INSTITUTO TECNOLÓGICO NACIONAL DE MEXICO CAMPUS ORIZABA

MATERIA: ESTRUCTURA DE DATOS

REPORTE DE PRACTICAS UNIDAD 1: INTRODUCCION A LAS ESTRUCTURA DE DATOS

MAESTRA: MARTINEZ CASTILLO MARIA JACINTA

INTEGRANTES:
BRAVO ROSAS YAMILETH
CRESCENCIO RICO JOSE ARMANDO

GRUPO: 4PM – 5PM HRS **CLAVE:** 3a3A

ESPECIALIDAD: ING. INFORMATICA

FECHA DE ENTREGA: 17/04/2023

Introducción.

En el siguiente reporte se presentarán las prácticas realizadas en relación a los arreglos unidimensionales y bidimensionales como introducción a las estructuras de datos.

Un array o arreglo es una estructura de datos en la programación que permite almacenar una colección de elementos homogéneos, donde cada elemento puede ser accedido mediante un índice o posición dentro del array. Estas estructuras se utilizan para manejar grandes cantidades de datos de una manera más eficiente y ordenada.

Los arreglos unidimensionales son una estructura de datos muy útil que nos permite almacenar múltiples valores del mismo tipo en una sola variable. En este informe se describirán los arreglos unidimensionales desordenados y ordenados, y las operaciones que se pueden realizar con ellos, como la inserción, búsqueda, eliminación y modificación.

Los arreglos bidimensionales, una estructura de datos en la que se pueden almacenar valores en una matriz de dos dimensiones. Se explicará la declaración de arreglos bidimensionales y las operaciones básicas que se pueden realizar con ellos, como la lectura, escritura y asignación.

Este informe tiene como objetivo proporcionar una comprensión más profunda de las estructuras de datos y las operaciones que se pueden realizar con ellas.

Competencia especifica.

Especifica(s):
Conoce y comprende las diferentes estructuras de datos, su clasificación y forma de manipularlas para buscar la manera más eficiente de resolver problemas.
Genéricas:

- Capacidad de análisis y síntesis.
- Habilidad en el manejo de equipo de cómputo.
- Capacidad de trabajar en equipo.

- Habilidad para buscar y analizar información proveniente de fuentes diversas.

Marco teórico.

1.1 Clasificación de las estructuras de datos.

Arreglos unidimensionales:

Durante el curso se aprendió sobre como en Java, un arreglo unidimensional (también conocido como vector o array) es una estructura de datos que permite almacenar un conjunto de elementos del mismo tipo. Los arreglos unidimensionales se declaran de la siguiente manera:

```
tipo[] nombreArreglo = new tipo[tamaño];
```

Donde "tipo" es el tipo de datos que almacenará el arreglo (como int, float, String, etc.), "nombreArreglo" es el nombre que le damos al arreglo y "tamaño" es la cantidad de elementos que queremos almacenar en el arreglo. Por ejemplo, para crear un arreglo de enteros de tamaño 5:

```
int[] numeros = new int[5];
```

Una vez declarado el arreglo podemos interactuar con el de 2 formas:

a) Arreglos desordenados:

   En Java, un arreglo desordenado es un arreglo que no tiene un orden específico en sus elementos, es decir, los elementos pueden estar en cualquier posición del arreglo. Además, con los arreglos desordenados se tienen diferentes métodos con los cuales podemos interactuar ya sea para agregar, buscar, eliminar o modificar un dato del arreglo. Por ejemplo:

   a. Inserción desordenada:

      El siguiente método tiene la función que mientras el array tenga espacio se almacenara un dato nuevo, en caso de estar lleno muestra “array lleno”.

```
public void almacenarDato() {
        if (p < datos.length) {
                datos[p + 1] = toolsList.leerInt("Escribe un numero");
                p++;
        } else {
                toolsList.imprimeErrorMsg("Array lleno");
        }
}
```

b. Imprime desordenada:

   Mediante un ciclo for nos encargamos de que el arreglo sea recorrido por cada uno de los elementos mediante su índice, y por medio de una cadena concatenar cada uno de esos elementos que contiene para después mostrar en pantalla cada uno de esos valores almacenado en el arreglo.

```
public String imprimeDatos() {
        String cad = "";

        for (int i = 0; i <= p; i++) {
                cad += i + "[" + datos[i] + "]" + "\n";
        }
        return ("\n " + cad);
}
```

c. Eliminación desordenada:

