

INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	Fundamentos de la Programación II				
TÍTULO DE LA PRÁCTICA:	Laboratorio 04: Arreglos de Objetos, Búsquedas y Ordenamientos				
NÚMERO DE PRÁCTICA:	04	AÑO LECTIVO:	2024 - B	NRO. SEMESTRE:	II
FECHA DE PRESENTACIÓN	22/09/2024	HORA DE PRESENTACIÓN	11:59:00 PM		
INTEGRANTE (s): Palma Apaza, Santiago Enrique				NOTA:	
DOCENTE: LINO JOSÉ PINTO OPPE					

SOLUCIÓN Y RESULTADOS
<p>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</p> <p>Clase Nave</p> <p>La clase Nave representa una unidad en un juego. Contiene atributos que describen las características de la nave, como su nombre, posición (fila y columna), estado y puntos. Los métodos de la clase permiten modificar y acceder a estos atributos.</p> <p>Métodos de la Clase Nave</p> <ol style="list-style-type: none">1. setNombre(String n)<ul style="list-style-type: none">○ Establece el nombre de la nave.2. setFila(int f)<ul style="list-style-type: none">○ Establece la fila en la que se encuentra la nave.3. setColumna(String c)<ul style="list-style-type: none">○ Establece la columna en la que se encuentra la nave.4. setEstado(boolean e)<ul style="list-style-type: none">○ Establece el estado de la nave (activa o inactiva).5. setPuntos(int p)<ul style="list-style-type: none">○ Establece el número de puntos que tiene la nave.6. getNombre()<ul style="list-style-type: none">○ Devuelve el nombre de la nave.7. getFila()

- Devuelve la fila en la que se encuentra la nave.

8. getColumna()

- Devuelve la columna en la que se encuentra la nave.

9. getEstado()

- Devuelve el estado de la nave.

10. getPuntos()

- Devuelve el número de puntos que tiene la nave.

```
11 public class Nave {
12     private String nombre;
13     private int fila;
14     private String columna;
15     private boolean estado;
16     private int puntos;
17
18     // Métodos mutadores
19     public void setNombre(String n) {
20         nombre = n;
21     }
22
23     public void setFila(int f) {
24         fila = f;
25     }
26
27     public void setColumna(String c) {
28         columna = c;
29     }
30
31     public void setEstado(boolean e) {
32         estado = e;
33     }
34
35     public void setPuntos(int p) {
36         puntos = p;
37     }
38
39     // Métodos accesorios
40     public String getNombre() {
41         return nombre;
42     }
43
44     public int getFila() {
45         return fila;
46     }
47
48     public String getColumna() {
49         return columna;
50     }
51
52     public boolean getEstado() {
53         return estado;
54     }
55
56     public int getPuntos() {
57         return puntos;
```

