INSTITUTO TECNOLÓGICO NACIONAL DE MEXICO CAMPUS ORIZABA

Materia:

Estructura De Datos

Maestra

Martínez Castillo María Jacinta

Integrantes:

Bravo Rosas Yamileth -21010175
Crescencio Rico José Armando - 21010001

Grupo:

3PM – 4PM HRS

Clave:

3a3A

Especialidad:

Ing. informática

Fecha De Entrega

22/04/2023

REPORTE DE PRACTICAS

UNIDAD 3

Introducción.

En el mundo de la programación, las estructuras de datos son un elemento fundamental para el desarrollo de algoritmos y soluciones eficientes. En este sentido, las estructuras lineales son de gran importancia, y entre ellas destacan las pilas, colas y listas.

En este reporte de prácticas se aborda el uso de estas estructuras en el lenguaje de programación y además de sus principales operaciones básicas (Push, Pop, Peek, Empty) en Java ya sea para una pila o cola, explorando su implementación y aplicación en distintos problemas y situaciones.

Se presentan ejemplos como la elaboración de un menú en concreto que permiten comprender su funcionamiento y su utilidad en el desarrollo de soluciones informáticas. Además, se discuten algunas de las ventajas y desventajas de cada estructura, así como algunas buenas prácticas para su implementación y uso. En definitiva, este reporte busca brindar una visión general sobre el uso de las estructuras lineales de pilas y colas en Java, y su aplicación en la resolución de problemas informáticos.

Competencia especifica.

Especifica(s):
Comprende y aplica estructuras de datos lineales para solución de problemas.

Genéricas:

- Habilidad para buscar y analizar información proveniente de fuentes diversas.
- Capacidad de análisis y síntesis.
- Habilidad en el manejo de equipo de cómputo.
- Capacidad para trabajar en equipo.
- Capacidad de investigación.
- Capacidad de aplicar los conocimientos en la práctica.

Marco teórico.

3.0 Clases genéricas, arreglos dinámicos.

Las clases genéricas en Java son clases que pueden trabajar con diferentes tipos de datos de forma paramétrica. En otras palabras, las clases genéricas permiten definir una clase una vez y reutilizarla con diferentes tipos de datos sin tener que definir una nueva clase para cada tipo.

Las clases genéricas se definen utilizando el operador de tipo comodín "<>" y un identificador de tipo <T>.

Durante la unidad 3 se tuvo en cuenta que existen 3 tipos de estructuras de datos más comunes utilizadas para almacenar elementos y realizar operaciones específicas en ellos como lo son las pilas y las colas.

3.1 Pilas.

Una pila es una estructura de datos que sigue el principio LIFO (Last In, First Out), lo que significa que el último elemento agregado a la pila es el primero en ser eliminado. Las operaciones básicas de una pila son "push" (agregar un elemento) y "pop" (eliminar el último elemento agregado). También hay operaciones para acceder al elemento superior de la pila sin eliminarlo ("peek"), para verificar si la pila está vacía ("isEmpty"), y para obtener el tamaño actual de la pila ("size").

En términos de programación, una pila es una colección ordenada de elementos que se pueden agregar y retirar.

3.1.1 Representación en memoria.

En Java, las pilas se pueden implementar utilizando un arreglo (array) o una lista enlazada (linked list). Ambas implementaciones tienen diferentes formas de representación en memoria.

En la implementación de una pila mediante un arreglo, se reserva un bloque contiguo de memoria para almacenar los elementos de la pila. El primer elemento se coloca en la posición inicial del arreglo, y los demás elementos se agregan en las posiciones siguientes. La pila tiene un puntero que indica la posición del elemento superior en la pila. Cuando se agrega un elemento a la pila, se incrementa el puntero superior, y cuando se elimina un elemento, se decrementa el puntero superior. Si la pila se llena, se puede crear una nueva pila con un arreglo de mayor tamaño y copiar los elementos de la pila original en la nueva pila.

