|  |  |
| --- | --- |
|  |  |
| 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 №1

на тему

«Программные библиотеки»

по дисциплине

«Технологии программирования»

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

Принял Cтарший преподаватель Мирзоян Д.И.

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

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

Москва, 2017

**Теоретическая часть**

Для увеличения скорости и повышения эффективности процесса создания программного обеспечения широко используется повторное использование кода. Однако, для многократного использования одного и того же кода, необходимо его каким-то образом оформить, чтобы компилятор мог его использовать. Набор фрагментов кода, подготовленных к повторному использованию и называется библиотекой. Библиотеки бывают разных видов:

1. Библиотеки исходного кода, содержащие нескомпилированный исходный код на каком-либо языке программирования. Преимуществом данных библиотек является высокая открытость, т.е. программист легко может узнать, как работает та или иная функция, и легкость доработки и изменения — тот же программист, заметив ошибку, может легко ее устранить поправив соответствующим образом код библиотеки. Недостатками являются плохая защита от несанкционированных изменений и нелегального копирования, а также существенное для больших библиотек увеличения времени компиляции. Также такие библиотеки сравнительно легко можно перенести с одной аппаратной платформы на другую при наличии соответствующего компилятора, но они ограничены возможностью непосредственного использования только с одним языком программирования.
2. Библиотеки машинного кода, содержащие скомпилированный машинный код. Такие библиотеки можно подключить к множеству языков, имеющих соответствующие инструменты, но нельзя перенести на другую аппаратную платформу. Также они сильнее защищены от изменений, и, соответственно, правка ошибок в таких библиотеках в значительной степени затруднена.
3. Статические библиотеки подключаются в момент компиляции и/или компоновки программы (сборки), поэтому их код всегда присутствует в исполняемом файле приложения. С одной стороны, пропадает необходимость иметь эту библиотеку в системе, с другой — увеличивается размер исполняемого файла программы.
4. Динамические библиотеки подключаются уже во время работы программы. При этом есть динамические библиотеки со статическим связыванием и динамические библиотеки с динамическим связыванием. Первые загружаются и связываются с кодом программы операционной системой автоматически при запуске программы. При отсутствии библиотеки программа не может быть запущена. Вторые загружаются операционной системой по запросу приложения, связывание осуществляется самим приложением с использованием соответствующих системных функций. Логика поведения при отсутствии библиотеки определяется приложением — оно может как аварийно завершиться, так и продолжить работу.
5. Разделяемые, которые могут использоваться несколькими приложениями одновременно;
6. Неразделяемые, которые могут одновременно использоваться только одним приложением.

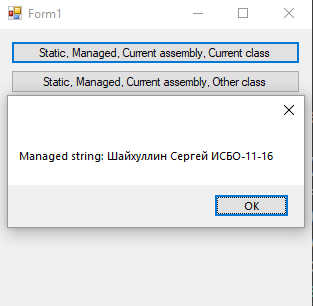
Статические библиотеки по факту могут быть только не разделяемыми — хотя при сборке двух разных приложений мог использоваться один и тот же файл статической библиотеки, при выполнении мы фактически имеем два несвязанных между собой одинаковых фрагмента кода.

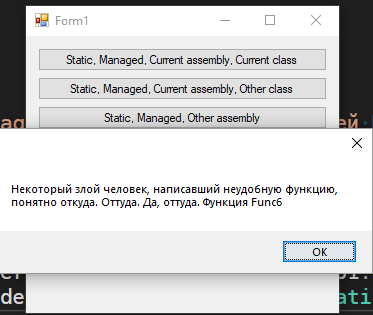
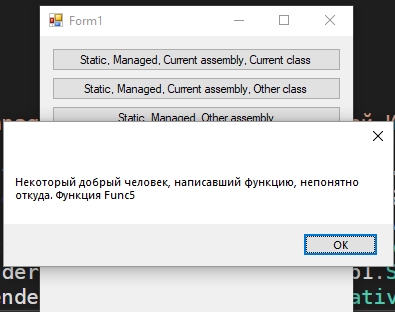
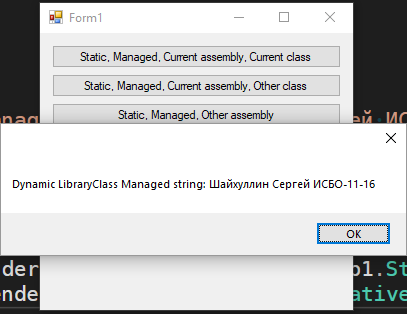
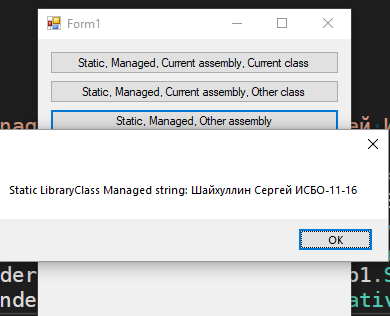
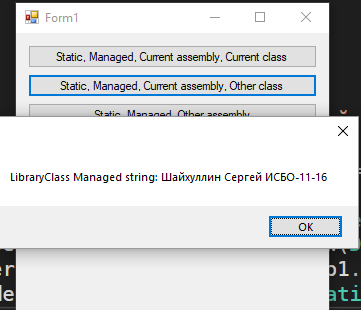
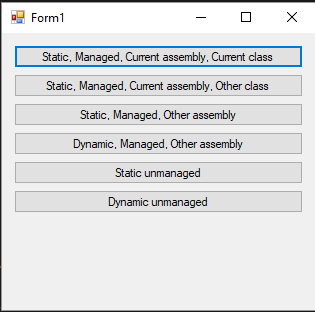
Библиотеки исходного кода могут быть динамическими в скриптовых языках (python, php, JavaScript) или в языках с поддержкой рефлексии.