DemoBatalla1.java

- a. Ingreso de datos de naves: Se utiliza un ciclo **for** para solicitar al usuario los detalles de cada nave (nombre, fila, columna, estado y puntos). Cada uno de estos datos es asignado a un objeto **Nave** usando los métodos *setter*.
- b. Mostrar naves creadas: Después de la creación de las naves, se llama al método **mostrarNaves** para imprimir los detalles de cada una de ellas, mostrando su nombre, fila, columna, estado y puntos.
- c. Búsqueda de naves por nombre y puntos: Se implementan los métodos **mostrarPorNombre** y **mostrarPorPuntos** que permiten buscar naves en el arreglo ya sea por el nombre ingresado por el usuario o por puntos menores o iguales al valor especificado.

```

1 package labFP2;
2
3 import java.util.*;
4
5 public class DemoBatalla1 {
6     public static void main(String[] args) {
7         Nave[] misNaves = new Nave[10];
8         Scanner sc = new Scanner(System.in);
9         String nomb, col;
10        int fil, punt;
11        boolean est;
12
13        for (int i = 0; i < misNaves.length; i++) {
14            System.out.println("Nave " + (i + 1));
15            System.out.print("Nombre: ");
16            nomb = sc.next();
17            System.out.print("Fila: ");
18            fil = sc.nextInt();
19            System.out.print("Columna: ");
20            col = sc.next();
21            System.out.print("Estado (true/false): ");
22            est = sc.nextBoolean();
23            System.out.print("Puntos: ");
24            punt = sc.nextInt();
25
26            misNaves[i] = new Nave();
27            misNaves[i].setNombre(nomb);
28            misNaves[i].setFila(fil);
29            misNaves[i].setColumna(col);
30            misNaves[i].setEstado(est);
31            misNaves[i].setPuntos(punt);
32        }
33
34        System.out.println("\nNaves creadas:");
35        mostrarNaves(misNaves);
36        mostrarPorNombre(misNaves);
37        mostrarPorPuntos(misNaves);
38        System.out.println("\nNave con mayor número de puntos: " + mostrarMayorPuntos(misNaves).getNombre());
39
40        System.out.println("Ingrese un nombre de nave para buscar:");
41        sc.nextLine();
42        String nombreNave = sc.nextLine();
43
44        int pos = busquedaLinealNombre(misNaves, nombreNave);
45        if (pos != -1) {
46            System.out.println("Nave encontrada en posición: " + pos);
47        } else {
48            System.out.println("Nave no encontrada.");
49        }
50
51        ordenarPorPuntosBurbuja(misNaves);
52        mostrarNaves(misNaves);
53
54        ordenarPorNombreBurbuja(misNaves);
55        mostrarNaves(misNaves);
56
57        pos = busquedaBinariaNombre(misNaves, nombreNave);
58        if (pos != -1) {
59            System.out.println("Nave encontrada en posición: " + pos);

```

```

61         System.out.println("Nave no encontrada.");
62     }
63
64     ordenarPorPuntosSeleccion(misNaves);
65     mostrarNaves(misNaves);
66
67     ordenarPorNombreSeleccion(misNaves);
68     mostrarNaves(misNaves);
69
70     ordenarPorPuntosInsercion(misNaves);
71     mostrarNaves(misNaves);
72
73     ordenarPorNombreInsercion(misNaves);
74     mostrarNaves(misNaves);
75 }
76
77 public static void mostrarNaves(Nave[] flota) {
78     for (Nave nave : flota) {
79         if (nave != null) {
80             System.out.println("Nombre: " + nave.getNombre() + ", Fila: " + nave.getFila() +
81                 ", Columna: " + nave.getColumna() + ", Estado: " + nave.getEstado() +
82                 ", Puntos: " + nave.getPuntos());
83         }
84     }
85 }
86
87 public static int busquedaLinealNombre(Nave[] flota, String s) {
88     for (int i = 0; i < flota.length; i++) {
89         if (flota[i] != null && flota[i].getNombre().equalsIgnoreCase(s)) {
90             return i;
91         }
92     }
93     return -1;
94 }
95
96 public static void ordenarPorPuntosBurbuja(Nave[] flota) {
97     for (int i = 0; i < flota.length - 1; i++) {
98         for (int j = 0; j < flota.length - i - 1; j++) {
99             if (flota[j] != null && flota[j + 1] != null && flota[j].getPuntos() > flota[j + 1].getPuntos()) {
100                 Nave temp = flota[j];
101                 flota[j] = flota[j + 1];
102                 flota[j + 1] = temp;
103             }
104         }
105     }
106 }
107
108 public static void ordenarPorNombreBurbuja(Nave[] flota) {
109     for (int i = 0; i < flota.length - 1; i++) {
110         for (int j = 0; j < flota.length - i - 1; j++) {
111             if (flota[j] != null && flota[j + 1] != null && flota[j].getNombre().compareToIgnoreCase(flota[j + 1].getNombre()) > 0) {
112                 Nave temp = flota[j];
113                 flota[j] = flota[j + 1];
114                 flota[j + 1] = temp;
115             }
116         }
117     }
118 }

