ЛАБОРАТОРНАЯ РАБОТА 5. КЛАССЫ, ОБЪЕКТЫ, НАСЛЕДОВАНИЕ

Цель лабораторной работы: Познакомиться с основой объектного подхода в языке C#, созданием объектов, классов и механизмом наследования.

5.1. Классы и обьекты

Класс является основой для создания объектов. В классе определяются данные и код, который работает с этими данными. Объекты являются экземплярами класса. Методы и переменные, составляющие класс, называются членами класса. При определении класса объявляются данные, которые он содержит, и код, работающий с этими данными. Данные содержатся в переменных экземпляра, которые определены классом, а код содержится в методах. В C# определены несколько специфических разновидностей членов класса. Это — переменные экземпляра, статические переменные, константы, методы, конструкторы, деструкторы, индексаторы, события, операторы и свойства.

Непосредственно инициализация переменных в объекте (переменных экземпляра) происходит в конструкторе. В классе могут быть определены несколько конструкторов.

Синтаксис класса:

```
class имя_класса{
        тип_доступа тип имя_переменной1;
        тип_доступа тип имя_переменной2;
        ...
        тип_доступа возвращаемый_тип имя_метода1(список_параметров)
        {
                тело_метода
        }
}
```

где тип_доступа может быть public, private, protected, internal.

Члены класса с типом доступа public доступны везде за пределами данного класса, с типом доступа protected – внутри членов данного класса и производных, с типом доступа private - только для других членов данного класса. Тип доступа internal применяется для типов, доступных в пределах одной сборки.

Пример:

```
class Animal{
        public string Name;
        private int Weight;
        protected int Type;
        public int Animal(int W, int T, string N){
```

```
            Weight=W;
            Type=T;
            Name=N;
        }
        public int GetWeight(){return Weight;}
}
```

Создание обьекта:
имя_класса имя_обьекта = new имя_класса();

При создании обьекта класса происходит вызов соответствующего конструктора класса.

5.2. Конструктор и деструктор

Конструктор класса – метод для инициализации объекта при его создании. Он имеет то же имя, что и его класс. В конструкторах тип возвращаемого значения не указывается явно. Конструкторы используются для присваивания начальных значений переменным экземпляра, определенным классом, и для выполнения любых других процедур инициализации, необходимых для создания объекта.

Все классы имеют конструкторы независимо от того, определен он или нет. По умолчанию в C# предусмотрено наличие конструктора, который присваивает нулевые значения всем переменным экземпляра (для переменных обычных типов) и значения null (для переменных ссылочного типа). Но если конструктор явно определен в классе, то конструктор по умолчанию использоваться не будет.

имя_класса(список_параметров) {тело_конструктора}

Деструктор – метод, вызывающийся автоматически при уничтожении объекта класса (непосредственно перед “сборкой мусора”). Деструктор не имеет параметров и возвращаемого значения.

~имя_класса() {тело_деструктора}

5.3. Наследование

Наследование — это свойство, с помощью которого один объект может приобретать свойства другого. При этом поддерживается концепция иерархической классификации, имеющей направление сверху вниз. Используя наследование, объект должен определить только те качества, которые делают его уникальным в пределах своего класса. Он может наследовать общие атрибуты от своих родительских классов.
Синтаксис:

```
class имя_класса : имя_родительского_класса
{
        тело_класса
}
```

Пример:

```
class Predator:Animal{
        private int Speed;
}
```

С помощью наследования создается иерархия классов (отношение ‘являться’). Кроме того, можно построить еще одну структуру – иерархию объектов (тогда, когда один объект является частью другого – отношение ‘часть-целое’).

5.4. Полиморфизм

Полиморфизм – одна из основных составляющих объектно-ориентированного программирования, позволяющая определять в наследуемом классе методы, которые будут общими для всех наследующих классов, при этом наследующий класс может определять специфическую реализацию некоторых или всех этих методов. Главный принцип полиморфизма: «один интерфейс, несколько методов». Благодаря ему, можно пользоваться методами, не обладая точными знаниями о типе объектов.

Основным инструментом для реализации принципа полиморфизма является использование виртуальных методы и абстрактных классов.

5.5. Виртуальные методы

Метод, при определении которого в наследуемом классе было указано ключевое слово virtual, и который был переопределен в одном или более наследующих классах, называется виртуальным методом. Следовательно, каждый наследующий класс может иметь собственную версию виртуального метода.

Выбор версии виртуального метода, которую требуется вызвать, осуществляется в соответствии с типом объекта, на который ссылается ссылочная переменная, во время выполнения программы. Другими словами, именно тип объекта, на который указывает ссылка (а не тип ссылочной переменной), определяет вызываемую версию виртуального метода. Таким образом, если класс содержит виртуальный метод и от этого класса были наследованы другие классы, в которых определены свои версии метода, при ссылке переменной типа наследуемого класса на различные типы объектов вызываются различные версии виртуального метода.

При определении виртуального метода в составе наследуемого класса перед типом возвращаемого значения указывается ключевое слово virtual, а при переопределении виртуального метода в наследующем классе

используется модификатор override. Виртуальный метод не может быть определен с модификатором static или abstract.

Переопределять виртуальный метод не обязательно. Если наследующий класс не предоставляет собственную версию виртуального метода, то используется метод наследуемого класса.

Переопределение метода положено в основу концепции динамического выбора вызываемого метода - выбора вызываемого переопределенного метода осуществляется во время выполнения программы, а не во время компиляции.

