



**Tecnológico Nacional de México
Campus Orizaba**

Estructura de Datos

Ingeniería en Sistemas Computacionales

**Tema 1:
Introducción a las estructuras de datos**

Integrantes:

**Castillo Solís Luis Ángel – 21010932
Muñoz Hernández Vania Lizeth – 21011009
Romero Ovando Karyme Michelle – 21011037**

**Grupo:
3g2B**

Fecha de entrega: 17/Abril /2023

1. Introducción

La programación se enfoca en la organización y manipulación de datos de diferentes tipos y tamaños. Es aquí donde entran en juego las estructuras de datos, que son formas específicas de almacenar y organizar datos en una computadora.

Se utilizan los datos de forma conjunta, la forma como estos datos serán agregados y organizados depende mucho de cómo serán utilizados y procesados, es decir, su búsqueda, los datos trabajados, su implementación y como los datos se relacionan. Se puede afirmar que un programa está compuesto de algoritmos y estructuras de datos, que juntos hacen que el programa funcione correctamente.

Los algoritmos manipulan los datos en estas estructuras de varias formas, para buscar un dato particular u ordenar los datos.

Los TDA son una herramienta importante en la programación, ya que permiten a los programadores abstraerse de los detalles internos de las estructuras de datos, lo que facilita la escritura de programas complejos. Además, el manejo de memoria es un aspecto crucial de la programación, ya que se debe tener en cuenta cómo se están asignando y liberando los recursos en la memoria de la computadora.

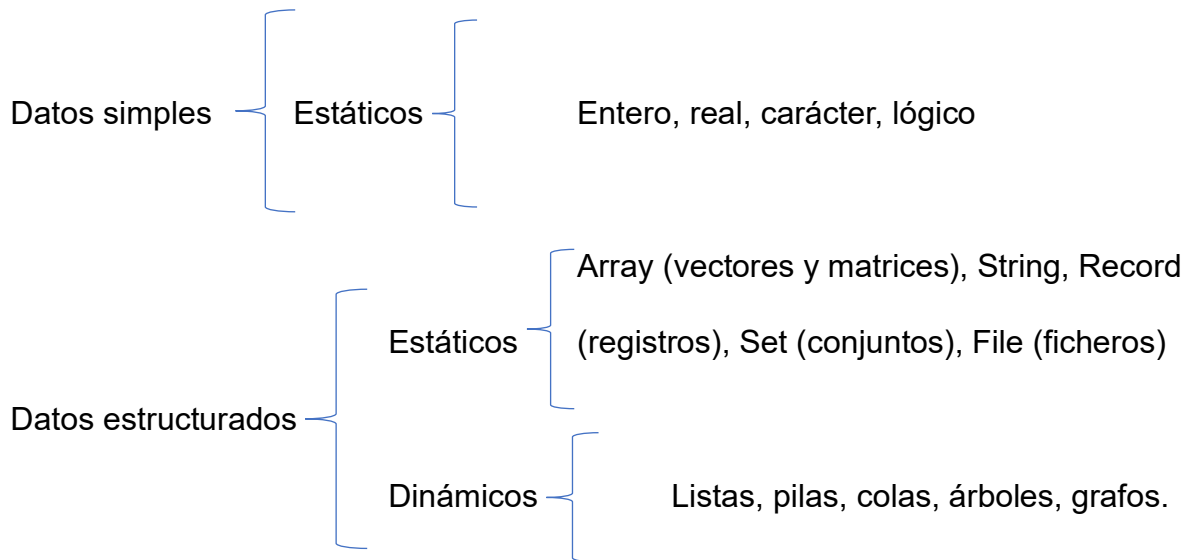
En resumen, el conocimiento y aplicación de las estructuras de datos, los tipos de datos abstractos y el manejo de memoria son fundamentales para cualquier programador que busque desarrollar programas eficientes.

2. Competencia específica

Conoce, comprende y aplica eficientemente estructuras de datos, métodos de ordenamiento y búsqueda para la optimización del rendimiento de soluciones a problemas del mundo real.

