Informe de Laboratorio 04

Tema: Arreglos de Objetos, Búsqueda y Ordenamiento de Burbuja

Nota

Estudiante	Escuela	Asignatura
Jhonatan David Arias Quispe jariasq@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programacion 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
04	Arreglos de Objetos, Búsqueda y Ordenamiento de Burbuja	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 20 Setiembre 2023	Al 27 Setiembre 2023

1. Actividades

- 1. Cree un Proyecto llamado Laboratorio4
- 2. Usted podrá reutilizar las dos clases Nave.java y DemoBatalla.java. creadas en Laboratorio 3
- 3. Completar el Código de la clase DemoBatalla

2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernell
- NeoVim
- OpenJDK 64-Bit 20.0.1
- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Creacion de programas con CLI

3. URL de Repositorio Github

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

4. Actividad 1

4.1. Para las pruebas de ejecucion

- Para la prueba de ejecucion de cada uno de los metodos que desarrollaremos, vamos a usar las siguientes naves:

```
Naves creadas:

Nave 1:
Nombre: cesar
Estado: true
Puntos: 23

Nave 2:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 3:
Nombre: Bannia
Estado: true
Puntos: 89
```

5. Completando los metodos faltantes

- La acitivdad del laboratorio 04 nos pide crear los siguientes metodos:

5.1. Busqueda Lineal

- La busqueda lineal es una busqueda simple, en la que se recorre todo el arreglo en busca de una informacion dada

Listing 1: Metodo de busqueda lineal

```
public static int busquedaLinealNombre(Nave[] flota, String s){
  for(int i = 0; i < flota.length; i++){
    if(flota[i].getNombre().equals(s))
      return i;
  }
  return -1;
}
```

- El metodo devuelve la posicion del elemento en caso encontrarlo
- Y de lo contrario devuelve -1 lo que indica que el elemento no se encontro

5.1.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 2: Prueba del metodo

```
Busqueda lineal, inserte un nombre:
Bannia
Nombre: Bannia
Estado: true
Puntos: 89
```

5.2. Ordenamiento por nombre: burbuja

- Ahora veremos como implementar el bubble sort para ordenar las naves con respecto a sus nombres

Listing 3: Metodo de ordenamiento burbuja por nombre

```
  public static void ordenarPorNombreBurbuja(Nave[] flota){
    for(int i = 0; i < flota.length - 1; i++){
      for(int j = 0; j < flota.length - 1 - i; j++){
        if(esMayor(flota[j].getNombre(), flota[j + 1].getNombre()))
          intercambiar(flota, j, j + 1);
      }
    }
  }
```

- Como vemos, estamos utilizando un metodo que compara dos strings y devuelve cual es mayor lexicograficamente:

Listing 4: Sub metodo para comparar dos strings

```
public static boolean esMayor(String s1, String s2){
  return s1.compareToIgnoreCase(s2) > 0;
}
```

- Ahora tenemos una manera de saber el orden de precedencia entre strings
- Adicionalmente a esto necesitamos otro metodo para intercambiar elementos de un arreglo de naves:

Listing 5: Sub metodo para intercambiar dos elementos

```
  public static void intercambiar(Nave[] flota, int i, int j){
    Nave aux = flota[i];
    flota[i] = flota [j];
    flota[j] = aux;
  }
```

5.2.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 6: Prueba del metodo

```
Ahora usaremos el algoritmo de ordenamiento burbuja, con respecto a Nombres.
Naves ordenadas:


Nave 1:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 2:
Nombre: Bannia
Estado: true
Puntos: 89

Nave 3:
Nombre: cesar
Estado: true
Puntos: 23
```

5.3. Ordenamiento por puntos: burbuja

- Ahora veremos como ordenar los elementos de un arreglo respecto a sus puntos usando el algoritmo burbuja
- Hacer esto es muy similar al ordenamiento por nombre, unicamente cambiando getNombre por getPuntos:

