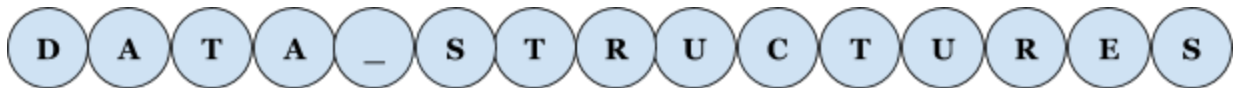

CS261 Data Structures

Assignment 2

v 1.06 (revised 1/20/2021)

Your Very Own Dynamic Array (plus Bag, Stack, and Queue)



```
da = DynamicArray(list("DATA"))
```



```
self.size = 4  
self.capacity = 4  
self.data = ['D', 'A', 'T', 'A']
```

```
da = DynamicArray(list("STRUCTURES"))
```



```
self.size = 10  
self.capacity = 16  
self.data = ['S', 'T', 'R', 'U', 'C', 'T', 'U', 'R', 'E', 'S', None, None, None, None, None, None]
```

Contents

General Instructions	3
 Part 1 - Dynamic Array Implementation	
Summary and Specific Instructions	4
resize()	5
append()	6
insert_at_index()	7
remove_at_index()	9
slice()	12
merge()	13
map()	14
filter()	15
reduce()	16
 Part 2 - Bag ADT Implementation	
Summary and Specific Instructions	17
add()	18
remove()	18
count()	19
clear()	19
equal()	20
 Part 3 - Stack ADT Implementation	
Summary and Specific Instructions	21
push()	22
pop()	22
top()	23
 Part 4 - Queue ADT Implementation	
Summary and Specific Instructions	24
enqueue()	25
dequeue()	25

General Instructions

1. Programs in this assignment must be written in Python v3 and submitted to Gradescope before the due date specified in the syllabus. You may resubmit your code as many times as necessary. Gradescope allows you to choose which submission will be graded.
2. In Gradescope, your code will run through several tests. Any failed tests will provide a brief explanation of testing conditions to help you with troubleshooting. Your goal is to pass all tests.
3. We encourage you to create your own test programs and cases even though this work won't have to be submitted and won't be graded. Gradescope tests are limited in scope and may not cover all edge cases. Your submission must work on all valid inputs. We reserve the right to test your submission with more tests than Gradescope.
4. Your code must have an appropriate level of comments. At a minimum, each method should have a descriptive docstring. Additionally, put comments throughout the code to make it easy to follow and understand.
5. You will be provided with a starter "skeleton" code, on which you will build your implementation. Methods defined in skeleton code must retain their names and input/output parameters. Variables defined in skeleton code must also retain their names. We will only test your solution by making calls to methods defined in the skeleton code and by checking values of variables defined in the skeleton code. You can add more methods and variables, as needed.
However, certain classes and methods cannot be changed in any way. Please see comments in the skeleton code for guidance. In particular, content of any methods pre-written for you as part of the skeleton code must not be changed.
6. Both the skeleton code and code examples provided in this document are part of assignment requirements. They have been carefully selected to demonstrate requirements for each method. Refer to them for the detailed description of expected method behavior, input/output parameters, and handling of edge cases. Code examples may include assignment requirements not explicitly stated elsewhere.
7. For each method, you can choose to implement a recursive or iterative solution. When using a recursive solution, be aware of maximum recursion depths on large inputs. We will specify the maximum input size that your solution must handle.

Part 1 - Summary and Specific Instructions

1. Implement a Dynamic Array class by completing the provided skeleton code in the file `dynamic_array.py`. The `DynamicArray` class will use a `StaticArray` object as its underlying data storage container and will provide many methods similar to those we are used to when working with Python lists. Once completed, your implementation will include the following methods:

```
resize()
append()
insert_at_index()
remove_at_index()
slice()
merge()
map()
filter()
reduce()
```

* Several class methods, like `is_empty()`, `length()`, `get_at_index()`, and `set_at_index()` have been pre-written for you.

