TAD HashTable
HashTable<K,V> = {nodeHashTable = arr_size, K key = null, V value = null}
{ inv : K & V are objects & arr_size = 307}
Operaciones Primitivas • HashTable Ninguno → HashTable • insert Objeto, Objeto → HashTable • insert HashTable, HashTable → HashTable • search Objeto → Objeto • search HashTable, Objeto → Objeto • delete Objeto → Boolean • delete HashTable, Objeto → Boolean • hashFunction → Integer

Constructura

HashTable() "Inicializa el constructor de la clase HashTable" { post : HashTable = { arr_size = 307 } }

Modificadora

insert(K key, V value) "Si no existen aún elementos en la hash table, crea un nuevo nodo con el elemento a añadir y se inserta en la hashTable dependiendo de la llave (key)" { pre : key & value ¡= null} { post : se ha añadido el nodo a la hashTable }

insert(HashTable ht, HashTable ht2) "Si ya existe más de un nodo, añade el nuevo nodo que contiene el objeto(value) en un espacio vacio de la hashTable" { pre : objeto ¡= null}

{ post : se ha añadido el nodo a la hashTable }

Analizadora

search(K key)

"Retorna el valor de la llave asociada a esa clave"
{ pre : key ¡= null }
{ post : valor asociado a la llave(key) }

search(HashTable ht, K key)

"Si existe mas de un objeto, busca entre ellos y retorna el valor de la llave asociada a esa clave"
{ pre : key ¡= null }
{ post : valor asociado a la llave(key) }

Destructura

delete(K key)

"Si solo hay un nodo en la HashTable, elimina el elemento contenido en ella y devuelve un booleano"

{ pre : key ¡= null }

{ post : return true si el elemento fue eliminado y false si no fue asi}

delete(K key)

"Si hay mas de un elemento en la HashTable, elimina el elemento contenido en ella y devuelve un booleano"

{ pre : key ¡= null }

{ post : return true si el elemento fue eliminado y false si no fue así}

TAD QUEUE

TAD Queue

Queue<T> = {Front = Null, Back = Null }
{ inv : Lista simplemente enlazada de objetos}
Operaciones Primitivas • enqueue Objeto → nodo • enqueue nodo, objeto → nodo • dequeue Ninguna → Objeto • front Ninguna → Objeto • isEmpty Ninguna → Boolean • Queue → queue

Constructura

Queue() “Inicializa el constructor de la clase Queue” { post : queue = { front = null}

Modificadora

Enqueue(T element) “Si no existen aún elementos en la cola, crea un nuevo nodo con el elemento a añadir y se inserta en la cola” { pre : objeto ¡= null} { post : front → nuevo nodo con el elemento }

Enqueue(nodo , objeto) “Si ya existe mas de un elemento en la cola, añade el nuevo nodo que contiene el objeto en un espacio vacio de la queue” { pre : objeto ¡= null} { post : back → nuevo nodo con el elemento }

Analizadora

front() “Retorna el valor actual de front” { pre : constructor ya inicializado } { post : front.getObject}

isEmpty() “Retorna un valor booleano indicando si la lista esta vacia o hay elementos en ella” { post : si front → null, entonces boolean = true sino boolean = false}

Destructura

dequeue() “Elimina el elemento contenido en front y devuelve ese elemento eliminado” { pre : front ¡= null } { post : return deletedObject}

TAD STACK

TAD Stack		
Stack<T> = {Top = null, Size = 0 }		
{ inv : Lista simplemente enlazada de objetos}		
Operaciones Primitivas • pop • push • size • top • isEmpty • Stack	Objeto	→ Stack → Stack → int → Objeto → Boolean → Stack

Constructura

Stack()

“Inicializa el constructor de la clase Stack”

{ post : stack = { top = null}

Modificadora

Push(T objeto)

“Si no existen aún elementos en la pila, crea un nuevo nodo con el elemento a añadir y se inserta en la pila”

{ pre : objeto ¡= null}

{ post : Pila con el nuevo y primer nodo añadido }

Push(nodo , objeto)

“Si ya existe mas de un elemento, añade el nuevo nodo que contiene el objeto en un espacio vacio de la cola”

{ pre : objeto ¡= null}

{ post : El nuevo nodo ha sido añadido a la pila}

Analizadora

Top()

“Retorna el valor actual de top”
{ pre : constructor ya inicializado }
{ post : top.getObject() }

isEmpty()

“Retorna un valor booleano indicando si la pila esta vacia o hay elementos en ella”

{ post : si → size = 0, entonces boolean = true pero si size >0 boolean = false}

size()

“Retorna un numero entero indicando la cantidad de objetos contenidos en la pila”

{pre: constructor ya inicializado }

{ post : size = n elementos en la pila}

Destructura

pop()

“Elimina el elemento contenido en top y devuelve ese elemento eliminado”

{ pre : top ¡= null }

{ post : return deletedObject}