**Presented by : Juan Esteban Ruiz – Tomas Quintero**

|  |
| --- |
| **ADT HASHTABLE** |
| **Invariants:**   * The Hash table does not contain duplicate keys. * Keys cannot be null. * The Hash table has a fixed size and automatically resizes when necessary. |
| **Primitive Operations:**  1. put(key, value) -> void  • Description: Inserts a key-value pair into the Hashtable or updates the value if the key already exists.  • Input: A key (key) and a value (value).  • Precondition: key is not null.  • Postcondition: The Hashtable contains the key-value pair (key, value).  2. get(key) -> value   * Description: Retrieves the value associated with the specified key. * Input: A key (key). * Precondition: key is not null. * Output: The value associated with the key or null if the key does not exist.   3. remove(key) -> void   * Description: Deletes the key-value pair associated with the specified key. * Input: A key (key). * Precondition: key is not null, and the key exists in the Hashtable. * Postcondition: The Hashtable does not contain the specified key.   4. containsKey(key) -> boolean   * Description: Checks if the Hashtable contains the specified key. * Input: A key (key). * Precondition: key is not null. * Output: true if the key exists in the Hashtable, false otherwise.   5. size() -> int   * Description: Gets the number of key-value pairs in the Hashtable. * Input: None. * Output: The number of elements in the Hashtable.   6. isEmpty() -> boolean   * Description: Checks if the Hashtable is empty. * Input: None. * Output: true if the Hashtable is empty, false otherwise.   7. clear() -> void   * Description: Removes all key-value pairs from the Hashtable. * Input: None. * Postcondition: The Hashtable is empty.   8. keys() -> Set<Key>   * Description: Retrieves a set of all keys in the Hashtable. * Input: None. * Output: A set of keys.   9. values() -> Collection<Value>   * Description: Retrieves a collection of all values in the Hashtable. * Input: None. * Output: A collection of values.   10. entrySet() -> Set<Entry<Key, Value>>   * Description: Retrieves a set of entries (key-value pairs) in the Hashtable. * Input: None. * Output: A set of entries. |

|  |
| --- |
| **ADT DYNAMICQUEUE** |
| **Invariants:**   * A dynamic queue contains a collection of elements. * The elements are ordered in a First-In-First-Out (FIFO) manner. |
| **Primitive Operations:**  1. enqueue(element) -> void   * Description: Adds an element to the back of the dynamic queue. * Input: An element. * Precondition: The element is not null. * Postcondition: The element is added to the back of the queue.   2. dequeue() -> element   * Description: Removes and returns the front element from the dynamic queue. * Input: None. * Precondition: The queue is not empty. * Output: The front element of the queue.   3. peek() -> element   * Description: Returns the front element of the dynamic queue without removing it. * Input: None. * Precondition: The queue is not empty. * Output: The front element of the queue.   4. isEmpty() -> boolean   * Description: Checks if the dynamic queue is empty. * Input: None. * Output: true if the queue is empty, false otherwise.   5. size() -> int   * Description: Returns the number of elements in the dynamic queue. * Input: None. * Output: The number of elements in the queue.   6. clear() -> void   * Description: Removes all elements from the dynamic queue. * Input: None. * Postcondition: The queue is empty. |

|  |
| --- |
| **ADT DYNAMICSTACK** |
| **Invariants:**   * A dynamic stack contains a collection of elements. * The elements are ordered in a Last-In-First-Out (LIFO) manner. |
| **Primitive Operations:**  **1. push(element) -> void**   * Description: Adds an element to the top of the dynamic stack. * Input: An element. * Precondition: The element is not null. * Postcondition: The element is added to the top of the stack.   **2. pop() -> element**   * Description: Removes and returns the top element from the dynamic stack. * Input: None. * Precondition: The stack is not empty. * Output: The top element of the stack.   **3. peek() -> element**   * Description: Returns the top element of the dynamic stack without removing it. * Input: None. * Precondition: The stack is not empty. * Output: The top element of the stack.   **4. isEmpty() -> boolean**   * Description: Checks if the dynamic stack is empty. * Input: None. * Output: true if the stack is empty, false otherwise.   **5. size() -> int**   * Description: Returns the number of elements in the dynamic stack. * Input: None. * Output: The number of elements in the stack.   **6. clear() -> void**   * Description: Removes all elements from the dynamic stack. * Input: None. * Postcondition: The stack is empty. |

|  |
| --- |
| **ADT PRIORITYQUEUE** |
| **Invariants:**   * A priority queue contains a collection of elements, each associated with a priority value. * Elements are removed from the priority queue based on their priority values, with higher-priority elements being removed first. |
| **Primitive Operations:**  **1. enqueue(element, priority) -> void**   * Description: Inserts an element with a specified priority into the priority queue. * Input: An element and its associated priority. * Precondition: The element is not null, and the priority is a valid priority value. * Postcondition: The element is inserted into the priority queue with the specified priority.   **2. dequeue() -> element**   * Description: Removes and returns the element with the highest priority from the priority queue. * Input: None. * Precondition: The priority queue is not empty. * Output: The element with the highest priority is removed and returned.   **3. peek() -> element**   * Description: Returns the element with the highest priority from the priority queue without removing it. * Input: None. * Precondition: The priority queue is not empty. * Output: The element with the highest priority.   **4. isEmpty() -> boolean**   * Description: Checks if the priority queue is empty. * Input: None. * Output: true if the priority queue is empty, false otherwise.   **5. size() -> int**   * Description: Returns the number of elements in the priority queue. * Input: None. * Output: The number of elements in the priority queue.   **6. clear() -> void**   * Description: Removes all elements from the priority queue. * Input: None.   Postcondition: The priority queue is empty. |