2. We will test your implementation with different types of objects, not just integers. We guarantee that all such objects will have correct implementation of methods `__eq__`, `__lt__`, `__gt__`, `__ge__`, `__le__`, and `__str__`.
3. The number of objects stored in the array at any given time will be between 0 and 100,000 inclusive. Array must allow for storage of duplicate objects.
4. Variables in the `DynamicArray` class are not marked as private. For this portion of the assignment, you are allowed to access and change their values directly. You are not required to write getter or setter methods for them.
5. **RESTRICTIONS:** You are not allowed to use ANY built-in Python data structures and their methods (lists, dictionaries, etc.). You must solve this portion of the assignment by importing and using objects of the `StaticArray` class (prewritten for you) and using class methods to write your solution.
You are also not allowed to directly access any variables of the `StaticArray` class (like `self.data._data[]`). All work must be done by only using `StaticArray` class methods.

resize(self, new_capacity: int) -> None:

This method changes the capacity of the underlying storage for the dynamic array elements. It does not change values or order of any elements currently stored in the dynamic array.

It is intended to be an “internal” method of the Dynamic Array class, called by other class methods such as `append()`, `remove_at_index()`, `insert_at_index()`, to manage the capacity of the underlying storage data structure.

The method should only accept positive integers for `new_capacity`. Additionally, `new_capacity` cannot be smaller than the number of elements currently stored in the dynamic array (which is tracked by the `self.size` variable). If `new_capacity` is not a positive integer or if `new_capacity < self.size`, this method should not do any work and just exit.

Example #1:

```
da = DynamicArray()
print(da.size, da.capacity, da.data)
da.resize(8)
print(da.size, da.capacity, da.data)
da.resize(2)
print(da.size, da.capacity, da.data)
da.resize(0)
print(da.size, da.capacity, da.data)
```

Output:

```
0 4 STAT_ARR Size: 4 [None, None, None, None]
0 8 STAT_ARR Size: 8 [None, None, None, None, None, None, None, None]
0 2 STAT_ARR Size: 2 [None, None]
0 2 STAT_ARR Size: 2 [None, None]
```

NOTE: Example 2 below will not work properly until the `append()` method is implemented.

Example #2:

```
da = DynamicArray([1, 2, 3, 4, 5, 6, 7, 8])
print(da)
da.resize(20)
print(da)
da.resize(4)
print(da)
```

Output:

```
DYN_ARR Size/Cap: 8/8 [1, 2, 3, 4, 5, 6, 7, 8]
DYN_ARR Size/Cap: 8/20 [1, 2, 3, 4, 5, 6, 7, 8]
DYN_ARR Size/Cap: 8/20 [1, 2, 3, 4, 5, 6, 7, 8]
```

append(self, value: object) -> None:

This method adds a new value at the end of the dynamic array.

If internal storage associated with a dynamic array is already full, you need to DOUBLE its capacity before adding a new value.

Example #1:

```
da = DynamicArray()
print(da.size, da.capacity, da.data)
da.append(1)
print(da.size, da.capacity, da.data)
print(da)
```

Output:

```
0 4 STAT_ARR Size: 4 [None, None, None, None]
1 4 STAT_ARR Size: 4 [1, None, None, None]
DYN_ARR Size/Cap: 1/4 [1]
```

Example #2:

```
da = DynamicArray()
for i in range(9):
    da.append(i + 101)
print(da)
```

Output:

```
DYN_ARR Size/Cap: 1/4 [101]
DYN_ARR Size/Cap: 2/4 [101, 102]
DYN_ARR Size/Cap: 3/4 [101, 102, 103]
DYN_ARR Size/Cap: 4/4 [101, 102, 103, 104]
DYN_ARR Size/Cap: 5/8 [101, 102, 103, 104, 105]
DYN_ARR Size/Cap: 6/8 [101, 102, 103, 104, 105, 106]
DYN_ARR Size/Cap: 7/8 [101, 102, 103, 104, 105, 106, 107]
DYN_ARR Size/Cap: 8/8 [101, 102, 103, 104, 105, 106, 107, 108]
DYN_ARR Size/Cap: 9/16 [101, 102, 103, 104, 105, 106, 107, 108, 109]
```

