

Tecnicatura Universitaria en Programación a Distancia

## **PROGRAMACIÓN II**

### **Trabajo Práctico 3: Introducción a POO**

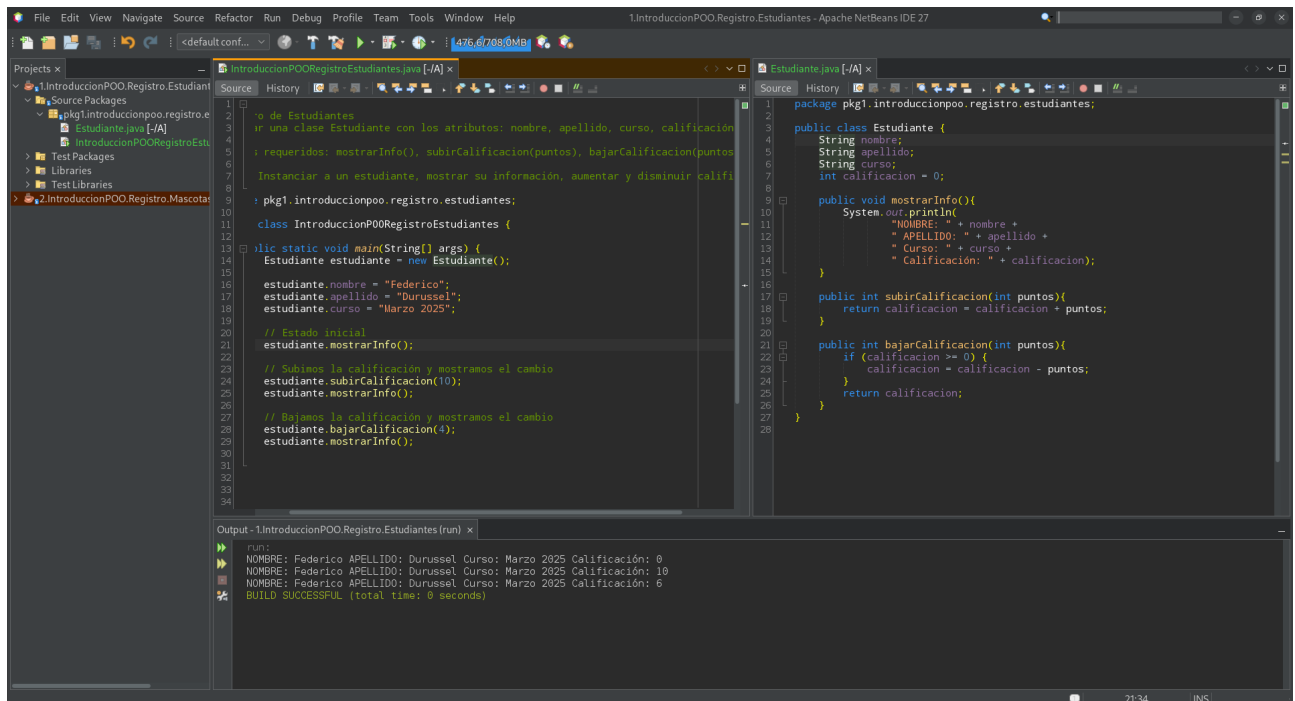
## Casos prácticos:

