1.- HOJA DE PRESENTACION:

MATERIA: ESTRUCTURA DE DATOS

INGIENERIA INFORMATICA

T2EDEQUIPO1 (UNIDAD 2)

EQUIPO #1: Castro Ramón David Alejandro 20010329

Martínez Ramos Rodrigo 20010347

GRUPO:

Fecha de entrega: 27-marzo-2023

INDICE DEL REPORTE

2) INTRODUCCIÓN (RESUMEN):

En este reporte de la unidad 3 se explicarán todos los códigos/programas que en este caso abarcan los temas de Datos Simples o arreglos y también vimos ejemplos fáciles como lo son los números Armstrong, una Suma de Divisores y una Validacon entre Amigos.

Este reporte tiene el propósito de que pongamos y documentemos todos lo que se hizo dentro de las diversas unidades en la metería de estructura de datos (en este caso es la Unidad 3) en el cual se mostrara paso a paso como fue que fimos evolucionando en temas además de nuestro avance con los códigos que se presentaron por parte de la maestra.

Además, nos beneficiara en la parte de que repasaremos todo lo visto en las unidades anteriores y recalcar nuestros conocimientos prácticos para resolver problemas por nuestra propia cuenta basándonos en apuntes y temas/documentos previos para saber si los resultados son los esperados al finalizar no solo cada unidad sino la materia de estructura de datos en el presente semestre

Se explicará a manera detallada cada una de las pares o apartados que se nos presenta en este documento y ver nuestros avances en esta materia el cual se presentaran apartados donde se incluya:

1) Una página de presentación (o portada)
2) Introducción (Resumen):
3) Competencia especifica:
4) Marco teórico:
5) Material y Equipo:
6) Desarrollo de la practica:
7) Resultados:
8) Conclusiones:
9) Bibliografía:

3) COMPETENCIA ESPECIFICA:

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

Comprende y aplica estructuras no lineales para la 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.

4) MARCO TEÓRICO:

En programación, una estructura de datos es una forma de organizar y almacenar datos para su uso eficiente y efectivo. Las estructuras de datos lineales son aquellas en las que los datos se organizan en una secuencia ordenada, como una lista, una pila o una cola. Estas estructuras tienen un único camino de acceso a los datos. Por otro lado, las estructuras de datos no lineales, como los árboles o los grafos, permiten múltiples caminos de acceso a los datos y su organización es más compleja. Estas estructuras se utilizan para problemas más complejos y pueden tener un mayor rendimiento en la resolución de problemas que las estructuras lineales. La elección de la estructura de datos adecuada depende del problema a resolver y del rendimiento requerido por la aplicación.

5) MATERIAL Y EQUIPO:

✓ ***Computadora:*** Cualquier dispositivo electrónico (Computafora o Laptop) que tenga buen funcionamiento en sistema y que pueda ser compatible con un lenguaje de programación para que se pueda instalar, además de una buena memoria de almacenamiento.

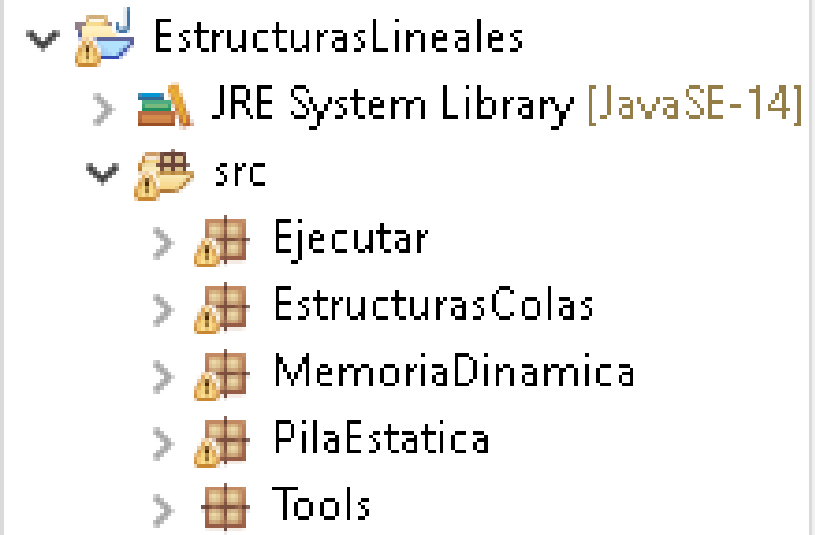
✓ ***Software y versión usados:*** En la creación de los programas se utilizó el programa/aplicación de Eclipse que se instala por medio de su JDK Y el IDE que se puede encontrar en la plataforma del lenguaje de programación

