ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

.dit/*
Informatics & Telematics

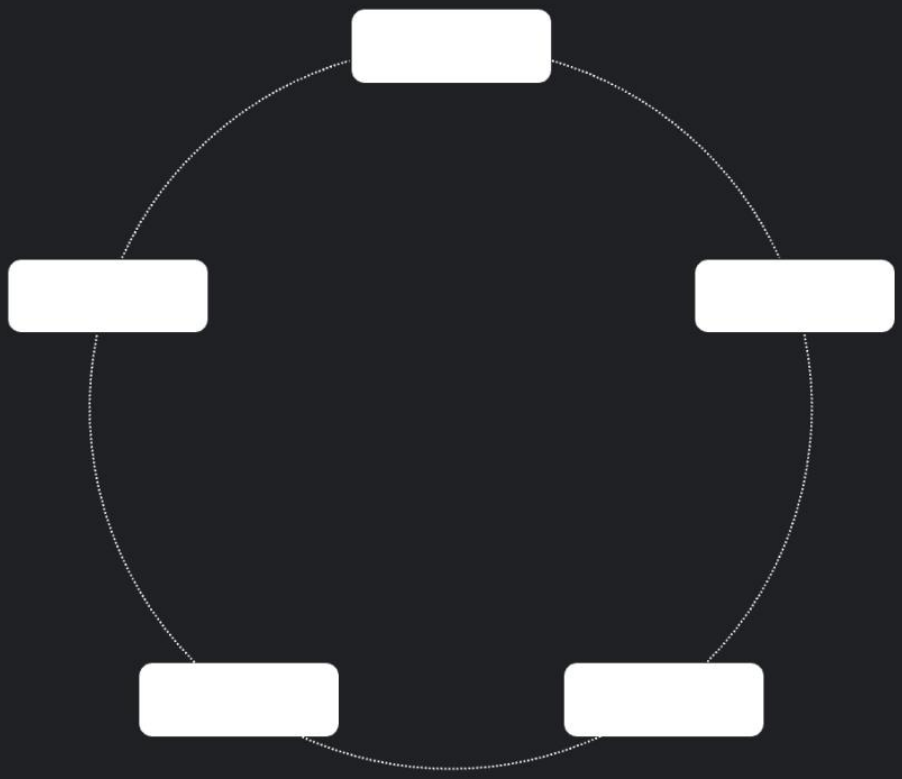
Circular Queue

21st December 2021

PROFESOR:
Dimitrios Michail

CLASS:
Data Structures

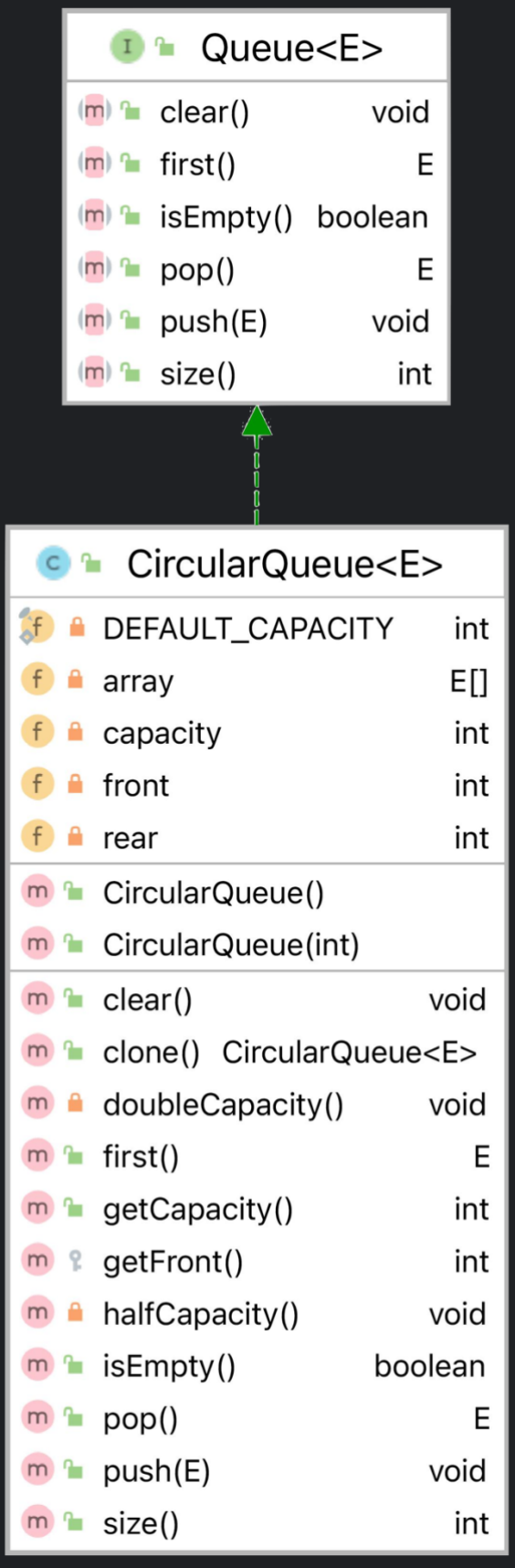
SEMESTER:
3rd



TEAM MEMBERS:
Konstantinos Kokkalis {22047} | Orestis Kritsotakis {22055} |Iasons Tsoukalas {22113}

Table of Contents

Project Structure



Methods

push (E elem)

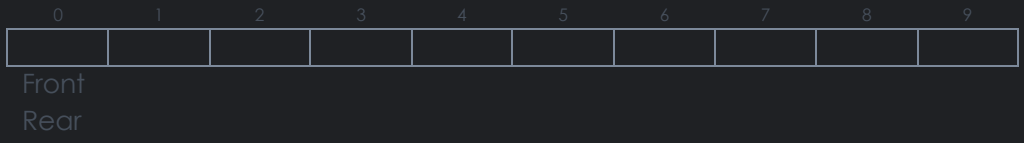
Adds a given element to the circular queue

Applied logic

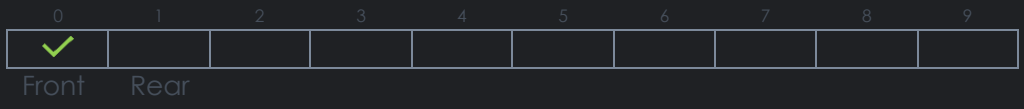
1. Checks if queue is full (up to next-to-last element) and uses doubleCapacity() if true.[1]
2. Copies element to the rear index slot.
3. Updates rear pointer to the next empty slot with modular arithmetic logic[2].

Examples

Regular

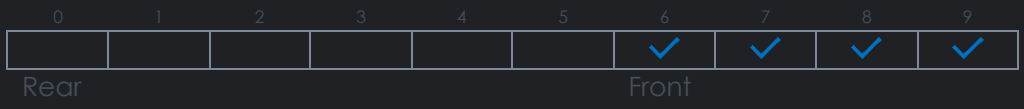