3.1.2 Operaciones básicas.

Las pilas en Java ofrecen una serie de operaciones básicas para administrar los elementos que contienen. A continuación, se describen algunas de las operaciones más comunes:

- **push**: Agrega un elemento al tope de la pila.
- **pop**: Elimina y devuelve el elemento del tope de la pila.
- **peek**: Devuelve el elemento del tope de la pila sin eliminarlo.
- **empty**: Verifica si la pila está vacía.
- **size**: Devuelve la cantidad de elementos que tiene la pila.

Estas operaciones se pueden implementar utilizando un arreglo o una lista enlazada.

3.1.3 Aplicaciones.

Las pilas en Java son una estructura de datos muy útil y versátil que se utiliza en una gran variedad de aplicaciones.

- Llamadas a subprogramas:
  Cuando dentro de un programa se realizan llamadas a subprogramas, el programa principal recordara el lugar donde se hizo la llamada de modo que pueda retornar allí. Esta operación se consigue disponiendo de las direcciones de la pila.

- Recursividad:

  Para simular un programa recursivo es necesario la utilización de pilas, ya que se está llamando continuamente a subprogramas que a la vez vuelven a llamarse a sí mismo.
- Tratamiento de expresiones aritméticas:

  Una expresión aritmética está formada por operando y operadores. Ejemplo: 3 + 34 * 5. Existen otras formas de escribir expresiones aritméticas, en el que se diferencian por la situación de los operadores respecto de los operandos. La forma en la que los operadores se colocan delate de los operandos es conocida como notación polaca o notación prefija.

  Ejemplo:

  Notación infija a * b / (a + c).

  Notación polaca */ab+ac

  En Java, se puede tratar una expresión aritmética utilizando una pila y el algoritmo de evaluación de expresiones postfix (notación postfija). El algoritmo postfix es una forma de escribir una expresión aritmética en la que los operadores se escriben después de sus operandos. Por ejemplo, la expresión 3 + 4 * 5 se escribiría como 3 4 5 * + en notación postfix.

- Ordenación rápida.

  En java Ordenación rápida se refiere a la operación de reorganizar los elementos mediante algoritmo

3.2 Colas.

Cola es otra estructura de datos que sigue el principio FIFO (First In, First Out), lo que significa que el primer elemento agregado a la cola es el primero en ser eliminado. Las operaciones básicas de una cola son "offer" (agregar un elemento),

"poll" (eliminar el primer elemento agregado), y "peek" (acceder al primer elemento sin eliminarlo).

3.2.1 Representación en memoria.

En Java, una cola puede ser implementada de varias maneras, pero la más común es a través de una estructura de datos conocida como "cola con punteros", que utiliza nodos enlazados para almacenar los elementos de la cola.

Cada nodo consta de dos partes: un campo "elemento" que almacena el valor del elemento y un campo "siguiente" que es un puntero que apunta al siguiente nodo en la cola. El primer nodo de la cola se llama "cabeza" y el último nodo se llama "cola". La cola comienza vacía con la cabeza y la cola como null.

Cuando se inserta un elemento en la cola, se crea un nuevo nodo y se asigna el valor del elemento al campo "elemento" del nodo. Si la cola está vacía, se establece el nodo como la cabeza y la cola. Si la cola no está vacía, se establece el puntero "siguiente" del nodo anterior para que apunte al nuevo nodo, y se establece el nuevo nodo como la nueva cola.

Cuando se elimina un elemento de la cola, se devuelve el valor almacenado en la cabeza de la cola y se actualiza la cabeza para que apunte al siguiente nodo en la cola. Si la cola está vacía después de la eliminación, la cabeza y la cola se establecen como null.

3.2.2 Operaciones básicas.

Estas operaciones son esenciales para manipular la cola de manera efectiva. La mayoría de las implementaciones de cola en Java proporcionan estas operaciones para interactuar con la cola.

