Module Twelve - Assignment Submission

The Assignment

> The Generics Assignment is to convert a non-generic Java class into a generic Java class and demonstrate functionality by instantiating with several different type The source code for the Doubly Linked List example in Java was copied, modified to move the attributes to the bottoms of classes, make the attributes private to Help information hiding, and comments changed from C to Java style and then saved in this Java source file The DLL class is written as a Doubly Linked List of ints. Modify the class to be generic and demonstrate with the following elements of the three types Integer: 1, 5, 10 Double: 2.0, 6.0, 12.0 String: "Dog", "Cat", "Horse" Submit per the "Assignment and Project Submission" doc

Design

There's really not much to design here. All we have to do is genericize the nodes and the functions of the class. Then, just write a function that constructs and displays the elements desired.

Implementation

```
package module12;

// A complete working Java program to demonstrate all
// Doubly Linked list Node
class Node<T> {

    // Constructor to create a new node
    // next and prev is by default initialized as null
    Node(T data) {
        this.data = data;
    }

    public T getData() {
        return this.data;
    }
    public void setData(T data) {
        this.data = data;
    }
    public Node<T> getPrev() {
        return this.prev;
    }
    public void setPrev(Node<T> prev) {
        this.prev = prev;
    }
    public Node<T> getNext() {
        return this.next;
    }
    public void setNext(Node<T> next) {
```

```
        this.next = next;
    }

    private T data;
    private Node<T> prev;
    private Node<T> next;
}

// Class for Doubly Linked List
public class DLL<T> {
    // Adding a node at the front of the list
    public void push(T new_data) {
        // 1. allocate node
        // 2. put in the data
        Node<T> new_Node = new Node<>(new_data);

        // 3. Make next of new node as head and previous as NULL
        new_Node.setNext(this.head);
        new_Node.setPrev(null);

        // 4. change prev of head node to new node
        if (this.head != null) {
            this.head.setPrev(new_Node);
        }

        // 5. move the head to point to the new node
        this.head = new_Node;
    }

    // Given a node as prev_node, insert a new node after the given node
    public void InsertAfter(Node<T> prev_Node, T new_data) {

        // 1. check if the given prev_node is NULL
        if (prev_Node == null) {
            System.out.println("The given previous node cannot be NULL ");
            return;
        }

        // 2. allocate node
        // 3. put in the data
        Node<T> new_node = new Node<>(new_data);

        // 4. Make next of new node as next of prev_node
        new_node.setNext(prev_Node.getNext());

        // 5. Make the next of prev_node as new_node
        prev_Node.setNext(new_node);

        // 6. Make prev_node as previous of new_node
        new_node.setPrev(prev_Node);

        // 7. Change previous of new_node's next node
        if (new_node.getNext() != null) {
            new_node.getNext().setPrev(new_node);
```

```
        }
    }

    // Add a node at the end of the list
    void append(T new_data) {
        // 1. allocate node
        // 2. put in the data
        Node<T> new_node = new Node<>(new_data);

        Node<T> last = this.head; // used in step 5

        // 3. This new node is going to be the last node, so make next of
it as NULL
        new_node.setNext(null);

        // 4. If the Linked List is empty, then make the new node as head
        if (this.head == null) {
            new_node.setPrev(null);
            this.head = new_node;
            return;
        }

        // 5. Else traverse till the last node
        while (last.getNext() != null) {
            last = last.getNext();
        }

        // 6. Change the next of last node
        last.setNext(new_node);

        // 7. Make last node as previous of new node
        new_node.setPrev(last);
    }

    // This function prints contents of linked list starting from the given
node
    public void printlist(Node<T> node) {
        Node<T> last = null;
        System.out.println("Traversal in forward Direction");
        while (node != null) {
            System.out.print(node.getData() + " ");
            last = node;
            node = node.getNext();
        }
        System.out.println();
        System.out.println("Traversal in reverse direction");
        while (last != null) {
            System.out.print(last.getData() + " ");
            last = last.getPrev();
        }
    }

    /* Drier program to test above functions */
    public static void main(String[] args) {
```

```
        /* Start with the empty list */
        DLL<Integer> dll_int = new DLL<>();
        DLL<String> dll_str = new DLL<>();
        DLL<Double> dll_dbl = new DLL<>();

        // Integer: 1, 5, 10
        dll_int.append(1);
        dll_int.push(5);
        dll_int.push(10);

        // String: "Dog", "Cat", "Horse"
        dll_str.append("Dog");
        dll_str.push("Cat");
        dll_str.push("Horse");

        // Double: 2.0, 6.0, 12.0
        dll_dbl.append(2.0);
        dll_dbl.push(6.0);
        dll_dbl.push(12.0);

        System.out.println("Integer List:");
        dll_int.printlist(dll_int.head);
        System.out.println("\nString List:");
        dll_str.printlist(dll_str.head);
        System.out.println("\nDouble List:");
        dll_dbl.printlist(dll_dbl.head);
    }

    private Node<T> head; // head of list
}
```

Output

```
Integer List:
Traversal in forward Direction
10 5 1
Traversal in reverse direction
1 5 10
String List:
Traversal in forward Direction
Horse Cat Dog
Traversal in reverse direction
Dog Cat Horse
Double List:
Traversal in forward Direction
12.0 6.0 2.0
Traversal in reverse direction
```