Ex. No.: 4	DELEGATES AND EVENTS
20-08-2024	

Objective

To write C# programs to demonstrate the concept of delegates and events for the given application.

Description

Delegates, Events, and Lambda Expressions are fundamental concepts in C# that enable flexible and dynamic method handling. Delegates are type-safe function pointers that allow methods to be passed as parameters and invoked. Events provide a way to implement the observer pattern, enabling objects to notify each other of changes or actions. Lambda expressions offer a concise syntax for defining anonymous methods, often used for inline delegate implementations and in LINQ queries.

1. Declaring, Instantiating, and Invoking Delegates

Delegates are used to define methods that can be passed as parameters and invoked dynamically.

Syntax for Declaring a Delegate:

```
public delegate returnType DelegateName(parameterList);
```

Syntax for Instantiating a Delegate:

```
DelegateName delegateInstance = (parameters) => expression;
```

Syntax for Invoking a Delegate:

```
result = delegateInstance(arguments);
```

2. Event Declaration, Binding, Triggering, and Handling

Events are declared and associated with delegates to notify when an action occurs. Events are triggered by invoking them.

Syntax for Declaring an Event:

```
public event Action EventName;
```

Syntax for Binding an Event (Subscribing):

```
eventInstance += HandlerMethod;
```

Syntax for Triggering an Event:

```
eventInstance?.Invoke();
```

Syntax for Handling an Event:

```
void HandlerMethod() { /* Event handling code */ }
```

3. Lambda Expression

Lambda expressions are a shorthand for writing anonymous methods and are used to create inline implementations of delegates.

Syntax for Lambda Expression:

```
(parameters) => expression;
```

Syntax for Using Lambda with Delegates:

```
DelegateName delegateInstance = (parameters) => expression;
```

Program
Matrix

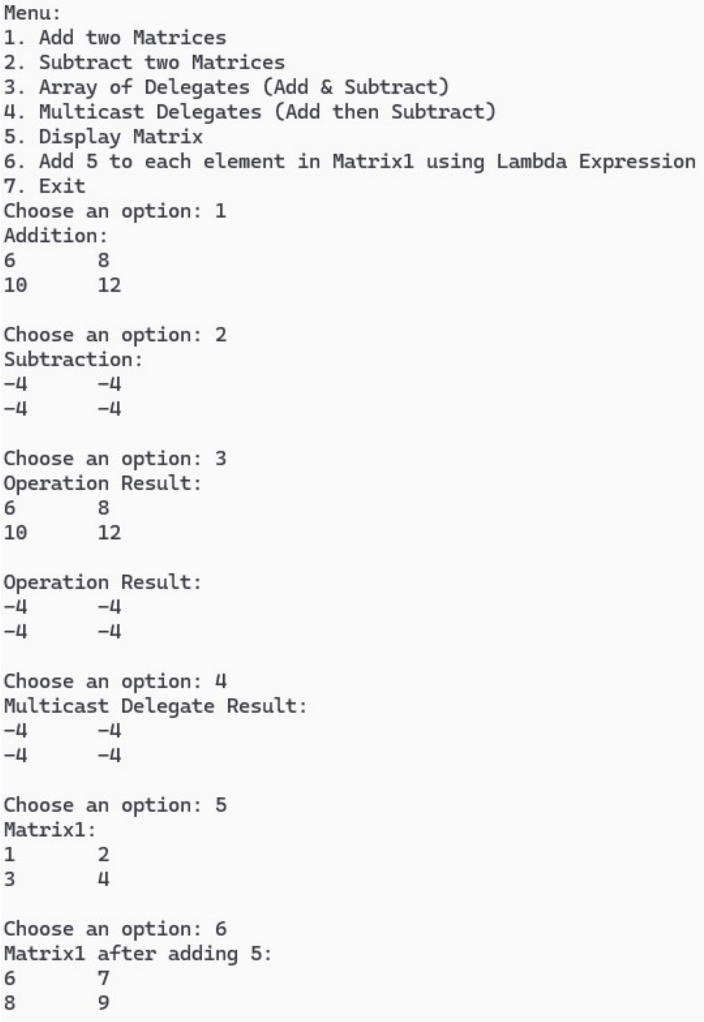
Design and develop a menu driven console application to perform the various operations on Matrix class. Closely follow the instructions given below.

