CSCI 2720 – Fall 2025

Programming Assignment 2: Sorted Doubly Linked List

Due by 11:59 PM on October 6th, 2025 on Odin

You can complete this assignment individually or as a group of two students.

1 Project Setup

Your source code should be kept under a package `cs2720.p2`. You are free to setup your project using Maven (recommended) or by creating the directory structure manually. Regardless, you must briefly document how to compile and run your project in your README file.

2 Project Description

The purpose of this assignment is to implement a **Sorted Doubly Linked List**. This program must accept all commands described later in this document.

Unlike with Assignment 1, you will not need an `ItemType` class for this assignment. Instead, you will use generics to make your program support three different data types (`int`, `double`, and `String`). The data type will be specified by the user before any operations are run.

2.1 Java Generics

One of the goals of the assignment is to make the list operations independent of any particular data type. The list data type will be provided by the user and you will then create the appropriate list in your `main` method using generics. For example:

```
public class NodeType<T extends Comparable<T>> {
    private T info;
    private NodeType<T> next;
    private NodeType<T> back;
}

public class DoublyLinkedList<T extends Comparable<T>> {
    private NodeType<T> head;

    public void deleteItem(T item) {
        // ...
    }

    // other methods will go here
}
```

During the program's execution, this "T" will get replaced by the data type selected by the user for the linked list. You need the `Comparable` class to compare generic objects.

Your program should support storing data of type `int`, `double`, and `String` depending on the input taken from the user at the very beginning of the program. The user should be able to enter 'i' for `int`, 'd' for `double`, or 's' for `String`. Please see the following sample output:

```
$ java DoublyLinkedListDriver string-input.txt
Enter list type (i - int, d - double, s - String): s
```

In the above example, the user has provided 's' as the input. In this case, you are going to create a doubly linked list that is going to store `String` items. You will open the input file name provided from the command prompt to read `String` data and store each `String` input in the doubly linked list like you did in Assignment 1. The insert, delete, print, and other commands should likewise support `String` inputs. If the user provides 'i' or 'd', your program should be able to work with `int` or `double` types respectively.

The user is responsible for selecting an appropriate type for a given program run depending on the provided input file and the values that are expected to be stored in the list. You will be provided sample text files for each appropriate data type: one for `int`, one for `double`, and one for `String`.

Similar to Programming Assignment 1 make sure your code performs error checking for file I/O operations. I.e., it should throw an error message if the file is not present or unable to be opened correctly. You should not hardcode the input file names in the `main` method.

Additionally, your `main` method should handle exceptions related to user input. For example, if the user has selected a `double` list, then trying to insert a `String` should be considered an error. See the Sample Output for an example. As a safeguard against infinite loops, you should also terminate the program if end-of-file is sent by the user. Typically, this causes the `Scanner` to throw a `NoSuchElementException`.

3 Functional Requirements

A functional requirement is *added* to your point total if satisfied.

NodeType.java should contain your `NodeType` class that uses generics. See the following code snippet for the instance variables. You should create your own getters, setters, and constructor(s) as necessary.

```
public class NodeType<T extends Comparable<T>> {
    private T info;
    private NodeType<T> next;
    private NodeType<T> back;
}
```

DoublyLinkedList.java should contain your doubly linked list class that uses generics.

Constructors:

`public DoublyLinkedList()`

Pre-Condition: None
Post-Condition: the list is created

Functions:

`public void insertItem(T item)`

Pre-Condition: the list exists and `item` is initialized
Post-Condition: the item is inserted into the list, maintaining a sorted order. You should throw an appropriate unchecked exception if the user tries to insert a duplicate item

`public void deleteItem(T item)`

Pre-Condition: the lists exists and `item` is initialized
Post-Condition: the node that contains the item is removed from the list. You should throw an appropriate unchecked exception if the user tries to delete an item that does not exist inside the list or if the user tries to delete from an empty list.

