Программирование “по-взрослому”

Немного о рефлексии. XML. Сериализация объектов. Создание редактора вопросов для игры “Верю-Не верю”

[Рефлексия](#_2frmvtfflj06)

[GetType, typeof](#_ylzbg2bb9i1y)

[Что такое XML?](#_iw9ekejdvn4m)

[Сериализация и десериализация](#_p5xt6242yhqw)

[Создаем Windows Forms приложение “Редактор вопросов для игры “Верю-Не верю”](#_lcgsk8qvvy7h)

[Классы для работы с данными](#_918t1fkqb33l)

[Приложение Windows Forms](#_5p1ahmxhmh2r)

[Дополнительные материалы](#_r18httkxppi0)

[Домашнее задание](#_3tfrjxxltv85)

[Используемая литература](#_pz2t80wjbpod)

# Рефлексия

Рефлексия представляет собой процесс выявления типов во время выполнения приложения. Каждое приложение содержит набор используемых классов, интерфейсов, а также их методов, свойств и прочих кирпичиков, из которых складывается приложение. И рефлексия как раз и позволяет определить все эти составные элементы приложения. Одними из самых простых способов получить информацию об объекте - это использование оператора typeof и метода GetType.

# GetType, typeof

Операцияtypeof**,** примененная к своему аргументу, возвращает его тип. В роли аргумента может выступать имя класса, как встроенного, так и созданного пользователем. Возвращаемый операцией результат имеет типType**.** К экземпляру класса применять операцию нельзя, но, зато, для экземпляра можно вызвать методGetType**,** наследуемый всеми классами, и получить тот же результат, что даетtypeof с именем данного класса. GetType и typeof на первый взгляд могут показаться бесполезными. Но с помощью них можно получить информацию о внутренней структуре класса, это еще называется рефлексией. Нам же typeof понадобиться для передачи информации о наших объектах в методы сериализации и десериализации XML данных.

|  |
| --- |
| using System;  class Program  {  static void Main()  {  // Используется для получения объекта System.Type для типа.  // Выражение typeof принимает следующую форму:  Type type = typeof(int);  Console.WriteLine(type);  // Для получения типа выражения во время выполнения можно  // воспользоваться методом платформы.NET GetType, как показано в следующем примере:  int i = 0;  Type type2 = i.GetType();  Console.WriteLine(type2);  }  } |

Еще пример использование рефлексии(“самопознания”):

|  |
| --- |
| using System;  using System.Reflection;  class Program  {  static PropertyInfo GetPropertyInfo(object obj,string str)  {  return obj.GetType().GetProperty(str);  }  static void Main(string[] args)  {  DateTime dateTime = new DateTime();  //dateTime.DayOfWeek  Console.WriteLine(GetPropertyInfo(dateTime, "DayOfWeek").CanRead);  Console.WriteLine(GetPropertyInfo(dateTime, "DayOfWeek").CanWrite);  Console.WriteLine(GetPropertyInfo(dateTime, "DayOfWeek").GetValue(dateTime, null));  Console.ReadKey();  }  } |

# Что такое XML?

XML очень похож на HTML. Но XML был создан для описания данных с прицелом на то, что представляют собой данные. HTML был создан для отображения данных с прицелом на то, как выглядят отображаемые данные.

* XML расшифровывается как Расширяемый язык разметки (EXtensible Markup Language);
* XML — это язык разметки, похожий на HTML;
* XML был создан для описания данных;
* Теги XML не предопределены. Вы можете использовать свои теги.

Пример HTML файла:

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <meta charset="UTF-8">  <title>Sample</title>  <link rel="stylesheet" href="style.css">  </head>  <body>  <div id="header">  <a href="#">Главная</a><span>/</span>  <a href="puzzles.html">Загадки</a><span>/</span>  <a href="quess.html">Угадайка</a><span>/</span>  <a href="03\_script\_mult.html">Угадайка мультиплеер</a><span>/</span>  </div> </body>  </html> |

Пример XML файла:

|  |
| --- |
| <?xml version="1.0"?>  <ArrayOfQuestion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  <Question>  <text>В Японии ученики на доске пишут кисточкой с цветными чернилами?</text>  <trueFalse>true</trueFalse>  </Question>  <Question>  <text>В Австралии практикуется применение одноразовых школьных досок? </text>  <trueFalse>false</trueFalse>  </Question>  <Question>  <text>Авторучка была изобретена еще в Древнем Египте?</text>  <trueFalse>true</trueFalse>  </Question>  </ArrayOfQuestion> |

