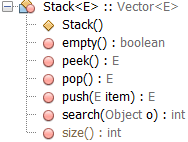
# Question 1

We want to test the predefined class java.util.Stack which represents a stack in memory (even if we want to believe that it is well encoded and tested by its original authors at Sun and Oracle).



Complete the unit test class below with the following test methods:

1. **testEmptyStackSize** **()** must verify that:
   1. The call to the *empty* () method must return "true" to a Stack object that has just been created (without adding any elements); otherwise the test displays the message "A new created stack must be empty".
   2. The call to the *size* () method returns zero; otherwise the test prints the message "A newly created stack should be of size 0".
2. **testEmptyStackPop ()** must verify that: The *pop ()* method throws an *EmptyStackException* exception when it is called on an empty stack.
3. **testStackSuccessfulPushPop ()** must verify that:
4. calling the *push ()* method with a given value increments the stack size by 1, so the test reads "A stack with 1 element should have size 1".
5. The call of *pop ()* afterwards returns the value given initially, otherwise the test displays "pop returns the top element of the stack",
6. and this last call will make the stack empty again; otherwise the test will display the message "After calling pop on a stack with 1 element, size must become 0".
7. **testStackElementsOrder ()** which verifies that:
8. The size of the stack becomes equal to 3 after the addition of three elements, for example the of strings "A", "B" and then "C";
9. The call of *pop()* three times successively will give the values ​​upside down, i.e. "C", then "B" and finally "A",
10. **testStackTimeout ()** which verifies that: the *push()* method called 1,000,000 times does not exceed 500 milliseconds.

public class StackTest {

@Test

public void testEmptyStackSize() {

final Stack emptyStack = new Stack();

}

@Test

public void testEmptyStackPop() {

final Stack<Object> emptyStack = new Stack();

}

@Test

public void testStackSuccessfulPushPop() {

final Stack<Object> stack = new Stack();

final String firstElement = "Anything...";

stack.push(firstElement);

// verify that size has become 1

Object top = stack.pop();

// verify that top is the same as firstElement

// verify that size becomes 0 again

}

@Test

public void testStackElementsOrder() {

final Stack<Object> stack = new Stack();

}

@Test

public void testStackTimeout() {

final Stack<Object> stack = new Stack();

for (int i = 0; i < 1000000; i++) {

stack.push(new String(String.valueOf(i)));

}

}

}

# Question 2

We want to test the predefined class java.util.PriorityQueue which represents a queue in memory (even if we want to believe that it is well encoded and tested by its original authors at Sun and Oracle).

Complete the unit test class below with the following test methods:

1. **testEmptyQueueSize ()** must verify that:
2. The call of the method *isEmpty* () must return "*true*" to an object of type PriorityQueue that has just been created (without adding any elements), otherwise the test displays the message "A newly created queue must be empty ".
3. The call to the *size* () method returns zero, otherwise the test prints the message "A newly created queue must be of size 0".
4. **testEmptyQueuePoll ()** must verify that: The *poll* () method returns null on an empty queue.
5. **testQueueAddNull** **()** must check that the *add* () method throws a *NullPointerException* when passing a "null" object as an argument to add it to a queue.
6. **testQueueSuccessfulAddPoll ()** must verify that:
7. Calling the *add* () method with a given value increments the size of the queue by 1, otherwise the test prints "Tailed with 1 elt should have size 1",
8. The call to *poll* () afterwards returns the value given initially, otherwise the test displays "poll returns the head of the queue",
9. and this last call will make the stack empty again, otherwise the test will display the message "After calling polls with 1 element, size is expected to become 0".
10. **testQueueElementsOrder** **()** which verifies that:
11. The size of the queue becomes equal to 3 after the addition of 3 elements, for example the chains of characters "A", "B" and then "C",
12. The call of poll () three times successively will give the values ​​with the same order, i.e. "A", then "B" and then finally "C".
13. **testQueueTimeout ()** which verifies that: the push call 1000 times does not exceed 500 milliseconds.

public class QueueTest {

public QueueTest() {

}

@Test

public void testEmptyQueueSize() {

final Queue emptyQueue = new PriorityQueue();

}

@Test

public void testEmptyQueuePoll() {

final Queue<Object> emptyQueue = new PriorityQueue<>();

Object head = emptyQueue.poll();

}

@Test

public void testQueueAddNull() {

final Queue<Object> queue = new PriorityQueue<>();

queue.add(null);

}

@Test

public void testQueueSuccessfulAddPoll() {

final Queue<Object> queue = new PriorityQueue();

final String firstElt = "Anything...";

queue.add(firstElt);

// verify that size has become 1

Object head = queue.poll();

// verify that head is the same as firstElement

// verify that size becomes 0 again

}

# Question 3

We want to test the predefined Java class java.util.ArrayList (even though it is believed that it is well programmed and well tested by its original authors). Complete the following unit test class with the following three test methods:

1. **testIsEmptyAndSize ( )**: must check that the call of the method *isEmpty ()* must return "true" on an ArrayList which has just been created and which contains no element, "false" otherwise. The method also checks that the *size ()* method also returns zero when the list is empty, otherwise a positive value.
2. **testGetExisting ( )**: must verify that searching for an element with *get (int index)* returns the correct object that exists at that location.
3. **testGetNonExisting ( )**: Check that the call to *get (int index)* with a negative value or greater than the size of the list returns an exception of type **IndexOutOfBoundException**.

import org.junit.Assert.\*;

import org.junit.\*;

public class ArrayListTest {

@Test

public void testIsEmptyAndSize() {

// write your test code here...

}

@Test

public void testGetExisting() {

// write your test code here...

}

@Test

public void testGetNonExisting() {

// write your test code here...

}

}

# Question 4

We want to test the "boolean contains (String)" method of the predefined Java class java.lang.String (even if we want to believe that it is well encoded and tested by its original authors at Sun and Oracle). Complete the unit test class below with the following three test methods and with the correct annotations (@ ...):

* 1. **testContainsNormal()**: must verify that the call to the *contains ()* method returns "true" on the string "samir" with valid parameters like "sam", "ami" or "mir", "false" otherwise as with "rim".
  2. **testContainsEmpty()**: must verify that the *contains ()* method returns "true" with a parameter representing an empty string "", regardless the value of the string object on which we call the method, i.e. whether it is an empty string or not.
  3. **testContainsNull()**: must verify that *the contains ()* call with the value "null" generates a **NullPointerException**.

import org.junit.Test;

import static org.junit.Assert.\*;

/\*\*

\* Test class for the contains method of the String class

\*/

public class StringContainsTest {

public void testContainsNormal() {

//...

}

public void testContainsEmpty() {

//...

}

public void testContainsNull() {

//...

}

}