



Universidad Tecnológica Nacional


Unidad 3: TP 3 - Introducción a la Programación Orientada a Objetos

Alumno: Calcatelli Renzo - rcalcatelli@gmail.com

Comisión: M2025-1

Materia: Programación II

Profesor: Ariel Enferrel

 [Enlace](#) al repositorio de GitHub

Objetivo General

El propósito de este trabajo es comprender los fundamentos de la Programación Orientada a Objetos, incluyendo conceptos clave como **clases**, **objetos**, **atributos** y **métodos**. El objetivo final es aplicar estos principios para estructurar programas de manera modular y reutilizable en Java.

Marco Teórico

La Programación Orientada a Objetos (POO) es un paradigma de programación que nos permite modelar el mundo real de manera más intuitiva. Los siguientes conceptos son fundamentales para este trabajo:

- **Clases y Objetos:** Las **clases** son plantillas que definen las características (atributos) y los comportamientos (métodos) de una entidad. Un **objeto** es una instancia de una clase, es decir, una entidad concreta creada a partir de esa plantilla. En este trabajo, modelamos entidades como **Estudiante**, **Mascota**, **Libro**, **Gallina** y **NaveEspacial**.
 - **Atributos y Métodos:** Los **atributos** son las propiedades de un objeto, como la **edad** o la **calificación**. Los **métodos** son las acciones o comportamientos que puede realizar el objeto, como **subirCalificacion()** o **cumplirAnios()**.
 - **Encapsulamiento:** Es el principio de ocultar los datos internos de un objeto y controlar el acceso a ellos. Se logra usando **modificadores de acceso** como **private**. Para manipular los atributos privados, utilizamos métodos especiales llamados **Getters** (para obtener el valor) y **Setters** (para establecer o modificar el valor).
-

Desarrollo del Caso Práctico

A continuación, se presentan los ejercicios desarrollados en Java, aplicando los conceptos de POO solicitados.

1. Registro de Estudiantes

Para este punto, se creó la clase **Estudiante** con los atributos **nombre**, **apellido**, **curso** y **calificación**. Se implementaron métodos para mostrar la información y modificar la calificación.

Explicación del Código:

- **Clase Estudiante:** Define la estructura de un estudiante. El constructor inicializa los atributos al crear un objeto.
- **Métodos:** **mostrarInfo()** imprime los datos del estudiante, mientras que **subirCalificacion()** y **bajarCalificacion()** ajustan el atributo **calificacion** de manera controlada.

```
public class Estudiante {
    // Atributos privados (encapsulamiento)
    private String nombre;
    private String apellido;
    private String curso;
    private double calificacion;

    // Constructor para inicializar el objeto
    public Estudiante(String nombre, String apellido, String curso, double
calificacion) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.curso = curso;
        setCalificacion(calificacion); // Usamos el setter para validar
    }

    // Método para mostrar información del estudiante
    public void mostrarInfo() {
        System.out.println("=== INFORMACIÓN DEL ESTUDIANTE ===");
        System.out.println("Nombre: " + nombre + " " + apellido);
        System.out.println("Curso: " + curso);
        System.out.printf("Calificación: %.2f%n", calificacion);
        System.out.println("Estado: " + obtenerEstadoAcademico());
        System.out.println("=====");
    }

    // Método para subir calificación con validación
    public void subirCalificacion(double puntos) {
        if (puntos <= 0) {
            System.out.println("Error: Los puntos deben ser positivos");
            return;
        }

        double nuevaCalificacion = this.calificacion + puntos;
        if (nuevaCalificacion > 10.0) {
            System.out.println("Advertencia: La calificación no puede
superar 10.0");
            this.calificacion = 10.0;
        } else {
            this.calificacion = nuevaCalificacion;
        }

        System.out.printf("Calificación aumentada en %.2f puntos. Nueva
calificación: %.2f%n",
            puntos, this.calificacion);
    }
}
```

```
// Método para bajar calificación con validación
public void bajarCalificacion(double puntos) {
    if (puntos <= 0) {
        System.out.println("Error: Los puntos deben ser positivos");
        return;
    }

    double nuevaCalificacion = this.calificacion - puntos;
    if (nuevaCalificacion < 0.0) {
        System.out.println("Advertencia: La calificación no puede ser negativa");
        this.calificacion = 0.0;
    } else {
        this.calificacion = nuevaCalificacion;
    }

    System.out.printf("Calificación reducida en %.2f puntos. Nueva calificación: %.2f%n",
        puntos, this.calificacion);
}

// Método auxiliar para determinar el estado académico
private String obtenerEstadoAcademico() {
    if (calificacion >= 7.0) return "Aprobado";
    else if (calificacion >= 4.0) return "Regular";
    else return "Libre";
}

// Getters y Setters
public String getNombre() { return nombre; }
public String getApellido() { return apellido; }
public String getCurso() { return curso; }
public double getCalificacion() { return calificacion; }

public void setCalificacion(double calificacion) {
    if (calificacion >= 0.0 && calificacion <= 10.0) {
        this.calificacion = calificacion;
    } else {
        System.out.println("Calificación inválida. Debe estar entre 0 y 10");
        this.calificacion = 0.0;
    }
}
}
```

