Національний технічний університет України
«КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки
Кафедра Інформаційних Систем та Технологій

# Комп’ютерний практикум №1

з дисципліни «Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

на тему

## «Узагальнені типи (Generic) з підтримкою подій. Колекції»

Виконав:
студент гр. ІС-11
Побережний Олександр

Київ 2023

## Лістинг програмного коду:

```
public class CustomLinkedList<T> : ICollection<T>
 {
     public delegate void OnActionEventHandler(object sender, EventArgs e);
     public event OnActionEventHandler OnAdding;
     public event OnActionEventHandler OnRemoving;
     public event OnActionEventHandler OnCleared;
     public event OnActionEventHandler OnCopied;
     public event OnActionEventHandler OnEndPlaced;
     public event OnActionEventHandler OnBeginPlaced;

     public Node<T>? First { get; private set;}
     public Node<T>? Last { get; private set;}
     public int Count { get; private set;}
     public bool IsReadOnly => false;

     public CustomLinkedList()
     {
         this.First = null;
         this.Last = null;
     }

     public void AddFirst(Node<T> node)
     {
         if(this.First == null && this.Count == 0)
         {
             this.First = node;
             this.Last = node;
         }
         else
         {
             this.First!.Previous = node;
             node.Next = this.First;
             this.First = node;
         }
         Count++;

         OnBeginPlaced(this, EventArgs.Empty);
     }

     public void AddLast(Node<T> node)
     {
         if (this.First == null && this.Count == 0)
         {
             this.First = node;
             this.Last = node;

         }
         else
         {
             this.Last!.Next = node;
             node.Previous = this.Last;
             this.Last = node;
         }
         Count++;

         OnEndPlaced(this, EventArgs.Empty);
     }

     public void Clear()
     {
         First = null;
         Last = null;
         Count = 0;
```

```
    OnCleared(this, EventArgs.Empty);
}

public void AddBefore(Node<T> newNode, Node<T> oldNode)
{
    if (this.First == null && this.Count == 0)
    {
        this.AddFirst(newNode);
    }
    else if (this.First == oldNode)
    {
        AddFirst(newNode);
    }
    else
    {
        Node<T>? prevNode = oldNode.Previous;

        newNode.Previous = prevNode;
        newNode.Next = oldNode;
        oldNode.Previous = newNode;
        prevNode!.Next = newNode;

    }
    Count++;

    OnAdding(this, EventArgs.Empty);
}

public void AddAfter(Node<T> newNode, Node<T> oldNode)
{
    if(this.First == null && this.Count == 0)
    {
        AddFirst(newNode);
    }

    if(this.Last == oldNode)
    {
        AddLast(newNode);
    }

    if(this.First == oldNode)
    {
        AddFirst(newNode);
    }

    Node<T>? prevNode = oldNode.Previous;

    prevNode!.Next = newNode;
    newNode.Previous = oldNode.Previous;
    newNode.Next = oldNode;
    oldNode.Previous = newNode;

    Count++;

    OnAdding(this, EventArgs.Empty);
}

public void RemoveFirst()
{
    this.First = this.First!.Next;
    this.First!.Previous = null;
    Count--;

    OnRemoving(this, EventArgs.Empty);
}

public void RemoveLast()
```

```
        {
            this.Last = this.Last.Previous;
            this.Last!.Next = null;
            Count--;

            OnRemoving(this, EventArgs.Empty);
        }

        public bool Remove(T node)
        {
            if(this.First == null && this.Count == 0)
            {
                Console.WriteLine($"Nothing to remove");
                return false;
            }
            else if(First!.Data!.Equals(node))
            {
                RemoveFirst();
                return true;
            }
            else
            {
                Node<T>? prevNode = First;
                Node<T>? currNode = prevNode.Next;

                while(currNode != null && !currNode.Data!.Equals(node))
                {
                    prevNode = currNode;
                    currNode = prevNode.Next;
                }

                if (currNode != null)
                {
                    prevNode.Next = currNode.Next;
                    currNode.Next = currNode.Previous;
                }
                Count--;
                OnRemoving(this, EventArgs.Empty);
                return true;

            }
        }

        public void Add(T item)
        {
            AddLast(new Node<T>(item));

            OnAdding(this, EventArgs.Empty);
        }

        public bool Contains(T item)
        {
            Node<T> node = First;
            while(!node!.Data!.Equals(item) && node.Next != null)
            {
                node = node.Next;
            }
            if (node.Data.Equals(item)) return true;
            else return false;

        }

        public void CopyTo(T[] array, int arrayIndex)
        {
            if (arrayIndex < 0 || arrayIndex > array.Length) throw new
ArgumentOutOfRangeException(nameof(arrayIndex));
```

```
        if (array.Length - arrayIndex < Count) throw new Exception("Not enough space
in array");

        Node<T>? node = First;

        while(node != null)
        {
            array[arrayIndex++] = node.Data;
            node = node.Next;
        }

        OnCopied(this, EventArgs.Empty);
    }

    public Node<T> Find(T item)
    {
        Node<T> node = First!;
        EqualityComparer<T> comparer = EqualityComparer<T>.Default;
        if(node != null)
        {
            if(item != null )
            {
                while (node != null)
                {
                    if (comparer.Equals(node.Data, item))
                    {
                        return node;
                    }
                    node = node.Next!;
                }
            }
            else
            {