### 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.



The screenshot shows the Apache NetBeans IDE with two Java files open. The left file, `IntroduccionPOORegistroEstudiantes.java`, contains a `main` method that creates an `Estudiante` object and calls its methods. The right file, `Estudiante.java`, defines the `Estudiante` class with its attributes and methods.

```
1 package pkg1.introduccionpoo.registro.estudiantes;
2
3 public class Estudiante {
4     String nombre;
5     String apellido;
6     String curso;
7     int calificacion = 0;
8
9     public void mostrarInfo(){
10         System.out.println(
11             "NOMBRE: " + nombre +
12             " APELLIDO: " + apellido +
13             " Curso: " + curso +
14             " Calificación: " + calificacion);
15     }
16
17     public int subirCalificacion(int puntos){
18         return calificacion + puntos;
19     }
20
21     public int bajarCalificacion(int puntos){
22         if (calificacion >= 0) {
23             calificacion = calificacion - puntos;
24         }
25         return calificacion;
26     }
27 }
28
```

```
1 // de Estudiantes
2 // ir una clase Estudiante con los atributos: nombre, apellido, curso, calificación
3 // requeridos: mostrarInfo(), subirCalificacion(puntos), bajarCalificacion(puntos)
4 // Instanciar a un estudiante, mostrar su información, aumentar y disminuir califi
5
6
7
8
9
10 package pkg1.introduccionpoo.registro.estudiantes;
11
12 class IntroduccionPOORegistroEstudiantes {
13
14     //lic static void main(String[] args) {
15     Estudiante estudiante = new Estudiante();
16
17     estudiante.nombre = "Federico";
18     estudiante.apellido = "Durussel";
19     estudiante.curso = "Marzo 2025";
20
21     // Estado inicial
22     estudiante.mostrarInfo();
23
24     // Subimos la calificación y mostramos el cambio
25     estudiante.subirCalificacion(10);
26     estudiante.mostrarInfo();
27
28     // Bajamos la calificación y mostramos el cambio
29     estudiante.bajarCalificacion(4);
30     estudiante.mostrarInfo();
31
32
33
34 }
```

Output - 1.IntroduccionPOO.Registro.Estudiantes (run) x

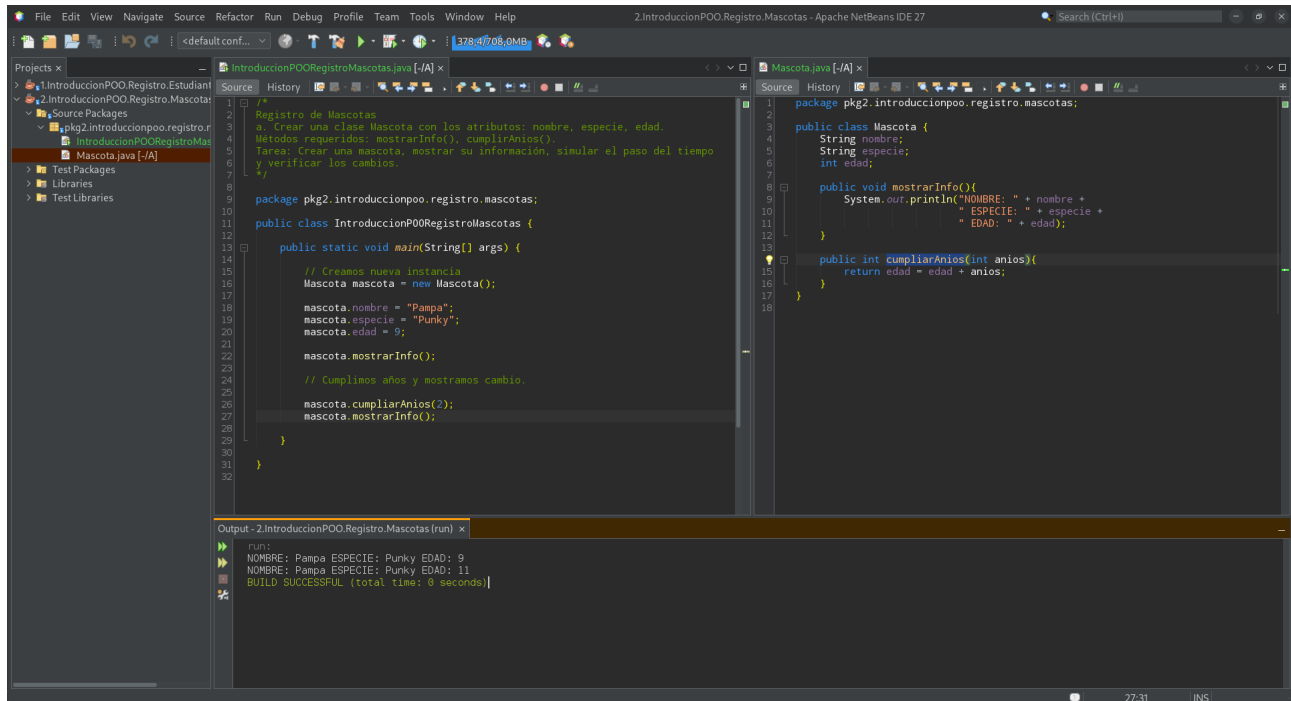
```
run:
NOMBRE: Federico APELLIDO: Durussel Curso: Marzo 2025 Calificación: 0
NOMBRE: Federico APELLIDO: Durussel Curso: Marzo 2025 Calificación: 10
NOMBRE: Federico APELLIDO: Durussel Curso: Marzo 2025 Calificación: 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 2. Registro de Mascotas

Crear una clase Mascota con los atributos: nombre, especie, edad.

Métodos requeridos: mostrarInfo(), cumplirAños().

Tarea: Crear una mascota, mostrar su información, simular el paso del tiempo y verificar los cambios.



