

Informe de Laboratorio 04

Tema: Laboratorio 04

Nota

Estudiante	Escuela	Asignatura
Victor Mamani Anahua vmamanian@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 20230489

Laboratorio	Tema	Duración
04	Laboratorio 04	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 17 Setiembre 2023	Al 24 Setiembre 2023

1. Tarea

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

2. Equipos, materiales y temas utilizados

- Sistema Operativo Ubuntu GNU Linux 23 lunar 64 bits Kernell 6.2.v
- Visual Studio Code.
- OpenJDK 64-Bits 19.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Actividades del Laboratorio 04.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/VictorMA18/fp2-23b.git>
- URL para el laboratorio 01 en el Repositorio GitHub.
- <https://github.com/VictorMA18/fp2-23b/tree/main/Fase01/Lab03>

4. Actividades del Laboratorio 04

4.1. Ejercicio DemoBatalla01

- En el primer commit agregamos un metodo `busquedaLinealNombre()` CREADO PARA LA BUSQUEDA DE LA PRIMERA NAVE QUE ES IGUAL AL NOMBRE QUE ESCRIBIMOS ESTA VA PASANDO POR TODAS LAS NAVES Y COMPARANDO SI ALGUNA DE ESTAS TIENE ESE NOMBRE Y SI HAY PONDRÁ LA PRIMERA CONCURRENCIA.
- El código, el commit y la ejecución sería el siguiente:

Listing 1: Commit

```
$ git commit -m "Completamos el metodo busquedaLinealNombre() donde el nombre ingresado  
por el usuario buscara la primera igualdad a esta y tambien devolvera su posicion  
que sera un int"
```

Listing 2: Las líneas de códigos del método creado:

```
public static int busquedaLinealNombre(Nave[] flota, String s){ //METODO CREADO PARA LA  
    BUSQUEDA DE LA PRIMERA NAVE QUE ES IGUAL AL NOMBRE QUE ESCRIBIMOS  
    System.out.println("*****");  
    int count = 0;  
    for(int i = 0; i < flota.length; i++){  
        if(flota[i].getNombre().equals(s)){  
            count++;  
            if(count > 0){  
                System.out.println(flota[i].toString());  
                break;  
            }else{  
                System.out.println("Nave no encontrada");  
            }  
            return i;  
        }  
    }  
    return -1;  
}
```

Listing 3: La ejecución dada:

```
Nave 1 :  
Nombre: Victor  
Fila: 2  
Columna: C  
Estado: true
```

```
Puntos: 45
Nave 2 :
Nombre: Victor
Fila: 3
Columna: A
Estado: true
Puntos: 653
Nave 3 :
Nombre: Victor
Fila: 4
Columna: A
Estado: true
Puntos: 9865

Ingrese el nombre para buscar a la primera nave:
Victor
*****
Nombre: Victor
Columna: C
Fila: 2
Estado: true
Puntos: 45
-----
```

4.2. Ejercicio DemoBatalla01

- En el segundo commit tambien completamos el metodo ordenarPorPuntosBurbuja() en la cual NOS PERMITE ORDENAR DE MENOR A MAYOR DE LA MANERA DE ORDENAR BURBUJA QUE SERIA CAMBIAR POSCIONES SI TU ELEMENTO DE ADELANTE ES MENOR AL ACTUAL Y ASI INTERCAMBIANDO CON LOS SIGUIENTES ELEMENTOS.
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 4: Commit

```
$ git commit -m "COMPLETAMOS ESTE METODO QUE NOS PERMITE ORDENAR DE MENOR A MAYOR DE LA
MANERA DE ORDENAR BURBUJA QUE SERIA CAMBIAR POSCIONES SI TU ELEMENTO DE ADELANTE ES
MENOR AL ACTUAL"
```

Listing 5: Las lineas de codigo del metodo completado:

```
public static void ordenarPorPuntosBurbuja(Nave[] flota){ //COMPLETAMOS ESTE METODO QUE
    NOS PERMITE ORDENAR DE MENOR A MAYOR DE LA MANERA DE ORDENAR BURBUJA QUE SERIA
    CAMBIAR POSCIONES SI TU ELEMENTO DE ADELANTE ES MENOR AL ACTUAL
    for(int i = 0; i < flota.length - 1;i++){
        if(flota[i].getPuntos() > flota[i + 1].getPuntos()){
            Nave temp = flota[i];
            flota[i] = flota[i + 1];
            flota[i + 1] = temp;
        }
    }
}
```

Listing 6: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: A
Estado: true
Puntos: 34
Nave 2 :
Nombre: Pepe
Fila: 3
Columna: E
Estado: true
Puntos: 76
Nave 3 :
Nombre: Lalo
Fila: 6
Columna: G
Estado: true
Puntos: 874