```

- d. Nave con mayor número de puntos: El método **mostrarMayorPuntos** recorre el arreglo de naves y encuentra la nave con el mayor número de puntos, cuyo nombre es luego mostrado.
- e. Búsqueda lineal por nombre: El método **busquedaLinealNombre** realiza una búsqueda secuencial en el arreglo para encontrar una nave con el nombre proporcionado. Si se encuentra, se devuelve su posición; si no, retorna **-1**.

```

61         System.out.println("Nave no encontrada.");
62     }
63
64     ordenarPorPuntosSeleccion(misNaves);
65     mostrarNaves(misNaves);
66
67     ordenarPorNombreSeleccion(misNaves);
68     mostrarNaves(misNaves);
69
70     ordenarPorPuntosInsercion(misNaves);
71     mostrarNaves(misNaves);
72
73     ordenarPorNombreInsercion(misNaves);
74     mostrarNaves(misNaves);
75 }
76
77 public static void mostrarNaves(Nave[] flota) {
78     for (Nave nave : flota) {
79         if (nave != null) {
80             System.out.println("Nombre: " + nave.getNombre() + ", Fila: " + nave.getFila() +
81                 ", Columna: " + nave.getColumna() + ", Estado: " + nave.getEstado() +
82                 ", Puntos: " + nave.getPuntos());
83         }
84     }
85 }
86
87 public static int busquedaLinealNombre(Nave[] flota, String s) {
88     for (int i = 0; i < flota.length; i++) {
89         if (flota[i] != null && flota[i].getNombre().equalsIgnoreCase(s)) {
90             return i;
91         }
92     }
93     return -1;
94 }
95
96 public static void ordenarPorPuntosBurbuja(Nave[] flota) {
97     for (int i = 0; i < flota.length - 1; i++) {
98         for (int j = 0; j < flota.length - i - 1; j++) {
99             if (flota[j] != null && flota[j + 1] != null && flota[j].getPuntos() > flota[j + 1].getPuntos()) {
100                 Nave temp = flota[j];
101                 flota[j] = flota[j + 1];
102                 flota[j + 1] = temp;
103             }
104         }
105     }
106 }
107
108 public static void ordenarPorNombreBurbuja(Nave[] flota) {
109     for (int i = 0; i < flota.length - 1; i++) {
110         for (int j = 0; j < flota.length - i - 1; j++) {
111             if (flota[j] != null && flota[j + 1] != null && flota[j].getNombre().compareToIgnoreCase(flota[j + 1].getNombre()) > 0) {
112                 Nave temp = flota[j];
113                 flota[j] = flota[j + 1];
114                 flota[j + 1] = temp;
115             }
116         }
117     }
118 }

```


- f. Ordenación de naves por puntos y nombre (Método Burbuja): Los métodos **ordenarPorPuntosBurbuja** y **ordenarPorNombreBurbuja** ordenan las naves en el arreglo utilizando el algoritmo de ordenación Burbuja, ya sea por puntos o por nombre. Después de cada ordenación, se muestra el arreglo ordenado.
- g. Búsqueda binaria por nombre: Se utiliza el método **busquedaBinariaNombre** para encontrar una nave por su nombre mediante una búsqueda binaria, lo cual requiere que el arreglo esté previamente ordenado.
- h. Ordenación usando Selección e Inserción: Los métodos de ordenación **ordenarPorPuntosSeleccion**, **ordenarPorNombreSeleccion**, **ordenarPorPuntosInsercion** y **ordenarPorNombreInsercion**

