Ejercicios aplicados de pilas

Ejercicio 1

El JAVA, para una pila construida a partir de un arreglo, el siguiente es el código:

```java
class Pila {
    private Object[] arreglo_pila;
    private int n = 0;

    public Pila(int totaln) {
        arreglo_pila = new Object[totaln];
    }

    public void Apilar(Object item) {
        if (n == arreglo_pila.length) {
            System.out.println("Pila llena, no se pueden agregar mas datos");
        }
        else{
            arreglo_pila[n++] = item;
        }
    }

    public void Borrar() {
        if (n == 0) {
            System.out.println("Pila vacia, no se pueden borrar datos");
        }
        else
        {
            arreglo_pila[--n] = null;
        }
    }

    public Object Ver_dato() {
        if (n == 0) {
            System.out.println("Pila vacia, no se puede mostrar dato");
            return null;
        }
        else
        {
            return arreglo_pila[n - 1];
        }
    }


    public int mostrar_N() {
        return n;
    }
}
```

Figura 1.

En el código de la Figura 1, se ha creado una nueva clase, es decir, un molde para crear nuevos objetos. En las líneas dos y tres se definen dos atributos para la pila: un arreglo de

objetos (los datos que puede recibir son enteros, caracteres entre otros.) y el tamaño o total de elementos (n). La variable n, indica además, cual es la siguiente posición donde se guardará un dato.

En la línea 5, se define el constructor, o método que inicializa cada nuevo objeto de la clase, recibiendo un parámetro que representa el total de elementos para la pila; en el interior del constructor, se define el tamaño para el arreglo de datos.

En la línea 9, se crea un nuevo método, que no retorna nada (*void*) y tiene la finalidad de agregar nuevos elementos a la pila (apilar), siempre y cuando, el total de elementos existentes en la pila (*arreglo_pila.length*), no supere el tamaño del arreglo, en cuyo caso muestra un mensaje informativo indicando que no puede agregar el dato; de lo contrario, lo guarda en la posición disponible e incrementando el índice del arreglo (n++).

En la línea 18, se crea un nuevo método, que no retorna nada y tiene la finalidad de borrar un elemento de la pila (borrar), siempre y cuando, la pila no se encuentre vacía. Al borrar el dato previamente, se realiza el decremento del índice de posiciones (—n).

En la línea 28, se crea un nuevo método para ver el dato actual, es decir, el último. Para esta finalidad se muestra la posición con el índice n disminuido en uno ($n - 1$).

El total de elementos registrados en la lista se muestra mediante el método *mostrar_N* en la línea 40.

La implementación o uso de la clase puede ser la siguiente:

```
1  public class MyClass {
2      public static void main(String args[]) {
3          Pila pila1=new Pila(5);
4          pila1.Apilar(10);
5          pila1.Apilar(11);
6          pila1.Apilar(50);
7          pila1.Apilar(40);
8          pila1.Borrar();
9          System.out.println(pila1.Ver_dato());
10         System.out.println(pila1.mostrar_N());
11     }
12 }
```

```
Result...
compiled and executed in 1.046 sec(s)
50
3
```

Figura 2.

En la Figura 4, se crea una nueva pila, llamada pila1 (línea 3). Se agregan datos a la pila invocando el método *Apilar* con el respectivo dato. Hasta la línea 7, existen en total 4 elementos en la lista y el último es 40. La línea 8 elimina este último dato y al invocar el

método *Ver_dato* se muestra 50, ya que efectivamente, el número 40 fue eliminado de la lista. Finalmente, se muestra el total de datos en la lista: 3.

