TAD STRUCTURES

Airline

Members:

- Alexis Jaramillo (A00395655)
- David Malte (A00368867)
- Juan Daniel Reina (A00394352)

TYPE	**HashNode<K,V>**
ATTRIBUTES	- **value:** T, value of type T stored in the node. - **- next:** Node<T>, reference to the next node in the list.
OPERATIOS	- **HashNode(K key, V value):** constructor that creates a new node with the given key and value. - **getKey():** returns the key of the node. - **getValue():** returns the value associated with the key of the node. - **setValue(V value):** sets the value associated with the node key. - **getNext():** returns the reference to the next node in the list. - **setNext(HashNode<K,V> next):** sets the reference to the next node in the list. - **getPrevious():** returns the reference to the previous node in the list. - **setPrevious(HashNode<K,V> previous):** sets the reference to the previous node in the list.

Method:	HashNode(K key, V value)
Description:	The constructor of the HashNode class that initializes the node with a key and a value.
Input:	**K key:** the key that will be used to look up this node in the hash table. **V value:** the value associated with this key.
Output:	None

Method:	getKey()
Description:	Returns the key of the node
Input:	None
Output:	The node's key of type K

Method:	getValue()
Description:	Returns the value of the node
Input:	None
Output:	The node's value of type V

Method:	setValue(V value)
Description:	Sets the value of the node to a new value.
Input:	V value: the new value to set in the node.
Output:	None

Method:	getNext()
Description:	Returns the next node in the hash list.
Input:	None.
Output:	The next node of type HashNode.

Method:	setNext(HashNode next)
Description:	Sets the next node in the hash list.
Input:	HashNode next: the next node to set.
Output:	None

Method:	getPrevious ()
Description:	Returns the previous node in the hash list.
Input:	None.
Output:	The previous node of type HashNode.

Method:	setPrevious(HashNode previous)
Description:	Sets the previous node in the hash list.
Input:	HashNode previous: the previous node to set.
Output:	None

TYPE	**HashTable<K, V>**
ATTRIBUTES	- **size:** int, current size of the table. - **table:** HashNode<K, V>[], array of nodes to store the elements.
OPERATIONS	- **HashTable(int capacity):** constructor that creates a new hash table with the given capacity. - **put(K key, V value):** inserts a key-value pair into the table. - **get(K key):** returns the value associated with a given key, or null if the key does not exist. - **remove(K key):** removes a key-value pair from the table and returns the value associated with the removed key, or null if the key does not exist. - **hash(K key):** hash function that calculates the index in the array for a given key. - **addSize(int newCapacity):** increases the size of the table and rehashes existing elements. - **getSize():** returns the current size of the table. - **toString():** returns a string representation of the hash table.

Method:	HashTable
Description:	A hash table data structure that stores key-value pairs and allows fast access to values by key.
Input:	A capacity value for the initial size of the hash table.
Output:	None

Method:	put
Description:	Adds a new key-value pair to the hash table or updates the value if the key already exists.
Input:	A key-value pair to be added or updated.
Output:	None

Method:	get
Description:	Returns the value associated with a given key in the hash table.
Input:	A key to search for in the hash table.
Output:	The value associated with the given key, or null if the key is not found in the hash table.

Method:	remove
Description:	Removes the key-value pair associated with a given key from the hash table.
Input:	A key to remove from the hash table.
Output:	The value associated with the given key that was removed, or null if the key is not found in the hash table.

Method:	hash
Description:	Computes a hash value for a given key to determine its index in the hash table.
Input:	A key to compute the hash value for.
Output:	The index in the hash table for the given key.

Method:	addSize
Description:	Increases the capacity of the hash table by creating a new, larger hash table and rehashing all existing key-value pairs into it.
Input:	The new capacity for the hash table.
Output:	None.

Method:	getSize
Description:	Returns the number of key-value pairs currently stored in the hash table.
Input:	None.
Output:	The number of key-value pairs in the hash table.

Method:	toString
Description:	Returns a string representation of the key-value pairs in the hash table.
Input:	None.
Output:	A string that lists all key-value pairs in the hash table, formatted as "[value1\nkey1: value1, key2: value2, ...]"

