```
import collections
from typing import List

import numpy as np


class BoundedQueue:
    """
        The BoundedQueue class represents or models the buffers within the data flow design of our memory optimization model
        for the stencil operators on FPGA.

        Notes:
            - implementation:               Uses two stacks as underlying data structure to ensure overall complexity of O(1)
                                            for appendleft() and pop().
            - maxsize for bounded queue:    Default behaviour is to remove oldest element of queue, therefore we have to check
                                            it and raise an exception.
            - reference:                    https://docs.python.org/3/library/collections.html#deque-objects
    """

    def __init__(self,
                 name: str,
                 maxsize: int,
                 swap_out: bool = False,
                 collection: List = [],
                 verbose: bool = False) -> None:
        """
        Create new BoundedQueue with given initialization parameters.
        :param name: name of the queue
        :param maxsize: maximum number of elements the queue can hold at a time
        :param swap_out: set whether or not the buffer is swapped out (might get overridden by the optimizer)
        :param collection: initial data in queue
        :param verbose: flag for console output logging
        """
        # save params
        self.maxsize: int = maxsize if maxsize > 0 else 1  # maxsize must be at least 1 to correctly forward data
        self.name: str = name
        # create queue
        self.queue: collection.dequeue = collections.deque(collection, self.maxsize)
        # init current size
        self.current_size: int = len(collection)
        # indication of where the buffer is located (slow memory or fast memory)
        self.swap_out = swap_out
        # flag for verbose console output
        self.verbose = verbose

    def __repr__(self):
        # override default implementation to return nice output e.g. if a collection of queue is being printed
        return str(self)
```

```
def __str__(self):
    # return an useful and human readable info string from the queue
    return "BoundedQueue: {}, current size: {}, max size: {}".format(self.name, self.current_size, self.maxsize)

def import_data(self, data):
    """
    Add data elements to queue.
    :param data: initial data in queue
    :return: nothing
    """
    if self.maxsize < len(data):
        raise RuntimeError("max size of queue ({}) is smaller than the data collection size ({})"
                           .format(self.maxsize, len(data)))
    else:
        self.queue: collections.deque = collections.deque(data, self.maxsize)
        self.current_size = len(data)

def export_data(self):
    """
    Return the current content of the hole queue.
    :return: numpy data array
    """
    return np.array(self.queue)[::-1]

def try_peek_last(self):
    """
    Return last data element (next element that gets dequeued) without removing it from the queue.
    :return: last data element on success, False otherwise
    """
    if self.current_size > 0:  # check bound
        return self.queue[self.current_size-1]
    else:
        return False

def size(self) -> int:
    """
    Get number of data items the queue currently contains.
    :return: current queue size
    """
    return self.current_size

def is_empty(self) -> bool:
    """
    Test if queue is empty.
    :return: if queue is empty
    """
    return self.size() == 0

def is_full(self) -> bool:
```

```
    """
    Test if queue is full.
    :return: if queue is full
    """
    return self.size() == self.maxsize

def enqueue(self, item) -> None:
    """
    Add data element to queue, causes an exception if queue is full.
    :param item: data element
    :return: None
    """
    if self.current_size >= self.maxsize:  # check bound
        raise RuntimeError("buffer {} overflow occurred".format(self.name))
    # add a new item to the left side
    self.queue.appendleft(item)
    # adjust counter
    self.current_size += 1

def dequeue(self):
    """
    Remove and return data element from queue, causes an exception if queue is empty.
    :return: data element
    """
    if self.current_size > 0:  # check bound
        # adjust size
        self.current_size -= 1
        # return and remove the rightmost item
        return self.queue.pop()
    else:
        raise RuntimeError("buffer {} underflow occurred".format(self.name))

def try_enqueue(self, item) -> bool:
    """
    Add data element to queue..
    :param item: data item
    :return: True: successful, False: unsuccessful
    """
    # check bound, do not raise exception in case of an overflow
    if self.current_size >= self.maxsize:
        # report: unsuccessful
        return False
    # add a new item to the left side
    self.queue.appendleft(item)
    # adjust counter
    self.current_size += 1
    # report: successful
    return True
```

```
    def try_dequeue(self):
        """
        Remove and return data item from queue.
        :return: data item: successful, False: unsuccessful
        """
        # check bound, do not raise exception in case of an underflow
        if self.current_size > 0:
            # adjust size
            self.current_size -= 1
            # return and remove the rightmost item
            return self.queue.pop()
        else:
            # report: unsuccessful
            return False

    def peek(self, index: int):
        """
        Returns data item at position 'index' without removal, causes an exception if index > BoundedQueue.current_size
        :param index: queue position of peeking element
        :return: data item
        """
        # check bound
        if self.current_size <= index:
            raise RuntimeError("buffer {} index out of bound access occurred".format(self.name))
        else:
            return self.queue[index]


if __name__ == "__main__":
    """
        Simple debugging example
    """
    # create dummy queue
    queue = BoundedQueue(name="debug",
                         maxsize=5,
                         collection=[1, 2, 3, 4, 5])
    # do some basic function calls and check if it crashes
    try:
        print("Enqueue element into full queue, should throw an exception.")
        queue.enqueue(6)
        print("Peek element at pos=3, value is: " + str(queue.peek(3)))
    except Exception as ex:
        print("Exception has been thrown.\n{}".format(ex.__traceback__))
```