✓***Materiales de apoyo para el desarrollo de la practica:*** Documentos de apoyo para la parte teórica presentados por la maestra el cual contenía los códigos y ejemplos de los programas que vemos y pasarlos con nuestros conocimientos al lenguaje de programación que elegimos en este caso Eclipse Versión 2023

6) DESARROLLO DE LA PRACTICA: ***7) RESULTADOS:***

EJERCICIOS DE PRACTICA

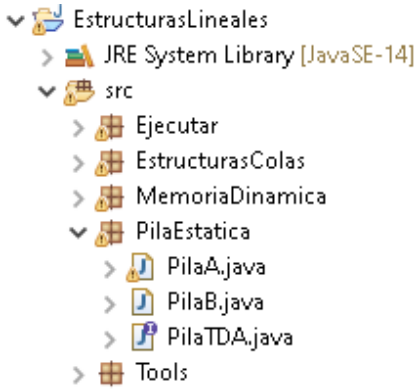
Primero comenzamos creando un nuevo proyecto el cual llamaremos Estructuras Lineales en el cual crearemos 5 paquetes los cuales se llamarán: Ejecutar, Estructuras colas, Memoria dinámica, Pila estática y Tools.



En el paquete Ejecutar tendremos nuestro menú el cual se encargará de llamar a todos nuestros métodos.

En el paquete Tools tendremos todos los métodos que nos ayudaran a leer los datos que inserte el usuario.

Empezaremos por los códigos del paquete Pila Estática.



Pila TDA

```
1  package PilaEstatica;
2
3  public interface PilaTDA<T>{
4
5      public boolean isEmpty();
6
7      public T pop();
8
9      public void push(T dato);
10
11     public T peek();
12
13     public void freePila();
14
15 }
16
```

Primero crearemos una interfaz a la cual le llamaremos PilaTDA.

Esta es una clase de interfaz la cual es una plantilla genérica que utilizaremos después para crear las diferentes pilas que usaremos, como será una plantilla genérica tenemos que declarar que será de tipo indefinido para después en tiempo de ejecución poder asignarle cualquier tipo que queramos.

En la línea 5:

Declaramos un método público que regresara un dato booleano el cual se llamara isEmpty() el cual se encargara de verificar si la pila está vacía o no.

En la línea 7:

Declaramos un método público que regresara un dato como no sabemos de qué tipo lo declaramos como genérico, esta clase se llamara pop el cual se encargara de sacar un dato de la pila.

En la línea 9:

Declaramos un método de tipo público que no regresará nada y se llamará push como parámetro recibirá un dato.

En la línea 11:

Declaramos un método público que regresara un dato de tipo indefinido por ahora el método se llamara peek el cual se encargara de regresar un dato de la pila.

En la línea 13:

Declaramos un método público que no regresara nada el cual llamaremos freePila el cual se encargara de vaciar la pila.

Pila A:

Creamos una clase llamada PilaA:

```
1  package PilaEstatica;
2
3  import Tools.ToolsPanel;
4
5  public class PilaA <T> implements PilaTDA<T>{
```

Esta clase se encargara de gestionar una pila a la cual se le dara el tamaño y el tipo en el menu, como en este momento no sabemos que tipo de datos entraran la clase sera generica osea que haceptara cualquier tipo de dato y sera de un tamaño indefinido hasta su ejecucion, a esta se le implementara la calse PilaTDA bista anterior mente para que tenga todos sus metodos.

```
7      private T pila[];
8      private byte tope;
```

En la linea 7:

Crearemos nuestra pila la cual sera privada osea que solo sera visible en esa clase la declaramos como tipo indefinido.

En la linea 8:

Creamos una variable la cual se encargara de llevar el conteo de datos que hay en nuestra pila la declaramos como tipo bite he igual solo sera vicible para esta clase.