[size: 0, capacity: 10]



[size: 1, capacity: 10]

Wrapped Around



[size: 4, capacity: 10]

[size: 5, capacity: 10]

---

[1] `size() == capacity - 1`
[2] `(rear + 1) % capacity`

pop()

Deletes and returns the first element of the circular queue

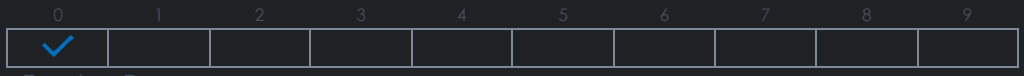
Applied logic

1. Checks if queue is empty[3] (if true throws NoSuchElementException)
2. Saves first element[4] and sets element's slot to null
3. Updates front pointer to the next queue slot using modular arithmetic logic[5].
4. Checks if queue needs downsizing (2 prerequisites):
    a. Slots used are less than 1/4th of the queue capacity[6]
    b. Halved capacity is more than 10[7]

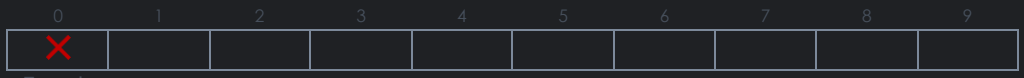
    The 2 prerequisites are used in order to avoid obsolete use of halfCapacity() method which is a high cost process.

