# Implementación de una Pila y Cola Dinámicas

Elaborado Por: Arnaldo Quintero Segura.

## Decisiones de Diseño

Para la implementación de los Tipos Abstractos de Datos (TAD) Cola (Queue) y Pila (Stack), se procederá a ver cada una de las operaciones que soportan estas estructuras. Así como las decisiones de diseño que han sido tomadas al momento de desarrollo.

### Node

Esta clase posee dos atributos: El valor del nodo, de tipo `Object`, y el apuntador al Nodo siguiente. Estos atributos de la clase son: `value` y `next` respectivamente. Haciendo referencia a las palabras en inglés para valor y siguiente. Estos dos atributos se encuentran encapsulados dentro de la clase, donde cada uno posee correspondientemente su `getter` y su `setter`.

Esta clase posee un único inicializador, que dado un parámetro de tipo `Object`, inicializa un nodo con el valor introducido. Al cual luego se podrá acceder mediante su `getter`: `getValue()`. Al igual que el nodo siguiente, que se encuentran inicialmente con un valor `null`.

### Propiedades de la Cola

Esta clase de Cola es llamada `Queue`, haciendo referencia a las palabras en inglés para Cola, posee tres atributos: La cantidad elementos presentes en la cola, un apuntador al comienzo de la cola, y un apuntador al final de la cola. Siendo cada uno de estos atributos `count`, `front`, y `rear` correspondientemente, haciendo referencia a las palabras en inglés para cantidad, frente, y posterior.

Los dos atributos `front` y `rear` de la clase son privados, dado que el usuario final no debe tener contacto con estos mismos. Si no que debe utilizar los métodos para interactuar con el TAD correspondiente.

El atributo `count` posee un `getter`: `getCount()`, pero será tratado posteriormente. Listemos ahora cada una de las operaciones que soporta la clase `Queue`.

#### Inicializar

Para inicializar una instancia de `Queue`, se utiliza el constructor de la clase. El cual inicializa la cantidad de elementos presentes en la cola a cero. Los punteros `front` y `rear` se mantienen en `null` dado que no hay nada dentro de la cola.

#### Insertar

Dado que se trata de un TAD Cola, la inserción siempre es hecha al final de la cola. Este es el motivo por el cual siempre se tiene el apuntador al final de la cola. Para que a momento de inserción, el proceso sea realizado más rápidamente. Para insertar en la Cola, se utiliza el método `enqueue()`, que recibe como parámetro un elemento de tipo `Object` a ser insertado en la cola.

Al momento de insertar un nuevo elemento, en caso de que haya sido exitoso, el método devuelve el valor booleano `true`, y en caso de que algo haya salido mal, devuelve el valor `false`. Para dar certeza así al usuario de cómo ha ido la operación realizada sobre la cola.

**Extraer**

Como se trata de una cola, la extracción de un elemento se realiza por el frente, extrayendo siempre el elemento más antiguo en la estructura. Para hacer esto, se utiliza el método `dequeue()`, que no recibe ningún parámetro, pero devuelve el valor desencolado correspondiente. Si la cola se encuentra vacía este método retorna el valor `null`.

**Contar**

Para contar la cantidad de elementos en la cola, se utiliza el `getter` del atributo `count`: `getCount()`, el cual nos devuelve la cantidad de elementos que posee la cola. Si la cola esta vacía, devuelve el valor cero. De cualquier otra manera, devolverá la cantidad de elementos presentes en la Cola.

**Inspeccionar**

Este método nos permite ver qué valor se encuentra actualmente en el frente de la cola, sin extraerlo. Para realizar esta función se utiliza el método `peek()`, que no requiere ningún parámetro. Y al igual que `dequeue()` en caso de estar la cola vacía, devuelve el valor `null`, en cualquier otro caso el valor de tipo `Object` que se encuentre en el frente de la cola.

**Buscar**

Este método nos permite saber si en la cola se encuentra un elemento en concreto. Para esto se utiliza el método `search()`, que recibe un parámetro de tipo `Object`, siendo este el elemento a buscar en la cola. De encontrarse el elemento en la cola, el método retornará la posición en la que se encuentra dicho elemento. Siendo el cero el frente de la cola y contando en positivo a medida que se acerca al posterior de la cola.

En caso de que el elemento no exista en la cola, el método devuelve el valor `-1`. Como referencia de una posición invalida.

Se escogió retornar un entero como resultado a esta operación, con posición invalida de valor `-1`, dado que ese es el estándar para el TAD cola que se maneja dentro de Java.

### Mostrar

Para imprimir los valores de la cola, tenemos una función `print` que ya nos imprime a la salida estándar, `stdOut`, el contenido de la cola, dentro de corchetes, los elementos separados por coma, cada uno dentro de comillas. Siendo el frente de la cola el elemento más a la izquierda en la representación, y el posterior el elemento más a la derecha.

