**Lab: Linear-Data-Structures**

Write Java code for solving the tasks on the following pages. Code should compile under the Java 8 and above standards you can write and locally test your solution with the Java 13 standard, however **Judge will run the submission with Java 10 JRE**. Avoid submissions with **features included after Java 10** release doing **otherwise** will result in **compile time error**.

Any code files that are part of the task are provided as **Skeleton**. In the beginning import the project skeleton, do not change any of the interfaces or classes provided. You are free to add additional logic in form of methods in both interfaces and implementations you are not allowed to delete or remove any of the code provided. Do not change the names of the files as they are part of the tests logic. **Do not change the packages** or move any of the files provided inside the skeleton if you have to add new file add it in the same package of usage.

Please follow the exact instructions on uploading the solutions for each task. Submit as **.zip archive** the files contained inside **"...\src\main\java"** folder this should work for all tasks regardless of current DS implementation.

In order for the solution to compile the tests **successfully** the project **must** have **single** **Main.java** file containing single **public static void main(String[] args)** method even empty one within the **Main class**.

Some of the problem will have simple **Benchmark** **tests** inside the skeleton. You can try to run those with **different** **values** and **different** **implementations** in order to **observe** behaviour. However **keep** in mind that the result comes **only as numbers** and this data may be **misleading** in some situations. Also the tests are not started from the command prompt which may **influence** the **accuracy** of the results. Those tests are only added as an **example** of **different** **data** **structures** **performance** on their **common** operations.

The Benchmark tool we are using is **JMH** (Java Microbenchmark Harness) and that is Java harness for building, running, and analyzing, **nano/micro/milli/macro** benchmarks written in Java and other languages targeting, the JVM.

**Additional** **information** can be found here: [JMH](https://openjdk.java.net/projects/code-tools/jmh/) and also there are other examples over the **internet**.

**Important:** when importing the skeleton **select** **import** **project** and then **select** **from** **maven** **module**, this way any following **dependencies** will be **automatically** **resolved**. The project has **NO** **default** **version** of **JDK so after the import you may (depends on some configurations) need to specify the SDK, you can download** **JDK 13** from [**HERE**](https://jdk.java.net/13/)**.**

## ArrayList

Your task is to implement the **ADS** **List<E>** inside the **ArrayList<E>** class provided. You can see that this class implements the **List<E>** interface you have to implement all the methods in order to solve the problem, however you are free to add more methods with any access modifier you want.

### ****Boolean add (****E element) – adds an element at the end of the sequence and returns true if successful (always returns true). This method should in addition increase the size of the structure and ensure that there is enough space for the addition to work. If needed you will have to resize the array.

### Boolean add (int index, E element) – the only difference from the above one is that now we have a specified index at which to add (insert) an element. This time you have to validate the index then add the element and shift the remaining elements if any from the index + 1 to the right (from the index + 1 to the last index + 1).

### E get (int index) – returns the element at the given index and does not remove it from the collection. If the index is invalid throw IndexOutOfBoundsException with a proper message of your chose (the message itself in not subjected to testing).

### E set (int index, E element) – sets the element at given index and returns the previously stored at that index element, again you should validate the index and throw IndexOutOfBoundsException if the validation fails.

### E remove (int index) – removes the element at specified index and returns it – again the same validation, here you should already have some way to reuse the index validation.

### Int size () – returns the number of elements.

### Int indexOf (E element) – returns the index of an element if the element is not present in the structure then return -1 as invalid array index.

### Boolean contains (E element) – returns true or false if the element is present inside the structure.

### Boolean isEmpty () – returns if there are elements stored or not.

### Boolean addMany(E[] elements): adds an array from the input at the end of the sequence and returns true if successful (always returns true). This method likes add function behaviors.t

### E[] sort(): return an ordered list in ascending order

### E[] reverse(): return a reversed list.

To implement your methods, let define a main method.