1. enqueue: Agrega un elemento al final de la cola.
2. dequeue: Elimina el elemento del frente de la cola y lo devuelve.

3. peek: Devuelve el elemento del frente sin eliminarlo.
4. isEmpty: verifica si la cola está vacía.
5. size: Devuelve el número de elementos de la cola.

3.2.3 Tipos de colas: Simples, circulares y colas dobles.

La clasificación más común de las colas se basa en su estructura interna y se pueden dividir en tres tipos:

- Colas simples: también conocidas como colas FIFO (First-In-First-Out), son la forma más común de cola. En estas colas, los elementos se agregan al final de la cola y se eliminan del frente de la cola. Las colas simples se implementan mediante una lista enlazada, donde cada nodo contiene un elemento y un puntero al siguiente nodo.
- Colas circulares: en estas colas, los elementos se almacenan en una matriz circular en lugar de una lista enlazada. Una vez que se llega al final de la matriz, el puntero de la cola se desplaza al inicio de la matriz. Las colas circulares se utilizan a menudo en situaciones donde se necesita una cola de tamaño fijo, como en la programación de sistemas embebidos.
- Colas dobles: también conocidas como deques (Double Ended Queues), son colas en las que los elementos se pueden insertar y eliminar desde ambos extremos. Las colas dobles se implementan mediante una lista doblemente enlazada, donde cada nodo contiene un elemento y un puntero al siguiente y al nodo anterior.

3.2.4 Aplicaciones.

Las colas tienen numerosas aplicaciones en Java y en la programación en general. A continuación, se mencionan algunas de las aplicaciones más comunes de las colas en Java:

- Algoritmos de búsqueda y procesamiento de datos: las colas se pueden utilizar para implementar algoritmos de búsqueda en anchura

(BFS), que se utilizan para recorrer árboles y grafos. Las colas también se pueden utilizar para procesar grandes conjuntos de datos en el orden en que se recibieron.

- Programación concurrente: las colas se utilizan a menudo en la programación concurrente para implementar patrones de diseño como el productor-consumidor. Los hilos que producen datos los agregan a una cola, mientras que los hilos que consumen datos los eliminan de la cola. Esto asegura que los datos se procesen de manera segura y sincronizada.
- Administración de procesos y tareas: las colas se utilizan para administrar tareas en sistemas informáticos. Los procesos o tareas que deben completarse se agregan a una cola, y luego se procesan en el orden en que se agregaron. Esto asegura que las tareas se completen de manera justa y equitativa.
- Implementación de servicios de red: las colas se utilizan para implementar servicios de red, como colas de correo electrónico y colas de mensajes en sistemas de mensajería instantánea. Los mensajes se agregan a una cola y se procesan en el orden en que se recibieron.

3.3 Listas.

En Java, una lista es una estructura de datos que se utiliza para almacenar una colección de elementos en un orden específico. Las listas en Java pueden ser de dos tipos:

1. Listas enlazadas: en este tipo de lista, los elementos se almacenan en nodos, cada uno de los cuales contiene un elemento y un puntero al siguiente nodo en la lista. La lista comienza con un nodo denominado cabeza y termina con un nodo denominado cola. Las listas enlazadas son flexibles y eficientes para insertar o eliminar elementos en cualquier posición de la lista.
2. Listas basadas en arreglos: en este tipo de lista, los elementos se almacenan en un arreglo y se acceden a ellos mediante un índice. Las listas basadas en arreglos son útiles cuando se necesita un acceso rápido a elementos

específicos de la lista, pero no son tan eficientes como las listas enlazadas para insertar o eliminar elementos en posiciones aleatorias.

