Практическая работа № 1.  
Рефлексия типов (Reflection)

**Цель практической работы**

Закрепление теоретических знаний по применению рефлексии типов.

**Постановка задачи**

Рефлексия — это мощная возможность в .NET, позволяющая программам исследовать свои собственные структуры данных и метаданные во время выполнения. С помощью рефлексии можно получать информацию о типах, методах, свойствах и других членах классов, а также динамически создавать экземпляры объектов и вызывать методы.

**Основные понятия рефлексии**

1. **Типы**: В C# типы могут быть классами, структурами, интерфейсами и перечислениями. Рефлексия позволяет исследовать все эти типы.
2. **Метаданные**: Это информация о типах, которая хранится в сборках (assemblies) .NET. Метаданные включают информацию о свойствах, методах, атрибутах и других членах типа.
3. **Assembly**: Это сборка, которая содержит код, метаданные и ресурсы. С помощью рефлексии можно загружать сборки и получать информацию о них.

Рефлексия может использоваться для различных целей, таких как:

* Динамическое создание объектов
* Вызываем методов по имени
* Получение и установка значений свойств
* Создание инструментов для сериализации и десериализации данных
* Реализация механизмов фильтрации и поиска

**Пример использования рефлексии: Сериализация и десериализация объектов**

В этом примере мы создадим приложение для сериализации и десериализации объектов в формате JSON. Мы будем использовать рефлексию для динамического получения значений свойств объектов и их установки.

**Шаг 1: Определение класса Person**

Создадим класс Person, который будет представлять человека с несколькими свойствами.

public class Person

{

public string Name { get; set; }

public int Age { get; set; }

public string Email { get; set; }

public override string ToString()

{

return $"Name: {Name}, Age: {Age}, Email: {Email}";

}

}

**Шаг 2: Реализация метода сериализации**

Создадим метод Serialize, который будет принимать объект и возвращать его представление в формате JSON.

using System.Text;

public static class JsonSerializer

{

public static string Serialize(object obj)

{

var type = obj.GetType();

var properties = type.GetProperties();

var jsonBuilder = new StringBuilder();

jsonBuilder.AppendLine("{");

var i = 0;

foreach (var property in properties)

{

var value = property.GetValue(obj);

jsonBuilder.Append($"\"{property.Name}\": \"{value}\"");

if (i < properties.Length - 1)

jsonBuilder.AppendLine(",");

i++;

}

// Удаляем последнюю запятую

if (jsonBuilder.Length > 1)

jsonBuilder.Length--;

jsonBuilder.AppendLine();

jsonBuilder.AppendLine("}");

return jsonBuilder.ToString();

}

}

**Шаг 3: Реализация метода десериализации**

Создадим метод Deserialize, который будет принимать строку JSON и возвращать объект заданного типа.

public static class JsonDeserializer

{

public static object Deserialize(Type type, string json)

{

object? obj = Activator.CreateInstance(type);

var properties = type.GetProperties();

// Удаляем фигурные скобки и разбиваем по запятой

var pairs = json.Trim('{', '}').Split(',');

foreach (var pair in pairs)

{

var keyValue = pair.Split(':');

var propertyName = keyValue[0].Trim().Trim('"');

var propertyValue = keyValue[1].Trim().Trim('"');

var property = type.GetProperty(propertyName);

if (property != null)

{

// Преобразуем строковое значение в тип свойства

if (property.PropertyType == typeof(int))

{

property.SetValue(obj, int.Parse(propertyValue));

}

else

{

property.SetValue(obj, propertyValue);

}

}

}

return obj;

}

}

**Шаг 4: Тестирование сериализации и десериализации**

Теперь протестируем наши методы в Main.

class Program

{

static void Main(string[] args)

{

// Создаем объект Person

var person = new Person

{

Name = "Alice",

Age = 30,

Email = "alice@example.com"

};

// Сериализуем объект в JSON

string json = JsonSerializer.Serialize(person);

Console.WriteLine("Сериализованный JSON:");

Console.WriteLine(json);

// Десериализуем JSON обратно в объект

var deserializedPerson = JsonDeserializer.Deserialize(typeof(person), json);

