

PROGRAMACIÓN II

Trabajo Práctico 3: Introducción a la Programación Orientada a Objetos

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:

CONCLUSIONES ESPERADAS

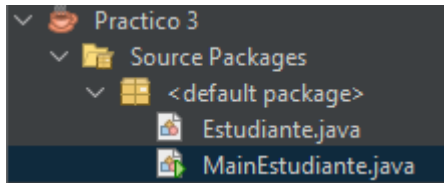
- Comprender la diferencia entre clases y objetos.
- Aplicar principios de encapsulamiento para proteger los datos.
- Usar getters y setters para gestionar atributos privados.
- Implementar métodos que definen comportamientos de los objetos.
- Manejar el estado y la identidad de los objetos correctamente.
- Aplicar buenas prácticas en la estructuración del código orientado a objetos.
- Reforzar el pensamiento modular y la reutilización del código en Java.

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.



> **class Estudiante**

```

*
* @author veron
*/
public class Estudiante {
    // Atributos
    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;
    }

    // Método para mostrar información
    public void mostrarInfo() {
        System.out.println("Nombre: " + nombre);
        System.out.println("Apellido: " + apellido);
        System.out.println("Curso: " + curso);
        System.out.println("Calificacion: " + calificacion);
    }

    // Método para aumentar la calificación
    public void subirCalificacion(double puntos) {
        this.calificacion += puntos;
        System.out.println("Calificacion aumentada a: " + calificacion);

        // Método para disminuir la calificación
        public void bajarCalificacion(double puntos) {
            this.calificacion -= puntos;
            System.out.println("Calificacion disminuida a: " + calificacion);
        }
    }
}

```

>Main

```

* @author veron
*/
public class MainEstudiante {
    public static void main(String[] args) {
        // Instanciar un estudiante
        Estudiante estudiantel = new Estudiante("Vero", "Guirin", "Fisica", 95.0);

        // Mostrar su información
        estudiantel.mostrarInfo();

        // Aumentar calificación
        estudiantel.subirCalificacion(5.0);

        // Disminuir calificación
        estudiantel.bajarCalificacion(3.0);

        // Mostrar la información nuevamente
        estudiantel.mostrarInfo();
    }
}

```

>SALIDA

```

Output - Practico 3 (run) x
run:
Nombre: Vero
Apellido: Guirin
Curso: Fisica
Calificacion: 95.0
Calificacion aumentada a: 100.0
Calificacion disminuida a: 97.0
Nombre: Vero
Apellido: Guirin
Curso: Fisica
Calificacion: 97.0
BUILD SUCCESSFUL (total time: 0 seconds)

```

RESUMEN del Proyecto:

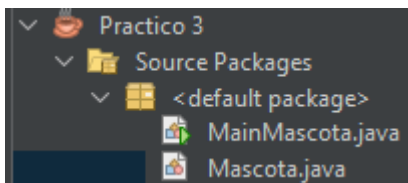
- **Clase Estudiante:** Contiene los atributos nombre, apellido, curso y calificación, y tiene métodos para mostrar la información del estudiante, y métodos para subir y bajar la calificación.
- **Clase MainEstudiante:** Se utiliza para instanciar un objeto de la clase Estudiante, mostrar su información y modificar su calificación.

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.



```
* @author veron
*/
public class Mascota {
    // Atributos
    private String nombre;
    private String especie;
    private int edad;

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

    // Método para mostrar información
    public void mostrarInfo() {
        System.out.println("Nombre: " + nombre);
        System.out.println("Especie: " + especie);
        System.out.println("Edad: " + edad + " años");
    }

    // Método para cumplir años
    public void cumplirAnios() {
        this.edad++;
        System.out.println(nombre + " ha cumplido un año más. Nueva edad: " + edad + " años.");
    }
}
```

```
* @author veron
*/
public class MainMascota {
    public static void main(String[] args) {
        // Crear una mascota
        Mascota mascotal = new Mascota("Jacky", "Perro", 5);

        // Mostrar su información
        mascotal.mostrarInfo();

        // Simular el paso del tiempo
        System.out.println("Simulando el paso del tiempo...");
        for (int i = 0; i < 3; i++) { // Simular 3 años
            mascotal.cumplirAños();
        }

        // Mostrar la información nuevamente
        mascotal.mostrarInfo();
    }
}
```

