PythonList

A list in Python is a fundamental, random access data structure. One main difference between Python lists and arrays / list-based structures in Java is *indexing*. In short, Python allows negative indices. For example, the index of the 'last' element in a Python list is indexed with `-1` and decreasing as we move toward the 'first' element (classic index 0). A few more examples follow.

For list `L = [8, -3, 4, 7, 100, -11]`, `L[-1] == -11` and `L[-6] == 8`.

Note that indices are not unrestricted with Python lists. In particular, given a list `L`, acceptable indices are in the interval

$$-L.size() \leq index < L.size()$$

and that there are no valid indices for empty lists.

This lab asks you to implement a `PythonList` class supporting negative indices as well as a set of verification tests.

Part I: `PythonList` Implementation

The `PythonList` class inherits from the `ArrayList` class and overrides the methods shown in Table 1. Each method must accept both positive and negatives in the appropriate range for the current list. Note: Table 1 is not a comprehensive list of all index-based methods defined by `ArrayList`; you are not required to implement more methods than listed in Table 1.

Table 1: Methods to Implement for the `PythonList<E>` class

Return Type	Method	Description
Constructor	`PythonList()`	Constructs an empty list with an initial capacity of ten.
Constructor	`PythonList (Collection<? extends E> c)`	Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`void`	`add(int index, E element)`	Inserts the specified element at the specified position in this list.
`E`	`get(int index)`	Returns the element at the specified position in this list.
`E`	`remove(int index)`	Removes the element at the specified position in this list.
`E`	`set(int index, E element)`	Replaces the element at the specified position in this list with the specified element.
`PythonList<E>`	`subList(int fromIndex, int toIndex)`	Returns a view of the portion of this list between the specified `fromIndex`, inclusive, and `toIndex`, exclusive. Throws an IllegalArgumentException if the *Java* (positive) indices have `fromIndex > toIndex`.

Each of these methods must be accompanied by proper docstring comments; consider looking up Java's `ArrayList` implementation to ease this burden. For example,

```
/**
 * Returns the element at the specified position in this list.
 * @param: index - index of the element to return
 * @returns the element at the specified position in this list
 * @throws IndexOutOfBoundsException - if the index is out of range (-size() <
                                                    index || index >= size())

 */
```

We strongly recommend implementing two `private`, utility methods. `rangeCheck` will verify the goodness (or not) of input index integer values. `standardizeIndex` will convert a valid Python-style index to a valid Java index.

Testing `PythonList` for Lab 1

Implement a main method (not in `PythonList.java`) that exercises each of the methods. Each method should be exercised individually (i.e., have a dedicated testing method).

Output should only occur in one of two cases. If a test fails, a description of the problem should be written to the console. ***A good description includes a brief description of the test, actual input, expected output, and actual output.***

The other case in which output is appropriate is when all tests succeed: indicate this fact with a simple statement.

Submitting Source Code

Your code should be well documented, including docstring comments of methods, blocks of code, and header comments in ***each*** file.

Testing code needs fewer comments as they should be self-descriptive; however, it is recommended that each individual test or family of tests be numbered and have a brief comment.

Header Comments

Your program must use the following standard comment at the top of *each source code file*. Copy and paste this comment and modify it accordingly.

```
/**
 * Write a succinct, meaningful description of the class here. You should avoid wordiness
 * and redundancy. If necessary, additional paragraphs should be preceded by <p>,
 * the html tag for a new paragraph.
 *
 * <p>Bugs: (a list of bugs and / or other problems)
 *
 * @author <your name>
 * @date   <date of completion>
 */
```

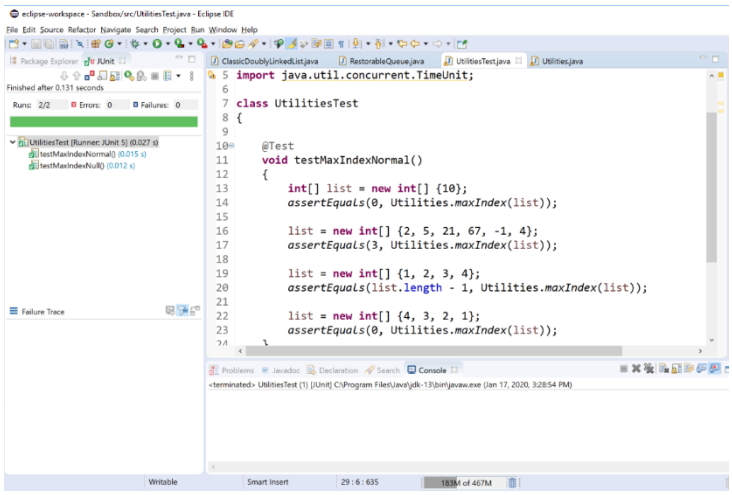
Inline Comments

Comment your code with a *reasonable amount of comments* throughout the program. Each method should have a comment that includes information about input, output, overall operation of the function, as well as any limitations that might raise exceptions; Javadoc comments are ideal. Each *block* of code (3-4 or more lines in sequence) in a function should be commented.

It is ***prohibited*** to use *long* comments to the right of lines of source code; attempt 80 to 100 character-wide text in source code files.

Submitting; Proof of Program Execution

Execute your code and take a screenshot of the associated output console. Place these screenshots into a word processing document (Word, OpenOffice, GoogleDocs, etc.). If multiple screenshots are necessary, label each clearly. Please make sure to crop and enlarge the screenshots so that the picture and / or text is clear (and doesn't strain my old eyes). For example, ***the screenshot below is not appropriately sized*** although it contains ideal information (output console, code, etc.). Create a PDF of this document and call it `evidence.pdf`.



Source Code Files

You are to submit your entire project folder (including any files provided to you).

Final Submission File

In a folder named `lab`, place (1) the project code folder and (2) `evidence.pdf`. Zip folder `lab` and label that zip file as `lab.zip`. This zip file is to be submitted via Moodle.