   Este método elimina un elemento de un arreglo en una posición específica y ajusta el tamaño del arreglo.

```
public void eliminaDato(int pos) {
        for (int j = pos; j <= p; j++) {
                datos[j] = datos[j + 1];
        }
        p--;
}
```

b) Arreglos ordenados:

   En cambio, un arreglo ordenado es un arreglo que tiene un orden específico en sus elementos, ya sea de forma ascendente o descendente.

a. Inserción ordenada:

El método "agregarOrdenado" inserta un nuevo valor en un arreglo ordenado en la posición correcta y ajusta el tamaño del arreglo. Es importante tener en cuenta que si el valor ya existe en el arreglo, no se insertará y se mostrará un mensaje de error. Además, es necesario validar que el arreglo no esté lleno antes de ejecutar este método.

```
public void agregarOrdenado() {
    int dato = 0;
    int pos = 0;

    if(validaVacio()) {
        datosOrdenados[posicion + 1] = toolsList.leerInt("Ingrese un valor");
        posicion++;
    } else {
        dato = toolsList.leerInt("Escribe el valor a insertar");
        pos = busquedaOrdenada(dato);
        if(pos > 0) {
            toolsList.imprimeErrorMsg("Dato existente");
        } else {
            pos = pos * (-1);
            corrimiento(pos);

            datosOrdenados[pos] = dato;
            posicion++;
        }
    }
}
```

b. Búsqueda ordenada:

El método toma un número entero como entrada y devuelve la posición del número en la lista ordenada, o una posición negativa si el número no se encuentra en la lista. El código utiliza un bucle while para buscar el número en la lista ordenada y utiliza la variable "i" para realizar un seguimiento de la posición actual.

```
public int busquedaOrdenada(int dato) {
    byte i = 0;

    while (i <= posicion && (Integer)datosOrdenados[i] < dato) {
        i++;
    }
    return (i > posicion || (Integer)datosOrdenados[i] > dato)? - i : i;
}
```

c. Eliminación ordenada:

   Elimina un elemento de una lista ordenada de números enteros llamada "datosOrdenados". El método toma una posición de índice "pos" como entrada y no devuelve ningún valor. El código elimina el elemento en la posición "pos" de la lista ordenada "datosOrdenados" y ajusta el índice de posición en consecuencia.

d. Modificación ordenada:

   El código modifica el elemento en la posición "pos" de la lista ordenada "datosOrdenados" según el valor de entrada "dato" y verifica que el nuevo valor sea coherente con los valores vecinos si es necesario.

```
public void modificaOrdenados(int dato) {
    int pos = 0;
    pos = busquedaOrdenada(dato);

    if(pos < posicion) {
        if(dato <= (Integer)datosOrdenados[0]) {
            do {
                pos = pos * (-1);
                dato = toolsList.leerInt("Ingresa un valor menor a: " + datosOrdenados[pos + 1]);
            } while (dato >= (Integer) datosOrdenados[pos * (-1) + 1]);
            datosOrdenados[pos * (-1)] = dato;
        } else {
            do {
                dato = toolsList.leerInt("Ingresa un valor menor a: '" + datosOrdenados[pos + 1] +
                        "'  mayor a: '" + datosOrdenados[pos - 1] + "'");
            } while (dato >= (Integer) datosOrdenados[pos + 1] && dato <= (Integer) datosOrdenados[pos - 1]);
            datosOrdenados[pos] = dato;
        }
    } else {
        do {
            dato = toolsList.leerInt("Ingresa un valor mayor a: " + datosOrdenados[pos - 1]);
        } while (dato <= (Integer) datosOrdenados[pos - 1]);
        datosOrdenados[pos] = dato;
    }
}
```

1.2 Tipos de datos abstractos (TDA).

En Java, un Tipo de Dato Abstracto (TDA) se refiere a una estructura de datos que se define por su comportamiento y operaciones, en lugar de por su representación concreta en memoria. Es decir, los TDAs son abstracciones que permiten a los programadores trabajar con datos de manera más abstracta, sin necesidad de conocer los detalles de su implementación.