Output - Practico 3 (run) ×

```
run:
Nombre: Jacky
Especie: Perro
Edad: 5 años
Simulando el paso del tiempo...
Jacky ha cumplido un año mas. Nueva edad: 6 años.
Jacky ha cumplido un año mas. Nueva edad: 7 años.
Jacky ha cumplido un año mas. Nueva edad: 8 años.
Nombre: Jacky
Especie: Perro
Edad: 8 años
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.

```

* @author veron
*/
public class Libro {
    // Atributos privados
    private String titulo;
    private String autor;
    private int añoPublicacion;

    // Constructor
    public Libro(String titulo, String autor, int añoPublicacion) {
        this.titulo = titulo;
        this.autor = autor;
        setAñoPublicacion(añoPublicacion); // Usar el setter para la validación
    }

    // Getters
    public String getTitulo() {
        return titulo;
    }

    public String getAutor() {
        return autor;
    }

    public int getAñoPublicacion() {
        return añoPublicacion;
    }

    // Setter con validación
    public void setAñoPublicacion(int añoPublicacion) {
        if (añoPublicacion < 1900 || añoPublicacion > 2023) {
            System.out.println("Año de publicacion invalido. Debe estar entre 1900 y 2023.");
        } else {
            this.añoPublicacion = añoPublicacion;
            System.out.println("Año de publicacion actualizado a: " + añoPublicacion);
        }
    }
}

```

```
* @author veron
*/
public class MainLibro {
    public static void main(String[] args) {
        // Crear un libro
        Libro libro = new Libro("Cronicas del Angel Gris", "Alejandro Dolina", 1987);

        // Intentar modificar el año con un valor inválido
        libro.setAñoPublicacion(1400); // Este año debería ser inválido

        // Intentar modificar el año con un valor válido
        libro.setAñoPublicacion(2020); // Este año es válido

        // Mostrar la información final
        System.out.println("Titulo: " + libro.getTitulo());
        System.out.println("Autor: " + libro.getAutor());
        System.out.println("Año de publicacion: " + libro.getAñoPublicacion());
    }
}
```

Output - Practico 3 (run) X



```
run:
Año de publicacion actualizado a: 1987
Año de publicacion invalido. Debe estar entre 1900 y 2023.
Año de publicacion actualizado a: 2020
Titulo: Cronicas del Angel Gris
Autor: Alejandro Dolina
Año de publicacion: 2020
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.

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

    // Constructor
    public Gallina(int idGallina) {
        this.idGallina = idGallina;
        this.edad = 0; // La gallina comienza con 0 años
        this.huevosPuestos = 0; // Inicialmente no ha puesto huevos
    }

    // Método para poner un huevo
    public void ponerHuevo() {
        huevosPuestos++;
        System.out.println("La gallina " + idGallina + " ha puesto un huevo.");
    }

    // Método para envejecer la gallina
    public void envejecer() {
        edad++;
        System.out.println("La gallina " + idGallina + " ha envejecido un año.");
    }

    // Método para mostrar el estado de la gallina
    public void mostrarEstado() {
        System.out.println("Gallina ID: " + idGallina);
        System.out.println("Edad: " + edad + " años");
        System.out.println("Huevos puestos: " + huevosPuestos);
    }
}
```



```
public class MainGallina {  
    public static void main(String[] args) {  
        // Crear dos gallinas  
        Gallina gallina1 = new Gallina(1);  
        Gallina gallina2 = new Gallina(2);  
  
        // Simular acciones de las gallinas  
        gallina1.ponerHuevo();  
        gallina1.envejecer();  
        gallina2.ponerHuevo();  
        gallina2.envejecer();  
        gallina2.envejecer(); // Envejece otra vez  
  
        // Mostrar el estado de las gallinas  
        System.out.println("\nEstado de las gallinas:");  
        gallina1.mostrarEstado();  
        gallina2.mostrarEstado();  
    }  
}
```

Output - Practico 3 (run) ×

run:
La gallina 1 ha puesto un huevo.
La gallina 1 ha envejecido un anio.
La gallina 2 ha puesto un huevo.
La gallina 2 ha envejecido un anio.
La gallina 2 ha envejecido un anio.

Estado de las gallinas:
Gallina ID: 1
Edad: 1 anios
Huevos puestos: 1
Gallina ID: 2
Edad: 2 anios
Huevos puestos: 1
BUILD SUCCESSFUL (total time: 0 seconds)

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.