`public int length()`

Pre-Condition: the list exists
Post-Condition: returns the length of the list

`public void print()`

Pre-Condition: the list exists
Post-Condition: items in the list are printed to standard output

`public void printReverse()`

Pre-Condition: the list exists
Post-Condition: items in the list are printed to standard output in reverse order
Note: You need to go to the end of the list and work from there.

Use the discussions from class to implement the above methods. You can make changes to the signatures of above methods, but make sure the formatting of their inputs and outputs match the sample output below.

Implement the following methods in your doubly linked list class using signatures of your choice.

`deleteSubsection` function – This method will take input from the user for the lower and upper bound (both inclusive) for a range of values that you will delete from the list.

- Example:

```
Enter lower bound: 9
Enter upper bound: 34
Original List: 3 5 10 20 34 56
Modified List: 3 5 56
```

  So in the example above you have deleted all the numbers between 9 and 34 from the list (including 34)
- Edge case: If the list has no values in the range, do nothing.

```
Enter lower bound: 15
Enter upper bound: 18
Original List: 3 5 10 20 34 56
Modified List: 3 5 10 20 34 56
```

- Edge case: If the list is empty, do nothing.

```
Enter lower bound: 3
Enter upper bound: 45
Original List:
Modified List:
```

In the README file give the pseudo code (steps) for your `deleteSubsection` operation. Using this pseudocode, explain the complexity (Big-O) of your `deleteSubsection` operation.

`reverseList` function – This method will return the reversed list. Use the original list and change the "next" and "back" references of nodes so that the list is reversed.

You are not allowed to create a new list that contains the elements in the reverse order. You will receive zero credit if you create a new list and copy the elements in the reverse order to that list.

- Examples:

```
Input list: 2 4 8 12 17 22 37
Reverse list: 37 22 17 12 8 4 2

Input list: 2
Reverse list: 2
```

In the README file give the pseudo code (steps) for your `reverseList` operation. Using this pseudocode, explain the complexity (Big-O) of your `reverseList` operation.

`swapAlternate` function – This method will swap every other node in the list. (For example: swaps nodes 1 and 2, nodes 3 and 4, nodes 5 and 6 and so on)

Note: You are not allowed to create a new list for this function. Like the `reverseList` method, modify the original list by changing the "next" and "back" references of nodes so that the nodes are swapped.

- Example:

```
Original List: 3 5 10 20 34 56
Swapped List: 5 3 20 10 56 34
```

- Edge case: If there is an odd number of nodes, leave the last node unswapped:

```
Original List: 45 67 89 102 120
Swapped List: 67 45 102 89 120
```

- Edge case: If the list is empty or has only one item then do nothing.

```
Original List:
Swapped List:

Original List: 5
Swapped List: 5
```

In the README file give the pseudo code (steps) for your `swapAlt` operation. Using this pseudocode, explain the complexity (Big-O) of your `swapAlt` operation.

Private members of `DoublyLinkedList`:

```
private NodeType<T> head;
```

- You can add helper functions (like `getHead`, etc.) that you will need to implement other functions in the class.
- You are also free to add other instance variables for a tail or header and trailer nodes.
- No functions should crash when called on an empty list. See some of the sample output below for examples.
- You can assume that the user is not going to perform insert, delete, or any other regular list functions after performing calling the reverse list and swap alternate functions.

Sample Outputs

The output of your program should match the sample output below. You don't need to match the exact whitespaces, newline etc. but your output should look like the sample output below. For the majority of functions in the sample output, the reverse list is also printed; this is to make sure that you properly link the "back" reference of the nodes in the doubly linked list.

For the following examples, the green "// comments" would not be printed by your program.

Blue text is entered by the user.