Ordenado por la cantidad de puntos del menor al mayor mediante el metodo burbuja:
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 34
*****
Nombre: Pepe
Columna: E
Fila: 3
Estado: true
Puntos: 76
*****
Nombre: Lalo
Columna: G
Fila: 6
Estado: true
Puntos: 874
*****
```

4.3. Ejercicio DemoBatalla01

- En el tercer commit completamos este metodo mostrarPorNombreBurbuja() donde aplicamos un for para ir por cada elemento comparando si este su primer caracter es mayor a la siguiente este cambiara su poscion haciendo que se ordene de una manera que sea de la A hasta la Z
- El codigo , el commit , la ejecucion seria el siguiente:

Listing 7: Commit

```
$ git commit -m "COMPLETAMOS ESTE METODO QUE NOS PERMITE ORDENAR DE A HASTA Z PARA ESTO
COMPARAMOS EL PRIMER CARACTER DE CADA UNA Y SI ESTA ACTUAL ES MAYOR A LA SIGUIENTE
ESTA CAMBIARA DE POSICION"
```

Listing 8: Las líneas de código del método completado:

```
public static void ordenarPorNombreBurbuja(Nave[] flota){ //COMPLETAMOS ESTE METODO QUE
    NOS PERMITE ORDENAR DE A HASTA Z PARA ESTO COMPARAMOS EL PRIMER CARACTER DE CADA
    UNA Y SI ESTA ACTUAL ES MAYOR A LA SIGUIENTE ESTA CAMBIARA DE POSICION
    for(int i = 0; i < flota.length - 1;i++){
        if(flota[i].getNombre().charAt(0) > flota[i + 1].getNombre().charAt(0)){
            Nave temp = flota[i];
            flota[i] = flota[i + 1];
            flota[i + 1] = temp;
        }
    }
}
```

Listing 9: La ejecución dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: A
Estado: true
Puntos: 34
Nave 2 :
Nombre: Pepe
Fila: 3
Columna: E
Estado: true
Puntos: 76
Nave 3 :
Nombre: Lalo
Fila: 6
Columna: G
Estado: true
Puntos: 874

Ordenado por las iniciales de cada nombre de A a la Z mediante el metodo burbuja:
Nombre: Pepe
Columna: E
Fila: 3
Estado: true
Puntos: 76
*****
Nombre: Lalo
Columna: G
Fila: 6
Estado: true
Puntos: 874
*****
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 34
*****
-----
```

4.4. Ejercicio DemoBatalla01

- En el cuarto commit completamos el metodo ordenarPorPuntosSeleccion donde usamos un for para pasar por todos los elementos y asi poder cambiar con el indice del proximo menor y en el otro metodo creado nos permite sacar el indice de este proximo menor de puntos usando un for pasando por todos elementos posteriores al actual y asi siguiendo.
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 10: Commit

```
$ git commit -m "METODO COMPLETADO QUE NOS PERMITE CAMBIAR LAS POSICIONES DE CADA
ARREGLO DEPENDIENDO DE LO QUE RETORNE EL METODO indProxMin que sera el indice cual
debemos cambiar con el actual y tambien METODO CREADO QUE NOS AYUDA A BUSCAR EL
INDICE DEL OBJETO Y NOS DICE CUAL ES EL PROXIMO MENOR APARTIR DEL QUE ESTAMOS Y VA
PASANDO POR TODOS LOS ELEMENTOS ASI QUE VA ACTUALIZANDOSE LA VARIABLE MINDEX"
```

Listing 11: Las lineas de codigo del metodo completado:

```
public static void ordenarPorPuntosSeleccion(Nave[] flota){ //METODO COMPLETADO QUE NOS
    PERMITE CAMBIAR LAS POSICIONES DE CADA ARREGLO DEPENDIENDO DE LO QUE RETORNE EL
    METODO indProxMin que sera el indice cual debemos cambiar con el actual
    for(int i = 0; i < flota.length - 1; i++){
        int j = indProxMin(flota, i);
        Nave temp = flota[i];
        flota[i] = flota[j];
        flota[j] = temp;
    }
}

public static int indProxMin(Nave[] flota, int index){ //METODO CREADO QUE NOS AYUDA A
    BUSCAR EL INDICE DEL OBJETO Y NOS DICE CUAL ES EL PROXIMO MENOR APARTIR DEL QUE
    ESTAMOS Y VA PASANDO POR TODOS LOS ELEMENTOS ASI QUE VA ACTUALIZANDOSE LA VARIABLE
    MINDEX
    int minindex = index;
    for(int i = index + 1; i < flota.length; i++){
        if(flota[i].getPuntos() < flota[minindex].getPuntos()){
            minindex = i;
        }
    }
    return minindex;
}
```

Listing 12: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: D
Estado: true
Puntos: 753
Nave 2 :
Nombre: Cristian
Fila: 4
Columna: B
Estado: true
```

```
Puntos: 234
Nave 3 :
Nombre: Lalo
Fila:
2
Columna: E
Estado: true
Puntos: 9866

Ordenado por la cantidad de puntos del menor al mayor mediante el metodo seleccion:
Nombre: Cristian
Columna: B
Fila: 4
Estado: true
Puntos: 234
*****
Nombre: Victor
Columna: D
Fila: 2
Estado: true
Puntos: 753
*****
Nombre: Lalo
Columna: E
Fila: 2
Estado: true
Puntos: 9866
*****
```

4.5. Ejercicio DemoBatalla01

- En el quinto commit completamos el metodo ordenarPorNombreSeleccion() donde usamos un for para pasar por todos los elementos y asi poder cambiar con el indice del proximo menor de las iniciales de los nombres y en el otro metodo creado nos permite sacar el indice de este proximo menor de los indices de los nombres usando un for pasando por todos elementos posteriores al actual y asi siguiendo.
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 13: Commit

