Informe de Laboratorio 04

Tema: Arreglos de objetos, Búsqueda y Algoritmos de ordenamiento

Nota

Estudiante	Escuela	Asignatura
Ryan Fabian Valdivia Segovia rvaldiviase@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
04	Arreglos de objetos, Búsqueda y Algoritmos de ordenamiento	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 18 Septiembre 2023	Al 24 Septiembre 2023

1. Tarea

1.1. Actividad: Demo Batalla

- Analice, complete y pruebe el Código de la clase DemoBatalla.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Home Single Language 64 bits 22621.2283
- VIM 9.0.
- Visual Studio Code 64 bits 1.82.2
- OpenJDK 64-Bits 11.0.16.1
- Git 2.41.0.windows.1
- Cuenta en GitHub con el correo institucional.
- Código parcial proporcionado por el profesor.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/RyanValdivia/fp2-23b.git`
- URL para el laboratorio 04 en el Repositorio GitHub.
- `https://github.com/RyanValdivia/fp2-23b/tree/main/fase01/lab04`

4. Actividades

4.1. Actividad 1

- Realicé un commit reutilizando el código ya hecho del laboratorio anterior.

Listing 1: Comentando el código parcial

```
$ git log Ahorcado.java
commit d44ccee0c41e9bfcd80323582d0cea9defa7667b
Author: RYAN VALDIVIA <rvaldiviase@unsa.edu.pe>
Date:  Wed Sep 27 16:18:18 2023 -0500
   Reutilizando el codigo del laboratorio anterior
```

- Lo primero que hice fue elaborar el método para intercambiar los valores en el arreglo, algo muy utilizado en los métodos de ordenamiento.

Listing 2: Intercambio

```
6   public static void intercambiar(Nave[] flota, int i, int j) {
7       Nave temp;
8       temp = flota[i];
9       flota[i] = flota[j];
10      flota[j] = temp;
11  }
```

- Seguidamente, comencé elaborando los métodos de ordenamiento, ya que para realizar búsqueda binaria en un arreglo, debe estar ordenado.
- Comencé con el ordenamiento por burbuja, realizando sus dos variantes, uno para datos enteros (el puntaje de las naves) y otro para Strings (los nombres).

Listing 3: Ordenamiento por burbuja: Enteros

```
12  public static void ordenarPorPuntosBurbuja(Nave[] flota) {
13      for (int i = 0; i < flota.length; i++) {
14          for (int j = 0; j < flota.length - 1; j++) {
15              if (flota[j].getPuntos() > flota[j + 1].getPuntos()) {
16                  intercambiar(flota, j, j + 1);
17              }
18          }
19      }
20  }
```

- En este algoritmo se intercambian valores de dos en dos para ordenar el arreglo.
- Este es un método conocido para ordenar, la variante se presenta con las cadenas o strings, donde en lugar de ir comparando los valores, uso el método compareTo() para ir comparando sus valores alfabéticos e irlos ordenando.

Listing 4: Ordenamiento por burbuja: Cadenas

```
21  public static void ordenarPorNombreBurbuja(Nave[] flota) {
22      for (int i = 0; i < flota.length; i++) {
23          for (int j = 0; j < flota.length - 1; j++) {
24              if (flota[j].getNombre().compareTo(flota[j + 1].getNombre()) > 0) {
25                  intercambiar(flota, j, j + 1);
26              }
27          }
28      }
29   }
```

- Una vez que ya tenía hecho un par de métodos de ordenamiento, procedí a la búsqueda binaria. La realicé para buscar un nombre dentro de una lista ordenada.

Listing 5: Búsqueda binaria: Cadenas

```
30  public static int busquedaBinariaNombre(Nave[] flota, String s) {
31      int baja = 0, media, alta = flota.length - 1;
32      while (baja <= alta) {
33          media = (alta + baja) / 2;
34          String nMedio = flota[media].getNombre();
35          int compare = s.compareTo(nMedio);
36          if (compare == 0) {
37              return media;
38          } else if (compare < 0) {
39              alta = media - 1;
40          } else {
41              baja = media + 1;
42          }
43      }
44      return -1;
45   }
```

- Además, cree un método para la búsqueda lineal en un arreglo (No necesita estar ordenado).

Listing 6: Método para obtener números desordenados

```
46   public static int busquedaLinealNombre(Nave[] flota, String s) {
47       for (int i = 0; i < flota.length; i++) {
48           if (flota[i].getNombre().equals(s)) {
49               return i;
50           }
51       }
52       return -1;
53   }
```

- Asímismo, creé el resto de métodos de ordenamiento, como el de selección.
- Realicé dos variantes, una para la búsqueda de valores enteros (Puntaje) y para cadenas (Nombres) utilizando compareTo de nuevo para obtener el menor valor alfabético del arreglo.

Listing 7: Ordenamiento por selección: Enteros

```
54  public static void ordenarPorPuntosSeleccion(Nave[] flota) {
55      for (int i = 0; i < flota.length - 1; i++) {
56          int menor = flota[i].getPuntos();
57          int indice = i;
58          for (int j = i + 1; j < flota.length; j++) {
59              if (flota[j].getPuntos() < menor) {
60                  menor = flota[j].getPuntos();
61                  indice = j;
62              }
63          }
64          if (indice != i) {
65              Nave pivot = flota[i];
66              flota[i] = flota[indice];
67              flota[indice] = pivot;
68          }
69      }
70  }
```

- Para este algoritmo, se parte del primer elemento del arreglo y se selecciona el menor valor de todo el arreglo, si este es menor que nuestro valor inicial, se intercambian, y asi seguimos intercambiando hasta que la lista esté ordenada.