Ejercicio 2

El código en JAVA para una pila construida a partir de nodos es el siguiente:

```
class Pila
    {
        class Nodo
        {
            public int info;
            public Nodo sig;
        }

        private Nodo inicio;

        public Pila()
        {
            inicio = null;
        }

        public void Apilar(int x)
        {
            Nodo nuevo;
            nuevo = new Nodo();
            nuevo.info = x;
            if (inicio == null)
            {
                nuevo.sig = null;
            }
            else
            {
                nuevo.sig = inicio;
            }
            inicio = nuevo;
        }

        public void Borrar()
        {
            if (inicio != null)
            {
                inicio = inicio.sig;
            }
        }

        public void Imprimir()
        {
            Nodo reco=inicio;
            System.out.println("Pila de datos.");
            while (reco!=null)
            {
                System.out.println(reco.info);
                reco=reco.sig;
            }
            System.out.println();
        }

    }
```

Figura 3.

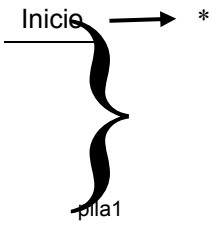
En la Figura 3, para la clase *Pila,* para declarar el nodo, se crea una nueva clase: *nodo*, con las propiedades *info* (información del nodo como valor entero) y referencia al próximo *nodo* llamado *sig*.

En el constructor de la clase (línea 11), se inicializa el puntero llamado inicio que permitirá acceder a los demás nodos. En primera instancia se asigna el valor de *null* (nulo) indicando que la pila está vacía.

Dicho constructor se invoca en el programa principal por medio de la instrucción:

Pila pila1 = new Pila();

Dando como resultado en memoria interna el siguiente esquema:



En el momento de agregar un nuevo elemento, se invoca el método *Apilar:*

pila1.Apilar(10);

El número 10 es asignado al parámetro de entrada *x* del método *Apilar*, y se crea el nuevo *nodo*, entregando 10 a la *info* del *nodo* y asignando el siguiente *nodo* nulo:

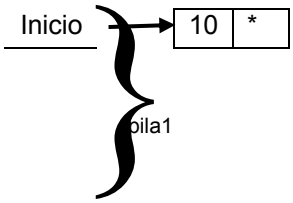
10	*

```
18          Nodo nuevo;
19          nuevo = new Nodo();
20          nuevo.info = x;
21          if (inicio == null)
22          {
23              nuevo.sig = null;
24          }
```

Líneas responsables de crear el nodo

El nodo se integra a la pila por medio de la línea 29:

```
29          inicio = nuevo;
```



Los siguientes nodos se enlazarán con la línea 27, ya que no está vacía la lista. Se actualizan los valores de inicio y apuntador al siguiente nodo.

Para borrar un nodo, se invoca el método definido en la línea 32. Se actualiza el apuntador de la pila (inicio) de manera que se dirija el siguiente nodo disponible. El último, al no tener una referencia hacia él, es eliminado por el sistema.

Para mostrar todos los datos de la pila, el método *Imprimir* crea un nodo que se posiciona sobre los nodos actuales hasta llegar a nulo.

El lenguaje de programación JAVA, al igual que muchos otros, incluye una clase interna que implementa y facilita el manejo de pilas. Dicha clase es *Stack*[1].

La Figura 4 presenta la implementación de dicha clase.

```java
import java.util.Stack;

public class MyClass {
    public static void main(String args[]) {
        Stack<String> personal = new Stack<>();
        personal.push("Rene");
        personal.push("Juan");
        personal.push("Carlos");
        personal.push("Angelina");
        System.out.println("Pila => " + personal);
        personal.pop();
        System.out.println("Pila => " + personal);
        String Ultimo = personal.peek();
        System.out.println("Dato dispobible => " + Ultimo);

    }
}
```

```
Result...
compiled and executed in 1.023 sec(s)
Pila => [Rene, Juan, Carlos, Angelina]
Pila => [Rene, Juan, Carlos]
Dato dispobible => Carlos
```

Figura 4.

Los métodos *push*, *pop* y *peek*, permiten apilar, borrar y mostrar el último dato respectivamente.

El siguiente video muestra el manejo de esta clase: https://youtu.be/D1X0VJESzE0

[1] https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html