

Trabajo Práctico N° 3

## Introducción a la POO

Comision 16

Marinoni Macarena <marinonimacarena@gmail.com>

**Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional**

**Programación II**

**Profesor:** Cortez, Alberto

**Tutor:** Bianchi, Neyén

25/09/2025

## ÍNDICE

<b>Objetivo general</b>	<b>3</b>
<b>Marco teórico</b>	<b>4</b>
<b>Caso práctico</b>	<b>5</b>
1. Registro de Estudiantes	5
2. Registro de Mascotas	7
3. Encapsulamiento con la clase libro	9
4. Gestión de Gallinas en granja digital	11
5. Simulación de nave espacial	14
<b>Conclusión</b>	<b>17</b>
<b>Anexo</b>	<b>18</b>

**Objetivo general**

Comprender y aplicar los fundamentos de la POO en Java (clases, objetos, atributos, métodos, estado, identidad y encapsulamiento) para estructurar programas de manera modular y reutilizable.

## Marco teórico

La Programación Orientada a Objetos (POO) es un paradigma que busca modelar el software siguiendo una lógica similar a la del mundo real. En lugar de pensar el programa como una secuencia de instrucciones, la POO organiza el código en clases y objetos, permitiendo mayor claridad, modularidad y escalabilidad.

Una clase puede entenderse como un molde que describe las características de un conjunto de objetos. Define atributos (propiedades que describen el estado) y métodos (acciones o comportamientos). Un objeto es una instancia concreta de esa clase, con su propio estado en memoria.

Los tres elementos centrales de todo objeto son:

- Estado: los valores actuales de sus atributos, que pueden cambiar durante la ejecución.
- Comportamiento: las acciones que el objeto puede realizar mediante sus métodos.
- Identidad: la referencia única que lo distingue de otros objetos, aunque tengan el mismo estado.

Dentro de la POO existen principios fundamentales conocidos como los cuatro pilares:

1. Abstracción: simplificar la realidad, enfocándose en lo esencial y ocultando los detalles irrelevantes.
2. Encapsulamiento: proteger la información interna de un objeto, ocultándose al exterior y brindando acceso controlado a través de métodos.
3. Herencia: posibilita crear nuevas clases a partir de otras existentes, reutilizando código y extendiendo funcionalidades.
4. Polimorfismo: permite que distintos objetos responden a un mismo mensaje de formas diferentes, favoreciendo la flexibilidad.

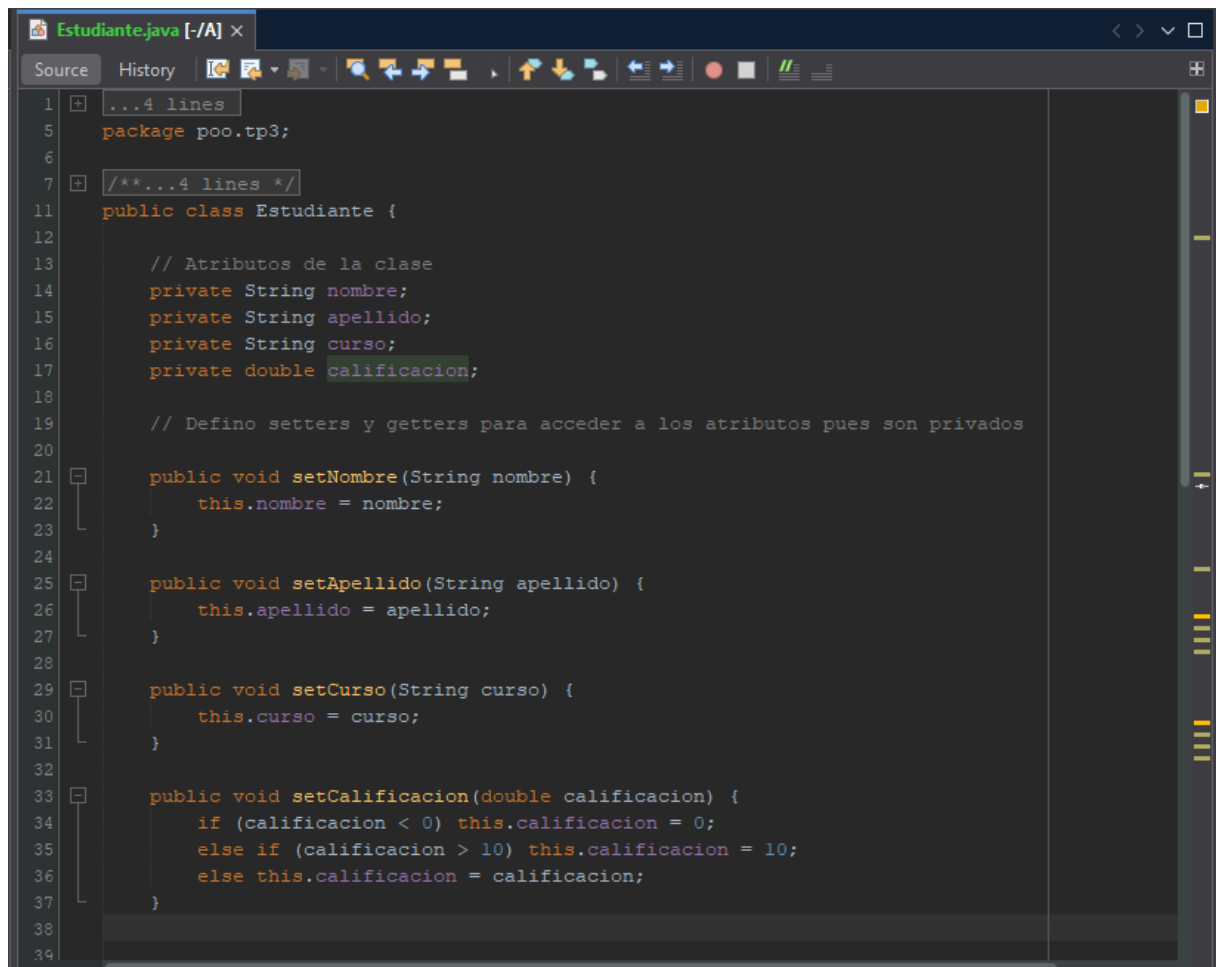
Un aspecto clave es el encapsulamiento, que en Java se implementa mediante modificadores de acceso (`private`, `public`, `protected`) y el uso de getters y setters. Esto garantiza que los atributos no puedan ser manipulados libremente, sino a través de reglas de negocio definidas en los métodos.

