Andrés Mayor Aldana – Jhonatan Arboleda

TAD Design

TAD HashTable<K,V>
HashTable = { Key = <key>, Value = <value> }
{ inv: HashTable.length >= Hashtable.size **}**
Operations: ● HashTable: None → HashTable<K,V> ● getValue: Key<K> x HashTable<K,V> → Value<V> ● insert: Key<K> x Value<V> x HashTable<K,V> → HashTable<K,V> ● delete: Key<K> x HashTable<K,V> → HashTable<K,V> ● contains: Key<K> x HashTable<K,V> → Boolean

HashTable()

“Crear una nueva HashTable sin elementos”

{pre: TRUE}

{post: HashTable = { } }

search(key<K>)

“Obtener el valor correspondiente a la llave dada”

{pre: TRUE}

{post: value<V>}

insert(key<K> , value<V>)

“Añadir un par de datos a la hashtable”

{pre: hashtable.length < hashtable.size }

{post: hashtable.length = hashtable.length+1 }

Andrés Mayor Aldana – Jhonatan Arboleda

delete(key<K>)

“Eliminar un par de datos de la tabla hash determinado por la llave dada y su respectivo valor”

{pre: hashtable.size > 0 }

{post: hashtable.length = hashtable.lenght - 1 }

contains(key<K>)

“Verificar si en la tabla hash se encuentra un par cuya llave coincida con la llave dada”

{pre: hashtable.size > 0 }

{post: TRUE, FALSE }

TAD Queue<T>		
Queue = { First = <first>, Latest = <latest>}		
{ inv: }		
Operaciones primitivas: ●		
Queue :		→ Queue<T>
● offer:	Queue<T> x Object<T>	→ Queue<T>
● peek:	Queue<T>	→ Object<T>
● poll:	Queue<T>	→ Object<T>

Queue()

“Create a new queue without items”

{pre: TRUE}

{post: queue = { First= null, Latest = null} }

Andrés Mayor Aldana – Jhonatan Arboleda

offer(queue, object)

“Add an object to the queue”

{post: queue = { First= first, Latest = latest} object ∈ T}

{post: queue = { First= first, Latest = object} }

peek(queue)

“Returns the first item in the queue”

post: queue = { First= first, Latest = latest} }

{post: <first>}

poll(queue)

“Returns he first item in the queue and removes it”

{post: queue = { First= first, Latest = latest} }

{post: <first> ^ queue = { First= first.Next, Latest = latest} }

TAD Stack<T>

Stack = { First = <first>}

{ inv: }

Primitive Operatio ɿs:

● Stack :		→ Stack<T>
● push:	Stack<T> x Object<T>	→ Stack<T>
● peek:	Stack<T>	→ Object<T>
● pop:	Stack<T>	→ Object<T>
● empty:	Stack<T>	→ Boolean

Andrés Mayor Aldana – Jhonatan Arboleda

Stack()

“Create a new stack without items”

{pre: TRUE}

{post: stack = { First= null} }

offer(stack, object)

“Add an object to the stack”

{post: stack = { First= element } object ∈ T}

{post: stack = { First= object} }

peek(stack)

“Returns the first item in the stack without removing it”

post: stack = { First= first } }

{post: <first>}

pop(stack)

“Returns he first item in the queue and removes it”

post: stack = { First= first } }

{post: <first> ^ stack = { First= first.Next } }

empty(stack)

“Informs if the stack has at least one item”

post: queue = { First= first } }

{post: False si stack.first = null. True de lo contrario }

Andrés Mayor Aldana – Jhonatan Arboleda

TAD Node<T>

Node = { Element = <element>, Next = <next>, Prior = <prior>}

{ inv: }

Operaciones primitivas:

● Node :	Object<T>	→	Node<T>
● getNext:	Node<T>	→	Node<T>
● getPrior:	Node<T>	→	Node<T>
● setNext:	Node<T> x Object<T>	→	Node<T>
● setPrior:	Node<T> x Object<T>	→	Node<T>
● getElement	Node<T>	→	Object<T>

Node(element)

“Create a new Node with the Next and Prior nulls”

{pre: element ∈ T ^ element != null }

{post: node = {Element:element, Next = null, Prior = null }

getNext(node)

“Returns Next node of the node”

{pre: node = {Element:element, Next = next, Prior = prior } }

{post: <next> }

getPrior(node)

“Returns Prior node of the node”

{pre: node = {Element:element, Next = next, Prior = prior } }

{post: <prior> }

setNext(node, n)

“Change Next node of the node”

{pre: node = {Element:element, Next = null, Prior = Node<T> } ^ n ∈ T }

{post: node = {Element:element, Next = n, Prior = prior } }

Andrés Mayor Aldana – Jhonatan Arboleda

setPrior(node, p)

“Change Prior node on the node”

{pre: node = {Element:element, Next = next, Prior = prior } ^ p ∈ T }

{post: node = {Element:element, Next = next, Prior = p } }

getElement(node)

“Returns the element of the node”

{pre: node = {Element:element, Next = next, Prior = prior } }

{post: <element> }