Лабораторная работа № 10.
НАСЛЕДОВАНИЕ И ВИРТУАЛЬНЫЕ ФУНКЦИИ

Цель. Получить практические навыки создания иерархии классов.

Часть 1. Создание иерархии классов.

Теоретические сведения.

Наследование – это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

Существующие классы называются **базовыми**, а новые – **производными**. Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа. С помощью наследования может быть создана иерархия классов, которые совместно используют код и интерфейсы.

Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах.

В иерархии производный объект наследует разрешенные для наследования компоненты всех базовых объектов (*public, protected*).

В иерархии классов соглашение относительно доступности компонентов класса следующее:

private – член класса может использоваться только функциями – членами данного класса и функциями – "друзьями" своего класса. В производном классе он недоступен.

protected – то же, что и ***private,*** но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – "друзьями" классов, производных от данного.

public – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к ***public*** - членам возможен доступ извне через имя объекта.

Синтаксис определения производного класса:
[атрибуты] [спецификаторы] class имя_класса [: предки] тело класса

В иерархии классов как базовые, так и производные классы могут иметь собственные конструкторы. При этом конструктор базового класса создает часть объекта, соответствующую базовому классу, а конструктор производного класса — часть объекта, соответствующую производному классу.

Если конструкторы определены и в базовом, и в производном классе, в процессе создания объектов должны выполниться конструкторы обоих классов. В этом случае необходимо использовать ключевое слово base, которое имеет два назначения:

- вызвать конструктор базового класса;
- получить доступ к элементу базового класса.

Виртуальным называется метод, объявляемый с помощью ключевого слова virtual в базовом классе и переопределяемый в одном или нескольких производных классах.

При использовании виртуальных методов, версию метода, который нужно вызвать, C# определяет по типу объекта, на который указывает ссылка, причем решение принимается динамически, во время выполнения программы. Следовательно, если имеются ссылки на различные объекты, будут выполняться различные версии виртуального метода.

Чтобы объявить метод в базовом классе виртуальным, его объявление необходимо предварить ключевым словом virtual. При переопределении виртуального метода в производном классе используется модификатор override. При переопределении метода сигнатуры типа у виртуального и метода-заменителя должны совпадать.

Виртуальный метод нельзя определять как статический (static) или абстрактный (abstract).

Переопределение виртуального метода формирует базу для одной из самых мощных концепций C#: динамической диспетчеризации методов (позднее связывание). Динамическая диспетчеризация методов — это механизм вызова переопределенного метода во время выполнения программы, а не в период компиляции. Именно благодаря механизму диспетчеризации методов в C# реализуется динамический полиморфизм.

Основное содержание работы.

Написать программу, в которой создается иерархия классов. Записать объекты классов в массив, выполнить просмотр элементов массива. Показать использование виртуальных функций.

Порядок выполнения работы.

1. Определить иерархию классов (в соответствии с вариантом).
2. Реализовать классы.
3. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в массив, после чего массив просматривается.
4. Реализовать 2 варианта программы: с помощью виртуальных и не виртуальных методов.
5. Без виртуальных функций программа будет работать неправильно! Объяснить почему. Объяснить необходимость виртуальных функций

Варианты заданий.

1) студент, преподаватель, **персона**, сотрудник;
2) служащий, **персона**, рабочий, инженер;
3) рабочий, **персона**, инженер, администрация;
4) **организация,** страховая компания, судостроительная компания, завод, библиотека;
5) тест, экзамен, выпускной экзамен, **испытание;**
6) **место,** область, город, мегаполис, адрес;
7) игрушка, продукт, **товар**, молочный продукт;
8) квитанция, накладная, **документ**, чек;
9) цех, мастерская, фабрика, **производство**;
10) **персона**, студент, школьник, студент-заочник;
11) автомобиль, поезд, **транспортное средство**, экспресс;
12) республика, монархия, королевство, **государство**;
13) млекопитающие, парнокопытные, птицы, **животное**;
14) **корабль**, пароход, парусник, корвет,
15) **двигатель,** двигатель внутреннего сгорания, дизель, турбореактивный двигатель;
16) журнал, книга, **печатное издание**, учебник.

Часть 2. Динамическая идентификация типов.

Теоретические сведения.

Динамическая идентификация типов (runtime type identification — RTTI) позволяет определить тип объекта во время выполнения программы, что необходимо во многих ситуациях. Например, можно совершенно точно узнать, на объект какого типа в действительности указывает ссылка на базовый класс.

В C# предусмотрено три ключевых слова, которые поддерживают динамическую идентификацию типов: is , as и typeof.