## Caso práctico

A continuación se desarrollaran diferentes actividades donde se aplican los conceptos antes mencionados

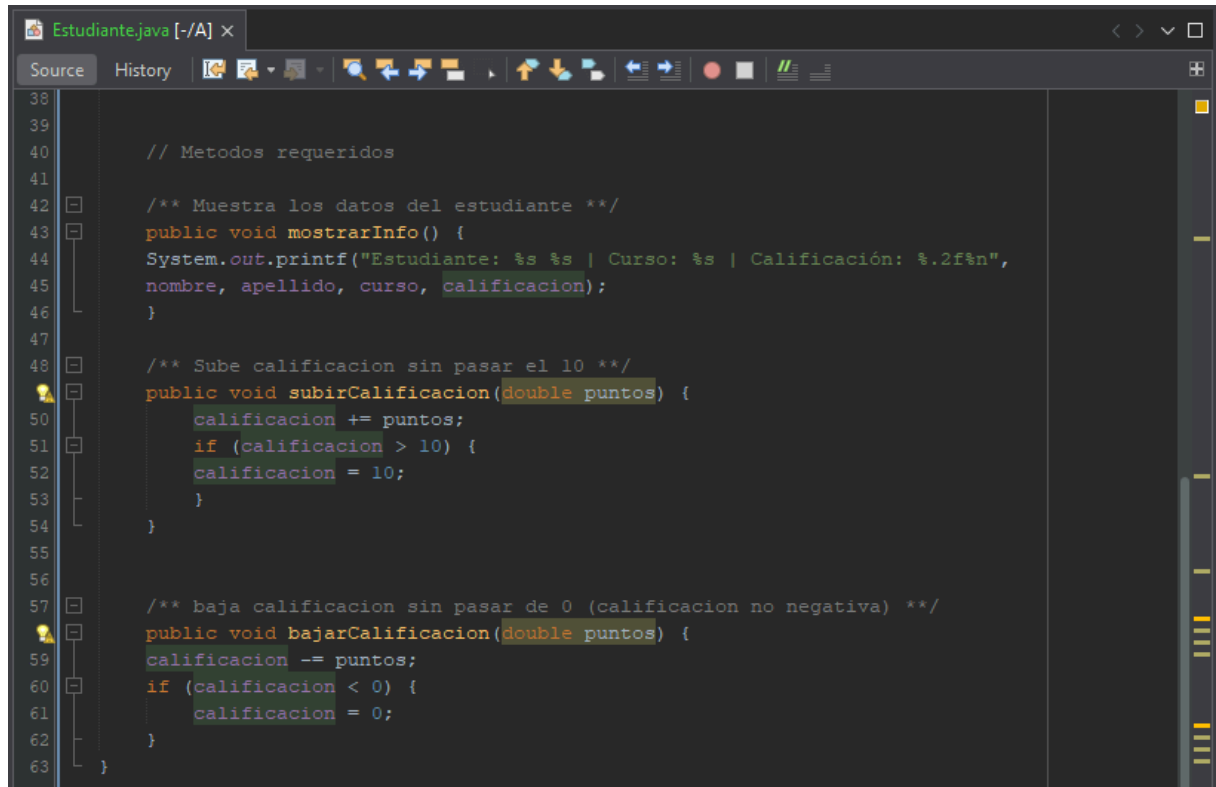
### 1. Registro de Estudiantes

En primer lugar se crea una clase Estudiante con los atributos: nombre, apellido, curso, calificación. Además, como se tienen atributos privados, se configuran setters y getters para acceder a ellos



```
1  ...4 lines
5  package poo.tp3;
6
7  /**...4 lines */
11 public class Estudiante {
12
13     // Atributos de la clase
14     private String nombre;
15     private String apellido;
16     private String curso;
17     private double calificacion;
18
19     // Defino setters y getters para acceder a los atributos pues son privados
20
21     public void setNombre(String nombre) {
22         this.nombre = nombre;
23     }
24
25     public void setApellido(String apellido) {
26         this.apellido = apellido;
27     }
28
29     public void setCurso(String curso) {
30         this.curso = curso;
31     }
32
33     public void setCalificacion(double calificacion) {
34         if (calificacion < 0) this.calificacion = 0;
35         else if (calificacion > 10) this.calificacion = 10;
36         else this.calificacion = calificacion;
37     }
38
39 }
```

Por otro lado, se desarrollan los métodos: mostrarInfo(), subirCalificacion(puntos), bajarCalificacion(puntos).



```
38
39
40 // Metodos requeridos
41
42 /** Muestra los datos del estudiante */
43 public void mostrarInfo() {
44     System.out.printf("Estudiante: %s %s | Curso: %s | Calificación: %.2f\n",
45         nombre, apellido, curso, calificacion);
46 }
47
48 /** Sube calificacion sin pasar el 10 */
49 public void subirCalificacion(double puntos) {
50     calificacion += puntos;
51     if (calificacion > 10) {
52         calificacion = 10;
53     }
54 }
55
56
57 /** baja calificacion sin pasar de 0 (calificacion no negativa) */
58 public void bajarCalificacion(double puntos) {
59     calificacion -= puntos;
60     if (calificacion < 0) {
61         calificacion = 0;
62     }
63 }
```

Para demostrar el funcionamiento se procede a instanciar a un estudiante, mostrar su información, aumentar y disminuir calificaciones como sigue:

The screenshot shows an IDE with a Java file named `TP_Unidad3.java`. The code defines a package `poo.tp3` and a class `TP_Unidad3` with a `main` method. The `main` method creates an `Estudiante` object, sets its attributes, and calls methods to modify and display its state.

```

1  ...4 lines
5  package poo.tp3;
6
7  /**...4 lines */
11 public class TP_Unidad3 {
12
13     /**
14      * @param args the command line arguments
15      */
16     public static void main(String[] args) {
17         Estudiante e = new Estudiante();
18         e.setNombre("Macarena");
19         e.setApellido("Marinoni");
20         e.setCurso("Programacion II");
21         e.setCalificacion(6);
22
23         // Estado inicial
24         e.mostrarInfo();
25
26         // Aumenta calificación +1.2
27         e.subirCalificacion(1.2);
28         e.mostrarInfo();
29
30         // Disminuye calificación -3.0
31         e.bajarCalificacion(3.0);
32         e.mostrarInfo();
33
34         // Intento de subir por encima de 10 -> se limita
35         e.subirCalificacion(20);
36         e.mostrarInfo();
37     }
38
39 }

