The purpose of this lab is to practice

A) using polymorphism in the code.

B) reading Java Documentation API and understand the relationships between objects. For this, I'm going to ask you to learn how to read from a file.

1. Prelude

A program is a piece of code that gets data (i.e. `input`), process it and send out the processed data (i.e. `output`). The inputs are sent from a variety of input devices such as keyboards, digital pens, scanners, or files. The output of a program can also be sent to a variety of devices such as screens, printers, or files.

The good thing about working with files is their involatile property. If you write something on a file, it stays forever (theoretically, if no technical problem happens) unless you remove it explicitly.

Two types of files can be defined, text and binary. Both types are stored as a series of bits (0 and 1). A text file is a file that can be opened by a text editor and usually have .txt extension. If a file is not a text file, it is a binary file. For example, an image or a voice is a binary file. In this lab we work with a text file only.

In java, data is transferred between sources and destinations as a `stream` of data, which simply represents a flow of data. An `input stream`, therefore, is a stream of data that flows from an input device to your program, while an `output stream`, is a stream of data that flows from your program to an output device.

An example of input stream is `System.in` that is used to read data to a program, while `System.out` is an output stream that is used to write data to the screen.

To read/write from/to a file, a correct input/output stream should be used. Java provides a variety of ways by which you can read and write data. One straight forward method to read data is to create a `Scanner` object that scans the input streams. For example, if a data is read from keyboard, the correct input stream is `System.in` and therefore the object is defined as:

```
Scanner scannerObject = new Scanner(System.in);
```

To read from a file, a couple of input stream can be used. One simple way is to simply pass the name of the file into the scanner object. For example, to read from `input.txt` the following code is used:

```
Scanner scannerObject = new Scanner(new File("inpu.txt"));
```

At this point I think you should have got some idea of how a file is read. What I need you to do is to visit https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/util/Scanner.html Whenever that is needed, when you implement the code for this lab.

Please note there are things that I have not explained here for which you need to do a bit of research in order to complete this lab. I list those here:

- Exception handling when reading from a file.
- An open file should finally be closed when it is not needed any more.

2. Setup

Please download `Lab6.zip` that is attached to this description.

- Open `eclipse`.
- Click on *`File`* and select *`Import`*.
- Choose *`Existing Projects into Workspace`* and click *`Next`*.
- Click on *`Select Archive File`* and then *`Browse`*. Find `Lab6.zip` and click *`Finish`*.
- Please make sure that you do not already have a project called `EECS2030_Lab6`, otherwise, eclipse cannot import it for you.

You should see two files, one is called `Container.java` and one `ContainerTester.java`. That's right :) Again, we will be working with containers in this lab. You should also see two text files in this project named `input.txt` and `in.txt`.

3. JavaDoc generation

The javaDoc has been written for you. All you need to do is to generate it as an HTML file to make it easier for navigation. For this, right click on `Container.java` -> select `export -> javaDoc -> Next`. It will ask you for the location in which you want to store the documentation. Enter the path and then click `Finish`.

If you look at the location in which you stored the documentation, you'll see there is a file called `index.html`. Clicking on this file, shows the documentation of the project in your browser.

4. Programming Task

A container is a storage that can store an unlimited number of objects.

In this lab, you are going to implement two types of the container including `Queue` and `Stack`. I have talked about `Queues` in the lecture, therefore if you need to, please revisit the lecture.

The second special type of containers is called a `stack`. A stack follows a `LIFO` (Last In, First Out) rule to insert/remove from/to the container. This means that the last item that is added to the stack, will be the first that is removed from the stack. You can imagine a stack as a plate dispenser. When you add a plate, you add it to the top of the dispenser. When you remove a plate, you remove it from the top again.

A stack has two main and two auxiliary methods:

`push(object)`: This method adds the object to the stack. In this lab we have changed the name of `push` to `add`, for the purpose of practicing polymorphism.

`pop()`: removes the last element that was added, from the stack. In this lab we have changed the name of `pop` to `remove`, for the purpose of practicing polymorphism.

`top()`: returns the element, which is at the top of the stack, without removing it.

`getSize()`: returns the number of elements in the stack.

Your job for this lab, is to complete `Container`, `Queue` and `Stack` class. I have written a comment, where you need to insert your code.

Task 1: Class Container

This class has a static final method that reads the file and creates a list that containing the read data. In fact, each line of the file is stored in one element of the list. This list is then returned from this method.

The names of the rest of the methods in this class explain themselves. If you need more clarification please read the javaDoc for these methods.

Task 2: Class Queue

While Queue Is-A container, it follows the rule of Queue data structure (FIFO), therefore all the methods should be implemented in a way that the rule is followed.

This class has an instance variable that holds all the items that is read from the file, which is labeled by "Queue". The constructor of this class removes this label and add the data to the queue.

The rest of the methods are self-explanatory.

Task 3: Class Stack

The Stack class is also is special type of a container that was explained above.

This class has a constructor that uses the list that contains the data that is read from the file and inserts the ones that are labeled with "Stack" to the instance variable of class Stack. Before adding the data, it removes the label.

To implement the rest of the methods please see their javaDoc.

5. Submit

You only submit one file that is called `Container.java` via eClass by clicking on the lab link.

You do not need to submit the tester or HTML files.