THOMPSON RIVERS UNIVERSITY

COMP 2230 – Data Structures and Algorithm Analysis

Assignment #2:

Due Date: Section 01- September 19th 2024; Section 02- September 20th 2024; at start of seminar

Seminar Activities

We will discuss advanced Object-Oriented topics and review programs that will be required for implementation of assignment2.

Problem #1

Implementation of the Size(), isEmpty(), and toString() methods into the provided ArrayStack class.

Input: an element to add onto the stack.

Output: elements removed from the stack.

Analysis of the problem:

Subtask and Analysis:

- Initialize stack(O(1))
  - Set initial capacity
  - top variable set to 0
- Push elements to the top of the stack(O(1))
  - Validate if stack is full and expands capacity if true
  - Set top index to passed element
  - Top variable incremented
- Pop elements from the top of the stack (O(1))
  - Validate if stack is empty and throws emptyCollectionException if true
  - Decrement top variable
  - Sets result variable to the top index element
  - Sets top index element to null
  - Returns result variable(element removed)
- Peek element at the top of the stack(O(1))
  - Validates if stack is empty and throws emptyCollectionException if true
  - Returns the top minus 1 index element
- Check is the stack is empty(O(1))
  - Returns true if the size() is 0
- Get the size of the stack(O(1))
  - Returns top variable value

- Format string to display stack(O(1))
  - Returns the stack in a string

Prototypes:

- **ArrayStack(int initialCapacity)**
  - Pre condition: non negative integer
  - Post condition: new arrayStack created with size of the nonnegative integer provided and top variable initialized at 0.
- **push(T element)**
  - Pre condition: stack that is not full
  - Post condition: elements added to top index of stack and top variable incremented
- T **Pop()**
  - Pre condition: non empty stack
  - Post condition: element at index top is removed and top variable decremented, removed element returned
- **T Peek()**
  - Pre condition: non empty stack
  - Post condition: top indexed element returned without being removed
- **Boolean isEmpty()**
  - Pre condition: none
  - Post condition: returns true if top index is null and false if the top index is not null
- **Int size()**
  - Pre condition: none
  - Post condition: returns the value of top variable indicating the number of elements in the stack.

<u>Test Cases:</u>

- Create an empty arraystack with initial capacity
- Push elements to capacity and then over capacity to test expandCapacity()(also tests push method)
- Test pop() method that removes the top of the stack
- Test peek() method that views the top of the stack without removing it
- Test isEmpty() method that checks if the stack is empty and returns true if it is and false if not
- Test size() method that returns the number of elements without modifying the stack
- Tests peek() while stack is empty to show it throws exception
- Test pop() while stack is empty to show it throws exception