Listing 7: Metodo para ordenar por puntos

```
public static void ordenarPorPuntosBurbuja(Nave[] flota){
  for(int i = 0; i < flota.length - i; i++){
    for(int j = 0; j < flota.length - 1 - i; j++){
      if(flota[j].getPuntos() > flota[j + 1].getPuntos())
        intercambiar(flota, j, j + 1);
    }
  }
}
```

- Ahora en lugar de comparar dos strings, directamente se comparan los valores numericos de sus puntos

5.3.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 8: Prueba del metodo

```
Ahora usaremos el algoritmo de ordenamiento burbuja, con respecto a Puntos.
Naves ordenadas:


```

```
Nave 1:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 2:
Nombre: cesar
Estado: true
Puntos: 23

Nave 3:
Nombre: Bannia
Estado: true
Puntos: 89
```

5.4. Ordenamiento por nombre: seleccion

- Ahora veremos el otro algoritmo de ordenamiento, el de seleccion
- La estructura basica del algoritmo, es ir seleccionando de izquierda a derecha los elementos mas chicos para ordenar el arreglo
- Su implementacion para ordenar una serie de naves lexicograficamente es la siguiente:

Listing 9: Metodo para ordenar por nombre

```
public static void ordenarPorNombreSeleccion(Nave[] flota){
  for(int i = 0; i < flota.length - 1; i++){
    int min = i;
    for(int j = i + 1; j < flota.length; j++){
      if(esMayor(flota[min].getNombre(), flota[j].getNombre()))
        min = j;
    }
    intercambiar(flota, i, min);
  }
}
```

5.4.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 10: Prueba del metodo

```
Ahora usaremos el algoritmo de ordenamiento seleccion, con respecto a Nombres.
Naves ordenadas:


Nave 1:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 2:
Nombre: Bannia
Estado: true
Puntos: 89

Nave 3:
Nombre: cesar
```

```
  Estado: true
  Puntos: 23
```

5.5. Ordenamiento por puntos: seleccion

- Ahora usaremos este mismo algoritmo para ordenar el array con respecto a los puntos
- Es la misma estructura y el mismo sentido, en lugar de unsar getNombre, usamos getPuntos

Listing 11: Metodo para ordenar por puntos

```
    public static void ordenarPorPuntosSeleccion(Nave[] flota){
  for(int i = 0; i < flota.length - 1; i++){
    int min = i;
    for(int j = i + 1; j < flota.length; j++){
      if(flota[j].getPuntos() < flota[min].getPuntos())
        min = j;
    }
    intercambiar(flota, i, min);
  }
}
```

5.5.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 12: Prueba del metodo

```
Ahora usaremos el algoritmo de ordenamiento seleccion, con respecto a Puntos.
Naves ordenadas:


Nave 1:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 2:
Nombre: cesar
Estado: true
Puntos: 23

Nave 3:
Nombre: Bannia
Estado: true
Puntos: 89
```

5.6. Ordenamiento por nombre: insercion

- Ahora veremos el algoritmo de insercion
- La idea cae en como ordenamos nuestras cartas cuando tenemos una baraja, agarramos la carta y la ïnsertamos.[en] su lugar
- Vamos a hacer esto pero con codigo, empezamos:

Listing 13: Metodo para ordenar por nombre

```
    public static void ordenarPorNombreInsercion(Nave[] flota){
  for(int i = 1; i < flota.length; i ++){
    Nave actual = flota[i];
    int j = i - 1;
    while(j >= 0 && esMayor(flota[j].getNombre(), actual.getNombre())){
      flota[j + 1] = flota[j];
      j--;
    }
    flota[j + 1] = actual;
  }
 }
```

- Estamos usando una variable para almacenar nuestro elemento a ubicar
- Luego recorremos tantos elementos a la derecha como elementos menores a nuestro pivote hayan
- Y finalmente colocamos nuestro pivote en la posicion que queda libre