3. Marco Teórico

Las estructuras de datos, como sabemos, son un conjunto de elementos de datos organizados, a los que por medio de operaciones es posible acceder y/o modificarlos. Podemos clasificarlos de la siguiente manera:



Los tipos de datos simples son aquellos que almacenan un único valor en una variable en un momento dado. Las estructuras de datos estáticas tienen un tamaño máximo definido antes de la ejecución del programa y no se pueden modificar durante la misma. Finalmente, los punteros son un tipo de datos que permiten la creación de estructuras de datos dinámicas.

Por otro lado, la abstracción de datos es la separación de las propiedades lógicas de los tipos de datos de su implementación. El tipo de dato abstracto es un tipo de dato definido por el usuario y se compone de dos partes: una parte pública que especifica como manipular los objetos del tipo y una parte privada que es usada internamente por el tipo para implementar el comportamiento especificado por la interfase pública. El TDA es un tipo de dato cuyas propiedades son especificadas independientemente de cualquier implementación particular.

Ahora bien, nuestros programas siempre ocuparán un espacio de memoria, de esta forma es los lenguajes de programación modernos proporcionan funciones de gestión de memoria para ayudar a los programadores a administrar la memoria de manera eficiente, incluyendo la asignación y liberación de memoria dinámica y el uso de recolectores de basura automáticos que liberan la memoria no utilizada. La comprensión del manejo de memoria en programación es esencial para crear programas eficientes y seguros. La memoria del ordenador se divide en diferentes áreas, cada una con un propósito específico:

- La memoria estática, se utiliza para almacenar variables globales y está reservada durante toda la ejecución del programa.
- La memoria dinámica, por otro lado, se reserva y libera durante la ejecución del programa y se utiliza para almacenar datos de tamaño desconocido o variables que cambian de tamaño durante la ejecución del programa.

Adentrándonos un poco más en cada uno de estos conceptos, podemos describir a la memoria estática como la porción de memoria que se asigna durante la compilación antes de que el programa se ejecute. Los objetos se crean en ese momento y se destruyen al finalizar el programa, y permanecen en la misma ubicación de memoria durante toda la ejecución del programa.

Los objetos administrados de ese modo son:

- Variables static
- Variables globales
- Miembros static de clases
- Literales cualquier tipo

Por otro lado, durante la ejecución del programa, la memoria dinámica se solicita mediante el uso de punteros. Conforme se necesitan más líneas de código, se pide más memoria al sistema operativo para su almacenamiento. La estructura de datos se ajusta de forma precisa a los requerimientos del programa, evitando la pérdida de datos o el uso innecesario de memoria. El programa se divide en cuatro partes: texto, datos estáticos, pila y una zona libre o heap. Es crucial liberar la memoria

inservible para evitar el agotamiento de la fuente de memoria dinámica. El tamaño de la pila cambia de manera dinámica según lo determinado por el sistema operativo.

Es importante administrar adecuadamente la memoria en un programa para evitar errores como la pérdida de memoria, la fragmentación de memoria y los desbordamientos de memoria, que pueden provocar el bloqueo o el fallo del programa.

4. Material y Equipo

El material y equipo que se necesita para llevar a cabo la práctica son:

- ✓ Computadora
- ✓ Software y versión usados
- ✓ Materiales de apoyo para el desarrollo de la práctica

5. Desarrollo de la practica

Pasos para desarrollar la practica:

1. Abrir el programa que sea de su agrado como Eclipse, NetBeans u otros
2. Crea un nuevo proyecto con el nombre de su agrado.
3. Cuando ya se creo el proyecto, de debe hacer lo siguiente:


 Los pasos para desarrollar la clase ArrayOperación son los siguientes:

1. Abrir el programa que sea de su agrado como Eclipse, NetBeans u otros.
2. Crear un proyecto nuevo.
3. Crear una clase en este caso se le puso el nombre de ArrayOperacion.
4. Teniendo la clase, es momento de declarar los atributos que serán de tipo privado, el primero será “dato” que es un array de tipo entero y “J” que es de tipo byte, está va a indicar la ultima posición ocupada en el arreglo.
5. Hacer un constructor de la clase que reciba como parámetro un entero de nombre “tam” para indicar el tamaño del array “datos”. Se crea un nuevo arreglo de “datos” de tamaño “tam” y establecer J-1.