implementan los algoritmos de Selección e Inserción, aplicándolos tanto para ordenar las naves por puntos como por nombre.

```
186         while (j >= 0 && flota[j] != null && key != null && flota[j].getNombre().compareToIgnoreCase(key) > 0) {
187             flota[j + 1] = flota[j];
188             j--;
189         }
190         flota[j + 1] = key;
191     }
192 }
193
194 public static Nave mostrarMayorPuntos(Nave[] flota) {
195     Nave naveMayor = flota[0];
196     for (int i = 1; i < flota.length; i++) {
197         Nave nave = flota[i];
198         if (nave != null && nave.getPuntos() > naveMayor.getPuntos()) {
199             naveMayor = nave;
200         }
201     }
202     return naveMayor;
203 }
204
205 public static void mostrarPorNombre(Nave[] flota) {
206     Scanner sc = new Scanner(System.in);
207     System.out.print("\nIngrese el nombre de la nave a buscar: ");
208     String nombreBuscado = sc.next();
209
210     boolean encontrado = false;
211     for (Nave nave : flota) {
212         if (nave != null && nave.getNombre().equalsIgnoreCase(nombreBuscado)) {
213             System.out.println("Nave encontrada: " + nave.getNombre() + ", Fila: " + nave.getFila() +
214                 ", Columna: " + nave.getColumna() + ", Estado: " + nave.getEstado() +
215                 ", Puntos: " + nave.getPuntos());
216             encontrado = true;
217         }
218     }
219     if (!encontrado) {
220         System.out.println("No se encontró ninguna nave con ese nombre.");
221     }
222 }
223
224 public static void mostrarPorPuntos(Nave[] flota) {
225     Scanner sc = new Scanner(System.in);
226     System.out.print("\nIngrese el número de puntos: ");
227     int puntosBuscados = sc.nextInt();
228
229     System.out.println("Naves con puntos menores o iguales a " + puntosBuscados + ":");
230     boolean encontrado = false;
231     for (Nave nave : flota) {
232         if (nave != null && nave.getPuntos() <= puntosBuscados) {
233             System.out.println("Nombre: " + nave.getNombre() + ", Fila: " + nave.getFila() +
234                 ", Columna: " + nave.getColumna() + ", Estado: " + nave.getEstado() +
235                 ", Puntos: " + nave.getPuntos());
236             encontrado = true;
237         }
238     }
239     if (!encontrado) {
240         System.out.println("No se encontraron naves con esos puntos.");
241     }
242 }
243 }
```

COMMITTS

NAVE.java

 MINGW64:/c/Users/Usuario24B/Desktop/sas

```
Usuario24B@DESKTOP-RC1N7NP MINGW64 ~ (master)
$ cd Desktop

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop (master)
$ cd sas

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git init
Initialized empty Git repository in C:/Users/Usuario24B/Desktop/sas/.git/

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git add Nave.java

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git commit -m "Reutilizada la clase Nave del ejercicio pasado, si es que se ne
cesitan m as ajustes a futuro, se realizar n,"
[master (root-commit) 2ed89f5] Reutilizada la clase Nave del ejercicio pasado, s
i es que se necesitan m as ajustes a futuro, se realizar n,
1 file changed, 47 insertions(+)
 create mode 100644 Nave.java

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git remote add origin https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_
ENRIQUE_LABORATORIO_04.git


Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git branch
* master

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git push origin master
remote: Permission to santiagopalma12/PALMA_APAZA_SANTIAGO_ENRIQUE_LABORATORIO_0
4.git denied to aaronQuinonez.
fatal: unable to access 'https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_
ENRIQUE_LABORATORIO_04.git/': The requested URL returned error: 403

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ !
```

DemoBatalla.java

Primer commit

 MINGW64:/c/Users/Usuario24B/Desktop/sas

```
Usuario24B@DESKTOP-RC1N7NP MINGW64 ~ (master)
$ cd Desktop

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop (master)
$ cd sas

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git add
Nothing specified, nothing added.
hint: Maybe you wanted to say 'git add .'
hint: Disable this message with "git config advice.addEmptyPaths false"

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git init
Reinitialized existing Git repository in C:/Users/Usuario24B/Desktop/sas/.git/

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git add DemoBatalla.java

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git commit -m "Reutilizado el ejercicio, anterior pero incluidos los métodos r
equeridos por el ejercicio, estos sin desarrollo, eliminando los que cumplían a
ntes la misma tarea"
[master 4b4f733] Reutilizado el ejercicio, anterior pero incluidos los métodos r
equeridos por el ejercicio, estos sin desarrollo, eliminando los que cumplían a
ntes la misma tarea
1 file changed, 162 insertions(+)
create mode 100644 DemoBatalla.java

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git remote add origin https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_
ENRIQUE_LABORATORIO_04.git
error: remote origin already exists.

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git branch
* master

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ git push origin master
remote: Permission to santiagopalma12/PALMA_APAZA_SANTIAGO_ENRIQUE_LABORATORIO_0
4.git denied to aaronQuinonez.
fatal: unable to access 'https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_
ENRIQUE_LABORATORIO_04.git/': The requested URL returned error: 403

Usuario24B@DESKTOP-RC1N7NP MINGW64 ~/Desktop/sas (master)
$ |
```