Despues de esto crearemos nuestro constructor Linea 10-13:

```
10      public PilaA(int max) {
11          pila=(T[])(new Object[max]);
12          tope=-1;
13      }
```

El cual sera el primero en ejecutarse en cuanto se llame a esta clase ya que este le dara el tipo de datos que almacenara y el tamaño a la pila, tambien se iniciara a tope con -1 ya que en este momento no tiene ningun dato dentro.

Implementamos el metodo isEmpty Lina 15-17:

```
15      public boolean isEmpty(){
16          return (tope==-1);
17      }
```

El cual se encargara de retornar un true o false dependiendo si nuestra variable tope es igual con -1 esto quiere decir que si es verdadero (true) nuestra pila esta vacia.

Creamos un metodo llamado isSpace Linea 19-21:

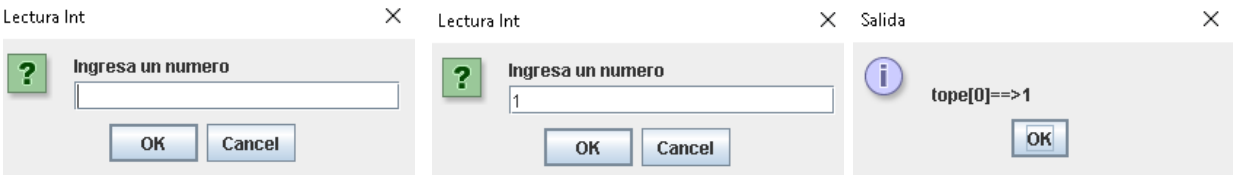
```
19      public boolean isSpace() {
20          return(tope<pila.length-1);
21      }
```

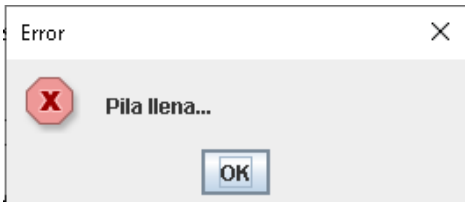
El cual se encargara de retornar un true o false dependiendo si nuestra pila esta llena, para esto compararemos el tamaño de nuestra pila (pila.length) con nuestra varibale tope el cual lleva el conteo de cuantos datos ay en nuestra pila, mientras tope sea menor que el tamaño de la pila regresara un true lo cual significa que aun ay espacion en la pila.

Implementamos el metodo push Linea 23-30:

```
23      public void push(T dato) {
24          if(isSpace()) {
25              tope++;
26              pila[tope]=dato;
27          }else {
28              ToolsPanel.imprimeError("Pila llena...");
29          }
30      }
```

El cual se encarga de insertar datos en nuestra pila, primero resiviremos como parametro el dato que queremos insertar, verificamos si nuestra pila tiene espacio para insertarlo, si hay espacio la variable tope aumentara uno y guardaremos el dato en la ultima posicionde la pila, si no hay espacio mandaremos un mensaje de pila llena.

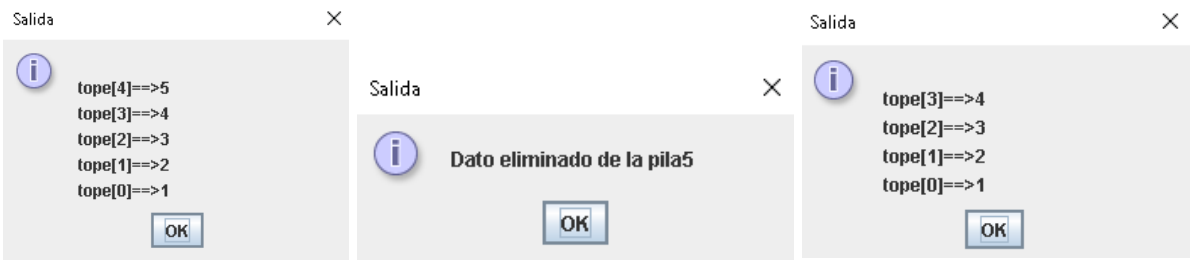




Implementamos el metodo pop linea 32-36:

```
32      public T pop() {
33          T dato = pila[tope];
34          tope--;
35          return dato;
36      }
```

