This work is about creating a double-linked list using certain methods and modifying and editing this linked-list.

The work was created in NetBeans environment and consists of 4 classes. These classes are DLNode, HW2Interface, LinkedList and Onat\_Bulut\_HW2 respectively. Among these classes, HW2Interface is created in interface type which is a class type.

To briefly explain the content of these classes, the DLNode class is the class that contains the basis for creating the nodes of the double-link list to be created, the HW2Interface class is a template designed to implement the methods to be used, the LinkedList class is the class where the algorithm of the methods to be used is built, and the Onat\_Bulut\_HW2 class is the class where these methods are called and applied on the list.

The methods used in the study are Insert, Delete, ReverseLink, SquashL, OplashL, Output, ROutput, toString and LinkedListException.

The purpose of the Insert method is to add the desired element to the list at the desired position. To explain the algorithm designed for this purpose, at first a node with the desired element value is created. This node will be added to the list at the desired position according to the value of the selected position. If the value entered is less than 0, since this value is an invalid value, an error is thrown using the LinkedListException method and the created node is not included in the list. If the value entered is 0, there are two cases. If the element head node is empty, the created node is directly included in the head of the list. If the head node is full, the created node is included just before the head node and the new head node becomes the created node.

The purpose of the Delete method is to remove the node at the specified position in the list. It is applied differently based on the value of the desired position. If the value is less than 0, an error is thrown as it is an invalid position. In the case where the value is 0, the position of the head node is moved to the next node, effectively removing the old head node. If the value is greater than 0, the method moves to the desired position with the temp node, and the connections of the node at that position with its right and left nodes are severed and linked to each other. This way, the node at that position is deleted.

The purpose of the ReverseLink method is to reverse a list. In the algorithm designed for this purpose, a node named current is created to represent the head node, and it traverses the list from the beginning to the end, swapping the right and left connections of each node. In this swapping process, a temporary node named temp, which temporarily holds the left connection, is used. Finally, the head node is added to the end of the new list.

The purpose of the SquashL method is to insert the sequential occurrence count of each element before that element in the list. For example, a list consisting of elements 1 1 2 3 will result in 1 2 2 1 3 1 after applying this method. In the algorithm designed to achieve this, operations are performed using a temp node representing the head node. This method occurs when the temp node is not empty. In the first stage, the algorithm checks how many consecutive occurrences there are for each element using a count variable. The initial value of the count variable is set to 1. This is because if the element in the list is not consecutive, a node with a value of 1 is added to its beginning. If the examined element has a consecutive occurrence, the count value becomes greater than 1. In this case, a new node holding the count value is created and added to the end of consecutive elements.

However, only one occurrence of the consecutive element should remain in the list. Therefore, the number of elements equal to one less than the count value is removed from the list, resulting in the final state of the list with elements followed by their counts.

The purpose of the OplashL method is to revert the original list before the application of the SquashL method. The algorithm designed for this involves adding to the list a number of elements equal to one less than the previously determined count of consecutive elements. Subtracting one is necessary because there is already one occurrence of that element in the list. Initially, a temp node is created to determine how many times the element occurs consecutively, and this information is stored in the count variable. If the count value is 1, it means that the element is consecutively repeated only once. In this case, the node holding the element value is kept constant, and the node containing the count of consecutiveness is removed. Subsequently, the temp node is shifted two nodes to the right to examine the consecutiveness of other nodes. If the count value is greater than 1, the node containing the count of consecutiveness is again removed. This time, a node containing the value of a number of elements equal to one less than this count is added. Temp is then shifted one node to the right to examine the consecutiveness of other nodes.

The purpose of the Output method is to print the list, which is initially created using the Insert method and later modified by other methods, to the console. For this purpose, a current node pointing to the head node is created, and a loop is used to traverse the list from the beginning to the end, printing the value of each node to the console. This loop continues as long as the current value is not empty.

The purpose of the ROutput method is to print the list in reverse order to the console. For this purpose, a current node pointing to the head node is initially created, and it traverses to the last node of the list. Then, it moves backward from that node, printing the value of each node to the console. The writing process concludes when the current value is empty.

The purpose of the toString method is the same as the Output method; however, its implementation is different. This method should return a string variable. Therefore, initially, an empty string variable named list is defined along with the current node. While traversing the list from the beginning to the end, the values are added to the list variable. Thus, the list variable, which is eventually returned, contains the elements to be printed to the console.

The purpose of the LinkedListException method is to encapsulate the exception thrown in the Insert and Delete methods. In this case, whenever an exception is thrown, the message "Invalid position" is displayed on the console.

In summary, this project involves creating a doubly linked list, modifying and organizing it using various algorithms, and utilizing interfaces. The implementation details of how these algorithms work have been covered, and there is a focus on exception handling as well.