

CSC216 Java Collections - In-Class Lab

[CSC216 Java Collections - In-Class Lab](#)

[Overview](#)

[Setup](#)

[Activities](#)

[Test CSC216ArrayList\(\)](#)

[Test add\(int, E\)](#)

[Test remove\(int\)](#)

[Test set\(int, E\)](#)

[Test get\(int\)](#)

[Stretch Goal](#)

[Troubleshooting](#)

[Non-fast forward notification when pushing](#)

[Merge Conflicts](#)

[major.minor.52 error](#)

Overview

Today we will test an implementation of `ArrayList` called `CSC216ArrayList`. The `CSC216ArrayList` class that you are testing is provided in a jar file, so you are unable to see the source. Instead, you will use the provided documentation to test `CSC216ArrayList`. Your goal is to write good tests that demonstrate the functionality of `CSC216ArrayList` and give you confidence that the library is implemented correctly. There are no intentionally seeded bugs, so we expect your tests to pass if your tests are appropriate. But there could be bugs. Let us know if you find one.

The documentation of the `CSC216ArrayList` class may be found by opening the `index.html` file in `Collections/doc/`.

If you run into any problem, check out the [Troubleshooting Section](#).

Setup

1. Download [Collections.zip](#).
2. Import into Eclipse
 - a. Right click on Package Explorer > Import > Existing Project Into Workspace > Next.
 - b. Select the radio button next to "Select archive file:" and browse for Collections.zip on your file system.
 - c. Make sure the Collections project is checked and click Finish.
 - d. If you already have a Collections project in your workspace, delete it by right clicking on the project and selecting Delete. Do NOT check the box to delete it from your file system unless you really want to delete it from your file system. You may need to rename the folder in your file system if you're still unable to import the Collections project.
3. Fix build path issues. You have a build path issue if there is a giant red exclamation point on the Collections project icon. Try the following:
 - a. Fix the JRE library. The Collections project contains a jar file with an implementation of ArrayList. The jar file was compiled using Java 1.7. You must be using Java 1.7 or higher to complete this exercise.
 - i. If you need a later version of Java, install it as described in the [Installation Tutorial](#). Remember the location of the Java installation. To associate the new Java install with Eclipse, do the following:
 1. Go to Window > Preferences > Java > Installed JREs
 2. If there is no installed Java 1.7 or Java 1.8 JRE, select Add > Standard VM. Search for the Directory for the installation. You want to select the top level of the directory, which will be named something like jdk1.7.0_76 or jre1.7.0_76.
 3. After selecting the directory, the list of JRE system libraries should be populated with files like rt.jar, etc. That means you selected an appropriate location.
 4. Click Finish. You can select the new library as the system default by selecting the checkbox next to the library. Click OK.
 - ii. If you have Java 1.7 or higher installed, you need to associate it with your project.
 1. Right click on the project and select Properties > Java Build Path. Select the Libraries tab.
 2. Select the current Java Library and Remove it.

3. Select Add Library > JRE System Library. Select the Workspace Default if it's Java 1.7 or higher. Otherwise, select an Alternate JRE from the drop down that is Java 1.7 or higher.
 4. Click Finish > Ok.
 - iii. If the error is related to JUnit, follow the project level steps in ii, but instead of adding a JRE System Library you will add a JUnit library. Make sure you're adding JUnit 4.
 - iv. If you have other build errors, please raise your hand for a member of the teaching staff to help you.
4. Run the test cases.
- a. The project comes with a starting set of test cases. We should start by making sure that they pass. Run the tests by opening the project, test folder, and package. Right click on `CSC216ArrayListTest` and select Run As > JUnit Test.
 - b. Make sure you have a green bar.
 - c. If you receive the following runtime error, go back to Step 3.ii, and make sure your project is using Java 1.7 or later!

```
java.lang.UnsupportedClassVersionError:
edu/ncsu/csc216/collections/CSC216ArrayListTest :
Unsupported major.minor version 51.0
```

5. Clone your repo
- a. Go to [github.ncsu.edu](https://github.com/ncsu) and select the CSC216 organization provided by your instructor.
 - b. Select or accept the invitation for the repo provided by your instructor for this class activity
6. Commit/push
- a. One team member should do the following:
 - i. Right click on the project and select Team > Share Project > Git > Next.
 - ii. Select the repository you just cloned and click Finish.
 - iii. Right click on the project and select Team > Add to Index.
 - iv. Right click on the project and select Team > Commit.
 - v. Enter a description commit message and click Commit and Push.
 - b. All other team members should retrieve the project from the repo by doing the following:
 - i. Open the Git Repositories view, right click on the repository, and select Pull.

- ii. Open the repository and Working Directory folder.
- iii. Right click on the Collections project and select Import Project.
- iv. In the wizard, select Import existing projects and make sure that Collections is highlighted. Click Next.
- v. Make sure Collections is checked and then click Finish.
- vi. The project should now be in your Package Explorer.

Activities

Work on this lab as a team. That means one person should be driving development and the others should be helping. You can periodically switch roles so that everyone gets a chance to code!

When testing a list, you should always consider how the element would behave when interacting with 1) an empty list, 2) the front of the list, 3) the middle of the list, and 4) the end of the list. The comments and details below cover these cases. As you work, identify which test scenarios are associated with each part of the list.

Test `CSC216ArrayList()`

The first thing that should be tested is that the `CSC216ArrayList` object is constructed correctly. A new constructed `CSC216ArrayList` object should be empty. Write a test that confirms the `list` object is constructed correctly by checking that all of the *non-adding* `CSC216ArrayList` methods behave correctly when called on an empty list. For example, attempting to remove the first element from an empty list should throw an `IndexOutOfBoundsException`.