С помощью оператора is можно определить, имеет ли рассматриваемый объект заданный тип. Общая форма его записи имеет следующий вид:

выражение is ТИП

Здесь тип элемента выражение сравнивается с элементом ТИП. ЕСЛИ ТИП элемента выражение совпадает (или совместим) с элементом ТИП, результат выполнения операции принимается равным значению ИСТИНА. В противном случае — значению

ЛОЖЬ. Следовательно, если результат истинен, выражение можно привести к типу, заданному элементом ТИП.

```
class Pair{}
class Fraction : Pair{}


Pair[] vector = new Pair[3];
. . . . .

int count1 = 0;
foreach (Pair p in vector)
        if (p is Fraction) count1++;
Console.WriteLine("В массиве " + count1 + " объектов типа Fraction");
```

Иногда во время работы программы требуется выполнить операцию приведения типов, не генерируя исключение в случае, если попытка окажется неудачной. Для этого предусмотрен оператор as, формат которого таков:

выражение as тип

```
int count2 = 0;
Fraction f = new Fraction();
foreach (Pair p in vector)
        {
            f = p as Fraction;
            if (f != null) count2++;

        }
Console.WriteLine("В массиве " + count2 + " объектов типа Fraction");
```

Основное содержание работы.

1. Реализовать метод для выполнения заданных запросов. При необходимости (для выполнения запроса) в класс могут быть добавлены новые поля (по сравнению с частью 1). В программе должно быть минимум ввода с клавиатуры. Поля объектов задаются в тексте программы. С клавиатуры вводятся только параметры запроса.
2. Реализовать не менее трех запросов, соответствующих иерархии классов (можно реализовать свои запросы).

Запросы.

1. Имена всех лиц мужского (женского) пола.
2. Имена студентов указанного курса.
3. Имена и должность преподавателей указанной кафедры.
4. Имена служащих со стажем не менее заданного.
5. Имена служащих заданной профессии.
6. Имена рабочих заданного цеха.
7. Имена рабочих заданной профессии.
8. Имена студентов, сдавших все (заданный) экзамены на отлично (хорошо и отлично).
9. Количество инженеров на заводе.
10. Имена всех монархов на заданном континенте.
11. Наименование всех деталей (узлов), входящих в заданный узел (механизм).
12. Наименование всех книг в библиотеке (магазине), вышедших не ранее указанного года.
13. Названия всех городов заданной области.
14. Наименование всех товаров в заданном отделе магазина.
15. Количество мужчин (женщин).

16. Количество студентов на указанном курсе.
17. Количество рабочих со стажем не менее заданного.
18. Количество рабочих заданной профессии.
19. Количество инженеров в заданном подразделении.
20. Количество товара заданного наименования.
21. Количество студентов, сдавших все экзамены на отлично.
22. Количество студентов, не сдавших хотя бы один экзамен.
23. Количество деталей (узлов), входящих в заданный узел (механизм).
24. Количество указанного транспортного средства в автопарке (на автостоянке).
25. Количество пассажиров во всех вагонах экспресса.
26. Суммарная стоимость товара заданного наименования.
27. Средний балл за сессию заданного студента.
28. Количество библиотек в городе.
29. Суммарное количество учебников в библиотеке (магазине).
30. Суммарное количество жителей всех городов в области.
31. Суммарная стоимость продукции заданного наименования по всем накладным.
32. Средняя мощность всех (заданного типа) транспортных средств в организации.
33. Средняя мощность всех дизелей, обслуживаемых заданной фирмой.
34. Средний вес животных заданного вида в зоопарке.
35. Среднее водоизмещение всех парусников на верфи (в порту).
36. Суммарный вес всех деталей в заданном узле.
37. Количество рабочих в заданном цехе.
38. Наименование всех цехов на данном заводе.
39. Количество жителей данного континента.
40. Наименование птиц в зоопарке.
41. Имена пароходов, приписанных к данному порту.
42. Количество различных типов ДВС, обслуживаемых автомастерской.
43. Наименование журналов, выписываемых библиотекой.
44. Суммарная стоимость всех деталей в механизме.
45. Суммарный страховой фонд всех страховых компаний региона.
46. Количество книг во всех библиотеках города.
47. Самый мощный автомобиль в данной организации.
48. Количество чеков на сумму превышающую заданную.
49. Общая сумма по всем чекам, выписанным в организации.
50. Самая дорогая и самая дешевая игрушка в магазине(наименование и стоимость).

Часть 3. Абстрактные классы и интерфейсы

Теоретические сведения.