3.3.1 Operaciones básicas.

En Java, las listas se pueden implementar mediante la interfaz List, que es una subinterfaz de la interfaz Collection. Algunas de las operaciones básicas que se pueden realizar en una lista en Java incluyen:

- Añadir elementos: se pueden añadir elementos a una lista utilizando los métodos add() o addAll().
- Eliminar elementos: se pueden eliminar elementos de una lista utilizando el método remove().
- Acceder a elementos: se puede acceder a elementos de una lista utilizando un índice o un iterador.
- Buscar elementos: se pueden buscar elementos en una lista utilizando el método indexOf() o contains().
- Ordenar elementos: se pueden ordenar los elementos de una lista utilizando el método sort().

3.3.2 Tipos de listas: simplemente enlazadas, doblemente enlazadas y circulares.

En Java, existen tres tipos principales de listas enlazadas, que son:

- Listas simplemente enlazadas: también conocidas como listas enlazadas unidireccionales, en estas listas cada nodo tiene un solo enlace o puntero que apunta al siguiente nodo en la lista. El último nodo de la lista tiene un enlace nulo. Las listas simplemente enlazadas son simples de implementar y son útiles para operaciones que requieren un recorrido de la lista desde el inicio hasta el final, pero no son tan eficientes para operaciones que necesitan acceso a nodos anteriores.

- Listas doblemente enlazadas: en estas listas, cada nodo tiene dos punteros: uno que apunta al nodo anterior y otro que apunta al nodo siguiente en la lista. Las listas doblemente enlazadas son más flexibles que las listas simplemente enlazadas, ya que se pueden recorrer en ambas direcciones y son más eficientes para la eliminación de nodos en posiciones aleatorias.
- Listas circulares: en estas listas, el último nodo de la lista apunta al primer nodo, formando un bucle. Las listas circulares se pueden implementar como listas simplemente enlazadas o doblemente enlazadas y se utilizan a menudo en aplicaciones que requieren un recorrido continuo de la lista.

3.3.3 Aplicaciones.

Las listas son estructuras de datos que se utilizan para almacenar una colección de elementos en Java. A continuación, se presentan algunas aplicaciones comunes de las listas en Java:

- Almacenamiento de datos: Las listas son útiles para almacenar una colección de datos. Por ejemplo, una lista puede almacenar nombres de personas, números de teléfono, direcciones de correo electrónico, etc.
- Implementación de pilas y colas: Las listas se pueden utilizar para implementar pilas y colas. En una pila, el último elemento agregado es el primero en salir (LIFO), mientras que en una cola, el primer elemento agregado es el primero en salir (FIFO).
- Ordenación de datos: Las listas se pueden utilizar para ordenar elementos en orden ascendente o descendente. Por ejemplo, se pueden utilizar para ordenar nombres en orden alfabético o para ordenar números en orden numérico.
- Búsqueda de elementos: Las listas se pueden utilizar para buscar elementos específicos en una colección. Por ejemplo, se puede buscar un nombre específico en una lista de nombres.

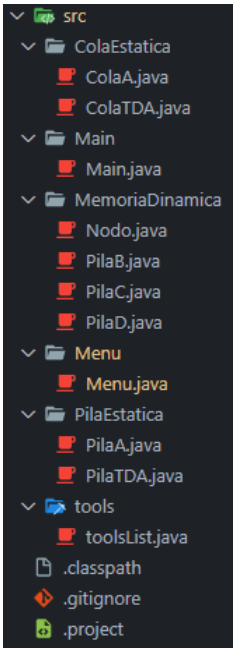
- Implementación de algoritmos: Las listas se pueden utilizar para implementar varios algoritmos, como el algoritmo de ordenación de burbuja, el algoritmo de ordenación rápida, etc.

Recursos materiales y equipo.

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

Desarrollo de la práctica.

