

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

Introducción

El presente trabajo práctico tiene como objetivo aplicar los conceptos fundamentales de la Programación Orientada a Objetos (POO) utilizando el lenguaje Java. En POO unimos atributos y métodos que trabajan sobre esos datos en una única entidad llamada objeto, lo que permite organizar el código de forma clara, modular y reutilizable.

Una clase es como un molde o plantilla que define cómo son los objetos, mientras que un objeto es un ejemplar de esa clase con sus propios datos. El objeto es una entidad que dispone de estado, comportamiento e identidad, y sus atributos son el conjunto de propiedades que lo caracterizan.

La abstracción es un principio que permite simplificar la complejidad de un sistema mostrando solo lo esencial y ocultando los detalles internos o innecesarios. En POO, esto significa enfocarse en qué hace un objeto, sin preocuparse por cómo lo hace.

En este TP se desarrollan diferentes ejercicios que permiten poner en práctica la creación de clases, la instanciación de objetos, la implementación de métodos y el manejo del estado de los objetos. A través de estas actividades, se busca reforzar el pensamiento modular y la aplicación de buenas prácticas en la estructuración del código orientado a objetos en Java.

Caso Práctico

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

1. Registro de Estudiantes

- 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.

En este ejercicio creamos un objeto llamado “**estudiante1**” a partir de la clase **Estudiante**, con atributos: **nombre, apellido, curso y calificación**. Primero mostramos toda su información en pantalla usando el método “`mostrarInfo()`”, que utiliza “`System.out.println()`” para imprimir cada dato. Luego aplicamos los métodos “`subirCalificacion()`” y “`bajarCalificacion()`” para modificar la nota y vemos cómo cambia con cada acción. Para seguir la evolución de la calificación sin repetir todos los demás datos, usamos el método “`mostrarCalificacion()`”, que imprime únicamente ese valor.

Opté por mantener la clase Estudiante y el main en el mismo archivo, separando claramente los métodos de la clase del código del main. Para acceder y modificar los atributos, utilicé **getters** y **setters**, y la palabra clave “`this`” para referirme a los atributos del objeto de manera explícita. Esto permite un código más ordenado y fácil de seguir, ideal para un ejercicio de práctica.

De esta manera, se puede observar de forma clara cómo un objeto de la clase Estudiante guarda su información y cómo podemos interactuar con ella dentro del programa, aplicando conceptos fundamentales de la Programación Orientada a Objetos como encapsulamiento, métodos y constructores.

Prueba 1: Aquí se demuestra como el aumento en 1 dio como resultado 9 partiendo de una nota que era 8 y luego su disminución en 2 quedando como nota final un: 7.

```
Output - Tp3Programacion (run) X
run:
Nombre: Sandra
Apellido: Martinez
Curso: Programación II
Calificación: 8.0
Calificación: 9.0
Calificación: 7.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Registro de Mascotas

- 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.

En este ejercicio creamos un objeto llamado "mascota1" a partir de la clase Mascota, con atributos: nombre, especie y edad. Primero mostramos toda su información en pantalla usando el método "mostrarInfo()", que utiliza "System.out.println()" para imprimir cada dato. Luego aplicamos el método "cumplirAnios()" para simular el paso del tiempo y observamos cómo cambia la edad de la mascota.

Al igual que el ejercicio anterior, mantuve la clase Mascota y el main en el mismo archivo, separando claramente los métodos de la clase del código del main. Para acceder y modificar los atributos, utilicé getters y setters, y la palabra clave "this" para referirme a los atributos del objeto de manera explícita. Esto permite un código más ordenado y fácil de seguir, ideal para un ejercicio de práctica.

Prueba 1: Aquí se demuestra cómo al usar la función "cumplirAnios()", que incrementa la edad en 1, Mike pasó de 3 años a tener 4, mostrando claramente el efecto del método sobre el atributo edad

```
Output - Tp3Programacion (run) X
run:
Nombre: Mike
Especie: Perro
Edad: 3
Después de cumplir años:
Nombre: Mike
Especie: Perro
Edad: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Encapsulamiento con la Clase Libro

- 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.

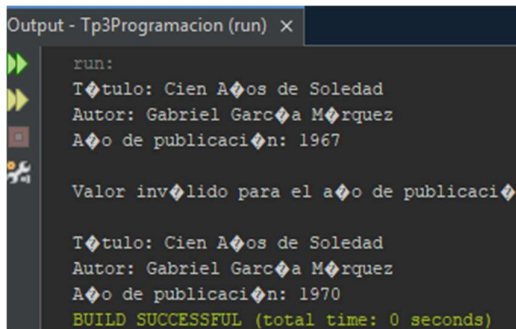
Creamos un objeto llamado "libro1" a partir de la clase Libro, con atributos: titulo, autor y añoPublicacion. Mostramos toda su información en pantalla mediante el método "mostrarInfo()", que utiliza "System.out.println()" para imprimir cada dato. Luego probamos modificar el año de

publicación usando el setter, primero con un valor inválido y luego con uno válido, observando cómo solo se acepta la actualización correcta.

NetBeans sugirió declarar los atributos título y autor como **final**, porque estos valores no se modifican después de asignarse en el constructor. Esto evita cambios accidentales, hace el código más seguro y claro, y es una buena práctica al trabajar con objetos que deberían mantener ciertos campos inmutables.