```
$ git commit -m "METODO COMPLETADO QUE NOS PERMITE CAMBIAR LAS POSICIONES DE CADA
ARREGLO DEPENDIENDO DE LO QUE RETORNE EL METODO indProxMin que sera el indice cual
debemos cambiar con el actual y tambien METODO CREADO QUE NOS AYUDA A BUSCAR EL
INDICE DEL OBJETO Y NOS DICE CUAL ES EL PROXIMO MENOR EN TERMINOS DE A HASTA Z
APARTIR DEL QUE ESTAMOS Y VA PASANDO POR TODOS LOS ELEMENTOS ASI QUE VA
ACTUALIZANDOSE LA VARIABLE MINDEX"
```

Listing 14: Las lineas de codigo del metodo completado:

```
public static void ordenarPorNombreSeleccion(Nave[] flota){ //METODO COMPLETADO QUE NOS
    PERMITE CAMBIAR LAS POSICIONES DE CADA ARREGLO DEPENDIENDO DE LO QUE RETORNE EL
    METODO indProxMin que sera el indice cual debemos cambiar con el actual
    for(int i = 0; i < flota.length - 1; i++){
```

```
        int j = indProxMin02(flota, i);
        Nave temp = flota[i];
        flota[i] = flota[j];
        flota[j] = temp;
    }
}

public static int indProxMin02(Nave[] flota, int index){ //METODO CREADO QUE NOS AYUDA
    A BUSCAR EL INDICE DEL OBJETO Y NOS DICE CUAL ES EL PROXIMO MENOR EN TERMINOS DE A
    HASTA Z APARTIR DEL QUE ESTAMOS Y VA PASANDO POR TODOS LOS ELEMENTOS ASI QUE VA
    ACTUALIZANDOSE LA VARIABLE MININDEX
    int minindex = index;
    for(int i = index + 1; i < flota.length; i++){
        if(flota[i].getNombre().charAt(0) < flota[minindex].getNombre().charAt(0)){
            minindex = i;
        }
    }
    return minindex;
}
```

Listing 15: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: D
Estado: true
Puntos: 753
Nave 2 :
Nombre: Cristian
Fila: 4
Columna: B
Estado: true
Puntos: 234
Nave 3 :
Nombre: Lalo
Fila:
2
Columna: E
Estado: true
Puntos: 9866

Ordenado por las iniciales de cada nombre de A a la Z mediante el metodo seleccion:
Nombre: Cristian
Columna: B
Fila: 4
Estado: true
Puntos: 234
*****
Nombre: Lalo
Columna: E
Fila: 2
Estado: true
Puntos: 9866
*****
Nombre: Victor
Columna: D
```



```
Fila: 2
Estado: true
Puntos: 753
*****
```

4.6. Ejercicio DemoBatalla01

- En el sexto commit completamos este metodo ordenarPorPuntosInsercion() onsisite en recorrer todo el array comenzando desde el segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en el lugar correcto entre todos los elementos con menor punto anteriores a él y asi por cada uno completando del mayor al menor
- El codigo , el commit y la ejecucion seria el siguiente:

Listing 16: Commit

```
$ git commit -m "Este metodo ordenarPorPuntosInsercion() consiste en recorrer todo el
array comenzando desde el segundo elemento hasta el final. Para cada elemento, se
trata de colocarlo en el lugar correcto entre todos los elementos con menor punto
anteriores a el y asi por cada uno completando del mayor al menor"
```

Listing 17: Las lineas de codigo del metodo creado:

```
public static void ordenarPorPuntosInsercion(Nave[] flota){ //Este metodo
    ordenarPorPuntosInsercion() consiste en recorrer todo el array comenzando desde el
    segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en el
    lugar correcto entre todos los elementos con menor punto anteriores a el y asi por
    cada uno completando del mayor al menor
    for(int i = 1; i < flota.length; i++){
        Nave temp = flota[i];
        int j = i - 1;
        while(j >= 0 && (temp.getPuntos() > flota[j].getPuntos())){
            flota[j + 1] = flota[j];
            j--;
        }
        flota[j + 1] = temp;
    }
}
```

Listing 18: La ejecucion dada:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: D
Estado: true
Puntos: 753
Nave 2 :
Nombre: Cristian
Fila: 4
Columna: B
Estado: true
Puntos: 234
Nave 3 :
```