En Java, se pueden implementar varios tipos de datos abstractos (TDA). A continuación, se presentan algunos de los TDAs más comunes:

- Pilas (Stack): Una pila es un TDA que sigue el principio LIFO (Last In First Out). En Java, la interfaz Stack se utiliza para implementar una pila. La clase Stack en Java es una implementación concreta de esta interfaz.

```java
import java.util.Stack;

Stack<Integer> pila = new Stack<Integer>();

// Agregar elementos a la pila
pila.push(10);
pila.push(20);
pila.push(30);

// Eliminar elemento de la pila
int elementoEliminado = pila.pop();

// Obtener el elemento superior de la pila
int elementoSuperior = pila.peek();

// Obtener el tamaño de la pila
int tamaño = pila.size();
```

- Colas (Queue): Una cola es un TDA que sigue el principio FIFO (First In First Out). En Java, la interfaz Queue se utiliza para implementar una cola. Las clases LinkedList y PriorityQueue son implementaciones concretas de la interfaz Queue.

```java
Queue<String> cola = new LinkedList<String>();

// Agregar elementos a la cola
cola.add("elemento1");
cola.add("elemento2");
cola.add("elemento3");

// Eliminar elemento de la cola
String elementoEliminado = cola.remove();

// Obtener el elemento frontal de la cola
String elementoFrontal = cola.peek();

// Obtener el tamaño de la cola
int tamaño = cola.size();
```

- Listas (List): Una lista es un TDA que se utiliza para almacenar y manipular elementos en una secuencia ordenada. En Java, la interfaz List se utiliza para implementar una lista. Las clases ArrayList y LinkedList son implementaciones concretas de la interfaz List.

```java
List<String> lista = new ArrayList<String>();

// Agregar elementos a la lista
lista.add("elemento1");
lista.add("elemento2");
lista.add("elemento3");

// Eliminar elemento de la lista
lista.remove(1);

// Obtener el elemento de la lista
String elemento = lista.get(0);

// Obtener el tamaño de la lista
int tamaño = lista.size();
```

- Mapas (Map): Un mapa es un TDA que se utiliza para almacenar y recuperar elementos en función de una clave única. En Java, la interfaz Map se utiliza

para implementar un mapa. Las clases HashMap y TreeMap son implementaciones concretas de la interfaz Map.

```java
Map<String, Integer> mapa = new HashMap<String, Integer>();

// Agregar elementos al mapa
mapa.put("llave1", 10);
mapa.put("llave2", 20);
mapa.put("llave3", 30);

// Eliminar elemento del mapa
mapa.remove("llave2");

// Obtener el valor del mapa
int valor = mapa.get("llave1");

// Verificar si el mapa contiene una llave
boolean contieneLlave = mapa.containsKey("llave1");

// Obtener el tamaño del mapa
int tamaño = mapa.size();
```

1.4 Manejo de memoria.

Java es un lenguaje de programación que maneja automáticamente la memoria, utilizando lo que se conoce como el "recolector de basura" (garbage collector) para gestionar la memoria dinámica. El recolector de basura es un proceso que se ejecuta en segundo plano y se encarga de detectar y eliminar los objetos que ya no se están utilizando, liberando la memoria correspondiente.

En Java, el stack y el heap son dos tipos diferentes de memoria utilizados por un programa durante su ejecución.

El stack en Java es utilizado para el almacenamiento de datos temporales y la ejecución de las operaciones de la pila. En este caso, el stack es utilizado para la creación y destrucción de objetos. Cada vez que un objeto es creado, se almacena una referencia a él en la pila. Cuando se llama a un método, se agregan sus argumentos y variables locales a la pila. Al finalizar la ejecución del método, estas variables se eliminan de la pila.

Por otro lado, el heap es utilizado para la asignación de memoria dinámica. Este tipo de memoria es utilizado cuando un objeto es creado a través de la palabra clave

new. El heap es responsable del almacenamiento y la gestión de los objetos durante la ejecución del programa.

1.4.1 Memoria estática-Operaciones básica.

Las operaciones básicas de la memoria estática en Java incluyen la declaración de variables estáticas, la asignación de valores a estas variables, el acceso a las variables desde otros lugares del código, y el uso de variables estáticas en métodos estáticos y no estáticos. Aquí se detallan estas operaciones:

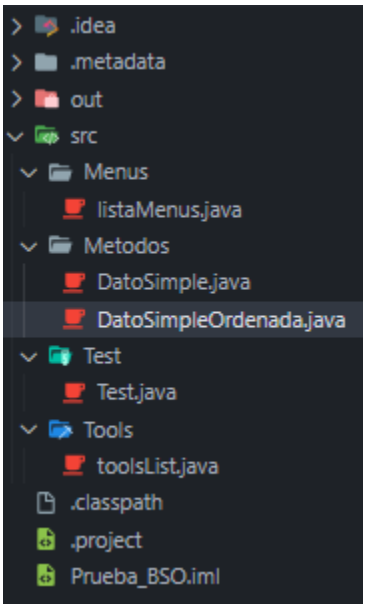
- Declaración de variables estáticas: se declara una variable estática usando la palabra clave "static" antes del tipo de datos.
- Asignación de valores a variables estáticas: se asigna un valor a una variable estática usando el nombre de la clase y el nombre de la variable estática.
- Acceso a variables estáticas: se accede a una variable estática usando el nombre de la clase y el nombre de la variable estática.
- Uso de variables estáticas en métodos estáticos: se puede acceder a variables estáticas en métodos estáticos sin la necesidad de crear una instancia de la clase.
- Uso de variables estáticas en métodos no estáticos: también se puede acceder a variables estáticas en métodos no estáticos, pero se necesita crear una instancia de la clase.

