**TAD stack data structure**

| **TAD** MyStack |
| --- |
| MyStack = [Element<T>] |
| {**inv:** you can just add an Element on the top (LIFO → last-in-first-out)} |
| Primitive Operations:   | **Methods** | **Operation type** | **Input** | **Output** | | --- | --- | --- | --- | | MyStack | Constructor |  | → MyStack | | push | Modifier | Element | → MyStack | | peek | Modifier |  | → MyStack | | pop | Modifier |  | → MyStack | | isEmpty | Analyzer |  | → Boolean | | toString | Constructor |  | → String | |

| **MyStak():** |
| --- |
| \*Create an empty Stack\* |
| {pre:} |
| {post: MyStack was created} |

| **push(Element):** |
| --- |
| \*Add an element on the top of MyStack\* |
| {pre: MyStack must to exist}  {pre: } |
| {post:A new element was added to MyStack} |

| **peek():** |
| --- |
| \*Return the top of the MyStack without deleting it\* |
| {pre: MyStack must to exist}  {pre: the top must be different from null} |
| {post: The top of MyStack was returned without deleting it} |

| **pop():** |
| --- |
| \*Return and delete the top element of MyStack\* |
| {pre:MyStack must to exist}  {pre: the top must be different from null} |
| {post:The top of MyStack was returned and deleted} |

| **isEmpty - EstaVacia ()** |
| --- |
| \*Verify if the stack has elements or not.\* |
| {pre: MyStack must to exist} |
| {post: True if MyStack == null  False otherwise } |

| **toString():** |
| --- |
| \*Print the elements of MyStack\* |
| {pre: MyStack must to exist}  {pre: MyStack can’t be empty} |
| {post:The elements of MyStack were printed} |

**TAD stack data structure**

| **TAD** MyLinkedList |
| --- |
| MyLinkedList={Node=<Node>} |
| {**inv**: MyLinkedList.Node.next >= MyLinkedList.Node.prev ⇔ (only if) MyLinkedList.Node.prev != null } |
| Primitive operations:  Node = Element of MyLinkedList  MyLinkedList = M.L.L   | **Métodos** | **Tipo Operación** | **Entradas** | **Salida** | | --- | --- | --- | --- | | createNode | Constructor |  | →Node | | insertNode | Modifier | Node x Node x Node |  | | isEmpty | Analyzer |  | → Boolean | | existingNode | Analyzer | Node x Node | → Boolean | | deleteNode | Modifier | Node |  | | getNode | Analyzer | Node | → Node | |

| **createNode():** |
| --- |
| \*Create a new Node. |
| {pre:TRUE} |
| {post: Node created. If the list M.L.L is empty, then the first Node of the list is n} |

| **insertNode(n, nPrev, nNext):** |
| --- |
| \*Node n is inserted into the linked list so that its next Node (nNext) (if it exists) is strictly greater and its previous element (nPrev) (if it exists) is less or equal. The current node becomes the inserted node. \* |
| {pre: M.L.L has to be initialized }  {pre: n exists (It must be different from null because we need to compare somehow)} |
| {post: The size of the list M.L.L increases one} |

| **isEmpty():** |
| --- |
| \*Verify if the linked list has elements (Nodes) or not.\* |
| {pre: Linked list must to exist} |
| {post: True if M.L.L == null  False otherwise } |

| **existingNode(n, actualNode):** |
| --- |
| \* Verify the existence of a Node n in the M.L.L list, the currentNode variable is used to navigate between the Nodes in the list and compare it with n until it is equal \* |
| {pre: Linked list must to exist}  {pre: n exists (It must be different from null because we need to compare somehow) } |
| {post: False if n doesn’t exist in M.L.L  True if n exists in M.L.L } |

| **deleteNode(n);** |
| --- |
| \*The current item is eliminated from the list as long as it exists. This verification is done with the ExistingNode() method. The Node before the deleted node has the next and previous relationships that the deleted Node had.\* |
| {pre: Linked list must to be initialized }  {pre: n exists (It must be different from null because we need to compare somehow) }  {pre: n exists in the list M.L.L } |
| {post: Node eliminated} |

| **getNode(n):** |
| --- |
| \* A Node n is obtained from the list as long as it exists. This verification is done with the ExistingNode() method. \* |
| {pre: Linked list must to be initialized }  {pre: n exists (It must be different from null because we need to compare somehow) }  {pre: n exists in the list M.L.L } |
| {post: Returns the Node to get} |