```

The Output window shows the execution results:

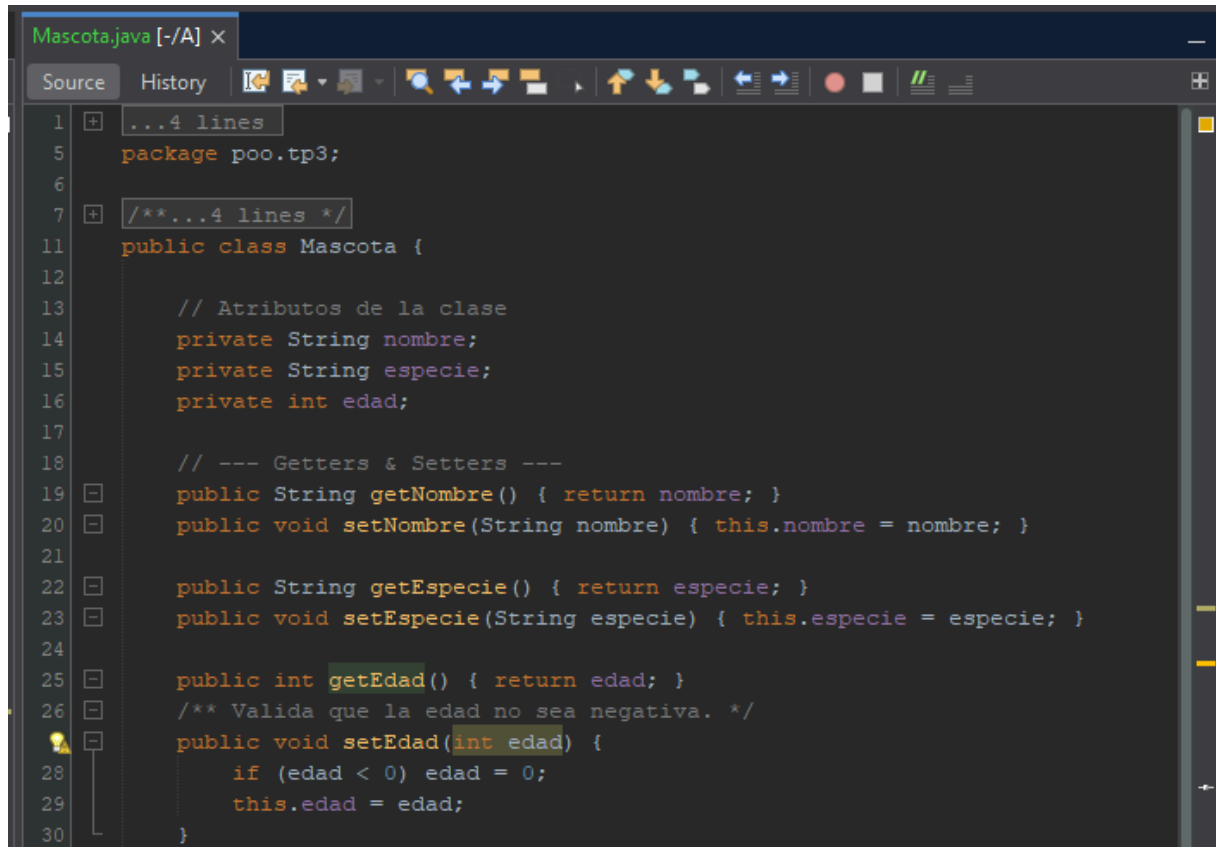
```

run:
Estudiante: Macarena Marinoni | Curso: Programacion II | Calificaci♦n: 6,00
Estudiante: Macarena Marinoni | Curso: Programacion II | Calificaci♦n: 7,20
Estudiante: Macarena Marinoni | Curso: Programacion II | Calificaci♦n: 4,20
Estudiante: Macarena Marinoni | Curso: Programacion II | Calificaci♦n: 10,00
BUILD SUCCESSFUL (total time: 0 seconds)

```

## 2. Registro de Mascotas

Se crea una clase `Mascotas` con los atributos: nombre, especie, edad. Además, como se tienen atributos privados, se configuran setters y getters para acceder a ellos

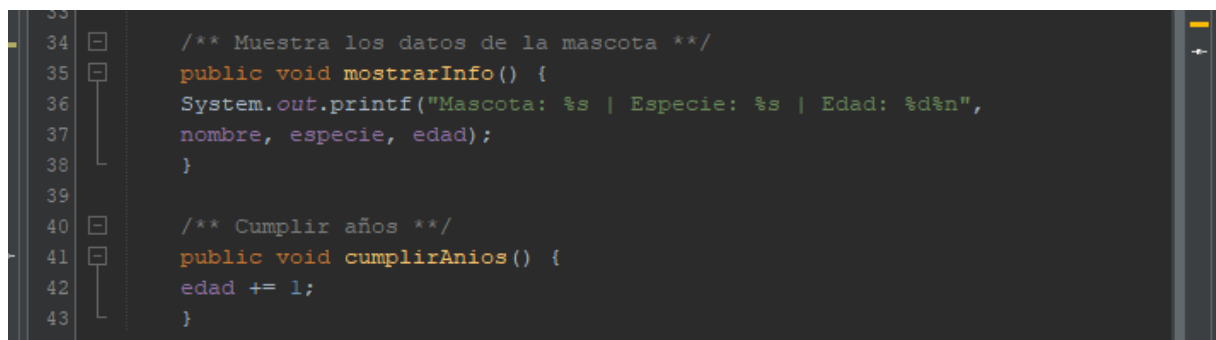


```

1  ...4 lines
5  package poo.tp3;
6
7  /**...4 lines */
11 public class Mascota {
12
13     // Atributos de la clase
14     private String nombre;
15     private String especie;
16     private int edad;
17
18     // --- Getters & Setters ---
19     public String getNombre() { return nombre; }
20     public void setNombre(String nombre) { this.nombre = nombre; }
21
22     public String getEspecie() { return especie; }
23     public void setEspecie(String especie) { this.especie = especie; }
24
25     public int getEdad() { return edad; }
26     /** Valida que la edad no sea negativa. */
27     public void setEdad(int edad) {
28         if (edad < 0) edad = 0;
29         this.edad = edad;
30     }