```
* @author veron
*/
public class NaveEspacial {
    // Atributos
    private String nombre;
    private int combustible;
    private int combustibleMaximo;

    // Constructor
    public NaveEspacial(String nombre, int combustible) {
        this.nombre = nombre;
        this.combustible = combustible;
        this.combustibleMaximo = 100; // Limite máximo de combustible
    }

    // Método para despegar
    public void despegar() {
        if (combustible > 0) {
            System.out.println(nombre + " ha despegado.");
        } else {
            System.out.println(nombre + " no puede despegar, no hay combustible.");
        }
    }
}
```

```

// Método para avanzar una distancia
public void avanzar(int distancia) {
    int consumo = distancia / 10; // 1 unidad de combustible por cada 10 de distancia
    if (combustible >= consumo) {
        combustible -= consumo;
        System.out.println(nombre + " ha avanzado " + distancia + " km.");
    } else {
        System.out.println(nombre + " no puede avanzar, combustible insuficiente.");
    }
}

// Método para recargar combustible
public void recargarCombustible(int cantidad) {
    if (combustible + cantidad > combustibleMaximo) {
        System.out.println("No se puede recargar, el combustible excede el límite máximo.");
    } else {
        combustible += cantidad;
        System.out.println("Recargado " + cantidad + " unidades de combustible. Total: " + combustible);
    }
}

// Método para mostrar estado
public void mostrarEstado() {
    System.out.println("Estado de la nave " + nombre + ":");
    System.out.println("Combustible: " + combustible);
}

```

```

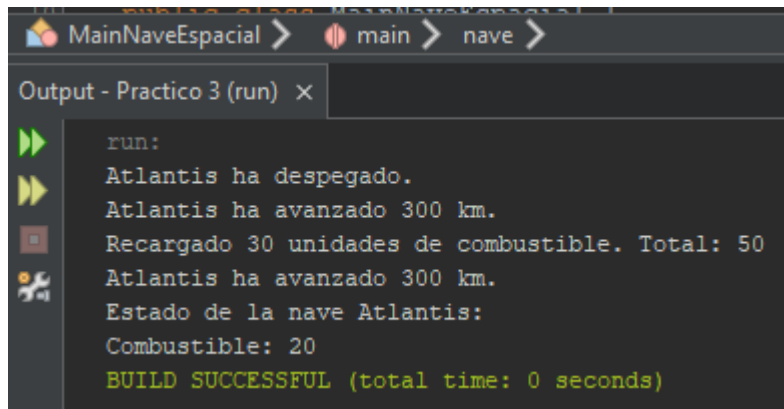
* @author veron
*/
public class MainNaveEspacial {
    public static void main(String[] args) {
        // Crear una nave con 50 unidades de combustible
        NaveEspacial nave = new NaveEspacial("Atlantis", 50);

        // Intentar avanzar sin recargar
        nave.despegar();
        nave.avanzar(300); // Debe fallar, no hay suficiente combustible

        // Recaudar combustible y avanzar correctamente
        nave.recargarCombustible(30); // Recargar combustible
        nave.avanzar(300); // Avanzar nuevamente

        // Mostrar el estado final
        nave.mostrarEstado();
    }
}

```



```
run:
Atlantis ha despegado.
Atlantis ha avanzado 300 km.
Recargado 30 unidades de combustible. Total: 50
Atlantis ha avanzado 300 km.
Estado de la nave Atlantis:
Combustible: 20
BUILD SUCCESSFUL (total time: 0 seconds)
```