



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



**TECNM**  
TECNOLÓGICO NACIONAL DE  
MÉXICO



**Tecnológico Nacional de México  
Campus Orizaba**

**Ingeniería en Sistemas Computacionales**

**Estructura de datos**

**Unidad 3**

**Tema:**

**Clases genéricas, arreglos dinámicos, Pilas y Colas**

**Integrantes:**

**Castillo Solis Luis Ángel – 21010932**

**Muñoz Hernández Vania Lizeth – 21011009**

**Romero Ovando Karyme Michelle – 21011037**

**Grupo:**

**3g2B**

**Fecha de entrega: 22/Abril /2023**

## 1. Introducción.

Una pila en Java es una estructura de datos que sigue el principio LIFO (Last In, First Out), lo que significa que el último elemento que se agrega a la pila es el primero en ser eliminado. La pila es similar a una pila de platos en la que los platos se agregan y eliminan del tope. En Java, puedes implementar una pila utilizando una clase de la biblioteca estándar llamada `java.util.Stack`. Esta clase ya tiene implementados los métodos comunes para agregar y eliminar elementos de la pila, así como para obtener el elemento en la parte superior de la pila sin eliminarlo. También puedes implementar tu propia clase de pila si lo deseas. Para agregar un elemento a la pila, utilizas el método `push()`, y para eliminar un elemento de la pila, utilizas el método `pop()`. Puedes obtener el elemento en la parte superior de la pila utilizando el método `peek()`. También puedes comprobar si la pila está vacía utilizando el método `empty()`. Es importante tener en cuenta que, al igual que con cualquier estructura de datos, debes manejar correctamente las excepciones y los errores que puedan surgir al trabajar con pilas en Java.

Por otra parte cola en Java es una estructura de datos que sigue el principio FIFO (First In, First Out), lo que significa que el primer elemento que se agrega a la cola es el primero en ser eliminado. Es similar a una fila de personas esperando en un supermercado: la primera persona en llegar es la primera en ser atendida. En Java, puedes implementar una cola utilizando una clase de la biblioteca estándar llamada `java.util.Queue`. Esta clase tiene implementados los métodos comunes para agregar y eliminar elementos de la cola, así como para obtener el elemento frontal sin eliminarlo. También puedes implementar tu propia clase de cola si lo deseas. Para agregar un elemento a la cola, utilizas el método `offer()`, y para eliminar un elemento de la cola, utilizas el método `poll()`. Puedes obtener el elemento frontal de la cola utilizando el método `peek()`. También puedes comprobar si la cola está vacía utilizando el método `isEmpty()`. Es importante tener en cuenta que, al igual que con cualquier estructura de datos, debes manejar correctamente las excepciones y los errores que puedan surgir al trabajar con colas en Java. Además, hay diferentes tipos de colas que puedes utilizar según tus necesidades, como colas de prioridad o colas circulares.

## 2. Competencia específica

Comprende y aplica estructuras de datos lineales para solución de problemas.

## 3. Marco Teórico

**Pilas:** Una pila es una estructura de datos lineal que permite almacenar y acceder a elementos siguiendo una política LIFO (Last-In-First-Out), es decir, el último elemento en entrar es el primero en salir. Se puede pensar en una pila como una pila de platos en un restaurante, donde los nuevos platos se colocan en la parte superior de la pila y los clientes siempre toman el plato superior.

Las operaciones básicas de una pila son "push" y "pop". "Push" se utiliza para agregar un elemento a la parte superior de la pila, mientras que "pop" se utiliza para eliminar el elemento superior de la pila. Además, también es común tener la operación "peek" para acceder al elemento superior de la pila sin eliminarlo.

**Colas:** Una cola es una estructura de datos lineal que permite almacenar y acceder a elementos siguiendo una política FIFO (First-In-First-Out), es decir, el primer elemento en entrar es el primero en salir. Se puede pensar en una cola como una línea en un cine, donde las personas que llegan primero son las primeras en entrar y las personas que llegan después tienen que esperar detrás de ellos.

Las operaciones básicas de una cola son "enqueue" y "dequeue". "Enqueue" se utiliza para agregar un elemento al final de la cola, mientras que "dequeue" se utiliza para eliminar el elemento del frente de la cola. Al igual que con las pilas, también es común tener la operación "peek" para acceder al elemento del frente de la cola sin eliminarlo.

**Listas:** Una lista es una estructura de datos lineal que permite almacenar y acceder a elementos de forma ordenada. A diferencia de las pilas y colas, las listas no tienen una política predeterminada para el acceso a elementos. En

cambio, los elementos se pueden acceder y manipular de varias maneras diferentes, dependiendo de cómo se implemente la lista. Existen varios tipos de listas, como las listas simplemente enlazadas, las listas doblemente enlazadas y las listas circulares. Cada tipo de lista tiene sus propias características y operaciones únicas, pero todas tienen en común que permiten agregar y eliminar elementos en cualquier posición de la lista, así como acceder a los elementos por índice o por valor.

#### **4. Material y Equipo**

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

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

#### **5. Desarrollo de la practica.**

Pasos para desarrollar la practica:

1. Ejecutar una aplicación que utilice java, como Eclipse, NetBeans, etc.
2. Crear un proyecto, al nuestro le llamaremos “Estructuras lineales”.
3. Una vez creado el proyecto crearemos paquetes que tengan alojados cada una de las clases que utilizaremos.
4. Finalizaremos con la creación de las diversas clases, cada una en su respectivo paquete.

Pasos para crear la interfaz de una plantilla generica “PilaTDA”.

- Sobre el paquete en donde alojaremos nuestra interface daremos clic derecho, seguido nos coloreamos sobre la palabra “new” y daremos clic sobre “Java Interface” para después ponerle el nombre.

- Una vez creada para hacerla una plantilla genérica colocaremos entre flechas la letra “T” tomándola como genérica como se muestra a continuación:

```
 public interface PilaTDA <T> {
```

- Para finalizar esta plantilla crearemos los métodos que utilizara la plantilla y será implementada dentro de las diferentes clases de pila que se crearan, estos métodos serán:
  - ❖ isEmpty() .- Este será un método booleano que regresara un true si la pila esta vacia.
  - ❖ push() .- El método push insertara el dato en el tope de la pila.
  - ❖ pop() .- Pop hará lo contrario a push, este en lugar de insertar un dato lo eliminara del tope de la pila.
  - ❖ peek() .- Por ultimo peek te regresara el elemento que este en el tope de la pila sin quitarlo.

#### Pasos para la creación de la clase “PilaA”

- Primero implementaremos la plantilla genérica que creamos antes, llamada “PilaTDA”.
- Seguimos creando el arreglo “pila[]” y una variable de tipo byte llamada “tope”.
- Despues crearemos un constructor que se encargue de crear una pila vacia llamada “PilaA” donde “max” sea el tamaño máximo de la pila que se especifica como argumento.
- Seguido de ello crearemos el método isEmpty usando un Override ya que estamos sobre escribiendo un método de la interfaz implementada, este método comprueba si la pila está vacía, si devuelve un true es que está vacía y si es false es que la pila tiene datos dentro, este método se utilizara para verificar si se puede eliminar un elemento de la pila si es que esta tiene datos o esta vacía.
- Ahora se crea el método isSpace que verificara si hay espacio dentro de la pila, si devuelve un true es porque aun hay espacio dentro de la

pila, de lo contrario arroja un false, este método puede utilizarse para evitar un desborde dentro de la pila.

- Seguiremos con el método pushPila, nuevamente agregaremos un Override, este método es el que ingresa cada uno de los datos a la pila, primero verificando si hay espacio disponible dentro de la misma usando el método antes creado, en caso de no haber espacio, mostrara un mensaje de que la pila esta llena.
- El método popPila es el siguiente, este método toma el ultimo dato agregado a la pila y lo elimina, en este método debemos tener mucho cuidado, ya que en caso de que la pila este vacía podría generarse una excepción.
- Ahora se creará el método peekPila, este método toma el ultimo dato ingresado y lo regresa para mostrarlo, se debe tener cuidado ya que si se llama el método teniendo la pila vacía, muy posiblemente tengamos una excepción.
- Por último crearemos un método llamado toString, utilizando la recursividad crearemos un código que llame a toString para crear una representación en formato de cadena de los elementos de la pila.

En este punto haremos una observación, ya que en lugar de crear una hipotética clase "PilaB" en su lugar utilizaremos la platilla por defecto que nos da java importando "java.util.Stack" directamente en la clase de nuestro menú.

Pasos para crear la clase "PilaC"

- Iniciaremos creando dos atributos, una ArrayList llamado pila y uno llamado tope.
- El siguiente paso es crear el constructor, este constructor inicializa la pila con el ArrayList y establece el valor del tope en -1, lo que indica que la pila esta vacia.
- Nuestro primer método a crear será el método Size, este método se utiliza para saber el tamaño actual de la pila, nos será útil para saber si la pila esta vacia antes de intentar agregar un elemento.

- Por ultimo creamos un método llamado Vaciar, este método eliminara todo lo que este dentro de la pila y así liberar el espacio en la memoria utilizado por la pila.

