

## PROGRAMACIÓN II

### Trabajo Práctico 3: Introducción a la Programación Orientada a Objetos

Alumna: García Cintia.

#### 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                     |

#### Caso Práctico

Desarrollar en Java los siguientes ejercicios aplicando los conceptos de

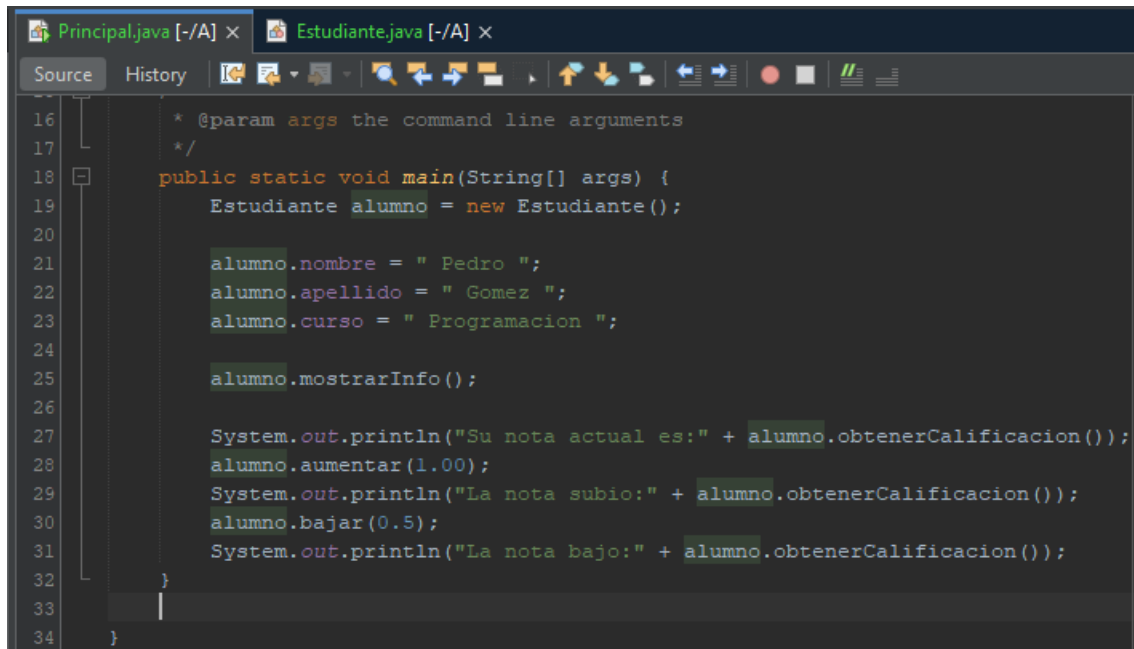
Programación orientada a objetos:

##### 1. Registro de Estudiantes

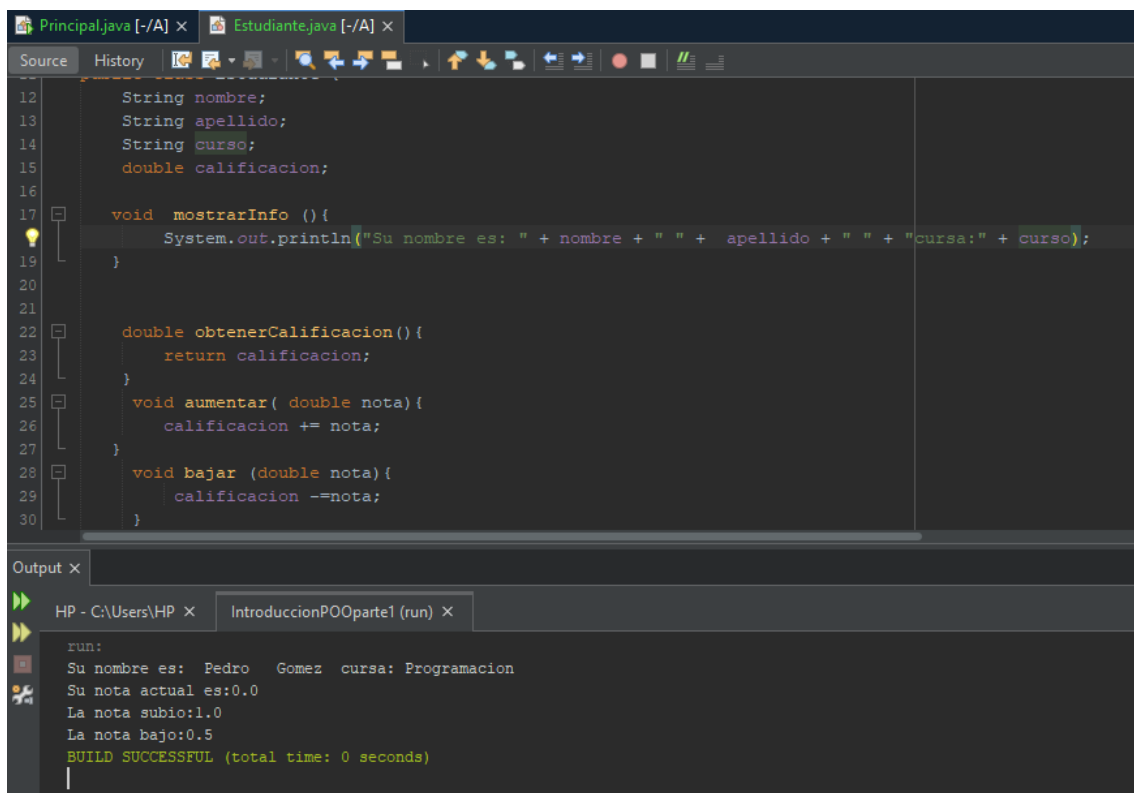
a. Crear una clase Estudiante con los atributos: nombre, apellido, curso, calificación.

Métodos requeridos: mostrarInfo(), subirCalificacion(puntos), bajarCalificacion(puntos).

Tarea: Instanciar a un estudiante, mostrar su información, aumentar y disminuir calificaciones.



```
16      * @param args the command line arguments
17      */
18      public static void main(String[] args) {
19          Estudiante alumno = new Estudiante();
20
21          alumno.nombre = " Pedro ";
22          alumno.apellido = " Gomez ";
23          alumno.curso = " Programacion ";
24
25          alumno.mostrarInfo();
26
27          System.out.println("Su nota actual es:" + alumno.obtenerCalificacion());
28          alumno.aumentar(1.00);
29          System.out.println("La nota subio:" + alumno.obtenerCalificacion());
30          alumno.bajar(0.5);
31          System.out.println("La nota bajo:" + alumno.obtenerCalificacion());
32      }
33
34  }
```



```
12      String nombre;
13      String apellido;
14      String curso;
15      double calificacion;
16
17      void mostrarInfo () {
18          System.out.println("Su nombre es: " + nombre + " " + apellido + " " + "curso:" + curso);
19      }
20
21
22      double obtenerCalificacion() {
23          return calificacion;
24      }
25
26      void aumentar( double nota) {
27          calificacion += nota;
28      }
29
30      void bajar (double nota) {
31          calificacion -=nota;
32      }
33  }
```

Output x

HP - C:\Users\HP x IntroduccionPOOparte1 (run) x

```
run:
Su nombre es: Pedro Gomez curso: Programacion
Su nota actual es:0.0
La nota subio:1.0
La nota bajo:0.5
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.

```

    /**
    public class Ejercicio2 {

        /**
        * @param args the command line arguments
        */
        public static void main(String[] args) {
            Mascota mascota = new Mascota();

            mascota.nombre = "Pipo";
            mascota.especie = "Perro";
            mascota.edad = 15;

            mascota.mostrarInfo();
            System.out.println("");
            mascota.cumplirAnios();
            System.out.println(" Su mascota cumplio un anio mas ");
        }
    }

```

HP - C:\Users\HP x Ejercicio2 (run) x

```

run:
Su nombre es : Pipo pertenece a la especie: Perro tiene 15 anios

Su nombre es : Pipo pertenece a la especie: Perro tiene 16 anios
Su mascota cumplio un anio mas
BUILD SUCCESSFUL (total time: 0 seconds)

```