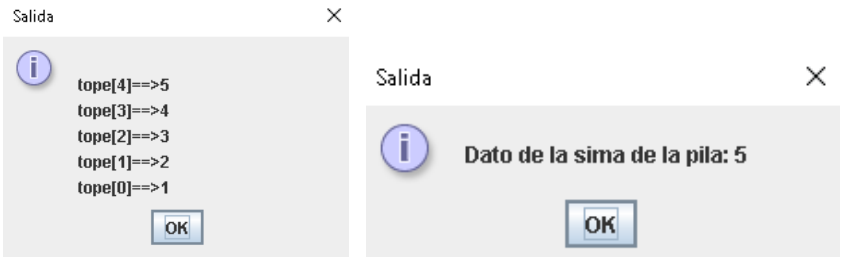
El cual se encargara de guardar el ultimo dato de la pila en una variable que llamaremos dato y a tope le quitaremos uno para eliminar el ultimo dato de la pila y al final retornamos el dato que fue eliminado.



Implementamos el metodo peek Linea 38-40:

```
38      public T peek() {
39          return pila[tope];
40      }
```

El cual solo se encarga de regresar el ultimo dato que hay en la pila.



Creamos los metodos toString Linea 42-48:

```
42    public String toString() {
43        return toString(tope);
44    }
45
46    public String toString(int i) {
47        return (i>=0)?"\n"+"tope["+i+"]==>"+""+pila[i]+" "+toString(i-1):"";
48    }
```

El primero se encargara de enviarle al segundo el tamaño del tope para asi poder manipularlo en el segundo toString, en el se ocupara la forma recursiva para ir imprimiendo los valores que existen en la pila uno por uno hasta llegar al final, en este caso como es una pila el ultimo digito que se inserto sera el primero en mostrarse.

Pila B:

Creamos una nueva clase la cual llamaremos pila B:

```
1  package PilaEstatica;
2
3  import java.util.Stack;
4
5  public class PilaB<T> implements PilaTDA<T>{
```

A esta clase se encargara de gestionar una pila en este caso utilisaremos la memoria stack y junto con sus metodos para poder agregar, eliminar, etc. Tambien implementaremos la clase PilaTDA para obtener sus metodos genericos.

Linea 7:

```
7      private Stack<T> pila;
```

Empezamos creando una pila privada de tipo generico en el stack.

Creamos un constructor Linea 9-11:

```
9     public PilaB() {
10        pila=new Stack<T>();
11    }
```

Que iniciara nuestra pila generica.

Implementamos el metodo isEmpty Linea 13-16

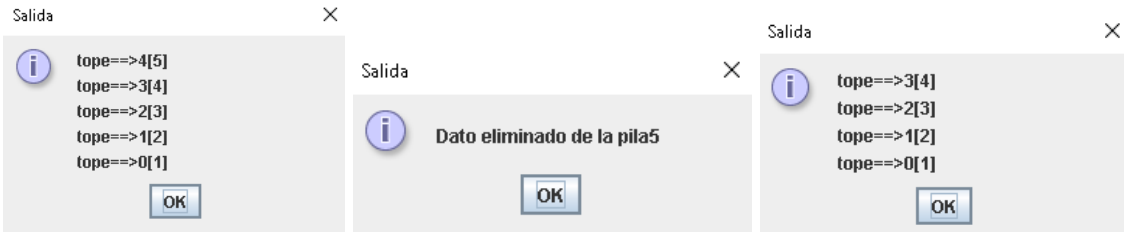
```
13      @Override
14      public boolean isEmpty() {
15          return (pila.empty());
16      }
```

Implementamos el metodo isEmpty el cual se encargara de retornar si nuestra pila esta vacia o no.

Implementamos el metodo pop Linea 18-24:

```
18      @Override
19      public T pop() {
20          T dato;
21          dato=(T)pila.peek();
22          pila.pop();
23          return dato;
24      }
```

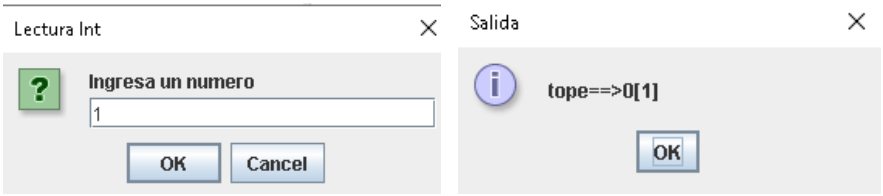
El cual se encargara de crear un variable generica llamada dato en la cual guardaremos el ultimo dato de la pila, el siguiente paso sera eliminar ese mismo dato y al final solo retornamos la variable dato para mostrar el dato eliminado, al eliminar este dato tambien se elimina su espacio en el stack cumpliendo asi el ultimo en entrar el primero en salir.



