PRE-LAB

1. What are the Advantages of Delegates and Events?

Solution:

- 1. Decoupling of Components: Delegates and events allow components to be loosely coupled. Publishers can raise events without needing to know or directly reference specific subscribers. This promotes better separation of concerns and easier maintenance of the codebase.
- 2. Flexibility and Extensibility: Delegates and events provide a flexible way to extend functionality. New subscribers can be easily added to handle events without modifying existing code, enabling easy extensibility of applications.
- 3. Asynchronous Programming: Delegates and events can be used to implement asynchronous programming patterns. This allows applications to perform long-running tasks without blocking the main thread, improving responsiveness and user experience.
- 4. Encapsulation and Information Hiding: Events allow classes to encapsulate their internal state and implementation details while still allowing external components to be notified of important changes. This helps maintain a clean and understandable API.
- 5. Support for Observer Pattern:Delegates and events provide a way to implement the Observer pattern, where an object (the subject) maintains a list of dependents (observers) that are notified of state changes. This pattern is useful for implementing UI updates, event-driven systems, and more.

IN-LAB:

TASK1: Task Delegates and events refers to task Collections

To add the following new functionalities to the project created in task Collections:

Include two events in the **CustomArray** type:

- The **OnChangeElement** event occurs when the indexer changes the element value (if the old and new element values match, the event is not raised)
- The **OnChangeEqualElement** event occurs if a value equal to the index of the changed element is written to the element (if the old and new values of the element match, the event is not raised)

Use the **ArrayHandler** delegate to create the event.

The event handler takes two parameters: an object **sender** - a reference to the **CustomArray** instance that generated the event, and an event argument **ArrayEventArgs** <**T**> **e**. In the event argument, write in the **Id** field the index by **Solution:**

which the user changes the element of the **CustomArray** array, in the **Value** field - the new value of the element by the Id index, in the **Message** field - an arbitrary string message.

```
        od
        File
        Edit
        View
        Git
        Project
        Build
        Debug
        Test
        Analyze
        Tools
        Extensions
        Window
        Help
        ✓ Search **
        Iab7

                                                                                                                                                                                                                                                                                                                                                     Sign in ⅓ — ♂ ×
 © - ⊕ ⊕ ⊕ © 9 - € - Debug - Any CPU - ▶ Continue - € - □ □ □ Uferyde Events - Thread:

Process: [12420] labTeve - □ Uferyde Events - Thread:

Process: [12420] labTeve - □ Uferyde Events - Thread:
                            using System;
                            public delegate void ArrayHandler<T>(object sender, ArrayEventArgs<T> e);
                        S references

| public class ArrayEventArgs<1> : EventArgs
| 1
                                 public int Id { get; set; }
                                  public T Value { get; set; }
                                  public string Message { get; set; }
             9
10
                                  Indexences
public ArrayEventArgs(int id, T value, string message)
                           3 references
Epublic class CustomArray<T>
            19
20
21
22
23
24
25
26
                                 private readonly T[] _array;
private readonly int _startIndex;
                                  public event ArrayHandler<T> OnChangeElement;
public event ArrayHandler<T> OnChangeEqualElement;
                                  | reference
| public Customarray(int length, int startIndex)
             27
28
29
38
31
32
                                  _array = new T[length];
_startIndex = startIndex;
                                 10 references
public T this[int index]
            33
34
35
26
| Registration | Regi
                                                                                                                                                                                                                                                                                                                                                                                                              Sign in ½ − Ø ×
                                                                                                                                          → % Program
             36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
60
61
62
64
                                                if (index < _startIndex || index >= _startIndex + _array.Length)
{
                                                        throw new IndexOutOfRangeException($"Index {index} is out of range.");
                                                }
return _array[index - _startIndex];
                                          if (index < _startIndex || index >= _startIndex + _array.Length)
{
    throw new IndexOutOfRangeException($*Index {index} is out of range.*);
}
                                         Toldvalue = _array[index - _startIndex];
    _array[index - _startIndex] = value;
    if (toldvalue.Equals(value))
    {
        OnChangeElement?.Invoke(this, new ArrayEventArgus()=(index, value, "Element changed"));
    }
                                 {
    OnChangeEqualElement?.Invoke(this, new ArrayEventArgscT>(index, value, "Element set to equal index"));
}
}
                                  public int Length => _array.Length;
            65
66
                                  public int FirstIndex => _startIndex;
                                  public int LastIndex => _startIndex + _array.Length = 1;
            69
70
                                         return _array;
                              0 ▲4 ↑ ↓ | 3/▼
```

Solution:

```
Sign is the five of Project Build Debug Test Assign fools Entension Window Rep P Search Build Search Search Search Build Search Search Search Build Search Bu
```

OUTPUT:

POST-LAB

1. Why and How the Delegates and Events are helpful to develop an Application?

Delegates and events are helpful in application development for several reasons:

- 1. Decoupling:Delegates and events enable loose coupling between components. Publishers can raise events without needing to know or directly reference specific subscribers. This promotes better separation of concerns and easier maintenance of the codebase.
- 2. Flexibility and Extensibility:Delegates and events provide a flexible way to extend functionality. New subscribers can be easily added to handle events without modifying existing code, enabling easy extensibility of applications.
- 3. Asynchronous Programming:Delegates and events can be used to implement asynchronous programming patterns. This allows applications to perform long-running tasks without blocking the main thread, improving responsiveness and user experience.
- 4. Encapsulation and Information Hiding:Events allow classes to encapsulate their internal state and implementation details while still allowing external components to be notified of important changes. This helps maintain a clean and understandable API.
- 5.Support for Observer Pattern: Delegates and events provide a way to implement the Observer pattern, where an object (the subject) maintains a list of dependents (observers) that are notified of state changes. This pattern is useful for implementing UI updates, event-driven systems, and more.

· •	lution	•
JU	uuuui	•