Práctica 7

Pilas

Matú Hernández Diana

Rojo Mata Daniel

Justificación de los algoritmos

- **Método constructor.**

```
1       public Pila() {
2         cima = null;
3         longitud = 0;
4       }
```

El algoritmo inicializa la estructura de datos Pila, en este caso, sin ningún elemento. El algoritmo es correcto porque cumple el propósito de crear una pila donde la cima es *null* y su longitud, por ende, debe ser 0.

- **Método apilar**

```
1       public void apilar( T elemento ) {
2         NodoPila nuevoNodo = new NodoPila(elemento);
3         nuevoNodo.siguiente = cima;
4         cima = nuevoNodo;
5         longitud ++;
6       }
```

Agregar un elemento en la cima de la pila implica asignar un espacio de memoria para el elemento a apilar, en este caso, el nuevo elemento se denomina como *nodoNuevo* (línea 2). Se

ajusta el nodo siguiente de *nodoNuevo* como la cima de la pila, línea 3, mientras que la cima se actualiza para que sea *nodoNuevo*. Finalmente la longitud aumenta en una unidad.

El algoritmo es correcto porque añade un elemento en la cima de la pila sin alterar los elementos ya existentes. La pila sigue manteniendo su estructura *LIFO (Last In, First Out)* después de la operación, pues la cima anterior se actualiza.

- **Método desapilar**

```
1       public void desapilar() throws Exception {
2         if (this.cima == null){
3           throw new Exception("La pila esta vacia");
4         }
5         else {
6           NodoPila actual = this.cima;
7           this.cima=cima.siguiente;
8           actual.siguiente = null;
9           actual = null;
10          longitud --;
11        }
12      }
```

El bloque if $2-4$ revisa el caso en que la cima de la pila sea *null*, esto es, la pila es vacia. En tal caso, se arroja una excepción.

El bloque *else* $5-11$ revisa la situación en que la pila no es vacia.

Cuando lo anterior ocurre, la cima se 'mueve' al siguiente nodo en la pila, línea 7 ($this.cima = cima.siguiente;$). Esto asegura que la cima de la pila ahora apunte al elemento que estaba debajo del elemento desapilado. El código establece las referencias $actual.siguiente = null;$ (línea 8) para asegurarse de que el nodo desapilado ya no esté conectado a ningún otro nodo. Luego, se asigna $actual = null;$ (línea 9) para liberar la memoria ocupada por el nodo desapilado. Finalmente la longitud de la pila disminuye en una unidad.

El algoritmo es correcto porque realiza las operaciones de desapilado de manera correcta además de que gestiona la memoria de manera precisa al modificar como *null* las referencias en 8 y 9. Además proporciona mensajes de error significativos para el caso en que se introduzca una pila vacía. Así pues, cumple con las reglas fundamentales de las pilas y asegura que la estructura de datos se maneje de forma coherente y segura en todas las situaciones posibles.

- **Método darElementoCima**

```
1       public T darElementoCima() throws Exception {
2           if (this.esVacia()) {
3             throw new Exception("La pila está vacía");
4           }
5           T aux = cima.elemento;
6           return aux;
7       }
```

El bloque $if\ 2-4$ revisa el caso en que la cima de la pila sea *null*, esto es, la pila es vacía. En tal caso, se arroja una excepción.
Se hizo uso de una variable auxiliar de tipo T que almacena el elemento de la cima, dicha variable es llama *aux*. El método se encarga de regresar lo almacenado en esta variable cuando la pila no es vacía.
El método es correcto pues se encarga de revisar las situaciones en donde la pila es vacía o no, y regresa de manera correcta (al acceder al campo) el elemento correspondiente.

- **Método esVacia**

```
1       public boolean esVacia() {
2           if (this.longitud == 0) {
3               return true;
4           }
```

```
5           else {
6               return false;
7           }
8       }
```

El bloque de código $if\ 2-4$ revisa el caso en que la longitud de la pila sea cero, lo que equivale a decir que la pila es vacia. En este caso se retorna el booleano *true* y en su defecto, cuando la longitud de la pila es distinta de cero, se retorna *false* pues indica que la pila no es vacía.
El algoritmo es correcto pues valida los casos que se requieren accediendo de manera adecuada a los campos de la EDD y retornando los valores booleanos adecuados.

