Практическая работа №1

Цель работы: получить навыки создания и описания классов на языке C#.

Теоретическая часть

Класс представляет собой шаблон, по которому определяется форма объекта. В нем указываются данные и код, который будет оперировать этими данными.

Класс является логической абстракцией. Физическое представление класса появится в оперативной памяти лишь после того, как будет создан объект этого класса. Таким образом, объект – экземпляр некоторого класса.

Члены класса подразделяются на два типа – данные-члены и функции-члены.

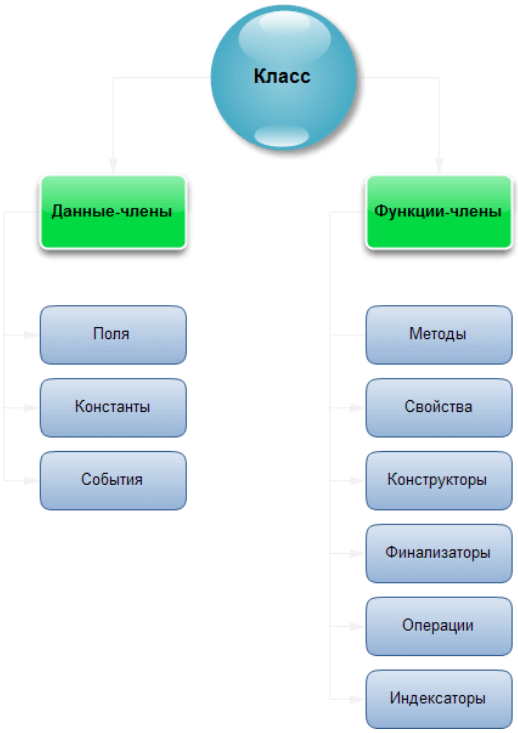

Рисунок 1 – Члены класса

Данные-члены — это те члены, которые содержат данные класса. Они могут быть статическими. Член класса является членом экземпляра, если только он не объявлен явно как static.

Функции-члены — это члены, которые обеспечивают некоторую функциональность для манипулирования данными класса.

Рассмотрим подробнее описание членов класса.

Поля (field)

Это любые переменные, ассоциированные с классом. При объявлении поля обязательно указывается его тип. Также можно указать модификатор доступа, обозначить поле статическим или проинициализировать.

```
private double length;
private double width;
private string name = "name";
```

Рисунок 2 – Поля класса

Константы (constant)

Могут быть ассоциированы с классом тем же способом, что и переменные. Константа объявляется с помощью ключевого слова const. Если она объявлена как public, то в этом случае становится доступной извне класса.

Следует помнить, что при компиляции значение константы хранится в метаданных сборки.

```
private const string Name = "name";
public const int Count = 0;
```

Рисунок 3 – Константы

События (event)

Это члены класса, позволяющие объекту уведомлять вызывающий код о том, что случилось нечто достойное упоминания, например, изменение свойства класса либо некоторое взаимодействие с пользователем.

По своей сути событие – это именованный делегат. При вызове этого делегата будут вызваны все подписавшиеся методы заданной сигнатуры.

```
public class EventClass
{
    public delegate void SampleEventHandler(string message);
    public event SampleEventHandler SampleEvent;

}
```

Рисунок 4 – Событие

```
void DisplayMessage(string message) => Console.WriteLine(message);

EventClass EventExample = new EventClass();

EventExample.SampleEvent += DisplayMessage;
EventExample.SampleEvent -= DisplayMessage;
```

Рисунок 5 – Обработчик события

Методы (method)

Это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора static.

```
public static int Add (int a, int b)
{
    return a + b;
}

ссылка: 1
public void Increment (ref int arg)
{
    arg += 10;
}

Ссылок: 0
public void Output ()
{
    int a = 10;
    Increment(ref a);
    Console.WriteLine($"Result: {a}");
}
```

Рисунок 6 – Методы класса

Свойства (property)

Это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса.

В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится. Это происходит автоматически на этапе компиляции, в том числе и при использовании автосвойств.

```
public double Length
{
    get
    {
        return length;
    }
    set
    {
        if (value > 0) this.length = value;
    }
}
```

Рисунок 7 – Свойство класса

Конструкторы (constructor)

Это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.

Если в классе не объявить конструктор, то для него будет задан конструктор по умолчанию. Он не принимает аргументов и используется только для выделения памяти под объект. Если же объявить конструктор с аргументами, то использовать конструктор по умолчанию не получится (если только перед этим не описать его вручную).

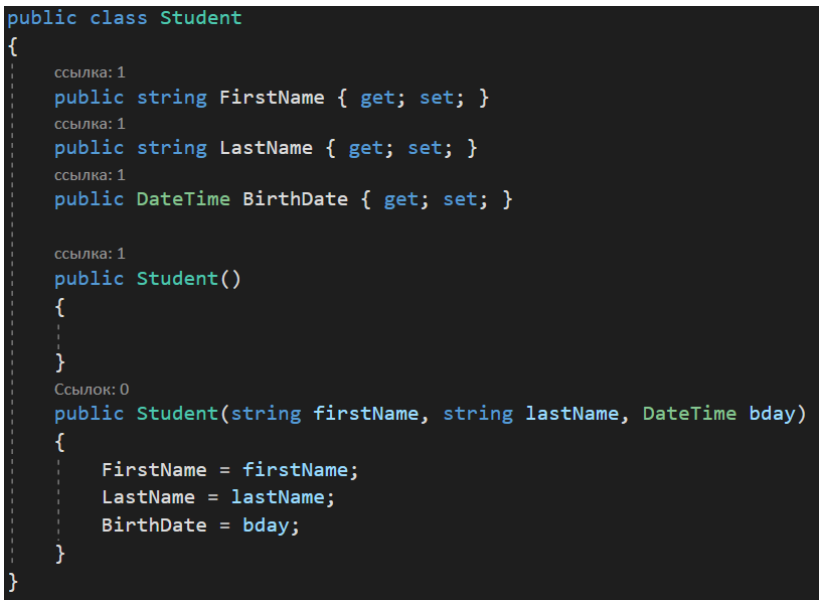

Рисунок 8 – Конструкторы класса

```
Student student1 = new Student();
Student student2 = new Student()
{ FirstName = "Ivan", LastName = "Ivanov", BirthDate = date };
Student student3 = new Student("Petr", "Petrov", date);
```

