TEST DESIGN

Scenario configuration:

Name	**Class**	**Scenario**
setUpStage1	HashTest	The scenario is a 10-space hashTable where we are going to perform several tests
setUpStage2	PriorityTest	The scenario is a 10-space priority queue where we are going to perform several tests
setUpStage3	StackTest	The scenario is a stack where we are going to perform several tests
setUpStage4	QueueTest	The scenario is a queue where we are going to perform several tests

Test case design:

Objective of the test:		The objective of this test in this class is to test various methods such as insert, delete, testing the Hash function, insert and search, and collision handling.		
Class	**Method**	**Scenario**	**Input values**	**Result**
HashTest	testInsert	setUpStage1	The entries are two objects with their key and their value	The outputs are two statements of two values that are expected to be obtained by searching for
HashTest	testDelete	setUpStage1	the entry is an object with its key and value	the output is the assertion that this object could be removed, and the false assertion that another object that does not exist can be removed.
HashTest	testFunctionHash	setUpStage1	Calls to hashTable.FunctionHash("key1") and hashTable.FunctionHash("key2").	Assertion with assertTrue to verify that the generated hash values are in the expected range (between 0 and 9).

HashTest	testInsertAndSearch	setUpStage1	Calls to hashTable.insert("key1", 1) and hashTable.insert("key2", 2) to add entries.	Assertion with assertEquals to verify that hashTable.search("key1") returns Integer.valueOf(1) and hashTable.search("key2") returns Integer.valueOf(2). Assert with assertNull to verify that hashTable.search("key3") returns null.
HashTest	testColisiones	setUpStage1	Call to hashTable.insert("cola", 4) to add an entry Additional call to hashTable.insert("cola", 3) to add another entry with the same key.	Assertion with assertEquals to verify that hashTable.search("cola") returns Integer.valueOf(4) after the first insertion. Assertion with assertEquals to verify that hashTable.search("cola") returns Integer.valueOf(3) after the second insertion, demonstrating that the collision was handled correctly.

Objective of the test:		The objective of this test is to perform several tests to the methods that are in the priorityTest, such as insert and remove, show the priorityQueue, also with different priorities, with the same priority and several tasks.		
Class	**Method**	**Scenario**	**Input values**	**Result**
PriorityTest	testInsertAndRemove	setUpStage2	Calls priorityQueue.insert(task1) and priorityQueue.insert(task2) to add tasks. Calls priorityQueue.remove() twice to remove tasks.	Verification that tasks are correctly inserted. Assertions with assertEquals to verify that tasks are removed as expected. Verification that the priority queue is empty at the end of the test.
PriorityTest	testShowPriorityQueue	setUpStage2	Calls to priorityQueue.insert(task1) and priorityQueue.insert(task2) to add tasks.	Verification that tasks are correctly inserted. Conversion of the priority queue to a string and assertion with assertTrue that it contains the task titles.

PriorityTest	testInsertAndRemoveWithDifferentPriorities	setUpStage2	Calls to priorityQueue.insert(lowPriorityTask) and priorityQueue.insert(highPriorityTask) to add tasks with different priorities. Calls to priorityQueue.remove() twice to remove tasks.	Verification that tasks with different priorities are correctly inserted. Assertions with assertEquals to verify that tasks are removed in the correct priority order
PriorityTest	testInsertAndRemoveWithSamePriority	setUpStage2	Calls to priorityQueue.insert(task1) and priorityQueue.insert(task2) to add tasks with the same priority. Calls to priorityQueue.remove() twice to remove tasks	Verification that tasks with the same priority are correctly inserted. Assertions with assertEquals to verify that tasks are removed in the order they were inserted.
PriorityTest	testShowPriorityQueueWithMultipleTasks	setUpStage2	Calls to priorityQueue.insert(task1) and priorityQueue.insert(task2) to add multiple tasks.	Verification that multiple tasks are correctly inserted. Conversion of the priority queue to a string and assertion with assertTrue that it contains the titles of all tasks.
PriorityTest	testRemoveOnEmptyPriorityQueue	setUpStage2	Check if the priority queue is empty and attempt to remove an item.	Verification that the priority queue is empty. Handling of an exception when trying to remove from an empty queue

Objective of the test:		The objective of this test is to perform various tests on the methods found in the Stack, such as adding an element to the stack, removing element from the stack, checking if the stack is empty and checking the number of elements in the stack.		
Class	**Method**	**Scenario**	**Input values**	**Result**
StackTest	testPushAndPeek	setUpStage3	Push integers into the stack, call “peek”	Verify that the value returned by “peek” matches the last pushed value.

StackTest	testPop	setUpStage3	Push integers into the stack, call “pop”	Verify that the value returned by “pop” matches the last pushed value, and check that the stack is empty after popping all values.
StackTest	testIsEmpty	setUpStage3	Push integers into the stack	Verify that “isEmpty” returns “true” if the stack is empty and “false” if it contains elements.
StackTest	testSize	setUpStage3	Push and pop integers from the stack	Check that “size” returns the correct number of elements in the stack..

Objective of the test:		The purpose of this test is to perform various tests on the methods found in the queue, such as adding an element to the end of the queue, removing and returning the element from the front of the queue, getting an element, checking if the queue is empty, knowing the number of elements in the queue, and print the queue.		
Class	**Method**	**Scenario**	**Input values**	**Result**
QueueTest	testEnqueueAndPeek	setUpStage4	Enqueue integers into the queue, call “peek”	Verify that the value returned by “peek” matches the first enqueued value.
QueueTest	testDequeue	setUpStage4	Enqueue integers into the queue, call dequeue	Verify that the value returned by “dequeue” matches the first enqueued value and check that the queue is empty after dequeuing all values.
QueueTest	testIsEmpty	setUpStage4	Enqueue integers into the queue	Verify that “isEmpty” returns “true” if the queue is empty and “false” if it contains elements.
QueueTest	testSize	setUpStage4	Enqueue and dequeue integers from the queue	Check that “size” returns the correct number of elements in the queue.
QueueTest	testRear	setUpStage4	Enqueue integers into the queue	Verify that the value returned by “rear“ matches the last enqueued value.
QueueTest	testDequeueOnEmptyQueue	setUpStage4	Try to dequeue from an empty queue	Verify that an “IllegalStateException”is thrown.