Example Output 1 (Doubly Linked List) - `int-input.txt`

```
Enter list type (i - int, d - double, s - string): i
Commands:
(i) - Insert value
(d) - Delete value
(p) - Print list
(l) - Length
(t) - Print reverse
(r) - Reverse list
(b) - Delete Subsection
(s) - Swap Alternate
(q) - Quit program


//1. Test PRINT(p)
Enter a command: p
The list is: 1 3 5 8 10 12 20


//2. Test PRINTREVERSE(t)
Enter a command: t
The reverse list: 20 12 10 8 5 3 1
```

```
//3. Test LENGTH(l)
Enter a command: l
The length of the list is 7


//4. Test INSERT(i)
//4a. Insert at the beginning
Enter a command: i
The list is: 1 3 5 8 10 12 20
Enter a number to insert: 0
The list is: 0 1 3 5 8 10 12 20
The reverse list: 20 12 10 8 5 3 1 0


//4b. Insert at the middle/end
Enter a command: i
The list is: 0 1 3 5 8 10 12 20
Enter a number to insert: 14
The list is: 0 1 3 5 8 10 12 14 20
The reverse list: 20 14 12 10 8 5 3 1 0
Enter a command: i
The list is: 0 1 3 5 8 10 12 14 20
Enter a number to insert: 33
The list is: 0 1 3 5 8 10 12 14 20 33
The reverse list: 33 20 14 12 10 8 5 3 1 0


//4c. Insert an item that already exists
Enter a command: i
The list is: 0 1 3 5 8 10 12 14 20 33
Enter a number to insert: 20
Item already exists
The list is: 0 1 3 5 8 10 12 14 20 33
The reverse list: 33 20 14 12 10 8 5 3 1 0


//5.Test DELETE(d)
//5a. Delete first element
Enter a command: d
The list is: 0 1 3 5 8 10 12 14 20 33
Enter a number to delete: 0
The list is: 1 3 5 8 10 12 14 20 33
The reverse list: 33 20 14 12 10 8 5 3 1
```

```
//5b. Delete last element or an element in the middle
Enter a command: d
The list is: 1 3 5 8 10 12 14 20 33
Enter a number to delete: 14
The list is: 1 3 5 8 10 12 20 33
The reverse list: 33 20 12 10 8 5 3 1
Enter a command: d
The list is: 1 3 5 8 10 12 20 33
Enter a number to delete: 33
The list is: 1 3 5 8 10 12 20
The reverse list: 20 12 10 8 5 3 1


//5c. Delete a non-existing item
Enter a command: d
The list is: 1 3 5 8 10 12 20
Enter a number to delete: 22
The item is not present in the list
The list is: 1 3 5 8 10 12 20
The reverse list: 20 12 10 8 5 3 1


//5d. Continue deleting until the only element left is 20 and delete the last element
//and then try to delete from an empty list
Enter a command: d
The list is: 20
Enter a number to delete: 20
Enter a command: d
Enter a number to delete: 4
You cannot delete from an empty list


//4d. Insert into empty list
Enter a command: i
Enter a number to insert: 4
The list is: 4
The reverse list: 4


//6. Test QUIT(q)
Enter a command: q
Exiting the program...
```

