|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«Московский технологический университет»**  **МИРЭА** | | |

Институт Информационных Технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ**

по Лабораторной Работе №6

на тему

«Playground. Рефлексия»

по дисциплине

«ООП»

Выполнил студент группы ИСБО-11-16 Шайхуллин С.В.

Принял Cтарший преподаватель Мирзоян Д.И.

Выполнено «\_\_\_» \_\_\_\_\_\_\_\_\_2017 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2017г.

Москва, 2017

**Теоретическая часть**

В мире .NET рефлексией (reflection) называется процесс обнаружения типов во время выполнения. С применением служб рефлексии те же самые метаданные, которые отображает утилита ildasm.exe, можно получать программно в виде удобной объектной модели. Например, рефлексия позволяет извлечь список всех типов, которые содержатся внутри определенной сборки \*.dll или \*.ехе (или даже внутри файла \*.netmodule если речь идет о многофайловой сборке), в том числе методов, полей, свойств и событий, определенных в каждом из них. Можно также динамически обнаруживать набор интерфейсов, которые поддерживаются данным типом, параметров, которые принимает данный метод, и других деталей подобного рода (таких как имена базовых классов, информация о пространствах имен, данные манифеста и т.д.).

Многие классы, поддерживающие рефлексию, входят в состав прикладного интерфейса .NET Reflection API, относящегося к пространству имен System.Reflection. Как и в любом другом пространстве имен, в System.Reflection (которое поставляется в составе сборки mscorlib.dll) содержится набор взаимосвязанных типов. Ниже описаны некоторые наиболее важные из этих типов:

Assembly

В этом абстрактном классе содержатся статические методы, которые позволяют загружать сборку, исследовать ее и производить с ней различные манипуляции

AssemblyName

Этот класс позволяет выяснить различные детали, связанные с идентификацией сборки (номер версии, информация о культуре и т.д.)

EventInfо

В этом абстрактном классе хранится информация о заданном событии

FieldInfo

В этом абстрактном классе хранится информация о заданном поле

MemberInfo

Этот абстрактный базовый класс определяет общее поведение для типов EventInfo, FieldInfo, MethodInfo и PropertyInfo

MethodInfo

В этом абстрактном классе содержится информация по заданному методу

Module

Этот абстрактный класс позволяет получить доступ к определенному модулю внутри многофайловой сборки

ParameterInfо

В этом классе хранится информация по заданному параметру

PropertyInfо

В этом абстрактном классе хранится информация по заданному свойству

Чтобы понять, каким образом использовать пространство имен System.Reflection для программного чтения метаданных .NET, необходимо сначала ознакомиться с классом System.Type.

Класс System.Type

Класс System.Type составляет ядро подсистемы рефлексии, поскольку он инкапсулирует тип данных. Он содержит многие свойства и методы, которыми можно пользоваться для получения информации о типе данных во время выполнения. Класс Type является производным от абстрактного класса System.Reflection.MemberInfo.

Следовательно, тип члена можно определить, проверив свойство MemberType. Так, если свойство MemberType имеет значение MemberTypes.Method, то проверяемый член является методом.

В класс MemberInfo входят два абстрактных метода: GetCustomAttributes () и IsDefined(). Оба метода связаны с атрибутами. Первый из них получает список специальных атрибутов, имеющих отношение к вызывающему объекту, а второй устанавливает, определен ли атрибут для вызывающего метода. В версию .NET Framework Version 4.0 внедрен метод GetCustomAttributesData(), возвращающий сведения о специальных атрибутах.

Класс System.Type имеет набор членов, которые могут применяться для изучения метаданных типа, и большинство из которых возвращает типы из пространства имен System.Reflection. Например, член Type.GetMethods() возвращает массив объектов типа MethodInfo, член Type.GetFields() — массив объектов типа FieldInfo. и т.д.

Рефлексия методов

С помощью методов и свойств класса Type можно получить подробные сведения о типе данных во время выполнения программы. Это довольно эффективное средство. Ведь получив сведения о типе данных, можно сразу же вызвать его конструкторы и методы или воспользоваться его свойствами. Следовательно, рефлексия позволяет использовать код, который не был доступен во время компиляции.

Прикладной интерфейс Reflection API весьма обширен и имеет ясную логическую структуру, поэтому, уяснив одну его часть, нетрудно понять и все остальное.

Чтобы проиллюстрировать базовый процесс рефлексии (и оценить пользу от System.Type), создадим новое консольное приложение. Это приложение будет отображать детали методов, свойств, полей и поддерживаемых интерфейсов (а также другие интересные элементы данных) для любого из типов, содержащихся как в самом приложении, так и в сборке mscorlib.dll (доступ к которой все приложения .NET получают автоматически)

// Для выполнения рефлексии должно импортироваться это пространство имен

using System.Reflection;

Далее потребуется модифицировать класс Program, определив в нем ряд статических методов так, чтобы каждый из них принимал единственный параметр System.Type и возвращал void.

Имея в своем распоряжении объект класса Type, можно получить список методов, поддерживаемых отдельным типом данных, используя метод GetMethods(). Ниже приведена одна из форм, подходящих для этой цели:

Methodlnfo[] GetMethods()

Этот метод возвращает массив объектов класса MethodInfo, которые описывают методы, поддерживаемые вызывающим типом. Класс MethodInfo находится в пространстве имен System.Reflection.

Класс MethodInfo является производным от абстрактного класса MethodBase, который в свою очередь наследует от класса MemberInfo. Это дает возможность пользоваться всеми свойствами и методами, определенными в этих трех классах. Например, для получения имени метода служит свойство Name. Особый интерес вызывают два члена класса MethodInfo: ReturnType и GetParameters().

Возвращаемый тип метода находится в доступном только для чтения свойстве ReturnType, которое является объектом класса Type. Метод GetParameters() возвращает список параметров, связанных с анализируемым методом. Ниже приведена его общая форма:

ParameterInfo[] GetParameters();

Сведения о параметрах содержатся в объекте класса ParameterInfо. В классе ParameterInfо определено немало свойств и методов, описывающих параметры. Особое значение имеют два свойства: Name — представляет собой строку, содержащую имя параметра, a ParameterType — описывает тип параметра, который инкапсулирован в объекте класса Type.

Рефлексия конструкторов, полей и свойств

В предыдущей статье мы рассмотрели пример использования рефлексии методов, таким же образом можно получить информацию о полях, свойствах и интерфейсах класса с помощью методов GetField() и GetInterfaces. Давайте модифицируем программу из предыдущей статьи и добавим обобщенные методы в класс Reflect, отображающие информацию о полях, свойствах и интерфейсах, добавив реализацию интерфейса в класс MyTestClass: Следует иметь в виду, что большинство из "получающих", т.е. get-методов в System.Type (GetMethods(), GetInterfaces() и т.д.), имеют перегруженные версии, принимающие значения из перечисления BindingFlags. Это позволяет более точно указать, поиск чего должен производиться (например, только статических членов, только общедоступных членов, включая приватные члены, и т.д.).

Сильные стороны рефлексии проявляются наиболее заметно лишь в том случае, если объект создается динамически во время выполнения. И для этого необходимо получить сначала список конструкторов, а затем экземпляр объекта заданного типа, вызвав один из этих конструкторов. Такой механизм позволяет получать во время выполнения экземпляр объекта любого типа, даже не указывая его имя в операторе объявления.

Конструкторы конкретного типа получаются при вызове метода GetConstructors() для объекта класса Type. Ниже приведена одна из наиболее часто используемых форм этого метода:

ConstructorInfо[] GetConstructors()

Метод GetConstructors() возвращает массив объектов класса ConstructorInfо, описывающих конструкторы. Класс ConstructorInfo является производным от абстрактного класса MethodBase, который в свою очередь наследует от класса MemberInfo. В нем также определен ряд собственных методов. К их числу относится интересующий нас метод GetConstructors(), возвращающий список параметров, связанных с конструктором. Этот метод действует таким же образом, как и упоминавшийся ранее метод GetParameters(), определенный в классе MethodInfo.