<u>Code:</u>

<u>ArrayStack.java</u>

```
import Ass2_2230.exceptions.*;
import java.util.Arrays;

/**
 * An array implementation of a stack in which the bottom of the
 * stack is fixed at index 0.
 *
 * @author Java Foundations
 * @version 4.0
 */
public class ArrayStack<T> implements StackADT<T>
{
    protected final static int DEFAULT_CAPACITY = 100;

    protected int top;
    protected T[] stack;

    /**
     * Creates an empty stack using the default capacity.
     */
    public ArrayStack()
    {
        this(DEFAULT_CAPACITY);
    }

    /**
     * Creates an empty stack using the specified capacity.
     * @param initialCapacity the initial size of the array
     */
    public ArrayStack(int initialCapacity)
    {
        top = 0;
        stack = (T[])new Object[initialCapacity];
    }

    /**
     * Adds the specified element to the top of this stack, expanding
     * the capacity of the array if necessary.
     * @param element generic element to be pushed onto stack
     */
    public void push(T element)
    {
        if (size() == stack.length)
            expandCapacity();
        stack[top] = element;
        top++;
    }

    /**
     * Creates a new array to store the contents of this stack with
     * twice the capacity of the old one.
     */
    private void expandCapacity()
    {
        stack = Arrays.copyOf(stack, stack.length * 2);
    }

    /**
     * Removes the element at the top of this stack and returns a
     * reference to it.
     * @return element removed from top of stack
     * @throws EmptyCollectionException if stack is empty
     */
    public T pop() throws EmptyCollectionException
    {
        if (isEmpty())
            throw new EmptyCollectionException("stack");

        top--;
        T result = stack[top];
        stack[top] = null;

        return result;
    }

    /**
     * Returns a reference to the element at the top of this stack.
     * The element is not removed from the stack.
     * @return element on top of stack
     * @throws EmptyCollectionException if stack is empty
     */
    public T peek() throws EmptyCollectionException
    {
        if (isEmpty())
            throw new EmptyCollectionException("stack");

        return stack[top-1];
    }

    /**
     * Returns true if this stack is empty and false otherwise.
     * @return true if this stack is empty
     */
    public boolean isEmpty()
    {
        return (size() == 0);
    }

    /**
     * Returns the number of elements in this stack.
     * @return the number of elements in the stack
     */
    public int size()
    {
        return top;
    }

    /**
     * Returns a string representation of this stack.
     * @return a string representation of the stack
     */
    public String toString()
    {
        return Arrays.toString(stack);
    }
}
```

