C Sharp Anonymous Methods and Lamda Expression

concept of lamda expression was introduced in C# 3.0. Basically, Lamda expression is a more concise syntax of anonymous method. It is just a new way to write anonymous methods. At compile time all the lamda expressions are converted into anonymous methods according to lamda expression conversion rules.The left side of the lamda operator "=>" represents the arguments to the method and the right side is the method body.

### Lambda expression Syntax

1. **(parameters) => expression-or-statement-block**

## Types of Lamda expression

## Statement Lamda

Statement lambda has a statement block on the right side of the lambda operator "=>".

* 1. **x => { return x \* x; };**

## Expression Lamda

Expression lambda has only an expression (no return statement or curly braces), on the right side of the lambda operator "=>".

* 1. **x => x \* x; *// here x\*x is expression***

## Features of lamda expression

1. Lambda expressions themselves do not have type. In fact, there is no concept of a lambda expression in the CLR.
   1. ***// ERROR: Operator '.' cannot be applied to***
   2. ***// operand of type 'lambda expression'***
   3. **Type type = ((int x) => x).ToString();**
2. A lambda expression cannot be assigned to an implicitly typed local variable since the lambda expressions do not have type.
   1. ***// ERROR: Cannot assign lambda expression to an***
   2. ***// implicitly typed local variable***
   3. **var thing = (x => x);**
3. Jump statements (break, goto, continue) are not allowed within anonymous method/lambda expression. Similarly, you cannot jump into the lambda expression/ anonymous method from outside.
4. Variables defined within a lambda expression are accessible only within the scope of the lambda expression body.
5. Lambda expressions are used generally with the Func and Action delegates. our earlier expression can be written as follows:
   1. **Func sqr = x => x \* x;**

## Anonymous method with Lambda and delegate

1. **class Program**
2. **{**
3. ***//delegate for representing anonymous method***
4. **delegate int del(int x, int y);**
6. **static void Main(string[] args)**
7. **{**
8. ***//anonymous method using expression lamda***
9. **del d1 = (x, y) => x \* y;**
10. ***// or (int x, int y) => x \* y;***
12. ***//anonymous method using statement lamda***
13. **del d2 = (x, y) => { return x \* y; };**
14. ***// or (int x, int y) => { return x \* y; };***
16. ***//anonymous method using delegate keyword***
17. **del d3 = delegate(int x, int y) { return x \* y; };**
19. **int z1 = d1(2, 3);**
20. **int z2 = d2(3, 3);**
21. **int z3 = d3(4, 3);**
23. **Console.WriteLine(z1);**
24. **Console.WriteLine(z2);**
25. **Console.WriteLine(z3);**
26. **}**
27. **}**
28. ***//output:***
29. **6**
30. **9**
31. **12**

**Example:** *Print all numbers less than 5 from a List*

List<int> arr = new List<int> { 1, 2, 3, 7, 2, 9, 4, 7, };

List<int> min = arr.FindAll(x => x < 5);

foreach(int i in min)

Console.WriteLine(i);

**What is an anonymous method?**  
In simple terms, anonymous method is a method without a name.   
  
  
  