Segundo commit

```
santi@5lofiu MINGW64 ~
$ cd Desktop
bash: cd: Desktop: No such file or directory

santi@5lofiu MINGW64 ~
$ cd Escritorio
bash: cd: Escritorio: No such file or directory

santi@5lofiu MINGW64 ~
$ cd C:

santi@5lofiu MINGW64 /c
$ cd users

santi@5lofiu MINGW64 /c/users
$ cd santi

santi@5lofiu MINGW64 /c/users/santi
$ cd OneDrive

santi@5lofiu MINGW64 /c/users/santi/OneDrive
$ cd Escritorio

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio
$ cd sas1

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1
$ git init
Initialized empty Git repository in C:/Users/santi/OneDrive/Escritorio/sas1/.git/

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ git add DemoBatal1a1
fatal: pathspec 'DemoBatal1a1' did not match any files

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ DemoBatal1a1.java
bash: DemoBatal1a1.java: command not found

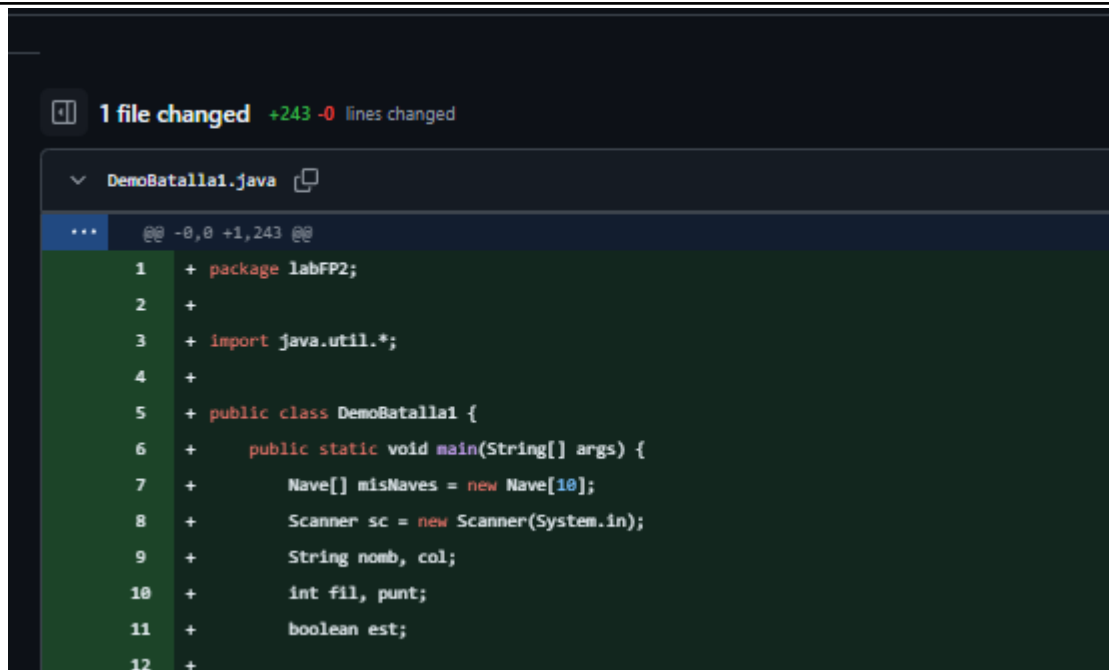
santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ git add DemoBatal1a1.java

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ git commit -m "Métodos con las búsquedas requeridas realizados, último paso
sería probar valores y corregir errores"
[master (root-commit) 6da159b] Métodos con las búsquedas requeridas realizados
, último paso sería probar valores y corregir errores
1 file changed, 243 insertions(+)
create mode 100644 DemoBatal1a1.java

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ git remote add origin https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_
ENRIQUE_LABORATORIO_04.git

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ git branch
* master

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas1 (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 1.73 KiB | 1.73 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_ENRIQUE_LABORATORIO_0
4.git
 * [new branch]      master -> master
```



```
1  + package labFP2;
2  +
3  + import java.util.*;
4  +
5  + public class DemoBatalla1 {
6  +     public static void main(String[] args) {
7  +         Nave[] misNaves = new Nave[10];
8  +         Scanner sc = new Scanner(System.in);
9  +         String nomb, col;
10 +         int fil, punt;
11 +         boolean est;
12 +
```