En caso de que la cola se encuentre vacía, se imprime el mensaje `Empty Queue`, que traduce del inglés a cola vacía.

Este método utiliza la sobreescritura del método `toString` de Java. Haciendo por lo tanto posible el uso de los métodos normales de Java para imprimir valores por consola, tales como `println` u otros.

## Propiedades de la Pila

Esta clase de Pila es llamada `Stack`, haciendo referencia a las palabras en inglés para Pila, posee dos atributos: La cantidad elementos presentes en la pila y un apuntador al comienzo de la pila. Siendo cada uno de estos atributos `count` y `top` correspondientemente, haciendo referencia a las palabras en inglés para cantidad y tope.

El atributo `top` de la clase es privados, dado que el usuario final no debe tener contacto con el mismo. Si no que debe utilizar los métodos para interactuar con el TAD correspondiente.

El atributo `count` posee un `getter`: `getCount()`, pero será tratado posteriormente. Listemos ahora cada una de las operaciones que soporta la clase `Stack`.

### Inicializar

Para inicializar una instancia de `Stack`, se utiliza el constructor de la clase. El cual inicializa la cantidad de elementos presentes en la pila a cero. El puntero `top` se mantienen en `null` dado que no hay nada dentro de la pila.

### Insertar

Dado que se trata de un TAD Pila, la inserción siempre es hecha al comienzo de la pila. Este es el motivo por el sólo se tiene un apuntador al comienzo de la pila. Para insertar en la Pila, se utiliza el método `push()`, que recibe como parámetro un elemento de tipo `Object` a ser insertado en la pila.

Al momento de insertar un nuevo elemento, en caso de que haya sido exitoso, el método devuelve el valor booleano `true`, y en caso de que algo haya salido mal, devuelve el valor `false`. Para dar certeza así al usuario de cómo ha ido la operación realizada sobre la pila.

### Extraer

Como se trata de una pila, la extracción de un elemento se realiza por el frente, extrayendo siempre el elemento más nuevo en la estructura. Para hacer esto, se utiliza el método `pop()`, que no recibe ningún parámetro, pero devuelve el valor desapilado correspondiente. Si la pila se encuentra vacía este método retorna el valor `null`.

### Contar

Para contar la cantidad de elementos en la pila, se utiliza el `getter` del atributo `count`: `getCount()`, el cual nos devuelve la cantidad de elementos que posee la pila. Si la pila esta vacía, devuelve el valor cero. De cualquier otra manera, devolverá la cantidad de elementos presentes en la Pila.

### Inspeccionar

Este método nos permite ver qué valor se encuentra actualmente en el frente de la pila, sin extraerlo. Para realizar esta función se utiliza el método `peek()`, que no requiere ningún parámetro. Y al igual que `pop()` en caso de estar la pila vacía, devuelve el valor `null`, en cualquier otro caso el valor de tipo `Object` que se encuentre en el frente de la pila.

### Buscar

Este método nos permite saber si en la pila se encuentra un elemento en concreto. Para esto se utiliza el método `search()`, que recibe un parámetro de tipo `Object`, siendo este el elemento a buscar en la pila. De encontrarse el elemento en la pila, el método retornará la posición en la que se encuentra dicho elemento. Siendo el cero el tope de la pila y contando en positivo a medida que se va adentrando en la pila.

En caso de que el elemento no exista en la pila, el método devuelve el valor `-1`. Como referencia de una posición invalida.

Se escogió retornar un entero como resultado a esta operación, con posición invalida de valor `-1`, dado que ese es el estándar para el TAD pila que se maneja dentro de Java.

### Mostrar

Para imprimir los valores de la pila, tenemos una función `print` que ya nos imprime a la salida estándar, `stdOut`, el contenido de la pila, dentro de corchetes, los elementos separados por coma, cada uno dentro de comillas. Siendo el frente de la pila el elemento más a la izquierda en la representación, y el posterior el elemento más a la derecha.

En caso de que la pila se encuentre vacía, se imprime el mensaje `Empty Stack`, que traduce del inglés a pila vacía.

Este método utiliza la sobreescritura del método `toString` de Java. Haciendo por lo tanto posible el uso de los métodos normales de Java para imprimir valores por consola, tales como `println` u otros.

## Prueba de funcionalidad

Como prueba de funcionamiento de estas clases, se encuentra el fichero `Examples.java`, el cual nos muestra paso a paso imprimiendo en consola cómo se crean estas estructuras, se añaden elementos, se revisa lo que hay en el tope, se remueven elementos, se buscan elementos validos e inválidos. Y se van imprimiendo los distintos valores que va tomando la pila por consola.

Se incluye además una serie de pruebas, tanto para la clase `Node` como para las clases `Queue` y `Stack`, en donde se prueba el funcionamiento de cada uno de los métodos públicos que posee la clase.

Dichas pruebas deben ser corridas utilizando la librería JUnit 5.