# PORTADA

UNIVERSIDAD TÉCNICA DE AMBATO

Facultad de Ingeniería en Sistemas, Electrónica e Industrial

Título: “Pilas y colas”

Carrera: Software

Nivel y Paralelo: 3RO “A”

Alumnos participantes: Barros Lozada Leonel

Jijón Viscaino Diego Patricio

López Garcés Elkin Joao

Paredes Lozada Ariel Mateo

Asignatura: Estructura de datos

Docente: Ing. Félix Fernandez M.Sc.

# INFORME DEL PROYECTO

## Título

Pilas y colas

## **Objetivos**

**Objetivo General**

Conocer el funcionamiento de pilas y colas.

**Objetivos Específicos**

* Describir las características y diferencias entre pilas y colas en Java, enfocándose en sus principios LIFO y FIFO.
* Implementar una aplicación en Java que inserte y extraiga datos de una pila y una cola, documentando el proceso con código y capturas.
* Analizar el comportamiento de pilas y colas mediante la aplicación, destacando sus aplicaciones prácticas.

## Palabras clave: (Pilas, Colas, LIFO, FIFO, Stacks, Queues)

## Introducción

En el ámbito de la programación, es fundamental entender cómo operan las diferentes estructuras de datos para seleccionar la más adecuada según las necesidades del problema a resolver. Las pilas (stacks) y las colas (queues) son dos de las estructuras de datos más utilizadas, cada una con características únicas que las hacen idóneas para ciertos tipos de operaciones y escenarios. Este informe se centra en explorar en profundidad estas dos estructuras de datos dentro del entorno de programación Java.

El objetivo principal de este estudio es conocer a fondo el funcionamiento de las pilas y las colas, identificando sus propiedades, diferencias y casos de uso específicos. Para lograr esto, se describirán sus características operativas básicas, incluyendo los principios de "último en entrar, primero en salir" (LIFO) para las pilas, y "primero en entrar, primero en salir" (FIFO) para las colas. Además, se implementará una aplicación en Java que permitirá insertar y extraer datos en ambas estructuras, proporcionando una plataforma para observar y comparar su comportamiento en tiempo real.

A través de la implementación práctica y el análisis comparativo, este informe no solo busca enriquecer la comprensión teórica de estas estructuras de datos, sino también demostrar su aplicabilidad y eficiencia en situaciones prácticas de programación.

## Resultados y Discusión (Desarrollo)

* + 1. **Pilas (Stacks)**

Una pila es una colección de elementos organizada según el principio de "último en entrar, primero en salir" (LIFO, Last In First Out). Esto significa que el último elemento añadido a la pila será el primero en ser eliminado.

**Operaciones:**

Las dos operaciones primarias en una pila son **push** (agregar un elemento a la pila) y **pop** (eliminar el elemento superior de la pila).

**isEmpty():** Verifica si la pila está vacía. Devuelve true si la pila no contiene elementos.

**size():** Devuelve el número de elementos en la pila. Esto es útil para controlar la cantidad de datos en la estructura de datos en cualquier momento.

**peek():** Permite ver el elemento en la cima de la pila sin eliminarlo. Este método es crucial para operaciones que requieren acceso al último elemento ingresado sin modificar la pila.

**Utilidad:** Ideal para algoritmos de navegación, como los utilizados en la función de deshacer (undo) en editores de texto, o para manejar llamadas recursivas en programación.

**Implementación:** En Java, las pilas se pueden implementar usando Stack<E> de la biblioteca Java Utilities (java.util), aunque también es común utilizar otras estructuras de datos, como listas enlazadas, para manejar dinámicamente los datos.

* + 1. **Colas (Queues)**

Una cola es una colección de elementos organizada según el principio de "primero en entrar, primero en salir" (FIFO, First In First Out). Esto significa que el primer elemento añadido será el primero en ser eliminado.

**Operaciones:  
isEmpty():** Similar a las pilas, este método verifica si la cola está vacía, devolviendo true si no hay elementos.

**size():** Retorna el número de elementos presentes en la cola, permitiendo el seguimiento del volumen de la cola.

**peek():** Permite acceder al primer elemento de la cola sin eliminarlo, proporcionando una forma de ver el próximo elemento a procesar sin modificar la cola.

**enqueue() / add():** Agrega un elemento al final de la cola. En Java, este método puede tener nombres como add(e) o offer(e), dependiendo de si la implementación debe devolver un valor booleano o lanzar una excepción si no puede agregar el elemento.

**dequeue() / remove():** Elimina el elemento al frente de la cola y lo devuelve. Similar al método add(), puede existir como remove() que lanza una excepción si la cola está vacía, o poll(), que devuelve null si la cola no tiene elementos.

**Utilidad:** Las colas son útiles en la programación de eventos, gestión de tareas y algoritmos de buffering, donde el orden de los elementos debe ser preservado.

**Implementación:** En Java, las colas se implementan comúnmente usando Queue<E> de la biblioteca Java Utilities (java.util), y se utilizan clases como LinkedList<E> o PriorityQueue<E> para diferentes comportamientos de ordenación y acceso.

En la comparación entre pilas (stacks) y colas (queues), ambos tipos de estructuras de datos manejan elementos con estrategias opuestas que influencian directamente su aplicación en escenarios de programación específicos. Las pilas, que operan bajo el principio de "último en entrar, primero en salir" (LIFO), son ideales para tareas que requieren un acceso inverso. Por otro lado, las colas, adheridas al principio de "primero en entrar, primero en salir" (FIFO), son cruciales en procesos donde se debe preservar el orden de llegada, tal como en la planificación de tareas y el manejo de colas de espera. Las pilas suelen implementarse con listas enlazadas que permiten expansión y contracción dinámica, facilitando operaciones rápidas de inserción y eliminación sin preocupaciones sobre el tamaño predefinido, mientras que las colas pueden usar tanto listas enlazadas como arreglos circulares, Esta distinción conceptual no solo afecta la elección técnica entre una estructura u otra, sino también la eficacia con la que se manejan los datos en diferentes contextos de programación, haciendo esencial una comprensión detallada de sus características y aplicaciones prácticas.

## Conclusiones

## Referencias bibliográficas