Ejecución y Tarea:

Se creó una instancia de la clase `TestEstudiante`, se mostró su información inicial y se modificó su calificación usando los métodos correspondientes.

```
public class TestEstudiante {
    public static void main(String[] args) {
        System.out.println("=== PRUEBA DE LA CLASE ESTUDIANTE ===\n");

        // Instanciar un estudiante
        Estudiante estudiante1 = new Estudiante("Renzo", "Calcatelli",
        "Programación II", 6.5);

        // Mostrar información inicial
        estudiante1.mostrarInfo();

        // Simular cambios en las calificaciones
        System.out.println("\n--- Modificando calificaciones ---");
        estudiante1.subirCalificacion(1.5); // Debería llegar a 8.0
        estudiante1.mostrarInfo();

        estudiante1.bajarCalificacion(3.0); // Debería llegar a 5.0
        estudiante1.mostrarInfo();

        // Probar validaciones
        System.out.println("\n--- Probando validaciones ---");
        estudiante1.subirCalificacion(6.0); // Debería limitarse a 10.0
        estudiante1.mostrarInfo();

        estudiante1.bajarCalificacion(15.0); // Debería limitarse a 0.0
        estudiante1.mostrarInfo();
    }
}
```

2. Registro de Mascotas

Se diseñó la clase `Mascota` con los atributos `nombre`, `especie` y `edad`. Los métodos implementados son `mostrarInfo()` y `cumplirAnios()`.

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

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

    // Mostrar información de la mascota
    public void mostrarInfo() {
        System.out.println("== INFORMACIÓN DE LA MASCOTA ==");
        System.out.println("Nombre: " + nombre);
        System.out.println("Especie: " + especie);
        System.out.println("Edad: " + edad + " años");
        System.out.println("Etapa de vida: " + determinarEtapaVida());
        System.out.println("=====");
    }

    // Método para simular el cumpleaños
    public void cumplirAnios() {
        edad++;
        System.out.println("¡Feliz cumpleaños " + nombre + "! Ahora tiene "
+ edad + " años.");

        // Verificar cambio de etapa de vida
        String nuevaEtapa = determinarEtapaVida();
        System.out.println("Etapa de vida: " + nuevaEtapa);
    }

    // Método privado para determinar la etapa de vida
    private String determinarEtapaVida() {
        if (especie.equalsIgnoreCase("Perro")) {
            if (edad < 1) return "Cachorro";
        }
    }
}
```

```
        else if (edad < 7) return "Adulto";
        else return "Senior";
    } else if (especie.equalsIgnoreCase("Gato")) {
        if (edad < 1) return "Gatito";
        else if (edad < 8) return "Adulto";
        else return "Senior";
    } else {
        if (edad < 2) return "Joven";
        else if (edad < 10) return "Adulto";
        else return "Senior";
    }
}

// Getters y Setters
public String getNombre() { return nombre; }
public String getEspecie() { return especie; }
public int getEdad() { return edad; }

public void setEdad(int edad) {
    if (edad >= 0) {
        this.edad = edad;
    } else {
        System.out.println("La edad no puede ser negativa");
        this.edad = 0;
    }
}
}
```

Ejecución y Tarea:

Se instanció un objeto `Mascota` y se usaron sus métodos para simular el paso del tiempo y verificar los cambios en la edad.

