**Exercise 5**

**Delegates and Lambda Expressions**

**Objective**

To develop C# Console applications using Lambda Expressions and Delegates.

**Delegate**

A delegate is a pointer to a method. That means, a delegate holds the address of a method which can be called using that delegate.

**Define a delegate**

We define a delegate just like we define a normal method. That is, delegate also has a return type and parameter. For example,

public delegate void MyDelegate(int a, int b);

Any method from any class that matches the delegate signature (return type and parameter) can be assigned to the delegate.

**How to store the address of a method in delegate?**

Suppose we have a sum method whose signature is the same as MyDelegate as given below

public static void sum(int a, int b)

{

Console.WriteLine(a + b);

}

Create an instance of MyDelegate and pass a method name as a parameter. For example,

MyDelegate d1 = new MyDelegate(sum);

//d1 is a pointer to the method sum

Hence, we can call the method using d1 delegate reference

d1(5, 6);

**Example Program**

using System;

namespace DemoApplication

{

class Program

{

public delegate void MyDelegate(int a, int b);

public static void sum(int a, int b)

{

Console.WriteLine(a + b);

}

static void Main(string[] args) {

MyDelegate d1 = new MyDelegate(sum);

d1(5, 6);

Console.ReadKey();

}

}

}

**Multicast Delegate in C#**

The multicast delegate is used to point to more than one method at a time. We use **+=** operator to add methods to delegate. For example,

using System;

namespace DemoApplication

{

class Calculator

{

public delegate void MyDelegate(int a, int b);

public void sum(int a, int b)

{

Console.WriteLine("Sum:" + (a + b));

}

public void multiply(int a, int b)

{

Console.WriteLine("Product: " + (a \* b));

}

public void RectArea(int l,int b)

{

Console.WriteLine("Area:" + (l\*b));

}

static void Main(string[] args) {

Calculator obj = new Calculator();

MyDelegate d1 = new MyDelegate(obj.sum);

d1 += obj.multiply; //multicast

d1 += obj.RectArea;

d1(5, 6);

Console.ReadKey();

}

}

}

**Lambda Expressions**

C# Lambda Expression is a short block of code that accepts parameters and returns a value. It is defined as an anonymous function (function without a name).

**General Syntax**

(parameterList) => lambda body

* parameterList - list of input parameters
* => - a lambda operator
* lambda body - can be an expression or statement.

The two types of lambda expressions are

* Expression Lambda
* Statement Lambda

**C# Lambda Expression**

C# Lambda Expression is a short block of code that accepts parameters and returns a value. It is defined as an anonymous function (function without a name). For example

num => num \* 7

Here, num is an input parameter and num \* 7 is a return value. The lambda expression does not execute on its own. Instead, it should be used inside other methods or variables.

Based on lambda body, the C# lambda expression is divided into two types.

Types of Lambda Expression

The two types of lambda expressions are:

1. Expression Lambda

2. Statement Lambda

**1. Expression Lambda**: Expression lambda contains a single expression in the lambda body. For example

(int num) => num \* 5;

The above expression lambda contains a single expression num \* 5 in the lambda body. It takes an int input, multiplies it by 5, and returns the output.

**Example code:**

using System;

namespace DemoApplication

{

class Program

{

public delegate int MyDelegate(int x);

static void Main(string[] args) {

MyDelegate square = (int x) => x \* x;

Console.WriteLine(square(5));

Console.ReadKey();

}

}

}

**2. Statement Lambda:** Statement lambda encloses one or more statements in the lambda body. We use curly braces {} to wrap the statements. For example,

(int a, int b) =>

{

var sum = a + b;

return sum;

};

The above expression is a statement lambda which contains two statements in the lambda body. This takes two int inputs and returns its sum.

Example code:

using System;

namespace DemoApplication

{

class Program

{

public delegate int MyDelegate(int x);

static void Main(string[] args) {

MyDelegate factorial = (int num) =>

{

int fact = 1;

for(int i = 2; i <= num; i++)

{

fact = fact \* i;

}

return fact;

};

Console.WriteLine("Factorial: " + factorial(5));

Console.ReadKey();

}

}

}

**Questions**

**(Select one question in each section based on Regno%3+1)**

**Section 1:** Create a C# console application to create lambda expression (expression lambda) to perform the following any one operation as per your Reg.No.

* 1. Area of circle
  2. Volume of cube
  3. Perimeter of cone

**Section 2:** Create a C# console application to create lambda expression (statement lambda) for the following (anyone as per your Reg.No)

1. Check the given number is Armstrong or not
2. Check the given number is palindrome or not
3. Check the given number is prime or not.

**Section 3:** Create a C# console applications for the following (any one) using Delegates

1 Create a complex number class and perform mathematical operations such as addition, subtraction, multiplication and division using individual methods.

a. Create a delegate ComplexDelegate

b. Assign all methods using multicast delegate

c. Call all methods through the single delegate object.

2 Create a Matrix class with two dimensional array data for storing 2x2 matrix using Constructor and perform mathematical operations such as addition, subtraction, multiplication and division using individual methods.

a. Create a delegate say MatrixDelegate

b. Assign all methods using multicast delegate

c. Call all the methods through the single delegate object.

3 Create a Distance class containing two data such as feet, inches. Perform addition, subtraction, multiplication and division using individual methods.

a. Create a delegate say DistanceDelegate

b. Assign all methods using multicasting delegate.

c. Call all methods through the single delegate object.