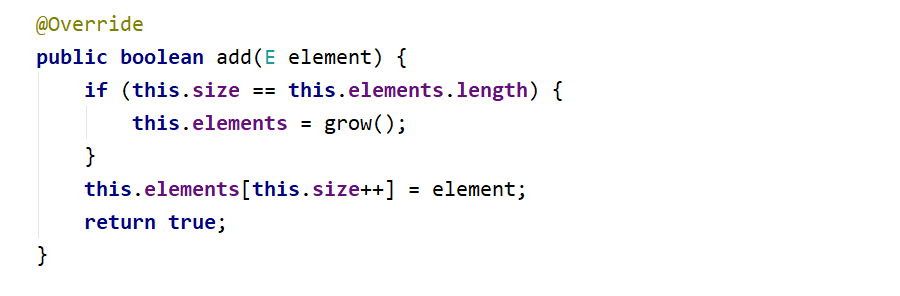
For example.

1. Declare and initialize an object of your ArrayList
2. Add 2 elements in this object.
3. Insert 1 element between the old elements.
4. Set 1 element in postion 1
5. Search an element.
6. Print all element in this object.

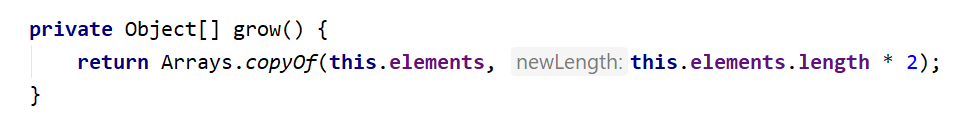
### Solution:

##### Define an interface List<E>:

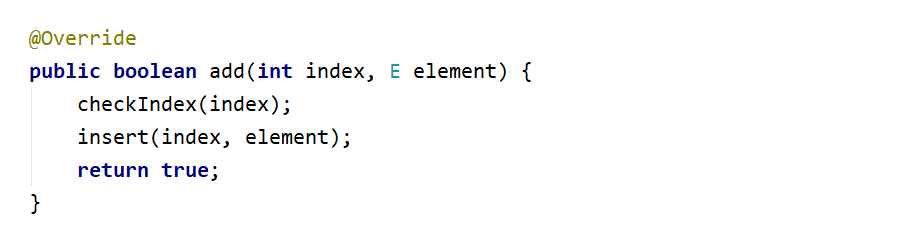
##### Boolean add (E element):



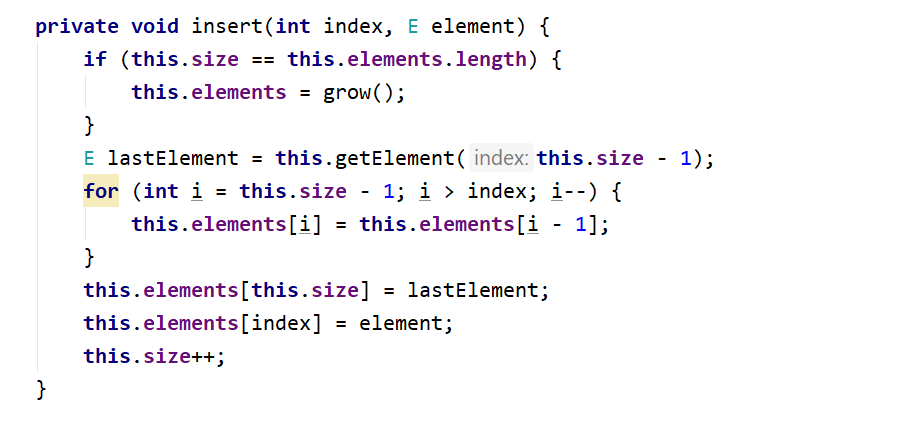
##### The – grow () helper method:



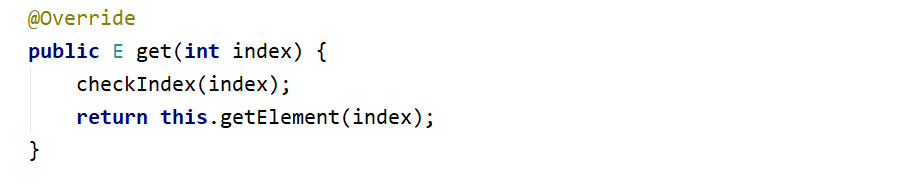
##### Boolean add (int index, E element):



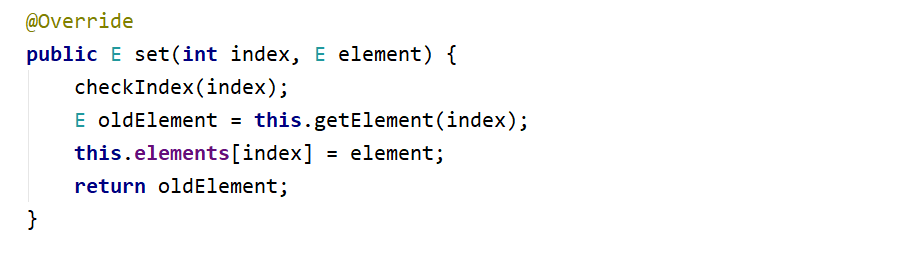
##### The insert (int index, E element) method:



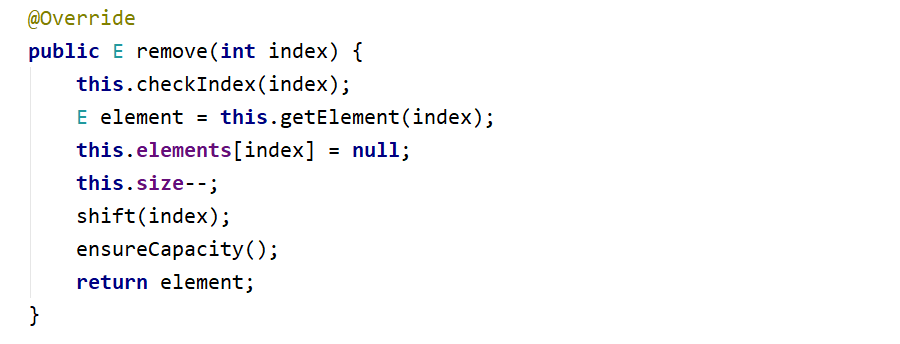
##### E get (int index):



##### E set (int index, E element):



##### E remove (int index):

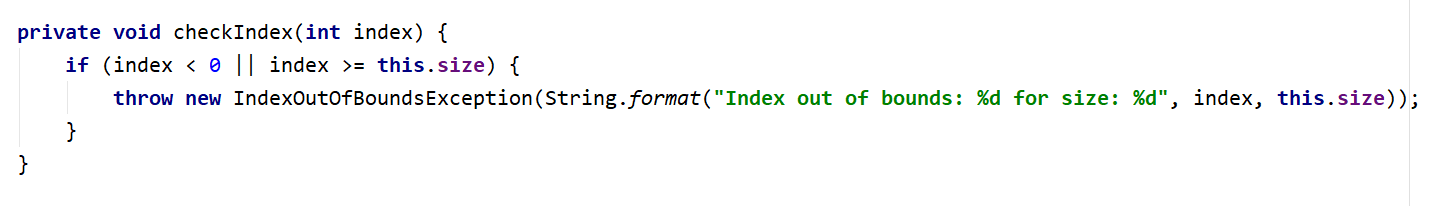


##### The ensureCapacity method: Take a look at those additional helper methods, you can reuse them whenever needed: First ensure capacity of the array if we have less than one third of the elements we can shrink the array.

##### The shrink method, looks a lot like grow with one major difference – we reduce the space:



##### And last but not least the check index method, feel free to modify the message.



All of the **other** **methods** are really **easy** and **straightforward** to be **implemented** so you won't need any help. If it doesn't work the first time **simply try different approach**.