Example Output 2 (Doubly Linked List) - `float-input.txt`

```
Enter list type (i - int, d - double, s - string): d
Commands:
(i) - Insert value
(d) - Delete value
(p) - Print list
(l) - Length
(t) - Print reverse
(r) - Reverse list
(b) - Delete Subsection
(s) - Swap Alternate
(q) - Quit program


//1. Test PRINT(p)
Enter a command: p
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7


//2. Test PRINTREVERSE(r)
Enter a command: t
The reverse list: 100.7 84.5 63.5 45.9 37.3 19.3 9.6 3.1


//3. Test LENGTH(l)
Enter a command: l
The length of the list is 8


//4. Test INSERT(i)
//4a. Insert at the beginning of the list
Enter a command: i
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7
Enter a number to insert: 2.7
The list is: 2.7 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7
The reverse list: 100.7 84.5 63.5 45.9 37.3 19.3 9.6 3.1 2.7
```

```
//4b. Insert at the middle/end
Enter a command: i
The list is: 2.7 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7
Enter a number to insert: 19.33
The list is: 2.7 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7
The reverse list: 100.7 84.5 63.5 45.9 37.3 19.33 19.3 9.6 3.1 2.7
Enter a command: i
The list is: 2.7 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7
Enter a number to insert: 120.4
The list is: 2.7 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7 120.4
The reverse list: 120.4 100.7 84.5 63.5 45.9 37.3 19.33 19.3 9.6 3.1 2.7


//4c. Insert an item that already exists
Enter a command: i
The list is: 2.7 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7 120.4
Enter a number to insert: 19.33
Item already exists
The list is: 2.7 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7 120.4
The reverse list: 120.4 100.7 84.5 63.5 45.9 37.3 19.33 19.3 9.6 3.1 2.7


//5.Test DELETE(d)
//5a. Delete first element
Enter a command: d
The list is: 2.7 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7 120.4
Enter a number to delete: 2.7
The list is: 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7 120.4
The reverse list: 120.4 100.7 84.5 63.5 45.9 37.3 19.33 19.3 9.6 3.1


//5b. Delete last element or an element in the middle
Enter a command: d
The list is: 3.1 9.6 19.3 19.33 37.3 45.9 63.5 84.5 100.7 120.4
Enter a number to delete: 19.33
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7 120.4
The reverse list: 120.4 100.7 84.5 63.5 45.9 37.3 19.3 9.6 3.1
Enter a command: d
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7 120.4
Enter a number to delete: 120.4
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7
The reverse list: 100.7 84.5 63.5 45.9 37.3 19.3 9.6 3.1
```

```
//5c. Delete a non-existing item
Enter a command: d
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7
Enter a number to delete: 20.3
The item is not present in the list
The list is: 3.1 9.6 19.3 37.3 45.9 63.5 84.5 100.7
The reverse list: 100.7 84.5 63.5 45.9 37.3 19.3 9.6 3.1


//5d. Continue deleting until the only element left is 100.7 and delete the last
//element then try to delete from an empty list
Enter a command: d
The list is: 100.7
Enter a number to delete: 100.7
Enter a command: d
Enter a number to delete: 12.4
You cannot delete from an empty list


//4d. Insert into empty list
Enter a command: i
Enter a number to insert: 12.4
The list is: 12.4
The reverse list: 12.4


//6. Test QUIT(q)
Enter a command: q
Exiting the program...
```

Example Output 3 (Doubly Linked List) - `string-input.txt`

```
Enter list type (i - int, d - double, s - string): s
Commands:
(i) - Insert value
(d) - Delete value
(p) - Print list
(l) - Length
(t) - Print reverse
(r) - Reverse list
(b) - Delete Subsection
(s) - Swap Alternate
(q) - Quit program


//1. Test PRINT(p)
Enter a command: p
The list is: Craig Dern Ford Goodman Macy


//2. Test LENGTH(l)
Enter a command: l
The length of the list is 5


//3. Test PRINTREVERSE(t)
Enter a command: t
The reverse list: Macy Goodman Ford Dern Craig


//4. Test INSERT(i)
//4a. Insert at the beginning of the list
Enter a command: i
The list is: Craig Dern Ford Goodman Macy
Enter a string to insert: Aba
The list is: Aba Craig Dern Ford Goodman Macy
The reverse list: Macy Goodman Ford Dern Craig Aba
```

```
//4b. Insert at the middle/end
Enter a command: i
The list is: Aba Craig Dern Ford Goodman Macy
Enter a string to insert: Elsa
The list is: Aba Craig Dern Elsa Ford Goodman Macy
The reverse list: Macy Goodman Ford Elsa Dern Craig Aba
Enter a command: i
The list is: Aba Craig Dern Elsa Ford Goodman Macy
Enter a string to insert: Roger
The list is: Aba Craig Dern Elsa Ford Goodman Macy Roger
The reverse list: Roger Macy Goodman Ford Elsa Dern Craig Aba


//4c. Insert an item that already exists
Enter a command: i
The list is: Aba Craig Dern Elsa Ford Goodman Macy Roger
Enter a string to insert: Roger
Item already exists
The list is: Aba Craig Dern Elsa Ford Goodman Macy Roger
The reverse list: Roger Macy Goodman Ford Elsa Dern Craig Ab


//5.Test DELETE(d)
//5a. Delete first element
Enter a command: d
The list is: Aba Craig Dern Elsa Ford Goodman Macy Roger
Enter a string to delete: Aba
The list is: Craig Dern Elsa Ford Goodman Macy Roger
The reverse list: Roger Macy Goodman Ford Elsa Dern Craig


//5b. Delete last element or an element in the middle
Enter a command: d
The list is: Craig Dern Elsa Ford Goodman Macy Roger
Enter a string to delete: Elsa
The list is: Craig Dern Ford Goodman Macy Roger
The reverse list: Roger Macy Goodman Ford Dern Craig
Enter a command: d
The list is: Craig Dern Ford Goodman Macy Roger
Enter a string to delete: Roger
The list is: Craig Dern Ford Goodman Macy
The reverse list: Macy Goodman Ford Dern Craig
```

```
//5c. Delete a non-existing item
Enter a command: d
The list is: Craig Dern Ford Goodman Macy
Enter a string to delete: Mike
The item is not present in the list
The list is: Craig Dern Ford Goodman Macy
The reverse list: Macy Goodman Ford Dern Craig


//5d. Continue deleting until the only element left is Macy and delete the last element
//then try to delete from an empty list
Enter a command: d
The list is: Macy
Enter a string to delete: Macy
Enter a command: d
Enter a string to delete: Bob
You cannot delete from an empty list


//4d. Insert into empty list
Enter a command: i
Enter a string to insert: Bob
The list is: Bob
The reverse list: Bob


//6. Test QUIT(q)
Enter a command: q
Exiting the program...
```