Рисунок 9 – Вызов конструкторов

Финализаторы (finalizer)

Вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды. Предсказать точно, когда будет вызван финализатор, невозможно.

Таким образом, в финализаторе описывается логика, которую необходимо перед тем, как сборщик мусора очистит память, выделенную под объект. При этом сами объекты, имеющие финализатор, помещаются в специальную область кучи – очередь финализации.

```
class Car
{
    Ссылок: 0
    ~Car()  // finalizer
    {
        // cleanup statements...
    }
}
```

Рисунок 10 – Финализатор

Операторы

Это простейшие действия вроде + или -. Когда вы складываете два целых числа, то, строго говоря, применяете операцию + к целым.

Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операции).

```
public static bool operator == (Circumference a, Circumference b)
{
    return a.Radius == b.Radius;
}

0 references
public static bool operator !=(Circumference a, Circumference b) => a.Radius != b.Radius;
```

Рисунок 11 – Операторы класса

Индексаторы

Позволяют индексировать объекты таким же способом, как массив или коллекцию.

```
float[] temps = new float[10]
{
56.2F, 56.7F, 56.5F, 56.9F, 58.8F,
61.3F, 65.9F, 62.1F, 59.2F, 57.5F
};
Ссылок: 0
public int Length => temps.Length;
Ссылок: 0
public float this[int index]
{
    get => temps[index];
    set => temps[index] = value;
}
```

Рисунок 12 – Индексатор

Разделяемый класс

Можно разделить определение класса, структуры, интерфейса или метода между двумя или более исходными файлами. Каждый исходный файл

содержит часть определения класса или метода, а во время компиляции приложения все части объединяются.

- Все части должны использовать ключевое слово partial.
- Для формирования окончательного типа все части должны быть доступны во время компиляции.
- Все части должны иметь одинаковые модификаторы доступа.
- Если какая-либо из частей объявлена абстрактной, то весь тип будет считаться абстрактным.
- Если какая-либо из частей объявлена запечатанной, то весь тип будет считаться запечатанным.
- Если какая-либо из частей объявляет базовый тип, то весь тип будет наследовать данный класс.

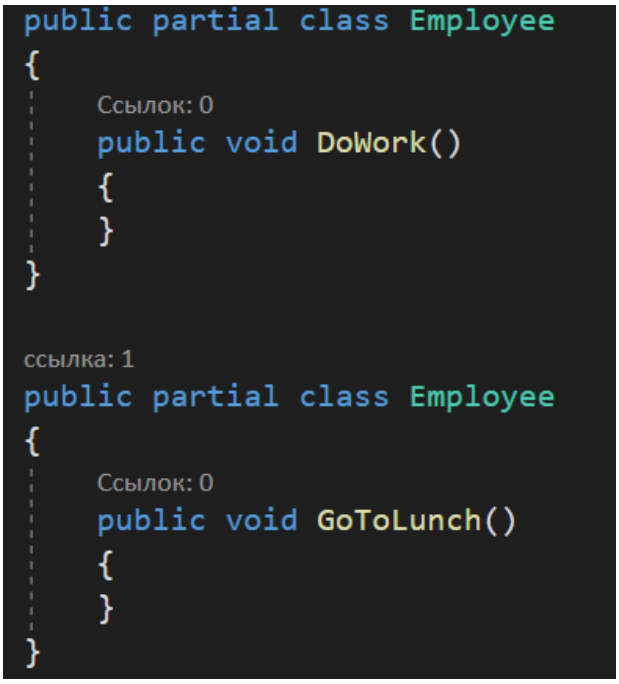

Рисунок 13 – Разделяемый класс

Статические классы

Статический класс может использоваться как обычный контейнер для наборов методов, работающих на входных параметрах, и не должен возвращать или устанавливать каких-либо внутренних полей экземпляра.

Такой класс:

- содержит только статические члены;
- создавать его экземпляры нельзя;
- является запечатанным;
- не может содержать конструкторы экземпляров.

По сути, создание статического класса аналогично созданию класса, содержащего только статические члены и закрытый конструктор. Нестатический класс может содержать статические методы, поля, свойства или события. Статический член вызывается для класса даже в том случае, если не создан экземпляр класса. Доступ к статическому члены всегда выполняется по имени класса, а не экземпляра. Существует только одна копия статического члена, независимо от того, сколько создано экземпляров класса.

Статические методы и свойства не могут обращаться к нестатическим полям и событиям в их содержащем типе, и они не могут обращаться к переменной экземпляра объекта, если он не передается явно в параметре метода. Статические методы могут быть перегружены, но не переопределены, поскольку они относятся к классу, а не к экземпляру класса. Доступ к статическому члену всегда выполняется по имени класса, а не экземпляра.

C# не поддерживает статические локальные переменные.

Практическая часть

Для диаграммы классов необходимо разработать соответствующие классы на языке C#. Описывать логику работы методов не нужно.