



## گزارش پروژه اول

نام و نام خانوادگی

نگار فتحی

شماره دانشجویی

۹۷۷۲۳۱۳۷

استاد

دکتر سعید پارسا

درس

کامپایلر پیشرفته

در پروژه انجام شده، به کمک کامپایلر دانت، رازلین، بررسی شده است که یک برنامه نوشته شده به زبان C#، از لحاظ موارد زیر، طبق اصول کد پاک هست یا خیر.

(۱) نام متغیر: باتوجه به مطالب ارائه شده در صفحه ۱۷ کتاب Clean Code، تمامی کلمات تشکیل دهنده نام متغیر باید معنی دار (جزء کلمات انگلیسی) بوده و اولین کلمه تشکیل دهنده نام متغیر نیز باید از جنس اسم باشد.

(۲) نام متد: باتوجه به مطالب ارائه شده در صفحه ۲۵ کتاب Clean Code، تمامی کلمات تشکیل دهنده نام متد باید معنی دار (جزء کلمات انگلیسی) بوده و اولین کلمه تشکیل دهنده نام متد نیز باید از جنس فعل باشد.

(۳) پارامترهای متد: باتوجه به مطالب ارائه شده در صفحه ۴۰ کتاب Clean Code، تعداد پارامترهای متد نباید از ۴ مورد تجاوز کند.

(۴) اندازه متد: باتوجه به مطالب ارائه شده در صفحه ۳۴ کتاب Clean Code، اندازه متد نباید از ۲۴ خط تجاوز کند.

(۵) یکتایی عملکرد متد: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، متد باید یک مسئولیت داشته باشد.

(۶) بلوک if: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، بلوک درون دستورالعمل if باید ۱ خط باشد. احتمالاً آن خط یک فراخوانی تابع است.

(۷) بلوک else: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، بلوک درون دستورالعمل else باید ۱ خط باشد. احتمالاً آن خط یک فراخوانی تابع است.

(۸) بلوک for: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، بلوک درون دستورالعمل for باید ۱ خط باشد. احتمالاً آن خط یک فراخوانی تابع است.

(۹) بلوک while: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، بلوک درون دستورالعمل while باید ۱ خط باشد. احتمالاً آن خط یک فراخوانی تابع است.

(۱۰) if تودرتو: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، دستورالعمل های if تودرتو نباید از ۲ سطح تجاوز کنند.

(۱۱) for تودرتو: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، دستورالعمل های for تودرتو نباید از ۲ سطح تجاوز کنند.

(۱۲) while تودرتو: باتوجه به مطالب ارائه شده در صفحه ۳۵ کتاب Clean Code، دستورالعمل های while تودرتو نباید از ۲ سطح تجاوز کنند.

جهت انجام این پروژه، گام های زیر طی شده است.

## ایجاد پروژه:

ابتدا یک پروژه Windows Forms Application با نام CleanCode در محیط ویژوال استودیو ایجاد می‌کنیم. بدین‌منظور باید مراحل زیر طی شود.

Visual Studio ⇒ File ⇒ New ⇒ Project ⇒ Visual C# ⇒ Windows Forms App (.NET Framework) ⇒ Name: CleanCode ⇒ Framework: .NET Framework 4.6.1 ⇒ Ok.

## نصب رازلین:

از آنجایی که این پروژه به کمک کامپایلر دات‌نت، رازلین، انجام شده‌است، باید مراحل نصب آن به‌صورت زیر طی شود.

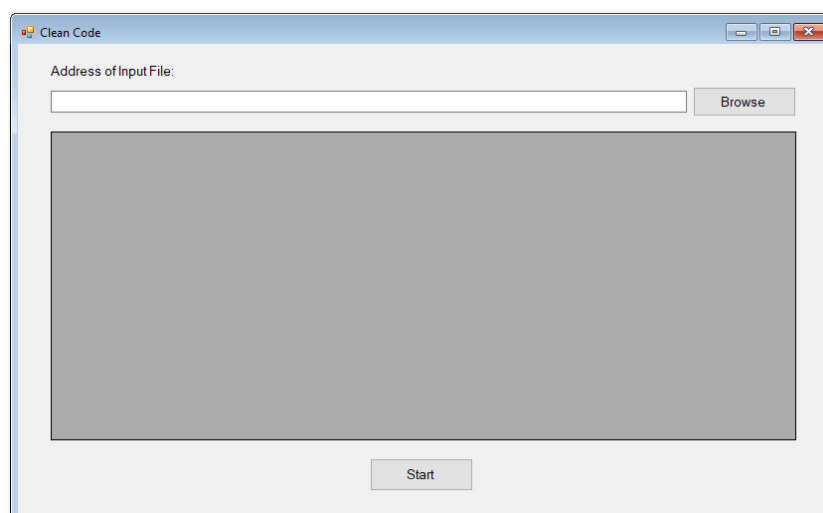
Visual Studio ⇒ Tools ⇒ Get Tools and Features... ⇒ Individual components ⇒ .NET Compiler Platform SDK ⇒ Modify.