```
Nombre: Lalo
Fila:
2
Columna: E
Estado: true
Puntos: 9866

Ordenado por la cantidad de puntos del mayor al menor mediante el metodo insercion:
Nombre: Lalo
Columna: E
Fila: 2
Estado: true
Puntos: 9866
*****
Nombre: Victor
Columna: D
Fila: 2
Estado: true
Puntos: 753
*****
Nombre: Cristian
Columna: B
Fila: 4
Estado: true
Puntos: 234
*****
```

4.7. Ejercicio DemoBatalla01

- En el septimo commit el metodo ordenarPorNombreInsercion() consiste en recorrer todo el array comenzando desde el segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en el lugar correcto entre todos los elementos con que sean menores a z anteriores a él y asi por cada uno completando por cada inicial de cada nombre de la Z hasta A
- El codigo , el commit y La ejecucion completa del codigo seria el siguiente:

Listing 19: Commit

```
$ git commit -m "Este metodo ordenarPorNombreInsercion() consiste en recorrer todo el
array comenzando desde el segundo elemento hasta el final. Para cada elemento, se
trata de colocarlo en el lugar correcto entre todos los elementos con que sean
menores a z anteriores a el y asi por cada uno completando por cada inicial de cada
nombre de la Z hasta A"
```

Listing 20: Las lineas de codigo del metodo creado:

```
public static void ordenarPorNombreInsercion(Nave[] flota){ //Este metodo
ordenarPorNombreInsercion() consiste en recorrer todo el array comenzando desde el
segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en el
lugar correcto entre todos los elementos con que sean menores a z anteriores a el y
asi por cada uno completando por cada inicial de cada nombre de la Z hasta A
for(int i = 1; i < flota.length; i++){
    Nave temp = flota[i];
    int j = i - 1;
```

```
while(j >= 0 && (temp.getNombre().charAt(0) > flota[j].getNombre().charAt(0))){  
    flota[j + 1] = flota[j];  
    j--;  
}  
flota[j + 1] = temp;  
}  
}
```

Listing 21: La ejecución del código completo:

```
Nave 1 :  
Nombre: Victor  
Fila: 2  
Columna: D  
Estado: true  
Puntos: 753  
Nave 2 :  
Nombre: Cristian  
Fila: 4  
Columna: B  
Estado: true  
Puntos: 234  
Nave 3 :  
Nombre: Lalo  
Fila:  
2  
Columna: E  
Estado: true  
Puntos: 9866  
  
Ordenado por las iniciales de cada nombre de Z a la A mediante el metodo insercion:  
Nombre: Victor  
Columna: D  
Fila: 2  
Estado: true  
Puntos: 753  
*****  
Nombre: Lalo  
Columna: E  
Fila: 2  
Estado: true  
Puntos: 9866  
*****  
Nombre: Cristian  
Columna: B  
Fila: 4  
Estado: true  
Puntos: 234  
*****
```

4.8. Ejercicio DemoBatalla01

- En el octavo commit creamos Estructura de control creado para mostrar el mensaje de (Nave no encontrada) debido a que esta comparando con los demás nombres de las otras naves

- El código y el commit sería el siguiente:

Listing 22: Commit

```
$ git commit -m "Estructura de control creado para mostrar el mensaje de (Nave no encontrada) debido a que esta comparando con los demás nombres de las otras naves"
```

Listing 23: Las líneas de código de lo creado:

```
if(pos == -1){ //Estructura de control creado para mostrar el mensaje de "Nave no encontrada" debido a que esta comparando con los demás nombres de las otras naves
    System.out.println("Nave no encontrada");
}
```

4.9. Ejercicio DemoBatalla01

- En el noveno commit Método completado que nos permite buscar la nave con el nombre ingresado este funciona en la que ponemos de límites una izquierda y una derecha que son los límites del arreglo y después este se va reduciendo su límite dependiendo en donde se encuentre nuestra nave desada capaz si esta atrás de nuestra mitad esta va reducir su derecha poniendo a esta como la mitad - 1 y si esta adelante de la mitad esta reducir su izquierda poniendo a esta como la mitad + 1 y así sucesivamente y va a parar cuando si la derecha sea menor a la izquierda y nos dará como resultado que la nave no fue encontrada y también antes de usar este método usamos el método ordenarPorNombreSelección(misNaves) usado para poder ordenar las palabras y así que esta pueda ser buscada por la búsqueda binaria ya para que esta sirva todos los nombres deben estar ordenados de la A hasta la Z y también una estructura de control if
- El código, el commit y la ejecución sería el siguiente:

Listing 24: Commit

```
$ git commit -m " Método completado que nos permite buscar la nave con el nombre ingresado este funciona en la que ponemos de límites una izquierda y una derecha que son los límites del arreglo y después este se va reduciendo su límite dependiendo en donde se encuentre nuestra nave desada capaz si esta atrás de nuestra mitad esta va reducir su derecha poniendo a esta como la mitad - 1 y si esta adelante de la mitad esta reducir su izquierda poniendo a esta como la mitad + 1 y así sucesivamente y va a parar cuando si la derecha sea menor a la izquierda y nos dará como resultado que la nave no fue encontrada y también antes de usar este método usamos el método ordenarPorNombreSelección(misNaves) usado para poder ordenar las palabras y así que esta pueda ser buscada por la búsqueda binaria ya para que esta sirva todos los nombres deben estar ordenados de la A hasta la Z y también una estructura de control if"
```

Listing 25: Las líneas de código de lo creado:

```
public static int busquedaBinariaNombre(Nave[] flota, String s){
    return 0;
    int left = 0;
    int right = flota.length - 1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(s.equals(flota[mid].getNombre())){
```

```
        return mid;
    }else if(s.charAt(0) > flota[mid].getNombre().charAt(0)){
        left = mid + 1;
    }else{
        right = mid - 1;
    }
}
return -1;
}
```

Listing 26: Las líneas de código de lo creado:

```
ordenarPorNombreSeleccion(misNaves);
pos=busquedaBinariaNombre(misNaves,searchedname01);
if(pos == -1){ //Estructura de control creado para mostrar el mensaje de "Nave no
    encontrada" debido a que esta buscando y comparando con los demas nombres de las
    otras naves
    System.out.println("Nave no encontrada");
}else{
    System.out.println(misNaves[pos].toString());
}
System.out.println("-----");
```

Listing 27: La ejecución del código completo:

```
Nave 1 :
Nombre: Victor
Fila: 2
Columna: C
Estado: true
Puntos: 7654
Nave 2 :
Nombre: Lalo
Fila: 2
Columna: E
Estado: true
Puntos: 7443
Nave 3 :
Nombre: Tato
Fila: 2
Columna: G
Estado: true
Puntos: 9765

Ingrese el nombre para buscar a la nave:
Victor
*****
Nombre: Victor
Columna: C
Fila: 2
Estado: true
Puntos: 7654
```

4.10. Ejercicio DemoBatalla01

- En el decimo commit Arreglando algunos errores y añadiendo comentarios a los metodos que faltaban y mostramos el codigo completo y su ejecucion
- El codigo y el commit seria el siguiente:

Listing 28: Commit

```
$ git commit -m " Arreglando algunos errores y aadiendo comentarios a los metodos que faltaban"
```

Listing 29: Las lineas de codigo de lo creado:

```
import java.util.*;
public class DemoBatalla01{
    public static void main(String [] args){
        Nave [] misNaves = new Nave [3]; // LE PONEMOS AL ARREGLO UN TAMAO DE 2 PARA SU
        POSTERIOR PRUEBA
        Scanner sc = new Scanner(System.in);
        String nomb, col;
        int fil, punt;
        boolean est;
        for (int i = 0; i < misNaves.length; i++) {
            System.out.println("Nave " + (i+1) + " : ");
            System.out.print("Nombre: ");
            nomb = sc.next();
            System.out.print("Fila: ");
            fil = sc.nextInt();
            System.out.print("Columna: ");
            col = sc.next();
            System.out.print("Estado: ");
            est = sc.nextBoolean();
            System.out.print("Puntos: ");
            punt = sc.nextInt();
            misNaves[i] = new Nave(); //Se crea un objeto Nave y se asigna su referencia a
            misNaves
            misNaves[i].setNombre(nomb);
            misNaves[i].setFila(fil);
            misNaves[i].setColumna(col);
            misNaves[i].setEstado(est);
            misNaves[i].setPuntos(punt);
        }
        System.out.println("\nNaves creadas:");
        System.out.println("-----");
        mostrarNaves(misNaves);
        System.out.println("-----");
        mostrarPorNombre(misNaves);
        System.out.println("-----");
        mostrarPorPuntos(misNaves);
        System.out.println("-----");
        System.out.println("Nave con mayor numero de puntos: \n" +
            mostrarMayorPuntos(misNaves));
        System.out.println("*****");
        System.out.println("-----");
        //leer un nombre
```

```
//mostrar los datos de la nave con dicho nombre, mensaje de no encontrado en caso
contrario
System.out.println("Ingrese el nombre para buscar a la primera nave: ");
String searchedname = sc.next();
int pos = busquedaLinealNombre(misNaves, searchedname);
if(pos == -1){ //Estructura de control creado para mostrar el mensaje de "Nave no
    encontrada" debido a que esta comparando con los demas nombres de las otras
    naves
    System.out.println("Nave no encontrada");
}else{
    System.out.println(misNaves[pos].toString()); // En caso de encontrarlo
    imprimira sus datos de esta nave en caso de no dara mensaja de (Nave no
    encontrada)
}
System.out.println("-----");
System.out.println("Ordenado por la cantidad de puntos del menor al mayor mediante
    el metodo burbuja: ");
ordenarPorPuntosBurbuja(misNaves);
mostrarNaves(misNaves);
System.out.println("-----");
System.out.println("Ordenado por las iniciales de cada nombre de A a la Z mediante
    el metodo burbuja: ");
ordenarPorNombreBurbuja(misNaves);
mostrarNaves(misNaves);
System.out.println("-----");
//mostrar los datos de la nave con dicho nombre, mensaje de no encontrado en caso
contrario
System.out.println("Ingrese el nombre para buscar a la nave: ");
String searchedname01 = sc.next();
System.out.println("*****");
ordenarPorNombreSeleccion(misNaves); //Metodo usado para poder ordenar las
    palabras y asi que esta pueda ser buscada por la busqueda binaria ya para que
    esta sirva todos los nombres deben estar ordenados de la A hasta la Z
pos=busquedaBinariaNombre(misNaves,searchedname01);
if(pos == -1){ //Estructura de control creado para mostrar el mensaje de "Nave no
    encontrada" debido a que esta buscando y comparando con los demas nombres de
    las otras naves
    System.out.println("Nave no encontrada");
}else{
    System.out.println(misNaves[pos].toString()); // En caso de encontrarlo
    imprimira sus datos de esta nave en caso de no dara mensaja de (Nave no
    encontrada)
}
System.out.println("-----");
System.out.println("Ordenado por la cantidad de puntos del menor al mayor mediante
    el metodo seleccion: ");
ordenarPorPuntosSeleccion(misNaves);
mostrarNaves(misNaves);
System.out.println("-----");
System.out.println("Ordenado por la cantidad de puntos del mayor al menor mediante
    el metodo insercion: ");
ordenarPorPuntosInsercion(misNaves);
mostrarNaves(misNaves);
System.out.println("-----");
System.out.println("Ordenado por las iniciales de cada nombre de A a la Z mediante
    el metodo seleccion: ");
```