```

Ejercicio2.java [-/A] x Mascota.java [-/A] x
source History
Tarea: Crear una mascota, mostrar su información, simular el paso del tiempo y
verificar los cambios.*/
public class Mascota {
    String nombre;
    String especie;
    int edad;

    void mostrarInfo(){
        System.out.println("Su nombre es : " + nombre + " pertenece a la especie: " + especie + " tiene " +
    }

    void cumplirAnios(){
        edad ++ ;
        System.out.println("Su nombre es : " + nombre + " pertenece a la especie: " + especie + " tiene " + e
    }
}

```

### 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.

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

```
*/  
public class Ejercicio3 {  
  
    /**  
     * @param args the command line arguments  
     */  
    public static void main(String[] args) {  
        Libro libro = new Libro ();  
  
        libro.setAutor("Florencia Bonelli");  
        libro.setTitulo("Caballos de Fuego");  
        libro.setAnioPublicacion(2000);  
  
        libro.mostrarInfo();  
        libro.setAnioPublicacion(2025);  
    }  
}
```

```
run:  
Titulo: Caballos de Fuego  
Autor: Florencia Bonelli  
Anio de publicacion: 2000  
BUILD SUCCESSFUL (total time: 0 seconds)
```

```

private String autor;
private int anioPublicacion;

public String getTitulo(){
    return titulo;
}

public String getAutor(){
    return autor;
}

public int getAnioPublicacion(){
    return anioPublicacion;
}

public void setTitulo(String nuevoTitulo) {
    titulo = nuevoTitulo;
}

public void setAutor(String nuevoAutor) {
    autor = nuevoAutor;
}

public void setAnioPublicacion(int anioNuevo){
    if (anioNuevo >= 1990 && anioNuevo <=2015){
        anioPublicacion = anioNuevo;
    }else {
        System.out.println("Anio Invalido, debe estar entre 1990 y 2015.");
    }
}

```

```

* @author HP
*/
public class Libro {
    private String titulo;
    private String autor;
    private int anioPublicacion;

    public String getTitulo(){
        return titulo;
    }
}

```

run:

```

Titulo: Caballos de Fuego
Autor: Florencia Bonelli
Anio de publicacion: 2000
Anio Invalido, debe estar entre 1990 y 2015.
BUILD SUCCESSFUL (total time: 0 seconds)

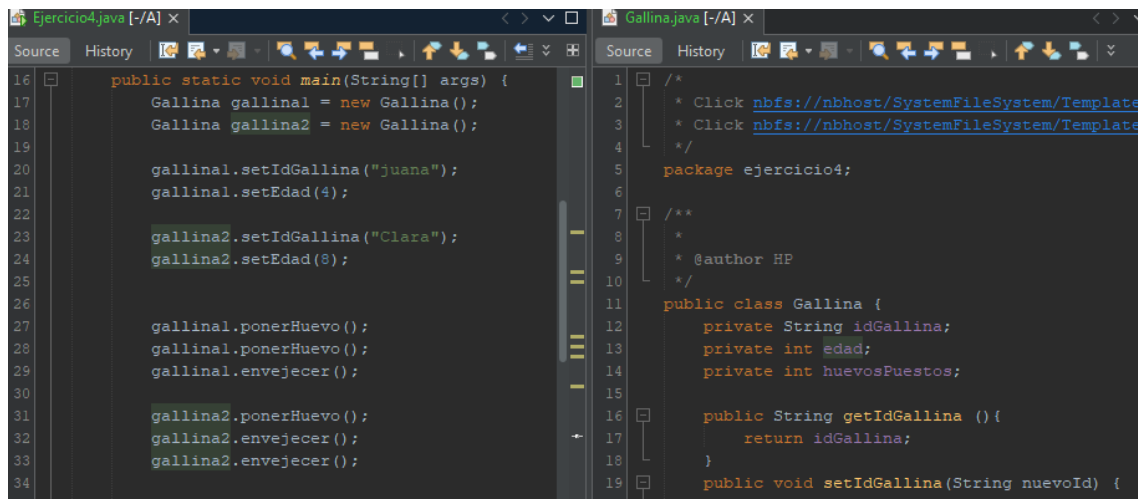
```

#### 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.



