

PROGRAMACIÓN II

Trabajo Práctico 3: Introducción a la Programación Orientada a Objetos

Alumno

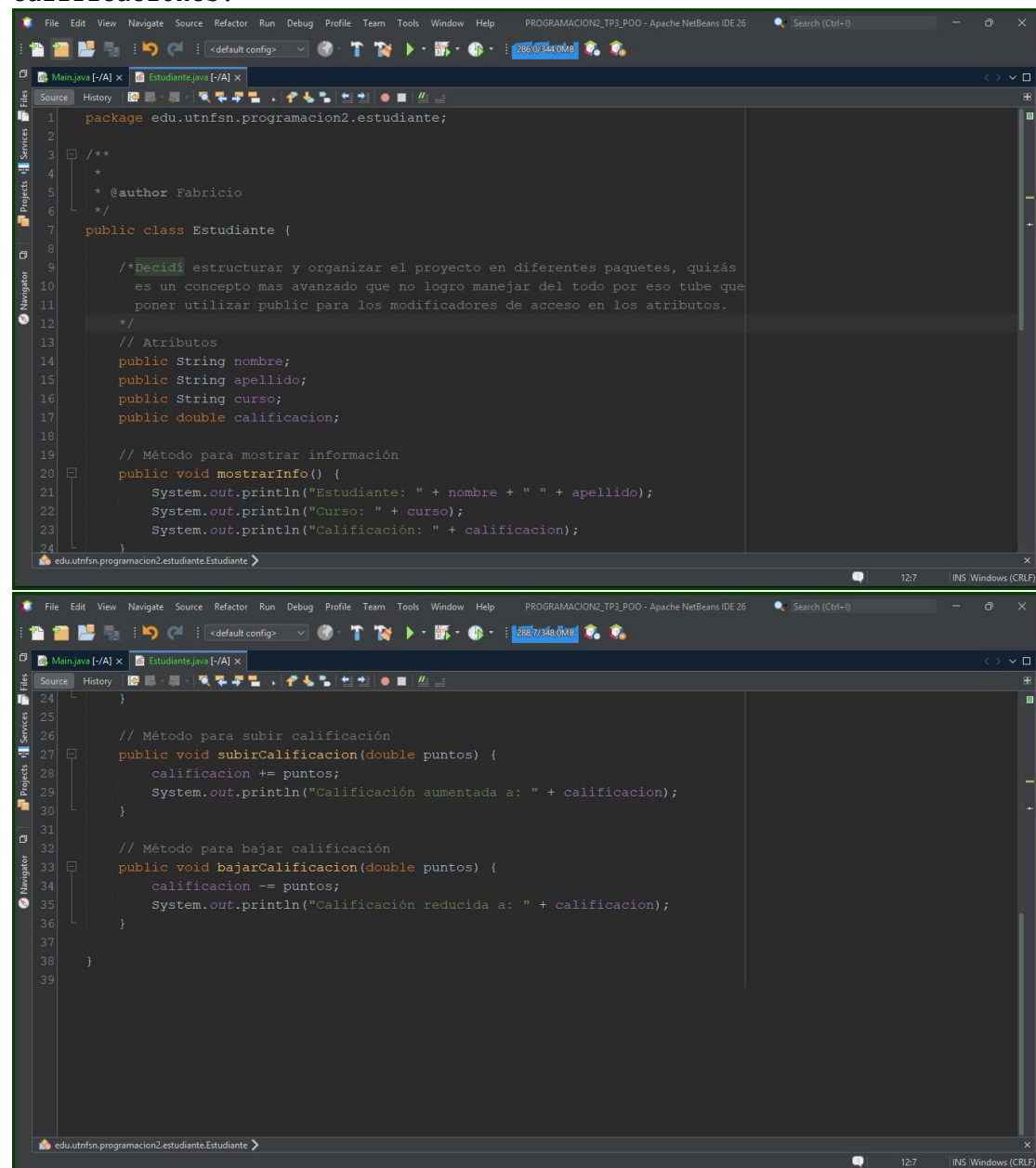
Fabrizio Nicolás Puccio

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.



```
package edu.utnfsn.programacion2.estudiante;

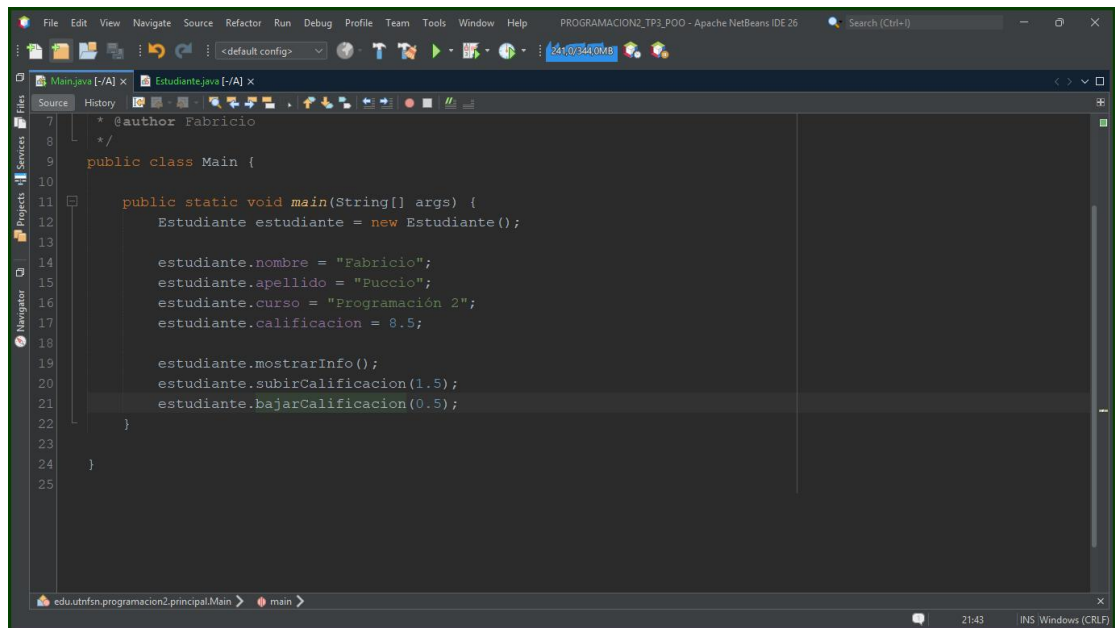
/**
 *
 * @author Fabrizio
 */
public class Estudiante {

    /**Decidi estructurar y organizar el proyecto en diferentes paquetes, quizás
     es un concepto mas avanzado que no logro manejar del todo por eso tube que
     poner utilizar public para los modificadores de acceso en los atributos.
     */
    // Atributos
    public String nombre;
    public String apellido;
    public String curso;
    public double calificacion;

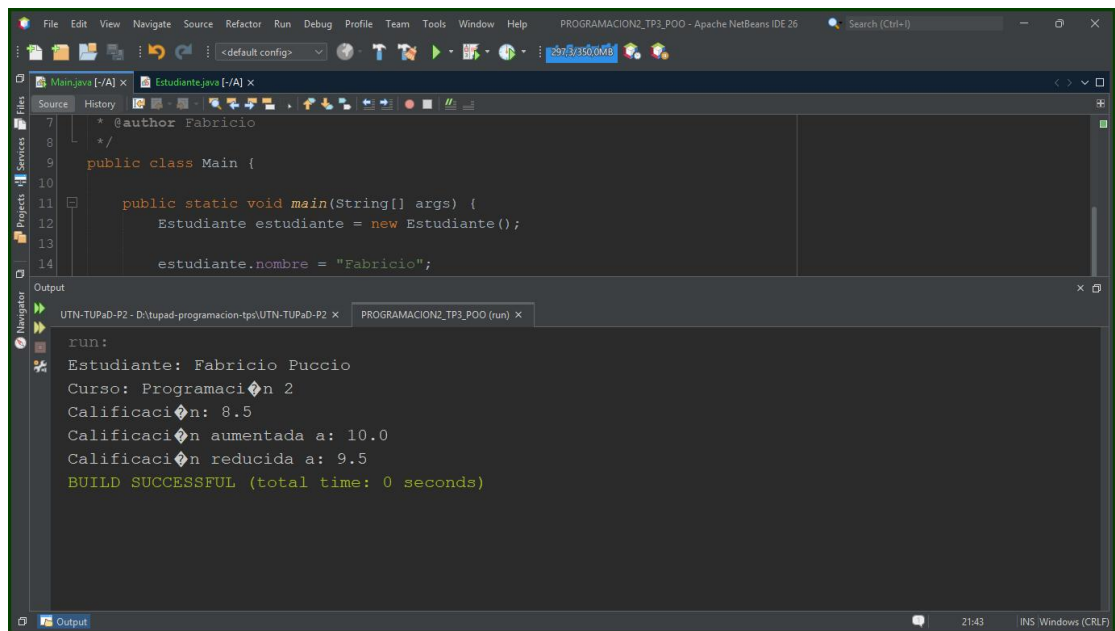
    // Método para mostrar información
    public void mostrarInfo() {
        System.out.println("Estudiante: " + nombre + " " + apellido);
        System.out.println("Curso: " + curso);
        System.out.println("Calificación: " + calificacion);
    }

    // Método para subir calificación
    public void subirCalificacion(double puntos) {
        calificacion += puntos;