Example Output 4 (Doubly Linked List) - `int-input.txt`

```
Enter list type (i - int, d - double, s - string): i
Commands:
(i) - Insert value
(d) - Delete value
(p) - Print list
(l) - Length
(t) - Print reverse
(r) - Reverse list
(b) - Delete Subsection
(s) - Swap Alternate
(q) - Quit program


//1a. Test DELETESUB(b)
Enter a command: b
Enter the lower bound: 4
Enter the upper bound: 11
The original list: 1 3 5 8 10 12 20
The modified list: 1 3 12 20
The reverse list: 20 12 3 1


//1b. Test DELETESUB(b)
Enter a command: b
Enter the lower bound: 3
Enter the upper bound: 10
The original list: 1 3 5 8 10 12 20
The modified list: 1 12 20
The reverse list: 20 12 1


//1c. Test DELETESUB(b)
Enter a command: b
Enter the lower bound: 8
Enter the upper bound: 33
The original list: 1 3 5 8 10 12 20
The modified list: 1 3 5
The reverse list: 5 3 1


//1d. Test DELETESUB(b)
Enter a command: b
Enter the lower bound: 13
Enter the upper bound: 17
The original list: 1 3 5 8 10 12 20
The modified list: 1 3 5 8 10 12 20
The reverse list: 20 12 10 8 5 3 1
```

```
//1e. Test DELETESUB(b)
Enter a command: b
Enter the lower bound: 33
Enter the upper bound: 44
The original list: 1 3 5 8 10 12 20
The modified list: 1 3 5 8 10 12 20
The reverse list: 20 12 10 8 5 3 1


//1f. Test DELETESUB(b)
Enter a command: b
Enter the lower bound: 1
Enter the upper bound: 20
The original list: 1 3 5 8 10 12 20
The modified list:


//1g. Test DELETESUB(b) empty list
Enter a command: b
Enter the lower bound: 2
Enter the upper bound: 5
The original list:
The modified list:


//1g. Insert invalid key
Enter a command: i
The list is: 0 1 3 5 8 10 12 14 20 33
Enter a number to insert: abcd
Invalid command try again: // execution continues...


//1i. Test DELETESUB(b) empty list
Enter a command: b
Enter the lower bound: abcd
Invalid command try again: // execution continues...


//1j. User sends end-of-file
Enter a command: b
Enter the lower bound: C-d // user types Ctrl+D
// Program immediately terminates
```