```
public static void main(String[] args) {
    Gallina gallina1 = new Gallina();
    Gallina gallina2 = new Gallina();

    gallina1.setIdGallina("juana");
    gallina1.setEdad(4);

    gallina2.setIdGallina("Clara");
    gallina2.setEdad(8);

    gallina1.ponerHuevo();
    gallina1.ponerHuevo();
    gallina1.envejecer();

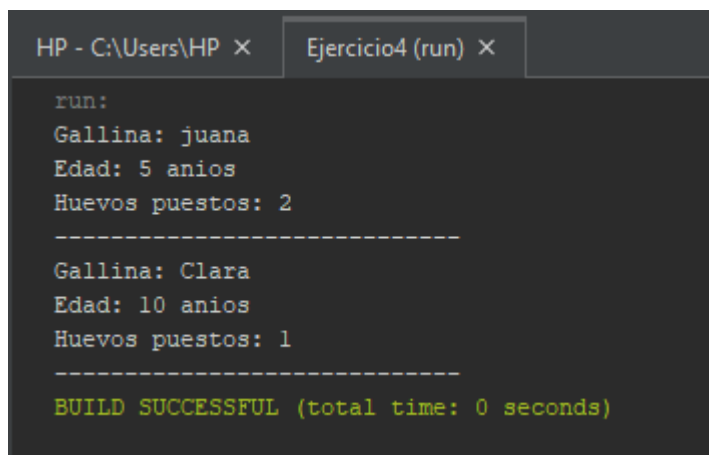
    gallina2.ponerHuevo();
    gallina2.envejecer();
    gallina2.envejecer();
}

/**
 * Click nbfs://nbhost/SystemFileSystem/Template
 * Click nbfs://nbhost/SystemFileSystem/Template
 */
package ejercicio4;

/**
 * @author HP
 */
public class Gallina {
    private String idGallina;
    private int edad;
    private int huevosPuestos;

    public String getIdGallina () {
        return idGallina;
    }

    public void setIdGallina(String nuevoId) {
```



```
HP - C:\Users\HP x Ejercicio4 (run) x
run:
Gallina: juana
Edad: 5 años
Huevos puestos: 2
-----
Gallina: Clara
Edad: 10 años
Huevos puestos: 1
-----
BUILD SUCCESSFUL (total time: 0 seconds)
```

Copio este código porque es demasiado largo para enviar print de pantalla

```
public static void main(String[] args) {

    Gallina gallina1 = new Gallina();

    Gallina gallina2 = new Gallina();

    gallina1.setIdGallina("juana");
```

```
gallina1.setEdad(4);
```

```
gallina2.setIdGallina("Clara");
```

```
gallina2.setEdad(8);
```

```
gallina1.ponerHuevo();
```

```
gallina1.ponerHuevo();
```

```
gallina1.envejecer();
```

```
gallina2.ponerHuevo();
```

```
gallina2.envejecer();
```

```
gallina2.envejecer();
```

```
// Mostrar estado final
```

```
gallina1.mostrarEstado();
```

```
gallina2.mostrarEstado();
```

```
}
```

```
}
```

```
public class Gallina {
```

```
    private String idGallina;
```

```
    private int edad;
```

```
    private int huevosPuestos;
```

```
    public String getIdGallina () {
```

```
        return idGallina;
```

```
}
```

```
public void setIdGallina(String nuevold) {  
    idGallina = nuevold;  
}  
  
public int getEdad () {  
    return edad;  
}  
  
public int getHuevosPuestos(){  
    return huevosPuestos;  
}  
  
public void setEdad(int nuevaEdad){  
    if (nuevaEdad >= 0){  
        edad = nuevaEdad;  
    } else {  
        System.out.println("Edad invalida");  
    }  
}  
  
public void ponerHuevo(){  
    huevosPuestos++;  
}  
  
public void envejecer(){  
    edad++;  
}  
  
public void mostrarEstado() {  
    System.out.println("Gallina: " + idGallina);
```



```

        System.out.println("Edad: " + edad + " años");

        System.out.println("Huevos puestos: " + huevosPuestos);

        System.out.println("-----");

    }
}

```

## 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.