II. PRUEBAS

¿Con qué valores comprobaste que tu práctica estuvo correcta?

Ingresé valores tipo String para los nombres de las naves y también para las columnas. usé enteros para las filas y para los puntos.

```
Nave 1
Nombre: santiago
Fila: 1
Columna: 1
Estado (true/false): true
Puntos: 10
Nave 2
Nombre: juan
Fila: 2
Columna: 2
Estado (true/false): true
Puntos: 2
Nave 3
Nombre: pedro
Fila: 3
Columna: 3
Estado (true/false): true
Puntos: 30
Nave 4
Nombre: jose
Fila: 5
Columna: 5
Estado (true/false): true
Puntos: 20
Nave 5
Nombre: nino
Fila: 5
Columna: 6
Estado (true/false): true
Puntos: 20
Nave 6
Nombre: pin
Fila: 7
Columna: 7
Estado (true/false): true
Puntos: 30
Nave 7
Nombre: filemon
Fila: 9
Columna: 9
Estado (true/false): true
Puntos: 20312
Nave 8
Nombre: josee
Fila: 10
Columna: 10
Estado (true/false): true
Puntos: 203
Nave 9
Nombre: ray
Fila: 30
Columna: 30
Estado (true/false): true
Puntos: 30
Nave 10
Nombre: aqua
Fila: 1
Columna: 4
```

```
Naves creadas:
Nombre: santiago, Fila: 1, Columna: 1, Estado: true, Puntos: 10
Nombre: juan, Fila: 2, Columna: 2, Estado: true, Puntos: 2
Nombre: pedro, Fila: 3, Columna: 3, Estado: true, Puntos: 30
Nombre: jose, Fila: 5, Columna: 5, Estado: true, Puntos: 20
Nombre: nino, Fila: 5, Columna: 6, Estado: true, Puntos: 20
Nombre: pin, Fila: 7, Columna: 7, Estado: true, Puntos: 30
Nombre: filemon, Fila: 9, Columna: 9, Estado: true, Puntos: 20312
Nombre: josee, Fila: 10, Columna: 10, Estado: true, Puntos: 203
Nombre: ray, Fila: 30, Columna: 30, Estado: true, Puntos: 30
Nombre: aqua, Fila: 1, Columna: 4, Estado: true, Puntos: 123
```

```
Ingrese el nombre de la nave a buscar: aqua
No se encontró ninguna nave con ese nombre.
```

```
Ingrese el número de puntos: 30
Naves con puntos menores o iguales a 30:
Nombre: santiago, Fila: 1, Columna: 1, Estado: true, Puntos: 10
Nombre: juan, Fila: 2, Columna: 2, Estado: true, Puntos: 2
Nombre: pedro, Fila: 3, Columna: 3, Estado: true, Puntos: 30
Nombre: jose, Fila: 5, Columna: 5, Estado: true, Puntos: 20
Nombre: nino, Fila: 5, Columna: 6, Estado: true, Puntos: 20
Nombre: pin, Fila: 7, Columna: 7, Estado: true, Puntos: 30
Nombre: ray, Fila: 30, Columna: 30, Estado: true, Puntos: 30
```

```
Nave con mayor número de puntos: filemon
```

¿Qué resultado esperas obtener para cada valor de entrada?