```
ordenarPorNombreSeleccion(misNaves);
mostrarNaves(misNaves);
System.out.println("-----");
System.out.println("Ordenado por las iniciales de cada nombre de Z a la A mediante
    el metodo insercion: ");
ordenarPorNombreInsercion(misNaves);
mostrarNaves(misNaves);
}
//Mtodo para mostrar todas las naves
public static void mostrarNaves(Nave [] flota){ //COMPLETAMOS EL METODO mostrarNaves
    Y NOS AYUDAMOS DE UN FOR EACH PARA EL MUESTREO DE LOS DATOS DE CADA OBJETO Y
    TAMBIEN USAMOS EL METODO toString()
    for(Nave ship: flota){
        System.out.println(ship.toString());
        System.out.println("*****");
    }
}
//Mtodo para mostrar todas las naves de un nombre que se pide por teclado
public static void mostrarPorNombre(Nave [] flota){ //COMPLETAMOS EL METODO
    mostrarPorNombre Y NOS AYUDAMOS DE UN FOR EACH CON TAL QUE SI EL NOMBRE
    INGRESADO ERA IGUAL AL OBJETO CREADO MOSTRABA LOS DATOS DEL OBJETO Y TAMBIEN
    USAMOS EL METODO toString()
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese el nombre para buscar a las naves: ");
    String name = sc.next();
    System.out.println("*****");
    for(Nave ship: flota){
        if(ship.getNombre().equals(name)){
            System.out.println(ship.toString());
            System.out.println("*****");
        }
    }
}
//Mtodo para mostrar todas las naves con un nmero de puntos inferior o igual
//al nmero de puntos que se pide por teclado
public static void mostrarPorPuntos(Nave [] flota){ // Completamos el metodo
    mostrarPorPuntos donde ingresamos un numero de puntos en esta usamos un for each
    que pase por todos los elementos donde si su numero de puntos es menor o igual a
    este numero ingresado se imprimira sus datos
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese un nmero puntos para buscar a las naves que son menor
        o igual a esta: ");
    int point = sc.nextInt();
    System.out.println("*****");
    for(Nave ship: flota){
        if(ship.getPuntos() <= point){
            System.out.println(ship.toString());
            System.out.println("*****");
        }
    }
}
//Mtodo que devuelve la Nave con mayor nmero de Puntos
public static Nave mostrarMayorPuntos(Nave [] flota){ //COMPLETAMOS ESTE METODO
    mostrarMayorPuntos DONDE CREAMOS UN OBJETO DE LA CLASE NAVE QUE ES SHIP EN ESTE
    PODREMOS GUARDAR LOS DATOS DE LA NAVE CON LA MAYOR CANTIDAD DE PUNTOS Y DESPUES
    RETONARNLO
```



```
Nave ship = new Nave();
for(int i = 0; i < flota.length - 1; i++){
    if(flota[i].getPuntos() < flota[i + 1].getPuntos()){
        ship = flota[i + 1];
    }else{
        ship = flota[i];
    }
}
return ship;
}

//Mtodo para buscar la primera nave con un nombre que se pidi por teclado
public static int busquedaLinealNombre(Nave[] flota, String s){ //METODO CREADO PARA
    LA BUSQUEDA DE LA PRIMERA NAVE QUE ES IGUAL AL NOMBRE QUE ESCRIBIMOS
    System.out.println("*****");
    for(int i = 0; i < flota.length; i++){
        if(flota[i].getNombre().equals(s)){
            return i;
        }
    }
    return -1;
}

//Mtodo que ordena por nmero de puntos de menor a mayor
public static void ordenarPorPuntosBurbuja(Nave[] flota){ //COMPLETAMOS ESTE METODO
    QUE NOS PERMITE ORDENAR DE MENOR A MAYOR DE LA MANERA DE ORDENAR BURBUJA QUE
    SERIA CAMBIAR POSCIONES SI TU ELEMENTO DE ADELANTE ES MENOR AL ACTUAL
    for(int i = 0; i < flota.length - 1; i++){
        if(flota[i].getPuntos() > flota[i + 1].getPuntos()){
            Nave temp = flota[i];
            flota[i] = flota[i + 1];
            flota[i + 1] = temp;
        }
    }
}