                while(node != null)
                {
                    if(node!.Data == null)
                    {
                        return node;
                    }
                    node = node.Next!;
                }
            }
        }
        return null;
    }

    public IEnumerator<T> GetEnumerator()
    {
        Node<T> node = First;

        while(node != null)
        {
            yield return node.Data;
            node = node.Next;
        }
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }
 }
public class MessageManager
{
    public void OnAdding(object sender, EventArgs e) =>
Console.WriteLine("Successfully added item");
    public void OnRemoving(object sender, EventArgs e) =>
Console.WriteLine("Successfully removed item");
```

```
    public void OnCleared(object sender, EventArgs e) =>
Console.WriteLine("Successfully cleared te list");
    public void OnCopied(object sender, EventArgs e) =>
Console.WriteLine("Successfully copied the list into array");
    public void OnEndPlaced(object sender, EventArgs e) =>
Console.WriteLine("Successfully added to the end");
    public void OnBeginPlaced(object sender, EventArgs e) =>
Console.WriteLine("Successfully added to the begin");

    public void InitHandlers<T>(CustomLinkedList<T> list)
    {
        list.OnAdding += this.OnAdding;
        list.OnRemoving += this.OnRemoving;
        list.OnCleared += this.OnCleared;
        list.OnCopied += this.OnCopied;
        list.OnEndPlaced += this.OnEndPlaced;
        list.OnBeginPlaced += this.OnBeginPlaced;
    }

    public void RemoveAddingHandler<T>(CustomLinkedList<T> list) => list.OnAdding -=
this.OnAdding;
    public void RemoveRemovingHandler<T>(CustomLinkedList<T> list) => list.OnRemoving
-= this.OnRemoving;
    public void RemoveClearHandler<T>(CustomLinkedList<T> list) => list.OnCleared -=
this.OnCleared;
    public void RemoveCopyHandler<T>(CustomLinkedList<T> list) => list.OnCopied -=
this.OnCopied;
    public void RemoveEndPlaceHandler<T>(CustomLinkedList<T> list) => list.OnEndPlaced
-= this.OnEndPlaced;
    public void RemoveBeginPlaceHandler<T>(CustomLinkedList<T> list) =>
list.OnBeginPlaced -= this.OnBeginPlaced;

}class Program
{
    static void InitHandlers<T>(CustomLinkedList<T> list, MessageManager mgr)
    {
        list.OnAdding += mgr.OnAdded;
        list.OnRemoving += mgr.OnRemoved;
        list.OnCleared += mgr.OnCleared;
        list.OnCopied += mgr.OnCopied;
        list.OnEndPlaced += mgr.OnEndPlaced;
        list.OnBeginPlaced += mgr.OnBeginPlaced;
    }

    static void ShowList<T>(ICollection<T> list)
    {
        foreach (T item in list)
        {
            Console.WriteLine(item);
        }
    }
    static void Main(string[] args)
    {
        CustomLinkedList<string>  list = new CustomLinkedList<string>();
        MessageManager mgr = new MessageManager();
        LinkedList<string> lst = new LinkedList<string>();

        InitHandlers(list, mgr);

        Console.WriteLine("Adding elements to the end:");

        for (int i = 0; i < 4; i++)
        {
            list.AddLast(new Node<string>($"element{i}"));
        }
        ShowList(list);

        Console.WriteLine();
```

```
Console.WriteLine("Adding items to the begin:");

list.AddFirst(new Node<string>("element10"));
ShowList(list);

Console.WriteLine();



Console.WriteLine("Adding before specified item:");

Node<string> elemX = new Node<string>("elementX");
Node<string> elem5 = new Node<string>("element5");
list.AddLast(elemX);
list.AddBefore(elem5, elemX);
ShowList(list);

Console.WriteLine();



Console.WriteLine("Adding after specified item:");

Node<string> elem6 = new Node<string>("element6");
list.AddAfter(elem6, elem5);
ShowList(list);

Console.WriteLine();


Console.WriteLine("Finding items:");

Console.WriteLine($"Found item - {list.Find("element2").Data}");

Console.WriteLine();


Console.WriteLine("Removing items:");

list.Remove("element3");
ShowList(list);

Console.WriteLine();



Console.WriteLine("Removing first item:");

list.RemoveFirst();
ShowList(list);

Console.WriteLine();


Console.WriteLine("Removing last item:");

list.RemoveLast();
ShowList(list);

Console.WriteLine();



Console.WriteLine("Copy into the array:");
```

```
            string[] arr = new string[list.Count];
            Console.WriteLine("Items of an array:");
            ShowList(arr);
            list.CopyTo(arr, 0);
            Console.WriteLine($"New items of the array");
            ShowList(arr);

            Console.WriteLine();


            Console.WriteLine("Find out if list contains item 'element11':");

            Console.WriteLine(list.Contains("element11"));

            Console.WriteLine();


        }
}
```

## Результати виконання:

```
Adding elements to the end:
Successfully added to the end
Successfully added to the end
Successfully added to the end
Successfully added to the end
element0
element1
element2
element3

Adding items to the begin:
Successfully added to the begin
element10
element0
element1
element2
element3

Adding before specified item:
Successfully added to the end
Successfully added item
element10
element0
element1
element2
element3
element5
elementX

Adding after specified item:
Successfully added item
element10
element0
element1
element2
element3
element6
element5
elementX
```

```
Finding items:
Found item - element2

Removing items:
Successfully removed item
element10
element0
element1
element2
element6
element5
elementX

Removing first item:
Successfully removed item
element0
element1
element2
element6
element5
elementX

Removing last item:
Successfully removed item
element0
element1
element2
element6
element5

Copy into the array:
Items of an array:




Successfully copied the list into array
New items of the array
element0
element1
element2
element6
element5

Find out if list contains item 'element11':
False
```