

# SOFTWARE ENGINEERING

## HOMEWORK 4

الطالبة : ريان ناطورة

الفئة :الخامسة

خطوات انشاء المشروع :

انشاء مجلد جديد ونكتب التعليمات في terminal:

```
dotnet new sln -n LoanApp
```

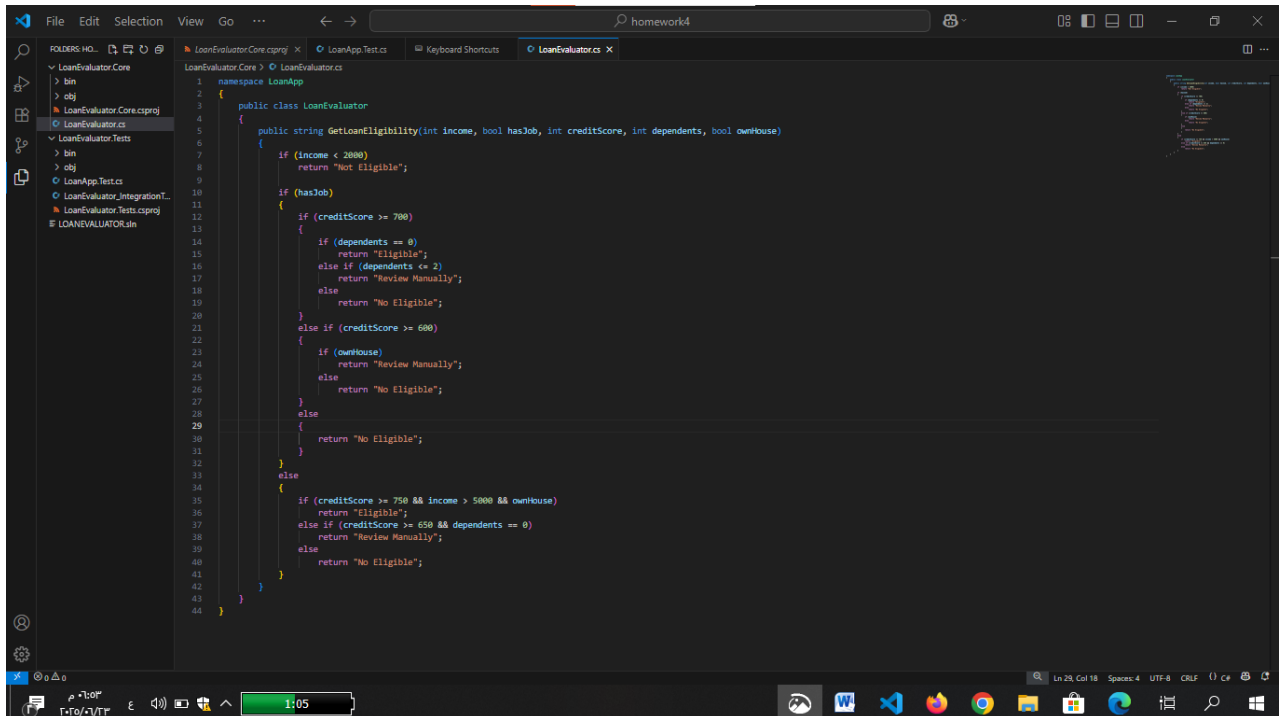
```
dotnet new classlib -n LoanApp.Core
```

```
dotnet new xunit -n LoanApp.Tests
```

```
dotnet sln add LoanApp.Core/LoanApp.Core.csproj
```