Una observación en este punto es que al igual que la PilaA, la PilaC también contiene los métodos creados en la primer pila, solamente con algunos ajustes pero dándonos el mismo funcionamiento, al igual que la clase PilaD tendrá los mismos métodos, con una variación de como se implementa pero teniendo a final de cuentas el mismo funcionamiento.

Pasos para crear la clase “Nodo”

- Primero crearemos dos atributos llamados info, este almacenara el valor del nodo y sig que será el que apunte al siguiente nodo de la lista.
- El constructor que crearemos aceptara un parámetro de tipo genérico que represente el valor del nodo y establezca el valor en el atributo info, el atributo sig se inicializara en null ya que aun no se a agregado ningún nodo adicional.
- Por ultimo agregaremos los setters y getters a la clase.

Antes de seguir con el tema de colas, se realizaron ejercicios de pilas, donde teníamos que crear una clase que almacenara una cadena y dividiera en partes dicha cadena, primero por símbolos, después por lo que se queda dentro de la pila y lo que se concatena dentro de la cadena.

¿Cómo se creo la clase Carácter?

La notación infija es la que usamos normalmente para escribir expresiones matemáticas, por ejemplo:  $(2 + 3) * 4 - 5 / 6$ .

En la notación posfija, los operadores se colocan después de los operandos, por ejemplo:  $2\ 3\ +\ 4\ *\ 5\ 6\ /\ -$ . La idea del algoritmo es recorrer la cadena de entrada carácter por carácter y procesar cada carácter de acuerdo a su tipo. Si el carácter es un paréntesis de apertura ('('), se agrega a la pila. Si es una letra, se agrega directamente a la cadena de salida. Si es un operador, se compara su jerarquía con la del operador en el tope de la pila. Si la jerarquía del operador en la pila es mayor o igual a la del operador actual, se saca el

operador de la pila y se agrega a la cadena de salida. Luego se agrega el operador actual a la pila. Si el carácter es un paréntesis de cierre (')'), se sacan los operadores de la pila y se agregan a la cadena de salida hasta encontrar el paréntesis de apertura correspondiente, que también se saca de la pila.

Al final del recorrido, se sacan los operadores restantes de la pila y se agregan a la cadena de salida.

La función `operador` verifica si un carácter es un operador válido (+, -, \*, /, ^). La función `jerarquia` devuelve la jerarquía de un operador (3 para '^', 2 para '\*' y '/', 1 para '+' y '-'). La variable `posfija` es la cadena de salida en notación posfija.

Es importante mencionar que este código no devuelve la cadena de expresión posfija, sino que simplemente la construye en la variable `posfija`. Para obtener la cadena de salida, se debe agregar al final de la función `cadenaPosfija` la instrucción `return posfija;`.

Seguido de ello el algoritmo utiliza una pila (objeto de tipo `Stack`) para almacenar los operadores y paréntesis de la expresión mientras se procesa. Se utiliza un ciclo `for` para recorrer cada carácter de la cadena de entrada (expresión infija) y se toman diferentes acciones en función del carácter:

- Si es un paréntesis de apertura ('('), se empuja a la pila.
- Si es una letra (a-z o A-Z), se agrega directamente a la expresión posfija y se agrega una fila a la tabla de la GUI para mostrar el estado actual de la pila y la expresión posfija.
- Si es un operador (+, -, \*, /, ^), se comprueba si la pila ya tiene operadores. Si el operador actual tiene una jerarquía menor o igual que el operador en la cima de la pila, se saca el operador de la pila y se agrega a la expresión posfija. Este proceso se repite hasta que el operador actual tenga una jerarquía mayor que el operador en la cima de la pila, o hasta que la pila esté vacía. Luego se empuja el operador actual a la pila.



- Si es un paréntesis de cierre (')'), se sacan operadores de la pila y se agregan a la expresión posfija hasta encontrar el paréntesis de apertura correspondiente. Luego se saca ese paréntesis de la pila.

Después de procesar todos los caracteres de la cadena de entrada, se sacan todos los operadores restantes de la pila y se agregan a la expresión posfija. Por último, se muestra la pila y la expresión posfija en una tabla de la GUI utilizando un objeto DefaultTableModel y un objeto JTable dentro de un JScrollPane.

En resumen, en esta clase se muestra un algoritmo para convertir una expresión aritmética infija a posfija y proporciona una GUI para visualizar el estado de la pila y la expresión posfija en cada paso del algoritmo.

Pasos para elaborar la interfaz de una plantilla genérica “ColaTDA”

- Para crear la plantilla debemos hacer exactamente los mismos pasos para crear la interface como lo hicimos con la plantilla de pila, solo cambiaremos las “pila” por “cola” ya que tendrán un uso similar con las colas.

Pasos para crear la clase “ColaA”

- Crearemos un método que recorra los datos de la pila llamado RecorrePosiciones, este método realiza un recorrido en la cola.
- ColaA es una calca de la clase PilaA, los únicos cambios son en el nombre en los atributos, ya que tenemos un funcionamiento similar, el cambio ahora es que cola en el método popCola eliminara el primer elemento insertado a diferencia de popPila que elimina el ultimo que fue insertado.

Pasos para crear la clase “ColaB”

- A diferencia de las pilas, en colas si creamos su clase B, aunque también importaremos una clase que ya esta definida por java, siendo esta clase una calca a la PilaC, con los mismos métodos solo que con adecuaciones para colas.

Pasos para crear la clase “ColaC”

- ColaC es igual a PilaD ya que se implementa una ArrayList, nuevamente el único cambio son la forma de crear los métodos, adecuando el mismo para colas.

5. El ultimo paso en la practica es crear un menú para cada una de las clases creadas (PilaA, Stack, PilaC, PilaD, Nodo, Carácter, ColaA, ColaB y ColaC).

## 6. Resultados

Interface PilaTDA

Código	
1	<code>package PilaEstatica;</code>
2	
3	<code>public interface PilaTDA &lt;T&gt; {</code>
4	
5	<code>    public boolean isEmptyPila(); //Regresa verdadero si la pila esta vacia.</code>
6	<code>    public void pushPila(T dato); // Inserta el dato en el tope de la pila.</code>
7	<code>    public T popPila(); //Elimina el elemento que esta en el tope de la pila.</code>
8	<code>    public T peekPila(); // Regresa el elemento que esta en el tope sin quitarlo.</code>
9	
10	<code>}</code>

## Clase PilaA

### Código

```
1  package PilaEstatica;
2  //datos estaticos
3  import EntradaSalida.Tools;
4
5  public class PilaA <T> implements PilaTDA<T> {
6
7      private T pila[];
8      private byte tope;
9
10     public PilaA(int max) {
11         pila = (T[]) (new Object[max]);
12         tope = -1;
13     }
14
15     @Override
16     public boolean isEmptyPila() {
17         return (tope == -1);
18     }
19
20     public boolean isSpace() {
21         return (tope < pila.length - 1);
22     }
23 }
```

```
24      @Override
25      public void pushPila(T dato){
26
27          if(isSpace()){
28              tope++;
29              pila[tope] = dato;
30          }
31          else Tools.errorMsj("PILA LLENA");
32      }
```

```
33
34      @Override
35      public T popPila(){
36          T dato = pila[tope];
37          tope--;
38          return dato;
39      }
```

```
40
41      @Override
42      public T peekPila(){
43          return pila[tope];
44      }
```

```
45
46      @Override
47      public String toString(){
48          return toString(tope);
49      }
50
51      private String toString(int i){
52          return (i >=0)? "tope ==>" + i + " [ " + pila[i] + " ] " + toString(i-1): "";
53      }
54  }
```


```

1 package estructuraslineales;
2 import EntradaSalida.Tools;
3 import javax.swing.JOptionPane;
4 import PilaEstatica.*;
5 public class EstructurasLineales {
6
7     // DESPLEGABLE
8
9     public static void operacionesPilaEstatica(String menu) {
10         PilaA <Integer> pila = new PilaA((byte)10);
11         //int op;
12         String op;
13         do {
14             op = desplegable(menu);
15             switch (op) {
16                 case "PUSH": pila.pushPila(Tools.leeInt("Escribe un dato entero:"));
17                     Tools.imprime("Datos en pila:\n" + pila.toString());
18                     break;
19                 case "POP": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
20                     else Tools.imprime("Dato eliminado de la cima: " + pila.popPila() + "\n" + pila.toString());
21                     break;
22
23                 case "PEEK": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
24                     else Tools.imprime("Dato en la cima de la pila: ==>" + pila.peekPila() + "\n" + pila.toString());
25                     break;
26                 case "FREE": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
27                     else {
28                         pila = null;
29                         pila = new PilaA((byte)10);
30                     }
31                     break;
32                 default: Tools.imprime("Programa Temrinado"); break;
33             } //switch
34         } while (! op.equals("SALIR"));
35     }
36     public static String desplegable(String menu) {
37         String valores[] = menu.split(",");
38         String res = (String) JOptionPane.showInputDialog(null, "M E N U", "Selecciona opcion:", JOptionPane.QUESTION_MESSAGE,
39             null, valores, valores[0]);
40         return (res);
41     }
42     public static void main(String[] args) {
43         String menu = "PUSH, POP, PEEK, FREE, SALIR";
44         operacionesPilaEstatica(menu);
45     }
46 }

```

## Ejecución


Selección de opción: × Int ×



**M E N U**

PUSH

Aceptar Cancelar




**Escribe un dato entero:**

Aceptar Cancelar

---


Int × Información ×



**Escribe un dato entero:**

19

Aceptar Cancelar



**Datos en pila:**

tope ==>0 [ 19 ]

Aceptar

Información



Datos en pila:

tope ==>3 [ 7 ] tope ==>2 [ 3 ] tope ==>1 [ 24 ] tope ==>0 [ 19 ]

Aceptar

Selecciona opcion:



M E N U

POP

Aceptar

Cancelar

Información



Dsto eliminado de la cima: 7

tope ==>2 [ 3 ] tope ==>1 [ 24 ] tope ==>0 [ 19 ]

Aceptar

Selecciona opcion:



M E N U

PEEK

Aceptar

Cancelar

Información



Dato en la cima de la pila: ==>3

tope ==>2 [ 3 ] tope ==>1 [ 24 ] tope ==>0 [ 19 ]

Aceptar

Selecciona opcion:



M E N U

FREE

Aceptar

Cancelar

Selecciona opcion:



M E N U

PEEK

Aceptar

Cancelar

Error.