```
public class TestMascota {
    public static void main(String[] args) {
        System.out.println("=== PRUEBA DE LA CLASE MASCOTA ===\n");

        // Crear una mascota
        Mascota miPerro = new Mascota("Angus", "Perro", 3);

        // Mostrar información inicial
        miPerro.mostrarInfo();

        // Simular el paso del tiempo
        System.out.println("\n--- Simulando el paso del tiempo ---");
        for (int i = 0; i < 5; i++) {
            miPerro.cumplirAños();
        }

        System.out.println("\n--- Estado final ---");
        miPerro.mostrarInfo();
    }
}
```

3. Encapsulamiento con la Clase Libro

Este ejercicio se enfoca en el **encapsulamiento**. Los atributos `titulo`, `autor` y `añoPublicacion` son privados. El acceso se controla mediante getters y un setter con validación.

Explicación del Código:

- **Atributos `private`:** Al ser privados, solo son accesibles dentro de la misma clase. Esto protege los datos de modificaciones no deseadas.
- **Getters:** Los métodos `getTitulo()`, `getAutor()` y `getAñoPublicacion()` permiten leer los valores de los atributos privados.
- **Setter con validación:** El método `setAñoPublicacion()` incluye una validación para asegurar que el año ingresado sea un valor válido.


```
public class Libro {
    // Atributos privados - principio de encapsulamiento
    private String titulo;
    private String autor;
    private int anioPublicacion;

    // Constructor
    public Libro(String titulo, String autor, int anioPublicacion) {
        this.titulo = titulo;
        this.autor = autor;
        setAnioPublicacion(anioPublicacion); // Usar setter para validación
    }

    // Getters - acceso controlado de lectura
    public String getTitulo() {
        return titulo;
    }

    public String getAutor() {
        return autor;
    }

    public int getAnioPublicacion() {
        return anioPublicacion;
    }

    // Setter con validación para año de publicación
    public void setAnioPublicacion(int anioPublicacion) {
        int anioActual = java.time.Year.now().getValue();

        if (anioPublicacion > 0 && anioPublicacion <= anioActual) {
            this.anioPublicacion = anioPublicacion;
            System.out.println("Año de publicación actualizado
correctamente: " + anioPublicacion);
        } else {
            System.out.println("Error: Año inválido. Debe estar entre 1 y "
+ anioActual);
        }
    }

    // Método para mostrar información del libro
    public void mostrarInfo() {
        System.out.println("=== INFORMACIÓN DEL LIBRO ===");
        System.out.println("Título: " + titulo);
        System.out.println("Autor: " + autor);
        System.out.println("Año de Publicación: " + anioPublicacion);
    }
}
```

```
        System.out.println("Antigüedad: " + calcularAntigüedad() + " años");
        System.out.println("=====");
    }

    // Método auxiliar para calcular antigüedad
    private int calcularAntigüedad() {
        return java.time.Year.now().getValue() - anioPublicacion;
    }
}
```

Ejecución y Tarea:

Se creó un objeto `Libro` y se demostró el funcionamiento del setter con validación, intentando modificar el año con un valor inválido y luego con uno válido.