```

package Principal;

/**
 *
 * @author HP
 */
public class NaveEspacial {

    private String nombre;

    private int combustible;

    private boolean despegue;

    public NaveEspacial(){

```

```
    combustible=50;

    despegue= false;
}

public void setNombre(String nuevoNombre) {

    nombre = nuevoNombre;
}

public void despegar(){

    despegue = true;

    System.out.println( nombre + " despego!");
}

public void avanzar(int distancia){

    if (despegue){

        if (combustible >= distancia){

            System.out.println(nombre + "avanza "+ distancia + "km");

            combustible =distancia;

        }else {System.out.println("No hay suficiente combustible");

        }

    }else {

        System.out.println("La nave "+ nombre + " aun no despego");

    }

}

public void recargarCombustible(int cantidad){

    if (cantidad + combustible <= 100){

        combustible += cantidad;
```

```

        System.out.println("Recarga " + cantidad + "litros de combustible completa");
    } else {
        System.out.println("La cantidad que desea cargar excede el limite maximo");
    }
}

public void mostrarEstado() {
    System.out.println("Nombre: " + nombre);
    System.out.println("Combustible: " + combustible + " unidades");
}

}

public static void main(String[] args) {
    NaveEspacial nave = new NaveEspacial();

    nave.setNombre("NaveUno");
    nave.mostrarEstado();
    nave.avanzar(20);
    nave.despegar();
    nave.avanzar(30);
    nave.recargarCombustible(40);
    nave.mostrarEstado();
}
}

```

```

public class NaveEspacial {
    private String nombre;
    private int combustible;
    private boolean despegue;

    public NaveEspacial() {
        combustible=50;
        despegue= false;
    }

    public void setNombre(String nuevoNombre) {
        nombre = nuevoNombre;
    }

    public void despegar() {
        despegue = true;
        System.out.println( nombre + " despego!");
    }

    public void avanzar(int distancia){
        if (despegue){
            if (combustible >= distancia){
                System.out.println(nombre + "avanza " + distancia + "km");
                combustible =distancia;