Example #3:

```
da = DynamicArray()
for i in range(600):
    da.append(i)
print(da.size)
print(da.capacity)
```

Output:

```
600
1024
```

insert_at_index(self, index: int, value: object) -> None:

This method adds a new value at the specified index position in the dynamic array. Index 0 refers to the beginning of the array. If the provided index is invalid, the method raises a custom "DynamicArrayException". Code for the exception is provided in the skeleton file. If the array contains N elements, valid indices for this method are [0, N] inclusive. Here, "indices" is referring to the method's index parameter.

If internal storage associated with the dynamic array is already full, you need to DOUBLE its capacity before adding a new value.

Example #1:

```
da = DynamicArray([100])
print(da)
da.insert_at_index(0, 200)
da.insert_at_index(0, 300)
da.insert_at_index(0, 400)
print(da)
da.insert_at_index(3, 500)
print(da)
da.insert_at_index(1, 600)
print(da)
```

Output:

```
DYN_ARR Size/Cap: 1/4 [100]
DYN_ARR Size/Cap: 4/4 [400, 300, 200, 100]
DYN_ARR Size/Cap: 5/8 [400, 300, 200, 500, 100]
DYN_ARR Size/Cap: 6/8 [400, 600, 300, 200, 500, 100]
```

Example #2:

```
da = DynamicArray()
try:
    da.insert_at_index(-1, 100)
except Exception as e:
    print("Exception raised:", type(e))
da.insert_at_index(0, 200)
try:
    da.insert_at_index(2, 300)
except Exception as e:
    print("Exception raised:", type(e))
print(da)
```

Output:

```
Exception raised: <class '__main__.DynamicArrayException'>
Exception raised: <class '__main__.DynamicArrayException'>
DYN_ARR Size/Cap: 1/4 [200]
```

Example #3:

```
da = DynamicArray()
for i in range(1, 10):
    index, value = i - 4, i * 10
    try:
        da.insert_at_index(index, value)
    except Exception as e:
        print("Cannot insert value", value, "at index", index)
print(da)
```

Output:

```
Cannot insert value 10 at index -3
Cannot insert value 20 at index -2
Cannot insert value 30 at index -1
DYN_ARR Size/Cap: 6/8 [40, 50, 60, 70, 80, 90]
```


remove_at_index(self, index: int) -> None:

This method removes the element at the specified index position from the dynamic array . Index 0 refers to the beginning of the array. If the provided index is invalid, the method raises a custom "DynamicArrayException". Code for the exception is provided in the skeleton file. If the array contains N elements, valid indices for this method are [0, N - 1] inclusive. Here, "indices" is referring to the method's index parameter.

When the number of elements stored in the array (before removal) is STRICTLY LESS than $\frac{1}{4}$ of its current capacity, the capacity must be reduced to TWICE the number of current elements. This check/capacity adjustment must happen BEFORE removal of the element.

If the current capacity (before reduction) is 10 elements or less, reduction should not happen at all. If the current capacity (before reduction) is greater than 10 elements, reduced capacity cannot become less than 10 elements. Please see examples below, especially example #3, for clarifications.

Example #1:

```
da = DynamicArray([10, 20, 30, 40, 50, 60, 70, 80])
print(da)
da.remove_at_index(0)
print(da)
da.remove_at_index(6)
print(da)
da.remove_at_index(2)
print(da)
```

Output:

```
DYN_ARR Size/Cap: 8/8 [10, 20, 30, 40, 50, 60, 70, 80]
DYN_ARR Size/Cap: 7/8 [20, 30, 40, 50, 60, 70, 80]
DYN_ARR Size/Cap: 6/8 [20, 30, 40, 50, 60, 70]
DYN_ARR Size/Cap: 5/8 [20, 30, 50, 60, 70]
```