```
1 package pkg2.introduccionpoo.registro.mascotas;
2
3 public class Mascota {
4     String nombre;
5     String especie;
6     int edad;
7
8     public void mostrarInfo(){
9         System.out.println("NOMBRE: " + nombre +
10                             " ESPECIE: " + especie +
11                             " EDAD: " + edad);
12     }
13
14     public int cumplirAños(int años){
15         return edad + edad + años;
16     }
17 }
18
```

```
1 // Registro de Mascotas
2 a. Crear una clase Mascota con los atributos: nombre, especie, edad.
3 Métodos requeridos: mostrarInfo(), cumplirAños().
4 Tarea: Crear una mascota, mostrar su información, simular el paso del tiempo
5 y verificar los cambios.
6
7
8 package pkg2.introduccionpoo.registro.mascotas;
9
10 public class IntroduccionPOORegistroMascotas {
11
12
13     public static void main(String[] args) {
14
15         // Creamos nueva instancia
16         Mascota mascota = new Mascota();
17
18         mascota.nombre = "Pampa";
19         mascota.especie = "Punky";
20         mascota.edad = 9;
21
22         mascota.mostrarInfo();
23
24         // Cumplimos años y mostramos cambio.
25         mascota.cumplirAños(2);
26         mascota.mostrarInfo();
27     }
28 }
29
30
31
32
```

Output - 2.IntroduccionPOO.Registro.Mascotas (run) x

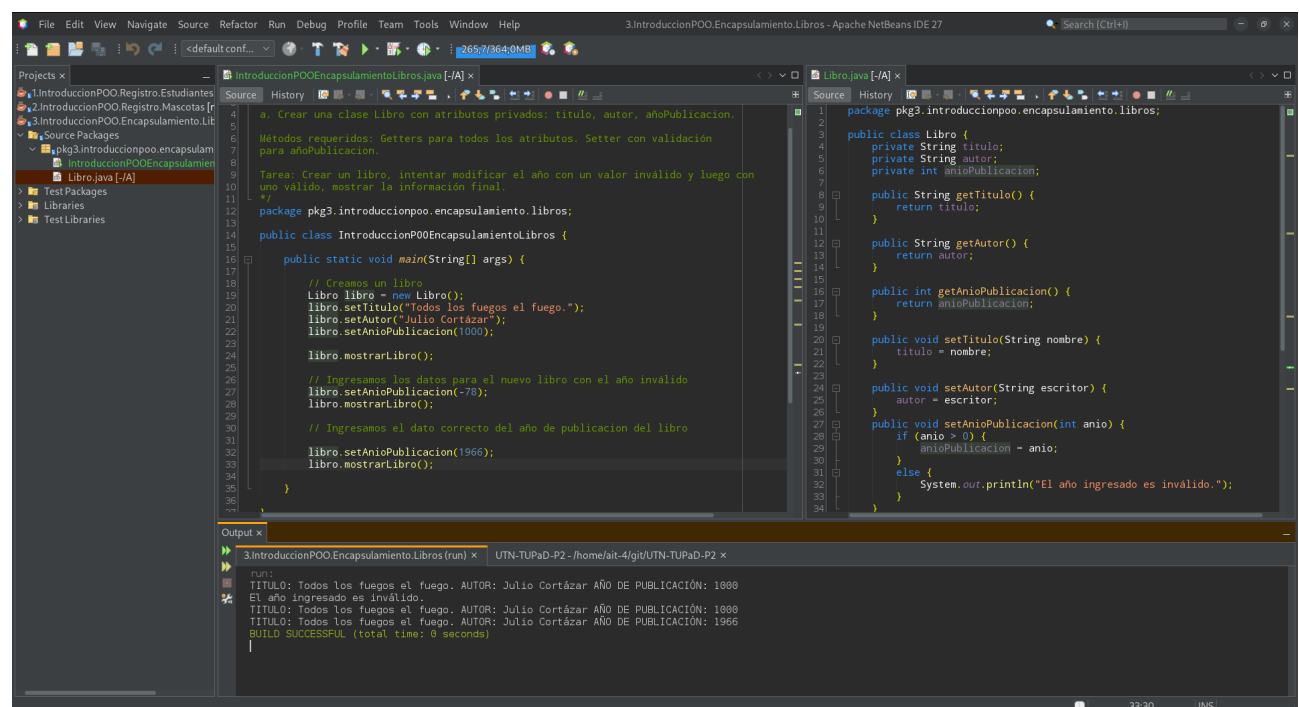
```
run:
NOMBRE: Pampa ESPECIE: Punky EDAD: 9
NOMBRE: Pampa ESPECIE: Punky EDAD: 11
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 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.



```
1 package pkg3.introduccionpoo.encapsulamiento.libros;
2
3 public class Libro {
4     private String titulo;
5     private String autor;
6     private int añoPublicacion;
7
8     public String getTitulo() {
9         return titulo;
10    }
11
12    public String getAutor() {
13        return autor;
14    }
15
16    public int getAñoPublicacion() {
17        return añoPublicacion;
18    }
19
20    public void setTitulo(String nombre) {
21        titulo = nombre;
22    }
23
24    public void setAutor(String escritor) {
25        autor = escritor;
26    }
27
28    public void setAñoPublicacion(int año) {
29        if (año > 0) {
30            añoPublicacion = año;
31        }
32        else {
33            System.out.println("El año ingresado es inválido.");
34        }
35    }
36 }
37
```

```
1 // Registro de Mascotas
2 a. Crear una clase Libro con atributos privados: titulo, autor, añoPublicacion.
3 Métodos requeridos: Getters para todos los atributos. Setter con validación
4 para añoPublicacion.
5 Tarea: Crear un libro, intentar modificar el año con un valor inválido y luego con
6 uno válido, mostrar la información final.
7
8 package pkg3.introduccionpoo.encapsulamiento.libros;
9
10 public class IntroduccionPOOEncapsulamientoLibros {
11
12     public static void main(String[] args) {
13
14         // Creamos un libro
15         Libro libro = new Libro();
16         libro.setTitulo("Todos los fuegos el fuego.");
17         libro.setAutor("Julio Cortázar");
18         libro.setAñoPublicacion(1000);
19
20         libro.mostrarLibro();
21
22         // Ingresamos los datos para el nuevo libro con el año inválido
23         libro.setAñoPublicacion(-78);
24         libro.mostrarLibro();
25
26         // Ingresamos el dato correcto del año de publicación del libro
27         libro.setAñoPublicacion(1966);
28         libro.mostrarLibro();
29     }
30 }
31
32
33
34
35
36
37
```

