[UTN-TUPaD-P2/módulo 3/TP3_POO at main · jazmin-herrera/UTN-TUPaD-P2]

Tp3: POO

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_1 {
    private final String nombre;
    private final String apellido;
    private final String curso;
    private double calificacion;

    public Estudiante_1(String nombre, String apellido, String curso, double calificacion) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.curso = curso;
        this.calificacion = calificacion;
    }

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

    public void subirCalificacion(double puntos) {
        calificacion += puntos;
    }

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

Main:

```java
class MainEstudiante {
    public static void main(String[] args) {
        Estudiante_1 estudiante = new Estudiante_1("Juan", "Perez", "Comision 1", 8.0);

        estudiante.mostrarInformacion();     // Muestra 8.0
        estudiante.subirCalificacion(2.0);   // Sube a 10.0
        estudiante.mostrarInformacion();     // Muestra 10.0
        estudiante.bajarCalificacion(1.0);   // Baja a 9.0
        estudiante.mostrarInformacion();     // Muestra 9.0
    }
}
```

Salida por pantalla:

```
run:
Nombre: Juan
Apellido: Perez
Curso: Comision 1
Calificacion: 8.0

Nombre: Juan
Apellido: Perez
Curso: Comision 1
Calificacion: 10.0

Nombre: Juan
Apellido: Perez
Curso: Comision 1
Calificacion: 9.0

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
public class Mascota_2 {
    private final String nombre;
    private final String especie;
    private int edad;

    // Constructor
    public Mascota_2(String nombre, String especie, int edad) {
        this.nombre = nombre;
        this.especie = especie;
        this.edad = edad;
    }

    // Métodos
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + " | Especie: " + especie + " | Edad: " + edad);
    }

    public void envejecer(int años) {
        this.edad += años;
    }

    // Main
    public static void main(String[] args) {
        Mascota_2 m1 = new Mascota_2("Firulais", "Perro", 5);
        m1.mostrarInformacion();

        m1.envejecer(2);
        m1.mostrarInformacion();
    }
}
```

Salida por pantalla:

```
run:
Nombre: Firulais | Especie: Perro | Edad: 5
Nombre: Firulais | Especie: Perro | Edad: 7
BUILD SUCCESSFUL (total time: 0 seconds)
```

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 final class Libro_3 {
    private final String titulo;
    private final String autor;
    private int anioPublicacion;

    // Constructor
    public Libro_3(String titulo, String autor, int anioPublicacion) {
        this.titulo = titulo;
        this.autor = autor;
        this.anioPublicacion = anioPublicacion;
    }

    // Getter y Setter
    public int getAnioPublicacion() {
        return anioPublicacion;
    }

    public void setAnioPublicacion(int anioPublicacion) {
        this.anioPublicacion = anioPublicacion;
    }

    // Método
    public void mostrarInformacion() {
        System.out.println("Titulo: " + titulo);
        System.out.println("Autor: " + autor);
        System.out.println("Año de publicacion: " + anioPublicacion);
    }

    // Main
    public static void main(String[] args) {
        Libro_3 libro1 = new Libro_3("Cien años de soledad", "Gabriel García Márquez", 0);
        libro1.mostrarInformacion();

        libro1.setAnioPublicacion(1967);
        libro1.mostrarInformacion();
    }
```

Salida por pantalla:

```
run:
Titulo: Cien a�os de soledad
Autor: Gabriel Garc�a M�rquez
A�o de publicacion: 0
Titulo: Cien a�os de soledad
Autor: Gabriel Garc�a M�rquez
A�o de publicacion: 1967
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.

parte1:

```java
public class Gallina {
    private String idGallina;
    private int edad;
    private int huevosPuestos;

    // Getters y setters
    public String getIdGallina() {
        return idGallina;
    }

    public void setIdGallina(String 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;
    }
```

parte2:

```java
    // Métodos
    public void ponerHuevo() {
        huevosPuestos++;
    }

    public void envejecer() {
        edad++;
    }

    public void mostrarEstado() {
        System.out.println("ID: " + idGallina + " | Edad: " + edad + " | Huevos puestos: " + huevosPuestos);
    }
}
```

main:

```java
public static void main(String[] args) {
    Gallina g = new Gallina();
    g.setIdGallina("abc123");
    g.setEdad(1);
    g.setHuevosPuestos(0);

    Gallina g2 = new Gallina();
    g2.setIdGallina("abc124");
    g2.setEdad(2);
    g2.setHuevosPuestos(5);

    // Estado inicial
    g.mostrarEstado();
    g2.mostrarEstado();

    // Ponen huevos
    g.ponerHuevo();
    g.ponerHuevo();
    g.ponerHuevo();

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

    // Envejecen
    g.envejecer();
    g2.envejecer();

    // Estado final
    g.mostrarEstado();
    g2.mostrarEstado();
}
```

Salida por pantalla:

```
run:
ID: abc123, huevos puestos: 0, edad: 1
ID: abc124, huevos puestos: 5, edad: 2
ID: abc123, huevos puestos: 3, edad: 2
ID: abc124, huevos puestos: 8, edad: 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Simulación de Nave Espacial

a. 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.

Parte 1:

```
class NaveEspacial {
    private String nombre;
    private int combustible;

    // Constructor
    public NaveEspacial(String nombre, int combustible) {
        this.nombre = nombre;
        this.combustible = combustible;
    }

    // Método para despegar
    public void despegar() {
        if (combustible >= 5) {
            combustible -= 5;
            System.out.println("La nave despego...");
        } else {
            System.out.println("No hay suficiente combustible para despega
        }
    }

    // Método para avanzar cierta cantidad de kilómetros
    public void avanzar(int km) {
        System.out.println("Intentando avanzar " + km + " km:");
        int consumo = km / 1; // 1 unidad de combustible por kilómetro

        if (combustible >= consumo) {
            combustible -= consumo;
            System.out.println("La nave avanzo " + km + " kms");
        } else {
            System.out.println("No hay suficiente combustible");
        }
```

Parte 2

```
    }

    // Método para recargar combustible
    public void recargarCombustible(int cantidad) {
        combustible += cantidad;
        System.out.println("Recargando " + cantidad + " unidades de combustible:");
    }

    // Método para mostrar el estado actual
    public void mostrarEstado() {
        System.out.println("Nombre: " + nombre + ", combustible: " + combustible);
    }
}
```

Main:

```
public class main_nave {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {


        // Crear la nave con 50 unidades de combustible
        NaveEspacial nave = new NaveEspacial("Exploradora 1", 50);

        // Estado inicial
        System.out.println("Estado inicial:");
        nave.mostrarEstado();

        // Despegar
        nave.despegar();
        nave.mostrarEstado();

        // Intentar avanzar 30 km sin suficiente combustible
        nave.avanzar(30);

        // Recargar combustible
        nave.recargarCombustible(40);

        // Avanzar 20 km
        nave.avanzar(20);

        // Estado final
        System.out.println("Estado final:");
        nave.mostrarEstado();
    }
}
```

Salida por pantalla:

```
run:
Estado inicial:
Nombre: Exploradora 1, combustible: 50
La nave despego...
Nombre: Exploradora 1, combustible: 45
Intentando avanzar 30 km:
La nave avanzo 30 kms
Recargando 40 unidades de combustible:
Intentando avanzar 20 km:
La nave avanzo 20 kms
Estado final:
Nombre: Exploradora 1, combustible: 35
BUILD SUCCESSFUL (total time: 0 seconds)
```