Как только будет обнаружен подходящий конструктор, для создания объекта вызывается метод Invoke(), определенный в классе ConstructorInfo. Ниже приведена одна из форм этого метода:

object Invoke(object[] parameters)

Любые аргументы, которые требуется передать методу, указываются в массиве parameters. Если же аргументы не нужны, то вместо массива parameters указывается пустое значение (null). Но в любом случае количество элементов массива parameters должно совпадать с количеством передаваемых аргументов, а типы аргументов — с типами параметров. Метод Invoke() возвращает ссылку на сконструированный объект.

Рефлексия обобщенных типов

При вызове Type.GetType() для получения описаний метаданных обобщенных типов должен обязательно применяться специальный синтаксис в виде символа обратной одинарной кавычки (') со следующим за ним числовым значением, которое представляет количество параметров, поддерживаемое данным типом. Например, чтобы отобразить описание метаданных обобщенного типа System.Collections.Generic.List<T>, приложению потребуется передать следующую строку:

System.Collections.Generic.List '1

Здесь используется числовое значение 1, поскольку List<T> имеет только один параметр. Для применения рефлексии в отношении типа Dictionary<TKey, TValue>, однако, пришлось бы указать значение 2:

System.Collections.Generic.Dictionary '2

Сборка несет в себе сведения о типах классов, структур и прочих элементов данных, которые в ней содержатся. Прикладной интерфейс Reflection API позволяет загрузить сборку, извлечь сведения о ней и получить экземпляры объектов любых открыто доступных в ней типов. Используя этот механизм, программа может выявлять свою среду и использовать те функциональные возможности, которые могут оказаться доступными без явного их определения во время компиляции. Это очень эффективный и привлекательный принцип. Представьте себе, например, программу, которая выполняет роль "браузера типов", отображая типы данных, доступные в системе, или же инструментальное средство разработки, позволяющее визуально составлять программы из различных типов данных, поддерживаемых в системе. А поскольку все сведения о типах могут быть извлечены и проверены, то ограничений на применение рефлексии практически не существует.

Для получения сведений о сборке сначала необходимо создать объект класса Assembly. В классе Assembly открытый конструктор не определяется. Вместо этого объект класса Assembly получается в результате вызова одного из его методов. Так, для загрузки сборки по заданному ее имени служит метод LoadFrom(). Ниже приведена его соответствующая форма:

static Assembly LoadFrom(string файл\_сборки)

где файл\_сборки-обозначает конкретное имя файла сборки.

Как только будет получен объект класса Assembly, появится возможность обнаружить определенные в нем типы данных, вызвав для него метод GetTypes() в приведенной ниже общей форме:

Type[] GetTypes()

Этот метод возвращает массив типов, содержащихся в сборке.

Для того чтобы продемонстрировать порядок обнаружения типов в сборке, потребуются два исходных файла. Первый файл возьмем из первого раздела данного руководства, где мы создали библиотеку классов fontinfo.dll. Вы можете использовать какую то свою библиотеку классов.

Динамически загружаемые сборки

При поиске внешних сборок, на которые ссылается текущая сборка, CLR-среда заглядывает в манифест сборки. Во многих случаях необходимо, чтобы сборки могли загружаться на лету программно, даже если в манифесте о них не упоминается. Формально процесс загрузки внешних сборок по требованию называется динамической загрузкой.

В пространстве имен System.Reflection поставляется класс по имени Assembly, с применением которого можно динамически загружать сборку, а также просматривать ее собственные свойства. Этот класс позволяет выполнять динамическую загрузку приватных и разделяемых сборок, причем находящихся в произвольных местах. По сути, класс Assembly предоставляет методы (в частности, Load() и LoadFrom(), которые позволяют программно поставлять ту же информацию, которая встречается в клиентских файлах \*.config.

Чтобы посмотреть, как обеспечивать динамическую загрузку на практике, создадим новый проект типа ConsoleApplication. Определим в нем метод Main(), который будет запрашивать у пользователя дружественное имя сборки для динамической загрузки. Кроме того, обеспечим передачу ссылки на эту сборку вспомогательному методу, который будет выводить имена всех содержащихся в сборке классов, интерфейсов, структур, перечислений и делегатов.

Обратите внимание, что статическому методу Assembly.Load() передается только дружественное имя сборки, которую требуется загрузить в память. Следовательно, чтобы подвергнуть рефлексии сборку fontinfo.dll с помощью этой программы, понадобится сначала скопировать двоичный файл fontinfo.dll в подкаталог bin\Debug внутри каталога приложения.

Чтобы сделать приложение более гибким, можно модифицировать код так, чтобы загрузка внешней сборки производилась с помощью метода Assembly.LoadFrom(), а не Assembly.Load():

try

{

ass = Assembly.LoadFrom(assemblyName);

Types(ass);

}

Метод Assembly.Load() имеет несколько перегруженных версий. Одна из них позволяет указать значение культуры (для локализуемых сборок), а также номер версии и значение маркера открытого ключа (для разделяемых сборок). Все вместе эти элементы, которые идентифицируют сборку, называются отображаемым именем (display name), формат которого выглядит так: сначала идет дружественное имя сборки, за ней строка разделенных запятыми пар "имя/значение", а потом необязательные спецификаторы (в любом порядке). Ниже приведен образец, которым следует пользоваться (необязательные элементы указаны в круглых скобках):

Имя (, Version = <старший номер>.<младший номер>.<номер сборки>.<номер редакции>)

(, Culture = <маркер культуры>) (, PublicKeyToken = <маркер открытого ключа>)

При создании отображаемого имени использование PublicKeyToken=null означает, что связывание и сопоставление должно выполняться со сборкой, не имеющей строгого имени, a Culture="" — что сопоставление должно выполняться с использованием культуры, которая принята на целевой машине по умолчанию:

// Выполнение загрузки fontinfo версии 1.0.0.0

//с использованием принятой по умолчанию культуры

Assembly а =

Assembly.Load(@"CarLibrary, Version=l.0.0.0, PublicKeyToken=null, Culture=""");

Следует также иметь в виду, что в пространстве имен System.Reflection поставляется тип AssemblyName, который позволяет представлять строковую информацию наподобие той, что была показана выше, в виде удобной объектной переменной. Обычно этот класс применяется вместе с классом System.Version, который позволяет упаковывать в объектно-ориентированную оболочку номер версии сборки. После создания отображаемое имя может передаваться перегруженной версии метода Assembly.Load().