6. Crear métodos de tipo público los cuales son: `arrayVacio()`, `agregarDatos()`, `imprimirArray()`, `buscarSecuencial(int dato)`, `eliminarDato(byte pos)` y `modificarDato(byte pos)`.
7. El metodo `arrayVacio()`, va a devolver true si J es igual a -1, de lo contrario, será false. Para el método de `agregarDatos()`, se verifica si J es menor que el tamaño del arreglo menos uno. Si es así, solicita al usuario que ingrese un valor entero y lo ingresa al arreglo en la posición J+1. Si J es igual al tamaño menos uno, se muestra un mensaje de error indicando que el arreglo está lleno.
8. Con el método de `imprimirArray()`, se crea una cadena, recorre a través del arreglo y agrega cada valor al final de la cadena, al final muestra una ventana con la cadena.
9. En el método de `buscarSecuencial(int dato)`, recorre a través del arreglo y verifica si el dato es igual a `datos[i]`. Si es así devuelve el valor de i, sino incrementa en uno y continua recorriendo. Si llega al final del arreglo y no se ha encontrado el dato, devuelve -1.
10. Con el método de `eliminarDato(byte pos)`, se recorre el arreglo y mueve cada valor una posición hacia la izquierda, luego decrementa J en uno. En el método de `modificarDato(byte pos)`, solicita al usuario un nuevo valor y lo almacena en la posición pos del arreglo datos.

- Precauciones o advertencias.

1. Verificar que el tamaño del arreglo “datos” sea mayor que cero.
2. Cuando se agreguen elementos al arreglo, es importante verificar si el arreglo ya está lleno antes de agregar un nuevo elemento para evitar errores.
3. Cuando se eliminan elementos del array, es importante verificar si la posición indicada existe en el arreglo antes de eliminar el elemento para evitar errores.
4. Al modificar elementos del array, se tiene que verificar si la posición indicada existe en el arreglo antes de modificarlo para evitar errores.
5. Es importante validar los valores ingresados por el usuario en los métodos que solicita.
6. Tener cuidado al usar el método `eliminarDato(byte pos)` ya que puede causar errores si se pasa una posición que no existe en el arreglo “datos”.

 Los pasos para desarrollar la clase DatosOrdenados son los siguientes:

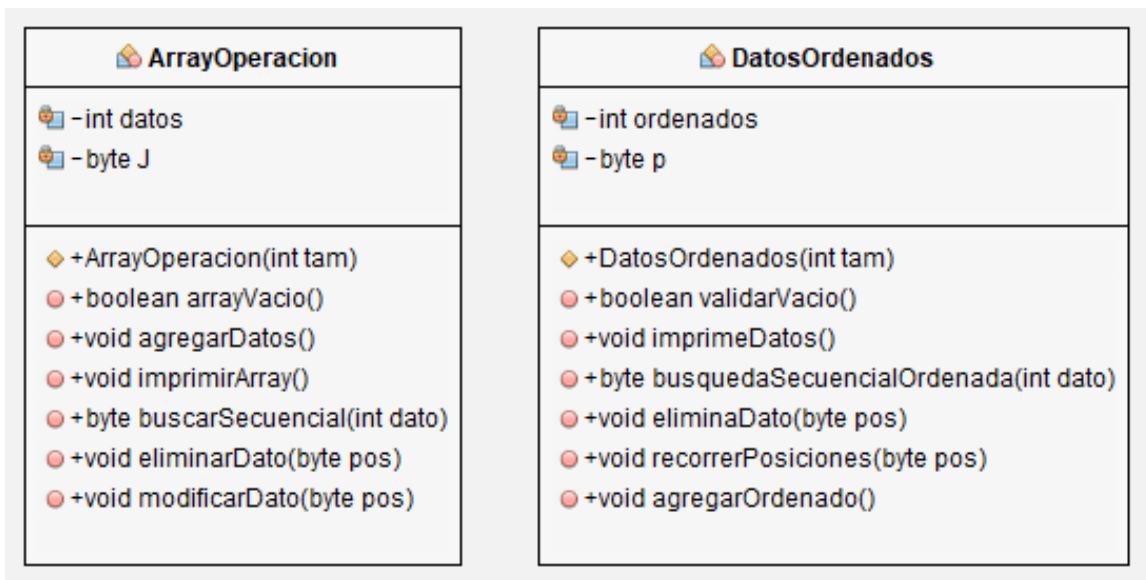
1. En el mismo proyecto se va a crear una nueva clase, en este caso llamada “DatosOrdenados”.