Visual Studio ⇒ View ⇒ Other Windows ⇒ Syntax Visualizer.

Visual Studio ⇒ Solution Explorer ⇒ References ⇒ Manage NuGet Packages... ⇒ Browse ⇒ Microsoft.CodeAnalysis (Version: 2.0.0) ⇒ Install ⇒ I Accept.

## ایجاد فرم‌های مورد نیاز:

با قرار دادن یک Label، یک TextBox، دو Button Browse و Start و یک DataGridView، فرمی مطابق شکل زیر ایجاد می‌کنیم.



ابتدا باید بتوان با کلیک کردن بر روی Browse Button، آدرس یک برنامه نوشته‌شده به زبان C# را گرفت. بدین‌منظور به قطعه کد زیر نیاز است.

```
using System.Windows.Forms;  
private void buttonBrowse_Click(object sender, EventArgs e)
```

```
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string fileName = openFileDialog.FileName;
        textBox.Text = fileName;
    }
}
```

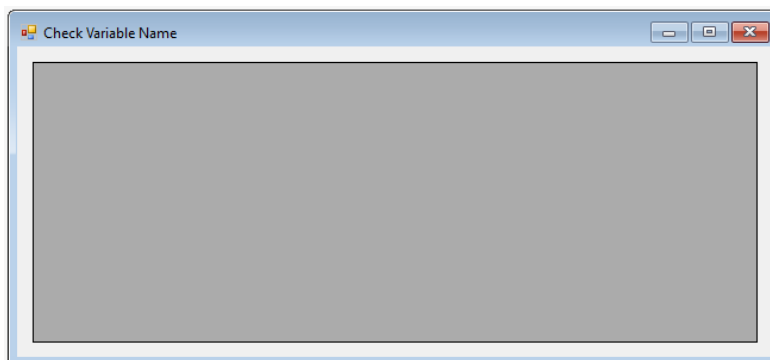
سپس باید بتوان با کلیک کردن بر روی Start Button، برنامه موردنظر را از لحاظ ۱۲ مورد ذکرشده در ابتدای گزارش، مورد تحلیل قرار داد. بدین منظور در قطعه کد زیر ۱۲ متد فراخوانی شده است که به ترتیب متناظر با ۱۲ مورد ذکرشده در ابتدای گزارش هستند.

```
private void buttonStart_Click(object sender, EventArgs e)
{
    CheckVariableName();
    CheckMethodName();
    CheckMethodParameters();
    CheckMethodSize();
    CheckMethodSingleResponsibility();
    CheckIfBlock();
    CheckElseBlock();
    CheckForBlock();
    CheckWhileBlock();
    CheckNestedIf();
    CheckNestedFor();
    CheckNestedWhile();
}
```

تمامی این ۱۲ متد، متدی به نام Show را فراخوانی می کنند که وظیفه دارد نتیجه تحلیل برنامه موردنظر را در DataGridView نمایش دهد. برطبق قطعه کد زیر، DataGridView دو ستون خواهد داشت که ستون اول توضیحی در مورد یکی از اصول کد پاک ذکرشده در ابتدای گزارش و ستون دوم تعداد دفعاتی است که این اصل در برنامه موردنظر نقض شده است.

```
public void Show(string x, string y)
{
    dataGridView.ColumnCount = 2;
    dataGridView.Columns[0].Name = "Clean Code Principle";
    dataGridView.Columns[0].MinimumWidth = 595;
    dataGridView.Columns[1].Name = "Count";
    dataGridView.Columns[1].MinimumWidth = 45;
    string[] row = new string[] { x, y };
    dataGridView.Rows.Add(row);
}
```

حال باید بتوان با دابل کلیک کردن بر روی هریک از سطرهای DataGridView جزئیات تحلیل صورت گرفته را در یک فرم دیگر مشاهده کرد. برای مثال با دابل کلیک کردن بر روی سطر اول DataGridView که مربوط به بررسی نام متغیرها است، باید فرمی مطابق شکل زیر نمایش داده شود که شامل یک DataGridView است.