# Сериализация и десериализация

Термин сериализация описывает процесс сохранения состояния объекта в потоке (например, файловом потоке). Последовательность сохраняемых данных содержит всю информацию, необходимую для реконструкции (или десериализации) состояния объекта с целью последующего использования. Применяя эту технологию, очень просто сохранять большие объемы данных с минимальными усилиями. Давайте рассмотрим этот процесс на примере сериализации созданого нами класса в формат XML файла. Класс, объекты которого подлежат сериализации и десериализации, снабжается атрибутом Serializable. Этот класс должен быть публичным, иначе методы, которые реализуют работу по сериализации и десериализации, не смогут получить к нему доступ.

|  |
| --- |
| [Serializable]  public class Student  {  // Чтобы поля можно было сериализовать, они должны быть открытыми  public string firstName;  public string lastName;  // Если поле не открыто, оно не будет сериализоваться  int age;  // Если мы не хотим нарушать принцип инкапсуляции,  // но хотим сериализовать поле, то должны реализовать  // доступ к нему через публичное свойство  public int Age  {  get { return age; }  set { age = value; }  }  // Для сериализации должен быть создан конструктор без параметров.  // Если конструктор по умолчанию не создан, он создается автоматически.  } |

Теперь класс для работы с сериализованным классом:

|  |
| --- |
| using System;  using System.IO;  using System.Xml.Serialization;  // Класс XmlSerializer требует, чтобы все сериализированные типы  // поддерживали конструктор по умолчанию (поэтому не забудьте его добавить, если определяли специальные конструкторы). Если этого не сделать, во время выполнения сгенерируется исключение InvalidOperationException.  namespace XMLSerializer  {  [Serializable]  public class Student  {  // Чтобы поля можно было сериализовать, они должны быть открытыми  public string firstName;  public string lastName;  // Если поле не открыто оно не будет сериализоваться  int age;  // Если мы хотим не нарушать принцип инкапсуляции, но хотим сериализовать поле, то должны реализовать доступ к нему через публичное свойство  public int Age  {  get { return age; }  set { if (value > 0) age = value; }  }  // Если конструктор по умолчанию не создан, он создается автоматически  }  class Program  {  static void SaveAsXmlFormat(Student obj, string fileName)  {  // Сохранить объект класса Student в файле fileName в формате XML  // typeof(Student) передает в XmlSerializer данные о классе.  // Внутри метода Serialize происходит довольна большая работа по постройке  // графа зависимостей для последующего создания xml файла.  // Процесс получения данных о структуре объекта называется рефлексией.  XmlSerializer xmlFormat = new XmlSerializer(typeof(Student));  // Создаем файловый поток(проще говоря, создаем файл)  Stream fStream = new FileStream(fileName, FileMode.Create, FileAccess.Write);  // В этот поток записываем сериализованные данные(записываем xml файл)  xmlFormat.Serialize(fStream, obj);  fStream.Close();  }  static Student LoadFromXmlFormat(string fileName)  {  Student obj=new Student();  // Считать объект Student из файла fileName формата XML  XmlSerializer xmlFormat = new XmlSerializer(typeof(Student));  Stream fStream = new FileStream(fileName,FileMode.Open, FileAccess.Read);  obj=(xmlFormat.Deserialize(fStream) as Student);  fStream.Close();  return obj;  }  static void Main(string[] args)  {  Student student = new Student();  student.Age = 20;  student.firstName = "Иван";  student.lastName = "Иванов";  SaveAsXmlFormat(student, "data.xml");  student=LoadFromXmlFormat("data.xml");  Console.WriteLine("{0} {1} {2}",student.firstName,student.lastName,student.Age);  Console.ReadKey();  }  }  } |