Examples

Regular

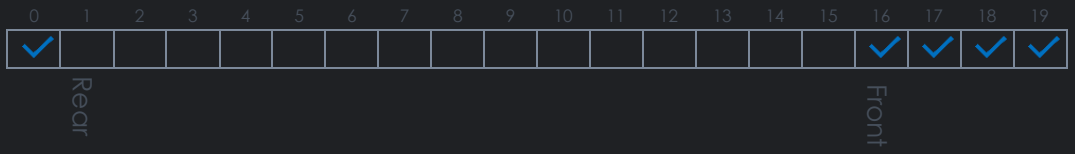


[size: 1, capacity: 10]

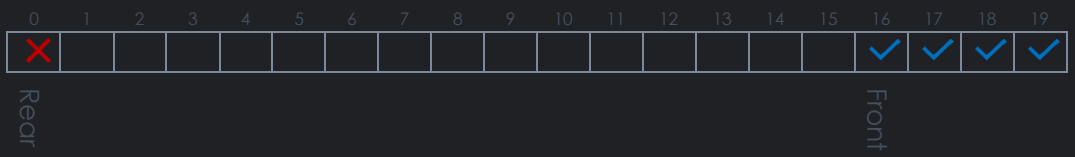


[size: 0, capacity: 10]

Wrapped Around



[size: 5, capacity: 20]



[size: 4, capacity: 20]

[size: 4, capacity: 10]

---

[3] Using isEmpty() method
[4] Using first() method
[5] `(front + 1) % capacity`
[6] `size() < capacity /4`
[7] `capacity/2>=10`

size()

Returns the amount of stored elements in the circular queue, calculated by front and rear indexes, using modular arithmetic logic[8].

Examples

Regular

[capacity: 10]

Returns: (4 - 0 + 10) % 10 = 14 % 10 = 4

Wrapped Around

[capacity: 10]

Returns: (4 - 6 + 10) % 10 = 8 % 10 = 8

[8] `(rear - front + capacity) % capacity;`

clear()

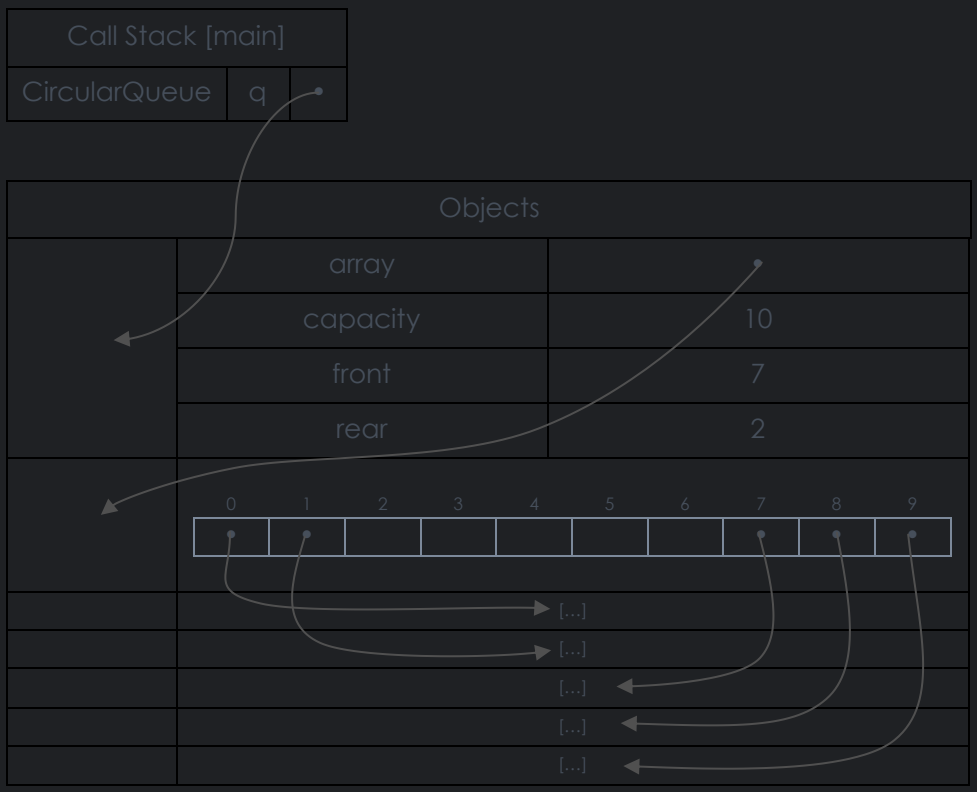
Clears the circular queue and resets rear index.

