6 | Pila con referencias

Meta

Que el alumno domine el manejo de información almacenada en una *Pila*.

Objetivos

Al finalizar la práctica el alumno será capaz de:

- Implementar el tipo de dato abstracto *Pila* utilizando nodos y referencias.

Antecedentes

Una `Pila` es una estructura de datos caracterizada por:

1. El último elemento que entra a la `Pila` es el primer elemento que sale.
2. Tiene un tamaño dinámico.

A continuación se define el tipo de dato abstracto `Pila`.

Definición 6.1: Pila

Una *Pila* es una estructura de datos tal que:

1. Tiene un número variable de elementos de tipo T.
2. Cuando se agrega un elemento, éste se coloca en el tope de la `Pila`.
3. Sólo se puede extraer al elemento en el tope de la `Pila`.

Nombre: `Pila`.
Valores: $\mathbb{N}$, T, con $\mathtt{null} \in$ T.

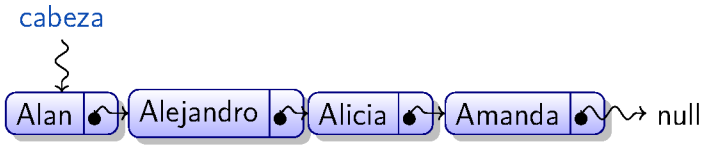

Figura 6.1 Representación en memoria de una pila, utilizando nodos y referencias.

Operaciones: sea `this` la pila sobre la cual se está operando.

Constructores :

Pila(): $\emptyset \to \text{Pila}$

Precondiciones: $\emptyset$

Postcondiciones :

- Una Pila vacía.

Métodos de acceso :

mira(this) $\to$ e: $\text{Pila} \times \mathbb{N} \to \text{T}$

Precondiciones: $\emptyset$

Postcondiciones :

- $e \in \text{T}$, e es el elemento almacenado en el tope de la Pila.

Métodos de manipulación :

expulsa(this) $\to$ e: $\text{Pila} \to \text{T}$

Precondiciones :$\emptyset$

Postcondiciones :

- Elimina y devuelve el elemento e que se encuentra en el tope de la Pila.

empuja(this, e): $\text{Pila} \times \text{T} \xrightarrow{?} \emptyset$

Precondiciones: $\emptyset$

Postcondiciones :

- El elemento e es asignado al tope de la Pila.

Actividad 6.1

Revisa la documentación de la clase Stack de `Java`. ¿Cuáles serían los métodos equivalentes a los definidos aquí? ¿En qué difieren?

Como se ilustra en la Figura 6.1, en esta implementación los datos se guardan dentro de objetos llamados *nodos*. Cada nodo contiene dos piezas de información:

- El dato[1] que guarda y
- la dirección del nodo con el siguiente dato.

Una clase, a la cual nosotros llamaremos `PilaLigada<E>`, tiene un atributo escencial:

[1] O la dirección del dato, si se trata de un objeto.

- La dirección del primer nodo, es decir, del nodo con el último dato que fue agregado a la pila.

Cada vez que se quiera empujar un dato a la pila, se creará un nodo nuevo para guardar ese dato. El nuevo nodo también almacenará la dirección del nodo que solía estar a la *cabeza* de la estructura, y la variable `cabeza` ahora tendrá la dirección de este nuevo nodo. Imaginemos que el nuevo dato acaba de *sumergir* un poco más a los datos anteriores (los empujó más lejos). Esos datos no volverán a ser visibles, hasta que el dato en la cabeza haya sido expulsado.

Para expulsar un dato se realiza el procedimiento inverso: la cabeza volverá a guardar la dirección del nodo siguiente y se devolverá el valor que estaba guardado en el nodo *de hasta arriba*, a la vez que se descarta el nodo que contenía al dato. Cuidado: al realizar estas operaciones en código es importante cuidar el orden en que se realizan, para no perder datos o direcciones en el proceso. A menudo requerirás del uso de variables temporales, para almacenar un dato que usarás depués. Pero recuerda: las variables temporales deben desaparecer cuando se termina la ejecución de un método, es decir, deben ser variables locales. Asegúrate de guardar todo lo que deba permanecer en la pila en atributos de objetos, ya sea en la `PilaLigada<E>` o algún `Nodo` adecuado.

Desarrollo

Se implementará el TDA `Pila` utilizando nodos y referencias. Para esto se deberá implementar la intefaz `IPila<E>` y extender `ColeccionAbstracta<E>`. Asegúrate de que tu implementación cumpla con las condiciones indicadas en la documentación de la intefaz. Por razones didácticas, no se permite el uso de ninguna clase que se encuentre en el api de Java, por ejemplo clases como `Vector<E>`, `LinkedList<E>` o cualquiera otra estructura del paquete `java.util`.

1. Programa la clase `Nodo`.

   Puedes crear esta clase dentro del paquete `ed.estructuras.lineales`. Esto te permitirá reutilizarlo cuando programes la siguiente estructura: la cola. Si eliges esta opción, dale acceso de paquete (es decir, la declaración de la clase omite el acceso e inicia con `class Nodo<E>...` en lugar de `public class Nodo...`). Esto es para no confundir este nodo con otros nodos que utilizarán futuras estructuras y que tienen características diferentes. Otra opción es programarlo como una clase estática interna de `PilaLigada<E>`, pero en ese caso, sólo la pila podrá usarlo.

2. Programa la clase `PilaLigada<E>`.

   - Implementa la interfaz `IPila<E>` y
   - extiende `ColeccionAbstracta<E>`.

Inicia con los métodos básicos.

3. Luego agrega el iterador que requiere `Collection<E>`. Puedes programar al iterador como una clase interna, en este caso debes implementar la interfaz `Iterator<E>` pero no es necesario que tu clase declare una nueva variable de tipo, la `<E>`. Esto se vería así:

```
import java.util.Iterator;
...
public class PilaLigada<E> ... {
  private class Iterador implements Iterator<E> {
    ...
  }
}
```

Aunque en una pila sólo se pueden agregar y remover elementos en un extremo, necesitaremos un iterador que permitar ver todos los elementos en la pila, desde el último insertado hasta el primero. Para esta práctica sólo programarás un constructor, que inicialice el iterador en el tope de la pila, y los métodos `next` y `hasNext` del iterador.

Preguntas

1. Explica, para esta implementación, cómo funciona el método `empuja`.
2. ¿Cuál es la complejidad, en el peor caso, de los métodos `mira`, `expulsa` y `empuja`?