```

Por otro lado, se desarrollan los métodos: mostrarInfo(), cumplirAños().



```

34 /** Muestra los datos de la mascota */
35 public void mostrarInfo() {
36     System.out.printf("Mascota: %s | Especie: %s | Edad: %d\n",
37         nombre, especie, edad);
38 }
39
40 /** Cumplir años */
41 public void cumplirAños() {
42     edad += 1;
43 }

```

Para demostrar el funcionamiento se procede a crear una mascota, mostrar su información, simular el paso del tiempo y verificar los cambios como sigue:



```

42
43     // CLASE MASCOTAS
44
45     Mascota m = new Mascota();
46     m.setNombre("Tita");
47     m.setEspecie("Gato");
48     m.setEdad(5);
49
50     // Estado inicial
51     m.mostrarInfo();           // Mascota: Luna (Gato) | Edad: 2
52
53     // Cambia el estado con un método
54     m.cumplirAños();
55     m.mostrarInfo();           // Edad: 3
56
57     // Probamos la validación de edad (no debería quedar negativa)
58     m.setEdad(-5);
59     m.mostrarInfo();           // Edad: 0
60
61 }

```

Output x Javadoc

TUPaD\_P2-C22025 - C:\Users\marin\TUPaD\_P2-C22025 x TP\_Unidad3 (run) x TUPaD\_P2-C22025 - C:\Users\marin\ >

run:

```

Mascota: Tita | Especie: Gato | Edad: 5
Mascota: Tita | Especie: Gato | Edad: 6
Mascota: Tita | Especie: Gato | Edad: 0
BUILD SUCCESSFUL (total time: 0 seconds)

```

### 3. Encapsulamiento con la clase libro

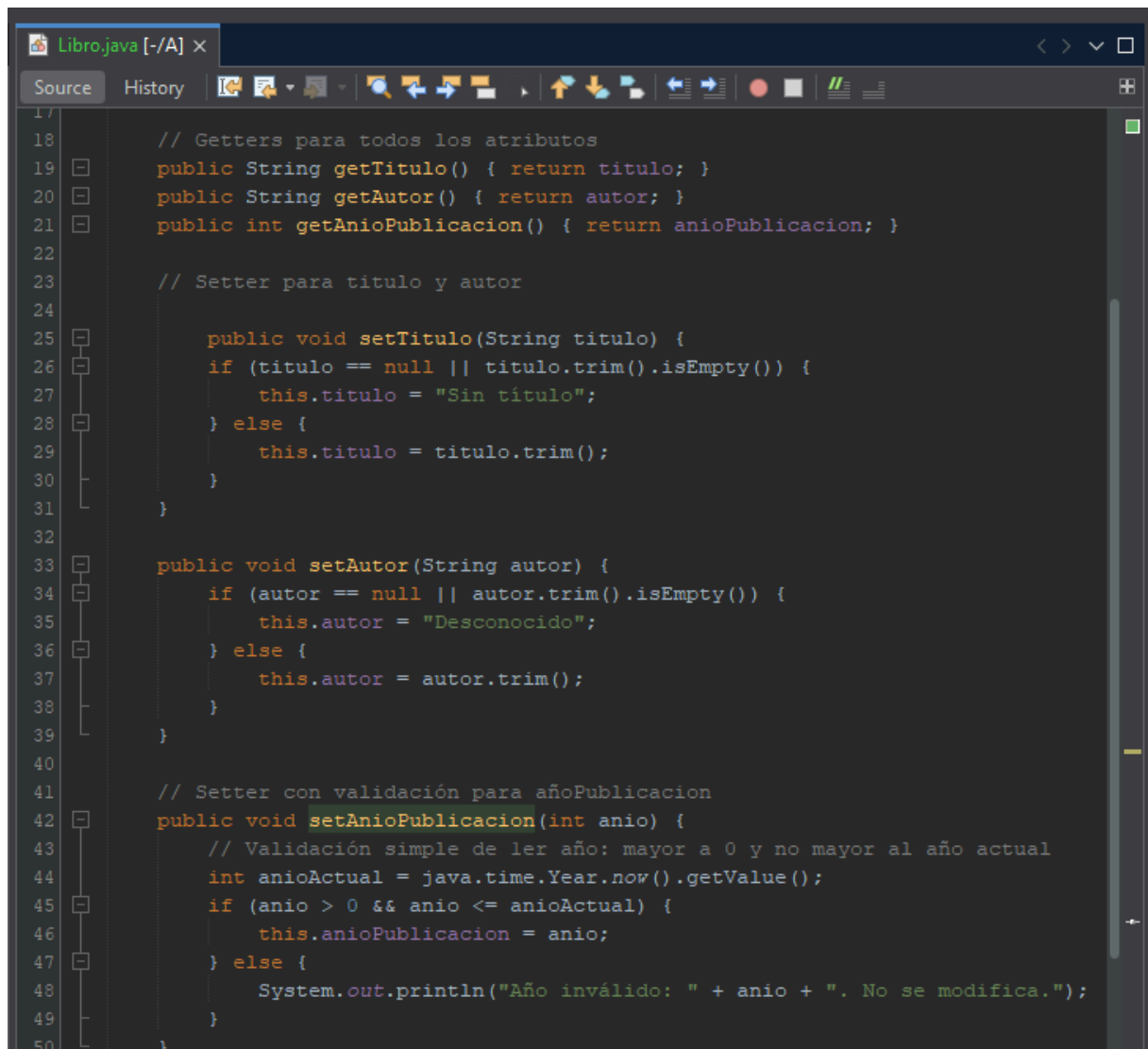
Se crea una clase Libro con los atributos: titulo, autor, añoPublicacion.

```

Libro.java [-/A] x
Source History
1  ...4 lines
5  package poo.tp3;
6
7  /**...4 lines */
11 public class Libro {
12
13     // Atributos de la clase
14     private String titulo;
15     private String autor;
16     private int anioPublicacion;
17