Para la realización de la práctica se creó un proyecto con el nombre “Estructuras Lineales” con la siguiente estructura de carpetas:



Pila Estática:

PilaA: Dentro de este paquete se define una clase llamada “PilaA” que implementa la interfaz “PilaTDA<T>”, que representa una pila (o stack) de elementos utilizando un array estático.

```java
public class PilaA<T> implements PilaTDA<T> {
  private T pila[];
  private byte tope;

  @SuppressWarnings("unchecked")
  public PilaA(int max) {
    pila = (T[])(new Object[max]);
    tope = -1;
  }
  public boolean isEmpty() {
    return (tope == -1);
  }
  public boolean isSpace() {
    return (tope < pila.length - 1);
  }
  public void push(T dato) {
    if(isSpace()) {
      tope ++;
      pila[tope] = dato;
    } else {
      toolsList.imprimeErrorMsg("PILA LLENA");
    }
  }
  public T pop() {
    T dato = pila[tope];
    tope--;
    return dato;
  }
  public T peek() {
    return pila[tope];
  }
  public String toString() {
    return toString(tope);
  }
  private String toString(int i) {
    return (i ≥ 0 )
      ? "\n" + "tope[" + i + "]⟹" + "" + pila[i] + "" + toString(i - 1)
      : "";
  }
}
```

```java
public interface PilaTDA<T>{
    public boolean isEmpty();
    public T pop();
    public void push(T dato);
    public T peek();
}
```

Memoria dinámica:

- PilaB: El código muestra la implementación de una pila genérica utilizando la clase Stack. La clase “PilaB<T>” implementa la interfaz PilaTDA<T>y tiene un miembro de datos “Stack<T>” llamado “Pila”.

```java
public class PilaB<T> implements PilaTDA<T>{
  private Stack<T> pila;

  public PilaB() {
    pila = new Stack<T>();
  }
  public int Size(){
    return pila.size();
  }
  public boolean isEmpty() {
    return (pila.empty());
  }
  public T peek() {
    return (T) (pila.peek());
  }
  public void vaciar() {
    pila.clear();
  }
  public void push(T dato) {
    pila.push(dato);
  }
  public T pop() {
    T dato;
    dato = (T) pila.peek();
    pila.pop();
    return dato;
  }
  public String toString() {
    return toString(pila.size()-1);
  }
  private String toString(int i) {
    return (i ≥ 0 )
      ? "\n" + "tope[" + i + "]⟹" + "" + pila.get(i) + "" + toString(i - 1)
      : "";
  }
}
```

- PilaC: El código define una clase genérica “PilaC” que implementa la interfaz "PilaTDA”. Esta clase utiliza un ArrayList para implementar la estructura de pila.

  Los campos de la clase son:

  - **pila**: un ArrayList que contiene los elementos de la pila.
  - **tope**: un entero que indica la posición del tope de la pila.

El constructor de la clase inicializa el ArrayList y el tope a -1 para indicar que la pila está vacía.

```java
public class PilaC<T>implements PilaTDA<T>{
  private ArrayList pila;
  int tope;

  public PilaC() {
    pila = new ArrayList();
    tope = -1;
  }
  public int Size() {
    return pila.size();
  }
  public boolean isEmpty() {
    return pila.isEmpty();
  }
  public void vaciar() {
    pila.clear();
  }
  public void push(Object dato) {
    pila.add(dato);
    tope ++;
  }
  public T pop() {
    T dato = (T) pila.get(tope);
    pila.remove(tope);
    tope --;
    return dato;
  }
  public T peek() {
    return (T) pila.get(tope);
  }
  public String toString() {
    return toString(tope);
  }
  private String toString(int i) {
    return (i ≥ 0 )
      ? "\n" + "tope[" + i + "]⟹" + "[" + pila.get(i) + "]\n" + toString(i - 1)
      : "";
  }
}
```