2. Se declaran las variables necesarias para la clase de tipo privada, en este caso, un arreglo de tipo int llamado "ordenados" y una variable llamada "p" de tipo byte que va a indicar cuantos valores se han insertado en el arreglo.
3. Se crea un constructor que va a recibir como parámetro un entero llamado tam para definir el tamaño del arreglo. En este constructor se inicializa el arreglo con el tamaño y se asigna el valor de -1 para "p" para indicar que el arreglo está vacío.
4. Con el método validarVacio(), que devuelve true si el arreglo está vacío y false en caso contrario.
5. El método imprimeDatos(), imprime una en una ventana emergente los valores del arreglo en orden.
6. El método busquedaSecuencialOrdenada(), recibe un parámetro entero y busca el valor en el arreglo ordenado. Este método utiliza un algoritmo de búsqueda secuencial para encontrar el valor en el arreglo y devuelve la posición en la que se encuentra.
7. Con el método de eliminaDato() va a recibir un parámetro byte que indica la posición del valor a eliminar en el arreglo. Este método recorre los valores posteriores a esa posición una casilla hacia atrás y disminuye en uno el valor de "p".
8. Para el metodo de recorrerPosiciones(), recibe un parámetro byte que indica la posición de la cual se deben recorrer las posiciones del arreglo. Con este método se mueven todos los valores posteriores a esa posición una casilla hacia adelante, dejando la casilla vacía en la posición indicada.
9. Con el método de agregarOrdenado, solicita al asuario un valor entero y lo inserta en el arreglo en la posición correspondiente para que se pueda mantener el orden, si el valor ya existe en el arreglo, se va a mostrar un mensaje de advertencia y no se agrega el valor.

- Precauciones o advertencias.

1. Cuando se inserta un valor en el arreglo, se debe mantener el orden para que el método de busqueda secuencial funcione correctamente.
2. Es necesario validar si el arreglo está vacío antes de realizar cualquier operación que dependa de los valores del arreglo.
3. Se deben realizar las ejecuciones del programa para asegurarnos de que los métodos funcionan correctamente.

➤ Diagrama de clase.



6. Resultados

🚦 **Datos Desordenados:**

Clase

Clase Array Operación:

```
1 package Metodos;
2 import EntradaSalida.Tools;
3 import javax.swing.JOptionPane;
4
5 public class ArrayOperacion {
6     private int datos[];
7     private byte J;
8
9     public ArrayOperacion(int tam) {
10         datos = new int[tam];
11         J = -1;
12     }
13
14     public boolean arrayVacio() {
15         return (J == -1);
16     }
17
18     public void agregarDatos() {
19         if (J < datos.length - 1) {
20             datos[J + 1] = Tools.leeEntero
21             (msg: "Escribe un valor: ");
22             J++;
23         } else {
24             JOptionPane.showMessageDialog
25             (parentComponent: null, message: "Array lleno.");
26         }
27     }
28 }
```



```

29 public void imprimirArray(){
30     String cad = "";
31     for (int i = 0; i <= J; i++) {
32         cad += "[" + i + "]" + " - " + datos[i] + "\n";
33     }
34     JOptionPane.showMessageDialog(parentComponent: null,
35         "Datos del Array:" + "\n" + cad);
36 }
37
38 public byte buscarSecuencial(int dato){
39     byte i =0;
40     while(i <= J && dato!= datos[i])
41         i++;
42     if(i<=J){
43         return i;
44     }
45     else{
46         return (-1);
47     }
48 }
49
50 public void eliminarDato(byte pos){
51     for (int k = pos; k <= J; k++) {
52         datos[k] = datos[k+1];
53     }
54     J--;
55 }
56
57 public void modificarDato(byte pos){
58     byte y = Tools.leeByte(msg: "Ingrese Nuevo Numero: ");
59     datos[pos] = y;
60 }

```

Funcionamiento

En esta clase se crearon siete métodos para poder manejar los datos desordenados. Como su nombre lo dice, estos datos no necesitan un orden a la hora de insertarlos, para ello, se necesita un array en donde se irán almacenando. En este tema son de gran importancia los métodos como: validar si el array está vacío, agregar datos, imprimir array, buscar secuencial, eliminar dato y modificar datos. Estos métodos son importantes para poder realizar nuestros proyectos sobre datos desordenados.