با طی کردن مراحل زیر می‌توان این فرم را به پروژه اضافه کرد.

Visual Studio ⇒ Solution Explorer ⇒ Project ⇒ Add ⇒ New Item... ⇒ Windows Form ⇒ Name: FormCheckVariableName ⇒ Add.

برطبق قطعه کد زیر، DataGridView موجود در این فرم، سه ستون خواهد داشت که ستون اول نام متغیری است که طبق اصول کد پاک نام‌گذاری نشده‌است و ستون دوم شماره خطی از برنامه است که این متغیر در آن خط تعریف شده‌است و در نهایت ستون سوم حاوی یک پیام مناسب است. قرار است اولین متد از ۱۲ متد ذکر شده در بالا یعنی متد CheckVariableName، در صورت تشخیص متغیری که طبق اصول کد پاک نام‌گذاری نشده‌است، متد Show تعریف شده در قطعه کد زیر را فراخوانی کند.

```
using System.Windows.Forms;

namespace CleanCode
{
    public partial class FormCheckVariableName : Form
    {
        public FormCheckVariableName()
        {
            InitializeComponent();
        }

        public void Show(string x, string y, string z)
        {
            dataGridView.ColumnCount = 3;
            dataGridView.Columns[0].Name = "Variable Name";
            dataGridView.Columns[0].MinimumWidth = 200;
            dataGridView.Columns[1].Name = "Line";
            dataGridView.Columns[1].MinimumWidth = 130;
            dataGridView.Columns[2].Name = "Message";
            dataGridView.Columns[2].MinimumWidth = 240;
            string[] row = new string[] { x, y, z };
            dataGridView.Rows.Add(row);
        }
    }
}
```

باید همانند فرم مثال زده شده، ۱۱ فرم دیگر ایجاد کرد. سپس به قطعه کد زیر نیاز است تا به ازای دابل کلیک کردن هر سطر DataGridView فرم اولیه، یکی از ۱۲ فرم ایجاد شده را با توجه به این که کدام سطر دابل کلیک شده‌است، نمایش دهد.

```

FormCheckVariableName formCheckVariableName = new FormCheckVariableName();
FormCheckMethodName formCheckMethodName = new FormCheckMethodName();
FormCheckMethodParameters formCheckMethodParameters = new FormCheckMethodParameters();
FormCheckMethodSize formCheckMethodSize = new FormCheckMethodSize();
FormCheckMethodSingleResponsibility formCheckMethodSingleResponsibility = new
FormCheckMethodSingleResponsibility();
FormCheckIfBlock formCheckIfBlock = new FormCheckIfBlock();
FormCheckElseBlock formCheckElseBlock = new FormCheckElseBlock();
FormCheckForBlock formCheckForBlock = new FormCheckForBlock();
FormCheckWhileBlock formCheckWhileBlock = new FormCheckWhileBlock();
FormCheckNestedIf formCheckNestedIf = new FormCheckNestedIf();
FormCheckNestedFor formCheckNestedFor = new FormCheckNestedFor();
FormCheckNestedWhile formCheckNestedWhile = new FormCheckNestedWhile();

private void dataGridView_RowHeaderMouseDoubleClick(object sender, DataGridViewCellMouseEventArgs)
{
    switch (dataGridView.SelectedRows[0].Cells[0].RowIndex)
    {
        case 0:
            formCheckVariableName.Show();
            break;
        case 1:
            formCheckMethodName.Show();
            break;
        case 2:
            formCheckMethodParameters.Show();
            break;
        case 3:
            formCheckMethodSize.Show();
            break;
        case 4:
            formCheckMethodSingleResponsibility.Show();
            break;
        case 5:
            formCheckIfBlock.Show();
            break;
        case 6:
            formCheckElseBlock.Show();
            break;
        case 7:
            formCheckForBlock.Show();
            break;
        case 8:
            formCheckWhileBlock.Show();
            break;
        case 9:
            formCheckNestedIf.Show();
            break;
        case 10:
            formCheckNestedFor.Show();
            break;
        case 11:
            formCheckNestedWhile.Show();
            break;
        default:
            break;
    }
}