Pila vacía

Aceptar

Selecciona opcion:



M E N U

SALIR

Aceptar

Cancelar

Información



Programa Temrinado

Aceptar

## Funcionamiento

El código define una clase llamada **PilaA** que implementa la interfaz **PilaTDA**. Esta clase representa una pila genérica que almacena elementos del tipo **T**. La pila se implementa mediante un arreglo **pila**, y un índice **tope** que indica la posición del elemento en la cima de la pila. La clase tiene métodos para verificar si la pila está vacía o si hay espacio disponible para agregar elementos a la pila, eliminar elementos de la pila, obtener el elemento en la cima de la pila y convertir la pila en una cadena de texto. Si se intenta agregar un elemento adicional a una pila llena, se muestra un mensaje de error mediante la herramienta Tools.

## Clase EstructurasLineales

### Código

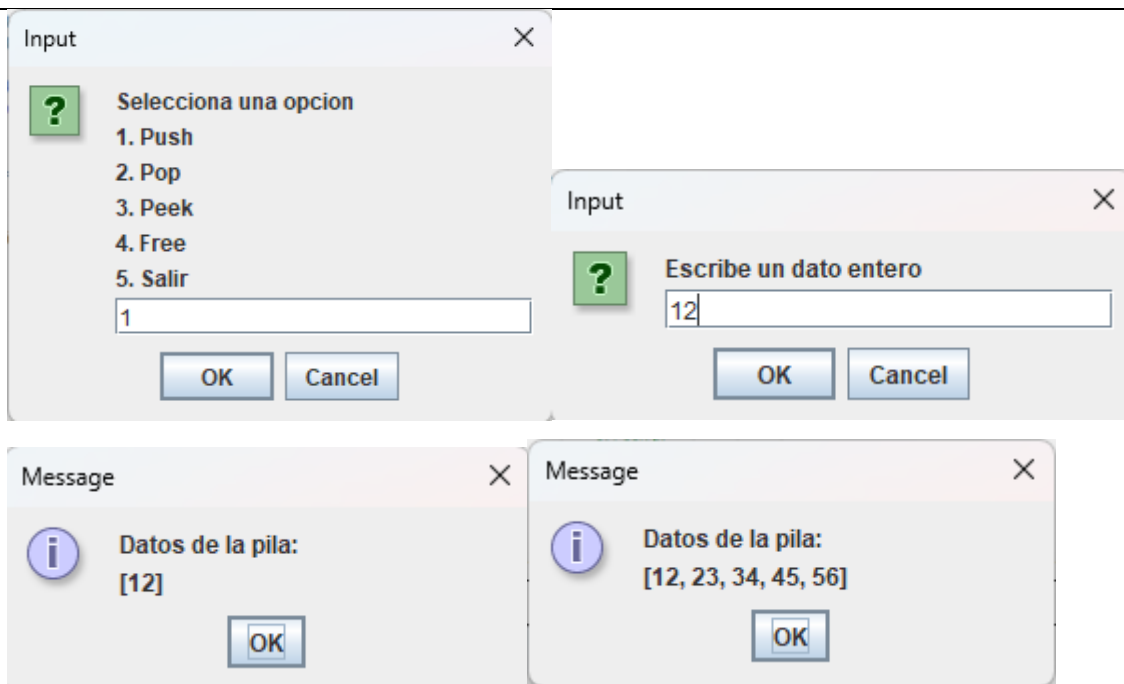
```
1 package PilaDinamica;
2
3 import EntradaSalida.Tools;
4 import java.util.Stack;
5
6
7 public class EstructurasLineales {
8
9     public static void MenuPilaEstatica(String Menu){
10
11         Stack pila = new Stack();
12         int sel;
13         do{
14             sel=Tools.leeEntero( msg: Menu);
15             switch(sel){
16                 case 1: pila.push( item: Tools.leeEntero( msg: "Escribe un dato entero"));
17                     Tools.ImprimeMensaje("Datos de la pila: \n" + pila.toString());
18                     break;
19                 case 2: if (pila.isEmpty()){
20                     Tools.ImprimeMensaje( msg: "Pila Vacía");
21                 }
22                 else{
23                     Tools.ImprimeMensaje("Dato eliminado de la cima de la pila: ==>" + pila.pop() + "\n" + pila.toString());
24                 }
25                 break;
26                 case 3: if (pila.isEmpty()){
27                     Tools.ImprimeMensaje( msg: "Pila Vacía");
28                 }
29                 else{
```

```

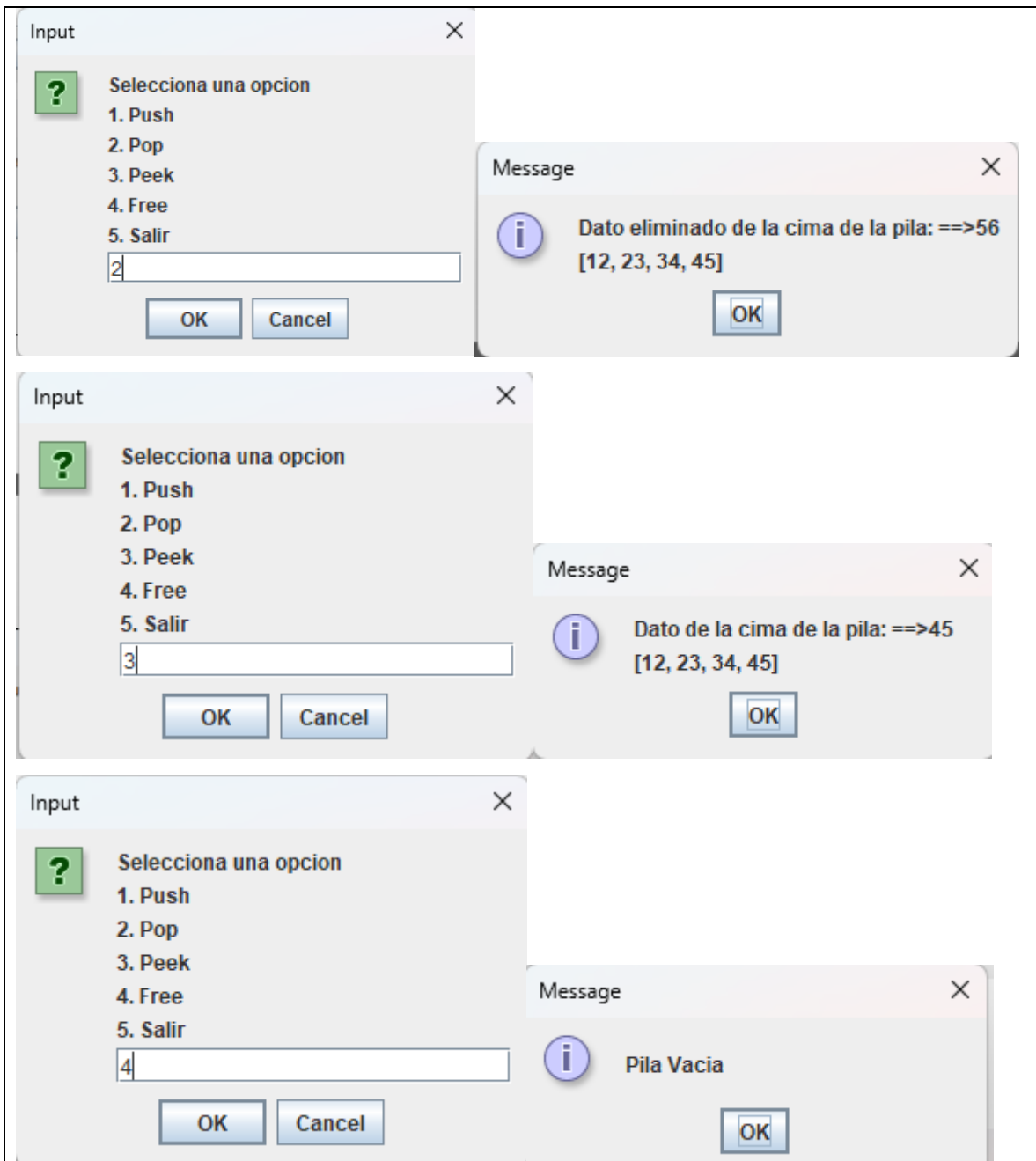
30         Tools.ImprimeMensaje("Dato de la cima de la pila: ==>" + pila.peek() + "\n" + pila.toString());
31     }
32     break;
33     case 4: if (pila.isEmpty()) {
34         Tools.ImprimeMensaje( msg: "Pila vacia");
35     }
36     else{
37         pila = null;
38         pila = new Stack();
39     }
40     break;
41     default: Tools.ImprimeMensaje( msg: "Opcion no definida, intenta de nuevo");
42 }
43 }
44 while(sell=5);
45 }
46 public static void main(String []args){
47     EstructurasLineales.MenuPilaEstatica("Selecciona una opcion \n"
48         +"1. Push\n"
49         +"2. Pop\n"
50         +"3. Peek\n"
51         +"4. Free\n"
52         +"5. Salir");
53 }
54
55 }

```

## Ejecución







### Funcionamiento

La pila se crea utilizando la clase Stack de Java.