- **Método darLongitud**

```
1       public int darLongitud() {
2           return this.longitud;
3       }
```

La línea 2 regresa el campo *longitud* accediendo a éste mediante el operador $\cdot$.
El algoritmo es correcto pues solo se está accediendo a un campo de la EDD.

3

Complejidad en tiempo y espacio

- **Método constructor.**

```
1       public Pila() {
2         cima = null;  // 1
3         longitud = 0;  //1
4       }
```

$$T(n) = 2$$

$$M(n) = 0$$

- **Método apilar.**

```
1       public void apilar( T elemento ) {
2         NodoPila nuevoNodo = new NodoPila(elemento);  //1
3         nuevoNodo.siguiente = cima;  //1
4         cima = nuevoNodo;  //1
5         longitud ++;  //2
6       }
```

$$T(n) = 5$$

$$M(n) = 1$$

- **Método desapilar.**

```
1       public void desapilar() throws Exception {
2         if (this.cima == null){  //2
3           throw new Exception("La pila esta vacia");
4         }
5         else {
```

```
6           NodoPila actual = this.cima;  //1
7           this.cima=cima.siguiente;  //1
8           actual.siguiente = null;  //1
9           actual = null;  //1
10          longitud --;  //2
11        }
12      }
```

$$T(n) = 8$$

$$M(n) = 0$$

- **Método darElementoCima**

```
1       public T darElementoCima() throws Exception {
2           if (this.esVacia()) {
3             throw new Exception("La pila está vacía");
4           }
5           T aux = cima.elemento;  //1
6           return aux;
7       }
```

$$T(n) = 1$$

$$M(n) = 0$$

- **Método esVacia**

```
1       public boolean esVacia() {
2           if (this.longitud == 0) {  //2
3               return true;
4           }
```

```
5           else {
6               return false;
7           }
8       }
```

$$T(n) = 2$$

$$M(n) = 0$$

- **Método darLongitud.**

```
1       public int darLongitud() {
2           return this.longitud;
3       }
```

$$T(n) = 0$$

$$M(n) = 0$$

Observación

Se puede observar que en todos los métodos $T(n) = a$ y $M(n) = b$, donde $a, b \in \mathbb{N}$, esto sugiere que las funciones pertenecen a $O(1)$, es decir, las funciones anteriores son constantes en tiempo y en almacenamiento.

Pseudocódigos

- **Método Constructor**
- **Entrada:** Ninguna.
- **Salida:** Objeto de tipo Pila.

```
1    function Pila()
2        cima <- null
3        longitud <- 0
4    end function
```

- **Método apilar**
- **Entrada:** Elemento de tipo T que se va a apilar en la pila.
- **Salida:** Ninguna.

```
1    function apilar(P, elemento)
2        nuevoNodo <- *NodoPila(elemento)
3        nuevoNodo.siguiente <- cima
4        cima <- nuevoNodo
5        P.longitud <- P.longitud + 1
6    end function
```

- **Método desapilar**
- **Entrada:** Pila P
- **Salida:** La pila P sin su cima

```
1     function desapilar(P)
2         if(P.cima = null)
3             ERROR "Pila vacia"
4         else
5             actual <- P.cima
6             cima <- P.cima.siguiente
7             actual.siguiente <- null
8             actual <- null
9             P.longitud <- P.longitud - 1
10        end if-else
11    end function
```

- **Método darElementoCima.**
- **Entrada:** Ninguna.
- **Salida:** Elemento de tipo T en la cima de la pila.
- **Excepción:** Lanza una excepción si la pila está vacía.

```
1     function darElementoCima(P)
2         if (P.longitud = 0)
3             entonces ERROR "Pila vacia"
4         end if
5         aux <- cima.elemento
6         retornar aux
7     end function
```

- **Método esVacia**
- **Entrada:** Ninguna.
- **Salida:** `true` si la pila está vacía, `false` en caso contrario.

```
1       function esVacia(P)
2           if (P.longitud = 0)
3               entonces true
4           else
5             entonces false
6           end if-else
7       end function
```

- **Método darLongitud**
- **Entrada:** Pila P
- **Salida:** Numero entero

```
1       function darLongitud(P)
2           return P.longitud
3       end function
```