Applied logic

1. Creates a new empty circular queue of same capacity and connects it to the array attribute of the circular queue. This way, there will be no reference to the old (filled) array so the garbage collector will destroy it. After that process, there will be no reference to the elements from the old array and garbage collector will destroy them as well.
2. Resets rear index to match the front index (indicating an empty queue).

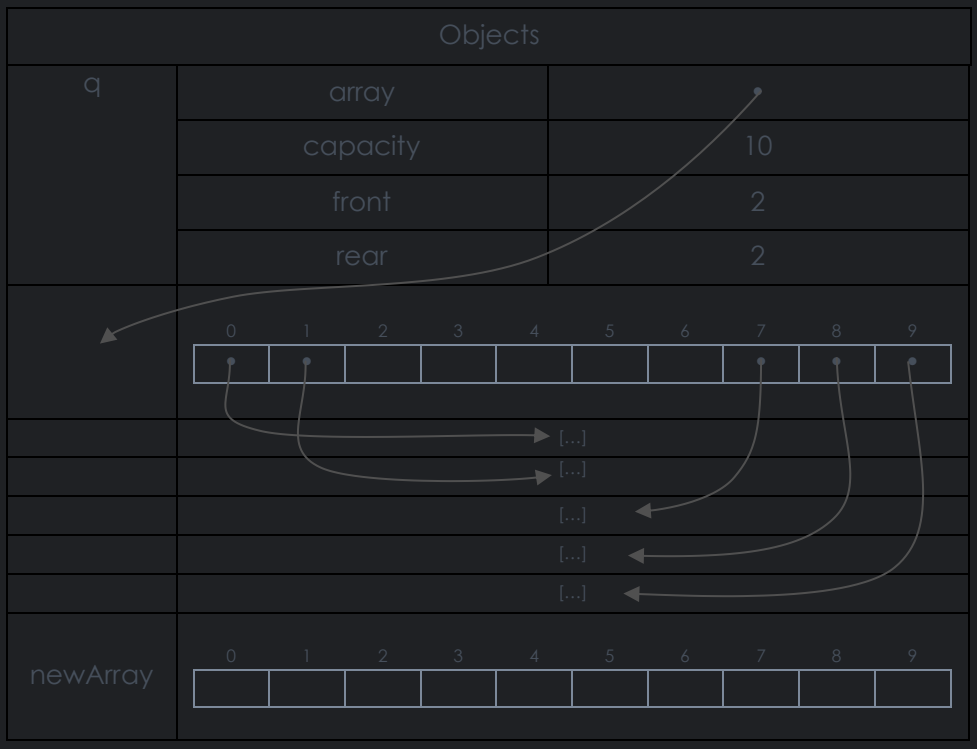
Example

Let's consider a CircularQueue q with 5 elements as following:

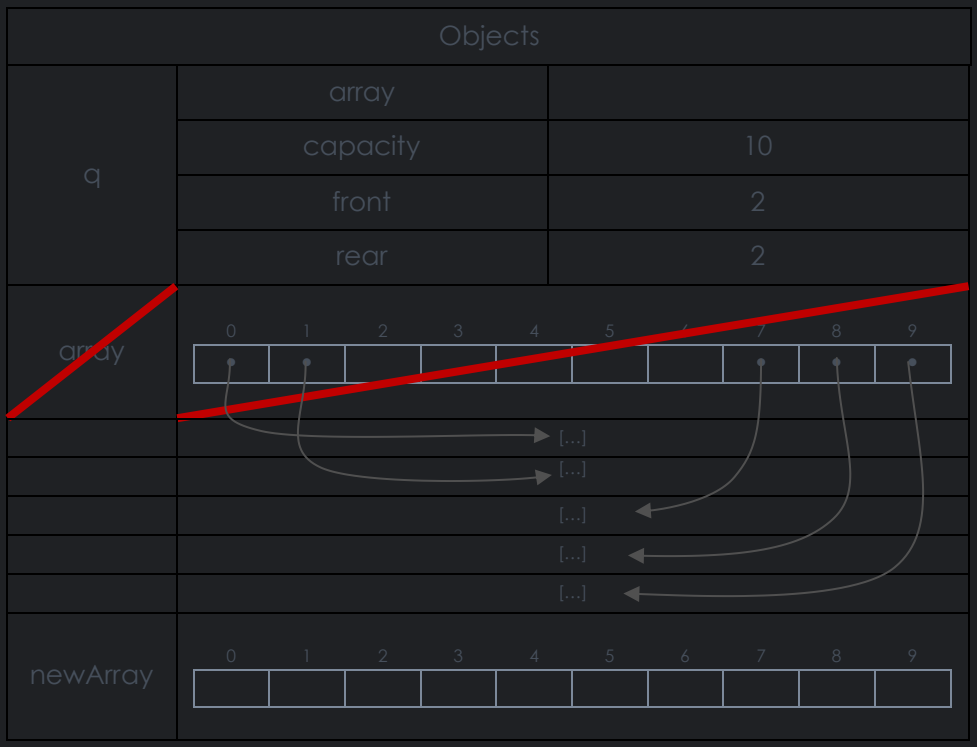


If clear() get called, the process in stages will be:

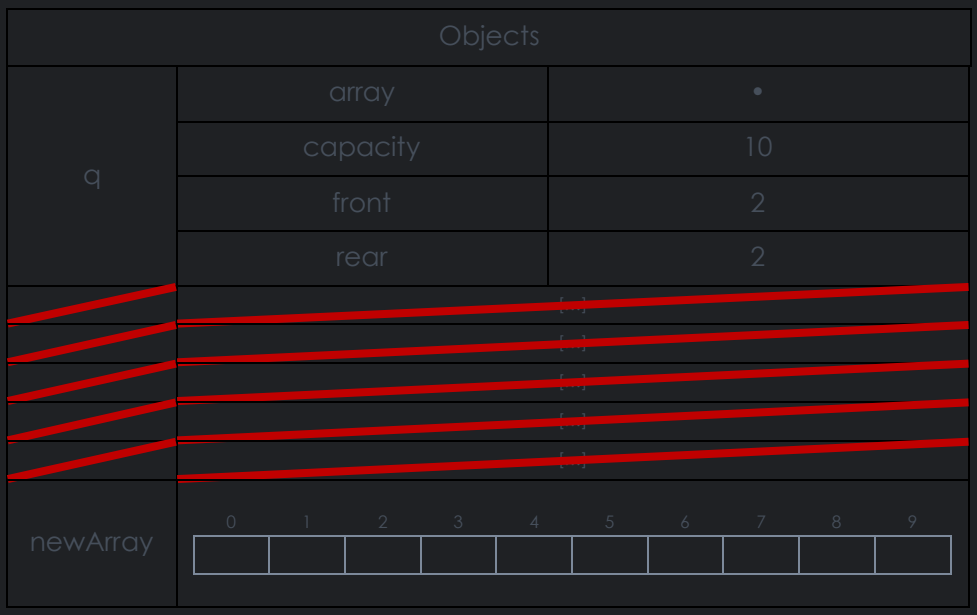
I. Making new array and updating front index



