

This lab has **five (5)** checkpoints.

Learning Outcomes

Upon completing this lab, students should be able to

- Define *generic* Stack class.
- Left justify text in a given line width.
- Be able to use Gradle commandline property definition to run a specific Java class.

Introduction

1. **Do not** type the code in and compile it. You are expected to trace the execution flow.

Consider the *circular, doubly-linked* list class at the end of this document.

- (a) What are the values of the `di.nodeCounter` and `di.list` fields after the following code executes:

```
DoublyLinkedListCircularList<Integer> di = new DoublyLinkedListCircularList<Integer>();
```

Draw the structure of the `di.list` value.

- (b) *Draw* the value of `di.list` when the next line runs after the line above.

```
di.append(100);
```

- (c) *Draw* the value of `di.list` when these lines run after the line above.

```
di.append(200);  
di.append(300);  
di.append(400);  
di.append(500);
```

- (d) Finally, what is the output of the following line if it is run after all of the previous lines?

```
System.out.println(di);
```

✓ Show and explain your answers to these questions to the lab instructor. Be prepared to explain the printed values for `length` and `nodeCounter`.

2. Initialize a Gradle application project in a new directory. Also initialize `git` in the root of the project.

The name of the project: `reverse`. Starting package: `reverse`.

Change the name of `reverse.App` to `reverse.ByInt`. This lab will have two *different* executable programs; you need to use the `getProperties` version of selecting the `mainClassName`:

```
application {  
    mainClassName = project.getProperties().getOrDefault("alternateMainClass", 'reverse.ByInt')  
}
```

Delete everything in `src/test/java` so that your newly renamed application can be built with `gradle build`.

Add the `run` block to `build.gradle` that connects gradle's standard input to that of your program when using `gradle run`.

Confirm that you can build your application with `gradle build`.

Confirm that you can run your program with `gradle run -q --console=plain`.

Commit your working initial version to your local git repository.

Modify `reverse.ByInt.main` to create a `ByInt` object, ignoring command-line arguments and have it call `run`.

Your `run` method should use a `Scanner` to read `int` values from `System.in` until there are no more to read. For the moment, print out each integer in the body of the loop (below you will put them into a `Stack<Integer>`).

Compile and run your code. It should read a bunch of integers from the user and terminate (cleanly) when the input file ends (you enter "`^d`" followed by `<Enter>`).

✓ Show the lab instructor your working code run with `gradle run -q --console=plain`.

3. Add a new package, `datastructure` and a new class, `datastructure.Stack<E>`. Your `Stack<E>` should be implemented as a *singly-linked non-circular* list (you are to implement this; do not use Java library classes).

The public-facing functions you must implement are

```
class Stack<E> {
    public Stack<E>();
    public boolean isEmpty();
    public void push(E data);
    public E pop(); // return null if isEmpty()
    public E peek(); // return null if isEmpty()
}
```

To test this out, add a variable to `ByInt.run` of type `Stack<Integer>` and rather than printing out the integers as the user types them, push each into the stack. Then, after the input ends, empty the stack, printing the integers in the order they come off the stack. (**Question:** if input is 1, 2, 3, 4, 5, what will the printed output be now?)

✓ Show the lab instructor your working code run with `gradle run -q --console=plain`.

Be prepared to show them the `datastructures.Stack<E>` source code and *explain* it to them.

4. Add another Java file to the `reverse` package, `reverse.ByWord`. Set up a standard construct/call `run` main method.

`ByWord` expects one command-line parameter: the path to an input file. Pass the file name in to the constructor.

In `ByWord.run`, open the file, wrap it in a `Scanner`, read the file **by word** and push the words into a `Stack<String>`. After the input ends, dump the stack contents, separated by a single space, to standard output.

Use `gradle` to run your program (**Note:** the `\` escape, right before the end-of-line, extends the line to include the following line):

```
$ gradle run -q --console=plain -PalternateMainClass='reverse.ByWord' \
  --args=someFile.txt
```

✓ Show the lab instructor your working `reverse.ByWord`.

5. The output of `ByWord` is one *very* long line. This is ugly. Better to print out some number of lines of no more than sixty characters in width.

You will *justify* the lines of output in a *width* of 60.

How? Inside your loop that prints it all out you must check if a new line needs to be started before printing the next word. Then, after the loop finishes, start a new line after the last partial line.

In pseudocode:

```
while (stack isn't empty)
  word = stack.pop
  if line_length + word.length + 1 > linewidth
    start new line
    set line_length to zero
  else
    print a space
  print the word
```

✓ Show the lab instructor your improved `reverse.ByWord`.

6. [Not a separate checkpoint but necessary] Add and commit your full `gradle` project (if you `git add` just the root directory, `.`, `git` will add all subdirectories recursively).

Create a repository in your Gitea turn-in organization, associate it with your lab work, and push the repo. This will be done at the end of every lab.

```

public class DoublyLinkedCircularList<E> {
    private int nodeCounter;
    private DLCLNode list;    // reference to list nodes
    private class DLCLNode<E> {
        E data;
        DLCLNode prev;
        DLCLNode next;
        public DLCLNode(DLCLNode prev, E data, DLCLNode next) {
            nodeCounter++;    // count calls to this constructor
            this.prev = prev;
            this.data = data;
            this.next = next;
        }
    }
    public DoublyLinkedCircularList() {
        nodeCounter = 0;    // initialize counter (no nodes made)
        list = new DLCLNode<E>(null, null, null);
        list.next = list;
        list.prev = list;    // initialize list (empty list)
    }
    public int length() {
        int len = 0;
        for(DLCLNode curr = list.next; curr != list; curr = curr.next, len++);
        return len;
    }
    public void append(E data) {
        DLCLNode newLast = new DLCLNode<E>(list.prev, data, list);
        list.prev.next = newLast;
        list.prev = newLast;
    }
    public String toString() {
        String retval =
            String.format("length=_%d, _nodeCounter=_%d\n", length(), nodeCounter);
        String separator = "";
        for(DLCLNode curr = list.next; curr != list; curr = curr.next) {
            retval += String.format("%s%s", separator, curr.data);
            separator = ",_";
        }
        return retval;
    }
}

```