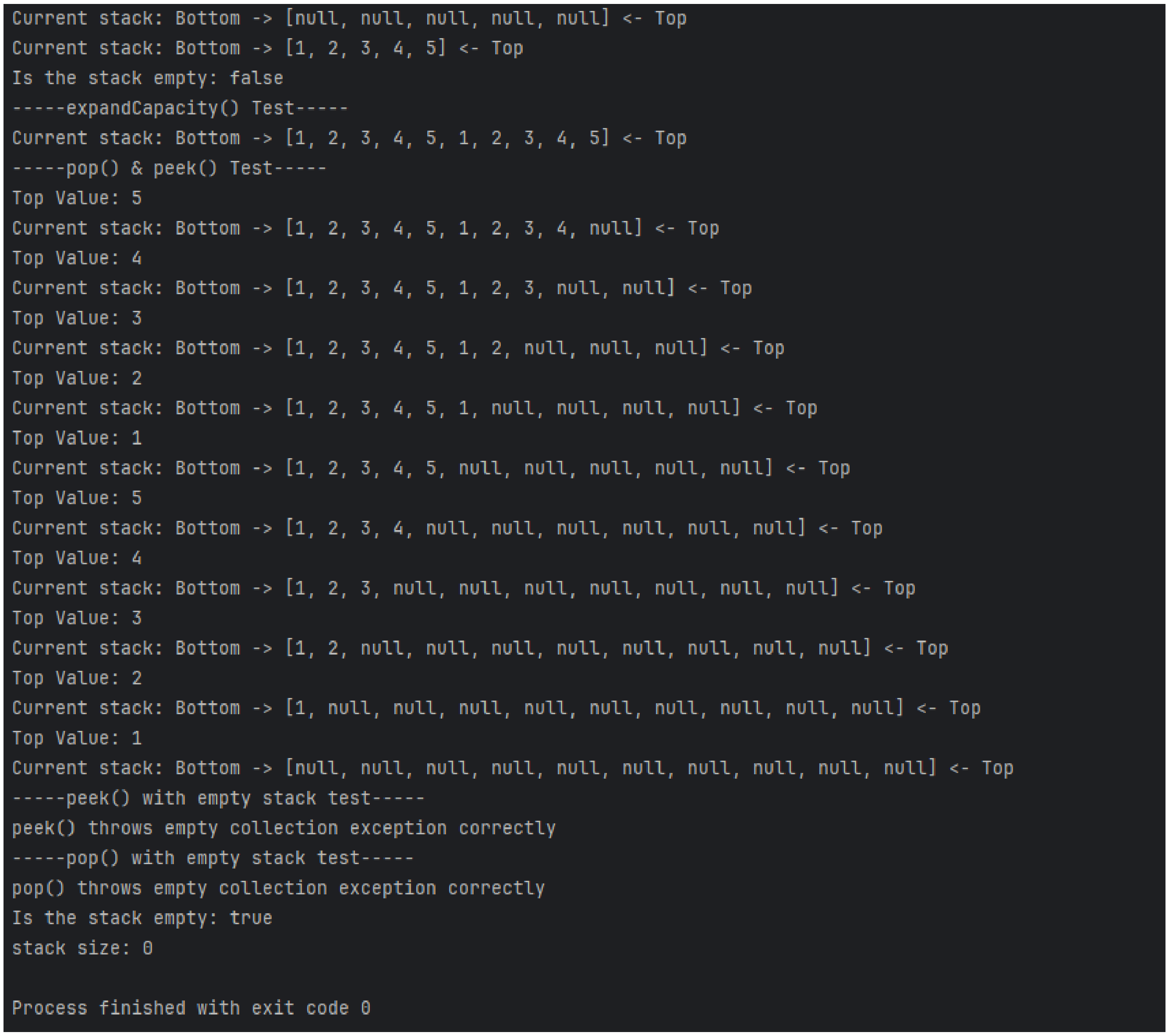
<u>ArrayStackTester.java</u>

```
package Ass2_2230;
```

```
import Ass2_2230.exceptions.EmptyCollectionException;

public class ArrayStackTester {

	public static void main (String[] args) throws Exception {

		ArrayStack<Integer> array = new ArrayStack<>(5);
		//initialization with null values and capacity 5
		System.out.println("Current stack: " + "Bottom -> " +
array.toString() + " <- Top");

		//populate stack to fill the initial capacity
		for (int i = 1; i < 6; i++) {
			array.push(i);
		}
		System.out.println("Current stack: " + "Bottom -> " +
array.toString() + " <- Top");

		System.out.println("Is the stack empty: " + array.isEmpty());

		//test expandCapacity() method
		System.out.println("-----expandCapacity() Test-----");
		for (int i = 1; i < 6; i++) {
			array.push(i);
		}
		System.out.println("Current stack: " + "Bottom -> " +
array.toString() + " <- Top");

		//test pop and peek method
		System.out.println("-----pop() & peek() Test-----");
		for (int i = 0; i < 10; i++) {
			System.out.println("Top Value: " + array.peek());
			array.pop();
			System.out.println("Current stack: " + "Bottom -> " +
array.toString() + " <- Top");
		}

		//test peek with empty method
		System.out.println("-----peek() with empty stack test-----");
		try {
```

```
                    array.peek();
            } catch (EmptyCollectionException e) {
                    System.out.println("peek() throws empty collection 
exception correctly");
            }

            //test pop with empty stack
            System.out.println("-----pop() with empty stack test-----");
            try {
                    array.pop();
            } catch (EmptyCollectionException e) {
                    System.out.println("pop() throws empty collection 
exception correctly");
            }
            System.out.println("Is the stack empty: " + array.isEmpty());
            System.out.println("stack size: " + array.size());
    }
}
```