//Mtodo que ordena por nombre de A a Z
public static void ordenarPorNombreBurbuja(Nave[] flota){ //COMPLETAMOS ESTE METODO
    QUE NOS PERMITE ORDENAR DE A HASTA Z PARA ESTO COMPARAMOS EL PRIMER CARACTER DE
    CADA UNA Y SI ESTA ACTUAL ES MAYOR A LA SIGUIENTE ESTA CAMBIARA DE POSICION
    for(int i = 0; i < flota.length - 1; i++){
        if(flota[i].getNombre().charAt(0) > flota[i + 1].getNombre().charAt(0)){
            Nave temp = flota[i];
            flota[i] = flota[i + 1];
            flota[i + 1] = temp;
        }
    }
}

//Mtodo para buscar la primera nave con un nombre que se pidi por teclado
public static int busquedaBinariaNombre(Nave[] flota, String s){ // Metodo
    completado que nos permite buscar la nave con el nombre ingresado este funciona
    en la que ponemos de limites una izquierda y una derecha que son los limites del
    arreglo y despues este se va reduciendo su limte dependiendo en donde se
    encuentre nuestra nave desada capaz si esta atras de nuestra mitad esta va
    reducir su derecha poniendo a esta como la mitad - 1 y si esta adelante de la
    mitad esta reducira su izquierda poniendo a esta como la mitad + 1 y asi
    sucesivamente y va a parar cuando si la derecha sea menor a la izquierda y nos
    dara como resultado que la nave no fue encontrada
    int left = 0;
```

```
int right = flota.length - 1;
while(left <= right) {
    int mid = left + (right - left) / 2;
    if(s.equals(flota[mid].getNombre())){
        return mid;
    }else if(s.charAt(0) > flota[mid].getNombre().charAt(0)){
        left = mid + 1;
    }else{
        right = mid - 1;
    }
}
return -1;
}

// Mtodo que ordena por nmero de puntos de menor a mayor
public static void ordenarPorPuntosSeleccion(Nave[] flota){ //METODO COMPLETADO QUE
    NOS PERMITE CAMBIAR LAS POSICIONES DE CADA ARREGLO DEPENDIENDO DE LO QUE RETORNE
    EL METODO indProxMin que sera el indice cual debemos cambiar con el actual
    for(int i = 0; i < flota.length - 1; i++){
        int j = indProxMin(flota, i);
        Nave temp = flota[i];
        flota[i] = flota[j];
        flota[j] = temp;
    }
}

public static int indProxMin(Nave[] flota, int index){ //METODO CREADO QUE NOS AYUDA
    A BUSCAR EL INDICE DEL OBJETO Y NOS DICE CUAL ES EL PROXIMO MENOR APARTIR DEL
    QUE ESTAMOS Y VA PASANDO POR TODOS LOS ELEMENTOS ASI QUE VA ACTUALIZANDOSE LA
    VARIABLE MINDEX
    int minindex = index;
    for(int i = index + 1; i < flota.length; i++){
        if(flota[i].getPuntos() < flota[minindex].getPuntos()){
            minindex = i;
        }
    }
    return minindex;
}

// Mtodo que ordena por nombre de A a Z
public static void ordenarPorNombreSeleccion(Nave[] flota){ //METODO COMPLETADO QUE
    NOS PERMITE CAMBIAR LAS POSICIONES DE CADA ARREGLO DEPENDIENDO DE LO QUE RETORNE
    EL METODO indProxMin que sera el indice cual debemos cambiar con el actual
    for(int i = 0; i < flota.length - 1; i++){
        int j = indProxMin02(flota, i);
        Nave temp = flota[i];
        flota[i] = flota[j];
        flota[j] = temp;
    }
}

public static int indProxMin02(Nave[] flota, int index){ //METODO CREADO QUE NOS
    AYUDA A BUSCAR EL INDICE DEL OBJETO Y NOS DICE CUAL ES EL PROXIMO MENOR EN
    TERMINOS DE A HASTA Z APARTIR DEL QUE ESTAMOS Y VA PASANDO POR TODOS LOS
    ELEMENTOS ASI QUE VA ACTUALIZANDOSE LA VARIABLE MINDEX
    int minindex = index;
    for(int i = index + 1; i < flota.length; i++){
        if(flota[i].getNombre().charAt(0) < flota[minindex].getNombre().charAt(0)){
            minindex = i;
        }
    }
}
```