Espero que las naves se guarden con los nombres respectivos, en las filas y columnas respectivas, con el valor booleano ingresado y por último con la cantidad de puntos ingresada. Luego que me enseñe todas las naves ingresadas, que me pregunte por buscar esta nave y enseñarme todos sus datos y por último pedirme un número para enseñar de manera desordenada los valores con esa o menor puntuación. Y finalmente mostrar la nave con más puntos.

¿Qué valor o comportamiento obtuviste para cada valor de entrada?

Lo esperado en la consigna anterior.

RUBRICA

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	x	2	
2. Commits	Hay capturas de pantalla de los	4	x	3	

	commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).				
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	x	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	x	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	x	1	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	x	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	x	2	
8. Madurez	El Informe muestra de manera general una	4	x	3	

	evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).				
TOTAL		20		17	

III. CONCLUSIONES

Eficiencia en la manipulación de datos: El programa implementa múltiples algoritmos de búsqueda y ordenación, lo que permite trabajar de manera eficiente con grandes conjuntos de datos representados por las naves. La utilización de algoritmos como la búsqueda binaria y la ordenación por burbuja, selección e inserción muestra cómo se pueden aplicar diferentes técnicas para resolver problemas de orden y búsqueda, dependiendo del contexto.

Modularidad del código: El diseño del programa se ha enfocado en la modularidad, organizando las diferentes tareas en métodos específicos. Esta separación facilita el mantenimiento y la comprensión del código, permitiendo agregar funcionalidades o modificar algoritmos sin alterar otras partes del sistema.

Flexibilidad en el manejo de objetos: El uso de arreglos de objetos de la clase **Nave** demuestra la flexibilidad y el poder de la Programación Orientada a Objetos (POO). Cada objeto almacena de forma independiente sus atributos, como nombre, puntos, posición y estado, lo que permite la personalización de cada nave y su tratamiento individual durante las operaciones de búsqueda y ordenación.

Aplicación de algoritmos clásicos: La implementación de algoritmos clásicos de búsqueda y ordenación permite comparar sus eficiencias en un contexto práctico. La búsqueda lineal es simple pero ineficiente para grandes volúmenes de datos, mientras que la búsqueda binaria optimiza la búsqueda cuando los datos están ordenados. Por su parte, los diferentes algoritmos de ordenación muestran distintas complejidades y resultados, destacando la importancia de elegir el método adecuado para el problema que se desea resolver.

METODOLOGIA

Análisis del problema: El primer paso fue comprender los requisitos del programa. Se identificó que el objetivo principal era crear una simulación en la que se manipularan múltiples objetos de tipo Nave y se realizaran operaciones de búsqueda y ordenación sobre ellos. Se seleccionaron los atributos y métodos necesarios para la clase Nave y se plantearon las interacciones clave con el usuario (entrada de datos y consultas).

Diseño modular del programa: Se decidió estructurar el programa de manera modular, separando las operaciones en métodos independientes. Esto facilitó la prueba individual de cada función (mostrar, ordenar, buscar) y permitió hacer cambios y mejoras a medida que se avanzaba en el desarrollo.

Implementación de algoritmos: Se implementaron diversos algoritmos de búsqueda y ordenación. Para cada uno de ellos, se evaluaron diferentes enfoques (lineal y binario para búsqueda; burbuja, selección e inserción para ordenación), lo que permitió que el código fuera más flexible y demostrara la aplicabilidad de cada algoritmo en distintos escenarios.

Pruebas y depuración: Una vez completada la implementación, se realizaron pruebas del programa, validando que las entradas de los usuarios generaran los resultados esperados. Las pruebas incluyeron la correcta creación de naves, la búsqueda de naves por nombre o puntos, la ordenación de las naves y la comparación de los resultados obtenidos con los esperados.

Optimización y documentación: El código fue optimizado para reducir redundancias y mejorar su legibilidad. Se agregaron comentarios explicativos y se realizó un análisis del rendimiento de los distintos algoritmos para elegir los más adecuados en función del tamaño del conjunto de naves.

REFERENCIAS Y BIBLIOGRAFÍA

Aedo López, M. W. (2019). *Fundamentos de programación I: Java Básico (1. ed.)*. Universidad Nacional de San Agustín. ISBN 978-612-4337-55-0.