CMPT225, Spring 2021
Homework Assignment 2
Due date: Friday, February 12, 2021, 23:59

You need to implement the following classes:
Part 1:
- *assignment2/CircularLinkedList.java*

Part 2:
- *assignment2/MyStackOperations.java*

Part 3:
- *assignment2/LinkedListNode.java*

Note that all files must be in under the package (folder) *assignment2***.**
You may add more classes to your solution if necessary.

Submit all your files in ***assignment2.zip*** to CourSys. Make sure your zip file can be unzipped using the command "*unzip assignmen2.zip*" in CSIL. All your solutions in the zip file must be under the **src** folder, same as in the provided *hw2-cmpt225-spring21.zip*.

Discussion with others: You may discuss the assignment with your classmates/tutors (or anyone else), but coding must be entirely your own.

References: You may use textbooks, wiki, stack overflow, geeksforgeeks, etc. If you do, specify the references in comments. Posting questions online asking for solutions (e.g. using chegg.com) is prohibited.

Readability: Your code should be readable using the standard Java conventions. Add comments wherever is necessary. If needed, write helper functions or add classes to improve readability.

Compilation: Your code MUST compile in CSIL using javac.
Make sure that your code compiles without warnings/errors.
If the code does not compile in CSIL the grade on that part will be 0 (zero).
Even if you can't solve a problem completely, make sure it compiles.

The assignment will be graded mostly **automatically**, with some exceptions.

Do not add main() to your solutions. The main() method will be in the test files.

Warnings: Warnings during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Testing: Test your code. Examples of tests are included. Your code will be tested using the provided tests as well as additional tests. You should create more tests to check your solution.

Good luck!

Part 1 [60 points] - class CircularLinkedList<T>

In this part you need to implement a circular linked list. This is a generic class representing a *doubly linked list of objects arranged on a circle*. In contrast to the usually linked list, it does not have head or tail. Instead the data structure has a pointer to one of the nodes in the list, and operations allowing to (i) get/set the value of the node under the pointer (iii) move pointer forward and backward (iii) add a new element before or after the pointer, (iv) remove the node under the pointer, and more. Specifically, you need to implement the following constructors and operations

`public CircularLinkedList()`
creates an empty circular linked list

`public CircularLinkedList(int initSize, T initValue)`
creates a circular linked list with initial `initSize` elements, all with the value `initValue`.

`public void moveForward()`: Moves the pointer one node forward.
If the list has one node, it has no effect.
If the list is empty, throws java.util.NoSuchElementException

`public void moveBackward()`: Moves the pointer one node backward.
If the list has one node, it has no effect.
If the list is empty, throws java.util.NoSuchElementException.

`public T get()`: Returns the value in the node under the pointer.
If the list is empty, throws java.util.NoSuchElementException.

`public T set(T value)`: Sets the value under the pointer.
Returns the previous value in the node under the pointer.
If the list is empty, throws java.util.NoSuchElementException.

`public void addBefore(T value)`: Adds a node with the given value before the pointer.
If the list was empty, the pointer is set to point to the new element.

`public void addAfter(T value)`: Adds a node with the given value after the pointer.
If the list was empty, the pointer is set to point to the new element.

`public T removeBefore()`: Removes the node right before the pointer.
Returns the element previously in the removed node.
If the list is empty, throws java.util.NoSuchElementException.

`public T removeAfter()`: Removes the node right after the pointer.
Returns the element previously in the removed node.
If the list is empty, throws java.util.NoSuchElementException.

`public boolean isEmpty()`: Checks if the list is empty.

Part 2 [45 points] - class MyStackMoreOperations<T>

In this part you are given a file mystack.jar that contains inside a class mystack.MyStack.
(Really, .jar is just a zip file with the class and some additional meta information).
For your convenience, the assignment also contains the implementation MyStack.java.
You can either add the java file to the project or the provided .jar.

The class MyStack<T> implements the standard stack operations: push(), pop(), isEmpty().
However, you don't get to see the implementation of MyStack, and can access it only using its public methods.

Your goal is to import the jar file into your project (or use the MyStack.java), and implement the class MyStackMoreOperations. The class has several public static methods manipulating MyStack.
The public methods are the following.

`public static int size(MyStack<T> s)`: returns the number of elements in s.

`public static <T> MyStack<T> clone(MyStack<T> orig)`: Makes a copy of orig.

`public static void reverse(MyStack<T> s)`: Reverses the order of the elements in s.

Part 3 [15 points] - class LinkedListNode<T>

In this part you are given a class representing a node in a one directional linked list. The node has two data fields: data and next. You goal is to write the following method:

`public int countReachableNodes()`: returns the number of nodes reachable from `this`.
For example in the structures below we have:

- the number of nodes reachable from X is 4 (X,Y,Z,W)
- the number reachable from W is 1 (only W)
- The number of nodes reachable from A is 5 (A,B,C,D,E)
- The number of nodes reachable from E is 4 (E,B,C,D)