**Delegates Real-time example in C#**

In this article, I am going to discuss the **delegates real-time example in C#**. The delegates are one of the most important concepts that you need to understand as a C# developer. In many interviews, most of the interviewers ask to explain the usage of delegates in the real-time project that you have worked on.  Please read the following two articles before proceeding to this article, where we discussed the basics of delegate and multicast delegates in C# with examples.

1. [**SingleCast Delegates in C#**](https://dotnettutorials.net/lesson/delegates-csharp/) – Here we discussed the basics of Delegates in C# with some examples.
2. [**Multicast Delegate in C#**](https://dotnettutorials.net/lesson/multicast-delegate-csharp/)– Here we discussed the Multicast Delegates in C# with some examples.

The Delegates in C# are extensively used by framework developers.  Let us understand delegates in C# with one real-time example. Let say we have a class called **Employee** as shown below.

**Employee.cs**

**namespace** DelegateRealtimeExample

**{**

**public** class Employee

**{**

**public** int ID **{** get; set; **}**

**public** string Name **{** get; set; **}**

**public** string Gender **{** get; set; **}**

**public** int Experience **{** get; set; **}**

**public** int Salary **{** get; set; **}**

**}**

**}**

The above **Employee** class has the following properties.

**Id**

**Name**

**Gender**

**Experience**

**Salary**

Now I want to write a method in the **Employee** class which can be used to promote the employees. The method that we are going to write will take a list of Employee objects as a parameter and then should print the names of all the employees who are eligible for a promotion.

But the logic based on which the employee gets promoted should not be hardcoded. At times we may promote employees **based on their experience** and at times we may promote them **based on their salary** or maybe **some other condition**. So, the logic to promote employees **should not be hard-coded within the method**.

**To achieve this we can make use of delegates. So now I would design my class as shown below.**

**namespace** DelegateRealtimeExample

**{**

**public** delegate bool EligibleToPromotion**(**Employee EmployeeToPromotion**)**;

**public** class Employee

**{**

**public** int ID **{** get; set; **}**

**public** string Name **{** get; set; **}**

**public** string Gender **{** get; set; **}**

**public** int Experience **{** get; set; **}**

**public** int Salary **{** get; set; **}**

**public** static **void** PromoteEmployee**(**List<Employee> lstEmployees, EligibleToPromotion IsEmployeeEligible**)**

**{**

**foreach** **(**Employee employee in lstEmployees**)**

**{**

**if** **(**IsEmployeeEligible**(**employee**))**

**{**

Console.WriteLine**(**"Employee {0} Promoted", employee.Name**)**;

**}**

**}**

**}**

**}**

**}**

In the above example, we created a delegate called**EligibleToPromote**. This delegate takes the **Employee** object as a parameter and returns a boolean. In the **Employee** class, we have **a PromoteEmpoloyee** method. This method takes a list of Employees and a Delegate of type **EligibleToPromote** as parameters.

The method then loops thru each employee object and passes it to the delegate. If the delegate returns true, then the Employee is **promoted**, else **not promoted**. So within the method, **we have not hardcoded any logic on how we want to promote employees**.

Now the client who uses the **Employee** class has the flexibility of determining the logic on **how they want to promote their employees**. First create the employee objects – emp1, emp2, and emp3. Populate the properties for the respective objects. We then create an employee List to hold all the 3 employees as shown below.

**namespace** DelegateRealtimeExample

**{**

**public** class Program

**{**

static **void** Main**()**

**{**

Employee emp1 = **new** Employee**()**

**{**

ID = 101,

Name = "Pranaya",

Gender = "Male",

Experience = 5,

Salary = 10000

**}**;

Employee emp2 = **new** Employee**()**

**{**

ID = 102,

Name = "Priyanka",

Gender = "Female",

Experience = 10,

Salary = 20000

**}**;

Employee emp3 = **new** Employee**()**

**{**

ID = 103,

Name = "Anurag",

Experience = 15,

Salary = 30000

**}**;

List<Employee> lstEmployess = **new** List<Employee>**()**;

lstEmployess.Add**(**emp1**)**;

lstEmployess.Add**(**emp2**)**;

lstEmployess.Add**(**emp3**)**;

EligibleToPromotion eligibleTopromote = **new** EligibleToPromotion**(**Program.Promote**)**;

Employee.PromoteEmployee**(**lstEmployess, eligibleTopromote**)**;

Console.ReadKey**()**;

**}**

**public** static bool Promote**(**Employee employee**)**

**{**

**if** **(**employee.Salary > 10000**)**

**{**

**return** **true**;

**}**

**else**

**{**

**return** **false**;

**}**

**}**

**}**

* **}**

Notice the Promote method that we have created. This method has the logic of how we want to promote our employees. **The method is then passed as a parameter to the delegate**. Also, note this method has the same signature as that of **EligibleToPromote** delegate. This is very important because the **Promote method cannot be passed as a parameter to the delegate if the signature differs**. This is the reason why delegates are called as **type-safe function pointers**.

**OUTPUT:**

Delegates Real-time example in C#

So if we did not have the concept of delegates it would not have been possible to pass a function as a parameter. As Promote method in the Employee class makes use of delegate, it is possible to dynamically decide the logic on how we want to promote employees.

In C# 3.0 **Lambda expressions**are introduced. So you can make use of lambda expressions instead of creating a function and then an instance of a delegate and then passing the function as a parameter to the delegate. The sample example rewritten using Lambda expression is shown below. Private Promote method is no longer required now.

* **namespace** DelegateRealtimeExample
* **{**
* **public** class Program
* **{**
* static **void** Main**()**
* **{**
* Employee emp1 = **new** Employee**()**
* **{**
* ID = 101,
* Name = "Pranaya",
* Gender = "Male",
* Experience = 5,
* Salary = 10000
* **}**;
* Employee emp2 = **new** Employee**()**
* **{**
* ID = 102,
* Name = "Priyanka",
* Gender = "Female",
* Experience = 10,
* Salary = 20000
* **}**;
* Employee emp3 = **new** Employee**()**
* **{**
* ID = 103,
* Name = "Anurag",
* Experience = 15,
* Salary = 30000
* **}**;
* List<Employee> lstEmployess = **new** List<Employee>**()**;
* lstEmployess.Add**(**emp1**)**;
* lstEmployess.Add**(**emp2**)**;
* lstEmployess.Add**(**emp3**)**;
* Employee.PromoteEmployee**(**lstEmployess, x => x.Experience > 5**)**;
* Console.ReadKey**()**;
* **}**
* **}**
* **}**

**OUTPUT:**

Delegates Real-time example in C#