Implementamos el metodo push Linea 26-29 :

```
26      @Override
27      public void push(T dato) {
28          pila.push(dato);
29      }
```

El cual nos trae un parametro generico llamado dato el cual sera ingresado al final de la pila, cunado este dato se agrega a la pila en el stack se crean nuevos espacios.



Implementamos el metodo peek Linea :

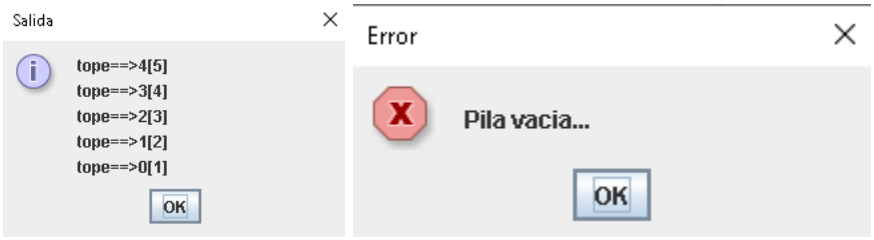
```
31      @Override
32      public T peek() {
33          return (T)(pila.peek());
34      }
```

El cual regresara el ultimo elemento que se encuentra en la pila.

Implementamos el metodo freePila Linea 36-39 :

```
36      @Override
37      public void freePila() {
38          pila.clear();
39      }
```

Este metodo se encargara de vacial la pila por completo eliminando todos los espacion de la pila en el stack.



Implementamos el metodo Size Linea 41-43:

```
41      public int Size() {
42          return pila.size();
43      }
```

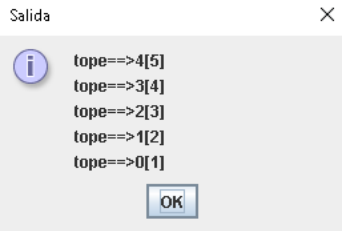
El cual se encargara de regresar el numero de elementos que existen dentro de nuesta pila.

Creamos los metodos toString Linea :

```
45      public String toString() {
46          return toString(Size()-1);
47      }
48      private String toString(int i) {
49          return (i>=0)?"tope==>"+i+"["+pila.get(i)+"]\n"+toString(i-1):"";
50      }
```

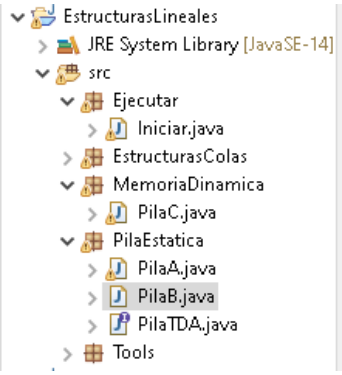
El primero se encargara de enviarle al segundo el tamaño del tope para asi poder manipularlo en el segundo toString, en el se ocupara la forma recursiva para ir

imprimiendo los valores que existen en la pila uno por uno hasta llegar al final, en este caso como es una pila el ultimo digito que se inserto sera el primero en mostrarse.



Pila C:

En el paquete de memoria dinamica creamos una nueva clase que llamaremos Pila C:




```
1  package MemoriaDinamica;
2
3  import java.util.ArrayList;
6
7  public class PilaC<T> implements PilaTDA<T>{
```

Esta clase se encargara de gestionar un arraylist el cual sera dinamico osea que se iran creando los espacion en tiempo de ejecucion, esto significa que no ay un limite asignado. A esta clase tambien le implementaremos la clase PilaTDA.

Empezamos por crear nuestra pila Linea 9-10:

```
 9      private ArrayList pila;
10
11      int tope=-1;
```

La cual sera privada de tipo ArrayList y el tope que llevara el conteo de cuantos datos ay en cada espacio de la pila lo inciamos con -1 indicando que no existe ningun dato aun.