        System.out.println("Calificación aumentada a: " + calificacion);
    }

    // Método para bajar calificación
    public void bajarCalificacion(double puntos) {
        calificacion -= puntos;
        System.out.println("Calificación reducida a: " + calificacion);
    }
}
```



```
7  * @author Fabricio
8  */
9  public class Main {
10
11     public static void main(String[] args) {
12         Estudiante estudiante = new Estudiante();
13
14         estudiante.nombre = "Fabricio";
15         estudiante.apellido = "Puccio";
16         estudiante.curso = "Programación 2";
17         estudiante.calificacion = 8.5;
18
19         estudiante.mostrarInfo();
20         estudiante.subirCalificacion(1.5);
21         estudiante.bajarCalificacion(0.5);
22     }
23
24 }
25
```



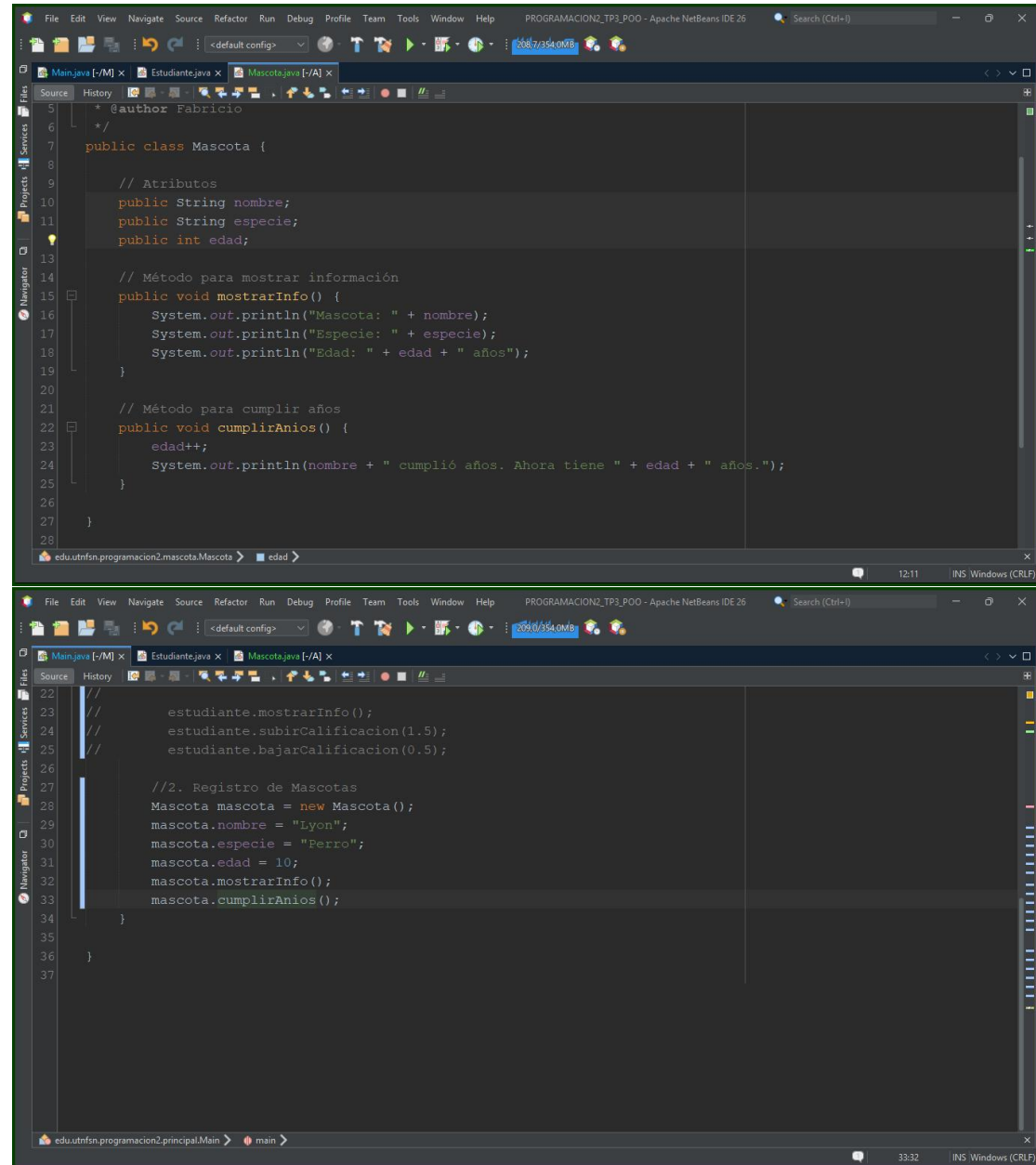
```
run:
Estudiante: Fabricio Puccio
Curso: Programaci n 2
Calificaci n: 8.5
Calificaci n aumentada a: 10.0
Calificaci n reducida a: 9.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.



The image consists of two screenshots of the Apache NetBeans IDE. The top screenshot shows the `Mascota.java` file with the following code:

```
5  * @author Fabricio
6  */
7  public class Mascota {
8
9      // Atributos
10     public String nombre;
11     public String especie;
12     public int edad;
13
14     // Método para mostrar información
15     public void mostrarInfo() {
16         System.out.println("Mascota: " + nombre);
17         System.out.println("Especie: " + especie);
18         System.out.println("Edad: " + edad + " años");
19     }
20
21     // Método para cumplir años
22     public void cumplirAnios() {
23         edad++;
24         System.out.println(nombre + " cumplió años. Ahora tiene " + edad + " años.");
25     }
26
27 }
28
```

The bottom screenshot shows the `Main.java` file with the following code:

```
22 //
23 //
24 //
25 //
26 //
27 //2. Registro de Mascotas
28 Mascota mascota = new Mascota();
29 mascota.nombre = "Lyon";
30 mascota.especie = "Perro";
31 mascota.edad = 10;
32 mascota.mostrarInfo();
33 mascota.cumplirAnios();
34 }
35
36 }
37
```

```
19 // estudiante.apellido = "Puccio";
20 // estudiante.curso = "Programación 2";
21 // estudiante.calificacion = 8.5;
22 //
23 // estudiante.mostrarInfo();
24 // estudiante.subirCalificacion(1.5);
25 // estudiante.bajarCalificacion(0.5);
26
```

```
run:
Mascota: Lyon
Especie: Perro
Edad: 10 años
Lyon cumplió años. Ahora tiene 11 años.
BUILD SUCCESSFUL (total time: 0 seconds)
```

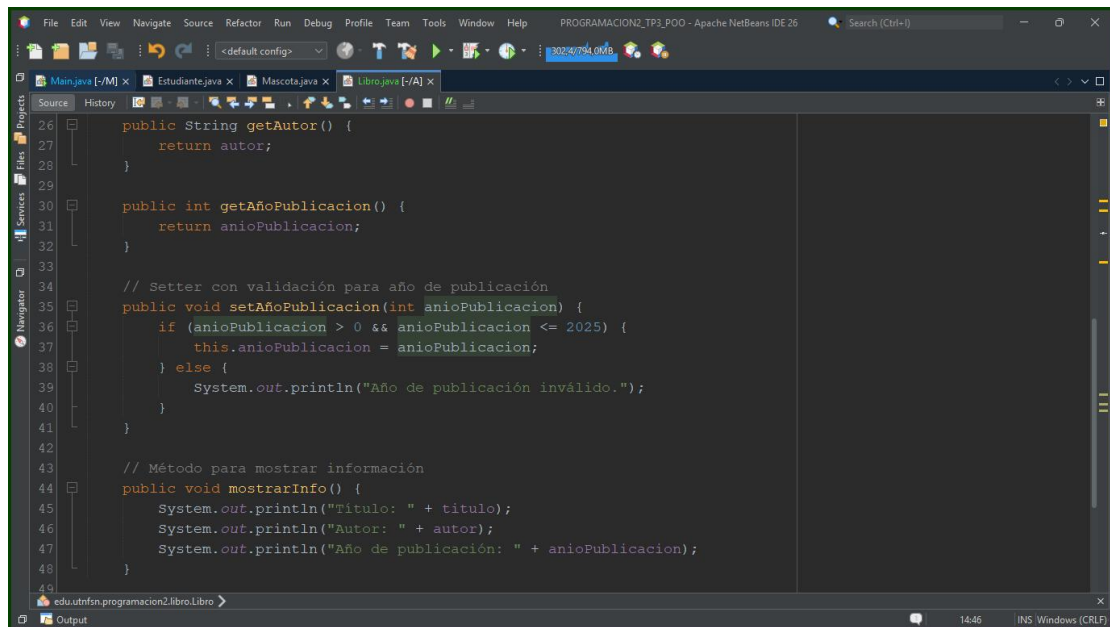
3. Encapsulamiento con la Clase Libro

a. Crear una clase Libro con atributos privados: titulo, autor, añoPublicacion.

Métodos requeridos: Getters para todos los atributos. Setter con validación para añoPublicacion.

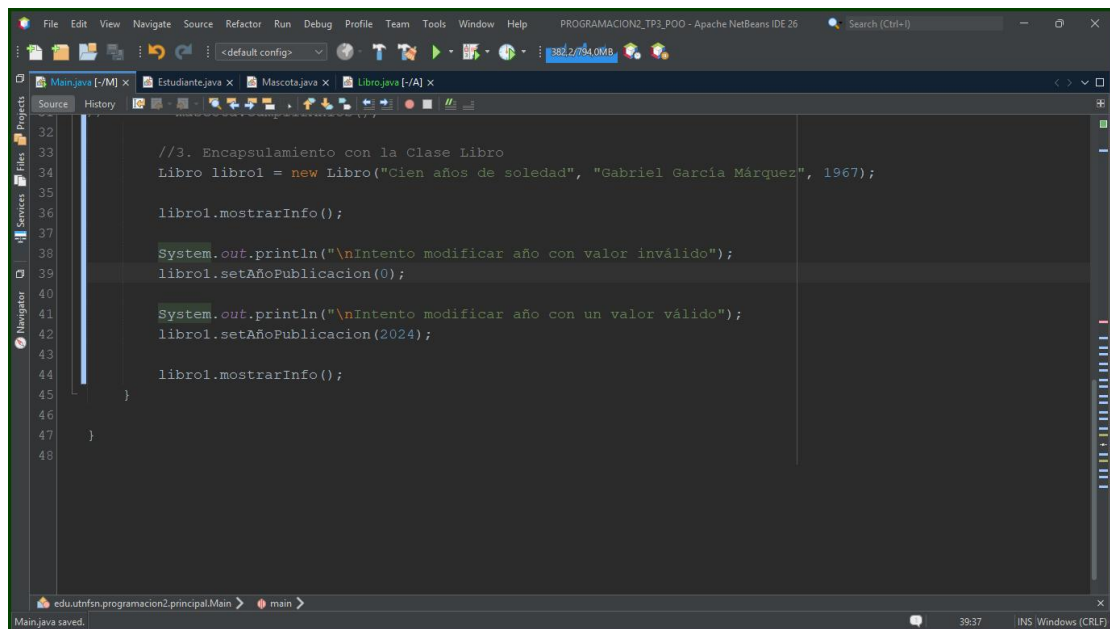
Tarea: Crear un libro, intentar modificar el año con un valor inválido y luego con uno válido, mostrar la información final.