**Задание**

1) Создать класс синтаксического дерева

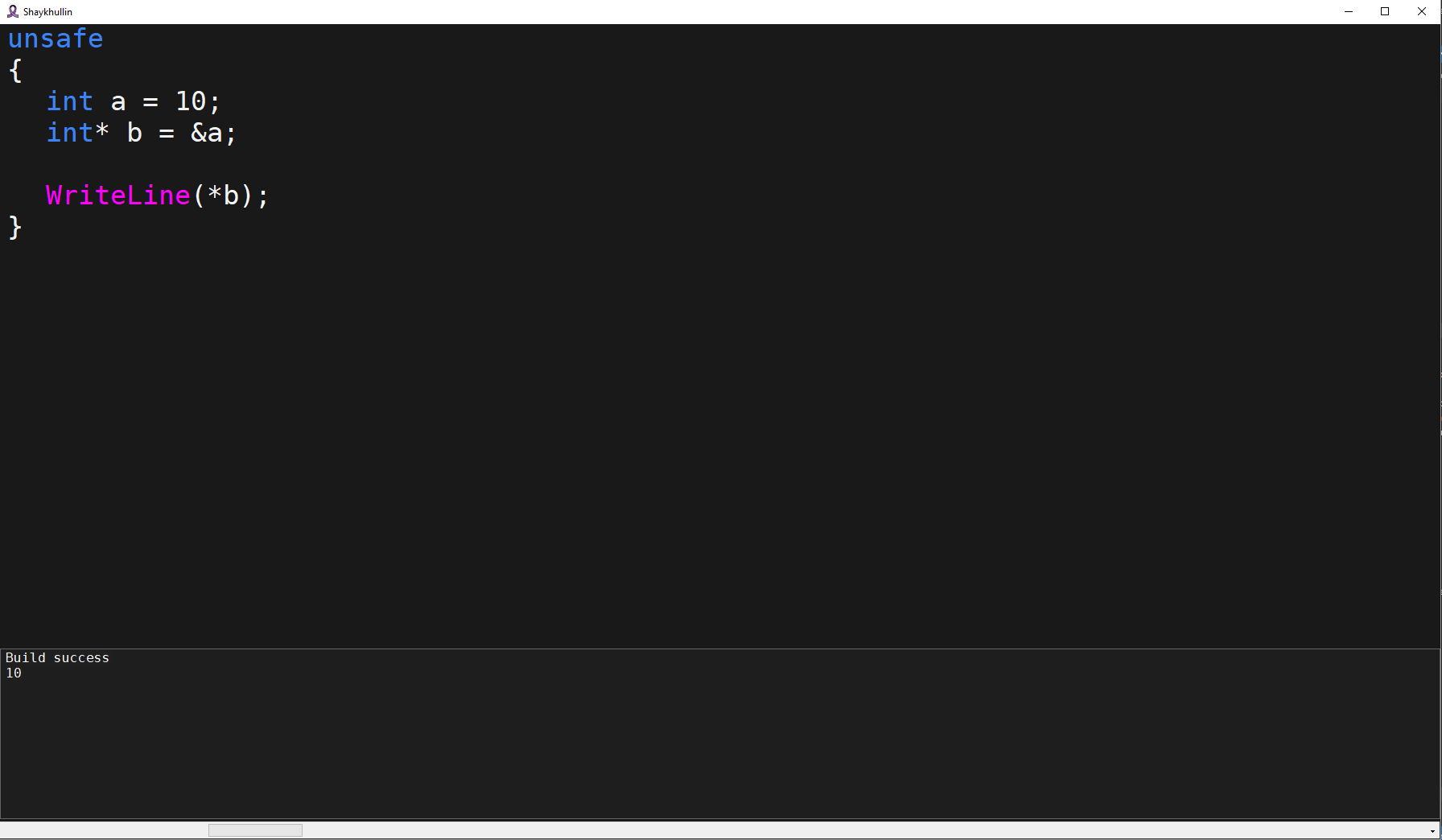
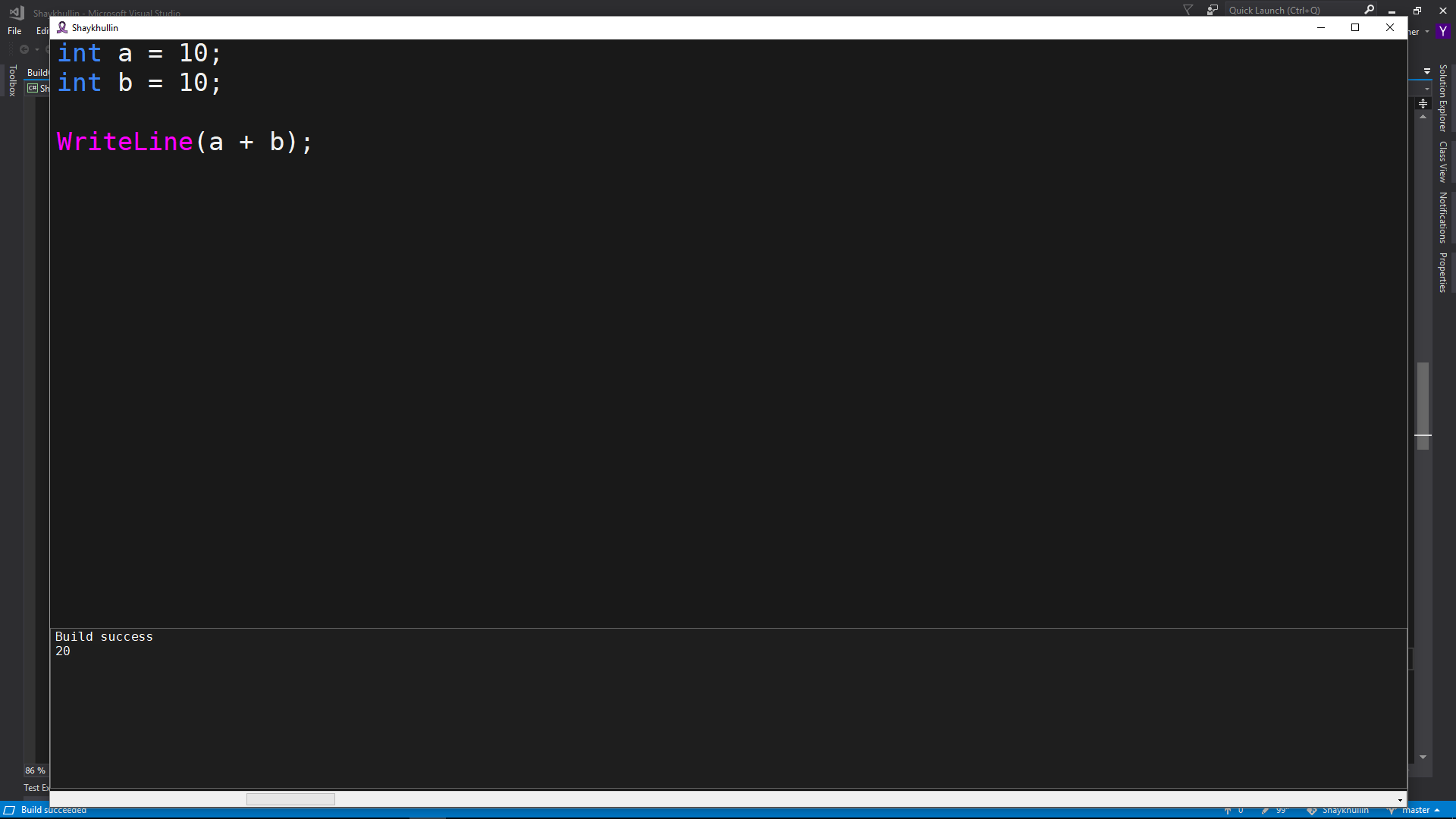
2) Поместить на форму все необходимые компоненты