Output x

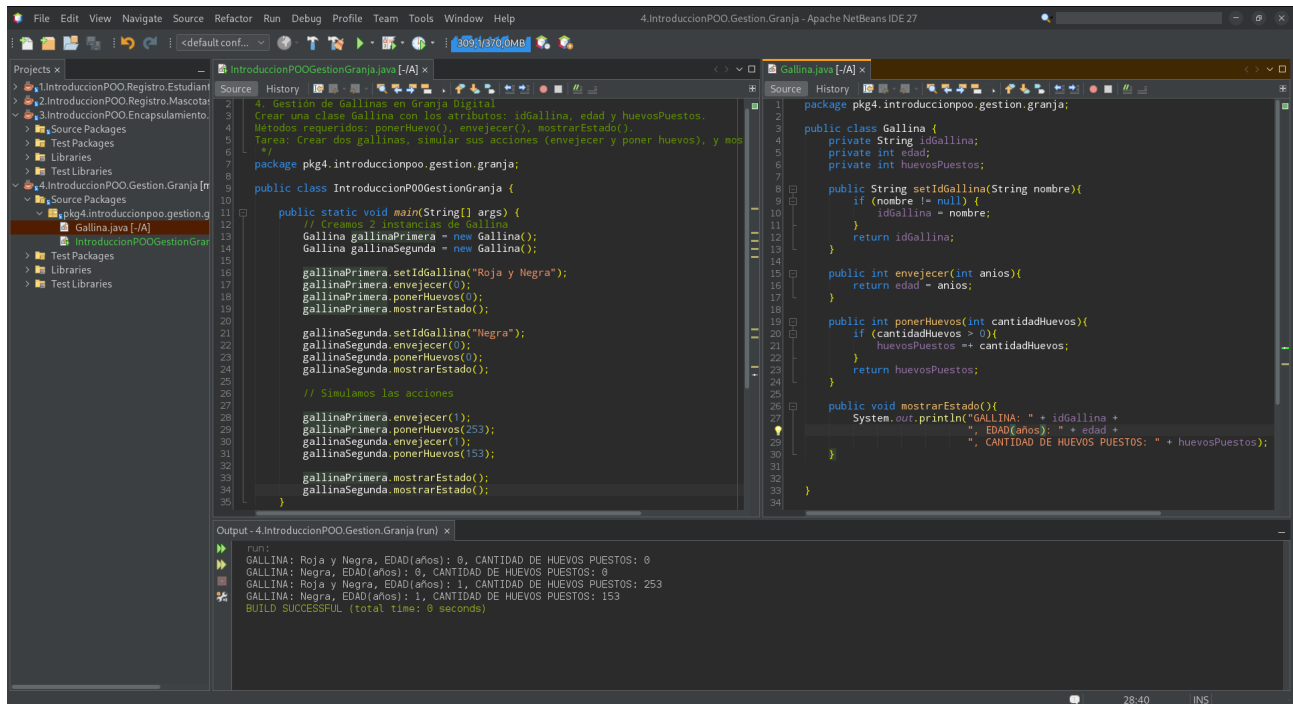
```
3.IntroduccionPOO.Encapsulamiento.Libros (run) x
UTN-TUPaD-P2 - /home/ait-4/git/UTN-TUPaD-P2 x
run:
TITULO: Todos los fuegos el fuego. AUTOR: Julio Cortázar AÑO DE PUBLICACIÓN: 1000
El año ingresado es inválido.
TITULO: Todos los fuegos el fuego. AUTOR: Julio Cortázar AÑO DE PUBLICACIÓN: 1000
TITULO: Todos los fuegos el fuego. AUTOR: Julio Cortázar AÑO DE PUBLICACIÓN: 1966
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.



```
4.IntroduccionPOO.Gestion.Granja - Apache NetBeans IDE 27

Source History
package pkg4.introduccionpoo.gestion.granja;

public class IntroduccionPOOGestionGranja {

    public static void main(String[] args) {
        // Creamos las instancias de gallinas
        Gallina gallinaPrimera = new Gallina();
        Gallina gallinaSegunda = new Gallina();

        gallinaPrimera setIdGallina("Roja y Negra");
        gallinaPrimera.envejecer(0);
        gallinaPrimera.ponerHuevos(0);
        gallinaPrimera.mostrarEstado();

        gallinaSegunda.setIdGallina("Negra");
        gallinaSegunda.envejecer(0);
        gallinaSegunda.ponerHuevos(0);
        gallinaSegunda.mostrarEstado();