```
1 package edu.utnfn.programacion2.libro;
2
3 /**
4  *
5  * @author Fabricio
6  */
7 public class Libro {
8
9     // Atributos privados
10    private String titulo;
11    private String autor;
12    private int añoPublicacion;
13
14    // Método Constructor, recibe parámetros para inicializar atributos
15    public Libro(String titulo, String autor, int añoPublicacion) {
16        this.titulo = titulo;
17        this.autor = autor;
18        setAñoPublicacion(añoPublicacion); // Usamos setter para validación
19    }
20
21    // Getters
22    public String getTitulo() {
23        return titulo;
24    }
25}
```



```
26 public String getAutor() {
27     return autor;
28 }
29
30 public int getAñoPublicacion() {
31     return anioPublicacion;
32 }
33
34 // Setter con validación para año de publicación
35 public void setAñoPublicacion(int anioPublicacion) {
36     if (anioPublicacion > 0 && anioPublicacion <= 2025) {
37         this.anioPublicacion = anioPublicacion;
38     } else {
39         System.out.println("Año de publicación inválido.");
40     }
41 }
42
43 // Método para mostrar información
44 public void mostrarInfo() {
45     System.out.println("Titulo: " + titulo);
46     System.out.println("Autor: " + autor);
47     System.out.println("Año de publicación: " + anioPublicacion);
48 }
49
```

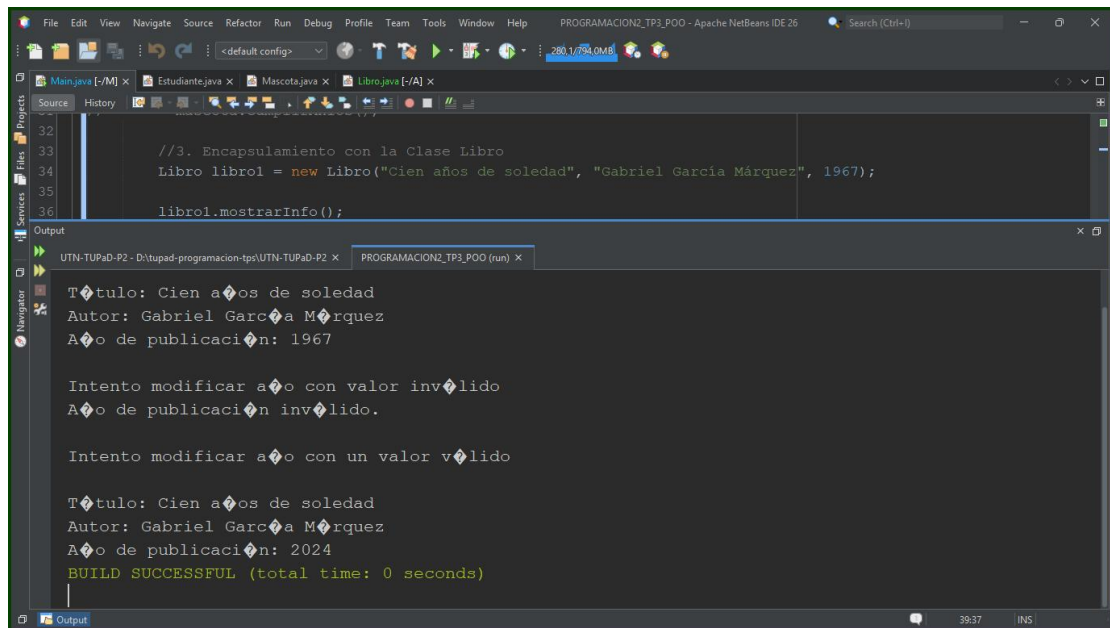
edu.utnfrsn.programacion2.libro.Libro



```
32
33 //3. Encapsulamiento con la Clase Libro
34 Libro librol = new Libro("Cien años de soledad", "Gabriel García Márquez", 1967);
35
36 librol.mostrarInfo();
37
38 System.out.println("\nIntento modificar año con valor inválido");
39 librol.setAñoPublicacion(0);
40
41 System.out.println("\nIntento modificar año con un valor válido");
42 librol.setAñoPublicacion(2024);
43
44 librol.mostrarInfo();
45 }
46
47 }
48
```

edu.utnfrsn.programacion2.principal.Main

Main.java saved.