Example #2:

```
da = DynamicArray([1024])
print(da)
for i in range(17):
    da.insert_at_index(i, i)
print(da.size, da.capacity)
for i in range(16, -1, -1):
    da.remove_at_index(0)
print(da)
```

Output:

```
DYN_ARR Size/Cap: 1/4 [1024]
18 32
DYN_ARR Size/Cap: 1/10 [1024]
```

Example #3:

```

da = DynamicArray()
print(da.size, da.capacity)
[da.append(1) for i in range(100)]          # step 1 - add 100 elements
print(da.size, da.capacity)
[da.remove_at_index(0) for i in range(68)]  # step 2 - remove 68 elements
print(da.size, da.capacity)
da.remove_at_index(0)                      # step 3 - remove 1 element
print(da.size, da.capacity)
da.remove_at_index(0)                      # step 4 - remove 1 element
print(da.size, da.capacity)
[da.remove_at_index(0) for i in range(14)]  # step 5 - remove 14 elements
print(da.size, da.capacity)
da.remove_at_index(0)                      # step 6 - remove 1 element
print(da.size, da.capacity)
da.remove_at_index(0)                      # step 7 - remove 1 element
print(da.size, da.capacity)

for i in range(14):
    print("Before remove_at_index(): ", da.size, da.capacity, end="")
    da.remove_at_index(0)
    print(" After remove_at_index(): ", da.size, da.capacity)

```

Output:

```

0 4
100 128
32 128
31 128
30 62
16 62
15 62
14 30
Before remove_at_index(): 14 30 After remove_at_index(): 13 30
Before remove_at_index(): 13 30 After remove_at_index(): 12 30
Before remove_at_index(): 12 30 After remove_at_index(): 11 30
Before remove_at_index(): 11 30 After remove_at_index(): 10 30
Before remove_at_index(): 10 30 After remove_at_index(): 9 30
Before remove_at_index(): 9 30 After remove_at_index(): 8 30
Before remove_at_index(): 8 30 After remove_at_index(): 7 30
Before remove_at_index(): 7 30 After remove_at_index(): 6 14
Before remove_at_index(): 6 14 After remove_at_index(): 5 14
Before remove_at_index(): 5 14 After remove_at_index(): 4 14
Before remove_at_index(): 4 14 After remove_at_index(): 3 14
Before remove_at_index(): 3 14 After remove_at_index(): 2 10
Before remove_at_index(): 2 10 After remove_at_index(): 1 10
Before remove_at_index(): 1 10 After remove_at_index(): 0 10

```

Example #4:

```
da = DynamicArray([1, 2, 3, 4, 5])
print(da)
for _ in range(5):
    da.remove_at_index(0)
    print(da)
```

Output:

```
DYN_ARR Size/Cap: 5/8 [1, 2, 3, 4, 5]
DYN_ARR Size/Cap: 4/8 [2, 3, 4, 5]
DYN_ARR Size/Cap: 3/8 [3, 4, 5]
DYN_ARR Size/Cap: 2/8 [4, 5]
DYN_ARR Size/Cap: 1/8 [5]
DYN_ARR Size/Cap: 0/8 []
```

slice(self, start_index: int, size: int) -> object:

This method returns a new Dynamic Array object that contains the requested number of elements from the original array starting with the element located at the requested start index. If the array contains N elements, a valid `start_index` value is in range `[0, N - 1]` inclusive. A valid `size` value is a non-negative integer.

If the provided start index or size is invalid, or if there are not enough elements between start index and the end of the array to make a slice of the requested size, this method raises a custom "DynamicArrayException". Code for the exception is provided in the skeleton file.