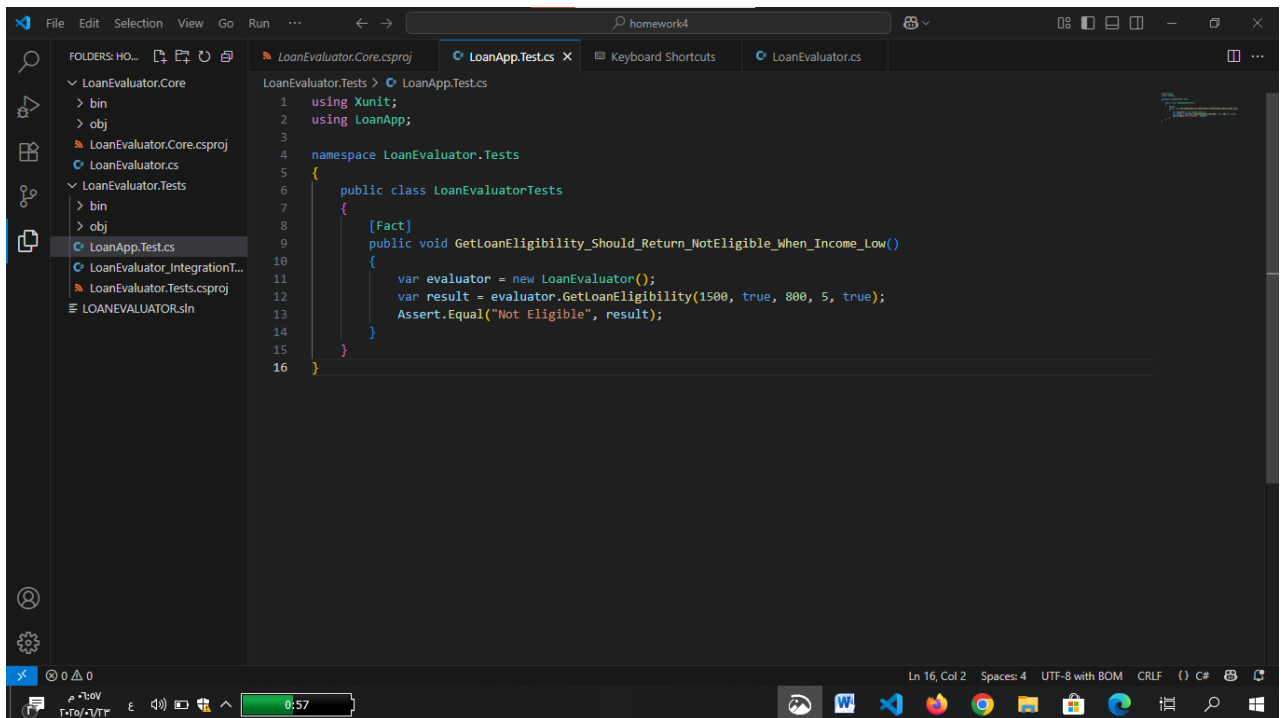
```
dotnet add LoanApp.Tests/LoanApp.Tests.csproj reference LoanApp.Core.csproj
```

١. محتوى ملف LoanEvaluator.cs (كود المحاضرة):