```
//3. Encapsulamiento con la Clase Libro
Libro libro1 = new Libro("Cien años de soledad", "Gabriel García Márquez", 1967);

libro1.mostrarInfo();
```

Output:

```
Título: Cien años de soledad
Autor: Gabriel García Márquez
Año de publicación: 1967

Intento modificar año con valor inválido
Año de publicación inválido.

Intento modificar año con un valor válido

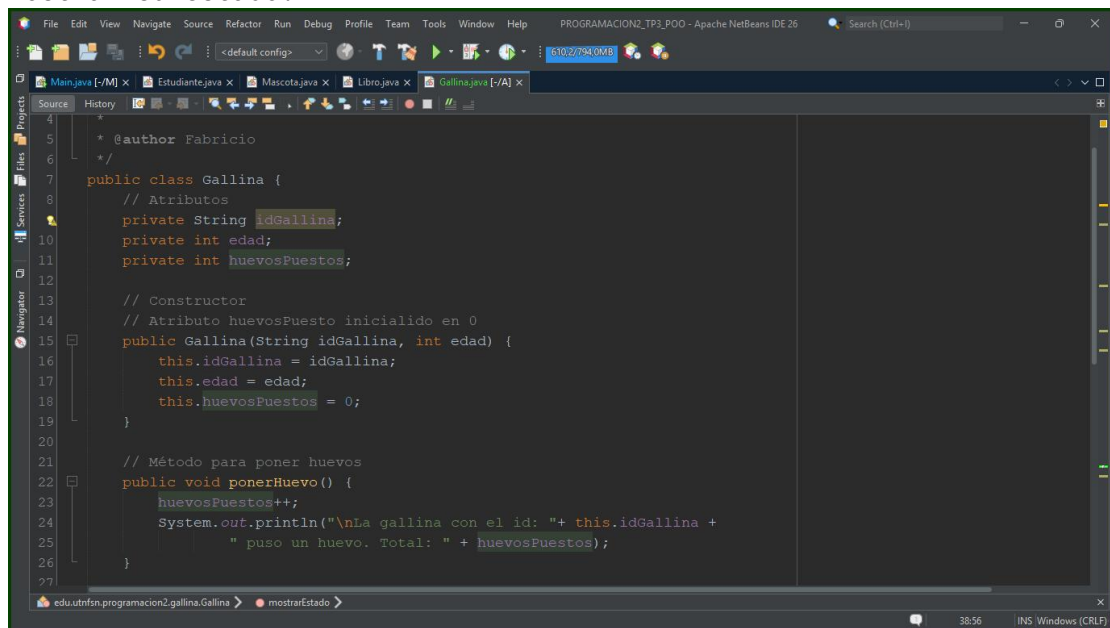
Título: Cien años de soledad
Autor: Gabriel García Márquez
Año de publicación: 2024
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.



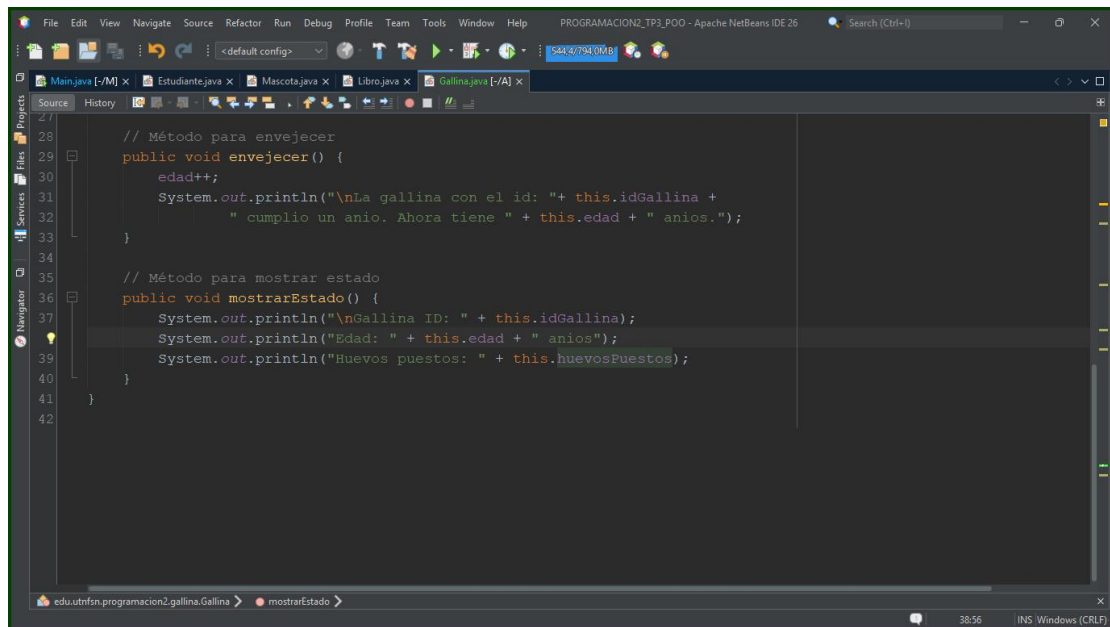
```
* @author Fabricio
*/
public class Gallina {
    // Atributos
    private String idGallina;
    private int edad;
    private int huevosPuestos;

    // Constructor
    // Atributo huevosPuesto inicializado en 0
    public Gallina(String idGallina, int edad) {
        this.idGallina = idGallina;
        this.edad = edad;
        this.huevosPuestos = 0;
    }

    // Método para poner huevos
    public void ponerHuevo() {
        huevosPuestos++;
        System.out.println("\nLa gallina con el id: " + this.idGallina +
            " puso un huevo. Total: " + huevosPuestos);
    }
}
```