TYPE	**Node<T>**
ATTRIBUTES	- **value:** T, value of type T stored in the node. - **next:** Node<T>, reference to the next node in the list.
OPERATIONS	- **Node(T value):** constructor that creates a new node with the given value. - **getValue():** returns the value stored in the node. - **setValue(T value):** sets the value stored in the node. - **getNext():** returns the reference to the next node in the list. - **setNext(Node<T> next):** sets the reference to the next node in the list.

Method:	Node(T value)
Description:	The constructor of the Node class that initializes the node with a value.
Input:	T value: the value to be stored in the node.
Output:	None.

Method:	getValue()
Description:	Returns the value stored in the node.

Input:	None.
Output:	The value of type T.

Method:	setValue(T value)
Description:	Sets the value of the node to a new value.
Input:	T value: the new value to set in the node.
Output:	None.

Method:	getNext()
Description:	Returns the next node in the linked list.
Input:	None.
Output:	The next node of type Node<T>.

Method:	setNext(Node<T> next)
Description:	Sets the next node in the linked list.
Input:	Node<T> next: the next node to set.
Output:	None.

TYPE	**Queue<T>** (Queue)
ATTRIBUTES	- **front:** Node<T>, reference to the front of the queue. - **back:** Node<T>, reference to the end of the queue. - **size:** int, current size of the queue.
OPERATIONS	- **Queue():** constructor that creates a new empty queue. - **enqueue(T data):** add an element to the end of the queue. - **dequeue():** removes and returns the element at the front of the queue. - **peek():** returns the element at the front of the queue without removing it. - **isEmpty():** returns true if the queue is empty, false otherwise. - **size():** returns the current size of the queue. - **toString():** returns a string representation of the queue.

Method:	Queue()
Description:	The constructor of the Queue class that initializes an empty queue.
Input:	None
Output:	None

Method:	enqueue(T data)
Description:	Adds an element to the back of the queue.
Input:	T data: the element to be added to the queue.
Output:	None.

Method:	dequeue()
Description:	Removes and returns the element at the front of the queue.
Input:	None.
Output:	The element of type T that was removed from the queue.

Method:	peek()
Description:	Returns the element at the front of the queue without removing it.
Input:	None
Output:	The element of type T at the front of the queue.

Method:	isEmpty()
Description:	Checks if the queue is empty.
Input:	None
Output:	true if the queue is empty, false otherwise.

Method:	size()
Description:	Returns the number of elements in the queue.
Input:	None
Output:	The number of elements in the queue as an integer.

Method:	toString()
Description:	Returns a string representation of the elements in the queue.
Input:	None.
Output:	A string representation of the elements in the queue, formatted as "[element1, element2, ...]".

TYPE	**Stack<T>** (Stack)
ATTRIBUTES	- **top:** Node<T>, reference to the top element of the stack. - **size:** int, current size of the stack.
OPERATIONS	- **Stack():** constructor that creates a new empty stack. - **push(T data):** push an element on top of the stack. - **pop():** removes and returns the element at the top of the stack. - **peek():** returns the element at the top of the stack without removing it. - **size():** returns the current size of the stack. - **isEmpty():** returns true if the stack is empty, false otherwise. - **toString():** returns a string representation of the stack.

Method:	Stack()
Description:	The constructor of the Stack class that initializes an empty stack.
Input:	None.
Output:	None.

Method:	push(T data)
Description:	Adds an element to the top of the stack.
Input:	T data: the element to be added to the stack.

Output:	None.

Method:	pop()
Description:	Removes and returns the element at the top of the stack.
Input:	None.
Output:	The element of type T that was removed from the stack.

Method:	peek()
Description:	Returns the element at the top of the stack without removing it.
Input:	None.
Output:	The element of type T at the top of the stack.

Method:	size()
Description:	Returns the number of elements in the stack.
Input:	None.
Output:	The number of elements in the stack as an integer.

Method:	isEmpty()
Description:	Checks if the stack is empty.
Input:	None.
Output:	true if the stack is empty, false otherwise.

Method:	toString()
Description:	Returns a string representation of the elements in the stack.
Input:	None.
Output:	A string representation of the elements in the stack, formatted as "[element1, element2, ...]".