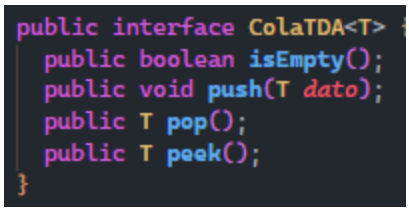
- PilaD: Este código define una implementación de la interfaz PilaTDA mediante una estructura de datos de pila basada en una lista enlazada. La lista enlazada se implementa mediante la clase “Nodo”, que contiene un dato de tipo “T” y un puntero al siguiente nodo de la lista.En la clase “PilaD”, el tope de la pila se representa mediante el nodo pila. Cuando se crea una nueva instancia de la pila (PilaD()), se inicializa el tope a null.

```java
public class PilaD<T> implements PilaTDA<T> {
  private Nodo pila;

  public PilaD() {
    pila = null;
  }
  public boolean isEmpty() {
    return (pila == null);
  }
  public void vaciar() {
    pila = null;
  }
  public void push(T dato) {
    Nodo tope = new Nodo(dato);
    if (isEmpty()) {
      pila = tope;
    } else {
      tope.sig = pila;
      pila = tope;
    }
  }
  public T pop() {
    T dato = (T) pila.getInfo();
    pila = pila.getSig();
    return dato;
  }
  public T peek() {
    return (T) (pila.getInfo());
  }
  public String toString() {
    Nodo tope = pila;
    return toString(tope);
  }
  private String toString(Nodo i) {
    return (i ≠ null)
      ? "Tope ⟹ " + "[" + i.getInfo() + "]\n" + toString(i.getSig())
      : "";
  }
}
```

```java
public class Nodo<T> {
  public T info;
  public Nodo sig;

  public Nodo(T info) {
    this.info = info;
    this.sig = null;
  }
  public T getInfo() {
    return info;
  }
  public void setInfo(T info) {
    this.info = info;
  }
  public Nodo getSig() {
    return sig;
  }
  public void setSig(Nodo sig) {
    this.sig = sig;
  }
}
```

Cola estática:

Dentro del paquete cola estática se crea una interfaz ColaTDA<T>



```
public interface ColaTDA<T> {
    public boolean isEmpty();
    public void push(T dato);
    public T pop();
    public T peek();
}
```

- ColaA: El código define una implementación de una cola estática genérica en Java, utilizando un arreglo de tamaño fijo para almacenar los elementos. Los métodos que se definen son los siguientes:

```java
public class ColaA<T> {
  private T cola[];
  private byte u;

  @SuppressWarnings("unchecked")
  public ColaA(int max) {
    cola = (T[])(new Object[max]);
    u =- 1;
  }
  public boolean isEmpty() {
    return (u == -1);
  }
  public boolean isSpace() {
    return (u < cola.length-1);
  }
  public void push(T dato) {
    if (isSpace()) {
      u++;
      cola[u] = dato;
    } else {
      toolsList.imprimeErrorMsg("Estructura llena");
    }
  }
  public T pop() {
    T dato = cola[0];
    return dato;
  }
  public T peek() {
    return (T)cola[0];
  }
  public String toString() {
    return toString(0);
  }
  private String toString(int i) {
    return (i ≤ u)
      ? "u ⟹ " + i + "[" + cola[i] + "]" + toString(i + 1)
      : "";
  }
}
```

- ColaB: Este código define una implementación de la interfaz ColaTDA utilizando una LinkedList de Java. La clase ColaB es genérica, lo que significa que puede manejar cualquier tipo de dato, que se especifica en tiempo de ejecución.

```java
public class ColaB<T> implements ColaTDA<T> {
  private Queue cola;

  public ColaB () {
    cola = new LinkedList();
  }
  public int Size () {
    return cola.size();
  }
  public boolean isEmpty () {
    return (cola.isEmpty());
  }
  public T peek () {
    return (T)(cola.element());
  }
  public void vaciar () {
    cola.clear();
  }
  public void push (T dato) {
    cola.add(dato);
  }
  public T pop () {
    T dato;
    dato = (T) cola.element();
    cola.remove();
    return dato;
  }
  public String toString () {
    String cad = "";
    byte j = 0;

    for (Iterator i = cola.iterator(); i.hasNext();) {
      cad += "[" + i.next() + "]" + j;
      j++;
    }
    return cad;
  }
}
```

Resultados.