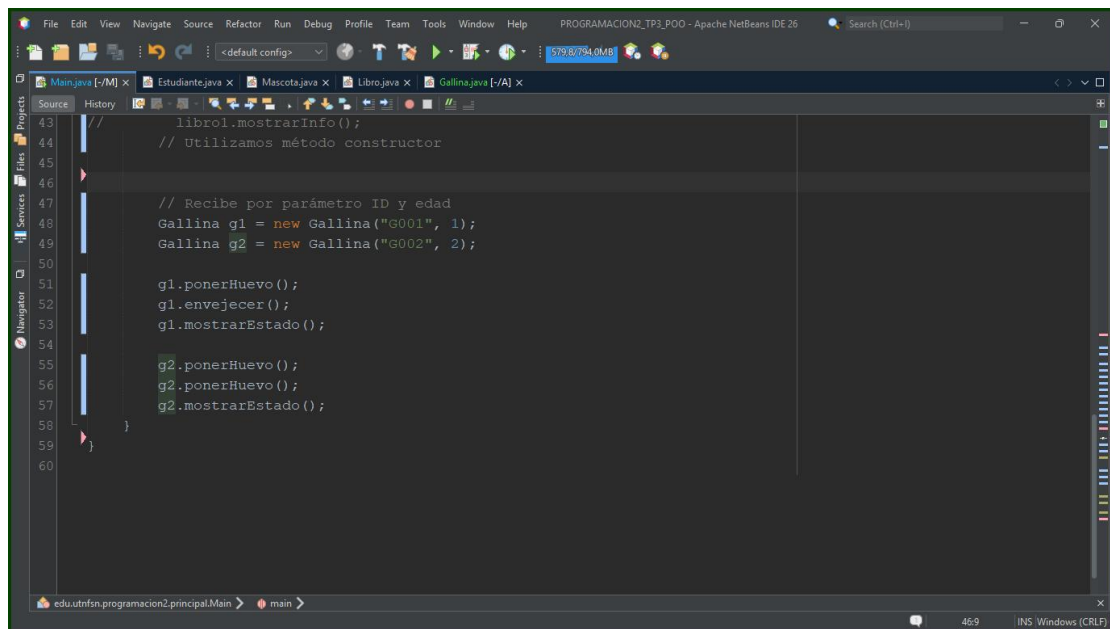
Output:

```
edu.utfrfsn.programacion2.gallina.Gallina mostrarEstado
```



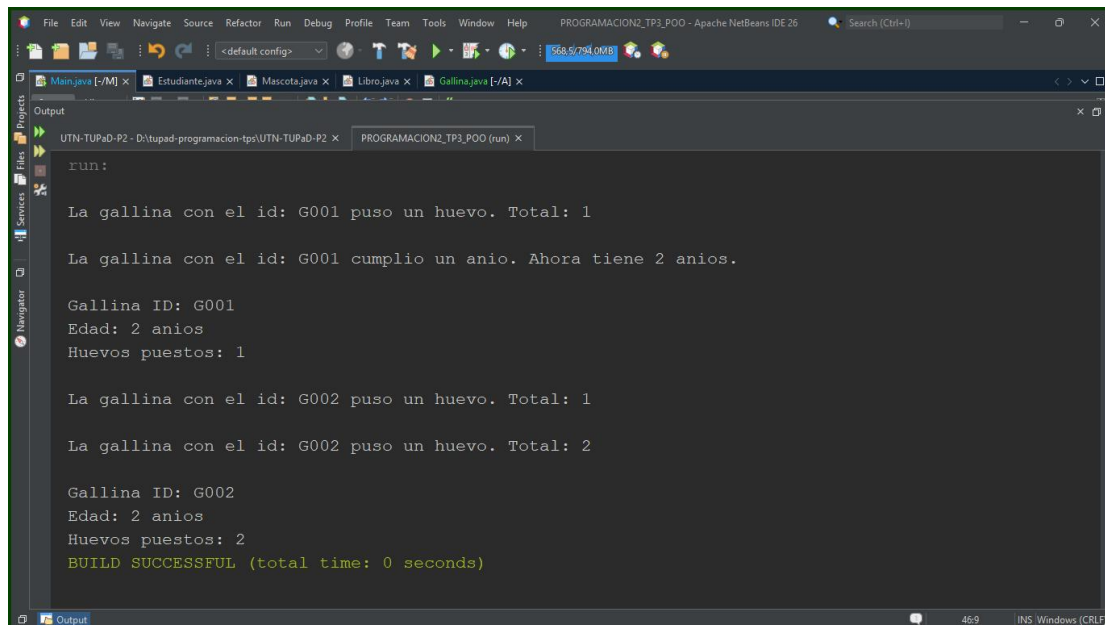
```
27
28 // Método para envejecer
29 public void envejecer() {
30     edad++;
31     System.out.println("\nLa gallina con el id: " + this.idGallina +
32         " cumple un año. Ahora tiene " + this.edad + " años.");
33 }
34
35 // Método para mostrar estado
36 public void mostrarEstado() {
37     System.out.println("\nGallina ID: " + this.idGallina);
38     System.out.println("Edad: " + this.edad + " años");
39     System.out.println("Huevos puestos: " + this.huevosPuestos);
40 }
41
42
```

edu.utnfrs.programacion2.gallina.Gallina > mostrarEstado



```
43 // librol.mostrarInfo();
44 // Utilizamos método constructor
45
46
47 // Recibe por parámetro ID y edad
48 Gallina g1 = new Gallina("G001", 1);
49 Gallina g2 = new Gallina("G002", 2);
50
51 g1.ponerHuevo();
52 g1.envejecer();
53 g1.mostrarEstado();
54
55 g2.ponerHuevo();
56 g2.ponerHuevo();
57 g2.mostrarEstado();
58 }
59
60
```