La función MenuPilaEstatica es la que maneja el menú y recibe como parámetro un String que contiene las opciones que el usuario puede seleccionar.

Dentro de un bucle do-while, la función muestra el menú utilizando el parámetro recibido y espera a que el usuario seleccione una opción mediante la lectura de un número entero utilizando el método leeEntero de la clase Tools.

Después de obtener la selección del usuario, se ejecuta un switch-case que realiza la acción correspondiente según la opción seleccionada. Las opciones son las siguientes:

push: agrega un nuevo elemento a la cima de la pila, el usuario debe proporcionar un valor entero mediante el método leeEntero de la clase Tools.

pop: elimina y devuelve el elemento en la cima de la pila, si la pila está vacía se muestra un mensaje indicándolo.

peek: devuelve el elemento en la cima de la pila sin eliminarlo, si la pila está vacía se muestra un mensaje indicándolo.

free: vacía completamente la pila, es decir, la deja sin elementos.

salir: termina la ejecución del menú.

La función main simplemente llama a MenuPilaEstatica con el menú definido y espera a que el usuario termine la ejecución del menú.

## Clase PilaC

Código
--------

```
1 package PilaDinamica;
2
3 import EntradaSalida.Tools;
4 import PilaEstatica.PilaTDA;
5 import java.util.ArrayList;
6 import javax.swing.JOptionPane;
7
8 public class PilaC <T> implements PilaTDA <T>{
9
10     private ArrayList pila;
11     int tope;
12
13     public PilaC() {
14         pila = new ArrayList();
15         tope = -1;
16     }
17
18     public int Size() {
19         return pila.size();
20     }
21
22     @Override
23     public boolean isEmptyPila() {
24         return pila.isEmpty();
25     }
```

```

26
27 public void vaciar() {
28     pila.clear();
29 }
30
31 @Override
32 public void pushPila(Object dato) {
33     pila.add(dato);
34     tope++;
35 }
36
37 @Override
38 public T popPila() {
39     T dato = (T) pila.get(tope);
40     pila.remove(tope);
41     tope--;
42     return dato;
43 }
44
45 @Override
46 public T peekPila() {
47     return (T) pila.get(tope);
48 }
49
50 @Override
51 public String toString() {
52     return toString(tope);
53 }
54
55 private String toString(int i) {
56     return ( i >= 0 ) ? "tope==> " + i + " [ " +
57         pila.get(i) + " ]\n" + toString(i-1)
58         : "";
59 }
60
61

```

Menú Ejecutable:

```

62 //***** MENU
63 public static void menuPilaC (String menu){
64     PilaC pila = new PilaC();
65     //int op;
66     String op;
67     do {
68         op = desplegable(menu);
69         switch (op) {
70             case "PUSH": pila.pushPila(Tools.leeInt("Escribe un dato entero:"));
71                         Tools.imprime("Datos en pila:\n" + pila.toString());
72                         break;
73             case "POP": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
74                         else Tools.imprime("Dato eliminado: " + pila.popPila()+ "\n" + pila);
75                         break;
76             case "PEEK": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
77                         else Tools.imprime("Dato en la cima de la pila: ==>" + pila.peekPila()+ "\n" + pila);
78                         break;
79             case "VACIAR": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
80                         else{
81                             pila.vaciar();
82                             // pila = new PilaC();
83                         }
84                         break;
85             default: Tools.imprime("Programa Temrinado"); break;
86         } //switch
87     } while (! op.equals("SALIR"));
88 }
89
90 public static String desplegable(String menu) {
91     String valores[] = menu.split(",");
92     String res = (String) JOptionPane.showInputDialog(null,"M E N U"," Selecciona opcion:",
93     JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
94     return (res);
95 }
96
97 public static void main(String[] args) {
98     String menu = "PUSH,POP,PEEK,VACIAR,SALIR";
99     menuPilaC(menu);
100 }
101

```

## Ejecución

Selecciona opcion:

?

M E N U

PUSH

Aceptar

Cancelar

Int

Escribe un dato entero:

44

Aceptar

Cancelar

Información

i

Datos en pila:

[44]

Aceptar

Información

i

Datos en pila:

[44, 77, 22, 111]

Aceptar

Selecciona opcion:

?

M E N U

POP

Aceptar

Cancelar

Información

i

Dato eliminado: 111

[44, 77, 22]

Aceptar



### Funcionamiento

En lugar de utilizar la clase Stack de Java, esta implementación utiliza un ArrayList para almacenar los elementos de la pila.

La clase PilaC implementa la interfaz PilaTDA<T>, que define los métodos básicos que debe tener una pila, como pushPila, popPila, peekPila, isEmptyPila, entre otros. La T en la definición de la clase y en la interfaz significa que la pila es genérica, es decir, puede almacenar elementos de cualquier tipo.

El constructor de la clase PilaC inicializa el ArrayList y establece el tope de la pila en -1, lo que indica que la pila está vacía.

El método Size devuelve el tamaño de la pila, es decir, el número de elementos que contiene.

El método isEmptyPila verifica si la pila está vacía y devuelve un valor booleano indicando si es así o no.

El método vaciar vacía la pila completamente, eliminando todos los elementos.

El método pushPila agrega un elemento a la cima de la pila, es decir, al final del ArrayList, y actualiza el tope de la pila.

El método popPila elimina y devuelve el elemento en la cima de la pila, es decir, el último elemento del ArrayList, y actualiza el tope de la pila.

El método peekPila devuelve el elemento en la cima de la pila sin eliminarlo.

El método toString devuelve una representación en forma de cadena de la pila. El método privado toString(int i) es un método recursivo que construye la cadena utilizando un índice i que comienza en el tope de la pila y se decrementa recursivamente hasta cero, imprimiendo cada elemento y su posición en la pila.

En resumen, esta implementación de una pila genérica utiliza un ArrayList para almacenar los elementos de la pila y proporciona los métodos necesarios para trabajar con la pila, como agregar, eliminar y obtener elementos, así como verificar su estado y vaciarla completamente.

## Clase PilaD

### Código

```
1 package PilaDinamica;
2
3 import PilaEstatica.PilaTDA;
4
5 public class PilaD <T> implements PilaTDA <T>{
6
7     private Nodo pila;
8     public PilaD(){
9         pila = null;
10    }
11
12    @Override
13    public boolean isEmptyPila(){
14        return (pila == null);
15    }
16    @Override
17    public void pushPila(T dato){
18        Nodo tope = new Nodo( info: dato);
19        if (isEmptyPila()) {
20            pila = tope;
21        }
22        else{
23            tope.sig = pila;
24        }
25    }
26    @Override
27    public T popPila(){
28        Nodo tope = pila;
29        T dato = (T) pila.getInfo();
```



```

30         pila = pila.getSig();
31         tope = null;
32         return dato;
33     }
34     @Override
35     public T peekPila(){
36         return (T)(pila.getInfo());
37     }
38     @Override
39     public String toString(){
40         Nodo tope = pila;
41         return toString(i: tope);
42     }
43     private String toString(Nodo i){
44         return (i != null)?"Tope ==>" + "[" + i.getInfo() + "]\n" + toString(i: i.getSig()):"";
45     }
46 }

```

```

1 package PilaDinamica;
2
3 import EntradaSalida.Tools;
4 import java.util.Stack;
5 import javax.swing.JOptionPane;
6 //pila dinamica
7 public class PilaD {
8     public static void opPilaDinamica(String menu){
9         Stack pila = new Stack();
10        String op;
11        do {
12            op = desplegable(menu);
13            switch (op) {
14                case "PUSH": pila.push(Tools.leeInt("Escribre un dato entero:"));
15                            Tools.imprime("Datos en pila:\n" + pila.toString());
16                            break;
17                case "POP": if(pila.empty()) Tools.errorMsj("Pila vacía");
18                            else Tools.imprime("Dato eliminado: " + pila.pop() + "\n" + pila);
19                            break;
20                case "PEEK": if(pila.empty()) Tools.errorMsj("Pila vacía");
21                            else Tools.imprime("Dato en la cima de la pila: ==>" +
22                            pila.peek() + "\n" + pila);
23                            break;

