Design of use cases - Integrative task one

- Priority queues test cases:

1. Test cases for add functionality:

Test objective: Verify that the add method adds an element to the queue with the correct priority according to the natural order.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	add(T element)	Priority queue is empty.	Element to insert: 10	.The priority queue should contain item 10.

Test objective: Verify that the add method of the PriorityQueue class correctly inserts an item into the priority queue when there are already items in the queue.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	add(T element)	The priority queue contains the elements [3, 5, 8]	Element to insert: 6	The priority queue should contain items [3, 5, 6, 8] in that order.

Test objective: Verify that the add method increases the internal capacity of the queue if necessary to store more elements.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	add(T element)	The priority queue is full and contains 10 items.	Element to insert: 15	After adding item 15, the internal capacity of the queue should automatically increase to allow for its storage. The priority queue should contain 11 items and the internal capacity of the queue should have increased by at least 1 unit.

2. Test cases for remove functionality:

Test objective: To test the functionality of removing an element from an empty priority queue.

Class	Method	Scenery	Input values	Expected result
PriorityQueue	remove()	An empty priority queue.	No input values.	The method should raise an error indicating that the priority queue is empty.

Test objective: To test the functionality of removing an element from a non-empty priority queue.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	remove()	A priority queue with several elements.	No input values.	The method should remove the highest-priority element from the queue and return it.

Test objective: To test the functionality of removing an element that is not in the priority queue.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	remove()	A priority queue with several elements.	An element that is not in the queue.	The method should raise a NoSuchElementException error indicating that the element is not in the queue.

3. Test cases for peek functionality:

Test objective: Verify peek() method returns None for an empty priority queue.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	peek()	An empty priority queue.	Empty priority queue	The system should send an exception that the priority queue is empty.

Test objective: Test the peek() method after adding and removing several elements.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	peek()	Priority queue with	A priority queue with	The method should return the minimum

		elements added and removed.	the elements [1, 3, 2, 6, 5] added in that order and then the remove() method is called twice.	element, which is 3.

Test objective: Test the peek() method after adding an element larger than the minimum.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
PriorityQueue	peek()	Priority queue with elements and an element larger than the mínimum.	A priority queue with the elements [1, 2, 3, 4] added in that order and then the add(5) method is called.	The method should return the minimum element, which is still 1.

Binary Search Tree test cases:

1. Test cases for insert functionality:

Test objective: Verify that the insert method adds a node to the binary search tree in the correct position according to the value of the node.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	insert(T value)	The binary search tree is empty.	10	Expected result: The binary search tree should contain a single node with value 10

.

Test objective: Verify that the insert method adds a node to the binary search tree in the correct position according to the value of the node.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearch Tree	insert(T value)	The binary search tree contains several nodes	6.	The binary search tree should contain a node with value 6 in the correct position based on its value.

Test objective: Verify that the insert method correctly inserts a new node in the binary search tree.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	insert(T value)	The binary search tree contains several nodes, and a new node with a value lower than the root node is inserted	4	The binary search tree should contain a new node with value 4 in the correct position

2. Test cases for Delete functionality:

Test objective: Verify that the delete method removes a node from the binary search tree correctly.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	delete(T value)	The binary search tree contains the node to be deleted.	10.	The binary search tree should not contain a node with value 10.

Test objective: Verify that the delete method does not change the binary search tree if the node to be deleted is not present.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	delete(T value)	The binary search tree does not contain the node to be deleted.	15.	The binary search tree should remain unchanged.

Test objective: Verify that the delete method correctly removes a node from the binary search tree				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	delete(T value)	The binary search tree contains several nodes, and a node with two children is removed	10	The binary search tree should not contain a node with value 10, and the remaining nodes should be properly organized

3. Test cases for search functionality:

Test objective: Verify that the search method correctly finds a node in the binary search tree.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	search(T value)	The binary search tree contains the node to be found	8	Expected result: The method should return the node with value 8.

Test objective: Verify that the search method correctly returns None when the node is not in the binary search tree.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	search(T value)	The binary search tree does not contain the node to be found.	15.	The method should return None.

Test objective: Verify that the method correctly finds a node in the binary search tree				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	search(T value)	The binary search tree contains several nodes, and a node on a side branch of the tree is searched.	8	The method should return the node with value 8.

4. Test cases for inOrder functionality:

Test objective: Verify that the inOrder method correctly returns the nodes in the binary search tree in ascending order.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	inOrder()	The binary search tree contains several nodes	N/A	The method should return the nodes in ascending order based on their values

Test objective: Verify that the inOrder method correctly returns the nodes in the binary search tree in ascending order when the tree only has one node.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	inOrder()	The binary search tree contains only one node with value 5.	N/A	The method should return a list with a single element containing the value 5.

Test objective: Verify that the inOrder method correctly returns the nodes in the binary search tree in ascending order when the tree has multiple nodes with the same value.				
Class	**Method**	**Scenery**	**Input values**	**Expected result**
BinarySearchTree	inOrder()	The binary search tree contains several nodes with value 8.	N/A	The method should return a list with all the nodes with value 8 in ascending order.