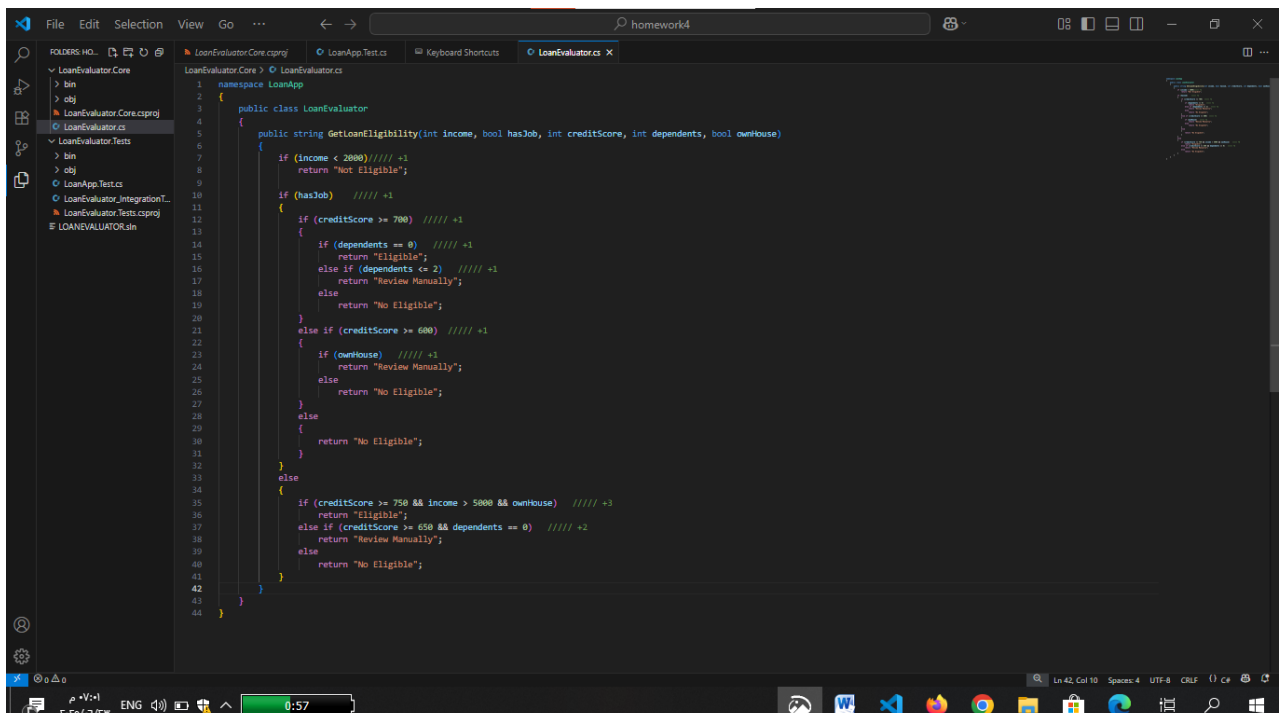
```
1 namespace LoanApp
2 {
3     public class LoanEvaluator
4     {
5         public string GetLoanEligibility(int income, bool hasJob, int creditScore, int dependents, bool ownHouse)
6         {
7             if (income < 2000)
8                 return "Not Eligible";
9
10            if (hasJob)
11            {
12                if (creditScore >= 700)
13                {
14                    if (dependents == 0)
15                        return "Eligible";
16                    else if (dependents <= 2)
17                        return "Review Manually";
18                    else
19                        return "No Eligible";
20                }
21                else if (creditScore >= 600)
22                {
23                    if (ownHouse)
24                        return "Review Manually";
25                    else
26                        return "No Eligible";
27                }
28                else
29                {
30                    return "No Eligible";
31                }
32            }
33            else
34            {
35                if (creditScore >= 750 && income > 5000 && ownHouse)
36                    return "Eligible";
37                else if (creditScore >= 650 && dependents == 0)
38                    return "Review Manually";
39                else
40                    return "No Eligible";
41            }
42        }
43    }
44 }
```

## ٢. محتوى ملف LoanApp.Test.cs



```
1 using Xunit;
2 using LoanApp;
3
4 namespace LoanEvaluator.Tests
5 {
6     public class LoanEvaluatorTests
7     {
8         [Fact]
9         public void GetLoanEligibility_Should_Return_NotEligible_When_Income_Low()
10        {
11            var evaluator = new LoanEvaluator();
12            var result = evaluator.GetLoanEligibility(1500, true, 800, 5, true);
13            Assert.Equal("Not Eligible", result);
14        }
15    }
16 }
```

## ٣. حساب التعقيد يدويا قبل التحسين Cyclomatic Complexity:



```
1 namespace LoanApp
2 {
3     public class LoanEvaluator
4     {
5         public string GetLoanEligibility(int income, bool hasJob, int creditScore, int dependents, bool ownHouse)
6         {
7             if (income < 2000) // +1
8                 return "Not Eligible";
9             if (hasJob) // +1
10            {
11                if (creditScore >= 700) // +1
12                {
13                    if (dependents == 0) // +1
14                        return "Eligible";
15                    else if (dependents <= 2) // +1
16                        return "Review Manually";
17                    else
18                        return "No Eligible";
19                }
20                else if (creditScore >= 600) // +1
21                {
22                    if (ownHouse) // +1
23                        return "Review Manually";
24                    else
25                        return "No Eligible";
26                }
27                else
28                {
29                    return "No Eligible";
30                }
31            }
32            else
33            {
34                if (creditScore >= 750 && income > 5000 && ownHouse) // +3
35                    return "Eligible";
36                else if (creditScore >= 650 && dependents == 0) // +2
37                    return "Review Manually";
38                else
39                    return "No Eligible";
40            }
41        }
42    }
43 }
44 }
```

المجموع = ١٢ نقاط القرار + (ثابت)

Cyclomatic Complexity = 13

## ٤. حساب التعقيد باستخدام أداة Codalyze:

The screenshot shows the Visual Studio IDE with a 'Code Complexity Report' window open. The report is for the method 'LoanApp::LoanEvaluator::GetLoanEligibility'. It displays the following data:

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
LoanApp::LoanEvaluator::GetLoanEligibility	5	42	▲ 13	37	▲ 5