```

```

24         case "FREE": if(pila.empty()) Tools.errorMsj("Pila vacía");
25                     else{
26                         pila = null;
27                         pila = new Stack();
28                     }
29                     break;
30         default: Tools.imprime("Programa Temrinado"); break;
31     } //switch
32     } while (! op.equals("SALIR"));
33 }
34
35 public static String desplegable(String menu) {
36     String valores[] = menu.split(",");
37     String res = (String) JOptionPane.showInputDialog(null,"M E N U"," Selecciona opcion:"
38     JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
39     return (res);
40 }
41
42 public static void main(String[] args) {
43     String menu ="PUSH,POP,PEEK,FREE,SALIR";
44     opPilaDinamica(menu);
45 }
46 }

```

## Ejecución

Selecciona opcion:
X
Int
X

**M E N U**

**Escribe un dato entero:**

Información
X
Información
X

**Datos en pila:**  
[44]

**Datos en pila:**  
[44, 77, 22, 111]

Selecciona opcion:
X
Información
X

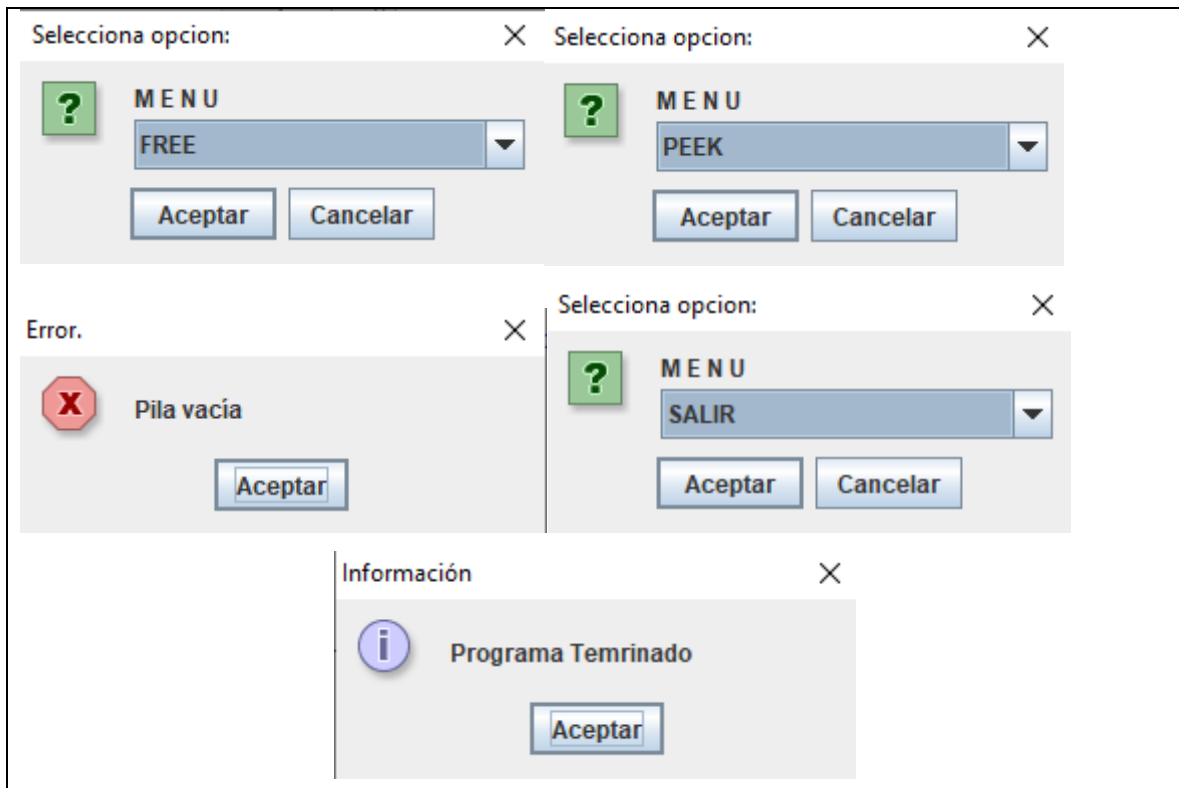
**M E N U**

**Dato eliminado: 111**  
[44, 77, 22]

Selecciona opcion:
X
Información
X

**M E N U**

**Dato en la cima de la pila: ==>22**  
[44, 77, 22]



### Funcionamiento

El código, resumidamente hace lo mismo que nuestra PilaA, la diferencia radica en que comenzamos a utilizar pilas dinámicas, es así que este código define una clase llamada PilaD que permite interactuar con una pila dinámica implementada mediante una instancia de la clase Stack. La clase tiene un método principal que despliega un menú con opciones para agregar elementos a la pila, eliminar elementos de la pila, obtener el elemento en la cima de la pila y liberar la memoria asignada a la pila. Tal cual hicimos en la pila estática y haremos en el resto de pilas dinámicas

El programa utiliza la herramienta JOptionPane para mostrar el menú y recibir la selección del usuario. Además, utiliza la clase Tools para mostrar mensajes de error o información en ventana.

Clase Nodo

Código
--------

```
1 package PilaDinamica;
2 public class Nodo<T> {
3     public T info;
4     public Nodo sig;
5
6     public Nodo(T info){
7         this.info = info;
8         this.sig = null;
9     }
10
11     public T getInfo() {
12         return info;
13     }
14
15     public void setInfo(T info) {
16         this.info = info;
17     }
18
19     public Nodo getSig() {
20         return sig;
21     }
22     public void setSig(Nodo sig) {
23         this.sig = sig;
24     }
25 }
```

```
1 package PilaDinamica;
2 import EntradaSalida.Tools;
3 import PilaEstatica.PilaTDA;
4 import javax.swing.JOptionPane;
5
6 public class PilaNodo <T> implements PilaTDA<T>{
7
8     private Nodo pila;
9     public PilaNodo(){
10         pila = null;
11     }
12
13     @Override
14     public void pushPila(T dato){
15         Nodo tope = new Nodo(dato);
16         if(isEmptyPila()) pila = tope;
17         else {
18             tope.sig = pila;
19             pila = tope;
20         }
21     }
22 }
```

```

23      @Override
24      public boolean isEmptyPila() {
25          return (pila == null);
26      }
27
28      @Override
29      public T popPila() {
30          Nodo tope = pila;
31          T dato = (T) pila.getInfo();
32          pila = pila.getSig();
33          tope = null;
34          return dato;
35      }
36
37      @Override
38      public T peekPila() {
39          return (T) pila.getInfo();
40      }
41
42      public void vaciar(){
43          pila = null;
44      }
45

```

```

46      @Override
47      public String toString(){
48          Nodo tope = pila;
49          return toString(tope);
50      }
51
52      public String toString(Nodo i){
53          return (i!= null)? //"tope ==>" +
54              " [ " + i.getInfo() + " ]\n"
55              + toString(i.getSig()) : "" ;
56      }
57
58
59      //*****

```

Menú ejecutable:

```

60 public static void opPilaNodo(String menu){
61     PilaNodo pila = new PilaNodo();
62     String op;
63     do {
64         op = desplegable(menu);
65         switch (op) {
66             case "PUSH": pila.pushPila(Tools.leeInt("Escribe un dato entero:"));
67                 Tools.imprime("Datos en pila:\n" + pila.toString());
68                 break;
69             case "POP": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
70                 else Tools.imprime("Dato eliminado: " + pila.popPila()+ "\n" + pila);
71                 break;
72             case "PEEK": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
73                 else Tools.imprime("Dato en la cima de la pila: ==>" +
74                     pila.peekPila()+ "\n" + pila);
75                 break;
76             case "FREE": if(pila.isEmptyPila()) Tools.errorMsj("Pila vacía");
77                 else{
78                     pila.vaciar();
79                 }
80                 break;
81             default: Tools.imprime("Programa Temrinado"); break;
82         } //switch
83     } while (! op.equals("SALIR"));
84 }

85
86 public static String desplegable(String menu) {
87     String valores[] = menu.split(",");
88     String res = (String) JOptionPane.showInputDialog(null,"M E N U"," Selecciona opcion:",
89         JOptionPane.QUESTION_MESSAGE,null,valores,valores[0]);
90     return (res);
91 }
92
93 public static void main(String[] args) {
94     String menu ="PUSH,POP,PEEK,FREE,SALIR";
95     opPilaNodo(menu);
96 }
97

```

## Ejecución

The screenshot displays the execution of the program through three sequential dialog boxes:

- Selecciona opcion:** A dialog box titled "Selecciona opcion:" with a close button (X) and a text label "Int". It contains a green question mark icon, a label "MENU", a dropdown menu showing "PUSH", and two buttons: "Aceptar" and "Cancelar".
- Escribe un dato entero:** A dialog box titled "Escribe un dato entero:" with a close button (X) and an information icon (i). It contains a text input field with the value "777" and two buttons: "Aceptar" and "Cancelar".
- Información:** A dialog box titled "Información" with a close button (X) and an information icon (i). It contains a label "Datos en pila:" followed by a list of values: "[ 444 ]", "[ 333 ]", "[ 1010 ]", and "[ 777 ]". Below the list is an "Aceptar" button.



Información

Selecciona opcion: X

?