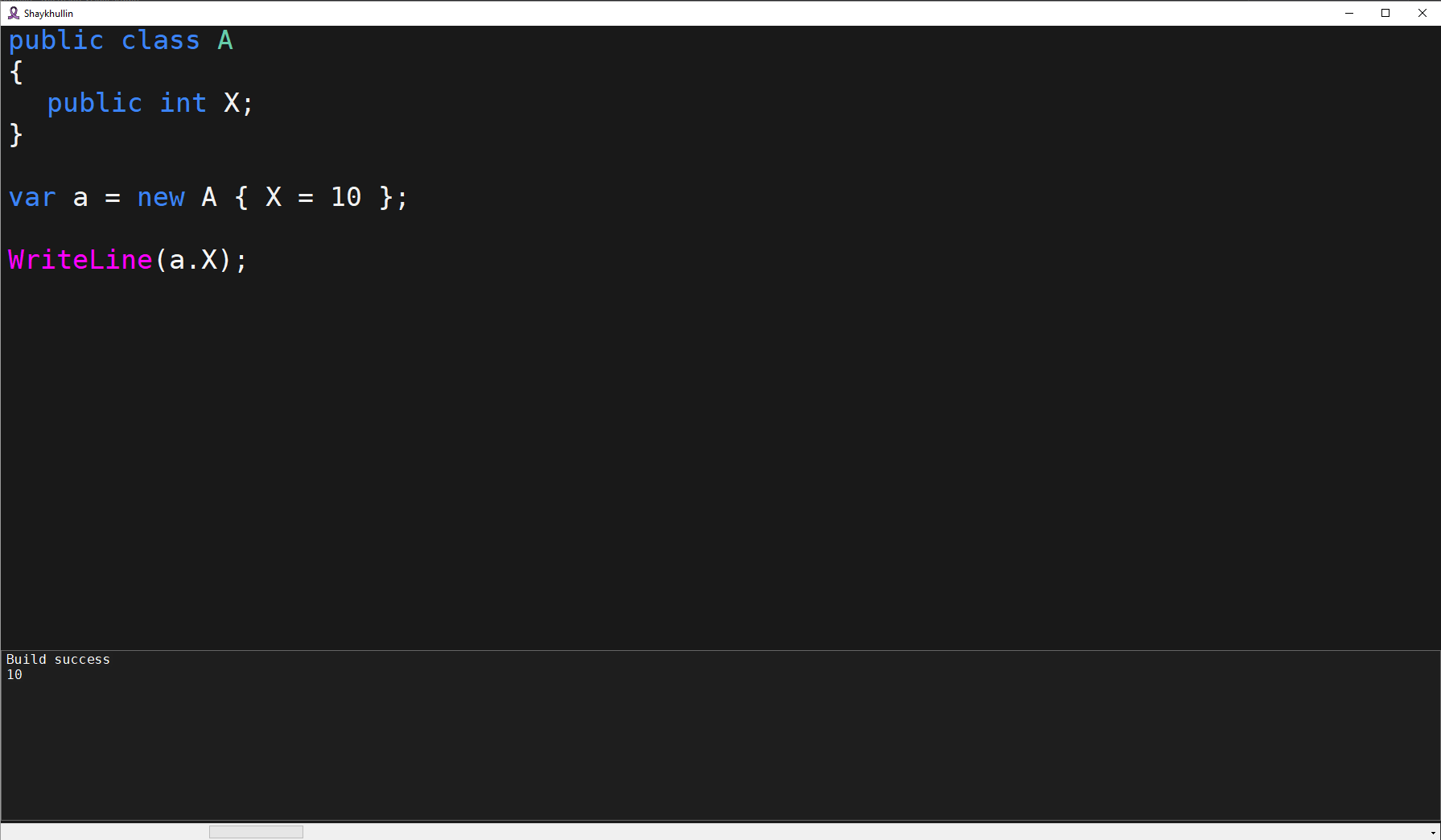
3) Реализовать динамическую загрузку другого проекта через рефлексию

**Заключение**

В данной лабораторной работе мы познакомились с реализацией динамически подгружаемых сборок.

**Пример работы**





**Исходный код**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace GeneratedNamespace

{

public static class GeneratedClass

{

private static Action<object> WriteLine;

public static void GeneratedMethod(Action<object> WriteLine)

{

GeneratedClass.WriteLine = WriteLine;

/\*METHODS\*/

}

/\*TYPES\*/

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class BuildCommand : Command

{

public override Keys Key => Keys.B;

public override bool RequireControl => true;

public override bool RequireShift => true;

public override void Apply(RichTextBox code)

{

MessageBox.Show("Building");

}

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public abstract class Command

{

public virtual bool RequireControl { get; }

public virtual bool RequireShift { get; }

public virtual bool RequireAlt { get; }

public abstract Keys Key { get; }

public bool IsSatisfied(Keys key, KeyState state)

{

return Key == key

&& RequireControl == state.ControlPressed

&& RequireShift == state.ShiftPressed

&& RequireAlt == state.AltPressed;

}

public abstract void Apply(RichTextBox code);

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class ControlEnterCommand : Command

{

public override Keys Key => Keys.Enter;

public override bool RequireControl => true;

public override void Apply(RichTextBox code)

{

var shiftAmount = 0;

for (var i = code.SelectionStart - 1; i > 0; i--)

{

if (code.Text[i] == '\n')

{

break;

}

shiftAmount++;

}

code.SelectionStart -= shiftAmount + 1;

}

}

}

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class ControlXCommand : Command

{

public override Keys Key => Keys.X;

public override bool RequireControl => true;

public override void Apply(RichTextBox code)

{

var lineNumber = code.GetLineFromCharIndex(code.SelectionStart);

var selection = code.SelectionStart;

code.Lines = ExceptIndex().ToArray();

code.SelectionStart = selection;

IEnumerable<string> ExceptIndex()

{

for (var i = 0; i < code.Lines.Length; i++)

{

if (i != lineNumber)

{

yield return code.Lines[i];

}

}

}

}

}

}

using System;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class EnterBracketCommand : Command

{

public override Keys Key => Keys.Enter;

public override void Apply(RichTextBox code)

{

var selected = code.SelectionStart;

if (code.Text.Length != selected

&& code.Text[selected - 1] == '{'

&& code.Text[selected] == '}')

{

Task.Run(() =>

{

CodeEditor.LockWindowUpdate(code.Handle);

Thread.Sleep(70);

code.Text = code.Text.Insert(selected + 1, "\t\n");

code.SelectionStart = selected + 2;

CodeEditor.LockWindowUpdate(IntPtr.Zero);

});

}

}

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class LeftCurlyBracketCommand : Command

{

public override Keys Key => Keys.OemOpenBrackets;

public override bool RequireShift => true;

public override void Apply(RichTextBox code)

{

var selected = code.SelectionStart;

code.Text = code.Text.Insert(code.SelectionStart, "}");

code.SelectionStart = selected;

}

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class LeftRoundBracketCommand : Command

{

public override Keys Key => Keys.D9;

public override bool RequireShift => true;

public override void Apply(RichTextBox code)

{

var selected = code.SelectionStart;

code.Text = code.Text.Insert(code.SelectionStart, ")");

code.SelectionStart = selected;

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Microsoft.CodeAnalysis;

using Shaykhullin.RoslynWrapper;

namespace Shaykhullin.Lab6.Commands

{

public class RunCommand : Command

{

public override Keys Key => Keys.F5;

public override bool RequireControl => true;

public override void Apply(RichTextBox code)

{

CodeEditor.OutputWindow.Text = "";

CodeEditor.OutputWindow.Focus();

CodeEditor.OutputWindow.Visible = true;

RunProgressBar();

var start = code.GetLineFromCharIndex(code.SelectionStart);

var end = code.GetLineFromCharIndex(code.SelectionStart + code.SelectionLength);

var executionCode = code.SelectionLength == 0

? code.Lines

: code.Lines.Skip(start).Take(end - start + 1).ToArray();

var assembly = new AssemblyComposer(

new CSharpSyntaxTreeCompiler(

new CSharpSyntaxTreeAnalyzer(GenerateTemplate(executionCode)),

new[]

{

typeof(object).Assembly,

typeof(IEnumerable<>).Assembly,

typeof(ArrayConverter).Assembly,

typeof(Constraint).Assembly,

typeof(Bitmap).Assembly,

typeof(Enumerable).Assembly,

typeof(StringBuilder).Assembly,

typeof(Task).Assembly,

typeof(Form).Assembly

},

OptimizationLevel.Debug))

.ComposeAssembly();

if (assembly.Success)

{

EvaluateMethod(assembly.Assembly);

}

else

{

EvaluateErrors(assembly.Errors);

}

}

private void EvaluateMethod(Assembly assembly)

{

CodeEditor.OutputWindow.AppendText("Build success\n");

var type = assembly.GetType("GeneratedNamespace.GeneratedClass");

var method = type.GetMethod("GeneratedMethod", BindingFlags.Public | BindingFlags.Static);

try

{

method.Invoke(null,

new object[]

{

(Action<object>)(obj => CodeEditor.OutputWindow.AppendText(obj.ToString() + "\n"))

});

}

catch (Exception e)

{

CodeEditor.OutputWindow.AppendText("Exception: " + e.InnerException.Message + "\n");

}

}

private void EvaluateErrors(IEnumerable<Diagnostic> errors)