Para comenzar con la explicación del código empezamos mostrando como es que se ejecuta el código.

Desde la clase Main estamos invocando la ejecución del menu para que le despliegue al usuario las respectivas opciones a ejecutar (pop, push, peek, free).

```java
package Main;

import Menu.Menu;

public class Main {
  public static void main(String[] args) {
    Menu.menu();
  }
}
```

Mediante este tipo de estructura de datos pilas y colas, se desarrolló un menú donde se crearon diferentes instancias de clase “PilaA, PilaB, PilaC, PilaD y ColaA,ColaB, ColaC, ColaD” en las que se implementaron clases genéricas que implementan la interfaz “PilaTDA” <T>. El parámetro de tipo “T” especifica el tipo de elemento que se almacenara en la pila

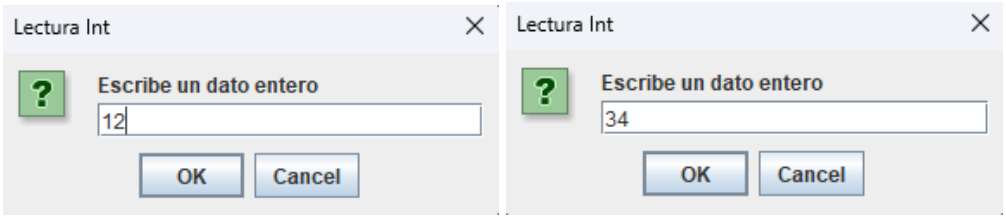
```
public class Menu {
  public static void menu() {

    PilaA<Integer> pila = new PilaA<Integer>((byte) 10);
    PilaB<Integer> pila = new PilaB<Integer>();
    PilaC<Integer> pila = new PilaC<Integer>();
    PilaD pila = new PilaD();

    ColaA<Integer> cola = new ColaA<Integer>((byte)10);

    String sel = "";

    do {
      sel = toolsList.boton("Pop,Push,Peek,Free,Salir");
      switch (sel) {
      case "Pop":
        if (cola.isEmpty()) {
          toolsList.imprimeErrorMsg("Pila vacia");
        } else {
          toolsList.imprimePantalla("Dato eliminado de la cima de la pila: " + cola.pop());
        }
        break;
      case "Push":
        cola.push(toolsList.leerInt("Escribe un dato entero"));
        toolsList.imprimePantalla("Datos de la pila\n" + cola.toString());
        break;
      case "Peek":
        if (cola.isEmpty()) {
          toolsList.imprimeErrorMsg("Esta vacio");
        } else {
          toolsList.imprimePantalla("Dato de la sima de la fila: " + cola.peek());
        }
        break;
      case "Free":
        if (cola.isEmpty()) {
          toolsList.imprimeErrorMsg("Pila vacia");
        }
        break;
      default:
      }
    } while (!sel.equalsIgnoreCase("Salir"));
  }
}
```

```
PilaA<Integer> pila = new PilaA<Integer>((byte) 10);
PilaB<Integer> pila = new PilaB<Integer>();
PilaC<Integer> pila = new PilaC<Integer>();
PilaD pila = new PilaD();

ColaA<Integer> cola = new ColaA<Integer>((byte)10);
```

Instancias con las cuales el usuario puede interactuar de manera dinámica mediante las diferentes opciones que proporciona el menú:

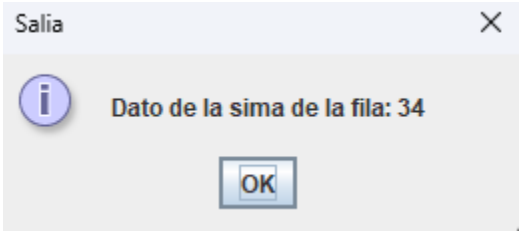
- Push: Añade un elemento en la cima.

```
case "Push":
  cola.push(toolsList.leerInt("Escribe un dato entero"));
  toolsList.imprimePantalla("Datos de la pila\n" + cola.toString());
  break;
```



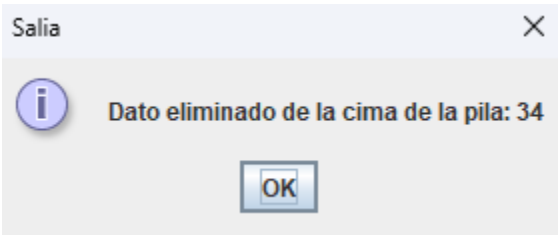
- Peek: Muestra el valor del elemento en la cima.

```
case "Peek":
  if (cola.isEmpty()) {
    toolsList.imprimeErrorMsg("Esta vacio");
  } else {
    toolsList.imprimePantalla("Dato de la sima de la fila: " + cola.peek());
  }
  break;
```



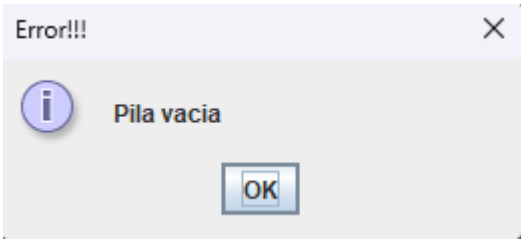
- Pop: Elimina el elemento de la cima de la es decir elimina el primer elemento añadido.

```
case "Pop":
  if (cola.isEmpty()) {
    toolsList.imprimeErrorMsg("Pila vacia");
  } else {
    toolsList.imprimePantalla("Dato eliminado de la cima de la pila: " + cola.pop());
  }
  break;
```



- Free: Vacía la cola completamente.

```
case "Free":
  if (cola.isEmpty()) {
    toolsList.imprimeErrorMsg("Pila vacia");
  }
  break;
```



Conclusión.

En conclusión, en este reporte se ha abordado el uso de las estructuras lineales de pilas y colas en Java, y se han presentado diversos ejemplos que permiten comprender su funcionamiento y su aplicación en distintos problemas y situaciones.

Se ha destacado la importancia de elegir la estructura adecuada en función de las necesidades específicas de cada problema, y se han presentado algunas buenas prácticas para la implementación y uso de estas estructuras.

En general, las estructuras de datos son un elemento fundamental en la programación, y las estructuras lineales de pilas y colas son muy utilizadas debido

a su simplicidad y eficiencia. En este sentido, el conocimiento y dominio de estas estructuras puede ser de gran utilidad para cualquier programador que busque desarrollar soluciones informáticas eficientes.

En resumen, este reporte de prácticas ha brindado una visión general del uso de las estructuras lineales de pilas y colas en Java, y esperamos que haya sido de utilidad para comprender su implementación y aplicación en la resolución de problemas informáticos.

Referencias.

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

Cola de Java: métodos de cola, implementación de cola y ejemplo - Otro. (s.f.). Los Comentarios, Juegos, Entretenimiento, abril 2023. https://spa.myservername.com/java-queue-queue-methods

Gonzalez, M. V. (2021, 24 de enero). Listas en Java. Codmind. https://blog.codmind.com/listas-en-java/

El uso de Listas en Java. (s.f.). Panama Hitek. https://panamahitek.com/el-uso-de-listas-en-java/#:~:text=El%20uso%20de%20listas%20en%20Java%20es%20una%20forma%20útil,de%20grandes%20cantidades%20de%20información.

Java ArrayList. Estructura dinámica de datos. (s.f.). Programación Java. http://puntocomnoesunlenguaje.blogspot.com/2012/12/arraylist-en-java.html