UNIVERSITÉ DU LUXEMBOURG

Array List from Scratch

Programming Fundamentals 2

Goals

- ✯ Learn the basics of JUnit and JavaDoc.
- ✯ Study the dynamic array data structure.
- ✯ Study the *selection sort* algorithm.
- ✯ **Relevant videos**:
  - Dynamic array data structure.
  - Testing your code.

Deliverables

1. The code on your Github repository generated by clicking here: `https://classroom.github.com/a/0vQxNaoS`
2. **Reviewer**: Gustavo Rocha Koglin (k0glin on Github).
3. **Automated testing** using the *JUnit* unit testing framework.

Exercise 1 – From scratch

In this exercise you are asked to implement your own dynamic array class. The cloned repository comes with a file `DynamicArray.java` with holes! Read the comments in the file and complete the unimplemented methods.

Exercise 2 – Unit testing

In order to test that the implemented methods have the right behaviour, you need to do unit testing. In the cloned repository, the file `DynamicArrayTest.java` tests your class with JUnit. There are 4 tests implemented, you can add more tests if you want to practice but you do not need to write anything in this exercise. The purpose is to get you more familiar with unit testing (you will have to write your own tests in later labs). Execute the command `mvn -Dtest=DynamicArrayTest test` to run the test, or simply `mvn test` to run all test files.

- Methods related to tests must have a JUnit annotation.
  - When you run a test, it executes every methods annotated with `@Test`.
  - The method annotated with `@BeforeEach` is executed before each `@Test` method is executed.
  - The annotation `@DisplayName` simply gives a name to a `@Test` method, so that the result of the test is more readable.

- Unit testing works with assertions. If one of the assertions is wrong, then the tests will fail.
  - `assertEquals(v1, v2)` asserts that the expected value `v1` equals the actual value `v2`.
  - `assertThrows(c, e)` asserts that the executable `e` well throws an exception of the type `c`.

→ You can find the official documentation of JUnit there for any further explanation.

Exercise 3 – Javadoc

When you write a code to be reused, you need to provide a documentation to help developers to understand the behaviour of the implemented classes and methods. Here is the official documentation of Java for instance. Similarly to the exercise about unit testing, you do not have to write anything but you can write your own documentation if you want to practice. With Javadoc, you can write the description of a method or a class with `/** ··· */` as it is written in the file `DynamicArray.java`. In addition to a description of the method, you may add tags (starting with `@`) for specific details.

- `@param` to describe a parameter used as an input of the method
- `@return` to describe the value returned by the method (its type and permissible range of values)
- `@throws` to specify if an exception can be thrown in the method and give the cause

→ you can find the official documentation of Javadoc there for any further explanation.

You can automatically generate the documentation using the command `mvn javadoc:javadoc`, and it produces HTML files in `target/doc/index.html`.

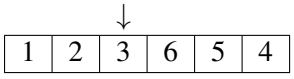
Exercise 4 – Sorting algorithm

The selection sort is a sorting algorithm which consists in finding the minimum element in the unsorted part of an array, and swapping it with the first unsorted element. It owes its name to the fact that it selects the next smallest element to place it in its final position. After the first pass, the smallest element is thus at the first position of the array, and after k passes, the first k elements of the array are ordered. What is the time complexity of this algorithm?

Example Consider the following array | 3 | 5 | 1 | 6 | 2 | 4 |. Here is a description of the first passage:
After the first pass, the minimum element 1 is found and swapped with the first unsorted element 3:

After the second pass, the minimum element 2 is found and swapped with the second unsorted element 5:

After the third pass, the minimum element 3 is found at correct position so the array is not changed:



The process continues until all elements are sorted:

At the end of the last pass, the array is fully sorted in ascending order.

1. In the file `Algorithm.java`, write the content of the static method `static void sort(DynamicArray array)` which implements the selection sort of the given array `array`.
2. Uncomment the fifth test in the file `DynamicArrayTest.java` to check your implementation.

Exercise 5 – Binary search

When an array is sorted, we can find an element faster than checking every element of the array with an algorithm called *binary search*. The idea is simple, let a be an array and x the element to search.

- Start with an interval $[l, u]$ such that $a[l] \leq x \leq a[u]$ (initially $[0, n-1]$ where n is the size of the array).
- Reduce that interval by taking the middle. If $a[mid] < x$ for instance, the interval becomes $[mid+1, u]$, otherwise it becomes $[l, mid]$.
- Start again with the new interval.

In the file `Algorithm.java` you can find another static method `static int search(DynamicArray array, int x)`. Write the content of this method to implement a binary search to find the index of the first occurrence of `x` in `array`. If the element is not present, it must return -1.