Test Output



```
Current stack: Bottom -> [null, null, null, null, null] <- Top
Current stack: Bottom -> [1, 2, 3, 4, 5] <- Top
Is the stack empty: false
-----expandCapacity() Test-----
Current stack: Bottom -> [1, 2, 3, 4, 5, 1, 2, 3, 4, 5] <- Top
-----pop() & peek() Test-----
Top Value: 5
Current stack: Bottom -> [1, 2, 3, 4, 5, 1, 2, 3, 4, null] <- Top
Top Value: 4
Current stack: Bottom -> [1, 2, 3, 4, 5, 1, 2, 3, null, null] <- Top
Top Value: 3
Current stack: Bottom -> [1, 2, 3, 4, 5, 1, 2, null, null, null] <- Top
Top Value: 2
Current stack: Bottom -> [1, 2, 3, 4, 5, 1, null, null, null, null] <- Top
Top Value: 1
Current stack: Bottom -> [1, 2, 3, 4, 5, null, null, null, null, null] <- Top
Top Value: 5
Current stack: Bottom -> [1, 2, 3, 4, null, null, null, null, null, null] <- Top
Top Value: 4
Current stack: Bottom -> [1, 2, 3, null, null, null, null, null, null, null] <- Top
Top Value: 3
Current stack: Bottom -> [1, 2, null, null, null, null, null, null, null, null] <- Top
Top Value: 2
Current stack: Bottom -> [1, null, null, null, null, null, null, null, null, null] <- Top
Top Value: 1
Current stack: Bottom -> [null, null, null, null, null, null, null, null, null, null] <- Top
-----peek() with empty stack test-----
peek() throws empty collection exception correctly
-----pop() with empty stack test-----
pop() throws empty collection exception correctly
Is the stack empty: true
stack size: 0

Process finished with exit code 0
```

Problem #2

Implementation of a drop out array stack that extends the array stack class. Overrides the push, pop, peek, and size methods to achieve an array that drops the bottom element and adds the top element when the stack is at capacity.

Input: Nonnegative integer

Output: Empty Array Stack with a capacity equivalent to the input

Analysis of the problem:

Subtask and Analysis:

- Initialize Drop out array(doa) stack(O(1))

    - Invoke Superclass initialization with n+1 initial capacity. N+1 prevents index out of bounds exception and prevents the arrays from auto expanding with the super class push method
- Override Push method from super class(O(1))
    - If the top variable is greater than or equal to n then top is set to 0.
    - Invokes the super class push method.
    - if the top is equivalent to bottom +1 and the top index of the stack is not null then increment the bottom variable
- Override pop method from super class(O(1))
    - If the top variable is equivalent to 0 then top is set to value of n
    - Return the pop method from super class
- Override peek method from super class (O(1))
    - If the top is equivalent to 0 then return the stack at index n-1
    - Else return the peek method from super class
- Override size method from super class
    - Initialize a size variable to value of n
    - If top is greater than bottom the size variable is set to top – bottom
    - Else if top is less than bottom the size variable is set to n – bottom + top
    - Else if the value at the bottom index of the stack is null the size variable is set to 0
    - Return the size variable

<u>Prototypes:</u>

- **DropOutArrayStack(int n)**
    - Pre condition: non negative integer
    - Post condition: new arrayStack created with max capacity of n +1
- **push(T element)**
    - Pre condition: stack that is not full
    - Post condition: element added to the top of the stack, if the stack is full then the bottom element is removed to make space for the new element.
- T **Pop()**
    - Pre condition: non empty stack
    - Post condition: element at the top index is removed and returned
- **T Peek()**
    - Pre condition: non empty stack
    - Post condition: top indexed element returned without being removed
- **Int size()**
    - Pre condition: none
    - Post condition: returns the value of the local size variable indicating the number of elements in the stack.

<u>Test Cases:</u>

- Create an empty DropOutArrayStack with initial capacity
- Push elements to capacity and then over capacity to test the drop out mechanism
- Test pop() method that removes the top of the stack
- Test peek() method that views the top of the stack without removing it
- Test isEmpty() method that checks if the stack is empty and returns true if it is and false if not

- Test size() method that returns the number of elements without modifying the stack
- Tests peek() while stack is empty to show it throws exception
- Test pop() while stack is empty to show it throws exception

Code:

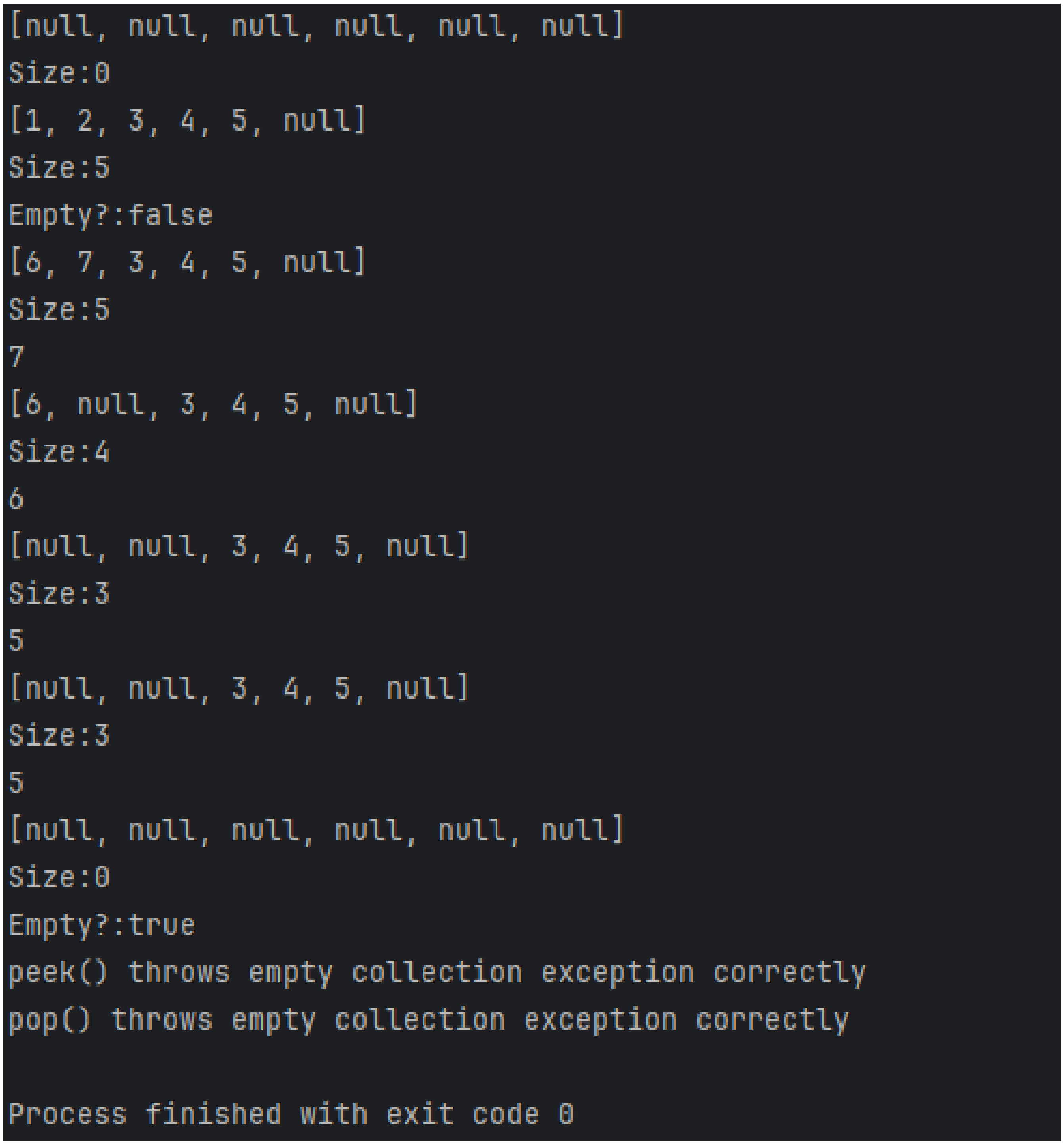
DropOutArrayStack.java

```
package Ass2_2230;

/**
 * An ArrayStack in which, if the size is n, when the n+1 element is
 * pushed, the oldest element (bottom of stack) is lost.
 *
 * @author Kaylee Crocker and Colton Isles
 */
public class DropOutArrayStack<T> extends ArrayStack<T> {

    private int n;  //the max size of the stack
    private int bottom = 0; //used only for calculating size() with O(1)
efficency

    /**
     * Creates an empty stack using the default capacity.
     */
    public DropOutArrayStack() {
        this(DEFAULT_CAPACITY);
    }

    /**
     * Creates an empty stack with the size n.
     * @param n the initial size of the array
     */
    public DropOutArrayStack(int n) {
        super(n + 1);
        /* Why array length n+1? The stack will only ever contain n
         * elements, however, ArrayStack.push() will expand the array
         * when the size of the stack reaches the length of the array.
         * This insures that the stack sizes never reaches that so the
         * array is not needlessly expanded. The last element of the
         * array will always be unused (null).
         */
        this.n = n;
    }



    /**
     * Adds the specified element to the top of this stack and removes
     * the bottom element if the size exceeds n by overwriting it.
     * @param element generic element to be pushed onto stack
     */
    @Override
     public void push(T element) {

        if (top >= n) {
            top = 0;
        }
        super.push(element);

        if (top == bottom + 1 && stack[top] != null) { //update the
bottom
            bottom++;
        }
    }

    /** removes the element on the top of the stack.
     * @return the element removed from the top of the stack
     */
    @Override
    public T pop() {
        if (top == 0) {
            top = n;
        }
       return super.pop();
    }

    /**displays the element at the top of the stack
     * @return the element on the top of the stack without removing it.
     */
    @Override
     public T peek() {
         if (top == 0) {
             return stack[n - 1];
         } else {
             return super.peek();
         }
     }
     /** Displays the number of elements in this stack.
      * @return difference between top and bottom if top is greater than
bottom, 0 if top == bottom, else n-bottom+top
      */
     @Override
      public int size() {
        int size = n;
          if (top > bottom) {
            size = top - bottom;
        } else if (top < bottom) {
            size = n - bottom + top;
        } else if (stack[bottom] == null) {
            size = 0;
        }
        return size;
      }
}
```