```

Además, se configuran Getters para todos los atributos y Setter con validación para añoPublicacion



```
17
18 // Getters para todos los atributos
19 public String getTitulo() { return titulo; }
20 public String getAutor() { return autor; }
21 public int getAnioPublicacion() { return anioPublicacion; }
22
23 // Setter para titulo y autor
24
25 public void setTitulo(String titulo) {
26     if (titulo == null || titulo.trim().isEmpty()) {
27         this.titulo = "Sin titulo";
28     } else {
29         this.titulo = titulo.trim();
30     }
31 }
32
33 public void setAutor(String autor) {
34     if (autor == null || autor.trim().isEmpty()) {
35         this.autor = "Desconocido";
36     } else {
37         this.autor = autor.trim();
38     }
39 }
40
41 // Setter con validación para añoPublicacion
42 public void setAnioPublicacion(int anio) {
43     // Validación simple de ler año: mayor a 0 y no mayor al año actual
44     int anioActual = java.time.Year.now().getValue();
45     if (anio > 0 && anio <= anioActual) {
46         this.anioPublicacion = anio;
47     } else {
48         System.out.println("Año inválido: " + anio + ". No se modifica.");
49     }
50 }
```

Para demostrar el funcionamiento se procede a crear un libro, intentar modificar el año con un valor inválido y luego con uno válido, mostrar la información final como sigue:

```

62
63 // CLASE LIBROS
64
65 Libro l = new Libro();
66
67 l.setTitulo("IT");
68 l.setAutor("King, Stephen");
69
70 // Intento inválido
71 l.setAnioPublicacion(3000); // mensaje: Año inválido... No se modifica.
72
73 // Intento válido
74 l.setAnioPublicacion(2008);
75
76 // Mostrar información final usando getters
77 System.out.println("Libro: " + l.getTitulo()
78 + " (" + l.getAutor() + ") - Año: " + l.getAnioPublicacion());
79
80

```

Output x Javadoc

TUPaD\_P2-C22025 - C:\Users\marin\TUPaD\_P2-C22025 x TP\_Unidad3 (run) x TUPaD\_P2-C22025 - C:\Users\marin\Docu

```

run:
Año inválido: 3000. No se modifica.
Libro: IT (King, Stephen) - Año: 2008
BUILD SUCCESSFUL (total time: 0 seconds)

```

#### 4. Gestión de Gallinas en granja digital

Se crea una clase Gallina con los atributos: idGallina, edad, huevosPuestos.

```

1 ...4 lines
5 package poo.tp3;
6
7 /**...4 lines */
11 public class Gallina {
12
13 // Atributos de la clase
14 private int idGallina;
15 private int edad;
16 private int huevosPuestos;

```

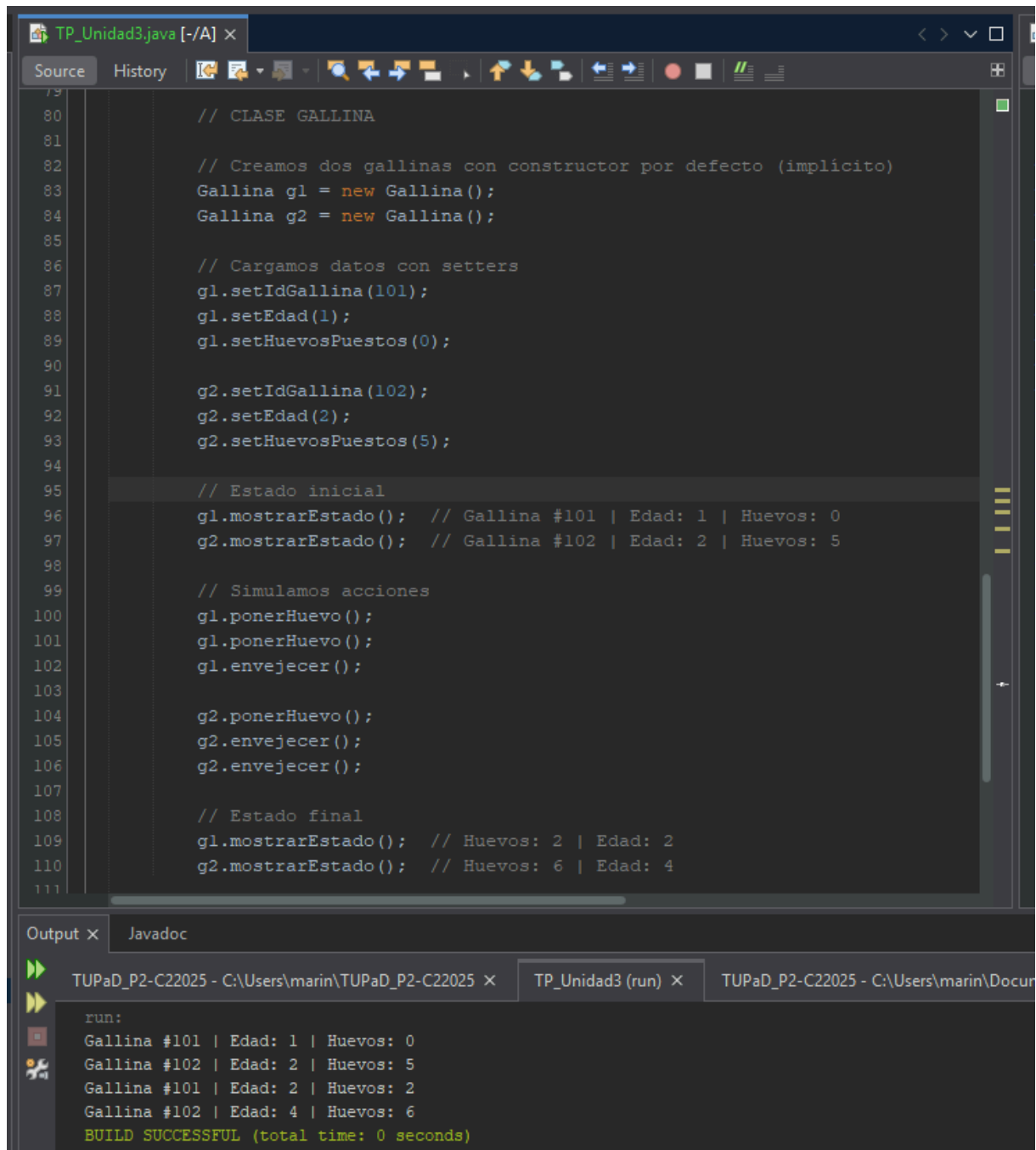
Se desarrollan los métodos ponerHuevo(), envejecer(), mostrarEstado()

```

18 // METODOS
19
20 /** Incrementa en 1 la cantidad de huevos puestos. */
21 public void ponerHuevo() {
22     huevosPuestos += 1;
23 }
24
25 /** Incrementa en 1 la edad */
26 public void envejecer() {
27     edad += 1;
28 }
29
30 /** Muestra el estado actual de la gallina. */
31 public void mostrarEstado() {
32     System.out.printf(
33         "Gallina #%d | Edad: %d | Huevos: %d\n",
34         idGallina, edad, huevosPuestos
35     );
36 }
37
38 // Getters Y Setters
39 public int getIdGallina() { return idGallina; }
40 public void setIdGallina(int idGallina) {
41     if (idGallina <= 0) {
42         System.out.println("idGallina inválido. Debe ser > 0. No se modifica");
43         return;
44     }
45     this.idGallina = idGallina;
46 }
47
48 public int getEdad() { return edad; }
49 public void setEdad(int edad) {
50     if (edad < 0) edad = 0; // evita negativos
51     this.edad = edad;
52 }
53
54 public int getHuevosPuestos() { return huevosPuestos; }
55 public void setHuevosPuestos(int huevosPuestos) {
56     if (huevosPuestos < 0) huevosPuestos = 0; // evita negativos
57     this.huevosPuestos = huevosPuestos;
58 }
59 }
60

