

CMPT225, Fall 2025
Homework Assignment 2
Due date: Sunday, October 12, 2025, 23:59

You need to implement the following classes:

Part 1:

- *assignment2.MyLinkedList.java*

Part 2:

- *assignment2.MyStackOperations.java*

Note that all files in your solution 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 assignment2.zip*” in CSIL. All your solutions in the zip file must be under the **src** folder, same as in the provided *cmpt225-hw2-fall25.zip*.

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.

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, please specify the references in comments. Asking others for solutions (online or in person) 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.

Do not add main() to your solutions. The main() method will be in the test files. Examples of such tests are provided in *TestMyLinkedList.java* and *TestMyStackOperations.java*.

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 [50 points] - class MyLinkedList<T>

In this part you need to implement a class MyLinkedList. This is a generic class representing a *list of objects*. The operations on the list are as follows:

- adding and removing elements from the left and from the right
- Reversing the list
- getting the size of the list
- checking if the list is empty

All operations must run in $O(1)$ time.

Specifically, you need to implement the following constructor and operations

public MyLinkedList()

Creates an empty linked list

public void addLeft(T item)

Adds the new item to the left of the list.

public void addRight(T item)

Adds the new item to the right of the list.

public T removeLeft()

Removes the leftmost item from the list and returns it.

If the list is empty, the method throws **NoSuchElementException**.

public T removeRight()

Removes the rightmost item from the list and returns it.

If the list is empty, the method throws **NoSuchElementException**.

public void reverse()

Reverses the list.

For example, if the list was 1-2-3-1-4, after running the method, it becomes 4-1-3-2-1.

public int size()

Returns the size of the list

public boolean isEmpty()

Checks if the list is empty.

Part 2 [50 points] - class `MyStackOperations<T>`

The interface `basicdatastructures.stack.Stack<T>` declares the standard operations: `push()`, `pop()`, `isEmpty()`.

Concrete implementations of a `Stack` are provided in the folder `basicdatastructures\stack\`.

Your goal is to implement the class `MyStackOperations`. The class has several public static methods manipulating a `Stack`. The public methods are as follows.

```
public static <T> int size(Stack<T> s)
```

Returns the number of elements in `s`.

After the methods return, the input stack must be in the same state as in the beginning.

```
public static <T> T removeBottom(Stack<T> s)
```

Removes the bottom element from the stack, and returns it.

The remaining elements are kept in the same order.

If `s` is empty, the method throws ***NoSuchElementException***.

```
public static <T> void reverse(Stack<T> s)
```

Reverses the order of the elements in `s`.

```
public static <T> boolean areEqual(Stack<T> s1, Stack<T> s2)
```

Checks if the two stacks have the same items in the same order.

The items in the stack are to be compared using `==` operator.

After the methods return, the input stacks must be in the same state as in the beginning.