Datos Ordenados.

Clase

```
ArrayOperacion.java x DatosOrdenados.java x
Source History

1 package Metodos;
2 import EntradaSalida.Tools;
3 import javax.swing.JOptionPane;
4
5 public class DatosOrdenados {
6     private int ordenados[];
7     private byte p; // cuantos valores se van insertando
8
9     public DatosOrdenados(int tam){
10         ordenados = new int[tam]; // se crea en tiempo de ejecucion
11         p = -1; // orden de cada arreglo, si p-1 es pq el arreglo esta vacio
12     }
13
14     public boolean validarVacio(){
15         return (p== -1); // retorna true o false
16     }
17
18     public void imprimeDatos(){
19         String cad = "";
20         for (int i = 0; i <= p; i++) {
21             cad += "[" + i + "]" + " - " + ordenados[i] + "\n";
22         }
23         JOptionPane.showMessageDialog(parentComponent: null,
24             "Datos del Array:" + "\n" + cad);
25     }
26
27     public byte busquedaSecuencialOrdenada(int dato){
28         byte i = 0;
29         while(i <= p && ordenados[i] < dato)
30             i++;
31
32         return (byte) ((i > p || ordenados[i] > dato)?-i:i);
33     }
34
35     public void eliminaDato(byte pos){
36         for (int k = pos; k < p; k++) {
37             ordenados[k] = ordenados[k+1];
38         }
39         p--;
40     }
41
42     public void recorrerPosiciones(byte pos){
43         // este metodo baja todos los elementos para
44         //que haya una casilla vacia
45         for (int j = p+1; j > pos; j--) {
46             ordenados[j] = ordenados[j-1];
47         }
48     }
49
50     public void agregarOrdenado(){
51         int dato = Tools.leeEntero(msg: "Ingrese Dato:");
52         if(validarVacio()){
53             ordenados[p+1] = dato;
```

```

54         p++;
55     } else {
56         byte pos = busquedaSecuencialOrdenada(dato);
57         //System.out.println(pos);
58
59         if(pos > 0){
60             JOptionPane.showMessageDialog(parentComponent: null,
61                 message: "El dato ya existe.");
62         } else{
63             pos*=-1;
64             recorrerPosiciones(pos);
65             ordenados[pos] = dato;
66             p++;
67         }
68     }
69 }
70
71 public void modificaDato(byte pos){
72     int dato = 0;
73     if(pos == 0){ //PRIMERO
74         do{
75             dato = Tools.leeEntero("Nuevo dato: " + "\n"
76                 + "El numero tiene que ser menor que: " + ordenados[pos]);
77             } while(dato>ordenados[pos+1]);
78             ordenados[pos] = dato;
79         } else{
80             if(pos == p) // SEGUNDO
81             do{
82                 dato = Tools.leeEntero("Nuevo dato: " + "\n"
83                     + "El numero tiene que ser menor que: " + ordenados[pos]);
84                 } while(dato>ordenados[pos]);
85                 ordenados[pos] = dato;
86             }
87             if(pos > 0 && pos < p)
88             do{
89                 dato = Tools.leeEntero("Nuevo dato: " + "\n"
90                     + "El numero tiene que ser" + "\n"
91                     + "Mayor que: " + ordenados[pos-1] + "\n"
92                     + "Menor que: " + ordenados[pos+1]);
93                 }while(dato>ordenados[pos+1] && dato>ordenados[pos-1]);
94                 ordenados[pos] = dato;
95             }
96         }
97     }
98 }
99
100
101
102

```

Funcionamiento

Para esta clase, vamos a ocupar los métodos anteriores solo con pequeñas modificaciones, pues como ahora estamos manejando datos ordenados, aquí sí importa tener orden en los datos que se vayan almacenando en el array.

Una vez teniendo esas dos clases con sus respectivos métodos, se ejecutaron para verificar su correcto funcionamiento. Se hizo un menú para Datos Desordenados y para Datos Ordenados, los cuales son sus pruebas respectivamente.

