# Lab: Unit testing

This document defines the lab overview for the ["Java OOP Advanced" course @ Software University](https://softuni.bg/java-advanced-oop). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1376/java-advanced-oop-july-2016).

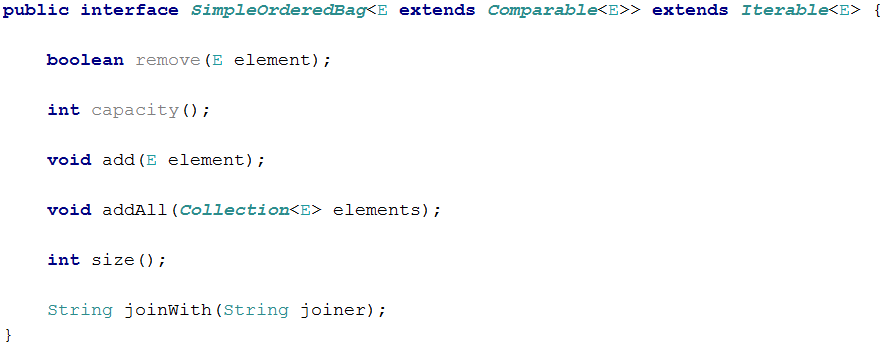
# Introduction

In the current lab piece we are going to extend the sorted data structure that we have a little bit and after that test if the whole functionality works correctly. If we work through the interface, in a later stage of our project, we may change the concrete implementation of the data structure, but the behavior will stay the same, so the only thing we will need to modify is the actual instantiation in the unit test class.

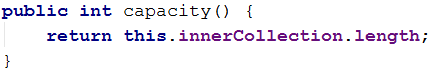
## Modify the SimpleOrderedBag and SimpleSortedList

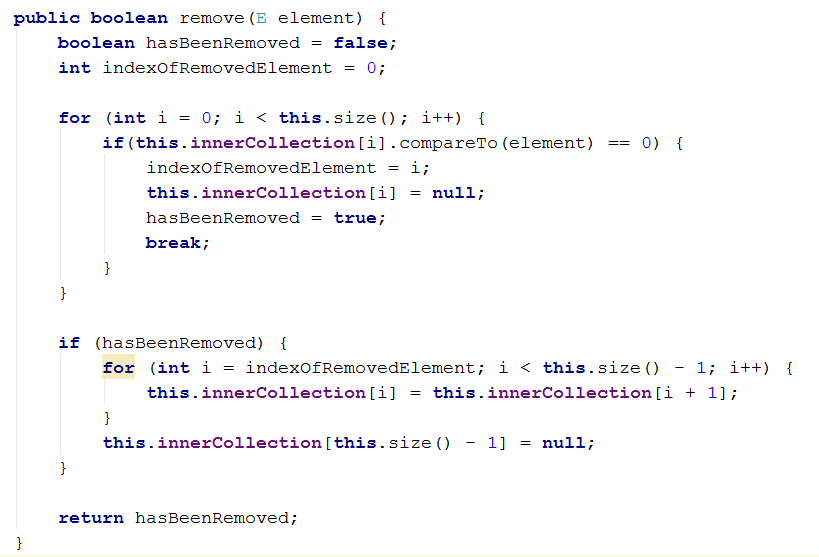
There are two things that we will add in the interface. The first one is a **remove** method that receives a <E> element and returns **true** or **false** - whether it has been removed or not.

The second thing is a method for the **capacity** of the data structure (length of the internal array). Here is how the interface should look after the modification:



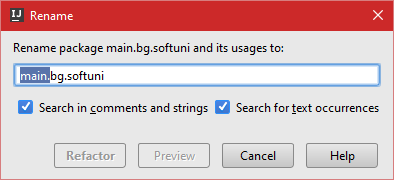
Now implement the missing functionality in the **SimpleSortedList**. Here is how the two methods should look:





## Writing Unit Tests

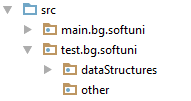
We will start with a little refactoring to accommodate our project for unit tests. First rename our main root package, so we can distinguish our main package from our test one:



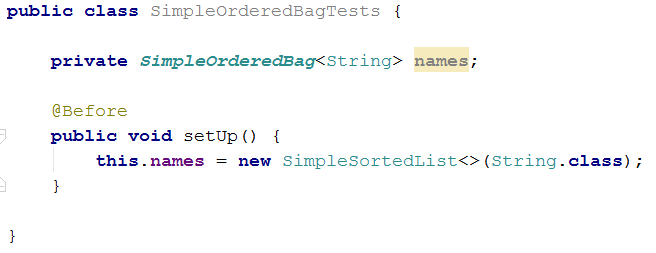
Remember to also fix the constants we used to store package and folder names in the **CommandInterpreter**:



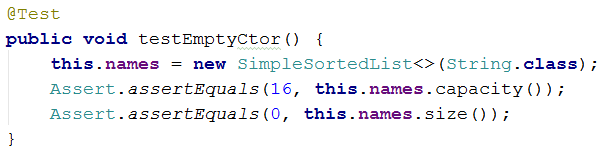
Now create a new package in the **src** folder. You can call it **test.bg.softuni**. Also create inside it two other packages called **dataStructures** and **other**:



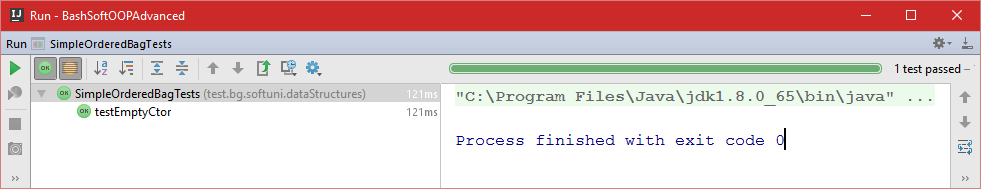
Finally we can start writing some tests. Crete a Class called **SimpleOrderedBagTests**. We can start with making a method that will initialize conditions for testing all other tests. Call it **setUp** and mark it with the @Before annotation - it ensures this method will be executed before all other tests. The method will initialize a field of type **SimpleOrderedBag<String>.** We work trough the interface so we can later change the implementation in the **setUp** and all our test will still work. This is how the class should look so far:



There is a problem though - what if the constructor doesn't even work? We need to test it too so let’s do that. Name the first test method **testEmptyCtor** and in it we will make a new instance of the **SimpleSortedList** and check the capacity and size values:



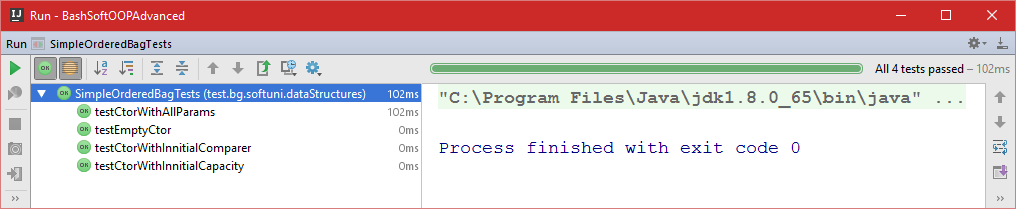
If you run the unit test everything should be according to plan and the result should be such:



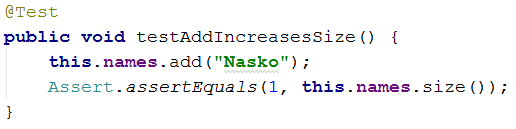
Now let’s make methods that test the other 3 constructors:



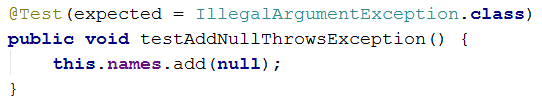
Resulting in:



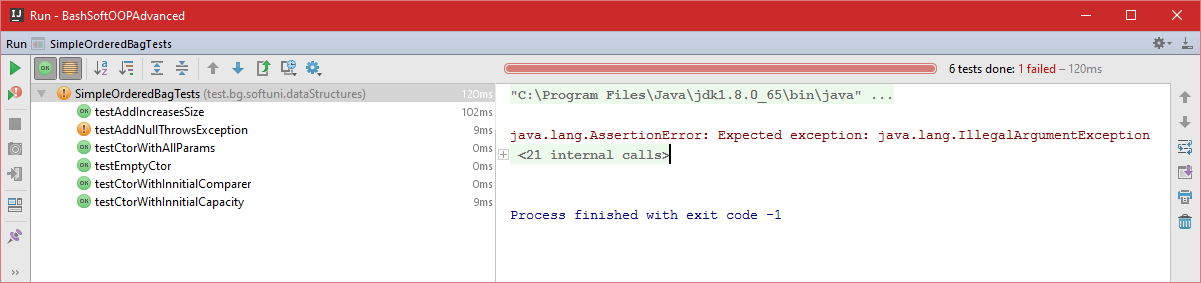
Since we are done with that, let’s check that the add method works as expected and increases the size of the collection:



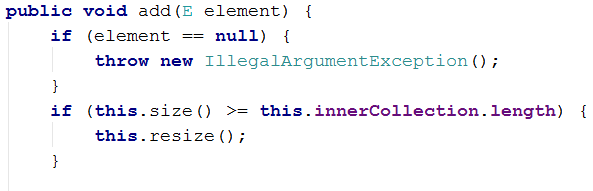
Now we should start thinking like QAs and think what should happen if someone adds **null**. We think it makes sense to throw a new argument null exception, because that does not match the conditions for what values can be held in our data structure:



Now let’s run the unit tests and see the results:



If we look at the implementation of the add method we will see, that we do not have any check for null and so we do not throw an **IllegalArgumentException**, anywhere. Let’s implement that functionality now:



If your add method begins like the picture above, everything is great.

Now we’ve shown you how to make unit tests. Try doing unit tests for the following functionality on your own:

**testAddUnsortedDataIsHeldSorted** – adds three Strings (“Rosen”, “Georgi”, “Balkan”) and checks by **iterating** the collection if it holds them in a sorted order: "Balkan", "Georgi", "Rosen"

**testAddingMoreThanInitialCapacity** – adds 17 elements and checks whether the Size is 17 and Capacity is NOT 16.

**testAddingAllFromCollectionIncreasesSize** – adds 2 elements to a List<String> and adds this list to the names using **addAll**. Then check whether the size of the collection equals 2.

**testAddingAllFromNullThrowsException** – adds a null value to the **addAll** and expects it to throw an **IllegalArgumentException**. Since this functionality was not considered in the implementation of the DS, we should now implement it.

**testAddAllKeepsSorted** – adds a collection with a few elements and after that check if the elements in the SimpleSortedList are sorted.

**testRemoveValidElementDecreasesSize** – adds an element, and removes the same element and then checks if the size has decreased. As you can see, this test should not pass, because we are not decreasing the size anywhere. Fix that and everything should be fine.

**testRemoveValidElementRemovesSelectedOne** – adds two elements (ivan, nasko) and then removes “ivan” from the collection and asserts it is not there.

**testRemovingNullThrowsException** – tries to remove null from the collection, which should result in an exception being thrown. Since we haven’t thought of that earlier, it’s now time to implement it.

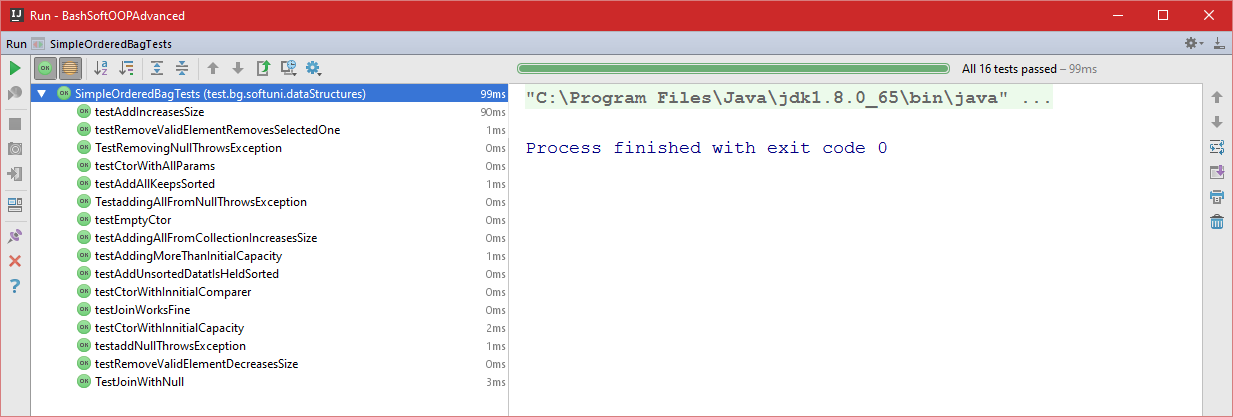
**testJoinWithNull** – adds a few elements and tries to join them with null joiner. This should throw an argument null exception, however the current implementation does not do so. Fix the implementation and the unit test should pass.

**testJoinWorksFine** – adds a few elements and tries to join them with “, ”.

These are only the common unit tests. You may want to make some more using the other constructors, to check if there are any problems with the comparators.

You can also make some performance tests to see how slow our simple implementation really is.

Final result:



This is your last lab for the current course and for the whole Java Fundamentals Module. Hope you’ve found it interesting and helpful. You can add more functionality on your own, or extend and refactor it even further. You can even upload it to your GitHub profile as it could be a part of your portfolio.