Интерфейс является крайним случаем абстрактного класса. Он определяет **поведение**, которое поддерживается реализующими этот интерфейс классами. В нем задается набор

абстрактных методов, свойств и индексаторов, которые должны быть реализованы

в производных классах. Основная идея использования интерфейса состоит в том, чтобы к объектам таких классов можно было обращаться одинаковым образом.

Каждый класс может определять элементы интерфейса по-своему. Таким образом, достигается полиморфизм: объекты разных классов по-разному реагируют на вызовы одного и того же метода.

Синтаксис интерфейса аналогичен синтаксису класса:

[атрибуты] [спецификаторы] interface имя_интерфейса [: предки]

тело_интерфейса

В списке предков класса сначала указывается его базовый класс, если он есть, а затем через запятую — интерфейсы, которые реализует этот класс. Таким образом, в C#

поддерживается одиночное наследование для классов и множественное — для интерфейсов. Это позволяет придать производному классу свойства нескольких базовых интерфейсов, реализуя их по своему усмотрению.

Сигнатуры методов в интерфейсе и реализации должны полностью совпадать. Для реализуемых элементов интерфейса в классе следует указывать спецификатор public. К этим элементам можно обращаться как через объект класса, так и через объект типа соответствующего интерфейса. Объекты типа интерфейса, так же как и объекты абстрактных классов, создавать нельзя.

В библиотеке классов ."NET определено множество стандартных интерфейсов, задающих желаемое поведение объектов.

Интерфейс IComparable определен в пространстве имен System. Он содержит всего один метод CompareTo, возвращающий результат сравнения двух объектов – текущего и переданного ему в качестве параметра:

interface IComparable

int CompareTo(object obj)

Метод должен возвращать:

– 0, если текущий объект и параметр равны;
– отрицательное число, если текущий объект меньше параметра;
– положительное число, если текущий объект больше параметра.

```
//Отсортировать вектор, в который записаны элементы классов SimpleFraction и
//DecimalFraction.
class SimpleFraction:IPair,IVisible,IComparable
    {
        int numenator;
        int denumenator;
        . . . . .
        public int CompareTo(object obj)//реализация интерфейса
        {
            SimpleFraction temp = (SimpleFraction)obj;//приведение к типу SimpleFraction
            if (this.numenator * temp.denumenator > temp.numenator * this.denumenator)
return 1;
            if (this.numenator * temp.denumenator < temp.numenator * this.denumenator)
return -1;
            return 0;
        }

    }
class DecimalFraction:IPair,IVisible
    {
        double number;
        public double Number
        . . . . .
        public static   implicit operator SimpleFraction(DecimalFraction d)
        {
            SimpleFraction temp=new SimpleFraction();
            double int_part=Math.Truncate(d.Number);
            double dec_part=d.Number-int_part;
            int power = 1;
            if (dec_part!=0)
            while (dec_part < 1) { dec_part *= 10; power *= 10; }
            temp.Num = Convert.ToInt32(int_part * power + dec_part);
            temp.Denum=power;
            return temp;
        }
}

static void Main(string[] args)
{
            . . . . . .
```

```
            SimpleFraction[] vector_sort = new SimpleFraction[3];
            vector_sort[0] = f1;
            vector_sort[1] = f2;
            vector_sort[2] = f3;
            Array.Sort(vector_sort);
            Console.WriteLine("Вектор отсортированный:");
            foreach (IPair p in vector_sort)
            {
                p.Show();
                Console.WriteLine();
            }
}
```

Интерфейс IComparer определен в пространстве имен System. Collections. Он содержит один метод CompareTo, возвращающий результат сравнения двух объектов, переданных ему в качестве параметров:

```
interface IComparer
{
        int Compare ( object obi, object ob2 )
}
```

