Команда (Дія) - це поведінковий патерн проєктування, який перетворює запити на об'єкти, даючи змогу передавати їх як аргументи під час виклику методів, ставити запити в чергу, логувати їх, а також підтримувати скасування операцій.

На даний момент цього патерна в моєму проекті немає, але є в моїх попередній лабі.

[Посилання на попередню лабораторну.]

В моїх роботі є інтерфейс користувача та програмна частина. В першій частині моїй лабі, мені потрібно було зробити свій список, який буде сортуватися різними методами, а також реалізувати легке їх під'єднання. Тобто в мене в інтерфейсі є багато кнопок, які відповідають за різні методи сортування та один список для сортування. І для підключення методів сортування я використав патерн **Команда.**

На скріншоті обведено кнопки, які відповідають за методи сортування.

```
public partial class Task1 : Window
{
    /// <summary>
    /// Main list to sort. This list are used for adding new elements, sorting by different methods.
    /// </summary>
    MyList<int> listToSort = new MyList<int> { };
    /// <summary>
    /// Sort event handler. Event for containing different sort methods.
    /// </summary>
    SortMethodsInvoker<int> subscribeSortMethods = new SortMethodsInvoker<int>();
    /// <summary>
    ///  Initialize UI and some elements.
    /// </summary>
```

В основному класі в мене є 2 поля. Перше поле це список для сортування, а інше це клас який підписує методи.

Цей клас виглядає так.

```
2 references
public class SortMethodsInvoker<T> where T : IComparable
{
    public MyList<T> listToSort;
    public SortEventHandler<T> subscribeSortMethods = new SortEventHandler<T>();
    public TextBox textBox;
    7 references
    public void SubscribeAndSort(ISortMethod<T> methodForSigning)
    {
        subscribeSortMethods.sortMethod += methodForSigning.Sort;
        subscribeSortMethods.Sort(listToSort);
        subscribeSortMethods.sortMethod -= methodForSigning.Sort;
        PrintList();
    }
    1 reference
    void PrintList()
    {
        textBox.Text = "";
        int iterator = 0;
        foreach (var item in listToSort)
        {
            textBox.Text += iterator == 0 ? item : " " + item;
            iterator++;

        }
    }
}
```

В ньому є вказівник на лист, який сортується, а також Подія до якої підписуються методи сортування, ще є вказівник на текстбокс. Метод `SubscribeAndSort` приймає метод сортування, підписує його та сортує список. Метод `PrintList` виводить елементи на текстбокс.

```
0 references
public Task1()
{
    InitializeComponent();
    subscribeSortMethods.listToSort = listToSort;
    subscribeSortMethods.textBox = textBox;
}
```