        // Simulamos las acciones

        gallinaPrimera.envejecer(1);
        gallinaPrimera.ponerHuevos(253);
        gallinaSegunda.envejecer(1);
        gallinaSegunda.ponerHuevos(153);

        gallinaPrimera.mostrarEstado();
        gallinaSegunda.mostrarEstado();
    }
}

Output - 4.IntroduccionPOO.Gestion.Granja (run) x
run:
GALLINA: Roja y Negra, EDAD(años): 0, CANTIDAD DE HUEVOS PUESTOS: 0
GALLINA: Negra, EDAD(años): 0, CANTIDAD DE HUEVOS PUESTOS: 0
GALLINA: Roja y Negra, EDAD(años): 1, CANTIDAD DE HUEVOS PUESTOS: 253
GALLINA: Negra, EDAD(años): 1, CANTIDAD DE HUEVOS PUESTOS: 153
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
package pkg4.introduccionpoo.gestion.granja;

public class Gallina {
    private String idgallina;
    private int edad;
    private int huevosPuestos;

    public String setIdGallina(String nombre){
        if (nombre != null){
            idgallina = nombre;
        }
        return idgallina;
    }

    public int envejecer(int anios){
        return edad = anios;
    }

    public int ponerHuevos(int cantidadHuevos){
        if (cantidadHuevos > 0){
            huevosPuestos += cantidadHuevos;
        }
        return huevosPuestos;
    }