Para manipular el atributo que puede cambiar, “anioPublicacion”, usamos un setter con validación. Los **getters** y la palabra clave “this” permiten acceder y referirse a los atributos del objeto de manera explícita.

Prueba 1: Aquí se demuestra cómo, al usar el setter con validación “setAnioPublicacion()”, un intento de asignar un valor inválido no cambia el atributo, mientras que al asignar un valor válido, el año de publicación se actualiza correctamente de 1967 a 1970, mostrando claramente el efecto del método sobre el atributo “anioPublicacion”.



```
Output - Tp3Programacion (run) X
run:
Título: Cien Años de Soledad
Autor: Gabriel García Márquez
Año de publicación: 1967

Valor inválido para el año de publicación

Título: Cien Años de Soledad
Autor: Gabriel García Márquez
Año de publicación: 1970
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Gestión de Gallinas en Granja Digital

- 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.

Creamos dos objetos llamados “gallina1” y “gallina2” a partir de la clase Gallina, con los atributos: idGallina, edad y huevosPuestos. Para mostrar sus datos, usamos el método “mostrarEstado()”, que imprime cada atributo en consola mediante “System.out.println()”.

Luego simulamos sus acciones: el método “envejecer()” incrementa la edad de la gallina en 1 mientras que el método “ponerHuevo()” aumenta el contador “huevosPuestos” en 1 cada vez que se llama.

Los métodos **getters**, como “getIdGallina()”, “getEdad()” y “getHuevosPuestos()”, permiten acceder de forma segura a los atributos, y la palabra clave “this” se usa dentro de los métodos para referirse explícitamente a los atributos del objeto.

Prueba 1: Llamamos a “mostrarEstado()” para ver los valores iniciales de las gallinas: la edad y la cantidad de huevos puestos. Luego aplicamos “envejecer()” a “gallina1” y usamos “ponerHuevo()” en ambas gallinas varias veces. Al volver a llamar a “mostrarEstado()”, podemos ver claramente cómo la

edad de “gallina1” aumentó en 1 y cómo los huevos puestos se incrementaron según las acciones realizadas.

```
Output - Tp3Programacion (run) x
run:
Gallina ID: 1
Edad: 3
Huevos puestos: 6

Gallina ID: 2
Edad: 3
Huevos puestos: 10
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.

Creamos un objeto llamado “nave1” a partir de la clase **NaveEspacial**, con los atributos: nombre y combustible. Para mostrar sus datos, usamos el método “mostrarEstado()”, que imprime en pantalla el nombre de la nave y la cantidad de combustible disponible.

Luego simulamos sus acciones:

- Con el método “despegar()” la nave intenta iniciar su vuelo, gastando 10 unidades de combustible si tiene suficiente.
- El método “avanzar(distancia)” permite mover la nave cierta distancia, reduciendo el combustible de acuerdo a la distancia recorrida y mostrando un mensaje si no hay suficiente combustible.
- Para recuperar combustible, usamos “recargarCombustible(cantidad)”, que aumenta el nivel de combustible sin superar el máximo permitido.

Los métodos getters, “getNombre()” y “getCombustible()”, permiten acceder de forma segura a los atributos, mientras que la palabra clave “this” se utiliza dentro de la clase para referirse explícitamente a los atributos del objeto.

Prueba 1: Primero llamamos a “mostrarEstado()” y vemos que la nave tiene 50 unidades de combustible. Intentamos avanzar 60 unidades con “avanzar(60)”, pero la nave no tiene suficiente combustible, por lo que el método imprime un aviso y el combustible permanece igual. Luego

recargamos 30 unidades con “recargarCombustible(30)”, llevando el combustible a 80. Al volver a avanzar 60 unidades, la nave puede moverse correctamente y el combustible se reduce a 20. Finalmente, al llamar nuevamente a “mostrarEstado()”, vemos claramente cómo los métodos modificaron los atributos de manera controlada y respetando las reglas de validación.

```
Output - Tp3Programacion (run) X
run:
Nave: Houston
Combustible inicial: 50.0
Houston no cumple con el avance de 60.0 unidades de combustible.
Houston recargó combustible. Nivel actualizado: 80.0
Houston avanzó 60.0 unidades. Combustible restante: 20.0
Nave: Houston
Combustible inicial: 20.0
BUILD SUCCESSFUL (total time: 0 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.

Conclusión:

A lo largo de este trabajo práctico, aplicamos conceptos fundamentales de la Programación Orientada a Objetos: creación de **clases**, **atributos**, **métodos**, **constructores**, **encapsulamiento** y uso de **getters** y **setters**.

Cada ejercicio permitió ver cómo los objetos almacenan información y cómo se logra interactuar con ellos mediante métodos, respetando reglas y validaciones. También se visualizaron las buenas prácticas, como el uso de **this** y la definición de atributos inmutables con **final**, lo que hace que el código sea más seguro, claro y fácil de mantener.

En conjunto, este TP nos ayudó a consolidar nuestra comprensión de la **POO** y a desarrollar un pensamiento lógico para modelar y simular distintos escenarios mediante objetos.

ENLACE GITHUB:

<https://github.com/Sandra86Martinez/Tp3Programacion/tree/master>