Source Code: 52eba966-504c-11f0-8edb-263d64ae3d6b.cs

Generated by Codalyze on 2025-06-23 16:08

## ٥. الكود بعد التحسين Refactoring:

The screenshot shows the Visual Studio IDE with the 'LoanApp::LoanEvaluator.cs' file open. The code is as follows:

```
1 namespace LoanApp
2 {
3     public class LoanEvaluator
4     {
5         public string GetLoanEligibility(int income, bool hasJob, int creditScore, int dependents, bool ownHouse)
6         {
7             if (income < 2000)
8                 return "Not Eligible";
9
10            return hasJob
11                ? EvaluateEmployed(creditScore, dependents, ownHouse)
12                : EvaluateUnemployed(income, creditScore, dependents, ownHouse);
13        }
14
15        private string EvaluateEmployed(int creditScore, int dependents, bool ownHouse)
16        {
17            if (creditScore >= 700)
18            {
19                if (dependents == 0)
20                    return "Eligible";
21                else if (dependents <= 2)
22                    return "Review Manually";
23                else
24                    return "No Eligible";
25            }
26            else if (creditScore >= 600)
27            {
28                return ownHouse ? "Review Manually" : "No Eligible";
29            }
30            else
31            {
32                return "No Eligible";
33            }
34        }
35
36        private string EvaluateUnemployed(int income, int creditScore, int dependents, bool ownHouse)
37        {
38            if (creditScore >= 750 && income > 5000 && ownHouse)
39                return "Eligible";
40            else if (creditScore >= 650 && dependents == 0)
41                return "Review Manually";
42            else
43                return "No Eligible";
44        }
45    }
46 }
```

## ٦. حساب التعقيد يدويا بعد التحسين :Cyclomatic Complexity

```
1 namespace LoanApp
2 {
3     public class LoanEvaluator
4     {
5         public string GetLoanEligibility(int income, bool hasJob, int creditScore, int dependents, bool ownHouse)
6         {
7             if (income < 2000) //+1
8                 return "Not Eligible";
9
10            return hasJob //+1
11                ? EvaluateEmployed(creditScore, dependents, ownHouse)
12                : EvaluateUnemployed(income, creditScore, dependents, ownHouse);
13        }
14
15        private string EvaluateEmployed(int creditScore, int dependents, bool ownHouse)
16        {
17            if (creditScore >= 700) //+1
18            {
19                if (dependents == 0) //+1
20                    return "Eligible";
21                else if (dependents < 2) //+1
22                    return "Review Manually";
23                else
24                    return "No Eligible";
25            }
26            else if (creditScore >= 600) //+1
27            {
28                return ownHouse ? "Review Manually" : "No Eligible"; //+1
29            }
30            else
31            {
32                return "No Eligible";
33            }
34        }
35
36        private string EvaluateUnemployed(int income, int creditScore, int dependents, bool ownHouse)
37        {
38            if (creditScore >= 750 && income > 5000 && ownHouse) //+1
39                return "Eligible";
40            else if (creditScore >= 650 && dependents == 0) //+1
41                return "Review Manually";
42            else
43                return "No Eligible";
44        }
45    }
46 }
```

المجموع = 9 نقاط القرار + (ثابت)