    public void mostrarEstado(){
        System.out.println("GALLINA: " + idgallina +
            ", EDAD(años): " + edad +
            ", CANTIDAD DE HUEVOS PUESTOS: " + huevosPuestos);
    }
}
```

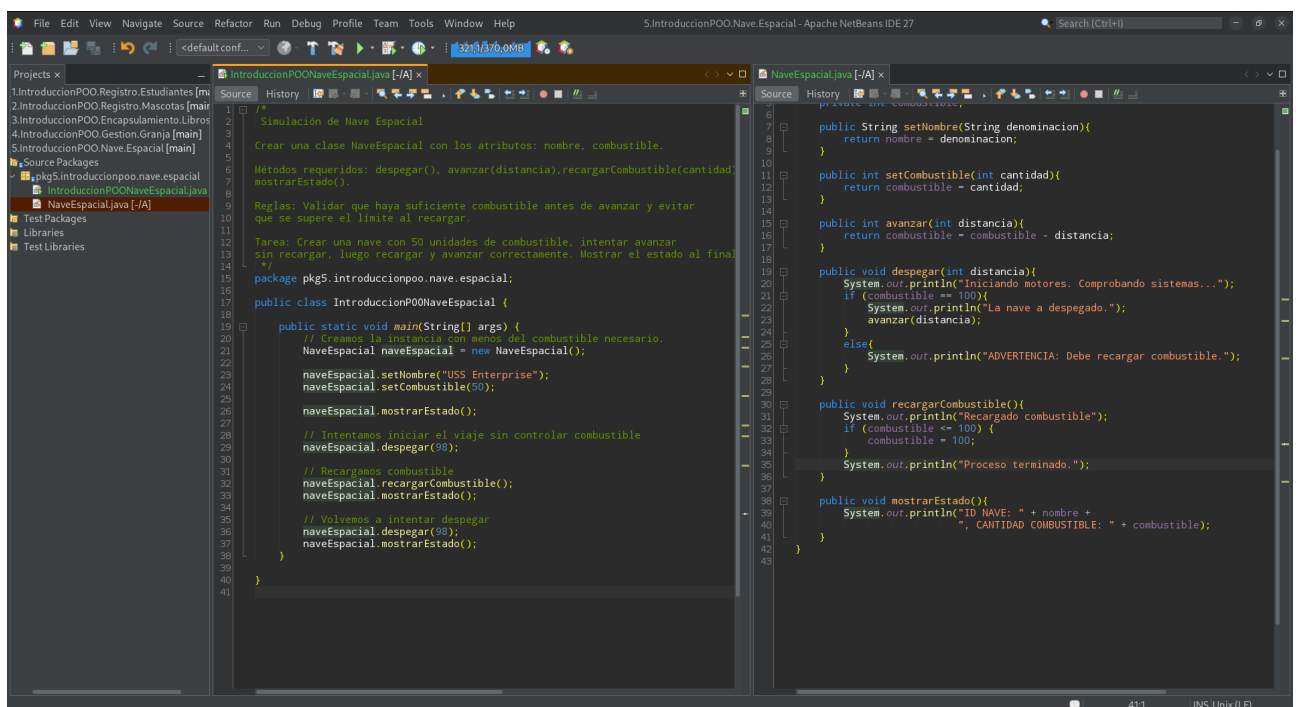
## 5. Simulación de Nave Espacial

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.IntroduccionPOO.Nave.Espacial - Apache NetBeans IDE 27

Source History
package pkg5.introduccionpoo.nave.espacial;

public class IntroduccionPOONaveEspacial {

    public static void main(String[] args) {
        // Creamos la instancia de la nave del combustible necesario.
        NaveEspacial naveEspacial = new NaveEspacial();

        naveEspacial.setNombre("USS Enterprise");
        naveEspacial.setCombustible(50);

        naveEspacial.mostrarEstado();

        // Intentamos iniciar el viaje sin controlar combustible
        naveEspacial.despegar(90);

        // Recargamos combustible
        naveEspacial.recargarCombustible();
        naveEspacial.mostrarEstado();

        // Volvemos a intentar despegar
        naveEspacial.despegar(90);