Recursos materiales y equipo.

- Computadora.
- Software IDE Eclipse For Java Developers,
- Lectura de los materiales de apoyo tema 1.
- Notas de clases.

Desarrollo de la práctica.

Para el presente reporte se creo la siguiente estructura de paquetes y clases para el correcto orden y ejecución del programa



Dentro del paquete “Metodos” se crearon 2 clases “Dato simple” y “Dato simple ordenada”:

Dato simple: Dentro de esta clase se define una lista de datos en un arreglo de tamaño fijo, además de contar con los siguientes métodos:

- "validaVacio" devuelve true si el arreglo está vacío (es decir, si "p" es igual a -1) y false en caso contrario.

```
public boolean validaVacio() {
  return (p == -1);
}
```

- "almacenarDato" agrega un nuevo dato al arreglo si el arreglo no está lleno. El método utiliza un método de la clase "toolsList" llamado "leerInt" para solicitar al usuario que ingrese un número, lo almacena en la siguiente posición disponible en el arreglo y actualiza "p" para indicar la nueva posición del último dato almacenado. Si el arreglo está lleno, el método muestra un mensaje de error utilizando otro método de la clase "toolsList" llamado "imprimeErrorMsg".

```java
public void almacenarDato() {
  if (p < datos.length) {
    datos[p + 1] = toolsList.leerInt("Escribe un numero");
    p++;
  } else {
    toolsList.imprimeErrorMsg("Array lleno");
  }
}
```

- El método "arreglo" devuelve una referencia al arreglo "datos".

```java
public Object[] arreglo() {
  return datos;
}
```

- "imprimeDatos" devuelve una cadena que contiene la lista de datos almacenados en el arreglo, con su posición correspondiente. El método utiliza un bucle for para recorrer los datos almacenados en el arreglo y construir la cadena de salida.

```java
public String imprimeDatos() {
  String cad = "";

  for (int i = 0; i ≤ p; i++) {
    cad += i + "[" + datos[i] + "]" + "\n";
  }
  return ("\n " + cad);
}
```

- "busquedaSecuencial" recibe un objeto "val" y busca ese objeto en el arreglo "datos" utilizando una búsqueda secuencial. Si el objeto se encuentra en el arreglo, el método devuelve la posición en la que se encuentra. Si el objeto no se encuentra en el arreglo, el método devuelve -2.

```java
public byte busquedaSecuencial(Object val) {
  byte i = 0;

  while (i ≤ p && !val.equals(datos[i])) {
    i++;
  }
  return (i ≤ p) ? i : -2;
}
```

- "eliminaDato" elimina el dato en la posición "pos" del arreglo y ajusta "p" para indicar la nueva posición del último dato almacenado. El método utiliza un bucle for para desplazar todos los datos posteriores a la posición "pos" una posición hacia atrás en el arreglo.

```java
public void eliminaDato(int pos) {
  for (int j = pos; j ≤ p; j++) {
    datos[j] = datos[j + 1];
  }
  p--;
}
```

- "busquedaOrdenada" recibe un arreglo de enteros "datos" y un entero "dato" y realiza una búsqueda binaria en el arreglo para encontrar la posición en la que "dato" debería ser insertado para mantener el orden del arreglo. Si "dato" ya se encuentra en el arreglo, el método devuelve su posición. Si "dato" no se encuentra en el arreglo, el método devuelve el negativo del índice donde debería ser insertado para mantener el orden del arreglo.

```java
public int busquedaOrdenada(int[] datos, int dato) {
  byte i = 0;
  while (i < datos.length && (Integer) datos[i] < dato) {
    i++;
  }
  return (i > datos.length || (Integer) datos[i] > dato) ? -i : i;
}
```

Dato simple ordenado:

- Constructor "DatoSimpleOrdenada": Recibe un parámetro de tipo byte que indica el tamaño del arreglo. El constructor inicializa los campos "datosOrdenados" y "posicion".

```java
public DatoSimpleOrdenada(byte tam) {
  datosOrdenados = new Object[tam];
  posicion = -1;
}
```

- "validaVacio": Retorna un valor booleano que indica si el arreglo está vacío o no

```java
public boolean validaVacio() {
  return (posicion == -1);
}
```

- "corrimiento": Recibe un parámetro de tipo int que indica la posición a partir de la cual se debe realizar un corrimiento hacia la derecha. El método mueve los elementos del arreglo desde la posición indicada hasta la posición siguiente, de forma que se libera espacio para agregar un nuevo elemento.

```java
public void corrimiento(int pos) {
  for(int i = posicion + 1; i > pos; i--)
    datosOrdenados[i] = datosOrdenados[i - 1];
}
```

- "agregarOrdenado": Este método se utiliza para agregar un nuevo elemento al arreglo de forma ordenada. Si el arreglo está vacío, simplemente se agrega el elemento en la primera posición. Si el arreglo no está vacío, se busca la posición adecuada para agregar el nuevo elemento utilizando el método.

```java
public void agregarOrdenado() {
  int dato = 0;
  int pos = 0;
  if(validaVacio()) {
    datosOrdenados[posicion + 1] = toolsList.leerInt("Ingrese un valor");
    posicion++;
  } else {
    dato = toolsList.leerInt("Escribe el valor a insertar");
    pos = busquedaOrdenada(dato);
    if(pos > 0) {
      toolsList.imprimeErrorMsg("Dato existente");
    } else {
      pos = pos * (-1);
      corrimiento(pos);
      datosOrdenados[pos] = dato;
      posicion++;
    }
  }
}
```

- "imprimeOrdenados": Retorna una cadena que representa los elementos del arreglo en orden.

```java
public String imprimeOrdenados() {
  String cad = "";
  for (int i = 0; i ≤ posicion; i++) {
    cad += i + "[" + datosOrdenados[i] + "]" + "\n";
  }
  return "\n" + cad;
}
```

- "modificaOrdenados": Recibe un parámetro de tipo int que indica el elemento que se desea modificar. El método busca la posición del elemento utilizando el método "busquedaOrdenada". Si el elemento está en la primera posición, se solicita un nuevo elemento menor que el siguiente elemento del arreglo. Si el elemento está en una posición intermedia, se solicita un nuevo elemento que sea mayor que el elemento anterior y menor que el elemento siguiente. Si el elemento está en la última posición, se solicita un nuevo elemento mayor

que el elemento anterior. Finalmente, se reemplaza el elemento existente por el nuevo elemento.

```
public void modificaOrdenados(int dato) {
  int pos = 0;
  pos = busquedaOrdenada(dato);

  if(pos < posicion) {
    if(dato ≤ (Integer)datosOrdenados[0]) {
      do {
        pos = pos * (-1);
        dato = toolsList.leerInt("Ingresa un valor menor a: " + datosOrdenados[pos + 1]);
      } while (dato ≥ (Integer) datosOrdenados[pos * (-1) + 1]);
        datosOrdenados[pos * (-1)] = dato;
    } else {
      do {
        dato = toolsList.leerInt("Ingresa un valor menor a: '" + datosOrdenados[pos + 1] +
        "'  mayor a: '" + datosOrdenados[pos - 1] + "'");
        } while (dato ≥ (Integer) datosOrdenados[pos + 1] && dato ≤ (Integer) datosOrdenados[pos - 1]);
        datosOrdenados[pos] = dato;
    }
  } else {
    do {
      dato = toolsList.leerInt("Ingresa un valor mayor a: " + datosOrdenados[pos - 1]);
    } while (dato ≤ (Integer) datosOrdenados[pos - 1]);
    datosOrdenados[pos] = dato;
  }
}
```

- "busquedaOrdenada": Recibe un parámetro de tipo int que indica el elemento que se desea buscar en el arreglo. El método realiza una búsqueda secuencial ordenada y retorna la posición del elemento si se encuentra en el arreglo, o un valor negativo que indica la posición donde debería estar el elemento si se quisiera insertar en el arreglo.

```
public int busquedaOrdenada(int dato) {
  byte i = 0;
  while (i ≤ posicion && (Integer)datosOrdenados[i] < dato) {
    i++;
  }
  return (i > posicion || (Integer)datosOrdenados[i] > dato)? - i : i;
}
public void eliminaDato(int pos) {
  for (int j = pos; j ≤ posicion; j++) {
    datosOrdenados[j]= datosOrdenados[j+1];
  }
  posicion--;
}
```