Надеюсь, как происходит сериализация одного объекта более-менее понятно. Сериализовать массив или коллекцию не намного сложнее.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Xml.Serialization;  namespace XMLSerializer\_List  {  [Serializable]  public class Student  {  // Чтобы поля можно было сериализовать, они должны быть открытыми  public string firstName;  public string lastName;  // Если поле не открыто, оно не будет сериализоваться  int age;  // Если мы хотим не нарушать принцип инкапсуляции, но хотим сериализовать поле, то должны реализовать доступ к нему через публичное свойство  public int Age  {  get { return age; }  set { if (value > 0) age = value; }  }  // Если есть отличный от конструктора по умолчанию конструктор, то пустой конструктор автоматически не создается...  public Student(string firstName, string lastName, int age)  {  this.firstName = firstName;  this.lastName = lastName;  this.age = age;  }  //...в этом случае для сериализации требуется самим создать пустой конструктор  public Student()  {  }  }  class Program  {  static void SaveAsXmlFormat(List<Student> obj, string fileName)  {  // Сериализовать список объектов не представляется большой проблемой  // Дело в том, что все объекты в C# наследуются от класса Object,  // который представляет собой дерево объектов  // подробней читайте Эндрю Троелсен Язык программирования C# 5.0  XmlSerializer xmlFormat = new XmlSerializer(typeof(List<Student>));  // Создаем файловый поток (проще говоря, создаем файл)  Stream fStream = new FileStream(fileName, FileMode.Create, FileAccess.Write);  // В этот поток записываем сериализованные данные (записываем xml файл)  xmlFormat.Serialize(fStream, obj);  fStream.Close();  }  static void LoadFromXmlFormat(ref List<Student> obj, string fileName)  {  // Считать класс List<Student> из файла fileName формата XML  // Обратите внимание, что этот пример показывает нам, что List<Student> не более, чем класс, его структура более сложная и для ее понимания потребуется некоторый опыт  XmlSerializer xmlFormat = new XmlSerializer(typeof(List<Student>));  Stream fStream = new FileStream(fileName, FileMode.Open, FileAccess.Read);  obj = (List<Student>)xmlFormat.Deserialize(fStream);  fStream.Close();  }  static void Main(string[] args)  {  List<Student> list = new List<Student>();  list.Add(new Student("Иван", "Иванов", 20));  list.Add(new Student("Петр", "Петров", 21));  SaveAsXmlFormat(list, "data.xml");  LoadFromXmlFormat(ref list, "data.xml");  foreach(var v in list)  {  Console.WriteLine("{0} {1} {2}",v.firstName, v.lastName, v.Age);  }  Console.ReadKey();  }  }  } |

Запустите приложение и найдите файлы xml, которые получились при сериализации объектов.

# Создаем Windows Forms приложение “Редактор вопросов для игры “Верю-Не верю”

В качестве демонстрации возможностей нашего класса создадим приложение, которое позволит продемонстрировать возможности Windows Forms, а также позволит создать игру “Верю-Не верю”.

Смысл игры довольно простой. Компьютер выдает нам информацию, а мы соглашаемся или не соглашаемся с этой информацией. Найдите в интернете вопросы для игры или придумайте сами. Это можно сделать позже.

Работа будет состоять из двух частей. В первой части нам нужно создать классы для работы с данными. Во второй создать Windows приложение, которое позволит пользователю создавать базу данных вопросов.

## Классы для работы с данными

Запустите Visual Studio. Создайте проект Windows Forms и назовите его BelieveOrNotBelieve. Для начала разработаем класс для работы с XML. Добавьте класс TrueFalse в проект со следующим содержимым:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Xml.Serialization;  namespace BelieveOrNotBelieve  {  // Класс для вопроса  [Serializable]  public class Question  {  public string text;//Текст вопроса  public bool trueFalse;//Правда или нет  // Здесь мы нарушаем правила инкапсуляции и эти поля нужно было бы реализовать через открытые свойства, но для упрощения примера оставим так  // Вам же предлагается сделать поля закрытыми и реализовать открытые свойства Text и TrueFalse  // Для сериализации должен быть пустой конструктор.  public Question()  {  }  public Question(string text, bool trueFalse)  {  this.text = text;  this.trueFalse = trueFalse;  }  }  // Класс для хранения списка вопросов. А так же для сериализации в XML и десериализации из XML  class TrueFalse  {  string fileName;  List<Question> list;  public string FileName  {  set { fileName = value; }  }  public TrueFalse(string fileName)  {  this.fileName = fileName;  list = new List<Question>();  }  public void Add(string text, bool trueFalse)  {  list.Add(new Question(text, trueFalse));  }  public void Remove(int index)  {  if (list != null && index<list.Count && index>=0) list.RemoveAt(index);  }  // Индексатор - свойство для доступа к закрытому объекту  public Question this[int index]  {  get { return list[index]; }  }  public void Save()  {  XmlSerializer xmlFormat = new XmlSerializer(typeof(List<Question>));  Stream fStream = new FileStream(fileName, FileMode.Create, FileAccess.Write);  xmlFormat.Serialize(fStream, list);  fStream.Close();  }  public void Load()  {  XmlSerializer xmlFormat = new XmlSerializer(typeof(List<Question>));  Stream fStream = new FileStream(fileName, FileMode.Open, FileAccess.Read);  list = (List<Question>)xmlFormat.Deserialize(fStream);  fStream.Close();  }  public int Count  {  get { return list.Count; }  }  }  } |

## 

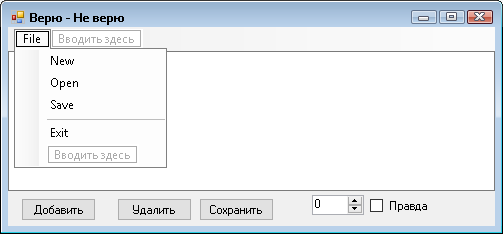
## Приложение Windows Forms

После того, как класс по работе с данными создан, выберите форму и добавьте на нее элементы как показано на рисунке.