Before you move on to the next test, do the following:

- Javadoc
- Run your tests! Does the test for the constructor pass?
- Commit & Push!
- Switch drivers

Test `add(int, E)`

Now that we are confident that the list is constructed correctly, let's test that elements are added to the list correctly. If we aren't confident in adding elements, the rest of our tests are invalid!

The `testAddIntE()` method has two tests implemented: 1) attempting to add an element to an out of bounds index and 2) adding an element to an empty list. You may use these tests as a model for writing your other tests. You should use different strings of input so that you can verify that the elements are in the correct order after adding. Each test is going to build on the code before it and each test should check that not only is the size of the list correct after the add, but that the order of the elements are correct after the add.

Write tests for the following scenarios:

- Add an element to the front of a list (with one element)
- Add an element to the middle of a list (with two elements)
- Add an element to the back of a list (with three elements)
- Attempt to add a null element. A `NullPointerException` should be thrown and there should be no change to the list.
- Attempt to add an element at index -1. An `IndexOutOfBoundsException` should be thrown and there should be no change to the list.
- Attempt to add an element at index 5 (of a four element list). An `IndexOutOfBoundsException` should be thrown and there should be no change to the list.
- The `CSC216ArrayList` starts with an array of 10 elements, but should appear to grow automatically to clients. Add 6 additional elements to the list. Then attempt to add the 11th element. The 11th element should be added.

Before you move on to the next test, do the following:

- Javadoc
- Run your tests! Do your tests pass?
- Commit & Push!
- Switch drivers

Test `remove(int)`

Now that we are confident that our `add` method is working correctly, let's test that elements can be removed from the list.

The `testRemoveInt()` method has some initial setup code for adding four elements to the list and ensuring that the list is correct and one test implemented for removing the middle element in the list.

Write tests for the following scenarios:

- Attempt to remove an element from an empty list. An `IndexOutOfBoundsException` should be thrown and the list should remain empty.
- Attempt to remove an element at a negative index from a list with 4 elements. An `IndexOutOfBoundsException` should be thrown and the list should remain the same.
- Attempt to remove an element at an index 4 or greater from a list with 4 elements. An `IndexOutOfBoundsException` should be thrown and the list should remain the same.
- Remove the first element of a 3 element list.
- Remove the last element of a 2 element list.
- Remove the only element from a list.

Before you move on to the next test, do the following:

- Javadoc
- Run your tests! Do your tests pass?
- Commit & Push!
- Switch drivers

Test `set(int, E)`

Now that we are confident that our add and remove methods are working correctly, let's test that the element at a given index can be set to a given value.

The `testSetIntE()` method has some initial setup code for adding four elements to the list and ensuring that the list is correct and one test implemented for setting a middle element in the list.

Write tests for the following scenarios:

- Attempt to set an element in an empty list. An `IndexOutOfBoundsException` should be thrown and the list should remain empty.
- Attempt to set an element at a negative index in a list with 4 elements. An `IndexOutOfBoundsException` should be thrown and the list should remain the same.
- Attempt to set an element at an index 4 or greater in a list with 4 elements. An `IndexOutOfBoundsException` should be thrown and the list should remain the same.
- Set the first element of a 4 element list.
- Set the last element of a 4 element list.
- Attempt to set an element to null. A `NullPointerException` should be thrown and the list should remain the same.

Before you move on to the next test, do the following:

- Javadoc
- Run your tests! Do your tests pass?
- Commit & Push!
- Switch drivers

Test `get(int)`

Now that we are confident that our `add`, `remove`, and `set` methods are working correctly, let's test that we can retrieve an element from a given index.

The `get(int)` method has already been tested with values in range of the list when writing the tests of the other methods. That means your other method should have not only tested that the size of the list is correct, but that the contents is correct. That's accomplished by getting each element in the list and checking the value. All that remains to test in `testGetInt()` method is the bounds. You'll start by writing code that will set up a list with 4 elements.

Write tests for the following scenarios:

- Attempt to get an element at a negative index in a list with 4 elements. An `IndexOutOfBoundsException` should be thrown and the list should remain the same.
- Attempt to get an element at an index 4 or greater in a list with 4 elements. An `IndexOutOfBoundsException` should be thrown and the list should remain the same.

Before you move on to the next test, do the following:

- Javadoc
- Run your tests! Do your tests pass?
- Commit & Push!
- Switch drivers

Stretch Goal

If you've still got time after writing all of the test cases for `CSC216ArrayList`, start creating a client program that uses `CSC216ArrayList` in some way. Perhaps you will want to create a `ToDo` application. Or maybe a grocery list application.

Create a new package for your work and see what you come up with! Make sure you document and test as you work!

Troubleshooting

Non-fast forward notification when pushing

This means there is something different between your local and remote repositories such that you cannot push to the remote repository. To resolve, do the following:

- If the Git Repositories view isn't open, open it by selecting Window > Show View > Other > Git > Git Repositories.
- Next to the [master] portion of the Repo Name, you will likely see arrows pointing up and down with different numbers. That has the number of changes from upstream and the number of changes that need to be pushed.
- In the Git Repositories view, right click on the repo and select Pull. This should change the arrows to only point up with changes for upstream.
- Right click on the repo and select Commit. Commit and Push.

Merge Conflicts

See <http://courses.ncsu.edu/csc216/common/tutorials/eclipse-github/eclipse-github.html#merge>.