```
    }  
    return minindex;  
}  
// Mtodo que muestra las naves ordenadas por nmero de puntos de mayor a menor  
public static void ordenarPorPuntosInsercion(Nave[] flota){ //Este metodo  
    ordenarPorPuntosInsercion() consiste en recorrer todo el array comenzando desde  
    el segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en  
    el lugar correcto entre todos los elementos con menor punto anteriores a l y asi  
    por cada uno completando del mayor al menor  
    for(int i = 1; i < flota.length; i++){  
        Nave temp = flota[i];  
        int j = i - 1;  
        while(j >= 0 && (temp.getPuntos() > flota[j].getPuntos())){  
            flota[j + 1] = flota[j];  
            j--;  
        }  
        flota[j + 1] = temp;  
    }  
}  
// Mtodo que muestra las naves ordenadas por nombre de Z a A  
public static void ordenarPorNombreInsercion(Nave[] flota){ //Este metodo  
    ordenarPorNombreInsercion() consiste en recorrer todo el array comenzando desde  
    el segundo elemento hasta el final. Para cada elemento, se trata de colocarlo en  
    el lugar correcto entre todos los elementos con que sean menores a z anteriores  
    a l y asi por cada uno completando por cada inicial de cada nombre de la Z hasta A  
    for(int i = 1; i < flota.length; i++){  
        Nave temp = flota[i];  
        int j = i - 1;  
        while(j >= 0 && (temp.getNombre().charAt(0) > flota[j].getNombre().charAt(0))){  
            flota[j + 1] = flota[j];  
            j--;  
        }  
        flota[j + 1] = temp;  
    }  
}  
}
```

Listing 30: La ejecucion del codigo completo:

```
Nave 1 :  
Nombre: Victor  
Fila: 2  
Columna: A  
Estado: true  
Puntos:  
5674  
Nave 2 :  
Nombre: Lalo  
Fila: 3  
Columna: E  
Estado: true  
Puntos: 23432  
Nave 3 :  
Nombre: Pepe  
Fila: 3  
Columna: F
```

```
Estado: true
Puntos: 456456

Naves creadas:
-----
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
Nombre: Lalo
Columna: E
Fila: 3
Estado: true
Puntos: 23432
*****
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
-----
Ingrese el nombre para buscar a las naves:
Victor
*****
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
-----
Ingrese un nmero puntos para buscar a las naves que son menor o igual a esta:
756758
*****
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
Nombre: Lalo
Columna: E
Fila: 3
Estado: true
Puntos: 23432
*****
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
-----
```

```
Nave con mayor numero de puntos:
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
-----

Ingrese el nombre para buscar a la primera nave:
Victor
*****
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
-----

Ordenado por la cantidad de puntos del menor al mayor mediante el metodo burbuja:
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
Nombre: Lalo
Columna: E
Fila: 3
Estado: true
Puntos: 23432
*****
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
-----

Ordenado por las iniciales de cada nombre de A a la Z mediante el metodo burbuja:
Nombre: Lalo
Columna: E
Fila: 3
Estado: true
Puntos: 23432
*****
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
```

Ingrese el nombre para buscar a la nave:

Lalo

Nombre: Lalo

Columna: E

Fila: 3

Estado: **true**

Puntos: 23432

Ordenado por la cantidad de puntos del menor al mayor mediante el metodo seleccion:

Nombre: Victor

Columna: A

Fila: 2

Estado: **true**

Puntos: 5674

Nombre: Lalo

Columna: E

Fila: 3

Estado: **true**

Puntos: 23432

Nombre: Pepe

Columna: F

Fila: 3

Estado: **true**

Puntos: 456456

Ordenado por la cantidad de puntos del mayor al menor mediante el metodo insercion:

Nombre: Pepe

Columna: F

Fila: 3

Estado: **true**

Puntos: 456456

Nombre: Lalo

Columna: E

Fila: 3

Estado: **true**

Puntos: 23432

Nombre: Victor

Columna: A

Fila: 2

Estado: **true**

Puntos: 5674

Ordenado por las iniciales de cada nombre de A a la Z mediante el metodo seleccion:

Nombre: Lalo

Columna: E

Fila: 3

Estado: **true**

Puntos: 23432

```
*****
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
-----
Ordenado por las iniciales de cada nombre de Z a la A mediante el metodo insercion:
Nombre: Victor
Columna: A
Fila: 2
Estado: true
Puntos: 5674
*****
Nombre: Pepe
Columna: F
Fila: 3
Estado: true
Puntos: 456456
*****
Nombre: Lalo
Columna: E
Fila: 3
Estado: true
Puntos: 23432
*****
```

4.11. Estructura de laboratorio 04

- El contenido que se entrega en este laboratorio04 es el siguiente:

```
/Lab04
|----DemoBatalla01.java
|----Latex
|    |----img
|    |    |----logo_abet.png
|    |    |----logo_episunsa.png
|    |    |----logo_unsa.jpg
|    |    +----pseudocodigo_insercion.png
|    |----Informe04.aux
|    |----Informe04.fdb_latexmk
|    |----Informe04.flx
|    |----Informe04.log
|    |----Informe04.out
|    |----Informe04.pdf
|    |----Informe04.synctex.gz
|    +----Informe04.tex
+----Nave.java
```

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 cumple 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 los ítems.
- 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	1	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	1	
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 están dentro de los plazos de fecha de entrega establecidos.	2	X	0.5	
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	2	
Total		20		12.5	

6. Referencias

- https://drive.google.com/file/d/1CoQAKeKW-QDYRmHLrBdbSopFB1Z_Qmk3/view