Example #1:

```
da = DynamicArray([1, 2, 3, 4, 5, 6, 7, 8, 9])
da_slice = da.slice(1, 3)
print(da, da_slice, sep="\n")
da_slice.remove_at_index(0)
print(da, da_slice, sep="\n")
```

Output:

```
DYN_ARR Size/Cap: 9/16 [1, 2, 3, 4, 5, 6, 7, 8, 9]
DYN_ARR Size/Cap: 3/4 [2, 3, 4]
DYN_ARR Size/Cap: 9/16 [1, 2, 3, 4, 5, 6, 7, 8, 9]
DYN_ARR Size/Cap: 2/4 [3, 4]
```

Example #2:

```
da = DynamicArray([10, 11, 12, 13, 14, 15, 16])
print("SOURCE:", da)
slices = [(0, 7), (-1, 7), (0, 8), (2, 3), (5, 0), (5, 3), (6, 1), (6, -1)]
for i, cnt in slices:
    print("Slice", i, "/", cnt, end="")
    try:
        print(" --- OK: ", da.slice(i, cnt))
    except:
        print(" --- exception occurred.")
```

Output:

```
SOURCE: DYN_ARR Size/Cap: 7/8 [10, 11, 12, 13, 14, 15, 16]
Slice 0 / 7 --- OK: DYN_ARR Size/Cap: 7/8 [10, 11, 12, 13, 14, 15, 16]
Slice -1 / 7 --- exception occurred.
Slice 0 / 8 --- exception occurred.
Slice 2 / 3 --- OK: DYN_ARR Size/Cap: 3/4 [12, 13, 14]
Slice 5 / 0 --- OK: DYN_ARR Size/Cap: 0/4 []
Slice 5 / 3 --- exception occurred.
Slice 6 / 1 --- OK: DYN_ARR Size/Cap: 1/4 [16]
Slice 6 / -1 --- exception occurred.
```

merge(self, second_da: object) -> None:

This method takes another Dynamic Array object as a parameter and appends all elements from the second array to the current one, in the same order as they are stored in the second array.

Example #1:

```
da = DynamicArray([1, 2, 3, 4, 5])
da2 = DynamicArray([10, 11, 12, 13])
print(da)
da.merge(da2)
print(da)
```

Output:

```
DYN_ARR Size/Cap: 5/8 [1, 2, 3, 4, 5]
DYN_ARR Size/Cap: 9/16 [1, 2, 3, 4, 5, 10, 11, 12, 13]
```

Example #2:

```
da = DynamicArray([1, 2, 3])
da2 = DynamicArray()
da3 = DynamicArray()
da.merge(da2)
print(da)
da2.merge(da3)
print(da2)
da3.merge(da)
print(da3)
```

Output:

```
DYN_ARR Size/Cap: 3/4 [1, 2, 3]
DYN_ARR Size/Cap: 0/4 []
DYN_ARR Size/Cap: 3/4 [1, 2, 3]
```

map(self, map_func) -> object:

This method creates a new Dynamic Array where the value of each element is derived by applying a given map_func to the corresponding value from the original array.

It works similarly to the built-in Python map() function. If you would like to review how Python's map() works, the following are good resources:

<https://docs.python.org/3/library/functions.html#map>

<https://www.geeksforgeeks.org/python-map-function/>

Example #1:

```
da = DynamicArray([1, 5, 10, 15, 20, 25])
print(da)
print(da.map(lambda x: (x ** 2)))
```

Output:

```
DYN_ARR Size/Cap: 6/8 [1, 5, 10, 15, 20, 25]
DYN_ARR Size/Cap: 6/8 [1, 25, 100, 225, 400, 625]
```

Example #2:

```
def double(value):
    return value * 2

def square(value):
    return value ** 2

def cube(value):
    return value ** 3

def plus_one(value):
    return value + 1

da = DynamicArray([plus_one, double, square, cube])
for value in [1, 10, 20]:
    print(da.map(lambda x: x(value)))
```

Output:

```
DYN_ARR Size/Cap: 4/4 [2, 2, 1, 1]
DYN_ARR Size/Cap: 4/4 [11, 20, 100, 1000]
DYN_ARR Size/Cap: 4/4 [21, 40, 400, 8000]
```

filter(self, filter_func) ->object:

This method creates a new Dynamic Array populated only with those elements from the original array for which filter_func returns True.