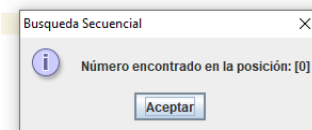
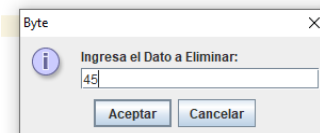
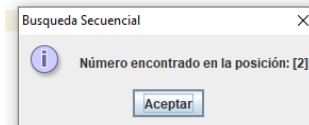
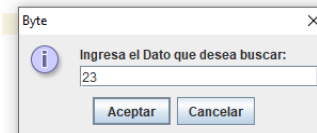
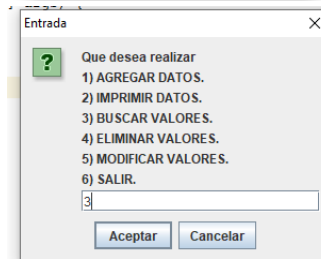
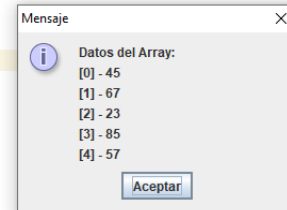
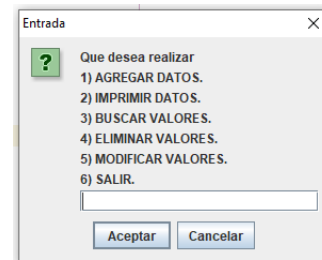
Datos Desordenados Menú:

Clase

Clase Menú:

```
1 package TestMetodos;
2 import EntradaSalida.Tools;
3 import Metodos.ArrayOperacion;
4 import javax.swing.JOptionPane;
5
6 public class Menu1 {
7     public static int menuOpciones(){
8         //variables locales: opciones y opcionElegida
9         String opciones="Que desea realizar\n";
10        opciones+="1) AGREGAR DATOS.\n";
11        opciones+="2) IMPRIMIR DATOS.\n";
12        opciones+="3) BUSCAR VALORES.\n";
13        opciones+="4) ELIMINAR VALORES.\n";
14        opciones+="5) MODIFICAR VALORES.\n";
15        opciones+="6) SALIR.\n";
16        int opcionElegida=Tools.leeEntero(msg: opciones);
17        return opcionElegida;
18    }
19
20    public static void menu(){
21        ArrayOperacion objM = new ArrayOperacion(5);
22        //DatosOrdenados objM = DatosOrdenados(5);
23        int opcion;
24        while((opcion=menuOpciones())!=6){
25            switch(opcion){
26                case 1: objM.agregarDatos();
27                break;
28                case 2: objM.imprimirArray();
29                break;
30                case 3:
31                    byte datoBuscado = Tools.leeByte
32                    (msg: "Ingresa el Dato que desea buscar: ");
33                    byte resultadoBusqueda =
34                        objM.buscarSecuencial(dato: datoBuscado);
35                    if(resultadoBusqueda > -1){
36                        JOptionPane.showMessageDialog
37                        (parentComponent:null, "Número encontrado en la"
38                            + " posición: ["+resultadoBusqueda+"]",
39                            title: "Busqueda Secuencial",
40                            messageType: JOptionPane.INFORMATION_MESSAGE);
41                    } else{
42                        JOptionPane.showMessageDialog
43                        (parentComponent:null, "No se encontro "
44                            + "el numero", title: "Busqueda secuencial",
45                            messageType: JOptionPane.WARNING_MESSAGE);
46                    }
47                    break;
48
49                case 4:
50                    byte datoBuscad = Tools.leeByte
51                    (msg: "Ingresa el Dato a Eliminar: ");
52                    byte resultadoBusqued = objM.buscarSecuencial(dato: datoBuscad);
53                    if(resultadoBusqued > -1){
54                        JOptionPane.showMessageDialog(parentComponent:null,
55                            "Número encontrado en la posición: [" +
56                                resultadoBusqued + "]",
57                            title: "Busqueda Secuencial",
58                            messageType: JOptionPane.INFORMATION_MESSAGE);
59                        objM.eliminarDato(pos: resultadoBusqued);
60                    } else{
61                        JOptionPane.showMessageDialog
62                        (parentComponent:null, "No se encontro el "
63                            + "numero", title: "Busqueda secuencial",
64                            messageType: JOptionPane.WARNING_MESSAGE);
65                    }
66                    break;
67            }
68        }
69    }
70 }
```