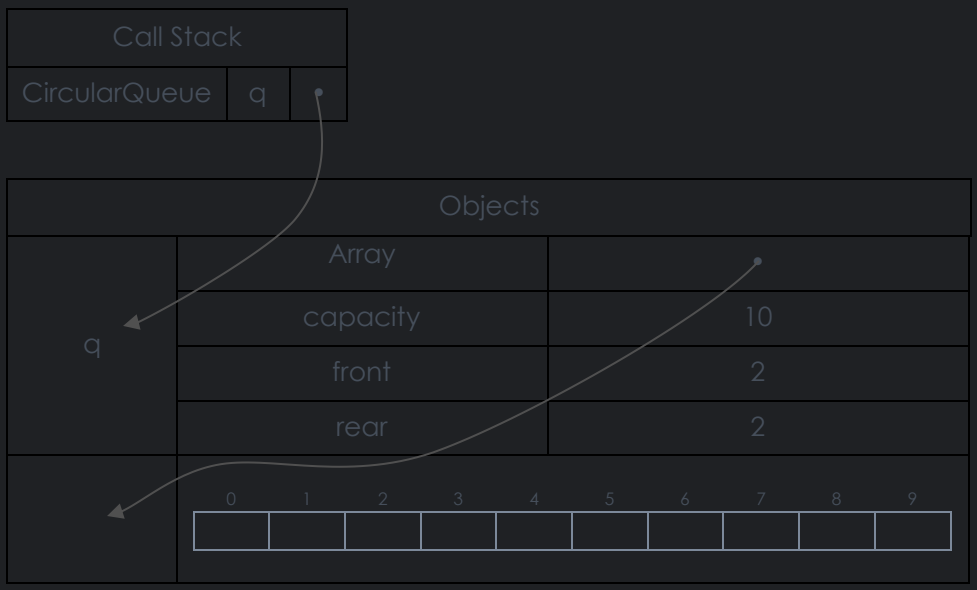
II. Updating q object's array reference



III. Exactly after q object's array reference updated



So when the control will return from clear() the memory will look like this:



isEmpty()

Applied Logic

Returns true if the circular queue is empty or else return false.

The result depends from the equality of front and rear indexes.

first()

Applied Logic

Returns first element using front index.

If circular queue is empty[9] throws NoSuchElementException.

[9] Calculated by isEmpty() method

Tests

Limits and circularity

The goal of this test is to push the structure to the limits. Further than that, we want to test the circular nature of the structure and make sure of the quality on managing the elements.

Stages

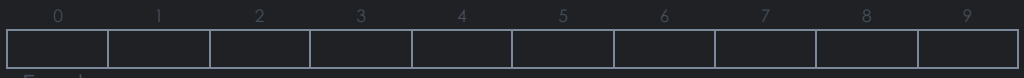
There are the following steps in the test:

1. Fills the queue at 90% of the capacity
2. Pops 60% of the capacity elements
3. Pushes new elements at 60% of the capacity
4. Pushes an element in order to reach 100% of the capacity so the queue will double it self
5. Pops all elements and checks the elements

Example

Stage 1

Let's set a small capacity at 10 elemetns.



Front
Rear
[size: 0, capacity: 10]

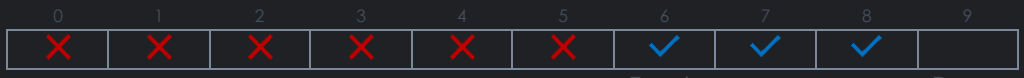
Pushes elements to the queue up to 90% capacity.

Front
Rear

[size: 9, capacity: 10]

Stage 2

Pops 60% of the elements inside the queue.