It works similarly to the built-in Python filter() function. If you would like to review how Python's filter() works, the following are good resources:

<https://docs.python.org/3/library/functions.html#filter>

<https://www.geeksforgeeks.org/filter-in-python/>

Example #1:

```
def filter_a(e):
    return e > 10

da = DynamicArray([1, 5, 10, 15, 20, 25])
print(da)
result = da.filter(filter_a)
print(result)
print(da.filter(lambda x: (10 <= x <= 20)))
```

Output:

```
DYN_ARR Size/Cap: 6/8 [1, 5, 10, 15, 20, 25]
DYN_ARR Size/Cap: 3/4 [15, 20, 25]
DYN_ARR Size/Cap: 3/4 [10, 15, 20]
```

Example #2:

```
def is_long_word(word, length):
    return len(word) > length

da = DynamicArray("This is a sentence with some long words".split())
print(da)
for length in [3, 4, 7]:
    print(da.filter(lambda word: is_long_word(word, length)))
```

Output:

```
DYN_ARR Size/Cap: 8/8 [This, is, a, sentence, with, some, long, words]
DYN_ARR Size/Cap: 6/8 [This, sentence, with, some, long, words]
DYN_ARR Size/Cap: 2/4 [sentence, words]
DYN_ARR Size/Cap: 1/4 [sentence]
```

reduce(self, reduce_func, initializer=None) ->object:

This method sequentially applies the `reduce_func` to all elements of the Dynamic Array and returns the resulting value. It accepts an optional `initializer` parameter. If this parameter is not provided, the first value in the array is used as the initializer. If the Dynamic Array is empty, the method returns the value of the initializer (or `None`, if it was not provided).

This method works similarly to the Python `reduce()` function. If you would like to review how Python's `reduce()` works, the following is a good starting point:

<https://www.geeksforgeeks.org/reduce-in-python/>

Example #1:

```
values = [100, 5, 10, 15, 20, 25]
da = DynamicArray(values)
print(da)
print(da.reduce(lambda x, y: x + y ** 2))
print(da.reduce(lambda x, y: x + y ** 2, -1))
```

Output:

```
DYN_ARR Size/Cap: 6/8 [100, 5, 10, 15, 20, 25]
1475
11374
```

Explanation:

$$1475 = 100 + 5^2 + 10^2 + 15^2 + 20^2 + 25^2$$

First value is not squared because it is used as an initializer.

$$11374 = -1 + 100^2 + 5^2 + 10^2 + 15^2 + 20^2 + 25^2$$

Example #2:

```
da = DynamicArray([100])
print(da.reduce(lambda x, y: x + y ** 2))
print(da.reduce(lambda x, y: x + y ** 2, -1))
da.remove_at_index(0)
print(da.reduce(lambda x, y: x + y ** 2))
print(da.reduce(lambda x, y: x + y ** 2, -1))
```

Output:

```
100
9999
None
-1
```


Part 2 - Summary and Specific Instructions

1. Implement a Bag ADT class by completing the provided skeleton code in the file `bag_da.py`. You will use the Dynamic Array data structure that you implemented in part 1 of this assignment as the underlying data storage for your Bag ADT.
2. Once completed, your implementation will include the following methods:

```
add()  
remove()  
count()  
clear()  
equal()
```

3. We will test your implementation with different types of objects, not just integers. We guarantee that all such objects will have correct implementation of methods `__eq__`, `__lt__`, `__gt__`, `__ge__`, `__le__`, and `__str__`.
4. The number of objects stored in the Bag at any given time will be between 0 and 100,000 inclusive. Bag must allow for storage of duplicate objects.
5. RESTRICTIONS: You are not allowed to use ANY built-in Python data structures and their methods. You must solve this portion of the assignment by importing the `DynamicArray` class that you wrote in part 1 and using class methods to write your solution.
You are also not allowed to directly access any variables of the `DynamicArray` class (like `self.size`, `self.capacity` and `self.data` in part 1). All work must be done by only using class methods.

add(self, value: object) -> None:

This method adds a new element to the bag.