        naveEspacial.mostrarEstado();
    }
}

Source History
package pkg5.introduccionpoo.nave.espacial;

private int combustible;

public String setNombre(String denominacion){
    return nombre = denominacion;
}

public int setCombustible(int cantidad){
    return combustible = cantidad;
}

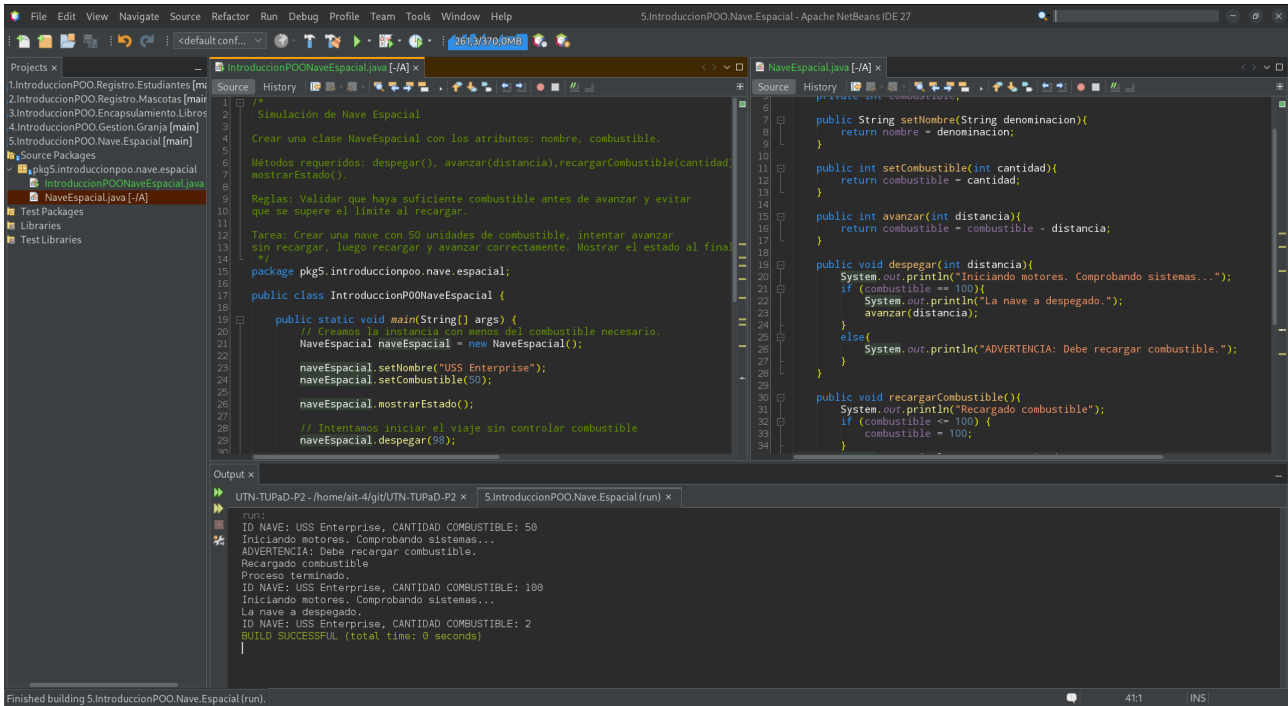
public int avanzar(int distancia){
    return combustible = combustible - distancia;
}

public void despegar(int distancia){
    System.out.println("Iniciando motores. Comprobando sistemas...");
    if (combustible == 100){
        System.out.println("La nave a despegado.");
        avanzar(distancia);
    } else {
        System.out.println("ADVERTENCIA: Debe recargar combustible.");
    }
}

public void recargarCombustible(){
    System.out.println("Recargado combustible");
    if (combustible <= 100) {
        combustible = 100;
    }
    System.out.println("Proceso terminado.");
}

public void mostrarEstado(){
    System.out.println("ID NAVE: " + nombre +
        ", CANTIDAD COMBUSTIBLE: " + combustible);
}
}
```