5.6.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 14: Prueba del metodo

```
Ahora usaremos el algoritmo de ordenamiento seleccion, con respecto a Nombres.
Naves ordenadas:


Nave 1:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 2:
Nombre: Bannia
Estado: true
Puntos: 89

Nave 3:
Nombre: cesar
Estado: true
Puntos: 23
```

5.7. Ordenamiento por puntos: insercion

- Ahora usaremos este mismo algoritmo para ordenar el array con respecto a los puntos
- Es la misma estructura y el mismo sentido, en lugar de unsar getNombre, usamos getPuntos

Listing 15: Metodo para ordenar por puntos

```
    public static void ordenarPorPuntosInsercion(Nave[] flota){
  for(int i = 1; i < flota.length; i++){
    Nave actual = flota[i];
    int j = i - 1;
    while(j >= 0 && flota[j].getPuntos() > actual.getPuntos()){
      flota[j + 1] = flota[j];
      j--;
```

```
    }
    flota[j + 1] = actual;
  }
 }
```

5.7.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 16: Prueba del metodo

```
Ahora usaremos el algoritmo de ordenamiento insercion, con respecto a Puntos.
Naves ordenadas:


Nave 1:
Nombre: Alvaro
Estado: true
Puntos: 6

Nave 2:
Nombre: cesar
Estado: true
Puntos: 23

Nave 3:
Nombre: Bannia
Estado: true
Puntos: 89
```

5.8. Busqueda binaria

- La busqueda binaria es una forma mas eficiente de buscar un elemento en una lista
- Como pre requisito, es que nuestro arreglo tiene que estar ordenado alfabeticamente, para eso usamos cualquier algoritmo de ordenamiento por nombre visto anteriormente
- El tiempo de complejidad es o(log n), lo que es muy bueno

Listing 17: Metodo para hacer una busqueda binaria

```
  public static int busquedaBinariaNombre(Nave[] flota, String s){
 ordenarPorNombreSeleccion(flota);
 int min = 0;
 int max = flota.length - 1;
 while(min <= max){
   int medio = (max + min) / 2;
   if(flota[medio].getNombre().equalsIgnoreCase(s))
    return medio;
   if(esMayor(s, flota[medio].getNombre()))
    min = medio + 1;
   else
    max = medio - 1;
 }
 return - 1;
}
```

5.8.1. Ejecucion

- Ahora veremos la prueba del metodo:

Listing 18: Prueba del metodo

```
   Busqueda binaria, inserte un nombre:
   cesar
   Nombre: cesar
   Estado: true
   Puntos: 23
```

6. Commits importantes

- Ahora veremos el registro de los commits principales:

```

commit 9b1f5460554796ac356fd9dcae4fc20bb3d3a123
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:  Wed Sep 27 01:46:26 2023 -0500

   Actividad 1: Implementacion del metodo de ordenamiento de Puntos por insercion

commit a68dcb2c45cf068ed3fdf3733bfd3113f16abe6c
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:  Wed Sep 27 01:18:23 2023 -0500

   Actividad 1: Implementacion del algoritmo de ordenamiento de puntos por seleccion

commit 18c3176724a4529cf0e77038215cdf5e180088ae
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:  Wed Sep 27 01:07:19 2023 -0500

   Actividad 1: Implementacion del metodo de busqueda binaria por nombre

commit 0db1011f08fff2f98effb4123c5d2136ea68879d
Author: JhonatanDczel <jariasq@unsa.edu.pe>
Date:  Tue Sep 26 23:54:17 2023 -0500

   Actividad 1: Implementacion del metodo de ordenamiento por puntos usando el algoritmo burbuja
```

- Cada commit representa el uso de un algoritmo diferente
- Esta el algoritmo burbuja, insercion y seleccion, asi como la busqueda binaria

7. Rúbricas

7.1. 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.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: 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 2: 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	4	
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 estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	1	
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	3	
Total		20		17	