Технологии программирования.

Курсовая работа

Часть №1

Тема: Программные библиотеки

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

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.

a) 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.

b) 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.

I. Разделяемые, которые могут использоваться несколькими приложениями одновременно;

II. 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.

Для изучения различных вариантов подключения библиотек создадим 4 однотипные функции:

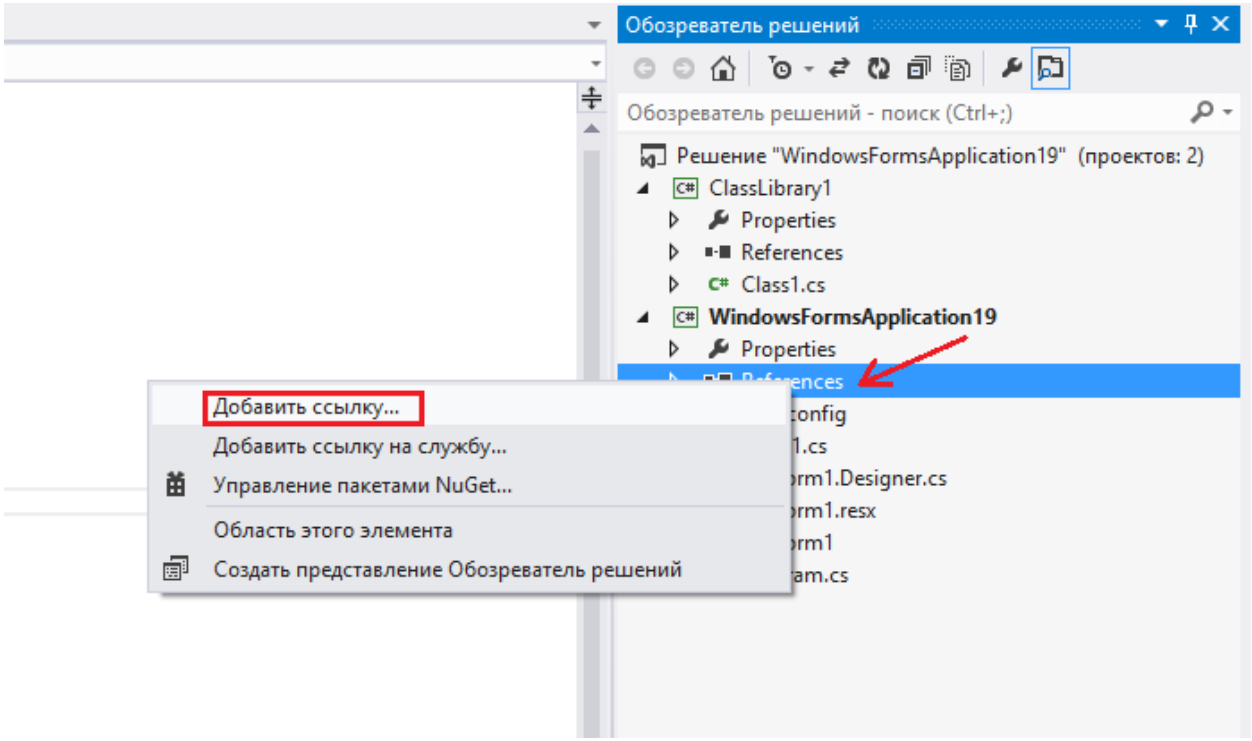
```
public string Func1()
{
    return "Василий Пупкин, ИСБО-13-66, функция Func1";
}
```

Одну поместим в тот же модуль приложения, в котором находится главная форма, другую — в другой модуль (другой класс в другом пространстве имен), который мы посчитаем статической библиотекой (в некоторых современных компиляторах, включая C#, отсутствуют явно выраженные статические библиотеки, они представлены только в виде «кэша» машинного кода во временных файлах *.o, *.obj, *.il), 3-я будет в статически подключенной динамической библиотеке, 4-я в динамически подключенной динамической библиотеке. Также мы подключим две неуправляемые библиотеки — одну статически, другую динамически. Соответственно, нам понадобится 3 проекта в одной папке (группе): основное приложение, содержащее код вызова 6 функций, библиотека со статическим связыванием с функцией 3, библиотека с динамическим связыванием с функцией 4.

Итого в приложении будет 6 кнопок, вызывающих соответствующие функции:

1. Функция Func1 — та же сборка, тот же модуль, то же пространство имен и класс.
2. Функция Func2 — та же сборка, другой модуль, другое пространство имен и класс («статическая библиотека»).
3. Функция Func3 — другая управляемая сборка, динамическая библиотека подключенная статически.
4. Функция Func4 — другая управляемая сборка, динамическая библиотека, загруженная и подключенная динамически.
5. Функция Func5 — другая неуправляемая сборка, динамическая библиотека подключенная статически.
6. Функция Func6 — другая неуправляемая сборка, динамическая библиотека, загруженная и подключенная динамически.

Для статического подключения функции в C#, в случае, если эта библиотека также является управляемой сборкой (т.е. предназначена для платформы .NET), достаточно добавить ссылку на нее в свойства проекта и добавить дикетиву using с необходимым пространством имен.



В случае, если библиотека является неуправляемой, потребуется ее импорт с помощью директивы DllImport и ключевого слова extern из пространства имен System.Runtime.InteropServices:

```
[DllImport("unmlib1.dll", CallingConvention=CallingConvention.StdCall)]
private static extern void Func5([MarshalAs(UnmanagedType.BStr)]out string data);
```

После статического подключения функцию (класс) можно использовать как обычно.

Динамическое связывание устроено значительно сложнее. Для этого необходимо загрузить библиотеку, запросить в ней искомые класс/метод/функцию и выгрузить библиотеку, когда в ней отпадет необходимость. Если библиотека является управляемой сборкой, то это легко можно сделать с помощью рефлексии (соответственно, нужно подключить пространство имен System.Reflection):

```
Assembly lib = Assembly.LoadFile(Application.StartupPath+"\\classlibrary1.dll");
Type myType = lib.GetType("ClassLibrary1.Class1");
dynamic obj = Activator.CreateInstance(myType);
MessageBox.Show(obj.Func4());
```

В случае неуправляемой библиотеки придется обратиться к системным функциям:

```
static class NativeMethods
{
    [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);
}
```

```
...

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

private void button2_Click(object sender, EventArgs e)
{
    IntPtr pDll =
NativeMethods.LoadLibrary(Application.StartupPath+"\\unmlib2.dll");
    IntPtr FuncAddr = NativeMethods.GetProcAddress(pDll, "Func6");

    Func6Delegate Func6 =
(FuncNDelegate)Marshal.GetDelegateForFunctionPointer(FuncAddr,
typeof(FuncNDelegate));

    String str1;
    Func6(out str1);
    MessageBox.Show(str1);
    NativeMethods.FreeLibrary(pDll);
}
```

Попробуйте менять библиотеки перед компиляцией, запуском, нажатием кнопок, взяв варианты ваших одногруппников. Объясните результат.