Example #1:

```
bag = Bag()
print(bag)
values = [10, 20, 30, 10, 20, 30]
for value in values:
    bag.add(value)
print(bag)
```

Output:

```
BAG: 0 elements. []
BAG: 6 elements. [10, 20, 30, 10, 20, 30]
```

remove(self, value: object) -> bool:

This method removes any one element from the bag that matches the provided “value” object. Method returns True if some object was actually removed from the bag. Otherwise, it returns False.

Example #1:

```
bag = Bag([1, 2, 3, 1, 2, 3, 1, 2, 3])
print(bag)
print(bag.remove(7), bag)
print(bag.remove(3), bag)
print(bag.remove(3), bag)
print(bag.remove(3), bag)
print(bag.remove(3), bag)
```

Output:

```
BAG: 9 elements. [1, 2, 3, 1, 2, 3, 1, 2, 3]
False BAG: 9 elements. [1, 2, 3, 1, 2, 3, 1, 2, 3]
True BAG: 8 elements. [1, 2, 1, 2, 3, 1, 2, 3]
True BAG: 7 elements. [1, 2, 1, 2, 1, 2, 3]
True BAG: 6 elements. [1, 2, 1, 2, 1, 2]
False BAG: 6 elements. [1, 2, 1, 2, 1, 2]
```

count(self, value: object) -> int:

This method counts the number of elements in the bag that match the provided "value" object.

Example #1:

```
bag = Bag([1, 2, 3, 1, 2, 2])
print(bag, bag.count(1), bag.count(2), bag.count(3), bag.count(4))
```

Output:

```
BAG: 6 elements. [1, 2, 3, 1, 2, 2] 2 3 1 0
```

clear(self) -> None:

This method clears the content of the bag.

Example #1:

```
bag = Bag([1, 2, 3, 1, 2, 3])
print(bag)
bag.clear()
print(bag)
```

Output:

```
BAG: 6 elements. [1, 2, 3, 1, 2, 3]
BAG: 0 elements. []
```

equal(self, second_bag: object) -> bool:

This method compares the content of the bag with the content of the second bag provided by the user. Method returns True if the bags are equal (have the same number of elements and contain the same elements without regards to the order of elements). Otherwise, it returns False.

Empty bag is only considered equal to another empty bag. This method should not change the contents of either bag.

Example #1:

```
bag1 = Bag([10, 20, 30, 40, 50, 60])
bag2 = Bag([60, 50, 40, 30, 20, 10])
bag3 = Bag([10, 20, 30, 40, 50])
bag_empty = Bag()

print(bag1, bag2, bag3, bag_empty, sep="\n")
print(bag1.equal(bag2), bag2.equal(bag1))
print(bag1.equal(bag3), bag3.equal(bag1))
print(bag2.equal(bag3), bag3.equal(bag2))
print(bag1.equal(bag_empty), bag_empty.equal(bag1))
print(bag_empty.equal(bag_empty))
print(bag1, bag2, bag3, bag_empty, sep="\n")

bag1 = Bag([100, 200, 300, 200])
bag2 = Bag([100, 200, 30, 100])
print(bag1.equal(bag2))
```

Output:

```
BAG: 6 elements. [10, 20, 30, 40, 50, 60]
BAG: 6 elements. [60, 50, 40, 30, 20, 10]
BAG: 5 elements. [10, 20, 30, 40, 50]
BAG: 0 elements. []
True True
False False
False False
False False
True
BAG: 6 elements. [10, 20, 30, 40, 50, 60]
BAG: 6 elements. [60, 50, 40, 30, 20, 10]
BAG: 5 elements. [10, 20, 30, 40, 50]
BAG: 0 elements. []
False
```

Part 3 - Summary and Specific Instructions

1. Implement a Stack ADT class by completing the provided skeleton code in the file `stack_da.py`. You will use the Dynamic Array data structure that you implemented in part 1 of this assignment as the underlying data storage for your Stack ADT.
2. Once completed, your implementation will include the following methods:

```
push()  
pop()  
top()
```

