|  |
| --- |
| **TAD** Queue. |
|  |
| {inv:} |
| **Operaciones primitivas:**   * Enqueue: Nodo 🡪 Queue * Dequeue: 🡪 Queue * Front: 🡪 Queue * IsEmpty: 🡪 Boolean * Size: 🡪 Entero |

**Operaciones modificadoras.**

|  |
| --- |
| **Enqueue ()**  “Agrega un nodo a la cola”  {pre: Nodo no está vacío}  {post: Queue.Size() 0} |

|  |
| --- |
| **Dequeue ()**  “Elimina y retorna el nodo que esta al principio de la cola”  {pre: Queue.Size() 0}  {post: Queue.Nodo & Queue.Size() – 1} |

**Operaciones analizadoras.**

|  |
| --- |
| **Front ()**  “Retorna el nodo que esta al principio de la cola”  {pre: Queue.Size() 0}  {post: Queue} |

|  |
| --- |
| **IsEmpty ()**  “Comprueba el estado de la cola”  {pre: TRUE}  {post: Retorna TRUE si la cola esta vacía. De lo contrario, retorna FALSE} |

|  |
| --- |
| **Size ()**  “Retorna el número de elementos almacenados en la cola”  {pre: TRUE}  {post: Retorna un Z+ *n*. De lo contrario, retorna 0} |

|  |
| --- |
| **TAD** Stack. |
|  |
| {inv:} |
| **Operaciones primitivas:**   * Push: Nodo 🡪 Stack * Pop: 🡪 Stack * Top: 🡪 Stack * IsEmpty: 🡪 Boolean * Size: 🡪 Entero |

**Operaciones modificadoras.**

|  |
| --- |
| **Push ()**  “Agrega un nodo a la pila”  {pre: Nodo no está vacío}  {post: Stack.Size() 0} |

|  |
| --- |
| **Pop ()**  “Retorna y elimina el nodo que se encuentra en la cima de la pila”  {pre: Stack.Size() 0}  {post: Stack.Nodo & Stack.Size() – 1} |

|  |
| --- |
| **Top ()**  “Retorna el nodo que está en la cima de la pila”  {pre: Stack.Size() 0}  {post: Stack.Nodo} |

**Operaciones Analizadoras.**

|  |
| --- |
| **IsEmpty ()**  “Comprueba el estado de la pila”  {pre: TRUE}  {post: Retorna TRUE si la pila esta vacía. De lo contrario, retorna FALSE} |
| **Size ()**  “Retorna el número de elementos almacenados en la pila”  {pre: TRUE}  {post: Retorna un Z+ *n*. De lo contrario, retorna 0} |

|  |
| --- |
| **TAD** Hash table. |
| Diagrama  Descripción generada automáticamente |
| {inv: HashTable.length > HashTable.size} |
| **Operaciones primitivas:**   * Add: Key, Value 🡪 HashTable * Search: Key 🡪 HashTable * Delete: Key 🡪 HashTable * returnHash: 🡪 Lista |

**Operaciones modificadoras.**

|  |
| --- |
| **Add ()**  “Agrega un nuevo nodo a la tabla Hash”  {pre: HashTable.length HashTable.size}  {post: HashTable.size + 1} |

|  |
| --- |
| **Search ()**  “Busca un nodo en la tabla Hash con una clave ingresada”  {pre: HashTable.size 0}  {post: HashTable} |

|  |
| --- |
| **Delete ()**  “Elimina y retorna un nodo de la tabla Hash con una clave ingresada”  {pre: HashTable.size 0}  {post: HashTable.size – 1 HashTable} |

**Operaciones analizadoras.**

|  |
| --- |
| **ReturnHash ()**  “Retorna una lista de todos los elementos almacenados en la tabla Hash”  {pre: HashTable.size 0}  {post: Lista} |

|  |
| --- |
| **TAD** Heap. |
| Diagrama  Descripción generada automáticamente |
| {inv:} |
| **Operaciones primitivas:**   * Heap: 🡪 Heap * BuildHeap: 🡪 Heap * Insert: Nodo 🡪 Heap * Heapify: Entero 🡪 Heap * GetHeapSize: 🡪 Entero * GetElements: 🡪 Lista |

**Operaciones constructoras.**

|  |
| --- |
| **Heap ()**  “Crea un heap sin elementos”  {pre: TRUE}  {post: Heap.size = 0} |

**Operaciones modificadoras.**

|  |
| --- |
| **BuildHeap ()**  “Modela la estructura del Heap”  {pre: Heap.size 0}  {post: Heap} |

|  |
| --- |
| **Insert ()**  “Agrega un nodo al Heap”  {pre: Heap.size 0}  {post: Heap.size + 1} |

|  |
| --- |
| **Heapify ()**  “Intercambia el elemento más pequeño del Heap con la raíz”  {pre: Heap.size 0}  {post: Heap} |

**Operaciones analizadoras.**

|  |
| --- |
| **GetHeapSize ()**  “Retorna el número de elementos almacenados en el Heap”  {pre: TRUE}  {post: Retorna un Z+ n. De lo contrario retorna 0} |

|  |
| --- |
| **GetElements ()**  “Retorna una lista de todos los elementos almacenados en el Heap”  {pre: Heap.size 0}  {post: Lista} |