Listing 8: Ordenamiento por selección: Cadenas

```
71  public static void ordenarPorNombreSeleccion(Nave[] flota) {
72      for (int i = 0; i < flota.length - 1; i++) {
73          String menor = flota[i].getNombre();
74          int indice = i;
75          for (int j = i + 1; j < flota.length; j++) {
76              if (flota[i].getNombre().compareTo(menor) < 0) {
77                  menor = flota[j].getNombre();
78                  indice = j;
79              }
80          }
81          if (indice != i) {
82              Nave pivot = flota[i];
83              flota[i] = flota[indice];
84              flota[indice] = pivot;
85          }
86      }
87  }
```

- Ahora sigue los algoritmos de ordenamiento por inserción, también dos variantes. Una para números enteros (el puntaje) y otra para los nombres, sin embargo. Ahora una variación más es para ordenar los nombres de la Z a la A, en vez de al revés como he estado haciendo hasta ahora.

Listing 9: Ordenamiento por inserción: Enteros

```
88  public static void ordenarPorPuntosInsercion(Nave[] flota) {
89      for (int i = 1; i < flota.length; i++) {
90          Nave valor = flota[i];
91          int j = i;
92          for (j = i; 0 < j && flota[j - 1].getPuntos() > valor.getPuntos(); j--) {
93              flota[j] = flota[j - 1];
94          }
95          flota[j] = valor;
96      }
97
98  }
```

- La lógica es, dado una lista desordenada, se parte desde el segundo término y se compara los valores hacia la izquierda, si es mayor el valor a la izquierda, se inserta hacia esa dirección hasta que toda la lista este ordenada.

Listing 10: Ordenamiento por inserción: Cadenas de la Z a la A

```
99  public static void ordenarPorNombreInsercion(Nave[] flota) {
100     for (int i = 1; i < flota.length; i++) {
101         Nave valor = flota[i];
102         int j = i;
103         for (j = i; 0 < j && flota[j - 1].getNombre().compareTo(valor.getNombre()) < 0;
                j--) {
104             flota[j] = flota[j - 1];
105         }
106         flota[j] = valor;
107     }
108 }
```

- Solo que en este caso, hacemos una modificación para que los valores mayores queden a la izquierda, ya que queremos que la lista esté ordenada de la Z a la A.
- Con esto ya tenemos algoritmos hechos para poder ordenar y buscar en arreglos.

Listing 11: Aplicando los métodos de búsqueda

```
109   int pos = busquedaLinealNombre(misNaves, nombre);
110     if (pos != -1) {
111         Nave n = misNaves[pos];
112         System.out.println("Nave " + pos + ":" + n.getNombre());
113         System.out.println("Posicion: " + n.getFila() + n.getColumna());
114         System.out.println("Puntos: " + n.getPuntos());
115         if (n.getEstado()) {
116             System.out.println("Sigue con vida");
117         } else {
118             System.out.println("Fue destruida");
119         }
120     } else {
121         System.out.println("Nave no encontrada");
122     }
123     ordenarPorPuntosBurbuja(misNaves);
```

- A continuación pruebo el código con cinco naves para no tener que ingresar tantos datos.

Listing 12: Prueba del código

```
$javac DemoBatalla.java Nave.java
$java DemoBatalla
Nave 1:
Connor 1 A true 60
Nave 2:
Delta 3 C true 25
Nave 3:
Whisky 6 J true 40
Nave 4:
Jayce 7 A true 55
Nave 5:
Abigail 8 B true 42
```

```
Naves creadas:
Nave 1:Connor
Posicion: 1A
Puntos: 60
Sigue con vida

Nave 2:Delta
Posicion: 3C
Puntos: 25
Sigue con vida

Nave 3:Whisky
Posicion: 6J
Puntos: 40
Sigue con vida

Nave 4:Jayce
Posicion: 7A
Puntos: 55
Sigue con vida

Nave 5:Abigail
Posicion: 8B
Puntos: 42
Sigue con vida
```

- Mostrando las naves.

```
Abigail
Nave 4:Abigail
Posicion: 8B
Puntos: 42
Sigue con vida
```

- Buscando un nombre.

```
Nave 1:Delta
Posicion: 3C
Puntos: 25
Sigue con vida

Nave 2:Whisky
Posicion: 6J
Puntos: 40
Sigue con vida

Nave 3:Abigail
Posicion: 8B
Puntos: 42
Sigue con vida

Nave 4:Jayce
Posicion: 7A
Puntos: 55
Sigue con vida

Nave 5:Connor
Posicion: 1A
Puntos: 60
Sigue con vida
```

- Ordenando por puntaje.

```
Naves creadas:
Nave 1:Connor
Posicion: 1A
Puntos: 60
Sigue con vida

Nave 2:Delta
Posicion: 3C
Puntos: 25
Sigue con vida

Nave 3:Whisky
Posicion: 6J
Puntos: 40
Sigue con vida

Nave 4:Jayce
Posicion: 7A
Puntos: 55
Sigue con vida

Nave 5:Abigail
Posicion: 8B
Puntos: 42
Sigue con vida
```

- Ordenando por nombre.

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

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 commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	2	
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	2	
6. Fechas	Las fechas de modificación del código fuente estan 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 evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		18	

6. Referencias

- Fundamentos de la programación 2 - Tópicos de la programación Orientada a Objetos (Marco Aedo)