```
public class TestLibro {
    public static void main(String[] args) {
        System.out.println("=== PRUEBA DE LA CLASE LIBRO ===\n");

        // Crear un libro
        Libro libro = new Libro("Don Quijote", "Miguel de Cervantes", 1605);

        // Mostrar información inicial
        libro.mostrarInfo();

        System.out.println("\n--- Probando validaciones ---");

        // Intentar modificar con año inválido (futuro)
        libro.setAnioPublicacion(2030);

        // Intentar modificar con año inválido (negativo)
        libro.setAnioPublicacion(-500);

        // Modificar con año válido
        libro.setAnioPublicacion(1615);

        System.out.println("\n--- Estado final ---");
        libro.mostrarInfo();
    }
}
```

4. Gestión de Gallinas en Granja Digital

Se creó la clase `Gallina` con los atributos `idGallina`, `edad` y `huevosPuestos`. Los métodos `ponerHuevo()`, `envejecer()` y `mostrarEstado()` simulan las acciones de una gallina.

```
public class Gallina {
    private static int contadorId = 1; // Variable estática para generar
    IDs únicos
    private int idGallina;
    private int edad;
    private int huevosPuestos;

    // Constructor
    public Gallina(int edad) {
        this.idGallina = contadorId++;
        setEdad(edad);
        this.huevosPuestos = 0;
    }

    // Método para poner huevos
    public void ponerHuevo() {
        if (edad < 1) {
            System.out.println("La gallina #" + idGallina + " es muy joven
para poner huevos");
            return;
        }

        if (edad > 8) {
            System.out.println("La gallina #" + idGallina + " es muy vieja,
pone pocos huevos");
            // Gallinas viejas ponen menos huevos
            if (Math.random() > 0.3) {
                return;
            }
        }

        huevosPuestos++;
        System.out.println("¡La gallina #" + idGallina + " puso un huevo!
Total: " + huevosPuestos);
    }

    // Método para envejecer
    public void envejecer() {
        edad++;
        System.out.println("La gallina #" + idGallina + " cumplió años.
Ahora tiene " + edad + " años");
    }
}
```

```
        if (edad == 1) {
            System.out.println("¡Ya puede empezar a poner huevos!");
        } else if (edad > 8) {
            System.out.println("La gallina está entrando en edad
avanzada");
        }
    }

    // Método para mostrar estado actual
    public void mostrarEstado() {
        System.out.println("== ESTADO DE LA GALLINA #" + idGallina + "
==");
        System.out.println("Edad: " + edad + " años");
        System.out.println("Huevos puestos: " + huevosPuestos);
        System.out.println("Productividad: " + evaluarProductividad());
        System.out.println("Estado: " + determinarEstado());
        System.out.println("=====");
    }

    // Método privado para evaluar productividad
    private String evaluarProductividad() {
        if (huevosPuestos == 0) return "Sin producción";
        else if (huevosPuestos < 10) return "Baja";
        else if (huevosPuestos < 50) return "Media";
        else return "Alta";
    }

    // Método privado para determinar estado
    private String determinarEstado() {
        if (edad < 1) return "Pollita";
        else if (edad <= 3) return "Joven y productiva";
        else if (edad <= 6) return "Adulta";
        else return "Senior";
    }

    // Getters
    public int getIdGallina() { return idGallina; }
    public int getEdad() { return edad; }
    public int getHuevosPuestos() { return huevosPuestos; }

    // Setter con validación
    public void setEdad(int edad) {
        if (edad >= 0) {
            this.edad = edad;
        } else {
            System.out.println("La edad no puede ser negativa");
            this.edad = 0;
        }
    }
}
```

```
}  
}  
}
```

Ejecución y Tarea:

Se crearon dos objetos `Gallina` y se simularon sus acciones. Al final, se mostró el estado de ambas para verificar los cambios.

```
public class TestGallina {  
    public static void main(String[] args) {  
        System.out.println("=== SIMULACIÓN DE GRANJA DIGITAL ===\n");  
  
        // Crear dos gallinas  
        Gallina gallina1 = new Gallina(0); // Pollita  
        Gallina gallina2 = new Gallina(2); // Gallina adulta  
  
        // Mostrar estado inicial  
        System.out.println("--- Estado inicial ---");  
        gallina1.mostrarEstado();  
        gallina2.mostrarEstado();  
  