- "eliminaDato": Recibe un parámetro de tipo int que indica la posición del elemento que se desea eliminar del arreglo. El método realiza un corrimiento hacia la izquierda a partir de la posición indicada para eliminar el elemento.

```java
public void eliminaDato(int pos) {
  for (int j = pos; j ≤ posicion; j++) {
    datosOrdenados[j]= datosOrdenados[j+1];
  }
  posicion--;
}
```

- "eliminarValor": Es un método estático que recibe dos parámetros: un arreglo bidimensional de enteros y un valor de tipo int que se desea eliminar del arreglo. El método busca el valor en el arreglo y elimina la fila completa en la que se encuentra el valor. Retorna un nuevo arreglo sin la fila eliminada.

```java
public static int[][] eliminarValor(int[][] arreglo, int valorAEliminar) {
  int filas = arreglo.length;
  int columnas = arreglo[0].length;
  for (int i = 0; i < filas; i++) {
    for (int j = 0; j < columnas; j++) {
      if (arreglo[i][j] == valorAEliminar) {
        int[][] nuevoArreglo = new int[filas - 1][columnas];
        int contador = 0;
        for (int k = 0; k < filas; k++) {
          if (k == i) {
            continue;
          }
          nuevoArreglo[contador] = Arrays.copyOf(arreglo[k], columnas);
          contador++;
        }
        return nuevoArreglo;
      }
    }
  }
  return arreglo;
}
```

Posteriormente después de hacer todas las implementaciones, se crea una función llamada "menu3" que toma un parámetro de cadena llamado "menu" la cual recibe como parámetro una cadena de las diferentes opciones que podrá hacer el usuario.

El propósito de la función "menu3" es proporcionar un menú interactivo para realizar varias operaciones en una lista de datos almacenados en una instancia de la clase "DatoSimpleOrdenada". La lista de datos está inicializada en la línea que declara y asigna un objeto "DatoSimpleOrdenada" a la variable "obj2".

```java
DatoSimpleOrdenada obj2 = new DatoSimpleOrdenada(tam);
```

Las opciones del menú y sus correspondientes acciones son las siguientes:

- "Agregar": llama al método "agregarOrdenado" del objeto "obj2" para agregar un nuevo dato a la lista.

```
case "Agregar":
  obj2.agregarOrdenado();
break;
```

- "Imprimir": llama al método "imprimeOrdenados" del objeto "obj2" para imprimir la lista de datos en orden, o muestra un mensaje de error si la lista está vacía.

```
case "Imprimir":
  if(obj2.validaVacio()) {
    toolsList.imprimeErrorMsg("Arreglo vacio");
  } else {
    toolsList.imprimePantalla("Datos del arreglo" + obj2.imprimeOrdenados());
  }
break;
```

- "Busqueda": solicita al usuario que ingrese un valor y llama al método "busquedaOrdenada" del objeto "obj2" para buscar el valor en la lista. Si se encuentra el valor, muestra su posición en la lista; de lo contrario, muestra la posición en la que se debería insertar para mantener la lista ordenada.

```
case "Busqueda":
  if(obj2.validaVacio()) {
      toolsList.imprimeErrorMsg("Arreglo vacio");
  } else {
    pos = obj2.busquedaOrdenada(toolsList.leerInt("Ingrese el valor"));
    if (pos ≥ 0) {
      toolsList.imprimePantalla("Se encuentra en la posicion: " + pos);
    } else {
      toolsList.imprimeErrorMsg("Se debe de insertar en la posicion" + pos * (-1));
    }
  }
break;
```

- "Modificar": solicita al usuario que ingrese un valor y llama al método "busquedaOrdenada" del objeto "obj2" para buscar el valor en la lista. Si se encuentra el valor, llama al método "modificaOrdenados" del objeto "obj2" para modificar el valor y luego muestra la lista actualizada; de lo contrario, muestra un mensaje de error.

```
case "Modificar":
  int dato = 0;

  if(obj2.validaVacio()) {
    toolsList.imprimeErrorMsg("El arreglo esta vacio");
  } else {
    dato = toolsList.leerInt("Ingrese el dato a modificar");
    pos = obj2.busquedaOrdenada(dato);
  }
  if(pos ≥ 0) {
    obj2.modificaOrdenados(dato);
    toolsList.imprimePantalla("Dato modificado." + obj2.imprimeOrdenados());
  } else {
    toolsList.imprimeErrorMsg("No se encuentra el dato");
  }
break;
```