Cyclomatic Complexity = 10

## ٧. حساب التعقيد باستخدام أداة Codalyze بعد التحسين:

### Code Complexity Report

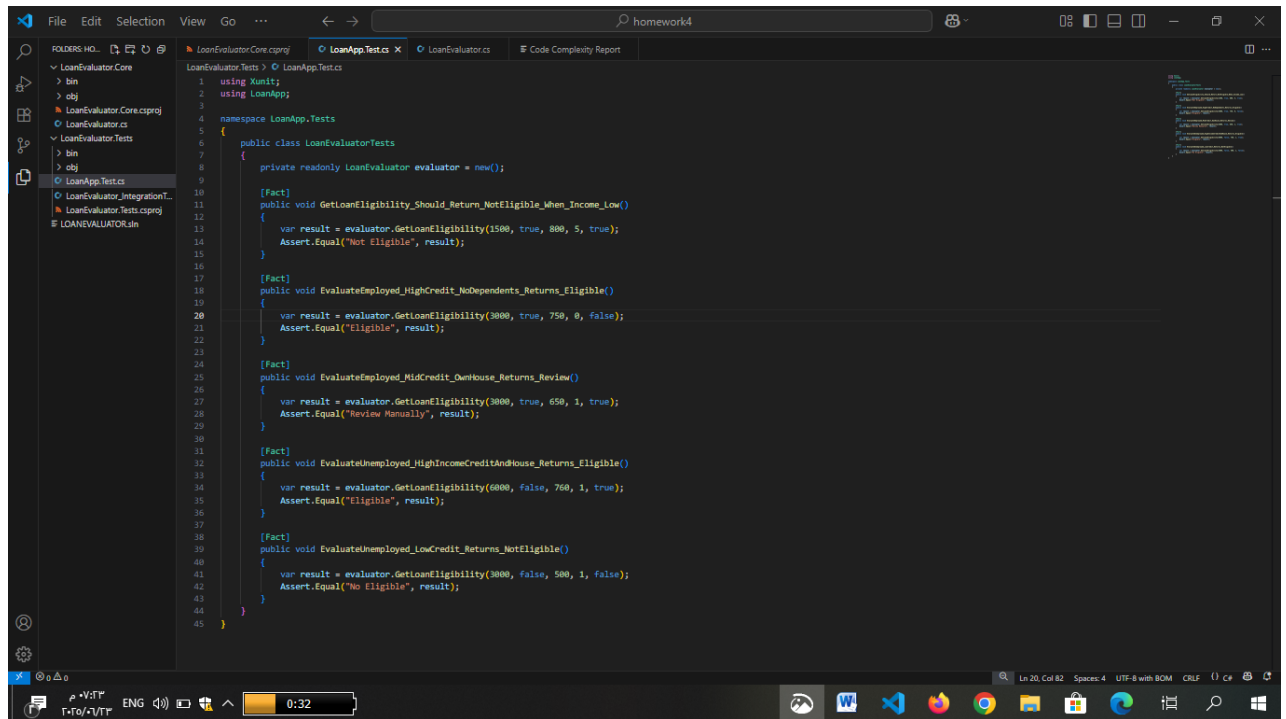
— Clean Code: A Handbook of Agile Software Craftsmanship Robert C. Martin

Function Name	Start Line	End Line	Cyclomatic Complexity (Threshold: 10)	Lines of Code (Threshold: 50)	Parameter Count (Threshold: 4)
LoanApp.LoanEvaluator.GetLoanEligibility	5	13	3	8	5
LoanApp.LoanEvaluator.EvaluateEmployed	15	34	6	20	3
LoanApp.LoanEvaluator.EvaluateUnemployed	36	44	6	9	4

Source Code: cf3a110a-504d-11f0-8edb-263d64ae3d6b.cs

Generated by Codalyze on 2025-06-23 16:19

## ٨. كود الاختبار لتغطية جميع الحالات:



```
1 using Xunit;
2 using LoanApp;
3
4 namespace LoanApp.Tests
5 {
6     public class LoanEvaluatorTests
7     {
8         private readonly LoanEvaluator evaluator = new();
9
10        [Fact]
11        public void GetLoanEligibility_Should_Return_NotEligible_When_Income_Low()
12        {
13            var result = evaluator.GetLoanEligibility(1500, true, 800, 5, true);
14            Assert.Equal("Not Eligible", result);
15        }
16
17        [Fact]
18        public void EvaluateEmployed_HighCredit_NoDependents_Returns_Eligible()
19        {
20            var result = evaluator.GetLoanEligibility(3000, true, 750, 0, false);
21            Assert.Equal("Eligible", result);
22        }
23
24        [Fact]
25        public void EvaluateEmployed_MidCredit_OwnHouse_Returns_Review()
26        {
27            var result = evaluator.GetLoanEligibility(3000, true, 650, 1, true);
28            Assert.Equal("Review Manually", result);
29        }
30
31        [Fact]
32        public void EvaluateUnemployed_HighIncomeCreditAndHouse_Returns_Eligible()
33        {
34            var result = evaluator.GetLoanEligibility(6000, false, 760, 1, true);
35            Assert.Equal("Eligible", result);
36        }
37
38        [Fact]
39        public void EvaluateUnemployed_LowCredit_Returns_NotEligible()
40        {
41            var result = evaluator.GetLoanEligibility(3000, false, 500, 1, false);
42            Assert.Equal("No Eligible", result);
43        }
44    }
45 }
```

شرح الحالات :

١. GetLoanEligibility\_Should\_Return\_NotEligible\_When\_Income\_Low

تتحقق هذه الحالة من أن أي شخص دخله أقل من ٢٠٠٠ يُرفض طلبه مباشرة، بغض النظر عن الوظيفة أو الائتمان أو أي عامل آخر.

