Callback method using Delegate

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. The Employee class has the properties like Id, Name, Gender, Experience and Salary.

Now we 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.

Example:

```
using System;
using System.Collections;
namespace DelegateExample
{
    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);
                }
            }
        }
    }
    public class Program
    {
        static void Main()
        {
            Employee emp1 = new Employee()
            {
                ID = 101,
                Name = "Rahul",
```

```
                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 = "Kishan",
                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;
            }
        }
    }
}
// Output:
//    Employee Priyanka Promoted
//    Employee Kishan Promoted
```

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 in example.

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.