<u>DOASTest.java</u>

```
package Ass2_2230;

public class DOASTest {
    public static void main(String[] args){
        DropOutArrayStack<Integer> doa = new DropOutArrayStack<>(5);
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        // Stack:
        // Size: 0
        doa.push(1);
        doa.push(2);
        doa.push(3);
        doa.push(4);
        doa.push(5);
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        System.out.println("Empty?:" + doa.isEmpty());
        // Stack: 1, 2, 3, 4, 5
        // Size: 5
        // Empty?: false
        doa.push(6);
        doa.push(7);
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        // Stack: 3, 4, 5, 6, 7
        // Size: 5
        System.out.println(doa.pop());
        // 7
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        // Stack: 3, 4, 5, 6
        // Size: 4
        System.out.println(doa.pop());
        // 6
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        // Stack: 3, 4, 5
        // Size: 3
        System.out.println(doa.peek());
        // 5
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        // Stack: 3, 4, 5
        // Size: 3
        System.out.println(doa.pop());
        // 5
        doa.pop();
        doa.pop();
        System.out.println(doa.toString());
        System.out.println("Size:" + doa.size());
        System.out.println("Empty?:" + doa.isEmpty());
        // Stack:
        // Size: 0
        // Empty?: true
        doa.peek();
        // throws EmptyCollectionException
    }
}
```

Test Cases



```
[null, null, null, null, null, null]
Size:0
[1, 2, 3, 4, 5, null]
Size:5
Empty?:false
[6, 7, 3, 4, 5, null]
Size:5
7
[6, null, 3, 4, 5, null]
Size:4
6
[null, null, 3, 4, 5, null]
Size:3
5
[null, null, 3, 4, 5, null]
Size:3
5
[null, null, null, null, null, null]
Size:0
Empty?:true
peek() throws empty collection exception correctly
pop() throws empty collection exception correctly

Process finished with exit code 0
```

Remember to verify the functionality of your programs.

Assignment Submission:

Submit a print-out of the program source code and a sample of the output, for each problem. Note you must follow the marking guidelines as identified in the LabMark document.