```

Para demostrar el funcionamiento se crean dos gallinas, se simulan sus acciones (envejecer y poner huevos), y se muestra su estado.



The screenshot shows an IDE window titled 'TP\_Unidad3.java' with a dark theme. The code defines a 'Gallina' class and simulates two chickens, g1 and g2, over time. The code is as follows:

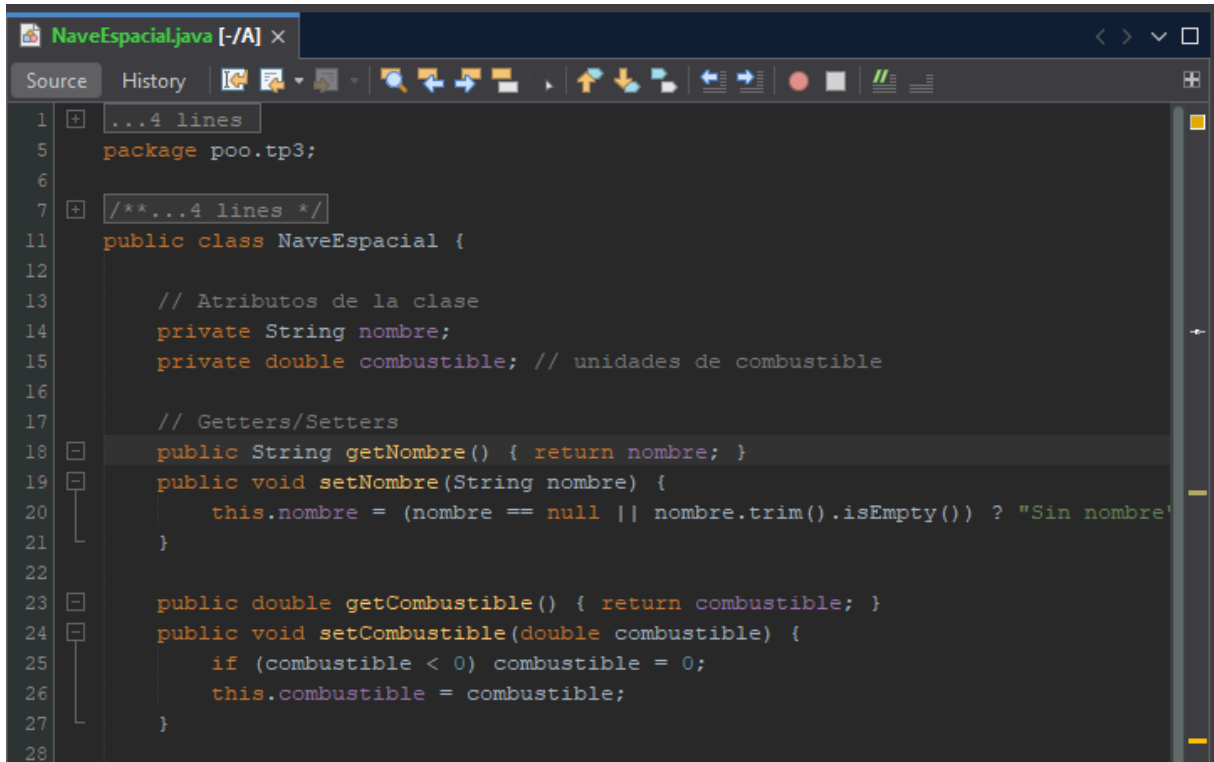
```
79
80 // CLASE GALLINA
81
82 // Creamos dos gallinas con constructor por defecto (implícito)
83 Gallina g1 = new Gallina();
84 Gallina g2 = new Gallina();
85
86 // Cargamos datos con setters
87 g1.setIdGallina(101);
88 g1.setEdad(1);
89 g1.setHuevosPuestos(0);
90
91 g2.setIdGallina(102);
92 g2.setEdad(2);
93 g2.setHuevosPuestos(5);
94
95 // Estado inicial
96 g1.mostrarEstado(); // Gallina #101 | Edad: 1 | Huevos: 0
97 g2.mostrarEstado(); // Gallina #102 | Edad: 2 | Huevos: 5
98
99 // Simulamos acciones
100 g1.ponerHuevo();
101 g1.ponerHuevo();
102 g1.envejecer();
103
104 g2.ponerHuevo();
105 g2.envejecer();
106 g2.envejecer();
107
108 // Estado final
109 g1.mostrarEstado(); // Huevos: 2 | Edad: 2
110 g2.mostrarEstado(); // Huevos: 6 | Edad: 4
111
```