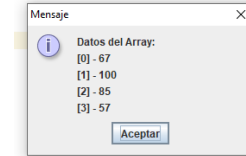
Ejecución



```

68 |
69 |         case 5:
70 |             byte datoM = Tools.leeByte(msg: "Ingrese el Dato a Modificar: ");
71 |             byte resulB = objM.buscarSecuencial(dato: datoM);
72 |             if(resulB > -1){
73 |                 JOptionPane.showMessageDialog
74 |                 (parentComponent:null, "Número encontrado en la "
75 |                 + "posición: [" + resulB + "]", "Busqueda "
76 |                 + "Secuencial",
77 |                 messageType: JOptionPane.INFORMATION_MESSAGE);
78 |                 objM.modificarDato(pos: resulB);
79 |             } else {
80 |                 JOptionPane.showMessageDialog
81 |                 (parentComponent:null, "No se encontro el"
82 |                 + " numero", title: "Busqueda secuencial",
83 |                 messageType: JOptionPane.WARNING_MESSAGE);
84 |             }
85 |             break;
86 |
87 |         case 6:
88 |             break;
89 |         default:
90 |             System.out.println("Error, opcion no definida.");
91 |     }
92 | }
93 |

```



Clase Main:

```

ArrayOperacion.java x DatosOrdenados.java x Principall.java x Menu1.java x
Source History
1
2 package TestMetodos;
3
4 public class Principall {
5
6     public static void main(String[] args) {
7         Menu1.menu();
8     }
9

```

Funcionamiento

Como se ve en este caso, el orden de almacenar los datos no importa, pues así como se vayan insertando, así ocupan su lugar en el array y no hay reglas al momento de insertarlos. Los métodos de eliminar y modificar son fundamentales para este tema.

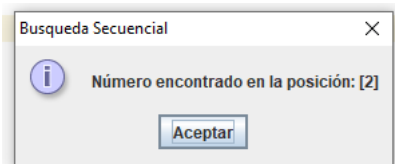
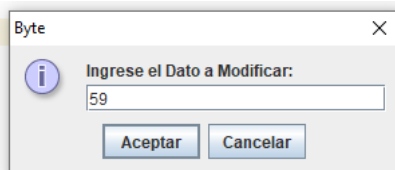
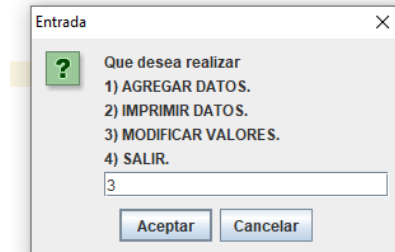
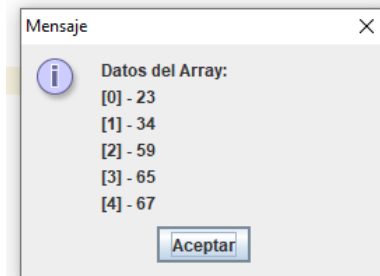
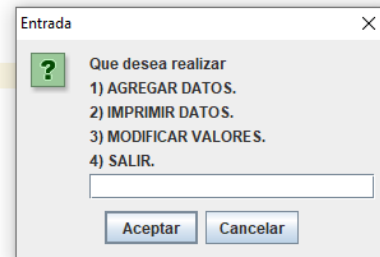
Datos Ordenados Menú:

Clase

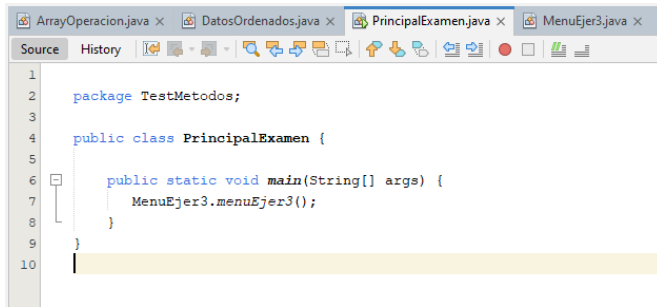
Clase Menú:

```
1 package TestMetodos;
2 import EntradaSalida.Tools;
3 import Metodos.MetodosEjer3;
4 import static TestMetodos.MenuEjer3.menuOpciones3;
5 import javax.swing.JOptionPane;
6
7 public class MenuEjer3 {
8     public static int menuOpciones3() {
9         String opciones="Que desea realizar\n";
10         opciones+="1) AGREGAR DATOS.\n";
11         opciones+="2) IMPRIMIR DATOS.\n";
12         opciones+="3) MODIFICAR VALORES.\n";
13         opciones+="4) SALIR.\n";
14         int opcionEligida=Tools.leeEntero(msg: opciones);
15         return opcionEligida;
16     }
17
18     public static void menuEjer3() {
19         MetodosEjer3 objM = new MetodosEjer3(ram: 5);
20
21         int opcion;
22         while ((opcion=menuOpciones3())!=4) {
23             switch(opcion) {
24
25                 case 1: objM.agregarOrdenado();
26                     break;
27                 case 2: objM.imprimeDatos();
28                     break;
29                 case 3:
30                     byte datoM = Tools.leeByte(msg: "Ingrese el Dato a Modificar: ");
31                     byte resulB = objM.búsquedaSecuencialOrdenada(dato: datoM);
32                     if(resulB > -1) {
33                         JOptionPane.showMessageDialog(parentComponent:null,
34                             "Número encontrado en la posición: [" + resulB + "]",
35                             title: "Busqueda Secuencial",
36                             messageType: JOptionPane.INFORMATION_MESSAGE);
37                         objM.modificaDato(pos: resulB);
38                     } else {
39                         JOptionPane.showMessageDialog(parentComponent:null,
40                             message:"No se encontro el numero",
41                             title: "Busqueda secuencial",
42                             messageType: JOptionPane.WARNING_MESSAGE);
43                     }
44                     break;
45                 case 4:
46                     break;
47                 default:
48                     System.out.println("Error, opcion no definida.");
49             }
50         }
51     }
52 }
53
```

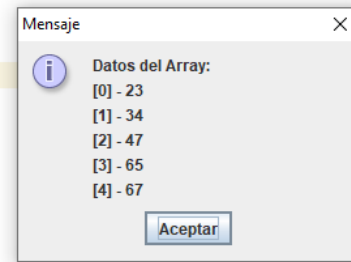
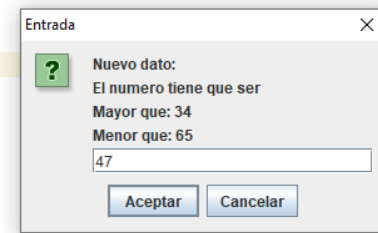
Ejecución



Clase Main:



```
1 package TestMetodos;
2
3
4 public class PrincipalExamen {
5
6     public static void main(String[] args) {
7         MenuEjer3.menuEjer3();
8     }
9 }
10
```



Funcionamiento

Cuando se trabaja con datos ordenados, no importa cómo se vayan insertando los datos, se irán acomodando en el array. A la hora de modificarlos se tiene que respetar entre que números debe de estar para que el orden no se altere.

4. Conclusiones

La clasificación de las estructuras de datos es un tema importante en la programación y se basa en la forma en que se organizan los datos. Estas estructuras tienen un conjunto de operaciones específicas que pueden realizarse sobre ellas, como la inserción, eliminación, búsqueda y ordenamiento.

El estudio de estructuras de datos, junto a su algoritmo, es parte esencial de los fundamentos de la programación. Cuando se quiere resolver un problema es importante seleccionar una estructura de datos adecuada.

Las estructuras de datos existen para ayudarnos en la compleja tarea de resolver problemas en programación, son útiles porque nos permiten tener una batería de herramientas para solucionar ciertos tipos de problemas, además, nos permiten hacer un software más eficiente optimizando recursos.

5. Bibliografía

Bibliografía

Facultad de Ciencias Exactas, Ingeniería y Agrimensura - UNR. (s.f.). *Estructura de Datos*. Obtenido de https://usuarios.fceia.unr.edu.ar/~sorribas/info1_notas_de_clase_4.pdf

Gallardo, J., & García, C. (s.f.). *Departamento de Lenguajes y Ciencias de la Computación*. Obtenido de Universidad de Málaga: <http://www.lcc.uma.es/~pepeg/modula/temas/tema8.pdf>

Sánchez, J. (2013). *Libro digital de Estructuras de Datos*. Obtenido de https://www.escom.ipn.mx/docs/oferta/matDidacticoISC2009/EDts/Libro_EstructuraDatos.pdf