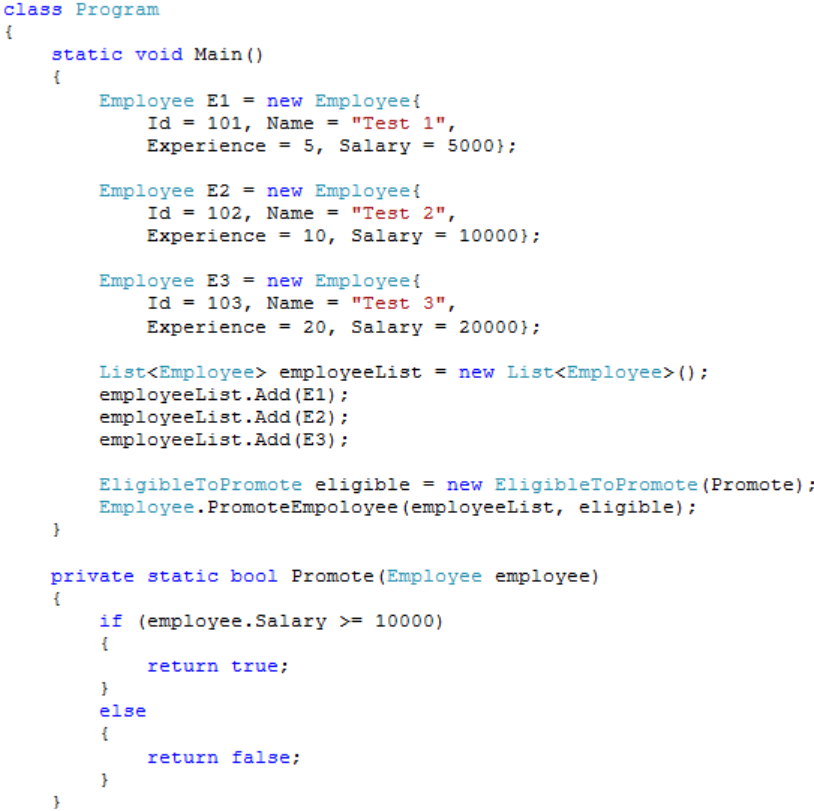
**Where did use delegates in your project - Part 2**

So now, the client who uses the **Employee** class has the flexibility of determining the logic on **how they want to promote their employees** a shown below. First create the employee objects - E1, E2 and E3. Populate the properties for the respective objects. We then create an employeeList to hold all the 3 employees.

Notice, the **private Promote method** that we have created. This method has the logic on 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 **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.



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 Sharp 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 expressions is shown below. Private Promote method is no longer required now.

Graphical user interface, text, application

Description automatically generated

**The output of the above program should be as below.**  
Employee Test 2 Promoted  
Employee Test 3 Promoted