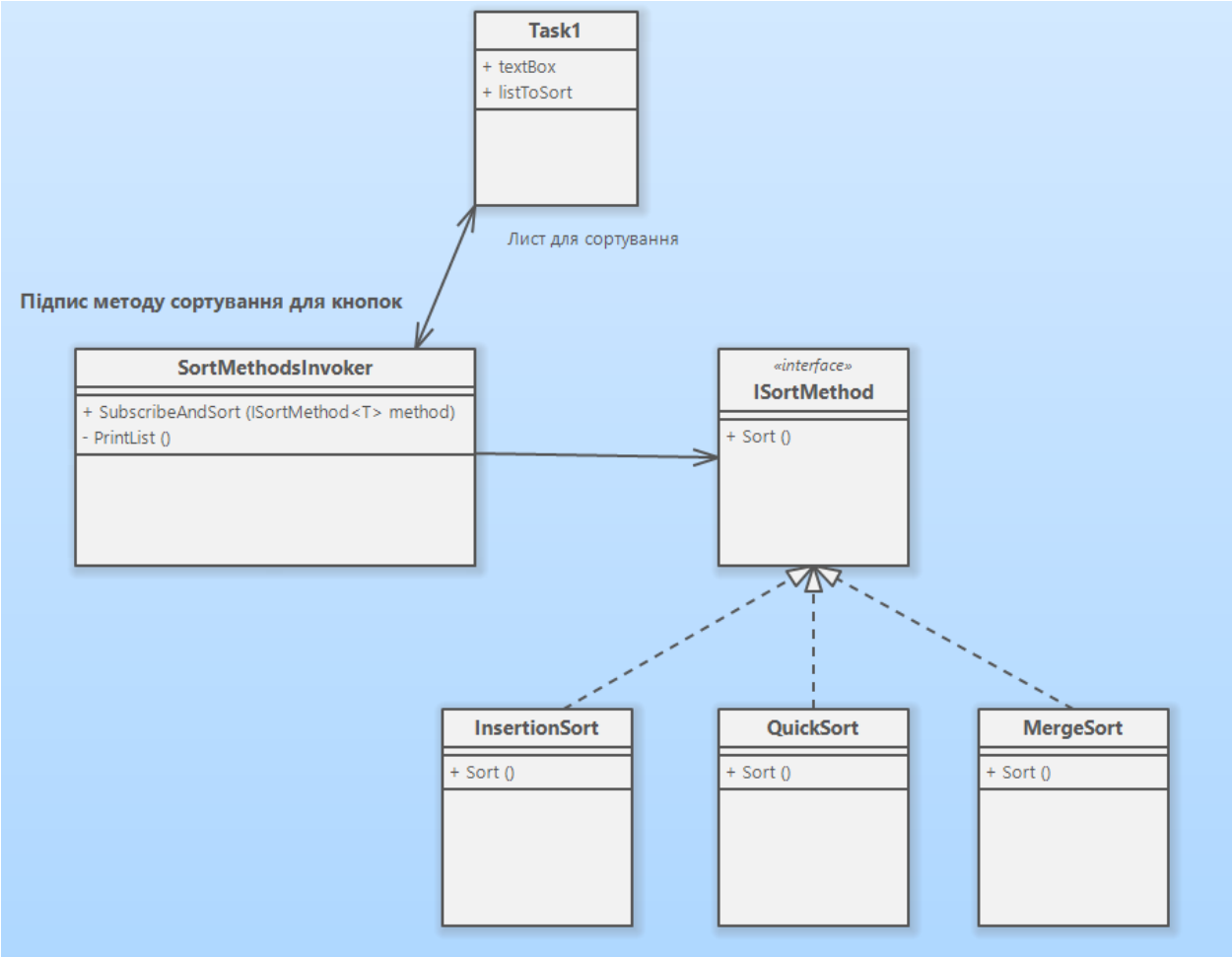
Метод ініціалізації дає нашому класу для підписування методів, всі потрібні елементи.

Після чого кожна кнопка-сортування має такий функціонал. Всередині неї створюється метод та визивається функція для підписання та сортування.

```
private void Button_InsertionSort(object sender, RoutedEventArgs e)
{
    var insertion = new InsertionSort<int>();
    subscribeSortMethods.SubscribeAndSort(insertion);
}
```

```
private void Button_QuickSort(object sender, RoutedEventArgs e)
{
    var quickSort = new QuickSort<int>();
    subscribeSortMethods.SubscribeAndSort(quickSort);
}
```

```
private void Button_MergeSort(object sender, RoutedEventArgs e)
{
    var mergeSort = new MergeSort<int>();
    subscribeSortMethods.SubscribeAndSort(mergeSort);
}
```



Тобто, всі методи сортування реалізують інтерфейс, за допомогою цього інтерфейсу клас SortMethodsInvoker прив’язує метод до листа (який потрібно відсортувати). А методи кнопок в основному класі використовують цей клас для сортування окремими методами.

Посилання на попередню лабораторну.

Класи можете знайти за шляхом:

1a\After\Task1.xaml.cs

1a\After\Task1\SortMethodsInvoker.cs

1a\After\Task1\SortingMethods.cs

1a\After\Task1\Sorting.cs

1a\After\Task1\MyList.cs