edu.utnfrs.programacion2.principal.Main > main



```
run:

La gallina con el id: G001 puso un huevo. Total: 1

La gallina con el id: G001 cumplio un anio. Ahora tiene 2 anios.

Gallina ID: G001
Edad: 2 anios
Huevos puestos: 1

La gallina con el id: G002 puso un huevo. Total: 1

La gallina con el id: G002 puso un huevo. Total: 2

Gallina ID: G002
Edad: 2 anios
Huevos puestos: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

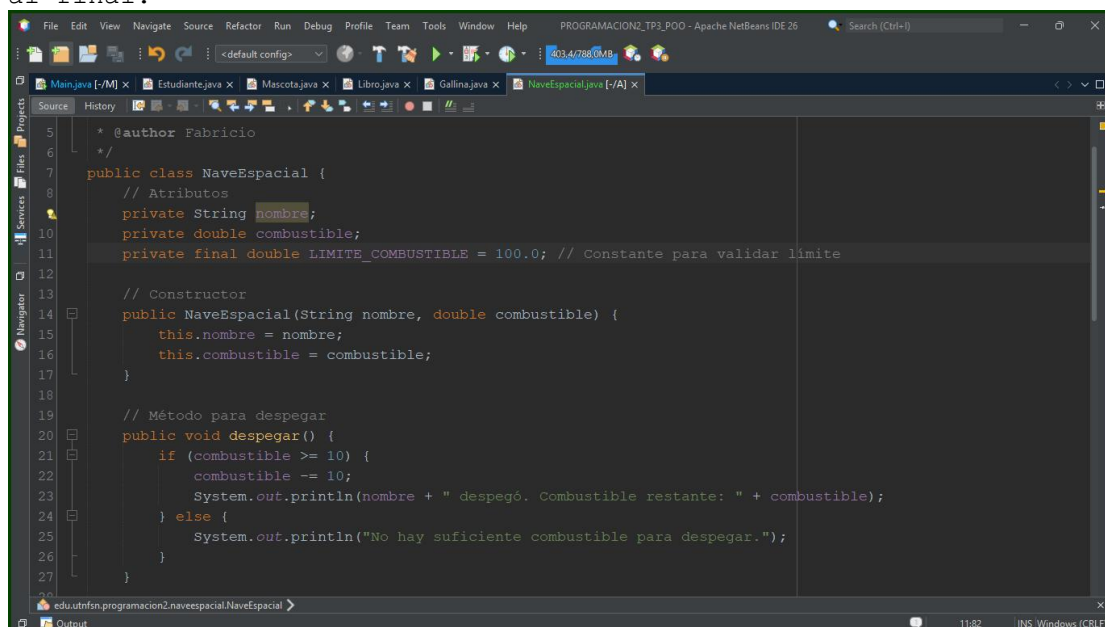
5. Simulación de Nave Espacial

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.



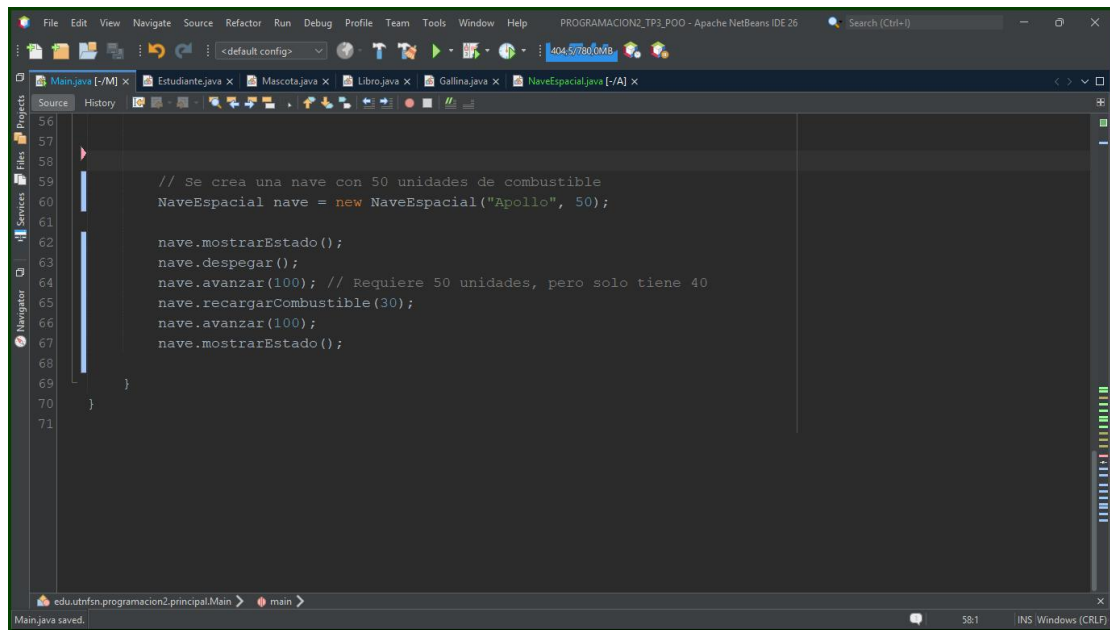
```
5  * @author Fabricio
6  */
7  public class NaveEspacial {
8      // Atributos
9      private String nombre;
10     private double combustible;
11     private final double LIMITE_COMBUSTIBLE = 100.0; // Constante para validar límite
12
13     // Constructor
14     public NaveEspacial(String nombre, double combustible) {
15         this.nombre = nombre;
16         this.combustible = combustible;
17     }
18
19     // Método para despegar
20     public void despegar() {
21         if (combustible >= 10) {
22             combustible -= 10;
23             System.out.println(nombre + " despegó. Combustible restante: " + combustible);
24         } else {
25             System.out.println("No hay suficiente combustible para despegar.");
26         }
27     }
28 }
```


This screenshot shows the NetBeans IDE with the file `NaveEspacial.java` open. The code defines two methods: `avanzar` and `recargarCombustible`. The `avanzar` method calculates the required fuel for a given distance and updates the remaining fuel. The `recargarCombustible` method checks if adding a certain amount of fuel would exceed a predefined limit.

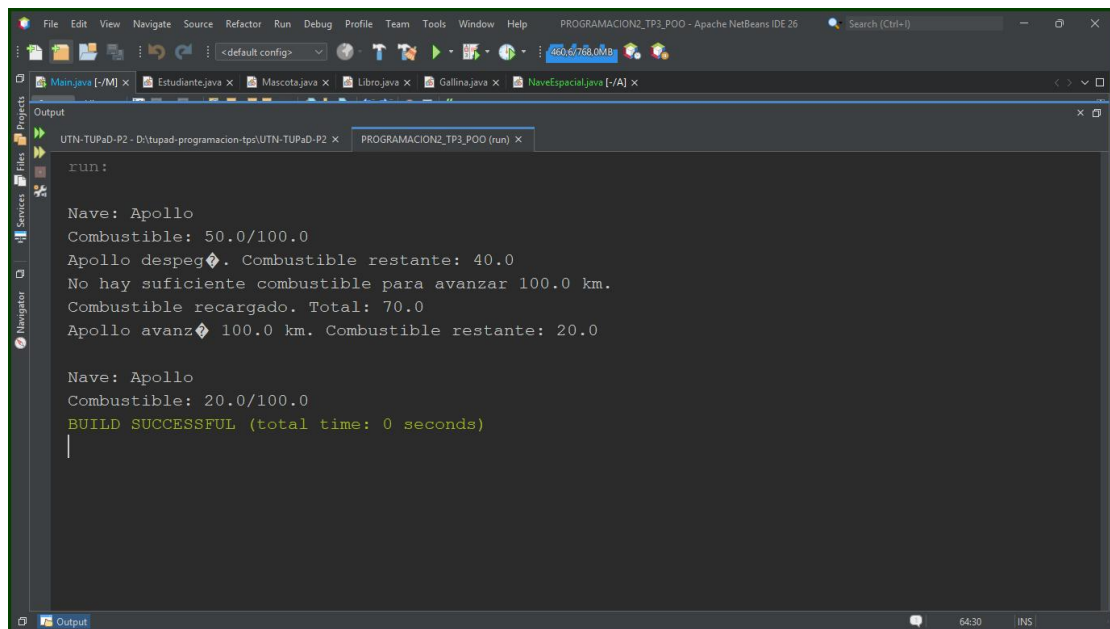
```
26 }
27
28
29 // Método para avanzar (validar combustible)
30 public void avanzar(double distancia) {
31     double combustibleNecesario = distancia * 0.5;
32     if (combustible >= combustibleNecesario) {
33         combustible -= combustibleNecesario;
34         System.out.println(nombre + " avanzó " + distancia + " km. Combustible restante: " + combustible);
35     } else {
36         System.out.println("No hay suficiente combustible para avanzar " + distancia + " km.");
37     }
38 }
39
40 // Método para recargar combustible (evitar superar límite utilizando constante para no repetir código)
41 public void recargarCombustible(double cantidad) {
42     if (combustible + cantidad <= LIMITE_COMBUSTIBLE) {
43         combustible += cantidad;
44         System.out.println("Combustible recargado. Total: " + combustible);
45     } else {
46         System.out.println("No se puede recargar, superaría el límite de " + LIMITE_COMBUSTIBLE);
47     }
48 }
49
```

This screenshot shows the continuation of the `NaveEspacial.java` file in the NetBeans IDE. It includes the `recargarCombustible` method (lines 41-48) and a new `mostrarEstado` method (lines 50-55) that prints the current state of the spacecraft, including its name and remaining fuel.

