

```
public class Mascota {
    private String nombre;
    private String especie;
    private int edad; // en años

    public Mascota(String nombre, String especie, int edad) {
        this.nombre = nombre;
        this.especie = especie;
        this.edad = Math.max(0, edad);
    }

    public void cumplirAnios() {
```

```
        this.edad += 1;
    }

    public void mostrarInfo() {
        System.out.println("Mascota: " + nombre + " (" + especie + ")");
        System.out.println("Edad: " + edad + " años");
        System.out.println("--------------------------");
    }
}
```

Prueba

```
// En el main
Mascota m1 = new Mascota("Luna", "Perro", 2);
m1.mostrarInfo();
m1.cumplirAnios();
m1.cumplirAnios();
m1.mostrarInfo();
```

Ejercicio 3 — Encapsulamiento con la clase Libro

Clase `Libro` con atributos **privados** `titulo`, `autor`, `anioPublicacion`. Proveer **getters** y un **setter con validación** para `anioPublicacion`.

Diseño y validación

- `anioPublicacion` debe ser razonable (por ejemplo, **>= 1400** y **<= año actual**).
- Si el valor es inválido, se ignora el cambio o se informa el error.

Código

```
import java.time.Year;

public class Libro {
    private String titulo;
    private String autor;
    private int anioPublicacion;

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

    public String getTitulo() { return titulo; }
    public String getAutor() { return autor; }
    public int getAnioPublicacion() { return anioPublicacion; }
```

```
    public void setAnioPublicacion(int anio) {
        int anioActual = Year.now().getValue();
        if (anio >= 1400 && anio <= anioActual) {
            this.anioPublicacion = anio;
        } else {
            System.out.println("Año inválido: " + anio + ". Se mantiene:
" + this.anioPublicacion);
        }
    }

    public String info() {
        return String.format("\"%s\" de %s (%d)", titulo, autor,
anioPublicacion);
    }
}
```

Prueba

```
// En el main
Libro l1 = new Libro("El Quijote", "Miguel de Cervantes", 1605);
System.out.println(l1.info());

l1.setAnioPublicacion(1200); // inválido, se rechaza
l1.setAnioPublicacion(2005); // válido
System.out.println(l1.info());
```

Ejercicio 4 — Gestión de Gallinas en Granja Digital

Clase `Gallina` con `idGallina`, `edad`, `huevosPuestos`, métodos `ponerHuevo()`, `envejecer()`, `mostrarEstado()`. Crear **dos** gallinas, simular acciones y mostrar estado.

Código

```
public class Gallina {
    private int idGallina;
    private int edad;          // en meses
    private int huevosPuestos; // acumulado

    public Gallina(int idGallina, int edadInicial) {
        this.idGallina = idGallina;
        this.edad = Math.max(0, edadInicial);
        this.huevosPuestos = 0;
    }

    public void ponerHuevo() {
        this.huevosPuestos += 1;
    }
```

```
    public void envejecer() {
        this.edad += 1;
    }

    public void mostrarEstado() {
        System.out.printf("Gallina #%d | Edad: %d meses | Huevos: %d\n",
                idGallina, edad, huevosPuestos);
    }
}
```

Prueba

```
// En el main
Gallina g1 = new Gallina(1, 10);
Gallina g2 = new Gallina(2, 8);

// Simulación simple
g1.envejecer();
g1.ponerHuevo();
g1.ponerHuevo();

g2.envejecer();
g2.ponerHuevo();

g1.mostrarEstado();
g2.mostrarEstado();
```

Ejercicio 5 — Simulación de Nave Espacial

Clase `NaveEspacial` con `nombre`, `combustible`, métodos `despegar()`, `avanzar(distancia)`, `recargarCombustible(cantidad)`, `mostrarEstado()`. Validar **combustible suficiente para avanzar** y **no exceder el límite al recargar**. Probar con una nave que inicia con 50 unidades de combustible.

Diseño

- Agrego `capacidadMaxima` para poder validar la recarga.
- Política de consumo: **1 unidad de combustible por unidad de distancia**.

Código

```
public class NaveEspacial {
    private String nombre;
    private double combustible;      // unidades actuales
    private final double capacidadMaxima; // tope

    public NaveEspacial(String nombre, double combustibleInicial, double
capacidadMaxima) {
        this.nombre = nombre;
```

```
        this.capacidadMaxima = Math.max(0, capacidadMaxima);
        this.combustible = Math.min(Math.max(0, combustibleInicial),
this.capacidadMaxima);
    }

    public void despegar() {
        if (combustible > 0) {
            System.out.println(nombre + " ha despegado.");
        } else {
            System.out.println("No hay combustible para despegar.");
        }
    }

    public void avanzar(double distancia) {
        if (distancia <= 0) {
            System.out.println("La distancia debe ser positiva.");
            return;
        }
        if (combustible >= distancia) {
            combustible -= distancia;
            System.out.println(nombre + " avanzó " + distancia + "
unidades.");
        } else {
            System.out.println("Combustible insuficiente para avanzar " +
distancia + " unidades.");
        }
    }

    public void recargarCombustible(double cantidad) {
        if (cantidad <= 0) {
            System.out.println("La cantidad debe ser positiva.");
            return;
        }
        if (combustible + cantidad <= capacidadMaxima) {
            combustible += cantidad;
            System.out.println("Recarga exitosa. Combustible actual: " +
combustible);
        } else {
            System.out.println("No se puede superar la capacidad máxima
(" + capacidadMaxima + ").");
        }
    }

    public void mostrarEstado() {
        System.out.printf("Nave: %s | Combustible: %.2f / %.2f\n",
nombre, combustible, capacidadMaxima);
    }
}
```

Prueba

```
// En el main
NaveEspacial nave = new NaveEspacial("Aurora", 50, 100);
nave.mostrarEstado();

nave.despegar();
nave.avanzar(60);      // falla por falta de combustible
nave.recargarCombustible(20); // 50 -> 70
nave.avanzar(60);      // ahora sí
nave.mostrarEstado();
```