Console.WriteLine("\nДесериализованный объект:");

Console.WriteLine(deserializedPerson);

}

}

**Запуск приложения**

Когда вы запустите приложение, вы увидите вывод, который показывает сериализованный JSON и десериализованный объект:

Сериализованный JSON:

{"Name": "Alice","Age": "30","Email": "alice@example.com"}

Десериализованный объект:

Name: Alice, Age: 30, Email: alice@example.com

**Пример создания плагина с использованием позднего связывания.**

1. Сначала создадим интерфейс для плагинов (проект №1):

// IPlugin.cs

public interface IPlugin

{

    void Execute();

}

1. Затем создадим класс плагина, который реализует этот интерфейс (проект №2, подключаем проект №1 с помощью добавления зависимостей):

// MyPlugin.cs

public class MyPlugin : IPlugin

{

    public void Execute()

    {

        Console.WriteLine("MyPlugin executed!");

    }

}

1. Теперь создадим основной класс, который будет загружать сборку плагина и вызывать метод Execute (проект №3, подключаем проект №1 с помощью добавления зависимостей):

// Program.cs

using System;

using System.IO;

using System.Reflection;

class Program

{

    static void Main(string[] args)

    {

        // Путь к сборке плагина

        string pluginPath = "MyPlugin.dll"; // Убедитесь, что этот файл существует в директории

        // Загружаем сборку плагина

        Assembly pluginAssembly = Assembly.LoadFrom(pluginPath);

        // Ищем типы, реализующие интерфейс IPlugin

        foreach (Type type in pluginAssembly.GetTypes())

        {

            if (typeof(IPlugin).IsAssignableFrom(type) && !type.IsInterface && !type.IsAbstract)

            {

                // Создаем экземпляр плагина

                IPlugin pluginInstance = (IPlugin)Activator.CreateInstance(type);

                // Вызываем метод Execute

                pluginInstance.Execute();

            }

        }

    }

}

**Задание на практическую работу**

1. Разработать приложение, которое будет использовать рефлексию для динамического создания полей ввода на основе свойств класса.
2. Создать библиотеку классов и добавить класс согласно варианту.
3. Создать Windows Forms приложение и подключить dll библиотеки классов с помощью позднего связывания (Assembly.LoadFrom) без добавления в зависимости.
4. При загрузке формы используйте рефлексию, чтобы получить все свойства класса и создать соответствующие элементы управления (например, TextBox и Label) для каждого свойства:
5. Добавьте кнопку "Сохранить", в которой реализуйте возможность сериализации объекта в формат JSON (из примера в методичке) и отображения его в текстовом поле.
6. Для успешной защиты:

* программа должна быть задокументирована с помощью комментариев,
* код программы не должен содержать не используемые блоки, должен быть «чистым» и стилистически верно оформлен,
* интерфейс программы должен быть гибким и удобным.

**Варианты заданий:**

1. **Car**

Поля: Model, Year, Mileage, Price, EnginePower, FuelConsumption

1. **Book**

Поля: Title, Author, Genre, PublicationYear, ISBN, PageCount

1. **Person**

Поля: FirstName, LastName, Age, Gender, Email, PhoneNumber

1. **Employee**

Поля: EmployeeID, Name, Department, Position, Salary, HireDate

1. **Product**

Поля: ProductID, Name, Category, Price, StockQuantity, Supplier

1. **House**

Поля: Address, SquareFootage, NumberOfBedrooms, NumberOfBathrooms, Price, YearBuilt