        // Simular el paso del tiempo  
        System.out.println("\n--- Simulando 3 años de vida ---");  
  
        for (int año = 1; año <= 3; año++) {  
            System.out.println("\n=== AÑO " + año + " ===");  
  
            // Envejecer las gallinas  
            gallina1.envejecer();  
            gallina2.envejecer();  
  
            // Simular puesta de huevos durante el año  
            for (int mes = 1; mes <= 12; mes++) {  
                // Cada gallina intenta poner 2-3 huevos por mes  
                for (int intento = 1; intento <= 3; intento++) {  
                    if (Math.random() > 0.3) { // 70% de probabilidad  
                        gallina1.ponerHuevo();  
                    }  
                    if (Math.random() > 0.3) {  
                        gallina2.ponerHuevo();  
                    }  
                }  
            }  
        }  
    }  
}
```

```

    }
}

// Mostrar estado anual
gallina1.mostrarEstado();
gallina2.mostrarEstado();
}
}
}

```

5. Simulación de Nave Espacial

Se creó la clase `NaveEspacial` con los atributos `nombre` y `combustible`. Se implementaron métodos como `despegar()`, `avanzar()`, `recargarCombustible()` y `mostrarEstado()`, incluyendo validaciones para el combustible.

Explicación del Código:

- **Validaciones:** El método `avanzar()` valida si hay suficiente combustible antes de realizar el viaje. El método `recargarCombustible()` evita que se supere el límite de combustible.

```

public class NaveEspacial {
    private String nombre;
    private double combustible;
    private static final double COMBUSTIBLE_MAXIMO = 100.0;
    private static final double CONSUMO_POR_KM = 0.5;
    private double distanciaRecorrida;

    // Constructor
    public NaveEspacial(String nombre, double combustibleInicial) {
        this.nombre = nombre;
        this.distanciaRecorrida = 0.0;
        recargarCombustible(combustibleInicial);
    }

    // Método para despegar
    public boolean despegar() {
        if (combustible < 10.0) {
            System.out.println("ERROR: Combustible insuficiente para despegar. Se necesitan al menos 10 unidades");
            return false;
        }
    }
}

```

```
    }

    combustible -= 10.0;
    System.out.println("¡" + nombre + " ha despegado exitosamente!");
    System.out.printf("Combustible restante: %.2f unidades%n",
combustible);
    return true;
}

// Método para avanzar
public boolean avanzar(double distancia) {
    if (distancia <= 0) {
        System.out.println("ERROR: La distancia debe ser positiva");
        return false;
    }

    double combustibleNecesario = distancia * CONSUMO_POR_KM;

    if (combustible < combustibleNecesario) {
        System.out.printf("ERROR: Combustible insuficiente para avanzar
%.2f km%n", distancia);
        System.out.printf("Se necesitan %.2f unidades, pero solo hay
%.2f%n",
            combustibleNecesario, combustible);
        return false;
    }

    // Realizar el viaje
    combustible -= combustibleNecesario;
    distanciaRecorrida += distancia;

    System.out.printf("%s avanzó %.2f km%n", nombre, distancia);
    System.out.printf("Combustible consumido: %.2f unidades%n",
combustibleNecesario);
    System.out.printf("Combustible restante: %.2f unidades%n",
combustible);

    return true;
}
```

```
// Método para recargar combustible
public void recargarCombustible(double cantidad) {
    if (cantidad <= 0) {
        System.out.println("ERROR: La cantidad de combustible debe ser
positiva");
        return;
    }

    double combustibleAnterior = combustible;
    combustible += cantidad;

    if (combustible > COMBUSTIBLE_MAXIMO) {
        combustible = COMBUSTIBLE_MAXIMO;
        double exceso = (combustibleAnterior + cantidad) -
COMBUSTIBLE_MAXIMO;
        System.out.printf("ADVERTENCIA: Tanque lleno. Se perdieron %.2f
unidades por exceso%n", exceso);
    }

    double recargado = combustible - combustibleAnterior;
    System.out.printf("Combustible recargado: %.2f unidades%n",
recargado);
    System.out.printf("Combustible total: %.2f/%.2f unidades%n",
combustible, COMBUSTIBLE_MAXIMO);
}