## Ejecución:



The screenshot displays the Apache NetBeans IDE environment. The **Projects** window on the left shows the project structure, with `IntroduccionPOO.Nave.Espacial` selected. The **Source** window is split into two panes. The left pane shows `IntroduccionPOONaveEspacial.java` with a `main` method that creates a `NaveEspacial` object, sets its name to "USS Enterprise", and calls `despegar()`. The right pane shows `NaveEspacial.java` with methods for `setNombre`, `setCombustible`, `avanzar`, `despegar`, and `recargarCombustible`. The **Output** window at the bottom shows the execution results, including the initial state of the ship, the refueling process, and the final state after refueling.

```
1 // Simulación de Nave Espacial
2
3 Crear una clase NaveEspacial con los atributos: nombre, combustible.
4
5
6 Metodos requeridos: despegar(), avanzar(distancia), recargarCombustible(cantidad,
7 mostrarEstado()).
8
9 Reglas: Validar que haya suficiente combustible antes de avanzar y evitar
10 que se supere el limite al recargar.
11
12 Tarea: Crear una nave con 50 unidades de combustible, intentar avanzar
13 sin recargar, luego recargar y avanzar correctamente. Mostrar el estado al final.
14 */
15 package pkg5.introduccionpoo.nave.espacial;
16
17 public class IntroduccionPOONaveEspacial {
18
19     public static void main(String[] args) {
20         // Creamos la instancia con menos del combustible necesario.
21         NaveEspacial naveEspacial = new NaveEspacial();
22
23         naveEspacial.setNombre("USS Enterprise");
24         naveEspacial.setCombustible(50);
25
26         naveEspacial.mostrarEstado();
27
28         // Intentamos iniciar el viaje sin controlar combustible
29         naveEspacial.despegar(90);
30     }
31 }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
run:
ID NAVE: USS Enterprise, CANTIDAD COMBUSTIBLE: 50
Iniciando motores. Comprobando sistemas...
ADVERTENCIA: Debe recargar combustible.
Recargado combustible
Proceso terminado.
ID NAVE: USS Enterprise, CANTIDAD COMBUSTIBLE: 100
Iniciando motores. Comprobando sistemas...
La nave a despegado.
ID NAVE: USS Enterprise, CANTIDAD COMBUSTIBLE: 2
BUILD SUCCESSFUL (total time: 0 seconds)
```