M E N U

POP

Aceptar

Cancelar

i

Dato eliminado: 444

[ 333 ]

[ 1010 ]

[ 777 ]

Aceptar

Información

Selecciona opcion: X

?

M E N U

PEEK

Aceptar

Cancelar

i

Dato en la cima de la pila: ==>333

[ 333 ]

[ 1010 ]

[ 777 ]

Aceptar

Selecciona opcion: X

?

M E N U

FREE

Aceptar

Cancelar

?

M E N U

PEEK

Aceptar

Cancelar

Error.

X

Pila vacía

Aceptar

Selecciona opcion: X

?

M E N U

SALIR

Aceptar

Cancelar

Información

i

Programa Temrinado

Aceptar

## Funcionamiento

El código define una clase **PilaNodo** que implementa la interfaz **PilaTDA** y representa una pila de datos mediante una estructura de nodos enlazados. La clase tiene un atributo **pila** que es una referencia al primer nodo de la pila y métodos para realizar las operaciones básicas de la pila, como agregar un elemento, quitar el elemento superior, revisar el elemento superior, vaciar la pila y mostrar los elementos de la pila en una cadena de texto; a los cuales, tal como en programas anteriores, se accede mediante un menú.

## Clase Character

### Código

```
1  package EjercicioClase;
2  import java.util.Stack;
3  import EntradaSalida.Tools;
4  import java.awt.BorderLayout;
5  import javax.swing.BoxLayout;
6  import javax.swing.JFrame;
7  import javax.swing.JPanel;
8  import javax.swing.JScrollPane;
9  import javax.swing.JTable;
10 import javax.swing.table.DefaultTableModel;
11
12 public class Character {
13     public void cadenaPosfija(String cad){
14
15         Stack pila = new Stack();
16         String posfija = "";
17
18         for(char i: cad.toCharArray()){
19
20             if(i == '('){
21                 pila.push(i);
22             }
23
24             if(Character.isLetter(ch: i)){
25                 posfija += i;
26             }
27
28             if(operador(cad: i)){
29                 while(!pila.isEmpty() && jerarquia((char)pila.peek()) >= jerarquia(cad: i)){
```

```

30         posfija += pila.peek();
31         pila.pop();
32     }
33     pila.push( item: i);
34 }
35
36     if(i == ' '){
37         while(!pila.isEmpty() && (char)pila.peek() != '('){
38             posfija += pila.pop();
39         }
40     }
41
42 }
43 while(!pila.isEmpty()){
44     pila.pop();
45 }
46 }
47
48 private static boolean operador(char cad){
49     return cad == '^' || cad == '*' || cad == '/' || cad == '+' || cad == '-';
50 }
51
52 public static byte jerarquia(char cad){
53     byte orden = 0;
54     switch(cad){
55         case '^': orden = 3; break;
56         case '/':
57         case '%':
58         case '*': orden = 2; break;
59         case '+':

```

```

60         case '-': orden = 1; break;
61     }
62     return orden;
63 }
64 public void CadenaPos(String cad) {
65
66     Stack pila = new Stack();
67     String posfija = "";
68
69     JFrame ventana = new JFrame();
70     JPanel panel = new JPanel();
71     panel.setLayout(new BorderLayout( target: panel, axis: BorderLayout.Y_AXIS));
72     JScrollPane scroll = new JScrollPane( view: panel);
73
74     String[] titulo = {"Simbolo", "Pila", "Cadena"};
75     DefaultTableModel mod = new DefaultTableModel( data: null, columnNames: titulo);
76     JTable tabla = new JTable( dm: mod);
77     JScrollPane tablaScroll = new JScrollPane( view: tabla);
78     panel.add( comp: tablaScroll);
79
80     for (int i = 0; i < cad.length(); i++) {
81         char c = cad.charAt( index: i);
82         Object[] datos = new Object[3];
83         datos[0] = c;
84         datos[1] = pila.toString();
85
86         if (c == '(') {
87             pila.push( item: c);
88         } else if (c >= 'a' && c <= 'z' || c >= 'A' && c <= 'Z') {
89             posfija += c;

```

```

90     datos[2] = posfija;
91     mod.addRow( row Data: datos);
92 } else if (operador( cod: c)) {
93     while (!pila.isEmpty() && jerarquia((char)pila.peek()) >= jerarquia( cod: c)) {
94         posfija += pila.peek();
95         datos[2] = posfija;
96         mod.addRow( row Data: datos);
97         pila.pop();
98     }
99     pila.push( item: c);
100 } else if (c == ')') {
101     while (!pila.isEmpty() && (char)pila.peek() != '(') {
102         posfija += pila.pop();
103         datos[2] = posfija;
104         mod.addRow( row Data: datos);
105     }
106     if (!pila.isEmpty() && (char)pila.peek() == '(') {
107         pila.pop();
108     }
109 }
110 }
111
112 while (!pila.isEmpty()) {
113     char c = (char)pila.pop();
114     posfija += c;
115     Object[] datos = new Object[3];
116     datos[0] = c;
117     datos[1] = pila.toString();
118     datos[2] = posfija;
119     mod.addRow( row Data: datos);

```

```

120 }
121
122 ventana.setLayout(new BorderLayout());
123 ventana.add( comp: scroll, constraints: BorderLayout.CENTER);
124 ventana.setSize( width: 500, height: 300);
125 ventana.setLocationRelativeTo( c: null);
126 ventana.setDefaultCloseOperation( operation: JFrame.EXIT_ON_CLOSE);
127 ventana.setVisible( b: true);
128 }
129 public static void main(String[] args) {
130     String Cadena = Tools.leeString( msg: "Ingresa una cadena: ");
131     Caracter Tabla = new Caracter();
132     Tabla.CadenaPos( cad: Cadena);
133 }
134

```

### Ejecución

Símbolo	Pila	Cadena
a	[()	a
b	[(, +]	ab
c	[(, +, ()]	abc
d	[(, +, (, -]	abcd
)	[(, +, (, -]	abcd-
)	[(, +]	abcd-+
e	[*, ()]	abcd-+e
f	[*, (, /]	abcd-+ef
g	[*, (, /, ()]	abcd-+efg
h	[*, (, /, (, ^]	abcd-+efgh
)	[*, (, /, (, ^]	abcd-+efgh^
)	[*, (, /]	abcd-+efgh^/
*	[]	abcd-+efgh^/*

### Funcionamiento

El método CadenaPos recibe como parámetro una cadena que representa una expresión aritmética en notación infija y muestra una ventana con una tabla que

muestra el proceso de conversión de la expresión en notación infija a notación posfija.

El método utiliza una pila para almacenar los operadores y paréntesis de la expresión en notación infija. El proceso de conversión se realiza recorriendo la expresión de izquierda a derecha, y en cada paso se realiza una de las siguientes acciones:

- Si el símbolo es un paréntesis de apertura, se agrega a la pila.
- Si el símbolo es una letra, se agrega directamente a la cadena de notación posfija.
- Si el símbolo es un operador, se sacan los operadores de mayor o igual jerarquía de la pila y se agregan a la cadena de notación posfija hasta encontrar un operador de menor jerarquía o un paréntesis de apertura, luego se agrega el operador actual a la pila.
- Si el símbolo es un paréntesis de cierre, se sacan los operadores de la pila y se agregan a la cadena de notación posfija hasta encontrar un paréntesis de apertura, luego se saca el paréntesis de apertura de la pila.

Al finalizar el recorrido de la expresión, se sacan los operadores restantes de la pila y se agregan a la cadena de notación posfija.

La tabla que se muestra en la ventana tiene tres columnas: "Símbolo", "Pila" y "Cadena". En la columna "Símbolo" se muestra el símbolo que se está procesando en ese momento, en la columna "Pila" se muestra el contenido de la pila en ese momento, y en la columna "Cadena" se muestra la cadena de notación posfija en ese momento. La tabla se va actualizando en cada paso del proceso de conversión.