```
41 public void recargarCombustible(double cantidad) {
42     if (combustible + cantidad <= LIMITE_COMBUSTIBLE) {
43         combustible += cantidad;
44         System.out.println("Combustible recargado. Total: " + combustible);
45     } else {
46         System.out.println("No se puede recargar, superaría el límite de " + LIMITE_COMBUSTIBLE);
47     }
48 }
49
50 // Método para mostrar estado
51 public void mostrarEstado() {
52     System.out.println("\nNave: " + nombre);
53     System.out.println("Combustible: " + combustible + "/" + LIMITE_COMBUSTIBLE);
54 }
55
56
```

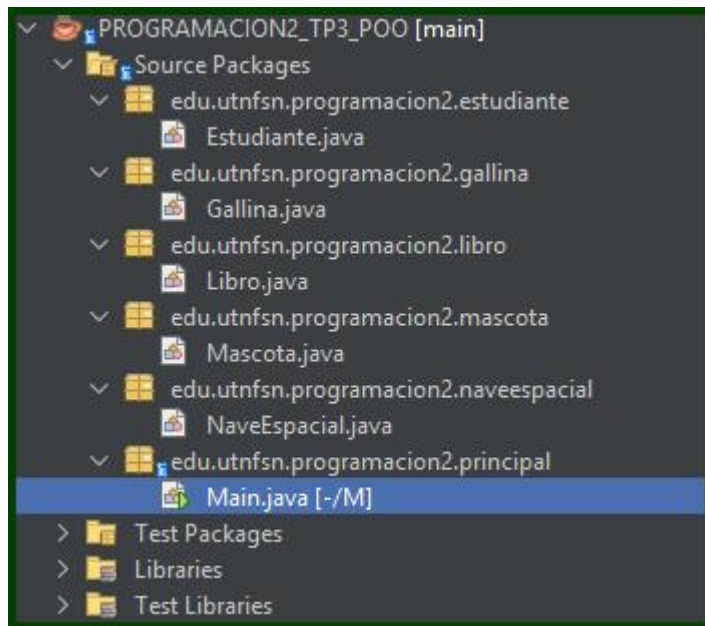


```
56
57
58
59 // Se crea una nave con 50 unidades de combustible
60 NaveEspacial nave = new NaveEspacial("Apollo", 50);
61
62 nave.mostrarEstado();
63 nave.despegar();
64 nave.avanzar(100); // Requiere 50 unidades, pero solo tiene 40
65 nave.recargarCombustible(30);
66 nave.avanzar(100);
67 nave.mostrarEstado();
68
69 }
70
71
```



```
run:
Nave: Apollo
Combustible: 50.0/100.0
Apollo despegó. Combustible restante: 40.0
No hay suficiente combustible para avanzar 100.0 km.
Combustible recargado. Total: 70.0
Apollo avanzó 100.0 km. Combustible restante: 20.0

Nave: Apollo
Combustible: 20.0/100.0
BUILD SUCCESSFUL (total time: 0 seconds)
```



Link a repositorio en GitHub: https://github.com/FabrizioPuccio/UTN-TUPaD-P2/tree/main/intro_prog_orientada_objetos/PROGRAMACION2_TP3_POO