The Output window at the bottom shows the execution results:

```
run:
Gallina #101 | Edad: 1 | Huevos: 0
Gallina #102 | Edad: 2 | Huevos: 5
Gallina #101 | Edad: 2 | Huevos: 2
Gallina #102 | Edad: 4 | Huevos: 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

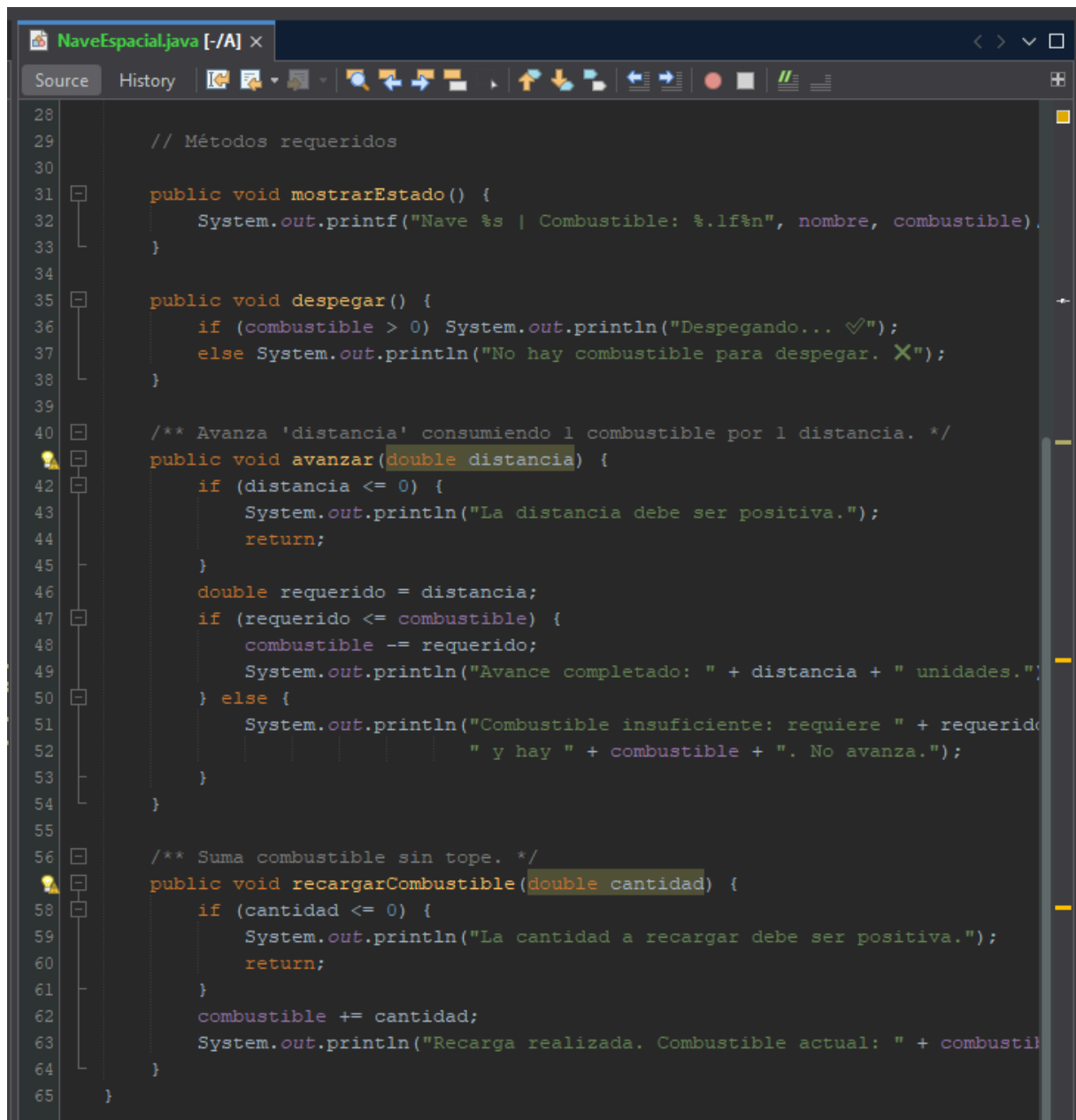
## 5. Simulación de nave espacial

Se crea una clase NaveEspacial con los atributos: nombre, combustible.



```
1  ...4 lines
5  package poo.tp3;
6
7  /**...4 lines */
11 public class NaveEspacial {
12
13     // Atributos de la clase
14     private String nombre;
15     private double combustible; // unidades de combustible
16
17     // Getters/Setters
18     public String getNombre() { return nombre; }
19     public void setNombre(String nombre) {
20         this.nombre = (nombre == null || nombre.trim().isEmpty()) ? "Sin nombre"
21     }
22
23     public double getCombustible() { return combustible; }
24     public void setCombustible(double combustible) {
25         if (combustible < 0) combustible = 0;
26         this.combustible = combustible;
27     }
28 }
```

Se desarrollan los métodos: despegar(), avanzar(distancia), recargarCombustible(cantidad), mostrarEstado(). Adicionalmente, se debe validar que haya suficiente combustible antes de avanzar y evitar que se supere el límite al recargar.

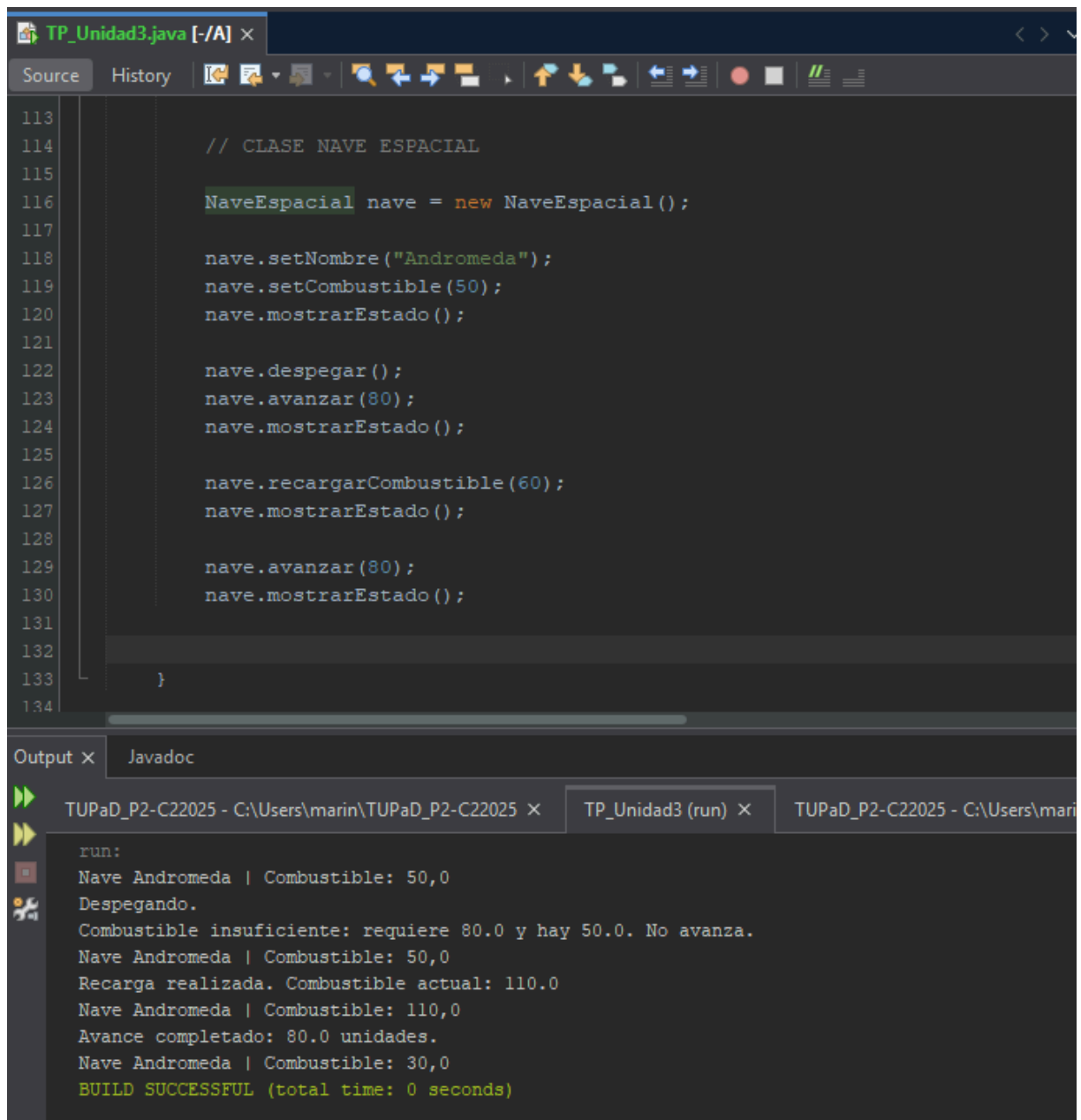