## Clase ColaA

### Código

```
1 package EstuCola;
2 import EntradaSalida.Tools;
3
4 public class ColaA <T> implements ColaTDA <T>{
5
6     private T cola[];
7     private byte u;
8
9     public ColaA(int max){
10         cola = (T[]) (new Object[max]);
11         u = -1;
12     }
13     @Override
14     public boolean isEmptyCola(){
15         return (u == -1);
16     }
17     public boolean isSpace(){
18         return (u < cola.length-1);
19     }
20     @Override
21     public void pushCola(T dato){
22         if(isSpace()){
23             u++;
24             cola[u] = dato;
25         }
26         else{
27             Tools.ImprimeMensaje( msg: "Cola llena.");
28         }
29     }
30 }
```



```

30  public void RecorrePosiciones(){
31      for (int J = 0; J < u; J++) {
32          cola[J] = cola[J+1];
33      }
34  }
35  @Override
36  public T popCola(){
37      T dato = cola[0];
38      RecorrePosiciones();
39      u--;
40      return dato;
41  }
42  @Override
43  public T peekCola(){
44      return cola[0];
45  }
46  public void vaciar(){
47      cola = null;
48      u = -1;
49  }
50  @Override
51  public String toString(){
52      return toString(i: u);
53  }
54  private String toString(int i){
55      return (i >= 0) ? "cola ==>" + i + "[" + cola[i] + "]\n" + toString(i-1): "";
56  }

```

Menú ejecutable:

```

58 public static void MenuColaA(String Menu){
59     ColaA pila = new ColaA( max: 10);
60     int sel;
61     do{
62         sel=Tools.leeEntero( msg: Menu);
63         switch(sel){
64             case 1: pila.pushCola( dato: Tools.leeString( msg: "Escribe un Nombre: "));
65                 Tools.ImprimeMensaje("Datos de la pila: \n" + pila.toString());
66                 break;
67             case 2: if (pila.isEmptyCola()){
68                 Tools.ImprimeMensaje( msg: "Pila Vacía");
69             }
70             else{
71                 Tools.ImprimeMensaje("Dato eliminado de cola: ==>" + pila.popCola() + "\n" + pila.toString());
72             }
73             break;
74             case 3: if (pila.isEmptyCola()) {
75                 Tools.ImprimeMensaje( msg: "Pila Vacía");
76             }
77             else{
78                 Tools.ImprimeMensaje("Dato de la cima de la cola: ==>" + pila.peekCola()+ "\n" + pila.toString());
79             }
80             break;
81             case 4: if (pila.isEmptyCola()) {
82                 Tools.ImprimeMensaje( msg: "Pila vacía");
83             }
84             else{
85                 pila = null;
86                 pila = new ColaA((byte)10);
87             }
88             break;
89             default: Tools.ImprimeMensaje( msg: "Opcion no definida, intenta de nuevo");
90         }
91     }
92     while(sel!=5);
93 }
94 public static void main(String []args){
95     ColaA.MenuColaA("Selecciona una opcion \n"
96         + "1. Push\n"
97         + "2. Pop\n"
98         + "3. Peek\n"
99         + "4. Free\n"
100        + "5. Salir");
101 }
102 }

```

## Ejecución

The image displays a sequence of five GUI dialog boxes illustrating the execution of a simulation. The first dialog is an 'Input' box titled 'Selecciona una opcion' with a list: 1. Push, 2. Pop, 3. Peek, 4. Free, 5. Salir. The input field contains '1'. The second dialog is an 'Input' box titled 'Escribe un Nombre:' with the input field containing 'Karyme'. The third dialog is a 'Message' box titled 'Datos de la pila:' showing 'cola ==>0[Karyme]'. The fourth dialog is a 'Message' box titled 'Datos de la pila:' showing 'cola ==>2[Luis]', 'cola ==>1[Vania]', and 'cola ==>0[Karyme]'. The fifth dialog is an 'Input' box titled 'Selecciona una opcion' with the input field containing '2'. The sixth dialog is a 'Message' box titled 'Dato eliminado de cola: ==>Karyme' showing 'cola ==>1[Luis]' and 'cola ==>0[Vania]'.

**Input**

Selecciona una opcion

1. Push
2. Pop
3. Peek
4. Free
5. Salir

1

OK Cancel

**Input**

Escribe un Nombre:

Karyme

OK Cancel

**Message**

Datos de la pila:

cola ==>0[Karyme]

OK

**Message**

Datos de la pila:

cola ==>2[Luis]

cola ==>1[Vania]

cola ==>0[Karyme]

OK

**Input**

Selecciona una opcion

1. Push
2. Pop
3. Peek
4. Free
5. Salir

2

OK Cancel

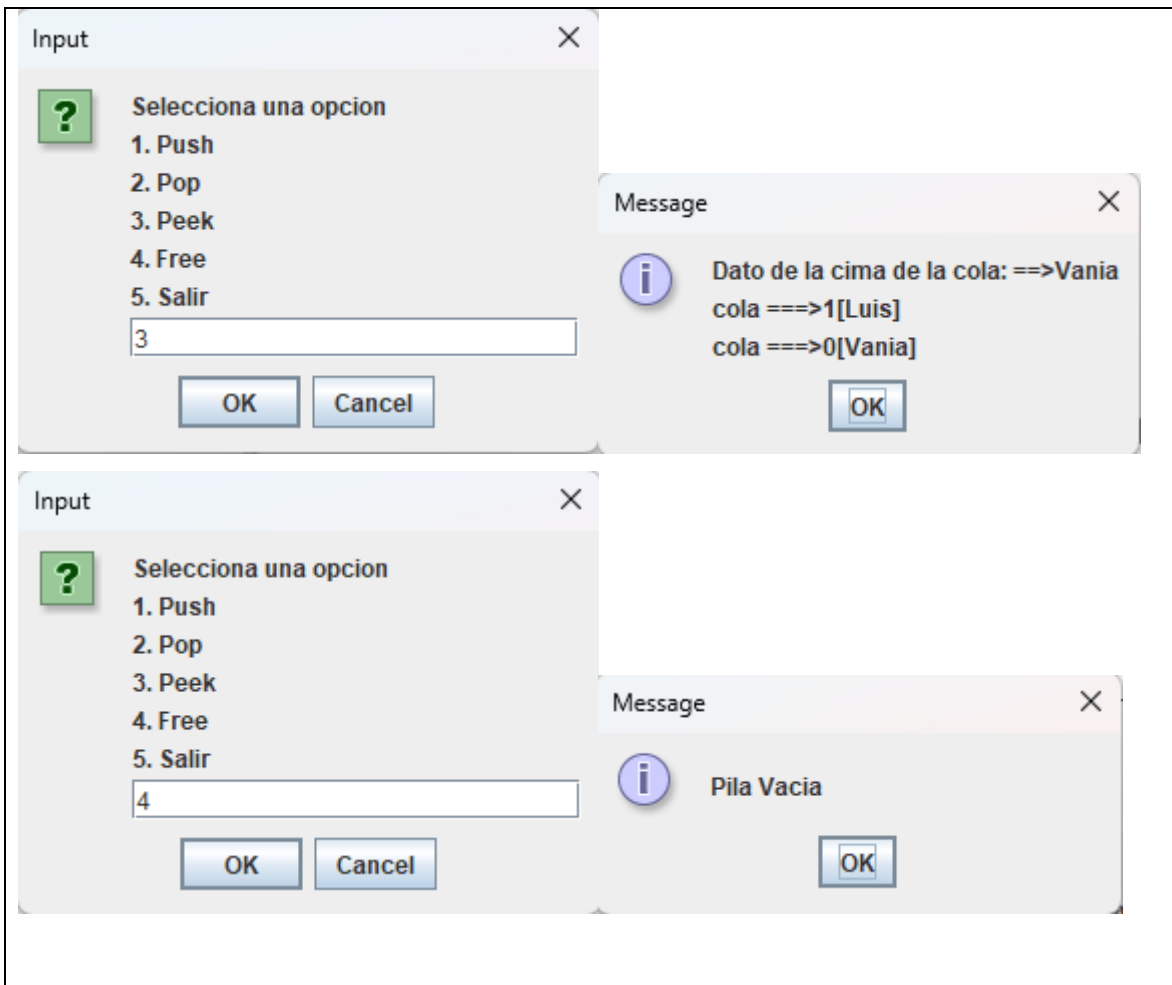
**Message**

Dato eliminado de cola: ==>Karyme

cola ==>1[Luis]

cola ==>0[Vania]

OK



### Funcionamiento

Este código define una clase genérica ColaA que implementa la interfaz InterfazCola y representa una cola (estructura de datos FIFO) mediante un arreglo. Los elementos se agregan al final de la cola y se eliminan del inicio. La clase cuenta con los siguientes métodos:

- isEmptyCola: devuelve true si la cola está vacía.
- isSpace: devuelve true si aún hay espacio en la cola.
- pushCola: agrega un elemento al final de la cola.
- popCola: elimina y devuelve el elemento al inicio de la cola.
- peekCola: devuelve el elemento al inicio de la cola sin eliminarlo. vaciar: elimina todos los elementos de la cola.
- toString: devuelve una representación en cadena de la cola.