{

CodeEditor.OutputWindow.AppendText("Build failed\n");

foreach (var error in errors)

{

var editor = CodeEditor.Editor;

var selected = editor.SelectionStart;

var normalizedStart = 0; //error.Location.SourceSpan.Start;

editor.Select(normalizedStart, error.Location.SourceSpan.Length + 1);

editor.SelectionFont = new Font(editor.SelectionFont, FontStyle.Underline);

editor.Select(normalizedStart, error.Location.SourceSpan.Length + 1);

editor.SelectionColor = Color.Red;

editor.SelectionStart = selected;

editor.SelectionFont = new Font(editor.SelectionFont, FontStyle.Regular);

CodeEditor.OutputWindow.AppendText(error + "\n");

}

}

private string GenerateTemplate(string[] codeLines)

{

var template = File.ReadAllText(

Path.Combine(Environment.CurrentDirectory, "template.cs"));

var types = DetectTypes().ToList();

var methods = codeLines.Except(types);

template = template.Replace("/\*METHODS\*/", string.Join("\n", methods));

template = template.Replace("/\*TYPES\*/", string.Join("\n", types));

return template;

IEnumerable<string> DetectTypes()

{

var typeFound = false;

var paranthesis = 0;

for (var i = 0; i < codeLines.Length; i++)

{

var line = codeLines[i];

if (line.Contains("delegate")

&& !line.Contains("{")

&& i + 1 < codeLines.Length

&& !codeLines[i + 1].Contains("{"))

{

yield return line;

}

if (HasType(line))

{

typeFound = true;

}

if (typeFound)

{

var lineDifference = line.Where(x => x == '{').Count()

- line.Where(x => x == '}').Count();

paranthesis += lineDifference;

yield return line;

if (!HasType(line) && paranthesis == 0)

{

typeFound = false;

}

}

}

bool HasType(string line)

{

return line.Contains("class")

|| line.Contains("interface")

|| line.Contains("enum")

|| line.Contains("struct");

}

}

}

private void RunProgressBar()

{

Task.Run(() =>

{

for (var i = 0; i < 10; i++)

{

CodeEditor.ProgressBar.PerformStep();

Thread.Sleep(50);

}

CodeEditor.ProgressBar.Value = 0;

});

}

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Commands

{

public class ShiftEnterCommand : Command

{

public override Keys Key => Keys.Enter;

public override bool RequireShift => true;

public override void Apply(RichTextBox code)

{

var shiftAmount = 0;

for (var i = code.SelectionStart; i < code.Text.Length; i++)

{

if (code.Text[i] == '\n')

{

break;

}

shiftAmount++;

}

code.SelectionStart += shiftAmount;

}

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6

{

public class KeyState

{

public bool ControlPressed { get; private set; }

public bool ShiftPressed { get; private set; }

public bool AltPressed { get; private set; }

public void Update(KeyEventArgs keyArgs)

{

ControlPressed = keyArgs.Control;

AltPressed = keyArgs.Alt;

ShiftPressed = keyArgs.Shift;

}

}

}

using System.Windows.Forms;

namespace Shaykhullin.Lab6.Core

{

public class ModelView<TViewModel> : Form

{

protected TViewModel model;

}

}

using System.Drawing;

namespace Shaykhullin.Lab6

{

public abstract class FunctionHighlighter : Highlighter

{

public override Color Color => Color.Fuchsia;

}

public class WriteLineHighlighter : FunctionHighlighter

{

public override string Keyword => "WriteLine";

}

}

namespace Shaykhullin.Lab6

{

public class AbstractHighlighter : KeywordHighlighter

{

public override string Keyword => "abstract";

}

}

namespace Shaykhullin.Lab6

{

public class AddHighlighter : KeywordHighlighter

{

public override string Keyword => "add";

}

}

namespace Shaykhullin.Lab6

{

public class AsHighlighter : KeywordHighlighter

{

public override string Keyword => "as";

}

}

namespace Shaykhullin.Lab6

{

public class BaseHighlighter : KeywordHighlighter

{

public override string Keyword => "base";

}

}

namespace Shaykhullin.Lab6

{

public class BoolHighlighter : KeywordHighlighter

{

public override string Keyword => "bool";

}

}

namespace Shaykhullin.Lab6

{

public class BreakHighlighter : KeywordHighlighter

{

public override string Keyword => "break";

}

}

namespace Shaykhullin.Lab6

{

public class ByteHighlighter : KeywordHighlighter

{

public override string Keyword => "byte";

}

}

namespace Shaykhullin.Lab6

{

public class CaseHighlighter : KeywordHighlighter

{

public override string Keyword => "case";

}

}

namespace Shaykhullin.Lab6

{

public class CatchHighlighter : KeywordHighlighter

{

public override string Keyword => "catch";

}

}

namespace Shaykhullin.Lab6

{

public class CharHighlighter : KeywordHighlighter

{

public override string Keyword => "char";

}

}

namespace Shaykhullin.Lab6

{

public class CheckedHighlighter : KeywordHighlighter

{

public override string Keyword => "checked";

}

}

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace Shaykhullin.Lab6

{

public class ClassHighlighter : KeywordHighlighter

{

public override string Keyword => "class ";

public override void Apply(RichTextBox editor)

{

base.Apply(editor);

var count = editor.Text.Substring(editor.SelectionStart)

.TakeWhile(x => x != ' ' && x != ':' && x != '\n' && x != '{' && x != '<')

.Count();

editor.SelectionLength = count;

editor.SelectionColor = Color.MediumAquamarine;

editor.SelectionStart += count;

editor.SelectionLength = 0;

}

}

}

namespace Shaykhullin.Lab6

{

public class ConstHighlighter : KeywordHighlighter

{

public override string Keyword => "const";

}

}

namespace Shaykhullin.Lab6

{

public class ContinueHighlighter : KeywordHighlighter

{

public override string Keyword => "continue";

}

}

namespace Shaykhullin.Lab6

{

public class DecimalHighlighter : KeywordHighlighter

{

public override string Keyword => "decimal";

}

}

namespace Shaykhullin.Lab6

{

public class DefaultHighlighter : KeywordHighlighter

{

public override string Keyword => "default";

}

}

namespace Shaykhullin.Lab6

{

public class DelegateHighlighter : KeywordHighlighter

{

public override string Keyword => "delegate";

}

}

namespace Shaykhullin.Lab6

{

public class DoHighlighter : KeywordHighlighter

{

public override string Keyword => "do";

}

}

namespace Shaykhullin.Lab6

{

public class DoubleHighlighter : KeywordHighlighter

{

public override string Keyword => "double";

}

}

namespace Shaykhullin.Lab6

{

public class ElseHighlighter : KeywordHighlighter

{

public override string Keyword => "else";

}

}

namespace Shaykhullin.Lab6

{

public class EnumHighlighter : KeywordHighlighter

{

public override string Keyword => "enum";

}

}

namespace Shaykhullin.Lab6

{

public class EventHighlighter : KeywordHighlighter

{

public override string Keyword => "event";

}

}

namespace Shaykhullin.Lab6

{

public class ExplicitHighlighter : KeywordHighlighter

{

public override string Keyword => "explicit";

}

}

namespace Shaykhullin.Lab6

{

public class ExternHighlighter : KeywordHighlighter

{

public override string Keyword => "extern";

}

}

namespace Shaykhullin.Lab6

{

public class FalseHighlighter : KeywordHighlighter

{

public override string Keyword => "false";

}

}

namespace Shaykhullin.Lab6

{

public class FinallyHighlighter : KeywordHighlighter

{

public override string Keyword => "finally";

}

}

namespace Shaykhullin.Lab6

{

public class FixedHighlighter : KeywordHighlighter

{

public override string Keyword => "fixed";

}

}

namespace Shaykhullin.Lab6

{

public class FloatHighlighter : KeywordHighlighter

{

public override string Keyword => "float";