Example Output 5 (Doubly Linked List) - `int-input.txt`

```
Enter list type (i - int, d - double, s - string): i
Commands:
(i) - Insert value
(d) - Delete value
(p) - Print list
(l) - Length
(t) - Print reverse
(r) - Reverse list
(b) - Delete Subsection
(s) - Swap Alternate
(q) - Quit program


//1a. Test SWAPALT(s) odd length list
//You can assume that other functions like insert or delete will not be called after
//calling the swapAlternate function
Enter a command: s
The original list: 1 3 5 8 10 12 20
The modified list: 3 1 8 5 12 10 20
The reverse list: 20 10 12 5 8 1 3


//1b. Test SWAPALT(s) even length list
Enter a command: s
The original list: 1 3 5 10 12 20
The modified list: 3 1 10 5 20 12
The reverse list: 12 20 5 10 1 3


//1c. Test SWAPALT(s) length two
Enter a command: s
The original list: 3 5
The modified list: 5 3


//1d. Test SWAPALT(s) length one
Enter a command: s
The original list: 5
The modified list: 5
The reverse list: 5


//1e. Test SWAPALT(s) empty list
Enter a command: s
The original list:
The modified list:
```

Example Output 6 (Doubly Linked List) - int-input.txt

```
Enter list type (i - int, d - double, s - string): i
Commands:
(i) - Insert value
(d) - Delete value
(p) - Print list
(l) - Length
(t) - Print reverse
(r) - Reverse list
(b) - Delete Subsection
(s) - Swap Alternate


//1a. Test Reverse list(r) odd length list
//You can assume that other functions like insert or delete will not be called after
//calling ReverseList function
Enter a command: r
The original list: 1 3 5 8 10 12 20
The reversed list: 20 12 10 8 5 3 1
Enter a command: t // This is used to check the "back" of the nodes
The reverse list: 1 3 5 8 10 12 20


//1a. Test Reverse list(r) even length list
Enter a command: r
The original list: 1 3 5 10 12 20
The reversed list: 20 12 10 5 3 1
Enter a command: t // This is used to check the "back" of the nodes
The reverse list: 1 3 5 10 12 20


//1b. Test Reverse list(r) length two
Enter a command: r
The original list: 1 3
The reversed list: 3 1


//1c. Test Reverse list(r) length one
Enter a command: r
The original list: 3
The reversed list: 3


//1d. Test Reverse list(r) empty list
Enter a command: r
The original list:
The reversed list:
```

NOTE: The sample outputs for reverseList, deleteSubsections, swapAlternate functions for `double` and `String` lists are not there, but you should test these functions (like the integer case above) for `double` and `String` cases as well before submitting the assignments.

Grading Rubric:

Doubly Linked List	
insert()	10%
delete()	10%
print()	5%
length()	5%
printReverse()	5%
Delete Subsection	10%
Reverse List	15%
Swap Alternate	10%
Works on all three input types	10%
Other Requirements	
`main` method (Reading input from file and handling commands)	10%
Exception handling	10%
Total:	100%

4 Nonfunctional Requirements

4.1 Project Structure

Your implementation should match the provided package structure and visibility of each class and method within reason. Use your best judgment when writing your own classes/methods not explicitly listed in this document. (10–100 points)

4.2 Project Dependencies

Your project must exclusively use your `NodeType` class to build the linked list. You may not use arrays or any Java collections. (100 points)

4.3 Compiling and Running

Code that fails to compile or code that compiles but fails to run will receive a grade of zero. Your program must accept a single command-line argument as a path to the input file.

4.4 README

Commands to run and compile the program should be documented clearly in the README file.

You must include your full name and university email address in your README file. If you are doing a project in a group of two, list the full names and email addresses of all two group members. If you are in a group, you must also describe the contributions of each group member in the assignment. (5 points)

4.5 Style Requirements

All submitted source files must conform to the CSCI 1302 style guide. Please see the guide on eLC for setting up style checks. (5 points for each style violation, up to 20)

A lack of appropriate inline comments may also constitute improper documentation.