En el constructor iniciaremos nuestra pila y el tope.

```
13      public PilaC() {
14          pila = new ArrayList();
15          int tope;
16      }
```

Creamos el metodo Size Linea 18-20:

```
18      public int Size() {
19          return pila.size();
20      }
```

El cual nos dara el dato exacto de cuantos elementos ay en nuestra pila.

Implementamos el metodo isEmpty Linea :

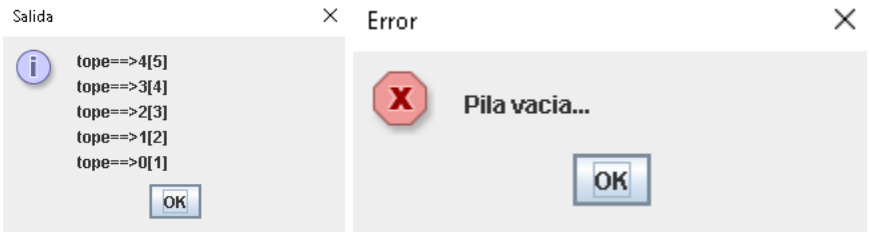
```
22      @Override
23      public boolean isEmpty() {
24          return pila.isEmpty();
25      }
```

El cual retornara si nuesta pila esta vacia o no.

Creamos el metodo vaciar Linea 27-29:

```
27      public void vaciar() {
28          pila.clear();
29      }
```

El cual se encargara de eliminar todos los datos de nuestra pila.



Implementamos el metodo push Linea 31-35:

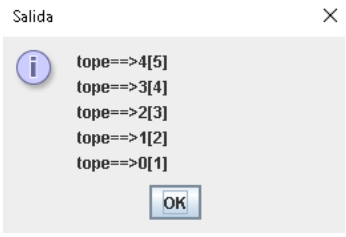
```
31      @Override
32      public void push(T dato) {
33          pila.add(dato);
34          tope++;
35      }
```

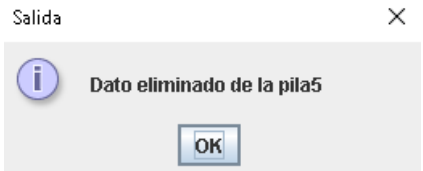
El cual resive como parametro generico dato, agregamos ese dato a nuestra pila y tope le aumentamos uno para saber cual es su posicion.

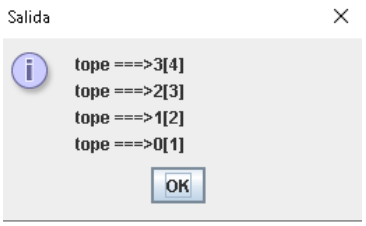
Implementamos el metodo pop Linea 37-43:

```
37      @Override
38      public T pop() {
39          T dato=(T)pila.get(tope);
40          pila.remove(tope);
41          tope--;
42          return dato;
43      }
```

El cual se encargara de sacar el ultimo dato de la pila y guardarlo en una bariable una ves guardada se eliminara de la pila y tope disminuira uno al final solo se mostrara el dato que fue removido.



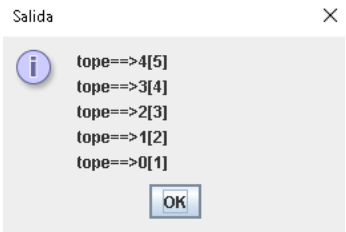


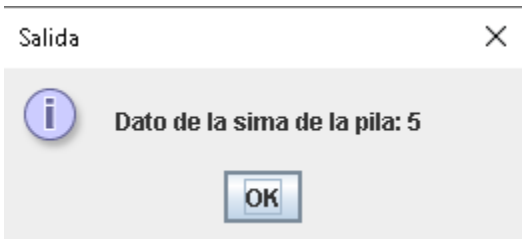


Implementamos el metodo peek Linea 45-48:

```
45      @Override
46      public T peek() {
47          return (T)pila.get(tope);
48      }
```

El cual solo se encarga de devolver el ultimo dato de la pila pero sin eliminarlo.

