Sesión 24: Pilas y Colas

Programación 2

Ángel Herranz

Abril 2019

Universidad Politécnica de Madrid

En capítulos anteriores

- 👍 Tema 1: Clases y Objetos
- 👍 Tema 2: Colecciones acotadas de Objetos
- 👍 Tema 4: Tipos Abstractos de Datos
- 👍 Tema 3: Programación Modular
- 👍 Tema 5: Herencia y Polimorfismo
- 👍 Tema 6: Excepciones
- 🕒 Tema 7: Implementación de TADs lineales
  - *A long time ago...* `Nodo<T>`
  - *Arrays* redimensionables
  - Cadenas *simplemente* enlazadas

En el capítulo de hoy

- 👍 Cadenas enlazadas
- 👍 *Stack*
  - *Puntero* al primero
- 👍 *Queue*
  - *Puntero* al primero
  - *Puntero* al primero y al último
  - Cadena enlazada circular

Cadenas enlazadas

Insertar `s` al final de `l`

```
Nodo<String> nuevo = new Nodo<String>(s);
if (l == null) {
  l = nuevo;
}
else {
  Nodo<String> aux = l;
  while (aux.siguiente != null) {
    aux = aux.siguiente;
  }
  aux.siguiente = nuevo;
}
```

Cambiar la posición `i` de `l` por `s`

```
Nodo<String> aux = l;
for (int j = 0; j < i; j++) {
  aux = aux.siguiente;
}
aux.dato = s;
```

Insertar `s` en la posición `i` de `l`

```
Nodo<String> nuevo = new Nodo<String>(s);
if (i == 0) {
  nuevo.siguiente = l;
  l = nuevo;
}
else {
  Nodo<String> aux = l;
  for (int j = 0; j < i - 1; j++) {
    aux = aux.siguiente;
  }
  nuevo.siguiente = aux.siguiente;
  aux.siguiente = nuevo;
}
```

Comprobar si `s` está en `l`[1]

```
Nodo<String> aux = l;
encontrado = false;
while (aux != null && !encontrado) {
  encontrado = s.equals(aux.dato);
  aux = aux.siguiente;
}
```

[1] usar la variable `encontrado`

Borrar el primero de `l`

```
l = l.siguiente;
```

Borrar el último de `l`

```
if (l.siguiente == null) {
  l = l.siguiente;
}
else {
  Nodo<String> aux = l;
  while (aux.siguiente.siguiente != null) {
    aux = aux.siguiente;
  }
  aux.siguiente = null;
}
```

Borrar el dato en la posición `i` de `l`

Borrar el dato `s` de `l`

Insertar `s` en orden en `l`

Stack

API de la clase **Stack<T>**

- `Stack()`
- **`boolean`** `isEmpty ()`
- **`void`** `push (T elemento)`
- `T peek ()` **`throws`** `EmptyStackException`
- `T pop()` **`throws`** `EmptyStackException`

Programar el TAD `Stack<T>`

```
public class Stack<T> {
  private Node<T> top;

  public Stack() {
    top = null;
  }

  // TODO: implement the whole API
}
```

Queue

interface QueueInterface<T>

- `boolean isEmpty ();`
- `void add (T elemento)`
- `T peek () throws EmptyQueueException`
- `T poll() throws EmptyStackException`

Programar SimpleQueue<T>

```
public class SimpleQueue<T>
  implements QueueInterface<E>
{
  private Node<T> last=null;

  public SimpleQueue() {
  }

  // TODO: implement the whole API
}
```

Programar FirstLastQueue<T>

```
public class FirstLastQueue<T>
  implements QueueInterface<E>
{
  private Node<T> first=null;
  private Node<T> last=null;

  public FirstLastQueue() {
  }

  // TODO: implement the whole API
}
```

Programar `CircularQueue<T>`

```
public class CircularQueue<T>
  implements QueueInterface<E>
{
  private Node<T> first=null;

  public CircularQueue() {
  }

  // TODO: implement the whole API
}
```