Front
Rear

[size: 3, capacity: 10]

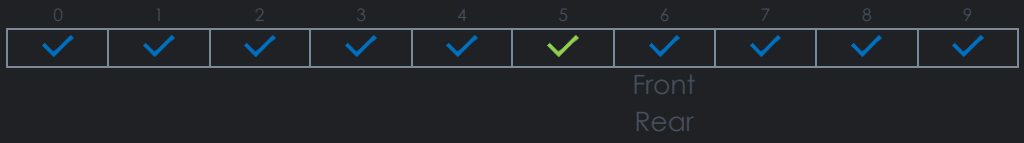
Stage 3

The previous stages achieve to increase the front index so the structure will add elements to the second half. At some point, it will start to add elements at the start of the queue (elements 0 to 4), applying the circular logic of its nature. If we push the same amount of poped elements form the 2nd stage, the queue will look like this:

[size: 9, capacity: 10]

Stage 4

If we push one more element, the queue will automatically double in capacity and order the elements by entry priority. If we take a snapshot exactly after pushing the element, the queue will look like this:



[size: 10, capacity: 10]

After pushing, the doubleCapacity[10] method will be called and return the following queue:

[size: 10, capacity: 20]

[10] `(size() == capacity - 1) == true`

Stage 5

Last but not least, popping the elements, the test, will check[11] the elements with expected value variables, guaranteeing the quality of the structure. The pop method will call the halfCapacity method, every time that the structure uses less than ¼ of its capacity with the lower limit of 10 elements capacity[12]. The proces on phases will look like that:

[size: 5, capacity: 20]

(halfCapacity called)

[size: 4, capacity: 10]

[size: 0, capacity: 10]

---

[11] Using assertEquals method: `assertEquals(next_pop_element++, (int) q.pop());`

[12] `(size() < capacity / 4 && capacity>10) == true`

Multiple elements and clear method test

The goal of this test, is to check the structure with multiple objects and also to run the clear() method.

Stages

There are the following steps in the test:

1. Makes a StringBuilder [input] from hard coded text
2. Pushes every character from the StringBuilder to the queue [q]
3. Makes a hard copy of the queue [q2]
4. Pops all elements and append it to a new StringBuilder [output] and checks if q is empty
5. Check if the tow StringBuilder [input and output] are identical
6. Calls clear() method to q2 and checks if q2 is empty

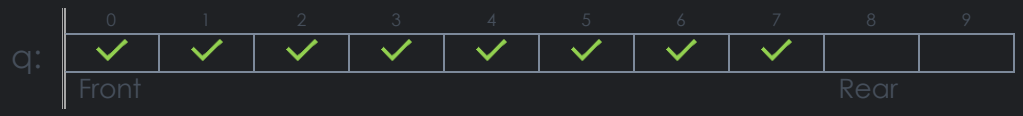
Example

Stage 1

Makes a StringBuilder [input] from hard coded text. For simplisity, lets consider a string builder with 8 chars.

Stage 2

Pushes every character from the StringBuilder to the queue [q]



[size: 8, capacity: 10]

Stage 3

Makes a hard copy of the queue [q2]

Stage 4

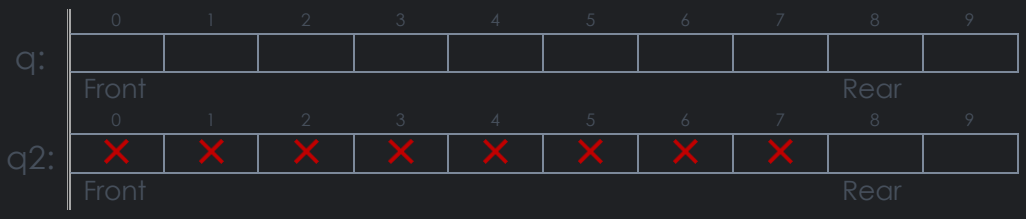
Pops all elements and append it to a new StringBuilder [output] and checks if q is empty

Stage 5

Check if the tow StringBuilder [input and output] are identical

Stage 6

Calls clear() method to q2 and after that checks if q2 is empty



Example conventions

✓ Newly pushed element

✓ Element

✗ Newly poped element