```

```
}  
}
```

## بررسی نام متغیرها:

در فرایند بررسی پاک بودن نام متغیرها، از آن جایی که این نام‌ها می‌توانند از کنارهم قرار گرفتن چندین کلمه به صورت camelCase تشکیل شده باشند، ابتدا باید بتوان با تشخیص حروف بزرگ موجود در آن‌ها، کلمات آن‌ها را از یکدیگر جدا کرد که این امر به کمک قطعه کد زیر حاصل خواهد شد.

```
public string SplitCamelCase(string x)  
{  
    return System.Text.RegularExpressions.Regex.Replace(x, "[A-Z]", " $1",  
        System.Text.RegularExpressions.RegexOptions.Compiled).Trim();  
}
```

بعد از جداسازی کلمات تشکیل دهنده نام متغیرها، باید به کمک یک مجموعه از لغات انگلیسی، تشخیص دهیم که کدام یک از این کلمات، یک لغت انگلیسی معتبر است و کدام یک نیست. نام تشکیل شده از یک کلمه که جزء مجموعه لغات انگلیسی نیست، یک نام ناپاک است. تشخیص قرار داشتن یک کلمه در مجموعه لغات انگلیسی به کمک قطعه کد زیر حاصل خواهد شد.

```
using NetSpell.SpellChecker.Dictionary;  
using NetSpell.SpellChecker;  
public bool IsEnglishWord(string x)  
{  
    WordDictionary dictionary = new WordDictionary();  
    dictionary.DictionaryFile = @"..\..\NewFolder\en-US.dic";  
    dictionary.Initialize();  
    Spelling spell = new Spelling();  
    spell.Dictionary = dictionary;  
    if (spell.TestWord(x))  
    {  
        return true;  
    }  
    return false;  
}
```

قطعه کد ارائه شده، نیازمند فراهم نمودن پیش نیازهای زیر است.

<https://github.com/loresoft/NetSpell> ⇒ Download.

Visual Studio ⇒ Solution Explorer ⇒ References ⇒ Manage NuGet Packages... ⇒ Brows ⇒ NetSpell (Version: 2.1.7) ⇒ Install ⇒ I Accept.

حال باید بررسی کرد که اولین کلمه تشکیل دهنده نام متغیرها و توابع که جزء مجموعه لغات انگلیسی بودند، چه برجستگی (اسم، فعل یا ...) دارند. این برجستگی باید برای متغیرها، اسم باشد. در غیر این صورت، نام گذاری‌ها ناپاک هستند. جهت برجستگی زدن از PoS Tagger استفاده می‌کنیم. بدین منظور باید مراحل زیر طی شده و از قطعه کد زیر استفاده کرد.

<https://sergey-tihon.github.io/Stanford.NLP.NET/StanfordPOSTagger.html> ⇒ Download Stanford POS Tagger full archive with models.

Visual Studio ⇒ Solution Explorer ⇒ References ⇒ Manage NuGet Packages... ⇒ Brows ⇒ Stanford.NLP.POSTagger (Version: 3.9.2) ⇒ Install ⇒ I Accept.

```
using edu.stanford.nlp.tagger.maxent;
using java.io;
using java.util;
using edu.stanford.nlp.ling;
using Console = System.Console;

public string Tag(string x)
{
    var model = new MaxentTagger(@"..\..\NewFolder\wsj-0-18-bidirectional-nodistsim.tagger", null, false);
    var sentence = MaxentTagger.tokenizeText(new StringReader(x)).ToArray();
    foreach (ArrayList i in sentence)
    {
        var taggedSentence = model.tagSentence(i);
        return SentenceUtils.listToString(taggedSentence, false);
    }
    return null;
}
```

حال نیازمند قطعه کدی هستیم که با فراخوانی مناسب سه قطعه کد بالا، فرایند بررسی پاک بودن نام متغیرها را به صورت یکپارچه انجام دهد که بدین منظور از قطعه کد زیر (متد CleanVariableName) استفاده می کنیم.

```
public void CleanVariableName(string x, string y)
{
    string text1 = SplitCamelCase(x);
    string[] text2 = text1.Split(' ');
    for (int i = 0; i < text2.Length; i++)
    {
        if (IsEnglishWord(text2[i]) != true)
        {
            formCheckVariableName.Show(x, y, "It contains a non-English word.");
            break;
        }
    }
    if (Tag(text2[0]) != text2[0].ToString() + "/NN")
    {
        formCheckVariableName.Show(x, y, "Its first word is not Noun.");
    }
}
```