}

}

namespace Shaykhullin.Lab6

{

public class ForeachHighlighter : KeywordHighlighter

{

public override string Keyword => "foreach";

}

}

namespace Shaykhullin.Lab6

{

public class ForHighlighter : KeywordHighlighter

{

public override string Keyword => "for";

}

}

namespace Shaykhullin.Lab6

{

public class GetHighlighter : KeywordHighlighter

{

public override string Keyword => "get";

}

}

namespace Shaykhullin.Lab6

{

public class GlobalHighlighter : KeywordHighlighter

{

public override string Keyword => "global";

}

}

namespace Shaykhullin.Lab6

{

public class GotoHighlighter : KeywordHighlighter

{

public override string Keyword => "goto";

}

}

namespace Shaykhullin.Lab6

{

public class IfHighlighter : KeywordHighlighter

{

public override string Keyword => "if";

}

}

namespace Shaykhullin.Lab6

{

public class ImplicitHighlighter : KeywordHighlighter

{

public override string Keyword => "implicit";

}

}

namespace Shaykhullin.Lab6

{

public class InHighlighter : KeywordHighlighter

{

public override string Keyword => "in ";

}

}

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace Shaykhullin.Lab6

{

public class InterfaceHighlighter : KeywordHighlighter

{

public override string Keyword => "interface";

public override void Apply(RichTextBox editor)

{

base.Apply(editor);

var count = editor.Text.Substring(editor.SelectionStart)

.TakeWhile(x => x != ' ' && x != ':' && x != '\n' && x != '{' && x != '<')

.Count();

editor.SelectionLength = count;

editor.SelectionColor = Color.FromArgb(245, 255, 158);

editor.SelectionStart += count;

editor.SelectionLength = 0;

}

}

}

namespace Shaykhullin.Lab6

{

public class InternalHighlighter : KeywordHighlighter

{

public override string Keyword => "internal";

}

}

namespace Shaykhullin.Lab6

{

public class IntHighlighter : KeywordHighlighter

{

public override string Keyword => "int";

}

}

namespace Shaykhullin.Lab6

{

public class IsHighlighter : KeywordHighlighter

{

public override string Keyword => "is";

}

}

using System.Drawing;

namespace Shaykhullin.Lab6

{

public abstract class KeywordHighlighter : Highlighter

{

public sealed override Color Color { get; } = Color.FromArgb(59, 131, 247);

}

}

namespace Shaykhullin.Lab6

{

public class LockHighlighter : KeywordHighlighter

{

public override string Keyword => "lock";

}

}

namespace Shaykhullin.Lab6

{

public class LongHighlighter : KeywordHighlighter

{

public override string Keyword => "long";

}

}

namespace Shaykhullin.Lab6

{

public class NamespaceHighlighter : KeywordHighlighter

{

public override string Keyword => "namespace";

}

}

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace Shaykhullin.Lab6

{

public class NewHighlighter : KeywordHighlighter

{

public override string Keyword => "new";

public override void Apply(RichTextBox editor)

{

base.Apply(editor);

var count = editor.Text.Substring(editor.SelectionStart)

.TakeWhile(x => x != ' ' && x != '[' && x != '\n' && x != '{' && x != '(')

.Count();

editor.SelectionLength = count;

editor.SelectionColor = Color.MediumAquamarine;

editor.SelectionStart += count;

editor.SelectionLength = 0;

}

}

}

namespace Shaykhullin.Lab6

{

public class NullHighlighter : KeywordHighlighter

{

public override string Keyword => "null";

}

}

namespace Shaykhullin.Lab6

{

public class ObjectHighlighter : KeywordHighlighter

{

public override string Keyword => "object";

}

}

namespace Shaykhullin.Lab6

{

public class OperatorHighlighter : KeywordHighlighter

{

public override string Keyword => "operator";

}

}

namespace Shaykhullin.Lab6

{

public class OutHighlighter : KeywordHighlighter

{

public override string Keyword => "out";

}

}

namespace Shaykhullin.Lab6

{

public class OverrideHighlighter : KeywordHighlighter

{

public override string Keyword => "override";

}

}

namespace Shaykhullin.Lab6

{

public class ParamsHighlighter : KeywordHighlighter

{

public override string Keyword => "params ";

}

}

namespace Shaykhullin.Lab6

{

public class PartialHighlighter : KeywordHighlighter

{

public override string Keyword => "partial";

}

}

namespace Shaykhullin.Lab6

{

public class PrivateHighlighter : KeywordHighlighter

{

public override string Keyword => "private";

}

}

namespace Shaykhullin.Lab6

{

public class ProtectedHighlighter : KeywordHighlighter

{

public override string Keyword => "protected";

}

}

namespace Shaykhullin.Lab6

{

public class PublicHighlighter : KeywordHighlighter

{

public override string Keyword => "public";

}

}

namespace Shaykhullin.Lab6

{

public class ReadonlyHighlighter : KeywordHighlighter

{

public override string Keyword => "readonly";

}

}

namespace Shaykhullin.Lab6

{

public class RefHighlighter : KeywordHighlighter

{

public override string Keyword => "ref";

}

}

namespace Shaykhullin.Lab6

{

public class RemoveHighlighter : KeywordHighlighter

{

public override string Keyword => "remove";

}

}

namespace Shaykhullin.Lab6

{

public class ReturnHighlighter : KeywordHighlighter

{

public override string Keyword => "return";

}

}

namespace Shaykhullin.Lab6

{

public class SbyteHighlighter : KeywordHighlighter

{

public override string Keyword => "sbyte";

}

}

namespace Shaykhullin.Lab6

{

public class SealedHighlighter : KeywordHighlighter

{

public override string Keyword => "sealed";

}

}

namespace Shaykhullin.Lab6

{

public class SetHighlighter : KeywordHighlighter

{

public override string Keyword => "set";

}

}

namespace Shaykhullin.Lab6

{

public class ShortHighlighter : KeywordHighlighter

{

public override string Keyword => "short";

}

}

namespace Shaykhullin.Lab6

{

public class SizeofHighlighter : KeywordHighlighter

{

public override string Keyword => "sizeof";

}

}

namespace Shaykhullin.Lab6

{

public class StackallocHighlighter : KeywordHighlighter

{

public override string Keyword => "stackalloc";

}

}

namespace Shaykhullin.Lab6

{

public class StaticHighlighter : KeywordHighlighter

{

public override string Keyword => "static";

}

}

namespace Shaykhullin.Lab6

{

public class StringHighlighter : KeywordHighlighter

{

public override string Keyword => "string";

}

}

namespace Shaykhullin.Lab6

{

public class StructHighlighter : KeywordHighlighter

{

public override string Keyword => "struct";

}

}

namespace Shaykhullin.Lab6

{

public class SwitchHighlighter : KeywordHighlighter

{

public override string Keyword => "switch";

}

}

namespace Shaykhullin.Lab6

{

public class ThisHighlighter : KeywordHighlighter

{

public override string Keyword => "this";

}

}

namespace Shaykhullin.Lab6

{

public class ThrowHighlighter : KeywordHighlighter

{

public override string Keyword => "throw";

}

}

namespace Shaykhullin.Lab6

{

public class TrueHighlighter : KeywordHighlighter

{

public override string Keyword => "true";

}

}

namespace Shaykhullin.Lab6

{

public class TryHighlighter : KeywordHighlighter

{

public override string Keyword => "try";

}

}

namespace Shaykhullin.Lab6

{

public class TypeofHighlighter : KeywordHighlighter

{

public override string Keyword => "typeof";

}

}

namespace Shaykhullin.Lab6

{

public class UintHighlighter : KeywordHighlighter

{

public override string Keyword => "uint";

}

}

namespace Shaykhullin.Lab6

{

public class UlongHighlighter : KeywordHighlighter

{

public override string Keyword => "ulong";

}

}

namespace Shaykhullin.Lab6

{

public class UncheckedHighlighter : KeywordHighlighter

{

public override string Keyword => "unchecked";