Меню – ToolStrip(ToolStripMenuItem.Name = miNew, miOpen, miSave, miExit)

Номер вопроса – NumericUpDown(Name = nudNumber)“Правда” – CheckBox(Name= cboxTrue)

Кнопки – Button(Name= btnAdd, btnDelete, btnSaveQuest)



Белое поле, занимающее больше всего места, это элемент TextBox. Переименуйте элементы, и создайте обработчики событий. Опишите в классе Form1 объект database класса TrueFalse.

В итоге у вас должно получиться следующее:

|  |
| --- |
| using System;  using System.Windows.Forms;  namespace BelieveOrNotBelieve  {  public partial class Form1 : Form  {  // База данных с вопросами  TrueFalse database;  public Form1()  {  InitializeComponent();  }  // Обработчик пункта меню Exit  private void miExit\_Click(object sender, EventArgs e)  {  this.Close();  }  // Обработчик пункта меню New  private void miNew\_Click(object sender, EventArgs e)  {  SaveFileDialog sfd = new SaveFileDialog();  if (sfd.ShowDialog() == DialogResult.OK)  {  database = new TrueFalse(sfd.FileName);  database.Add("123", true);  database.Save();  nudNumber.Minimum = 1;  nudNumber.Maximum = 1;  nudNumber.Value = 1;  };  }  // Обработчик события изменения значения numericUpDown  private void nudNumber\_ValueChanged(object sender, EventArgs e)  {  tboxQuestion.Text = database[(int)nudNumber.Value - 1].text;  cboxTrue.Checked = database[(int)nudNumber.Value - 1].trueFalse;  }  // Обработчик кнопки Добавить  private void btnAdd\_Click(object sender, EventArgs e)  {  if (database==null)  {  MessageBox.Show("Создайте новую базу данных","Сообщение");  return;  }  database.Add((database.Count+1).ToString(), true);  nudNumber.Maximum = database.Count;  nudNumber.Value = database.Count;  }  // Обработчик кнопки Удалить  private void btnDelete\_Click(object sender, EventArgs e)  {  if (nudNumber.Maximum == 1 || database==null) return;  database.Remove((int)nudNumber.Value);  nudNumber.Maximum--;  if (nudNumber.Value>1) nudNumber.Value = nudNumber.Value;  }  // Обработчик пункта меню Save  private void miSave\_Click(object sender, EventArgs e)  {  if (database!= null) database.Save();  else MessageBox.Show("База данных не создана");  }  // Обработчик пункта меню Open  private void miOpen\_Click(object sender, EventArgs e)  {  OpenFileDialog ofd = new OpenFileDialog();  if (ofd.ShowDialog()==DialogResult.OK)  {  database = new TrueFalse(ofd.FileName);  database.Load();  nudNumber.Minimum = 1;  nudNumber.Maximum = database.Count;  nudNumber.Value = 1;  }  }  // Обработчик кнопки Сохранить (вопрос)  private void btnSaveQuest\_Click(object sender, EventArgs e)  {  database[(int)nudNumber.Value-1].text = tboxQuestion.Text;  database[(int)nudNumber.Value - 1].trueFalse = cboxTrue.Checked;  }  }  } |

Запустите приложение. Убедитесь, что кнопки работают. Проверьте, что вопросы добавляются, сохраняются и загружаются.

# Дополнительные материалы

1. [typeof (справочник по C#)](https://msdn.microsoft.com/ru-ru/library/58918ffs.aspx) - посмотреть, что такое рефлексия.

# Домашнее задание

1. а) Создайте приложение показанное на уроке;  
   б) изменить интерфейс программы, увеличив шрифт, поменяв цвет элементов, и добавив другие косметические улучшения на свое усмотрение;  
   в) Добавить в приложение меню “О программе” с информацией о программе (автор, версия, авторские права и др.);  
   г) Добавить в приложение сообщение с предупреждением при попытке удалить вопрос;  
   д) Добавить пункт меню Save As в котором можно выбрать имя для сохранения базы данных (элемент SaveFileDialog);
2. \*Используя полученные знания и класс TrueFalse, разработать игру “Верю-Не верю”
3. \*\*Написать программу-преобразователь из CSV в XML-файл с информацией о студентах (6 урок).

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Эндрю Троелсен. “Язык программирования C# 5.0 и платформа .NET 4.5”, 2013 г.
2. Петцольд Ч. “Программирование на C#. Т1”, 2001 г.
3. Петцольд Ч. “Программирование с использованием Windows Forms”, 2006 г.
4. Г.Шилдт. “C# 4.0. Полное руководство”.
5. [MSDN](https://msdn.microsoft.com/ru-ru/default.aspx)