درنهایت، قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckVariableName فراخوانی شود که برنامه موردنظر را از لحاظ پاک بودن نام متغیرهای استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به اعلان متغیرها را استخراج نموده و برای هریک از آنها، متد CleanVariableName را فراخوانی می کنیم. لازم به ذکر است که متغیرهایی که جهت استفاده به عنوان شمارنده حلقه های for تعریف می شوند، نباید مورد بررسی قرار گیرند که بدین منظور



از ریشه درخت شروع به پیمایش کرده و ساختارهای مربوط به دستورالعمل‌های for را استخراج نموده و درون یک ArrayList قرار می‌دهیم و سپس با قراردادن یک جمله شرطی که نام متغیرها را با نام‌های موجود در ArrayList مقایسه می‌کند، از بررسی متغیرهای تعریف‌شده به‌عنوان شمارنده حلقه‌های for اجتناب می‌کنیم.

```
using Microsoft.CodeAnalysis.CSharp;
using Microsoft.CodeAnalysis.CSharp.Syntax;
using System.Linq;

public void CheckVariableName()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var variableDeclarator = root.DescendantNodes().OfType<VariableDeclaratorSyntax>();
    var forStatement = root.DescendantNodes().OfType<ForStatementSyntax>();
    ArrayList arrayList = new ArrayList();
    foreach (var i in forStatement)
    {
        arrayList.Add(i.DescendantNodes().OfType<IdentifierNameSyntax>().First().Identifier.ToString());
    }
    foreach (var i in variableDeclarator)
    {
        if (!arrayList.Contains(i.Identifier.ToString()))
        {
            count = count + 1;
            CleanVariableName(i.Identifier.ToString(),
                tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
        }
    }
    Show("All words forming the variable name should be meaningful (English word). The first word forming the variable name should be Noun. Clean Code book. Page 17.", count.ToString());
}
```

بررسی نام متدها: 

در فرایند بررسی پاک‌بودن نام متدها نیز همانند فرایند بررسی پاک‌بودن نام متغیرها، نیازمند متدهای SplitCamelCase، IsEnglishWord و Tag هستیم. همچنین به یک متد دیگر، مشابه متد CleanVariableName با نام CleanMethodName نیاز است که کد آن به‌صورت زیر است.

```
public void CleanMethodName(string x, string y)
{
    string text1 = SplitCamelCase(x);
    string[] text2 = text1.Split(' ');
    for (int i = 0; i < text2.Length; i++)
    {
        if (IsEnglishWord(text2[i]) != true)
        {
            formCheckMethodName.Show(x, y, "It contains a non-English word.");
            break;
        }
    }
    if (Tag(text2[0]) != text2[0].ToString() + "/VB")
```

```

    {
        formCheckMethodName.Show(x, y, "Its first word is not Verb.");
    }
}

```

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckMethodName فراخوانی شود که برنامه موردنظر را از لحاظ پاک بودن نام متدهای استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به اعلان متدها را استخراج نموده و برای هریک از آنها، متد CleanMethodName را فراخوانی می کنیم.

```

public void CheckMethodName()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var methodDeclaration = root.DescendantNodes().OfType<MethodDeclarationSyntax>();
    foreach (var i in methodDeclaration)
    {
        count = count + 1;
        CleanMethodName(i.Identifier.ToString(),
            tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
    }
    Show("All words forming the method name should be meaningful (English word). The first word forming the method name should be Verb. Clean Code book. Page 25.", count.ToString());
}

```

 بررسی پارامترهای متد:

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckMethodParameters فراخوانی شود که برنامه موردنظر را از لحاظ تعداد پارامترهای متدهای استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به اعلان متدها را استخراج نموده و برای هریک از آنها، تعداد پارامترها را شمارش می کنیم که این تعداد نباید بیش تر از ۴ مورد باشد.

```

public void CheckMethodParameters()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var methodDeclaration = root.DescendantNodes().OfType<MethodDeclarationSyntax>();
    foreach (var i in methodDeclaration)
    {
        if (i.ParameterList.Parameters.Count > 4)
        {
            count = count + 1;
            formCheckMethodParameters.Show(i.Identifier.ToString(),
                tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
        }
    }
}

```

```

    }
    Show("The number of the method parameters should not exceed 4 cases. Clean Code book. Page 40.", count.ToString());
}