}

}

namespace Shaykhullin.Lab6

{

public class UnsafeHighlighter : KeywordHighlighter

{

public override string Keyword => "unsafe";

}

}

namespace Shaykhullin.Lab6

{

public class UshortHighlighter : KeywordHighlighter

{

public override string Keyword => "ushort";

}

}

namespace Shaykhullin.Lab6

{

public class UsingHighlighter : KeywordHighlighter

{

public override string Keyword => "using";

}

}

namespace Shaykhullin.Lab6

{

public class ValueHighlighter : KeywordHighlighter

{

public override string Keyword => "value";

}

}

namespace Shaykhullin.Lab6

{

public class VarHighlighter : KeywordHighlighter

{

public override string Keyword => "var";

}

}

namespace Shaykhullin.Lab6

{

public class VirtualHighlighter : KeywordHighlighter

{

public override string Keyword => "virtual";

}

}

namespace Shaykhullin.Lab6

{

public class VoidHighlighter : KeywordHighlighter

{

public override string Keyword => "void";

}

}

namespace Shaykhullin.Lab6

{

public class VolatileHighlighter : KeywordHighlighter

{

public override string Keyword => "volatile";

}

}

namespace Shaykhullin.Lab6

{

public class WhereHighlighter : KeywordHighlighter

{

public override string Keyword => "where";

}

}

namespace Shaykhullin.Lab6

{

public class WhileHighlighter : KeywordHighlighter

{

public override string Keyword => "while";

}

}

namespace Shaykhullin.Lab6

{

public class YieldHighlighter : KeywordHighlighter

{

public override string Keyword => "yield";

}

}

using System.Drawing;

using System.Windows.Forms;

namespace Shaykhullin.Lab6

{

public abstract class Highlighter

{

public abstract string Keyword { get; }

public abstract Color Color { get; }

public bool IsSatisfied(string keyword, int index)

{

for (var i = 0; i < Keyword.Length; i++)

{

if (index + i >= keyword.Length || keyword[index + i] != Keyword[i])

{

return false;

}

}

return true;

}

public virtual void Apply(RichTextBox editor)

{

editor.SelectionLength = Keyword.Length;

editor.SelectionColor = Color;

editor.SelectionStart += Keyword.Length;

editor.SelectionLength = 0;

}

}

}

using System.Reflection;

using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following

// set of attributes. Change these attribute values to modify the information

// associated with an assembly.

[assembly: AssemblyTitle("Shaykhullin.Lab6")]

[assembly: AssemblyDescription("")]

[assembly: AssemblyConfiguration("")]

[assembly: AssemblyCompany("")]

[assembly: AssemblyProduct("Shaykhullin.Lab6")]

[assembly: AssemblyCopyright("Copyright © 2017")]

[assembly: AssemblyTrademark("")]

[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible

// to COM components. If you need to access a type in this assembly from

// COM, set the ComVisible attribute to true on that type.

[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM

[assembly: Guid("74552892-0f97-4245-9359-94ef1e60b943")]

// Version information for an assembly consists of the following four values:

//

// Major Version

// Minor Version

// Build Number

// Revision

//

// You can specify all the values or you can default the Build and Revision Numbers

// by using the '\*' as shown below:

// [assembly: AssemblyVersion("1.0.\*")]

[assembly: AssemblyVersion("1.0.0.0")]

[assembly: AssemblyFileVersion("1.0.0.0")]

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated by a tool.

// Runtime Version:4.0.30319.42000

//

// Changes to this file may cause incorrect behavior and will be lost if

// the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

namespace Shaykhullin.Lab6.Properties

{

/// <summary>

/// A strongly-typed resource class, for looking up localized strings, etc.

/// </summary>

// This class was auto-generated by the StronglyTypedResourceBuilder

// class via a tool like ResGen or Visual Studio.

// To add or remove a member, edit your .ResX file then rerun ResGen

// with the /str option, or rebuild your VS project.

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedResourceBuilder", "4.0.0.0")]

[global::System.Diagnostics.DebuggerNonUserCodeAttribute()]

[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

internal class Resources

{

private static global::System.Resources.ResourceManager resourceMan;

private static global::System.Globalization.CultureInfo resourceCulture;

[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Performance", "CA1811:AvoidUncalledPrivateCode")]

internal Resources()

{

}

/// <summary>

/// Returns the cached ResourceManager instance used by this class.

/// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]

internal static global::System.Resources.ResourceManager ResourceManager

{

get

{

if ((resourceMan == null))

{

global::System.Resources.ResourceManager temp = new global::System.Resources.ResourceManager("Shaykhullin.Lab6.Properties.Resources", typeof(Resources).Assembly);

resourceMan = temp;

}

return resourceMan;

}

}

/// <summary>

/// Overrides the current thread's CurrentUICulture property for all

/// resource lookups using this strongly typed resource class.

/// </summary>

[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]

internal static global::System.Globalization.CultureInfo Culture

{

get

{

return resourceCulture;

}

set

{

resourceCulture = value;

}

}

}

}

//------------------------------------------------------------------------------

// <auto-generated>

// This code was generated by a tool.

// Runtime Version:4.0.30319.42000

//

// Changes to this file may cause incorrect behavior and will be lost if

// the code is regenerated.

// </auto-generated>

//------------------------------------------------------------------------------

namespace Shaykhullin.Lab6.Properties

{

[global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]

[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.SettingsDesigner.SettingsSingleFileGenerator", "11.0.0.0")]

internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase

{

private static Settings defaultInstance = ((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new Settings())));

public static Settings Default

{

get

{

return defaultInstance;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using Shaykhullin.Lab6.Commands;

namespace Shaykhullin.Lab6.ViewModels

{

public class CodeEditorViewModel

{

private readonly IList<Command> commands;

private readonly IList<Highlighter> highlighters;

private readonly KeyState state;

public CodeEditorViewModel()

{

commands = typeof(Command).Assembly.GetTypes()

.Where(type => !type.IsAbstract)

.Where(type => typeof(Command).IsAssignableFrom(type))

.Select(type => (Command)Activator.CreateInstance(type))

.ToList();

highlighters = typeof(Highlighter).Assembly.GetTypes()

.Where(type => !type.IsAbstract)

.Where(type => typeof(Highlighter).IsAssignableFrom(type))

.Select(type => (Highlighter)Activator.CreateInstance(type))

.OrderByDescending(h => h.Keyword.Length)

.ToList();

state = new KeyState();

}

public void UpdateState(KeyEventArgs args)

{

state.Update(args);

}

public void TryExecuteCommand(RichTextBox editor, KeyEventArgs args)

{

for (var i = 0; i < commands.Count; i++)

{

if (commands[i].IsSatisfied(args.KeyCode, state))

{

commands[i].Apply(editor);

return;

}

}

}

public bool TryExecuteHighlighter(RichTextBox editor, string keyword, int index)

{

for (var i = 0; i < highlighters.Count; i++)

{

if (highlighters[i].IsSatisfied(keyword, index))

{

highlighters[i].Apply(editor);

return true;

}

}

return false;

}

public void ResetSelectionToStart(RichTextBox editor)

{

editor.SelectionStart = editor.SelectionLength = 0;

}

public void SetNextSelectionAndColor(RichTextBox editor)

{

editor.SelectionLength = 1;

editor.SelectionColor = Color.WhiteSmoke;

editor.SelectionStart++;

editor.SelectionLength = 0;

}

public void SetAllTextToRegularFont(RichTextBox editor)

{

editor.SelectAll();

editor.SelectionFont = new Font(editor.SelectionFont, FontStyle.Regular);

}

public void SetSelectedUnderscoreFont(RichTextBox editor)

{

}

}

}

using System;

using System.Runtime.InteropServices;

using System.Windows.Forms;

using Shaykhullin.Lab6.Core;

using Shaykhullin.Lab6.ViewModels;

namespace Shaykhullin.Lab6

{

public partial class CodeEditor : ModelView<CodeEditorViewModel>

{

private string prevText;

public CodeEditor()