```

28
29 // Métodos requeridos
30
31 public void mostrarEstado() {
32     System.out.printf("Nave %s | Combustible: %.1f%n", nombre, combustible);
33 }
34
35 public void despegar() {
36     if (combustible > 0) System.out.println("Despegando... ✔");
37     else System.out.println("No hay combustible para despegar. ✖");
38 }
39
40 /** Avanza 'distancia' consumiendo 1 combustible por 1 distancia. */
41 public void avanzar(double distancia) {
42     if (distancia <= 0) {
43         System.out.println("La distancia debe ser positiva.");
44         return;
45     }
46     double requerido = distancia;
47     if (requerido <= combustible) {
48         combustible -= requerido;
49         System.out.println("Avance completado: " + distancia + " unidades.");
50     } else {
51         System.out.println("Combustible insuficiente: requiere " + requerido
52             + " y hay " + combustible + ". No avanza.");
53     }
54 }
55
56 /** Suma combustible sin tope. */
57 public void recargarCombustible(double cantidad) {
58     if (cantidad <= 0) {
59         System.out.println("La cantidad a recargar debe ser positiva.");
60         return;
61     }
62     combustible += cantidad;
63     System.out.println("Recarga realizada. Combustible actual: " + combustible);
64 }
65 }

```

Para demostrar el funcionamiento se crea una nave con 50 unidades de combustible, intentar avanzar sin recargar, luego recargar y avanzar correctamente:



The screenshot displays an IDE window titled 'TP\_Unidad3.java [-/A]'. The 'Source' tab is active, showing Java code for a 'Nave Espacial' class. The code includes a constructor, methods for setting name and fuel, and methods for launching, moving, and refueling. Line numbers 113 to 134 are visible on the left. Below the code editor, the 'Output' tab is active, showing the execution results of the 'TP\_Unidad3 (run)' task. The output includes the initial state of the ship, a launch attempt that fails due to insufficient fuel, a successful refueling, and a successful movement, followed by a final state report and a 'BUILD SUCCESSFUL' message.

```
113
114         // CLASE NAVE ESPACIAL
115
116         NaveEspacial nave = new NaveEspacial();
117
118         nave.setNombre("Andromeda");
119         nave.setCombustible(50);
120         nave.mostrarEstado();
121
122         nave.despegar();
123         nave.avanzar(80);
124         nave.mostrarEstado();
125
126         nave.recargarCombustible(60);
127         nave.mostrarEstado();
128
129         nave.avanzar(80);
130         nave.mostrarEstado();
131
132
133     }
134
```

Output × Javadoc

TUPaD\_P2-C22025 - C:\Users\marin\TUPaD\_P2-C22025 × TP\_Unidad3 (run) × TUPaD\_P2-C22025 - C:\Users\marin

run:  
Nave Andromeda | Combustible: 50,0  
Despegando.  
Combustible insuficiente: requiere 80.0 y hay 50.0. No avanza.  
Nave Andromeda | Combustible: 50,0  
Recarga realizada. Combustible actual: 110.0  
Nave Andromeda | Combustible: 110,0  
Avance completado: 80.0 unidades.  
Nave Andromeda | Combustible: 30,0  
BUILD SUCCESSFUL (total time: 0 seconds)



## Conclusión

El presente trabajo evidencia que se logró aplicar los fundamentos de la Programación Orientada a Objetos en Java a partir de clases sencillas —Estudiante, Mascota, Libro, Gallina y NaveEspacial— se modelaron entidades reales para mostrar, en la práctica, la interacción entre estado, comportamiento e identidad. En todos los casos, se reforzó el encapsulamiento mediante atributos `private` y el uso de `getters/setters` con validaciones básicas, evitando estados inválidos (por ejemplo, calificaciones fuera de 0–10, edades negativas o años de publicación incorrectos).

Asimismo, se demostró criterio en el control del estado a través de métodos que modifican los atributos de forma segura (por ejemplo, `subirCalificacion`, `cumplirAnios`, `ponerHuevo`, `avanzar`), y organizó el código en módulos claros, probados desde un `main` independiente por consigna. Esta estructura favorece la legibilidad, la mantenibilidad y sienta bases sólidas para incorporar progresivamente conceptos más avanzados.

## **Anexo**

Link repositorio [https://github.com/MaquiMarinoni/TUPaD\\_P2-C22025.git](https://github.com/MaquiMarinoni/TUPaD_P2-C22025.git)