```

### بررسی اندازه متد:

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckMethodSize فراخوانی شود که برنامه موردنظر را از لحاظ اندازه متدهای استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به اعلان متدها را استخراج نموده و برای هریک از آنها، شماره خط شروع و پایان را محاسبه می‌کنیم که تفاضل این دو نباید بیش‌تر از ۲۴ خط باشد.

```

public void CheckMethodSize()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var methodDeclaration = root.DescendantNodes().OfType<MethodDeclarationSyntax>();
    foreach (var i in methodDeclaration)
    {
        var startLinePosition = tree.GetLineSpan(i.Span).StartLinePosition.Line;
        var endLinePosition = tree.GetLineSpan(i.Span).EndLinePosition.Line;
        if (endLinePosition - startLinePosition + 1 > 24)
        {
            count = count + 1;
            formCheckMethodSize.Show(i.Identifier.ToString(),
            tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
        }
    }
    Show("The size of the method should not exceed 24 lines. Clean Code book. Page 34.",
    count.ToString());
}

```

### بررسی یکتایی عملکرد متد:

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckMethodSingleResponsibility فراخوانی شود که برنامه موردنظر را از لحاظ یکتایی عملکرد متدهای استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به اعلان متدها را استخراج می‌کنیم. حال باید از ریشه درخت مربوط به هریک از این ساختارها، شروع به پیمایش کرده و ساختارهای مربوط به دستورالعمل‌های return را استخراج نموده و تعداد آنها را شمارش کنیم. این تعداد، در صورتی که عمل return بر روی عبارتهای مختلفی صورت گرفته باشد، نباید بیش‌تر از ۱ باشد.

```

public void CheckMethodSingleResponsibility()

```

```

{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var methodDeclaration = root.DescendantNodes().OfType<MethodDeclarationSyntax>();
    foreach (var i in methodDeclaration)
    {
        var returnStatement = i.DescendantNodes().OfType<ReturnStatementSyntax>();
        if (returnStatement.Count() > 1)
        {
            ArrayList arrayList = new ArrayList();
            foreach (var j in returnStatement)
            {
                arrayList.add(j.Expression);
            }
            for (int k = 1; k < arrayList.size(); k++)
            {
                if (arrayList.toArray()[0].ToString() != arrayList.toArray()[k].ToString())
                {
                    count = count + 1;
                    formCheckMethodSingleResponsibility.Show(i.Identifier.ToString(), tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
                    break;
                }
            }
        }
    }
    Show("The method should have one responsibility (single responsibility principle). Clean Code book. Page 35.", count.ToString());
}

```

### بررسی بلوک if

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckIfBlock فراخوانی شود که برنامه موردنظر را از لحاظ پاک نوشته شدن بدنه دستورالعمل های if استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل های if را استخراج می کنیم. حال باید از ریشه درخت مربوط به هریک از این ساختارها، شروع به پیمایش کرده و اولین ساختار مربوط به دستورالعمل else را استخراج نموده و برای این دو، شماره خط شروع را محاسبه کنیم (شماره خط پایان یک دستورالعمل if برابر با شماره خط شروع اولین دستورالعمل else بعد از آن است) که تفاضل این دو باید ۴ (یک خط دستورالعمل if، یک خط پرانتز باز، یک خط بدنه دستورالعمل if و یک خط پرانتز بسته) باشد.

```

public void CheckIfBlock ()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var ifStatement = root.DescendantNodes().OfType<IfStatementSyntax>();
}

```

```

foreach (var i in ifStatement)
{
    var elseClause = i.DescendantNodes().OfType<ElseClauseSyntax>().First();
    var startLinePosition = tree.GetLineSpan(i.Span).StartLinePosition.Line;
    var endLinePosition = tree.GetLineSpan(elseClause.Span).StartLinePosition.Line - 1;
    if (endLinePosition - startLinePosition + 1 != 4)
    {
        count = count + 1;
        formCheckIfBlock.Show(tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString(
    ));
    }
}
Show("The block within If statement should be 1 line. Probably that line should be a function call.
Clean Code book. Page 35.", count.ToString());
}

```

بررسی بلوک else:

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckElseBlock فراخوانی شود که برنامه موردنظر را از لحاظ پاک نوشته شدن بدنه دستورالعمل‌های else استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل‌های if را استخراج می‌کنیم. حال باید از ریشه درخت مربوط به هریک از این ساختارها، شروع به پیمایش کرده و آخرین ساختار مربوط به دستورالعمل else را استخراج نموده و برای آن، شماره خط شروع و پایان را محاسبه کنیم که تفاضل این دو باید ۴ (یک خط دستورالعمل else، یک خط پرانتز باز، یک خط بدنه دستورالعمل else و یک خط پرانتز بسته) باشد.

```

public void CheckElseBlock()
{
    int count = 0;
    ArrayList arrayList = new ArrayList();
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var ifStatement = root.DescendantNodes().OfType<IfStatementSyntax>();
    foreach (var i in ifStatement)
    {
        var elseClause = i.DescendantNodes().OfType<ElseClauseSyntax>().Last();
        var startLinePosition = tree.GetLineSpan(elseClause.Span).StartLinePosition.Line;
        var endLinePosition = tree.GetLineSpan(elseClause.Span).EndLinePosition.Line;
        if (endLinePosition - startLinePosition + 1 != 4)
        {
            if
            (arrayList.Contains(tree.GetLineSpan(elseClause.Span).StartLinePosition.Line.ToString()) !=
            true)
            {
                count = count + 1;
                arrayList.Add(tree.GetLineSpan(elseClause.Span).StartLinePosition.Line.T
                oString());
                formCheckElseBlock.Show(tree.GetLineSpan(elseClause.Span).StartLineP
                osition.Line.ToString());
            }
        }
    }
}

```

```

    }
}
Show("The block within Else statement should be 1 line. Probably that line should be a function call.
Clean Code book. Page 35.", count.ToString());
}

```

### بررسی بلوک for

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckForBlock فراخوانی شود که برنامه موردنظر را از لحاظ پاک نوشته شدن بدنه دستورالعمل‌های for استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل‌های for را استخراج نموده و برای هریک از آن‌ها، شماره خط شروع و پایان را محاسبه می‌کنیم که تفاضل این دو باید ۴ (یک خط دستورالعمل for، یک خط پرانتز باز، یک خط بدنه دستورالعمل for و یک خط پرانتز بسته) باشد.

```

public void CheckForBlock()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var forStatement = root.DescendantNodes().OfType<ForStatementSyntax>();
    foreach (var i in forStatement)
    {
        var startLinePosition = tree.GetLineSpan(i.Span).StartLinePosition.Line;
        var endLinePosition = tree.GetLineSpan(i.Span).EndLinePosition.Line;
        if (endLinePosition - startLinePosition + 1 != 4)
        {
            count = count + 1;
            formCheckForBlock.Show(tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
        }
    }
    Show("The block within For statement should be 1 line. Probably that line should be a function call.
Clean Code book. Page 35.", count.ToString());
}

```

### بررسی بلوک while

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckWhileBlock فراخوانی شود که برنامه موردنظر را از لحاظ پاک نوشته شدن بدنه دستورالعمل‌های while استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل‌های while را استخراج نموده و برای هریک از آن‌ها، شماره خط شروع و پایان را محاسبه می‌کنیم

که تفاضل این دو باید ۴ (یک خط دستورالعمل while، یک خط پرانتز باز، یک خط بدنه دستورالعمل while و یک خط پرانتز بسته) باشد.

```
public void CheckWhileBlock()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var whileStatement = root.DescendantNodes().OfType<WhileStatementSyntax>();
    foreach (var i in whileStatement)
    {
        var startLinePosition = tree.GetLineSpan(i.Span).StartLinePosition.Line;
        var endLinePosition = tree.GetLineSpan(i.Span).EndLinePosition.Line;
        if (endLinePosition - startLinePosition + 1 != 4)
        {
            count = count + 1;
            formCheckWhileBlock.Show(tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString());
        }
    }
    Show("The block within While statement should be 1 line. Probably that line should be a function call. Clean Code book. Page 35.", count.ToString());
}
```

بررسی if تودرتو:

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckNestedIf فراخوانی شود که برنامه موردنظر را از لحاظ تعداد دستورالعمل های if تودرتو استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل های if را استخراج می کنیم. حال باید از ریشه درخت مربوط به هریک از این ساختارها، شروع به پیمایش کرده و ساختارهای مربوط به دستورالعمل های if را استخراج کنیم. این کار را دوباره تکرار کرده و سپس تعداد دستورالعمل های if را شمارش می کنیم که این تعداد نباید بیش تر از ۱ مورد باشد.

```
public void CheckNestedIf()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var ifStatement1 = root.DescendantNodes().OfType<IfStatementSyntax>();
    foreach (var i in ifStatement1)
    {
        var ifStatement2 = i.DescendantNodes().OfType<IfStatementSyntax>();
        foreach (var j in ifStatement2)
        {
            var ifStatement3 = j.DescendantNodes().OfType<IfStatementSyntax>();
            if (ifStatement3.Count() > 0)
            {
                count = count + 1;
            }
        }
    }
}
```

```

        formCheckNestedIf.Show(tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString() + " ---> " + tree.GetLineSpan(j.Span).StartLinePosition.Line.ToString() + " " + tree.GetLineSpan(ifStatement3.First().Span).StartLinePosition.Line.ToString());
    }
}
Show("The nested If statements should not exceed 2 levels. Clean Code book. Page 35.", count.ToString());
}

```

#### بررسی for تودرتو:

قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckNestedFor فراخوانی شود که برنامه موردنظر را از لحاظ تعداد دستورالعمل‌های for تودرتو استفاده شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با در دست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل‌های for را استخراج می‌کنیم. حال باید از ریشه درخت مربوط به هریک از این ساختارها، شروع به پیمایش کرده و ساختارهای مربوط به دستورالعمل‌های for را استخراج کنیم. این کار را دوباره تکرار کرده و سپس تعداد دستورالعمل‌های for را شمارش می‌کنیم که این تعداد نباید بیش‌تر از ۱ مورد باشد.

```

public void CheckNestedFor()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var forStatement1 = root.DescendantNodes().OfType<ForStatementSyntax>();
    foreach (var i in forStatement1)
    {
        var forStatement2 = i.DescendantNodes().OfType<ForStatementSyntax>();
        foreach (var j in forStatement2)
        {
            var forStatement3 = j.DescendantNodes().OfType<ForStatementSyntax>();
            if (forStatement3.Count() > 0)
            {
                count = count + 1;
                formCheckNestedFor.Show(tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString() + " ---> " + tree.GetLineSpan(j.Span).StartLinePosition.Line.ToString() + " " + tree.GetLineSpan(forStatement3.First().Span).StartLinePosition.Line.ToString());
            }
        }
    }
    Show("The nested For statements should not exceed 2 levels. Clean Code book. Page 35.", count.ToString());
}

```

#### بررسی while تودرتو:



قرار است بعد از کلیک کردن بر روی Start Button فرم اولیه، متدی به نام CheckNestedWhile فراخوانی شود که برنامه موردنظر را از لحاظ تعداد دستورالعمل‌های while تودرتو استفاده‌شده در آن تحلیل کند. بدین جهت باید در بدنه این متد، قطعه کد زیر را قرار داد. در این قطعه کد، درخت نحو برنامه موردنظر را ایجاد و با دردست داشتن ریشه درخت، شروع به پیمایش درخت کرده و ساختارهای مربوط به دستورالعمل‌های while را استخراج می‌کنیم. حال باید از ریشه درخت مربوط به هریک از این ساختارها، شروع به پیمایش کرده و ساختارهای مربوط به دستورالعمل‌های while را استخراج کنیم. این کار را دوباره تکرار کرده و سپس تعداد دستورالعمل‌های while را شمارش می‌کنیم که این تعداد نباید بیش‌تر از ۱ مورد باشد.

```
public void CheckNestedWhile()
{
    int count = 0;
    var tree = CSharpSyntaxTree.ParseText(System.IO.File.ReadAllText(textBox.Text));
    var root = (CompilationUnitSyntax)tree.GetRoot();
    var whileStatement1 = root.DescendantNodes().OfType<WhileStatementSyntax>();
    foreach (var i in whileStatement1)
    {
        var whileStatement2 = i.DescendantNodes().OfType<WhileStatementSyntax>();
        foreach (var j in whileStatement2)
        {
            var whileStatement3 = j.DescendantNodes().OfType<WhileStatementSyntax>();
            if (whileStatement3.Count() > 0)
            {
                count = count + 1;
                formCheckNestedWhile.Show(tree.GetLineSpan(i.Span).StartLinePosition.Line.ToString() + " ---> " + tree.GetLineSpan(j.Span).StartLinePosition.Line.ToString() + " ---> " + tree.GetLineSpan(whileStatement3.First().Span).StartLinePosition.Line.ToString());
            }
        }
    }
    Show("The nested While statements should not exceed 2 levels. Clean Code book. Page 35.", count.ToString());
}
```