## Clase ColaB

### Código

```
1 package EstuCola;
2 import EntradaSalida.Tools;
3 import java.util.Iterator;
4 import java.util.LinkedList;
5 import java.util.Queue;
6
7 public class ColaB <T> implements ColaTDA<T>{
8
9     private Queue cola;
10    public ColaB(){
11        cola = new LinkedList();
12    }
13    public int Size(){
14        return cola.size();
15    }
16    @Override
17    public boolean isEmptyCola(){
18        return(cola.isEmpty());
19    }
20    @Override
21    public T peekCola(){
22        return(T)(cola.element());
23    }
24    public void vaciar(){
25        cola.clear();
26    }
27    @Override
28    public void pushCola(T dato){
29        cola.add( dato);
30    }
```

```

31      @Override
32      public T popCola(){
33          T dato;
34          dato = (T) cola.element();
35          cola.remove();
36          return dato;
37      }
38      @Override
39      public String toString(){
40          String cad = "";
41          byte j = 0;
42          for (Iterator i = cola.iterator(); i.hasNext();){
43              cad += " " + "[" + i.next() + "]" + " " + j;
44              j++;
45          }
46          return cad;
47      }

```

Menú ejecutable:

```

48      public static void MenuColaB(String Menu){
49
50          ColaB cola = new ColaB();
51          int sel;
52          do{
53              sel=Tools.leeEntero( msg: Menu);
54              switch(sel){
55                  case 1: cola.pushCola( dato: Tools.leeString( msg: "Escribe un dato"));
56                      Tools.ImprimeMensaje("Datos de la cola: \n" + cola.toString());
57                      break;
58                  case 2: if (cola.isEmptyCola()){
59                      Tools.ImprimeMensaje( msg: "Cola Vacía");
60                  }
61                  else{
62                      Tools.ImprimeMensaje("Dato eliminado de la cola: ==>" + cola.popCola() + "\n" + cola.toString());
63                  }
64                  break;
65                  case 3: if (cola.isEmptyCola()){
66                      Tools.ImprimeMensaje( msg: "Cola Vacía");
67                  }
68                  else{
69                      Tools.ImprimeMensaje("Dato de la cola: ==>" + cola.peekCola() + "\n" + cola.toString());
70                  }
71                  break;
72                  case 4: if (cola.isEmptyCola()){
73                      Tools.ImprimeMensaje( msg: "Cola vacía");
74                  }
75                  else{
76                      cola = null;
77                      cola = new ColaB();

```

```

78     }
79     break;
80     default: Tools.ImprimeMensaje( msg: "Opcion no definida, intenta de nuevo");
81 }
82 }
83 while(sell=5);
84 }
85 public static void main(String []args){
86     ColaB.MenuColaB("Selecciona una opcion \n"
87         +"1. Push\n"
88         +"2. Pop\n"
89         +"3. Peek\n"
90         +"4. Free\n"
91         +"5. Salir");
92 }
93 }

```

## Ejecución

The execution sequence is as follows:

- Input Dialog:** Titled "Input", it displays a menu "Selecciona una opcion" with options: 1. Push, 2. Pop, 3. Peek, 4. Free, 5. Salir. The user has entered "1". Buttons: OK, Cancel.
- Input Dialog:** Titled "Input", it prompts "Escribe un dato" (Write a data). The user has entered "Luis". Buttons: OK, Cancel.
- Message Dialog:** Titled "Message", it displays "Datos de la cola: [Luis] 0". Button: OK.
- Message Dialog:** Titled "Message", it displays "Datos de la cola: [Luis] 0 [Karyme] 1 [Vania] 2". Button: OK.

## Funcionamiento

Este código implementa una cola genérica utilizando una estructura de datos de cola predeterminada en Java llamada LinkedList. La clase ColaB tiene los siguientes métodos:



- Un constructor que inicializa la cola.
- El método Size() que devuelve el tamaño de la cola.
- El método isEmptyCola() que devuelve si la cola está vacía o no.
- El método peekCola() que devuelve el primer elemento de la cola sin eliminarlo.
- El método vaciar() que elimina todos los elementos de la cola.
- El método pushCola(T dato) que inserta un elemento al final de la cola.
- El método popCola() que devuelve el primer elemento de la cola y lo elimina de la misma.
- El método toString() que devuelve una cadena que representa la cola.

En resumen, el código implementa una cola genérica utilizando la clase LinkedList de Java, proporcionando los métodos necesarios para manipular la cola.

Clase ColaC

Código

```

1 package EstuCola;
2 import EntradaSalida.Tools;
3 import java.util.ArrayList;
4
5 public class ColaC <T> implements ColaTDA <T> {
6
7     private ArrayList cola;
8     byte u;
9
10    public ColaC(){
11        cola = new ArrayList();
12        u = 0;
13    }
14    public int Size(){
15        return cola.size()-1;
16    }
17    @Override
18    public boolean isEmptyCola(){
19        return cola.isEmpty();
20    }
21    public void vacia(){
22        cola.clear();
23    }
24    @Override
25    public void pushCola(Object dato){
26        cola.add( e: dato);
27        u++;
28    }
29    @Override
30    public T popCola(){

```

```

31     T dato = (T) cola.get( index: 0);
32     cola.remove( index: 0);
33     u--;
34     return dato;
35 }
36 @Override
37 public T peekCola(){
38     return (T) cola.get( index: 0);
39 }
40 @Override
41 public String toString(){
42     return toString( i: 0);
43 }
44 private String toString(int i){
45     return (i < u)? "" + i + "[" + cola.get( index: i) + "]" ==> " + toString(i+1):"";
46 }

```

Menú ejecutable:

```

47 public static void MenuColaC(String Menu){
48
49     ColaC cola = new ColaC();
50     int sel;
51     do{
52         sel=Tools.leeEntero( msg: Menu);
53         switch(sel){
54             case 1: cola.pushCola( dato: Tools.leeString( msg: "Escribe un dato entero"));
55                 Tools.ImprimeMensaje("Datos de la cola: \n" + cola.toString());
56                 break;
57             case 2: if (cola.isEmptyCola()){
58                 Tools.ImprimeMensaje( msg: "Cola Vacía");
59             }
60             else{
61                 Tools.ImprimeMensaje("Dato eliminado de la cola: ==>" + cola.popCola() + "\n" + cola.toString());
62             }
63             break;
64             case 3: if (cola.isEmptyCola()) {
65                 Tools.ImprimeMensaje( msg: "Cola Vacía");
66             }
67             else{
68                 Tools.ImprimeMensaje("Dato de la cola: ==>" + cola.peekCola() + "\n" + cola.toString());
69             }
70             break;
71             case 4: if (cola.isEmptyCola()) {
72                 Tools.ImprimeMensaje( msg: "Cola vacía");
73             }
74             else{
75                 cola = null;
76                 cola = new ColaC();

```


```

77     }
78     break;
79     default: Tools.ImprimeMensaje( msg: "Opcion no definida, intenta de nuevo");
80 }
81 }
82 while(sell!=5);
83 }
84 public static void main(String []args){
85     ColaC.MenuColaC("Selecciona una opcion \n"
86         +"1. Push\n"
87         +"2. Pop\n"
88         +"3. Peek\n"
89         +"4. Free\n"
90         +"5. Salir");
91 }
92 }

```

## Ejecución


Input


**Selecciona una opcion**  
1. Push  
2. Pop  
3. Peek  
4. Free  
5. Salir

OK

Cancel


Input


**Escribe un dato entero**

OK


Cancel

Message


**Datos de la cola:**  
0[Vania] ==>

OK

Message


**Datos de la cola:**  
0[Vania] ==>1[Luis] ==>2[Karyme] ==>

OK

Input

?

Selecciona una opcion

1. Push

2. Pop

3. Peek

4. Free

5. Salir

2

OK

Cancel

Message

i

Dato eliminado de la cola: ==>Vania  
0[Luis] ==>1[Karyme] ==>

OK

Input

?

Selecciona una opcion

1. Push

2. Pop

3. Peek

4. Free

5. Salir

3

OK

Cancel

Message

i

Dato de la cola: ==>Luis  
0[Luis] ==>1[Karyme] ==>

OK

Input

?

Selecciona una opcion

1. Push

2. Pop

3. Peek

4. Free

5. Salir

4

OK

Cancel

Message

i

Cola Vacía

OK

Funcionamiento
Este código implementa una cola utilizando una ArrayList. La cola se inicializa vacía con un ArrayList y un índice u igual a 0. Los métodos pushCola y popCola agregan y eliminan elementos en la cola, respectivamente. El método peekCola devuelve el primer elemento de la cola, y el método toString devuelve una cadena que representa la cola completa. Además, el código tiene métodos para verificar si la cola está vacía, vaciar la cola y obtener su tamaño.

## 7. Conclusiones.

En conclusión, en Java, las pilas y colas son estructuras de datos que se implementan utilizando clases predefinidas en el lenguaje. La clase Stack se utiliza para implementar una pila y la clase Queue se utiliza para implementar una cola. Ambas clases ofrecen métodos que permiten agregar, eliminar y acceder a elementos de la estructura de datos, y proporcionan diferentes políticas de acceso a los elementos (LIFO para las pilas y FIFO para las colas). Es importante tener en cuenta que estas clases son solo implementaciones de las estructuras de datos y que se pueden crear estructuras personalizadas utilizando las clases predefinidas en Java, como ArrayList o LinkedList. En general, las pilas y colas son estructuras de datos esenciales en Java y su uso es muy común en programación.

## 8. Bibliografía.

Estructura de Datos JP. (2015, noviembre). Pilas y colas en Java [entrada de blog]. Recuperado el 23 de abril de 2023, de

<http://estructuradedatosjp.blogspot.com/2015/11/pilas-y-colas-en-java.html>

Guía de Pilas y Colas en Java [Solución]. (2012). Universidad Carlos III de Madrid. Recuperado el 23 de abril de 2023, de

[http://www.it.uc3m.es/java/2012-13/units/pilas-colas/guides/4/guide\\_es\\_solution.html](http://www.it.uc3m.es/java/2012-13/units/pilas-colas/guides/4/guide_es_solution.html)

Oportunity, R. (s.f.). Java Pilas y Colas [Diapositivas]. Recuperado el 23 de abril de 2023, de

<https://es.slideshare.net/RoverOportunity2012/java-pilas-y-colas>

Programación Básica Java. (s.f.). Listas, Pilas y Colas [Sitio web]. Recuperado el 23 de abril de 2023, de

<https://sites.google.com/site/programacionbasicajava/listas-pilas-y-colas?pli=1>