4.6 Unit Tests

You are *encouraged* to write unit tests for your project. In CSCI 1302, you may have done this using custom classes, variables, and methods. Here, we recommend using a framework like JUnit to test your linked list features. Like integrating Checkstyle with your build process, testing frameworks provide another convenient way to speed-up your development process.

A sample test file is included on eLC for Project 1. Although the package for `SortedLinkedListTest` is the same as `SortedLinkedList`, note that the test file is under `src/test/java/cs2720/p1`, while the main file is under `src/main/java/cs2720/p1`. Feel free to modify this template file to adhere to the Project 2 requirements.

Check out the JUnit 5 documentation for more details.

You should also make sure your JUnit and Surefire dependencies are compatible.

See the Project 1 instructions on eLC for more details.

4.6.1 Note on Checkstyle

Unit tests are meant to be descriptive by design, particularly since they are often only a few lines of code. Because of this, you are not explicitly required to Javadoc methods that are given the `@Test` annotation (aka the actual unit tests that are run). Bear in mind that you are still expected to document a longer test whose purpose is not immediately clear.

4.7 Late Submission Policy

Projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score.
- If submitted 24-48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.
- Assignment extensions will not be granted, regardless of the situation. If you need extra time, submit up to two days late and consider the late penalty waiver option discussed in the syllabus.

4.8 Submission Notes

You must include your full name and university email address in your README.txt file. If you are doing a project in a group of two, list the full names and email addresses of all two group members. If you are in a group, you must also describe the contributions of each group member in the assignment.

Submit the following files on Odin:

- DoublyLinkedList.java
- NodeType.java
- DoublyLinkedListDriver.java
- README.txt

Odin Login Instructions

Projects should be submitted through the School of Computing's Odin server. You can ssh into Odin using your myID credentials. For example,

```
$ ssh abc12345@odin.cs.uga.edu
```

Group Submissions:

****Note: This assignment can be done individually or in a group of two students. The members of the group should be from the same section. Submit a README file with your names, UGA emails and any other information that you want to give the TA to run your code. If you are doing it in a group of two make sure to include the contribution of each student in the README file.**

If you are working in a group of two, your submission folder must be named as follows:

```
LastName1_LastName2_assignment2
```

When you are submitting the folder you should use the submit command as follows:

```
$ submit LastName1_LastName2_assignment2 <odin-submission-destination>
```

Make sure to replace "`<odin-submission-destination>`" with the correct destination from the table on the bottom of this document.

If you are working in a group, you must **submit the assignment from the Odin account of only one of the members** in the group

Individual Submissions:

If you are working individually, your submission folder must be named as follows:

```
LastName_assignment2
```

Submission command should be used as follows:

```
$ submit LastName_assignment2 <odin-submission-destination>
```

Make sure to replace "`<odin-submission-destination>`" with the correct destination from the table on the bottom of this document.

****For all submissions, the folder name and README file are very important in order to be able to locate your assignment and grade it.**

****Note: Make sure you submit your assignment to the right folder to avoid grading delays.**

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

You can check your current working directory by typing:

```
$ pwd
```

For example if your assignment is in the following directory,

`/documents/csci2720/assignments/LastName_assignment2`

Before executing the submit command, change to the assignment's immediate parent directory:

```
$ cd /documents/csci2720/assignments/
```

Then execute the submit command:

```
$ submit LastName_assignment2 <odin-submission-destination>
```

Make sure to replace "`<odin-submission-destination>`" with the correct destination from the table on the bottom of this document.

To check if your submission command was valid, there should be a receipt file.

```
$ ls LastName_assignment2
```

This should include something like "`rec54321`."

Selecting the Odin Submission Destination

You must select your Odin submission destination using the following mapping based on the section of the course you are registered to.

Section	Class Time	Odin Submission Destination
CRN 42634	TR 11:10 AM	`csci-2720f`