Assignment 7

Dynamic Array of Integers Class 50 pts.

General Requirements

- *Add comments to the source code you are writing:*
  - *Describe the purpose of every variable*
  - *Explain the algorithm you are using for solution*
  - *Add proper comments for all methods. Include @param, @return, and @throws tags*
- *Archive the entire project using ZIP or RAR utility. Turn it in into the digital drop box.*

Dynamic Array of Integers Class

Create a class named `DynamicArray` that will have convenient functionality similar to JavaScript's Array object and Java's ArrayList class. The class allows to store array of integers that can grow and shrink as needed, search and sort itself, add and remove elements.

You are not allowed to use ArrayList object as well as any methods from java.util.Arrays class.

Please see the list of required features and methods below.

1. `private int array[]` field. You MUST store the data internally in a regular partially-filled array of integers. Please DO NOT USE ArrayList. The size of the allocated array is its capacity and will be discussed below.
2. `private int size` field. This variable stores the number of "occupied" elements in the array. Set to 0 in the constructor.
3. *Constructor with parameter*. The parameter defines the capacity of initial array. Allocates array of given capacity, sets size field to 0. In case the parameter given to constructor is less than 0, IllegalArgumentException is being thrown.
4. *No-argument constructor*. Allocates array of size 10, sets size field to 0.
5. *Copy constructor*. The constructor takes an object of type DynamibcArray as a parameter and copies it into the object it creates. The constructor throws IllegalArgumentException if the object that was passed to copy from is null.
6. `int getSize()` returns the size – a number of occupied elements in the array.
7. `int [] toArray()` accessor returns the array. Make sure you DO NOT return the private array field. Instead, allocate memory for the new array, copy your array field into that new object, and return the new array.
8. `public void push(int num)` adds a new element to the end of the array and increments the size field.
   If the array is full, you need to increase the capacity of the array:

   a. Create a new array with the size equal to double the capacity of the original one.
   b. Copy all the elements from the array field to the new array.
   c. Add the new element to the end of the new array.
   d. Use new array as an array field.
9. `public int pop() throws RuntimeException` removes the last element of the array and returns it. Decrements the size field. If the array is empty a RuntimeException with the message "Array is empty" must be thrown. At this point check the capacity of the array. If the capacity is 4 times larger than the number of occupied elements (size), it is time to shrink the array:
   a. Create a new array with the size equal to half of the capacity of the original one.
   b. Copy all the elements from the array field to the new array.
   c. Use new array as an array field.
10. `int get(int index) throws IndexOutOfBoundsException` returns element of the array with the requested index. If the index provided is too large or negative, the IndexOutOfBoundsException is thrown with the message "Illegal index".
11. `int indexOf(int key)` returns the index of the first occurrence of the given number. Returns -1 when the number is not found.
12. `void add(int index, int num) throws IndexOutOfBoundsException` adds a new element (passed as parameter num) to the location of the array specified by index parameter. If the index is larger than size of the array or less than 0, IndexOutOfBoundsException is thrown. When adding the element into the middle of the array, you'll have to shift all the elements to the right to make room for the new one. If the array is full and there is no room for a new element, the array must be doubled in size. Please follow the steps listed in the push() method description to double the capacity of the array.
13. `int remove ( int index) throws IndexOutOfBoundsException` removes the element at the specified position in this array. When the element is removed from the middle of the array, all the elements must be shifted to close the gap created by removed element. If the index value passed into the method is more or equal to the size or less than 0 the IndexOutOfBoundsException must be thrown.
    At this point check the capacity of the array. If the capacity is 4 times larger than the number of occupied elements (size), it is time to shrink the array.
14. `boolean isEmpty()` returns true if the size of the array is 0.
15. `void sort()` – sorts the elements of the array. You can choose any sorting algorithm you prefer.
16. `void shuffle()` – shuffles the elements of the array. Please do not use any methods from any collection objects to do the shuffling. Instead apply popular Fisher–Yates shuffle algorithm (find algorithm and even its Java encoding online).

17. `int findMin()throws RuntimeException` returns the smallest element in the array. If the array is empty throws `RuntimeException`
18. `int findMax()throws RuntimeException` returns the largest element in the array. If the array is empty throws `RuntimeException`
19. `String toString()` – returns an array as a string of comma-separated values.
20. `boolean equals(DynamicArray obj)` compares two objects (this one and the one passed as parameter) element-by-element and determines if they are exactly the same. The capacity of two compared objects is not being compared.

Name your file `DynamicArray.java`

Testing your Class

Test all the methods you wrote in main().
Please do not use user input when running your tests. Instead call your methods with hard-coded arguments – this code will create a record of your test cases. In comments explain why you chose the test cases you used. Make sure to test all exception-generating cases.
Use try-catch blocks to handle exceptions. You can comment out the exception-causing code later.

Name your file `ArrayDemo.java` .