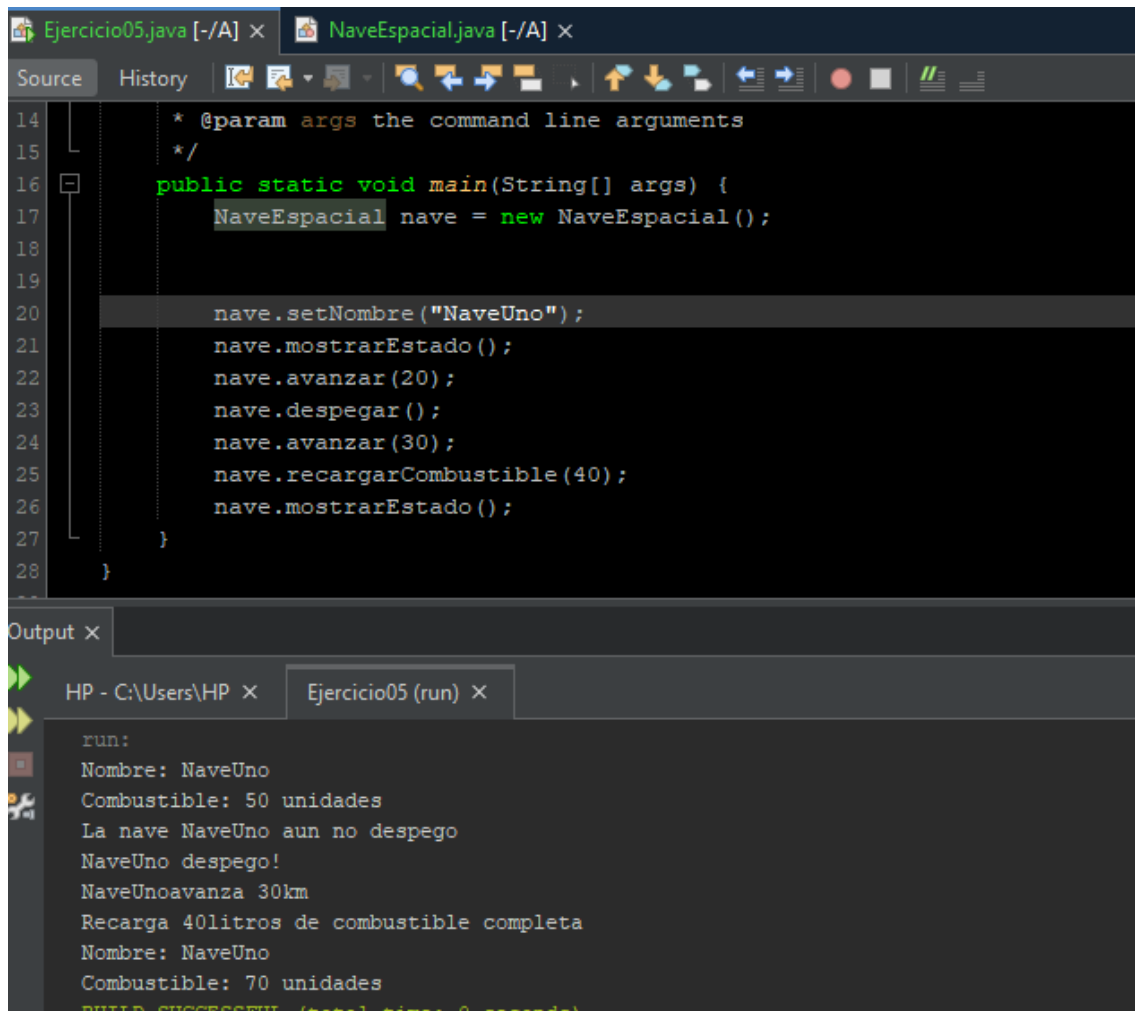
            }else {System.out.println("No hay suficiente combustible");
            }
        }else {
            System.out.println("La nave " + nombre + " aun no despego");
        }
    }
}

```

```

28     if (despegue){
29         if (combustible >= distancia){
30             System.out.println(nombre + "avanza " + distancia + "km");
31             combustible =distancia;
32
33             }else {System.out.println("No hay suficiente combustible");
34             }
35         }
36     }else {
37         System.out.println("La nave " + nombre + " aun no despego");
38     }
39
40 }
41 public void recargarCombustible(int cantidad){
42     if (cantidad + combustible <= 100){
43         combustible += cantidad;
44         System.out.println("Recarga " + cantidad + "litros de combustible completa");
45     } else {
46         System.out.println("La cantidad que desea cargar excede el limite maximo");
47     }
48 }
49
50 public void mostrarEstado() {
51     System.out.println("Nombre: " + nombre);
52     System.out.println("Combustible: " + combustible + " unidades");
53 }
54
55 }

```



The screenshot shows an IDE with two tabs: 'Ejercicio05.java' and 'NaveEspacial.java'. The 'NaveEspacial.java' tab is active, displaying the following Java code:

```
14  * @param args the command line arguments
15  */
16  public static void main(String[] args) {
17      NaveEspacial nave = new NaveEspacial();
18
19
20      nave.setNombre("NaveUno");
21      nave.mostrarEstado();
22      nave.avanzar(20);
23      nave.despegar();
24      nave.avanzar(30);
25      nave.recargarCombustible(40);
26      nave.mostrarEstado();
27  }
28  }
```

Below the code editor, the 'Output' window is open, showing the execution results for 'Ejercicio05 (run)':

```
run:
Nombre: NaveUno
Combustible: 50 unidades
La nave NaveUno aun no despegó
NaveUno despegó!
NaveUno avanza 30km
Recarga 40 litros de combustible completa
Nombre: NaveUno
Combustible: 70 unidades
BUILD SUCCESSFUL (total time: 0 seconds)
```

Enlace de GitHub con los ejercicios-

<https://github.com/Cigarcia1307/Programacion2-UTN-GarciaCintia.git>

## 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.