1. **Movie**

Поля: Title, Director, ReleaseYear, Genre, Duration, Rating

1. **Student**

Поля: StudentID, Name, Major, GPA, EnrollmentYear, GraduationYear

1. **Animal**

Поля: Species, Age, Weight, Habitat, Diet, ConservationStatus

1. **Smartphone**

Поля: Brand, Model, ReleaseYear, StorageCapacity, RAM, Price

1. **Recipe**

Поля: RecipeName, Ingredients, CookingTime, Servings, Cuisine, Instructions

1. **Event**

Поля: EventName, Date, Location, Duration, Participants, Organizer

1. **BankAccount**

Поля: AccountNumber, AccountHolder, Balance, AccountType, InterestRate, OpenDate

1. **Course**

Поля: CourseName, CourseCode, Credits, Instructor, Semester, Prerequisites

1. **Laptop**

Поля: Brand, Model, Processor, RAM, Storage, Price

1. **Game**

Поля: Title, Genre, Platform, ReleaseDate, Rating, Developer

1. **InsurancePolicy**

Поля: PolicyNumber, PolicyHolder, CoverageAmount, Premium, StartDate, EndDate

1. **TravelPackage**

Поля: Destination, Duration, Price, Inclusions, Exclusions, TravelAgency

1. **Subscription**

Поля: ServiceName, MonthlyFee, StartDate, EndDate, Features, User Count

1. **Plant**

Поля: Species, WateringFrequency, SunlightNeeds, SoilType, Height, GrowthRate

# Содержание пояснительной записки

1. Постановка задачи. Приводится теоретический материал, использованный при написании приложения.

2. Формулировка задания и вариант. Приводится задание на лабораторную работу и вариант этого задания.

3. Описание выполняемых действий. Необходимо привести описание последовательности разработки программы, реализации используемых методов, алгоритмов, блок-схем.

4. Анализ результатов. Привести анализ входных и выходных данных. Показать результаты выполнения программного кода. Предоставить скриншоты обработки тестовых примеров. Сделать выводы.

5. Листинг программы. Привести листинг разработанного программного кода, содержание файлов входных и выходных данных.

# Используемое программное обеспечение

1. Среда программирования MS Visual Studio Community 2022 (Свободно распространяемое программное обеспечение (в учебных целях));
2. Microsoft Office Standard 2007 (Open License: 42267924);
3. Open Office (Свободно распространяемое программное обеспечение).
4. Браузер (Свободно распространяемое программное обеспечение).

# Список литературы

* + - 1. Мейер Б. Объектно-ориентированное программирование и программная инженерия [Электронный ресурс]/ Мейер Б. – Электрон. текстовые данные. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 285 c.
      2. Биллиг, В. A. Основы объектного программирования на С# (C# 3.0, Visual Studio 2008) [Электронный ресурс] : учебное пособие / В. A. Биллиг. — Электрон. текстовые данные. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2017. — 583 c. — 978-5-4487-0145-0. — Режим доступа: http://www.iprbookshop.ru/72339.html
      3. Павловская, Т. А. Программирование на языке высокого уровня C# [Электронный ресурс] / Т. А. Павловская. — 2-е изд. — Электрон. текстовые данные. — М. : Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 245 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/73713.html
      4. Агапов, В. П. Основы программирования на языке С# [Электронный ресурс] : учебное пособие / В. П. Агапов. — Электрон. текстовые данные. — М. : Московский государственный строительный университет, ЭБС АСВ, 2012. — 128 c. — 978-5-7264-0576-6. — Режим доступа: http://www.iprbookshop.ru/16366.html
      5. Медведев, М. А. Программирование на СИ# [Электронный ресурс] : учебное пособие / М. А. Медведев, А. Н. Медведев ; под ред. А. В. Присяжный. — Электрон. текстовые данные. — Екатеринбург : Уральский федеральный университет, ЭБС АСВ, 2015. — 64 c. — 978-5-7996-1561-1. — Режим доступа: http://www.iprbookshop.ru/69667.html
      6. Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual С# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 c
      7. Уйманова Н.А. Основы объектно-ориентированного программирования [Электронный ресурс]: практикум/ Уйманова Н.А., Таспаева М.Г.— Электрон. текстовые данные.— Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2017.— 156 c.— Режим доступа: http://www.iprbookshop.ru/78808.html.— ЭБС «IPRbooks»
      8. Новиков П.В. Объектно-ориентированное программирование [Электронный ресурс]: учебно-методическое пособие к лабораторным работам/ Новиков П.В.— Электрон. текстовые данные.— Саратов: Вузовское образование, 2017.— 124 c.— Режим доступа: http://www.iprbookshop.ru/64650.html.— ЭБС «IPRbooks»