In order to change the starting code to generic I started with the node class adding the type parameter which is <T> and this gets specified later in the code at the main method. Also did this for the class DLL to make sure that when added to the collection it is able to be any type of numeric type. Then class DLL was changed to a generic to allow for the allocation of the different numeric types to be able to be allocated with out any casting. Once those were converted to generics I was able to input different the three different types into the doubly linked list. Started first with the integers by declaring referencing the generic class by doing class-name<type-arg-list> var-name = new class-name<type-arg-list>(). Then using the methods of append, and push to implement the integer numbers. This was repeated to allow for the Double, and String type.

package generic;

```
package generic;

//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<T>(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<T>(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<T>(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 = new DLL<Integer>();
        dll.append(1);
        dll.push(5);
        dll.push(10);
        System.out.println("Created DLL is: ");
        dll.printlist(dll.head);

        DLL<Double> dll2 = new DLL<Double>();
        dll2.append(2.0);
        dll2.push(6.0);
        dll2.push(12.0);
        System.out.println("\nCreated DLL2 is: ");
        dll2.printlist(dll2.head);

        DLL<String> dll3 = new DLL <String>();
        dll3.append("Dog");
        dll3.push("Cat");
        dll3.push("Horse");
        System.out.println("\nCreated DLL3 is: ");
        dll3.printlist(dll3.head);
    }

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

```
<terminated> DLL (1) [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (Apr 17, 2021, 9:06:05 AM)
Created DLL is:
Traversal in forward Direction
10 5 1
Traversal in reverse direction
1 5 10
Created DLL2 is:
Traversal in forward Direction
12.0 6.0 2.0
Traversal in reverse direction
2.0 6.0 12.0
Created DLL3 is:
Traversal in forward Direction
Horse Cat Dog
Traversal in reverse direction
Dog Cat Horse
```