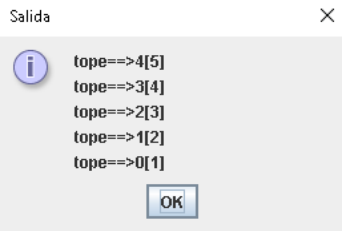




Creamos los metodos toString Linea :

```
50      public String toString() {
51          return toString(tope);
52      }
53
54      public String toString(int i) {
55          return (i>=0)?"tope ===>"+i+"["+pila.get(i)+"]\n"+toString(i-1):"";
56      }
```

El primero se encargara de enviarle al segundo el tamaño del tope para asi poder manipularlo en el segundo toString, en el se ocupara la forma recursiva para ir imprimiendo los valores que existen en la pila uno por uno hasta llegar al final, en este caso como es una pila el ultimo digito que se inserto sera el primero en mostrarse.



ColaTDA:

Ahora en el paquete estructuras colas creamos una clase interface llamada ColaTDA que sera una plantilla generica.

```
1  package EstructurasColas;
2
3  public interface ColaTDA <T>{
4
5      public boolean isEmptyCola();
6
7      public void pushCola(T Dato);
8
9      public T popCola();
10
11     public T peekCola();
12
13     public void freeCola();
14
15 }
```

En la linea 5 creamos los metodos isEmptyCola como booleano

En la linea 7 pushCola no regresara nada pero tendra un parametro generico llamado Dato

En la linea 9 popCola regresara un dato

En la linea 11 peekCola regresara un dato

En la linea 13 al final freeCola el cual no regresara nada ni llevara parametros.

Cola A:

Creamos una nueva clase llamada cola A:

```
1  package EstructurasColas;
2
3  import Tools.ToolsPanel;
4
5  public class ColaA<T> implements ColaTDA<T>{
```

Le implementaremos la clase ColaTDA osea nuestra plantilla generica, esta clase se encargara de administrar nuestra lista llamada cola en este caso el primero que entra es el primero que sale.

Primero iniciamos dos variables privadas Linea 7-8:

```
7       private T cola[];
8       private byte u;
```

Una llamada cola de tipo generica y la otra llamada u de tipo byte.

Iniciamos nuestro constructor Linea 10-13:

```
10      public ColaA(int max) {
11          cola = (T[]) (new Object[max]);
12          u=-1;
13      }
```

El cual resivira como parametro el tamaño que tendra nuestra cola, iniciamos nuestro arreglo cola con el tamaño resivido y u lo inciamos con -1 esto significa que no ay datos en el arreglo actual mente.

Implementamos el metodo isEmpty Linea:

```
15      @Override
16      public boolean isEmptyCola() {
17          return (u==-1);
18      }
```

El cual se encargara de decirnos si nuestro arreglo esta vacio o no.

Creamos el metodo isSpace Linea 20-22:

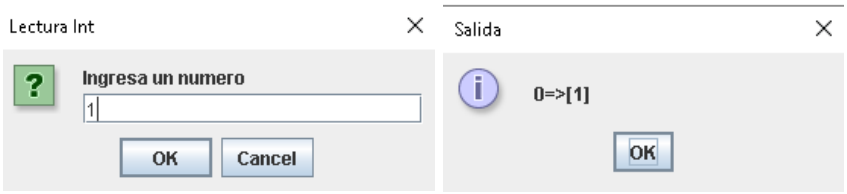
```
20      public boolean isSpace() {
21          return (u<cola.length-1);
22      }
```

El cual nos dira si existe espacio en nuestro arreglo comparando la variable u el cual lleva el conteo de cuantos datos ay en el arreglo contra el tamaño que tiene la cola.

Implementamos el metodo pushCola Linea 24-32:

```
24      @Override
25      public void pushCola(T Dato) {
26          if(isSpace()) {
27              u++;
28              cola[u]=Dato;
29          }else {
30              ToolsPanel.imprimeError("Cola llena...");
31          }
32      }
```

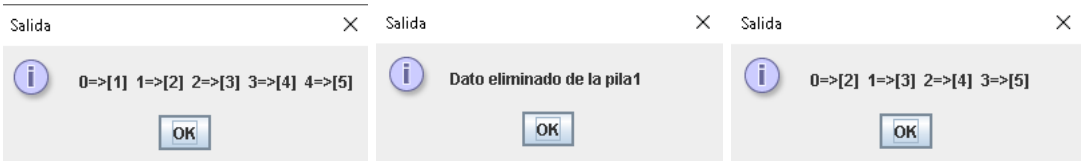
Resivira un parametro llamado dato, primero se confirmara si existe espacio en la cola o ya esta llena, si existe espacio en la cola u aumentara en uno y se agregara en la ultima posicion de la cola el dato resivido anteriormente. En caso de que no exista espacio en la cola se mandara un mensaje que dira que la cola esta llena.