3. We will test your implementation with different types of objects, not just integers. We guarantee that all such objects will have correct implementation of methods `__eq__`, `__lt__`, `__gt__`, `__ge__`, `__le__`, and `__str__`.
4. The number of objects stored in the Stack at any given time will be between 0 and 100,000 inclusive. Stack must allow for storage of duplicate objects.
5. **RESTRICTIONS:** You are not allowed to use ANY built-in Python data structures and their methods. You must solve this portion of the assignment by importing the `DynamicArray` class that you wrote in part 1 and using class methods to write your solution.
You are also not allowed to directly access any variables of the `DynamicArray` class (like `self.size`, `self.capacity` and `self.data` in part 1). All work must be done by only using class methods.

push(self, value: object) -> None:

This method adds a new element to the top of the stack.

Example #1:

```
s = Stack()
print(s)
for value in [1, 2, 3, 4, 5]:
    s.push(value)
print(s)
```

Output:

```
STACK: 0 elements. []
STACK: 5 elements. [1, 2, 3, 4, 5]
```

pop(self) -> object:

This method removes the top element from the stack and returns its value. If the stack is empty, the method raises a custom "StackException". Code for the exception is provided in the skeleton file.

Example #1:

```
s = Stack()
try:
    print(s.pop())
except Exception as e:
    print("Exception:", type(e))

for value in [1, 2, 3, 4, 5]:
    s.push(value)

for i in range(6):
    try:
        print(s.pop())
    except Exception as e:
        print("Exception:", type(e))
```

Output:

```
Exception: <class '__main__.StackException'>
5
4
3
2
1
Exception: <class '__main__.StackException'>
```

top(self) -> object:

This method returns the value of the top element of the stack without removing it. If the stack is empty, the method raises a custom "StackException". Code for the exception is provided in the skeleton file.

Example #1:

```
s = Stack()
try:
    s.top()
except Exception as e:
    print("No elements in stack", type(e))

s.push(10)
s.push(20)
print(s)

print(s.top())
print(s.top())
print(s)
```

Output:

```
No elements in stack <class '__main__.StackException'>
STACK: 2 elements. [10, 20]
20
20
STACK: 2 elements. [10, 20]
```

Part 4 - Summary and Specific Instructions

1. Implement a Queue ADT class by completing the provided skeleton code in the file `queue_da.py`. You will use the Dynamic Array data structure that you implemented in part 1 of this assignment as the underlying data storage for your Queue ADT.
2. Once completed, your implementation will include the following methods:

```
enqueue()  
dequeue()
```

3. We will test your implementation with different types of objects, not just integers. We guarantee that all such objects will have correct implementation of methods `__eq__`, `__lt__`, `__gt__`, `__ge__`, `__le__`, and `__str__`.
4. The number of objects stored in the Queue at any given time will be between 0 and 100,000 inclusive. Queue must allow for storage of duplicate elements.
5. RESTRICTIONS: You are not allowed to use ANY built-in Python data structures and their methods. You must solve this portion of the assignment by importing the `DynamicArray` class that you wrote in part 1 and using class methods to write your solution.
You are also not allowed to directly access any variables of the `DynamicArray` class (like `self.size`, `self.capacity` and `self.data` in part 1). All work must be done by only using class methods.

enqueue(self, value: object) -> None:

This method adds a new value to the end of the queue.

Example #1:

```
q = Queue()
print(q)
for value in [1, 2, 3, 4, 5]:
    q.enqueue(value)
print(q)
```

Output:

```
QUEUE: 0 elements. []
QUEUE: 5 elements. [1, 2, 3, 4, 5]
```

dequeue(self) -> object:

This method removes and returns the value from the beginning of the queue. If the queue is empty, the method raises a custom "QueueException". Code for the exception is provided in the skeleton file.

Example #1:

```
q = Queue()
for value in [1, 2, 3, 4, 5]:
    q.enqueue(value)
print(q)
for i in range(6):
    try:
        print(q.dequeue())
    except Exception as e:
        print("No elements in queue", type(e))
```

Output:

```
QUEUE: 5 elements. [1, 2, 3, 4, 5]
1
2
3
4
5
No elements in queue <class '__main__.QueueException'>
```