{

InitializeComponent();

Editor = editor;

OutputWindow = outputWindow;

ProgressBar = toolStripProgressBar1;

model = new CodeEditorViewModel();

OnTextChanged(null, null);

}

public static TextBox OutputWindow { get; private set; }

public static RichTextBox Editor { get; private set; }

public static ToolStripProgressBar ProgressBar { get; private set; }

[DllImport("user32.dll")]

public static extern bool LockWindowUpdate(IntPtr hWndLock);

private void OnKeyDown(object sender, KeyEventArgs args)

{

model.UpdateState(args);

LockWindowUpdate(editor.Handle);

model.TryExecuteCommand(editor, args);

LockWindowUpdate(IntPtr.Zero);

}

private void OnTextChanged(object sender, EventArgs e)

{

if (prevText == editor.Text)

{

return;

}

prevText = editor.Text;

OnOutputDisable(sender, e);

var selection = editor.SelectionStart;

LockWindowUpdate(editor.Handle);

model.SetAllTextToRegularFont(editor);

model.ResetSelectionToStart(editor);

while (editor.SelectionStart < editor.Text.Length)

{

if (model.TryExecuteHighlighter(editor, editor.Text, editor.SelectionStart))

{

model.SetNextSelectionAndColor(editor);

}

else

{

var current = editor.Text[editor.SelectionStart];

while (char.IsLetterOrDigit(current) && editor.Text.Length - 1 != editor.SelectionStart)

{

current = editor.Text[++editor.SelectionStart];

}

editor.SelectionStart++;

}

}

editor.SelectionStart = selection;

LockWindowUpdate(IntPtr.Zero);

}

private void OnOutputDisable(object sender, EventArgs e)

{

outputWindow.Visible = false;

}

}

}

namespace Shaykhullin.Lab6

{

partial class CodeEditor

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

System.ComponentModel.ComponentResourceManager resources = new System.ComponentModel.ComponentResourceManager(typeof(CodeEditor));

this.editor = new System.Windows.Forms.RichTextBox();

this.statusStrip1 = new System.Windows.Forms.StatusStrip();

this.toolStripDropDownButton2 = new System.Windows.Forms.ToolStripDropDownButton();

this.outputWindow = new System.Windows.Forms.TextBox();

this.toolStripProgressBar1 = new System.Windows.Forms.ToolStripProgressBar();

this.statusStrip1.SuspendLayout();

this.SuspendLayout();

//

// editor

//

this.editor.AcceptsTab = true;

this.editor.Anchor = System.Windows.Forms.AnchorStyles.None;

this.editor.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(25)))), ((int)(((byte)(25)))), ((int)(((byte)(25)))));

this.editor.BorderStyle = System.Windows.Forms.BorderStyle.None;

this.editor.Font = new System.Drawing.Font("DejaVu Sans Mono", 25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

this.editor.ForeColor = System.Drawing.SystemColors.MenuBar;

this.editor.ImeMode = System.Windows.Forms.ImeMode.On;

this.editor.Location = new System.Drawing.Point(0, -2);

this.editor.Margin = new System.Windows.Forms.Padding(3, 3, 3, 0);

this.editor.MaximumSize = new System.Drawing.Size(2333, 2000);

this.editor.MinimumSize = new System.Drawing.Size(583, 500);

this.editor.Name = "editor";

this.editor.ScrollBars = System.Windows.Forms.RichTextBoxScrollBars.None;

this.editor.ShowSelectionMargin = true;

this.editor.Size = new System.Drawing.Size(1790, 993);

this.editor.TabIndex = 0;

this.editor.TabStop = false;

this.editor.Text = "int a = 10;\nint b = 10;\n\nWriteLine(a + b);";

this.editor.Click += new System.EventHandler(this.OnOutputDisable);

this.editor.TextChanged += new System.EventHandler(this.OnTextChanged);

this.editor.Enter += new System.EventHandler(this.OnOutputDisable);

this.editor.KeyDown += new System.Windows.Forms.KeyEventHandler(this.OnKeyDown);

//

// statusStrip1

//

this.statusStrip1.AllowMerge = false;

this.statusStrip1.AutoSize = false;

this.statusStrip1.Items.AddRange(new System.Windows.Forms.ToolStripItem[] {

this.toolStripDropDownButton2,

this.toolStripProgressBar1});

this.statusStrip1.Location = new System.Drawing.Point(0, 991);

this.statusStrip1.Name = "statusStrip1";

this.statusStrip1.Padding = new System.Windows.Forms.Padding(16, 0, 1, 0);

this.statusStrip1.RightToLeft = System.Windows.Forms.RightToLeft.Yes;

this.statusStrip1.Size = new System.Drawing.Size(1790, 22);

this.statusStrip1.SizingGrip = false;

this.statusStrip1.Stretch = false;

this.statusStrip1.TabIndex = 2;

this.statusStrip1.Text = "statusStrip1";

//

// toolStripDropDownButton2

//

this.toolStripDropDownButton2.DisplayStyle = System.Windows.Forms.ToolStripItemDisplayStyle.Image;

this.toolStripDropDownButton2.Name = "toolStripDropDownButton2";

this.toolStripDropDownButton2.Size = new System.Drawing.Size(13, 20);

this.toolStripDropDownButton2.Text = "toolStripDropDownButton2";

//

// outputWindow

//

this.outputWindow.Anchor = System.Windows.Forms.AnchorStyles.None;

this.outputWindow.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(30)))), ((int)(((byte)(30)))), ((int)(((byte)(30)))));

this.outputWindow.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;

this.outputWindow.Font = new System.Drawing.Font("DejaVu Sans Mono", 12F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

this.outputWindow.ForeColor = System.Drawing.SystemColors.Window;

this.outputWindow.Location = new System.Drawing.Point(0, 776);

this.outputWindow.Multiline = true;

this.outputWindow.Name = "outputWindow";

this.outputWindow.ReadOnly = true;

this.outputWindow.ShortcutsEnabled = false;

this.outputWindow.Size = new System.Drawing.Size(1790, 212);

this.outputWindow.TabIndex = 3;

this.outputWindow.TabStop = false;

this.outputWindow.Visible = false;

this.outputWindow.Leave += new System.EventHandler(this.OnOutputDisable);

//

// toolStripProgressBar1

//

this.toolStripProgressBar1.Alignment = System.Windows.Forms.ToolStripItemAlignment.Right;

this.toolStripProgressBar1.Margin = new System.Windows.Forms.Padding(1400, 3, 1, 3);

this.toolStripProgressBar1.Name = "toolStripProgressBar1";

this.toolStripProgressBar1.Size = new System.Drawing.Size(117, 16);

//

// CodeEditor

//

this.AutoScaleDimensions = new System.Drawing.SizeF(7F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.BackColor = System.Drawing.Color.FromArgb(((int)(((byte)(49)))), ((int)(((byte)(49)))), ((int)(((byte)(49)))));

this.ClientSize = new System.Drawing.Size(1790, 1013);

this.Controls.Add(this.outputWindow);

this.Controls.Add(this.statusStrip1);

this.Controls.Add(this.editor);

this.DoubleBuffered = true;

this.Font = new System.Drawing.Font("DejaVu Sans Mono", 8.25F, System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point, ((byte)(0)));

this.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedSingle;

this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));

this.Name = "CodeEditor";

this.Text = "Shaykhullin";

this.statusStrip1.ResumeLayout(false);

this.statusStrip1.PerformLayout();

this.ResumeLayout(false);

this.PerformLayout();

}

#endregion

private System.Windows.Forms.RichTextBox editor;

private System.Windows.Forms.StatusStrip statusStrip1;

private System.Windows.Forms.ToolStripDropDownButton toolStripDropDownButton2;

private System.Windows.Forms.TextBox outputWindow;

private System.Windows.Forms.ToolStripProgressBar toolStripProgressBar1;

}

}