- "Eliminar": solicita al usuario que ingrese un valor y llama al método "busquedaOrdenada" del objeto "obj2" para buscar el valor en la lista. Si se encuentra el valor, llama al método "eliminaDato" del objeto "obj2" para eliminar el valor y luego muestra su posición en la lista; de lo contrario, muestra un mensaje de error.

```
case "Eliminar":
  if (obj2.validaVacio()) {
    toolsList.imprimeErrorMsg("Arreglo Vacio");
  } else {
    pos = obj2.busquedaOrdenada(toolsList.leerInt("Ingrese el valor"));
    if (pos ≥ 0) {
      toolsList.imprimePantalla("Dato eliminado en la posicion: " + pos);
      obj2.eliminaDato(pos);
    } else {
      toolsList.imprimeErrorMsg("Dato no encontrado");
    }
  }
break;
```

- "Salir": Termina el bucle para finalizar la ejecución del programa.

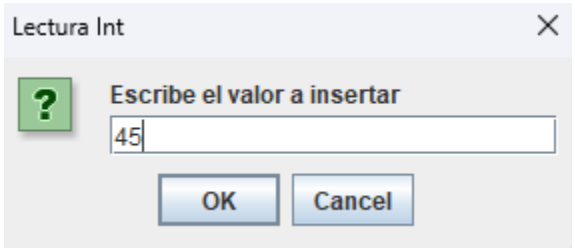
Resultados.

Como resultado después de aplicar todas las implementaciones en las clases “DatoSimple” y “DatoSimpleOrdenado” junto con la creación y utilización de sus metodos para la utilización de los mismos en el menu. El funcionamiento del menu funciona tanto para datos simple y dato simple ordenador.

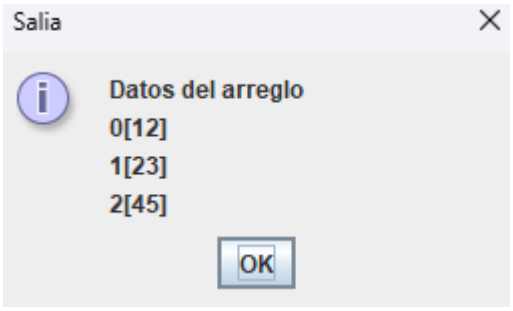
```
DatoSimple obj = new DatoSimple(tam);
DatoSimpleOrdenada obj2 = new DatoSimpleOrdenada(tam);
```

Una vez ejecutado el programa mostrara las siguientes opciones que puede realizar el usuario:

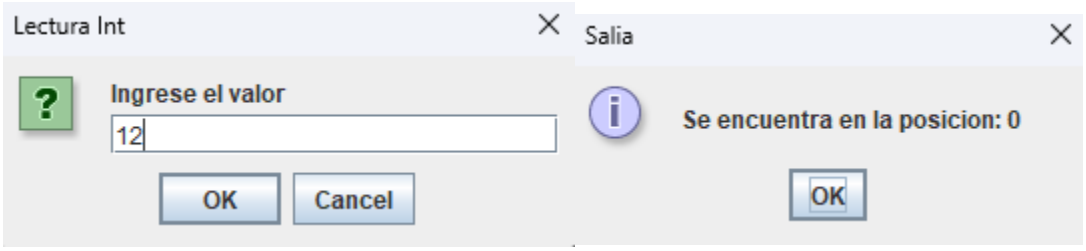
- Agregar:



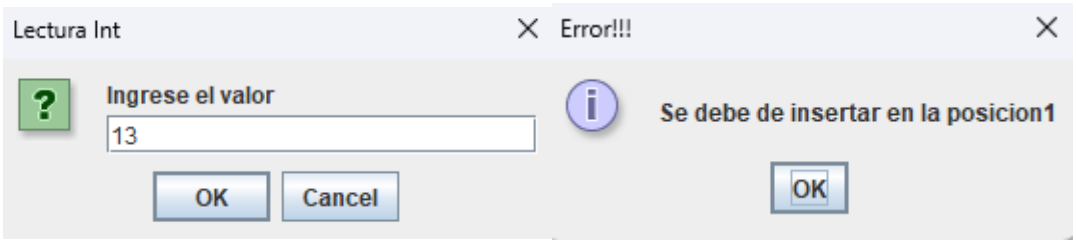
- Imprimir:



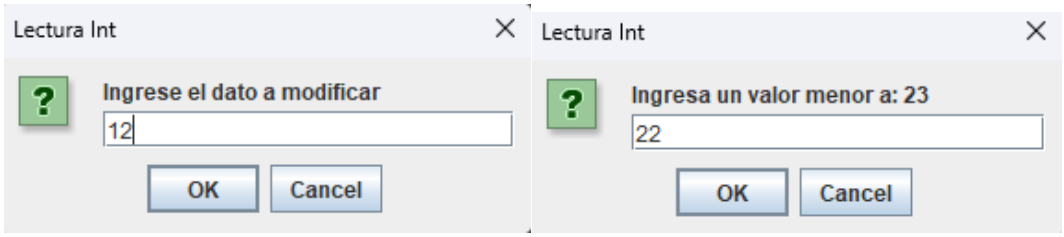
- Búsqueda:

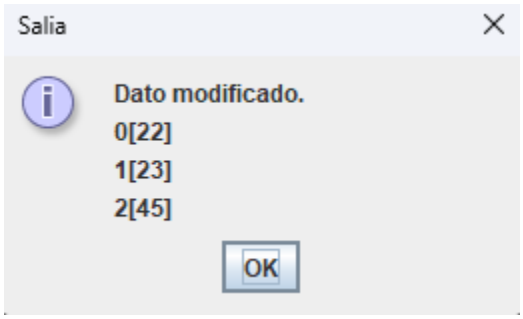


En caso de no encontrarse mostrará la posición donde deberá insertarse el nuevo dato.

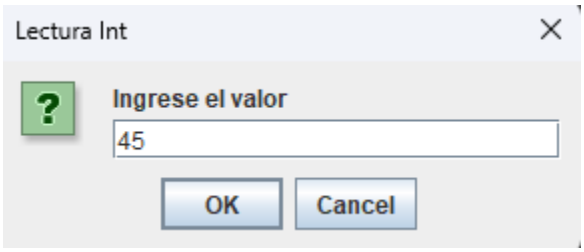


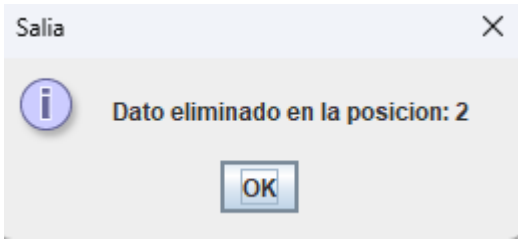
- Modificar:



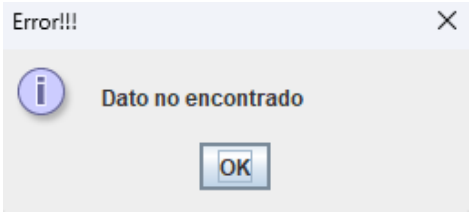


- Eliminar:





  - De no encontrarse un dato mostrara:



Conclusiones.

En conclusión, los arrays o arreglos son una estructura de datos fundamental en la programación que nos permiten almacenar y manipular grandes cantidades de datos del mismo tipo de manera eficiente y ordenada. Son ampliamente utilizados en muchos lenguajes de programación y nos permiten representar matrices, vectores, listas y otras estructuras de datos.

Además, los arrays ofrecen una variedad de operaciones que se pueden realizar con ellos, como la inserción, eliminación, ordenamiento, búsqueda y modificación de elementos, lo que hace que sean una herramienta esencial para cualquier programador.

En resumen, el conocimiento y el manejo de los arrays son imprescindibles en la programación y es importante conocer su uso y operaciones para poder manejar grandes cantidades de datos y realizar operaciones de manera más eficiente y efectiva.

Referencias.

Martínez Castillo, M. J. (2023). Estructuras de datos fundamentales PDF. Capítulo 1. (Obra original publicada en 2023)

Diferencias entre heap y stack. (s.f.). Blog de tecnologías de la información. https://codingornot.com/diferencias-entre-heap-y-stack

Asignaciones – Heap y Stack. (s.f.). Preparando SCJP. https://preparandoscjp.wordpress.com/2011/09/19/asignaciones-heap-y-stack/

Estructura de datos en Java ¡Infórmate! | tokioschool.com. (s.f.). Tokio School. https://www.tokioschool.com/noticias/estructura-datos-java/#:~:text=datos%20en%20Java.-,Tipos%20de%20datos%20en%20Java,por%20tipos%20primitivos%20o%20compuestos.

CC30A Algoritmos y Estructuras de Datos: Tipos de datos abstractos. (s.f.). DCC | Universidad de Chile. https://users.dcc.uchile.cl/~bebustos/apuntes/cc30a/TDA/