a) Create a Matrix class
   1. Declare integer variables to represent the rows and columns of the matrix
   2. Declare a two-dimensional integer array
   3. Add instance constructor and overloaded constructor which takes rows, columns and
   4. integer two-dimensional array as arguments.
   5. Override the ToString() function to display the Matrix object as a 2D matrix
   6. Add functions to Add and Subtract two Matrix objects.
   7. Add a display function

b) Create a MatrixTest class
   1. Declare required delegates and event
   2. Add Main method here and demonstrate the following as a menu driven program.
      - Add two Matrix using single delegate
      - Subtract two Matrix using single delegate
      - Array of Delegates
      - Multicast Delegates
      - Event handling for display function
      - Lambda Expression to Add 5 to a Matrix object

```
using System;
using System.Linq;
namespace URK21CS1041LAB4 {
  public class Matrix {
    private int rows, columns;
    private int[,] matrixArray;
    public Matrix(int rows, int columns) {
      this.rows = rows; this.columns = columns;
      matrixArray = new int[rows, columns];
    }
    public Matrix(int[,] array) {
      rows = array.GetLength(0); columns = array.GetLength(1);
      matrixArray = array;
    }
    public override string ToString() {
      string result = "";
      for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++)
          result += matrixArray[i, j] + "\t";
        result += "\n";  }
      return result;
    }
    public Matrix Add(Matrix other) {
      int[,] resultArray = new int[rows, columns];
      for (int i = 0; i < rows; i++)
        for (int j = 0; j < columns; j++)
          resultArray[i, j] = matrixArray[i, j] + other.matrixArray[i, j];
      return new Matrix(resultArray);
    }
    public Matrix Subtract(Matrix other) {
         int[,] resultArray = new int[rows, columns];
         for (int i = 0; i < rows; i++)
            for (int j = 0; j < columns; j++)
               resultArray[i, j] = matrixArray[i, j] - other.matrixArray[i, j];
         return new Matrix(resultArray);
      }
      public event Action DisplayEvent;
      public void Display() {
         DisplayEvent?.Invoke();
         Console.WriteLine(this.ToString());
      }
      public void AddFive() {
         matrixArray = matrixArray.Cast<int>().Select(x => x + 5).ToArray().Reshape(rows, columns);
      }}
   public static class ArrayExtensions {
      public static T[,] Reshape<T>(this T[] array, int rows, int cols) {
         T[,] result = new T[rows, cols];
         for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
               result[i, j] = array[i * cols + j];
         return result;
      }}
   public class MatrixTest {
      public delegate Matrix MatrixOperation(Matrix a, Matrix b);
      static void Main(string[] args) {
         Matrix matrix1 = new Matrix(new int[,] { { 1, 2 }, { 3, 4 } });
         Matrix matrix2 = new Matrix(new int[,] { { 5, 6 }, { 7, 8 } });
         MatrixOperation addOperation = (a, b) => a.Add(b);
         MatrixOperation subtractOperation = (a, b) => a.Subtract(b);
         Console.WriteLine("\nMenu:");
         Console.WriteLine("1. Add two Matrices");
         Console.WriteLine("2. Subtract two Matrices");
         Console.WriteLine("3. Array of Delegates (Add & Subtract)");
         Console.WriteLine("4. Multicast Delegates (Add then Subtract)");
         Console.WriteLine("5. Display Matrix");
         Console.WriteLine("6. Add 5 to each element in Matrix1 using Lambda Expression");
         Console.WriteLine("7. Exit");
         while (true) {
            Console.Write("Choose an option: ");
            string choice = Console.ReadLine();
            switch (choice) {
               case "1":
                  Console.WriteLine("Addition:\n" + addOperation(matrix1, matrix2));
                  break;
               case "2":
                  Console.WriteLine("Subtraction:\n" + subtractOperation(matrix1, matrix2));
                  break;
               case "3":
                  MatrixOperation[] operations = { addOperation, subtractOperation };
                  foreach (var operation in operations)
                     Console.WriteLine("Operation Result:\n" + operation(matrix1, matrix2));
                  break;
```

```
  case "4":
    MatrixOperation multicastOperation = addOperation + subtractOperation;
    Console.WriteLine("Multicast Delegate Result:\n" + multicastOperation(matrix1, matrix2));
    break;
  case "5":
    matrix1.DisplayEvent += () => Console.WriteLine("Matrix1: ");
    matrix1.Display();
    break;
  case "6":
    matrix1.AddFive();
    Console.WriteLine("Matrix1 after adding 5:\n" + matrix1);
    break;
  case "7":
    return;
  default:
    Console.WriteLine("Invalid choice.");
    break;
}}}}}
```



```
Menu:
1. Add two Matrices
2. Subtract two Matrices
3. Array of Delegates (Add & Subtract)
4. Multicast Delegates (Add then Subtract)
5. Display Matrix
6. Add 5 to each element in Matrix1 using Lambda Expression
7. Exit
Choose an option: 1
Addition:
6       8
10      12

Choose an option: 2
Subtraction:
-4      -4
-4      -4

Choose an option: 3
Operation Result:
6       8
10      12

Operation Result:
-4      -4
-4      -4

Choose an option: 4
Multicast Delegate Result:
-4      -4
-4      -4

Choose an option: 5
Matrix1:
1       2
3       4

Choose an option: 6
Matrix1 after adding 5:
6       7
8       9
```

Result

Thus, the C# program to demonstrate the concept of delegates and events was implemented and verified successfully.