Технологии программирования
Курсовая работа
Часть №3
Тема: RTTI

Задание: Создать приложение, выводящее информацию RTTI для каждого узла дерева объектов.

Вывести:

1) Название класса
2) Название класса-предка
3) Область видимости класса
4) Название сборки, в которой класс определен
5) Публичный интерфейс класса:
   А) список полей с типами
   Б) список свойств с типами и указанием на наличие методов считывания и установки значения
   В) список методов с типами возвращемых значений, названиями и типами аргументов
6) Определяемый классом интерфейс:

   А) список полей с типами
   Б) список свойств с типами и указанием на наличие методов считывания и установки значения
   В) список методов с типами возвращемых значений, названиями и типами аргументов
   Для каждого элемента следует также указать спецификатор области видимости. Если метод виртуальный, необходимо это указать, а также указать класс, в котором находится реализация этого метода и класс, в котором он был впервые объявлен.

Дополните данную схему классов информацией о членах (полях, методах и свойствах) каждого класса. Если виртуальный элемент впервые объявлен в данном классе, выделите его зеленым, если виртуальный элемент переопределен — желтым. Невиртуальные элементы должны иметь черный цвет, переопределенные —красный.

```
public class RTTIRoot
{
    internal int A;
    public int PA;
    public int Prop1 { get; set; }
    internal void DoIt() { }
    public virtual void DoThat() {}
}

internal class RTTIClass1: RTTIRoot
{
    private int B;
    public string C;
    internal new void DoIt() { }
    public override void DoThat() { }
}

private class RTTIClass11 : RTTIClass1
{
    internal float D;
    private float PropD { get { return D; } }
    internal new void DoIt() { }
}

internal class RTTIClass12 : RTTIClass1
{
    public override void DoThat() { }
}

public class RTTIClass2 : RTTIRoot
{
    protected int Proto1;
    protected int PropProto { set { Proto1 = value; DoAnother();
} }
    private int DoAnother() { return 25; }
    public new virtual void DoThat() { }
}

protected class RTTIClass21 : RTTIClass2
{
    static int St1;
    public RTTIClass21() { St1 = 100500; }
    internal new void DoIt() { }
}

public class RTTIClass22 : RTTIClass2
{
    public override void DoThat() { }
}
```

Данные классы следует делать вложенными в класс формы, чтобы все поля были доступны.

Выполнение

RTTI (Run-Time Type Information) — информация о типах времени выполнения, специальный механизм, позволяющий учитывать фактический тип переменной в процессе исполнения программы. В случае обычных переменных ее тип определяется использованием данного участка памяти и определить характер фактически хранящихся в переменной данных в общем случае невозможно. При использовании RTTI программа в каждый момент времени обладает информацией о фактическом типе значения переменной.

RTTI используется для:

1) Поддержки динамической типизации в языках, в которых она есть: Basic, Java, Python.

2) Поддержки механизмов наследования и полиморфизма для обеспечения ООП

3) Обеспечения безопасности кода и вычислений (.NET)

Вообще, в платформе .NET RTTI (CTS) используется для всех трех целей, мы рассмотрим применение для 1й и 2й.

Реализуем объектное дерево, указанное в задании. Код описания классов приведен в тексте задания. Реализация всех методов может быть пустой (кроме возвращающих значение, в них можно подставить любую константу). При запуске приложения для работы необходимо создать по одному объекту каждого типа.

Для работы нам потребуется выводить достаточно большой объем информации, поэтому воспользуемся компонентом TextBox с установленным в true свойством Multiline для вывода информации и копирования ее в отчет.

На потребуется какой-то механизм получения информации, причем этот механизм не зависит от конкретного объекта — реализуем его в виде подпрограммы, принимающей параметр типа object и выводящей информацию в TextBox.

Однако откуда возьмется RTTI в объекте? Как мы знаем, в C# все объекты динамические, т.е. представляют собой указатели. Если мы посмотрим на содержимое объекта (например путем прямого чтения памяти), то увидим, что у каждого созданного объекта в начале есть несколько полей размером с указатель неизвестного нам назначения. Это и есть указатель на структуру RTTI.

Прямая работа с указателями и областями памяти в C# довольно затруднительна (хотя и возможна при определенных навыках программиста), поэтому мы воспользуемся средствами, которые нам предоставляет технология интроспекции/рефлексии, предназначенная как раз для работы с данными RTTI.
Основная чисть информации о типе содержится в структуре System.Type, получить которую можно вызвав функцию GetType для любого объекта:

```
void DumpObject(object obj)
{
    Type obj_type = obj.GetType();
    textBox1.AppendText("================================
"+obj_type.Name+" ================================\r\n");
```

```
    textBox1.AppendText("parent:
"+obj_type.BaseType.Name+"\r\n");
}
```

К сожалению, прямой доступ к VMT средствами интроспекции невозможен по соображениям безопасности. Поэтому мы запросим список полей, методов и свойств у объекта-описателя класса:

```
void DumpObject(object obj)
{
    Type obj_type = obj.GetType();
    textBox1.AppendText("================================
"+obj_type.Name+" ================================\r\n");

    textBox1.AppendText("parent:
"+obj_type.BaseType.Name+"\r\n");

    textBox1.AppendText("public interface:\r\n");
    textBox1.AppendText("fields:\r\n");

    FieldInfo[] fields = obj_type.GetFields();
    foreach (FieldInfo field in fields)
        textBox1.AppendText(field.FieldType.Name+"
"+field.Name+"\r\n");
    textBox1.AppendText("properties:\r\n");

    PropertyInfo[] properties = obj_type.GetProperties();
    foreach (PropertyInfo property in properties)
        textBox1.AppendText(property.PropertyType.Name+"
"+property.Name+" { "+(property.CanRead ? "get; " : "") +
(property.CanWrite ? "set; " : "") +"}\r\n");

    textBox1.AppendText("methods:\r\n");
    MethodInfo[] methods = obj_type.GetMethods();
    foreach (MethodInfo method in methods)
    {
        string m;
        m = method.ReturnType.Name + " " + method.Name+"(";
        foreach (ParameterInfo param in method.GetParameters())
            m+=(param.IsOut ? "out " : "")
+param.ParameterType.Name+" "+param.Name+", ";
        m += ")\r\n";
        textBox1.AppendText(m);
        if (method.IsVirtual)
            textBox1.AppendText("  defined in: " +
method.DeclaringType.Name+"\r\n");
    }
}
```

Остальную информацию получите, пользуясь свойствами и методами типа System.Type и пользуясь документацией MSDN.