**Задание**

Создать приложение, вызывающее сходные функции текстового вывода из различным образом подключенных программных библиотек. Продемонстрировать результаты подмены библиотек.

**Пример работы**





**Заключение**

В данной лабораторной работе мы научились подключать к программе различные сторонние библиотеки

**Исходный код**

using System;

using System.Windows.Forms;

namespace Lab1

{

public partial class MainWindow : Form

{

public MainWindow()

{

InitializeComponent();

}

public string GetManagedString() => "Managed string: Шайхуллин Сергей ИСБО-11-16";

private void StaticManagedCurrentCurrent(object sender, EventArgs e) => Show(this.GetManagedString());

private void StaticManagedCurrentOther(object sender, EventArgs e) => Show(new Library.LibraryClass().GetManagedString());

private void StaticManagedOther(object sender, EventArgs e) => Show(new ManagedLibrary.LibraryClass().GetManagedString());

private void DynamicManagedOther(object sender, EventArgs e) => Show(DynamicManaged.GetManagedString());

private void StaticUnmanaged(object sender, EventArgs e) => Show(Lab1.StaticUnmanaged.GetUnmanagedString());

private void DynamicUnmanaged(object sender, EventArgs e) => Show(Lab1.DynamicUnmanaged.GetUnmanagedString());

private void Show(string message) => MessageBox.Show(message);

}

}

using System;

using System.IO;

using System.Reflection;

namespace Lab1

{

static class DynamicManaged

{

public static string GetManagedString()

{

var asm = Assembly.LoadFrom(Path.Combine(Environment.CurrentDirectory, "Lab1.DynamicManagedLibrary.dll"));

var type = asm.GetType("Lab1.DynamicManagedLibrary.LibraryClass");

var m = type.GetMethod("GetManagedString", BindingFlags.Public | BindingFlags.Instance);

return (string)m.Invoke(Activator.CreateInstance(type), null);

}

}

}

namespace Lab1.Library

{

public class LibraryClass

{

public string GetManagedString() => "LibraryClass Managed string: Шайхуллин Сергей ИСБО-11-16";

}

}

using System;

using System.Runtime.InteropServices;

using System.Windows.Forms;

namespace Lab1

{

static class DynamicUnmanaged

{

[UnmanagedFunctionPointer(CallingConvention.StdCall)]

private delegate void Func6Delegate([MarshalAs(UnmanagedType.BStr)]out string data);

[DllImport("kernel32.dll")]

public static extern IntPtr LoadLibrary(string dllToLoad);

[DllImport("kernel32.dll")]

public static extern IntPtr GetProcAddress(IntPtr hModule, string procedureName);

[DllImport("kernel32.dll")]

public static extern Int32 GetLastError();

[DllImport("kernel32.dll")]

public static extern bool FreeLibrary(IntPtr hModule);

public static string GetUnmanagedString()

{

var dllPtr = LoadLibrary(Application.StartupPath + "\\unmlib2.dll");

var methodPtr = GetProcAddress(dllPtr, "Func6");

var unmanagedStringMethod = (Func6Delegate)Marshal.GetDelegateForFunctionPointer(methodPtr, typeof(Func6Delegate));

unmanagedStringMethod(out string str1);

FreeLibrary(dllPtr);

return str1;

}

}

}

using System.Runtime.InteropServices;

namespace Lab1

{

static class StaticUnmanaged

{

[DllImport("unmlib1.dll", CallingConvention = CallingConvention.StdCall)]

private static extern void Func5([MarshalAs(UnmanagedType.BStr)]out string data);

public static string GetUnmanagedString()

{

Func5(out string data);

return data;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.DynamicManagedLibrary

{

public class LibraryClass

{

public string GetManagedString()

{

return "Dynamic LibraryClass Managed string: Шайхуллин Сергей ИСБО-11-16";

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1.ManagedLibrary

{

public class LibraryClass

{

public string GetManagedString() => "Static LibraryClass Managed string: Шайхуллин Сергей ИСБО-11-16";

}

}