Принцип применения этого интерфейса состоит в там, что для каждого критерия сортировки объектов описывается небольшой вспомогательный класс, реализующий этот интерфейс. Объект этого класса передается в стандартный метод сортировки массива в качестве второго аргумента (существует несколько перегруженных версий этого метода).

```
//Отсортировать вектор, в который записаны элементы класса Student по полю name (age,
//rating, course).
abstract class Person
    {
        protected string name;
        protected int age;
        public abstract void Init();
        public abstract void Show();
    }

    class Student : Person,IComparable
    {
        protected int course;//курс
        protected double rating;

        public string Name
        {
            set { name = value; }
            get { return name; }
        }
        public int Age
        {
            set { if (value > 0)age = value; else age = 1; }
            get { return age; }
        }
        public int Course
        {
            set
            {
                if ((value >= 1) && (value <= 6)) course = value;
                else course = 1;
            }
            get { return course; }
        }
        public double Raiting
        {
            set
            {
```

```
                if ((value > 0) && (value <= 5)) rating = value;
                else rating = 0.0;
            }
            get { return rating; }
        }

        public Student(string n, int a, int c, double r)
        {
            name = n; age = a; course = c; rating = r;
        }
        public Student()
        {
            name = ""; age = 1; course = 1; rating = 0.0;
        }
        public override void Init()
        {
            string buf;
            Console.WriteLine("ФИО:");
            buf = Console.ReadLine();
            name = Convert.ToString(buf);
            Console.WriteLine("Возраст:");
            buf = Console.ReadLine();
            age = Convert.ToInt32(buf);
            Console.WriteLine("Курс:");
            buf = Console.ReadLine();
            course = Convert.ToInt32(buf);
            Console.WriteLine("Рейтинг:");
            buf = Console.ReadLine();
            rating = Convert.ToDouble(buf);
        }
        public override void Show()
        {
            Console.WriteLine(name + ", " + age + ", " + course + " курс, рейтинг: " +
rating);
        }

public int CompareTo(object obj)//реализация интерфейса
        {
            Student temp = (Student)obj;//приведение к типу Student
            if (String.Compare(this.name, temp.name)>0 ) return 1;
            if (String.Compare(this.name, temp.name) < 0) return -1;
            return 0;
        }

}
public class SortByName: IComparer
        {
            int IComparer.Compare(object ob1, object ob2)
            {
                Student s1 = (Student)ob1;
                Student s2 = (Student)ob2;
                return String.Compare(s1.Name, s2.Name);
            }

    }
static void Main(string[] args)
{
Student[] StudentGroup = new Student[5];
            StudentGroup[0] = new Student("Иванов", 17, 1, 4.5);
            StudentGroup[1] = new Student("Петров", 18, 1, 3.5);
            StudentGroup[2] = new Student("Иванова", 20, 3, 4.0);
            StudentGroup[3] = new Student("Сидоров", 22, 5, 4.5);
            StudentGroup[4] = new Student("Кузнецова", 18, 1, 4.5);
            Console.WriteLine("Сортировка по имени");
            Array.Sort(StudentGroup, new SortByName());
```

```
            foreach (Student s in StudentGroup)
                s.Show();
}
```

Клонирование - это создание копии объекта. Копия объекта называется клоном. При присваивании одного объекта ссылочного типа другому копируется ссылка (адрес), а не сам объект. Если необходимо скопировать в другую область памяти поля объекта, можно воспользоваться методом MemberwiseClone(), который любой объект наследует от класса object. При этом объекты, на которые указывают поля объекта, в свою очередь являющиеся ссылками, не копируются. Это называется поверхностным клонированием. Для создания полностью независимых объектов необходимо глубокое клонирование, когда в памяти создается дубликат всего дерева объектов, то есть объектов, на которые ссылаются поля объекта, поля полей и т. д. Алгоритм глубокого клонирования весьма сложен, поскольку требует рекурсивного обхода всех ссылок объекта и отслеживания циклических зависимостей. Объект, имеющий собственные алгоритмы клонирования, должен объявляться как наследник **интерфейса ICloneable** и переопределять его единственный метод C1one().

```
class Person:ICloneable
    {
        string name;
        int age;
        public string Name
        {
            set { name = value; }
            get { return name; }
        }
        public int Age
        {
            set { age = value; }
            get { return age; }
        }
        public Person(string s, int a)
        {
            name = s; age = a;
        }
        public void Show()
        {
            Console.WriteLine(name + ", " + age);
        }
        public Person ShallowCopy() //поверхностное копирование
        {
            return (Person)this.MemberwiseClone();
        }
        public object Clone()
        {
            return new Person("Клон"+this.name, this.age);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Person[] mas = new Person[]{new Person("Иванов",25), new Person("Петров",34),
new Person("Сидоров",55)};
            foreach (Person x in mas)
                x.Show();

            Person p1 = new Person("",0);
            p1 = mas[0];
            p1.Show();
```

```
            p1 = mas[1].ShallowCopy();
            p1.Show();

            p1 =(Person) mas[2].Clone();
            p1.Show();
        }
    }
```

1. Составить иерархию классов в соответствии с вариантом. Иерархия должна содержать хотя бы один интерфейс и хотя бы один абстрактный класс.
2. Создать массив интерфейсных элементов и поместить в него экземпляры различных классов иерархии.
3. Реализовать сортировку элементов массива, используя стандартные интерфейсы и методы класса Array.
4. Реализовать поиск элемента в массиве, используя стандартные интерфейсы и методы класса Array.
5. Реализовать в одном из классов метод клонирования объектов. Показать клонирование объектов.