٢. EvaluateEmployed\_HighCredit\_NoDependents\_Returns\_Eligible

تتحقق هذه الحالة من أن الموظف الذي لديه دخل جيد، ودرجة ائتمان عالية (٧٥٠ أو أكثر)، ولا يعيل أحد، يعتبر مؤهلاً للحصول على قرض.

٣. EvaluateEmployed\_MidCredit\_OwnHouse\_Returns\_Review

تتحقق هذه الحالة من أن الموظف الذي لديه درجة ائتمان متوسطة (بين ٦٠٠ و٦٩٩)، ويملك منزلًا، يتم تحويل طلبه إلى مراجعة يدوية.

٤. EvaluateUnemployed\_HighIncomeCreditAndHouse\_Returns\_Eligible

تتحقق هذه الحالة من أن الشخص غير الموظف، إذا كان دخله مرتفعًا (أكثر من ٥٠٠٠)، ودرجته الائتمانية عالية (٧٥٠ أو أكثر)، ويملك منزلًا، يعتبر مؤهلاً.

٥. EvaluateUnemployed\_LowCredit\_Returns\_NotEligible

تتحقق هذه الحالة من أن الشخص غير الموظف الذي لديه درجة ائتمان ضعيفة (أقل من ٦٠٠) ولا يملك منزلًا يتم رفض طلبه بشكل مباشر.

The screenshot shows a Visual Studio IDE with a C# test file named `LoanApp.Tests.cs` in the `LoanApp.Tests` namespace. The code defines a `LoanEvaluatorTests` class with three test methods: `GetLoanEligibility_Should_Return_NotEligible_When_Income_Low()`, `EvaluateEmployed_HighCredit_NoDependents_Returns_Eligible()`, and `EvaluateEmployed_MidCredit_OwnHouse_Returns_Review()`. Each method uses `Assert.Equal` to verify the output of the `LoanEvaluator` class.

The terminal output shows the successful execution of the tests:

```

Restore complete (1.7s)
PS C:\Users\AboOmar\Desktop\homework4> dotnet test
Restore complete (5.0s) -> LoanEvaluator.Core\bin\Debug\net9.0\LoanEvaluator.Core.dll
LoanEvaluator.Tests succeeded (2.2s) -> LoanEvaluator.Tests\bin\Debug\net9.0\LoanEvaluator.Tests.dll
[xunit.net 00:00:01.60] Discovering: LoanEvaluator.Tests
[xunit.net 00:00:01.85] Discovered: LoanEvaluator.Tests
[xunit.net 00:00:02.47] Starting: LoanEvaluator.Tests
[xunit.net 00:00:02.47] Finished: LoanEvaluator.Tests
LoanEvaluator.Tests test succeeded (7.2s)

Test summary: total: 5; failed: 0; succeeded: 5; skipped: 0; duration: 7.2s
Build succeeded in 16.9s
PS C:\Users\AboOmar\Desktop\homework4>

```

مقارنة قبل وبعد عملية التحسين:

- قبل:
- تعقيد حلقي (Cyclomatic Complexity) عالي = ١٣
- صعب الفهم والصيانة
- يصعب اختبار الحالات الداخلية كل على حدة
- دالة واحدة تحتوي كل منطق القرار

بعد:

- تقليل التعقيد الحلقي (Cyclomatic Complexity) إلى ٣ في كل تابع فرعي بدلاً من ١٣ في تابع واحد.
- تبسيط منطق القرار عبر توزيع الشروط إلى توابع صغيرة متخصصة.
- تحسين قابلية القراءة والفهم بفضل فصل الحالات (موظف / غير موظف) في توابع مستقلة.
- سهولة اختبار كل حالة على حدة عبر كتابة اختبارات وحدة منفصلة لكل تابع فرعي.
- رفع قابلية الصيانة والتطوير، حيث يمكن تعديل منطق فئة معينة دون التأثير على الأخرى.
- إعادة استخدام التوابع الفرعية عند الحاجة دون تكرار الكود.
- تهيئة الكود ليكون أكثر مرونة للتوسع مستقبلاً مثل إضافة قواعد جديدة أو شروط خاصة دون تعقيد إضافي.