Синтаксис:

virtual тип имя (список_параметров){тело_метода};

5.6. Абстрактные классы

В абстрактном классе определяются лишь общие предназначения методов, которые должны быть реализованы в наследующих классах, но сам по себе этот класс не реализует один, или несколько подобных методов, называемых абстрактными (для них определены только некоторые характеристики, такие как тип возвращаемого значения, имя и список параметров).

При объявлении абстрактного метода используется модификатор abstract. Абстрактный метод автоматически становится виртуальным, так что модификатор virtual при объявлении метода не используется. Абстрактный класс предназначен только для создания иерархии классов, нельзя создать объект абстрактного класса.

Пример:

```
abstract class Animal
{
        public string Name;
        protected int Weight;
        private int Type;
        abstract void Feed();
        public int Animal(int W, int T, string N)
        {
                Weight=W;
                Type=T;
                Name=N;
        }
        public int GetWeight()
        {
                return Weight;
        }
}
```

```
class Predator:Animal
{
	private int Speed;
	override void Feed(int Food)
	{
		Weight += Food;
	}
}
```

5.7. Интерфейсы

В C# для полного отделения структуры класса от его реализации используется механизм интерфейсов.

Интерфейс является расширением идеи абстрактных классов и методов. Синтаксис интерфейсов подобен синтаксису абстрактных классов. Объявление интерфейсов выполняется с помощью ключевого слова interface. При этом методы в интерфейсе не поддерживают реализацию. Членами интерфейса могут быть методы, свойства, индексаторы и события.

Интерфейс может реализовываться произвольным количеством классов. Один класс, в свою очередь, может реализовывать любое число интерфейсов. Каждый класс, включающий интерфейс, должен реализовывать его методы. В интерфейсе для методов неявным образом задается тип public. В этом случае также не допускается явный спецификатор доступа.

Синтаксис:

[атрибуты] [модификаторы] interface Имя_интерфейса [:список_родительских_интерфейсов]

```
{
	обьявление_свойств_и_методов
}
```

Пример:

```
interface Species
{
	string Species();
	void Feed();
}
class Cheetah:Animal,Species{
	private string ScientificName;
	public string Species()
	{
		return ScientificName;
	}
	public void Feed()
	{
		Weight++;
```

```
        }
}
```

Можно объявлять ссылочную переменную, имеющую интерфейсный тип. Подобная переменная может ссылаться на любой объект, который реализует ее интерфейс. При вызове метода объекта с помощью интерфейсной ссылки вызывается версия метода, реализуемого данным объектом.

Возможно наследование интерфейсов. В этом случае используется синтаксис, аналогичный наследованию классов. Если класс реализует интерфейс, который наследует другой интерфейс, должна обеспечиваться реализация для всех членов, определенных в составе цепи наследования интерфейсов.

5.8. Делегаты

Делегат — это объект, имеющий ссылку на метод. Делегат позволяет выбрать вызываемый метод во время выполнения программы. Фактически значение делегата – это адрес области памяти, где находится точка входа метода. Важным свойством делегата является то, что он позволяет указать в коде программы вызов метода, но фактически вызываемый метод определяется во время работы программы, а не во время компилирования.

Делегат объявляется с помощью ключевого слова delegate, за которым указывается тип возвращаемого значения, имя делегата и список параметров вызываемых методов.

Синтаксис:
delegate тип_возвращаемого_значения имя_делегата (список_параметров);

Характерной особенностью делегата является возможность его использования для вызова любого метода, который соответствует подписи делегата. Это дает возможность определить во время выполнения программы, какой из методов должен быть вызван. Вызываемый метод может быть методом экземпляра, ассоциированным с объектом, либо статическим методом, ассоциированным с классом. Метод можно вызвать только тогда, когда его подпись соответствует подписи делегата.

5.9. Многоадресность делегатов

Многоадресность — это способность делегата хранить несколько ссылок на различные методы, что позволяет при вызове делегата инициировать эту цепочку методов. Для создания цепочки методов необходимо создать экземпляр делегата, и пользуясь операторами + или += добавлять методы к цепочке. Для удаления метода из цепочки используется оператор - или -=. Делегаты, хранящие несколько ссылок, должны иметь тип возвращаемого значения void.

5.10. Индивидуальные задания

Построить иерархию классов в соответствии с вариантом задания:

1. Студент, преподаватель, персона, заведующий кафедрой
2. Служащий, персона, рабочий, инженер
3. Рабочий, кадры, инженер, администрация
4. Деталь, механизм, изделие, узел
5. Организация, страховая компания, нефтегазовая компания, завод
6. Журнал, книга, печатное издание, учебник
7. Тест, экзамен, выпускной экзамен, испытание
8. Место, область, город, мегаполис
9. Игрушка, продукт, товар, молочный продукт
10. Квитанция, накладная, документ, счет
11. Автомобиль, поезд, транспортное средство, экспресс
12. Двигатель, турбина, дизель, электромотор
13. Республика, монархия, королевство, государство
14. Млекопитающее, парнокопытное, птица, животное
15. Корабль, пароход, парусник, корвет
16. Дерево, растение, мох, гриб
17. Винтовка, гранатомет, оружие, кинжал.
18. Вертолет, дельтаплан, самолет, транспорт.
19. Дом, комната, жилье, квартира.
20. Радиоволны, электромагнитное излучение, ультрафиолет, рентгеновские лучи.
21. Математика, геометрия, наука, биология
22. Дождь, погода, снег, ветер
23. Доска, стройматериалы, кирпич, брус
24. Семья, нация, народность, общество
25. Полк, дивизия, армия, бригада