Implementamos el metodo popCola Linea 34-42:

```
34      @Override
35      public T popCola() {
36          T Dato = cola[0];
37          for(int k=0; k<=u; k++) {
38              cola[k]=cola[k+1];
39          }
40          u--;
41          return Dato;
42      }
```

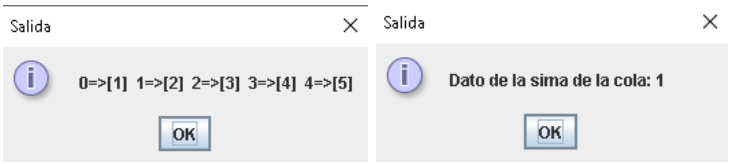
Primero guardamos el dato que eliminaremos en una variable llamada dato, en este caso el que eliminaremos sera el primero, para eliminarlo simplemente tendremos que recorrer todos los datos una casilla antes y u le disminuimos uno, al final solo regresamos el dato que fue eliminado.



Implementamos el metodo peekCola Linea 44-47:

```
44      @Override
45      public T peekCola() {
46          return (T)cola[0];
47      }
```

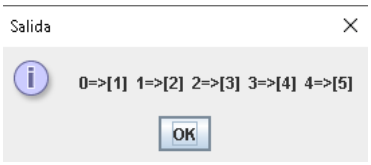
Este metodo solo regresara el mimer dato que existe en la cola.



Creamos los metodos toString Linea :

```
49      public String toString() {
50          return toString(0);
51      }
52
53      public String toString(int i) {
54          return (i<=u)?" "+i+"=>["+""+cola[i]+"] "+toString(i+1):"";
55      }
```

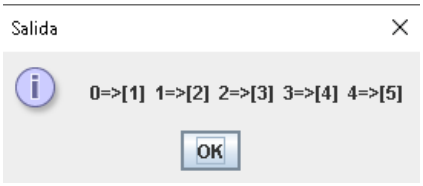
El primero se encargara de enviarle al segundo el tamaño del tope para asi poder manipularlo en el segundo toString, en el se ocupara la forma recursiva para ir imprimiendo los valores que existen en la cola uno por uno hasta llegar al final, en este caso como es una pila el primer digito que se inserto sera el primero en salir.

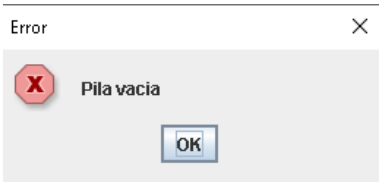


Implementamos el metodo freeCola Linea :

```
57      @Override
58      public void freeCola() {
59          u=-1;
60      }
```

Este metodo solo volvera a poner a u en -1 haciendo que la cola se vuelva a reiniciar y se pierdan todos los datos.





8) CONCLUSIONES:

Los resultados esperados fueron los resultados obtenidos debido a que seguimos paso por paso las instrucciones prestadas y dadas por la docente a lo largo de la unidad 3.

Además, todo nos salió gracias a los trabajos de los ejercicios y actividades realizadas tanto como en el salón de clases como en el laboratorio de centro de computo en donde pudimos aclarar y aprender temas básicos a la introducción de la estructura de datos haciendo ejercicios básicos como:

Memoria estática-Memoria dinámica:

- En esta unidad pudimos utilizar tanto la memoria estática y la dinámica manipulando los datos de diferente manera aun que en su mayoría los resultados arrojados son muy similares en cada clase el método que se emplea es diferente y cada uno tiene sus veneficios y desventajas dependiendo de cómo quieras utilizarlas

9) Bibliografía:

Tonkseverus. (n.d.). *Estructuras lineales y no lineales*. http://estructuradedatos10111248.blogspot.com/2015/07/estructuras-lineales-y-no-lineales.html