// Método para mostrar estado
public void mostrarEstado() {
    System.out.println("=== ESTADO DE LA NAVE " + nombre.toUpperCase() +
" ===");
    System.out.printf("Combustible: %.2f/%.2f unidades (%.1f%%)%n",
        combustible, COMBUSTIBLE_MAXIMO,
(combustible/COMBUSTIBLE_MAXIMO)*100);
    System.out.printf("Distancia recorrida: %.2f km%n",
distanciaRecorrida);
    System.out.println("Estado operativo: " +
determinarEstadoOperativo());
    System.out.println("Autonomía restante: " + calcularAutonomia() + "
km");
    System.out.println("=====");
}
```



```
// Método privado para determinar estado operativo
private String determinarEstadoOperativo() {
    if (combustible >= 50) return "Excelente";
    else if (combustible >= 25) return "Bueno";
    else if (combustible >= 10) return "Precaución";
    else return "Crítico";
}

// Método privado para calcular autonomía
private double calcularAutonomia() {
    return combustible / CONSUMO_POR_KM;
}

// Getters
public String getNombre() { return nombre; }
public double getCombustible() { return combustible; }
public double getDistanciaRecorrida() { return distanciaRecorrida; }
}
```

Ejecución y Tarea:

Se creó una nave con 50 unidades de combustible y se demostró el funcionamiento de las validaciones, intentando avanzar con combustible insuficiente y luego recargando para poder hacerlo.

```
public class TestNaveEspacial {
    public static void main(String[] args) {
        System.out.println("=== SIMULACIÓN DE NAVE ESPACIAL ===\n");

        // Crear una nave con 50 unidades de combustible
        NaveEspacial nave = new NaveEspacial("Enterprise", 50.0);

        // Mostrar estado inicial
        System.out.println("--- Estado inicial ---");
        nave.mostrarEstado();

        // Intentar despegar
        System.out.println("\n--- Intento de despegue ---");
        if (nave.despegar()) {
            nave.mostrarEstado();
        }
    }
}
```

```
// Intentar avanzar sin recargar
System.out.println("\n--- Intento de viaje largo sin recargar
---");
nave.avanzar(100); // Necesita 50 unidades, pero solo tiene ~40

// Intentar avanzar distancia menor
System.out.println("\n--- Viaje corto ---");
nave.avanzar(20); // Debería funcionar
nave.mostrarEstado();

// Recargar combustible
System.out.println("\n--- Recarga de combustible ---");
nave.recargarCombustible(60); // Debería limitarse a 100 total
nave.mostrarEstado();

// Ahora intentar el viaje largo
System.out.println("\n--- Viaje largo después de recargar ---");
nave.avanzar(100);
nave.mostrarEstado();

// Simular varios viajes
System.out.println("\n--- Simulación de múltiples viajes ---");
double[] distancias = {30, 25, 40, 20};

for (double distancia : distancias) {
    System.out.println("\n--- Viaje de " + distancia + " km ---");
    if (!nave.avanzar(distancia)) {
        System.out.println("Recargando combustible para
continuar...");
        nave.recargarCombustible(50);
        nave.avanzar(distancia);
    }
}

// Estado final
System.out.println("\n--- ESTADO FINAL ---");
nave.mostrarEstado();
}
```

Conclusiones

A través de este trabajo, pude comprender la diferencia fundamental entre **clases** (plantillas) y **objetos** (instancias concretas). La implementación de cada clase me permitió aplicar el principio de **encapsulamiento** para proteger los datos, utilizando **getters y setters** para un acceso controlado. Además, logré definir los **comportamientos** de los objetos a través de sus métodos, lo que me ayudó a entender cómo los objetos pueden cambiar su **estado** e **identidad** de manera correcta. Este ejercicio reforzó el pensamiento modular y la reutilización del código en Java, ya que cada clase representa una entidad independiente y funcional. En general, considero que el trabajo me proporcionó una base sólida para continuar explorando la programación orientada a objetos.