**Let's understand how a method can exist without a name**

usingSystem**;**

usingSystem.Collections.Generic**;**

classProgram

**{**

publicstaticvoidMain**()**

**{**

List<Employee>listEmployees=newList<Employee>**()**

**{**

newEmployee**{**ID=101**,**Name="Mark"**},**

newEmployee**{**ID=102**,**Name="John"**},**

newEmployee**{**ID=103**,**Name="Mary"**},**

**};**

// Step 2: Create an instance of Predicate<Employee>

// delegate and pass the method name as an argument

// to the delegate constructor

Predicate<Employee>predicateEmployee=

newPredicate<Employee>**(**FindEmployee**);**

// Step 3: Now pass the delegate instance as

// the argument for Find() method

Employeeemployee=

listEmployees.Find**(**x=>predicateEmployee**(**x**));**

Console.WriteLine**(**"ID = {0}, Name {1}"**,**

employee.ID**,**employee.Name**);**

// Anonymous method is being passed as an argument to

// the Find() method. This anonymous method replaces

// the need for Step 1, 2 and 3

employee=listEmployees.Find**(**delegate**(**Employeex**)**

**{**returnx.ID==102**; });**

Console.WriteLine**(**"ID = {0}, Name {1}"**,**

employee.ID**,**employee.Name**);**

**}**

// Step 1: Create a method whose signature matches

// with the signature of Predicate<Employee> delegate

privatestaticboolFindEmployee**(**EmployeeEmp**)**

**{**

returnEmp.ID==102**;**

**}**

publicclassEmployee

**{**

publicintID**{**get**;**set**; }**

publicstringName**{**get**;**set**; }**

**}**

**}**

In this example, **Find**() method expects a delegate to be passed as the argument. If you want to look at the signature of the delegate, right click on **Find()**method and select **"Go To Definition"**from the context menu. At this point you should see the following method.  
publicTFind**(**Predicate<T>match**);**  
  
Right click on Predicate<T> and select  **"Go To Definition"**  
  
Now you should see the signature of the Predicate delegate.  
publicdelegateboolPredicate<in**T**>**(T**obj**);**  
  
Notice that the delegate returns bool and expects an object of Type <T>. In our case T isEmployee.   
  
So, to the **Find**() method we need to pass an instance of Predicate<Employee> delegate as an argument. Delegates are function pointers. This means when we create an instance of a delegate, we pass the name of the method as an argument to the delegate constructor.   
  
**Step 1:**Create a method whose signature matches with the signature of Predicate<Employee> delegate

privatestaticboolFindEmployee**(**EmployeeEmp**)**

**{**

returnEmp.ID==102**;**

**}**

**Step 2:** Create an instance of Predicate<Employee> delegate and pass the method name as an argument to the delegate constructor

Predicate<Employee>predicateEmployee=

newPredicate<Employee>**(**FindEmployee**);**

**Step 3:** Now pass the delegate instance as the argument for Find() method

Employeeemployee=

listEmployees.Find**(**x=>predicateEmployee**(**x**));**

**Anonymous methods were introduced in C# 2** and they eliminate the need for Step 1, 2 & 3, that is they provide us a way of creating delegate instances without having to write a separate method.  
  
**Now let us see, how to pass anonymous method as an argument to Find() method.**  
employee=listEmployees.Find**(**delegate**(**Employeex**) {**returnx.ID==102**; });**

**Anonymous methods and Lambda expressions are very similar.** Anonymous methods were introduced in C# 2 and Lambda expressions in C# 3.   
  
**To find an employee with Id = 102, using anonymous method**

Employeeemployee=listEmployees.Find

**(**delegate**(**EmployeeEmp**) {**returnEmp.ID==102**; });**

**To find an employee with Id = 102, using lambda expression**  
Employeeemployee=listEmployees.Find**(**Emp=>Emp.ID==102**);**  
  
**You can also explicitly specify the Input type but not required**  
employee=listEmployees.Find**((**EmployeeEmp**)**=>Emp.ID==102**);**  
  
**Notice that with a Lambda expression you don't have to use the delegate keyword explicitly and you don't have to specify the input parameter type explicitly.** The parameter type is inferred. Lambda expressions are more convenient to use than anonymous methods. Lambda expressions are particularly helpful for writing LINQ query expressions.  
  
**=>** is called lambda operator and read as GOES TO.  
  
**In most of the cases Lambda expressions supersedes anonymous methods.** To my knowledge, the only time I prefer to use anonymous methods over lambdas is, when we have to omit the parameter list when it's not used within the body.   
  
Anonymous methods allow the parameter list to be omitted entirely when it's not used within the body, where as with lambda expressions this is not the case.